MARCO PREDATORO

TEAM D
CLIFTON OBSERVATORY PARK

VASUDEV MENON - TEAM MANAGER
MICHAEL WYRLEY-BIRCH - LEAD PROGRAMMER
CARLA ROSARIO - LEAD DESIGNER
NIKIL CHANDRASEKAR
ALYSSA CHAN
EMIL JINO

Contents

Contributions

- Vasudev Menon: 1.0
- Michael Wyrley-Birch: 1.0
- Carla Rosario: 1.0
- Alyssa Chan: 1.0
- Emil Jino: 1.0
- Nikil Chandrasekar: 1.0

I, confirm that I:

- Worked at least 400 hours
- Made an effort to have equal contribution to the team efforts
- Participated in daily meetings
- Helped weekly play testing

Signature: Vasudev Menon 

Signature: Michael Wyrley-Birch 

Signature: Carla Rosario 

Signature: Alyssa Chan 

Signature: Emil Jino 

Signature: Nikil Chandrasekar 

1 Individual Contributions

Vasudev Menon

Research into Flagship tech, Arduino, General VR/AR - 20h

Research into the Azure Body Tracking SDK and Azure Sensor SDK. Learnt to use the Arduino IDE and understood different types of available Arduino models for our specific use case.

Acquiring Tech - Constant

Explored and collated the necessary tech and accessories required after a consensus was reached with the team. Conveyed these requirements to Tech Hub, Hackspace, and Engineering Labs personnel.

Point of Contact with Panel / Tech from Luke and Adrian - Constant

As the Team Manager, I relayed information between the team and the Panel regarding progress and issues. Organization of room requirements, sending across the documents, posters and submissions.

Body mapping into Unity - 20h

Worked on inserting the body tracking from the Azure Kinect into Unity. Looked into different ways people have made similar games online and how we can incorporate it into our own game.

Synchronisation of Kinects - 20h

Worked on getting the 2 Azure Kinect Cameras synchronized to each other to solve occlusion, within our given space.

Border Compass - 10h

Learnt to use the ARCore and helped in creating the border compass implementation on the AR.

Buttons - 10h

Learnt how to use the Arduino Uno and built a large red button connected to an Arduino, which, when pressed, opens the box.

Simon Says - 15h

Brainstormed task ideas for Polos. Implemented the Arduino-connected Simon Says task with 4 differently coloured buttons across the ends of the room.

Eggs - 20h

Identified how the eggs fit into the theme of our game and helped implement egg harvesting as Polo's singular streamlined goal. Assisted in constructing and preparing egg nests, as well as equipping all the eggs with RFID tags and QR codes for seamless gameplay.

Egg expiration - 15h

Facilitated the integration between the augmented egg markers and RFID code to add an egg expiration feature. Real eggs have 5 seconds to be scanned before they turn Rotten, achieved by sending an Event after the timer is up.

Area Theming - 25h

Prototyped set design by generating examples of Stable Diffusion models like Midjourney and Lexica. Balancing technical constraints and audience accessibility with our creative ideas, I helped organize the game space into an unforgettable, immersive experience.

Asset Design ideation - 10h

Used stable diffusion models like DALLE and Midjourney to test and iterate through different possible asset design ideas.

Asset Design Modelling - 40h

Learnt how to use Blender and created the 3D models for the Claws, Marco head and Polo body. Created the 3D forest clearing scene that both sides of the game use.

Box - 10h

Made a loading bay with cardboard and motors to open based on a button press across the room.

3D Prints - 25h

Learnt how to use Cura and the Ultimaker 3D Printer. Modelled and 3D printed the 4 wrist braces for the AR phones to be placed on Polo players' forearms.

Planning Organization and Meetings - 20h

Scheduled regular meetings and deadlines while delegating tasks based on team members' skills, and establishing communication channels. Organized feedback from playtests so that they can be discussed by the team in future meetings.

Report - 25h

Set up and aided in the careful organization and editing of the report. Refined its content to enhance its clarity and presentation for a polished final result.

General team discussions - 80h

Brainstormed ideas, held weekly scrum meetings to review feedback, and enhanced gameplay.

Testing and Debugging - 120h

Conducted weekly internal and bi-weekly external playtests, and improved game robustness through debugging and fixes.

Michael Wyrley-Birch

Kinect Sensor and Body tracking - 30h

Researching the Kinect Sensor and body tracking SDK. Using the Microsoft Azure Kinect Sample GitHub to take the body tracking data and place it in unity.

Synchronising Azure Kinect - 80h

Researching different possible ways to synchronise the two cameras together.
Creating a synchronisation workflow that can be used to place the cameras in virtual space.
First place the main camera in virtual space using a person standing in the centre of the space. The difference vector between the rendered skeleton and $0, 0, 0$ is then the location of the camera and its roll and pitch being calculated from the accelerometer. This is then repeated many times and the results are averaged to make sure that we get an accurate location.
Second place the subordinate camera in relation to the main camera. We Calculated the difference vector and difference quaternion between the main and subordinate bodies to find the correct location for the camera. This was run many times for each bone in order to get a very accurate body location.
Creating a UI that clearly shows what steps you need to take in order to run the synchronisation.

Merging two Body Tracking Skeletons 16h

The two cameras each produce their own list of bodies that need to be merged. First calculate the distance between each body, if it is below some error value it is part of the same human. Render one of the bodies based on which distance to its camera is shorter, this is because the body tracking accuracy worsens with distance. These are then sorted based on which body is closer to the main camera to reduce the amount of flickering that occurs due to the bodies being tracked in a different order in each frame. Any bodies that are not within both cameras are just rendered as normal, being sorted by the distances to their respective cameras. The top two sorted bodies are then rendered with the camera's transform and rotation being applied to the skeleton.

RFID Reader 8h

Using an empty text field in Unity that is constantly active to read when the RFID is activated. Then hijacking the resulting value to find which specific RFID tag was used.

Arduino 16h

Research methods to get the motors and buttons working together and communicating with the Arduino.

Terror Radius Sound. 16h

Using the Euclidean distance between the Polos and Marco to play different sounds to add a sense of terror.

Torch Shader - 16h

Researching and implementing a shader that makes the object it is applied to visible whenever a spotlight is shining on it. First, the vector between the light and material is calculated, and then it is compared to the direction of the spotlight using the dot product. As the spotlight beam is a cone add the cosign of this angle halved this is due to the dot product producing a scaled cosign of the two vectors. The resulting sum can then be scaled by an arbitrary value to strengthen the effect.

Points Bar Shader - 8h

Researching and implementing a shader and script that displays the current points score as a mix of two colours where the higher the mixing point the better the player is doing, and the lower the worse they are. First, the points are turned into a percentage between the maximum and minimum points values and sent to the shader. The distance between the x coordinate of the pixel and the points percentage is then calculated through the smooth step function and used to decide the amount of mixing of the two colours.

Timer - 5h

Researched and implemented a script that displays the current time left in the game and calculates the winner once the game ends.

Set up the game world in VR - 16h

Added fog and skybox to the VR world.

Introduction Video - 24h

Creating a specialist model trained via DreamBooth on each team member and a specific drawing style.
Turning a video of two team members into an AI-generated video via stable diffusion.
Using the runway's Machine Learning based background removal to rotoscope the video.
Learning DaVinci Resolve in order to create the introduction video. Compiling multiple clips we took throughout the project, aswell as taking new videos, into one cohesive 5min video.

General team discussions - 80h

We dedicated time to brainstorming ideas and weekly scrum meetings to organize our plan.

Testing and Debugging - 120h

Weekly internal playtests as well as bi-weekly external playtests. General debugging based on feedback from playtests.

Carla Rosario

Setting up AR phones in Unity - 5 hours

Researched Android phone integration with Unity, initialised ARFoundation template and configured build settings to work with given Motorola phones.

Researching Unity, Azure SDK/Synchronisation - 40 hours

Contributed to group research, particularly in learning Unity and examining Azure SDK documentation. Explored Azure SDK sample repository and helped consider how to implement Kinect synchronisation.

AR Networking - 20 hours

Researched Photon and collaborated with VR & Kinect side in connecting AR phones to the network. Added join/leave room buttons, and received events.

AR Spatial tracking and Marco model view - 20 hours

Researched ARFoundation and its plane & spatial tracking framework. Augmented plane tracking scripts to place the grid on the floor, which was used to debug 3D space calibration. This was important for 3D AR Marco visualisation; the grid helped users infer the distance of Marco and helped us to validate that he was appearing where he should be. Added recalibration buttons for when calibration went awry, and configured 2 phones to spawn at different points in coordinate space.

AR Compass & Border Compass - 30 hours

Removed 3D Marco model from AR phone view and used his networked location to rotate a compass arrow on the AR phone instead. Researched ARFoundation further, and used ARsession to determine 2D angle between the phone user and Marco. In a later iteration, the compass feature was reconfigured to be a block moving around the edge of the phone. The location of the block corresponded to Marco's direction.

AR UI - 15 hours

Designed and implemented tech-hud style UI for the AR phone. This included concept art and brainstorming, through to Adobe Photoshop generation.

AR Terror radius implementation - 5 hours

Worked with VR side to augment their terror radius code for the AR phones. Determine Euclidean distances, which activated different AudioClips increasing in tension the shorter the distance. Researched Unity's 3D sound system and implemented it by attaching AudioSource to Marco and AudioListener to the ARsession.

Powerups - 15 hours

Brainstormed potential powerups for Polo side, and built on Unity's ARMarker system to increment powerup count when powerups QRcodes are scanned. Added buttons to send power-up events to Marco's side, allowing for two-way communication between the physical and virtual sides of the game.

Eggs - 30 hours

Conceptualised and co-implemented egg harvesting as Polo's singular unified goal. Came as a result of much discussion and brainstorming regarding Polo's gameplay, particularly how it could be streamlined and made more intuitive.

- *Egg markers:* Created markers for physical eggs, and added them to ARMarker system. Eggs were either 'Real' or 'Rotten', and spawned Canvas prefabs saying which. Real eggs are Polo's way to gain points.
- *Egg expiration:* Augmented egg markers and RFID code to add an egg expiration feature. Real eggs have 5 seconds to be scanned before they turn Rotten, achieved by sending an event after the timer is up.
- *Egg nests:* Conceptualised and built real-life themed nests, where Polos search for eggs.

Area Theming - 10 hours

Brainstormed and co-planned Polo area theming. Aimed to complement the VR Marco scene, creating a unified game ambience and atmosphere. Involved in research and concept art, using AI-generated imagery to inspire own sketches and plans. This also involved sourcing various materials, and building aspects of the 3D space: namely nests and leaves.

Asset Design & Concept Art - 25 hours

Conceptualised general game atmosphere, and worked on asset design to achieve this.

- *VR Scene:* Generated concept art and mood board to communicate the desired look and feel. This involved sourcing reference images and using these to draw concept art that was then used to build tree models for Marco's forest.
- *Game marketing and merchandise:* Used self-drawn concept art to make exciting and communicative game posters, as well as a team logo.
- *AR UI and Polo Area Theming*

Report writing & editing - 30 hours

Team meetings & discussion - 80 hours

Testing and debugging - 120 hours

Alyssa Chan

Learning - 5 weeks

Being unfamiliar with all of the software we made use of - Unity, Unity VR, Unity AR, Photon, Maya, and the Azure Kinect SDK - I had to devote significant amounts of time to learning and understanding each piece of kit, and making decisions on what parts were relevant to the project. I also put research into figuring out the implementations of each of our features, and of course, many many hours were put into figuring out how to debug them too.

Networking - 3 weeks

I had the responsibility of setting up networking across each of our 3 applications. This included:

- The research and learning of Photon's networking options, and the subsequent explanation of the structure of a Photon room to the rest of the group.
- Evaluating which options would be the most appropriate for our project
- The implementation of our chosen methods of synchronisation for each feature, in each of the applications; creating event managers and listeners, events and the raising thereof, the serialisation of values and creation of synchronised prefabs to be instantiated over the network.

Music - 3 days

Thorough research and analysis went into not only finding the music references to be handed off to the composers, but also the isolation of specific, desirable traits of each and every track we submitted, and even examples of what we had wanted to avoid. This involved listening to dozens of tracks from different games and separating each of them into their composite layers, pinpointing features and listing them as either desirable or undesirable.

Art and design - 1 week

In order to come up with a design for Marco, I researched all of the different egg-laying kingdoms of animals and came up with concepts based on each one. We eventually settled on more of an insectile look as we decided that that would be the most unnerving. I continued to iteratively brainstorm and get feedback from the team until I could draw a refined version of the face and claws, which I handed off to Vasudev to model. I also worked with Carla to create some concept art for the Polos, which I then modelled.

Animation - 2 weeks

I rigged Polo by manually inserting a skeleton into the mesh and painting on the influence weights for each bone. Multiple attempts had to be made due to poor documentation and inexperience.

I spent two days researching animation synchronisation options, eventually settling on the implementation of the synchronised Kinect-driven animations explained in detail in the Technical content section.

Making this custom system work with Michael's camera synchronisation code was a tricky and finicky process that required constant testing and adjustment.

Team meetings - 80 hours

Every day that we agreed to come onto campus to work, the team would have a meeting that followed an agile format. Every week, we would do an evaluation of the previous week, and decide on what tasks to focus on for the current week.

Furthermore, we discussed the feedback we had received from each of our playtests and discussed in detail how it should impact the trajectory of current and future weeks.

Team meetings were also integral to the co-development of features, especially ones which required multiple components, such as setting up networking, or testing whether or not each of the three applications interacted with each other correctly. Additionally, they were essential to being able to understand each others' code and subsequently build on it.

Testing and debugging - 120 hours

Miscellaneous

Trawling through documentation, cleaning up code and debugging took up almost the majority of the time. Also coded small prototypes such as the "Simon Says" minigame which was later scrapped.

Also frequently spoke with staff and TAs for advice, direction, and technical tips to solve various problems as they arose.

Nikil Chandrasekar

Learning Unity, Photon, Oculus Integration- 30 hrs

During the first 2 weeks, I learned and researched about VR-based unity projects and how they are set up. This involved reading through unity and photon documentation and numerous YouTube videos. Reading up on Oculus-unity integration

Setting up Oculus in Unity- 5hrs

Configured Oculus integration with Unity, researched and learned Oculus Unity Toolkits that allowed us to set up a working VR game. Researched and Implemented hand and headset tracking by using the XR interaction tool kit. Implemented scripts that set up VR controller characteristics for the game.

Photon Networking- 20hrs

Set up initial online networked room Researching and learning Photon to integrate into unity. Learned about Photon view scripts and implemented a Network Manager that managed the joining, leaving, and instantiation of players.

VR- Marco Model Implementation-35hrs

This involved setting up multiple scripts in different machines to make sure Marco's movements are networked to all systems that join the online room. Adding transform and photon view scripts to Marco's hands and head objects to synchronize movements.

Marco Claw was implemented by adding scripts like Hand Presence Physics that regulated the claw physics, The script initializes helps achieve smooth, physics-based motion. The Marco claws also needed colliders that would send an event when an in-game collision occurs

VR - Polo Model Integration-25hrs

Imported and configured polo models to have colliders and Implemented a first iteration fade in and out animation for Polos in the VR world, this involved writing scripts that render the Polo prefab material based on a timer. Imported and integrated the Polo Torch shader.

Events systems-15hrs

I also helped set up the points and collision by sending and receiving events based on the game state. Helped implemented events manager script that sent major game state changes as events.

Controller Haptics based on collision-10hrs

This involved creating scripts that would incorporate Haptic feedback into the meta-quest controllers based on collisions in the game.

VR World Model(skybox, lighting)- 5hrs

Imported and helped implement VR world ambience like adding a skybox, and regulating lighting.

Particle System-10hrs

Implemented a particle system for a fog-like ambience in the game scene, also added blood splatter for every Collision with polo.

Helped implement Torch feature-20hrs

I wrote scripts and implemented a torch feature for the Marco side. Implemented a deplete and replenish feature for the torches by adding custom scripts that handled the increase and reduction logic of the spotlight angle, Also implemented the UI for this in Unity.

VR side threat radius-20hrs

Integrating the threat radius on the VR side involved adding multiple scripts to the Polo and Marco prefabs to make sure the different audio clips are played based on the size of the radius.

Marco side Flashbang feature implementation-10hrs

Added a power-down feature for Marcos which allowed polos to inflict damage on Marco. This involved creating a canvas and writing a script that triggered a white canvas image when a flash-bang event was received.

Points Bar- 4hrs

Implemented the UI for the points system in the VR world using unity, and created a World space canvas with respect to the camera. Used the custom Events Manager script to regulate the increase and decrease of points and reflected these changes in the UI

Integration of Polo with bones - 10hrs

Kinects track the Polo's movement and convert it into bone data that is sent over the network using a Photon-Serialize view.

General Brainstorming and Scrum Meetings-80hrs

We spent time brainstorming ideas and had weekly scrum meetings to plan what we would do

Testing and Debugging - 120hrs

We had weekly play tests within our group and bi-weekly play tests with external members. We also worked in an agile way when working with the VR.

Emil Jino

Learning Unity, Photon - 30hrs

Spent time learning about Unity, and how we could utilize it to its full capacity. I also spent time researching Photon as this is how would we network our game.

Setting up Oculus integration in Unity - 5hrs

Set up Oculus integration in Unity for our VR project. Imported the package into the Unity project and added any necessary scripts or components to enable Oculus functionality. To test the scene, I built and run the application on the Oculus.

Scaling virtual space to that of the physical space - 20hrs

To ensure that Polo and Marco were contained in the same area, we scaled the scene in the VR to match that of the physical space. We created a 4x4m space where gameplay would occur

Creating Marco/Hand and headset tracking - 30hrs

Co-developed the Marco prefab and mapped hands and the head to the Oculus controllers and headset. Created our own "hands" and "head" in Blender, to which we added relevant colliders, rigid bodies and scripts.

Networking using Photon - 20hrs

Set up Photon Pun networking on both Marco and Polo devices so that both game objects would be visible on the other device.

Co-developed sending and receiving events, e.g collisions, points, Marco movement networked - 40hrs

Implemented an Event manager script as a team. This updated the points on both sides when a collision occurred between Marco and Polo. The points were displayed on a canvas at the start however we went on to change it to points bar.

Co-developed further features such as Haptics for collisions - 10hrs

Added haptics to the VR controllers by making a script so that the player would get instant feedback when they hit the Polo.

Imported Polo spacesuit game object - 1hr

Our team created a spacesuit model in Blender and I imported it and used that as our Polo model.

Co-developed implementation of space skybox - 5hr

To make the game fit our design of being on an alien planet, we added a skybox with a space image, we also implemented a fog effect to the scene.

Threat radius for Marco - 20hr

We added a threat radius system, it has 3 rings that play sounds as the Polo gets closer, with the sounds intensifying with each smaller ring. Done using Euclidean distance between Polos.

Added Torch feature - 10hrs

We co-developed the torch feature to ensure a more exciting experience for Marco. This works by pressing the trigger to light up the scene and find the Polos, who would otherwise be invisible. Shader material was applied to Polo to allow this. We implemented a deplete and replenish feature, to stop Marco from using it indefinitely.

Added decoy Polos - 10hrs

Implemented a power-up for the Polo to use. This decoy effect works by Polo pressing a button on their AR phone which instantiates a number of random Polos in Marco's scene.

Added torch bar - 4hrs

Implemented a torch bar and integrated it into Marco's arms for a more flush game experience.

Integration of Polo with bones - 10hrs

Implemented Polo which has bone data. Kinects track the Polo's movement and convert it into bone data that is sent over the network using a photon serialized view to the VR and thus allowing more realistic movement for the Polo.

Implementation of Particle Systems - 10hrs

Implemented fog and blood splatter particle system for better visuals and user experience

General brainstorming and scrum meetings - 80hrs

We spent time brainstorming ideas and had weekly scrum meetings to plan what we would do each week.

Testing and debugging - 120hrs

We had weekly play tests within our group, but on top of this, we had bi-weekly play tests with external members, such as friends, lecturers etc. We also worked in an agile way when working with the VR, as we tested each feature one by one.

2 Top 5 Contributions

1. Seamlessly integrated **two physical game spaces** into **one virtual environment**, enabling **interactions** between game spaces, such as power-ups or attacks.
2. Reliable **body tracking** from **two synchronized cameras** with all the joint and bone data **networked across applications** to simulate the character in our game
3. Made **custom models and 3D scenes** with **custom shaders** to improve to appeal and ambience of our game
4. We networked **three distinct applications** - our main game, VR game, and AR game - to form a cohesive, unified gaming experience.
5. We integrated **physical game components**, eggs hidden in textural nests, with tangible impacts on the virtual game.

Check out our video:https://youtu.be/seFJKZjpeTE.

3 Nine Aspects

3.1 Team Process

- Conducted daily meetings to ensure continuous, high-quality project development and alignment among team members (see Section 5.1).
- Used Asana Kanban Boards to efficiently track tasks and monitor contributions (see Section 5.2).
- Utilized GitHub Organization to effectively manage and organise different VR and AR applications involved in the game (see Section 5.6).
- Engaged in pair programming, aiding the development of robust and reliable solutions (see Section 5.5).
- Had regular brainstorming sessions, generating innovative ideas and approaches, using a collaborative Miro board (see Section 5.3).
- Held weekly reflective debrief sessions to discuss successes, challenges, and areas for improvement following each playtest. Built a supportive environment for discussion. (see Section 5.4).
- Adopted a user-driven approach, with weekly game tests and feedback reviews (see Section 6.2).

3.2 Technical Understanding

- Researched the Azure Kinect Documentation and synchronized the two Azure Kinect cameras together (see Section 6.1.4,6.1.3, 7.2).
- Explored Azure Kinect Body Tracking SDK to figure out how to render images from two cameras into a single interpolated body (see Section 6.1.4,7.1, 7.2).
- Researched and implemented PUN 2 network (see Section 7.9).
- Learnt steps to track collisions in Unity, manipulating game object components (see Section 7.8).
- Sent body animations over the network (see Section 7.11.9).
- Used built-in render pipeline over the Universal Render Pipeline as it caused significant compatibility issues in the VR game (see Section 6.1.1).
- Discovered significant limitations in the accuracy of AR spatial tracking for live Marco position visualisation. (see Section 7.5.1, 7.5.2).
- Researched and experimented with ARFoundation's reference image tracking capabilities (see Section 7.6.1)
- Researched and explored the implementation of Wi-Fi/Blue networked Arduino buttons, taking into consideration both physical and software constraints(see Section 7.7).

3.3 Flagship Technologies Delivered

- Room-scale VR - Incorporated 3D body mapping across two physical game spaces into one single virtual environment. Allowed for unencumbered, instinctive movement, using world-scale to lessen the learning curve. (see Section 7.1, 7.2, 7.3).
- Physicality - Integrated VR, AR and physical components into one unified game allowing for dynamic, immersive and physical gameplay. Bridged physical eggs into the virtual game via radio-frequency identification (RFID) and marker systems. (see Section 7.3.3, 7.6, 7.10.1).

3.4 Implementation & Software

- Implemented a synchronous Azure Kinect body tracking system that could track 2 players without occlusion (see Section 7.1, 7.2).
- Networked several devices to create a seamlessly connected game (see Section 7.9).
- Implemented a reliable VR hand & head tracking system (see Section 7.3.2).
- Implemented a collision system that sends events across the network (see Section 7.8.1).
- Implemented a robust AR Marker scanning system triggering networked events (see Section 7.6.1).
- We meticulously applied the core principles of Object-Oriented Programming throughout the development process (see Code).
- Implemented powerups sent across the network, triggering 'power-downs' on the opposing side (see Section 7.6.3, 7.3.3)
- Integrated an RFID system to scan Eggs and trigger point events (see Section 7.6.2).
- Created a robust tug-of-war points system via Photon events sent to all applications (see Section 7.8).
- Animated Polo model with Kinect-tracked bones, serialized over the network (see Section 7.11.9, 7.9).

3.5 Tools, Development & Testing

- We used the Unity game engine to create the three separate applications (see Section 6.1.1,6.3).
- The Azure Body Tracking SDK and Azure Sensor SDK were used to capture body tracking information (see Section 6.1.4).
- Oculus XR was used to create an immersive virtual reality experience (see Section 6.1.5).
- Unity's ARFoundation was used to build and test a wide array of AR functionality (see Section 6.1.6).

- We relied on GitHub to maintain our codebase securely while ensuring accessibility for all team members (see Section 5.6).
- Blender, Maya, Adobe Illustrator, Adobe Photoshop, and Ultimaker Cura were used for modelling and design purposes (see Section 6.1.8).
- Regular user testing was conducted internally and externally, using Google forms and discussion to gather both qualitative and quantitative feedback (see Section 6.2).
- Used project planning and brainstorming tools Asana and Miro to guide development (see Section 5.2, 5.3)

3.6 Game Playability

- Follows natural instincts: running from threats, chasing prey. Low learning curve for controls. Room-scale reduces the potential for motion sickness (see Demo, Section 4.3).
- Onboarding video: Health and safety instructions and how to play the game. Particularly VR induction. (see Demo).
- Intuitive UI VR: Torch charge fades up and down claws (see Section 7.4.2).
- HUD-style AR phone strapped to wrist: Wrist brace increases automatism. UI presents just 2 clear and elegant functions - powerups and egg scanning. (see Section 7.11.10, 7.11.11).
- Calibration: Initial camera calibration ensures robust and consistent body tracking (see Section 7.2).
- Minimap for the Polo player to orient themselves in the virtual game space, and see Marco's location (see Section 7.5.3).
- Haptic feedback alerts players of enemy collisions (see Section 7.8.2).
- A timer tells both sides how much time to win (see Demo).
- An intuitive tug-of-war points system that is easy to comprehend by all players (see Section 7.8).
- Threat radius audio adds way to determine enemy proximity (see Section 7.5.4, 7.4.3).

3.7 Look & Feel

- Designed custom 3D models in Blender and Maya to build an otherworldly game space (see Section 7.11.1, 7.11.2).
- We crafted an eerie & immersive atmosphere for the VR game using particle systems (see Section 7.11.4).
- Built a tense and otherworldly atmosphere in the physical game space for the Polo side(see Section 7.10.1).
- Collaborated with 2 composers to design music that would increase tension and atmosphere (see Section 7.10.2).
- Designed, built and animated custom Polo models with networked Kinect bones (see Section 7.11.8, 7.11.9).
- Designed, built and textured unique alien Marco model, using custom shaders to convey crucial game state information (see Section 7.11.6, 7.11.7, 7.4.2)
- Wrote custom shaders with shader graphs to create an authentic alien planet atmosphere. Mimicked wind on leaves and glow in the bark. (see Section 7.11.3).
- Used post-processing effects for enhanced visuals and atmosphere (see Section 7.11.5).
- Built futuristic tech-hud style UI for AR phone to increase sci-fi feel (see Section 7.11.11).
- Created a introduction video, using Stable Diffusion, that helps set the scene for the game, and introduce player missions (see Section 7.10.5).
- Built a cohesive and engaging game brand through custom logos and posters (see Section 7.10.4).

3.8 Novelty & Uniqueness

- Merging two physical spaces into a single virtual game space creates a seamlessly immersive and novel experience for players (see Section 7.2).
- There is a duality to the game concept; there is an underlying moral question as to which side is the right side (see Section 4.2)
- Body mapping allows both Marco and Polo to view each other's movements and positions in real-time, adding a sense of urgency and excitement to the game (see Section 7.1).
- The game relies on natural instincts, such as running and catching, making it accessible and engaging for players of various skill levels (see Demo, 4.3).
- A strong network supports the game's infrastructure, enhancing its capabilities and functionality (see Section 7.9).
- The game's visual design is highly immersive and interesting, allowing players to fully engage with the setting and atmosphere (see Demo, Section 7.10.1).
- Players on both sides can influence the other's progress directly, encouraging competition and tension (see Section 7.3.3,7.6.3, 7.8).

- Wrist-strapped datapads that help Polo interface with the virtual world are a unique and exciting element that increase connectivity with the virtual game (see Section 7.11.10, 7.6.1).

3.9 Report & Documentation

- The report contains extensive technical information and development discussion (see Section 7, 5, 6).
- Our code includes useful comments, which helped support collaborative development (see Code).
- We have created extra documentation to help anyone with the required tech run the game (see Github README.md).
- The report is written professionally in LaTeX, using a custom template created by our team (see Report formatting).
- We have made clear diagrams and drawings to understand and explain different concepts to each other and to anyone learning about our game (see Figures in Report).

4 Abstract

4.1 Story

The great, great, great, great grandchild of the playground game Marco Polo. Human Polo has landed on an alien planet – they need to harvest a compound that will cure a pandemic ravaging the human race. It is only found in the eggs of the alien species Marco. Alien Marco is defending his eggs from thieves – hunting the human invaders.

In this report, we discuss the development of Marco Predatoro, including various game tests and consequent iterations. Throughout the report when we refer to Marco, it is the VR alien side, and when we refer to Polo, it is the AR and physical human side.

4.2 Concept

Marco Predatoro is a multiplayer room-scale game, where one side is played in fully immersive VR and the other is played in a themed real-life room. When you step up to play you choose a side; are you the good guy or the bad guy? Is there any such thing?

If players choose Marco they play as an alien in VR and are immersed in an eerie forest clearing, where human invaders run around in the darkness stealing Marco's eggs. Marco's goal is to kill the human invader before they can steal all its eggs and escape back to Earth. Marco must chase them, using its alien powers to illuminate the forest. When Marco attacks the Polo, the Polo loses points and gets haptic feedback, via a datapad attached to their wrist, to let them know that they have been hurt.

If players choose Polo, they frantically search alien nests for eggs containing a compound that will save the human race. They must scan the eggs with their datapad to confirm that they are fresh and take them to the loading bay to extract their juice before the timer runs out. From the moment eggs are removed from the nest, it is only a matter of time before they turn rotten. Sending rotten eggs to earth will unleash a deadly virus that could wipe out humanity, and make Polo lose major points.

While tasked with this earth-saving mission, Polo is pursued by an invisible alien. The only way to see this elusive creature is through a cutting-edge SpectroQuantum camera mounted onto their ship. This advanced technology allows them to detect the alien's energy signature and track its movements on the ship's screen, heightening the suspense and thrill of the game. This is projected onto the wall or the physical space.

While evading Marco, Polo has various tactics at their disposal to hinder and outsmart their pursuer. They can temporarily blind Marco to slow him down or deploy clever decoys to create confusion and throw him off their trail. These strategies add depth to the gameplay and provide players with additional ways to interact with their enemy.

All this is achieved by virtually mapping two physical rooms into a single virtual space. The body of the Polo is tracked by Kinect cameras and displayed in Marco's VR space. While Marco's head and hand coordinates are tracked by the MetaQuest and broadcast back to the Polo side.

The eggs are scanned using an AR phone strapped to Polo's wrist. After confirming the eggs' freshness, they must be placed in the loading bay to be juiced: an RFID scanner. The Polo gains points by juicing the most eggs and whereas Marco gains points by attacking the Polos. The points system works in a tug-of-war fashion and the side that has gained the when time is up emerges as the winner.

4.3 How to Play

There are 2 sides to our game and, so we decided to make gameplay last 5 minutes, so each player can attempt both sides. We aimed in all our developments to make a game that felt balanced on both sides, so both are equal parts challenging and intuitive.

How to play Marco

- Move around the VR space as you would move around in the real world. This is possible because it is room-scale.
- Make your claws glow by pressing the trigger on the back of the MetaQuest controller. This will illuminate the dark forest clearing around you, and help you find Polos.
- Your glow powers will deplete with use. Take breaks to let glow replenish.

- Seek out spacesuit Polo in the 4*4m clearing, hit them with your claws to gain points. No button press is required - just have the controllers in hand.
- Observe the tug-of-war points bar to see who is winning.

How to play Polo

- Play in the real world, moving around a 4*4m game space
- Search physical alien nests for eggs, and scan them by placing in view of the AR phone camera attached to your wrist.
- Scan real eggs on the RFID scanner in the loading dock (a physical box), before the timer on the AR phone runs out, to gain points.
- Observe a screen map of yourself and Marco in the space. Use this to evade him.
- Engage powerups by clicking Decoy or Blind buttons on AR wrist datapad.
- Observe the tug-of-war points bar on the map screen to see who is winning.

5 Team Process and Project Planning

5.1 Organisation & Meeting Structure

At the start of the Games project, we committed to consistent weekly meetings in person. After the 5 week sprint to create the MVP game, we adopted an agile workflow, working in fortnightly/weekly sprints to iterate our game based on reflections and feedback from the prior version. Each week the tasks for that sprint were informed and confirmed by several meetings:

- Monday - Check-in meeting with Tilo and panel members: We met every week prior to the check-in meeting to set out the key points we wanted feedback/opinions on. After each meeting we distilled the relevant action points we gained from it. Mondays were key for confirming the next steps and defining the tasks necessary to achieve them.
- Thursday - Gameplay test: After the MVP, we resolved to do a weekly Gameplay test, inviting peers to play the game and give feedback on our changes. Thursdays provided a deadline to aim for, which helped to ensure we were achieving something each week. After the Gameplay tests, we gathered feedback using either a Google Form or by documenting people's opinions in a Google Doc.
- Friday - Gameplay test reflection: After every Games test, we met the following day to discuss feedback and reflect on the future direction of the game. This reflective process is explained in more depth later.

Figure 1: Collaborative discussions and team building

- Meetings with our TA, Armand: We had fortnightly meetings with our mentor Armand, who was available for quick queries via Teams, as well as longer catch-ups about our progress. For the latter, we prepared bullet points to discuss and get feedback on. We also discussed implementation details on Unity with Jordan.
- Meetings with tech hub, Adrian and Luke: Throughout the project we were in close contact with the Tech Hub, meeting at least once a week to discuss our tech requirements and gain advice on the best ways to implement new ideas.
- Consistent work meetings: Aside from the regular meetings in our week, outlined above, we met at least 4 days a week to work on the project together. These meetings stretched the entire working day, and involved discussion, coding, debugging and testing.

5.2 Planning: Kanban board & Whatsapp

Since each member of our team had different skills, we split up tasks according to people's strengths. This meant ultimately our work in different aspects - tech, design, planning, report - resulted in even contributions overall.

We found the best way to delegate and organise these tasks for each sprint was to use a Kanban board, as we could easily see who was doing what and what was left to be done. We chose to use Asana for this as we were familiar with the user interface and were able to keep track of each member's present tasks. In addition to the typical columns of To-Do, In Progress and Complete, we had an extra To-Do column for future tasks/ideas which had been identified but were not relevant to the current sprint. This was useful to divide longer-term tasks from shorter weekly tasks and to bookmark possible ideas for future iterations. Within

each Kanban block, we allocated tasks to individuals, outlined subtasks, and, when applicable, provided links to relevant research or resources. We also used What-

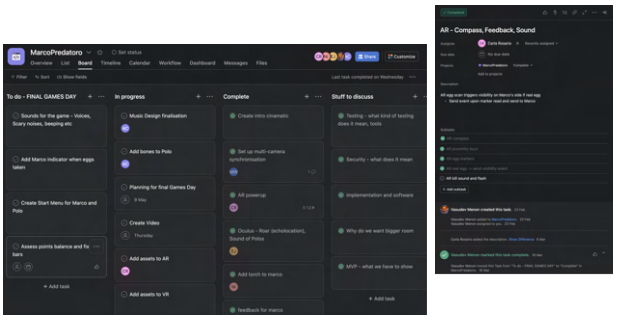


Figure 2: Asana board and individual block, including subtasks and assignee

sApp as a continuous communication channel, both for organising meetings and sending interesting resources or ideas we found. Whatsapp's relative usability on both mobile and desktop was a key factor in our choosing this.

5.3 Brainstorming: Miro Board

Throughout the project, we used a Miro board, which acted as the collective creative brain of the team. This approach facilitated the gathering of research and ideas in an unstructured manner before consolidating them into tasks while offering the flexibility to adapt dynamically as needed. The flexibility of Miro was conducive to the brainstorming process as ideas did not have to fit into specific tasks: hence it worked well as an intermediary tool, prior to task delegation via the Kanban board.

It also worked well as a continuous store of all our ideas that we could dip back into when a discussion was going stale. And it helped us to communicate the game vibe to the composers, as we were able to share the visual and musical references we had collected. insert screenies

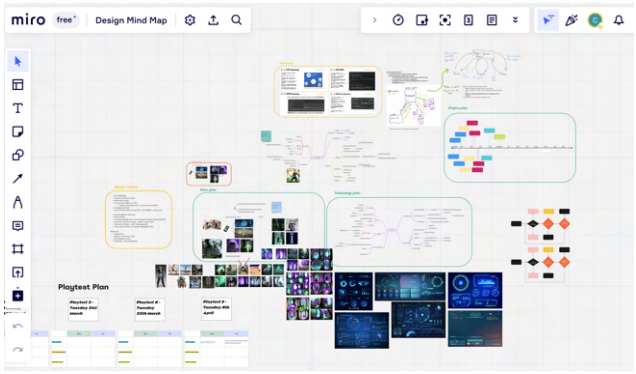


Figure 3: Miro board used for brainstorming across all game aspects

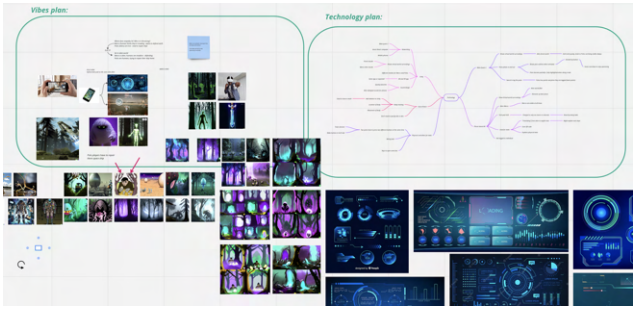


Figure 4: Detail of miro board, using mindmaps to brainstorm and reference images to generate mood boards

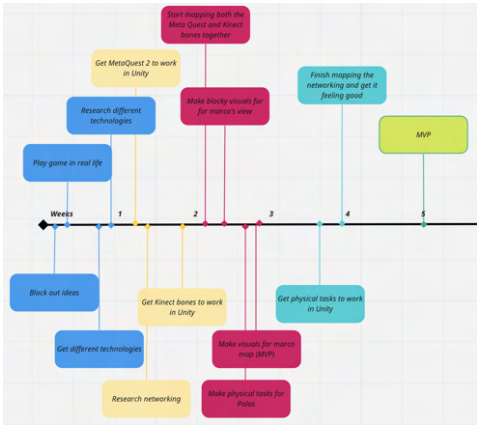


Figure 5: Plan for weekly sprints prior to MVP, created on Miro Board

5.4 Reflective Discussion

Earlier on in our project, we found that people often had different perceptions of the tasks to be done and general game direction. This was only discovered after features had been made, or in Monday panel meetings when we were explaining our next steps to lecturers. We, therefore, updated our team process mid-way through our development, to enforce reflective group discussion.

Each week began with a discussion of the bigger picture, ensuring we were all aligned with the overall direction of the game, including both look & feel and gameplay practicalities. We then worked on extracting smaller action points from this, informed by feedback from previous game tests, and set out each person's tasks for that week.

This framework was useful to ensure we did not lose sight of our overall aims and meant each week we were in some way moving towards achieving our final goal.

These were not always easy discussions, with Games tests often resulting in divided opinions about the correct direction to take the game. However, the most difficult discussions were also often the most fruitful; we worked through the pros and cons of different approaches and considered a wide range of perspectives.

Conflict resolution

As in any team project, there were times when disagreements and misunderstandings led to group tensions. Rather than let these fester, we allocated time in our weekly meetings for people to voice any concerns or feelings and worked to make sure everybody felt heard and valued.

A key factor in our success as a team was our resilience to disagreement and our openness to testing many different possibilities. For discussions where there was no clear-cut decision as to which direction we should choose, we resolved that the best way to find out would be to make it and play it. In all iterations, we ultimately let the gameplay decide our direction; the priority throughout was to make a game that was balanced, coherent and, above all else, fun.

5.5 Pair-Programming

Pair programming was a key part of our development process, collaborating on large coding features and bug fixing. After initially taking a more solo approach, we found pair programming improved our team robustness in two major ways.

1. *Ensured features developed were in-line with final goals:* Rather than having one team member misinterpret a group idea, having two people collaborate meant the perception of the task to be done was cross-checked.

2. *Robustness to member absence:* If a member was absent for whatever reason, as is inevitable in long-term group work, pair programming meant at least one other person understood their code and could continue their tasks.

5.6 Version Control: GitHub

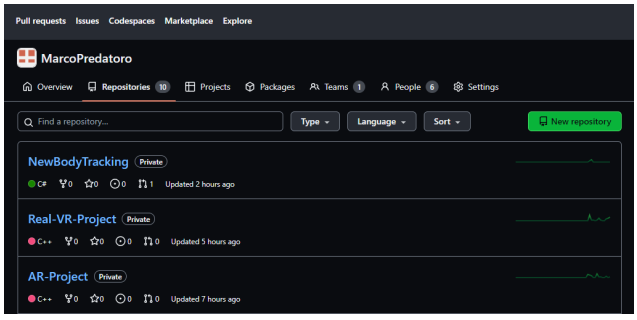


Figure 6: Github organization with our 3 distinct applications

GitHub was a critical tool in version control. Given we were collaborating on 3 separate applications, across Marco VR, Polo body tracking and Polo AR, Github was essential for maintaining a consistent and up-to-date code-base, with a detailed paper trail of previous iterations and features.

5.6.1 GitHub Organisation

From the start of our development process, we made a GitHub organisation. This was particularly useful because it allowed us to have separate repositories for each application, under the same umbrella organisation.

We determined that three separate repositories were a better approach than just using one with different branches because of the complexity of each application. A huge priority was avoiding merge conflicts, which we discovered early on were particularly difficult to solve in Unity, so having our GitHub process as streamlined as possible was the optimum way to approach this. It also meant we could use feature branches without having to ensure they were stemming from the correct application branch, hence reducing the possibility of mistakes.

5.6.2 GitHub Branches

GitHub branches were particularly important when multiple people were working on separate features within the same application. Given we were using GitHub Organisation, we were unable to implement branch protection rules without upgrading to GitHub Teams/Enterprise or making our repositories public. We felt keeping the repositories private was important to avoid spying or vandalism, so we compromised by verbally agreeing to only merge branches after at least 1 approving member had reviewed the pull request. This worked sufficiently as we were only 6 people and in constant communication with each other.

6 Software, Tools and Development

6.1 Development Software & Tools

6.1.1 Unity

Unity was the game engine we used to develop the core logic and models for the VR-AR game. Unity was used as a unified platform that seamlessly integrates with several technologies such as the XR Interaction Toolkit, Photon, ARCore, AR Foundation, and Kinect SDK.

Using Unity's versatile building blocks like GameObjects, Prefabs, Assets, and Scripts we modelled environments, interactive objects, and multiplayer rooms and defined their behaviour/logic by attaching scripts and components to these game objects.

We used the built-in render pipeline for this project. This was due to the materials not being updated properly on the VR side when changing to the Universal Rendering pipeline. Attempting this shift caused all to become pink error materials. Using a shader graph was also a problem in the Virtual reality project due to issues in compatibility with Metaquest's Android compiler.

6.1.2 Photon

For networking, we used PUN 2 (Photon Unity Networking), a Unity package for creating multiplayer games. This allowed us to synchronise objects, variables and structures over the internet between the different machines required to run the game, such as points, animations, and events for collision detection and egg delivery.

6.1.3 Azure Kinect Development Kit

We used the Microsoft Azure Kinect Development Kit to help with computer vision and body tracking development. It can control an array of sensors, including microphone arrays, an RGB camera, a depth camera, IR emitters, an accelerometer and a gyroscope [9].

6.1.4 Azure Sensor SDK & BodyTracking SDK

The Azure Sensor SDK allowed us to interface with the Kinect sensors and cameras [5], while the Body-Tracking SDK allowed us to interface with the Kinect cameras and computer vision algorithms[1].

6.1.5 Oculus XR

We used the Unity Oculus XR Plug-in, to create the virtual reality (VR) experience for our Oculus Metaquest 2.

This provided us with access to the Oculus platform's features, including hand tracking, spatial mapping, and cross-platform functionality. This also included the Oculus SDK, which includes APIs for rendering, input, and audio.

It extends Unity's existing XR platform, which provides a set of APIs for working with different VR and AR devices, to include support for Oculus devices and features.

The plug-in supports both single and multiplayer modes which is essential as it allows us to network Polo into Marco's game.

6.1.6 AR Foundation

We used AR Foundation to build our AR application. It is a Unity platform that combines core features from ARKit and ARCore, Apple and Google's augmented reality SDKs, to build applications that are portable across platforms. We used its plane tracking and image tracking features in particular to address different features throughout the development of the Polo side of gameplay.

6.1.7 Arduino

We used the Arduino platform, along with its integrated development environment (IDE) to combine the capabilities of various microcontrollers and sensors. We used it to make the buttons and the Simon Says task in our game, which we eventually removed from the final game.

6.1.8 VS Code Live Share

When we were not able to meet in person, we found the VS Code Live Share plugin useful to carry out pair programming. As mentioned in Section 5.5, this enabled us to provide support to each other during development.

6.1.9 Blender & Maya

We used both Maya and Blender for modelling due to different team members having experience with one or the other.

Maya is a 3D animation and visual effects software. It was used to create and rig a 3D model, which was then imported to Unity, where the controller was then driven by the skeleton generated by the Azure Kinect.

Blender is a free and open-source 3D computer graphics software toolset. It was used to create the 3D modelled versions of Marco's claws, the virtual space and the Polo body displayed on both the Screen and the VR headset.

6.1.10 Ultimaker Cura

We utilized Ultimaker Cura to generate precise 3D Prints for the Arm brace that holds the AR phone, keeping the phone and user safe from hazards. It is a powerful slicing software that streamlines the 3D printing process by efficiently converting digital models into optimized machine instructions.

6.1.11 Adobe Photoshop and Illustrator

Photoshop and Illustrator were used to design custom UI elements for the Polo side, as well as to develop the Game brand identity through logos and posters.

6.2 Testing

We self-organised weekly Game tests, where we tested out our weekly changes with peers who both had and hadn't played the game before. This allowed us to collect 2 sorts of perspectives: opinions on the current iteration versus the previous, and opinions on the current iteration as a standalone experience. We considered the standalone experience to be important, as the game should be successful for brand new players, but often the feedback from players who'd played before was useful in determining whether most recent changes improved, or actually made the game worse.

Our development process was often not linear; if a new feature did not improve gameplay, we were not averse to reverting back to older more successful ideas. The Polo side is a key example, wherein we trialled many different ways of visualising Marco from the Polo side (see Section 7.5), searching for the solution which would be equal parts fun *and* intuitive. We found our simplest and earliest solution, a screen minimap, was actually the best one; people found it the most playable, and most enjoyable, and it showcased our flagship achievement of mapping the two physical

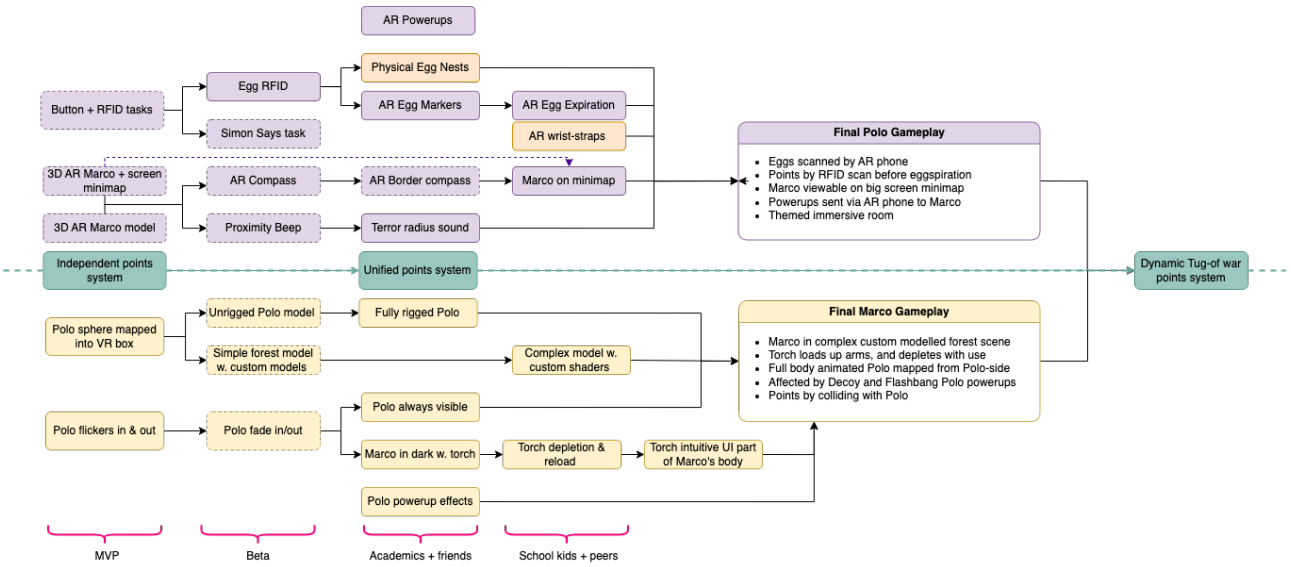


Figure 7: Flowchart of how features evolved across significant playtests

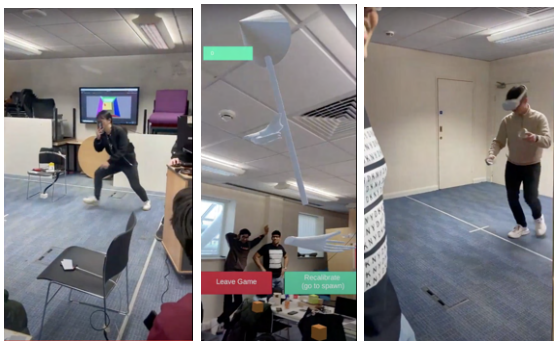


Figure 8: External games test, synchronous shots of Polo gameplay, AR phone and Marco gameplay

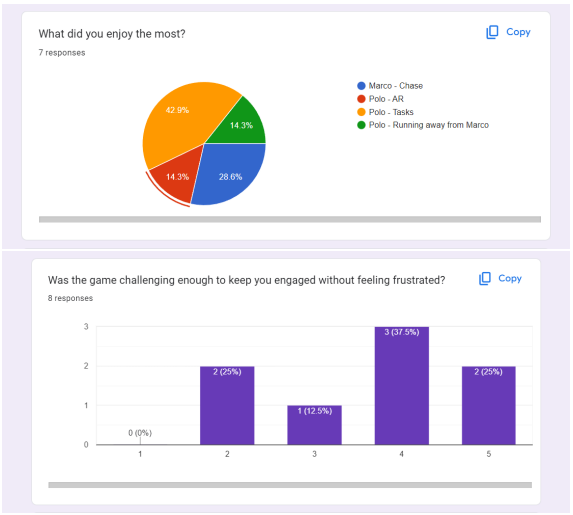


Figure 9: Example of Google Form stats generated from MVP

spaces together. Despite having worked long and hard on building more complex AR solutions, we were ultimately led by our user testing. This was key learning throughout the project; complex does not mean better, robustness above all.

In addition to peer testing, we also gathered feedback from panel members in the MVP and Beta testing. This feedback was particularly useful as it was more diverse than that of our peer group. For example, the physicality of our game was highlighted in ways that hadn't been flagged by our peers; would having our game very physically fast-paced be exclusionary of certain groups, or unsustainable for a 5-10 minute timeframe? Collating these wider opinions allowed us to reflect on how our game would sit in the real world, and how we could improve it to be more intuitive and accessible for all players.

Overall we had an estimated 50 people test our game, ranging in age and gaming experiences. We leveraged these game tests to gather both quantitative and qualitative feedback, using Google Forms to collect scorings and word of mouth for opinions and general feedback. We took notes on all feedback, ready for us to review in reflective discussion meetings, as discussed in Section 5.4.

Fig 7 shows how game features evolved across different play tests, and led to our final game version. They are separated into game tests that were key turning points in our development: MVP, Beta, Academics & Friends, School kids & peers. The takeaways and justifications for each feature change are discussed in detail in Section 7.

6.3 Final Game Tech

The final game uses a wide variety of hardware, networked into one cohesive game:

- Azure Kinect Cameras
- MetaQuest 2
- Android Phone
- Eggs tagged with RFID and AR markers

The final gameplay features of Marco and Polo's sides are outlined in Fig 7. They include 3 main software applications:

- *VR Metaquest application:* Running the Marco side game on MetaQuest2 headset and its two controllers.
- *AR Android phone application:* Running the game on an AR phone strapped onto the Polo's arm.
- *Body-Tracking application:* Running the main server application while displaying Polo's 'ship camera' view of the virtual scene with Polo and Marco in it. All three of these are seamlessly networked together into one cohesive and elegant game. If anyone were to add further functionality, they could do some by pulling from the repo and using the above hardware and software tools.

7 Technical Content

7.1 Body Tracking

We built all of our body tracking programs from Microsoft's sample projects [2], this allowed us to quickly adapt what we wanted and learn how the Kinect worked without having to build from scratch. It works by translating the bone data from the body tracking SDK to bones in Unity.

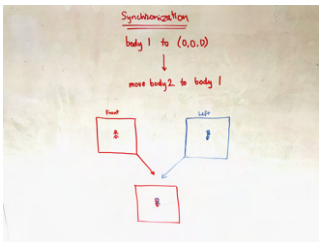
We found that with only one Kinect, users can easily occlude the other (e.g. if they are in front of the camera), which was one of the deciding factors in using two Kinect sensors given we were considering a multiplayer Polo side. The other was space constraints for the cameras. They have a maximum body tracking range of 4m but in practice become less reliable the further from the camera you are, making it almost impossible to track the 4*4 space robustly with only one camera.

Whilst we did get body-tracking working for two players, we ultimately decided to only have one player on the final Games day. We were not able to conduct sufficient play tests because of the tight schedule and the need to prioritize other tasks. Using two players had consequent concerns about the robustness of our synchronisation algorithms. We also felt that we didn't have enough feedback about the game experience with two Polos, so it wouldn't make sense in our user-driven development process. The 1v1 player is the best version of our game and we wanted to put our best foot forward during our demo on Games Day.

7.2 Mapping 2 physical spaces into 1 virtual space

For our flagship technology, we needed to map the real world into the virtual space. To do this we found the camera's location and rotation in Unity, so the skeletons could be accurately positioned. To accomplish this we decided two synchronisation steps were necessary:

1. Place the main camera in the virtual scene.
2. Place the secondary camera in the scene based on the synchronisation of the first camera.



7.2.1 Synchronisation step 1: Main camera

After much deliberation, we decided to use the skeleton generated in Unity to obtain the translation and floor location of the camera, with its roll and pitch being calculated via the built-in accelerometer.

The virtual camera starts at (0,0,0), so all skeletons generated will be placed in the scene with an offset from this. We used this fact to our advantage, by placing a real human at the centre of the scene we can calculate the difference between the virtual skeleton and (0,0,0) thus calculating the location of the camera.

For efficiency's sake, we calculated the difference vector of the spine and feet, due to the spine being the most central bone and the feet giving us the position of the floor. Using more bones would slow down the synchronisation process, without adding much more in terms of precision.

This is run multiple times, with a delay between each iteration to obtain an average between frames for greater accuracy and less overfitting of one skeleton. The camera's translation is then $(\text{spine difference}x, \text{foot difference}y, \text{spine difference}z)$. The roll and pitch can be calculated using, $Roll = \phi = \arctan \frac{a_y}{a_z}$ and $Pitch = \theta = \arcsin \frac{a_x}{g}$ [11]. These are used to set the camera's rotation.

Note that this method assumes the absence of yaw rotation for the camera. As our camera consistently maintains a central position, additional computations related to yaw rotation would be deemed unnecessary.

7.2.2 Synchronisation step 2: Secondary camera

For the second synchronisation step, we calculate the translation and rotation of the second camera in relation to the first.

To do this we place a person in the centre of the real world with their body rotated to be between the two cameras, as in Fig 10.

The rotation is calculated from the average of the difference between quaternions for each bone in the skeleton. The formula for the difference between two quaternions is $Q_{\text{difference}} = Q_0 * Q_1^{-1}$ (where $Q_{\text{difference}}$ is the difference quaternion, Q_0 is the first bodies bone rotation and Q_1^{-1} is the inverse of the second body's bone rotation). We use an accumulator variable to get the average for all bones, where spherical linear interpolation (Slerp) between the accumulator and the

Figure 10: Sychronising the second camera

difference quaternion with a weight of 0.5, forms the average.

As in step 1, this is run multiple times using the same averaging method and delay between each iteration for greater accuracy and less overfitting to one skeleton

The translation of the second camera can be calculated in the same way as in synchronisation step 1 but the difference vector will be between bones in body 0 and body 1.

7.2.3 Deciding which skeleton to render

Each camera keeps track of its own skeletons with their coordinates relative to its camera coordinate space. In order to only render one body per human in the scene, the skeletons from each camera must be merged in some manner.

To find which skeletons correspond to which person, given more than 2 people in the scene, we take the 3D coordinates of the spine from camera space and translate them into world space using the rotation and translation of the cameras, Translation $= V_{\text{Camera Vector}} + V_{\text{Pelvis Vector}}$ and Rotation $= Q_{\text{Camera Rotation}} * Q_{\text{Pelvis Rotation}}$. If the Euclidean distance between two skeletons is within a certain error margin, they are assumed to come from the same human.

To decide which of the two skeletons to render we used the distances to their respective cameras. The closer body is always rendered, as the further away a person is from a camera the worse the body tracking software is. Any bodies that don't overlap are assumed to only be in view of one camera and are rendered normally.

As the final iteration of the game is only one player, only the body with the shortest distance to its respective camera was chosen to be rendered to limit the interference of people in the periphery and produce an overall smoother experience. We could increase and decrease this limit to test the efficacy of different numbers of Polo players.

7.3 Marco: VR Gameplay

7.3.1 Configuring room-scale VR

The first task in developing Marco gameplay was generating the room-scale virtual space. Marco plays the game in a separate physical area from Polo, but with

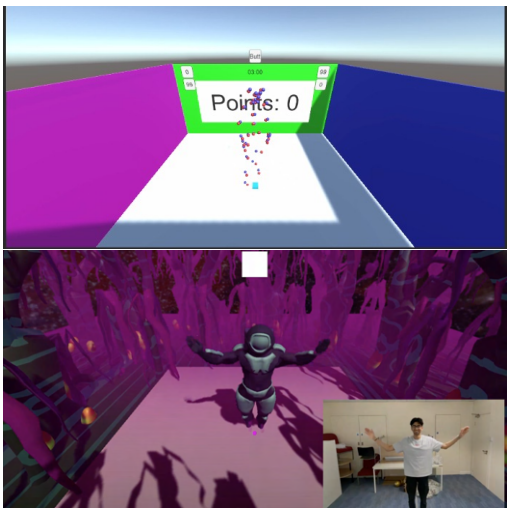


Figure 11: 1:Synchronisation step, point cloud shows skeletons generated by 2 cameras. 2: Successfully calibrated body mapping

identical dimensions, wearing the VR headset. We, therefore, had to make sure the scaling of the space correlated to that of the real world, which in this case was a 4*4m area. This was also important to reduce possible VR-driven motion sickness [8], as it would mean player motion would be the same scale as in real life.

Usually in Unity, the scaling is quite straight forward and 1 unit equates to 1m in real life. However, we encountered an unexpected problem where the room was scaled incorrectly, due to difficulties exporting the model from Blender's coordinate system to Unity's. We, therefore, experimented with the scaling system to make sure it was indeed mapped to 4x4m, otherwise, the Gameplay would not be fluid and the mapping would be misaligned once the Polos from the real-life space were introduced.

7.3.2 Hand and Headset Tracking

Once we had the virtual space with the correct dimensions, the next step was to add hands and a head for Marco. We initially did this by using a cube game object for Marco's head and using pre-installed hands from the Oculus package. However, we then went on to create our own custom head and claws.

To make sure the VR heads and hands were accurately mapped to the physical locations of the controllers and headset respectively, we created a C# script to link the game objects to the positions of the controllers and headset.

The Hand Presence Physics script regulated the claw physics. It initializes the object's Rigid-Body component and locates the Controller Transform in the scene. It then updates the claw's velocity to match the controller's movement using FixedUpdate() for smooth, physics-based motion. One problem we encountered during this was the hands registering collisions with eachother; this is discussed in depth in Section 7.8.1.

This allowed the person in VR to feel like they were actually in the virtual world, as their actions were

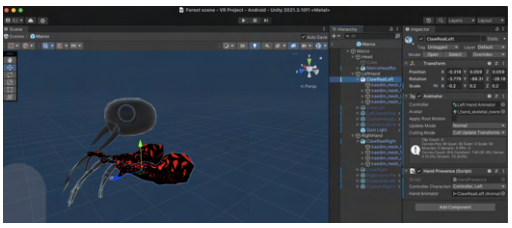


Figure 12: Marco Hand and Headset tracking in Unity

mapped to hands they could see in the space.

7.3.3 VR Powerups

As discussed in Section 7.6.3, powerups were a key tool for increasing the connection and interactivity between the two sides of the game. Once powerup events were received on the Marco side, they triggered either Blind or Decoy effects:

1. *Blind:* When triggered Marco gets white-screened and loses all sense of direction. This gives Polo time to strategize their moves. We implemented this feature by adding a white image attached to a canvas game object on the VR scene. This canvas would then be activated solely upon receiving a 'Blind' Photon event. The canvas was set to a world-space view in Unity improving its effect. The scripts also regulated how the screen faded in and the duration of the flash, making for a believable flashbang-like effect. We also added a ringing sound when this happens to allow for more a multisensory experience.

2. *Decoy:* When activated, Marco is surrounded by a randomly generated group of 5 Polos, and we have the ability to set the quantity of Polos in the group. This creates confusion regarding the location of the real Polo. This was done by implementing a script that instantiates the Polo prefab multiple times at random points within the 4*4 game space. The locally instantiated game objects are then removed after a set time period by calling the Destroy() function in Unity.

7.4 Marco: Polo visualization

It was important that Marco could see the Kinect-tracked Polos accurately positioned in the virtual world. To do this we created a prefab called "Polo" that would be instantiated when Polos entered the Photon room, by entering the Kinect-tracked game space.

Since the interaction between Marco and Polo is crucial, we tested several iterations of how Marco sees Polo.

7.4.1 Fade in and out

In one of the early tests, we made the Polo game object fade in and out every 5-10 seconds. This was done by adding scripts that changed the alpha value of the Polo game object, smoothly altering Polo's opacity. In tests, this posed several issues:

- *Players didn't explore the space:* Marco stood still and waited the 5-10 seconds to pounce on Polo wherever he reappeared. This lost the chase aspect of the game.

- *Players had no sense of Polo's direction:* In the intervals with no visual cues, Marco had no way to know where to chase Polo.

- *Players had no way to influence their chances:* Due to their lack of influence on Polo's visibility, they had nothing to due in periods of invisibility, and felt it was more a game of chance.

7.4.2 Torch

Implementing a 'torch' feature, affected Polo's visibility without directly altering the Polo game object. We hoped this would encourage space exploration and chasing, as Marco felt control over Polo's visibility and could in turn increase their success by searching around the space.

We darkened the VR scene and attached a Torchlight to Marco's claws that could illuminate Polo, who is otherwise shrouded in darkness. The illumination of Polo dictates the pace of the VR gameplay and it was imperative we still kept it fast and fun.

After numerous playtests, the user consensus was that the torch unbalanced the game in favour of Marco. We, therefore, constrained players from illuminating the scene indefinitely, by gradually reducing the torch scope and range as they hold down the trigger. Releasing the trigger gradually replenishes the torch battery, forcing Marco to have periods of darkness. We achieved this by accessing the input device's controller characteristics and ensuring that any trigger input would initiate the torch's gradual decrease.

Given a physical torch does not fit well with Marco's alien lore, we needed UI to somehow convey the torch as an alien power. To accomplish this we used the Spotlight object in Unity and attached the object to the Hand/Controller/Claw object. We then used two separate shaders for rendering the effect: one that only revealed an object when it was within the torch's light, and the other to change the visibility of the arm holding the torch based on how much torch energy was left. This aimed to intuitively convey to Marco how much torch light was left, by making the claw glow and deplete with torch use.

1. The first shader calculates the dot product between the vector $\vec{v}$ from the spotlight to the object and the direction $\vec{d}$ of the spotlight. Since the spotlight is a cone, we then combine this with the cosine of half the spotlight angle $\theta_s/2$: $\cos(\theta_s/2) + (\vec{v} \cdot \vec{d})$.
   The result can then be scaled by an arbitrary value to strengthen the fall-off effect.

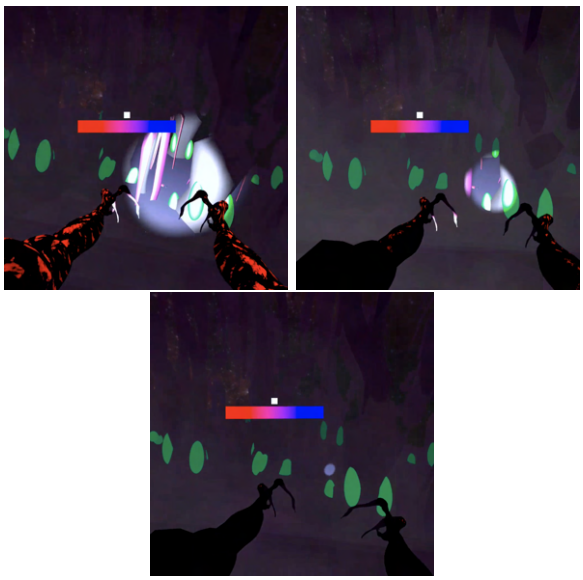


Figure 13: Marco's claws showing 'torch battery' depletion

2. The second shader uses Unity's smooth step function to calculate if the current pixel should be rendered or not based on the percentage of torch energy left, plus some blend width to make it smoother.

7.4.3 Threat radius & 3D sound

To add more depth to the VR gameplay we added distance-based sound effects. These sounds conveyed Polo's proximity and served as a hint for Marco to track the enemy down, particularly when his vision was limited.

To implement this we added scripts that used the Euclidean distance between Polo and Marco to play different sounds that increased in urgency the closer the radius. This script is the same as the one explained in detail in Section 7.5.4, incorporating Unity's Spatial audio in the same manner.

7.5 Polo: Marco visualization

7.5.1 AR live model

The first iteration of Polo gameplay included an AR live model of Marco, visible through a handheld phone. In order to map the 3D space, we utilised ARFoundation's PlaneTracking features to detect the first horizontal plane and place a grid on top of it. We reasoned that if the gamespace was clear of obstructions, this should place the grid on the floor. The grid also functioned as a debugging aid, in that if the grid corners didn't correspond to the corners of the gamespace we would know that the AR coordinate space was out of sync with the VR and Kinect applications. Joining the room would then instantiate a Marco model prefab, which moved in AR view according to its networked coordiantes.

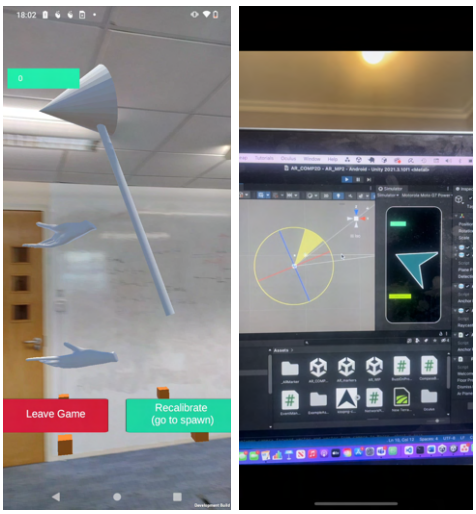


Figure 14: In-game screenshot of the AR live model, and AR compass development

In development, we also tried using ARFoundation's 2D Image Tracking to place the grid on a marker that was placed on the floor, but we found this was less reliable and less useful for debugging the 3D space mapping, as grid placement was not explicitly tied to the AR coordinate space.

In Game Tests this felt novel and exciting, but had various drawbacks:

- *Inaccurate mapping of Marco in the 3D space:* ARFoundation's native spatial tracking technology dynamically updates the coordinate space as it detects more reference points and is moved around. This meant it changed through gameplay and led to inaccuracies in Marco's location on occasion.

- *Vulnerable to fast or swift movements:* Moving the phone too fast, or placing the phone down caused the AR coordinate system to reset, thus positioning Marco incorrectly.

- *Difficult to perceive depth and distance:* The Marco model was an arbitrary shape with no defined dimensions or shadow, so it was difficult to intuit distance.

- Players were torn between looking into the phone and interacting with the physical space - the two did not mesh well, and it lacked natural flow

The only way to solve the coordinate system malfunctioning was to return to the spawn location and reset the tracking, which was hugely disruptive to gameplay.

A key finding from the Games test was that players often chose to ignore the phone and leave Marco's location to chance; we, therefore, reasoned that future iterations would need to make the location information more automatic and intuitive.

7.5.2 AR Compass & Border Compass

Our first proposed solution to the above issues was an AR compass with a metal detector-like beep. We thought this could provide more immediate Marco location feedback, bringing Polo players back into the physical space. It would also reduce the effect of minor inaccuracies in the AR spatial tracking, thus improving the gameplay experience.

To implement the compass we calculated the 2D angle between the ARSession and the networked Marco head. We used this angle to rotate an Arrow sprite on a 2D canvas, which was updated via Unity's onGUI() function. The development of this can be seen in Fig 14.

In Games Tests this was more robust than the AR model, but had several drawbacks:

- *AR phone had too many functions:* In this Game test the phone was also used to scan AR markers. User feedback said it was confusing and inelegant to have to scan markers while a compass was in the centre of the screen.
- *Players felt disconnected from virtual space:* The lack of Marco visuals meant it was difficult to perceive the threat, and it was a less immersive experience

To address the first drawback, we tried a toned-down approach, where instead of a central compass we migrated it to the edges of the phone screen: Border compass. This meant it no longer blocked the marker scanning zone.

The implementation of the Border Compass used the same angle calculated in the original compass but updated a Rectangle sprite on the 2D canvas instead. The key difference in the code was that instead of rotating the Rectangle, we had to change its position to the point on the border of the screen where the angle would point to.

In Gameplay tests this solved the multifunctional problem somewhat, but the players still felt detached from the virtual space; the mapping of the 2 physical play spaces was lost. Given this was one of our key flagship technologies and one of the *coolest* aspects of the game, we needed to find a way for it to be communicated to players on the Polo side.

7.5.3 Minimap

In various early playtests we used a minimap screen alongside the AR Marco visualisation. We chose to continue down the AR route initially due to concerns that the mental mapping from a birds-eye map would feel unnatural. A key issue with the AR solutions, however, was that ultimately players felt detached from Marco's side.

During a playtest with our friends, we put the minimap screen on, see Fig 15 to help explain how the game worked. It was immediately more of a hit than the AR Compass they had been testing, and the game required a lot less explanation. It felt more elegant, more playable and more exciting. We attributed its success to various factors:

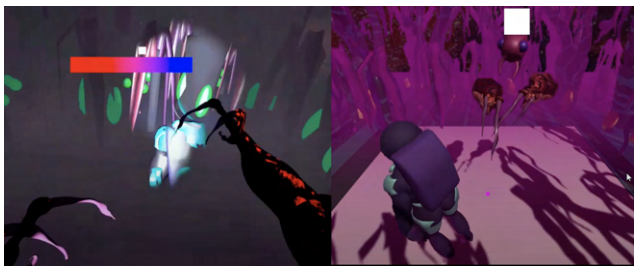


Figure 15: In-game footage, left side shows Marco's view, right side shows Polo's minimap screen view

- *Polos can see themselves in the virtual space:* Seeing their real-life actions mapped into the VR scene was exciting and novel. We'd been desensitised to this by our many hours of playtests, so having people see it for the first time reminded us where the most impressive aspects of our game lay.
- *Polos can see their effect on Marco:* When Polos send a power-up attack to Marco, they can see him react in real-time.
- *Robust Marco visualisation:* This was the most robust solution to Marco visualisation; if Marco collided with Polo on screen, Polo's felt handheld vibration and points reduced.

7.5.4 Threat radius & 3D sound

To make the game feel more immersive, proximity-based sound was proposed, so the closer Marco was, the more intense the music became. This was initially introduced as a pairing for the AR compass to give some audio feedback as to Marco's proximity. It added a layer of tension to gameplay that we felt enhanced the feeling of being hunted.

We enforced 3 concentric rings of different 'threat radii' around the AR phone; if Polo entered a ring, a particular sound was played. This used the ARsession and Marco's head coordinates to calculate the Euclidean distance between the 2 and activate different sounds according to the proximity. The AudioListener was attached to the ARSession, and the AudioSource was attached to Marco; this meant in addition to the different beep speeds indicating proximity, Unity's built-in Spatial Audio would also be in effect.

In later versions we used the Kinect-tracked Polo coordinates to calculate Euclidean distance, instead of the ARSession. This made it more robust, as we were no longer relying on the temperamental AR spatial tracking system.

We also reduced the number of threat circles to just 1, as due to the size of the space and the fast pace of the game, 3 concentric circles felt slightly confusing and disorienting.

7.6 Polo: Gameplay

After trialling a variety of Polo tasks, we concluded that we needed to find 1 key Polo mission that was intuitive and robust. This would crystallise the Polo gameplay and fit better with the Marco side: two missions directly opposed to one another.

Stealing Marco's eggs made sense in terms of its simplicity and keeping with the lore of the game; this is the point at which our game became Human Egg stealers vs. Alien Egg protectors. Polos would have to look for Eggs in the play space and send them back to Earth via the ship's loading dock to deliver life-saving compounds to disease-ravaged Earth.

To add layers of difficulty to this one task, we decided to include Rotten Eggs, which if sent back to Earth via the Loading Dock, would unleash a deadly virus/a baby alien. This would require Polos to use their datapad (AR phone) to scan markings on the egg and determine if they were safe or rotten, all whilst continuing to evade Marco.

7.6.1 AR Eggs

In order to determine which eggs are Real or Rotten, players must use the AR phone camera to scan codes on each egg. To implement this we extended ARFoundation's ARmarker package, using a custom C# script, to spawn different prefabs for different codes. The AR has two main functions with respect to the eggs:

1. *Determine Real or Rotten:* Scanning the marker on the egg spawns a UI prefab on the datapad displaying 'Real' or 'Rotten'.

2. *Egg expiration:* Designed to add urgency to the game. Scanning a Real marker begins a timer and if the egg has not been docked after 5 seconds it *expires*: Fig 16. In practise this is achieved by sending a Photon event across the network, which maps the Egg's unique ARmarker to its corresponding RFID code and then updates the RFID code's value to 'Rotten'. The egg, therefore, registers *negative* points if scanned.

7.6.2 RFID cards & eggs

Polos can gain points by sending the eggs to earth via the loading dock, where an RFID chip on the egg is scanned by an RFID reader, and point updates are sent over the PUN network.

The earliest iteration of this concept used just RFID cards. Though this iteration worked well as a proof of concept, many players were able to find ways to "beat the system" and gain way more points by either leaving the cards on the scanner or tapping it multiple times to gain points before coming around again to do the same thing again. This led us to consider a system wherein we enforced single-use only.

Attaching RFID tags to the physical egg concept enforced this in two ways:

- *Loading dock takes eggs away:* The loading dock concept requires players to deposit their eggs, and they are therefore unable to repeat the same RFID

- *Each egg is uniquely mapped to markers and RFID in the code:* Once a Real egg has been scanned by the AR, there are only 5 seconds to deposit it - after this its RFID value is changed so it can gain no points, and the AR marker cannot be rescanned. This means it can only be scanned to gain points once, within those 5 seconds.

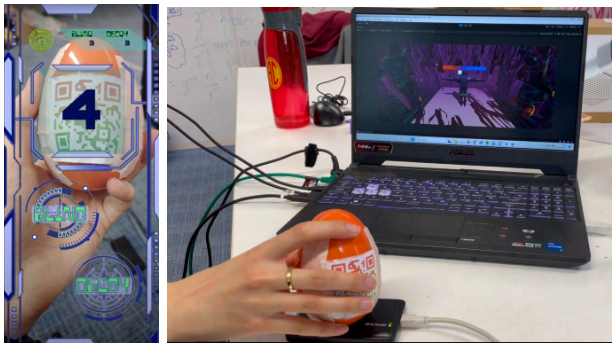


Figure 16: Illustrated above is the ARmarker and RFID scanning process of an egg

Physical plastic eggs interface with the game via RFID chips as follows. Each egg has a unique RFID tag attached to it, where each tag has a unique string of characters associated with it [4]. The RFID scanner is integrated into Unity via and Text box on a UI Canvas. Every time an egg is scanned the box reads the RFID code as a string and then checks this against a dictionary of known eggs. The points are then incremented or decremented based on the value of the RFID in the dictionary: 'Real' or 'Rotten,

7.6.3 AR Powerups

The Polos can engage 2 powerups during gameplay: Blind and Decoy. These powerups can be triggered from the datapad android phone that is strapped to their wrist. When a powerup button is pressed, a Photon event is sent across the network, and received by the VR-side. This then triggers the Blind/Decoy effect in Marco's gameplay, and the power-up count is decremented on the Polo datapad.

The consequent effects of the powerups on the Marco side as discussed in length in Section 7.3.3.

The two powerups were not initially key to the gameplay. But from user feedback, it became clear that Polos wanted to have more interaction with Marco. It helped to balance the game and increased the interactivity of the 2 sides, therefore making gameplay more immersive and interconnected. This built on the success of our flagship technology, in mapping 2 separate spaces together.

7.7 Polo: Arduino & Buttons

During our various playtests, we explored numerous tasks involving buttons. Initially, we'd imagined an escape-room format to Polo's gameplay, wherein they would need to perform a sequence of tasks to achieve points. Buttons constituted one of these said tasks.

7.7.1 Button to open RFID card box

Buttons were used in conjunction with the RFID card game test, wherein a button press was needed to open a box containing the cards. At the point in our development when we were still considering 2 Polos, we thought the buttons could enforce a teamwork element. One person would have to hold the button to keep the box on the other side of the space open. We implemented this by following the simple example button sketch and diagram provided in the Arduino IDE [3]. We connected a big red button with wires, LEDs and an Arduino Uno, to a server motor that opened the box. Based on this initial test we learnt the following things:

- *Tactile feedback was appreciated by all:* Players liked the physicality of hitting a big red button
- *Too simple of a task:* Though this added an extra element to the game it wasn't one that added any complexity or strategy

Taking the above into account we leaned into a more complex Polo task incorporating buttons: Simon Says.

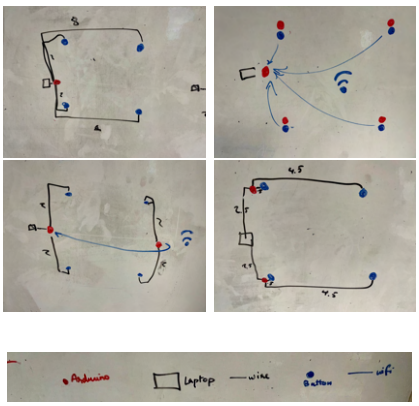
7.7.2 Simon Says

Figure 17: The LED light-up buttons that were planned to be used for the Simon Says task

Drawing inspiration from Simon Says, we made a task where the Polos needed to hit a sequence of buttons in a colour-coded order. This would open the loading bay, inside of which is the RFID scanner for the Eggs. The colour code would appear once on the AR phone strapped to the Polo's arm, hence adding a memory challenge to the task.

On playtesting this we stumbled into the following issues:

- *Connecting the buttons together was difficult:* We came up with multiple ways that we could possibly connect four colour-coded buttons in our 4*4 space. Below are the different ways we proposed, considering a variety of wireless and wired solutions:



This proved harder to actually implement. The HC-05 Bluetooth Serial Transceiver and EPS8266 WiFi Module that connect to the Arduino weren't reliable enough to ensure robust connectivity. Furthermore, integrating this into Unity with the Ardity plugin didn't meet our specific needs.

- *Diminished enjoyment:* Playtesting revealed that most participants did not enjoy this particular aspect of the game. They found it tedious and frustrating.

We, therefore, decided to streamline the Polo side to one egg-stealing task, as described in Section 7.6.1, which meant that we ended up completely scrapping buttons. This allowed us to prioritize ironing out the gameplay experience on the Polo side, with less hardware to connect to our already complex project.

7.8 Points and Haptics

An essential part of the gameplay is the point system; players need to know if they are winning. Points were communicated across the network via a Photon EventManager. The Marco side sent point-updating events when Marco collided with Polos, and Polos sent them after successfully scanning eggs, as detailed in Section 7.6.1.

The first iteration of the point system was very basic: Marco and Polo had separate point counts, and whoever had the highest points at the end won. In both the AR and VR applications, we displayed both these counts on a Unity UI canvas. Game tests yielded several downsides to this approach:

- *Players felt separated from their opponent:* Given they are in separate physical spaces, this was already a challenge for us; separate points systems aggravated this and both sides felt they could have no effect on the other player
- *Points not instinctively interpretable:* Having two point counts on the UI canvas meant players had to read both and distinguish which was higher. In such a fast-paced game, this was not ideal

We, therefore, decided to try one *unified* points system, by having Marco collisions decrement points, and Polo eggs increment points. This winner was Marco

if the endpoint count was negative, and Polo if it was positive. After testing we found this was better in that players felt stronger interactivity with the enemy side, but the UI was still difficult to intuit.

Hence the final version of the game includes a tug-of-war style points bar, that both changes in gradient colour and has an indicator that moves left or right to indicate who is winning. The latter was included as we considered potential accessibility issues for Visually Impaired players if we relied on the colour gradient alone.

We drew on our own previous experience of games to design this, in the hope that for many players, the affordance of a familiar points bar would help reduce the learning curve and increase gameplay engagement. We anticipated even players with no gaming experience would find this interface more accessible due to the familiar concept of tug-of-war.

After player testing, this was by far our most successful points system for several reasons:

- *Players felt they could directly impact their enemy's success:* Having your point increase also have the effect of decreasing enemy points felt exciting and motivating.
- *Players could understand who was winning in a single glance:* The graphic representation of points allowed for faster processing and ease of play, leveraging familiar affordances for ease of understanding.

7.8.1 Collisions

In order for Marco to gain points, we had to register when he collided with Polo and send a CollisionEvent over the network, as discussed in Section 7.9. To detect collisions we added collider components to the Marco Claws and Polo prefabs and implemented custom scripts.

In the HapticInteractable script, collisions are handled by the OnCollisionEnter() method, which is called automatically by Unity when a game object with a collider component comes into contact with another game object with a collider component. The method checks if the collided object is Polo and if the canSendCollision flag is true. If these conditions are met, it sends a haptic impulse to the target device, calls the SendCollisionEvent() method, and starts a Coroutine for collision cooldown, to make sure we don't send points every frame.

We discovered a problem when adding colliders to the claws, in that they often collided with each other, causing glitchy hand tracking. To fix this problem we used Unity's Layers to avoid self-collision by assigning each hand to its own layer. We then used the Physics Manager to configure the collision matrix, specifying which layers should not interact with each other.

7.8.2 Haptics

On both the Marco and Polo side, we implemented haptic feedback, to alert both sides when a collision was made. Given the 2 sides are not in the same physical space, this was a particularly important feature to alert players when they had collided with their invisible enemy. It also increased the sense of connection between the 2 spaces, to feel a tangible physical impact from the other side.

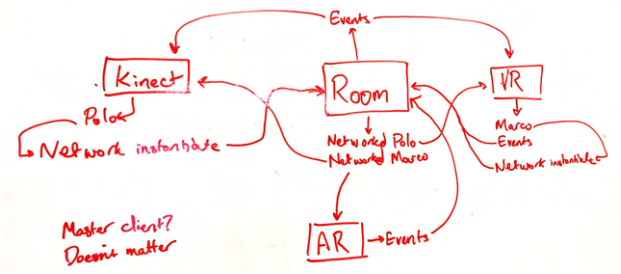


Figure 18: Diagram of implemented Photon network

On Marco's side, haptics was implemented in the script where collision detection occurred, so Marco would get vibration feedback via the controllers when they hit the Polo. We did this by using the XR interaction toolkit and accessing the controller characteristics from the input device(VR controller) and sending a vibration of the duration and strength of our choice.

On the Polo side Unity's built-in Handheld.Vibrate() was called when a collision Photon event was received from the Marco side.

7.9 Networking

Given we were using three separate applications, networking was essential to link them into a single united game. We used PUN 2 to do this, wherein each application joined the same Photon room.

Our network structure employed a decentralised event-based framework: Fig 18. We settled on this because each application needed to send and receive information equally, especially in earlier iterations. The Photon room structure lent itself well to this. There is no inherent "server" instance; each client joins a "room" and one is randomly assigned as the "master".

We employed a plethora of Photon networking methods to synchronise the various aspects of our game.

The majority of the game occurs client-side on each laptop and is transmitted via Photon's event system. This included the Marco-side collisions, the Polo-side power-ups & egg-scanning, and the events signalling the game start and the game end. In previous iterations, this also included the scrapped Simon Says minigame. We wanted to keep as much client-side as possible, as we determined low latency was important to the game's playability. This required us to create events, functions to raise said events, and event handlers which were distinct to the application it would be integrated into.

The remainder of the game was networked via serialisation. Photon offers a means to instantiate objects over the network, linking instances of objects using a "PhotonView". However, it is possible to link two different Unity prefabs to the same PhotonView,

which we took advantage of for our asymmetric gameplay. For instance, the Polo game objects on the VR and Kinect sides were different. Photon also provided methods to serialise the position and orientation of these objects through a PhotonTransformView.

However, for several parts of our game, these were not appropriate solutions.

In particular, we could not use the PhotonAnimatorView for animating the Polo models. We determined that putting a PhotonTransformView on each and every bone in the skeleton would produce too much overhead, especially as we wanted to keep the latency low. Instead, we created a custom data structure to store the orientation of every bone in the skeleton and wrote custom methods to pack, serialise, unpack, and control the receiving skeleton. The packing and unpacking were, unfortunately, necessary, since Photon does not support the serialisation of complex structures such as Unity's Quaternions, or structures-in-structures, such as vectors in dictionaries.

7.10 Atmosphere creation

Due to the physicality of our game, building an immersive experience on the Polo side was important to improve the look and feel. We wanted to mirror this on the Marco side, creating a united aura and ambience.

7.10.1 Area Theming

Since the Polo side is not in VR, we had to build on the physicality to make it as immersive as the Marco side and mirror the same atmosphere.

- *Eggs:* We used physical eggs, bought and augmented with AR markers and RFID tags, to enforce Polo's tasks as described in Sections 7.6.1 and 7.6.2. This added visual appeal to the task, as well as a materiality that tied to the natural physicality of Polo's gameplay.

- *Egg Nests:* To build the alien feeling, we created textural alien nests coloured brightly and furry in patches. Inside the nest the eggs are buried in water beads, making the search for eggs an engaging and tactile experience.

To contain the water beads we covered the top with a stretchy fishnet fabric that built further texture and kept the inside visible while containing the possible mess.

- *Lighting:* We added mood-lighting LEDs to fit the theme of our game. This is coordinated across both sides of the game, using alien purple and blue hues.

- *Costume:* We will be dressing up our players in fun costumes to increase their immersion in our game. Polo will be wearing a spacesuit helmet. Our team will also be dressed up in capes, alien sunglasses and headgear.

- *Biophilic design:* To enrich the narrative of our game, which is set in a forest clearing, we incorporated netting with leaves to amplify the immersive atmosphere. This creative addition not only strengthens the connection to the natural environment but also elevates the players' overall experience.

7.10.2 Sound & Music design

Initially, we gave our composers a list of tracks that we wanted, and a general feel we wanted from each of them (as well as several references); for instance, a background track, a chase theme, a menu theme, a victory/loss sting, and several sound effects. Attempting to convey the moods we wanted with specificity, but without technical music knowledge, proved very tricky. The composers were able to deliver in the end.

As we further developed the game, we encountered further issues. These include figuring out what defines a chase in a 4m x 4m square. The majority of individuals have a reach exceeding 0.5m. Therefore, it is essential to consider how often a "chase" would be initiated when a player stands at the centre of the play space and spins around. Eventually, we pivoted from a "chase" track to an "anxious" one, but new features continued to demand new sounds.

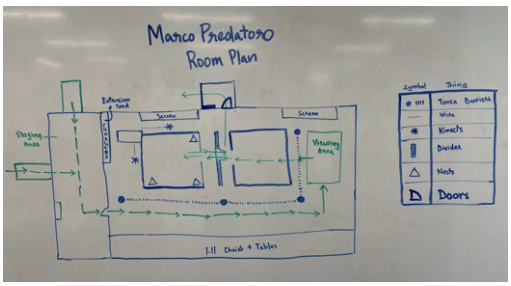


Figure 19: Room plan for Games Day

Figure 20: Different logo iterations, same core concept

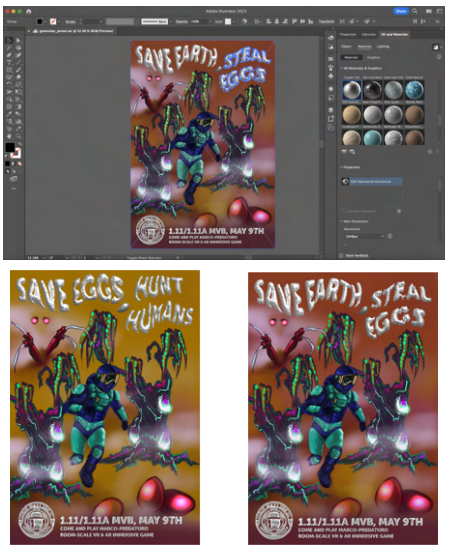


Figure 21: Poster design in Illustrator, and final 2 outputs

As with everything else, the asymmetric nature of our gameplay demanded a different track for each side. However, we also had to consider the cohesion between the two sides' music and understand that the differences between them would be highlighted as it is one of the few things both sides have in common.

7.10.3 Room planning and Queue organization

Our Azure Kinect cameras are particularly vulnerable to minor disturbances, which could compromise the Body Tracking and synchronisation: key features in the success of our flagship technologies. It is therefore crucial to effectively manage the room organization for Games Day. After spending countless hours in our given room 1.11/1.11A we decided on the plan in Fig 7.10.3. This ensured a smooth flow of players while keeping all our tech safe from mishaps, and spectators out of view of the Azure Kinects.

7.10.4 Visual Merchandising

To accompany the game, it was important to create graphic logos and posters that communicated the visual imagery and reinforced the look and feel. Doing this built a coherent and united brand.

- *Logo:* The concept for the logo included elements from both game sides, Marco's claws and Polo's space helmet, in one simple graphic image. This was conceptualised and sketched in iPad App Sketchbook, and then edited into an inflated chrome effect in Adobe Photoshop: Fig 20.
- *Posters:* To draw people to our game, we adapted and added to previously drawn concept art in Sketchbook, and then assembled with text in Adobe Illustrator and Photoshop. We made 2 versions of the poster, to allude to the duality of gameplay; there are two sides to every story, Fig 21.

7.10.5 Introduction Video

To set the scene for our game, we needed to make a video that was engaging and informative, communicating key game missions for each player. After mood boarding, we compiled existing gameplay clips and recorded extra video to convey the game backstory. To make the extra video clips more engaging, we used Corridor Digital's Stable Diffusion method to convert them into sci-fi, anime-style animation. Stable Diffusion is a generative art tool. To do this we trained a Dream booth model[10] on specific subjects and art style. And then used this alongside Stable Diffusion[**?**] to convert the video frames into animation. Finally we applied various post-processing steps, including manual deletion of bad frames, and rotoscoping[6] to cut out the subject of the video and place them on a pre-generated background.

After collating many clips of playtests, team work and Marco's viewpoint. We combined these with the animated video, and added composer-made music, to communicate gameplay and its development. This ultimately conveyed game story and experience well.

7.11 Modelling & Animation

Modelling and Animation is a crucial aspect of game development, as it serves as the foundation for creating immersive and visually engaging experiences. We gave this aspect of our game a lot of attention from the initial vibe board to actually modelling the assets.

7.11.1 Virtual Scene: Design

Before modelling, we came up with thoughtful concept art that aligned with our Game story and lore. This stage involved brainstorming and refining ideas which we captured on the Miro Board. We used Stable Diffusion models like Midjourney, DALLE and Lexica to spark ideas for the general atmosphere, and used these

to inspire our own custom concept art created by digitally hand-drawing: Fig 22. We conceptualised an

Figure 22: Alien tree custom-drawn concept art

alien forest clearing with trees inspired by custom concept art; they have an otherworldly shape and glowing orbs embedded in the bark. The atmosphere is eerie and has a fog particle system to add to the ambience. Once we settled on the functionality and had the design ideas in place, it acted as the groundwork for the modelling in the next stage.

7.11.2 Virtual Scene: Modelling

Once we knew our desired vibe we began the modelling process. This involved countless hours trying to understand how to use Blender and its features in the most efficient way. Throughout our different playtests there was an improvement in our models as we became more proficient with Blender.

All this is situated inside an alien forest clearing with trees inspired by custom concept art; they have an otherworldly shape and glowing orbs embedded in the bark. The atmosphere is eerie and coloured fog obscures vision.

7.11.3 Virtual Scene: Texturing & Animating

To help the look and feel of our game we used a shader graph to create some basic shaders, in Unity's shader graph builder, that animate our scene.

1. The first shader makes the leaves on the trees move slightly as though there is wind. This was done by moving through Gradient Noise based on the current time of the game and applying this noise to the top x component of the mesh in world space, making it wiggle around but stay in the same y and z position.

2. The second shader adds emissive patterns to the eggs, giving them an alien glow. This is done by moving through the Gradient Noise, as before, but then applying it to the emission component of the material.

3. The final shader adds patterns to the trees, to make them feel textural and dangerous. Noise is animated in the same manner as before, and is used to animate a rounded rectangle. This creates patterns which are then applied to the emission component of the material.

These shaders worked without fault in the body tracking app, which is used to render the Polo minimap (Section7.5.3).

However, due to issues with compatibility across Android, Unity versions and rendering pipelines, discussed in Section 6.1.1, we had difficulties implementing them in the Metaquest.

After days of trying and failing to find a solution, we ultimately resolved to use shaders without shader graphs on the VR side. This meant the Marco side still has the same colours, but unfortunately no animated elements.

7.11.4 VR: Particle Systems

We worked on other game aspects, such as adding a fog-particle system, to build the same ambience in different ways, such as adding more depth and atmosphere to the game.

The fog system works by emitting particles that are rendered as small, semi-transparent sprites. We imported a cloud image and applied it to each sprite to achieve a natural look. We then manipulated these particles by changing their size, speed, colour, and opacity properties, creating a cloud-like appearance. To achieve our desired fog effect, we also had to experiment with different emission rates and maximum numbers of particles. This was especially important due to the hardware limitations of the Oculus; too high of an emission rate or too many particles caused serious lag, which impacted the player experience. We found a balance by reducing the emission rate and the number of particles but increasing the particle size.

We attempted to apply post-processing, discussed in Section 7.11.5, to the VR side however due to the way the VR camera system is set up, it did not work well. We therefore decided to implement another particle system, to enhance the VR in other ways. We created a Blood Splatter particle system, that is triggered every time Marco collides with Polo, to improve Marco's gameplay experience: Fig 7.11.4. This was implemented similarly to the fog system, by importing an image with a blood splatter effect, that was applied to particle sprites. We tweaked some parameters of the system such as emission rate, particle size, speed and angle to achieve a more realistic blood splatter effect. We added some custom code to the HapticInteractable script, discussed in Section 7.8.1, to instantiate the blood splatter when a collision occurred. The particle system is then destroyed after 1.5secs using Destroy().

7.11.5 Virtual Scene: Post-processing

We employed some of Unity's built-in post-processing packages, to polish our game visuals and enhance the feel.

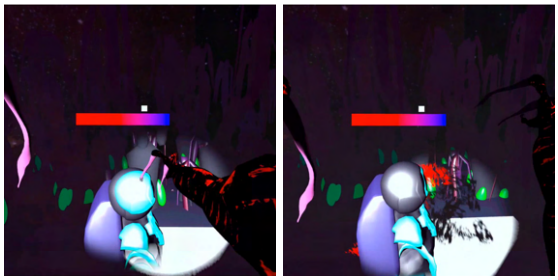


Figure 23: Blood Splatter effect visual

To do this a post-processing layer, and PostProcessingVolume component was added to the main camera in the scene. The post-processing layer serves as an interface between the camera and the post-processing system, while the PostProcessingVolume defines the settings of the effects applied.

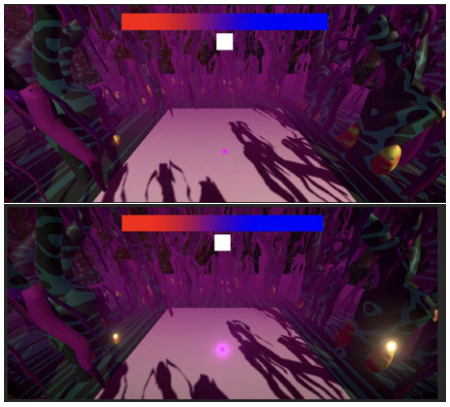


Figure 24: Virtual Scene before and after post-processing applied

Of all Unity's post-processing effects, we found the following to work best with our game. These can be seen in effect in Fig 24.

1. *Bloom:* Creates a glow effect around bright areas of the scene, adding to realism and immersion.

2. *Vignette:* Fades the corners of view, creating a sense of claustrophobia and thus heightening the fear factor.

7.11.6 Marco & Polo model: Design

Lots of research was put into the design for Marco; we ultimately wanted to achieve a design that could be both scary and loveable, in keeping with the duality of our game story. Given this was a difficult aim, we worked cyclically, iterating over research, prototype, and evaluation steps. We researched egg-laying animal kingdoms and drew concept art of a monster based on each. We settled on insects, as we found them to be the most unsettling. We then researched different types of carnivorous insects, settled on mantises, and brainstormed concepts from there. This eventually brought us to our final Marco design, wherein mantis heads and claws have been reimagined; combining elements

Figure 25: Early design process sketches for alien Marco

of reptilian and dinosaur anatomy to achieve an otherworldly design.

For Polo, we made use of a similar design loop. We looked at different types of helmets, space suits, boots, and gadgets such as lamps, in order to conceptualise a dynamic and futuristic space person. We collated our results and drew concepts based on them.

7.11.7 Marco model: Modelling and Texturing

We made intricate models for Marco's claws and head in Blender, using our detailed concept sketches as a starting point. The shaders used to texture Marco's claws are discussed in depth in Section 7.4.2, as they tied into a crucial part of Marco's gameplay: his 'torch'.

7.11.8 Polo Model: Modelling and Rigging

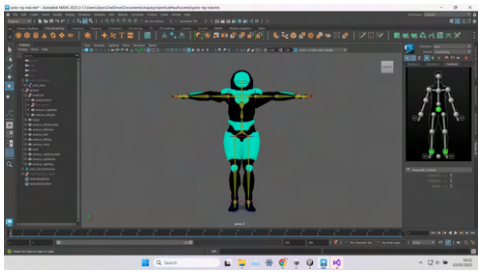


Figure 26: The final polo model being animated in Maya

The Polo models emulate futuristic astronaut spacesuits. Once we had the initial designs drawn up by our team, we started making very basic models with simple shapes. These eventually evolved into slightly more intricate models following tutorials online. Once we had the model, modelling and rigging were done in Maya. This was done by taking the mesh we had modelled, binding it to a skeleton, and manually painting the influence weights of each bone onto the mesh to make it into a "skin". Though binding the skin to the skeleton was arduous, due largely to our inexperience and poor documentation, the real challenge came in

figuring out how to synchronise an animation driven by the skeleton generated by the Azure Kinect.

7.11.9 Polo Model: Synchronised Kinect-Driven Animations

Synchronising our Kinect-body tracking to our Polo model was one of the key challenges in our development process. We felt it was hugely important that Polo was animated to showcase our flagship body mapping and to improve Marco's overall experience.

While there exists a Photon solution to animation in the Photon Animator View, it is only capable of synchronising Mecanim animations, which are controlled by an Animation State Machine. Our animations come from the Azure Kinect skeleton rather than states. Ergo, though we could animate models locally, we were unable to synchronise animations across the network without a different solution. Discovering this took time, but we worked efficiently to find a solution once we were aware of the issue.

Among the brainstormed solutions was creating a system of states that would be detected from the angles and heights of bones in the Kinect skeleton; or driving animation solely by the position and orientation of the head. However, the solution we ultimately committed to was to create a structure to store the orientation of each bone in the skeleton and serialise that. Because each bone is parented by another in the skeleton, save for the root bone, serialising positions were unnecessary.

Custom code was then written to unpack (and pack) this structure and drive animation through Unity's own Animator component using the Azure BodyTracking SDK.

There were further issues with the VR side receiving these animations. First, the head and armour of the model would stay stationary in the centre of the room, rather than moving with the rest of the model. After much exploring and testing, we found several bugs contributing to the effect:

While the position of the skeleton was being successfully synchronised, the position of the model as a whole was not. We found applying the root motion of the skeleton onto the animation was not enough for the Marco side to work, because of its dependence on the geometry of the models rather than their skeletons. To solve this, we used the synchronised position of the "pelvis" bone generated by the Kinect; this transformed the model as a whole.

Though that corrected the position of the armour, it did not fix the orientation of each of the pieces or the head. This turned out to be due to us not exporting constraint data from Maya. After remedying this, we were able to successfully animate the Polo model on the VR side.

7.11.10 AR Datapad: Wrist strap/3D printing

In order to make the phone feel as natural and unencumbering as possible, we decided to strap it to the player's forearms. This also served the look and feel in making it feel more sci-fi, drawing from references like *Predator's wrist gauntlet*[7].

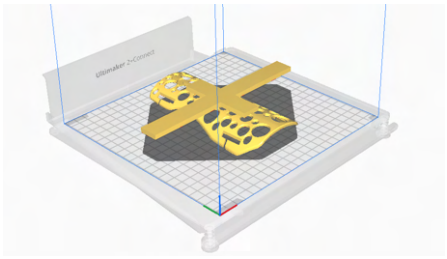


Figure 27: Final model of the wrist brace in the Ultimaker Cura software

We went through 4 iterations of this before we decided on our final design. Our initial prototype was made of cardboard and duct tape to test viability. Once we tested this in our playtest on Schools Day, we moved on to 3D printing this to have a more robust solution. We prototyped different 3D models with a phone holder on a sci-fi arm brace so we could add straps to them: Fig 27. After a couple of failed attempts, we simplified our design to simple support with a superglued phone case mounted on top and wrist straps attached.

7.11.11 AR Datapad: UI Design & Modelling

To build the sci-fi, futuristic feel of the Polo side, we wanted the UI of the AR phone to have a fun datapad, 'tech-hud' look. We collated mood boards of inspiration images and made the 2D elements in Adobe Photoshop: Fig 28. These were then imported to Unity as 2D sprites, and used to texture Canvas Images.

Figure 28: AR UI mood board and Photoshop development

References

[1] Azure Kinect Body Tracking SDK download. https://learn.microsoft.com/en-us/azure/kinect-dk/body-sdk-download, sep 2 2022. [Online; accessed 2023-04-24].

[2] Azure-Kinect-Samples/body-tracking-samples/sample_unity_bodytracking at master · microsoft/Azure-Kinect-Samples. https://github.com/microsoft/Azure-Kinect-Samples/tree/master/body-tracking-samples/sample_unity_bodytracking, jun 3 2022. [Online; accessed 2023-04-24].

[3] How to wire and program a button. https://docs.arduino.cc/built-in-examples/digital/Button, apr 26 2023. [Online; accessed 2023-05-03].

[4] Amsler, S. Rfid (radio frequency identification). https://www.techtarget.com/iotagenda/definition/RFID-radio-frequency-identification, March 2021. [Online; accessed 2023-05-04].

[5] Brent-A. Azure Kinect Sensor SDK download. https://learn.microsoft.com/en-us/azure/kinect-dk/sensor-sdk-download, sep 21 2022. [Online; accessed 2023-04-24].

[6] Fleischer, M. Method of Producing Moving-Picture cartoons. https://patents.google.com/patent/US1242674A/en, oct 9 1917. [Online; accessed 2023-04-24].

[7] Gordon, L., Silver, J., Davis, J., Thomas, J., Thomas, J., Silvestri, A., McAlpine, D. M., and Winston, S. *Predator*. 20th Century Fox, 1987.

[8] Patrão, B., Pedro, S., and Menezes, P. How to Deal with Motion Sickness in Virtual Reality, 2020.

[9] qm13. Azure Kinect DK hardware specifications. https://learn.microsoft.com/en-us/azure/kinect-dk/hardware-specification, may 23 2023. [Online; accessed 2023-04-24].

[10] Ruiz, N., Li, Y., Jampani, V., Pritch, Y., Rubinstein, M., and Aberman, K. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. https://arxiv.org/abs/2208.12242, aug 25 2022.

[11] Wrona, M. Roll and pitch angles from accelerometer sensors. https://mwrona.com/posts/accel-roll-pitch/, nov 27 2020. [Online; accessed 2023-04-24].