Group Big Tasty

University of BRISTOL

Somnia

By
Brooke Ayers Evans
Erich Reinholtz
Jacob Chalk
Thomas Raybould
Finn Wilkinson
Owen Coyne
Haris Kountouris

Signed Members Page

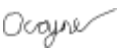
Brooke Ayers Evans
Weighting : 0.8
Roles : Team Manager
Signed : 

Erich Reinholtz
Weighting : 0.8
Roles : Modelling and Effects
Signed : 

Jacob Chalk
Weighting : 1.5
Roles : Lead Programmer
Signed : 

Thomas Raybould
Weighting : 1.5
Roles : Programmer and systems developer
Signed : 

Finn Wilkinson
Weighting : 0.8
Roles : Research, Modelling, Mini Systems Dev
Signed : 

Owen Coyne
Weighting : 0.8
Roles : Story, Game Design, Testing
Signed : 

Haris Kountouris
Weighting : 0.8
Roles : Audio Research and Modelling
Signed : 

Top Ten Team Contributions

1. We created 35 models in Maya from scratch, along with 5 custom textures created in GIMP.
2. We integrated our game in the Unity Game Engine along with their built-in physics engine.
3. We wrote 4,000 lines of source code. Including C# scripts and HLSL shader code. This was designed and optimised to be as efficient as possible, as VR places extra demands on the GPU due to twice the draw calls to render a scene (Once for each eye lense).
4. We designed our entire game dynamic to suit the horror theme. Planning out elements such as: story, suspense building, planning the layout of the map and scripted scare events. The map was planned to match certain story/gameplay elements and then created in the Unity editor. The suspense/story element needed to be carefully planned as the short 10 minute window meant that the traditional suspense building element of horror has to be condensed down, but still effective.
5. We integrated a monster patrol system, using Unity's built-in path finder. This was combined with a ray-casting field-of-view system, this allowed the monster to "chase" the player when they are in view (e.g. not occluded) and within a certain radius of the monster. We added in editor scripts which helped to visualise the monster's range and field-of-view in the Unity editor.
6. We used Unity's built-in Occlusion culling system, this is once again for the sake of efficiency, as occluded models would no longer be rendered. This drastically reduced the draw calls made for the large map and considerably improved performance.
7. We created our own particle system. This was then to be used on the monster for effect.
8. We coded an arduino in order to add a 4D element to the game via creating "chills" using a fan. This was initially planned to be used on Games day.
9. We rigged a humanistic model in order to create a "rag-doll" effect. These bodies would then populate the map and even be used in a scripted event to create suspense and unease.
10. We designed an entire framework of movement, interaction and action performing with the HTC Vive. The only library was SteamVR which was used to read input from the Vive only. Everything else was coded and implemented by us. We implemented features such as how button presses change action depending on the player's state e.g. are they holding an object already? Or trying to grab an object currently? This framework extends to an easily extendable interactable objects set. This follows OOP principles and allows different objects to be easily added, which will inherit and override parent class functions to create DRY code.

Categorised Team Contributions

1. Professional team communication and problem resolution

· As a team we made sure we met regularly to ensure that everybody was on the same page and that everyone could continue with their individual tasks.

· Our team leader contacted relevant members of the faculty in order to ensure we had the relevant material to overcome any problems and that we had all the equipment available to enhance our gameplay.

· Jira and Slack groups were created to keep track of tasks.

2. Technical Depth and Technical Understanding:

· During the development cycle we had to write about 4000 lines of code in C#. These include, augmenting or editing existing unity scripts and creating our own movement from scratch to match the needs of our game.

· We were able to extract data from a heartbeat sensor over Bluetooth.

· Coding an Arduino to communicate over WIFI.

3. Technical Implementations, Solutions and achievements delivered:

· Our movement and interaction framework gives the player an immersive experience. which makes cases of motion sickness rare as we have heard and applied suggestions. from multiple members of the panel who had tested the game. Additionally, it has been thoroughly tested to be extremely smooth and bug free.

· We created most of the models used in our game from scratch in Maya along with 5 textures in GIMP.

· We used occlusion to stop the engine from overworking the CPU, which made the game more efficient and allowed us to improve the quality of our models.

· We created a particle system to give our monster a mysterious yet hunting appearance.

4. Professional Development and Testing Cycle using Professional Tools:

· Using Jira, we were able to assign individual tasks and note their significance.

· We split into smaller groups to make our progress more efficient while also following agile development to make sure that the pieces each team was able to create work as intended in the project.

· We followed the OOP principles and created class diagrams to allow everyone to follow and contribute to the project since it was easy to add more features or edit the existing ones.

5. Game Playability, Interactivity and Controls

· At the start of the game there is a tutorial area to teach the player the controls and how to interact with the world. This area can also help the player get familiar with the physics of the world. For example, how far the player can throw objects or how fast the player can move.

· The controls are extremely easy and simple to learn and mimic real movements when possible. For example, to grab an object, you just have to press the trigger on the back of the VIVE controller which is how you would grab an object in real life.

6. Look & Feel, Graphics Made/Generated, and Physical game

· We created 35 Models in Maya from scratch as well as 5 textures in GIMP because we had to make sure that the objects in our world fit the theme and the atmosphere we were trying to create. This also gave us the opportunity to create objects in our world which were easily interactable and gave a realistic feeling when handled by the player.

· We created a story and a world which constantly puts the player “at the edge of their seat”. Carefully planned jump scares and suspense constantly building up were what we constantly had in mind when we designed every aspect of our game.

· Sound is extremely important in horror games, so we explained our vision to the producer carefully and made sure that the soundtrack can deliver the atmosphere we were looking for.

7. Game Novelty and Unique Product Aspects

· During games day we would have used the hackspace to further immense the player into the experience before even putting on the headset by dimming the lights and recreating a hospital-like smell.

· We developed an Arduino which during games day, would have switched on fans directed at the player remotely, depending on in-game scenarios.

Abstract

Somnia is a virtual reality horror we have been working on. The game sets up players in the role of an unknown character who suddenly wakes up on a hospital bed. The player must then explore a seemingly decrepit hospital, marred throughout by graffiti, ruin and the effects of unexplainable forces. Through exploration of this hospital they will come to find out more about the nature of the hospital and what has been happening there. However as they progress further through the game things will start to seem more out of touch with reality and the presence of a monster formed from mist begins to become more apparent.

The crux of our game design was a series of encounters with the mist monster, slowly building tension, which lead to a final climax where the player must escape. The initial room is set up as a typical hospital ward and is designed to allow the player to become accustomed to the controls and setting without any danger. They then progress through a hallway with a flickering light, which breaks as they walk towards it, forcing them to use the flashlight they have just collected. This leads to a reception where our first encounter occurs, the mist monster can be seen in the window to another ward as the player approaches it. The window then shatters forcing the player to flee into a side door which locks behind them. They then pass through a ward where a machine emanates a flatlining sound and a body shape can be seen in one of the hospital beds. After the bed is passed the machine begins to report a pulse and the body is gone.

Leaving this room through a different exit the player will be confused to find themselves back in the same reception room they just left, however the conditions of the room have changed. This is one of the first instances of the game mechanics being used to warp the players sense of location and trust in the game. The objects throughout the room have been scattered and a path of destruction caused by the monster will be visible in this and the next room they enter (hospital beds strewn about, marks on the wall etc.). From here the player will enter a corridor which closes up behind them as they turn the corner. This is followed by the window at the far end of the corridor, that they have to pass, beginning to emanate mist.

Past this corridor we planned to have a large atrium with a few small areas such as shops and a security room. This would be supplemented by a lot of structures within the atrium such as fountains and pillars that provided “cover” for the player. This room would be the setting for the climax of the game where the player passes through the atrium only to find the exit door locked requiring a keycard. When the player turns around to find the keycard, the objects in the atrium have been disrupted, as in previous rooms, and the monster is in full view. The final gameplay segment would require the player to navigate to the security room and retrieve the keycard, however the monster would be constantly patrolling the room, obstructing their path and often directly chasing the player. This is the only point in the game in which the monster poses a direct threat and so really ramps up the tension. After collecting the keycard the player can escape to the safety of the corridor past the door they were trying to exit.

This escape would lead to a door flooded with white light from inside, when the player passed through they would “wake up” in the same hospital bed as the beginning in a clean and new looking hospital ward.

Our goal was to represent the decrepit hospital as facets of the characters subconscious during a stay in a real hospital. The ambiguity of the “mist” monster and the increasingly abstract layout of the hospital would also lend itself to this interpretation. Our plan was to also scatter throughout the game more notes and objects that would hint at this outcome if inspected.

The game is played with the HTC vive virtual reality headset, with which the players will be able to look around and freely navigate the environment. This was key to the design of the game as the immersion provided by the headset allowed us to build an intimate horror narrative that really ramped up the tension throughout. The player could move around within the game space using both physical movement (within the bounds of the play space) or the trackpads on the HTC vive controller, this allowed us to cater to player’s tastes and alleviate some players tendency towards nausea in VR games. The controllers could also be used to interact with the two key genres of objects present in the game:

- Objects which were used to interact with the environment or were needed to complete sections of the game (e.g. doors, flashlights and keycards).
- Objects scattered around the in-game environment that can be inspected to gain a wider understanding of the background and story of the game (e.g. doctor’s notes or small significant items)

In the case of objects which needed to be repeatedly accessed throughout the game, such as the flashlight, we implemented a belt-like inventory system where the player could store and retrieve objects throughout play.

The Team Process and Project Planning

During the course of game development we implemented a number of different strategies to aid collaborative development which I will discuss in this section.

We first created a team leader role that was held by Brooke throughout the process of game development. She made the majority of executive decisions about how the development process should run. Any decisions made by the team were conducted by a vote system and the final decision was chosen by Brooke.

Initially Brooke created a slack channel for us to stay in contact and discuss the technical aspects of the game with each other. We had different channels for different sub-groups within our team. We also created a facebook messenger group to discuss more informally and ensure everyone knew when meetings were occurring. When a member couldn't attend, we would message a brief overview of the meeting and where we were heading next.

We decided to give the role of Lead Developer to Jacob who made all the executive decisions related to the code. He decided how certain things should be implemented and ensured all code was up to standard. This allowed us to produce clean and efficient code for our game. Jacob decided that a pair programming system should be implemented for our development process and helped organise these groups with Brooke.

During this process, we made full use of Github to share and collaborate on each task. This also allowed us to have multiple branches at once, one of which was a main development branch for us to merge all changes that worked together after completion and a master branch which was only ever pushed to when we had a fully tested and working implementation at a particular stage in our project. This allowed us to progress freely when we could and have a backup version that we knew worked correctly if our game was to be shown to a staff member at any particular time.

We also decided to use Jira during the development process to dedicate tasks. Brooke was in charge of ensuring the tasks were up to date and dedicated to the correct team groups.
We met weekly as a group which was organised by the team leader to discuss what tasks we had completed during the week and what needed to be completed next. This often involved an agenda and taking meeting minutes which were then added to Jira as tasks after the meeting was finished. This weekly meeting allowed members to express their concerns frequently so that if anything was falling behind, team members could be moved around to assist in certain tasks that were of a higher priority.

The majority of work during this development was undertaken as a pair programming task. This mainly involved one member primarily writing the code while an additional team member would actively search code for any errors while the main member was typing. In addition to this, often a third member of the team would be present to physically test the game during the motion control development stage. This allowed for issues to be found and fixed quickly before progressing onto the next task.

We initially set lots of fixed deadlines for each subtask to be completed within but found these were too tight and struggled to reach them. Throughout the whole development process, we found estimating workload and time durations for each task to be difficult which did present with issues with workload not being evenly distributed at each stage.

We then implemented a different system where we ranked each task as a different priority on Jira and decided to delegate groups to tackle the highest priority tasks first. Once these were completed we could then move onto lower priority tasks such as modelling or texturing objects. Without having strict deadlines anymore, just a general strategy of each week the highest priority task was to be completed, we managed to stay on for the majority of the year.
After returning in January, we found that we had fallen behind in progress after the Christmas period so decided to have two weeks of intensive development sessions to produce a working prototype ready for feedback.

During development, we ran into some issues and to further explain our problem solving process we have discussed one problem we encountered below in detail.

A problem we discovered after feedback from the panel of markers, was that there were varied opinions on how our locomotive system worked. Some found it nauseating and unrealistic whereas other members of the panel found it fitted the style of our game well and was better than most options for motion sickness. To solve this issue, we decided to research and create a list of other methods we could use for movement within our game. We then decided to test each method and create a pros and cons list together as a team. We took our concerns to a lecturer, Carl Henrik to seek further ideas and deeper opinions about what we had discussed. We then shared further opinions as a large group and ranked the methods together in order of preference. We each voted on a primary method and a secondary as we believed that offering two different locomotive systems the user can choose between was beneficial to all players. One method was more involved in the game and allowed a greater range of movement and another which offered sitting down for those who struggle with motion sickness. Throughout this whole process, all members' opinions were equally considered at each stage to reach a conclusion that satisfied all.

During the whole year we found that working physically together provided a much better coherency between all parts of our project. We were able to work together for the majority of the year in the upstairs of MVB. This allowed us to have some team members working on the

look and feel of the game as the mechanics were being developed so they could be seen alongside and adapted as needed.

Unfortunately, during the covid-19 crisis, we have been unable to work as a team anymore as the equipment we use is located in MVB and we have all since moved home. As our equipment requires a large space to test, we were unable to continue developing gameplay any further. We have been since using messenger and Google Docs to collaborate to complete our report on time for submission.

If we were to complete this development process again, there would be a number of things we would do differently which I will discuss below.

Firstly, we would have spent more time in the initial stages mapping out which parts of the project needed to be completed. We would use rough time estimates to distribute work more evenly between members. There were many points within our project where some people had too much work to do and others not enough resulting in an imbalance of workload. This was due to us not discussing workload estimates enough and not reviewing these each week in case the guess now seemed inaccurate. We stuck mostly to a solid system of two people groups working on a task which did not work as effectively as we expected.

Secondly, we would utilise slack more as we found that by the end of our project, Slack was not used very often by the group. Certain members would have smaller group communication with each other which often resulted in some development points not being shared with all members immediately. Slack would resolve this as all members would have access to all channels and could keep on top of any issues or changes.

Thirdly, we found that not many members were using all aspects of Jira fully so Brooke found it difficult to track what work was being completed by whom and when it was finished by. This also contributed to the imbalance of workload at times as she was unable to follow where exactly we were in the development pipeline. Ideally, Jira should’ve been used to its full extent to track hours spent on each task for each member.

In conclusion, we believe that we have worked well as a team throughout this whole process. All members engaged with the project but we did struggle with workload imbalance which we have discussed post development for ways we could fix this if we were to complete this again.

Individual Contributions

Brooke

Team leader role throughout

Initial admin setup - 2 hours
Organising and overseeing initial meetings to discuss ideas - 4 hours
Creating Slack and their channels - 1 hour
Creating Jira - 1 hour

Weekly Roles
During the entirety of the project I was present at almost all development team meetings/programming sessions where I undertook a number of different activities which included: *Roughly 10-14 hours weekly*

- Offering advice or answering questions about aspects of the game
- Approving/aiding with development of ideas for the game
- Aiding with debugging/coding
- Assisting with gameplay testing
- Assisting with final decisions on game mechanics or implementation ideas

Updating Jira, Liasing with staff members and industry mentor, organising weekly meetings - *1 hour weekly*
Aiding Owen Coyne with continuous development of story and map design for game - *1 hours weekly*
Weekly team briefing meeting - *2 hours weekly*

Individual Tasks
Creating an assortment of models for game: *3 hours*

- Battery
- Fountain
- Tray
- Desk

Creating an assortment of textures in GIMP for game: *6 hours*

- Ceiling tiles
- Multiple Flooring textures
- Multiple Wall textures
- Chair fabric

Aiding investigation into alternative locomotion/room scale VR - *5 hours*
Researching into tech we would need to produce the game and sorting collection - *5 hours*
Research into ethics surrounding use of heart rate monitor, ethics applications, meeting with Conor Houghton to discuss ethics application - *3 hours*
Organising and meeting with Carl Henrik to discuss game mechanic options and graphics - *1 hour*
Research into heart monitor usage - *10 hours*

Erich

Roles: Modelling, Research, Visuals

Individual contributions:

1. Research

Investigating different real life environments and setups that would provide useful context for our game. I selected elements that could be relevantly integrated within the game, to enhance the overall horror experience. Such elements include: objects present (collectible or non-collectible), materials for the surroundings, etc. – *2 hours*

2. Modelling

Taking part in the production of our custom-made 3D models. For the design of the geometry, as well as for texturing, I made use of Autodesk Maya and Arnold Renderer. Examples of models include: hospital beds, desk with retractable drawers, the head and face of the monster. – *9 hours*

3. Interactions

Experimenting techniques, in Unity, for creating manipulability in terms of the location and motion of certain components of my models. For example, allowing the drawers of the desk model to move backward and/or forward, in order for the player to be able to control its state during gameplay. – *3 hours*

4. Developing effects

Exploring, creating and testing visual effects for the monster entity in Unity. Investigated different emitters, lighting schemes, variations of colours, surrounding materials, etc. – *8 hours*

Weekly contributions:

- Was present at all group meetings, constantly providing suggestions and feedback. I got involved in particular details surrounding the game and tried to contribute to resolving as many issues as possible. – *2 hours weekly*

- Kept up to date with and understood every teammate's part, from story writing all the way to coding. – *1 hour weekly*

- Was involved in testing sessions for different details of our software (i.e. collider functionality), as well as the overall gameplay. – *1 hour weekly*

Pair work contributions:

- Contributed, along with Finn, to the research and development of our own custom made Particle System. Provided aid in comparing different methods and techniques, investigating ways of optimisation, as well as debugging and testing. – *7 hours*

Jacob

Features further decomposed if faced with significant issues/errors. Features that are not further broken down still include hours for areas such as: initial set up, add-ons/improvements to the stated feature and debugging/error fixing.

Roles: Lead Programmer. (Majority of the hours stated below were spent paired programming with Tom Raybould)

Game features

- Setting up Steam VR/adding additional controls - 2 hours
- Movement - 20 hours total
    - Initial set up - 8 hours.
    - Add ons after set up: gravity - 4 hours
    - Debugging/Testing alternative methods - 8 hours
- Object interaction - 16 hours total
    - Basic interaction setup - 4 hours
    - Add ons after set up: Socket Inventory - 4 hours
    - Error fixing such as held orientation - 8 hours
- Object functionality e.g. Flashlight and phone - 8 hours
- VR-friendly doors - 24 hours total
    - Initial setup - 4 hours
    - Debugging and error fixing - 20 hours
- Setting up and baking occlusion culling - 4 hours
- Door Portal - 12 hours total
    - Initial setup - 2 hours
    - Debugging - 10 hours
- Tutorial - 4 hours
- Monster patrol system along with FOV - 8 hours
- Creating and timing scripted events - 8 hours
- Optimising particle system - 8 hours
- Optimising general areas of code/reformatting to adhere to DRY code principles - 16 hours
- Organising the Unity physics layers to work with map code - 4 hours
- Setting up test player to work without VR headset - 4 hours
- Menu (e.g. scene transitioning and VR point) - 8 hours
- Fire Exit sign highlighting key areas to impose some linearity to the game - 2 hours
- Interactable hospital curtains - 4 hours

Map

- Room creation and object placement - 8 hours
- Lighting - 4 hours
- Basic Unity 3D sound around the map e.g. light flicking, ambient noise, ECG, phone call etc. - 4 hours
- Small tweaks to design/room layout of map for more coherent gameplay - 2 hours
- Materials - 2 hours

Modelling

- Modelling 17 in-game models - 20 hours
- Modelling and rigging humanoid model and setting up ragdoll physics in unity - 12 hours
- Retopologising high resolution models (ECG machine, Doors, etc.) - 8 hours

Restructures/reformats

- General code reformatting for readability - 12 hours
- Map restructure after initial panel showing - 8 hours

Shaders

- Vignette shader - 4 hours
- Custom glass - 2 hour
- Camera-texture-to-plane - 2 hours
- Arranging and meeting Carl Henrik Ek to discuss shader implementation - 1 hour

Testing

- Testing implemented features for functionality and stress testing for special case scenarios - 12 hours
- General “glitch-proofing” not specific to implemented features e.g. ensuring a player can’t walk through walls - 8 hours

Other

- Bindings UI breaking. Considerable time spend finding and then fixing the bug (Required setting up the project again) - 8 hours
- Creating UML diagrams of the framework in order to make it clearer to the group on how to extend the code - 2 hours
- Research into optimisation techniques - 4 hours
- Profiling and analysing code to diagnose problem areas - 4 hours
- Research into shader methods and learning HLSL - 4 hours

Weekly:

- Meetings and discussions with programming partner Tom Raybould to evaluate progress, improvements that could be made and current issues/bugs that need addressing - 2 hours

Total time: 285 hours for individual tasks + (2 * X for weekly hours, approx. 24 assuming a 12 week process)

Finn

Particle System

- Research into implementation and relevant physics - 8hrs
- Particle system creation - 25hrs
- Implementing into main Unity Project - 2hrs
- Debugging - 10hrs
- Texture research, creation and implementation - 4hrs

Heart Rate Monitor

- C# Bluetooth communication and data retrieval research - 10hrs
- Heart Rate monitor app creation and debugging - 10hrs

Research

- Alternative user inputs such as treadmill to use as movement controller - 5hrs
- Possible graphical methods such as real-time ray tracing - 12hrs

Modelling

- ECG machine - 6hrs
- Mock body with draped cloth - 3hrs
- Starting room and first corridor - 6hrs

Misc

- Initial HTC Vive setup on Desktop and using in Unity - 5hrs
- Curtain physics and trials - 2hrs

Weekly

- Attend weekly team briefings - 1hrs
- Catch up with core game development and other systems progress - 1hrs
- Gameplay testing - 1hrs

Tom

Majority of the hours stated below were spent pair programming with Jacob Chalk

Mechanics and Features

- Object Interaction - 8 hours
    - Object Interaction Framework - 4 hours
    - Socket system for holding/storing objects - 4 hours
- Scripted Gameplay Events - 24 hours
    - Framework/Base for scripted in-game events - 4 hours
    - Writing individual scripted events - 20 hours
- VR Controls - 18 hours
    - Movement controls - 8 hours
    - Testing movement options (room scale, seated, alternative controls) - 8 hours
    - Object interaction controls - 2 hours
- In-game Phone controls/scripts - 5 hours
- Flashlight controls/scripts - 5 hours
- Vignette Shader (Intended to be used with heart rate monitor) - 4 hours
- Interactive curtains - 10.5 hours
    - Investigating Unity cloth, methods for implementing interactive curtains with vr - 2 hours
    - Setting up and writing scripts for cloth interaction with player - 5 hours
    - Customising standard shader to prevent backface culling on curtains - 0.5 hours
    - Testing and bug fixing - 3 hours
- Doors - 24 hours
    - Initial setup/writing scripts - 4 hours
    - Debugging/testing - 20 hours

Map

- 3D modelling objects - 1 hour
- Making prefabs of objects - 3 hours
- Materials - 2 hours
- Map design - 2 hours
- Room building and object placement - 10 hours
- Lighting - 4 hours
- Materials - 2 hours

Optimisation/Performance

- Researching ways to improve code performance - 2 hours
- Rewriting unoptimised scripts - 10 hours
- Using the profiler on builds - 10 hours
- Controlling curtains to prevent constant cloth physics - 5 hours

Testing

- Testing and preventing glitches caused by multiple colliders from the VR Rig - 10 hours
- Gameplay testing - 20 hours

Misc Tasks

The following are time estimates over the entire project

- Setting up and managing the project repository - 2 hours
- Fixing merge errors from other people's branches - 4 hours
- Steam VR bindings UI broke at start of project, finding the issue took a considerable amount of time - 8 hours

Owen

Research

- Map design and game structure - 5 Hours
- Game Design fundamentals - 10 Hours
- Unity audio and Audio Spatializer SDK - 4 Hours
- Horror game genre and player psychology - 4 Hours
- Efficacy and implementation of cutscenes in VR - 2 Hours

Game Design

- Developing consistent aesthetic for the game world - 3 Hours
- Map design - 15 Hours
- Design and choreography of game encounters - 20 Hours
- Story writing - 5 Hours
- Test work on unity audio spatializer SDK - 4 Hours

Weekly Roles

- Gameplay testing - 2 Hours weekly
- Bug checking - 1 Hour weekly
- Communicating game design and story choices to the team and adapting them based on group discussion (with Brooke Ayers Evans) - 2 Hours weekly
- Guiding model design - 1 Hour weekly

Misc Tasks

- Liasing with the game's composer and working in tandem to help shape an appropriate game soundtrack - 5 Hours

Haris

Modelling

- Flashlight: 5h
- IV stand: 5h
- Doors: 6h
- Fixing models to fit game requirements: 2h

Research

- Finding suitable Audio SDK and implementing Steam Audio SDK: 20h
- Switch a plug on remotely using arduino : 10h

Weekly

- Attending team meetings - 1h
- Integrating into development - 1h

Software

The Gameplay

When a member of the panel runs our game, there is a numbered amount of stages that they will go through:

Stage 1: The Menu

This is the first visible element after the game is first run. It will feature the title of the game, as well as the option to begin.

Stage 2: The Tutorial

After the game is initiated, the player will initially be relocated to a start room. Here, an automatic tutorial will be run, with the informative purpose of getting the member of the panel to understand how to play the game. The player will be instructed about the different controls on the pads, and how each option allows the player to move through the environment or interact with it. For example, they will learn how to pick up objects, interact with objects, open and close doors, etc. This stage is also a good opportunity for the user to do the first steps in getting used to the locomotion associated to this game, which is partly similar to real life (i.e. looking around, turning around, squatting), but which does have different aspects (i.e. the lack of usage of one's real legs).

Stage 3: Exploration

Once the tutorial is completed, the user begins the actual quest of the game. The purpose, which is stated at Stage 1, is for the player to make his way out of the hospital. The player then begins progressing through the rooms, whilst finding and interacting with different objects.

For example, a torch can be found, which can help investigate dark rooms or corners. The torch has an "on" and "off" mode, as well as an energy level. The player needs to be really careful about when to use it because, if it is kept operational for too much time, the battery runs out, and the object cannot be used anymore.

Another element that contributes to this stage is that of the scripted events, which are meant to enhance the overall ambience associated with the horror genre. One such example occurs when the player enters a long, narrow corridor. The dim lights begin to automatically flicker, the further the player advances through the hallway.

One of the elements which are present in this stage, yet not heavily emphasised until the next one, is the monster. It is an entity surrounded by a mysterious fog that makes its appearance on certain occasions. Its movement and behaviour belongs to the aforementioned class of scripted events.

The player must continue to look for the way out of the hospital, while simultaneously avoiding getting caught by the monster. If that were to happen, the player would lose the game, and the “bad ending” would be displayed.

Stage 4: The Final Showdown

This is the stage of the game that has remained uncompleted, due to the global pandemic. It was meant to feature a concluding confrontation between the player and the monster.

This was ought to be a culminating moment for the game, which can be regarded similarly to the traditional “boss battle” from classic games.

Maintaining the software during development

In order to keep track of our software material, we used GitHub to store and transfer everything. We had different branches for different users or for certain aspects of the game.

For instance, we used separate branches for individual rooms, or for certain elements of a room that required more testing or trial-and-error evaluation. One such example is the custom-implemented particle effect used for the fog shroud which surrounds the monster.

To accomplish certain goals, we would often entrust ourselves individual tasks or work within subgroups of two or three. We believe that this method of working concurrently has helped boost the overall efficiency of our team.

Adding new functionality to the game

Since certain aspects of the game remain unfinished, the question of adding a new functionality to our game is a very practical and potentially useful one for us in the future.

As we worked for this project, we have always tried to keep the material as organised and tidy as possible, both in the repositories as well as within the software tools we worked with (i.e. Unity and Maya). Therefore, any addition of a new model, script, scene, etc. can be done easily, by following the right path, and can then be integrated within the game.

Technical Content

Overview

Creating a game within the horror genre provides a difficult challenge of making a story and environment which is both scary and believable to some degree. One key to implementing horror is suspense. This can be achieved by alluding to a monster or entity whilst not showing the player directly until a late stage in the game. Another key aspect is in-game audio; having unexplained noises coupled with heightened noises of say footsteps or a heartbeat creates an unnerving atmosphere for a player. We were in the process of creating a custom sound engine, which would realistically reflect sounds off the in-game objects and surfaces. This also has the benefit that we could add features and customise it as we saw fit to produce the effect we wanted for the game. Unfortunately, due to the COVID-19 outbreak, this feature was not completed and so is not present in our uploaded project.

There is also the challenge of implementing some interactive gameplay element within this story, especially as we are utilising VR hardware. Simply creating a game with no objective is harder to engage the player and does not show off the utilised hardware to its full capabilities. Knowing this, we had to make the decision of what the player should aim to achieve within the fictional world we created; whether it be to complete various puzzles, overcome the villain/monster that the story revolves around, or to level up throughout the game and defeat a final boss. Given the time constraint of 10 minutes, we decided to try and combine some puzzle solving along with escaping the monster before escaping at the end (if you were successful!). This allowed us to make use of the main benefits of VR (more intuitive interaction and movement) to engage the player with the game, whilst also providing the player with a distinct end goal in order to keep their attention and interest throughout gameplay.

Through initial group discussion we came up with various ideas and settings where our story could be set and ended up deciding on a hospital setting based inside the protagonist's (the player) subconscious. We were all in agreement that a hospital setting could be made very eery and disturbing, whilst also being a setting where many different objects, rooms, and props would fit into the surroundings naturally (such as bodies, beds, desks, medical equipment, etc.). Once this had been decided, Owen took on the task of storyboarding and creating a rough outline for the playable game map, which we thought was a good fit due to his interest with the horror game and movie genres. Often, he would relay back with the team with his ideas and we were able to implement them into our Unity project.

Map creation

Creating the playable game area proved more challenging than initially thought. The map needed to fit with the storyline of the game well and give chances for scripted events to occur naturally, however, we also wanted to convey our story point of being in someone's subconscious through the map itself. Inspiration was taken from optical illusions where it looks as if a surface is never ending or repeating. We managed to implement this by teleporting a player in the game when they walk through a certain set of doors to another part of the map, which they have already been through. This in itself posed technical challenges such as: showing a different part of the map through windows than were actually there, alluding to the fact that the player has been in this area before but they shouldn't have been able to via the route they took, and teleporting the player without any in game signs of this (stuttering, skipped frames, view changes of any sort).

Implementation of the teleportation feature was done using a custom shader which displayed what a separate camera (looking out of the target doors matching the relative position of the player) in the game was viewing as a texture on a Unity plane, which was placed behind a set of fake doors. This made sure that the player saw the correct scenery through the windows in the doors. From this, the player position is then changed via a transform so that they are teleported to the corresponding position in the target room.

Custom Vignette Shader

A custom Vignette shader was created to simulate a tunnel vision effect under certain conditions.
We had the idea to simulate the tunnel vision effect if the players heart-rate went over a certain threshold, or if the player got too close to the monster to act as a visual deterrent whilst also adding to the atmosphere of the game. The latter ended up being the final implementation within the game as we were not able to incorporate the heart rate monitor into the project due to Windows 7 limitations with low energy Bluetooth devices. Despite this, we were able to retrieve real time heart rate data from a chest or wrist mounted heart rate monitor through a C# application on a Windows 10 device. The application would look for the first heart rate monitor connected to the computer via Bluetooth (prior to executing the app) using the Bluetooth device description label, and then retrieve only the relevant data using the Windows Bluetooth C# libraries. The data was singled out easily as the bluetooth standards mean that all data communicated has a descriptor attached to it, allowing for easy identification of devices and data.

The shader itself was implemented using 2 maps (colour and alpha maps) and was applied to a plane which matched the relative position and orientation of the players head. The maps used a smooth step function in order to fade towards the edges of the screen. The colour and

alpha maps ensured that the blackened edges of the shader were not transparent, but the center was, providing a correct vignette effect. The reason we chose this method was because post-processing can provide extra expense to performance. Factor in that this would have to be performed to each lens, it would be too expensive to use this. Furthermore, post processing on each lens doesn't produce the exact effect we were after, as it would be a vignette effect for each eye, rather than the player's field of view. The floating plane resolved both issues with performance and with matching the player's field of view, and so an effective vignette effect was in place.

Tutorial and introduction to gameplay

Due to the recent development and use of VR equipment, it was necessary for us to develop an introductory sequence in which the player is taught how to use the VR controllers within our game. This introduction needed to be intuitive for someone to go through so that anyone was able to play, but it also gives us a chance to set up the in-game story. This ensures that the player is not taken out of the game's atmosphere, thereby enhancing their experience.
The use of visual onscreen aids were used in order to tell the player what each controller button did, and audio cues were used - such as a phone call from a mobile phone in the start room - in order to both convey the story and guide the player through the different areas of the initial room.

Controls and object interaction framework

Regarding the control system itself, we created a custom player movement, interaction and action performing framework which allowed us to add functionality as we saw fit by following the main principles of Object Oriented Programming. This allowed us to perform different actions based on the current player state, giving us a lot of flexibility on how a player interacted with the world. This framework was also easily extended to deal with objects that the player would be able to interact with; doors, trolleys, beds, books, flashlight, keycard etc. The framework is constructed around a single Interactable class, with other child classes using this as a blueprint (as per standard OOP principles). These child classes focus in on different aspects of a players interaction such as Grabbable - a script added to in-game object allowing a player to hold or squeeze, Squeezable - a child which inherits similar functionality to grabbable objects, except the trigger must be held in order to keep hold of the object e.g. door handles or rolling trays. There were also classes for the player belt storage and most importantly their in game hands.

A Trigger Event framework was also constructed to allow various player defined actions to trigger an ingame response automatically and dynamically. These range from a player walking

through a certain door or part of a map causing lights to flicker, or the player using a key card against a locked door in order to unlock it.

Modelling, culling and optimising draw calls

Another key aspect of immersing the player is the in game models. Although more challenging, we decided to create all the models ourselves in Maya in order to ensure that our game had the correct overall aesthetic by not relying on the textures that some pre-existing model would have. Custom models also give us the opportunity to easily modify any model if needed, either for aesthetic changes or performance gains. We split up the modelling task between all the team members that had completed the Character and Set Design unit as these team members would have had extensive experience using Maya for modelling and texturing models. Occasionally, we would retopologize high resolution models in order to reduce the vert/tri could in order to gain performance.

Maintaining a smooth framerate is a key concern and challenge when creating a VR game. This is due to the fact that every draw call needs to be performed twice - once for each of the two screens present in a VR headset. One way we combatted this was to implement Unity's occlusion culling, where the scene is baked prior to the game being built, allowing for objects only in a camera's field of view to be rendered by the engine. This dramatically improves performance as significantly less objects are rendered for each frame. We aided this occlusion process by breaking up larger objects such as walls and ceilings, so that only the visible parts of these would be rendered. This also gave us the advantage that we could have more detailed objects if needed, again improving and solidifying the immersion that we are trying to create within the game.

The Monster

Our vision of the monster which is alluded to and stalks the player throughout the game was to surround it in smoke to a) create mystery, and b) so it would leave a trail where it had recently been. Both these aspects would add suspense to the gameplay, as well as engaging and immersing the player. We initially looked at Unity's inbuilt particle system, which while is very fleshed out, we didn't find that we could achieve the exact effects we were after. The decision was then made to create a custom particle system which could be used for our monster smoke effects, as well as being easily adaptable to accommodate effects such as water fountains, air-born dust, and rain.

Technically, the creation of the particle system is complex in regards to both the physics needed and the amount of variables that are in play to alter such things as particle size,

number of particles emitted per second, fall speed, growth and shrink rate over lifetime, life span, emission radius about emitter to name a few. All of these can be changed within the Unity project in order to create the suitable effect for the situation. Unity's inbuilt rigidbody system was used in order to have gravity act upon the particles, with an additional variable used to alter how great this effect was. An additional interface layer was also added to our custom interface framework so that the particles were able to interact and be stopped by other in-game objects, whilst not being able to collide with the other particles. This was necessary to avoid visual and performance issues.

The particle system itself is composed of two C# scripts; a Particle Emitter script which acts as a parent to all of the Particle scripts, with a prefab also being created for the smoke particles used for the monster. The particle emitter script randomly determines the direction and speed each particle will have (within the specified limits), as well as updating the particle positions for a moving emitter and deciding which particles available to it will be triggered 'alive' next. The bank of usable particles is instantiated by the emitter script, with a size determined in the Unity editor. From here, the list is cycled through, with the position of the next particle to be activated being stored using an array index pointer. The particles themselves handle size increases and decreases as necessary, as well as hiding and 'killing' themselves once their lifespan has been completed. Messages from the emitter to particles and vice versa are communicated through stored object references, which while slightly more inefficient memory wise, is much faster when it comes to execution as we are not having to search for each particle, or the emitter, at runtime.

Initially for the physical appearance of the particles we used the quad prefab found within Unity, as it is composed of only two tris per object. However, this made particles of most sizes look very 2-dimensional as you could only see the texture from one direction. Ultimately we used the cube prefab in order to solve these two issues, with performance surprisingly not taking a hit from this decision. The cube texture itself (for the monster) was made by sampling a smoke texture found online, and then applying one of the Unity legacy mobile shaders in order to create a semi-transparent effect, as well as applying a colour. The resulting system looked realistic and performed better than our expectations, matching the in-game performance of the Unity particle system with equivalent settings, whilst giving us the customizability as desired.

Not only was the aesthetic important for the monster, but also the functionality. In terms of scripting for the monster, we had two areas to focus on: A patrol system and a field of view system. The first allows the monster to patrol a specific environment in the game and the latter allows the monster to deviate from this patrol, should the player be detected. The patrol system was implemented by a custom script which worked in tandem with Unity's built-in navigation system and also the custom FOV system. The monster essentially has 3 states: patrolling, scanning an area and chasing the player. The monster can only be in one of these states at a time and its state is chosen based on a set of conditions and its current state. If the monster does not have sight on a player, and it is not within a certain range of some predetermined way

points, then the monster is patrolling randomly between the way points and will focus on one particular way point to travel to. If the monster does not have sight on a player, but it IS within a certain range of some predetermined way points, then the monster will stop at the waypoint, and scan the area for a set period of time. If the monster has a visual on the player, then all other states are ignored and the monster is “chasing” the player until it loses sight, if it does lose sight, it will return to the patrolling state.

The monster FOV system used a ray-cast system. We implemented it in such a way that we could control the FOV range, angle and height that it was able to search in. The monster would periodically cast rays within the restricted range and angle of the direction it was facing (i.e. its range of view vs its field of view). The system would then check if these rays intersected with the player, without intersecting with other objects first. If this was the case, then the FOV system would communicate with the patrol system, signalling that the player was sighted and that it should begin pursuit.

4D Elements

For games day, we had originally planned to add some ‘4D’ elements to our game experience in order to add something new and intriguing, whilst again trying to further immerse the player. Our plan for this was to be able to control a fan or air conditioning unit within the room so that during certain sections of our game, the player would feel a cold breeze pass over them. To achieve this we planned to communicate to an Arduino (which was running a listener script) over WiFi to act as a switch between a plugged in appliance via a breadboard. This was sadly not fully developed due to not receiving the appropriate additional hardware in time before the COVID outbreak, however, a solution which illuminated LEDs instead of turning on a fan was developed successfully and so we are confident that in different circumstances our solution would have performed as described.

VR Doors

One of the more surprising technical challenges for us was in-game doors, which required us to write some custom scripts relating to interaction and swing physics. In theory, a door has simple interactions; pull or push. However in practice, and especially VR, there are some more aspects to consider. In order to pull a door, the player needs to be able to grab and hold onto part of the door (the handle) and then apply a force to open the door. In our implementation we have a logical handle which the player can grab onto and move, and the physical representation which the player can see. Repositioning the logical door handle will change its displacement and therefore results in velocity in the eyes of the Unity physics system. This initially resulted in the door flying open when the player lets go of the logical handle. This is

due to the rigid body of the door and the handle being connected. This doesn't portray a heavy fire door, or any door for that matter, as the player would expect the door to stop moving once the handle (and the force applied to it) had been released. To achieve this response, we created a door handle script which would temporarily disable the physics for the door and physical handle when the player grabs onto the logical handle. This was done by enabling kinematics on the door, meaning the object in question does not respond to the physics system provided by Unity. This is necessary as the physical handle will naturally follow the Rigidbody of the logical door handle when the player is trying to open the door, and so it will get some fixed (or stored) velocity causing the erratic hinging behaviour, even if the door is stationary. We could not have simply set the velocity value to 0 due to the colliders used for object-to-object interaction. Turning on Kinematics ignores this stored velocity in the physical handle, and so we achieve a more realistic door action when the player tries to open them via the door handles. Once the player releases the door handle, we turn back on Kinematics and so the door will obey the in-game physics once again (say if the player threw an object at the door, it would react accordingly). The player is still able to slam doors depending on how hard we close them by applying some velocity manually depending on the real-world velocity of the player's hand.

VR Hospital Curtains

We set about implementing interactive curtains by making use of Unity Cloth. The first challenge faced whilst designing the curtains was that by default, the backface of a cloth would be culled, so from one angle, the curtains were invisible. The solution to the problem was simple, by including 'Cull Off' as an option in the sub shader, the back-faces were no longer culled. A problem we faced with many instances of the curtains was that the continuous cloth physics and rendering reduced the performance of the game to an unacceptably low frame rate. By default the cloth would always be running, and the skinned mesh renderers were being ignored by the occlusion culler. We went about adding some control to the curtains, so that we could decided if they were enabled or not, based on how close the player was to the relevant part of the map, i.e. if the player is in the atrium, there is no need to have the curtains in the wards active. We used colliders to detect when the player was within range of a group of curtains. We made sure that only the player could trigger the events that controlled curtains. We implemented code that would add the correct colliders to each curtain on Awake(). It was easier to populate the list of colliders this way, as it would ensure that every single curtain in the game could interact with the player if they chose too. Before COVID, we had been continuing the development of curtains, to include a method to allow any object in the game to interact with the curtain. In testing, we had this functionality for the rollable tray tables, all that was required was to make the functionality more general. Unity Cloth only interacts with sphere or capsule colliders, so we would have had to implement a method to dynamically create a capsule collider for an incoming object if it didn't already have one. Alternatively, we could

have made sure that every asset in the game that ‘could’ interact with the curtain, already had a capsule or sphere collider on it.

Smashing Glass Event

One of the scripted events in the game was a window which would break and shatter into the corridor whilst the player walked past. There were two window models, one was fixed, and the other was broken up into shards. By using a collider, we trigger the scripted event when the player reaches the correct point. The change between fixed window and smashed window occurred when a rag-doll patient was thrown towards the window. When the patient was close enough, the script would switch the windows between enabled and disabled, so that the broken window was active. Next it applied velocity to each shard of glass. With some small tweaks, we were able to achieve the effect of the body smashing the glass window.