Projet de Fin d'Etude

Outil Pédagogique de Storyboard Collaboratif

Keywords: Unity, C#, Réseau, Education, Art

Master Informatique Image et Son

Réalisés par

Jules COMBARNOUS, Emma Gendre, Mathilde KIEFFER et Nathan SALIN

Professeur.es encadrant.es

Vincent Casamayou

Juliette Le Meudec

CONTENTS

1. Etat de l'art / existant **2**

1.1. Applications similaires . . . 2

1.2. L'importance de la collaboration . . . 2

1.3. Interface graphique pour favoriser la créativité . . . 3

2. User stories **4**

2.1. Création d'un Serveur . . . 4

2.2. Déplacement libre d'un utilisateur . . . 4

2.3. Déposer un objet . . . 4

2.4. Sélectionner un objet . . . 4

2.4.1. Mouvement Libre . . . 5

2.4.2. Utilisation des gadgets . . . 5

2.4.2.1. Positionnement avec gadget . . . 5

2.4.2.2. Rotation avec gadget . . . 5

2.4.2.3. Redimensionnement avec gadget . . . 5

2.4.3. Changement de couleur . . . 5

2.4.4. Appartenance de l'objet . . . 6

2.5. Suppression d'un objet . . . 6

2.6. Desélectionner un objet . . . 6

2.7. Caméra . . . 6

2.7.1. Activation de l'interface caméra . . . 6

2.7.2. Sélection d'une caméra . . . 6

2.7.3. Application de filtres . . . 6

2.7.4. Changement de caméra . . . 6

2.7.5. Activation de point de fuite côté client . . . 6

2.7.6. Activation de point de fuite côté serveur . . . 7

2.7.7. Quitter l'interface avec une option active . . . 7

2.7.8. Rejoindre une caméra avec une option active . . . 7

2.7.9. Activation du multi point de vue . . . 7

2.7.10. Rotation de la caméra . . . 7

2.7.11. Capture d'écran d'une caméra . . . 7

2.8. Lumières . . . 7

2.9. Bouton d'aide 8

3. Choix logiciels **9**

3.1. Unity 9

3.2. Fish-Net 9

4. Conception **11**

4.1. Interface Graphique 11

4.1.1. Interface graphique principale 11

4.1.2. Interface graphique des points de vue et comparaison des points de vue 13

4.1.3. Barre d'outils 16

4.1.4. Bouton Curseur 16

4.1.5. Bouton couleur 16

4.1.6. Bouton gadget transform 16

4.1.7. Bouton gadget rotation 17

4.1.8. Bouton gadget scale 17

4.1.9. Bouton caméra 17

4.1.10. Bouton interface déroulante 17

4.1.11. Bouton aide 18

4.1.12. Tooltip 18

4.2. Caméra 18

4.2.1. Propriétés de la caméra 19

4.3. Déplacement des utilisateurs 19

4.4. Sélection d'un objet 20

4.5. Mouvement libre des objets 20

4.6. Modifier les propriété d'un objet 20

4.6.1. Modifier les propriétés d'un objet avec les gadgets 20

4.6.2. Modifier les propriétés avec l'interface utilisateur 20

4.6.3. Modifier le nom de l'objet 20

4.6.4. La couleur 21

4.6.5. Modification des autres propriétés des objets 21

4.7. Créer ou supprimer un objet 21

4.8. Aspect collaboratif 22

4.8.1. Avatar 22

4.8.2. Limitations de sélection 23

4.8.3. Récupération de l'historique des actions 23

4.9. Système de rôle 24

4.9.1. Menu d'accueil 25

5. Réalisation **27**

5.1. Touche du clavier 27

5.2. Interaction dans l'espace 3D . . . 27
5.2.1. Ajout d'objets dans l'espace 3D . . . 27
5.2.2. Sélection d'un objet . . . 29
5.2.3. Informations de l'objet sélectionné . . . 30
5.2.4. Changer la couleur d'un objet . . . 31
5.2.5. Mouvements libres de l'utilisateur dans l'espace . . . 31
5.2.6. Manipulation libre des objets . . . 33
5.2.7. Ajout du gadget de transformation pour la manipulation des objets . . . 33
5.2.8. Suppression d'un objet . . . 34
5.2.9. Récupérer ou renommer un objet . . . 35
5.3. Caméra, création et manipulation dans l'espace 3D . . . 35
5.3.1. Construction des caméras . . . 35
5.3.2. Ajout de caméras . . . 36
5.3.3. Camera Manager . . . 37
5.3.4. Cacher les caméras . . . 38
5.3.5. Les filtres des caméras . . . 38
5.3.6. Capture d'écrans . . . 39
5.3.7. Création et manipulation de points de fuite . . . 39
5.4. Lumière, création et manipulation dans l'espace 3D. . . . 41
5.4.1. Visuel des lumières . . . 42
5.4.2. Ajout de lumières dans la scène . . . 43
5.4.3. Modification des lumières . . . 43
5.5. Serveur et Client . . . 45
5.5.1. Récupération de l'historique des actions . . . 45
5.6. Interface Utilisateur . . . 46
5.6.1. Barre d'outils . . . 46
5.6.2. Interface redimensionnable . . . 47
5.6.3. Interface réactive . . . 49
5.6.4. Interface d'apparition des objets . . . 50
5.6.5. CursorManager . . . 51
5.6.6. Modification par valeurs numériques . . . 52
5.6.7. Éditions des propriétés uniques des objets . . . 54
5.6.8. Changement des palettes de couleurs . . . 56
5.6.9. Page d'aide . . . 57
5.6.10. Tooltip . . . 57

6. Tests et Résultats **59**
6.1. Fiches des participants . . . 59
6.2. Information sur l'ensemble des testeurs . . . 60
6.3. Notes sur le déroulement des tests . . . 60
6.4. Résultats et Conclusions . . . 62

7. Conclusion et Perspective **66**

Références **68**

A. Annexes **69**

INTRODUCTION

Les technologies numériques ont révolutionné de nombreux aspects de notre société y compris l'éducation et l'apprentissage. Ces avancées étaient largement explorées dans les disciplines scientifiques telles que les mathématiques mais moins dans les domaines plus artistiques. Cependant, il est important de reconnaître le potentiel des outils numériques dans des domaines tels que les arts visuels où des concepts souvent abstraits ou cachés peuvent être rendus plus accessibles grâce à ces technologies.

Dans le domaine du dessin, certaines notions fondamentales telles que la perspective, les proportions et l'utilisation de la lumière sont essentielles mais peuvent parfois être difficiles à comprendre et à appliquer dans des situations concrètes. Par ailleurs, l'intégration d'environnements numériques pour la collaboration entre étudiants peut soulever des interrogations concernant la créativité et la subjectivité.

Dans ce contexte[1], notre projet a pour but de remédier au manque d'outils numériques dédiés à l'enseignement des pratiques artistiques en se concentrant spécifiquement sur la création de storyboards. Les storyboards jouent un rôle important dans de nombreux processus artistiques et de production visuelle et permettent de conceptualiser et de valider des idées avant de passer à des étapes plus avancées. Les solutions existantes sont souvent limitées et peu adaptées à la collaboration à distance en particulier lorsqu'il s'agit d'intégrer des éléments en 3D. En effet, il est très rare de nos jours de trouver des exemples mélangeant ces multiples concepts.

Ainsi, notre projet consiste à développer un outil permettant la collaboration à distance pour la création de storyboards en 3D. En utilisant une banque de fonctionnalités préexistantes, les utilisateurs pourront composer des scènes en trois dimensions, placer des éléments et des caméras et capturer des rendus 2D de la scène pour une utilisation ultérieure. L'aspect collaboratif de l'outil ouvrira de nouvelles possibilités d'interactions et d'échanges entre les étudiants et favorisera ainsi la créativité et l'apprentissage collectif.

Pour se faire, au sein du projet, nous allons implémenter des outils pédagogiques tels que l'affichage des points de fuite pour aider les utilisateurs à comprendre et à appliquer les principes fondamentaux du dessin et de la perspective. Ce projet combine les technologies numériques, l'enseignement des arts visuels et la collaboration à distance, ce qui pourrait considérablement améliorer l'apprentissage dans ce domaine.

Dans ce rapport, nous détaillerons les différentes étapes du développement de cet outil en mettant en avant les différentes fonctionnalités du projet, les problèmes rencontrés et les perspectives d'avenir pour son utilisation et son amélioration.

[1] *Sujet du projet :* `https://masterinfo.emi.u-bordeaux.fr/wiki/lib/exe/fetch.php?media=wiki:iis:pfe-storyboard.pdf`

1 ETAT DE L'ART / EXISTANT

Ce projet répond au besoin d'avoir une application artistique, éducative et collaborative. Elle doit respecter ces principes tout en étant facile à prendre en main et compréhensible pour une population qui pourrait être néophyte dans le domaine de la création 3D ou tout simplement dans la création numérique.

1.1. Applications similaires

On dénombre de nombreuses applications similaires qui ont pour but de créer des storyboards de manière numérique. Dans cet existant, on va trouver des applications conçues pour le storyboarding que cela soit en logiciel **[8]** ou en application web **[9]**. Toutefois, ces applications sont pour la plupart dans un environnement 2D.

Notre application se veut aussi collaborative et éducative. Il existe des applications qui combinent ces deux aspects en permettant l'apprentissage collaboratif d'un sujet particulier par plusieurs utilisateurs **[10]**.

Dans le domaine de la 3D, il existe de nombreux exemples d'applications. Ces logiciels vont utiliser de la modélisation par ordinateur et du rendu 3D **[6]**. Ce type de logiciel facilite la manipulation en temps réel d'objets 3D ou d'éléments spécifiques, comme un mannequin **[5]**, en offrant une interface graphique épurée et intuitive pour apporter des modifications.

1.2. L'importance de la collaboration

Notre application étant développée à des fins éducatives, la collaboration de ses utilisateurs en est un point clé. L'apprentissage collaboratif présente de nombreux avantages par rapport à des efforts compétitifs et/ou individualistes **[3]**. Les différents avantages sont les suivants : production de résultats plus satisfaisants, plus grande productivité, relations plus chaleureuses, solidaires et engagées entre collaborateurs ainsi qu'une meilleure santé psychologique, une plus grande compétence sociale et une meilleure estime de soi.

Le domaine de l'éducation devrait encourager davantage les activités collaboratives et créatives. En effet, ces activités favorisent le développement des compétences de travail en groupe et de l'intelligence collective chez les étudiants. Cela pourrait avoir un impact positif sur leur avenir **[1]**.

L'utilisation d'un espace collaboratif afin d'expérimenter un sujet particulier va donc favoriser sa compréhension ainsi que le travail d'équipe **[10]**.

1.3. Interface graphique pour favoriser la créativité

Selon une étude de Anne-Laure Sellier et Darren W Dahl **[7]**, il a été constaté que la restriction des options pour les utilisateurs expérimentés peut favoriser leur créativité tandis que les utilisateurs moins expérimentés sont souvent plus créatifs lorsqu'ils disposent de davantage de choix. Ainsi, afin de répondre aux besoins de tous les utilisateurs sans générer de frustration, notre application vise à trouver un juste équilibre. Nous cherchons à créer un environnement qui favorise la créativité, quel que soit le niveau d'expertise de l'utilisateur. Pour atteindre un tel objectif, nous pourrions par exemple envisager de personnaliser les palettes de couleurs en fonction des préférences individuelles de chaque étudiant.

Il est crucial de mettre en place un environnement collaboratif qui stimule la créativité des utilisateurs en favorisant un climat de confiance, d'encouragement et d'exploration sans risque **[2]**. Les applications collaboratives réussissent à stimuler la créativité de leurs utilisateurs en favorisant leur immersion dans l'espace collaboratif **[10]**. Il est préférable d'opter pour des choix de couleurs pastel et de proposer un espace infini afin d'éviter à l'utilisateur de se sentir confiné, ce qui pourrait réduire sa créativité.

Nous nous inspirerons également des principes de Gestalt, des règles de Shnerderman ainsi que de celles de Nielsen **[4]** pour concevoir une interface utilisateur ergonomique. Ces principes visent respectivement à prendre en compte les processus cognitifs humains dans la conception d'une interface graphique, à présenter les attentes fondamentales en matière d'interface utilisateur et enfin à fournir des lignes directrices générales pour une interface graphique efficace.

2 USER STORIES

De multiples scénarios d'application peuvent être envisagés pour notre logiciel. Ci-dessous, nous présentons des exemples de scénarios qui mettent en lumière les fonctionnalités offertes par le logiciel :

2.1. Création d'un Serveur

Un utilisateur **A** crée un serveur puis rejoint son propre serveur en tant que client devenant ainsi le propriétaire du serveur. Ensuite, les utilisateurs **B** et **C** rejoignent sa session en tant que clients. L'utilisateur **A** décide de quitter le serveur transférant ainsi la propriété à l'utilisateur **B** qui est le premier à s'être connecté après l'ancien propriétaire.

2.2. Déplacement libre d'un utilisateur

Un utilisateur **A** se déplace librement dans la scène. En parallèle, l'utilisateur **B** observe en temps réel le déplacement de **A** tout en se déplaçant lui-même. Lorsqu'un utilisateur **C** rejoint le serveur, il voit la position actuelle de **A** et **B** suite à leurs déplacements.

2.3. Déposer un objet

L'utilisateur **A** souhaite déposer un objet dans la scène. Pour ce faire, il clique sur l'icône correspondant à l'objet souhaité dans la barre inférieure de l'interface. En maintenant son clic, il glisse l'objet dans la scène et relâche ensuite le clic pour le déposer. Une fois déposé, l'objet apparaît sur la scène et peut être manipulé par les différents utilisateurs.

2.4. Sélectionner un objet

Lorsqu'un utilisateur **A** sélectionne un objet de la scène qui n'est pas déjà sélectionné par un autre utilisateur, l'objet change de matériau pour indiquer sa sélection. Pour l'utilisateur **A**, l'objet apparaît d'une certaine couleur, tandis que pour les autres utilisateurs, il est affiché avec une couleur désignant l'appartenance actuelle à **A**.

2.4.1. Mouvement Libre

L'utilisateur **A** active l'outil curseur. Il peut ensuite déplacer librement l'objet dans la scène en maintenant le clic de sa souris sur l'objet sélectionné et en déplaçant le curseur sur la scène. Lorsque le clic est relâché, l'objet cesse de suivre le curseur.

2.4.2. Utilisation des gadgets

L'utilisateur **A** active l'outil de positionnement, de rotation ou de redimensionnement. En conséquence, un gadget correspondant à l'outil choisi apparaît sur l'objet sélectionné. Les autres utilisateurs présents sur le serveur ne voient pas apparaître le gadget et ne peuvent pas modifier l'objet sélectionné par **A**.

2.4.2.1. Positionnement avec gadget

L'utilisateur **A** active l'outil de positionnement et maintient le clic sur l'un des axes du gadget. Ensuite, il déplace sa souris pour faire bouger l'objet le long de l'axe sélectionné. Par la suite, il maintient le clic sur la boule au centre du gadget et peut bouger librement l'objet de la même manière qu'avec l'outil curseur.

2.4.2.2. Rotation avec gadget

L'utilisateur **A** active l'outil de rotation et maintient le clic sur l'un des axes du gadget. Ensuite, il déplace sa souris pour faire pivoter l'objet autour de l'axe sélectionné.

2.4.2.3. Redimensionnement avec gadget

L'utilisateur **A** active l'outil de mise à l'échelle et maintient le clic sur l'un des axes du gadget. Ensuite, il déplace sa souris pour agrandir l'objet le long de l'axe sélectionné. Par la suite, en maintenant le clic sur le cube au centre du gadget, l'objet s'agrandit uniformément sur tous les axes en fonction du déplacement de la souris.

2.4.3. Changement de couleur

L'utilisateur **A** souhaite modifier la couleur de l'objet qu'il a sélectionné. Il choisit de cliquer sur l'une des couleurs proposées sur la palette de couleur et l'objet prend la couleur sélectionnée. Les utilisateurs **B** et **C** voient en temps réel le changement de couleur. L'utilisateur **B**, quant à lui, décide de changer sa palette de couleur en spectre colorimétrique. N'étant pas certain de la couleur qu'il veut choisir, il maintient son clic sur le spectre et déplace sa souris. L'objet que **B** a sélectionné change donc en temps réel de couleur à chaque déplacement de la souris. Tous les changements seront visibles par **A** et **C**. **B** choisit de relâcher son clic et l'objet qu'il a sélectionné adopte la dernière couleur qu'il a choisie.

2.4.4. Appartenance de l'objet

L'utilisateur **A** sélectionne un objet. Les utilisateurs **B** et **C** tentent de cliquer sur l'objet pour le sélectionner à leur tour. Cela échoue car **A** a déjà sélectionné l'objet. **B** et **C** tentent également de cliquer sur les outils gadgets de leur interface utilisateur mais aucun gadget n'apparaît car ils ne peuvent pas sélectionner l'objet.

2.5. Suppression d'un objet

L'utilisateur **A** n'est finalement pas satisfait de l'objet sélectionné et décide de le supprimer.

2.6. Desélectionner un objet

L'utilisateur **A** souhaite arrêter de modifier l'objet qu'il a sélectionné. Pour le désélectionner, il effectue un clic droit avec sa souris. Les autres utilisateurs constatent ensuite que l'objet perd son indication visuelle indiquant qu'il n'est plus soumis à la sélection. Désormais, l'objet est sélectionnable par tous les utilisateurs.

2.7. Caméra

2.7.1. Activation de l'interface caméra

L'utilisateur **A** appuie sur le bouton "caméra et capture d'écran" et active une nouvelle interface utilisateur. L'effet de ce bouton n'apparaissant que pour l'utilisateur qui l'a déclenché, aucun des changements d'interface générés ne seront visibles par les autres utilisateurs.

2.7.2. Sélection d'une caméra

L'utilisateur **A** visualise dans la partie droite de l'interface utilisateur la liste des caméras présentes dans la scène et en sélectionne une en la cochant. Sa vue est alors remplacée par celle de la caméra cochée.

2.7.3. Application de filtres

L'utilisateur **A** décide ensuite d'appliquer un filtre sur la caméra sélectionnée et fait son choix en cliquant parmi ceux disponibles sur le menu déroulant dans la partie droite de l'interface utilisateur. Le filtre s'applique ensuite sur la vue de la caméra sélectionnée. Les utilisateurs **B** et **C** ne voient pas l'application de filtre.

2.7.4. Changement de caméra

L'utilisateur **A** veut changer de point de vue; il change de caméra en désactivant celle précédemment sélectionnée et en sélectionne une autre.

2.7.5. Activation de point de fuite côté client

L'utilisateur **A** souhaite maintenant activer le point de fuite de la caméra et coche l'option correspondante dans l'interface utilisateur pour l'afficher. Le point de fuite apparaît alors pour l'utilisateur **A** mais pas pour les autres utilisateurs.

2.7.6. Activation de point de fuite côté serveur

L'utilisateur **A** désire ensuite permettre la visualisation de ce point de fuite aux autres utilisateurs. Pour ce faire, il appuie sur le bouton de l'interface utilisateur "point de fuite serveur" permettant l'affichage du point de fuite aux autres utilisateurs du serveur.

2.7.7. Quitter l'interface avec une option active

L'utilisateur **A** quitte le point de vue de la caméra et récupère ainsi le point de vue de son avatar. Puis, il quitte l'interface "caméra et capture d'écran" en sélectionnant l'outil curseur. Tout clic sur un bouton de la barre d'outils se situant à gauche de l'interface utilisateur permet cette action à l'exception du bouton aide.

2.7.8. Rejoindre une caméra avec une option active

L'utilisateur **B** quant à lui décide d'activer à son tour l'interface "caméra et capture d'écran" en appuyant sur le bouton du même nom. Il sélectionne la caméra avec le point de fuite apparent et constate que l'option d'activation du point de fuite côté serveur est déjà activée. Cette option ayant été activée au préalable par l'utilisateur **A** avant de quitter l'outil caméra.

2.7.9. Activation du multi point de vue

L'utilisateur **B** décide d'avoir les points de vue de quatre caméras différentes et les sélectionne dans l'interface utilisateur. Les points de vue apparaissent dans l'ordre suivant : en haut à gauche, en haut à droite, en bas à gauche et en bas à droite. Pour mieux visualiser les objets de la scène, l'utilisateur **B** choisit de cacher les caméras en cliquant sur l'option du même nom dans l'interface utilisateur.

2.7.10. Rotation de la caméra

L'utilisateur **B** estime que le point de vue de la caméra avec le point de fuite côté serveur affiché n'est pas satisfaisant. Il applique donc une rotation à la caméra en désélectionnant l'ensemble des caméras, en sélectionnant l'outil gadget rotation et en effectuant la rotation souhaitée. Le point de fuite suit le point de vue de la caméra dans l'ensemble de ses mouvements.

2.7.11. Capture d'écran d'une caméra

L'utilisateur **A** décide de prendre une capture d'écran de la scène depuis son point de vue. En appuyant sur la touche F12, il génère une capture qui est ensuite enregistrée dans le dossier des données de l'application de l'utilisateur **A**.

2.8. Lumières

L'utilisateur **A** place une lumière dans la scène. Il a préalablement choisi un type de lumière parmi les lumières ponctuelles, directionnelles et projecteurs. Une fois placée, il indique son intensité et le nombre

de rebonds possibles grâce à l'interface utilisateur. Enfin, l'utilisateur **A** choisit de changer la couleur de la lumière qui sera perceptible par tous les utilisateurs présents sur le serveur sur les objets de la scène.

2.9. Bouton d'aide

L'utilisateur **A** appuie sur le bouton "aide" de l'interface utilisateur. Une page s'ouvre fournissant des indications sur les commandes et le fonctionnement de l'application. Cette page est spécifique à l'utilisateur **A** et n'est pas affichée aux utilisateurs **B** et **C**.

3 CHOIX LOGICIELS

Le sujet du projet a pour contrainte l'utilisation de Unity et par conséquent du C# qui est le langage utilisé pour les scripts produits par Unity ainsi que l'utilisation de Fish-Net qui est un module facilitant la création de serveur dans Unity.

3.1. Unity

Unity[1] est une plateforme de développement en temps réel. C'est un logiciel qui permet la création de jeux, d'applications en 3D ou en 2D. Il a l'avantage de posséder une licence gratuite sous contrainte permettant une libre utilisation et création d'applications. Il est réputé pour sa simple prise en main facilitant son accessibilité et possède un large panel de fonctionnalité pouvant plaire aux débutants comme aux expérimentés. Unity a pour avantage de posséder des outils déjà implémentés comme les lumières ou les ombres, ce qui va donc faciliter certains points de notre projet. Les nombreux avantages énumérés confirment clairement que cette application est le choix idéal pour notre projet. Elle répond parfaitement à notre objectif de créer une application dans un environnement 3D permettant la manipulation d'objets en trois dimensions.

3.2. Fish-Net

Fish-Net[2] est un module conçu pour être intégré à Unity permettant ainsi l'implémentation d'un système Client-Serveur au sein de notre application. Cette extension simplifie la configuration du réseau en étendant les fonctionnalités d'Unity avec des classes supplémentaires telles que ***NetworkBehaviour*** qui permettent de définir des caractéristiques spécifiques pour les méthodes indiquant ainsi si elles doivent réagir côté client ou serveur. Cet outil offre la possibilité d'envoyer au serveur une méthode à l'aide de l'annotation ***[ServerRPC]*** et de spécifier si cette méthode doit être envoyée aux observateurs ou à une cible particulière à l'aide respectivement des annotations ***[ObserverRPC]*** et ***[TargetRPC]***. De plus, Fish-Net propose des fonctionnalités permettant de distinguer si un objet appartient à un client (***base.Owner***) ou au serveur (***base.Server***). Ces fonctionnalités se révéleront extrêmement utiles pour mettre en place efficacement et simplement un système Client-Serveur sur Unity. Par ailleurs, notre projet s'inscrit dans un cadre de recherche où nous accordons une importance capitale à l'adoption de solutions à la fois gratuites et open source. Après une analyse minutieuse des plugins réseau disponibles et compatibles avec Unity, Fish-Net est l'unique solution répondant à ces exigences. Outre sa gratuité, Fish-Net se distingue par sa documen-

[1] `https://unity.com/fr`

[2] *Télécharger FishNet :* `https://assetstore.unity.com/packages/tools/network/fish-net-networking-evolved-207815` | *Documentation de Fish-Net :* `https://fish-networking.gitbook.io/docs`

tation exhaustive et sa compatibilité avec toutes les principales plateformes d’hébergement de serveurs, ce qui en fait le choix idéal pour notre projet.

4 CONCEPTION

4.1. Interface Graphique

4.1.1. Interface graphique principale

Dans notre démarche théorique, la première étape consiste à concevoir une interface graphique. Cette application est destinée aux artistes qu'ils soient confirmés ou en devenir, ce qui impose plusieurs contraintes en termes d'ergonomie :

1. L'interface doit être familière et compréhensible pour notre public cible.
2. Elle ne doit pas être trop envahissante visuellement et ainsi laisser la majeure partie de l'écran dédiée à la visualisation de la scène.

Notre interface doit également répondre à plusieurs contraintes fonctionnelles, ce qui implique que l'interface graphique doit comprendre les éléments suivants :

1. Une barre d'outils permettant de sélectionner les divers outils de manipulation de la scène.
2. Une section permettant de modifier numériquement les différents paramètres des objets de la scène ou des outils.
3. Une zone permettant d'ajouter directement des objets à la scène.

Ces éléments sont essentiels pour assurer une utilisation efficace et intuitive de l'application. La première contrainte nous pousse à recourir au transfert de connaissances. L'utilisation d'une interface graphique similaire à celles des applications existantes présente deux avantages majeurs. Tout d'abord, pour les débutants, cela leur permet d'acquérir des mécanismes et des connaissances transférables à d'autres logiciels standard de l'industrie. Ensuite, pour les artistes expérimentés, une interface familière facilite grandement la prise en main du logiciel.

Pour répondre à la deuxième contrainte, notre objectif est de garantir que la majorité de l'espace d'affichage soit dédiée à la visualisation de la scène. Cela permettra aux artistes de se concentrer pleinement sur leur création sans être distraits ou contraints par une interface encombrante. En favorisant la clarté et la simplicité, nous assurons une utilisation intuitive de l'application, ce qui est essentiel pour encourager la créativité et la productivité de nos utilisateurs.

Pour répondre à cette contrainte, nous avons envisagé deux solutions. La première consistait à concevoir une interface des plus minimaliste. Voici le prototype résultant de cette idée :

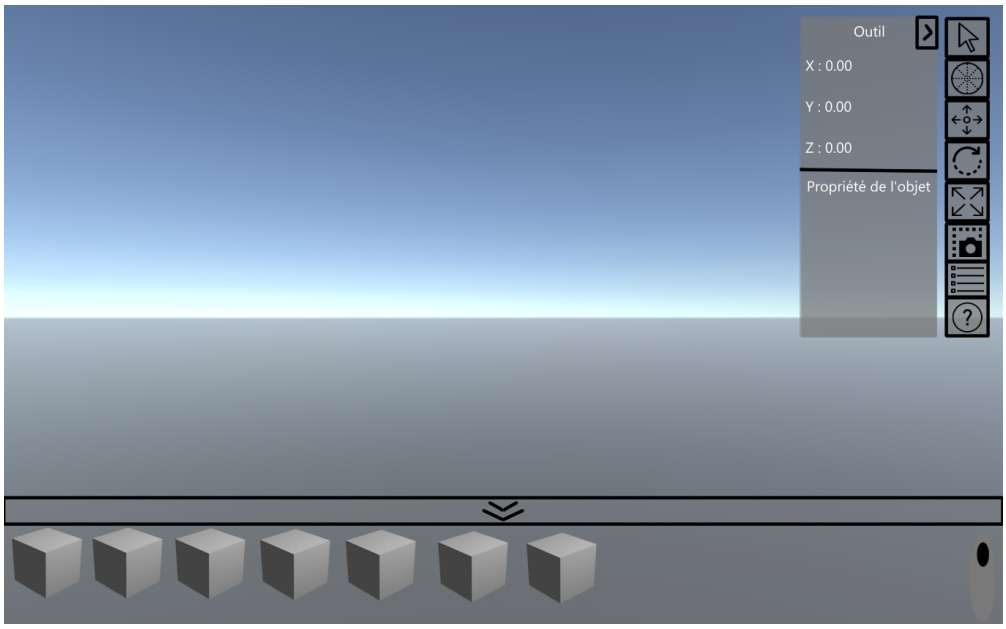


Figure 4.1.: Prototype de l'interface graphique minimaliste

Néanmoins, après une réflexion approfondie en collaboration avec nos clients, nous avons réalisé que cette version masquait trop d'informations ce qui risquait de perturber à la fois les débutants et les utilisateurs confirmés. En effet, dans cette version, les actions et les informations n'étaient pas toujours facilement accessibles rendant l'interface lourde et lente à utiliser. De plus, cette interface graphique limitait considérablement l'espace disponible pour les différents outils, ce qui aurait pu rendre l'application plus lourde à utiliser à long terme et compliquer grandement l'ajout de nouveaux outils coté développeur.

Nous avons donc opté pour la seconde option qui consistait à nous appuyer sur les normes de l'industrie mais en laissant la possibilité à l'utilisateur de redimensionner les fenêtres à sa guise.

L'idée théorique du redimensionnement repose sur la modification de la taille des fenêtres du logiciel lorsqu'on déplace le curseur vers leurs bords, de la même manière que les logiciels sur un bureau. Lorsque l'utilisateur clique sur sa souris, il peut alors ajuster la taille de l'interface dans la direction où le curseur est déplacé. Par exemple, si la sélection est effectuée sur la partie supérieur de la fenêtre, l'utilisateur peut redimensionner l'interface de haut en bas. De même, si la sélection est faite sur les bords gauches ou droits, l'utilisateur peut ajuster la taille de la fenêtre de gauche à droite.

Pour concevoir le prototype de notre interface graphique, nous nous sommes largement inspirés des standards de l'industrie du dessin tels que **Photoshop**[1] ou **Krita**[2] qui offrent des fonctionnalités similaires à ce que nous souhaitons proposer. Ces logiciels présentent une barre d'outils sur la partie gauche de l'écran tandis que la partie droite est entièrement dédiée aux calques et aux différents paramètres des outils et des calques. Dans le cadre de notre logiciel, nous envisageons de remplacer les calques par les objets de notre scène. Cependant, ces logiciels ne disposent pas d'une interface permettant spécifiquement d'ajouter

[1] https://www.adobe.com/fr/products/photoshop.html
[2] https://krita.org/fr/

directement des objets sur leurs calques. En adoptant un style d'interface similaire, la seule zone encore disponible sur l'écran est la partie inférieure. Par conséquent, nous avons décidé de placer la partie de l'interface chargée d'ajouter des objets dans la scène dans cette partie inférieure de l'écran.

Nous avons finalement conçu le prototype visuel ci-dessous, qui a servi de fondement pour la création de l'interface graphique de notre application :

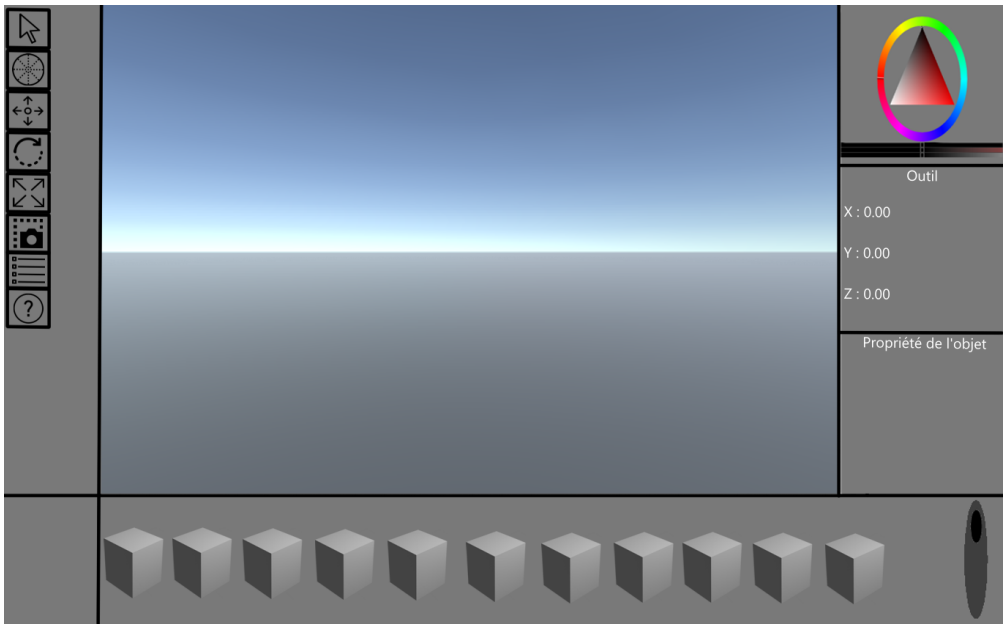


Figure 4.2.: Prototype de l'interface graphique finale

4.1.2. Interface graphique des points de vue et comparaison des points de vue

Dans une application collaborative d'art, il est crucial d'explorer et de manipuler les différents points de vue de la scène selon les besoins. Pour répondre à cette exigence, nous avons décidé de créer un outil dédié à la manipulation des points de vue utilisant l'icône de la caméra et du rectangle en pointillé, comme illustré dans la figure 4.2.

Cet outil doit être capable d'accomplir plusieurs tâches :

1. Manipuler les propriétés uniques du point de vue.
2. Faciliter la comparaison entre différents points de vue.
3. Différencier clairement les points de vue.

Pour comparer les points de vue, l'interface graphique devra afficher une liste de tous les points de vue présents dans la scène. Cette liste sera placée dans la partie droite de l'interface réservée aux paramètres de l'outil. Idéalement, il devrait être facile de distinguer visuellement les points de vue observés des points de vue non observés. À cette fin, nous avons choisi un indicateur visuel : un œil. Un œil fermé signifie que le point de vue n'est pas utilisé ou observé par l'utilisateur tandis qu'un œil ouvert indique que l'utilisateur est en train de l'observer ou de le manipuler.

Il est également essentiel de pouvoir comparer différents points de vue de manière efficace. Pour cela, les comparaisons doivent être lisibles et ne pas altérer la représentation visuelle des points de vue. Nous avons donc décidé de limiter théoriquement le nombre de points de vue comparés à quatre. Cette limitation permet de conserver le rapport d'aspect du point de vue d'origine tout en assurant une bonne visibilité sur tous les types d'écrans.

Pour rendre plus claire la distinction entre les différents points de vue lors de la comparaison, une solution simple consisterait à afficher les noms correspondants pendant la comparaison.

L'espace disponible restant dans la partie droite de l'interface sera réservé à la manipulation des propriétés uniques des différents points de vue.

Nous avons finalement conçu le prototype visuel ci-dessous pour l'outil point de vue :

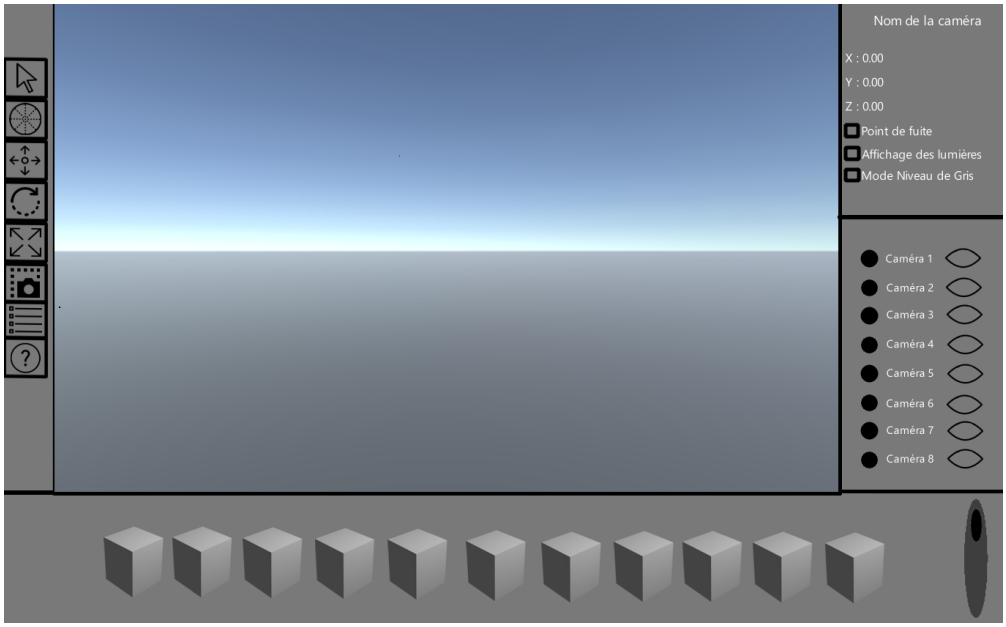


Figure 4.3.: Prototype de l'interface graphique de l'outil point de vue sans comparaison

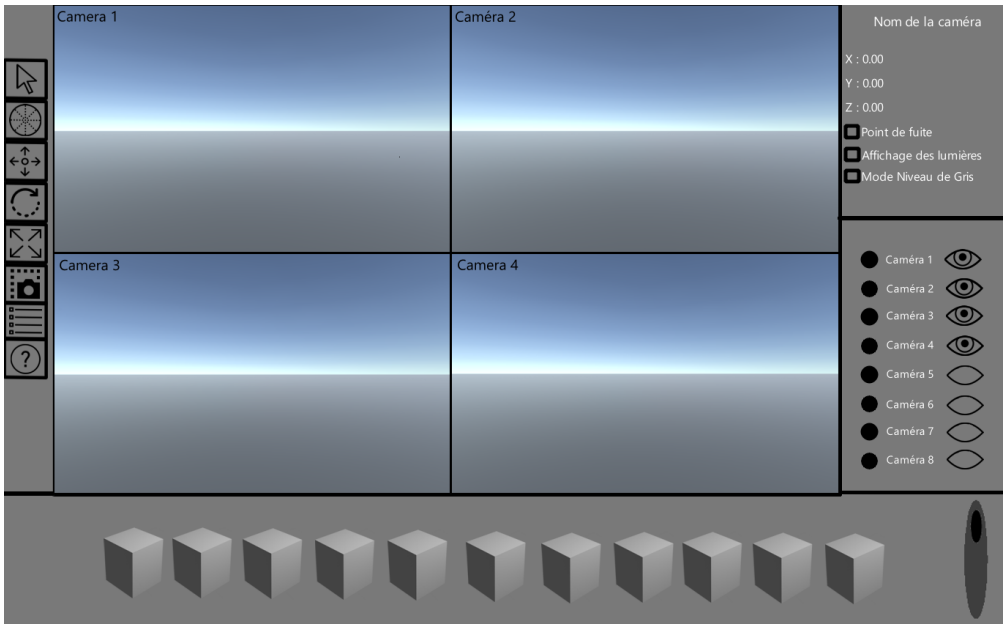


Figure 4.4.: Prototype de l'interface graphique de l'outil point de vue avec comparaison

Nous avons également envisagé de simplifier davantage l'interface lorsque le mode de comparaison est activé. Dans ce mode, il est impossible de modifier la scène ce qui rend la barre d'outils et la barre d'ajout d'objets non essentielles. Ainsi, en les cachant, il serait possible d'offrir plus d'espace visuel pour les comparaisons.

Cela se traduirait par l'interface graphique suivante :

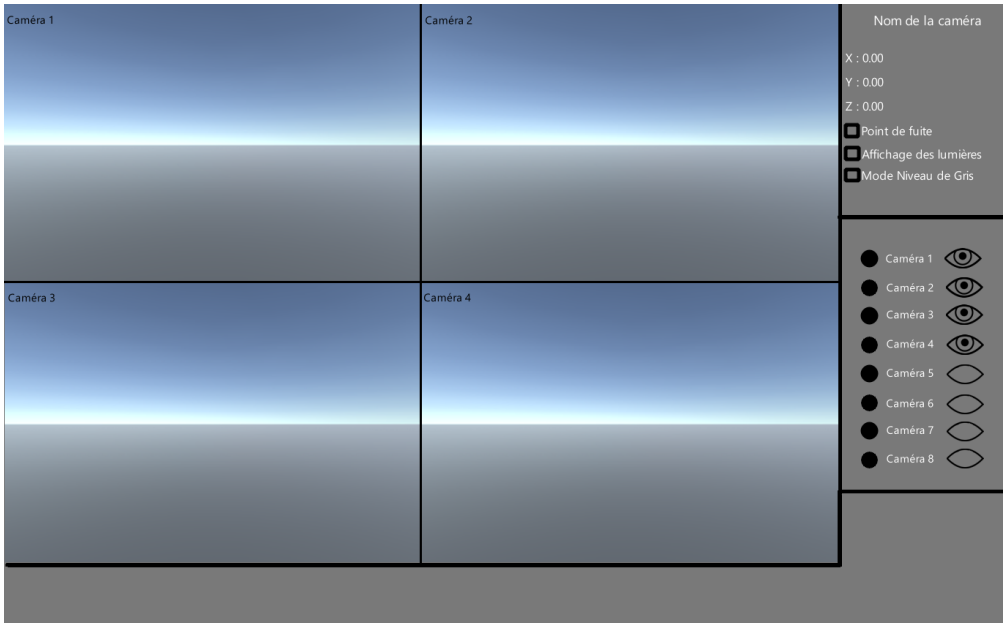


Figure 4.5.: Deuxième version du prototype de l'interface graphique de l'outil point de vue

4.1.3. Barre d'outils

À gauche de l'interface utilisateur, la barre d'outils regroupe principalement l'ensemble des actions réalisables par un utilisateur sur les objets de la scène. Ces différents outils sont représentés sous forme de boutons permettant d'activer et/ou de désactiver les fonctionnalités/outils.

4.1.4. Bouton Curseur

Le bouton curseur est représenté par l'image suivante dans la barre d'outil :

Figure 4.6.: Image de l'outil curseur

L'outil curseur possède plusieurs fonctionnalités. La première permet la sélection de l'objet dans la scène et d'autoriser son déplacement libre. La seconde consiste à éditer les paramètres uniques de l'objet sélectionné en les affichant dans la partie droite de l'interface.

4.1.5. Bouton couleur

Ce bouton est représenté par une roue des couleurs :

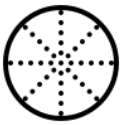


Figure 4.7.: Image du bouton couleur

Ce bouton permet de changer le type de palette de couleurs d'un simple clic. Deux options sont disponibles : une palette simplifiée de 50 couleurs et une palette représentée sous forme de spectre colorimétrique.

4.1.6. Bouton gadget transform

Le bouton gadget transform est représenté par l'image suivante dans la barre d'outil :

Figure 4.8.: Image de l'outil gadget transform

Cet outil permet de faire apparaître le gadget de déplacement de l'objet sur l'objet sélectionné. Le gadget est alors manipulable par l'utilisateur. Le gadget de déplacement sera composée de 4 éléments. Une sphère permettant de déplacer librement l'objet dans une scène et de trois flèches représentant les axes x, y, z dans la scène. En cliquant sur les axes, l'objet pourrait être déplacé en suivant l'axe sélectionné.

4.1.7. Bouton gadget rotation

Le bouton gadget rotation est représenté par l'icône suivante dans la barre d'outils :

Figure 4.9.: Image de l'outil gadget rotation

Cet outil permet de faire apparaître le gadget de rotation. Ce gadget est composé de trois cercles représentant les axes de rotation x, y et z. Comme pour le gadget de transform, si l'utilisateur clique sur l'un des cercles, il pourra alors faire pivoter l'objet en fonction de l'axe sélectionné en déplaçant la souris.

4.1.8. Bouton gadget scale

Le bouton gadget scale est représenté par l'icône suivante dans la barre d'outils :

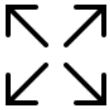


Figure 4.10.: Image de l'outil gadget scale

Cet outil permet d'afficher le gadget de mise à l'échelle / scaling de l'objet. Il est composé de quatre éléments : trois tiges à bout carré et un carré à la base des tiges. Les tiges permettent de redimensionner l'objet selon les axes x, y et z. Le carré à la base permet d'effectuer un redimensionnement libre, agissant sur tous les axes simultanément. Le fonctionnement de cet outil est similaire à celui des outils précédents.

4.1.9. Bouton caméra

Le bouton caméra est représenté par l'icône suivante dans la barre des outils :

Figure 4.11.: Image de l'outil Caméra

Ce outil permet simplement d'ouvrir la fenêtre de l'outil caméra sur la droite de l'interface présenté dans la partie 4.1.2.

4.1.10. Bouton interface déroulante

Le bouton interface déroulante est représenté par l'icône suivante dans la barre d'outils :

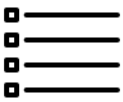


Figure 4.12.: Image de l'outil interface déroulante

Ce bouton permet de transformer la partie droite de l'interface en une liste déroulante. Cette liste contiendra tous les objets présents dans la scène. Le principe de cette liste est de permettre à l'utilisateur de sélectionner rapidement des objets dans la scène, peu importe leur emplacement et celui de l'utilisateur dans la scène.

4.1.11. Bouton aide

Le bouton d'aide est représenté par l'icône suivante dans la barre d'outils :

Figure 4.13.: Image du bouton Aide

Ce bouton ouvre un menu d'aide qui contiendra tous les éléments nécessaires pour comprendre le fonctionnement de notre application. Cela pourrait inclure :

- Les contrôles
- La sélection des objets dans la scène
- L'ajout d'objet dans la scène
- Le fonctionnement détaillé des différents outils

4.1.12. Tooltip

Du fait que notre application sera un outil d'apprentissage, il est important d'aider l'utilisateur à comprendre comment utiliser les différentes fonctionnalités. Pour ce faire, l'utilisation de tooltips peut être un bon guide pour les utilisateurs. Il s'agit de petites fenêtres contextuelles qui apparaissent généralement lorsque l'on survole un élément avec le curseur et qui fournissent des informations supplémentaires ou des conseils sur cet élément. Dans notre cas, l'apparition du tooltip se fera lors du survol des boutons représentant les outils et lors du survol des objets à placer dans la scène.

4.2. Caméra

Une méthode très simple pour manipuler les points de vue est d'utiliser des caméras. Pour la création de ces caméras, deux options nous ont semblé pertinentes :

1. Un objet virtuel sans présence physique, visible sous forme d'image flottante.

2. Un objet physique avec un maillage en 3D.

La première option ne nous semble pas la plus pratique. Le manque de présence physique des caméras pourrait être déroutant pour l'utilisateur débutant et rendrait leurs sélections et leurs manipulations complexes au sein de l'environnement 3D. Pour ces raisons, nous avons choisi la seconde option.

Les caméras seront des objets physiques placés dans la scène avec un modèle 3D. Ce modèle visuel inclura en son sein un objet "Camera" fourni par Unity permettant de visualiser la scène à travers la caméra. La direction de la caméra sera alignée avec celle du modèle 3D. Ainsi, le modèle 3D servira de guide visuel à l'utilisateur pour manipuler facilement la caméra dans l'espace 3D.

4.2.1. Propriétés de la caméra

Pour fournir aux artistes un large éventail de possibilités, nous souhaitons leur offrir autant d'options que possible pour manipuler les points de vue et les caméras. L'approche la plus simple serait d'introduire des filtres sur les caméras. Ces filtres pourraient être un véritable atout pédagogique pour notre application permettant de masquer ou de mettre en évidence certains éléments de la scène.

Dans la lignée des filtres pédagogiques, un filtre qui pourrait s'avérer particulièrement utile est celui mettant en évidence les ombres produites par les sources lumineuses. En art, la compréhension des ombres et des lumières est un aspect particulièrement complexe qui pose problème à de nombreux artistes. Un tel filtre, en plus de la manipulation en direct des lumières, pourrait aider les étudiants à mieux appréhender ce concept.

Un autre filtre couramment utilisé à mettre en place est le filtre noir et blanc. Généralement, les storyboards sont réalisés en noir et blanc, avec très peu voire pas de couleur. L'ajout d'un tel filtre permettrait de simuler ce type de planche.

Cependant, l'un des atouts de notre application par rapport à un storyboard classique réside dans sa palette de couleurs qui offre un aspect pédagogique supplémentaire à explorer. De plus, un autre aspect souvent difficile à appréhender pour de nombreux étudiants est la colorimétrie d'un dessin ou d'une scène. Pour les aider à mieux comprendre les nuances de couleur et leur répartition sur différents objets, une solution envisageable serait de proposer différents filtres. Ces filtres pourraient mettre en évidence certaines couleurs tout en masquant ou atténuant les autres.

4.3. Déplacement des utilisateurs

Il est important que l'utilisateur ne se sente pas limité dans ses mouvements pour ne pas entraver sa créativité. C'est pourquoi notre objectif est de faire en sorte que l'utilisateur ne soit pas affecté par la gravité et puisse "voler" dans l'espace 3D. Nous estimons que des déplacements plus rapides et libres en trois dimensions permettent non seulement de faciliter la prise de point de vue sur la scène pour l'utilisateur, mais également de faciliter certaines actions, comme par exemple empiler des objets.

4.4. Sélection d'un objet

Afin de manipuler la scène, la sélection des différents objets sera nécessaire. Pour ce faire, un simple clic de souris suffira. Une fois l'objet sélectionné, un contour de couleur spécifique apparaîtra autour de celui-ci. La couleur du contour correspond à la couleur liée à l'utilisateur. Pour faciliter la sélection, lorsque l'utilisateur survole un objet avec sa souris, un contour violet apparaîtra pour indiquer que l'objet est manipulable. Ce contour disparaîtra si l'utilisateur retire sa souris de l'objet. La désélection d'un objet peut être effectuée de deux manières différentes : soit en cliquant sur un autre objet sélectionnable ou en dehors de la scène, soit en utilisant le clic droit de la souris.

4.5. Mouvement libre des objets

Dans l'optique de mettre en place un système de déplacement des objets ne restreignant pas la créativité des utilisateurs, nous avons décidé d'ajouter une option permettant le mouvement libre des objets. L'utilisateur pourra s'il le souhaite, sans afficher le gadget de mouvement, déplacer l'objet dans la direction qu'il souhaite. Pour cela, l'utilisateur clique sur l'objet qu'il souhaite déplacer, l'objet suivra alors la souris et ses déplacements. Pour arrêter cette fonctionnalité, il lui suffira de lâcher le clic de la souris.

4.6. Modifier les propriété d'un objet

4.6.1. Modifier les propriétés d'un objet avec les gadgets

Pour la modification des propriétés d'un objet présent dans la scène, nous avons décidé de concevoir des gadgets que l'on retrouve dans beaucoup d'applications pour le graphisme 3D.

Les designs des modèles des gadgets ainsi que leur fonctionnement ont largement été inspirés par ceux proposés dans Unity[3] et sont décrits dans la partie 4.1.6.

4.6.2. Modifier les propriétés avec l'interface utilisateur

Lorsqu'un objet sera sélectionné, la partie droite de l'interface sera modifiée pour pouvoir montrer les propriétés de l'objet sélectionné à l'utilisateur. Nous avons estimé que l'utilisateur pourrait vouloir faire des modifications précises sur les valeurs de position, rotation et d'échelle d'un objet et avoir une alternative à l'utilisation des gadgets. En fonction du gadget sélectionné par l'utilisateur dans la partie gauche de l'interface utilisateur, les informations modifiables seront différentes. Par exemple, le gadget de transformation fait apparaître les coordonnées de position de l'objet sélectionné. Ses coordonnées seront alors modifiables par l'utilisateur.

4.6.3. Modifier le nom de l'objet

Dans le but de permettre à l'utilisateur de facilement identifier les objets qu'il manipule, nous avons décidé de mettre en place une fonctionnalité permettant à l'utilisateur de renommer chaque objet qui peut être créé via l'interface utilisateur. Cette fonctionnalité facilitera également la collaboration des différents utilisateurs

[3] https://unity.com/fr

en leur permettant d'affecter des noms particuliers à chaque objet manipulé, rendant leurs communications plus simples et facilitant la cohésion du groupe.

4.6.4. La couleur

Dans un storyboard, les formes dessinées sont généralement en noir et blanc. Comme nous voulons proposer le plus d'options possibles aux utilisateurs pour la création de leur storyboard, nous avons décidé de leur donner la possibilité de changer la couleur de chaque objet et lumière à leur disposition. Ce simple ajout leur permettra de visualiser plus facilement la colorimétrie d'une scène.

Dans la pratique, l'utilisateur devra d'abord sélectionner un objet puis choisir une couleur dans un spectre colorimétrique présent dans l'interface utilisateur. Il peut effectuer sa sélection de couleurs de deux manières différentes :

1. Par un simple clic : l'utilisateur peut changer la couleur de l'objet en temps réel en cliquant simplement sur le spectre colorimétrique.
2. Par un clic prolongé : L'utilisateur maintient le clic et déplace sa souris dans l'ensemble du spectre. La couleur de l'objet se met à jour en temps réel à chaque changement de position de la souris. Lorsque le clic est relâché, l'objet conserve la dernière couleur sélectionnée avant le relâchement.

Pour parvenir à ce résultat, l'idée est de récupérer la position de la souris lors du clic sur notre palette de couleurs qui sera une simple image. Nous récupérons ensuite la couleur du pixel correspondant à cette position et l'appliquons au matériau de l'objet.

4.6.5. Modification des autres propriétés des objets

Certains objets, tels que les lumières ou les caméras, possèdent des propriétés uniques. Il est également possible que d'autres objets à l'avenir possèdent des propriétés uniques. Pour cela, il sera nécessaire de fournir des outils pour les manipuler. Les caméras auront leurs propres outils en raison de leurs nombreuses propriétés uniques énoncées précédemment. Quant aux objets autres que les caméras, il sera possible d'accéder à leurs propriétés uniques à l'aide de l'outil curseur. En plus de servir d'outil de sélection, il affichera dans la partie droite de l'interface les différentes propriétés des objets. Les propriétés uniques seront affichées dans un menu déroulant. Ce menu déroulant contiendra un champ adapté à chaque propriété permettant de les modifier. Par exemple, pour les lumières, cela pourrait inclure leur puissance ou leur portée qui seront modifiables à l'aide d'un champ d'entrée acceptant des valeurs numériques uniquement.

4.7. Créer ou supprimer un objet

La partie dédiée à la création d'un objet se situe dans la partie inférieure de l'interface 4.2. Pour faire apparaître un objet, le processus est assez simple. Il suffit à l'utilisateur de cliquer sur l'objet, de maintenir le clic enfoncé et de le déposer à l'endroit désiré dans la scène en déplaçant la souris à cet endroit. Pour que l'objet soit déposé, il est nécessaire qu'il y ait un autre objet ou un sol à l'emplacement du dépôt.

La suppression d'un objet est également simple à effectuer. Pour cela, il suffit de sélectionner un objet à l'aide de tout outil permettant la sélection. Une fois l'objet sélectionné, il suffit à l'utilisateur d'appuyer sur la touche "SUPPR" pour le retirer de la scène. En théorie, la suppression d'un objet ordinaire ne devrait poser aucun problème. Cependant, nous avons dû être attentifs à la suppression des caméras qui sont des objets plus complexes. En effet, le cas le plus problématique survient si un utilisateur supprime une caméra alors qu'un autre utilisateur utilise son point de vue. La solution théorique la plus simple à ce problème consiste à empêcher la suppression de la caméra lorsqu'elle est utilisée par un utilisateur, quel qu'il soit.

4.8. Aspect collaboratif

Dans ce projet, la collaboration entre les différents utilisateurs est un élément essentiel à prendre en compte dans le développement de l'application. En effet, chaque utilisateur doit pouvoir utiliser pleinement l'application sans pour autant perturber le travail de création de ses pairs au sein de la scène. Ces perturbations peuvent revêtir plusieurs formes allant de la simple maladresse à des actes intentionnels à des fins humoristiques ou malveillantes. Pour limiter au maximum ces perturbations, nous avons envisagé d'intégrer différentes solutions.

4.8.1. Avatar

Afin de bien visualiser les différents utilisateurs d'un serveur, nous avons pensé à la mise en place d'un système d'avatars. Suite à la phase de connexion à un serveur, une page (Figure 4.14) permet aux utilisateurs de créer un petit personnage qui les représentera dans le monde virtuel. Ils pourront choisir un pseudo, une couleur ou encore un accessoire distinctif tel qu'un chapeau ou encore une ceinture. Lors de cette phase de création, un utilisateur ne sera pas en mesure de sélectionner une couleur déjà utilisée par un autre avatar présent sur le serveur. Grâce à cette simple personnalisation, les utilisateurs pourront s'identifier personnellement à leur personnage. Ces différentes personnalisations permettront une meilleure reconnaissance de chaque utilisateur.

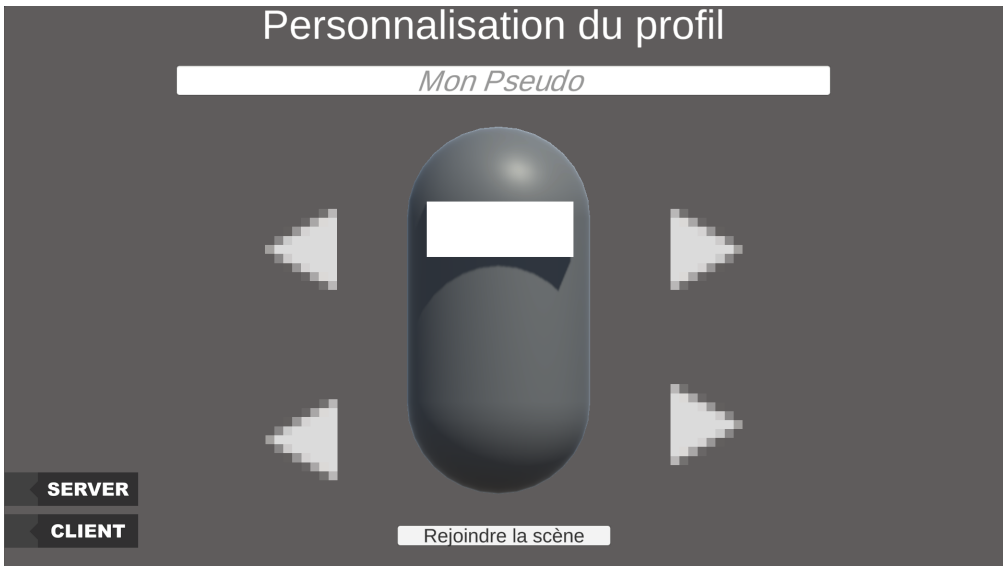


Figure 4.14.: L'interface de personnalisation de l'avatar

4.8.2. Limitations de sélection

Lors de la collaboration, il est possible que plusieurs personnes aient envie de modifier ou de manipuler un même objet présent sur les serveurs. Cela peut entraîner divers problèmes majeurs, notamment des conflits au sein du groupe. Si plusieurs personnes veulent déplacer un même objet de manière différente, il n'y aura aucun moyen de donner raison à l'une des parties. Outre les problèmes de litige, cela peut également retarder le groupe dans sa tâche. Pour éviter ce genre de cas, si un utilisateur sélectionne un objet, lui et lui seul pourra le manipuler jusqu'à sa désélection. Pour repérer facilement qu'un objet est déjà en cours de manipulation, il sera surligné de la couleur de l'avatar qui l'a sélectionné. Un utilisateur peut également verrouiller un objet, ce qui empêche toute modification de cet objet dans la scène. En effet, une telle fonctionnalité permettrait d'éviter les erreurs si quelqu'un modifie l'environnement alors que ce n'était pas nécessaire. Cependant, cette fonctionnalité pourrait aussi engendrer d'autres problèmes, comme la monopolisation d'un objet par l'un des utilisateurs alors qu'il est nécessaire de le modifier. C'est pourquoi une telle fonctionnalité ne serait disponible que pour les enseignants considérés comme des "super-utilisateurs" et où eux seuls pourront verrouiller un objet pour l'ensemble des élèves.

4.8.3. Récupération de l'historique des actions

Pour mieux comprendre les actions des autres utilisateurs, identifier l'auteur d'une action problématique ou garder une trace des actions ayant conduit à un bon résultat, une fenêtre d'historique est intégrée dans un coin de l'interface utilisateur pour afficher tous les changements importants effectués dans la scène. Les différents types de changements possibles sont :

- Connexion d'un utilisateur dans la scène
- Ajout d'objets dans la scène
- Mouvement/échelle/rotation d'un objet dans la scène
- Changement de valeur dans l'inspecteur d'un objet
- Changement de couleur d'un objet
- Suppression d'un objet dans la scène
- Déconnexion d'un utilisateur dans la scène

Chacune de ces actions seront décrites comme ceci :

$$Log[Numero] : [Action][NouvelleValeur][Utilisateur][Date]$$

$Numero$: Ce numéro désignera le nombre lié à ce log, afin de le différencier des autres appels fait dans l'historique.
$Action$: Cela désignera l'action faite sur le joueur, cela peut être toutes les modifications décrites ci-dessus.
$NouvelleValeur$: Cette valeur est optionnelle, elle sera utilisée pour le changement de couleur par exem-

ple. Permettant ainsi d'indiquer à l'ensemble des utilisateurs la nouvelle couleur choisie par l'initiateur du changement.
$Utilisateur$: Cela indiquera à qui appartient le changement dans la scène.
$Date$: La date liée au moment où a été mis en place le changement. Elle sera écrite en suivant le format suivant : $JJ/MM/YY\ hh:mm:ss$.

4.9. Système de rôle

La mise en place de rôles pour les utilisateurs peut permettre une meilleure gestion de la collaboration et une limitation des actions sur le serveur. Il faut garder à l'esprit que cette application doit être utilisée dans un cadre pédagogique où les utilisateurs seront soit des élèves soit des enseignants. De ce fait, ces deux profils n'utiliseront pas nécessairement l'application de la même manière ou dans le même but.

Par exemple, un enseignant peut avoir besoin d'utiliser l'application pour expliquer une notion à ses élèves sans être perturbé par des actions effectuées par les étudiants sur le serveur. Un autre scénario est possible : plusieurs élèves travaillent ensemble sur un même serveur mais il y a une mauvaise collaboration entre eux. Par conséquent, en fonction du degré de perturbation, il devrait être possible de restreindre temporairement les actions de certains utilisateurs, voire de bannir temporairement un utilisateur malveillant. Avec ces différents scénarios à l'esprit, nous avons réfléchi à l'implémentation de trois rôles :

- Élève : Destiné aux élèves, il est le rôle par défaut attribué à toute personne se connectant à un serveur déjà hébergé. En termes de droits, il peut utiliser toutes les fonctionnalités et outils de l'application de storyboarding et peut, s'il le souhaite, verrouiller la modification des objets présents dans la scène aux autres utilisateurs.

- Hébergeur : Destiné à tous types d'utilisateurs, il est le rôle attribué aux personnes créant et/ou hébergeant un serveur. En plus des droits du rôle Élève, il peut également déverrouiller la modification des objets. Il possède également des droits de gestion d'utilisateurs, tels que la possibilité de bloquer certains accès aux outils et fonctionnalités de storyboarding, le bannissement temporaire ou encore le transfert de droits. Il peut les attribuer de différentes manières :

  - Déconnexion : Lors de sa déconnexion, son rôle et tous ses droits sont légués au premier client ayant rejoint le serveur.

  - Transfert de rôle : L'hébergeur peut choisir de changer de rôle et de devenir un simple client. De ce fait, il sélectionne le nouvel hébergeur et récupère les droits correspondant à son nouveau statut.

  - Don : S'il le souhaite, il peut attribuer des droits aux personnes possédant le rôle Élève.

  Un serveur ne peut avoir qu'un unique utilisateur possédant ce rôle.

- Enseignant : Destiné aux enseignants, ce rôle possède tous les droits. Pour obtenir ce rôle sur un serveur, un utilisateur doit utiliser un code secret reconnu par l'application. Ce code doit être renseigné dans l'espace menu lorsqu'un utilisateur tente de rejoindre un serveur, de l'héberger ou encore de le

créer. Ce rôle possède les mêmes droits que l'Hébergeur, à la différence qu'il n'est pas autorisé à céder son rôle à un autre utilisateur. Un utilisateur ayant ce rôle peut également avoir simultanément le rôle d'Hébergeur s'il est le créateur du serveur ou si le précédent hébergeur lui a transmis ses droits. Dans ces cas de figure, la transmission du rôle d'hébergeur par un rôle Enseignant est similaire à celle d'un simple Hébergeur. Tout comme les autres rôles, il peut effectuer des actions sur la scène et bloquer des fonctionnalités, outils et droits aux autres utilisateurs. Étant le rôle avec le plus de droits, il a la priorité sur toutes les actions possibles sur le serveur et aucun autre rôle ne peut annuler ses actions.

4.9.1. Menu d'accueil

Lorsqu'un utilisateur lance l'application, il arrive sur une page d'accueil lui permettant différentes actions dans le but de le diriger vers le serveur de travail souhaité. Sur cette page d'accueil, plusieurs actions sont possibles :

1. Rejoindre un serveur :
   Sur la page principale de l'accueil (Figure 4.15), l'utilisateur peut retrouver une liste de tous les serveurs en cours d'utilisation. En cliquant sur le nom du serveur, il peut obtenir plus de détails sur ce dernier, notamment les pseudonymes des personnes présentes et leurs rôles. L'utilisateur peut également rechercher rapidement un serveur en renseignant son nom dans la barre de recherche. S'il souhaite rejoindre le serveur, il pourra cliquer sur le bouton "Rejoindre" et ainsi intégrer le serveur avec le rôle d'élève. Si l'utilisateur est un enseignant, il devra renseigner son mot de passe secret avant de cliquer sur le bouton, ce qui lui permettra de rejoindre le serveur avec le rôle d'Enseignant. En cas d'erreur de mot de passe, un message d'erreur s'affichera et il devra renseigner de nouveau son mot de passe ou se connecter comme un élève.

Figure 4.15.: Le menu principal

2. Création d'un serveur :
   L'utilisateur souhaite créer un nouveau serveur et l'héberger afin de démarrer un nouvel espace de travail collaboratif. Pour ce faire, il va donc cliquer sur le bouton "Créer un serveur", ce qui le redirigera

vers une page (Figure 4.16) lui permettant de nommer ce dernier. Il pourra finaliser la création en cliquant sur le bouton ”Créer” et deviendra ainsi son hébergeur. Il peut également créer le serveur en étant enseignant en renseignant son mot de passe secret avant de cliquer sur le bouton ”Créer”.

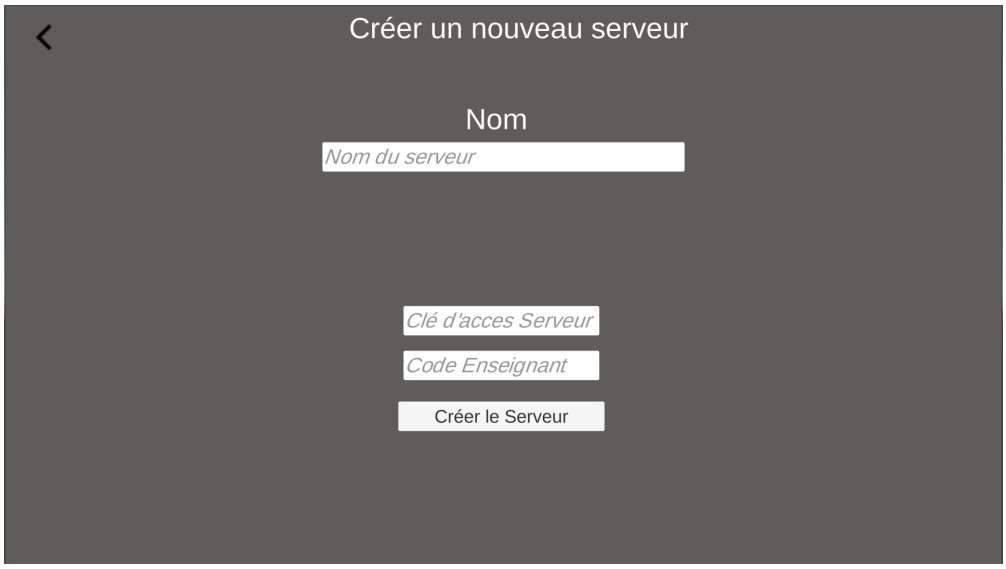


Figure 4.16.: La page de création du serveur

3. Hébergement d’un serveur :
   Sur la page principale, l’utilisateur ne voit pas le serveur qu’il recherche dans la liste des serveurs en ligne. Cependant, il possède une sauvegarde du serveur qui a déjà été utilisée par le passé. Pour utiliser sa sauvegarde et relancer le serveur, il cliquera sur le bouton ”Héberger un serveur”. Il sera alors redirigé vers une autre page (Figure 4.17) lui affichant une liste de toutes les sauvegardes qu’il possède. Tout comme sur la page précédente, il pourra sélectionner un serveur et effectuer des recherches par nom. Après avoir sélectionné la sauvegarde, il cliquera sur le bouton ”Héberger”, en choisissant éventuellement de renseigner un code enseignant.

Figure 4.17.: La page d’hébergement d’un serveur

5 RÉALISATION

5.1. Touche du clavier

L'application utilise quelques touches qui ne sont pas utilisées au sein de l'interface utilisateur, les voici :

Touches du clavier	
Z/↑	Avancer
Q/←	Gauche
D/→	Droite
S/↓	Reculer
Space	Monter
Ctrl	Descendre
F12	Screenshot
Suppr	Supprimer un objet
Alt	Faire apparaître/disparaître la souris

Pour le choix des touches, nous nous sommes grandement inspirés des contrôles traditionnels utilisés dans le domaine du jeu vidéo. Nous avons donc choisi d'associer les mouvements aux touches Z, Q, S et D. Nous avons laissé la possibilité à l'utilisateur d'utiliser les flèches directionelles dans le cas où il n'aurait pas l'habitude de ce type de contrôle. Toujours en puisant dans l'inspiration du domaine du jeu vidéo, nous avons associé les touches ***Space*** et ***Ctrl*** pour respectivement permettre de s'envoler et de descendre. Dans de nombreux jeux, la touche ***Space*** est couramment utilisée pour les sauts, tandis que la touche ***Ctrl*** est souvent associée à l'action de s'accroupir. Par conséquent, nous avons choisi d'utiliser ces touches pour des fonctions similaires au sein de notre application. Nous avons aussi choisi ***Alt*** pour une raison pratique. En effet, celle-ci est disponible, ne représente pas une lettre et reste aussi proche des autres commandes. De même, nous avons envisagé d'associer la fonction de capture d'écran à la touche ***F12*** étant donné qu'elle est souvent réservée aux fonctionnalités spéciales. Enfin, le choix de la touche ***Suppr*** pour la fonction de suppression découle de son association directe avec l'action de suppression.

5.2. Interaction dans l'espace 3D

5.2.1. Ajout d'objets dans l'espace 3D

L'apparition d'un objet dans Unity est généralement une tâche simple mais, dans notre cas, cela est plus complexe en raison de la composante réseau de notre projet. En effet, il est nécessaire de faire apparaître l'objet non seulement dans l'application de l'utilisateur mais aussi sur le serveur et dans les applications des

autres utilisateurs. Pour pouvoir faire apparaître nos objets dans la scène, nous avons dû résoudre quatre problèmes :

1. Faire en sorte que l'apparition des objets soit modulaire
2. Faire apparaître l'objet sur toutes les applications connectées sur le serveur
3. Où faire apparaître l'objet
4. Faire en sorte que l'objet n'apparaisse pas dans le sol

Le script ***DragSpawnGO*** est spécifiquement conçu dans notre projet pour gérer l'apparition des objets. Il contient la méthode ***CreateGameObjectOnCusor*** qui permet l'apparition d'objets dans la scène. La résolution du premier problème a été assez aisée. Nous avons simplement défini une variable publique de type GameObject permettant ainsi de spécifier l'objet à faire apparaître dans la scène. Cette variable peut être modifiée depuis l'interface Unity ce qui offre ainsi la possibilité de choisir dynamiquement l'objet à faire apparaître à l'aide de ce script. Concernant le deuxième problème, sa résolution n'a pas posé de difficulté majeure. Tous les objets en réseau ont accès à un objet appelé ServerManager qui facilite les interactions avec le serveur grâce à ses méthodes préexistantes. Parmi ces méthodes, nous nous sommes particulièrement intéressés à ***Spawn(NetworkObject, NetworkConnection, Scene)*** qui permet de faire apparaître un objet directement sur le serveur et de le distribuer à tous les utilisateurs connectés à une scène donnée. Cette méthode requiert trois paramètres :

1. NetworkObject est l'objet que nous souhaitons faire apparaître. Cet objet doit obligatoirement avoir un composant réseau attaché à lui.
2. NetworkConnection permet d'affecter un propriétaire à l'objet.
3. Scene représente la scène Unity. Si ce paramètre n'est pas fourni, l'objet apparaîtra dans la scène où l'appel de cette méthode a été effectué.

Bien que l'objet ServerManager soit accessible à tous les objets étant des NetworkObject, son utilisation n'est pas exempte de contraintes. En effet, les méthodes de ServerManager ne peuvent être exécutées que depuis un serveur. Dans le cadre de la méthode ***Spawn***, le NetworkObject passé en paramètre doit être instancié par le serveur exclusivement avant d'être utilisé. Pour surmonter ces contraintes, l'utilisation d'une requête RPC s'avère nécessaire. C'est précisément le rôle de la fonction ***InstantiateObjectOnScene(Vector3 spawnPosition)*** qui est une ServerRPC, c'est-à-dire une partie du code que le client demande au serveur d'exécuter. Cette fonction se divise en trois parties : la première consiste à créer l'objet sur le serveur via la méthode ***Instantiate(Object original)*** d'Unity permettant ainsi de faire apparaître un GameObject sur la scène du serveur. Ensuite, la position de l'objet est modifiée pour correspondre à celle passée en paramètre. Enfin, l'objet est transmis à **ServerManager.Spawn** afin qu'il puisse être instancié dans les autres instances de l'application.

Le troisième problème était davantage une question de méthode de résolution. La solution la plus évidente aurait consisté à faire apparaître l'objet directement devant le joueur en récupérant la position de la caméra et en décalant le point d'apparition dans la direction de son regard. Cependant, cette solution, bien que

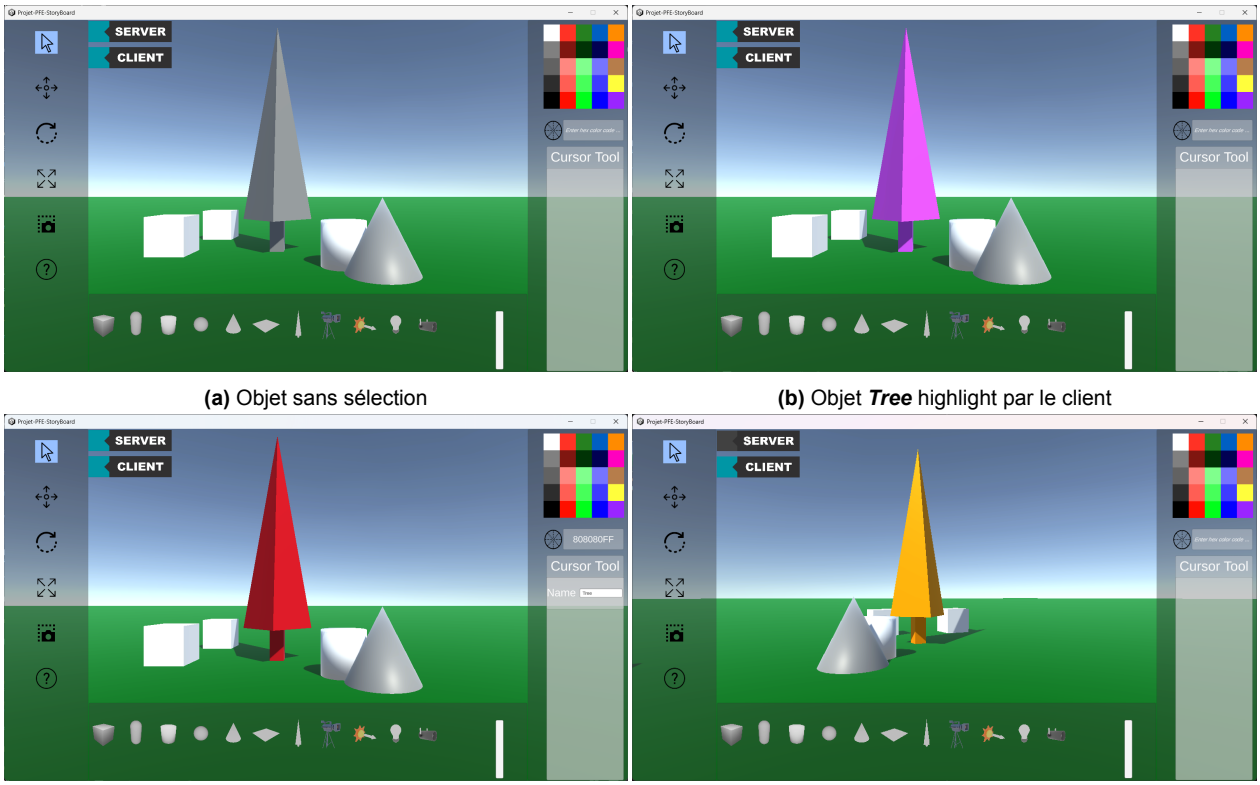
simple, manquait de flexibilité et aurait eu un impact majeur sur le flux de travail de l'application. Nous avons donc opté pour une approche différente : faire apparaître l'objet à l'emplacement du curseur. Pour ce faire, lorsque nous souhaitons faire apparaître un objet, nous lançons un rayon depuis le curseur de l'application. À cet effet, Unity fournit une méthode associée à la classe Camera, ***Camera.ScreenPointToRay(Vector3 pos)***, qui permet de convertir la position du curseur de la souris en espace écran en un rayon utilisable dans la scène 3D. Une fois le rayon construit, il suffit de l'envoyer à l'aide de la méthode ***Physics.Raycast***. Cette méthode renvoie un objet de type RaycastHit contenant toutes les informations sur le trajet du rayon. Deux informations sont particulièrement intéressantes : d'abord, la détection d'intersection avec un élément et ensuite, la position de cette intersection. Ainsi, si notre rayon touche un objet, il suffit alors d'instancier notre objet à la position de l'intersection. Si le rayon ne touche aucun objet, l'apparition de l'objet est annulée.

Le dernier problème n'a malheureusement pas pu être résolu depuis Unity. En effet, comme dans toutes les applications utilisant la 3D, les objets ont une origine qui peut être située n'importe où à l'intérieur ou à l'extérieur de l'objet. Il est donc toujours difficile de les faire apparaître correctement en fonction de leur origine. Bien qu'il existe probablement des méthodes qui ne dépendent pas de l'origine, nous n'avons pas pu en trouver durant la durée du projet. Cependant, ce problème d'origine peut être résolu de manière externe. En effet, tous les modèles actuellement utilisés dans l'application ont été créés par l'un d'entre nous. Nous avons donc décidé d'appliquer la contrainte suivante à tous nos objets : leur origine doit être située au point le plus bas du modèle. Ainsi, nous sommes toujours assurés que l'objet apparaîtra au-dessus du sol lorsqu'il est instancié. Cette solution n'est pas sans inconvénient car si à l'avenir nous souhaitons permettre aux utilisateurs d'ajouter leurs propres modèles, cette contrainte deviendra un obstacle pour eux. De plus, si l'utilisateur ne respecte pas cette règle, il est possible que son objet n'apparaisse pas correctement dans la scène. La meilleure solution à ce problème serait de trouver un algorithme ou une solution universelle qui ne dépendrait peut-être pas de l'origine de l'objet pour son apparition.

5.2.2. Sélection d'un objet

L'utilisateur peut sélectionner des objets dans la scène. Pour mettre en place ce fonctionnement, nous avons décidé d'ajouter un tag "Selectable" aux objets pouvant être sélectionnés et manipulés par l'utilisateur. Si l'utilisateur déplace sa souris sur un objet possédant le tag "Selectable", alors l'objet devient highlight et change de matériau pour visuellement notifier l'utilisateur. Un objet highlight ne l'est que jusqu'à ce que l'utilisateur enlève sa souris de l'objet. Un objet highlight peut ensuite être selected lorsque l'utilisateur appuie sur le clic gauche de sa souris. Nous avons décidé d'utiliser deux matériaux différents du matériau d'origine de l'objet pour représenter visuellement à l'utilisateur le fait que celui-ci soit highlight ou selected. Certains objets peuvent être composés de plusieurs objets nommées "objets fils". Nous avons décidé de voir la sélection comme une liste contenant l'ensemble des objets fils qui composent l'objet sélectionné par l'utilisateur. Cela permet donc de sélectionner l'objet dans son intégralité plutôt qu'une partie seulement. Dans notre liste, les objets sélectionnés se trouvent toujours en début de liste et les objets highlight en fin de liste. L'ensemble des traitements associés à la sélection d'objets et à leur manipulation après sélection se trouvent dans la classe ***Selection***. Si l'utilisateur appuie sur le clic droit de sa souris alors qu'un objet est sélectionné, celui-ci est désélectionné. L'objet retrouve alors son matériau d'origine et l'ensemble de ses objets fils dans la liste sont supprimés. Pour certains, la méthode la plus intuitive pour désélectionner un objet

serait de cliquer avec le bouton gauche de la souris sur un objet différent de celui actuellement sélectionné. Cependant, nous pensons personnellement que le fait que la désélection se trouve sur un autre clic que celui de la sélection permet d'éviter des manipulations non voulues dans le cas où de nombreux objets sont regroupés en un même endroit. Nous avons par la suite décidé d'ajouter un troisième matériau visible uniquement par les clients autres que l'utilisateur ayant sélectionné l'objet. Ce matériau permet à ces clients d'être visuellement notifiés de la sélection d'un objet par un des utilisateurs.



(a) Objet sans sélection

(b) Objet ***Tree*** highlight par le client

(c) Objet ***Tree*** sélectionné par le client

(d) Vu de l'Objet ***Tree*** sélectionné par un autre client

Figure 5.1.: Représentation du processus de sélection d'un objet

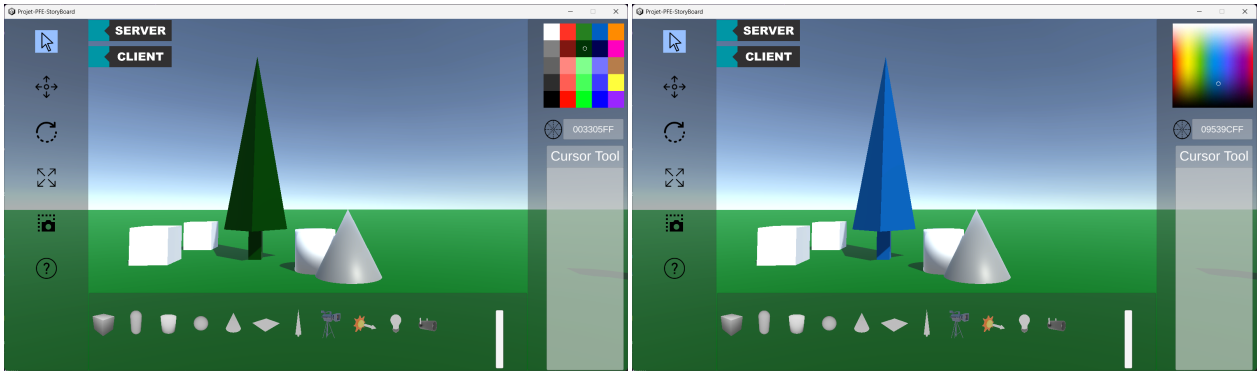
Sur l'image 5.1a, nous avons le cas où aucun client du serveur n'a cliqué sur un objet. L'image suivante 5.1b représente le cas où le client survole à l'aide de sa souris un objet, l'objet en question devient donc *highlight*, qui est ici représenté par sa couleur rose. L'image 5.1c représente la sélection de l'objet de l'image 5.1b, qui est représenté par sa couleur rouge. Enfin, du point de vue d'un autre client, l'objet sera donc coloré d'une couleur particulière, ici jaune, indiquant à cet utilisateur que l'objet est déjà utilisé et qu'il ne peut donc pas lui même le modifier, comme indiqué sur l'image 5.1d.

5.2.3. Informations de l'objet sélectionné

Les données de l'objet sélectionné (position dans l'espace, rotation, taille) sont stockées dans une classe nommée ***InformationObject***. Ces informations sont ensuite visualisables par l'utilisateur grâce à l'UI tant que l'objet est sélectionné. Ces données peuvent être modifiées en temps réel en utilisant l'UI.

5.2.4. Changer la couleur d'un objet

Suite à la sélection d'un objet, l'utilisateur peut choisir de changer sa couleur. Pour ce faire, il doit cliquer sur la couleur de son choix dans la palette mise à sa disposition qui sera appliquée au matériau de l'objet puis actualisée en temps réel. Si l'utilisateur maintient le clic actif sur la palette et déplace sa souris, le programme actualisera sa couleur tant qu'il déplacera sa souris, lui permettant ainsi de trouver celle qu'il souhaite. Quand le clic est relâché, l'objet adoptera la dernière couleur sélectionnée. Afin de permettre ce type de sélection, les palettes présentes dans le programme sont des images. Elles sont définies comme arrière-plan d'un bouton sur lequel nous effectuons un ***GetPixel(x,y)*** en fonction de la position de la souris sur l'image au moment du clic. L'un des éléments problématiques lors de la programmation a été les conversions. En effet, les différents éléments présents sur le canvas ont leur propre repère (repère local) mais les getters de position peuvent parfois exprimer la position d'un élément dans un autre repère, comme le repère monde ou encore le repère d'un objet parent tel qu'un panneau ou un canvas.



(a) Changement de couleur du ***Tree*** via la palette de 50 couleurs **(b)** Changement de couleur du ***Tree*** via le spectre colorimétrique

Figure 5.2.: Changement de la couleur d'un objet

5.2.5. Mouvements libres de l'utilisateur dans l'espace

Nous avons tout d'abord imaginé un déplacement utilisateur s'inspirant fortement des mécaniques du mode créatif du jeu vidéo *Minecraft*[1]. Dans ce mode de déplacement, les utilisateurs ne subissent pas les effets de la gravité et peuvent donc se déplacer librement sur leurs axes x et z grâce aux touches z, q, s et d ou aux flèches directionnelles. La barre espace et le contrôle gauche leur permettent de se déplacer sur l'axe y en les faisant respectivement prendre et perdre de la hauteur. Appuyer deux fois sur la barre espace de manière successive permet à l'utilisateur de changer entre le mode de déplacement où il subit la gravité et celui où il ne la subit pas. Si l'utilisateur ne subit pas la gravité, alors il devient capable de passer au travers de tous les objets. Appuyer sur la touche Maj permet à l'utilisateur de prendre de la vitesse et de se déplacer plus rapidement tant que la touche est maintenue.

Nous avons ensuite décidé de mettre en place un mouvement lié à l'endroit où pointe la souris de l'utilisateur. Il n'est alors plus nécessaire pour l'utilisateur d'appuyer sur la barre espace ainsi que le contrôle gauche pour prendre ou perdre de la hauteur puisqu'il lui suffit de regarder dans la direction dans laquelle il souhaite se déplacer et d'appuyer sur la touche z ou la flèche directionnelle haute pour avancer dans la direction souhaitée. Par exemple, s'il souhaite prendre de la hauteur, il lui suffit de regarder au dessus de lui et

[1] `https://www.minecraft.net/fr-fr`

d'avancer dans cette direction en appuyant sur la touche z ou la flèche directionnelle haute. Les touches q, s et d ainsi que les flèches directionnelles gauche, droite et basse permettent d'effectuer le mouvement qui leur est associé par rapport à la direction dans laquelle l'utilisateur regarde. L'utilisateur ne subit pas la gravité dans ce mode de déplacement mais il ne peut pas traverser des objets de la scène. Il peut cependant passer au travers des gadgets.

Par soucis de compréhension pour les néophytes, nous avons finalement décidé de revenir sur notre première idée en enlevant le fait de pouvoir remettre la gravité sur l'utilisateur. En effet, nous estimons qu'avoir deux touches supplémentaires, dont le fonctionnement est expliqué dans l'interface du bouton aide, est plus compréhensible qu'une mécanique de mouvement nécessitant moins de touche mais qui demande un certain temps d'adaptation avant de pouvoir bouger à sa guise. Le fonctionnement de la touche Maj tel que décrit précédemment est toujours valable dans cette version du déplacement.

L'association des mouvements vers l'avant, la gauche, l'arrière et la droite respectivement aux touches z, q, s et d ainsi qu'aux flèches directionnelles haute, gauche, basse et droite s'est fait automatiquement grâce à un module proposé par l'éditeur de Unity. En effet, nous avons associé au préfabriqué de l'avatar de l'utilisateur le module "character controller" qui s'occupe de l'association de ces touches aux mouvements en deux dimensions du joueur. Ces mouvements n'ont donc nécessité aucune implémentation de notre part. Cependant, pour rendre le mouvement véritablement libre, il faut qu'il se fasse en trois dimensions. Nous avons donc commencé par supprimer l'effet de la gravité sur le joueur directement depuis l'éditeur **Unity**. Pour cela, nous avons simplement supprimé le module "rigidbody" initialement associé au préfabriqué de l'avatar de l'utilisateur. L'avatar ne peut alors plus subir les effets de la gravité. Nous avons ensuite créé un script ***Player_Controller_FPS*** qui s'occupe d'associer des actions à l'ensemble des autres touches de mouvement dont nous avons pu parlé dans cette section. Pour les mouvements devant s'effectuer en tout temps ou pour les touches pouvant être maintenues pour conserver leur effet, leurs actions ont été implémentées dans la méthode ***Update***. La touche "Left ctrl" permet donc à l'utilisateur d'atterrir et la touche "Espace" de s'envoler en modifiant la valeur de l'axe y dans la direction de son prochain mouvement. La touche "Maj", quant à elle, permet de changer la vitesse du déplacement. En effet, lorsque l'utilisateur appuie sur la touche et tant qu'il la maintient, le booléen "isRunning" passe à vrai. Ce booléen implique que les mouvements horizontaux et verticaux ne sont plus multipliés par la variable "walkingSpeed" mais par "runningSpeed". Lorsque la touche est relâchée, le booléen repasse à faux.

Pour finir, nous avons mis en place un booléen "canMove" dans le même script qui permet de savoir si la souris est active pour l'utilisateur ou non. En effet, le module "Player Character" associé au préfabriqué de l'utilisateur lui permet de suivre la souris. La caméra étant également associée au préfabriqué, elle effectue le même mouvement. Nous ne voulons donc pas que le regard de l'utilisateur bouge alors qu'il essaie par exemple de sélectionner un outil dans l'interface utilisateur ou de déplacer un objet sur la scène. Le bouton "Alt gauche" permet d'alterner entre le mode sélection dans lequel la souris est active et le mode déplacement où elle ne l'est pas et où elle n'est pas visible.

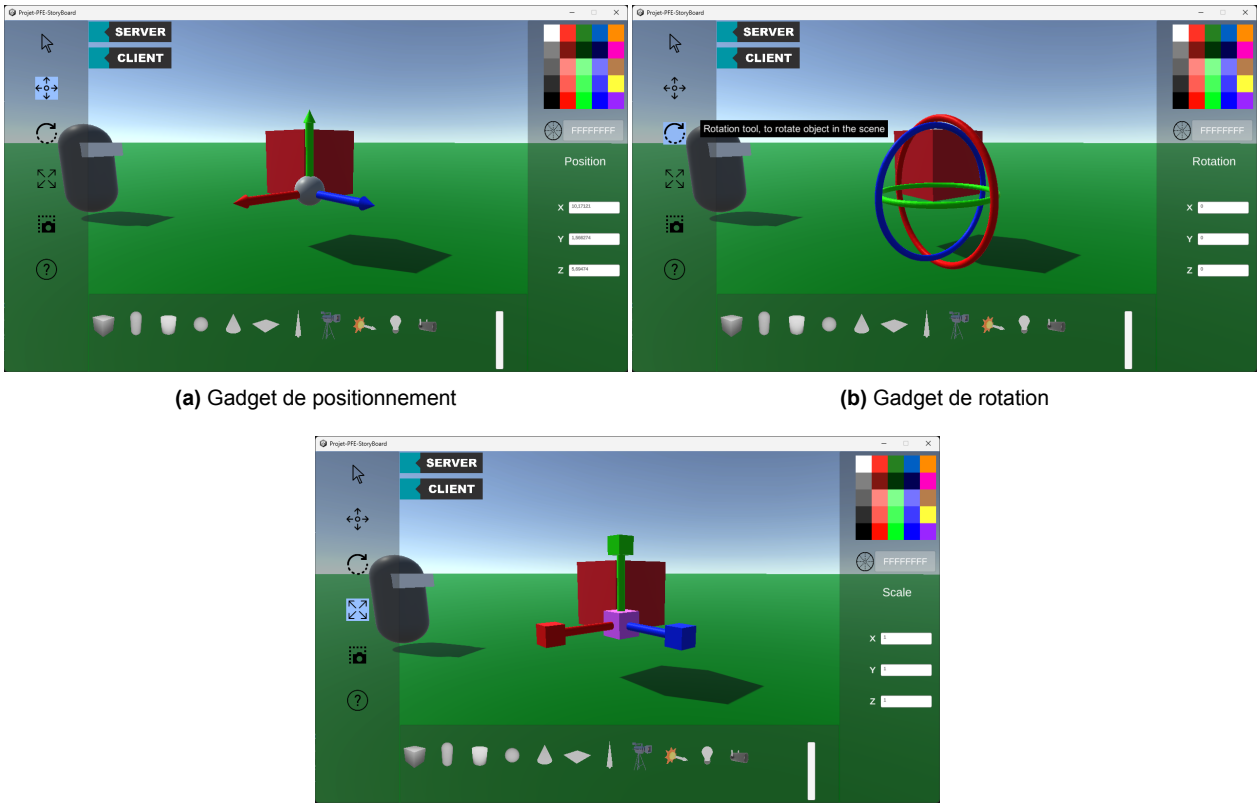
5.2.6. Manipulation libre des objets

Une fois qu'un objet a été sélectionné, l'utilisateur peut faire un clic gauche sur l'objet puis glisser son curseur pour qu'il soit déplacé librement sur l'ensemble des axes grâce à la classe ***DragGizmo***. Le mouvement à effectuer est calculé en faisant dans la méthode ***Update*** la différence entre la position courante de l'objet sélectionné et celle du curseur de l'utilisateur, toutes deux dans l'espace monde.

La première version du mouvement libre des objets sélectionnés offrait à l'utilisateur la possibilité de rapprocher ou d'éloigner l'objet de l'utilisateur grâce au mouvement de la molette souris. Plus précisément, elle rapprochait ou écartait l'objet au centre du point de vision de l'utilisateur. L'implémentation de cette fonctionnalité s'est avérée plus compliquée que prévue. En effet, cette fonctionnalité avait été pensée pour fonctionner en même temps qu'un mouvement libre et avait donc été implémentée dans la méthode ***Update*** de la classe ***DragGizmo***. Cependant, l'appel à la méthode s'effectuant à chaque frame, les deux mouvements de l'objet n'arrivaient pas à s'exécuter en même temps. Seul un des deux mouvements pouvait s'effectuer à la fois. Nous avons finalement abandonné l'idée car l'ajout du gadget de transformation permet d'avoir les mêmes fonctionnalités que celles que nous avons pu obtenir avec notre première version du mouvement libre, c'est-à-dire qu'un seul mouvement peut être effectué à la fois.

5.2.7. Ajout du gadget de transformation pour la manipulation des objets

Lorsque l'utilisateur a sélectionné un objet et qu'il a choisi un mode de transformation dans l'UI, un gadget de transformation apparaîtra pour mettre en place les transformations désirées. Pour cela, les différents gizmos seront dès le début présents dans la scène. Ils sont positionnés à un lieu qui n'est pas visible par les utilisateurs et sont désactiver pour qu'ils ne soient pas visibles sur la scène. Lorsqu'un utilisateur va vouloir sélectionner un objet et lui appliquer l'une des transformations désirées, le gadget de transformation choisi va récupérer les positions de l'objet afin de se placer en son centre et s'active pour qu'il soit visible par l'utilisateur. L'objet sélectionné ne sera plus détecté par un lancer de rayon pour que seul le gadget de transformation puisse l'être. Le lancer de rayon se fera sur l'ensemble des fils du gadget de transformation. Les fils du gadget de transformation représentent les différents axes de transformation. Si l'utilisateur clique sur l'un des axes puis applique un mouvement dans la direction de l'axe, alors l'objet va se déplacer positivement dans l'axe choisi. S'il se déplace vers une direction inverse à l'axe alors l'objet va se déplacer négativement dans l'axe choisi. Pour chaque gadget de transformation, la forme au centre permet une transformation sur l'ensemble des axes. Les trois transformations ont quelques différences de calcul. Pour le cas du transform, son déplacement prend en compte sa position initiale additionnée au mouvement appliqué à l'axe choisi. Pour le redimensionnement, son cas est assez semblable. Toutefois, il y a un cas spécial pour la forme globale. Pour ce cas, le calcul du déplacement prend en compte la somme des distances de chaque axe qui est par la suite multipliée pour chaque axe de déplacement. Pour le cas de la rotation, nous nous sommes inspirés du fonctionnement de la rotation d'Unity pour créer la nôtre. En effet, on ne prend plus en compte le déplacement mais simplement la distance et l'axe de transformation pour calculer la rotation. Les transformations sont appelées par la classe ***DragGizmoAxes*** et sont mises en place par ***NetworkManagementPrefab***.



(a) Gadget de positionnement

(b) Gadget de rotation

(c) Gadget de redimensionnement

Figure 5.3.: Les différents gadgets

5.2.8. Suppression d'un objet

Il est possible de supprimer un objet de la scène une fois qu'il est sélectionné par l'utilisateur. Pour ce faire, l'utilisateur doit sélectionner l'objet qu'il souhaite supprimer et appuyer sur la touche "Suppr" de son clavier. Cette action déclenche la méthode ***DestroyBehaviour*** de la classe ***Selection***. Tout d'abord, cette méthode désélectionne l'objet en appelant ***DeselectBehaviour*** de la même classe. Ensuite, une instance de la classe ***NetworkManagementPrefab*** est utilisée pour supprimer définitivement l'objet côté serveur et client. La méthode ***DestroyObjectServerSide(GameObject gameObject)*** de la classe ***NetworkManagementPrefab*** est utilisée pour supprimer l'objet côté serveur. Compte tenu de l'aspect réseau de notre programme, il est essentiel que la suppression soit propagée à tous les clients. Pour cela, nous utilisons la méthode ***Despawn(GameObject gameObject)*** de l'objet ***ServerManager*** qui détruit l'objet sur le serveur et informe tous les clients de cette suppression. La suppression d'objets standards ne pose généralement pas de problème. Cependant, nous avons accordé une attention particulière aux caméras, car plusieurs clients peuvent interagir avec elles simultanément. Nous avons ainsi modifié la classe ***CameraPrefabScript*** pour y ajouter une méthode ***GetisBeingWatch***. Cette méthode renvoie un booléen indiquant si la caméra est utilisée par un ou plusieurs utilisateurs. Dans le cas où la caméra est utilisée, la suppression de l'objet est bloquée pour éviter toute perturbation.

5.2.9. Récupérer ou renommer un objet

Nous avons créé la classe ***NamePrefabManager*** qui a pour but de renommer ou récupérer le nom de l'objet sélectionné. Cette classe est une classe abstraite qui a comme classe fille l'ensemble des types d'objets existants que l'utilisateur peut poser tels qu'une ***Capsule*** ou une ***PointLight***. L'utilisateur aura donc la possibilité de renommer l'objet à partir de l'UI.

Lorsque l'utilisateur ajoute un objet à la scène, l'objet aura un nom prédéfini. Son nom est le nom de l'outil posé plus un numéro s'il y en a plusieurs sur la scène. Si par exemple, l'utilisateur pose un objet capsule pour la première fois, cette objet se nommera "Capsule". S'il repose un objet capsule, ce second objet se nommera "Capsule (1)". La numérotation se fait grâce à un entier déclaré comme static se trouvant dans toutes les classes filles qui va s'incrémenter à chaque appel du start de la classe. La classe mère a pour but de récupérer directement le nom ou le nom de base de l'objet à l'aide des méthodes ***GetName*** et ***GetBaseName***. Elle permet aussi de changer au niveau du serveur et des différents clients le nom de l'objet à l'aide de la méthode ***SetNameServeurSide***.

Afin de palier à une contrainte du réseau, nous avons mis en place un booléen déterminant si l'objet a été modifié avant l'arrivée d'un autre utilisateur. Ce qui va permettre de ne pas écraser le nom modifié de l'objet par son nom originel.

5.3. Caméra, création et manipulation dans l'espace 3D

Les caméras sont un élément clef du projet. En effet, c'est au travers de ces caméras que les utilisateurs pourront visualiser et manipuler leurs différents plans. Les caméras doivent être aussi simples d'utilisation que possible afin de favoriser la créativité et de réduire le temps passé à travailler sur le plan. Nos caméras devront posséder plusieurs fonctions :

1. Possibilité de prendre le point de vue de la caméra
2. Possibilité de comparer le point de vue de plusieurs caméras
3. Possibilité de prendre une capture d'écran
4. Possibilité d'appliquer à chaque caméra une série de filtres
5. Visualiser les points de fuite

5.3.1. Construction des caméras

Les caméras proposées par notre application sont construites de manière précise pour pouvoir assurer toutes ces fonctionnalités. Le premier élément de nos caméras est un modèle 3D d'une caméra. Ce modèle a été réalisé par notre équipe en s'inspirant largement des caméras de cinéma moderne tout en ajoutant un aspect rétro au design de la caméra. En effet, nous avons choisi de positionner la caméra sur un trépied. Cette décision est justifiée par le souci de ne pas déstabiliser l'utilisateur avec une caméra volante. La taille du modèle est ajustée de manière à ce que, lorsque les pieds de la caméra touchent le sol, le point de vue fourni par cette caméra soit le même que si l'utilisateur était à l'emplacement de la caméra. Cependant,

cette approche n'est pas sans problèmes car certains plans tels que les contre-plongées ou les plongées nécessiteront une caméra dans les airs. Dans ces situations, le trépied devient plus un inconvénient qu'un atout. Une solution pourrait être de proposer une version de la caméra sans trépied.

Figure 5.4.: Modèle 3D de la caméra

Au-delà de l'aspect visuel, notre caméra est composée d'éléments invisibles pour l'utilisateur. En réalité, notre caméra est composée de deux caméras distinctes. La première, nommée ***PlayerViewCam*** est utilisée lorsque l'utilisateur veut prendre le point de vue de la caméra. La seconde caméra nommée ***RenderTextureViewCam*** est plus particulière. En effet, cette caméra ne renvoie pas directement ce qu'elle voit mais transcrit ce qu'elle voit dans une texture/image. Cette caméra nous permet facilement de réaliser des captures d'écran ou de projeter son visuel sur différents objets dans la scène. L'existence de cette seconde caméra découle d'une limitation de Unity. En effet, une caméra ne peut pas être utilisée comme vue principale de l'utilisateur tout en créant une texture ou une image de ce qu'elle voit. Ce problème était particulièrement délicat lors de la mise en place de la comparaison des points de vue. Un autre élément indirectement caché aux utilisateurs est constitué des objets ***FrustrumClient*** et ***FrustrumServer***. Ces objets sont utilisés pour définir les points de fuite côté client et serveur. Le dernier élément qui compose physiquement notre caméra est un ***Volume*** fourni par Unity. Ce volume est une boîte permettant d'appliquer des effets de post-processing sur les caméras se trouvant à l'intérieur. Nos deux caméras, ***PlayerViewCam*** et ***RenderTextureViewCam*** se trouvent à l'intérieur de ce volume.

Il existe également deux scripts attachés à nos caméras. Le premier, ***CameraPrefabScript***, est un script qui permet de manipuler et de déclarer aisément une caméra dans notre espace 3D. Le deuxième script, ***CameraVanishingPoint***, est entièrement dédié à la création et à la manipulation des points de fuite des caméras.

5.3.2. Ajout de caméras

L'ajout d'une caméra dans la scène se fait comme tout autre objet ajoutable dans la scène. Il existe un bouton dans la partie inférieure de l'UI permettant de drag and drop une caméra dans la scène. Cependant, comparée aux autres objets, chaque caméra doit déclarer son existence au Camera Manager présent dans la scène.

5.3.3. Camera Manager

Le gestionnaire de caméra est un objet invisible dans la scène qui recense toutes les caméras présentes. Cet objet est constitué d'un seul script complexe portant le même nom. Son rôle est de contrôler et de manipuler tous les éléments liés aux caméras dans la scène. Ce script possède plusieurs éléments indispensables au bon fonctionnement des caméras et des éléments d'interface qui y sont liés :

- Une liste contenant tous les filtres applicables aux différentes caméras.
- Une liste de toutes les caméras dans la scène.
- Une référence à l'utilisateur.
- Une référence au script contrôlant les sélections.
- Une référence à l'interface de sélection des caméras.
- Une référence à l'interface de comparaison des caméras.
- Une liste des caméras ayant un filtre appliqué sur elles.

Ce script a trois objectifs principaux :

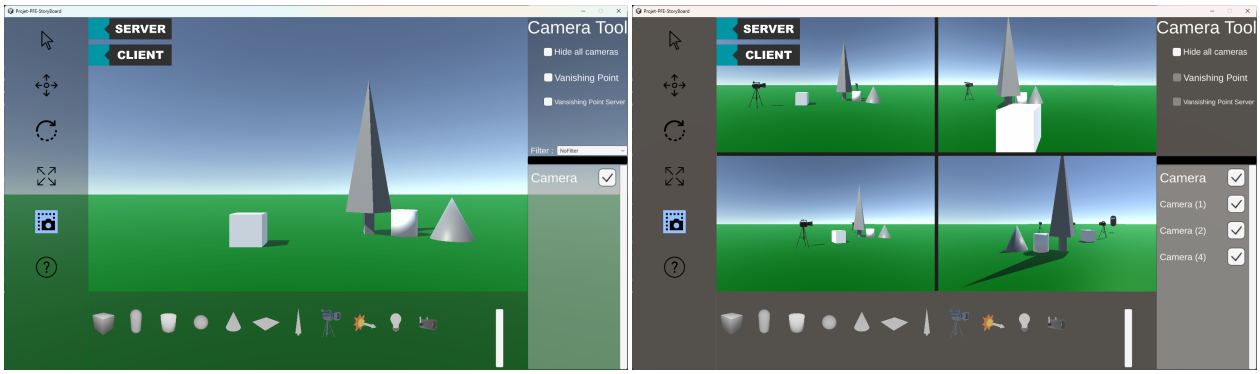
1. Assurer la liaison entre les caméras et les autres codes nécessitant les caméras.
2. Assurer le bon fonctionnement des filtres.
3. Assurer le bon fonctionnement du mode de comparaison des points de vue.

De nombreux codes utilisent des lancers de rayons pour fonctionner comme c'est le cas de la sélection. Or, la caméra actuellement utilisée par l'utilisateur n'est pas forcément celle qui est attachée au corps de l'utilisateur. Il peut par exemple manipuler la scène à travers l'une des caméras existantes dans la scène. Le gestionnaire de caméra contient toujours la caméra courante de l'utilisateur et cette caméra est facilement obtenable via la méthode ***GetCurrentCamera***.

Un autre aspect fondamental du script CameraManager réside dans sa capacité à garantir le bon fonctionnement des filtres de traitement d'images associés à chaque caméra. En tirant parti des filtres définis dans ***listOfPostProcessingEffect***, le script permet l'application individuelle des filtres sur les caméras présentes dans la scène. La méthode ***ApplyFilterToCurrentCamera*** permet d'appliquer un filtre à la caméra courante de l'utilisateur en se basant sur l'indice du filtre dans ***listOfPostProcessingEffect***.

La comparaison des points de vue est également assurée par cet objet. Les méthodes ***ActivateCameraToRender*** et ***DeactivateCameraToRender*** permettent de changer et d'afficher plusieurs caméras. Cette classe adapte son comportement au nombre de caméras que l'utilisateur souhaite visualiser. Si l'utilisateur sélectionne une seule caméra, la vue de l'utilisateur devient alors celle de la caméra. Si plusieurs caméras sont activées, le script active l'interface de comparaison et applique le visuel de la caméra à un des quatre panneaux de comparaison disponibles. Cette affectation des panneaux de comparaison est réalisée par un

autre script, le script ***CompareCameraPannelScript*** qui est attaché à l'interface de comparaison dans la scène.



(a) Point de vue d'une caméra **(b)** Comparaison de plusieurs point de vue de caméra

Figure 5.5.: Comparaisons des points de vue

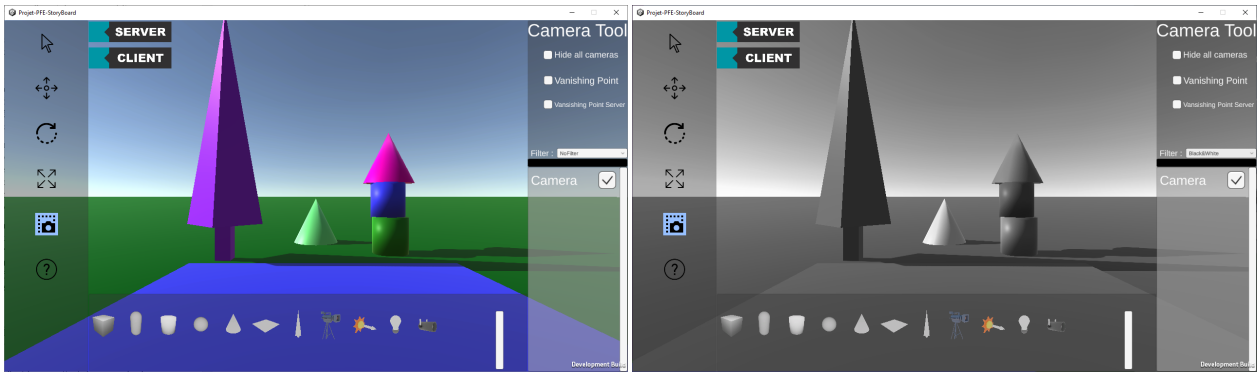
5.3.4. Cacher les caméras

L'utilisateur peut choisir de cacher les caméras disposées sur la scène à l'aide d'une option visible sur l'interface utilisateur lorsqu'une caméra est sélectionnée. Une fois cette option sélectionnée, l'ensemble des caméras placées sur la scène ne seront plus visibles jusqu'à ce que l'utilisateur en décide autrement. Les caméras fonctionneront toujours de la même manière, seul l'objet caméra ne sera plus visible sur la scène. Cette option permettra à l'utilisateur d'avoir une vision claire de la scène et de ses objets, ce qui peut notamment être utile lorsque l'utilisateur souhaite comparer les points de vue des différentes caméras sur la scène sans être gêné par la présence d'objets caméras.

5.3.5. Les filtres des caméras

Il existe plusieurs façons d'appliquer des filtres sur les caméras dans Unity. Nous avons décidé d'utiliser des effets de post-processing avec l'Universal Render Pipeline (URP) fourni par Unity. Ce pipeline permet d'appliquer une grande variété d'effets de post-processing à différentes caméras de notre scène. Pour appliquer ces effets à nos caméras, deux solutions sont possibles : soit appliquer directement un effet à chaque caméra, soit appliquer l'effet à un ***Volume*** qui l'applique à toutes les caméras qu'il contient. Nous avons opté pour la seconde option car nous avons deux caméras, ce qui facilite grandement la manipulation des filtres.

L'URP permet d'utiliser des ***Volume Profiles***, une manière simple de sauvegarder les effets de post-processing. Ces profils de volume peuvent ensuite être directement appliqués aux volumes de nos caméras. Ainsi, depuis Unity, il est possible de créer rapidement et facilement un grand nombre d'effets de post-processing et de filtres pour nos caméras.



(a) Sans filtre **(b)** Avec le filtre noir et blanc

Figure 5.6.: Vue d'une caméra en fonction des filtres

5.3.6. Capture d'écrans

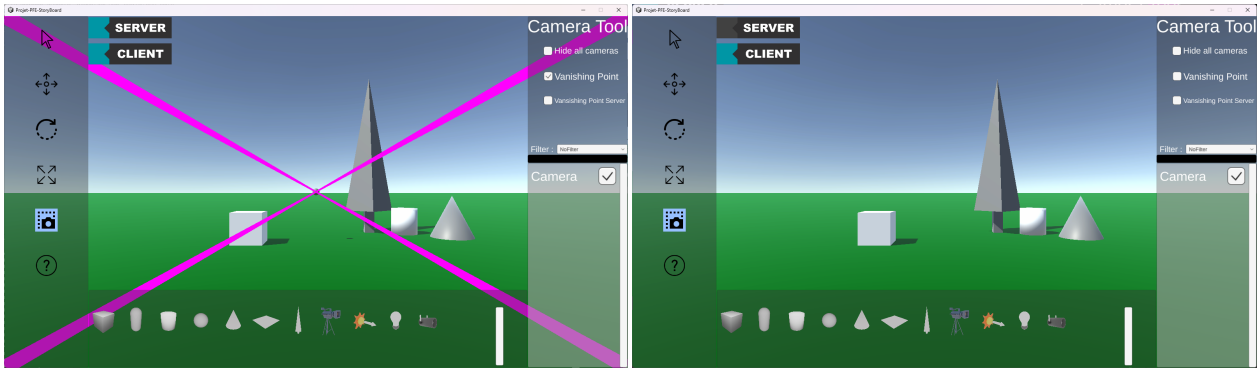
Un aspect clé de notre application est la capture d'écran. Ces captures permettent de conserver une preuve du travail réalisé tout en offrant aux utilisateurs la possibilité de l'utiliser dans d'autres applications. La capture d'écran doit prendre en compte trois éléments : tout d'abord, la gestion de l'utilisation de différentes caméras ; ensuite, la prise en compte des filtres de la caméra ; enfin, la nécessité que la capture d'écran n'affiche pas l'interface utilisateur mais seulement la scène elle-même. La prise en compte de la seconde partie est indirectement assurée par le ***Volume*** présent dans chacune de ces caméras.

Pour procéder à la capture d'écran, deux solutions pouvaient être facilement mises en place. La première consiste à utiliser la classe ***ScreenshotHandler*** fournie par Unity. Cette classe permet de créer une reproduction fidèle de ce que voit l'utilisateur. Bien que cette solution soit la plus simple, elle ne correspond pas totalement à nos besoins car si l'on utilise simplement ***ScreenshotHandler***, la capture d'écran obtenue inclut également l'interface utilisateur, ce qui n'est pas souhaitable. Une solution à ce problème existe, mais elle n'est pas des plus correcte. Cette solution impliquerait de passer en référence au script qui gère les captures d'écran tous les ***Canvas*** représentant l'interface utilisateur. Avec ces références, le script pourrait temporairement masquer l'interface, prendre la capture d'écran, puis réactiver l'interface une fois la capture d'écran effectuée. Une autre solution pour prendre une capture d'écran tout en évitant de capturer également l'interface utilisateur serait de réaliser la capture d'écran à la fin du rendu d'une frame. Dans Unity, l'interface est le dernier élément à être rendu à l'écran. Cette approche permettrait de capturer uniquement la scène en ignorant l'interface utilisateur. Cependant, cette solution élimine tous les éléments d'interface. Si, à l'avenir, certains éléments de l'interface utilisateur tels que du texte devaient apparaître sur les captures d'écran, cette méthode de capture d'écran ne serait pas optimale.

5.3.7. Création et manipulation de points de fuite

L'utilisateur peut faire apparaître un point de fuite dans la vision de la caméra sélectionnée. Pour faire apparaître ce point de fuite, nous utilisons la classe ***CameraVanishingPoint***. Dans un premier temps, on récupère la caméra enfant ***"RenderTextureViewCam"*** de l'objet caméra. Ensuite, on va lui assigner une profondeur que l'utilisateur pourra choisir dans l'UI. On va mettre en place quatre gameObject LineRender qui vont servir de ligne pour le point de fuite créé et on va instancier une sphère qui va représenter ce

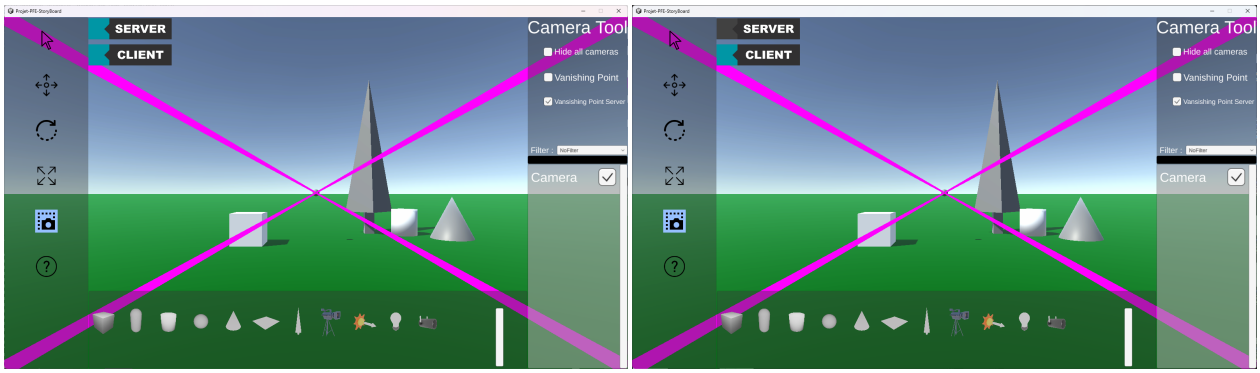
point de fuite. Dans la méthode ***Update*** de la classe, on récupère les coins de la vue caméra en espace monde, pour pouvoir ensuite créer des lignes reliant ces points à la sphère créée grâce au LineRenderer. Le ***Update*** permet de faire en sorte que le point de fuite suive la vision de la caméra en temps réel. Pour cela, nous avons décidé de créer un objet parent frustum regroupant les quatre lignes et le point de fuite que nous décalons le long de l'axe x de la caméra d'une distance choisi par l'utilisateur via l'interface utilisateur. L'ajout d'un point de fuite permettra à l'utilisateur d'avoir une meilleur vision de la perspective depuis la caméra pour l'assister dans le placement des objets sur la scène.



(a) Apparition des points de fuite vue client seul **(b)** Point de vue d'un autre client de la même caméra

Figure 5.7.: Visualisation des points de fuites depuis une caméra

Toutefois, cette méthode n'est fonctionnelle qu'en client. Selon le choix de l'utilisateur, les points de fuite peuvent être rendus visibles uniquement pour lui-même ou pour tous les utilisateurs. Nous avons retravaillé cette fonctionnalité afin qu'elle puisse être utilisable en réseau. Dans la classe ***CameraVanishingPoint***, nous avons enlevé la méthode ***Update*** pour la remplacer par un ensemble de méthode. Dans un premier temps, nous lions deux gameObjects nommés ***FrustumClient*** et ***FrustumServer*** à toutes les caméras. Les frustrums suivront ainsi parfaitement les mouvements de la caméra. A chacun de ces frustums est lié un gameObject ***VanishingPoint***. Par la suite, ***CameraVanishingPoint*** va appeler une méthode ***InitFrustums*** qui va servir à initialiser les deux frustums ***FrustumClient*** et ***FrustumServer*** à l'aide d'un appel à la méthode ***InitFrustumsClient*** et ***InitFrustumsServer***. Ces deux méthodes vont servir à appeler l'initialisation des lignes du frustum et de la sphère qui servira de point d'arrivée des lignes. L'initialisation côté serveur se fera dans la classe ***CameraPrefabScript*** qui appellera la classe et sa méthode ***InitFrustums***. Dans cette classe, nous allons ajouter deux méthodes ***ToggleFrustumClient*** et ***ToggleFrustumServer*** qui serviront respectivement à activer ou désactiver les frustums côté client et serveur. Ces deux méthodes seront appelées dans la classe ***Sélection*** lorsque l'utilisateur aura activé dans l'UI l'apparition des frustums côté client seul ou serveur.



(a) Apparition des points de fuite vue côté serveur **(b)** Point de vue d'un autre client de la même caméra

Figure 5.8.: Visualisation des points de fuite en dehors d'une caméra

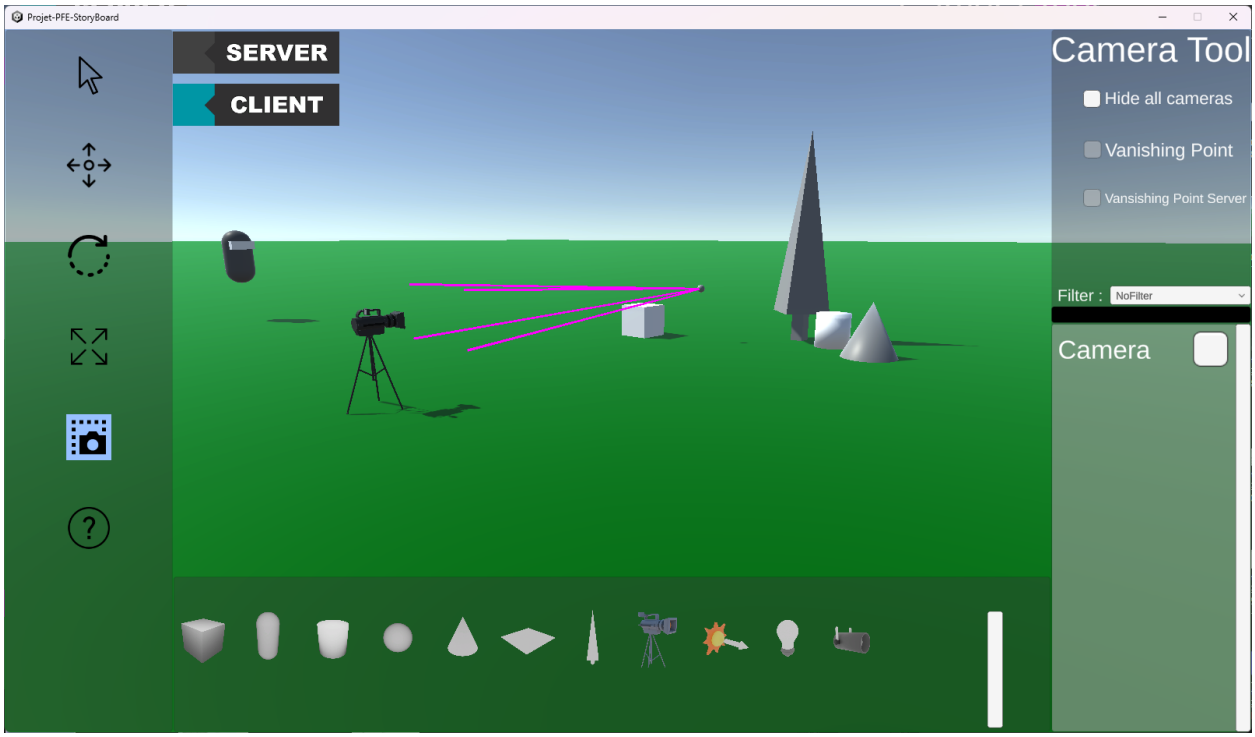


Figure 5.9.: Point de vue d'un client du point de fuite mis sur la caméra

5.4. Lumière, création et manipulation dans l'espace 3D.

La lumière est également un élément important dans le dessin, parfois complexe à comprendre pour les étudiants et artistes. Il est donc essentiel que leur utilisation soit claire et simple.

Unity fournit trois types de lumières qui peuvent être utilisées et manipulées directement depuis une scène :

1. Les lumières ponctuelles (*Point Light*) qui éclairent autour d'elles dans un certain périmètre.
2. Les projecteurs (*Spot Light*), des lumières qui éclairent dans un cône devant elles.

3. Les lumières directionnelles (*Directional Light*) qui éclairent toute la scène dans une certaine direction.

5.4.1. Visuel des lumières

Pour faciliter la reconnaissance des différents types de lumière dans la scène, il était important pour nous que chacune des lumières disposent d'un modèle distinctif permettant de les différencier au premier coup d'œil et de comprendre de quel type de lumière il s'agit. Pour ce faire, nous avons créé trois modèles 3D pour les trois types de lumières.

Pour la lumière ponctuelle (*Point Light*), nous nous sommes une fois de plus inspirés de Blender et de Unity. Dans ces logiciels, ce type de lumière est représenté par une petite ampoule, ce qui correspond parfaitement au fonctionnement de ce type de lumière. Nous avons donc réalisé un modèle simple d'ampoule pour représenter cette lumière.

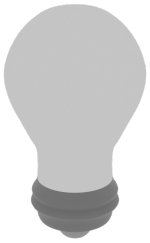


Figure 5.10.: Modèle 3D de la Lumière ponctuelle (*Point Light*)

Pour les projecteurs (*Spot Light*), nous avons directement puisé notre inspiration dans leur nom. Nous avons réalisé un modèle 3D d'un spot lumineux en nous inspirant des spots de cinéma ou de théâtre. Ces types de spot ont un aspect visuel simple mais distinct des autres modèles de lumière.

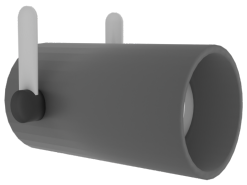


Figure 5.11.: Modèle 3D des projecteurs (*Spot Light*)

Pour les lumières directionnelles (*Directional Light*), nous nous sommes encore inspirés de Blender et Unity. Ils représentent ce type de lumière par un soleil, ce qui correspond également parfaitement au fonctionnement de cette lumière. Ainsi, nous avons concrétisé cette idée en créant un modèle très basique d'un soleil composé d'une sphère centrale et de contours pointus comme dans les dessins des jeunes enfants.

Nous avons également ajouté une grande flèche à l'avant du modèle. Cette flèche permet d'indiquer de manière claire la direction de la lumière.

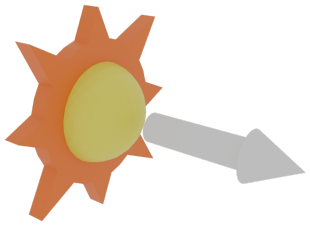


Figure 5.12.: Modèle 3D de la lumière directionnelle (*Directional Light*)

5.4.2. Ajout de lumières dans la scène

L'ajout de lumières dans la scène se déroule de la même manière que l'ajout d'autres objets dans l'environnement. Un bouton dédié est disponible dans la partie inférieure de l'interface utilisateur permettant de faire glisser et déposer (drag and drop) les différentes lumières dans la scène.

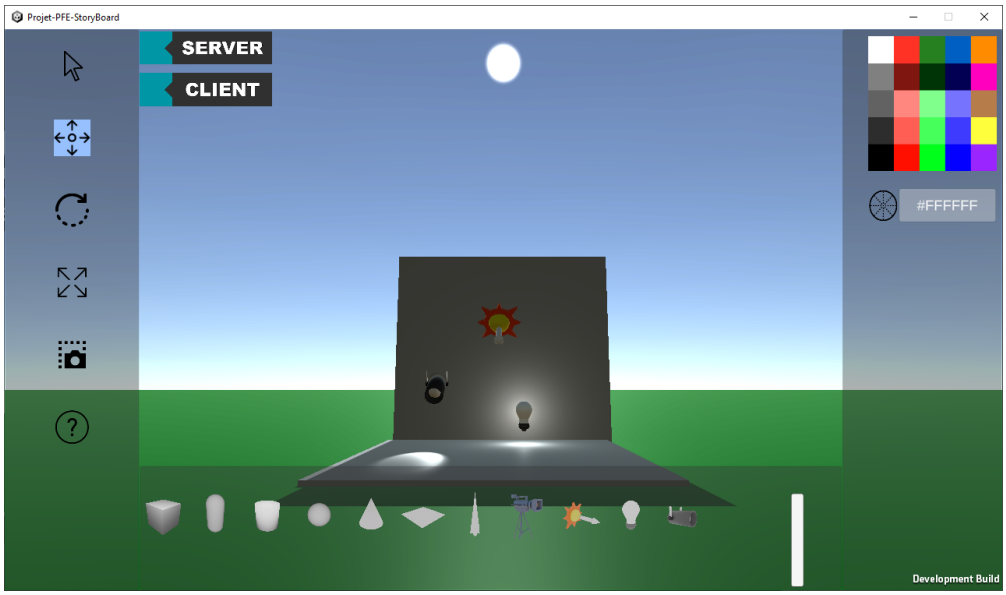


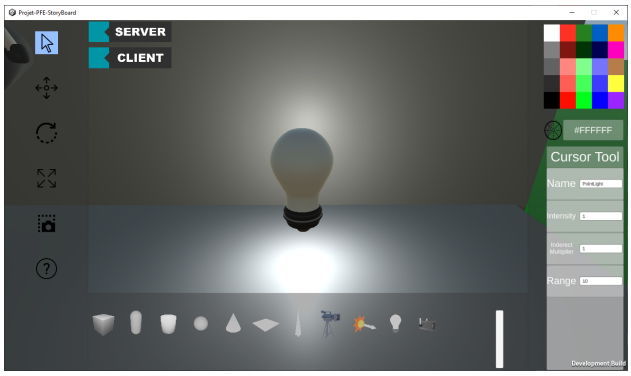
Figure 5.13.: Les trois lumières placées dans une scène

5.4.3. Modification des lumières

A l'aide de l'outil curseur, il est possible de modifier les champs les plus importants de la lumière. Lorsqu'une lampe est sélectionnée avec cet outil, des zones de saisie apparaissent sur l'interface à droite indiquant le nom du champ et sa valeur. Les champs actuellement modifiables dans notre application sont les suivants :

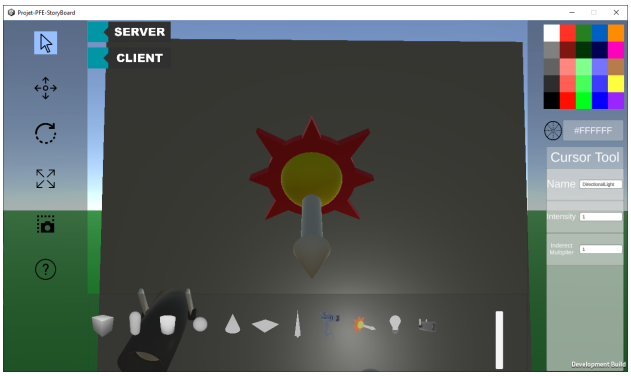
- La puissance de la lumière.

- La portée de la lumière.
- Le nombre de rebonds de la lumière sur les objets.



(a) Propriétés uniques des lumières ponctuelles

(b) Propriétés uniques des spots lumineux



(c) Propriétés uniques des lumières directionnelles

Figure 5.14.: Visualisation des différentes propriétés uniques des lumières

Il est également possible, comme pour tous les autres objets de la scène, de manipuler la couleur de la lumière. Cela se fait en sélectionnant une couleur comme pour un objet dans la scène. Cependant, au lieu de modifier la couleur du modèle 3D, c'est la couleur de la lumière qui est modifiée.

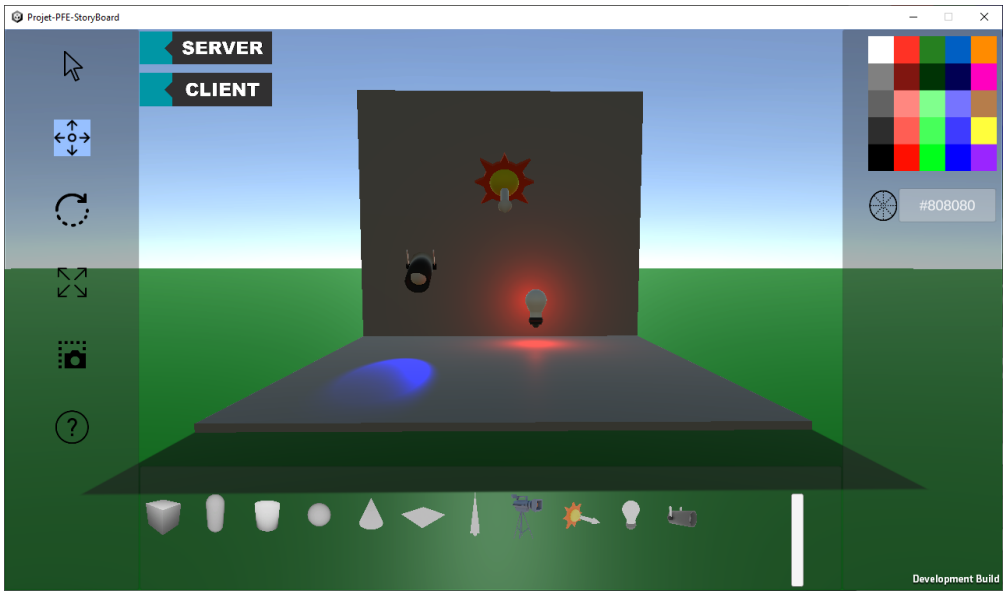


Figure 5.15.: Les trois lumières aux couleurs changées

Toutes les modifications sont possibles grâce au script ***NetworkManagementPrefab***. À la différence des autres objets utilisant ce script, les lumières sont déclarées comme un ***NetworkManagementPrefab*** de type LIGHT via un booléen. Ce booléen permet aux codes utilisant les ***NetworkManagementPrefab*** de différencier les différents types d'objets. Il existe dans ***NetworkManagementPrefab*** un ensemble de méthodes permettant de modifier directement les champs côté serveur et côté client.

5.5. Serveur et Client

5.5.1. Récupération de l'historique des actions

Notre système intègre un mécanisme d'historique des actions, également connu sous le nom de *log*, afin de suivre l'ensemble des événements collaboratifs se produisant dans la scène. À cette fin, nous avons développé la classe ***LogHistory*** conçue pour créer et enregistrer un fichier log. Cette classe propose d'enregistrer quatre types d'actions distinctes : les actions simples sans valeur numérique, les actions impliquant une valeur numérique, les actions de placement d'objets dans la scène ainsi que les connexions et déconnexions des utilisateurs de la scène. Pour ce faire, un objet nommé "Log" sera intégré à la scène et directement lié à la classe ***LogHistory***. Cet objet permettra aux éléments exécutant ces actions d'écrire facilement dans le fichier log. Notre étape suivante consistait à adapter les classes existantes pour prendre en charge l'écriture dans les logs. Nous avons commencé par la classe ***Selection*** qui représente une part importante des logs. Cette classe comporte deux méthodes appelées toutes deux ***CallLog*** qui permettent respectivement de déclencher une action simple sans valeur numérique et une action avec une valeur numérique. Grâce à cette classe, nous pouvons aisément détecter la suppression d'un objet, son déplacement avec ou sans outils, les changements de couleur de l'objet ainsi que les modifications effectuées dans l'inspecteur de l'objet. Un autre avantage significatif de cette classe pour les logs est sa capacité à identifier l'utilisateur à l'origine de ces actions. Ensuite, nous avons travaillé sur l'ajout de l'enregistrement des apparitions d'objets dans les logs et c'est la classe ***DragSpawnGO*** qui est chargée de cette tâche. Nous avons

donc ajouté une nouvelle méthode ***CallLog*** qui est invoquée lorsque qu'un utilisateur fait apparaître un objet. Cependant, en raison du fonctionnement de l'apparition au sein de cette classe, l'identification de l'auteur était plus complexe. En effet, l'apparition de l'objet et donc l'appel de ***CallLog*** doivent être réalisés sur le serveur car l'apparition de l'objet a lieu au niveau du serveur. Il est donc impossible de procéder comme dans ***Selection*** pour récupérer l'auteur de l'action. Pour résoudre ce problème, nous avons modifié ***InstiateObjectOnScene*** afin qu'elle prenne en paramètre l'identifiant de la personne réalisant l'action sous la forme d'un string. Avec le recul et les connaissances que nous avons aujourd'hui, il aurait été possible d'utiliser le paramètre optionnel des RPC, ***NetworkConnection conn***, qui théoriquement contient l'identifiant de l'utilisateur réalisant l'action. La dernière étape a consisté à ajouter les logs relatifs à la connexion et à la déconnexion des utilisateurs. Pour cela, nous avons ajouté deux nouvelles méthodes, ***CallLogConnection*** et ***CallLogDisconnection***, à la classe ***FPSPlayerControler*** qui représente l'utilisateur et son corps sur le serveur. Ces méthodes sont respectivement appelées lors de l'arrivée et le départ d'un utilisateur de la scène. Pour la connexion, nous vérifions lors de la première frame avec la méthode ***Update*** si l'utilisateur est enregistré dans les logs à l'aide du booléen ***hasBeenDeclared***. Si le booléen est faux, nous appelons alors ***CallLogConnection*** et mettons le booléen à vrai. Pour la déconnexion, nous devons enregistrer un log avant que l'utilisateur et son corps ne soient détruits par le serveur. ***Fish-Net*** propose une méthode ***OnDespawnServer*** à redéfinir. Cette méthode est appelée avant que l'objet ne soit détruit par le serveur et nous permet ainsi d'appeler ***CallLogDisconnection*** et de définir l'identifiant unique de l'utilisateur se déconnectant. À l'origine, nous avions l'intention de rendre les logs accessibles via l'interface de l'application. Malheureusement, en raison de contraintes de temps, cette fonctionnalité n'a pas pu être implémentée. Cependant, l'historique des logs est disponible dans les données du programme sous le nom de ***Log.txt***.

5.6. Interface Utilisateur

5.6.1. Barre d'outils

Pendant le développement de l'application, la barre d'outils a subi une série de changements. Le premier changement majeur concerne la position du bouton de couleur. En effet, l'usage d'un outil de couleur semblait de moins en moins pertinent, surtout parce que la seule fonction de ce bouton était de modifier la palette de couleur. Ainsi, le bouton de couleur a été retiré de la barre d'outils pour être placé en dessous des palettes de couleurs. Cette position est en corrélation avec sa fonctionnalité, ce qui facilitera son association par l'utilisateur. Le deuxième changement majeur concerne le bouton d'aide. Lui aussi, au fil du développement, a perdu son statut d'outil. Il a semblé plus judicieux de le séparer des autres outils. Bien que sa position soit restée inchangée, son fonctionnement n'est plus lié aux autres boutons de la barre d'outils. Le dernier changement concerne le bouton d'interface déroulante. Au fur et à mesure du développement, l'intérêt d'un tel bouton nous a semblé limité. En plus de l'intérêt limité de cette fonctionnalité, le temps nécessaire pour l'implémenter aurait été important et aurait limité notre temps sur des fonctionnalités plus essentielles du projet. Par conséquent, ce bouton et sa fonctionnalité ne sont pas présents dans le projet final.

5.6.2. Interface redimensionnable

L'un des premiers éléments conçus dans notre interface est la capacité d'agrandir ou de réduire la taille des différents éléments de l'interface. Au cours du projet, deux implémentations de cette fonctionnalité ont été envisagées. La première version consistait à utiliser une énumération pour indiquer de quel côté de l'interface il était possible de sélectionner pour redimensionner cet élément. Ensuite, une valeur numérique en virgule flottante était définie pour indiquer la taille de la zone sur laquelle l'utilisateur peut cliquer pour redimensionner la fenêtre. Avec ces valeurs en place, il était alors possible de calculer une zone où la souris permettait le redimensionnement. Pour cela, il nous suffisait de calculer la position du bord décrit par l'énumération et de la garder en mémoire. Si la position de la souris de l'utilisateur lors d'un clic se trouvait entre ce bord et la taille de la zone décrite par la valeur flottante, nous suivions alors l'évolution de distance pour agrandir ou rétrécir l'élément graphique sélectionné. Cependant, nous avons rencontré un problème majeur avec cette implémentation. Pour définir les coordonnées dans un espace d'écran, Unity offre un système d'ancrage, utilisé pour permettre à l'interface de s'adapter à toutes les résolutions. Cependant, les coordonnées des éléments sont exprimées en fonction des points d'ancrage. Ainsi, un rectangle positionné quelque part à l'écran n'a pas les mêmes coordonnées selon ses ancrages, que ce soit au centre de l'écran ou dans un coin. Cela nous a posé deux problèmes majeurs :

1. En raison des différents ancrages, les coordonnées de l'interface pouvaient varier.
2. Forcer l'ancrage des panneaux aurait diminué la flexibilité du code et réduit nos capacités à rendre l'interface réactive.

Le premier problème était le plus préoccupant. En effet, les coordonnées variant en fonction du point d'ancrage, il était particulièrement difficile de calculer la position réelle de la zone de redimensionnement de la fenêtre. Cela rendait complexe la détermination précise de la position du bord de l'interface graphique. La seule solution pour résoudre ce problème aurait été de forcer les points d'ancrage, mais même avec cela, la précision des zones de redimensionnement restait peu fiable et parfois imparfaite. Nous avons donc dû chercher une solution qui ne dépendait pas des coordonnées des éléments de notre interface graphique pour calculer la zone de redimensionnement. Nous avons donc mis en place une deuxième version du redimensionnement qui est la version toujours utilisée dans notre code. Dans cette version, il est toujours nécessaire de déclarer au moyen d'une énumération de quel côté de l'élément graphique le redimensionnement est possible. Dans le cadre de notre code, seules quatre méthodes de redimensionnement sont possibles :

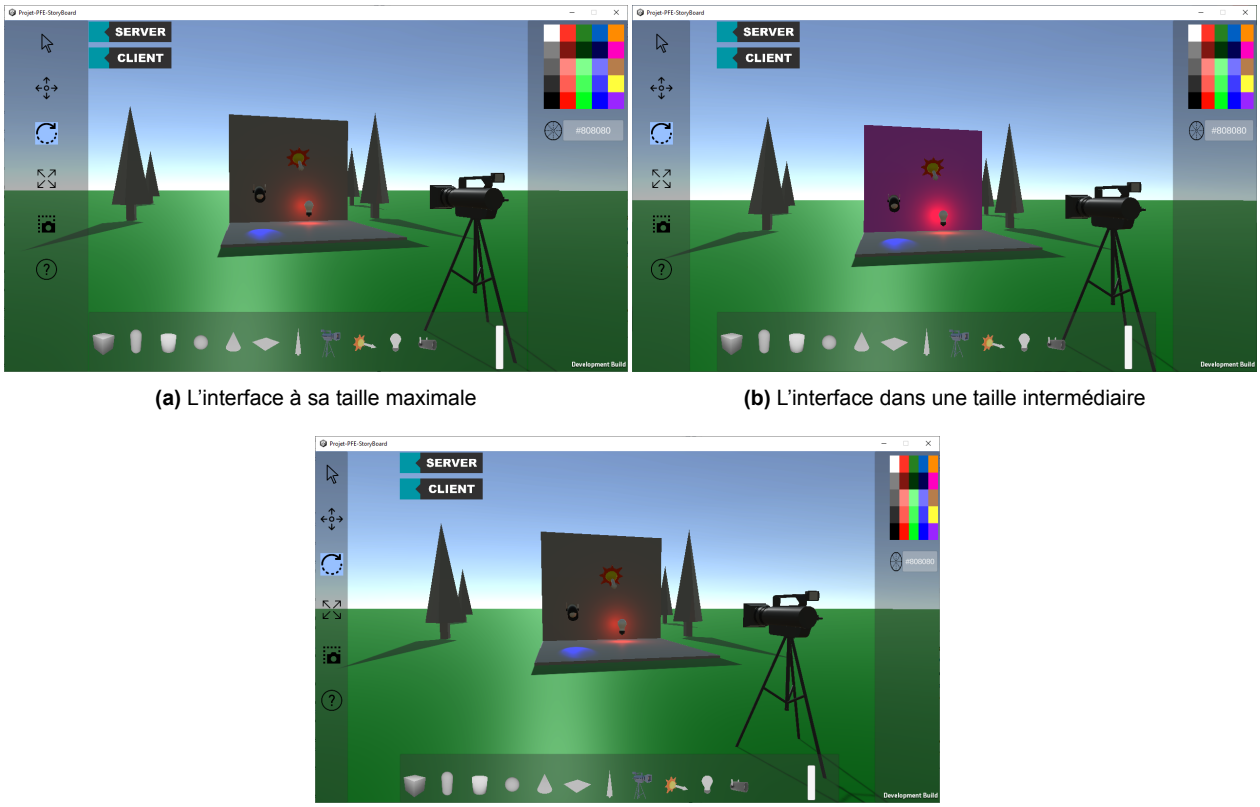
- Haut
- Bas
- Gauche
- Droite

En fonction de cette valeur, il est possible de déterminer quel côté du rectangle doit être manipulé. Nous avons positionné un objet invisible d'une certaine largeur sur le bord correspondant. Cet objet invisible

délimite la zone où le redimensionnement est possible lorsque l'utilisateur passe la souris. Pour détecter si l'utilisateur maintient le clic dans cette zone invisible, Unity fournit les interfaces ***IDragHandler*** et ***IEndDragHandler***. Ces deux interfaces sont conçues spécifiquement pour être utilisées par les éléments de l'interface graphique. Pour les intégrer à notre script, il nous suffit d'implémenter ces deux interfaces en définissant les méthodes ***OnDrag(PointerEventData eventData)*** et ***OnEndDrag(PointerEventData eventData)***. Ces deux méthodes sont respectivement appelées lorsque l'utilisateur maintient le clic sur l'objet et relâche le clic. De plus, les paramètres de ces méthodes, ***PointerEventData eventData***, contiennent toutes les informations sur le curseur, telles que sa position à l'écran, ou ce qui est particulièrement pertinent pour nous, la position du curseur à l'intérieur de l'objet cliqué. Avec tous ces éléments, il nous est possible de nous affranchir totalement de l'utilisation des coordonnées dans l'espace écran et ainsi régler les problèmes de la version précédente. Ainsi, à l'aide de ces deux méthodes, il nous est donc possible de déterminer trois éléments importants :

- Si l'utilisateur est train de maintenir le clic sur la zone
- Où l'utilisateur a sa souris
- Quand il relâche la souris

Ainsi, lorsque la méthode ***OnDrag(PointerEventData eventData)*** est appelée, le redimensionnement est déclenché. Le redimensionnement s'arrête lorsque ***OnEndDrag(PointerEventData eventData)*** est appelé. Ensuite, il nous suffit de calculer la taille du redimensionnement. Pour ce faire, nous avons simplement décidé de récupérer la position de la souris à l'intérieur de notre zone de sélection. Par la suite, nous calculons la distance entre la position de la souris et le bord de notre objet, puis nous redimensionnons l'élément de l'interface graphique en fonction de cette distance. Pour éviter que le redimensionnement ne rende les fenêtres trop grandes ou trop petites, nous avons ajouté deux paramètres supplémentaires. Ces paramètres, représentés par des Vector2, indiquent la taille maximale et minimale en hauteur et en largeur des fenêtres graphiques. Ces valeurs sont facilement accessibles et modifiables afin de s'adapter à toutes les fenêtres qui pourront être créées à l'avenir.



(a) L'interface à sa taille maximale

(b) L'interface dans une taille intermédiaire

(c) L'interface à sa taille minimale

Figure 5.16.: Différentes tailles de l'interface redimensionnable

5.6.3. Interface réactive

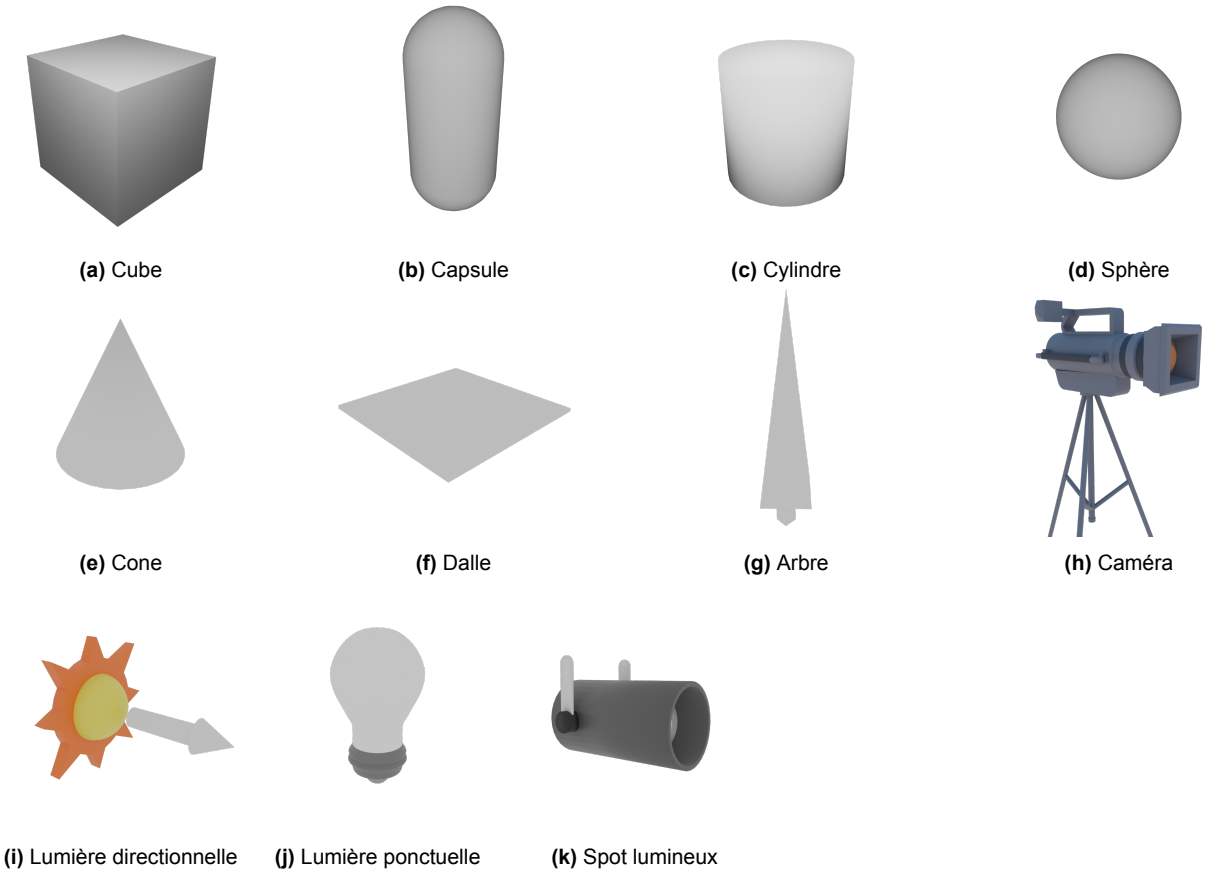
Un autre aspect crucial de l'interface graphique est sa capacité à s'adapter aux différentes résolutions d'écran. Une interface réactive est une interface qui s'ajuste automatiquement à la taille de l'application sur l'écran de l'utilisateur assurant ainsi une expérience cohérente indépendamment de la taille de l'écran. Pour atteindre cet objectif, nous avons utilisé les points d'ancrage de Unity comme expliqué précédemment. Cependant, en raison de la présence d'interfaces redimensionnables, cette tâche s'est révélée particulièrement complexe. Nous avons dû gérer plusieurs cas, chacun nécessitant une approche différente pour garantir la réactivité de l'interface.

Les éléments d'interface redimensionnables : Cette catégorie englobe les grands éléments de l'interface pouvant être redimensionnés. Les contraintes d'ancrage pour ces objets varient en fonction de leur nature. Pour les objets pouvant être réduits en hauteur, nous avons mis en place un ancrage personnalisé pour que seule la largeur de l'objet s'adapte à la résolution de l'application. En revanche, pour les objets dont la largeur ne peut pas être réduite, nous avons configuré l'ancrage de manière à ce que seule la hauteur de l'objet s'ajuste proportionnellement à la résolution de l'application. Le contenu des interfaces redimensionnables pouvant être légèrement déformé : Cette deuxième catégorie a été traitée de manière similaire à la première. En fonction de la direction dans laquelle le contenu peut être déformé, nous avons ajusté les points d'ancrage pour que seule la hauteur ou la largeur de l'élément graphique évolue proportionnellement avec la taille de l'application ou de l'interface. Par exemple, pour la palette de couleurs de notre application, nous avons maintenu la hauteur constante afin de garantir sa manipulation aisée tandis que seule la largeur

s'adaptait aux différentes tailles d'interface. Le contenu des interfaces redimensionnables devant conserver leur aspect d'origine : Cette dernière catégorie a nécessité un traitement au cas par cas car les objets de cette catégorie ont des formes et des emplacements variés. Heureusement, Unity offre la possibilité de créer des ancrages uniques pour chaque objet. Une solution courante pour préserver l'aspect de ces objets malgré les variations de taille de l'interface a été de lier les points d'ancrage à la taille de l'objet lui-même.

5.6.4. Interface d'apparition des objets

Une fois le code d'apparition mis en place conformément aux instructions fournies dans la section 5.2.1, il reste à lier cette fonctionnalité à l'interface graphique. Cette fonctionnalité est située dans la partie inférieure de l'interface graphique qui regroupe tous les objets susceptibles d'apparaître. Ces objets sont représentés par des boutons comportant une image les symbolisant. Actuellement, les objets disponibles dans notre version sont les suivants :



(a) Cube **(b)** Capsule **(c)** Cylindre **(d)** Sphère

(e) Cone **(f)** Dalle **(g)** Arbre **(h)** Caméra

(i) Lumière directionnelle **(j)** Lumière ponctuelle **(k)** Spot lumineux

Figure 5.17.: Icône de tous les objets disponibles

Étant donnée notre méthode d'apparition des objets sur le curseur, la seule solution pour appliquer ce choix est le glisser-déposer. Pour ce faire, nous nous sommes inspirés de notre travail sur les interfaces redimensionnables. En utilisant l'interface **IEndDragHandler**, il nous est possible de redéfinir la méthode ***OnEndDrag(PointerEventData eventData)***. Ainsi, en l'implémentant dans ***DragSpawnGO***, il est possible de détecter le lâcher du clic de l'utilisateur et de récupérer la position du curseur. Il suffit ensuite d'appeler la méthode ***CreateGameObjectOnCursor*** pour faire apparaître l'objet à l'emplacement de la souris.

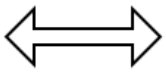
5.6.5. CursorManager

Un moyen simple de rendre l'interface plus claire et intuitive est l'utilisation de changements visuels. Un des changements visuels les plus simples est celui de la souris. Nous avons implémenté dans un premier temps un changement visuel de la souris dans deux cas différents :

1. Le redimensionnement de l'interface
2. Les boutons ou éléments interactifs.

Sans le changement de visuel, il est presque impossible pour un utilisateur de savoir que l'interface est redimensionnable. Pour indiquer cette option, nous avons créé deux curseurs différents correspondant aux deux types de redimensionnement :

(a) Curseur pour le redimensionnement en hauteur



(b) Curseur pour le redimensionnement en largeur

Figure 5.18.: Visuel des différents curseurs de redimensionnement

Maintenant que les visuels des curseurs sont prêts, il est nécessaire de les implémenter correctement dans l'application. Pour faciliter cette manipulation à travers toute l'application, nous avons créé un objet nommé ***CursorManager*** dans chaque scène accompagné d'un script du même nom. Ce script contient toutes les images potentielles de curseur ainsi que les méthodes nécessaires pour changer le curseur. Ainsi, tout élément souhaitant modifier le curseur n'a qu'à récupérer l'instance de ***CursorManager*** et appeler la méthode appropriée. Dans Unity, changer le curseur de l'application est une opération simple. La classe Cursor fournie par Unity offre un large éventail de méthodes pour manipuler le curseur. Parmi ces méthodes, la méthode ***SetCursor(Texture2D texture, Vector2 hotspot, CursorMode cursorMode)*** permet de modifier l'apparence et le comportement du curseur. Cette méthode possède trois paramètres :

La texture du curseur (Texture2D) : il s'agit de l'image utilisée comme curseur de la souris, généralement une texture 2D chargée dans le projet Unity. Si la valeur fournie est égale à***null***, alors le curseur redevient celui du système d'exploitation.

Le hotspot (Vector2) : il représente la position du curseur sur l'écran. Dans notre cas, afin de garantir un positionnement correct indépendamment de la taille du curseur, le hotspot est défini à largeur/2, hauteur/2 de l'image du curseur.

Le mode du curseur (CursorMode) : ce paramètre spécifie le comportement d'affichage du curseur. Nous avons choisi de laisser la valeur par défaut, CursorMode.Auto, qui force le curseur à être affiché selon les règles du système d'exploitation.

L'étape suivante a consisté à intégrer les fonctions de manipulation du curseur dans les éléments appropriés. Pour les éléments d'interface graphique, les méthodes de ***CursorManager*** seront invoquées dans

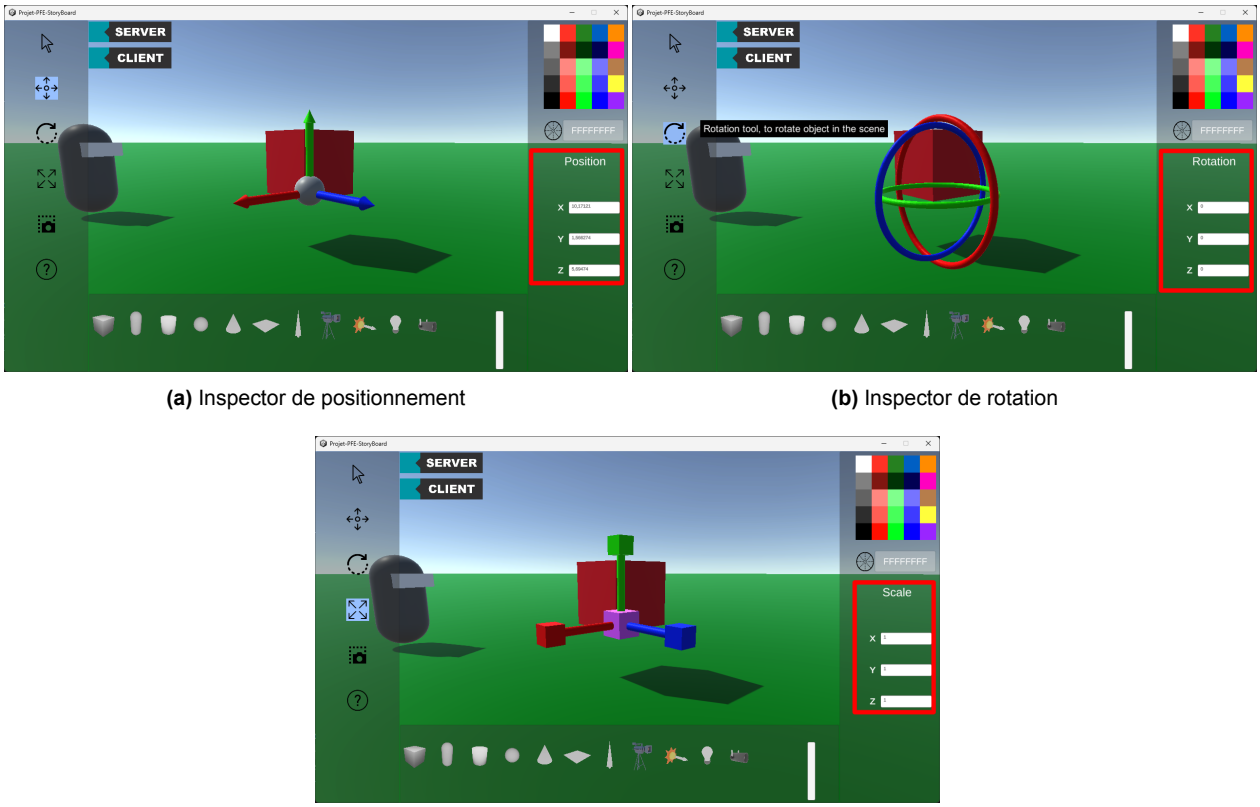
les méthodes ***OnPointerEnter*** et ***OnPointerExit***. Ces méthodes sont respectivement déclenchées lorsque le curseur entre et sort de l'élément graphique. Afin d'indiquer qu'un élément est interactif, nous souhaitons également modifier le curseur lorsque l'utilisateur survole cet élément. Cette fonctionnalité sera particulièrement utile pour la barre d'outils et pour signaler que les images des objets peuvent être manipulées pour les faire apparaître. À cet effet, nous utiliserons l'image de curseur suivante :

Figure 5.19.: Curseur d'interaction

Pour afficher ce curseur, nous ajoutons une nouvelle méthode dans ***CursorManager*** permettant de changer le curseur avec cette image. Cette méthode sera une fois de plus appelée par les méthodes ***OnPointerEnter*** et ***OnPointerExit*** qui sont implémentées pour chaque bouton de l'interface graphique d'apparition.

5.6.6. Modification par valeurs numériques

Lorsqu'un utilisateur souhaite modifier les propriétés d'un objet, nous avons déjà mentionné qu'il pouvait soit utiliser des gadgets de transformation, soit simplement déplacer l'objet en utilisant le curseur pour effectuer des déplacements libres. Cependant, l'interface utilisateur propose une autre manière d'effectuer ces transformations, à savoir une modification manuelle des valeurs numériques. Dans la partie droite de l'interface utilisateur, une zone appelée l'***Inspector*** s'adapte en fonction de l'outil sélectionné. En effet, lorsque l'utilisateur sélectionne un objet puis active l'outil de positionnement, de rotation ou de redimensionnement, les propriétés de l'objet sont représentées dans des champs de saisie de type **Inputfield**. Il peut ensuite modifier directement les valeurs indiquées qui seront automatiquement appliquées en appuyant sur la touche entrée.



(a) Inspector de positionnement

(b) Inspector de rotation

(c) Inspector de redimensionnement

Figure 5.20.: Les différents visuels de l'Inspector

Pour parvenir à ce résultat, lors de la sélection de l'objet, le script ***ObjectManipulation*** fait appel à la fonction ***GetInformationObject***, liée à la classe ***Selection***, permettant de récupérer toutes les informations concernant notre objet. Ensuite, notre script extrait uniquement les données correspondant à l'outil activé sous la forme d'un vecteur x, y, z et les transmet aux **Inputfields** associées.

Lors de chaque modification de l'utilisateur, les valeurs des champs sont récupérées et transformées en un vecteur et fournies à notre objet lors de l'appui sur entrée.

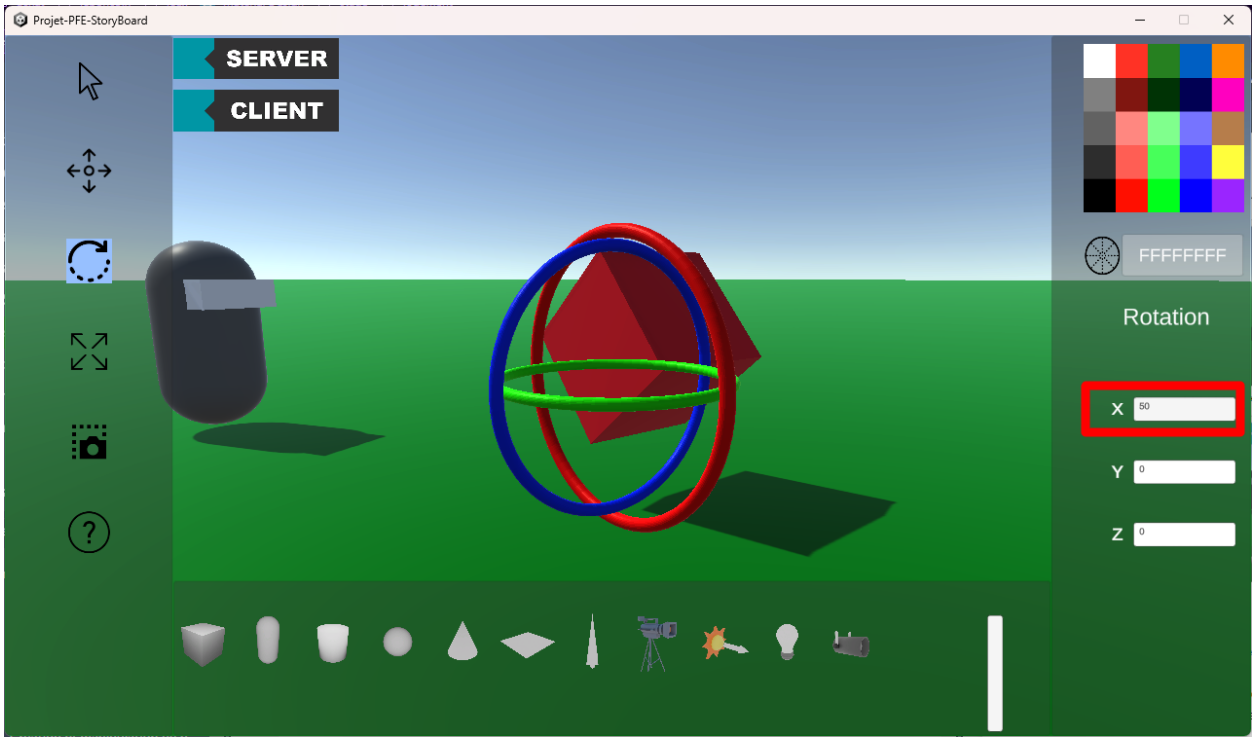


Figure 5.21.: Rotation de 50 degrés autour l'axe x

Lors du développement, plusieurs contraintes devaient être prises en compte, telles que la possibilité de renseigner des positions et des rotations négatives, tout en bloquant la possibilité de renseigner un redimensionnement négatif. Pour signaler à l'utilisateur que les valeurs saisies étaient invalides, nous avons programmé le texte pour qu'il devienne rouge lorsque la valeur entrée ne respecte pas les critères définis. Un autre détail auquel nous nous sommes heurtés est l'unité d'expression de la rotation. En effet, un être humain exprime la rotation d'un objet en degrés, alors que l'objet lui-même stocke ses propriétés de rotation sous forme de quaternions. Par conséquent, notre script doit effectuer une conversion avant de restituer les valeurs de rotation à l'interface graphique, et inversement lors des modifications effectuées par l'utilisateur.

5.6.7. Éditions des propriétés uniques des objets

De nombreux objets, tels que les lumières, présentent des propriétés uniques ce qui nécessite un outil permettant à l'utilisateur d'accéder et de modifier ces champs spécifiques. Nous avons intégré cette fonctionnalité à l'outil curseur en modifiant une fois de plus la partie droite de l'interface utilisateur pour y ajouter une section "Cursor Tool" regroupant toutes les valeurs uniques des objets. Cette section est équipée d'un ***Scroll Rect*** transformant le menu en un menu déroulant similaire à celui de l'interface d'apparition des objets. Ainsi, si le nombre de propriétés uniques d'un objet devait être trop important pour être affiché dans l'interface, il serait toujours possible d'y accéder en faisant défiler l'interface graphique à l'aide de la souris. Le fonctionnement de cette interface graphique reste assez basique : pour chaque propriété unique, un élément graphique permettant sa modification est créé. Dans l'application, cela se traduit par un élément graphique comportant le nom de la propriété et un ***InputField*** de Unity. Lors de la sélection de l'objet, ses

champs uniques sont affichés en fonction de son type tandis que les autres champs sont masqués à l'aide d'un script dédié aux propriétés uniques. Actuellement, nous ne gérons que deux types d'objets ayant des propriétés uniques. Le premier type est commun à tous les objets de notre scène : il s'agit du nom des objets. Cette propriété est gérée par le script ***NameToolManager***. Ce script est responsable de l'élément graphique associé au champ unique du nom ainsi que de l'***InputField*** lié à ce champ. Il dispose de trois méthodes :

1. ***SetCurrentGameObject(GameObject gameObject)*** : Cette méthode permet d'assigner à l'objet script l'objet actuellement sélectionné dans la scène. Pour que l'objet soit utilisable par le script, il vérifie que celui-ci possède la classe ***NamePrefabManager***, la classe responsable de la gestion des noms. Si ce n'est pas le cas, l'objet est rejeté.

2. ***UnSetCurrentGameObject*** : Cette méthode permet au script de libérer l'objet fourni par la méthode ***SetCurrentGameObject***.

3. ***OnValueEndName*** : Cette méthode est appelée par l'***InputField*** lorsque qu'une nouvelle valeur est entrée. Le texte entré est alors appliqué à l'objet à l'aide de la classe ***NamePrefabManager*** et de sa méthode ***SetNameServeurSide***.

Le script ***NameToolManager*** est également chargé d'afficher le champ unique si l'objet sélectionné possède un nom. Pour cela, à chaque frame, le code vérifie si un objet est sélectionné. S'il est sélectionné, il vérifie ensuite par sécurité que l'objet appartient bien à la classe ***NamePrefabManager*** et possède un nom. Si tel est le cas, le champ de modification apparaît sur l'interface ; sinon, il reste caché.

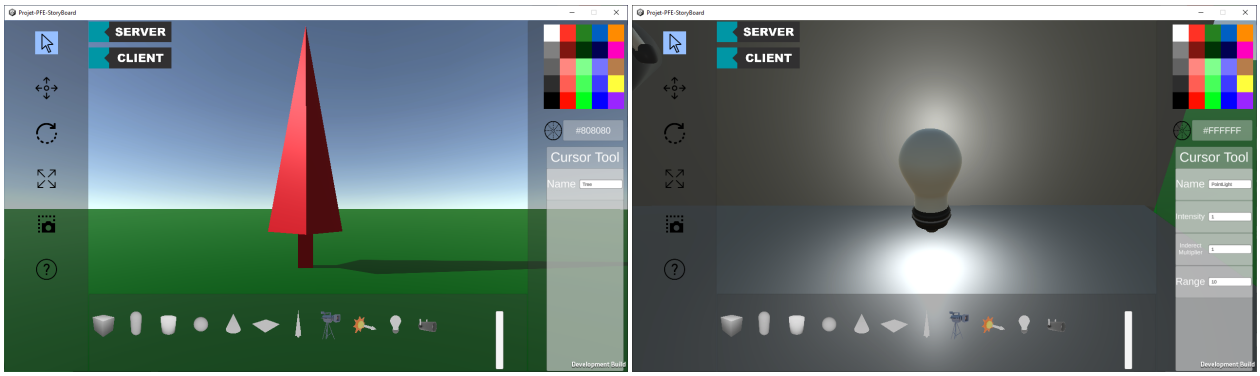
Les lumières constituent un autre type d'objets comportant des champs uniques. Le script dédié à la gestion de ces champs spécifiques, ***LightToolManager***, fonctionne de manière très similaire à ***NamePrefabManager***. ***LightToolManager*** dispose également de méthodes pour sélectionner/désélectionner l'objet en cours et pour appliquer les valeurs entrées par l'utilisateur dans les ***InputField*** correspondants. La seule distinction entre ***LightToolManager*** et ***NamePrefabManager*** réside dans la façon dont les champs sont affichés. Alors que ***NamePrefabManager*** gère automatiquement l'affichage de ses champs uniques, ce n'est pas le cas pour ***LightToolManager***. Ce dernier propose une méthode d'affichage, ***ShowCorresponding-InputField***, qui affiche les champs en fonction du type de lumière. Cette méthode doit être appelée lors de la sélection de l'objet.

Nous avons adopté ces deux modes de fonctionnement pour deux raisons :

1. Facilité d'utilisation : ***NamePrefabManager*** est plus simple à utiliser car son affichage est autonome et ne nécessite pas d'appel externe. Cependant, les vérifications de son état sont effectuées à chaque frame, soit environ toutes les 0,016 à 0,018 secondes. Bien que ces vérifications soient simples, elles pourraient avoir un impact sur les performances de notre application à long terme.

2. Performance : En revanche, ***LightToolManager*** est moins simple à utiliser car l'affichage des champs uniques est géré par une méthode et non de manière automatique. Cependant, cette complexité d'utilisation évite un appel de vérification de l'état à chaque frame.

Malheureusement, nous n'avons pas eu le temps de tester l'impact réel de ces deux approches sur les performances de notre application.

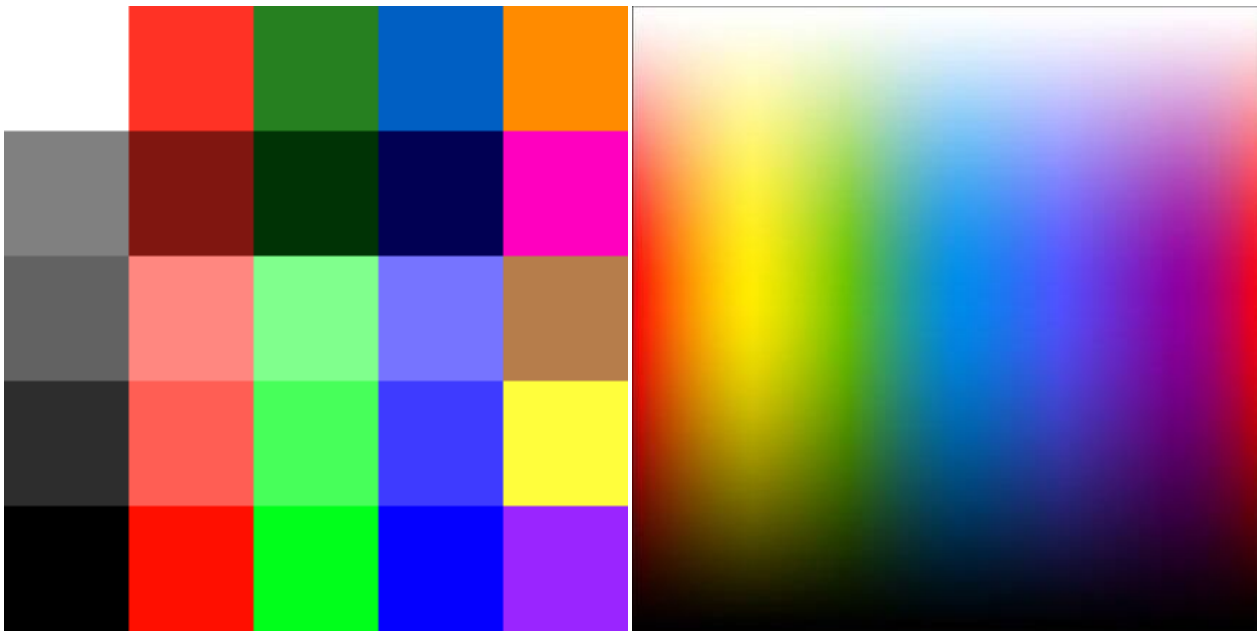


(a) Propriétés uniques des noms **(b)** Propriétés uniques d'une lumière

Figure 5.22.: Visualisation des différentes propriétés uniques

5.6.8. Changement des palettes de couleurs

Pour permettre une expérience plus personnalisable, nous souhaitons offrir à l'utilisateur la possibilité de changer de palette de couleurs. Le changement de palette de couleurs est assuré par la classe ***SwitchPalet***. Cette classe est composée de deux méthodes : ***SetColorSelector***, une méthode privée et ***OnClickSwitchPalet***, une méthode publique. La méthode ***SetColorSelector*** permet de parcourir toutes les palettes de couleurs disponibles dans la classe ***SwitchPalet***. La liste des palettes disponibles est représentée par une liste d'images (Sprite) qui peut être directement modifiée depuis Unity. Dans la version actuelle du code, seulement deux palettes sont supportées : une palette aux couleurs limitées et le spectre des couleurs. La méthode ***OnClickSwitchPalet*** appelle simplement la méthode ***SetColorSelector***. ***OnClickSwitchPalet*** est appelée lors de l'appui sur le bouton de couleur.



(a) Palette de couleur restreinte **(b)** Spectre des couleurs

Figure 5.23.: Les différentes palettes de couleur

5.6.9. Page d'aide

La page d'aide est un canvas composé d'explications pour l'utilisateur. En effet, cette page permet à l'utilisateur d'assimiler les différents contrôles de l'application. Cette aide s'affiche automatiquement lors de la connexion au serveur ou lors du clic sur le bouton associé. Pour fermer la page, l'utilisateur peut soit cliquer sur la croix, soit cliquer à nouveau sur le bouton aide de la barre d'outils.

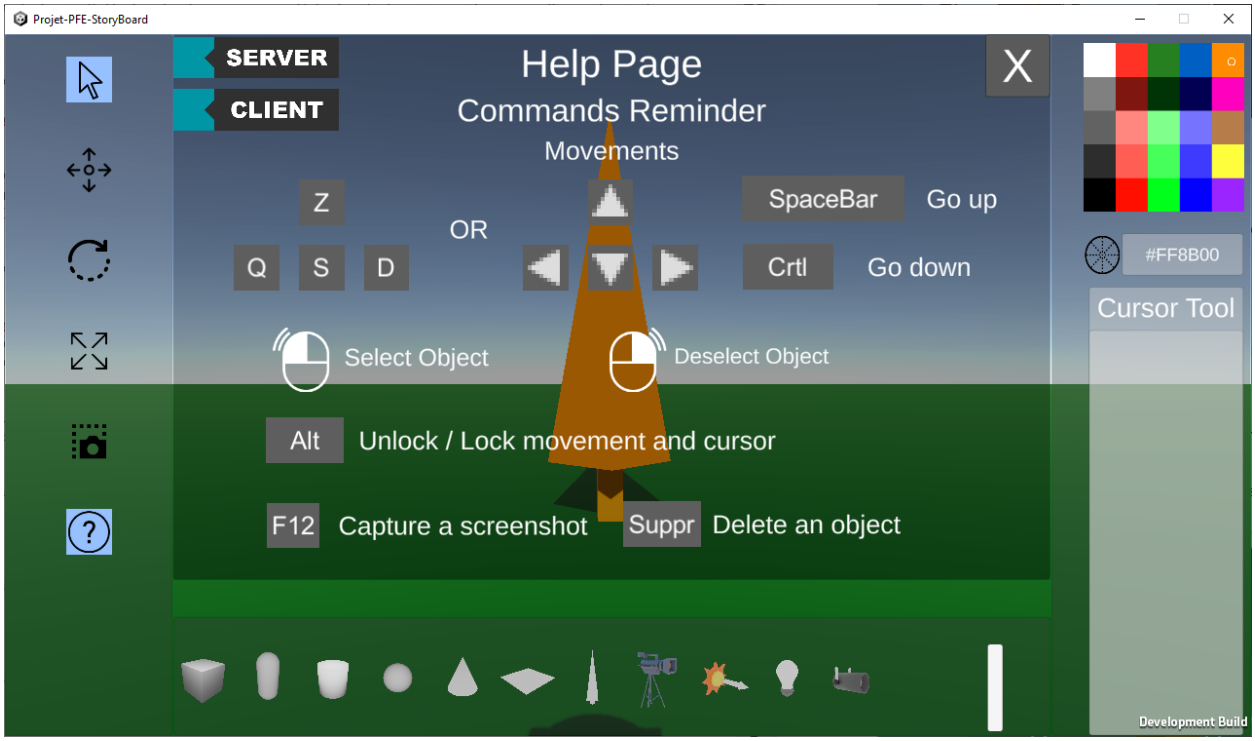


Figure 5.24.: Page d'aide

5.6.10. Tooltip

Pour créer le tooltip, nous avons besoin de deux éléments :

- d'un arrière-plan représentant la boîte du tooltip
- d'un texte explicatif de l'élément survolé

Ainsi, notre tooltip est composé d'un panel noir qui fera office d'arrière-plan et d'une zone de texte.

De plus, cette aide doit répondre à plusieurs critères :

- n'apparaître que lors du survol des éléments souhaités
- adapter sa taille et son texte en fonction des informations à transmettre

Pour gérer l'apparition du tooltip, la classe ***Tooltip***, responsable de la gestion des tooltips, propose en public la méthode ***ShowTooltip(string tooltipString)***. Cette méthode prend en paramètre le texte devant

être affiché par le tooltip. Elle est aussi responsable de l'ajustement de la taille de la boîte représentant le tooltip. Ainsi, l'arrière-plan ajuste sa taille automatiquement en fonction de la longueur du texte tout en ajoutant un padding pour une meilleure lisibilité de l'aide. En ce qui concerne sa position, le tooltip se place à la position de la souris avec un léger décalage sur la coordonnée en x afin d'obtenir un espace entre la souris et le tooltip. Comme pour les curseurs, la methode ***ShowTooltip(string tooltipString)*** sera appelée dans les méthodes ***OnPointerEnter*** et ***OnPointerExit*** des classes nécessitant un tooltip.

Néanmoins, cette version des tooltips reste assez basique. Bien que la taille de la boîte s'adapte au texte, nous n'avons pas mis en place de vérification vis-à-vis de la position de la boîte. En conséquence, il est possible que des textes longs dépassent l'écran de l'utilisateur. La solution la plus simple serait de vérifier si la boîte est partiellement en dehors de l'écran grâce à l'origine de la boîte et à sa taille. Si l'on remarque que la boîte est en dehors de l'écran, deux solutions seraient alors possibles :

- Soit on essaie de redimensionner la boîte de manière à ce qu'elle soit dans l'écran
- Soit on change la boîte de côté. Ainsi, par exemple, si la boîte dépasse sur la partie droite de l'écran, le tooltip se déplacera sur le côté gauche de la souris

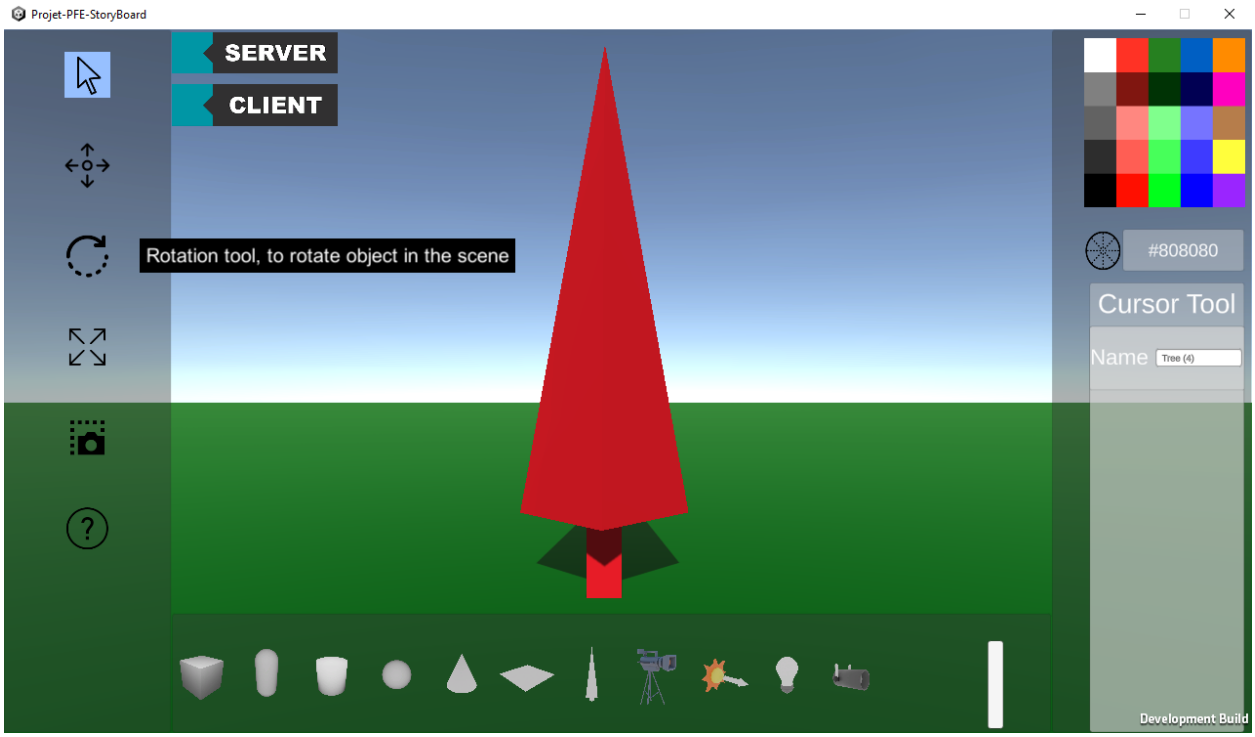


Figure 5.25.: Exemple du tooltip sur l'outil rotation

6 TESTS ET RÉSULTATS

6.1. Fiches des participants

Pour déterminer l'efficacité de notre application, nous avons mis en place un protocole d'expérimentation (Cf Annexe A.1). Ce protocole se divise en plusieurs étapes. La première étape consiste à recueillir des informations sur les habitudes personnelles des utilisateurs (Cf Annexe A.2) telles que leur temps d'utilisation des écrans, s'ils jouent à des jeux vidéo, s'ils ont souvent recours à des logiciels de traitement de texte, logiciels graphiques ou de 3D. Cette première partie nous renseignera sur leur propension à une prise en main facile de l'outil.

Dans un second temps, les utilisateurs auront à accomplir une série de douze objectifs pré-définis (Cf Annexe A.3) qui sont:

1. Se déplacer dans la scène
2. Poser un objet dans la scène
3. Déplacer un objet librement dans la scène
4. Effectuer des modifications sur les propriétés d'un objet à l'aide de gizmo (mouvement, rotation et redimension d'objet)
5. Manipuler les champs uniques de ces objets dans l'inspector (à l'aide de l'interface utilisateur)
6. Changer la couleur d'un objet
7. Manipuler le champ unique de chaque type de lumière
8. Visualiser la scène depuis le point de vue d'une caméra
9. Ajouter un filtre à une caméra
10. Faire apparaître le point de fuite d'une caméra
11. Prendre un screenshot depuis le point de vue d'une caméra
12. Mettre en place une scène complète de manière cohérente

Pour se faire, les utilisateurs seront mis devant l'application et devront utiliser les informations fournies par le logiciel. Les 11 premières étapes sont là pour inviter les utilisateurs à utiliser plusieurs fonctionnalités du

projet. La dernière étape, nous sert à déterminer si l'utilisateur a bien compris l'ensemble des objectifs et à déceler le moindre bug apparent. Durant chaque étape, chaque objectif sera chronométré pour déterminer avec quelle aisance l'utilisateur a accompli une tâche et si le temps passé sur cette tâche est équivalent aux autres étapes.

Enfin, l'utilisateur sera invité à procéder à son auto-évaluation après avoir accompli l'ensemble des objectifs définis (Cf Annexe A.4). Il devra évaluer chaque tâche en utilisant une échelle de notation composée de cinq niveaux qui sont les suivants :

1. Echec de la tâche
2. A réussi avec de l'assistance
3. A réussi avec difficulté
4. A réussi avec un temps d'adaptation
5. Réussite immédiate de la tâche

L'utilisateur sera ainsi invité à évaluer la difficulté de chaque tâche sur une échelle de cinq. Ensuite, s'il le désire, il pourra ajouter un commentaire pour expliquer son ressenti ou donner son avis sur la tâche. Enfin, s'il le souhaite, il pourra rédiger un avis global sur le logiciel afin de nous fournir des conseils et des critiques pour son amélioration (Cf Annexe A.5).

6.2. Information sur l'ensemble des testeurs

Nous avons expérimenté notre logiciel sur une population de 7 personnes. La moyenne des âges de nos utilisateurs est de 40 ans avec un écart-type de 17.5. Cette population est composé de 3 femmes et 4 hommes. Nos utilisateurs ont tous une grande consommation d'écran que cela soit pour une utilisation professionnelle ou pour le plaisir. A une exception près, la totalité des testeurs jouent à des jeux vidéos. Cependant, il est important de noter que la familiarité avec les logiciels de traitement de texte, de graphisme et de modélisation 3D varie considérablement d'un individu à l'autre au sein de notre population de test. Il convient de souligner que ce test a été conçu pour dégager une tendance plutôt que pour vérifier pleinement le fonctionnement de notre application.

6.3. Notes sur le déroulement des tests

Nous allons maintenant décrire de manière générale ce que nous avons remarqué ainsi que les observations des utilisateurs sur ces différentes tâches :

- Tâche 1: Pour cette tâche, lors de notre premier test, nous avons remarqué qu'il est très difficile pour l'utilisateur de sortir correctement de la fenêtre d'aide. Suite à cette évaluation, une croix a été ajoutée pour faciliter la fermeture de la page d'aide. Néanmoins, malgré cette modification, certains testeurs ont encore rencontré des difficultés pour quitter la page. Lors de nos premiers essais, l'indication de l'utilisation de la touche ***Alt*** sur la page d'aide n'était pas suffisamment claire. Nous avons donc changé

le texte associé à la touche afin que son fonctionnement soit compréhensible. Les utilisateurs nous ont aussi informé que la touche n'était pas intuitive.

- Tâche 2: En général, tous les utilisateurs ont trouvé cette tâche intuitive. Cependant, il nous a été conseillé d'ajouter des explications dans l'aide afin que la tâche soit plus facilement maîtrisée par tous les utilisateurs.

- Tâche 3: Lors des différents tests, le problème de la désélection revenait souvent. En effet, il n'est pas intuitif pour les utilisateurs de désélectionner à l'aide du clique droit. Malgré ça, les utilisateurs finissaient par s'habituer à la fonctionnalité et parvenaient à la maîtriser.

- Tâche 4: Les utilisateurs nous ont informé que les outils de modification étaient faciles d'utilisation, mais ils ont rencontrés différents problèmes. Tout d'abord, ils ont rencontré des difficultés de compréhension. En effet, l'utilisation du mode ”gizmo” dans notre documentation de test était source de confusion car ce terme spécifique n'est pas universellement connu. De plus, les utilisateurs ont souligné la complexité d'utilisation du gadget de rotation. Ils ont exprimé le désir de pouvoir ”tourner” les cercles de l'outil de rotation sur eux-mêmes, alors qu'il n'était possible d'utiliser cet outil qu'en effectuant des mouvements horizontaux avec la souris. Cette limitation rendait la rotation des cercles positionnés verticalement peu intuitive.

- Tâche 5: Le problème principal de cette tâche ne réside pas dans sa difficulté mais dans un manque de compréhension de l'objectif donné. En effet, notre fiche de test pouvait ne pas être claire sur certains points. Nous avons manqué de temps pour perfectionner la fiche afin qu'elle soit le plus compréhensible possible. Le terme ”champ unique” n'était pas un terme compréhensible pour la plupart des testeurs. Malgré ça, lorsque les utilisateurs finissaient par comprendre la question, la tâche était correctement accomplie.

- Tâche 6: Lors de la première réalisation de cette tâche, nous avons découvert un bug. Il survenait lorsque l'utilisateur remettait sa souris sur l'objet sélectionné après avoir choisi une nouvelle couleur. Nous avons pu résoudre ce problème après avoir pris en compte ce retour. Certains utilisateurs ont noté qu'il n'était pas pratique de percevoir le changement de couleur en raison de la présence de la couleur utilisée pour indiquer visuellement la sélection de l'objet. En effet, la couleur de la sélection prend le pas sur le choix de couleur de l'objet, ce qui rend difficile la visualisation de la couleur actuellement appliquée.

- Tâche 7: Dans l'ensemble, les utilisateurs ont réussi à atteindre cet objectif avec succès. Cependant, quelques exceptions ont rencontré des difficultés en raison d'un manque de compréhension de la notion de lumière dans l'ensemble des ressources proposées.

- Tâche 8: Pour cette tâche, les avis sont assez tranchés. Certains vont trouver la tâche intuitive, l'icône étant considérée comme suffisamment explicite. D'autres considèrent que la gestion des caméras est compliquée, ce qui rend la réalisation de la tâche complexe.

- Tâche 9: Dans l'ensemble, la tâche a été faite correctement. Une grande partie des utilisateurs ont considéré cette tâche comme étant facile de prise en main et compréhensible. Un des testeurs a noté que l'emplacement de l'onglet des filtres n'était pas idéal et que son utilisation lui semblait peu intuitive.
- Tâche 10: Dans la majorité des cas, cette tâche a été réalisée correctement sans rencontrer de problème.
- Tâche 11: Certains des utilisateurs ont eu du mal à effectuer cette tâche. En effet, certains utilisateurs ont oublié la plupart des touches présentées dans la page d'aide au début du test. Après avoir eu le réflexe de revenir sur la page d'aide, la tâche a été simplement réalisée. Nous avons recueilli des avis divergents concernant la touche à utiliser pour effectuer une capture d'écran. Certains considèrent que ***F12*** est une touche convenable, tandis que d'autres pensent qu'il serait plus logique que la touche ***Impécr*** s'occupe de la capture d'écran. De plus, il nous a été conseillé d'ajouter une icône pour indiquer qu'une capture d'écran a été réalisée avec succès.
- Tâche 12: Après avoir pris en main l'ensemble des commandes à l'aide des précédentes tâches, les utilisateurs ont pu facilement réaliser cette tâche et ont été amenés à explorer le logiciel librement.

6.4. Résultats et Conclusions

Pour commencer, examinons la distribution des notes attribuées par l'ensemble des utilisateurs.

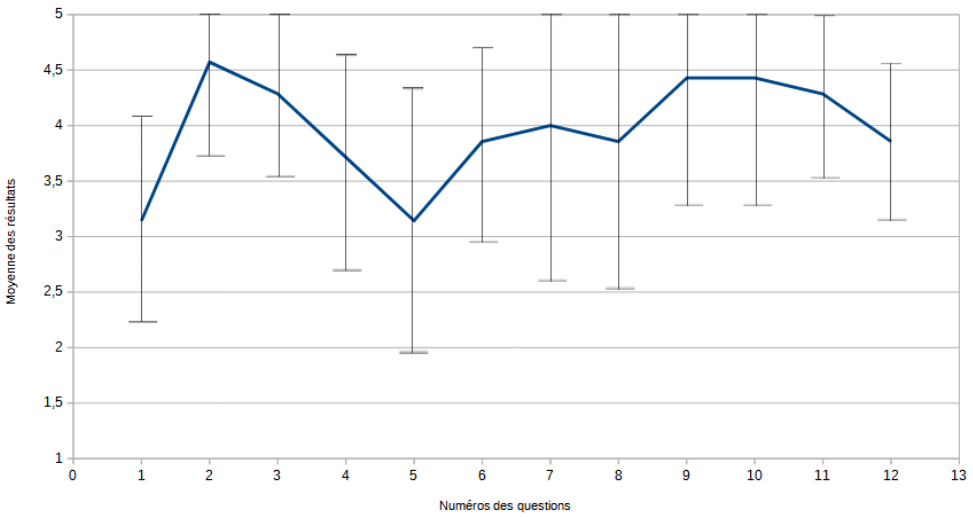


Figure 6.1.: Courbe de la moyenne de notation de chaque tâche

La Figure 6.1 montre la moyenne des notes attribuées par l'ensemble des utilisateurs à chaque question, accompagnée de leur écart-type respectif. Nous pouvons voir que toutes les moyennes des notes se situe dans l'intervalle [3.14,4.57]. Nous pouvons ainsi conclure que malgré quelques incompréhensions de la part de certains utilisateurs, les tâches ont globalement été comprises. Il faut noter que les scores les plus bas

sont attribués aux questions 1 et 5. Cela peut s'expliquer par le fait que la première question est la première tâche à laquelle les utilisateurs sont confrontés. Ils doivent donc s'habituer à l'application ce qui peut présenter certaines difficultés initiales. Les résultats relativement bas de la cinquième question s'expliquent par le fait que de nombreux utilisateurs rencontraient des difficultés à comprendre la tâche. En effet, l'expression "champ unique" a suscité beaucoup d'incertitude chez les utilisateurs. En revanche, la deuxième tâche a obtenu le meilleur résultat indiquant que cette tâche a bien été assimilée par l'ensemble des testeurs. De plus, il est à noter que les étapes finales ont été globalement bien exécutées ce qui témoigne d'une appropriation adéquate de l'application.

Lorsque l'on calcule la moyenne générale pour l'ensemble des utilisateurs, on obtient une moyenne de 3,96 avec un écart-type de 0,49. Cette moyenne est encourageante car elle dépasse la moyenne théorique possible. De plus, l'écart-type relativement faible indique que la plupart des résultats convergent autour de cette moyenne ce qui témoigne d'une bonne cohérence dans les évaluations.

Nous allons maintenant aborder les durées consacrées par les testeurs à chaque tâche individuelle.

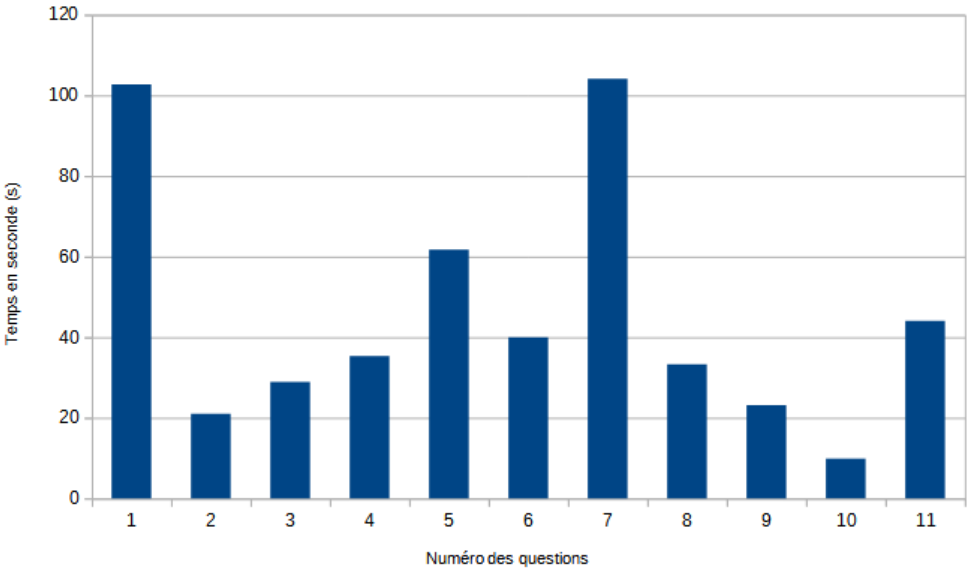


Figure 6.2.: Histogramme de la moyenne des temps passés sur une tâche

La Figure 6.2 montre que les durées consacrées aux différentes tâches varient considérablement. En effet, certaines tâches ont demandé beaucoup plus de temps que d'autres. Il est également notable que la première étape a été l'une des plus longues. En effet, on obtient une moyenne de 1 minute et 42 secondes par utilisateur sur cette étape. Cette observation montre bien que la première tâche a été une étape de prise en main du logiciel, ce qui a entraîné des difficultés lors de cette première tâche. On peut aussi voir que l'étape 7 a été également une étape qui a demandé beaucoup de temps. Cela montre la difficulté rencontrée par certains testeurs pour comprendre la notion de lumière dans le logiciel, en plus de celles liées à la compréhension de l'objectif de la tâche. L'étape 5 s'est avérée être le troisième objectif nécessitant le plus de

temps. Cette étape a suscité de nombreuses interrogations chez les utilisateurs en raison du vocabulaire spécialisé utilisé pour décrire la tâche. Nous pouvons aussi constater que la dixième étape a été celle qui a demandé le moins de temps à l'ensemble des testeurs. Cette observation est en corrélation avec le score élevé obtenu pour cette question. Cela indique que la méthode utilisée pour faire apparaître les points de fuite dans l'application est intuitive.

A la suite de toutes les expériences, les utilisateurs nous ont fait part de plusieurs remarques supplémentaires. Dans un premier temps, les utilisateurs ont souligné que le bouton Aide ne devrait pas être situé sur la même colonne que le reste des boutons d'outils à gauche de l'interface utilisateur. De plus, certains testeurs ont noté un manque flagrant de retour utilisateur sur certaines fonctionnalités telles que la capture d'écran à l'aide de la touche ***F12*** et encore du changement de mode avec ***ALT***. Un autre aspect relevé par de nombreux testeurs est l'absence de raccourcis clavier courants tels que ***CTRL-A***, ***CTRL-Z***, ou ***CTRL-C*** ***CTRL-V***. Bien que nous ayons envisagé d'intégrer ces raccourcis, leur utilisation dans un contexte collaboratif pourrait poser plusieurs problèmes majeurs susceptibles d'avoir un impact significatif sur notre projet à long terme. On nous a également signalé que les tâches pouvaient être difficiles à comprendre pour des personnes qui ne sont pas familières avec le domaine des sciences ou des mathématiques. Il aurait donc été nécessaire de revoir la fiche de test auparavant. De plus, les utilisateurs ont suggéré de mettre en place de manière visible un référentiel sur la carte pour faciliter la navigation et savoir où se déplacer. En effet, lorsqu'on effectue des changements de valeurs numériques, il est difficile de prévoir dans quelle direction l'objet va se déplacer car nous n'avons pas de référentiel pour nous diriger. Bien que l'ajout d'objet soit assez intuitif, le manque de prévisualisation de l'emplacement de l'objet avant son ajout a désorienté plusieurs testeurs. On nous a également suggéré d'intégrer un bouton permettant de nettoyer entièrement la scène c'est-à-dire de supprimer tous les objets qui y sont placés. De plus, il a été souligné que la manière de tourner un objet avec le gadget de rotation n'est pas intuitive. Sur un plan artistique, certains utilisateurs ont fait remarquer qu'il était surprenant pour un logiciel de storyboard de ne pas permettre l'ajout de zones de texte à la manière d'une bande dessinée. Cependant, tous les utilisateurs s'accordent à dire qu'après une période d'adaptation pour prendre en main le logiciel, il devient intuitif et facile à manipuler.

De notre côté, nous avons pu observer plusieurs aspects. Dans un premier temps, nous avons remarqué que certains testeurs n'était pas à l'aise avec l'anglais. Comme toutes les descriptions des outils disponibles sont en anglais et affichées lorsque les utilisateurs survolent ces outils avec leur souris, il leur était difficile de comprendre certaines fonctionnalités de notre application. Nous avons également constaté que certains utilisateurs ne portaient pas suffisamment attention à la page d'aide du logiciel, voire ne la consultaient pas du tout. Cela leur rendait probablement certaines tâches plus complexes qu'elles ne le sont en réalité. Nous avons également observé qu'un utilisateur peut penser avoir fait disparaître un objet lorsqu'il le déplace, en fonction du point de vue de sa caméra, alors qu'en réalité, il l'a simplement positionné en dessous de la plate-forme servant de sol dans la scène. De plus, nous avons constaté que les utilisateurs essaient souvent de placer des objets dans les airs lorsqu'ils les positionnent pour la première fois, particulièrement avec les différentes lumières. Cependant, dans l'état actuel, notre code ne supporte pas cette fonction.

En conclusion, les résultats sont concluants malgré quelques difficultés liées à une mauvaise compréhension ou à des problèmes de prise en main. Les différents conseils des utilisateurs seront précieux pour

améliorer le logiciel afin qu'il soit plus compréhensible par tous. Ces résultats laissent entrevoir un avenir prometteur pour l'application.

7 CONCLUSION ET PERSPECTIVE

Nous vous avons présenté dans ce document notre application servant d'outil pédagogique de storyboard collaboratif. Cette application, à destination première des étudiants dans le domaine de l'art, propose un espace créatif permettant à quiconque d'expérimenter sur des concepts artistiques au travers d'une scène en trois dimensions. Nous avons pour cela restreint au minimum les actions des utilisateurs en les laissant se déplacer librement là où ils le souhaitent sur la scène et en leur permettant de manipuler des objets à leur guise.

Les seules limites sont celles imposées par l'aspect collaboratif de notre application. En effet, jusqu'à trois utilisateurs peuvent se connecter sur le même serveur pour modifier une scène. Ces utilisateurs se voient mutuellement sur la scène et peuvent tous manipuler des objets mais pas simultanément. Tant qu'un objet est sélectionné par un utilisateur, les autres utilisateurs sont visuellement alertés de ce choix et ne peuvent pas sélectionner ce même objet tant qu'il ne l'a pas désélectionné. Pour renforcer cet aspect collaboratif, nous avons également mis en place un prototype de log qui permet à chaque utilisateur de connaître les actions importantes qui ont été effectuées sur le serveur ainsi que l'identifiant de la personne qui les a réalisées.

L'aspect éducatif de notre application provient tout d'abord de la possibilité pour chaque utilisateur de visualiser le point de fuite des caméras. Ce point de fuite qui peut être observé depuis n'importe quel angle de vue grâce aux mouvements libres des utilisateurs leur favorise la compréhension de la perspective. L'ajout des lumières sur la scène permet aux utilisateurs de créer l'éclairage qui leur convient. L'évolution en temps réel des ombres avec la modification des lumières facilite la compréhension de leur fonctionnement par les utilisateurs. La possibilité d'ajouter un filtre à une caméra peut permettre de visualiser et de favoriser une meilleure compréhension de nombreux aspects. Pour la version de l'application rendue avec le rapport, un filtre noir et blanc est proposé à l'utilisateur, ce qui lui permet de donner à la scène une apparence de storyboard traditionnel. Cela pourrait lui permettre d'apprendre les limitations et les avantages d'un storyboard papier traditionnel sans même avoir à en réaliser un.

Étant conçue de manière à permettre une intégration fluide des éléments supplémentaires, cette application ouvre la voie à des extensions futures, sans contrainte majeure, laissant ainsi place à de nombreuses possibilités d'amélioration. À titre d'exemple, de nouveaux éléments pourraient être ajoutés ce qui offrirait aux utilisateurs la possibilité de les manipuler. De plus, des filtres supplémentaires pourraient être développés pour les caméras, permettant par exemple d'accentuer les contours des objets ou de faciliter l'observation de leurs ombres. Étant développée sur Unity 3D, il est envisageable qu'à l'avenir, cette application puisse être adaptée à la réalité virtuelle ouvrant ainsi la possibilité d'atteindre un public plus vaste.

Les retours des utilisateurs fournissent des perspectives d'amélioration considérables pour notre projet. Le manque de retour utilisateur peut être résolu en intégrant une icône ou une zone dédiée à l'écran, indiquant l'état actuel de l'utilisateur. Pour remédier aux problèmes rencontrés lors de la capture d'écran, l'utilisation de l'audio ou d'effets visuels sur l'écran pourrait potentiellement résoudre le problème. Les difficultés de contrôle observées lors des phases de test suggèrent qu'il est nécessaire de revoir partiellement les contrôles et les touches utilisés par notre application. La fonction de sélection a également fait l'objet de nombreuses critiques, principalement attribuées au fonctionnement actuel de celui-ci. En mettant en œuvre notre concept initial de sélection, les problèmes de clarté associés à cette fonction devraient disparaître. Certains gadgets, notamment celui de rotation, présentent des problèmes majeurs de clarté et de fonctionnement. Pour les résoudre, il serait nécessaire de rendre le fonctionnement du gadget plus intuitif et de permettre à la position de la souris de déterminer l'angle de rotation plutôt que de faire tourner directement l'objet. Des modifications importantes pourraient également être apportées à la fonction d'ajout d'objets, notamment en ajoutant un retour utilisateur sous la forme d'une version transparente de l'objet indiquant sa position d'apparition. Il serait également envisageable de permettre l'apparition d'objets dans le ciel. Enfin, il serait crucial d'intégrer des options d'accessibilité telles que le changement de langue, de police, et un mode adapté aux handicaps visuels tels que le daltonisme.

RÉFÉRENCES

[1] Jonali Baruah and Paul B. Paulus. “Collaborative creativity and innovation in education.” In: *Creativity Under Duress in Education? Resistive Theories, Practices, and Actions* (2019), pp. 155–177.

[2] Linda Candy Lena Mamykina and Ernest Edmonds. “Collaborative creativity.” In: *Communications of the ACM* 45.10 (2002), pp. 96–99.

[3] Laal Marjan and Ghodsi Seyed Mohammad. “Benefits of collaborative learning.” In: *Procedia-social and behavioral sciences* 31 (2012), pp. 486–490.

[4] Fourcan Karim Mazumder and Utpal Kanti Das. “Usability guidelines for usable user interface.” In: *International Journal of Research in Engineering and Technology* 3.9 (2014), pp. 79–82.

[5] *PoseMyArt*. URL: `https://posemy.art/`.

[6] Ton Roosendaal. *Blender*. Dernière version : 5 décembre 2023. 1994. URL: `https://www.blender.org/`.

[7] Anne-Laure Sellier and Darren W Dahl. “Focus! Creative success is enjoyed through restricted choice.” In: *Journal of Marketing Research* 48.6 (2011), pp. 996–1007.

[8] *Storyboarder*. URL: `https://wonderunit.com/storyboarder/`.

[9] *StoryboarderThat*. URL: `https://www.storyboardthat.com/fr/collaboration-en-temps-r%C3%A9el`.

[10] Christine Webster et al. “VR Auditory Space, Spatialisation du Son en Immersion VR.” In: *Journées d’informatique musicale, Strasbourg, GREAM/IRMA* (2020).

A ANNEXES

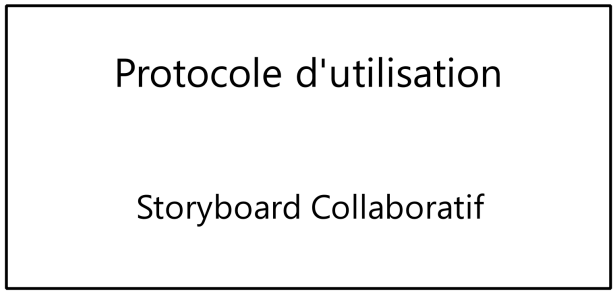


Figure A.1.: Première page du protocole d’utilisation

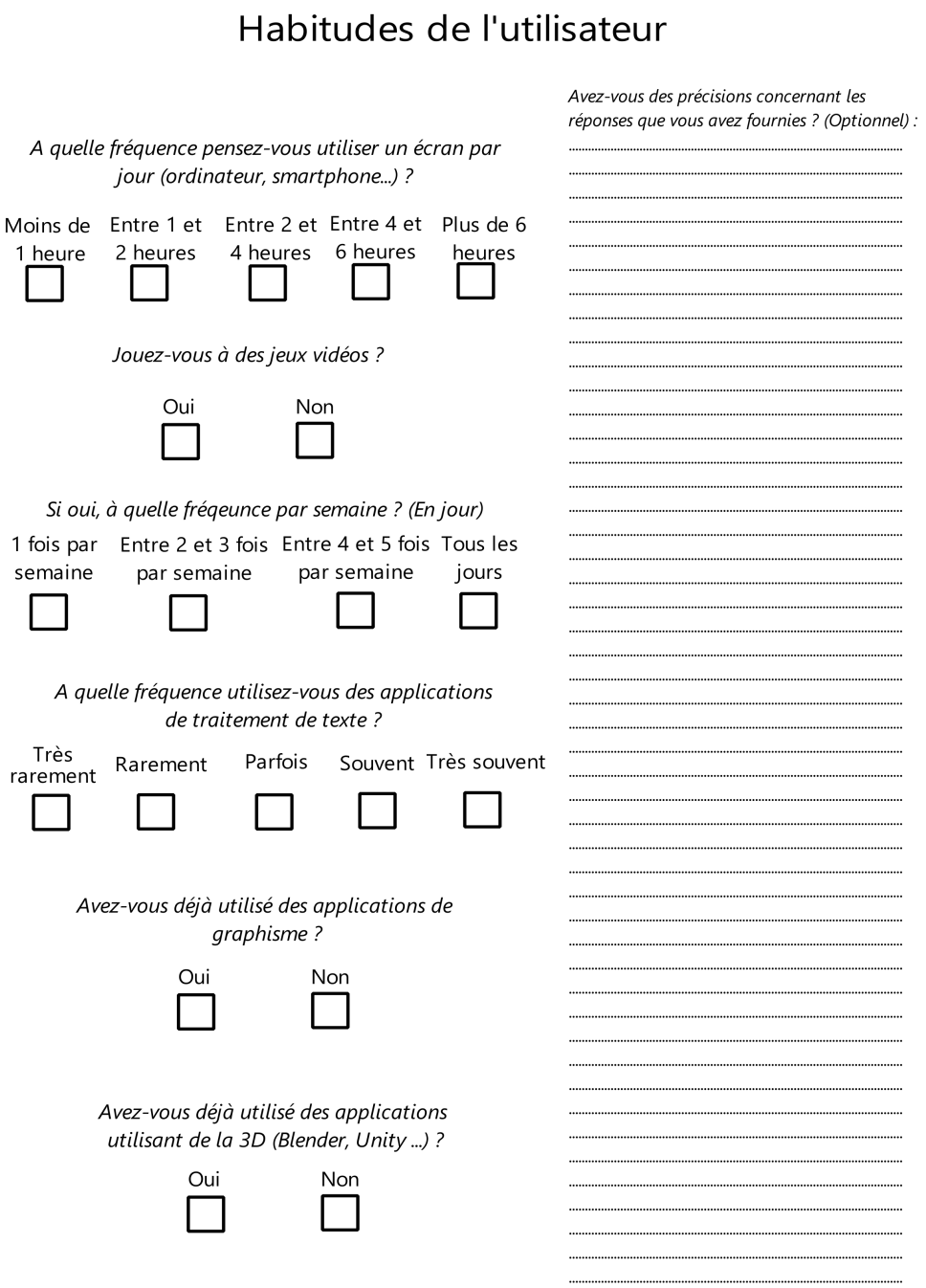


Figure A.2.: Deuxième page du protocole d'utilisation

Tâches à réaliser

Plusieurs tâches vous seront proposées, vous allez essayer de toutes les réaliser. Si vous n'y arrivez pas, vous pouvez demander de l'assistance à l'un d'entre-nous et si malgré cela vous n'y arrivez toujours pas, vous pouvez abandonner. Si vous effectuez ce test sans la présence réelle de l'un des développeurs, veuillez vous chronométrer à chaque étape.

1 . Se déplacer dans la scène

2 . Poser un objet dans la scène

3 . Déplacer un objet librement dans la scène

4 . Effectuer des modifications sur les propriétés d'un objet à l'aide de gizmo (mouvement, rotation et redimension d'objet)

5 . Manipuler les champs uniques de ces objets dans l'inspector (à l'aide de l'interface utilisateur)

6 . Changer la couleur d'un objet

7 . Manipuler le champs unique de chaque type de lumière

8 . Visualiser la scène depuis le point de vue d'une caméra

9 . Ajouter un filtre à une caméra

10 . Faire apparaître le point de fuite d'une caméra

11 . Prendre un screenshot depuis le point de vue d'une caméra

12 . Mettre en place une scène complète de manière cohérente

Figure A.3.: Troisième page du protocole d’utilisation

Autoévaluation des tâches

Maintenant que vous avez effectué les différentes tâches, vous allez pouvoir vous autoévaluer. Pour cela, un barème est à votre disposition. Vous allez écrire à côté de chacune de ces tâches la valeur numérique correspondant à la difficulté rencontrée. Si vous rencontrez une quelconque difficulté, n'hésitez pas à demander au développeur. Vous pouvez aussi, si vous le souhaitez ou que vous en trouvez la nécessité, écrire un commentaire sur la tâche en question (cela peut être une critique, un conseil...).

1 : Echec de la tâche 2 : A réussi avec de l'assistance 3 : A réussi avec difficulté	4 : A réussi avec un temps d'adaptation 5 : Réussite immédiate de la tâche

1 . ..../5
Commentaire : .....
.....
.....
.....

2 . ..../5
Commentaire : .....
.....
.....
.....

3 . ..../5
Commentaire : .....
.....
.....
.....

4 . ..../5
Commentaire : .....
.....
.....
.....

5 . ..../5
Commentaire : .....
.....
.....
.....

6 . ..../5
Commentaire : .....
.....
.....
.....

7 . ..../5
Commentaire : .....
.....
.....
.....

8 . ..../5
Commentaire : .....
.....
.....
.....

9 . ..../5
Commentaire : .....
.....
.....
.....

10 . ..../5
Commentaire : .....
.....
.....
.....

11 . ..../5
Commentaire : .....
.....
.....
.....

12 . ..../5
Commentaire : .....
.....
.....
.....

Figure A.4.: Quatrième page du protocole d'utilisation

Avez-vous d'autres avis à nous faire parvenir, des conseils, des critiques à apporter sur l'ensemble de l'application ?

Figure A.5.: Cinquième page du protocole d'utilisation