

Rapport de stage

“Vue anamorphique sous Unity3D”

Mihai ANCA

Tuteur: Pascal DESBARATS

LaBRI

Enseignant référent: Lionel CLEMENT

**Université
de BORDEAUX**

Table des matières

Introduction	2
1 Environnement de développement	3
1.1 Recherche initiale	3
1.2 Création du projet	3
1.3 Installation d'OpenCV	4
1.4 Détection du visage	5
2 Méthodes	6
2.1 Anamorphose	6
2.1.1 Anamorphose statique	6
2.1.2 Anamorphose dynamique	10
2.2 Anaglyphe rouge/cyan	13
2.3 Billboards 3D	15
3 Application	17
3.1 Recherche	17
3.2 Idées	19
3.3 Implémentation	20
3.3.1 Intérieur	20
3.3.2 Extérieur	25
3.3.3 Prix final	26
3.3.4 Menu principal	26
3.3.5 Générique de fin	27
3.4 Résultats	28
Conclusion	30
Bibliographie	33

Introduction

De nos jours, l'état de la technologie nous offre la possibilité de développer diverses vues 3D à l'aide d'équipements spécialisés. Il existe des appareils comme les écrans 3D, les caméras 3D et les lunettes AR et VR. Malheureusement, ceux-ci ne sont pas encore accessibles au grand public, ce qui les rend impratiques pour la plupart des gens.

Pour cette raison, nous voyons l'émergence d'alternatives qui répliquent l'effet 3D sur des écrans 2D. Pour en citer quelques-unes, nous avons l'anaglyphe rouge/cyan, la vue **Side-By-Side** et l'anamorphose. Le grand avantage de ces derniers est le coût bien faible des matériels nécessaires pour que l'effet soit perceptible à l'œil humain.

Quant à l'anamorphose, il y en a deux types : statique et dynamique. L'anamorphose statique crée une déformation d'un objet qui le rend visible seulement à partir de certains points de vue. L'une des applications que nous pouvons déjà observer déployées sont les *billboards*, des panneaux incurvés qui affichent une publicité anamorphosée. L'anamorphose dynamique généralise cet effet au rendu en temps réel afin de donner l'impression de profondeur d'un écran lors des rotations de tête de l'utilisateur. Ceci reste aussi assez coûteux, à cause de la nécessité des caméras 3D qui peuvent encore coûter 300€ ! Alors, nous avons décidé de nous servir de l'apprentissage automatique afin d'éliminer ce besoin, en ayant remplacé la détection exacte de la tête avec une estimation effectuée à partir d'une *webcam* classique.

En ce qui concerne l'anaglyphe rouge/cyan, deux versions décalées de la même image, une passée par un filtre rouge, une autre cyan, sont superposées afin de créer une impression de profondeur en utilisant une paire de lunettes spécifiques qui ont un coût bien faible.

Suite à la recherche de stage que j'ai lancée par mail, j'ai reçu le sujet proposé par M. Pascal DESBARATS, représentant du LaBRI. Il s'agit du sujet intitulé "Vue anamorphique sous Unity3D", qui me donne la tâche d'implémenter la vue «anamorphique» sous ses deux formes, statique et dynamique, afin d'en créer une démo d'un jeu de plateforme 3D sous Unity. La durée du stage a été fixée à un mois et demi.

Dû à des contraintes personnelles, nous sommes convenus d'effectuer ce stage en télétravail, ce qui nous a posé des contraintes supplémentaires. Pour communiquer, nous avons choisi d'échanger par mail et d'avoir des séances régulières sur Zoom pour faire le point sur l'état d'avancement du projet.

Dans ce document, je présente le processus de développement de cette application, en passant par les différentes étapes requises : le préparation du projet, l'installation et l'utilisation de la bibliothèque OpenCV sous Unity, l'implémentation des deux formes d'anamorphose, de l'anaglyphe rouge/cyan et des *billboards*, ainsi que leurs utilisations dans le processus de développement d'un *platformer* 3D.

Chapitre 1

Environnement de développement

Dans ce chapitre, je présenterai la partie préparatrice de mon stage, en commençant avec une recherche initiale que j'ai effectuée sur l'anamorphose, les outils de gestion de projet que j'ai mis en place afin de rester organisé tout au long du processus, l'installation de la bibliothèque OpenCV, ainsi que l'inclusion du code de détection du visage qui a été fourni dans le même tutoriel.

1.1 Recherche initiale

Je me suis renseigné sur les méthodes existantes et j'ai trouvé plusieurs travaux déjà effectués sur le thème de l'anamorphose qui m'ont semblé intéressants :

- Un jeu entier (qui n'est malheureusement plus disponible en téléchargement) a été créé en utilisant l'anamorphose statique pour créer des illusions optiques utiles à la création de différents puzzles. [Don15]
- Un projet Unity qui implémente l'anamorphose dynamique à l'aide d'un casque AR et dont le dépôt git est public. [McK22]
- Deux vidéos sur YouTube qui montrent l'effet perçu quand l'anamorphose dynamique est utilisée. [Bul11] [Sof13]

J'ai également trouvé deux éléments qui me semblaient potentiellement utiles, mais qui ne m'ont enfin pas servi :

- Un dépôt git avec un projet de *Lens Flare* anamorphique qui ne semble pas être le même effet que celui que l'on cherche. [kei17]
- Une propriété Unity de la caméra qui contrôlerait l'anamorphose à l'aide d'une valeur flottante quand les propriétés physiques sont activées [Unib], mais je n'ai pas pu la faire fonctionner dans le contexte du projet. Après une mise à jour d'Unity, j'ai pu l'inclure dans un script de test, mais il n'y avait aucun effet visible.

1.2 Crédit du projet

Avant de me lancer dans le sujet, j'ai dû mettre en place plusieurs logiciels et outils de gestion de projet :

- Visual Studio Community 2019 - l'environnement de programmation utilisé par Unity ;
- Unity 2021.3.25f1 LTS - la dernière version d'Unity disponible depuis mon compte étudiant ;
- un dépôt Github du projet qui est disponible ici :
<https://github.com/hyperion2022/anamorphic-view> ;
- un espace Trello pour la gestion des tâches et la sauvegarde de documents importants ;
- un projet Overleaf pour la rédaction du rapport de stage et d'autres éventuels documents en L^AT_EX.

Malgré ma réticence initiale face à l'utilisation des IA, je me suis également autorisé d'utiliser chatGPT en tant qu'outil de travail afin d'optimiser mon processus de recherche et de déboggage de code.

Ainsi, j'ai créé un projet Unity à partir du *Third Person Template*, qui me fournit un personnage qui peut

effectuer les mouvements basiques (marche, course, saut etc.) et du décor qui met en valeur ces propriétés. Ce prototype m'a facilité la tâche de création d'un jeu vidéo en me donnant une base solide sur laquelle j'ai pu tester l'anamorphose.

Ensuite, je l'ai chargé sur le dépôt `git` (avec un `.gitignore` adapté) et j'ai mis en place le `Trello` avec le sujet de stage, les articles de recherche et des liens utiles sur l'utilisation et des *assets* qui auraient pu me servir à la suite.

Pour tester l'anamorphose sur un objet classique, j'ai trouvé le fameux modèle de la théière que j'ai placée dans la scène :

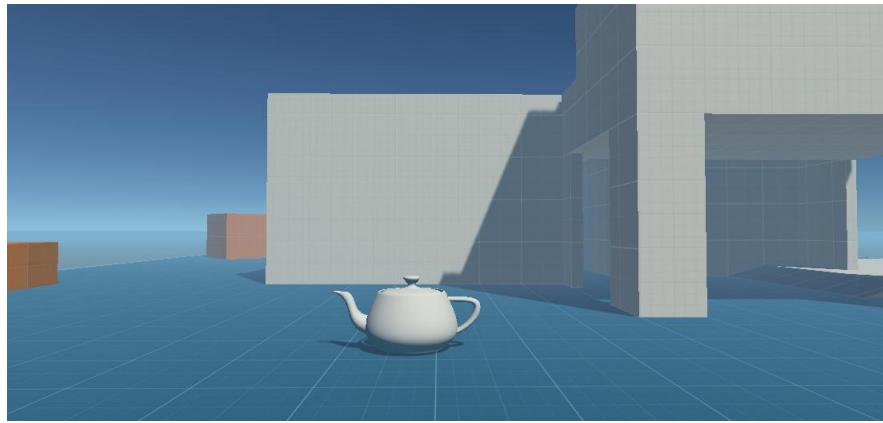


FIGURE 1.1 – Modèle "teapot" dans la scène du *Third Person Template*

Avec ces outils mis en place, j'ai pu me lancer dans le sujet en tenant compte du rôle de chacun. Ceci m'a aidé à garder la trace des rendez-vous et de mon travail à chaque pas.

1.3 Installation d'OpenCV

OpenCV est une bibliothèque *open-source* de vision par ordinateur, ce qui offre des fonctionnalités qui ne sont pas incluses dans le projet `Unity` de base. Je m'en suis servi pour récupérer le flux de la *webcam* afin d'effectuer des traitements spécifiques dessus et de passer les résultats à la boucle de rendu d'`Unity`.

Le processus d'installation et de mise en place de la bibliothèque manuellement sur PC a été assez sinueux, à cause des différentes étapes qui demandent des traitements spécifiques au système d'opération.

Tout commence avec ce tutoriel en trois parties qui m'a été suggéré [Mou17] ; il s'agit d'un très bon guide, mais qui n'est malheureusement plus disponible sur le site personnel de la personne qui l'a créé.

Par contre, je venais d'apprendre de l'existence d'une fonctionnalité du site *Internet Archive* qui s'appelle "*WayBack Machine*" [way]. Ceci m'a permis de retrouver facilement le contenu du site à un instant du passé où le tutoriel s'y retrouvait.

Ainsi, le processus comporte 3 étapes :

1. L'installation effective de la bibliothèque : téléchargement des fichiers source, compilation avec `CMake`, création de fichiers spécifiques à l'aide de `Visual Studio`, copie des fichiers résultants vers un endroit spécifique aux logiciels (dans mon cas, `C:\Program Files (x86)`) et la mise à jour des variables de l'environnement pour créer le lien entre mon code et la bibliothèque.
2. La mise en place du projet : création d'un nouveau projet C++ sous `Visual Studio`, passage sous x64, spécification de bas niveau des propriétés du projet (à l'aide des variables de l'environnement créées à l'étape 1), mention des dépendances `.dll` et enfin l'étape de *Build* en mode *Debug* et *Release*.
3. Le passage des données de détection vers `Unity` : copie de dépendances `.dll` dans le projet `Unity`, écriture de script qui retrouve les fonctions externes du projet `OpenCV` pour les utiliser à la suite.

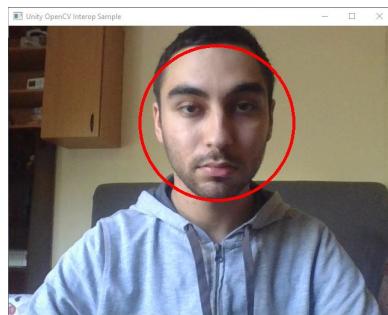


FIGURE 1.2 – La fenêtre de débogage de la détection du visage sous Unity

Après ce long processus, j'ai eu accès aux données capturées par la *webcam* dans le code C# afin de les utiliser avec l'API d'Unity et une fenêtre qui sert au débogage.

Les détails de l'implémentation seront détaillés dans la partie suivante.

1.4 Détection du visage

Le code nécessaire pour ce traitement a également été fourni dans le tutoriel d'installation, ce qui pourrait ensuite être amélioré avec la détection individuelle des yeux, mais la détection du visage est l'élément nécessaire pour l'anamorphose dynamique (voir [chapitre 2](#)).

Pour ceci, nous considérons une structure **Circle** qui passe des informations sur le(s) visage(s) détecté(s) sous la forme d'une liste de cercles (composés par la position du centre et la longueur du rayon) afin de rendre l'ajout aussi léger que possible. Celle-ci sera définie en C++ dans le projet défini à l'étape 2 du tutoriel.

L'image est capturée sous forme de flux vidéo et la détection s'effectue à l'aide d'un fichier .xml qui contient les données nécessaires sous la forme d'une "Cascade LBP" qui, selon l'auteur du tutoriel, serait plus efficace qu'une "Cascade de Haar" classique.

A la suite, il y a quatre fonctions (**Init**, **Close**, **SetScale** et **Detect**) qui sont implémentées de façon à être reconnaissables par C# avec un format spécifique.

Leur implémentation effectue, dans l'ordre, l'initialisation du fichier cascade et du flux vidéo, la fermeture du flux vidéo, la mise à jour du paramètre **scale** utilisé pour le passage des données à taille réduite et la détection effective.

Du côté Unity, le code devra recevoir ces données afin de les utiliser à la suite.

Pour cela, une nouvelle classe C# est créée avec l'attribut `[DllImport("UnityOpenCVSample")]` pour chacune des fonctions exportées du code C++ (avec de légères variations selon le besoin).

Ensuite, la structure **Circle** est redéfinie afin de recevoir le nombre équivalent d'octets que le code C++ enverra.

Le code final initialise la plupart des paramètres dans la fonction **Start()**, utilise **OnApplicationQuit()** pour fermer le flux vidéo et dans **Update()**, la détection se passe afin de remplir une `List<Vector2>` avec les positions 2D des visages détectés. Cette dernière sera ensuite utilisable dans le contexte de l'anamorphose dynamique.

Nous venons de voir l'étape initiale du stage qui rend l'implémentation effective des méthodes d'effets 3D possible.

Chapitre 2

Méthodes

Dans ce chapitre, je présente en détail quatre effets 3D auxquels je me suis intéressé au cours du stage : l'anamorphose statique et dynamique, l'anaglyphe rouge/cyan et les *billboards*. Chacune des méthodes choisies sera expliquée et sera accompagnée par des compléments, comme l'implémentation, la boîte englobante ou les cartes de profondeur.

2.1 Anamorphose

2.1.1 Anamorphose statique

Lecture d'article (1)

J'ai choisi l'article [HC07] que nous avons utilisé dans le projet de PdP en M1 dans le but de recréer l'effet anamorphique.

Celui-ci commence avec une description de l'effet et avec un court historique des essais de reproduire l'effet "à la main".

L'une des contributions les plus anciennes qui est donné en tant qu'exemple classique de l'anamorphose est le tableau "Les Ambassadeurs" par Hans Holbein le Jeune (qui date de 1536) dont le crâne en bas n'est visible que si l'on s'éloigne vers la droite du cadre.

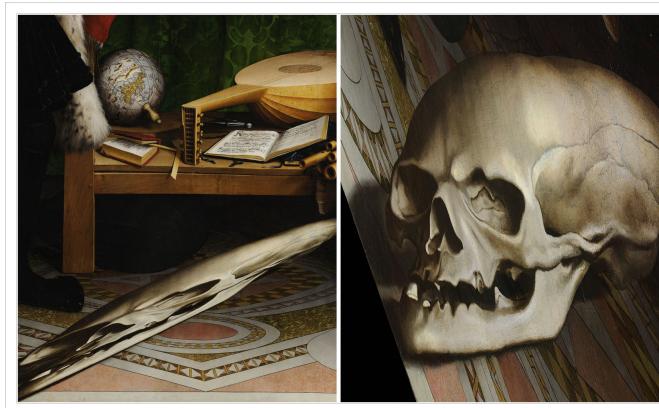


FIGURE 2.1 – Vues frontale et latérale droite du détail anamorphique de “Les Ambassadeurs”

L'algorithme présenté se base sur des notions de géométrie 3D (des transformations de l'espace) et sur le

fonctionnement d'une caméra sous OpenGL. Il s'agit d'une méthode qui transforme un maillage de points afin d'obtenir un effet anamorphique.

Pour modéliser une caméra en projection perspective, nous considérons que toutes les données capturées de l'environnement convergent vers un seul point - le centre de projection (**cop**). Ceci peut être paramétré par rapport à l'angle d'ouverture (*Field of View ou FOV*), et par les dimensions du **frostm**, une pyramide tronquée qui englobe les données "visibles" depuis la caméra.

Plus précisément, nous disposons du **cop** vers lequel convergent les données, l'angle d'ouverture θ , la distance **n** vers le plan **near** et la distance **f** vers le plan **far** entre lesquels les données sont prises en compte pour le rendu final. **h** représente la hauteur de l'image ; le plan **near** est effectivement notre écran.

Ainsi, nous pouvons relier les données **n**, **h** et θ à l'aide de la formule :

$$\tan(\theta/2) = \frac{h/2}{n}$$

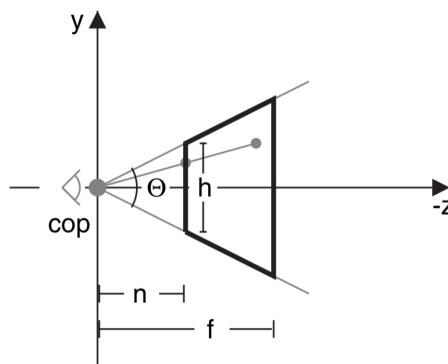


FIGURE 2.2 – Schéma du fonctionnement d'un frostum

L'algorithme présenté est le suivant :

ENTREE : le maillage à modifier, les données de la caméra ($0 \leq \theta \leq 180^\circ$), des facteurs pour **n** et pour **f** ;
SORTIE : le maillage modifié ;

1. Définir une boîte englobant le maillage donné **b**.
2. Choisir une face de cette boîte (en tant que plan **near**) devant laquelle placer le **cop**. La distance vers la face opposée sera notée **d**.
3. Déterminer **n** à partir de θ et de **h**.
4. Calculer $f = n + d$.
5. Calculer $n_p = facteur_n * n$ et $f_p = facteur_f * f$.
6. Passer les sommets du repère local au repère de la caméra.
7. Calculer les nouveaux points :

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} \frac{b_x * p_z}{n} \\ \frac{b_y * p_z}{n} \\ \frac{r * n_p * f_p}{r * n_p + f - b_z} \end{bmatrix}, \quad r = \frac{f - n}{f_p - n_p}$$

8. Remettre les points du repère de la caméra au repère local.

Pour résumer la méthode, nous effectuons une "réctification", c'est-à-dire que les points qui suivent des trajectoires convergeant vers le **cop** seront alignés sur des trajectoires parallèles afin d'être retransformés en "réctification" inverse. Il s'agit donc, en algèbre linéaire, d'un "changement de base".

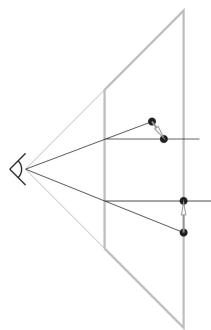
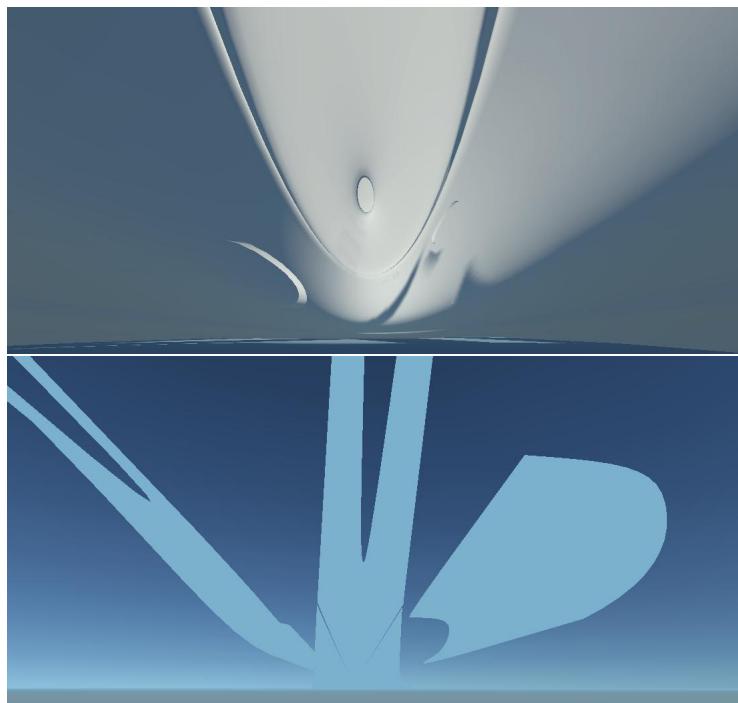


FIGURE 2.3 – Représentation de l'étape de "réctification"

Boîte englobante

Pour commencer l'algorithme, j'ai eu besoin d'une boîte englobante et la solution proposée par **Unity** n'a pas été assez développée pour mes besoins.

Ce qui est offert dans l'API est la classe **Bounds** [[Unia](#)], qui représente les boîtes englobantes alignées aux axes (**AABB**), ce qui n'est pas adapté à nos besoins d'alignement par rapport aux caméras. Vu la nature rigide de cette classe, j'ai eu des résultats non satisfaisants :

FIGURE 2.4 – Versions buguées de l'anamorphose en utilisant une **AABB**

Une version ultérieure dont la méthode a été corrigée ne faisait qu'aplatir le théière jusqu'à donner l'impression d'une texture 2D, mais l'effet restait différent de celui attendu.

A cause de cet inconvénient, j'ai pensé à écrire une version d'une boîte englobante orientée (**Oriented Bounding Box** ou **OBB**) en mettant ensemble dans une nouvelle classe une instance de la classe **Bounds** et une composante **Transform** afin de retenir l'orientation (position, rotation et échelle) de la boîte séparément.

Ceci n'a malheureusement pas fonctionné à cause d'un défaut de l'idée de base. Ne considérer que la rotation peut laisser des sommets en dehors de la boîte si l'objet est plus long que large.

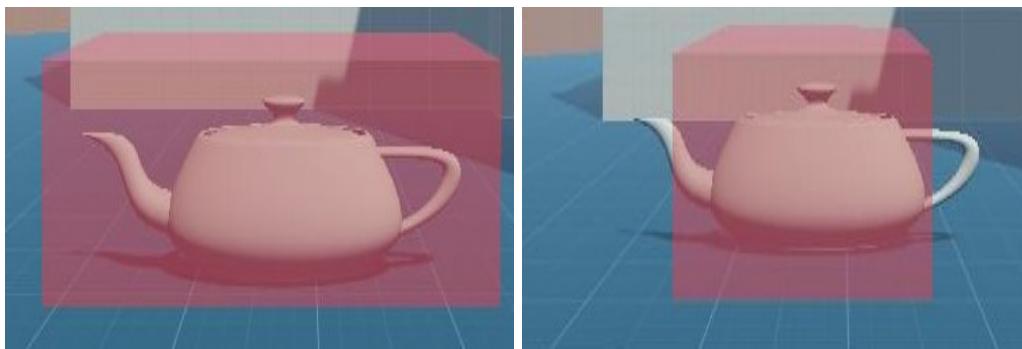


FIGURE 2.5 – Exemple d'une boîte englobante le modèle et la même boîte tournée à 90°

A la suite, j'ai considéré une implémentation complète d'une OBB afin de la rendre entièrement fonctionnelle. Cette fois-ci, j'ai rédigé le code entièrement à l'aide de **chatGPT** afin de pouvoir me concentrer sur l'algorithme d'anamorphose. Ceci est dû au manque de ressources en ligne sur les boîtes englobantes 3D ; j'en ai trouvées uniquement sur la version 2D pour encadrer les objets détectés lors de l'usage du **ML** dans les images.

Ainsi, cette nouvelle classe dispose de plusieurs fonctions utiles pour recréer l'effet :

- `public int GetClosestFaceIndex(Camera camera)`
- `public float CalculateDistanceBetweenClosestAndOpposingFace(Camera camera)`
- `public float CalculateNearDistance(Camera camera)`

et d'autres fonctions liées à l'implémentation géométrique de bas niveau de la boîte.

Après avoir testé plusieurs versions de l'effet anamorphique, cette implémentation semble être la plus fiable et sans effets de bord visibles.

Implémentation

Le premier choix à faire a été de calculer l'effet qu'une seule fois dans la fonction `Start()` pour enlever le risque de croissance exponentielle du maillage ou de complexité trop élevée. Ceci a entraîné le besoin d'une fonction d'initialisation de la boîte englobante pour enlever le risque d'accès aux données non-initialisées.

La traduction des étapes en code a été assez directe, les seules incertitudes provenant de l'implémentation de certaines fonctions de l'API d'Unity.

J'ai créé une fonction `private void calculateAnamorphosis(Mesh mesh, float nFactor, float fFactor, Camera camera)` que j'appelle depuis la fonction `Start()` avec le maillage de l'objet **Unity** sur lequel je rajoute le **script**, une caméra arbitraire et deux facteurs que je choisis aussi arbitrairement.

Une fois le maillage référencé sous forme de tableau de `Vector3`, je récupère la boîte englobante attachée au même objet (étape 1). Les étapes 2, 3 et 4 s'effectuent à l'aide des fonctions spécifiques à la classe `OrientedBoundingBox`. L'étape 5 n'est représenté que par deux multiplications élémentaires.

Pour la transformation effective, le rapport `r` est calculé et un nouveau tableau de sommets `p` est créé. Une boucle `for` met dans chaque `p[i]` les valeurs des `b[i]` remis dans le repère de la caméra. Les nouveaux points sont calculés selon la formule de l'étape 7 et ils sont remis en repère local. Une fois la boucle finie, les sommets sont remis dans le maillage et les bornes sont recalculées (selon la spécification d'Unity).

Les passages entre les différents repères sont effectués à l'aide des fonctions `Camera.WorldToViewportPoint` et `Camera.ViewportToWorldPoint`. Suite à un déplacement inattendu du maillage, l'étape 8 (optionnelle) a été omise.

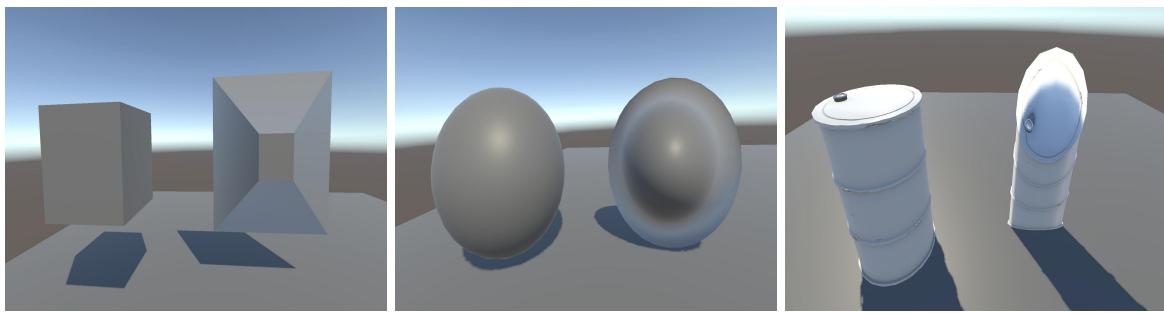


FIGURE 2.6 – Effet de l'implémentation courante sur un cube, une sphère et un baril

J'ai également rencontré plusieurs soucis avec le maillage de la théière à cause des normales qui étaient mauvaises ou qui manquaient. L'une des versions initiales ne faisait qu'aplatiser les objets jusqu'à les rendre quasiment bi-dimensionnels. Alors, j'ai testé le **script** en forme finale sur plusieurs formes de base d'Unity et sur des objets du magasin officiel qui avaient des normales bien calculées.

Enfin, j'ai des résultats qui répliquent l'effet anamorphique, mais dont le paramétrage n'est pas entièrement fonctionnel pour créer des différences notables par rapport aux orientations de la caméra.

2.1.2 Anamorphose dynamique

Lecture d'article (2)

L'article mentionné par le sujet de stage [GP11] décrit une méthode basée sur l'utilisation du capteur Kinect de la Xbox. Celui-ci était très connu pour l'inclusion dans les consoles de Microsoft, comme l'analogique de la Wii Remote par Nintendo, et pour avoir été relativement accessible.

Le dispositif était composé par deux capteurs :

- un projecteur infra-rouge ;
- une caméra infra-rouge qui capte le modèle projeté par le premier et calcule une image de profondeur de résolution 640×480 pixels.

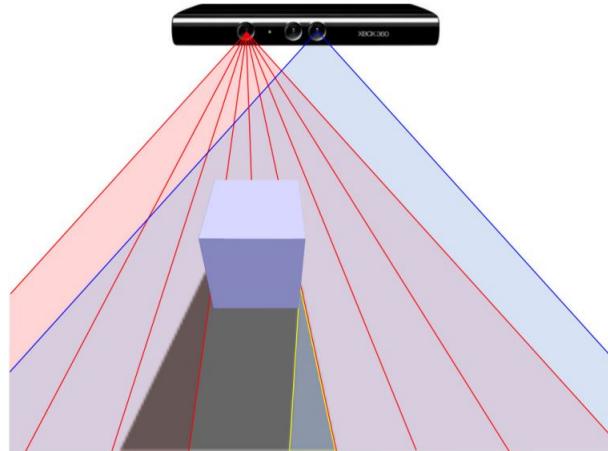


FIGURE 2.7 – Fonctionnement du système de capteurs de la Kinect

Vu la nature plus simple de la tâche courante, je n'ai extrait que les informations pertinentes aux caméras classiques. Ainsi, j'ai pu trouver une formule qui, à partir d'une valeur de gris d'une carte de profondeur, donne une approximation de la distance entre l'objet et la caméra (en cm) :

$$d'(u, v) = 5.28 \times \tan(d_{raw}(u, v)/694.77) + 17.72,$$

avec $d_{raw}(u, v)$ la valeur de gris de la carte de profondeur à la position (u,v).

Ici, on considère aussi que ces valeurs font partie de l'ensemble [[384, 1024]] selon le fonctionnement du Kinect, alors j'ai dû tenir compte du fait que les valeurs de gris d'une image classique capturée par OpenCV sont dans l'intervalle [[0, 255]]. Pour passer de l'un à l'autre, j'ai calculé une suite d'opérations de base sur les intervalles :

$$[[0, 255]](\times \frac{640}{255}) \longrightarrow [[0, 640]](+384) \longrightarrow [[384, 1024]]$$

Pour passer des coordonnées (u, v) aux coordonnées cartésiennes, nous avons les formules de passage suivantes :

$$\begin{aligned} x &= (u - 320) \times \frac{1.025 \times d}{640}, \\ y &= (u - 240) \times \frac{1.025 \times d}{640}, \\ z &= d, \end{aligned}$$

où $d = d'(u, v)$.

Ceci nous donne

$$h = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (2.1)$$

Ces formules semblent dépendre de la hauteur et de la largeur prédéfinies ($320 = 640 / 2$ et $240 = 480 / 2$ avec une division par 640 - la largeur), alors en manque de formes généralisées, la carte de profondeur devra correspondre à la résolution de 640×480 .

Nous pouvons atténuer les mouvements fins en calculant $\bar{h} = (1 - \alpha)\bar{h} + \alpha h$, avec $\alpha = \frac{1}{4}$ et la valeur initiale de \bar{h} étant h .

Pour effectuer la projection de la scène 3D, nous avons plusieurs transformations à réaliser :

1. Passer les coordonnées de la tête du repère de la caméra en repère monde à l'aide de la matrice T calculée en utilisant la solution de Horn à l'aide de paires de coordonnées de trois sphères : $h' = T \cdot \bar{h}$. Dans notre cas, les sphères sont des points de l'espace 3D et la version la plus intuitive est de considérer les 4 coins du plan *near* de la boîte englobante de l'objet en question. Ainsi, nous aurons un plan de référence défini par les 4 points à partir duquel seront considérées les distances vers l'utilisateur.

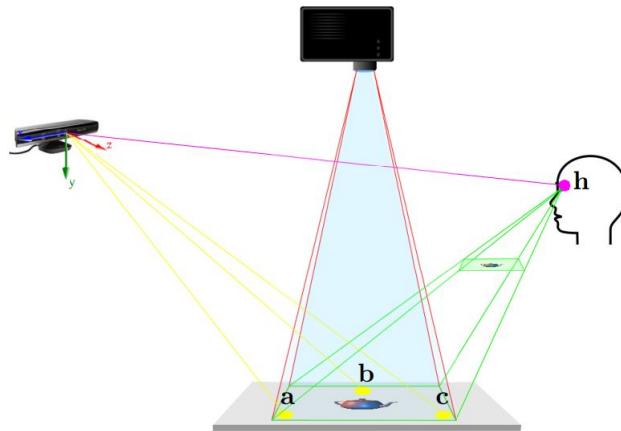


FIGURE 2.8 – Représentation des trois sphères de calibrage

2. Calculer la matrice de projection perspective P .

$$P = \begin{bmatrix} \frac{2d_n}{x'_2 - x'_1} & 0 & \frac{x'_2 + x'_1}{x'_2 - x'_1} & 0 \\ 0 & \frac{2d_n}{y'_2 - y'_1} & \frac{y'_1 + y'_2}{y'_2 - y'_1} & 0 \\ 0 & 0 & -\frac{d_f + d_n}{d_f - d_n} & -\frac{2d_f + d_n}{d_f - d_n} \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

avec $d_n = 1, d_f = f - h'_z$ et

$$x'_1 = (x_1 - h'_x)/n$$

$$x'_2 = (x_2 - h'_x)/n$$

$$y'_1 = (y_1 - h'_y)/n$$

$$y'_2 = (y_2 - h'_y)/n$$

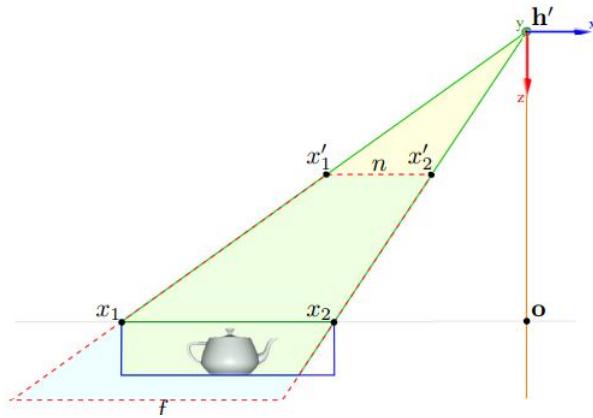


FIGURE 2.9 – Schéma du frostum modifié en fonction de la vue de l'utilisateur

La détection de la tête requise par cette méthode sera confiée à `OpenCV` et la carte de profondeur sera générée à partir d'une seule image capturée par la *webcam*. Cette dernière sera décrite à la suite.

Carte de profondeur

Afin de générer une carte de profondeur à partir d'une seule image, j'ai eu besoin d'une solution d'estimation de la profondeur à l'aide du *Machine Learning*.

Parmi les solutions existantes, celles que j'ai trouvées sont `DenseDepth` [ial19], `MiDaS` [io19], `stereoDNN` [IOT18] et `monodepth2` [mia18]. Celle qui m'a été directement utile a été `DenseDepth`, car elle m'a offert dès le départ des modèles pré-compilés, ce qui m'a évité d'apprendre comment le faire sous python.

Ainsi, j'ai trouvé le fichier correspondant à la *NYU Depth Dataset V2* [NSF12], respectivement `nyu.h5`. Malheureusement, ce format de fichiers n'est pas compatible avec la bibliothèque `OpenCV`, ce qui m'a amené à trouver un format adapté, respectivement le format ouvert `.onnx` (*Open Neural Network Exchange*).

En manque de version `.onnx`, j'ai dû faire le lien avec le modèle de ML codé sous python, alors j'ai réussi générer (à l'aide de `chatGPT`) et adapter (à cause d'un souci de compatibilité de fonction) un `script` de conversion que j'ai utilisé sous `WSL2`.

Une fois les différentes bibliothèques installées et le `script` lancé, je me suis retrouvé avec le fichier `densedepth.onnx` que j'ai enfin pu copier-coller dans le projet `OpenCV`.

A la suite, j'ai eu plusieurs soucis avec l'installation d'`OpenCV` suite aux modifications de code suivantes :

- La structure `Circle` évolue afin de retenir également la profondeur détectée au centre
- Dans la fonction `Detect`, nous utilisons le modèle `DenseDepth` : chargement de fichier sous forme de "blob", création et normalisation de la `Depth Map` et son affichage.

Quand j'ai essayé de compiler ce nouveau code dans le projet C++, j'ai eu des soucis de compatibilité avec le réseau de neurones (de nouvelles bibliothèques requises) provenant de l'installation d'`OpenCV` précédente et liés aux liaisons entre les différents fichiers et le projet (dans le `Linker`). Ceci m'a obligé de réinstaller plusieurs versions différentes de la bibliothèque ainsi :

- La version 4.7.0 ne fonctionnait plus, malgré les essais multiples de refaire certains pas du tutoriel [Mou17]. Nous avons eu peur que ce soit à cause d'une incompatibilité entre `OpenCV` et `Unity` ;
- J'ai tout supprimé et installé la version décrite dans le tutoriel, la 3.1.0. Par contre, elle ne supporte pas le module `dnn` (*Deep Neural Networks*) ;
- J'ai de nouveau tout supprimé et installé la première version qui supportait le module `dnn`, respectivement la 3.3.0. Par contre, elle ne supportait pas la fonction `readNetFromONNX` ;
- J'ai enfin tout supprimé et réinstallé la version de départ, la 4.7.0, et le projet compilait de nouveau. J'ai également dû remplacer plusieurs dépendances de base avec le fichier `opencv_world470.dll`.

Du côté `Unity`, j'ai implémenté les modifications suivantes :

- J'ai remis à jour la structure `Circle` ; La nouvelle variable de profondeur est de type `byte` (l'équivalent d'`unsigned char` en C++) , alors la taille totale de la structure augmente d'un octet.
- J'ai rajouté une liste de profondeurs que j'initialise à la suite ;
- Je la vide et j'y rajoute les profondeurs de chaque visage détecté à chaque tour de boucle.

Malheureusement, une fois réintégré dans le projet `Unity`, le code C++ me faisait planter l'application à cause du fichier `opencv_world470.dll` à chaque nouvelle exécution. Pour cette raison, j'ai décidé d'enlever tous les fichiers du projet C++ contenus dans le projet `Unity` afin de réaliser une nouvelle installation propre. Ceci m'a donné des erreurs de dépendances manquantes, sans me donner plus de détails sur les fichiers concernés. Je n'ai non plus trouvé de détails sur Internet ou en utilisant `chatGPT`, alors je n'ai malheureusement pas pu poursuivre l'implémentation de l'anamorphose dynamique.

La seule option qui m'est restée était le projet mentionné dans le ??, mais il a été créé pour fonctionner à l'aide d'une casque AR et nécessite les positions des coins de l'écran.

2.2 Anaglyphe rouge/cyan

A cause des soucis avec les différents types d'anamorphose, nous avons décidé de trouver une alternative au cas où jusqu'à la fin du stage, aucun des deux effets ne serait complètement fonctionnel.

Ainsi, j'ai trouvé ce dépôt sur `Github` [rya21], qui implémente l'effet 3D d'anaglyphe rouge/cyan au niveau de l'URP (*Universal Render Pipeline*).

Initialement, j'ai téléchargé la dernière version (v3.1.0), mais celle-ci provoquait des erreurs dans `Unity`. Après une demande auprès de l'auteur, j'ai eu une réponse rapide qui m'a dirigé vers la version 2.1.1, qui elle est compatible avec ma version d'`Unity` et de l'URP.

Ensuite, j'ai pu l'implémenter sans souci ; tout ce que j'ai eu à faire après l'importation du package a été de l'ajouter dans les *features* de l'objet `Asset_Renderer` qui se trouve dans le répertoire `Assets` et de lui passer la référence du `shader` correspondant.

L'effet est satisfaisant, mais il y a des lignes horizontales qui peuvent être considérées comme un effet de bord. Il existe également un bug qui fait que l'effet se perde une fois que les objets se croisent avec la `skybox`.

L'effet est visible à l'aide d'une paire de lunettes rouge/cyan qui peuvent être trouvés à un prix bien réduit. Ceci rend l'effet reproductible sur des machines classiques sans d'autres dépenses excessives.

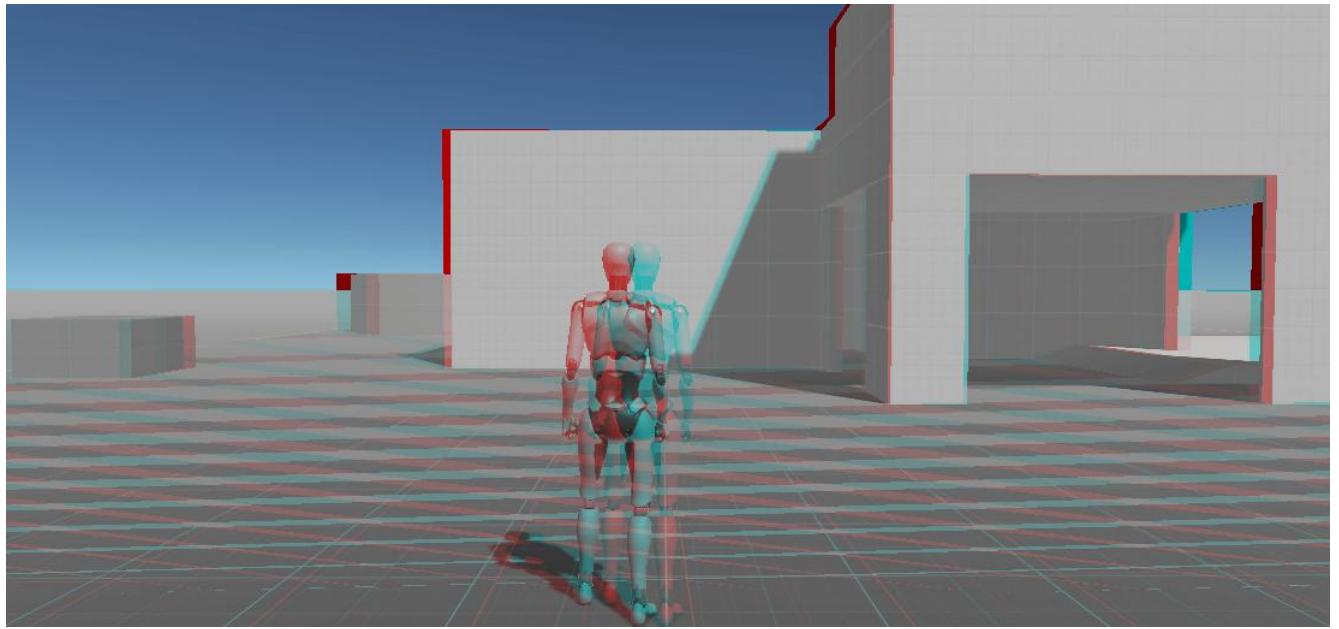


FIGURE 2.10 – Fonctionnement de l'anaglyphe en version 2

Ensuite, j'ai implémenté un court **script** qui active et désactive l'effet à chaque fois que l'utilisateur appuie sur une touche ;

Je l'ai considérée la touche du chiffre "1" vu que d'autres filtres pourraient être rajoutés ultérieurement dans la liste de touches numériques. En raison de temps limité, j'ai décidé de signaler cette possibilité à l'aide d'un message affiché par **Debug.Log**.

En mettant ultérieurement à jour **Unity** vers la version 2022.3.2f1, j'ai utilisé la version 3 de ce **package** et j'ai ainsi réussi éliminer les artéfacts horizontaux.

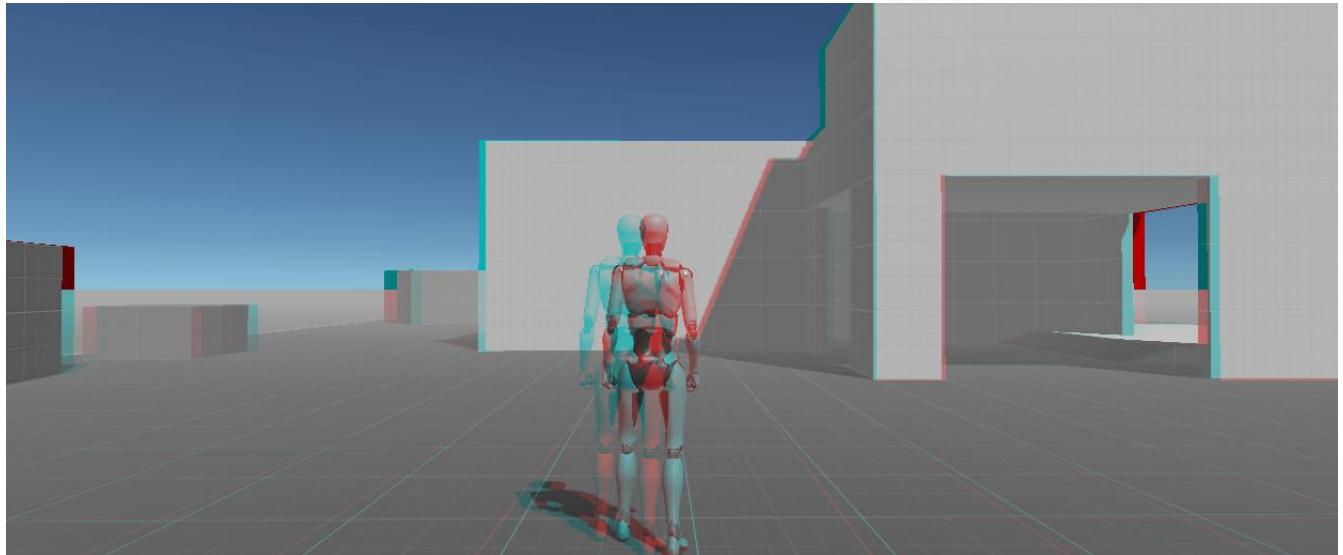


FIGURE 2.11 – Fonctionnement de l'anaglyphe en version 3

Par contre, le problème de perte de l'effet au contact avec la **skybox** persiste. Selon ce que je peux observer, l'objet est rendu invisible tandis que deux copies, une rouge et une cyan, sont décalées et affichées. Quand l'objet s'intersecte avec le ciel, l'effet est perdu à cause des couleurs inversées et d'une troisième figure apparaît,

celle de base qui est normalement invisible. Ce problème est documenté sur la page du package [rya22] et serait dû au fonctionnement des `skybox` sous Unity.

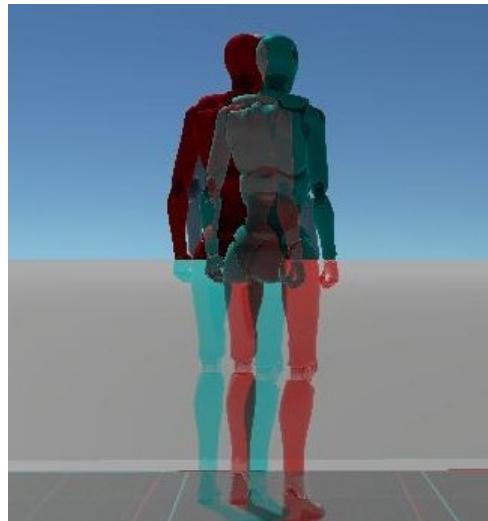


FIGURE 2.12 – Bug de perte de l’effet par rapport à la `skybox` (visible sur le coude gauche)

L’implémentation de l’effet est de bas niveau et demande des connaissances sur le fonctionnement de l’URP qui dépassent les miennes. Vu le temps réduit, je me suis contenté d’utiliser l’effet tel quel, mais de me restreindre à couvrir la `skybox` autant que possible pour le garder fonctionnel dans le contexte du jeu.

2.3 Billboards 3D

Dans le même but que l’anaglyphe, j’ai essayé de répliquer plusieurs vidéos sur youtube afin d’obtenir un effet de *billboard* anamorphique.

Il s’agit de vidéos affichés sur un écran incurvé qui donnent l’impression que les objets en sortent, ce qui n’est visible qu’à partir d’une position fixe du public. La technique est bien particulière, comme la vidéo effective est plus petite et que le reste de l’espace est rempli avec les couleurs capturées à partir des objets autour du *billboard*. Je pourrais décrire plus sur cet effet, mais comme une image vaut mille mots, je cite un exemple parlant sur youtube. [Sto22]

J’ai créé une nouvelle scène pour essayer toutes les ressources dont je disposais.

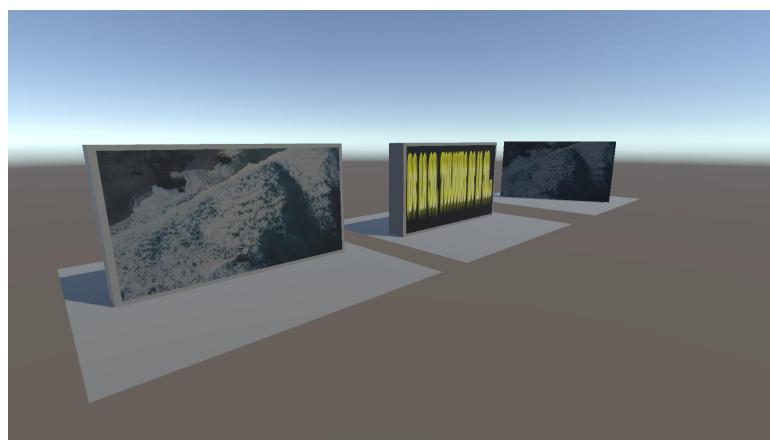


FIGURE 2.13 – La scène avec des essais de billboards

Ainsi, j'ai tenté quatre techniques différentes :

1. La première vidéo [Gam21] décrit comment placer une vidéo dans la scène en la collant à un objet 3D en tant que texture "dynamique". Il s'agit d'un simple glisser-déposer du fichier sur un objet, notamment un plan, et d'ensuite le paramétriser et l'utiliser dans la scène tel quel.
2. La deuxième technique [Ame21] montre comment animer des textures pour tourner en boucle comme les vraies publicités, mais en utilisant des **shaders**. J'ai suivi le tutoriel qui créait un graph avec des noeuds prédéfinis, mais dans ma version, l'un d'entre eux ne se comportait pas comme prévu. Plus précisément, une texture était censé bouger après avoir fait un lien avec une variable de "vitesse", mais pour moi, c'était statique. Il s'agit aussi d'un type différent de graph (qui est considéré son équivalent) comme la version d'**Unity** utilisée dans la vidéo ne correspondait pas parfaitement à la mienne.
3. Pour la troisième version [Tod23], j'ai tenté de faire une texture suivre la caméra peu importe son orientation. Ceci ne marchait pas bien avec les plans, comme ils restaient toujours orientés à plat par rapport à la caméra, mais j'ai trouvé une méthode encore plus simple dans les commentaires. Il s'agit d'une composante **Unity** qui effectue tout ce travail sans souci d'orientation - "*Look At Constraint*".
4. La dernière méthode [tbaA22] réplique effectivement le fonctionnement d'un vrai *billboard* anamorphique dans Blender. Le souci est que je n'ai pas assez d'expérience avec Blender, et donc, j'ai dû chercher chaque raccourci de clavier (aucun n'était marqué). Après 10 minutes de recherche pour environ 2 minutes de travail dans la vidéo, j'ai tout remis à zéro par erreur et je ne voyais pas de raison à continuer avec dans le peu de temps qui me restait.

J'ai finalement tenté de trouver plus de vidéos pertinentes, mais je n'en ai pas pu trouver davantage.

Pour faciliter le mouvement de la caméra et vérifier le fonctionnement dynamique des différents billboards dans la scène, j'ai réutilisé un **script** que j'avais créé pour un ancien stage : **Smooth Mouse Look**.

Nous venons de voir les techniques 3D choisies, ainsi que l'état d'avancement de chacune. A la suite, nous allons voir comment j'ai pu créer une démo en utilisant celles qui m'ont semblé les plus adaptées.

Chapitre 3

Application

Dans ce chapitre, je présente le jeu vidéo que j'ai conçu en utilisant l'anaglyphe et les billboards. Il s'agit d'un niveau de casino que j'ai créé lors d'une recherche sur les différentes caractéristiques d'un *platformer* et d'une étape de collecte d'idées.

3.1 Recherche

Pour effectuer la tâche finale, j'ai commencé en me posant la question de ce que c'est vraiment un *platformer*. A l'aide de plusieurs recherches en ligne, de chatGPT et de l'expérience personnelle, j'ai pu trouver les caractéristiques suivantes de ce type de jeu. Je présenterai également une étude de cas avec des exemples de trois séries de jeux de plateforme qui sont représentatifs pour le genre : les jeux "Mario" (par Nintendo), les jeux "Crash Bandicoot" (par Naughty Dog et Activision) et les jeux "Psychonauts" (par Double Fine).

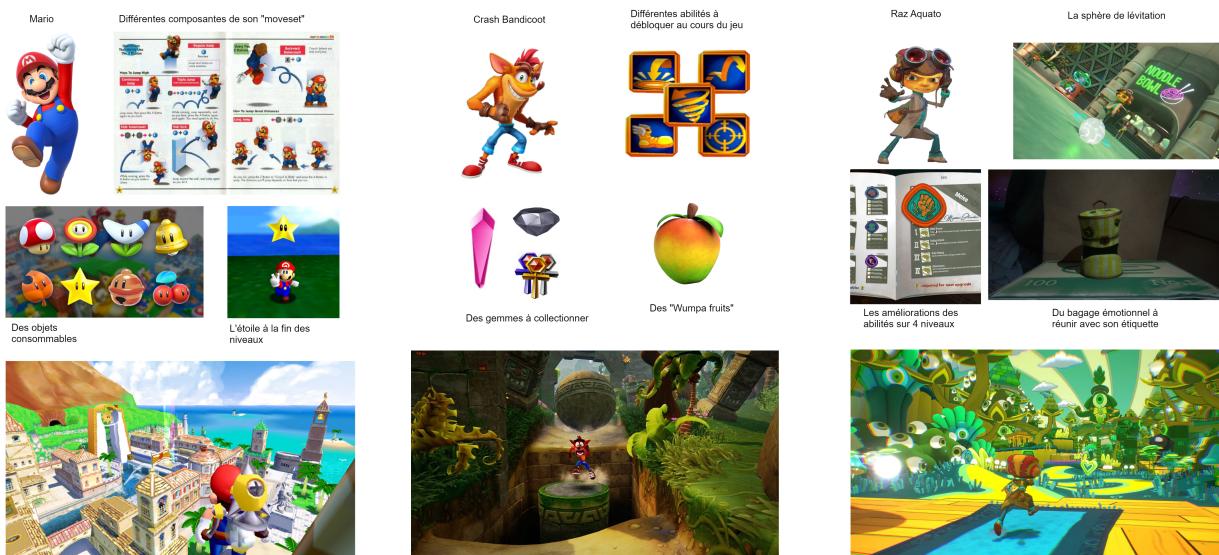


FIGURE 3.1 – Des caractéristiques des différents jeux décrits

1. **Le personnage principal est souvent une "mascotte"**, il s'agit d'un personnage avec des caractéristiques physiques qui le mettent en évidence, des caractères qui dénotent souvent du courage et de la confiance en soi. Il donne aussi souvent le nom de la série.

- Mario est souvent heureux et prêt à sauver ses amis. Il est petit, avec des caractéristiques physiques arrondies pour le rendre plus mignon.
 - Crash Bandicoot est un marsupial avec un grand sourire, des yeux très expressifs et des vêtements humains, ce qui le rend spécialement remarquable.
 - Razputin "Raz" Aquato est un enfant de 10 ans avec des caractéristiques bien exagérées : une tête rectangulaire presque entièrement remplie par les yeux et avec une casque en cuir et des lunettes rouges rondes par dessus. Il est souvent enthousiasmé et essaie de prendre ses devoirs au sérieux, même s'il cela le rend souvent embarrassé.
2. **Il y a un moveset expansif**, un ensemble d'animations que nous pouvons utiliser telles quelles et/ou combiner afin d'arriver à certains points d'intérêt. Dans certains cas, l'environnement peut donner plusieurs possibilités de mouvement pour effectuer un trajet, ce qui laisse la liberté à l'utilisateur et crée un élément de pensée critique qui rend l'expérience plus immersive et moins automatisée. Un moveset de base inclut la marche et la course dans toutes les directions et le saut.
- Mario présente différentes animations spécifiques à certains de ses jeux : la rotation en l'air, le triple saut, le saut long etc.
 - Crash peut également faire une rotation, mais par le sol et dans tous ses jeux.
 - Raz a des abilités qui lui servent également au moveset, par exemple la sphère de lévitation qui lui évite de toucher la terre électrifiée ou qu'il peut utiliser comme un ballon pour descendre en toute sécurité sur des distances plus longues. Il peut aussi utiliser la "connexion mentale" comme un grappin afin d'arriver à des endroits plus hauts.
3. **Il y a des abilités qui peuvent être modifiées**. Les modificateurs peuvent augmenter (ou parfois même diminuer) les pouvoirs du personnage.
- Mario en a plusieurs sous la forme de différents objets consommables comme des champignons ou des fleurs : il peut être transformé en abeille et voler, devenir géant, devenir invincible et éliminer tous les ennemis en les touchant etc.
 - Crash adopte une philosophie différente où les abilités sont gagnées une fois et retenues pendant le reste du jeu : la triple rotation, les chaussures de vitesse, un lance-roquette qui lance des fruits etc.
 - Raz a plusieurs "*PSI Powers*" qu'il gagne progressivement dans le premier jeu et qui sont liées au thème de la psyché : la pyrokinésie, la télékinésie, l'invisibilité etc.
4. **Il y a des objets à collectionner**. Un sous-genre important des *platformers* est le *collectathon*, c'est-à-dire les jeux qui se concentrent sur la collecte d'ensembles d'objets autour de l'environnement à des fins spécifiques :
- Dans les jeux "Mario", chaque niveau nous demande de collecter l'étoile/le soleil/la lune qui marque la fin. Selon le jeu, il y a aussi 8 pièces rouges cachées dans le niveau ou 3 étoiles vertes en plus.
 - Crash a certains niveaux qui requièrent des gemmes à trouver dans d'autres niveaux.
 - Psychonauts a une pléthore d'objets à collectionner : des figurines qui représentent des idées du cerveau dans lequel se passe le niveau, les "pépites de sagesse", les bagages émotionnels (qui sont effectivement des bagages) et les "vestiges de mémoire" parmi autres. Les vestiges de mémoire sont des coffres à attraper qui contiennent des images avec des moments additionnels du passé.
5. **Il y a des systèmes de progression**. La progression est l'évolution des capacités offertes au joueur, comme la marche plus rapide, le coût diminué des matériaux ou une plus grande distance de détection des ennemis.
- Dans les jeux "Mario", il n'y a pas de tels systèmes permanents, mais l'évolution se voit dans l'apparition de nouveaux niveaux ou, selon le cas, de nouvelles chambres à explorer à l'aide du nombre total d'étoiles ramassées. Pour chaque 100 pièces en or, Mario reçoit également une nouvelle vie.
 - Crash a un système classique par niveaux aussi, mais il y a des modificateurs permanents décrits dans la partie sur les abilités. Comme dans les jeux "Mario", pour chaque 100 fruits ramassés, Crash reçoit une nouvelle vie.
 - Psychonauts dispose d'une version basique d'un arbre de compétences, où l'on peut choisir quels aspects améliorer en fonction de la quantité courante de crédit. Les niveaux sont mieux intégrés dans l'histoire et se passent majoritairement dans les psychés des personnages principaux.
6. **L'exploration est le but principal**. Les *platformers* sont des jeux de découverte, où les abilités doivent être enchaînées et combinées afin de passer des obstacles qui peuvent cacher des objets à collectionner ou des détails de l'environnement qui complètent l'histoire.

- Les jeux "Mario" contiennent beaucoup d'environnements différents, souvent avec des chemins secondaires et des secrets cachés partout.
- Les jeux "Crash" ont moins de variété et sont des jeux assez linéaires vu leur nature limitée avec beaucoup de niveaux avec une trajectoire de la caméra fixée. Les objets sont souvent cachés en dehors du plan de vue selon un axe non-utilisé par la caméra.
- Les deux "Psychonauts" contiennent des zones ouvertes avec beaucoup d'objets cachés, souvent dans des coins ou des plateformes moins visibles où l'on peut trouver beaucoup d'objets à collectionner ainsi que des détails de l'environnement qui complémentent le cadre de l'histoire.

3.2 Idées

Une fois l'objectif fixé, j'ai commencé une étape de *brainstorming* pour trouver l'identité du jeu que j'allais créer. Vu que je ne dispose pas d'un département artistique, je me suis limité à utiliser les *assets* que j'allais trouver sur Internet gratuitement. Ainsi, l'*Asset Store* d'*Unity* en avait pas mal, mais il m'a semblé assez limité en modèles gratuits. Je connaissais aussi deux sites qui m'ont beaucoup servi : Mixamo [[Ado](#)] fait par *Adobe* et qui offre des modèles de personnages et des animations gratuites qui sont compatibles avec *Unity* et Sketchfab [[Ske](#)] qui offre beaucoup de ressources variées, dont un grand nombre sans frais.

Cela dit, j'ai trouvé plusieurs idées qui m'ont semblé faisables au départ :

- Pour trouver le nom du personnage, j'ai créé un jeu de mots à partir de l'anamorphose et je suis arrivé sur Murphy (à partir de Morphy) ;
- Vu que la technique anamorphose est employée, le personnage pourrait se transformer dans d'autres personnages comme mécanique de jeu (à partir du mot grec *metamorphosis*, qui se traduit par "transformation") ;
- Parmi les modèles de Mixamo, ceux qui m'ont semblé les plus adaptés ont été celui d'un mannequin de magasin et celui d'un acteur en costume de *motion capture*.
- Afin d'utiliser l'anamorphose statique, j'ai eu l'idée d'en créer avec des codes pour ouvrir des portes ou des coffres. Bien sûr que ces codes ne seront visibles qu'à partir d'un certain angle ;
- Les différents billboards peuvent servir de décor ou être inclus dans les mécaniques du jeu afin d'en extraire des informations utiles à la résolution des défis. Une alternative serait d'y afficher des conseils pour que l'utilisateur démarre bien le jeu (ce que fait chaque touche, comment les animations se combinent, quel outil utiliser pour quel but etc.) ;
- Un outil peut servir à plusieurs buts. Par exemple, une montre connectée qui sert de menu, inventaire, arme et de grappin.
- Un *moveset* compréhensif incluerait au moins la marche, la course, le saut et l'accroupissement.

Après avoir considéré le temps restant et les ressources dont je disposais, j'ai créé le concept suivant :

- Le jeu s'appelle "Murphy at the Casino".
- Le protagoniste sera le modèle "*The Boss*" de Mixamo, qui ressemble à un chef de mafia, avec des caractéristiques exagérées d'une caricature.
- Il est composé d'un seul grand niveau au sein d'un casino fermé afin d'éviter que l'anamorphose ne soit perçue correctement.
- Vu l'état des deux types d'anamorphose, j'ai choisi d'utiliser l'anaglyphe et les billboards en tant que techniques 3D.
- L'anaglyphe est la mécanique centrale qui crée la dualité entre le monde normal et le monde filtré par les lunettes rouge/cyan. Il y a des objets statiques qui persistent dans les deux mondes et des objets spécifiques à chacune des vues.
- Le but principal est de ramasser des objets sur plusieurs niveaux :
 1. Il y a 52 cartes de jeu qui créent un paquet organisé par symboles. Chacun des modèles de carte de la scène est différents pour représenter chacune des cartes de façon unique.
 2. Pour chaque paire de 13 cartes du même symbole, le symbole apparaît en fin d'un défi et il est également à ramasser.
 3. Une fois les 4 symboles collectés, le jeu est fini. La fin est marquée avec un "prix" - une animation qui remet les cartes dans un paquet afin de marquer la collection complète.
- Il y a de multiples interactions qui seront intégrées à l'environnement afin de rendre l'exploration moins dirigée.

- L'interface est dynamique par rapport à l'état d'avancement dans le jeu, tout en restant minimaliste. Chacun des éléments disparaît une fois que leur but est atteint.

3.3 Implémentation

Après avoir passé la plupart de la période de stage concentré sur comment faire fonctionner l'anamorphose, je n'ai disposé que d'une semaine et demi pour créer le jeu sous la forme d'une démo d'un seul niveau afin de maximiser le nombre de systèmes que je puisse rajouter à l'expérience. Ainsi, j'ai créé plusieurs scènes : un intérieur de casino, une scène extérieure et un menu avec un générique de fin.

3.3.1 Intérieur

Importation de modèles

J'ai réalisé vite que j'allais importer différents types d'objets 3D dans Unity, alors j'ai dû trouver des moyens pour me simplifier la tâche de conversion. J'ai trouvé un package qui s'appelle GLTFUtility [Sic19] et qui importe des modèles 3D sous forme de fichier glTF, un format bien commun parmi les différents assets de Sketchfab.

J'ai rencontré deux problèmes récurrents en important des modèles :

- les modèles étaient blancs à cause des textures qu'il fallait convertir pour fonctionner avec l'URP (à l'aide d'une simple option d'Unity) ;
- les textures n'étaient plus liées aux modèles, ce qui m'a demandé d'utiliser l'option d'extraction des textures.

Personnage principal

J'ai téléchargé le personnage de Mixamo et j'ai suivi un tutoriel pour remplacer le modèle du template sans enlever des fonctionnalités. Alors, j'ai enlevé l'ancien modèle (objet enfant d'un ensemble d'objets qui définissent le personnage), j'ai rajouté le nouveau en tant que nouvel objet enfant et j'ai remplacé l'"avatar" de la composante Animator afin de rendre l'animation de ce nouveau squelette possible.

Pour rajouter un détail qui rendrait le jeu plus vérifique, j'ai rajouté une paire de lunettes rouge/cyan au personnage (aussi trouvées sur Sketchfab) et elles ne sont visibles que quand l'anaglyphe est activée. J'ai eu quelques soucis pour trouver l'objet qui correspond au mieux à la tête pour le rendre parent des lunettes afin que les mouvements soient synchronisées, mais ceci a été corrigé.

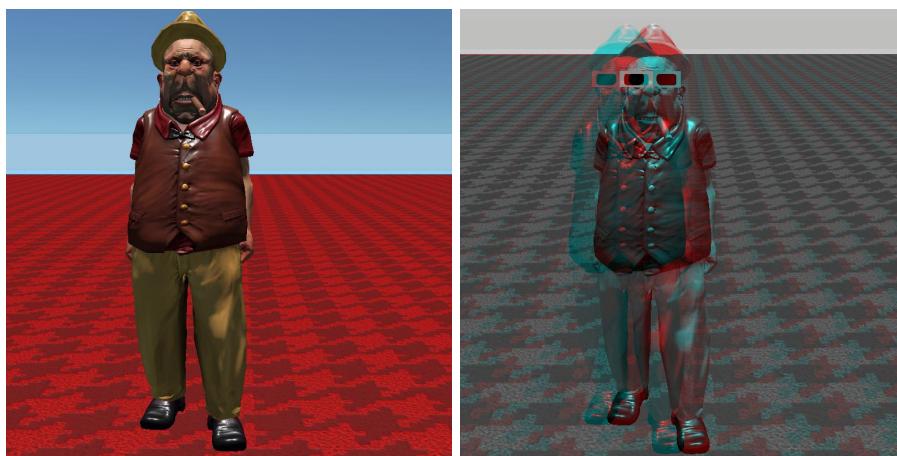


FIGURE 3.2 – Personnage principal sans/avec les lunettes rouge/cyan

Objets de collection

J'ai commencé avec la partie intégrale de l'expérience, trouver les objets à collectionner et les rendre interactibles. Ainsi, j'ai trouvé un modèle avec toutes les 52 cartes sur Sketchfab et je l'ai importé dans *Unity*. J'ai passé du temps à renommer chaque carte avec le numéro et le symbole correspondant et à les organiser sous un même objet parent. Ceci m'a donné la possibilité de les séparer par symbole pour faciliter le comptage des objets collectionnés et d'effectuer des changements sur chacun des objets par le code. J'ai également trouvé les quatre symboles (coeur, carreau, trèfle et pique) en tant que modèles sur Sketchfab.

Pour collectionner les objets, chacun doit contenir un **Collider** qui sera détecté par la boucle de rendu d'*Unity* en contact avec la composante **Character Controller** du personnage principal.

Pour ceci, j'ai rajouté la fonction `OnControllerColliderHit` dans le **script** *Third Person Controller* prédéfini par le **template**. A chaque détection de carte, un son est joué, l'objet est désactivé, il est compté et un son différent joue si les 13 cartes du même symbole sont ramassées. Pour que chacune des cartes ait les mêmes propriétés, j'ai créé un **script** *CardsProperties* qui rajoute une composante **Box Collider** et une étiquette correspondant au symbole sur chacune des cartes. Pour rendre les cartes plus visibles, j'ai créé un **script** *CardRotation* pour les faire tourner autour de leurs axes verticaux. Le même traitement est appliqué aux quatre symboles à ramasser, mais avec un son différent.

Le son des symboles est un peu bas, mais je n'ai pas eu le temps de chercher plus en détail sur comment le corriger vu que le paramètre de volume de la fonction `AudioSource.PlayClipAtPoint` ne le dépassait pas. J'ai également voulu rajouter des lumières au-dessus de chacune des cartes, mais je n'ai pas trouvé de version satisfaisante par le code.

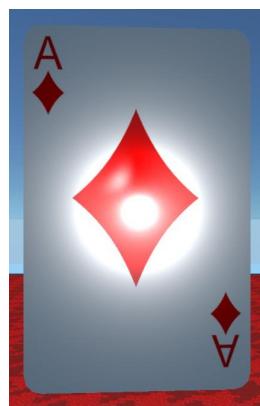


FIGURE 3.3 – Effet d'une lumière rajoutée aux cartes

Système d'interaction

Pour donner à l'utilisateur plus d'interactivité et de possibilités d'extraire des détails de l'espace que j'allais créer, j'ai décidé d'inclure un système d'interaction (selon un tutoriel sur youtube [Pos22]) qui crée une classe *Interactor* qui sera attribuée au joueur. Réellement, ce qui se passe est que nous attribuons une sphère au joueur ; à chaque collision entre la sphère et un objet concerné, un message est affiché dans l'interface et si l'on appuie sur la touche "E", une action arbitraire aura lieu. Chacun des objets que l'on veut rendre "dynamique" possédera un **script** qui va implémenter une interface **IInteractable** en définissant un comportement propre à lui et un **collider**. Un objet interactif sera reconnu à l'aide de son **Layer** "*Interactable*". Il existe également un **script** qui contrôle l'UI afin de l'afficher au bon moment et de la faire tourner vers la caméra à tout moment.

Le seul souci que j'ai eu pour cette partie a été le fait que j'ai confondu le **Layer** avec le **Tag** de l'objet, mais je l'ai appris assez vite pour pouvoir rajouter d'autant plus d'objets interactifs dans la scène.

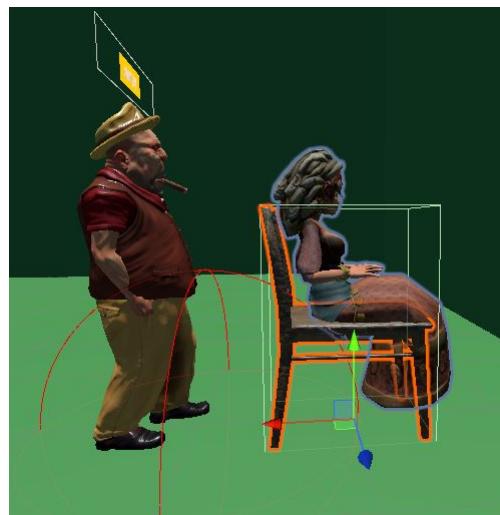


FIGURE 3.4 – La sphère (rouge) intersecte le **collider** (vert) de la chaise afin d'afficher l'UI (rectangle jaune).

Pour les fenêtres de dialogue, j'ai conçu un ensemble d'images et de fenêtres de texte qui forment un panneau paramétrable par le **script Text Scroller** pour chaque intervention :

- Titre/Nom du personnage ;
- Corps de texte ;
- Bouton et instruction ;
- Nombre de la page ;
- Un tableau de **string** à afficher un à un.

Chacun des textes change sa taille dynamiquement afin de remplir au mieux l'espace de la case qui lui est allouée.

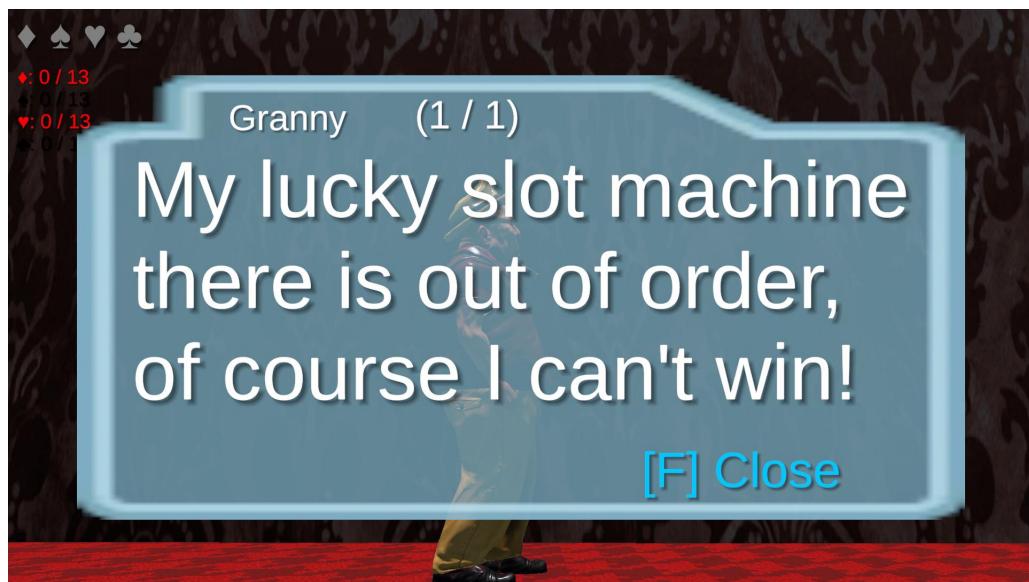


FIGURE 3.5 – Fenêtre de dialogue

Interface

J'ai rajouté une interface minimalist composée par cinq lignes de texte :

- une ligne avec les quatre symboles : au début, ils sont coloriés en gris ; une fois les 13 cartes d'un même symbole ramassées, le symbole respectif devient rouge/noir ; une fois l'objet du symbole ramassé, la composante de l'interface qui lui correspond disparaît ;

- quatre lignes : un compteur pour chacun des symboles, pour montrer combien de cartes de chaque catégorie ont été collectées sur 13. Une fois qu'un ensemble est complété, la ligne qui lui correspond se désactive aussi.

Le problème le plus important que j'ai rencontré a été de garder le ratio des éléments d'UI pour différentes résolutions. Ceci a été possible grâce à l'option UI Scale Mode mise à *Scale With Screen Size*.

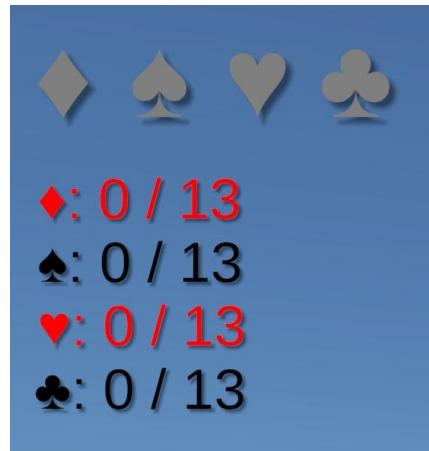


FIGURE 3.6 – L'interface au début du jeu

Les quatre puzzles

En utilisant les différentes techniques fonctionnelles que j'avais développées jusque-là, j'ai créé quatre puzzles, à la fin desquels se trouvent chacun des symboles :

1. Une suite de plateformes qui apparaissent et disparaissent alternativement entre le monde normal et sous l'effet de l'anaglyphe. Elles ont nécessité assez de tests pour rendre les plateformes non-triviales sans rendre l'expérience frustrante ;
2. Un labyrinthe de dés qui "regardent" l'utilisateur (en utilisant la `Look At Constraint` trouvée lors de mes recherches sur les *billboards*. J'ai dû faire attention à bien écarter les murs pour former des couloirs accommodés aux mouvements des dés sans laisser de passages ouverts ;
3. Une maison de cartes avec deux chambres, dont une fermée jusqu'à la collecte de toutes les cartes de trèfle ; la chambre initialement fermée contient des cartes suspendues en l'air qui sont enlevées quand l'anaglyphe est activée. J'ai également dû y rajouter des composantes `Box Collider` ;
4. Des cercles concentriques de piles de jetons de poker qui alternent entre le monde réel et le monde anaglyphe. J'ai dû les rapprocher afin de créer des cercles fermés.

Devant chacun de ces défis se trouve un poteau avec un message qui donne un indice sur chacun d'entre eux de façon subtile.

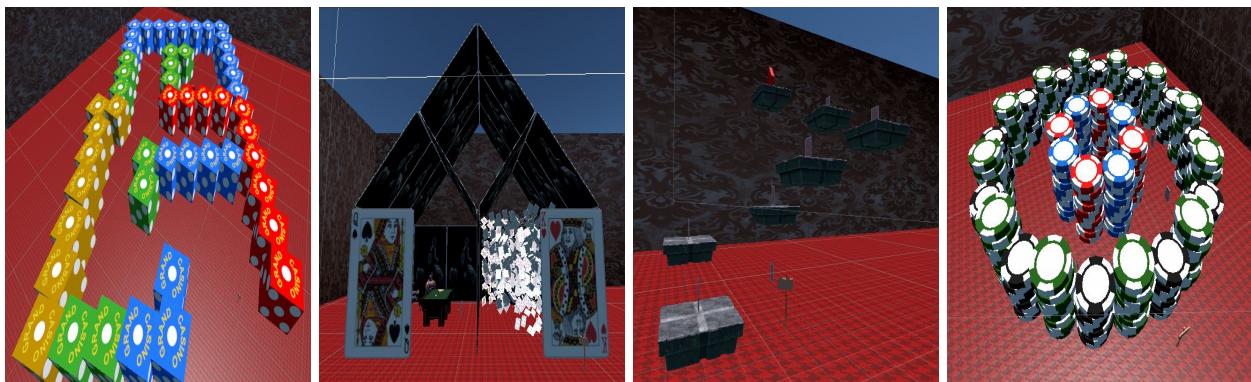


FIGURE 3.7 – Les 4 puzzles (en ordre : labyrinthe, maison de cartes, plateformes et piles de jetons)

Atmosphère

Le son peut changer drastiquement l'attitude du joueur, alors j'ai choisi d'intégrer une musique de fond que j'ai trouvée sur un site libre de droits. Il s'agit d'un morceau très entraînant qui est représentatif pour le thème de casino et que j'ai mis en boucle.

J'ai créé un décor de casino en cherchant divers modèles sur Sketchfab, ainsi :

- J'ai un objet "capsule" qui est un terminal au centre du casino avec un chandelier au-dessus ;
- A gauche et à droite de l'entrée, j'ai un mur de machines à sous avec des chaises ;
- Les murs sont des cubes transformés avec une texture trouvée en ligne ;
- Un matériel rouge est également mis sur le plan du sol pour simuler un tapis de casino ;
- Une table de billards est placée dans la chambre de gauche de la maison de cartes.

J'ai également rajouté une pléthora de NPC (personnages non-joueur) pour populer l'endroit et le rendre moins vide. Chacun a aussi une réplique de dialogue qui lui correspond dans l'esprit de l'autodérision et du ludisme de l'ensemble de personnages proposés par Mixamo. Vu que je ne dispose pas de plusieurs acteurs de voix, j'ai décidé de me faciliter la tâche en utilisant le système d'interaction pour afficher ce que chacun des personnages dit.

Utilisation de l'anaglyphe

L'anaglyphe était initialement utilisable à l'aide de la touche "1", mais en pratique, j'ai trouvé qu'utiliser le bouton droit de la souris était plus confortable.

Pour faire apparaître et disparaitre des objets arbitraires, j'ai créé un script "*ToggleURPFeature*" qui active et désactive deux listes de **GameObject** de façon alternative. Ces listes sont publiques et peuvent être remplies à la main avec des objets de la scène.

J'ai choisi d'y inclure certaines machines à sous décalées vers l'avant, certaines plateformes, les cartes de jeu suspendues et les piles de jetons de poker.

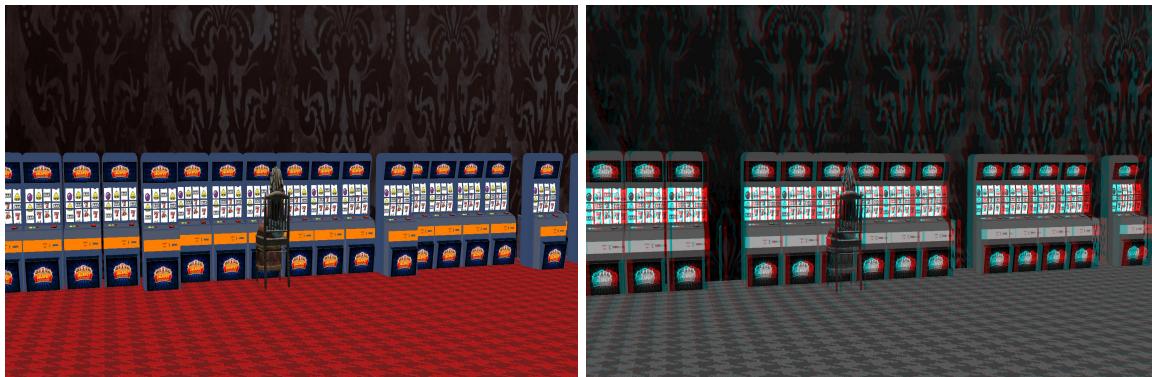


FIGURE 3.8 – Effets de l'anaglyphe sur les machines à sous

La "capsule"

La "capsule" est l'objet central du jeu, il s'agit du modèle "NTGL Just Vegas POD" qui est une sorte de console qui donne les indications sous forme de tutoriel texte. Le joueur y est redirigé au départ afin d'apprendre plus sur le but du jeu, comment utiliser les lunettes anaglyphe et les défis.

Une fois tous les symboles collectés, un message redirige le joueur vers ce repère afin de recevoir un message de félicitations et d'être transféré vers la scène du prix final.

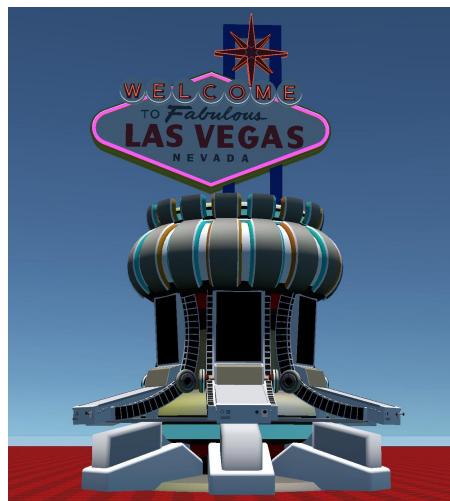


FIGURE 3.9 – La "capsule"

Cacher les objets

J'ai choisi de cacher les cartes de la façon suivante :

- 8 cartes au milieu de la scène afin que le joueur puisse interagir avec pour savoir quel est leur comportement ;
- 12 cartes derrière les machines à sous et trouvables en utilisant l'anaglyphe ;
- les autres partagées parmi les chemins des quatre puzzles pour donner des indices indirects, en faisant correspondre leurs symboles avec le symbole qui se cache à la fin.

3.3.2 Extérieur

La plupart des éléments sont les mêmes que pour l'intérieur. La musique est plus calme, l'interface n'apparaît plus sauf pour celle d'interaction et l'anaglyphe est désactivée, comme les lunettes sont offertes à l'utilisateur par la hôte avant d'entrer.

J'ai choisi une musique différente à fond pour marquer le sentiment de rester à l'extérieur d'une fête bruyante et les NPC ont des répliques liées à l'exclusion de la fête ou de la construction du bâtiment.

Cette fois-ci, j'ai choisi une pyramide avec deux entrées, l'une bloquée par un NPC, l'autre contient un Trigger qui change la scène vers l'intérieur pour recréer l'effet de passer par une porte. La hôte devant la porte libre donne un message de bienvenue, offre les lunettes rouge/cyan et encourage le joueur à aller vers la "capsule".

La pyramide m'a révélé un souci sur le plan qui représentait le sol et qui faisait mon personnage tomber en-dessous. J'avais seulement agrandi la taille d'un composant du plan, mais ce n'était pas celui qui contenait la composante de collision.

La *skybox* a été remplacée pour refléter la nuit et le brouillard bleu caractéristique à Unity sur les objets à distance a été désactivé.

La scène est un long couloir bloqué par des arbres rapprochés qui contiennent des Colliders. J'ai aussi rajouté des tables pour donner plus de contexte aux NPC et une statue entre les deux portes pour rendre l'endroit plus vraisemblable.

J'ai eu quelques soucis en cherchant des modèles d'arbres à cause d'un dysfonctionnement qui rendait les textures visibles (un tel arbre est souvent représenté par un ensemble de surfaces texturées orientées dans tous les sens).



FIGURE 3.10 – La scène extérieure

3.3.3 Prix final

Pour la fin, j'ai voulu trouver une animation qui met toutes les cartes dans un paquet, mais je n'ai eu le temps d'apprendre comment l'utiliser dans Unity. Ainsi, au lieu d'un modèle qui anime les cartes que j'ai trouvé sur Sketchfab, j'ai mis un autre qui représente une figurine avec plusieurs cartes qui essaient de créer une pyramide ensemble.

Je les ai écartées afin de rajouter une carte sur laquelle j'ai marqué un message de remerciement pour le joueur. Afin de marquer cet achèvement, j'ai rajouté un son de gain.



FIGURE 3.11 – La scène finale du prix

3.3.4 Menu principal

J'ai décidé de rendre le jeu plus *user-friendly* en rajoutant un menu qui offre trois possibilités :

- Play - lance la scène extérieure ;
- Credits - lance la scène du générique de fin ;
- Quit - ferme l'application.

Pour ceci, j'ai appris le fonctionnement des boutons sous Unity et comment leur changer l'apparence entre les trois états de base (non-activé, quand la souris est placée dessus et quand il est appuyé) en utilisant un

tutoriel [Bra17]. J'ai ensuite attribué à chacun des boutons une fonction d'un nouveau `script` spécifique en utilisant l'interface spécifique.

J'ai également rajouté un fond d'écran pour rendre le texte plus lisible.

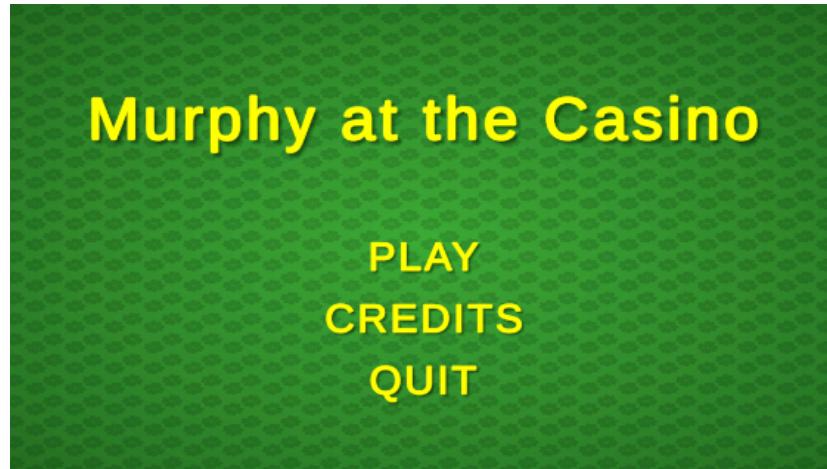


FIGURE 3.12 – Le menu principal

3.3.5 Générique de fin

Tout au long du processus de création de la scène, j'ai gardé de côté tous les liens des modèles que j'ai utilisés, comme ils étaient libres de droits, mais avec la remarque qu'il fallait les mentionner dans les travaux où ils seraient ensuite utilisés.

J'ai trouvé un tutoriel sur YouTube [Veg17] qui montre comment, à partir de deux fenêtres de texte, nous pouvons animer le mouvement de bas en haut progressif du générique en utilisant une composante `Animator`. Il suffit de marquer une position de début et une autre pour la fin au moment souhaité et une interpolation est créée pour lier les deux états de façon uniforme.

J'ai également rajouté un petit `script` qui sorte de la scène vers le menu principal si la fin a été atteinte ou que l'utilisateur a appuyé sur la touche "E" (ce qui est marqué comme possibilité dans le coin en bas à droite). Un autre morceau de musique a été choisi pour accompagner la liste afin de la rendre moins monotone.

La partie la plus longue a été la transcription des noms des modèles, des auteurs et des liens afin de donner des références uniformément complètes. Je les ai également organisés par catégories (3D, 2D, audio).

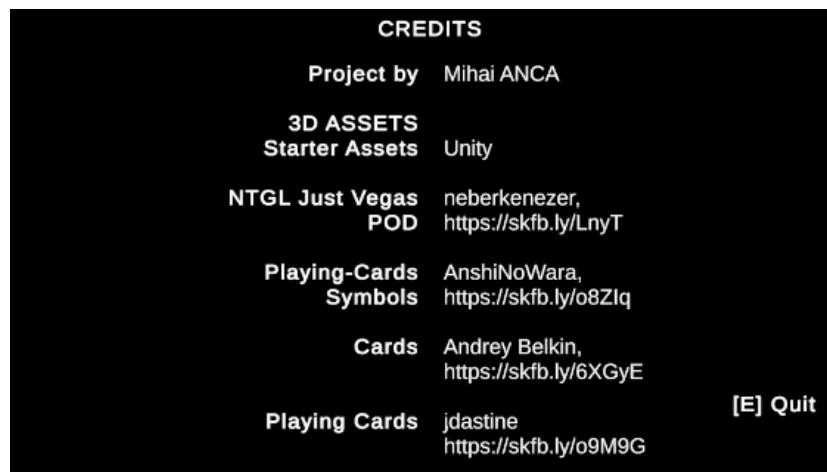


FIGURE 3.13 – Le générique de fin

3.4 Résultats

Après ces étapes, j'ai créé un *build* du jeu pour Windows comme plateforme cible de base afin de tester son fonctionnement en plein écran. Une fois que les éléments d'interface m'ont semblé bien positionnés et le jeu gagnable, j'ai regardé les options que j'avais pour paramétriser le logiciel (le nom de l'auteur, l'icône, s'il peut être joué en plein écran et à quelle résolution etc.).

Afin de rendre le jeu jouable plus facilement, je me suis rappelé qu'Unity offre la possibilité de publication gratuite des *builds* effectués en WebGL sur leur site, alors je m'y suis lancé. Après avoir installé le plugin requis et changé de plateforme, j'ai également dû réintégrer le **template** suite à un problème d'incompatibilité et recopier mon code du **script Third Person Controller**. Le manque de certains fichiers faisait que la souris ne fonctionnait plus correctement et que le personnage ne pouvait marcher que dans les directions dictées par les boutons du clavier.

J'ai choisi une option qui effectue une forte compression des données afin de rendre le logiciel au plus léger possible (les données affichées sur une page web ne peuvent pas dépasser les 2 Go). Après plus de 20 minutes de compilation, j'ai eu un logiciel de moins de 200 Mo, ce qui m'a permis facilement de publier ceci sur le site Unity Play :

<https://play.unity.com/mg/other/webgl-build-1097>

Pour le tester, j'ai créé un *Google Form* que j'ai publié plusieurs fois sur les différentes communautés autour d'Unity sur reddit, mais je n'ai malheureusement pas eu de retour malgré les centaines de vues.

The form contains five sections:

- Have you enjoyed your experience with it? ***: A 5-point Likert scale from "Not at all" to "Absolutely". The "3" button is selected.
- What did you like about it? ***: A text input field labeled "Your answer".
- What didn't you like about it? ***: A text input field labeled "Your answer".
- Have you encountered any bugs or inconsistencies? ***: A text input field labeled "Your answer".
- What could be improved (and how)? ***: A text input field labeled "Your answer".

FIGURE 3.14 – Les questions du formulaire envoyé sur reddit

Entre temps, j'ai également observé que la version web garde un bug qui fait qu'en mode plein écran, la souris reste visible et fonctionnelle uniquement jusqu'aux bords, mais elle ne peut pas les dépasser afin de faire tourner la caméra dans toutes les orientations possibles.

La solution que j'ai trouvée est qu'il faut sortir du mode plein écran et s'y remettre afin que le pointeur disparaisse et que le mouvement s'effectue correctement.

Quand j'ai voulu mettre cette nouvelle version sur mon dépôt [Github](#), j'ai eu une erreur que j'avais eu avant sur un fichier qui dépassait la limite imposée de 100 Mo. J'ai voulu le supprimer du commit, mais en utilisant l'interface *desktop*, j'ai supprimé le travail que j'avais effectué sur les génériques de fin. Au moins, je les avais publiées en version WebGL. Pour les récupérer, je les ai enregistrés en utilisant un outil de capture vidéo d'écran et j'ai mis le résultat dans le dépôt du projet.

Enfin, pour avoir une version enregistrée du *gameplay*, je me suis enregistré une session de la version antérieure dès le début et jusqu'à la fin et je l'ai postée sur YouTube avec la vidéo des génériques collée à la fin (un parcours du jeu direct faire dure 6 minutes environ) :

<https://www.youtube.com/watch?v=r8BiLExZjMI>

Nous avons vu le processus de développement de la démo des techniques 3D sous la forme d'un jeu vidéo de plateforme, jouable sur plusieurs formes (Windows et dans le navigateur). Ceci marque la fin de stage avec un produit utilisable et encore paramétrable.

Conclusion

Suite à ce stage, j'ai appris à travailler avec la géométrie 3D de bas niveau dans **Unity**, à mieux rechercher des informations et à organiser mon travail de façon plus efficiente. J'ai compris davantage sur les différentes techniques de production d'effets 3D sur des écrans 2D, ce qui vient en complément des connaissances déjà acquises dans l'UE de PdP de ce semestre.

Ainsi, j'ai créé une démo d'un jeu qui utilise une technique 3D nécessitant du matériel à un prix dérisoire. J'ai également réussi apprendre plus sur la technique anamorphique en essayant de l'implémenter. Ceci m'a également demandé d'apprendre plus sur les boîtes englobantes et la détection des visages. L'une des tâches la plus difficile a été l'installation et les liaisons de bas niveaux entre les différents fichiers d'**OpenCV**.

J'ai enfin pu utiliser les bases que j'ai acquises en **Unity** lors du dernier stage et crée un jeu que je considère comme finalisé. Sous la forme actuelle, il est jouable dans le navigateur et un tutoriel est également fourni sur YouTube.

Je compte utiliser les nouveaux acquis de ce stage comme base pour le chemin professionnel que j'envisage effectuer dans le domaine des jeux vidéo, en apprenant plus sur les effets 3D et sur la philosophie de création d'un jeu. J'envisage un avenir très prometteur en ce qui concerne les nouvelles expériences 3D et la disponibilité des technologies requises au grand public.

En tenant compte de l'état d'avancement de chacune de mes tâches, j'ai compilé des listes d'améliorations possibles pour chacune afin de montrer qu'avec plus de temps et de ressources, le projet courant aurait pu avancer encore plus.

Je considère qu'un grand facteur décisif sera représenté par la normalisation de ces méthodes et l'avancement dans les domaines de la 3D afin d'avoir encore plus de méthodes et de technologies disponibles en accès libre. Il y a beaucoup de potentiel de voir ce domaine se développer vite.

Anamorphose statique

- La méthode devrait être corrigée ou changée avec une autre.
- Il faut effectuer plus de recherche sur les boîtes englobantes et leurs propriétés géométriques afin d'en créer une avec la sûreté de l'implémentation.
- Il serait envisageable de trouver un moyen de pré-calculer les normales pour tout maillage afin de rendre tout objet compatible avec la méthode.
- Il peut y avoir de multiples utilisations dans le jeu sous la forme d'une mécanique de jeu ou juste pour donner des détails environnementaux. Du texte pourrait être caché sous forme anamorphosée sur les murs, des objets pourraient créer encore une dualité entre le monde normal et anamorphosé ou des concepts plus créatifs pourraient être inclus avec, comme la géométrie non-euclidienne (voir la vidéo [Cod18]).

Anamorphose dynamique

- L'installation d'`OpenCV` devra être corrigée, éventuellement à l'aide d'un forum ou d'une question ciblée à l'auteur du tutoriel utilisé.
- Une autre méthode de détection pourrait être trouvée, possiblement parmi les `packages` d'`Unity`.
- Si le problème de la carte de profondeur est résolu, la méthode décrite dans l'article pourra être implémentée.
- Avec plus d'expertise, le dépôt `git` qui l'implémente pour les casques AR pourrait être adapté.
- Une fois la version qui détecte le visage fonctionnelle, une amélioration qui prend en compte la position et l'orientation des yeux pourrait être mise en place.
- La vue dynamique offerte par cette méthode peut donner la possibilité de cacher des éléments de l'environnement proche des bords de la vue afin que l'utilisateur puisse inspecter le reste en utilisant sa tête comme modalité de contrôle du jeu.

Anaglyphe

- Le bug de la skybox devra être corrigé, soit par le créateur du dépôt, soit par moi-même si j'arrive à gagner de l'expertise dans ce domaine.
- Sinon, une autre implémentation devra être trouvée ou créée.
- Il y a plusieurs types d'anaglyphe, il pourrait y en avoir plusieurs qui soient inclus séparément. (voir [3dt])
- Plusieurs utilisations de l'anaglyphe en tant que mécanique de jeu pourraient être trouvées.

Billboards 3D

- Plus de vidéos et de méthodes de production des *billboards* anamorphiques pourraient être recherchées.
- Avec du temps, j'aurais pu apprendre les bases (et les raccourcis) de `Blender` afin de suivre la vidéo initiale.
- Avec plus de recherche et d'aide de la part des forums, j'aurais pu retenter le graph de `shader` qui n'a pas fonctionné au départ.
- Il y a de multiples usages possibles dans le jeu, à partir du décor urbain jusqu'à une méthode alternative de faire passer des messages depuis l'environnement et sans utiliser des cases de texte qui couvrent l'UI.

Jeu de plateforme 3D

- Le logiciel pourrait être déployé sur plusieurs plateformes et en plusieurs langues.
- Une sortie physique pourrait être envisagée avec une paire de lunettes rouge/cyan incluse.
- Du soin particulier pourrait être apporté à l'accessibilité (taille et couleur des polices, filtres photos pour les différents défauts visuels, des descriptions audio etc.)
- Un menu compréhensif d'options pourrait être créé.
- Les manettes pourraient être supportées, avec le choix des touches.
- Des effets de *post-processing* comme l'anti-aliasing ou un effet de *film grain* rendraient l'image plus stable et lui offriraient un aspect particulier (de film).
- Plusieurs niveaux pourraient être conçus : un plateau de tournage de film, une table avec des jeux de société dont une tour de Jenga à grimper, une pièce de théâtre avec des projecteurs qu'il faut éviter afin de rester caché etc.
- Une histoire pourrait être développée afin de lier les niveaux ensemble et de créer une motivation de plus.
- Une implémentation de l'idée de base où le personnage se transforme en d'autres personnages pourrait également être réalisée.
- Les NPC pourraient être ensuite animés et avoir plusieurs lignes de dialogue spécifiques.
- Le `moveset` aurait pu évoluer avec plus d'animations spécifiques, mais cela m'aurait demandé plus de temps pour bien comprendre comment le `template` est organisé.

- Des ennemis contrôlés par des IA pourraient être rajoutés pour créer un élément de difficulté (avec un système de barre de vie et des sauts qui peuvent tuer le personnage en fonction de la hauteur).
- Autre que l'anaglyphe, plusieurs filtres différents pourraient être rajoutés avec leurs propres effets sur le monde, comme un filtre sépia qui fait le lien entre le présent et le passé. Un nouveau menu pour choisir le filtre assigné à la touche droite de la souris pourrait y être rajouté. Les personnages pourraient aussi réagir différemment en fonction du mode utilisé.
- Un système d'argent pourrait également être introduit, avec un magasin et des filtres photo à acheter en plus d'autres objets utiles. L'argent pourrait être gagné lors des mini-jeux d'arcade du casino.
- Plusieurs types de *pick-ups* pourraient être rajoutés : de l'argent, des modificateurs, des points de vie, d'endurance etc.
- Des lumières peuvent être rajoutées aux *pick-ups* pour les rendre encore plus visibles.
- Avec un réel budget, je pourrais acheter des assets plus sophistiqués et ne plus me restreindre les recherches créatives pour rendre l'espace mieux décoré et l'expérience plus immersive.
- Avec plus de temps, j'aurais lancé un appel à mon réseau de connaissances, au sein de l'université et sur plusieurs sites pour recueillir des avis des joueurs. La création de pages dédiées sur les réseaux sociaux seraient aussi envisageables.
- Les puzzles peuvent encore être améliorés avec plus de difficulté : un labyrinthe plus long et sinueux, des plateformes plus difficiles, plus de cercles et/ou une autre forme des piles de jetons de poker et un puzzle plus difficile pour la maison de cartes.
- Le seul bug que j'ai repéré est la rotation de certaines cartes du labyrinthe par rapport à un point commun au lieu de leurs propres axes.
- Le code pourrait toujours être amélioré est mieux factorisé : les interactions de la classe *Third Person Controller* avec les cartes et les symboles seraient mieux placées au sein du système d'interaction, les fenêtres de dialogues pourraient être généralisées afin d'en créer une à partir de plusieurs **string** au lieu de copier-coller l'objet-parent depuis la fenêtre de l'**Inspector** et changer ses enfants individuellement.
- Une meilleure hiérarchisation des *pick-ups* et des autres objets en général aiderait sur la lisibilité et la logique du code.
- Un menu pourrait aider à mieux comprendre quelles cartes ont été collectionnées par le joueur. Celui-ci contiendrait des *sprites* de chacune des cartes posés sur une grille de 4×13 . Chacun serait initialisé grisé et ensuite colorié une fois que l'objet soit collectionné.

Bibliographie

- [3dt] 3dtv.at. Anaglyph methods comparison. https://3dtv.at/Knowhow/AnaglyphComparison_en.aspx. Accédé le 7 juillet 2023.
- [Ado] Adobe. Mixamo. <https://www.mixamo.com/#/>. Accédé le 20 juin 2023.
- [Ame21] Mubasher Amer. How to make animated billboard or texture animation shaders in unity 3d || mobile optimized - urp. https://www.youtube.com/watch?v=j_G8CNALDU8, 2021. Accédé le 5 juillet 2023.
- [Bra17] Brackeys. Start menu in unity. https://www.youtube.com/watch?v=zc8ac_qUXQY, 2017. Accédé le 7 juillet 2023.
- [Bul11] Silviu Buligan. glasses 3d rubik cube (better than ios 7 parallax effect). <https://www.youtube.com/watch?v=DLplItqq9YA>, 2011. Accédé le 7 juin 2023.
- [Cod18] CodeParade. Non-euclidean worlds engine. <https://www.youtube.com/watch?v=kEB11PQ9Eo8&t=29s>, 2018. Accédé le 7 juillet 2023.
- [Don15] Joe Donnelly. Anamorphosis is a puzzle game with perspective. <https://www.rockpapershotgun.com/anamorphosis-puzzle-game>, 2015. Accédé le 7 juin 2023.
- [Gam21] MetalStorm Games. Unity basics - 3d video billboards inside your game. <https://www.youtube.com/watch?v=M8ZorqPoNSM>, 2021. Accédé le 5 juillet 2023.
- [GP11] Jens Garstka and Gabriele Peters. View-dependent 3d projection using depth-image-based head tracking. 06 2011.
- [HC07] D. Hansford and Daniel Collins. Anamorphic 3d geometry. *Computing (Vienna/New York)*, 79 :211–223, 04 2007.
- [ial19] ialhashim. Densedepth. <https://github.com/ialhashim/DenseDepth>, 2019. Accédé le 14 juin 2023.
- [io19] isl org. Midas. <https://github.com/isl-org/MiDaS>, 2019. Accédé le 14 juin 2023.
- [IOT18] NVIDIA AI IOT. stereodnn. <https://github.com/NVIDIA-AI-IOT/redtail/tree/master/stereoDNN>, 2018. Accédé le 14 juin 2023.
- [kei17] keijiro. Anamorphic lens flare effect for unity. <https://github.com/keijiro/KinoStreak>, 2017. Accédé le 7 juin 2023.
- [McK22] Theodore McKenzie. A neat anamorphic illusion set up in ar with unity. <https://80.lv/articles/a-neat-anamorphic-illusion-set-up-in-ar-with-unity/>, 2022. Accédé le 7 juin 2023.
- [Mou17] Thomas Mountainborn. [tutorial] using c++ opencv within unity. <https://forum.unity.com/threads/tutorial-using-c-opencv-within-unity.459434/>, 2017. Accédé le 8 juin 2023.
- [nia18] nianticlabs. monodepth2. <https://github.com/nianticlabs/monodepth2>, 2018. Accédé le 14 juin 2023.
- [NSF12] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.
- [Pos22] Dan Pos. Unity interaction system | how to interact with any game object in unity. <https://www.youtube.com/watch?v=THmW4Yo1Dok>, 2022. Accédé le 6 juillet 2023.
- [rya21] ryanslikesocool. Anaglyph3d. <https://github.com/ryanslikesocool/Anaglyph3D>, 2021. Accédé le 15 juin 2023.

- [rya22] ryanslikesocool. dark objects 2. <https://github.com/ryanslikesocool/Anaglyph3D/issues/2>, 2022. Accédé le 20 juin 2023.
- [Sic19] Sicc city. Gltfutility. <https://github.com/Sicc city/GLTFUtility>, 2019. Accédé le 7 juillet 2023.
- [Ske] Sketchfab. Sketchfab. <https://sketchfab.com/feed>. Accédé le 20 juin 2023.
- [Sof13] Quaternion Software. Unity3d kinect : "holographic" effect using projection matrix. <https://www.youtube.com/watch?v=KYeTEtxQqVc>, 2013. Accédé le 7 juin 2023.
- [Sto22] Nike Careers : Our Stories. Nike japan's air max day 3d billboard. <https://www.youtube.com/watch?v=N6v7HQiCNIQ>, 2022. Accédé le 5 juillet 2023.
- [tbaA22] Learn to become an Animator. How to create 3b billboard videos on curved screen | 3d billboard | blender tutorial. <https://www.youtube.com/watch?v=chqwnEIfr9Q>, 2022. Accédé le 5 juillet 2023.
- [Tod23] What I Learned Today. Billboard in unity : How to create 2d/3d sprites that always face the camera. <https://www.youtube.com/watch?v=sVCaAkocdj4>, 2023. Accédé le 5 juillet 2023.
- [Unia] Unity. Bounds. <https://docs.unity3d.com/ScriptReference/Bounds.html>. Accédé le 9 juin 2023.
- [Unib] Unity. Camera.anamorphism. <https://docs.unity3d.com/2022.2/Documentation/ScriptReference/Camera-anamorphism.html>. Accédé le 7 juin 2023.
- [Veg17] Jimmy Vegas. Mini unity tutorial - how to create scrolling credits scene - beginners tutorial. <https://www.youtube.com/watch?v=cj6hwCjiVZE&t=9s>, 2017. Accédé le 7 juillet 2023.
- [way] Internet archive wayback machine. <https://archive.org/web/>. Accédé le 8 juin 2023.