

# Drum Druid Circle

GROUP F, THIRD-YEAR GAMING PROJECT



Callum Ward:

Jack Walker

Paul Vladislav

Ena Balatinac

Max Prasertsan

Angus Purcell

Patrick Lee



University of  
**BRISTOL**

# Signed Contribution Page

**Team Manager:** Callum Ward

Weighting: 1.18



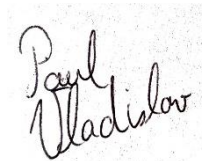
**Lead Designer:** Jack Walker

Weighting: 1.00



**Lead Programmer:** Paul Vladislav

Weighting: 0.94



**Team Member:** Ena Balatinac

Weighting: 0.97



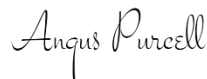
**Team Member:** Max Prasertsan

Weighting: 0.95



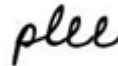
**Team Member:** Angus Purcell

Weighting: 0.94



**Team Member:** Patrick Lee

Weighting: 1.02



# Top Five Contributions Page

1. We implemented a real-time procedural generated tree-growing system based on the L-System generation model. The growth rate is linked to how well the players score in-game.
2. We developed a motion tracking system for multiple bodies using the Kinect Camera; replicating the movement of players to in-game models.
3. We developed custom force-sensitive sensors and scripts to perform low-latency hit extraction and debouncing with MIDI support. Each is packaged into a modular laser-cut drum featuring asynchronous LED animations.
4. We utilised the DAFNI Immersive data tent with a 5.1 surround sound system to create a 270-degree surround environment for the players to experience the game.
5. We implemented a Python library to acquire the drum tracks' spectral flux data and created a timestamped map of the peaks representing the loudest point of each drum hit. This was used in conjunction with a MIDI file reader to get the onset time of each drum hit.

Demo Vid: [https://www.youtube.com/channel/UCwsdtnaA\\_oA6RN8iqoSYMZQ](https://www.youtube.com/channel/UCwsdtnaA_oA6RN8iqoSYMZQ)

# Nine Aspects

## 1.Team Process

---

- 1.1. Held consistent meetings three times a week, in-person, Monday, Wednesday and Friday. (Page 7: Team process and Project Plannig section)
- 1.2. Assigned, categorised and prioritised over 150 tasks on the GitHub Kanban board. As well as creating a Discord server with channels for each branch with further details and progress logs.
- 1.3. Used flowcharts to assist development priotisation and object orientied communication patterns (

## 2.Technical Understanding

---

- 2.1. Used Librosa library from Python to read the beats from music tracks.
- 2.2. In-depth research and development of FSRs, microcontrollers and signal processing for the creation of the drum
- 2.3. Used Azure Kinect Bodytracking Unity API for transfer of Kinect tracking data into Unity
- 2.4. Research and used the DAFNI immersive tent as part of the game experience

## 3.Flagship Technologies Delivered

---

- 3.1. Implemented a procedural growth system for a variety of trees based on L-system
- 3.2. Developed multi-body tracking and organisation for the Kinect camera
- 3.3. Fabricated custom FSR sensors and modular drum controller, developing signal filtering scripts performing debouncing and peak extraction

## 4.Implementation & Software

---

- 4.1. Developed beat generators reading from composer-sourced MIDI files
- 4.2. Created an algorithm to generate trees procedurally

## 5.Tools, Development & Testing

---

- 5.1. Utilised agile development cycle with consistent team meetings and evaluations on Monday afternoon
- 5.2. Managed more than 400 commits to the GitHub repository and over 10 branches that are categorised into each section of the project
- 5.3. Had over 25 testers to play our game, age 17 and above.

## 6.Game Playability

---

- 6.1. The beats generated on the screen have been controlled so that the speed is within human reaction time.
- 6.2. Presented clear instructions on how to play the game
- 6.3. Developed a convenient menu control that uses the drums as a way to navigate
- 6.4. Designed the gameplay style based on the feedback received throughout the game development cycle

## 7.Look & Feel

---

- 7.1. Used a high polygon design with a semi-realistic feel to it
- 7.2. Handmade assets in the game except for Skybox
- 7.3. Bespoke music from composers gives a better experience to the drumming
- 7.4. Used DAFNI immersive data space to bring the players into the game space
- 7.5. Decorated the play area to increase the immersion

## 8.Novelty & Uniqueness

---

- 8.1. Handcrafted drum controllers
- 8.2. Handmade assets in the game except for Skybox
- 8.3. Created a unique rhythm game experience with high-octane and relaxing beats within the same game
- 8.4. Created a cooperative drumming experience for three players
- 8.5. Developed procedurally growing trees

## 9.Report & Documentation

---

- 9.1. Documented images on MVP showcase (See appendix)
- 9.2. Documented images on Drums building process (See appendix)
- 9.3. Documented DAFNI Immersive data space building process (<https://youtu.be/RLKAojntfvo>)
- 9.4. Documented images on Testing process (See appendix)
- 9.5. Documented images on Video making session (See appendix)

# Abstract

---

## Overview

---

Druid Drum Circle is a multiplayer rhythm game with up to three players. Players work together aiming to stay in rhythm with the beats coming down the screen. Each player has two drums using our custom sensors. Hitting the drum when a beat is in the target area reproduces the instrument sounds extracted from each track. The game features three biomes, a forest, a mountain and a beach. Where each biome plays a unique song with varying difficulty. Better performance results in a higher score and additional trees growing in the environment with more visual effects.

## Story and setting

---

A trio of druids wonder the wasteland that once flourished with gifts from **Jörð** (Jord), daughter of Night and mother of Earth. They attempt to fix the damage caused by humanity using their musical power. Channelled through special wands, and drums made from the remains of the world tree.

In this post-apocalyptic world, the druids are armoured by pieces of world tree and bones from creatures that once dominated the Earth. Ready to answer the call from Mother Nature once again.

The journey starts from deep inside the forest, where the players must regrow the local flora. Then up the mountain to an old shrine. And finally down to the beach. Each area is based on their mythology, culture and region.

The Forest is heavily inspired by Norse mythology. This map is the core concept of our game with the theme behind being more relatable to our team due to the popularity of the mythology. The concept behind this map was that the players, a group of druids, were on an adventure to help stabilise the world tree or '**Yggdrasill**'. The music in this map represents the step required to heal the Earth, one symphony at a time.

For the mountain map, we took inspiration from the East. Mainly Japanese culture. The story of **Izanagi**

and **Izanami** was the base of this track. The three druids must calm nature which is now in chaos due to the damage. To purify and cleanse the land of all the chaos and taint. The influence of the Shinto culture can also be seen on the map along with the assets such as Shinto style shrine, Torii gate and stone lantern. The tree for this map is a cherry blossom, custom-made for this specific track.

For the beach map, the main inspiration was from the Caribbean, mostly Puerto Rico (Borikén), Hispaniola, and Cuba. This was a mythology that is not widely known, and higher difficulty to research. The lore behind this track was that the druid's trio must please '**Atabey**', the goddess of music, fertility and beauty. This ancestral spirit is mad with how the human had been treating nature. This map includes palm trees, beach style buildings and rocks.

## Roles

---

Players can play as one of the three druids, each with their own unique drum track to follow. They should then try their best to play along as accurately as possible to ensure as high a score as possible and to maximise the growth of the trees in landscape. For most of the level, the player's aim to work together, playing separate drum tracks which form one larger soundtrack. At different points throughout the level, players will get to perform an individual freestyle, whilst the other players continue with their tracks. This will allow the given player to create their own twist on the music, and earn points based on the complexity and timing of their freestyle. At the end of the song players will be given an individual score, as well as a team score that they can record.

## Gameplay

---

From the start, players can use their drum controllers to navigate the menu. The first option will lead them to a page with three icons, representing the three maps or music tracks. Players can cycle the option by using the left drum and confirm with the right drum. Once the

map has been chosen, all players will get right into the game. Starting with a countdown of 3 seconds. Following this a series of Celtic symbols will appear relative to the drum beats in the song. Each player has their own drum track to play along to so they all have a unique experience. If players hit the beat at the correct time, a drum is heard audibly, and the player earns points. While playing in time the players will also be playing along to the music and the music will compound and build. Also, while players hit notes without missing, they will build a 'combo' that will amplify the earned score. If players miss, they are notified by their section of the screen shaking, turning red, the build- up of music degrading and the combo being reset.

As combo builds, orbs of energy are fired out of the player's (in-game) drum. This will fly away and strike the ground at a random location within view, triggering a tree to spawn. The spawned trees will then grow using procedurally generated branches. The speed of the trees growing will be relative to the player's combo multiplier and they will also have a unique glow based on the player who spawned them.

Once a song is finished, the camera will pan around and closer to the spawned trees, showing the players the new



Figure 1: Team testing out the drum controllers

tree-scape they have created. They will also be presented with their scores, and player 1 will be able to tap their drum to return to the level select screen.

## Controls

Other than initially starting the game on a computer, the controls are purely operated by tapping the sensor on top of each drum. Each player has two drums that they use to play the game and navigate the menus.

Menus are navigated by scrolling down with the left drum and confirming the chosen selection with the right drum. Once a scene is chosen, the players are sent into that specific druid world. Beats will spawn on the left or right of a player's track and this will correspond to the relative drum. The aim of the game is to hit the sensor on top of each drum when the corresponding beat is within the target zone. Hitting at the correct time will play the beat within the song and initiate procedural tree growth on the level. The closer to the center of the target, the more points are scored.

Each player has a specific colour, and this colour is shown both in game and on the physical drum. This allows the player to easily distinguish which notes they should be hitting, as well as which trees they are responsible for growing.



Figure 2: UI layout of the game



# Team Process and Project Planning

## Overview

Our team process and project planning had been in an agile week-long sprint system, where we planned and organised tasks on Monday. Gaining and reconsolidated them on Friday. We made Flowchart for the game concept to map out the general guideline for the development. We explored other software to help track tasks and progress, but GitHub kanban board was the main method of tracking for us.

## Weekly meetings

As part of our team process and project planning, we held weekly meetings three days a week: Monday, Wednesday, and Friday. These meetings were crucial in ensuring that our team stayed on track and worked effectively towards each project goal.

On Mondays, we held meetings to set our weekly goals and priorities. During these meetings, we discussed the tasks that needed to be completed for the week and assigned them to team members based on their areas of focus and abilities. We also discussed any potential roadblocks or challenges that we might face during the week and brainstorm solutions to overcome them.

In addition to our weekly meetings, we also used Discord channels to post small reports on our progress and share research links to assist with others' tasks. This allowed us to stay connected and informed about each

other's progress. Providing an efficient way to get quick feedback and advice on how to tweak our solutions.

On Wednesdays, we held check-in meetings to assess each member's task progress. During these meetings, team members reported on the tasks they had completed and any challenges they had encountered. We also discussed any changes or adjustments that needed to be made to our weekly goals and priorities. This included extending tasks into the following week or assigning additional tasks to get started on. Some tasks proved to be more difficult than we anticipated, in which case we assigned more members to help solve the issue.

On Fridays, we held evaluation meetings to review our progress for the week. We discussed what we had accomplished and what still needed to be done. We also conducted testing of our game to identify any bugs or issues that needed to be addressed. These meetings were crucial in ensuring that we stayed on track and made steady progress towards completing our project.

We stayed consistent with this meeting structure throughout the project, which helped us maintain a clear sense of direction and focus. By using week-long development sprints as part of an agile development approach, we were able to quickly adapt to changing requirements and make rapid progress.

Overall, our weekly meetings, use of Discord channels, and adoption of agile development practices were essential parts of our team process and project

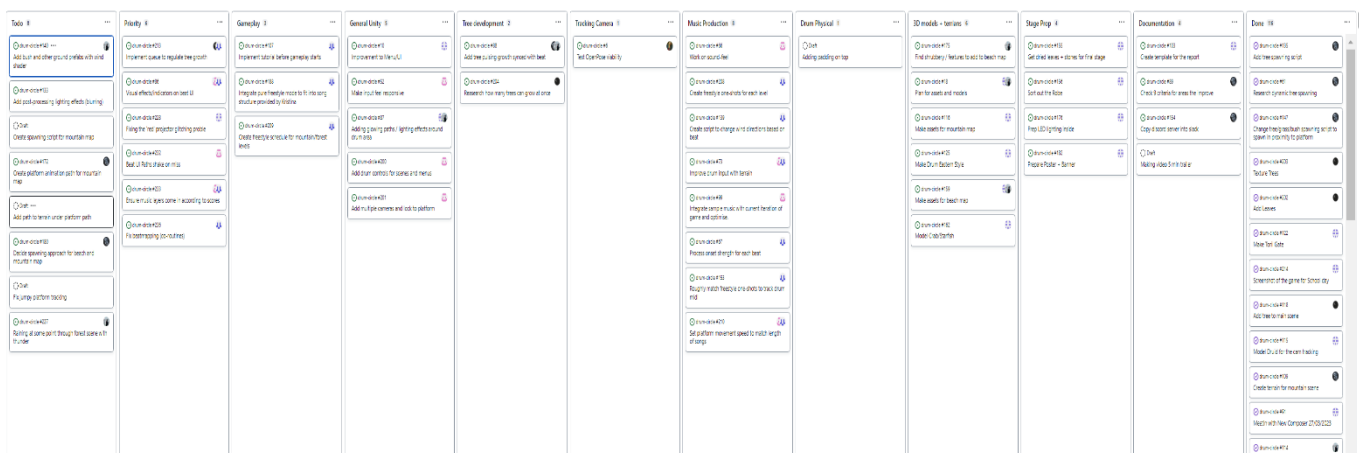


Figure 3: GitHub Kanban task management screenshot



planning. They helped us stay organized, focused, and motivated as we worked towards completing our game.

## Repository management

---

As mentioned previously, we have used Github to host a remote repository of our source code. We initially set up branches to drive the development of the core components of the game. Aiming to produce an MVP as fast as possible. At the same time creating Kanban categories to reflect branches. Doing this avoided confusion about which branch would be used for each task. We made a point of creating rules around making contributions to the repository. Always rebase or merge from the main branch before creating a pull request. Test your code before committing to a branch to prevent bugs from being merged into the main branch. These rules aimed to ensure main stayed free of bugs and was up to date with previous pull requests. We also aimed to commit early and often preventing merge issues between multiple members working on the same branch. This rule also applied to staying up to date with main. Falling behind by too many commits increases the chance of merge conflicts. When dealing with Unity proprietary file types, it can become very difficult to decide between versions.

Throughout the project development lifecycle, there have been 527 commits across 12 branches with 77 pull requests into the main branch. With varying distributions on contribution due to some elements of the game being developed outside of the repository. Such as drum development and models being created in Maya.

## Task management

---

For the task management process, we used GitHub to create a Kanban board for tracking our tasks. We organised tasks into categories for each area of development, including Priority, Gameplay, General Unity, Tree development, Kinect tracking, Music production, Drum, Terrain, 3D-Models and Stage props. Each task was assigned a status of Todo, In Progress, or Completed.

Using a Kanban board allowed us to easily visualize our progress and see which tasks were completed and which still needed to be done. This helped us stay organized and focused throughout the development lifecycle. The Kanban board also allowed us to easily assign tasks to

team members and track their progress. Ensuring that everyone was contributing to the project and that tasks were being completed on time.

The use of categories provided a way to prioritise tasks within each crucial area of development. Supporting steady progress towards our project

milestones, and equally ensuring that we did not overlook any important tasks. Having a Kanban

board integrated into GitHub meant that we could link and close tasks with pull requests into the main branch. Creating an easy way to track our progress through commit history and see how the codebase evolved.

We also used our Discord server to create channels for each area of development. These channels complemented our task tracking by allowing us to provide more detailed descriptions of what needed to be done. We also used the channels for posting bugs and getting help from other team members. By the end of the project, we had completed over 160 tasks. GitHub and Discord proved to be effective ways of documenting and planning our progress.

If we were to start again, one change we would make would be to have team members switch between different areas of development. This prevents members from getting tunnel vision in one area, making the work more enjoyable, in turn increasing productivity. We believe it would strengthen the codebase by using fresh eyes to spot bugs and suggest efficient alternatives saving time.

## MVP Flowchart

---

Initially, we created the MVP diagram to better visualise the steps involved in completing each area of the game. With this structure, we were able to structure our development within Unity into clear units. It was also useful in highlighting our priority tasks. Without this we found it would be easy to drift into more specific tasks that would not be necessary for meeting the MVP requirements. This approach to development also assisted in setting up our repository with branches and with the creation of our task-tracking Kanban project.

With Unity using C# as the scripting language, it was important to leverage the object-oriented approach within our software structure. Creating the script communication hierarchy helped us visualise the dependencies and control flow within the final program. As a result, we had a clear picture of how to develop each script and create public communication functions before other members' areas of development had been

completed. Overall reducing the time and complexity when integrating our tasks.

## Contribution management

For our development cycle, we broke down the term crisis into three main categories, personal well-being, hardware/facilities, and software/technology.

We realised that people are the most important asset. To ensure that we were prepared for any potential health issue (physical or mental), we shared the tasks on Monday and if there's issue occurred during Monday and Friday, we would delegate that task to a teammate working on a similar area or push back on the timeline and try to wrap up the task within the next week cycle. We decided to prioritise health over work hours.

Hardware and facilities were more relevant once the DAFNI tent had been set up. Due to the lack of a manual and guide on the tent, there were ongoing problems, for example, the projectors not working as the intended or sound system was not displaying the base correctly. By devising a checklist for testing, we were able to keep the tent in the best possible condition.

Software and technology were more toward the issues with GitHub and Unity software. There were considerable rebasing when we tried to merge between branches. To prepare for massive conflicts, we dedicated time to the weekly cycle for the integration process. Unity also crashed at a certain point. To combat this issue, we saved regularly and push to GitHub once we were done with our subtask.

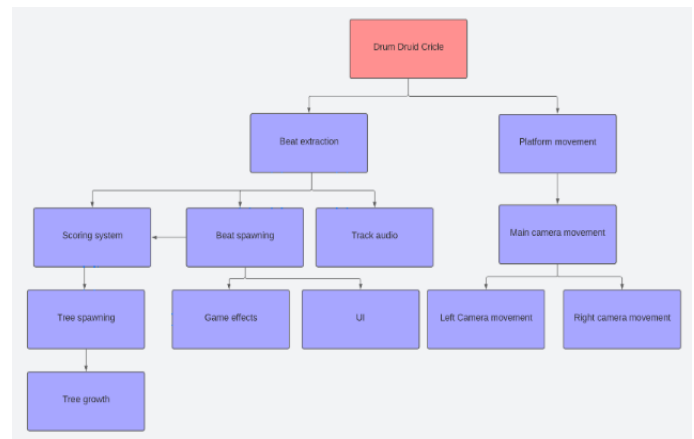


Figure 4: Script communication heirarchy

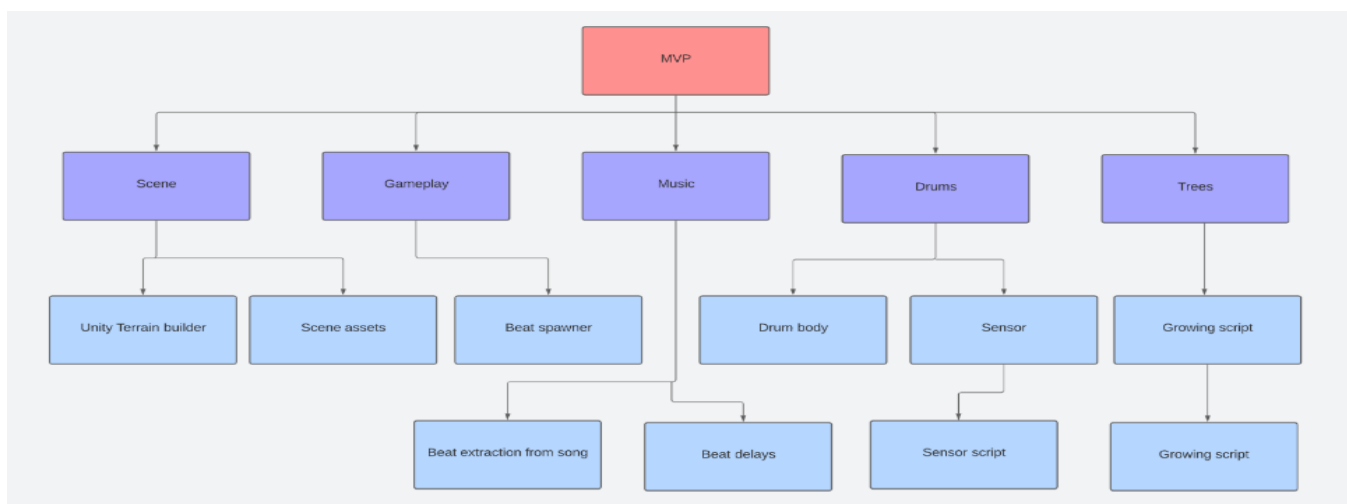


Figure 5: MVP development plan

# Individual Contributions

## Callum Ward - Team Manager

---

Focus: Drum sensors, Scene design, Gameplay

### Team Management

My team manager role meant that I took on administrative duties for the group. From the beginning of the project, I set up all lines of communication. Including a Discord server with channels dedicated to each area of the project. To support agile development, I researched each technology and provided resources for team members to get up to speed. Additionally, to kickstart the project I set up our GitHub repository with branches and a Kanban board, dividing tasks into focused groups. I set a standard for regular meetings and progress reports, making sure everyone had a clear image of the current game state. Another major part of my role has been creating a vision for the final product and setting goals to keep everyone on track. Also being a point of contact with staff allowed me to document suggested areas of improvement and integrate them into our plans. On average this role took 4.5 hours a week to complete.

### Drums

Developing the sensors, drum bodies and modular controller has been my largest contribution. This included initial testing of existing piezo's and force-sensitive resistors (FSR), finding the most effective solution. Testing each sensor involved setting up a microcontroller to read signal inputs. After analysing data from each, I developed custom scripts extracting player hits and velocities. Followed by fabricating and refining numerous FSR designs to improve reliability. Tested each implementation with Jack and Patrick leading to constant modifications throughout the project. Additionally, to make our drum interaction unique and integrated into the game, I designed and built six wooden drums using a laser cutter. Incorporating addressable LEDs asynchronously controlled by a central teensy microcontroller utilising the FastLED library for animations. To gain performance I changed the microcontroller from Arduino Uno to Teensy. Followed by switching the signal type from serial to MIDI. Due to the extensive

testing and development stage, this took approximately 10 weeks alongside my other contributions.

### Scene design & movement

Each composer provided a unique track and feel. To match the tracks I designed forest, beach and mountain scenes using Unity's terrain system. Importing textures and additional tools to aid design and make each landscape feel unique. This also involved developing shaders like the water shader used to simulate an ocean on the beach scene. With performance being a main concern, I converted the project to use Unity's universal render pipeline, making several optimisations. The most impactful optimisation leveraged static object declaration and mixed over real-time lighting sources, allowing me to bake light into each scene improving performance. I developed a modular waypoint system to move the player platform along pre-defined paths for each scene. On top of this, I set up a camera tracking system to follow the platform with a panoramic view. Developing different lock-on tracking approaches for each scene. This took 3 weeks.

### Tree spawning

Alongside terrain and drum development, I created scripts to procedurally spawn trees for each scene. Taking varying approaches to match the layout of each scene. Regularly testing with Ena and Paul to modify existing strategies. Using an OOP development style to link control flow between each area of development, creating a versatile spawning system. Which can accept procedural and static trees as well as other vegetation. This took 1 week.

### Report

I took the responsibility of structuring and writing fundamental sections of the report. Including Nine Aspects, Team Process and Project Planning and Technical content. This took 1 week.

# Max Prasertsan

*Focus: Administration, Assets making, Immersive tent management*

## Administration

I set and reminded my teammates about our weekly meeting time and location by using Discord mainly. I am also the point of contact for university staff and composers. Mainly to do with allocating resources to our group such as the budget for the hardware required to make the drums, set up the immersive tent and organised decoration and risk assessment/management tasks. Another task of mine was to organise and clean up the GitHub Kanban board so that my team could easily locate their weekly tasks. I spend roughly 3-4 hours per week. I also collaborated with my teammates in other parts such as the music direction alongside Jack and Patrick. Art direction with Ena and Paul. And assisted Callum in the production of the drums.

## DAFNI Immersive tent

My main responsibility for this project was with the tent. I was responsible for assembling and maintenance of the DAFNI Immersive Data space which is a 3x3 m metal frame tent with 3 projectors on top (more detail in the technical content section). This took about 2 weeks to set up, aligning the projections to the screens, and sorting out the cables.

Also, I was responsible for weekly checking of the tent by running a set of tests to ensure there were no hardware or software problems. The 5.1 surround sound system also required setting up (Software update, wires connection and testing). Multiple tests were carried out with our composers' music to adjust the system so that all the music is coming out in the right direction. Fixing this took about a week and a half.

## Video

I was in charge of recording the video, setting the script, scene and planning for the final product. This took about a week.

## Assets

I was responsible for creating assets by using Autodesk Maya on the mountain map such as the Shrine, Torii Gate, Japanese Red Lantern, stone lantern, and the first iteration of the platform. Also made the forest hut and the first iteration of the beach chair and the beach hut



Figure 6: Druid Model Iteration 3

utilising the skills that I learned from the CGI unit such as vertices manipulation, Boolean geometry mesh joining and material texturing. I also created the player model that can be seen in-game, rigging the skeleton system so that it can be used with the Kinect camera. Overall, the asset-making took approximately 7-8 weeks alongside other work.

## Documentation

I was responsible for taking photos during significant events, including the first MVP release showcase, the school day showcase, and the work in progress.

by

# Patrick Lee

---

Focus: Music integration, Gameplay, UI

## Music Integration

Much of my contribution involved creating an engaging visualization of the drum tracks in the form of descending 'beat' markers which would line up with the target area in time with the audible music. The quality of the gameplay relies heavily on this aspect, therefore it was important to ensure the mapping of the beats was as accurate as possible. I also succeeded in making this method of beat mapping such that it could be used generally i.e. for any given drum track. I initially researched existing projects which aim to solve the problem of 'beat' detection in an audio file, ultimately making use of the Librosa project and its corresponding Python library. This allowed me to write a Python script which could create a map of beat onsets against time, including a differentiator to determine if the beat was part of the underlying rhythm (e.g. a kick drum) or more of an off-beat transient (e.g. a hi-hat or tom) and store the resulting data as a JSON array. The composers later provided us with the relevant MIDI files which gave us auxiliary data on onset time and note velocity. Alongside the underlying music analysis, I also created a spawner which, given the JSON and MIDI data could spawn a visual representation of a beat at the correct time and, alongside Jack, implemented the functionality for the 'beat' markers to descend at the correct speed and the animations and scoring as when a player hit beat (in)correctly. Overall, the music integration as a whole took the bulk of the first 5 weeks of development, with continual refactoring and improvement afterwards.

## UI

Although we had created a provisional UI using 3D assets, once we implemented moving cameras, it

became necessary to switch to a 2D overlay. Jack created the foundations for this, and then I created much of the functionality to animate and apply effects to this UI. I created a base UI effects class which contained some common functionality, such as glowing, and derived from this to create specialized 'lane' and 'button' versions for effects such as the lanes shaking when the player misses a beat or the buttons swelling when the drums are hit. Including the initial 3D UI, around 4 weeks have been spent on this.

## Gameplay/Game modes

I implemented the functionality for a freestyle section, such that players could be given solo sections in a song where they would be able to play the drum notes unrestricted by rhythm markers. I created extensions to the UI whereby the beat markers would rise from the drums (instead of descending) and a player's lane would glow, indicating that it was their turn for the freestyle. I spent around 2 weeks developing this feature.

## MIDI Input

I made use of the Minis plugin [6] to integrate MIDI input for interfacing with the physical drum units. I implemented a listener to listen for MIDI 'Note On' events and store them in an array to be used by other components. This only took around 3 days to implement and test.

## Integration of music with environment

Due to centrality of music within the game, it was important that events such as tree spawning and grass swaying occurred in time with drum music being played. I spent time implementing the functionality required for this, including the spawning of 'fire balls' on-beat, when a user had hit enough successful beat markers, which would towards a location on the terrain and trigger the spawning of a procedural tree on impact. The 'fire ball' and explosion effects themselves were created using Unity's in-built particle systems, and the movement of them was based on quaternion-based rotation effects such that the fire-balls would 'home in' on a tree spawning location. The functionality for this music-interaction was slowly built up throughout the project as more features were added and improved.



# Jack Walker – Lead Design

---

Focus: Music integration, Gameplay, UI, Composer Communications

## Lead Design

As Lead Design, my job was focused on making sure the game fit together well, as well as sounded and looked good. Initially, I helped with the design and functionality of the game's core features, not long after I arranged meetings with the composers and worked closely with them to design music to our needs. Later on, my primary focus was on redesigning the UI to fit the game and look as good as possible.

## Core Systems and Music Integration

The initial sprint for the project added the core functionality of the game. For us this was acquiring the beats we needed from a song, visualising them on the screen, and then giving the player some indication of when they need to 'hit' the beat. While Patrick worked on the beat acquisition, I worked on the spawning and timing of the music side of the game. I implemented an Audio Manager that allowed us to easily add songs, and sound effects to be manipulated. I also set up the timings for the visual beats falling down the screen into the target area based on the data given from Patrick's beat acquisition. Next, I added a scoring system that calculated a score based on proximity to the 'perfect' timing and the player's 'combo', and then I finally connected the first iteration of the drums to the game scripts to enable our core functionality. This part of the coursework took roughly 5 weeks at the start of the unit with constant updates and development throughout the entire project.

## User Interface

Our initial implementation of the user interface (UI) using 3D assets that were instantiated into the game world and kept in positions relative to the camera. Initially, this was fine for our needs and allowed us to test and implement many parts of the game. As the project developed it became necessary for us to change our UI to a 2D overlay.

The 2D overlay UI would allow us to move the camera and create dynamic assets with no chance of said assets obscuring the UI or adding extra overhead. This new system proved to be very complicated to use for our needs, but I did manage to create a functional UI that

was integrated with our drum controls, allowed the selection of levels, and provided the core functionality of a playable game. After this, I worked closely with Patrick to add visual effects and style so the player can feel immersed and fully aware of their state in the game. In total, the UI likely took around 4 weeks to develop.

## Music

I had several meetings with the composers to talk about what we wanted for the game, how to best implement the music and then to show them where we were and to talk about any changes needed. From these meetings, we decided to develop three levels, and each level's song is split into multiple layers that we would reconstruct on our side. This allowed us to isolate the drums for each player so we can play/cut the expected sounds when they hit the drums, as well as allowing us to provide a compounding build-up of music that sounded increasingly impressive as the player progressed and did well. Between meetings and the implementation of the music system, this took roughly 2 weeks.

## DAFNI Immersive Tent

My main contribution was the immersive tent. I helped with the initial set-up of the tent and also configured the 5.1 surround sound to work with the music our game was using. I spent just under a week working on this part of the project.

# Ena Balatinac

---

Focus: Asset creation and optimisation, shaders, textures, visual effects

I've spent the overwhelming majority of my time creating different assets, materials, and shaders for them. Further, some assets that were imported needed adjustments and/or optimisations which I've further conducted. I've also done scene design alongside Callum for the beach environment and some on the forest one. I've spent some time creating visual effects. Below are examples of the work I've done.

## Shaders

Originally, the first iteration of the tree growth was created by using a shader graph, however, it was just a placeholder for the procedural trees. It worked on the principle of revealing colour from transparency as the growth parameter was changing. I've further created a grass shader made with a shader graph that has the effect of swaying to the wind which can be altered with external variables such as chosen strength or speed of the wind.

## Procedural Textures

One of the most significant textures was a procedural rock texture for the cliffs in the beach scene. It was created using a shader graph in Blender. To achieve the desired look of this complex texture I've used different texture and displacement nodes as well as procedural Musgrave texture which allowed me to generate this complex pattern without requiring any external reference image files. To make it more realistic, I've separated the base colour on the Z axis, and after using some maths nodes as well as Musgrave texture nodes I've created the procedural moss on the top parts on the Z axis of the cliff. After adopting different parameters and getting to the stage where we were satisfied with looks, the texture needed to be baked before exporting it to Unity because the material was made procedurally. After baking multiple maps such as ambient occlusion, UV, and Normals, I connected them into a new graph which was transferable to unity.

## Visual Effects

The thunderstorm effect in the forest environment was created by using 3 different elements- clouds, rain, and thunder. Clouds and rain were made using particle systems and the thunder effect was made by using a VFX graph and shader graph. Base cloud texture was created in the external image editing software as I needed to enhance realism in the cloud system and nothing premade was good enough.

## Assets

I've made numerous assets ranging from simple single-leaf to full-scale iterations of realistic high-poly trees, grass, cliffs, platforms, and others. As high poly models are made from a costly amount of vertices and faces, I've spent a significant amount of time trying to optimise them. I've done this by doing retopology. Even though I've tested industry-standard software for that, I've found better success by using tools in Maya to reduce the number of triangles and then further manually edit the unnecessarily complicated meshes. This was significant for the foliage meshes that I've created to be used with procedural tree growth. Lastly parts of some imported assets needed to be changed which I've done by exporting the into Maya, then I've changed/ built upon them to make them more appropriate and then imported them back to Unity.



# Angus Purcell

---

Focus: 3D Player Tracking

## Kinect Body Tracking

My sole focus throughout the project was to incorporate player motion data into the Druid Drum Circle game. As a key technology, this is an important part of the game to get right, and because of this the first few weeks were spent researching different methods to obtain player data and transfer them into Unity. My prototype used Openpose to track the movements of people and an API that could transfer this into Unity, making use of a webcam. However, this could only recover 2D skeleton data from a video, which posed limitations as our game is a 3D game. Considering this, I researched different methods to analyse this 2D data to extrapolate a 3D pose which is a huge topic, but doing this in real-time on a single PC was not feasible without high latency, even if a system to extrapolate was achieved.

At this time, I got an Azure Kinect to test with. The Kinect could obtain 3D player data, so it already solved the problem with Openpose. In the process of prototyping whether this option was feasible I discovered almost no examples of a Kinect being used to represent game characters live, and none with multiple unique players in real-time. To integrate players' data into our game, I made use of an API developed by a Microsoft employee, which had almost no documentation. This meant most of the time I spent on this in the early stages was on understanding how it worked and experimentation. I implemented an improved ID system to identify and track multiple players given how far from the Kinect they are.

On top of this, I had to use the data for each player to rig and move character models. The rigging of multiple characters was time-consuming as the data was not consistent for each character model, even though this wasn't the case for the skeletons that are separate from the models. To remedy this, I had to give each character model a separate data pipeline from the skeleton renderer to the character model animator, not an ideal solution but allows the scripts to be readable. This also meant coordinating with Max to ensure that the character models had skeletons inside them that would be cohesive with the rigging system that I had implemented. Due to the distance

measurement not being as consistent when players are a similar distance, such as inside the Immersive Igloo, I had to update the way the scripts calculated which player was closer. Comprehensive testing and development was challenging without 3 group members present which limited how much development could occur.

## DAFNI Immersive tent

I have also been given a couple of days to help fix issues with the Immersive Igloo along with a few other group members. This includes aligning projectors and locating the source of a projector malfunction.

# Paul Vladislav – Lead Programmer

---

Focus: Procedural tree generation

## Procedural Trees

During the initial phase of the project, I researched various methods of generating trees. I considered different techniques for procedurally generation, including space colonization algorithms, fractals, and L-systems. After careful analysis and testing, I decided to implement the L-systems approach for generating the tree structure.

Once I decided on the approach, I worked on developing and integrating the L-systems-based tree structure into the game. I started by experimenting with predefined shapes for constructing the tree trunk, but soon realized that this approach was limiting. To overcome this challenge, I switched to procedurally generating the mesh, which allowed for more flexibility in creating the tree trunk's shape and structure.

As part of my responsibilities, I also worked on making the trees grow and respond to the players performance in the game. I developed a system that determines the tree growth rate and behavior based on the players' score .

In addition to developing the generation systems, I also focused on optimisation. Generating the mesh is computationally demanding, as such I implemented a level-of-detail system that generates less detailed meshes as the camera moves away from the object, as well as a queueing system that limits the number of simultaneously growing trees.

Besides the technical aspects, I also worked on the visual design of the trees. I was responsible for creating the materials used to texture the trunks and designing the overall appearance of the trees, including their branching structure and leaf density. I collaborated with Ena to design the leaves used in the generation process. We experimented with different colors, shapes, and sizes to create realistic and visually appealing trees.

# Software, Tools and Development

## Overview

The game is made using the Unity game engine which is free software for students. With the help of packages and plugins, we can create an enjoyable game. Alongside Unity, we also used Autodesk Maya to create professional 3D assets.

## Unity

Unity game engine is one of the most popular game development applications. With many tools, plugins and packages that help create high-quality 3D games with ease. Half-way through the project we switched from Legacy rendering to the Universal Render Pipeline.



Figure 7: Beach map terrain in Unity

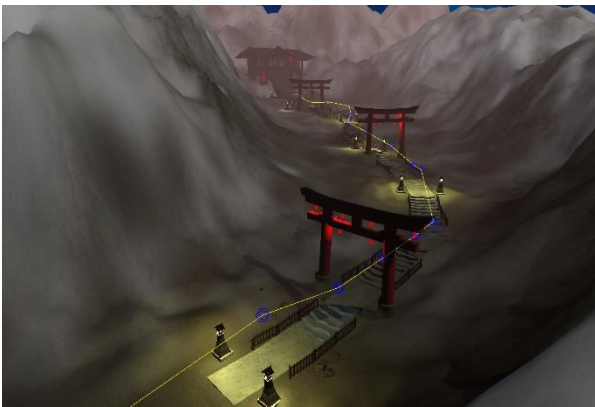


Figure 8: Mountain terrain system

Providing additional post effects and a lightweight build to improve performance. Each scene was then designed and structured using Unity's built-in terrain system. Specifically on the mountain scene, we experimented with Unity's new mixed lighting system. Allowing us to bake in lighting but also have real-time lighting from each of the lamps shown in the Figure 8. Overall, improving performance at the cost of a larger project

size. We also made extensive use of Unity's Shader graph system. These are crucial in developing any dynamic effect on objects. We implemented shaders for glowing objects and ocean simulation.

We used Unity's new UI Builder to build our UI. While this is more complicated than the old method of building a UI out of game objects, this gave us the tools needed to produce the exact interface we wanted. The UI Builder enabled us to build a 2D overlay for the UI, rather than a 3D one in the game space so the game world had no chance of interacting with the UI. It also allowed for much better scaling, so the UI would work with screens of any resolution.

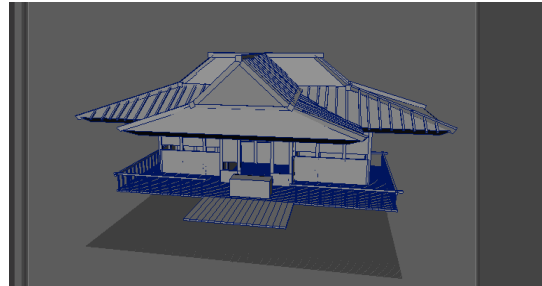


Figure 9: Shrine model in Autodesk Maya

## Autodesk Maya

All 3D in-game assets were created using Autodesk Maya. Maya is a professional 3D software with a robust toolset for creating realistic characters and visual effects. Maya uses a node-based architecture, where each node represents a function that can be applied to an object on the scene. Maya also supports various export formats and easy integration with other software, which in our case is Unity[14].

For the game assets, we try to make assets as realistic as the performance allows. This forced us to balance between the number of vertices and the performance of the game. By using soft edges, we were able to make a smooth-looking geometry without the significant increase in overall vertices. The assets were shaped from basic geometry such as cubes and cylinders. By using the tools provided such as extrude, nonlinear deform and vertices manipulation, we were able to create complex polygons that once combined form many of the assets shown on the map.

## Software Maintainability & Git

Usage of Git during this project was paramount to ensuring software maintainability. Given the size of the project, challenges with maintainability could quickly arise. To address this issue, the whole project was hosted on a remote GitHub repository every member of the team could develop their own aspects seamlessly, without causing disruptions other members' work. Each feature or task was developed in its own dedicated branch, allowing other members to work individually without interfering with either the main branch or the branches associated with other features. This way we could reduce the number of potential conflicts before they could occur.

Prior to merging into main, we followed a strict process where we can rebase or merge from main to update our work with latest changes from main. The benefit of this process is that it provided an opportunity to resolve any merge conflicts or bugs that might arise during the merge into main, but this way they can be identified before that could happen. By addressing these issues prior to merging, we could maintain and functioning main branch. Also meaning, that any member merging from main could be confident it was a working build.

## Testing

During individual development, we concentrated on our specific areas of the game to test. This approach has allowed us to focus on identifying bugs at a lower level. Meaning that when components of the game are brought together and tested in small groups within the team, the focus can be on the concept rather than the implementation. This ensures cohesive development of all elements within the game.

Since as a group we were limited to the feedback of the seven of us, we recognised the importance of external parties. These perspectives allowed us access to opinions and insights that we might have otherwise overlooked. During both showcase days and the school day we were able to identify areas that required work to deliver a fun and engaging experience. Through this method we were able to prioritise elements of the game that were crucial to the feel and experience, whilst simultaneously identifying aspects that had less impact. Using this method, we obtained feedback from panel members and school children, as well as friends and family of group members. This wide range of perspectives gave us a comprehensive guide of how to further improve Druid Drum Circle. We changed the UI

layout and effects based on the feedback received on blind testing. The core concern was that some of the UI features were hard to see and track, which was a red flag in our game because the players must be able to quickly identify the beats in order for a great gaming experience. The complexity had been dialed down and the icons were simplified for ease of use of the players.

We also created a systematic test for the DAFNI Immersive data space due to the fact that it was relatively new technology with no clear user manual and guide, beside the setting up manual. The testing consisted of four different stages. Progressing from the most crucial part or the most problematic part down to the most stable asset within the tent.

- Test the three projectors by running them one at a time, then two at a time and all three at once. With the Unity game running on each screen. If the issue persists, testing with other software or applications (such as youtube or chrome) to see if the issue is the Unity itself or the processing power of the PC.
- Test the HDMI cable by swapping the cables to the front projector which was the most consistent one out of the three. Alternatively if there is a consistent problem with any cable, carry out more testing with extra HDMI cable provided in spare equipment box.
- Test the sound system by running the music files provided by the composers and adjusting the amp accordingly. Also standing on top of the MVB staircase to test the loudness of the tent itself, adjusting to ensure minimum noise disturbance to the surrounding area.
- Test the drums connection to the Igloo pc once all three projectors have been tested. Start with the input reading, then menu selection using all drums, and finally testing with the game and adjust any delay or inconsistency in each drum.

The projectors were the wildcard in the testing due to the inconsistency of the issues. The problem alternated between HDMI cable and the projectors themselves. To study the issue more, we would carry out the testing twice a week since the tent had been assembled and increased the testing up to once every two days a week before school day showcase event. There were many speculations regarding the issues with the flickering screen. At the end, we found out it was caused by faulty HDMI cable. Swapping that out and carrying regular testing ensured that the DAFNI Immersive data space or the tent was in top condition and ready for Game day showcase.

Player models made in Autodesk Maya had this one issue, geometry breaking. We made sure that there was no gap in the models themselves. The druid models that can be seen in game were 'skin' that wrapped around a pre-inserted skeleton system. There was still uncertainty when the models were rigging. Certain movement might bended or twisted the skins in an unexpected way. To test for this, we created a short animation for the skeleton in Maya to follow, which included waving with one hand and both hands, drum hitting motion (moving hand up and down in front of the body), and finally twisting the body from the waist. Any ripped section was patched and merged then retested and send to be test with Kinect motion camera input.

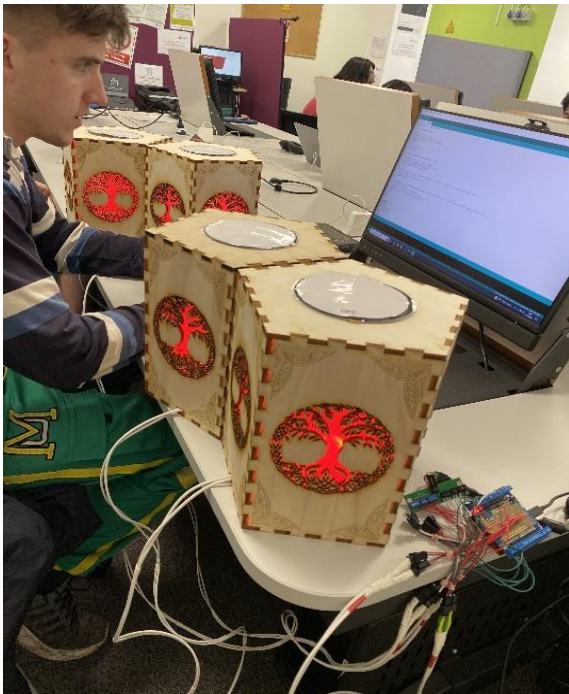


Figure 10: Testing sensors and game input

On top of this, testing of the player tracking was a challenging task. As the nature of the technology dictates, we required the presence of at least three members of the group. This hindered the consistency and availability that the feature could be tested. To address problem of limited opportunities to test compare to other elements within the game, we developed as much of the player tracking aspects as we could without testing. By focussing on building the

feature without testing, we could optimise the time spent when three or more members where available. Also, by creating separate processes for development and testing, we could maintain a constant pace of development, a well as providing a focussed and productive environment to identify and fix bugs.

Every new part of code added had it's functionality tested in an induvially, before adding tests for functionality in the greater system. These tests were then kept in the code for later development to ensure all aspect of the code worked together appropriately. The tests were later removed when a section was deemed 'complete' or functionally 'stable'.

## Blender

In order to develop procedural texture for the cliffs, we used computer graphics software tool called Blender. As the Blender doesn't have a direct integration with Unity, all the textures and maps had to be baked before exporting however it was the optimal solution when it comes to building such things as it offers the ability to work with nodes that allow procedural pattern generation which with other similar programs we either wouldn't be able to use at all or would have to overcompensate with using more complex shader graphs with suboptimal results.

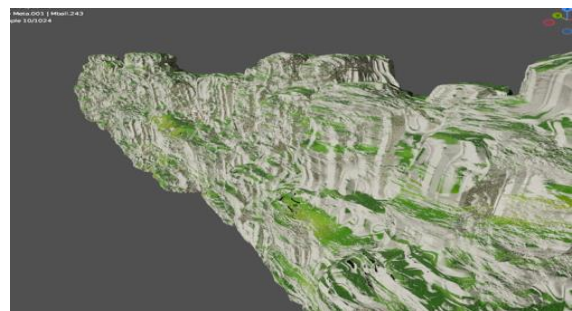


Figure 11: procedurally generated cliff rendered



# Technical Content

## Music Integration

### Audio Analysis

After researching several existing projects which attempt to solve the problem of beat detection and structural analysis of audio files, we based our audio analysis technology on work by McFee et al. [1], making use of their corresponding Python library.

Spectral flux representations (the distribution of energies over a set of frequencies) form the foundation of the audio analysis. To detect beats and transients, both those of lower frequency sounds representing the underlying rhythm and those shorter, higher frequency sounds which often represent off-beat drum hits, this spectral analysis must also be reconciled with a timescale.

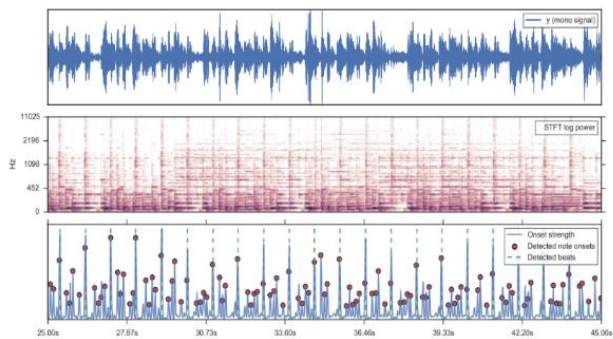


Figure 12: Signal, Spectral, and Onset representations

First, onset detection is performed, which first calculates an onset strength curve from the spectral flux over a series of time frames and then selects peak positions from this curve. The method of peak selection is based on work by Boek et al. [2] which involves applying a logarithmic filter bank to the spectrogram, and then applying half-wave rectification. This effectively divides the onset-strength curve into discrete frequency bands and sums up the positive differences between two consecutive time spectra and, if this meets a given threshold, determines it to be a peak.

The track as a whole is also analyzed to estimate the tempo. This is done by the detection of the recurring structure of the onset curve in a timeframe. This

estimated tempo is used to set the target spacing between selected peaks in the onset curve, from which the underlying 'beats' can be identified.

**Figure 13** shows the raw audio signal, followed by the logarithmic spectral flux, followed by the onset strength curve with detected peaks shown with red dots, and detected 'beats' shown with dotted green lines.

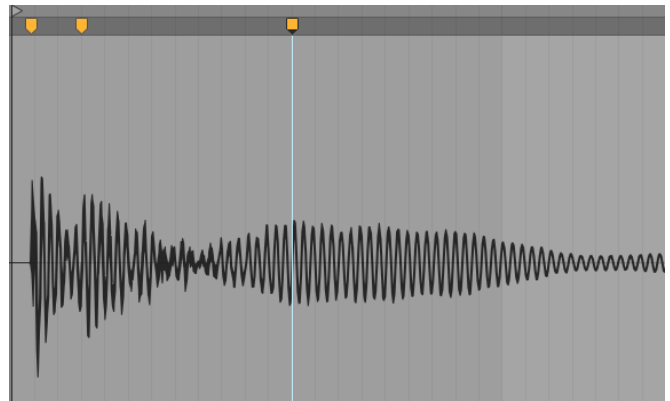


Figure 13: Waveform of a drum hit with possible onset markers.

As this detection method returns onset events as frame indices, we had to make an implementation which would make a conversion to absolute time so that it could be integrated with the game. This implementation effectively creates an array indexed by milliseconds. At each of these time points, it is indicated whether a regular onset, a beat (part of the underlying rhythm) or no event is occurring. This was then stored in JSON format so that it could be loaded in-game.

### MIDI Analysis

MIDI is a technical standard, describing a protocol for interfacing between musical instruments and computers [3]. A MIDI file contains a set of instructions, for example, a 'note on event' with a given velocity, note number, duration, start time etc. An example of a series of events is shown in **Figure 14**. The DryWetMIDI Library [4] was used to parse the MIDI files given to us by the composers. We could then calculate a tempo map from the file, iterate through the notes and calculate the millisecond timestamp at which each note occurred, in

this case, drum hit, as well as its velocity (hit strength) and note number (pitch). From this, we could then form a time-indexed mapping array, analogous to the JSON array described in the previous **Audio Analysis** section.

The use of this MIDI analysis was a later development in the project as it transpired that the composers could supply us with the required specific MIDI files and we therefore weren't reliant on the audio files alone. However, relying solely on MIDI data would have the disadvantage of only indicating the very start of the note onset. With certain drum sounds, this can fail to pinpoint the peak amplitude of the drum. **Figure 12** shows three yellow markers, indicating different time points of a drum hit used in the track for the Mountain Level. The MIDI analysis would always treat the first yellow marker as the 'hit' point of the note, whereas the **Librosa**-based audio analysis could determine it to be any of the three.

Put another way, the MIDI analysis is certain to capture every drum hit and contain the exact pitch and velocity, but possibly with some discrepancy concerning the time difference between note onset and peak onset strength. And the Librosa analysis is **almost** certain to capture every drum hit and will pinpoint the time of peak onset strength. It also allows general use in the sense that we can use it to analyze any audio file, without relying on bespoke MIDI files which aren't always available.

Our solution was to marry the two, and use an average timestamp of each drum hit alongside the pitch and velocity information provided by MIDI.

## Beat Spawning

With an accurate array of drum onsets, timestamped by milliseconds, we were then able to create beat spawning in time with the music. This array is queried every frame update with the current playback time in milliseconds - offset by the time it would take for a descending marker to reach the target area - and if a drum hit is indicated at that time, a marker can be spawned in with its attributes such as its size and colour corresponding to the stored information (inc. velocity and note number).

Due to the time between frame updates varying and generally being more than a single millisecond (but less than 10), the array would be queried up to 10 milliseconds on either side of the current time to ensure a beat wouldn't be missed. Due to the resulting overlap between the bounds of consecutive queries, a found note would then be indicated as used, to avoid spawning the same note twice.

MetaEvent DeviceName SmartMusic SoftSynth 1	start : 0	delta : 0	Note Ch : 1 key : 67	: VEL 96 : [3072 - 4096]
MetaEvent SequenceName Instrument 2	start : 0	delta : 0	Note Ch : 1 key : 67	: VEL 96 : [4096 - 5120]
CC Ch: 1 C: MAIN_VOLUME value: 101	start : 0	delta : 0	Note Ch : 1 key : 66	: VEL 96 : [5120 - 6144]
CC Ch: 1 C: PANPOT value: 64	start : 0	delta : 0	Note Ch : 1 key : 62	: VEL 96 : [6144 - 7168]
ON: Ch: 1 key: 67 vel: 96	start : 3072	delta : 3072	Note Ch : 1 key : 64	: VEL 96 : [7168 - 7680]
OFF: Ch: 1 key: 67 vel: 0	start : 4096	delta : 1024	Note Ch : 1 key : 62	: VEL 96 : [7680 - 8192]
ON: Ch: 1 key: 67 vel: 96	start : 4096	delta : 0	Note Ch : 1 key : 60	: VEL 96 : [8192 - 9216]
OFF: Ch: 1 key: 67 vel: 0	start : 5120	delta : 1024	Note Ch : 1 key : 62	: VEL 96 : [9216 - 10240]
ON: Ch: 1 key: 66 vel: 96	start : 5120	delta : 0	Note Ch : 1 key : 64	: VEL 96 : [10240 - 11264]
OFF: Ch: 1 key: 66 vel: 0	start : 6144	delta : 1024	Note Ch : 1 key : 67	: VEL 96 : [11264 - 12800]
ON: Ch: 1 key: 62 vel: 96	start : 6144	delta : 0	Note Ch : 1 key : 66	: VEL 96 : [12800 - 13312]
OFF: Ch: 1 key: 62 vel: 0	start : 7168	delta : 1024	Note Ch : 1 key : 67	: VEL 96 : [13312 - 14336]
ON: Ch: 1 key: 64 vel: 96	start : 7168	delta : 0	Note Ch : 1 key : 66	: VEL 96 : [14336 - 15360]
OFF: Ch: 1 key: 64 vel: 0	start : 7680	delta : 512	Note Ch : 1 key : 67	: VEL 96 : [15360 - 17408]
ON: Ch: 1 key: 62 vel: 96	start : 7680	delta : 0	Note Ch : 1 key : 66	: VEL 96 : [17408 - 18432]
OFF: Ch: 1 key: 62 vel: 0	start : 8192	delta : 512	Note Ch : 1 key : 62	: VEL 96 : [18432 - 19456]
ON: Ch: 1 key: 60 vel: 96	start : 8192	delta : 0	Note Ch : 1 key : 67	: VEL 96 : [19456 - 20480]
OFF: Ch: 1 key: 60 vel: 0	start : 9216	delta : 1024	Note Ch : 1 key : 65	: VEL 96 : [20480 - 21504]
ON: Ch: 1 key: 62 vel: 96	start : 9216	delta : 0	Note Ch : 1 key : 67	: VEL 96 : [21504 - 22528]

Figure 14: Example series of MIDI messages

This also provides a solution to the unlikely event of more than one frame update occurring in a single millisecond.

## MIDI Input

To handle MIDI input from the Arduino hardware, we made use of both Unity's Input System framework [5] and the Minis plugin [6]. This allowed us to create a listener for MIDI 'Note On' events which would then map the 'Note Number' to a drum index and assign an updated input 'velocity' or hit strength to that drum in-game to be checked in the main update cycle and handled appropriately.

## DAFNI Immersive Data Space or Igloo Tent

### Structure

The tent is a 3x3 metre space with metal frames, which is constructed of two pieces of frame bolted together (a total of six metal frames, two per side), on three sides and a blackout curtain as the front entry side.

The inside screens consist of two parts. The black protective section goes on top and the white section on \which the screen is displayed. It was crucial to prevent the white section from becoming dirty or damaged during our use of it. These screens are fitted into metal frames.

The top part of the tent is kept together by cross beams, which consist of one cross-type beam and four metal beams that are held together by hammering in the bolts.

Hanging from the cross beams are three projectors pointing toward the back, left and right screens. Power cables run through the roof opening along the cross beams. The AV cables also go through the same opening, into the Igloo pc which was set up on the right side of the tent.



The outside screens are simply for safety and protection against the metal frames.



Figure 15: Inside DAFNI Immersive tent

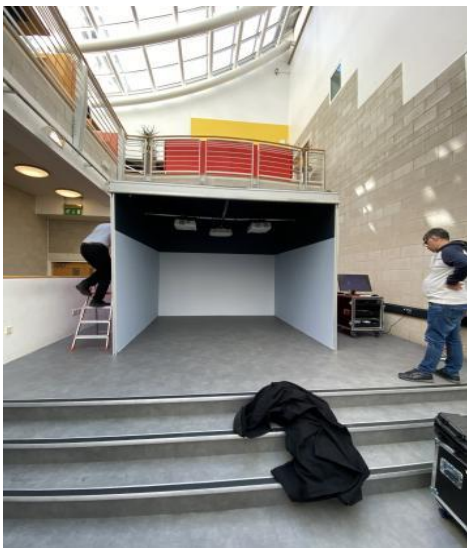


Figure 9: The DAFNI tent during the construction

## Igloo PC

The pc itself is a strong computer consisting of two SSD drives, 0.5 TB of C drive and 1 TB of D drive. The connection to the internet happens through a WAN ethernet socket (the router provided DHCP to the Igloo network on the access point side). All the software required for this is already preinstalled on the desktop.

## Sound System

The DAFNI uses the ‘Creative Lab SoundBlaster X4’, which is connected to Media Server via a USB. We manually installed the driver for the SoundBlaster ourselves. The other audio devices were muted to prevent audio from coming out of the projectors themselves or the control monitor.

The rear audio outputs on the sound card are connected to the amp or audio power amplifier in this configuration. Front speakers (including front left, front right and front centre) go into channels 1 & 2. The centre/subwoofer goes to channels 3 & 4. The rear speakers (including the left rear and right rear) go into channels 5 & 6.

The outputs on the amp then connect to the speakers in this configuration. Channel 1 to the front left. Channel 2 to the front right. Channel 3 to the front centre. Channel 4 is bridged to the subwoofer. Channel 5 to the rear left and Channel 6 to the rear right.

By setting the sound system up in this configuration, the music is properly localised and all the acoustic sources are uniform for the centre-positioned audience.

## Drum sensors



Figure 17: Piezo element left, FSR right.

As a rhythm game focused on drums, we had to develop a means for capturing players hits with a high degree of accuracy. We researched existing solutions and found three common approaches to sensing impacts. The first used microphones and extracted hits by filtering the audio out for amplitudes above a threshold. All three methods used a thresholding approach. However, with up to 6 active drums as well as game audio it was impractical to reliably extract peaks produced from a single drum. The last two approaches measured the force exerted on a surface or vibrations induced. Where force-sensitive

resistors and Piezo elements are the most affordable implementations for musical applications.

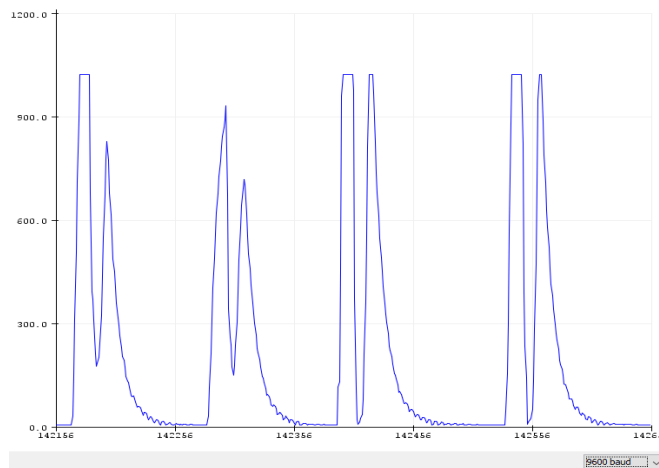


Figure 18: Piezo impact signals

To determine the most effective solution we used an Arduino Uno microcontroller to collect the analogue output from both sensors. Arduino [5] is the most popular open-source electronics platform, making it easy to write and upload scripts to any supported microcontroller. Each Piezo element generates an electrical charge when subjected to pressure or force, which can be used to measure the force applied to each drum. Testing showed that a direct impact produced multiple peaks. Oscillations can be caused for the following reasons: mechanical vibrations, sensor overload and signal amplification. Where multiple peaks cause double hits to be registered in-game, affecting playability.

Another important factor in choosing a sensor type was zero return. This is how quickly a sensor output returns to zero after a peak. Fast zero return decreases the time restraint between drum hits, allowing us to use higher bpm tracks in-game. To improve the zero return, we compared signal



Figure 19: 4th iteration FSR



20: 4th iteration FSR drum implementation

waveforms using a range of pull-down resistors. Most of the examples we found used drumsticks to trigger signals. They create a much larger pressure due to the small impact surface area. Reproducing similar forces using players' hands was uncomfortable and unrealistic for less able players. Equally, the only sensor sizes available were too small making it difficult to reliably play while looking at a screen. Fabricating larger versions would have been impractical given the time restraint and materials. Comparing the results in Figure 18 and Figure 19 we can see FSRs produce cleaner and shorter peaks without needing additional components. However, with a diameter of ~1cm, it was too small to start building a drum around. To develop our own, we researched what materials were needed and found that commercial FSRs had a simple design consisting of 3 layers. A single semi-conductive layer, a conductive layer and a spacer to prevent constant. When force is applied the semi-conductor becomes thinner, reducing resistance and increasing the voltage across an external resistor. Acting as a voltage divider circuit. Initially, we developed prototypes using readily available anti-static bags as a semiconductor and copper tape for the conductors. We wanted to achieve a sensor with a larger surface area for players to effectively



Figure 21: 5th iteration FSR

interact with while playing. While also being sensitive enough to detect small forces, accommodating kids and fatigue after playing long games.

We went through several prototype stages varying semiconductor thickness, copper tape width, sensor shape, spacer thickness and methods used to connect wires. Producing a reliable, repeatable design required tweaking all variables in the design. Figure 21 shows the 4<sup>th</sup> iteration where we layered copper tape onto a card to create conductive surfaces. Overlapping the tape caused small increases in height at different positions across the surface. Which lead to false positives in sensor output, to combat this we added foam offsets separating each layer. However, this was ineffective due to the conductive surfaces not being ridged enough to maintain separation during playtesting.

Our first drum body implementation used bins, leveraging the plastic lips to provide an offset for foam inserts to separate sensors. Each drum is connected to an analogue pin on the Arduino Uno. Hit signals were then encoded into the serial output of the Uno and received by Unity. Another issue we encountered at this stage was sensor consistency. With each sensor being handmade and highly sensitive, we found the resting resistance varied massively. To solve the inconsistency, we made our final iteration using sheets of copper and laminating each layer together. Using sheets eliminated the variance in conductive surface height between each sensor. Additionally, laminating helped maintain a constant force between layers which also reduced variance in resting resistance. Each layer was cut using a bladed compass and laminated using conventional A4 laminators. Adding copper legs provided a surface to solder wires to each surface for a consistent connection.

## Signal filtering

Reading and filtering all six analogue signals with low latency was equally important in achieving a responsive feel to the game. Our initial Arduino script relied on setting a constant voltage threshold to trigger a serial signal to be forwarded to Unity. When a player hits the sensor, it induces mechanical vibrations into the surface of the drum. Translating into oscillations in the analogue signal. Producing false positives that lowered players' scores and the rhythm of the game. Another challenge was extracting the peak value from the signal produced.

Traditionally this involves recording the signal until it decreases below the threshold to obtain a peak value. During playtesting we found some players would take

longer than others to take their hand off the drum after each hit. Often taking over 100ms before a hit signal would be relayed into the game. Making drums feel unresponsive and the music out of beat. To solve this, we took a dynamic approach to signal peak extraction and hit consistency, developing our debouncing software.

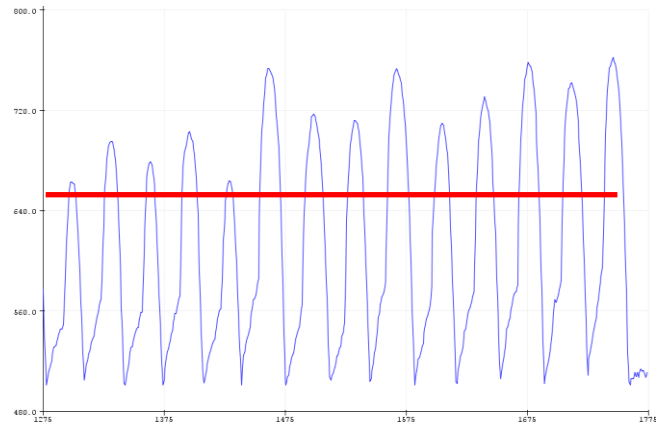


Figure 23: Signal thresholding

Figure 23 shows an example of the signal produced from a single drum hit. When the voltage increases past a threshold the signal delay starts, lasting 25ms. During this time the script records the maximum voltage read. After 25ms a hit signal is forwarded to Unity, encoding the drum number and peak value recorded. We tuned this delay to extract a realistic peak value within an industry-standard delay of 40ms. Allowing an additional 15ms for Unity to receive and add a response in-game. Our script then waits for the signal to fall below the threshold before starting the debounce delay, lasting 30ms. Which prevents false positives from being recorded as the player takes their hand off the drum. Again, we tuned this delay to be short enough to not hinder follow-up hits and also prevent false positives. We can see in Figure 23, peak 4 would have been registered as an additional hit if a debounce delay had not been used. After numerous rounds of in-lab and group playtesting we found our

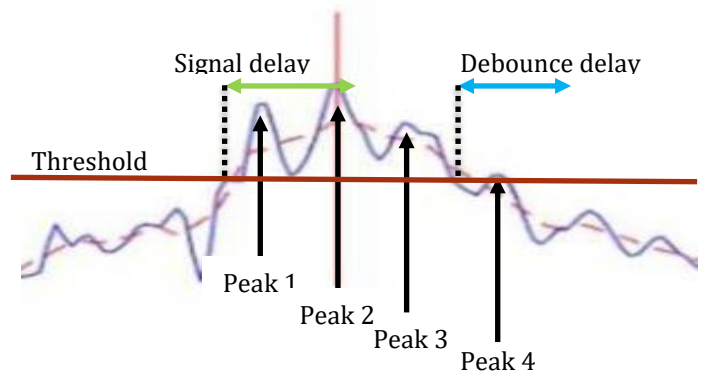


Figure 24: Signal filtering visualisation



design reduced false positives by ~95%. Additionally, the constant signal delay increased our max bpm by over 100%.

## Micro-controller & Lighting

We found that the Uno's limited 16MHz clock speed and serial buffer caused it to crash during playtesting. Instead, we switched to a Teensy LC microcontroller with a 48MHz clock. Allowing us to also drive six led strips through independent digital channels. Lighting up each drum with animations to match a character in-game added to the overall experience. To connect each sensor and LED strip, totalling 12 connections, we used a breakout board with screw terminals. After testing a range of LEDs and configurations we chose Neopixel addressable LED strips. Neopixel supports the FastLed library, providing asynchronous updates to each LED strip. Lowering impact on sensor performance.

When it came to installing the LEDs, we can and soldering the trips into triangles to distribute the light evenly within the drum body. The next stage was connecting each drum's electronics to the drum controller in a modular way. Modularity meant that if a single drum's electronics became faulty the rest would not be affected. With only 4 core cable available and 5 connections coming from each drum we opted to double up the cable. The benefit to this was that we now had an 8-core connection. That allowed us to double up wires with each cable, reducing resistance for the LED power

lines. This was crucial because we purposefully made each drum cable 2m in length to allow for different spacing between players on drums day. With each LED requiring 30mA at peak load, we needed at minimum an 8A power supply for safety. After soldering on 4-pin connectors we had a fully modular 6 drum controller. We made a simplified component diagram shown in Figure 25 using Fritzing [10].

## Drum body

Matching the theme of our game was important when designing a drum body. We chose the Tree of Life to be our logo, making that the main theme of the box. Starting with a PNG image, we made modifications to add support structures so that it would stay in one piece after laser cutting. We then converted from PNG to SV, providing a defined outline for the laser cutter. To design a template for each box, we used MakerCase [8], creating a pentagonal shape. Including edge joints for added strength. Additional modifications were made to the SVG files in Fusion360 [9], adding cutouts for our sensor connectors and main cables, and providing power and data lines.

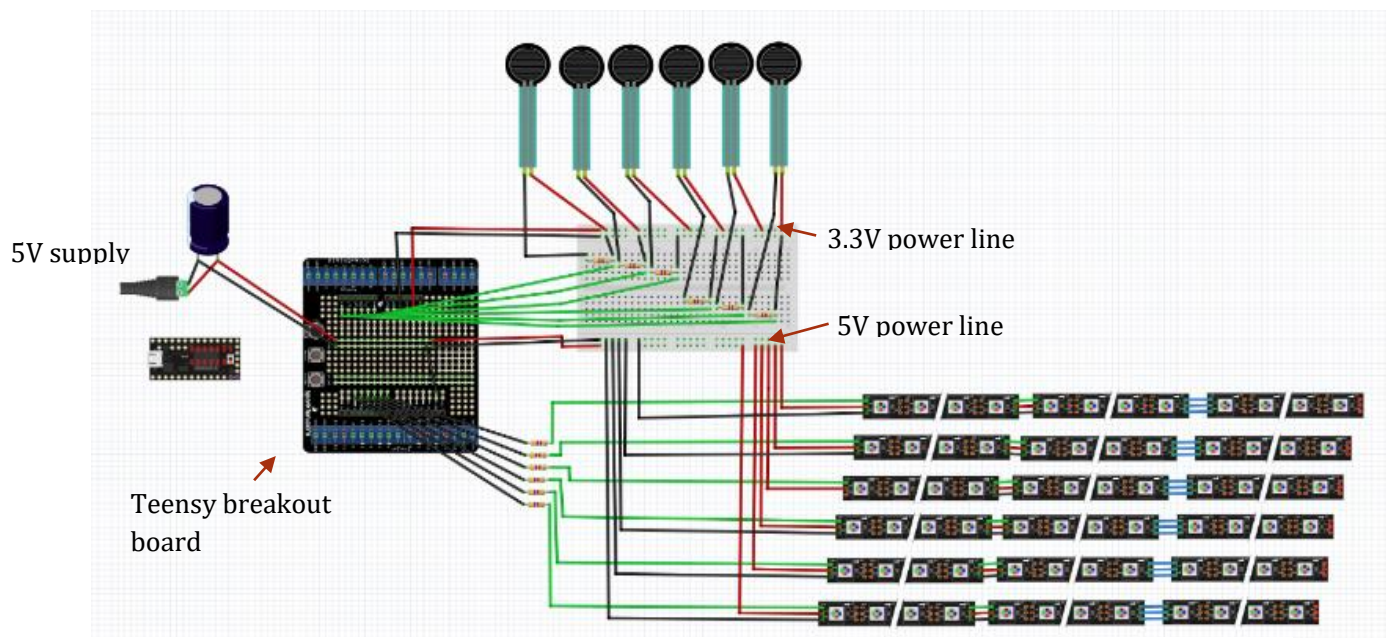


Figure 25: Component diagram

Finally, we combined our box templates and custom designs using CorelDraw. Designs were then cut and engraved from plywood using the Trotec laser cutter.

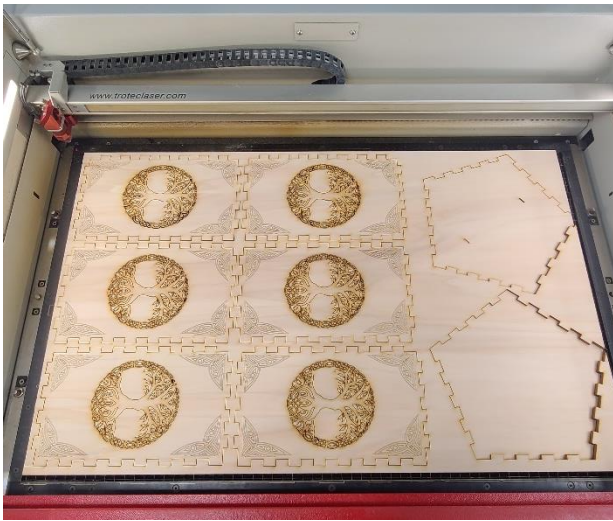


Figure 26: Trotec laser cutting process

To add support to each Tree of Life pattern we laser cut acrylic circles, acting as windows into the box. For all trees to have a glowing effect we sanded down the acrylic panels, diffusing the LED light. During construction, we used hot glue to secure each component to the inside of the box. As a final addition to make playing the drums more comfortable, we cut disks of silicon to pad each sensor.



Figure 27: Final drum assembly

## Procedural Trees

A key component of our game is the players' interaction with the environment through growing trees. We want each tree to be unique and to have an organic look as it grows. To that end we decided to procedurally generate the trees at run time so the trees are always different in every run of the game.

### Tree growth model

After researching several methods of procedural generation, we decided on implementing the generation model outlined in this Jinmo Kim's paper on modelling trees for use in virtual reality[11]. The advantage of this approach is that it allows for a high degree of control over the tree's structure and shape, as well as the ability to introduce natural variations and randomness.

The skeleton of the tree model is generated using an implementation of L-systems. Each tree is composed of branches that have two vector components: a position

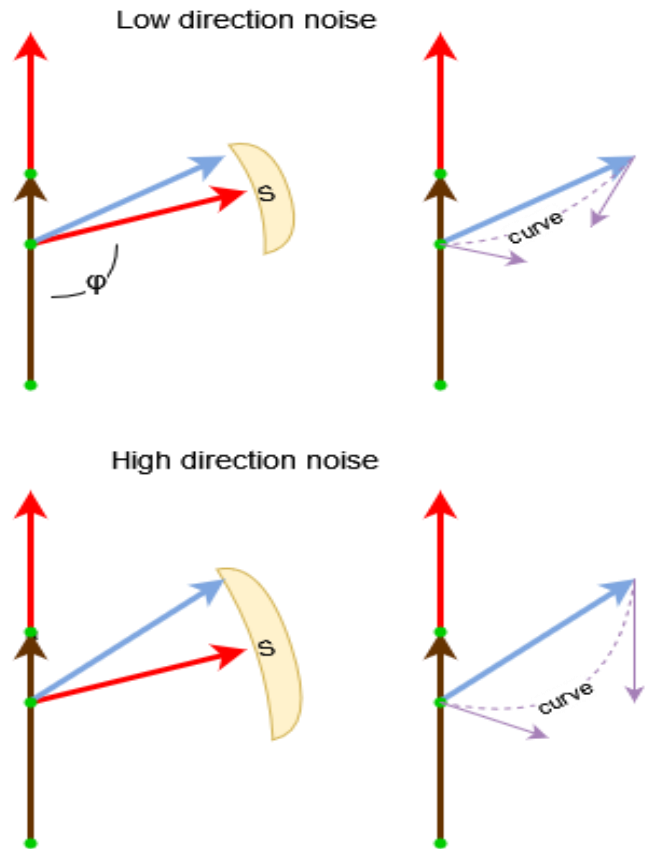


Figure 28: Effects of direction noise on tree generation.  $S$  is the space where the growth vector can be in;  $\varphi$  is the angle of the ideal growth vector to the parent branch.

vector, and a growth vector. The branches are organised in a binary tree structure, with one child branch having its position at the end of the parent branch and the other having its position vector along the growth vector of the parent branch, as shown in figure 29. For branches that grow from the side of the parent, the position vector is randomised to create more variation between the trees.

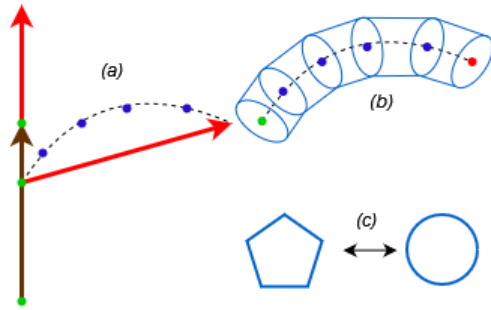


Figure 29:

- a) Sample points along the branch's curve
- b) Volume of a segmented branch

There are several parameters that can be changed to alter the tree's appearance, the most impactful of which being the length and width of the branches, their orientation, and the direction noise. While it is clear what how changing a branch's length, width, and orientation will affect its shape, the direction noise influences the variation in the shape of the branches. It represents how far the angle of the growth vector can deviate from its ideal value, as well as the distance of the Bezier curve's control points from the ends of the branch.

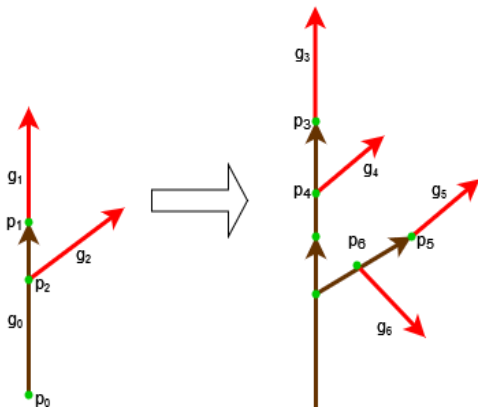


Figure 30: The structure of a tree and the vectors that make it up.  $p_i$  is the position vector;  $g_g$  is the growth

Figure 28 depicts how direction noise affects the size of the space that the growth vector can occupy, as well as the curvature of the branch.

## Mesh Generation

All the meshes of the trees are generated at runtime since it offers the most flexibility and the greatest amount of variation in their shape. A branch's mesh is created by sampling its curve at equidistant points, generating a layers of vertices in the shape of rings around them, and then connecting these layers.

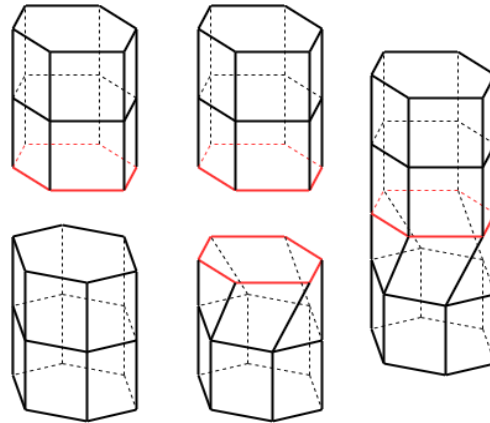


Figure 31: The process of copying the last layer of vertices from the branch above.

Figure 30 illustrates how sample points define the segments of the mesh, and the number of vertices in each ring determines the number of faces that a branch has. As the resolution of these points increases, the geometry closer approximates the desired curvature, but it comes at a higher computational cost.

To balance visual quality with performance, the number of segments and faces is dynamic and depends on the tree's distance from the camera.

The trees around the players have a much higher level of detail, and as the camera moves the mesh generation resolution adjusts automatically. In practice, the transition is almost seamless and it greatly improves performance since there are fewer polygons to render and fewer vertices to compute for the trees that occupy less space on the screen.

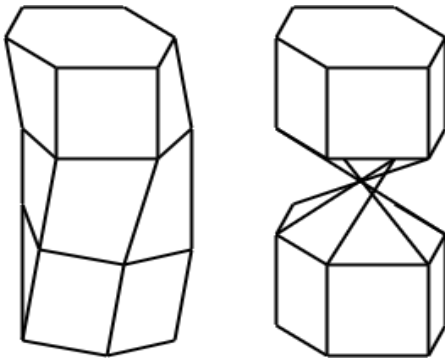
Overall, the process of mesh generation was the most challenging aspect to integrate. Since every branch handles its own mesh, aligning two branches end to end proved to be a complex task, as it required ensuring that the vertices and normals of the two meshes were



seamlessly connected. Our solution was to copy the last layer of vertices from the child branch to the parent, as shown in figure 31.

Another issue we encountered when generating meshes is “pinching”, depicted in figure 32. This occurs when two layers are rotated beyond a certain degree and the edges that connect them cross over.

To solve this problem we implemented a systems that connects the layers by picking the vertices that are closest to each other, thus minimising rotation.



32: Example of “pinching”

On the left: acceptable amount of rotation between layers

On the right: layers rotated beyond acceptable level

## Animation

Our trees differ from the ones described in mr. Kim’s paper[11] through the fact that they are dynamic, and grow. This effect is achieved by scaling the growth

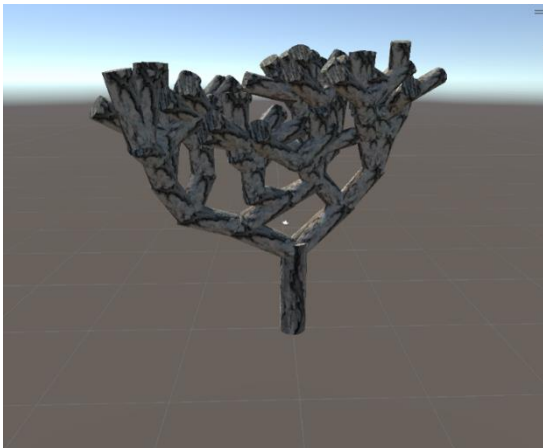


Figure 33: Early tree iteration

vector of every branch until it has reached its full length and recalculating the mesh. When a branch is fully grown, new branches are added until the tree has reached the maximum depth. The rate at which the animation progresses is dependent on the score multiplier, so the players have a real effect on the environment.

## Design

Creating procedural trees is as much an art as it is a science. Our trees have gone through many iterations before arriving at their current look.

When texturing the trees we decided on using realistic textures to give the trees more life and a sense of presence in the game.

Further, foliage components were designed in Maya in order to match different enviromants. In the process we designed cherry blossom flowers, oak and palm leaves. During the development we’ve tested different iteration of each and dedided that finalised versions would be clusters of oak and cherry blossoms respectively and single palm leaves.

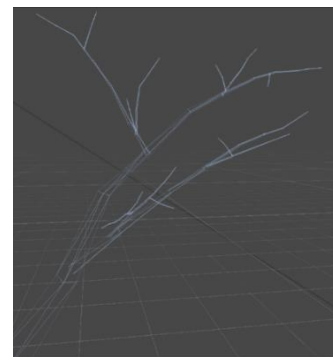
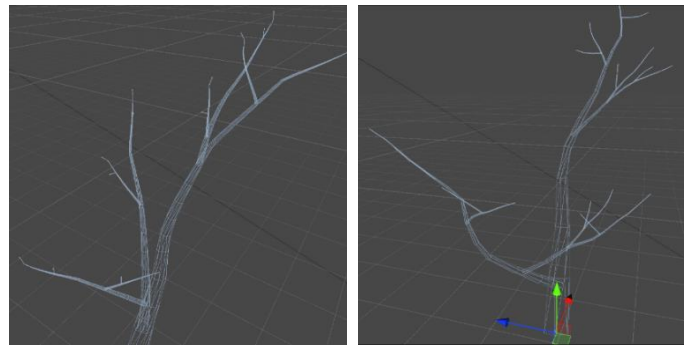


Figure 35: Tree meshes at different distances from the camera.

## Integration

Trees spawn around the platform duing gameplay. The spawning system generates both procedural trees and static tree models at regular timings. The placement



location is determined using different rules for each level in order to avoid collisions with other assets. Through the level, the forest grows around you, but this process is expensive and can seriously impact performance when there are many trees growing at the same time. We addressed this issue by implementing a queueing system for tree spawning so only a certain number of procedural trees can grow at once.

## Player Tracking

We understood that tracking players and replicating their movements in the game would enhance the overall experience. Our initial plan was to use Openpose [11] with a webcam, expecting this to be the simplest method to implement this. However, after development of a system that could take real-time position data from Openpose and using this to animate 2D skeletons we decided that having 2D avatars in a 3D would detract from the immersive experience promised by this technology. This led to us attempting to explore approaches to transform this 2D skeletal data into data that could be used to animate 3D character models. During this research, it became apparent that to apply a process that could achieve this for three players in real-time would have much too high latency to perform well. Thus, we had to rethink our approach on tracking players movements.

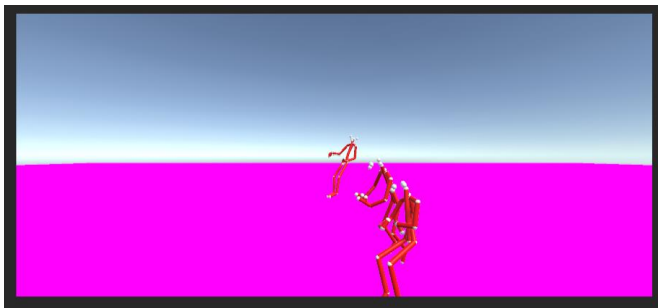


Figure 36: Multiple skeletons being rendered inside Unity

We decided to trial an Azure Kinect [12] as our next step. The Azure Kinect was promising as it has integrated body tracking capabilities, so it was the next logical step in our development process. The Kinect combines a depth sensor and a camera to effectively detect human bodies within the frame. The players movements can then be transferred into the Unity game engine by using an API [13]. Within Unity, the data can then be rendered into skeletons that realistically imitate players. The hips are the foundation reference point for the skeletons, each sequential bone from the hips has a

local rotation to determine its exact position in that frame.

We have to accommodate three individual players within the game, so we designed the scripts to sort players by distance and render only the three closest players into the game environment. Due to the setup of the players within the Immersive Igloo, the three people playing the game will always be the three closest. However, the representation of the skeletons was not visually pleasing. To combat this shortcoming, we rigged character models with pre-existing skeletons. The character models can then emulate movements of detected players. Whilst, the characters models are not as accurate of a representation of character movements, that fact is balanced by a more visually appealing experience.

Furthermore, we implemented a component to ensure that players are aligned with their corresponding in-game characters. When a player is on the right hand side of the physical setup, then to maintain a cohesive experience the associated avatar is also on the right side of the screen. This can be achieved through comparing distances as the Azure Kinect is off-centre, so each player will be definitively closer or further away and their representation in-game can be adjusted based on this.

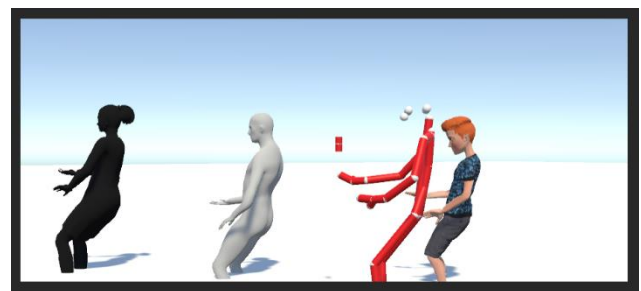


Figure 37: Models rigged to mimic skeleton of a player

# References

[1] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in Python." In Proceedings of the 14th Python in science conference, pp. 18-25. 2015.

[2] Böck, Sebastian, Florian Krebs, and Markus Schedl. Evaluating the Online Capabilities of Onset Detection Methods. In 11th International Society for Music Information Retrieval Conference (ISMIR 2012).

[3] MIDI, <https://en.wikipedia.org/wiki/MIDI>

[4] DryWetMIDI, <https://github.com/melanchall.drywetmidi>

[5] Unity Input System, <https://docs.unity3d.com/Packages/com.unity.input-system@1.5/manual/index.html>

[6] Minis, <https://github.com/keijiro/Minis>

[7] <https://www.arduino.cc/en/software>

[8] <https://en.makercase.com/#/>

[9] <https://www.autodesk.co.uk/products/fusion-360/personal>

[10] <https://fritzing.org/>

[11] Kim, Jinmo. (2016). Modeling and Optimization of a Tree Based on Virtual Reality for Immersive Virtual Landscape Generation. Symmetry. 8. 93. 10.3390/sym8090093.

[12] Openpose, <https://github.com/CMU-Perceptual-Computing-Lab/openpose>

[13] Azure Kinect, <https://learn.microsoft.com/en-us/azure/kinect-dk/>

[14] Azure Kinect Unity API, [https://github.com/microsoft/Azure-Kinect-Samples/tree/master/body-tracking-samples/sample\\_unity\\_bodytracking](https://github.com/microsoft/Azure-Kinect-Samples/tree/master/body-tracking-samples/sample_unity_bodytracking)