CLERMONT AUVERGNE INP Isima

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Internship Report of 2nd Year Engineering Student
Major: Interactive Systems Computing for Embedded, Robotics, and Virtual Reality

Achieving effective gaze on screen-based robot faces

Presented by : Thomas EKINDY NDAME

Academic mentor : Olivier RAYNAUD
Company mentor : Patrick LYNCH

Defense Date : 6 August 2023
Internship Duration : 4 months

Campus des Cézeaux . 1 rue de la Chébarde . TSA 60125 . 63178 Aubière CEDEX

Acknowledgments

I want to express my sincere gratitude to all those who have supported and guided me during my internship at the Department of Mechanical, Manufacturing, and Biomedical Engineering at Trinity College Dublin. Your assistance has been invaluable in making this experience enriching and memorable.

A special thank you goes to my academic supervisor, Olivier Raynaud. His consistent support and reassurance throughout my internship reflect his commitment to my personal and professional growth. Patrick Lynch, my dedicated mentor at Trinity College Dublin, deserves my heartfelt appreciation for fostering an engaging learning environment with his open discussions and enthusiasm. His dynamic approach and unwavering support made my lab experience both enjoyable and productive. I also appreciate the impactful contributions of Associate Professor Kevin Kelly, whose insights were instrumental in shaping my project. Moreover, I would like to extend my gratitude to Moyin Otubela, who has recently achieved her PhD and has greatly contributed to the organization of our work.

I'm grateful for the camaraderie and shared experiences of my fellow ISIMA interns, which enriched my time in Ireland both professionally and personally. The positive work environment, facilitated by the collaborative and friendly atmosphere among the Irish students in the lab, made our joint efforts memorable.

Lastly, I extend my heartfelt thanks to my family for their unwavering support, reminding me of the significance of my pursuits and encouraging me to excel.

List of Figures

Résumé étendu

Introduction

Le RAIL est un groupe de recherche qui se concentre sur la création de solutions techniques nouvelles et innovantes pour des défis importants dans la société. L'un de ces défis concerne l'interaction entre les humains et les robots, dans le but de rendre ces interactions aussi naturelles que possible du point de vue humain. Cela implique de reproduire les méthodes de communication humaine chez les robots. Le laboratoire met principalement l'accent sur le prototype de robot avancé appelé Stevie dans ce domaine. Stevie dispose de diverses méthodes de communication, allant des expressions faciales et de la voix synthétisée aux mouvements des bras et à l'orientation du corps, ce qui le rend interactif socialement avec les humains.

L'aspect de la communication étudié est l'un des plus cruciaux dans l'interaction humaine, et il concerne le regard. Pour améliorer cela dans le robot Stevie, il a été décidé de mener des expériences pour en évaluer l'efficacité. Initialement, ces expériences ont eu lieu dans le laboratoire RAIL, impliquant des personnes ordinaires, le robot Stevie et un membre du projet. La procédure consistait à placer des balles de tennis à hauteur des yeux autour du robot, à déplacer sa tête et ses yeux de manière spécifique, à positionner le participant humain et à lui demander d'identifier quelle balle de tennis, selon lui, le robot regardait.

Cependant, cette approche posait des défis en termes d'équilibre entre les données collectées et le temps consacré à l'organisation des expériences. Comme solution, l'idée de créer une simulation virtuelle en ligne a émergé, accessible via des ordinateurs, afin d'atteindre un public plus large et de recueillir davantage de données.

Le segment suivant plongera dans la dynamique entourant le contexte de travail et mettra en évidence les défis et les obstacles que la simulation devra surmonter.

La section suivante examinera en détail comment la solution a été développée.

Enfin, la troisième section résumera le travail mis en œuvre, abordera ses limites et suggérera des étapes futures potentielles pour la poursuite du projet pour le RAIL.

Méthodologie

Méthodologie Agile

Au sein du RAIL, j'ai fait partie d'un groupe d'étudiants de l'ISIMA, impliqués dans différents projets assignés à notre arrivée. Les méthodologies de travail ont été introduites par Kevin Kelly, professeur associé. L'adoption de la méthodologie Agile a été un choix unanime en raison de notre environnement collaboratif.
L'utilisation d'un tableau Agile a permis de gérer efficacement les tâches et les réunions de scrum étaient au cœur de notre routine quotidienne. Des réunions avec mon superviseur, Patrick Lynch, ont également été organisées pour examiner les progrès du projet. La planification des projets a évolué pour s'adapter aux besoins. Le passage d'un tableau physique à une alternative virtuelle a été motivé par des considérations pratiques.

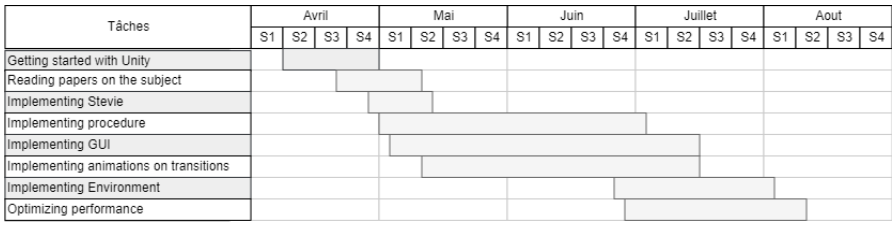


Figure 1: Diagramme de Gantt

De par la flexibilité et le principe de fonctionnement de la méthodologie Agile aucun diagramme de Gantt prévisionnel n'a été employé durant ce projet. Cependant un diagramme de Gantt réel, visible à la figure 2.4, a été conçu à l'issu du projet.

Documentation

L'adaptabilité est une contrainte majeure du projet, déterminant sa valeur et son utilité pour les futurs projets du RAIL. Des mesures ont été prises pour rendre le code plus compréhensible.

La documentation créée à l'aide de Doxygen a été implémentée, générant automatiquement une documentation organisée à partir du code source. Cette documentation améliore la lisibilité et la maintenabilité du code, en fournissant des explications complètes sur les fonctions et leur utilisation. Elle sert de référence précieuse pour les développeurs actuels et futurs, facilitant la compréhension et la modification du code.

Dans le contexte du projet de simulation, utiliser Doxygen pour documenter le code C# est essentiel pour garantir que la logique, le comportement et les composants de la simulation sont bien documentés.
Cette documentation facilite la navigation dans le code et permet aux développeurs de prendre des

décisions éclairées.

En somme, l'expérience au RAIL a été enrichie par la méthodologie Agile et grandement facilitée par l'utilisation de la documentation Doxygen, fournissant un cadre collaboratif pour une communication efficace, une adaptation et une amélioration continue dans nos projets dynamiques.

Matériel

Outil de développement

Lors de notre première réunion, nous avons discuté du choix des outils pour développer le projet de simulation, évaluant les avantages de Blender, Webots et Unity.
Finalement, Unity a été privilégié en raison de son interface conviviale qui facilite la création de simulations complexes sans nécessiter une expertise en programmation, de sa bibliothèque d'éléments pré-construits qui accélère le processus de développement, de sa communauté active et de sa documentation approfondie.
La décision a été prise d'utiliser Unity comme principal outil de développement pour la simulation, en tenant compte de son potentiel pour garantir une progression fluide du projet, augmenter l'efficacité et faciliter la collaboration entre les membres de l'équipe.

Unity

Unity constitue le socle de notre simulation, combinant habilement conception et programmation. Les GameObjects, éléments fondamentaux de Unity, sont au cœur de cette puissance.
Ils représentent les entités dans la simulation, comme objets, personnages et caméras. Leur hiérarchie permet d'organiser et de lier les éléments.
En ajoutant des
glsscripts aux GameObjects, j'ai créé des comportements dynamiques, et l'utilisation du GameManager d'Unity a géré les mécanismes globaux de la simulation. En somme, Unity crée une expérience immersive pour Stevie, notre robot virtuel.

Procédure

La procédure expérimentale est essentielle pour notre étude et découle des travaux précédents d'Eoghan O'Leary sur le regard de Stevie. Une réunion a été tenue pour discuter de sa mise en place au sein du projet. Nous avons défini des objectifs clairs et structuré la procédure en deux parties distinctes pour explorer comment les gens perçoivent le regard d'un robot dans différentes situations. Nous avons aussi examiné l'effet de la position du testeur et introduit de l'aléatoire dans les mouvements des yeux et de la tête du robot pour améliorer la réalisme.
La première partie de la procédure implique la révélation du robot, où l'humain est sollicité pour indiquer où il pense que le robot regarde. Ensuite, les mouvements des yeux et de la tête du robot

sont exécutés, suivis d'une nouvelle interrogation de l'humain sur la direction du regard du robot. Lors de la révélation du robot, nous évaluons son interprétation en tant que simple spectateur. Après les mouvements du robot, nous testons l'impact de l'affichage des mouvements des yeux et de la tête sur la perception de l'humain, ce qui diffère du scénario précédent. Cela nous permet d'évaluer l'importance des mouvements dans l'interprétation du regard. Dans la deuxième partie de la procédure, une légère modification est apportée pour évaluer d'autres facteurs. Une fois le robot révélé, l'humain est invité à se déplacer là où il pense que le robot regarde. Cela place l'humain en tant que participant dans une conversation simulée avec le robot. Ensuite, les yeux et la tête du robot sont déplacés, et enfin, l'humain donne son avis sur l'objet observé, après avoir été témoin des mouvements des yeux et de la tête. Cette étape examine l'interprétation du regard du point de vue de quelqu'un directement observé par le robot, plutôt que de celui d'un simple observateur extérieur.
Cette procédure améliorée s'inscrit dans la continuité de recherches antérieures et permet des modifications futures tout en maintenant l'intégrité du code. La contrainte d'adaptabilité m'a poussé à concevoir une mise en œuvre flexible de la procédure. Les événements de la procédure sont définis dans le script GameManager, agissant comme un composant central. Cela assure une exécution fluide grâce à un système d'états et de transitions géré par le GameManager. Cette approche facilite les modifications futures de la procédure sans altérer l'ensemble du code.

Robot Stevie

L'implémentation du robot Stevie dans la simulation se décline en deux parties : statique et dynamique. Dans la première, nous utilisons le modèle préexistant de Stevie, mais avec un défaut : l'absence des yeux. Nous avons donc introduit un modèle d'œil trouvé pendant les expériences en laboratoire.

Dans la partie dynamique, nous avons ajusté les positions des pupilles dans les yeux pour reproduire les mouvements réels de Stevie. Pour réaliser des mouvements fluides et cohérents des yeux et des pupilles, nous avons utilisé le concept de Delta Time, qui nous permet de récupérer le temps écoulé entre deux images consécutives (frames). Cette technique a également été appliquée aux mouvements de la tête de Stevie pour les rendre plus fluides.

Cependant, la complexité du modèle de Stevie a affecté les performances de la simulation, provoquant un ralentissement. Nous avons résolu ce problème en simplifiant le maillage du modèle, ce qui a considérablement amélioré les performances de rendu tout en préservant la qualité visuelle globale.

Environnement

L'environnement joue un rôle crucial dans la mise en œuvre de l'aspect utilisabilité de l'expérimentation. L'objectif était de donner aux testeurs une meilleure perception de la perspective et de la profondeur,

notamment en ce qui concerne la distance entre le robot et les objets environnants. Pour ce faire, des actifs d'environnement préfabriqués ont été sélectionnés dans le "Unity Asset Store". Ces actifs comprenaient des objets du quotidien permettant aux testeurs d'estimer l'échelle, ce qui est essentiel pour les distances entre les éléments de simulation.

Cependant, les environnements ont également affecté les performances de la simulation, tout comme le robot Stevie. Des environnements moins complexes, modélisés avec moins de polygones, ont montré de meilleures performances. La qualité des ombres a également été identifiée comme un facteur ayant un impact négatif sur les performances. Les ombres, bien que cruciales visuellement, peuvent être gourmandes en ressources. En ajustant les paramètres de qualité des ombres, notamment le type d'ombre et la résolution, il était possible d'améliorer considérablement les performances sans sacrifier la qualité visuelle.

Ces facteurs soulignent la nécessité de trouver un équilibre entre la fidélité visuelle et les performances pour garantir une expérience utilisateur fluide et réactive dans la simulation. Bien que la précalculation de l'éclairage puisse offrir des avantages à court terme, elle présente des complexités et des limitations potentielles, nécessitant une intervention manuelle et une gestion des données d'éclairage générées. Cette approche peut être moins adaptée aux projets à long terme, en raison des défis liés à la maintenance, à la compatibilité et à la collaboration entre développeurs.

Interfaces Graphiques Utilisateurs

L'interface utilisateur graphique (GUI) était un élément essentiel de l'expérience de la simulation. Unity offre un système puissant avec le composant "Canvas" pour créer des interfaces interactives. Les Canvas agissent comme des surfaces virtuelles pour les éléments d'interface, tels que les boutons et le texte. Ils sont organisés en hiérarchie, avec différents sous-Canvas ayant des fonctions spécifiques.

Un sous-Canvas permet aux utilisateurs de choisir l'objet que le robot observe, utilisant des boutons "Toggle". La soumission de la réponse nécessite la sélection et le maintien du bouton pour éviter les erreurs. Un autre sous-Canvas facilite le positionnement de l'utilisateur par rapport à la direction du robot via un curseur rotatif. L'interface d'instruction guide les utilisateurs à travers les étapes avec des flèches interactives.

Dans le sous-canvas d'instruction, une palette de couleurs apaisantes, basée sur des recherches, a été choisie pour éviter la fatigue visuelle des testeurs. Des flèches interactives et une animation fluide ont été utilisées pour faciliter la navigation entre les pages d'instructions. La conception a été pensée de manière à garantir une expérience utilisateur agréable et conviviale.

Ces interfaces améliorent l'expérience utilisateur en favorisant une interaction fluide et visuellement agréable. L'utilisation de fonctionnalités polyvalentes de Unity et d'optimisations telles que la simplification de maillage et l'ajustement des ombres contribue aux performances et à la qualité de la simulation.

Réalisation Finale

Le produit final permet des simulations d'expérience sur ordinateur local. Les instructions interactives en bleu fournissent des informations essentielles aux testeurs tout en minimisant les distractions. Les testeurs peuvent revenir en arrière s'ils manquent une instruction.
La sélection d'objets par les testeurs et leur positionnement sont facilités par une interface intuitive et transparente. Cependant, des limitations de flexibilité dans le code ont été identifiées. Les environnements pré-conçus ont posé des problèmes de performances, soulignant l'importance de l'équilibre entre esthétique et efficacité.
La simulation a été conçue avec une adaptabilité et une utilisabilité élevées, mais des ajustements au code pourraient améliorer sa polyvalence. Les résultats actuels servent de base pour des améliorations futures, bien que le déploiement en ligne reste à accomplir.

Limitations et Recommandations Futures

Intégration de la Simulation Basée sur le Web

Malgré la mise en place de la simulation, son déploiement en ligne reste en suspens. Cependant, l'utilisation d'Unity facilitera ce processus, notamment grâce à WebGL, une API JavaScript qui permet le rendu de graphismes 3D et 2D directement dans les navigateurs web sans nécessiter de plugins.
Cette technologie permet aux projets Unity d'être exécutés et expérimentés dans les navigateurs, élargissant ainsi l'accessibilité du contenu 3D créé avec Unity au-delà des installations de logiciels traditionnelles.

Limitations de l'Adaptabilité

Lors de la création d'une simulation à partir de zéro, j'avais initialement imaginé un haut degré d'adaptabilité, mais j'ai rencontré des obstacles majeurs en réalité. Les avantages potentiels étaient considérables, notamment la possibilité d'ajuster facilement des variables telles que le nombre de réponses disponibles pour les testeurs, de créer une interface réactive adaptable à des besoins divers et de modifier dynamiquement le nombre d'objets autour du robot.
Cependant, la réalité s'est avérée différente. J'ai codé la simulation d'une certaine manière, et ajuster ces paramètres nécessite désormais de recoder une partie substantielle de l'application. J'ai

perdu de vue la flexibilité que j'avais initialement intégrée, restreignant involontairement le rôle de la simulation. Cette prise de conscience a marqué un tournant où l'adaptabilité a été entravée.

En essence, la structure actuelle de la simulation permet des ajustements mineurs, mais des changements substantiels sont contraints par les bases rigides sur lesquelles elle repose. Bien que de nombreux composants puissent être réutilisés pour des expériences futures en ligne d'interactions humain-robot, la simulation elle-même manque de l'adaptabilité inhérente nécessaire à différentes utilisations pour lesquelles elle était initialement prévue. Cette prise de conscience découle d'une sous-estimation de l'importance primordiale de maintenir la conception de la simulation ultra-adaptable, comme elle l'était au cours du premier mois du projet. Cette erreur de calcul m'a conduit à adopter une solution qui manque finalement de la flexibilité recherchée.
En conclusion, bien que la capacité d'adaptabilité de la simulation offre la possibilité d'améliorations progressives dans son périmètre initial, elle présente des limites lorsqu'il s'agit de transformer sa structure fondamentale.

Choix d'Implémentation de l'Environnement

En réfléchissant à mes choix tout au long de ce projet, une autre décision que je regrette concerne la mise en place d'un environnement à partir de l'Unity Asset Store. Cette décision a involontairement entraîné des défis d'optimisation des performances, principalement en raison du fait que les environnements pré-conçus sont souvent associés à des demandes de ressources exceptionnellement élevées.

Une approche plus avisée aurait été de créer un environnement riche et immersif permettant aux testeurs de se sentir immergés dans la scène et d'évaluer les relations spatiales entre les éléments de simulation, en partant de zéro. Cela aurait pu être réalisé en assemblant des composants d'environnement plus petits provenant de l'Unity Asset Store. Par exemple, créer une pièce simple avec six murs et l'aménager avec des objets comme une table et divers éléments aurait probablement évité les problèmes de performances, répondant ainsi à la plupart des exigences environnementales prévues.

De plus, une quantité importante de temps a été investie dans la manipulation de ces environnements étendus, ce qui malheureusement n'a pas abouti à des retours proportionnels en termes d'utilisabilité et de performances. Repenser le processus de création de l'environnement et opter pour une approche de construction plus personnalisée, modulaire et efficace aurait été une voie plus judicieuse à suivre.

Il est évident que j'ai beaucoup appris en cours de route. Bien que mon objectif final de déployer la simulation en ligne n'ait pas encore été atteint, mes réalisations actuelles sont notables et fonctionnelles.

Les principales fonctionnalités de l'application sont en place et fonctionnent bien pour les simulations locales. C'est un progrès important et cela forme la base de mon travail futur. Bien que l'application ne soit pas entièrement flexible en termes de code, elle remplit efficacement son objectif initial.

Cependant, j'ai également reconnu une opportunité manquée. Avec une meilleure flexibilité du code, l'application aurait pu être plus polyvalente et bénéfique pour mon travail au sein du RAIL. Cette prise de conscience montre qu'il y a de la place pour l'amélioration et l'expansion au-delà de ce que j'ai réalisé jusqu'à présent.

Conclusion

Afin de progresser dans le domaine des interactions homme-robot au sein du RAIL et en tirant parti des travaux antérieurs dans ce domaine, j'ai été chargé de contribuer à mesurer l'efficacité des robots à écran pour transmettre leur regard aux humains. Initialement réalisées en laboratoire avec des humains et le robot Stevie, ces mesures posaient d'importants défis organisationnels. Ainsi, une simulation 3D de l'expérience a été développée pour une utilisation en ligne, permettant d'économiser considérablement du temps dans la collecte de données des testeurs. Mon rôle consistait à contribuer à la réalisation de cette simulation.
Dans ce rapport, nous explorons en détail la solution que j'ai apportée au problème. J'ai réussi à créer une simulation fonctionnelle qui a relevé de nombreuses contraintes, tout en maintenant le réalisme et les performances de la simulation. Cependant, la simulation n'a pas pu être déployée en ligne en raison de problèmes techniques liés à l'outil Unity. Bien que l'outil soit intuitif en surface, des problèmes techniques peuvent surgir dans des aspects plus complexes, sans toujours trouver de solutions sur les forums.

Bien que le travail achevé ne soit pas totalement terminé, ma contribution a considérablement fait avancer la création de cette simulation en ligne. Néanmoins, malgré la documentation disponible, le manque de flexibilité dans les mécanismes sous-jacents de la simulation pourrait exiger que les futurs développeurs possèdent une expertise significative en programmation et en développement Unity pour poursuivre et faire progresser efficacement le projet.
En rétrospective, je pense que même si cela a ralenti la phase intermédiaire de mon stage, j'aurais dû conserver la flexibilité présente dès le début dans mon code. Cela aurait pu simplifier le développement global de la simulation à long terme.

Je suis d'avis que le développement ultérieur de cet outil devrait être entrepris par une personne qualifiée et expérimentée en développement Unity. Bien que la charge de travail restante pour le déploiement de l'application sur un site web soit encore importante, elle est considérablement moindre que celle nécessaire pour construire la simulation à partir de zéro. Ainsi, la poursuite de

son développement est une option envisageable.

Si la simulation que j'ai créée est jugée insuffisamment flexible pour un développement ultérieur, il serait avisé de récupérer certains éléments de cette simulation pour en développer une nouvelle. Les composants les plus importants à mon avis seraient le modèle de Stevie ainsi que ses scripts d'interaction, les éléments graphiques de l'interface utilisateur, les scripts qui les gèrent et toutes les informations incluses dans ce rapport. Ces éléments pourraient grandement contribuer à la création d'une nouvelle simulation.

Abstract

Gaze is an important part of communication. Many humanoid robots use screens as a face, and conveying gaze using a face on a 2D screen is difficult. Building on previous work by the Robotics and Innovation Laboratory (RAIL) at Trinity College Dublin, particularly the robot **'Stevie'**, this project explores the efficacy of conveying gaze through 2D animations, robotic neck movements, and a combination of both.

To address this matter, it was determined to develop a **3D simulation**. This **simulation** is intended to be accessible **online** through a web platform on personal computers. This **simulation** mirrors the actions of the **'Stevie'** robot, specifically the eyes and head movements, following a particular procedure. Within the **simulation**, an interactive graphical interface prompts users to respond to posed questions. The collected data serves in measuring the effectiveness of 2D eye animations and robotic neck movements.

This internship report outlines the conception of the **3D simulation** using the **Unity** framework, **C#** programing language and **WebGL**, encompassing procedural and ergonomic choices, encountered technical challenge related to performance optimization, and strategies to ensure operational flexibility. While currently functional for local use, where users can engage with the **simulation** executable on their computers and respond to questions throughout the experimentation, the **simulation** remains a **work in progress**. As it is designed for future continuation by other developers, the deployment of the **simulation online** is pending.

Key-words: Unity, C#, WebGL, 3D simulation, online, work in progress

Contents

Introduction

The RAIL is a research group that focuses on creating new and innovative technical solutions for important challenges in society. One of these challenges is about how humans and robots interact, with the goal of making these interactions as natural as possible from a human perspective. This involves replicating human communication methods in robots. The lab's primary focus in this area is the advanced robot prototype called Stevie. Stevie has various communication methods, ranging from facial expressions and synthesized voice to arm movements and body orientation, making it socially interactive with humans.

The communication aspect under study is one of the most crucial in human interaction, and it pertains to gaze. To enhance this in the Stevie robot, a decision was made to conduct experiments to gauge its effectiveness. Initially, these experiments took place in the RAIL lab, involving ordinary people, the Stevie robot, and a project member. The procedure included placing tennis balls at eye level around the robot, moving its head and eyes in specific ways, positioning the human participant, and asking them to identify which tennis ball they thought the robot was looking at.

However, this approach posed challenges in terms of the balance between data collected and time spent organizing the experiments. As a solution, the idea of creating a virtual simulation online emerged, accessible through computers, to reach a broader audience and gather more data.

This following segment will delve into the dynamics surrounding the work context and highlight the challenges and obstacles that the simulation will have to overcome.

The next section will offer a thorough examination of how the solution was developed.

Finally, the third section will summarize the implemented work, address its limitations, and suggest potential future steps for the project's continuation for the RAIL.

1 Context

1.1 Work Environment

The internship leading to the writing of this report was conducted at the RAIL of Trinity College, a well-known university in Dublin.

1.1.1 Trinity College Dublin

1.1.1.0.1

Trinity College Dublin, commonly referred to as Trinity College, is a prestigious institution located in the heart of Dublin, Ireland, with its organizational chart displayed in the figure. 1.1. Established in 1592, it stands as one of Europe's oldest universities and boasts a global reputation as a hub for academic excellence, cutting-edge research, and innovation.

Trinity College is renowned for providing a dynamic and challenging learning environment, where students are encouraged to cultivate critical thinking, creativity, and problem-solving skills. The university offers a wide range of undergraduate and graduate programs in various fields, including humanities, social sciences, natural sciences, engineering, arts, and health sciences.

The academic prowess of Trinity College is evident in its highly qualified and dedicated faculty, comprised of experts and researchers renowned in their respective fields. Students have the opportunity to engage with these professors and participate in groundbreaking research projects that contribute to advancing knowledge and addressing global issues.

Beyond its commitment to academic excellence, Trinity College stands out for its iconic historic campus. The campus is home to grand historic buildings, exceptional libraries, and state-of-the-art modern facilities. The Trinity College Library, in particular, is famous for housing the Book of Kells, a ninth-century illuminated manuscript considered one of the world's most precious medieval treasures.

As an internationally renowned institution, Trinity College Dublin maintains partnerships and

Date: 12 July 2023

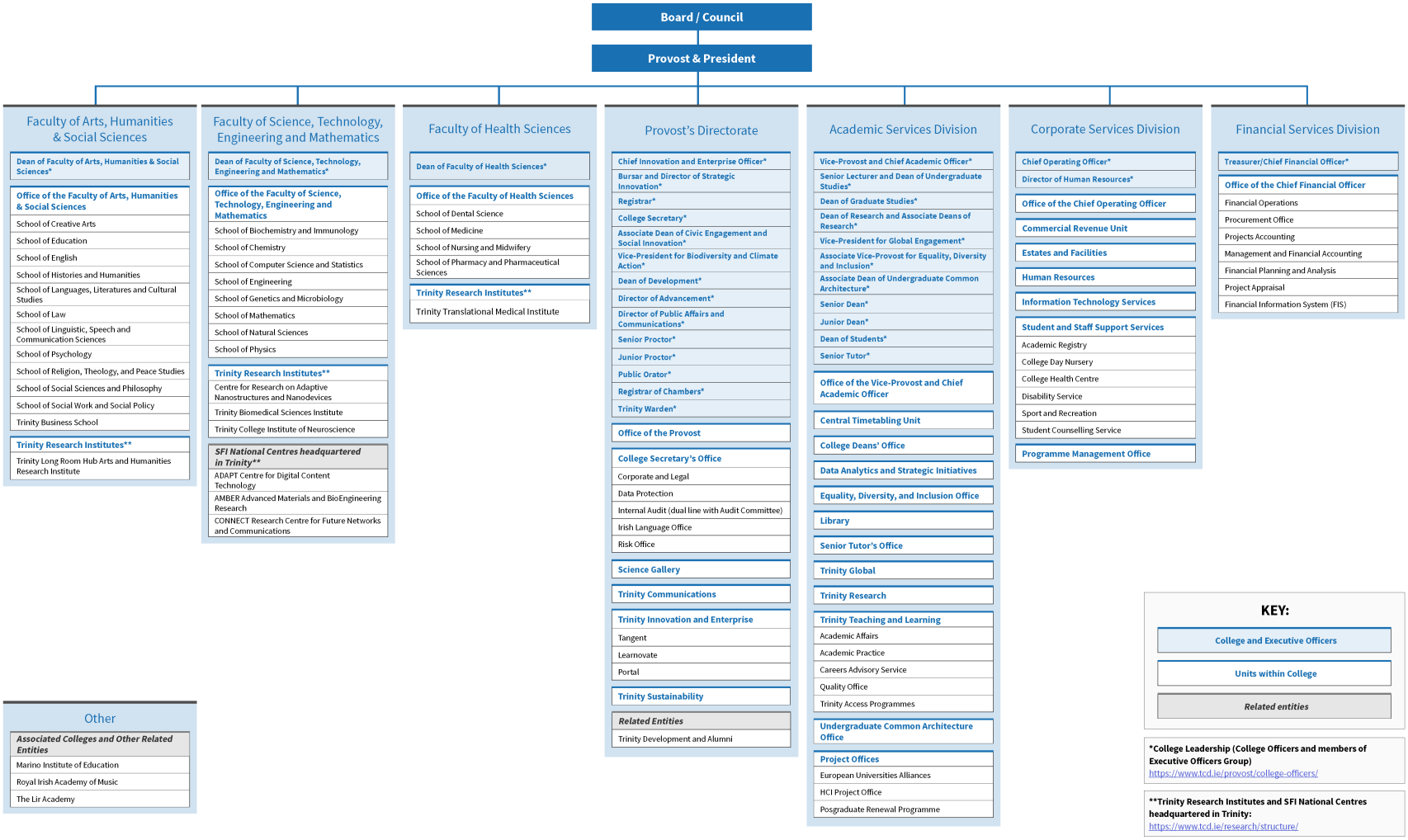


Figure 1.1: Trinity College Dublin's organization chart

collaborations with other universities, businesses, and research organizations worldwide. This global outlook promotes knowledge exchange, cultural diversity, and the creation of innovative solutions to contemporary challenges.

In summary, Trinity College Dublin embodies academic excellence, cutting-edge research, and a global perspective. It provides students with an invaluable educational experience that fosters intellectual, personal, and professional growth, while actively contributing to the advancement of society and knowledge on a global scale.

1.1.2 Robotic And Innovation Laboratory

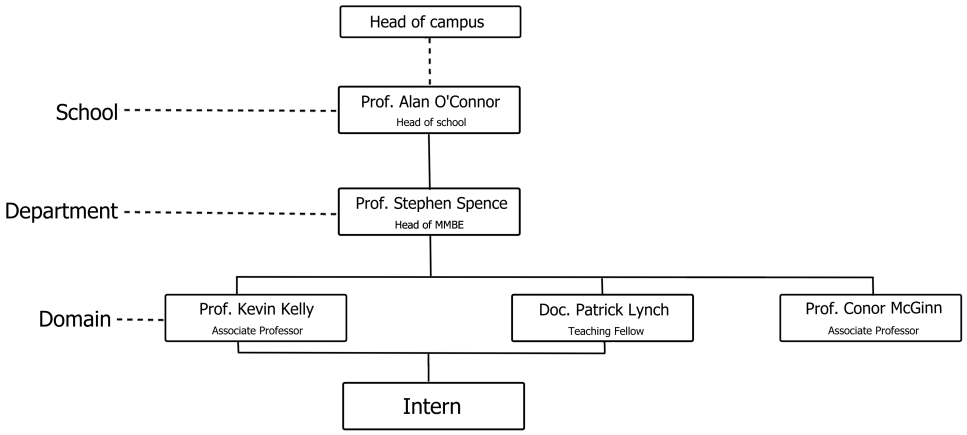


Figure 1.2: RAIL's organization chart

The Robotics and Innovation Laboratory (RAIL) at Trinity College Dublin is a dynamic research center dedicated to pushing the boundaries of robotics and fostering innovation, its organization chart is displayed on the figure 1.2. It serves as a creative space where cutting-edge ideas flourish, and research efforts are channeled towards addressing real-world challenges. RAIL's impressive arsenal includes advanced facilities and an array of robotic platforms, enabling researchers to delve into intricate studies in fields such as human-robot interaction, artificial intelligence, and machine learning.

RAIL's collaborative spirit is evident through its partnerships with industry leaders and academic institutions, facilitating the exchange of ideas, knowledge, and expertise. These collaborations not only accelerate the development of groundbreaking technologies but also contribute to the practical implementation of innovative solutions.

Education is a cornerstone of RAIL's mission, with the laboratory actively engaging in workshops, seminars, and outreach programs. By fostering learning and exploration, RAIL nurtures the

growth of future innovators, ensuring a continuous stream of creative minds contributing to the field of robotics.

In essence, the Robotics and Innovation Laboratory stands as a nexus of technological advancement, collaboration, and education, shaping the landscape of robotics and driving innovation forward.

1.2 Project Scope

To recap, the project aims to create an accessible online 3D simulation for conducting experiments on how the robot Stevie communicates its gaze. The primary goal of this simulation is to make organizing experiments easier, allowing us to measure the robot's effectiveness and gather more data compared to in-person experiments.

1.2.1 Maintaining Procedure Consistency

Gaze is a crucial element in communication, influencing factors like familiarity, enjoyment, activity, and even performance, as shown in a study that explored how the robot's gaze impacts human impressions [2]. However, the experiments planned for the simulation are focused on measuring how well the robot can convey to humans where it is looking.
Understanding how Stevie functions is essential to know how it can adjust its gaze.

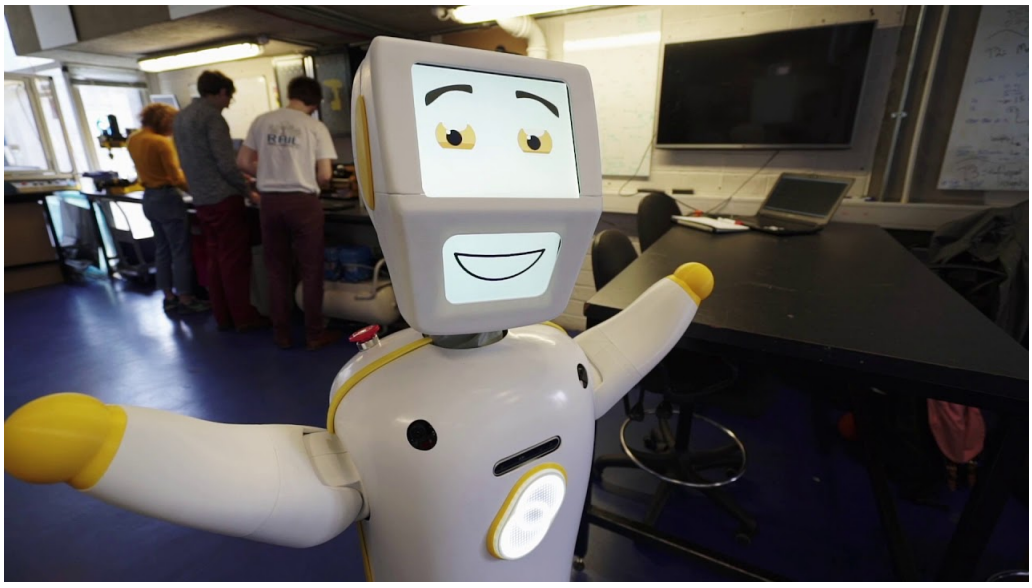


Figure 1.3: Photo of Stevie in motion

As seen in Figure 2.10, Stevie employs various ways to convey information, such as arm movements and facial expressions. Of particular significance to the project is its equipped 2D screen on its face, depicting eyes, eyebrows, and pupils, combined with the ability to turn its head. It's worth

noting that when the robot's eyes move, its pupils move within its eyes to enhance the naturalness of the motion.
These details present an initial technical challenge: accurately replicating the robot's gaze functionality as it exists in reality. Ensuring that the simulation closely resembles real-world conditions is crucial for obtaining valuable results.

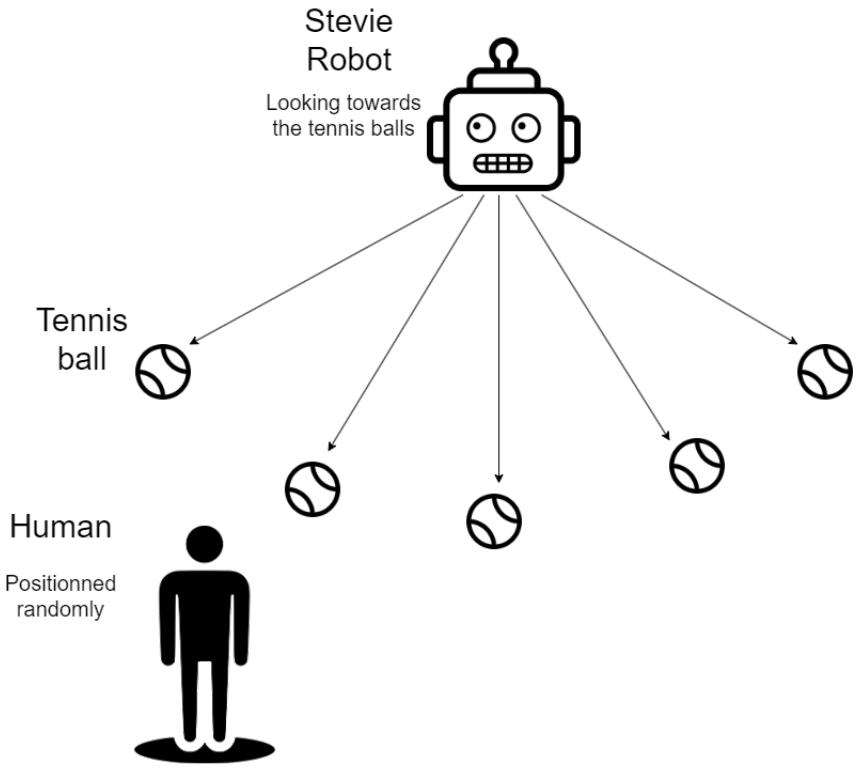


Figure 1.4: Experimental Setup Diagram

Furthermore, it's important to establish a well-thought-out experimental procedure. Initially, the assessment of Stevie's gaze effectiveness was conducted directly in the laboratory following a specific process.
This involved placing 5 tennis balls at eye level on poles, as depicted in Figure 1.4, and positioning the tester at a specific spot within the robot's field of view but not between it and the balls.
All of the robot's head movements were conducted along the vertical axis, and during the eye movements, the pupils also shifted within the robot's eyes.

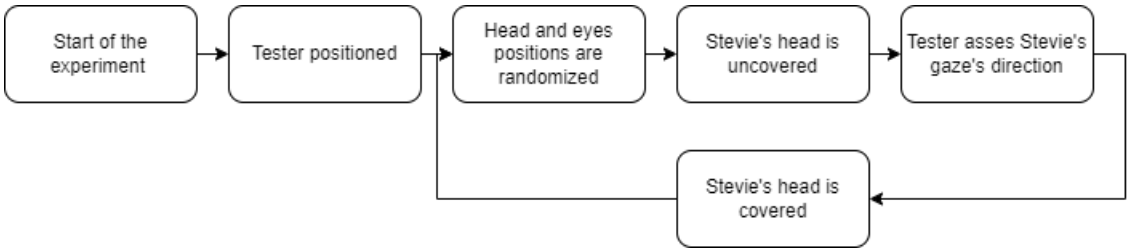


Figure 1.5: First Procedure's Functional Diagram

In the first stage, the steps outlined in Figure 1.5 were followed. The robot's head movements and eye movements were adjusted to have it focus on a particular location, and the tester was then asked to identify the tennis ball that the robot was observing.

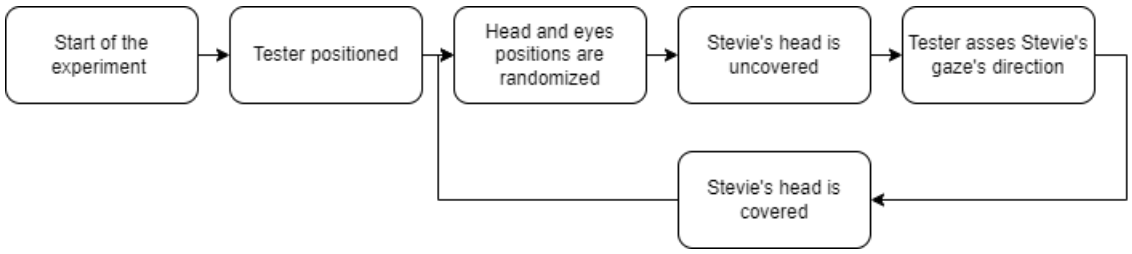


Figure 1.6: Experimental Functional Diagram

An additional variation was introduced, as illustrated in Figure 1.6. This variation involved covering Stevie's face with a cloth before revealing it to the tester, who would then find the robot already looking in a certain direction. Unlike the first procedure, in this case, the tester could not see the robot's head or eye movements.

Furthermore, the robot's range of potential head and eye positions was intentionally restricted. Five eye positions and three head positions were preconfigured, offering a carefully limited yet diverse array of feasible orientations.

Since the simulation will be viewed by a tester on a computer screen, it may be challenging for them to have a good sense of distances between objects, as compared to the real-world experimental setup. To provide them with a sense of perspective in the space where the experimentation will take place, it is necessary to incorporate an environment.

Moreover, the testers provided their answers through a questionnaire. The presence of this interaction also implies the existence of an interface through which the user can respond to questions posed by the simulation through the same interface.

This experiment was conducted by Eoghan O'Leary, a student from Trinity College, during the

writing of his dissertation on achieving effective gaze direction readability in embodied robots with screen-based faces.

The work described in this report is thus a continuation of his research, highlighting the significance of considering the conditions of his experiment.
It is evident that the experiment will need to be revised to suit its new format, which is an online simulation, while still adhering to the constraints of similarity with the real world.

1.2.2 Compatibility challenges

As we move into conducting experiments online, it's true that testers no longer need to physically come to the lab. However, they do need the right digital tools to access the simulation. In today's modern era, having a computer at home with an internet connection is quite common. We can assume that most people have a computer or, at the very least, the absence of a computer is not a factor that would discourage us from choosing to conduct an online simulation for this experiment.

Nevertheless, we need to consider the diversity of computers. Computers come in various models and capabilities, and this variation matters. The simulation we are dealing with is quite complex, involving 3D graphics and visuals, which can be resource-intensive for some computers. While certain computers might handle the simulation smoothly, others might struggle due to their limitations. This diversity in the capabilities of different computers is an important aspect to keep in mind.

While it may be plausible to overlook the subset of the population devoid of computer access, it is pivotal not to underestimate those who possess computers with lower capabilities. This brings forth a challenge: how to ensure the simulation runs well on both stronger and weaker computers. This is where our research comes into play. We aim to strike a balance – to ensure that the simulation is accessible to all while still meeting our scientific objectives.

The crux of the matter lies in finding a harmonious fit between the requirements of the experiments and the varying capabilities of different computers. This means it is needed to discover how to make the simulation function smoothly on computers with fewer resources, all while maintaining its quality.

1.2.3 Adaptability challenges

The initial goal of the simulation is to effectively carry out a specific experiment. Looking ahead, it's undoubtedly wise to anticipate that all future experiments related to human-robot interactions at the RAIL will greatly benefit from being conducted online through simulations. As such, it becomes crucial to make this simulation adaptable, meaning it should be easily modifiable and upgradable

by other developers, allowing them to introduce new features.

Even without delving into new features, it's essential for upcoming developers to have the capability to conveniently adjust parameters that can slightly alter the experiment. The reason for this lies in the uncertainty of whether the final product will forever remain in its most optimized state. The field of research and technology is ever-evolving, and what works best today might not be the same tomorrow.

Considering the dynamic nature of scientific progress and the continuous development of new methodologies, being able to tweak the simulation without taking over the entire system becomes a significant asset. This flexibility ensures that as new insights emerge, the simulation can be aligned with the latest advancements, thereby enabling the most relevant and accurate experimental outcomes.

1.2.4 Usability challenges

Initially, when the experimentation was conducted directly at the RAIL, an active project member was present to carry out the procedure and ensure that the experimental conditions were consistent for each participant.

With an online simulation replacing an in-person experiment, it becomes challenging to ascertain whether users are conducting it under the intended conditions. For instance, users might get distracted by external elements outside the simulation, disrupting the conditions, or even prematurely terminating the session. There's also a possibility that users might not comprehend the provided instructions, leading to skewed experimental data.

As a result, it becomes imperative to implement measures, including ergonomic considerations, to enhance the user experience. This ensures users remain engaged and focused on the experiment, while also prioritizing intuitiveness to minimize misunderstanding and errors in execution.

As we've just discussed, the simulation is poised to encounter a multitude of challenges. We've shed light on these obstacles, encompassing the transformation of real-world elements into a virtual environment without compromising the experimentation's integrity. Additionally, we've addressed concerns regarding the simulation's compatibility with less powerful computers, its adaptability for easy modification and enhancement, and its usability to ensure that testers engage in the simulation without distractions and complete the process diligently.

Moving forward, we'll delve into the strategies employed to tackle these constraints effectively.

2 Development

2.1 Methodology

2.1.1 Agile method

During my time at the RAIL, we were a group of 9 ISIMA students, myself included, embarked on separate projects, each assigned to us upon our arrival. As we entered the laboratory, we were introduced to the work methodologies by Kevin Kelly, an Associate Professor.

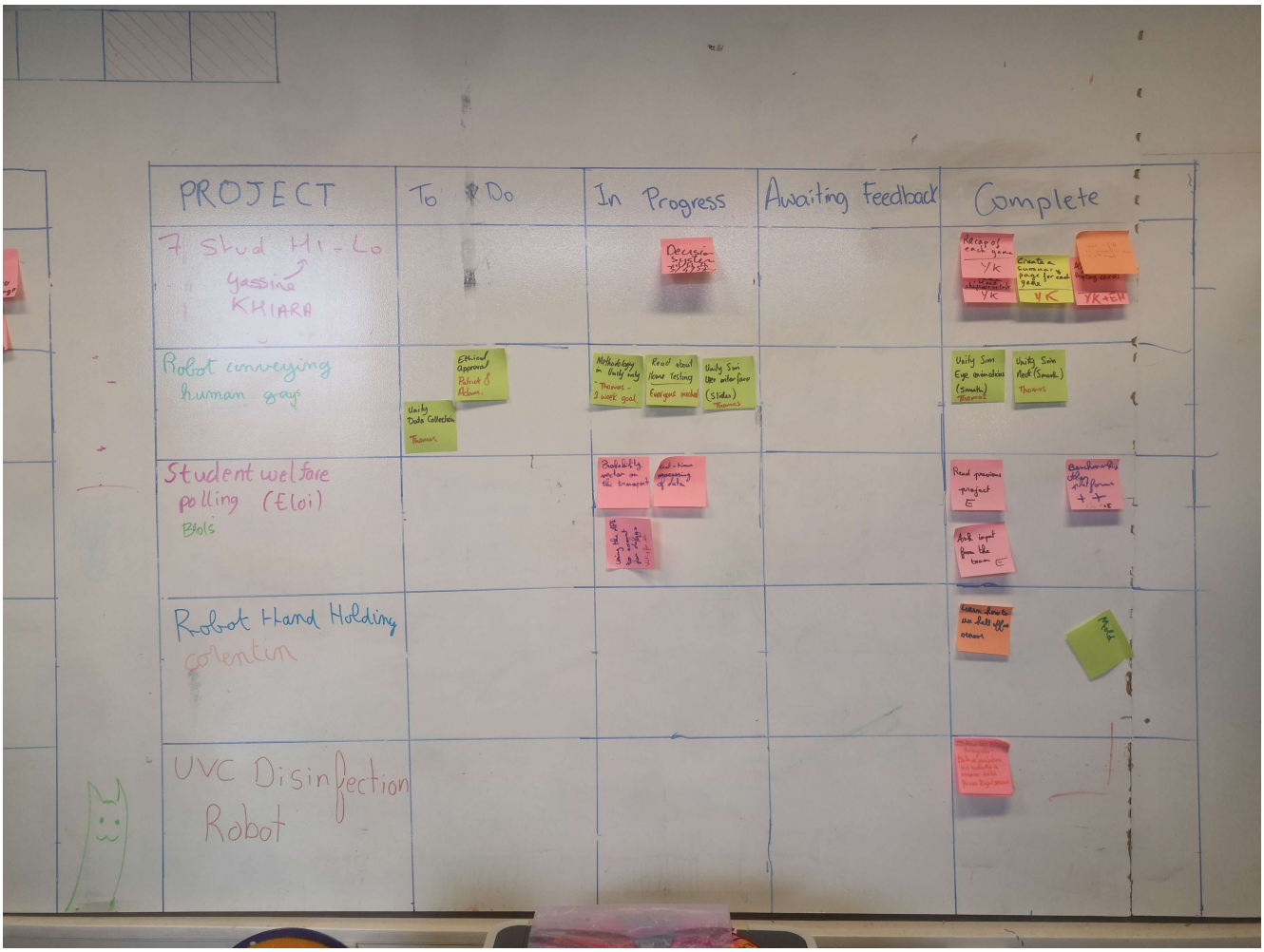


Figure 2.1: Photo of the Agile board

Given our shared workspace and collaborative environment, it was unanimously decided to adopt the Agile methodology. Our work process was effectively managed through the use of an Agile board as seen on the Figure 2.1. This board served as a dynamic tool, where tasks were meticulously documented along with their progress, and potential assistance requests were marked using post-it notes.

The heart of our daily routine consisted of morning scrum meetings. These were structured in a way that half of the interns attended one day and the other half the next day. During these meetings, each of us took turns reporting on our progress, outlining the remaining tasks, and addressing any challenges encountered. The scrum master, a role that rotated among a PhD candidate, Moyin Otubela, and our supervisors, Patrick Lynch or Kevin Kelly, facilitated these discussions. Their valuable input, opinions, and advice were integral to our progress. Moreover, significant questions related to specific projects were further discussed between the relevant intern and their respective tutor, either immediately after the scrum meeting or throughout the day.

Date	10/05/2023
Minute taker:	Yassine
Scrum Master	Kevin

PROJECT TEMPLATE:

Project:	Robot Gazing
Updates:	-Finished the slider prototype -Finished the design of the interface canvas
TO-DO	-Create a smooth transition when pressing the button -Theorizing on the submit button
Challenges	Smooth transition implies some error that need to be fixed
Comments	Making the UI translucent could be a good idea.

Figure 2.2: Example of a minute's section

In order to maintain detailed records of the meetings, one intern was designated as the minute taker during each scrum, tasked with recording notes within a shared Google Drive document 2.2.

Initially, our projects were organized in 2-week sprints. However, over time, we discovered that this timeframe often proved unrealistic due to uncertainties in predicting the exact workload required for each task.

					Monday	Tuesday	Wednesday	Thursday	Friday
10	KANBAN								
11	Tasks	Project	Status	Due Date	31/07/2023	01/08/2023	02/08/2023	03/08/2023	04/08/2023
12	Doxygen docum	Robot Gazing	In Progress	04/08/2023	6.00	3.00	1.00	2.00	
13	Timeline diagran	Robot Gazing	To Do	07/08/2023					
14	Writing report	Robot Gazing	To Do	21/08/2023				3.00	
15	Creating custom	Robot Gazing	In Progress	07/08/2023	1.00		3.00		
16	BUG	Robot Gazing	In Progress	10/08/2023		4.00	3.00	2.00	
17	Reducing lags b	Robot Gazing	To Do	14/08/2023					2.0

Figure 2.3: Virtual Agile board 2.0

The transition to a Google Sheets-based alternative for the Agile board, as depicted in Figure 2.3, occurred due to practical considerations. This virtual approach retained the functions of the physical board, listing ongoing and completed tasks, while also enabling us to track the time dedicated to each task. Additionally, this choice was also driven by environmental concerns within the RAIL, where environmental impact is taken into account, including reducing paper consumption.

Noteworthy are the series of meetings I held with my supervisor, Patrick Lynch. These discussions served as a forum to review project progress, reassess objectives, address needs, and exchange valuable insights.

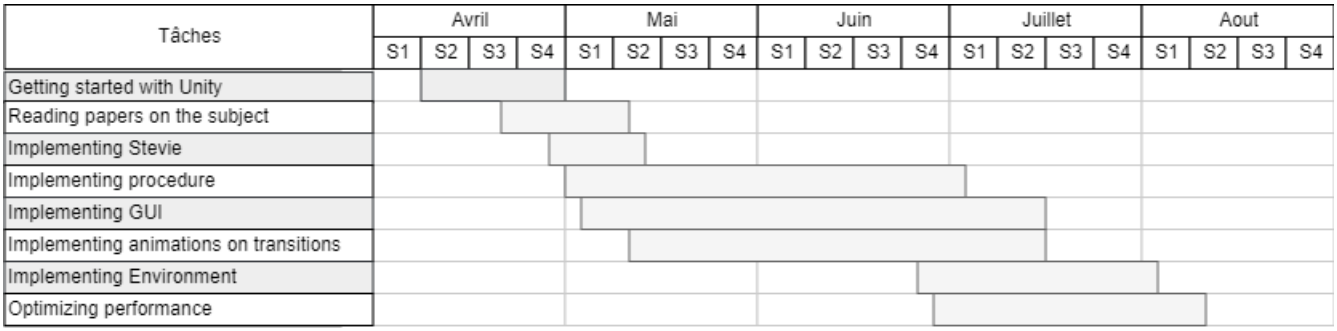
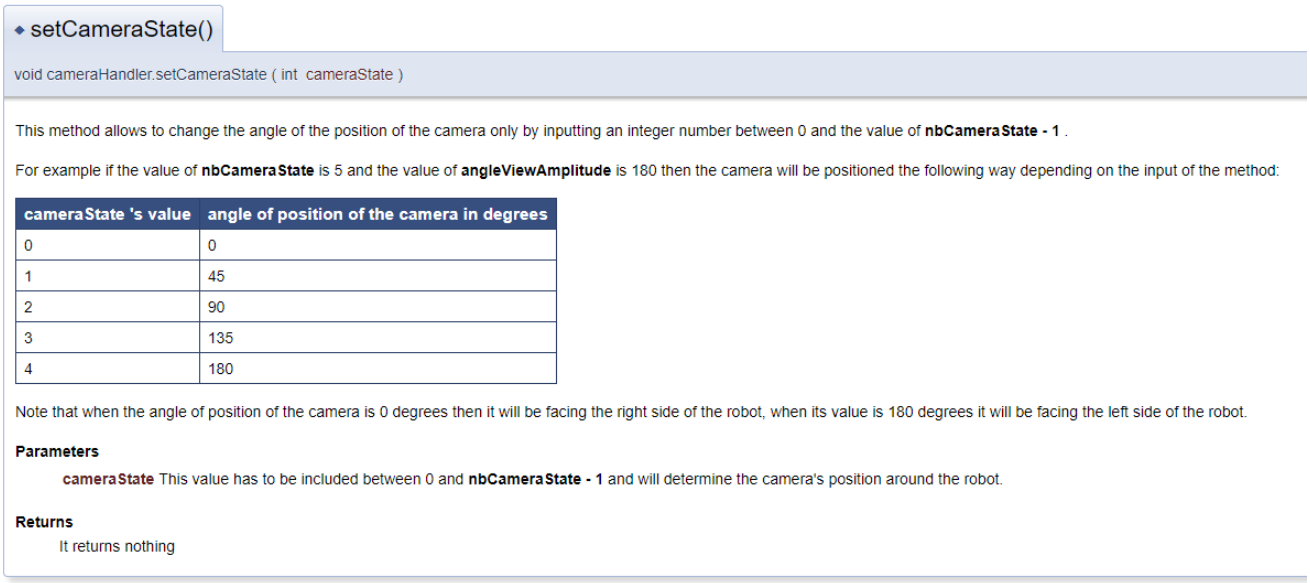


Figure 2.4: Gantt Chart

Having applied the Agile methodology, no predictive Gantt chart was created. However, the Gantt chart that was produced is the one depicted in Figure 2.4.

2.1.2 Documentation

Adaptability is a major constraint of the project; it is what will determine its value and utility in future RAIL projects. Measures have thus been taken into consideration, firstly to make the code more comprehensible.



◆ setCameraState()

void cameraHandler.setCameraState (int cameraState)

This method allows to change the angle of the position of the camera only by inputting an integer number between 0 and the value of **nbCameraState - 1** .

For example if the value of **nbCameraState** is 5 and the value of **angleViewAmplitude** is 180 then the camera will be positioned the following way depending on the input of the method:

cameraState 's value	angle of position of the camera in degrees
0	0
1	45
2	90
3	135
4	180

Note that when the angle of position of the camera is 0 degrees then it will be facing the right side of the robot, when its value is 180 degrees it will be facing the left side of the robot.

Parameters

cameraState This value has to be included between 0 and **nbCameraState - 1** and will determine the camera's position around the robot.

Returns

It returns nothing

Figure 2.5: Doxygen Unity's Camera's Documentation Example

Indeed, documentation created using Doxygen has been implemented as shown by the example in the Figure 2.5.

Doxygen is a powerful tool designed to automatically generate documentation from the source code of a software project. It works by parsing through the code files, extracting comments, and structuring them into a well-organized documentation format.

The process of documenting code with Doxygen involves adding specially formatted comments directly within the code. These comments include information such as the purpose of functions, their input parameters, return values, and any other relevant details. By consistently adding these comments, it can create a clear and informative documentation source that can be generated into various formats, such as HTML, PDF, or even XML.

Documenting the code using Doxygen is highly beneficial for several reasons. Firstly, it enhances code readability and maintainability by providing developers with comprehensive explanations of each component's purpose and usage. This is particularly useful when multiple team members are collaborating on a project or when revisiting code after some time has passed.

Furthermore, Doxygen-generated documentation serves as a valuable reference for anyone working on or using the codebase, including future developers who might join the project. It provides insights into the functionality of various elements, making it easier to understand and modify the code.
In the context of the simulation project, using Doxygen to document the C# code is crucial for ensur-

ing that the intricacies of the simulation's logic, behavior, and components are well-documented. By providing clear explanations and context, the documentation helps developers navigate the codebase more efficiently and make informed decisions.

It's not impossible that a future developer may find that certain things coded in my code are not coded in the right way at all, and they may decide to modify them. This documentation would also enable developers to modify my code in a safer manner as they will be aware of the impact of each line of code on others during their modifications.

In summary, my experience at the RAIL was enriched by the implementation of Agile methodology and greatly facilitated by the use of doxygen documentation. This provided a collaborative framework that ensured effective communication, adaptation, and continuous improvement within our dynamic projects.

2.2 Materials

2.2.1 Development tool

During the initial meeting with my supervisor, we engaged in a conversation concerning the selection of tools to employ in the development of this simulation project. My supervisor presented three distinct options, each combining the realms of 3D modeling and computer programming, as a means to accomplish this task:

- Blender
- Webots
- Unity

When considering the creation of the simulation, Unity emerges as a preferable choice over the alternatives of Webots and Blender. Several key factors underscore this choice, making Unity a compelling platform for the task at hand.

First and foremost, Unity boasts a user-friendly interface that facilitates the creation of intricate simulations without necessitating extensive programming expertise. This is of paramount importance, especially for projects involving a team with varying technical backgrounds. Unity's intuitive drag-and-drop functionality and visual scripting options enable efficient collaboration among team members with diverse skill sets.

Moreover, Unity offers a wide array of pre-built assets and resources that can be readily incorporated into the simulation. This expansive library encompasses 3D models, textures, animations, and more. Such resources significantly expedite the simulation development process, as they eliminate the need for painstakingly creating every component from scratch. This efficiency translates into saved time and increased productivity, essential factors for a successful project timeline.

Unity's robust community and extensive documentation are additional assets. The platform boasts a vast online community of developers and enthusiasts who actively share insights, advice, and solutions. This collaborative environment empowers project teams to troubleshoot challenges effectively and learn from the experiences of others.

While Webots and Blender also have their merits, Unity's advantages are compelling. Webots offers specialized robotics simulation capabilities, but it may necessitate a steeper learning curve due to its narrower focus. Blender excels in 3D modeling and animation, but lacks the seamless integration of interactive elements that Unity effortlessly provides.

In conclusion, the decision to utilize Unity for creating the simulation stems from its user-friendly interface, extensive asset library, compatibility, flexibility, strong community support, and comprehensive documentation. These factors collectively contribute to a smoother development process, increased efficiency, and enhanced collaboration.
Therefore, Unity has been chosen as the primary development tool.

2.2.2 Unity

Unity serves as the cornerstone of creating our simulation, bringing together design elements and programming functionalities seamlessly. The power of Unity lies in its concept of GameObjects, which are the building blocks of our virtual environment. These GameObjects represent distinct entities within the simulation, such as objects, characters, cameras, and more.
The fascinating part of GameObjects is their hierarchical structure, akin to a family tree. This hierarchy allows me to organize and nest GameObjects within one another, creating a logical relationship. A prime example of this hierarchy is Stevie, our robot. Stevie's head GameObject is nested within its body GameObject, which, in turn, is a part of the root GameObject representing the entire robot.

Through these, I've been able to create a virtual representation of our robot, Stevie. Each GameObject encapsulates various attributes and behaviors by attaching components. Components are like the essential modules that define how a GameObject behaves and interacts. For instance, I've assigned a MeshRenderer component to Stevie's head GameObject, which enables its visual rendering

Unity's strength becomes more evident when combining these GameObjects with scripts. These scripts act as the guiding intelligence behind the simulation. They are what make Stevie's movements smooth, lifelike, and responsive. By attaching scripts to GameObjects, I've achieved dynamic behaviors like turning the robot's head towards different directions. This direct connection between scripts and GameObjects facilitates a high level of control and interactivity.

Furthermore, Unity's GameManager has played a pivotal role in managing the simulation's overall mechanics. I've employed the GameManager to handle essential functions, such as keeping track of the robot's interactions and responses. For instance, the script managing the procedure of the experiment is assigned to the GameObject GameManager and executed by it. This allows me to manage the simulation's logic centrally, enhancing the cohesion and functionality of the simulation.

In summary, my utilization of Unity in the simulation development process revolves around creating and managing GameObjects, customizing their behavior through scripts, and leveraging Unity's GameManager for global control. These tools work together to bring Stevie to life within the simulation, ensuring a cohesive and immersive experience for users.

From the beginning of the project, even though I had some prior knowledge of Unity, I had to practice to regain a good grasp of the tool.

2.3 Procedure

2.3.1 Theory

As mentioned earlier, the way we conduct our experiments is really important because it shapes the whole purpose of our study.

Following the ideas of Eoghan O'Leary, the student who first explored how Stevie looks at things, it made sense for us to build on his work.

To make sure we were on the right track, a meeting was held at the RAIL, where the main people involved in the project got together. We talked about how we should set up our experiments in the simulation. This was a crucial step to make sure everything was going in the right direction.

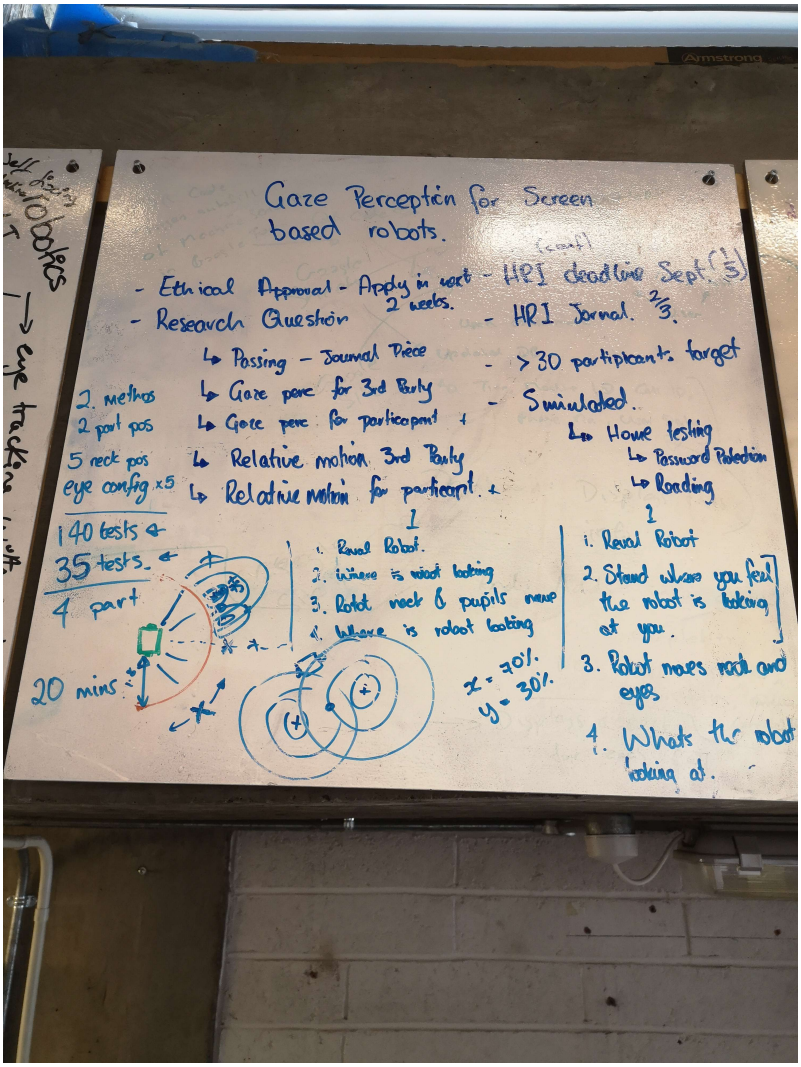


Figure 2.6: Simulation's goal for the Procedure

After the meeting, we had clear goals laid out on a board 2.6. Just like in the lab experiments, the tester's task was to pick an object in the space to indicate where they thought the robot was looking.

We wanted to explore how people perceive a robot's gaze direction differently in two scenarios. First, when they see the robot already looking in a certain direction, and second, when they can see the robot's eye and head movements as it transitions to that direction.

Varying this aspect and factoring it into future analyses would help us determine whether the smooth eye and head movement we plan to implement actually matters in how people interpret a robot's gaze direction.

Additionally, we considered another factor – the tester's position. To mimic a conversation-like context, we made a distinction between when the robot was looking at the tester, treating them as

a participant, and when the tester was a bystander in the conversation and not directly looked at by the robot.

Our hypothesis was that these two states – being part of the conversation and being an outsider – significantly influenced how the tester understood the robot's gaze. It was important for us to prove this and see how these factors played into our findings.

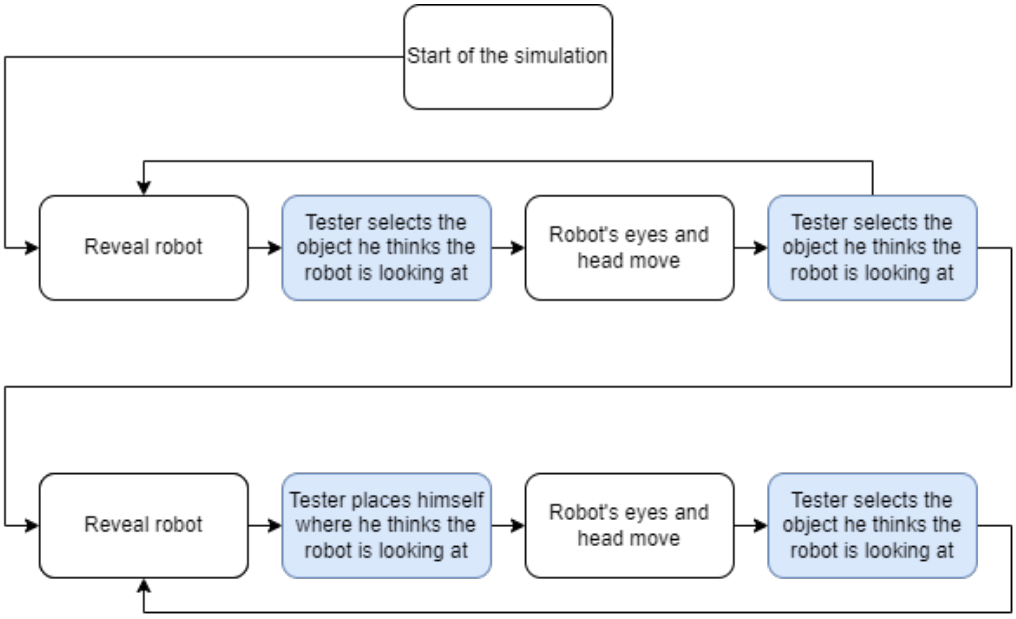


Figure 2.7: Procedure diagram of the simulation

The procedure, as depicted in Figure 2.7, which emerged from this meeting, can be broken down into two parts.

The first part involves unveiling the robot, asking the human where they believe the robot is looking, then moving the robot's eyes and head, followed by asking the human again where the robot is looking.
At the moment of the robot's revelation, we test how it is interpreted as a bystander.
After the robot's movement, we test the impact of displaying the robot's eye and head movements to the human, a change from the previous scenario. This allows us to assess the significance of movement in gaze interpretation.

In the second part of the procedure, a slight modification is made to evaluate other factors. After the robot is revealed, the human is asked to move to where they think the robot is looking. This positions the human as a participant in a simulated conversation with the robot. Subsequently, the robot's eyes and head are moved, and finally, the human provides their opinion on the observed

object, having witnessed the eye and head movement. This step examines gaze interpretation from the perspective of someone directly observed by the robot, rather than as an outsider.

It was essential to introduce randomness into the robot's eye and head movements at certain stages of the procedure.

A technique exists that can make a combination of numbers random while staying within known maximum values. This technique was implemented to create two separate lists of distinct combinations. These lists ensured that the positions for the head and eyes would be different for each iteration of every part of the procedure.
In this way, the technique ensured a variety of positions for both the head and eyes, enhancing the overall realism and diversity of the robot's movements throughout the procedure.

This refined procedure aims to investigate an additional factor's impact on gaze interpretation, extends the continuity of prior research on this topic, and can be effectively executed within a 3D simulation. This comprehensive procedure allows us to delve deeper into our investigation while building on previous work and providing a meaningful context for our findings.

2.3.2 Adaptability Challenge

However, the adaptability constraint urged me to develop a rather specific implementation for the procedure. It appeared clear to me that in the future, even in the near term, the procedure might need slight modifications, such as swapping the two procedure parts, for instance.
That's why I opted to design the procedure in a way that allows for such changes. Firstly, the sequence of events within the procedure is coded in the script linked to the GameManager, as it serves as a central component of the simulation. It's like crafting a flexible puzzle; this approach is similar to creating a kind of non-deterministic finite-state machine. The states and transitions are described within the "Update" function of the GameManager script. This "Update" function is called repeatedly with each new frame of the simulation, making it an essential part of the simulation's functioning.

```
//Explanation of the methodology screen
else if (state==2)
{
    //Code executed when the token enters this state
    if (stateEntry)
    {
        canvasHandler.setButtonsEnabled(true,true);
        stateEntry = false;
    }
    //Exiting the state to the previous page
    else if (canvasHandler.leftClicked)
    {
        state=1;
        transitionState = true;
        canvasHandler.textsTranslationLeft();
        stateEntry = true;
    }
    //Exiting the state to the next page
    else if (canvasHandler.rightClicked)
    {
        state=3;
        transitionState = true;
        canvasHandler.mainCanvasTranslationOut();
        headHandler.smoothHeadMovement(order0[testIndex*2]/5);
        eyesHandler.smoothEyesMovement(order0[testIndex*2]%5);
        stateEntry = true;
    }
}
```

Figure 2.8: Example of State's implementation

The state machine in question comprises several states, an example of which is depicted in Figure 2.8.
Each time the simulation enters a new state, the entry instructions for that state are executed. These instructions can, for instance, activate the ability to click a button on the user interface.

With each frame, the instructions associated with the current state of the procedure being executed are carried out.

Generally, there are no looping instructions within our simulation.
Every exit from a state is conditioned by a more or less straightforward requirement, like pressing a button through the user interface, and triggers instructions.
Whenever the GameManager exits a state, it temporarily loops through a transitional state. This transitional state allows smooth animations to play out before enabling user interaction with the simulation, thereby preventing conflicts and potential bugs.

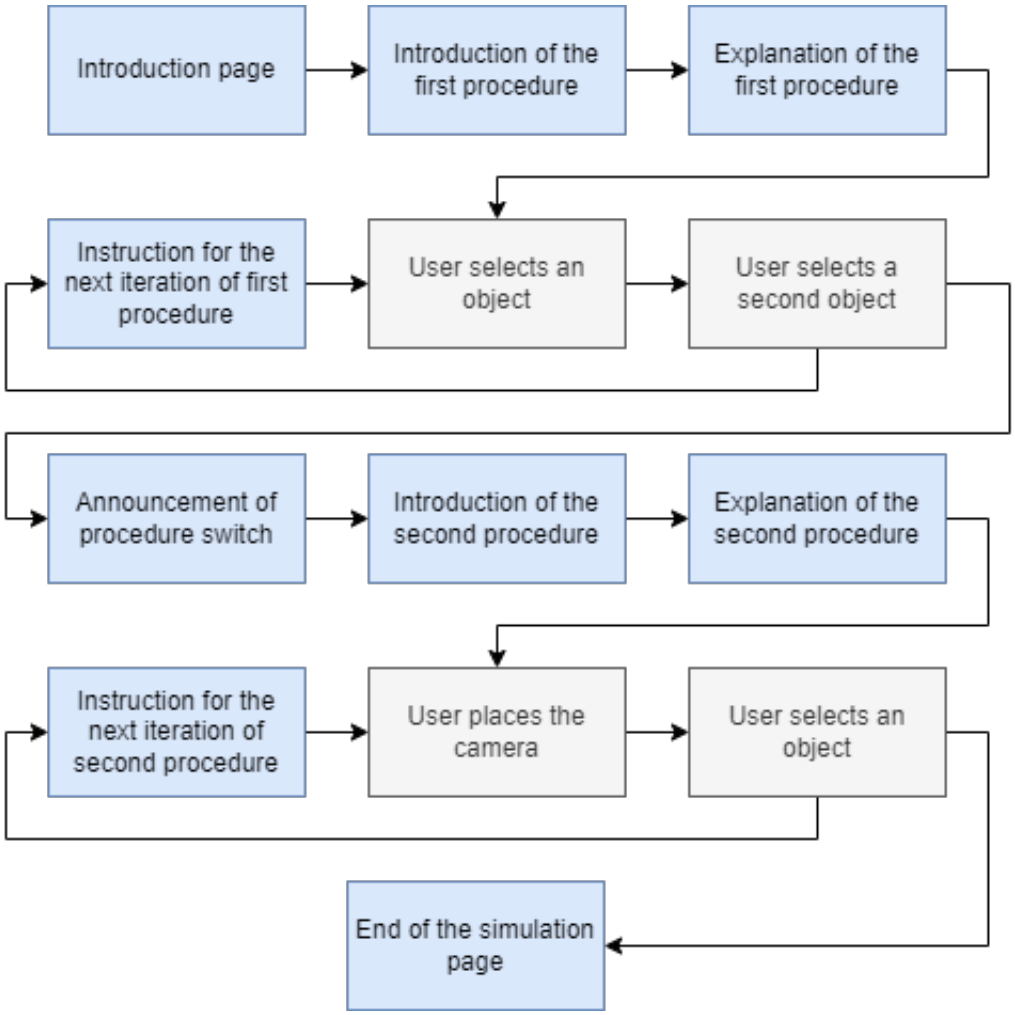


Figure 2.9: State Machine Diagram

The resulting state machine diagram is shown in the Figure 2.9.

Thanks to this implementation, we can easily define entry instructions for states, looping instructions, conditions, and state exit instructions. This approach also allows us to repeat the same sequences of states multiple times by revisiting previous states.
Consequently, we ensure that future developers can freely alter the experimentation procedure without the need to overhaul the entire codebase.

2.4 Stevie Robot

2.4.1 Static Part of the Implementation

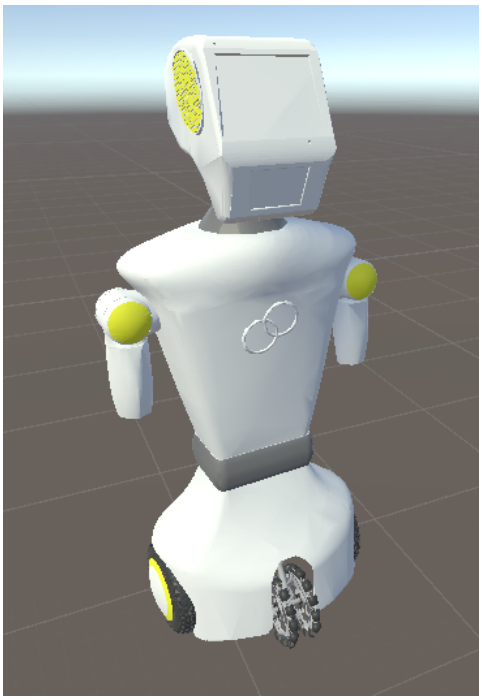


Figure 2.10: Stevie's eyes's image

Implementing the Stevie robot is also a significant part of the work, as we were assessing Stevie's effectiveness in our situation. Recreating Stevie's model from scratch in Unity would have been hard and time-consuming. Thankfully, a Ph.D. student from the lab had a file of Stevie's model in the form of a package that contained the Stevie model, as shown in Figure 2.10. However, a downside was that a crucial part was absent in the model: the eyes.

Figure 2.11: Stevie's eyes's image

It was while exploring the work conducted during the study where the experiments were directly

carried out in the lab that I came across an eye model for the Stevie robot, as shown in Figure 2.11, in "png" image format.

2.4.2 Dynamic Part of the Implementation

Similar to the in-person experiments, it was necessary to move the pupils within the eyes. Using image processing tools, the image was divided into two parts: one for the pupils and the other for the whites of the eyes. To make them easy to control within the Unity simulation, these two images were treated as separate textures applied to Stevie's face.

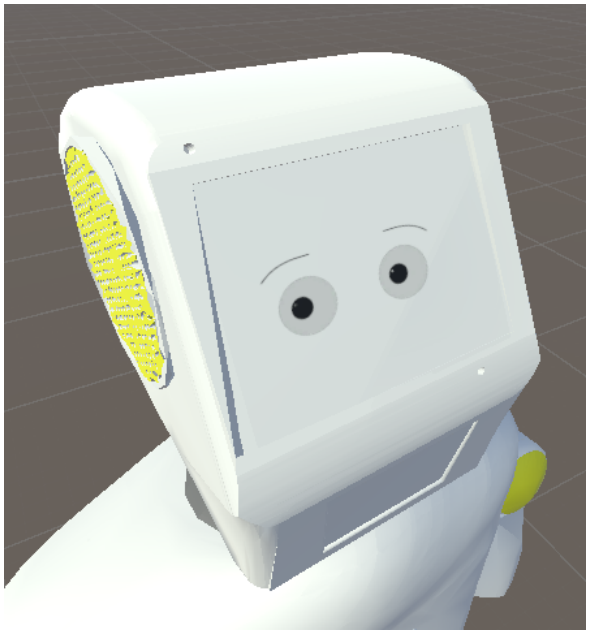


Figure 2.12: Stevie's eyes in the simulation

Once considered as textures, it became possible to translate them from right to left on Stevie's face, as shown in Figure 2.12, and therefore move the eyes and pupils to positions similar to those obtained in reality with the actual robot. However, these relocations were merely positional shifts. To infuse the simulation with a sense of motion, it was imperative to transform these adjustments into fluid, continuous movements.

To achieve fluid eye and pupil movements in the simulation, I employed the concept of Delta Time (DeltaTime) in Unity. DeltaTime measures the time elapsed between two consecutive frames and is used to normalize movement calculations, ensuring consistent behavior across different frame rates. By multiplying the desired movement speed by DeltaTime, I was able to achieve smooth and consistent eye and pupil movements, regardless of the frame rate. This technique enhanced the realism and immersion of the simulation, allowing for a more accurate representation of Stevie's gaze direction dynamics.

As for the movements of Stevie's head, the same technique was applied to make them smoother, using DeltaTime.

2.4.3 Compatibility Challenge

As the environment was just added to the simulation, it started experiencing lag. The number of frames per second significantly dropped during simulation execution, making it practically unusable. In my quest to identify the source of these issues, I turned to Unity's rendering statistics window for performance analysis. This window provides real-time rendering information about the application, helping to diagnose the problem.

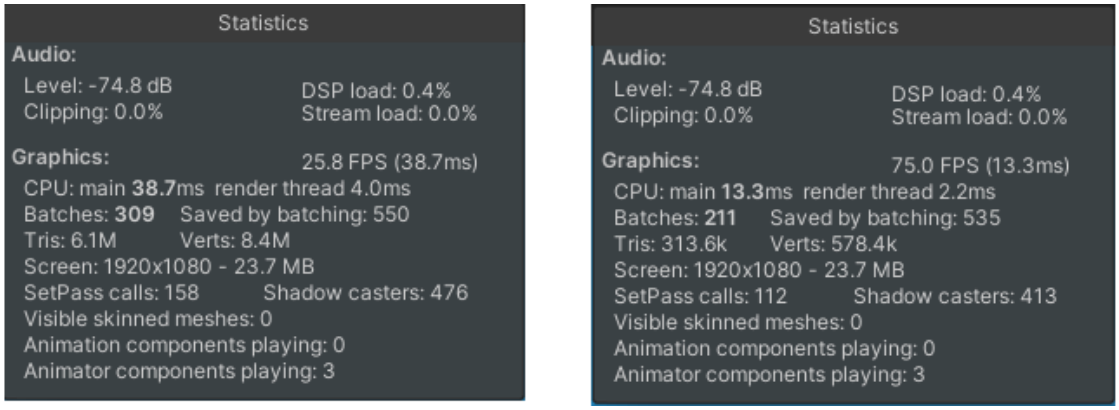


Figure 2.13: Stats Window Comparison With and Without Stevie

During this analysis, an interesting observation came to light, as illustrated in Figure 2.13. The striking disparity in the "Tris" and "Verts" values was evident, particularly with significantly more in the version containing Stevie's model compared to the version without it.

"Tris" and "Verts" values respectively describe the number of triangles and edges that Unity calculates per frame. Since the model accounted for over 90% of the total triangles and edges processed, it became crucial to address this issue.
In reality, the model had been designed without considering its potential use in a simulation. Consequently, certain parts of the robot were extremely resource-intensive due to suboptimal creation methods.

For instance, Stevie's ears and wheels were needlessly detailed for the intended usage.
Thus, the decision was made to simplify the model's mesh to reduce its resource demands. To achieve this, a technique known as Mesh Simplifying was applied. This computational method focuses on optimizing 3D models by decreasing their complexity while maintaining an acceptable level of visual quality. This adjustment aimed to enhance simulation performance by reducing the computational load caused by the highly detailed model of Stevie.

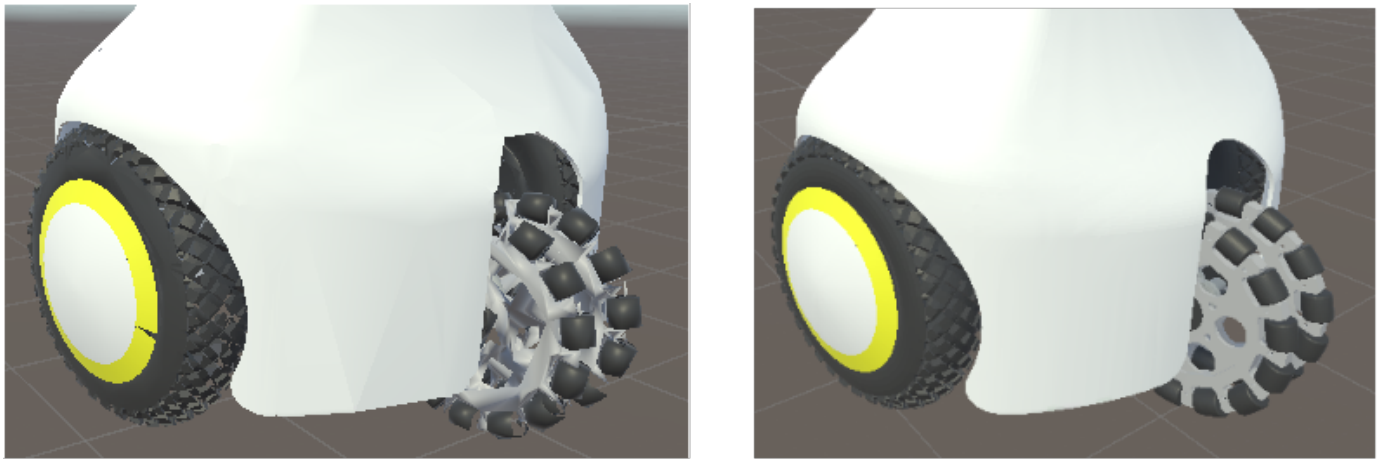


Figure 2.14: Wheels Comparison With and Without Mesh Simplifier

By applying a mesh simplifier to Stevie's model, as seen on the Figure 2.14, we can intelligently reduce the number of polygons or vertices that make up the model's geometry. This reduction translates to a decrease in computational load, making it easier for the simulation to render and animate the model swiftly. The key lies in removing unnecessary intricacies that might not be perceivable in the simulation context, without compromising the overall appearance.

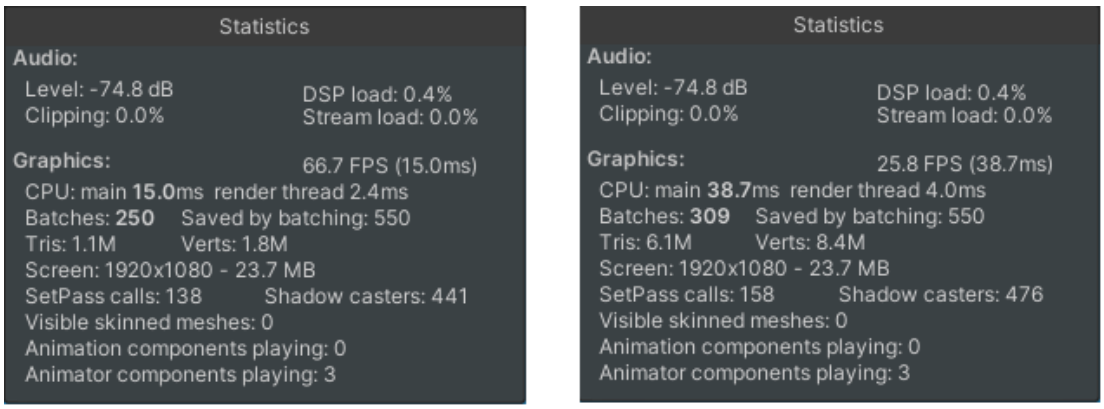


Figure 2.15: Stats Window Comparison With and Without Mesh Simplifier

Incorporating a mesh simplifier into our development process provides tangible benefits. It optimizes the rendering process, as seen on the Figure 2.15, making it possible to maintain high frame rates and responsiveness even on hardware with limited capabilities without noticing any decrease in visual quality. This optimization is especially noticeable when rendering scenes with multiple objects and complex interactions, ensuring a seamless and engaging user experience.

2.5 Camera's Adaptability Challenge

In our simulation, the camera served as the tester's viewpoint. In Unity, defining a camera and its initial position and orientation in space is not a complex task.
However, potential future modifications to certain elements of the simulation could require adjusting these settings.
Among these elements is the presence of a GUI fixed to the camera's viewpoint, which might potentially obscure an important scene element, for instance. For reasons of robot visibility, there might also be a need to change the camera's position and orientation.
Hence, a section in the code has been established to modify the camera's initial parameters. This allows for flexibility in case adjustments are needed due to changes in the simulation's requirements or visual considerations.

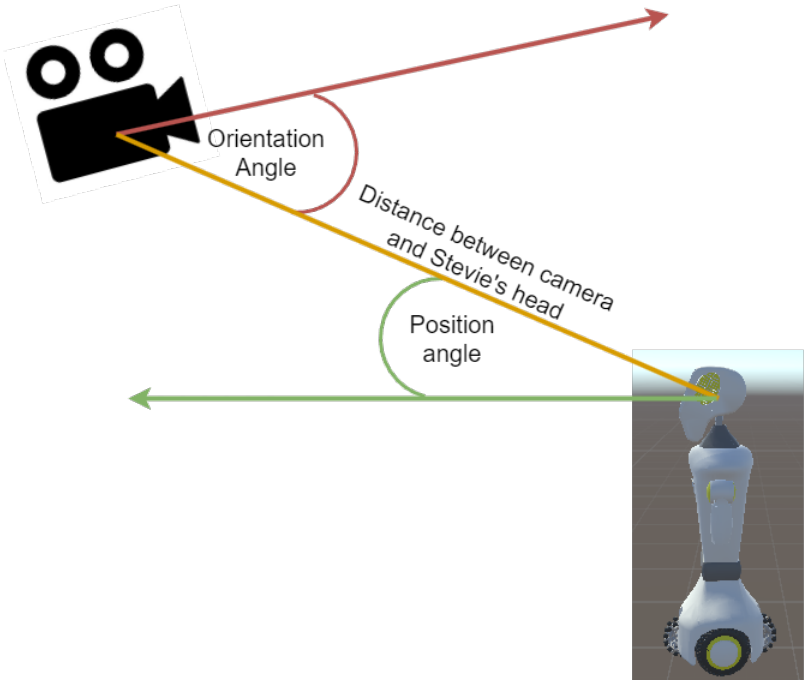


Figure 2.16: Diagram representing camera's initial position parameter

It has been made possible to modify three parameters, as shown in Figure 2.16:

- The camera's positional angle, which corresponds to the angle formed between the line connecting the robot's head and the camera, and the line formed by the direction of the robot's gaze.
- The camera's orientation angle, which is the angle formed between the line connecting the robot's head and the camera, and the line formed by the direction of the camera's orientation.
- The distance between the camera and the robot's head.

The addition of this capability to adjust such parameters has addressed the challenge of flexibility in the camera's implementation within the simulation, and consequently, the tester's viewpoint. This enhancement ensures that the simulation can adapt to potential changes and optimize the tester's experience.

2.6 Environment

2.6.1 Usability

The environment was considered a crucial part of implementing the usability aspect of the experimentation. The aim was to provide testers with a better sense of perspective and depth, specifically regarding the distance between the robot and surrounding objects.

To facilitate this, a decision was made to search for "Prefab" environment assets directly in the "Unity Asset Store." A selection of "assets", ranging from realistic to highly simplified in terms of detail, was made. What these assets had in common, beneficial for usability, was the inclusion of everyday objects that allowed testers to gauge scale during the experiment, aiding in considering distances between simulation elements.

Further choices regarding environment assets depended on addressing other challenges.

2.6.2 Compatibility

As mentioned earlier in the section discussing the implementation of Stevie and the compatibility issue due to insufficient performance, environments also triggered a significant drop in simulation performance. Upon opening the Unity editor's stats window to access rendering performance information, I observed that, much like with the robot, less complex environments - those modeled with fewer polygons - exhibited better performance than others. However, the difference wasn't immediately apparent, yet the simulation's frame rate still decreased noticeably.

Another performance-diminishing factor was subsequently identified, namely shadow quality. Shadows are a crucial visual element in rendering scenes, but they can be resource-intensive to compute. Unity provides various settings to adjust shadow quality, allowing developers to balance visual fidelity with performance impact.

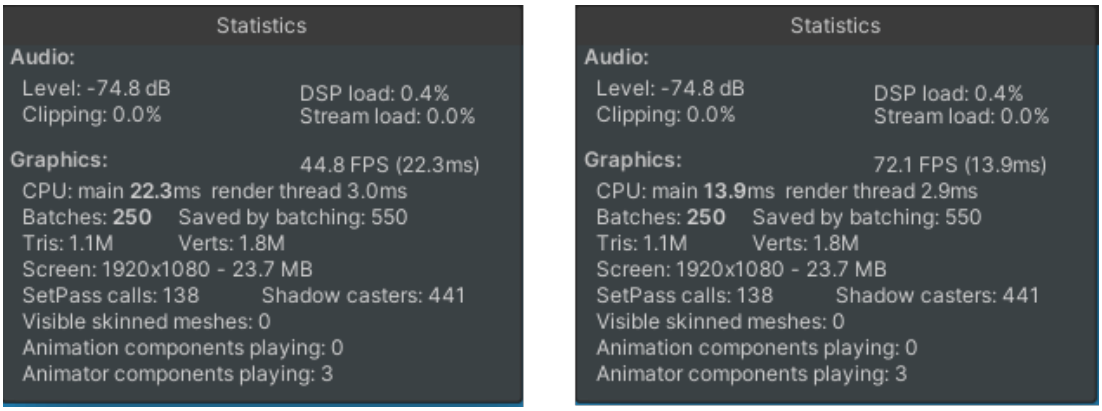


Figure 2.17: Stats Window Comparison With and Without Very High Quality Shadow

In our simulation, I noticed that higher shadow quality settings, which produce more realistic and detailed shadows, placed a substantial burden on performance as shown in the Figure 2.17. This was particularly noticeable in scenes with complex models and environments. By reducing the shadow quality, I was able to significantly alleviate the strain on system resources and improve the overall simulation performance without lowering the quality of the simulation.

The parameters that were changed are the shadow type and the resolution.

Soft shadows are achieved through complex calculations that simulate the natural scattering of light. While they offer a more realistic appearance, they are computationally intensive and can slow down the simulation. On the other hand, hard shadows provide a simpler, sharper shadow effect that demands fewer resources, thus improving overall performance.

Similarly, using low resolution shadows instead of very high resolution ones reduces the computational load on the system. High resolution shadows involve rendering detailed shadow maps, which can consume significant processing power. Switching to low resolution shadows sacrifices some visual fidelity but results in smoother performance, making the simulation more responsive.

The interaction of these factors - model complexity, environment intricacy, and shadow quality - emphasized the need to strike a balance between visual fidelity and performance in order to create a smooth and responsive user experience within the simulation.

I had also attempted to pregenerate lighting and shadows to ensure that there would be no shadow calculations during the simulation. However, I encountered a technical issue during the preprocessing of global illumination. The process became stuck at a certain stage, preventing the precalculation of shadows.

After researching online forums, an interesting response[1] was found.

The forum post raises an important point about the potential challenges associated with pre-calculating lighting solutions in Unity environments. The forum user shared a workaround to address issues with Unity's GI cache and the "Auto Generate" option, which can lead to build process disruptions. Their solution involves manually triggering the "Calculate Lighting" function when different light calculations are needed, allowing Unity to generate and store the lighting data alongside the scene.

While this solution might offer immediate relief for the specific issues faced by the forum user, it also highlights the potential drawbacks of relying solely on pre-calculated lighting. Manually initiating the lighting calculations and managing the generated data introduces an additional layer of complexity to the development process. Developers need to remember to trigger the calculations and maintain the consistency of the data across different environments.

Moreover, the forum user's suggestion to include the pre-calculated lighting data in version control systems (VCS) underscores the challenge of maintaining and sharing this data among developers working on diverse machines and platforms. This approach could lead to potential conflicts or issues related to version control, especially in collaborative settings.

Ultimately, the workaround shared in the forum post points to the intricacies and potential limitations of pre-processing lighting data. While it can provide short-term benefits, it may not be a robust solution for long-term projects. The need for manual intervention, data management, and compatibility concerns may outweigh the performance gains, particularly when considering the principle of adaptability and ease of collaboration for future developers.

2.7 Graphical User Interface

Similar to the environment, the graphical user interface (GUI) was a crucial component of the simulation's usability. It is through the GUI that testers interact with the simulation. If the GUI isn't engaging and immersive enough, testers might find the simulation dull and exit prematurely.

Unity provides a powerful UI system through its "Canvas" component, allowing developers to create dynamic and interactive graphical interfaces.

A "Canvas" in Unity is a virtual surface that holds UI elements like buttons, text, images, and more. It acts as a screen space overlay that can be positioned within the game world or as an overlay on top of it. The Canvas handles the layout, rendering order, and interaction of its UI elements. It also comes with its own rendering camera, ensuring that the UI remains crisp and unaffected by the main camera's settings.

The hierarchy of the UI elements starts with the main camera in the scene, to which a Canvas component is attached. This Canvas acts as the root of the UI hierarchy, providing a base layer for all UI elements to be displayed. Within this Canvas, there are three separate child Canvas elements, each with a specific purpose.

The first child Canvas is designed to enable the user to choose which object they believe the robot is looking at. This selection can be crucial for evaluating the robot's gaze accuracy.

The second child Canvas appears when the user needs to position themselves where the robot is looking using a slider.

The third and final child Canvas is dedicated to displaying instructions to the user. It provides information and guidance on how to interact with the simulation effectively.

During my work, I had to create mock-up designs for the user's response submission graphical interface.

After receiving multiple suggestions from my colleagues, I proceeded to develop and implement high-quality, functional designs that share similarities
.

2.7.1 Object Selection Canvas

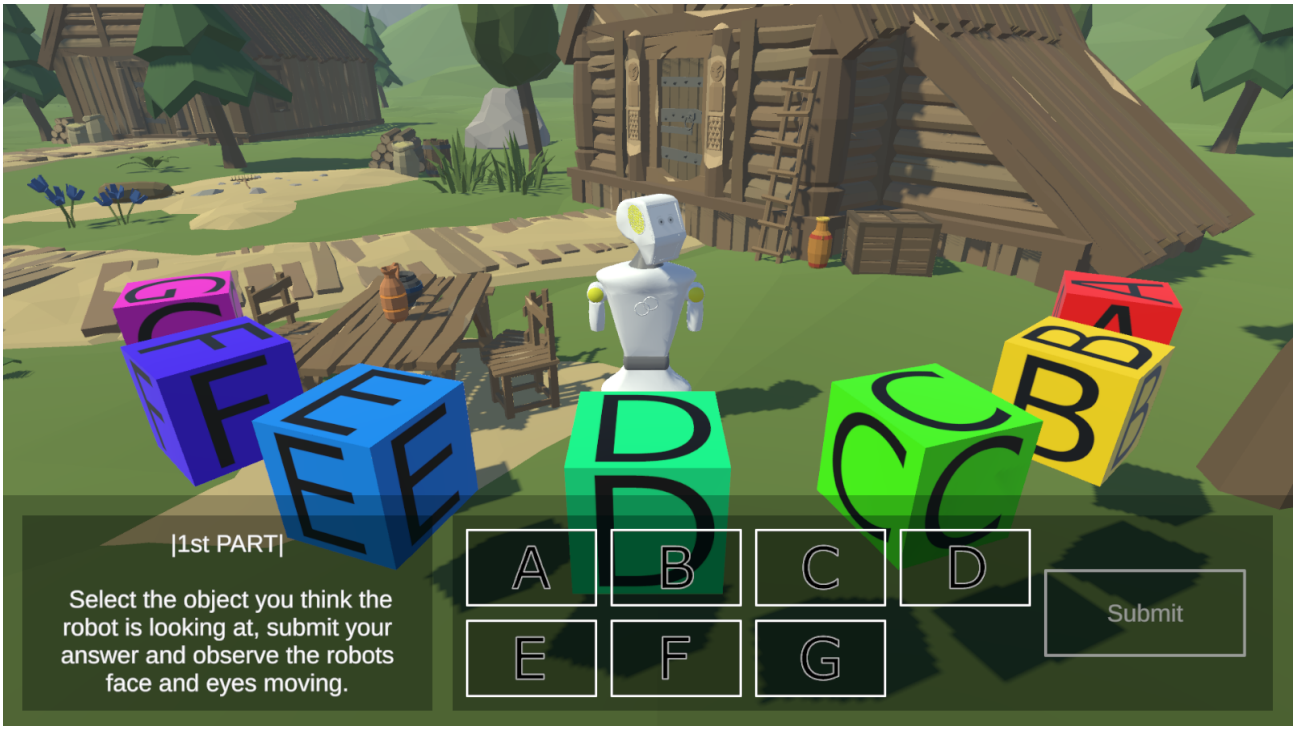


Figure 2.18: Graphical User Interface for Object Selection

This Canvas, shown in Figure 2.18, is designed to display a list of elements marked as "Toggle." Each "Toggle" button represents an object that the robot might be observing.

To confirm a choice on this interface, you simply need to click one of the toggle buttons and then submit your response by pressing the "Submit" button. The advantage of using "Toggle" buttons is their selectability.
When the mouse hovers over them, they highlight in blue, and when selected, they remain blue. Clicking on an already selected button deselects it.

Figure 2.19: Submit Button Being Pressed

The submission button, responsible for finalizing and submitting user responses, is intentionally presented in a grayed-out state upon loading the interface. This design choice makes the button

unclickable and effectively communicates to the user that the response cannot be submitted until certain conditions are met.

To activate the submission button, the user is required to select a specific "Toggle" button. Once this selection is made, the submission button becomes clickable. However, an additional layer of interaction is introduced to prevent hasty submissions. Users are instructed to maintain the click on the submission button for a predefined period. This duration is designed to ensure that users consciously intend to submit their response, minimizing the chances of inadvertent submissions.

As illustrated in Figure 2.19, the user interface provides a visual representation of the process. Users are guided to hold down the click on the submission button, allowing the system to validate and process the response during this time. This deliberate delay adds an extra step that acts as a safeguard against unintended submissions, aligning with the user-centered approach of the interface design.

2.7.2 Location Selection Canvas

Figure 2.20: Graphical User Interface for Location Selection

This canvas empowers users to establish where they believe the robot is looking.
The slider mechanism attached to the canvas is instrumental in achieving this functionality. As users

slide the control along the accompanying bar, the camera undergoes a rotation spanning 180 degrees around the robot. This innovative approach ensures a comprehensive view adjustment, enabling users to intuitively explore different perspectives.
The slider's design prioritizes simplicity and efficiency. Its straightforward operation grants testers the ability to swiftly manipulate the camera while maintaining precision. The smooth incorporation of movement control exemplifies a user-centric design approach, fostering both speed and accuracy in camera adjustments.

A notable similarity between the two interfaces is their translucency, a design choice that minimizes obstruction to the user's field of vision. This transparent design approach prioritizes usability, allowing users to seamlessly engage with the content while interacting with the interface elements.

Transitioning between these two canvases is a captivating experience. An elegant fade-in and fade-out animation occurs during both entry and exit, effectively retaining the tester's attention throughout the interaction. This animation strategy not only enhances visual appeal but also contributes to the overall cohesion of the interface.

2.7.3 Instruction Canvas

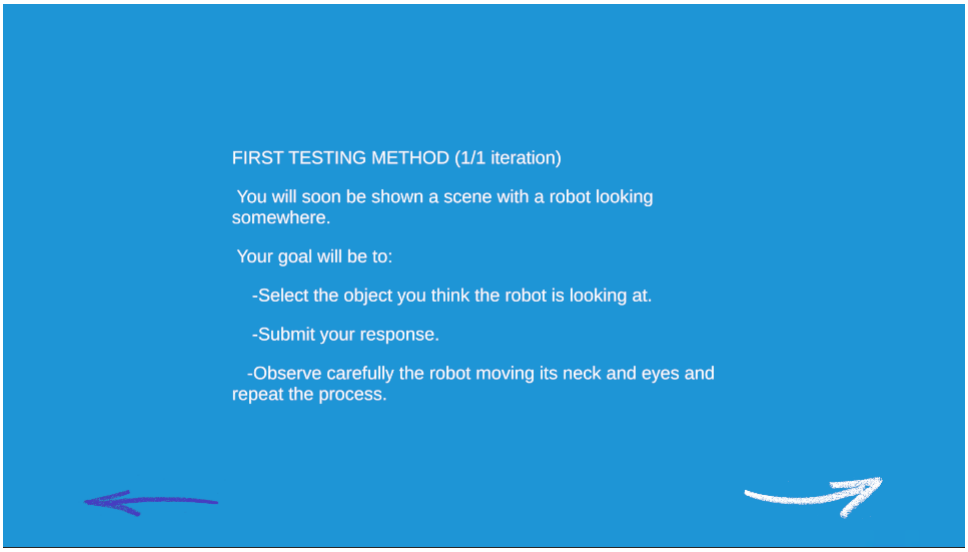


Figure 2.21: Graphical User Interface for Instructon Display

After conducting research, I came across an article [3] that highlighted the color blue as representing calmness and relaxation. That's why this color was selected as the background color for Canvas 2.21, aiming to counteract potential reading fatigue for testers.

Interactive arrow buttons were incorporated, designed as navigational aids. By default, they appear grayed out but lighten when the mouse hovers over them. Clicking on either arrow allows users to progress to the next page or return to the previous one.

In cases where a user lands on an instruction page directly from a tester response canvas, meaning no preceding instruction page exists, the left arrow for going back is absent.

Figure 2.22: Animation of Exit of the Instruction Canvas

The design of this canvas and its entry/exit transitions prioritizes smoothness. To achieve this effect, the instruction canvas smoothly translates between angles during both entry and exit. Additionally, an undulating pattern is introduced to cover or uncover the simulation, reducing abruptness during transitions, as illustrated in Figure 2.22.

It's essential to note that the text shown in the images does not reflect the final version of the texts to be used in the simulation. The step involving text selection and composition has not yet been reached, and these texts serve as placeholders for illustration purposes.

The integration of thoughtfully designed graphical interfaces has significantly improved the project's user-friendliness. The buttons and controls work smoothly, and the way elements change on the screen is pleasing. The use of colors and transparent elements also contributes to making the interface easier to use. This enhances user comfort during interaction and ultimately contributes to

the project's success by fostering increased user satisfaction.

In this development phase, I meticulously addressed the various challenges that arose during the creation of the simulation environment, robot behavior, and user interface. The adaptability constraint was at the forefront of my decisions, leading me to craft solutions that not only met the immediate requirements but also allowed for future adjustments with ease. From refining the robot's movements and gaze to devising a flexible event-driven procedure, every step was carefully taken to ensure a seamless user experience.
The integration of Unity's versatile features, such as the Canvas component for the graphical user interface and the application of various optimization techniques, was pivotal in achieving the project's objectives. The implementation of mesh simplification, dynamic shadow adjustments underscored my commitment to optimizing performance without compromising quality.

It's now time to step back and reflect on the work that has been done. It will be necessary to assess the relevance of the choices that have been made and potentially suggest avenues for improvement.

3 Results and Discussion

3.1 Outcome Achieved

In the end, the final product allows for simulating the experiment on a local computer.

When the application is started, blue instruction pages are shown. They effectively pass important information to testers by limiting distractions through their color and interactive features. If a user misses an instruction, they can go back a page to read it again.

Next, the tester is given the choice to select the object they think the robot is looking at. The interface for making this choice doesn't cause any vision problems as it's see-through. It's also really easy to use and understand. For instance, if no object is picked using the buttons, the answer submission button can't be pressed. This ensures things are clear and straightforward.

After the first part of the process is done, new instruction pages appear. These pages are meant to explain how the second part of the process will happen. This helps guide the tester through the next steps.

During the second phase, the second tester response submission interface comes into play. Here, the tester has the opportunity to position themselves where they believe the robot is looking, utilizing a slider control.

This GUI, built upon the same principles as the preceding one, provides the tester with a rapid and accurate means of selecting their response by using a slider.

The focal point was placed on ensuring usability constraints were met, evident in the fact that every element of the simulation was meticulously crafted to uphold this standard.

In terms of compatibility, the end result is a simulation that exhibits extremely little signs of frame rate drop, even on a laptop without a dedicated graphics card.

Several measures have been undertaken to ensure adaptability, particularly concerning the implementation approach of the procedure, which can be modified at any point. Considerations for

potential future environments have also been outlined in the report, aiming to curtail performance declines within the simulation.
Notably, the text content within designated areas is directly modifiable in the code through function calls.

In this manner, the simulation has been effectively implemented, mindful of the challenges it faced in terms of usability, adaptability, and compatibility.

3.2 Limitations and Future Recommendations

3.2.1 Web-Based Simulation Integration

Despite the fact that the simulation has been implemented, it hasn't been deployed online. However, the choice of the Unity tool greatly facilitates future work in this regard.
WebGL (Weg Graphic Library) is a JavaScript API that enables rendering 3D and 2D graphics directly in web browsers, without requiring any plugins or external software installations. It is closely tied to Unity through Unity's ability to export projects to WebGL format. Unity, a popular game development engine, provides the capability to create interactive 3D applications and games. When a Unity project is exported to WebGL, it essentially converts the project's assets, scenes, and logic into a format that can be executed within a web browser using WebGL technology.

This export process allows Unity developers to take advantage of WebGL's capabilities to display and interact with the 3D content they have created. Users can access and experience Unity-based applications and simulations directly within web browsers, enabling broader accessibility across various devices and platforms. This linkage between Unity and WebGL facilitates the distribution and engagement of interactive content by eliminating the need for users to install specific software or plugins.

In essence, WebGL serves as the technology that enables Unity projects to be executed and experienced within web browsers, extending the reach and accessibility of Unity-created content beyond traditional software installations.

3.2.2 Adaptability Limitations

In the journey of creating a simulation from scratch, I initially envisioned a high degree of adaptability, which, in hindsight, faced significant hurdles. The potential advantages for the RAIL were immense - the ability to effortlessly adjust variables like the number of responses available to testers, creating a responsive interface adaptable to diverse needs, and dynamically changing the number of objects around the Robot. However, the reality of the situation unfolded differently.

Having coded the simulation in a certain manner, making adjustments to these parameters now requires recoding a substantial portion of the application. Somewhere in the development process, I inadvertently lost sight of the very flexibility I had initially incorporated, and inadvertently restricted the simulation's role to a less flexible one. This realization emerged as a turning point where adaptability faced constraints.

In essence, the current structure of the simulation makes minor refinements possible, but substantial changes are constrained by the rigid foundations upon which it was built. While many components can be repurposed for future online experiments in Human-Robot interactions, the simulation itself lacks the inherent adaptability needed for a range of different use cases it was originally intended for. This realization stems from underestimating the paramount importance of keeping the design of the simulation ultra-adaptable as it was during the first month of the project. This miscalculation led me to embrace a solution that ultimately lacks the sought-after flexibility.

In conclusion, while the simulation's capacity for adaptability offers the potential for incremental enhancements within its original scope, it faces limitations when it comes to transforming its fundamental structure.

3.2.3 Environment Implementation Choices

Reflecting on my choices throughout this project, another decision that I regret involves implementing an environment from the Unity Asset Store. This choice inadvertently led to performance optimization challenges, driven solely by the fact that pre-designed environments often come with unusually high resource demands.

A more prudent approach to achieving a rich and immersive environment that allows testers to feel immersed in the scene and gauge the spatial relationships between simulation elements would have been to create it from scratch. This could have been achieved by piecing together smaller environment components sourced from Unity's Asset Store. For instance, crafting a simple room with six walls and furnishing it with objects like a table and various items would likely have sidestepped performance issues, meeting most of the environment's envisioned requirements.

Moreover, a significant amount of time was invested in manipulating these sprawling environments, which unfortunately didn't yield proportional returns in terms of usability and performance. Reimagining the environment creation process and opting for a more customized, modular, and efficient construction approach would have been a wiser route to take.

In hindsight, this aspect reveals the necessity of considering not only the aesthetics and functionalities of the environment but also its performance implications. Balancing these factors becomes

pivotal to creating an efficient and seamlessly operable simulation.

As we come to the end of the "Results and Discussion" section, it's clear that we've learned a lot along the way. Although our final goal of deploying the simulation online hasn't been reached yet, our current achievements are noteworthy and functional.

The application's main features are in place and working well for local simulations. This is important progress, and it forms the basis for future work. While the application isn't fully flexible in terms of its code, it still serves its intended purpose effectively.

However, we've also recognized a missed opportunity. With better code flexibility, the application could have been more versatile and beneficial for the RAIL. This realization shows us that there's room for improvement and expansion beyond what we've achieved so far.

4 Conclusion

In order to advance the technological progress of the RAIL in human-robot interactions, and building upon previous work in this area, I was tasked with assisting in measuring the effectiveness of screen-based robots in conveying their gaze to humans. Previously, this was done through direct measurements in the lab using humans and the robot Stevie, but this posed significant organizational challenges. Thus, the decision was made to create a 3D simulation of this experiment that could be accessed online, significantly saving time in gathering tester data. My role was to contribute to the realization of this simulation.

Throughout this report, we have explored in detail the solution I provided to the problem. Ultimately, I succeeded in creating a functional simulation that addressed many constraints, such as maintaining realism and simulation performance. However, the simulation wasn't able to be deployed online due to a multitude of technical issues related to the use of the Unity tool. While on the surface it's an intuitive tool, when delving into more technical aspects, there can be bugs and there's not always help available to fix these issues on the forums. While the completed work isn't entirely finished, my contribution has certainly pushed forward the creation of this online simulation. Nevertheless, despite the presence of documentation, the lack of flexibility in the underlying workings of the simulation might require future developers to possess significant programming and Unity development expertise to efficiently continue and advance the project.
Reflecting on my experience, I believe that even if it resulted in a slower middle phase of my internship, I should have maintained the flexibility that was present from the beginning in my code. This could have simplified the overall development of the simulation in the long run.

I believe that the further development of this tool should be taken up by a qualified individual experienced in Unity development. The remaining workload for deploying the application on a website is still substantial but considerably less than building the simulation from scratch, so continuing its development is an option worth considering.
If the simulation I created is considered not flexible enough for further development, it would be wise to retrieve some of the elements from the simulation I've built to develop a new one. The most important components, in my opinion, would be Stevie's model along with its interaction scripts, the user interface graphics, the scripts that manage them, and all the information included in this report. These elements could greatly contribute to the creation of a new simulation.

Bibliography

[1] Glockenbeat. *Unity hangs making build during Light Transport*. Forum. Apr. 2012. URL: https://forum.unity.com/threads/unity-hangs-making-build-during-light-transport.467722/.

[2] Takayuki Kanda, Hiroshi Ishiguro, and Toru Ishida. "Psychological analysis on human-robot interaction". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*. Vol. 4. IEEE. 2001, pp. 4166–4173.

[3] Naz Kaya and Helen H Epps. "Relationship between color and emotion: A study of college students". In: *College student journal* 38.3 (2004), pp. 396–405.

Lexique

asset : In the context of Unity, an asset refers to any digital resource, such as 3D models, textures, scripts, audio files, or animations, that is utilized within the Unity game development environment to create and enhance interactive experiences. Assets serve as building blocks for constructing virtual worlds, characters, and interactions, contributing to the overall content and functionality of Unity projects.. 15, 27

GameManager : In Unity, a GameManager refers to either a script or a GameObject responsible for overseeing and managing different aspects of a game's logic and progression. It functions as a central control system, coordinating elements like scoring, level transitions, player lives, and overall game state. The GameManager often serves as a communication hub between various components, facilitating organization and efficiency in game development by centralizing crucial gameplay functions.. vi, vii, 16, 19, 20

GameObject : In Unity, a GameObject is a fundamental object used to represent entities within a scene. It serves as a container for components, scripts, and other attributes that define an object's behavior, appearance, and functionality in the game world. GameObjects are the building blocks of scenes and play a central role in constructing interactive experiences within Unity's game engine.. vi, 15, 16

GUI : In the context of Unity, GUI refers to the visual elements and controls that users interact with in a game or application. It encompasses menus, buttons, sliders, text fields, and other graphical components that allow players to navigate, make selections, and interact with the game's interface. GUI elements are typically created and manipulated using scripts to respond to user input and provide feedback. Unity provides tools and functionality to design and implement GUI elements seamlessly within the game environment.. viii, 26, 29

package :A package contains features to fit the various needs of your project. This can include any core Unity features included during the Editor installation, or other packages that you can install as needed.. 22

RAIL : Robotics and Artificial Intelligence Laboratory at Trinity College. The RAIL is a research facility at Trinity College, where the internship took place. It focuses on advancing robotics and

AI through research and technology development for human-robot interactions, autonomous systems, and intelligent machines.. iii–vi, xi, xiii, xvi, 1, 3, 7–9, 12, 14, 16, 37, 39, 40

script : In Unity, a script refers to a set of instructions written in the C programming language that defines the behavior and interactions of GameObjects within a scene. Scripts are associated with GameObjects to define how they respond to user input, environmental changes, and other events. By attaching scripts to GameObjects, developers can customize and control the functionality of game elements, enabling them to create dynamic and interactive experiences in Unity.. vii, 16, 19, 40