FACULTY OF ENGINEERING, COMPUTING AND THE ENVIRONMENT

School of *Computer Science & Mathematics*

BSc DEGREE
IN
Computer Games Programming

PROJECT REPORT

Name: Maciej Baranowski

ID Number: K2120853

Project Title: DJ Simulator

https://github.com/LightBlazeMC/dj-simulator
https://lightblazemc.github.io/gamedev_djsimulator.html

Date: 24/April/2025

Supervisor: Andreas Hoppe

Kingston University London

Plagiarism Declaration

The following declaration should be signed and dated and inserted directly after the title page of your report:

Declaration

I have read and understood the University regulations on plagiarism and I understand the meaning of the word *plagiarism*. I declare that this report is entirely my own work. Any other sources are duly acknowledged and referenced according to the requirements of the School of Computer Science and Mathematics. All verbatim citations are indicated by double quotation marks ("…"). Neither in part nor in its entirety have I made use of another student's work and pretended that it is my own. I have not asked anybody to contribute to this project in the form of code, text or drawings. I did not allow and will not allow anyone to copy my work with the intention of presenting it as their own work.

Date: 24/April/2025

Signature: 

Table of contents

Plagiarism Declaration **2**
Declaration *2*
Table of contents **3**
Introduction **5**
Aims and objectives *5*
Personal developmental goals *5*
Summary of upcoming analysis *5*
State of the Art Review (Literature Review) **6**
Niche review *6*
Beat Saber VR *7*
Vinyl Reality *7*
GTA Online – Nightclub Missions *8*
Product analysis review *9*
Tools and Technologies **9**
Game engine review *9*
Secondary features review *10*
Entertainment review *11*
Development cycle review *11*
Conclusion of Literature review *11*
Analysis, requirements specification and development methodology **11**
Game engine analysis *11*
Secondary features analysis *12*
Project specific functional & non-functional requirements *13*
Requirements prioritisation *13*
SWOT analysis *13*
Risk and Contingency planning *14*
Development methodology analysis *14*
Conceptual Game Design **15**
Product overview *15*
Game outline *16*
Gameplay *17*
Level design *18*
Character design *19*
Graphical User Interface design *19*
Music library planning *20*
Meeting requirements via design *20*
Technical Game Design **21**
Class structure *21*
Animation state machines *22*
Component interfacing *22*
Implementation **22**
Early prototype stage *22*
Interim prototype stage *25*
Final prototype stage *27*
Testing and Evaluation **29**
Black box testing *29*
Functional testing *30*
Player feedback *30*
Key observations *31*
Legal, Social, Security and Ethical Issues **31**
Copyright & licencing issues *31*
Data protection & security *32*
Ethical & social impact & implications *32*
Critical Review and Conclusion **33**

Summary of achievements *33*
Personal experiences *33*
Conclusion *33*
Technical Appendices **33**
External assets sources *33*
References **34**

Introduction

DJ Simulator is a strategic simulation game made in Unity 2023 LTS. The game has been developed mainly on macOS, however anticipated platform deployment is both Windows and macOS. The game is set in a low-budget nightclub, where the player plays as the DJ who must entertain the crowd with good music choices.

Aims and objectives

The primary aim of the game is to perform excellent DJ sets that score plenty of points within a 5-minute time limit. In order to gain points, the player must ensure that they are selecting tracks that are of a genre that is the most popular within the crowd.

The main game objective is to score the most points within the 5-minute time limit. In order to score the most points, the player must first identify the most popular genre within the crowd. Once identified, the player must play a track of that genre in order to start accumulating points. The crowd will react accordingly. The player must continue to monitor the crowd, due to the fact that the crowd will change in numbers. As the popular genre changes within the crowd, the player must choose new music that fits accordingly. Once the 5-minute timer elapses, the game will notify the player of their score. Not selecting appropriate genres, or not changing the music based on the crowd's changes in favourite genres will not award many points.

Personal developmental goals

When embarking on this project, I had 3 main goals in mind:

- To create a convincing simulation of tasks & experiences of a typical beginner DJ
- To realistically simulate crowd behaviours that can be observed within nightclubs
- To simulate a low-budget nightclub environment that feels 'alive' and responsive to the player's actions.

Being a DJ myself, I intended to create a game that is able to simulate and showcase the typical workflow and tasks that a beginner DJ in a low-budget venue would experience. This is an experience that many DJs can relate to, and in order to deliver on a genuine representation of this experience, I set the 3 main goals above as my developmental goals since I believe that achieving these well will result in a realistic depiction of the intended experience.

Summary of upcoming analysis

This report will delve into an analysis of how this project was developed and formed to achieve the personal developmental goals outlined above. It is important to analyse both the infancy and late development stages of the project thoroughly.

The early planning stages will be covered in the form of a Literature Review, where sources of inspiration will be explored. The information collected from this research will then be thoroughly analysed and translated into a development methodology and requirements specification, where a clear developmental plan can be established.

At this point, clear requirements and goals will be apparent, therefore conceptual and technical game design can be discussed. A deep insight into the reasoning of how the game is designed and why those design choices have been made is crucial, due to the many intentional

mechanics that are implemented in order to create an immersive simulation. Many of those design choices are implemented in unique and abstract way, therefore a detailed analysis of the technical implementation is also necessary.

Since this game contains a diverse simulated system, thorough testing is necessary. The report will detail how this has been achieved to ensure that the game's simulation systems are robust, and as many simulation cases as possible have been tested. Legal and moral issues will also be considered here.

Finally, the report will conclude with a critical evaluation and conclusion of the information and research conducted during the development of this project. This will be followed by technical appendices and references.

State of the Art Review (Literature Review)

Niche review

DJ Simulator aims to depict the humble beginnings of a new DJ. Most DJs, when starting their career, will perform in low-budget, run-down venues [7], therefore it is important that this project depicts the gritty atmosphere well. There are certain industry standards within the DJ world, such as the hardware, which often tends to be Pioneer CDJs or Technics SL-1200s [8]. It is important that the game represents those expected industry standards faithfully in order not to alienate the player – which is highly likely to be (or aspiring to be) a DJ.



Figure 1 – A typical Pioneer DJ setup.

Not many gaming products exist within the niche of the DJing world. There does not seem to be a strong link between video games and DJing. Despite that, several gaming products and

generic niche conventions have been found and analysed. While no singular product has satisfied the entirety of the developmental objectives of this project, the following products each contain features that have proven insightful for the development of this project. Identifying those, and adapting them for the needs and goals of this project will help create a cohesive DJing experience. Beat Saber VR has been selected as an example of reactive environment design, Vinyl Reality has been selected as an example of realistic DJ hardware simulation, and finally GTA Online has been selected as an example of a realistic crowd simulation system.

Beat Saber VR



Figure 2 – In-game screen shot of Beat Saber VR.

Beat Saber VR is a music rhythm game where the player must break blocks with their sword in beat with the music. This game features heavily adaptable environments, which change based on the player's actions and song choice. Visual effects may appear or morph when the player triggers an event system [1]. This means that the game adapts visually based on what they player is doing and how well the player is doing.

This level of visual feedback is a great device for signalling important gameplay messages to the player, and creating an immersive, engaging and 'breathing' in-game world [2]. Based on this observation, it has been decided that in aid of improving environmental immersion and player feedback, audio reactive lighting will be implemented into the project.

Audio reactive lighting is a very typical element of most conventional nightclubs [3], therefore including it in the game is a great way of combining visual feedback, environmental immersion and generic convention within the DJ niche; most DJs would have had experience performing in a venue with reactive lighting.

Vinyl Reality



Figure 3 - In-game screen shot of Vinyl Reality (via Steam Store page).

Vinyl Reality is a game that deeply focuses on track selection, and the performance of said track. It is a deep emulation of a vinyl-based DJ system. Being developed by professional DJs [4], this product provides an excellent representation of typical tasks for a performing DJ. Most of the expected DJ system mechanics are thoroughly and realistically portrayed within this product.

The implementation of allowing the player to drag-and-drop music [5] on the turntable is an excellent way of giving the player control over the in-game world. This product is a strong source of inspiration for the simulation of the interactable in-game systems within the project. It allows for realistic interfacing and control over the in-game world, while retaining it within a format that is very synonymous within the DJ niche.

The turntable depicted within Vinyl Reality is the Technics SL-1200. It is a turntable that is a staple within the world of DJing, and has been around since the 1970's, making it a crucial piece of hardware ad heritage within the DJ world [6]. More importantly, the selection of such widely recognised hardware means that most DJs can easily interact with the in-game depiction of the turntable, since it is highly likely that they have used the real-life counterpart. Using this renowned system reduces the risk of confusion or alienation of the player, by utilising a familiar layout and system.

GTA Online – Nightclub Missions



Figure 4 – A crowd and a DJ inside the Music Locker nightclub in GTA Online.

The nightclub missions form a relatively small element within Grand Theft Auto Online's expansive and busy world. They are however a perfect example of excellent crowd simulation within a nightclub environment. There is a multitude of NPCs that have unique behavioural patterns [9], and enhance the mise-en-scene of the nightclub drastically.

While in GTA Online the crowd is mostly utilised for an aesthetic purpose, within DJ Simulator the crowd will provide visual aid for the player. The player should be able to identify popular music genres by looking at the crowd.

A realistic implementation of crowd simulation within the nightclub can greatly benefit the project by highly enhancing the environment (and its realism) and also as a means of providing visual feedback to the player on their performance and areas of improvement.

Product analysis review

To summarise, based on this analysis, utilising the reactive environmental systems of Beat Saber, the realistic crowd simulation systems from GTA Online nightclubs, along with the detailed, recognisable and robust hardware simulation systems of Vinyl Reality, is highly beneficial for creating a lively and detailed simulation of a DJ setup in a nightclub. Combining these systems creates a synergy between functional, immersive, 'breathing' environments and recognisable and realistic DJ hardware. These sources of inspiration provide an insight into the ingredients required for a robust, practical and non-alienating simulation of a DJ's environment within a nightclub.

Tools and Technologies

Game engine review

There are several essential features that the game engine of choice must provide for this project:

- High performance – large crowds and real-time visual effects can be computationally expensive. Therefore, a well optimised engine is needed that will not suffer from performance issues when simulating detailed scenes.
- Powerful scripting – there are many features that will need to be implemented that require complex logic to work. The engine must provide useful tools and libraries in order to make this process easier.
- Stability – since this project is not utilising any novel/evolving concepts (such as generative AI), an engine version with the greatest stability is of highest priority.
- Versatile animation system – in order to achieve realistic crowd simulation, an engine that can handle and calculate animations well is important.
- Environmental sensing – due to the large number of stimuli within the scene, the engine must be able to identify and act accordingly to player and environmental triggers.

Based on this list of requirements, two prospective engines have been considered: Unity 2023 LTS and Unreal Engine 5. Both of these engines satisfy the majority of these requirements much better than other game engines such as Godot or GameMaker, which either lack the necessary functionality, or are not as capable with it.

Unity is known for excellent performance, even with complex scenes. Therefore, scenes with large crowds should not cause performance issues for this engine. C# is a very powerful scripting language and is the one used by Unity. C#, along with MonoBehaviour [10] provide extremely powerful scripting opportunities. The 2023 LTS version of Unity provides all necessary features while being very stable. No preview (or more recent) version is needed here, and the project will benefit from using the very stable LTS branch. Unity's Animator is very powerful and will allow for complex NPC simulations. Unity's colliders and map constructing systems work very well with events and triggers.

On the other hand, Unreal Engine provides high fidelity graphics at ease [11], albeit with a large performance cost. The usage of C++, which is a very powerful language, along with the versatile and practical Blueprint system offers huge scripting capabilities. Unreal Engine offers no LTS solution, and crashes (especially on macOS) are very frequent. Unreal Engine provides a very powerful [12] suite of animation and simulation tools. They are very detailed, however overkill for this application. Unreal Engine's environment sensing is excellent, especially when paired with the Blueprint system.

Secondary features review

Due to this project being a DJ Simulator game, audio analysis algorithms are very important here. Unity offers the GetSpectrumData function, which provides the block of audio frequencies (spectrum data) of the AudioSource that is currently playing [13]. This is essential for the reactive crowd and lighting that is envisioned within this game.

A complex AI pathfinding, such as Dijkstra's Algorithm may not be necessary for DJ Simulator. A random crowd distribution model can instead be opted for since it provides realistic and convincing randomised NPC behaviours. Furthermore, the traversal of the NPCs is not important in this game as much as the actual portrayal of them on the dancefloor.

To simulate NPC behaviours, an animation state machine can be opted for. This is because the animation state machine system within Unity allows for various animation triggers, based on Boolean states, and also enables smooth transitions [14]. These features provide smooth and convincing crowd NPC behaviour modelling.

Entertainment review

It is important that DJ simulator is a fun game. According to Raph Koster in his book "A Theory Of Fun For Game Design", a fun game can be defined as one that is "a source of enjoyment" [15]. He states that "fun is the act of mastering a problem mentally" [15]. The problem in this scenario being the goal of entertaining the crowd with the right music. Based on this, it can be concluded that DJ Simulator is a fun game due to the fact that it provides the player "a source of enjoyment" [15] by allowing the player to "master a problem mentally" [15].

Development cycle review

For a successful and efficient development cycle, it is critical that the project follows a development methodology. For this project, there are two methodologies that could be considered: Agile and Waterfall. Waterfall is "particularly useful for large, complex projects with very specific and unchanging requirements" [16], whereas Agile "works well for projects that prize learning and are seeking or refining their product" [16].

Conclusion of Literature review

Based on this literature review, the project now has information based on the niche market of DJing and also a technological and developmental overview of potential possibilities and opportunities that can be used for the development of DJ Simulator.

Analysis, requirements specification and development methodology

Having considered the research conducted within the literature review, it is now essential to streamline this information into segments that will be advantageous for this project. Having multiple approaches for many tasks means that now the best approach must be considered and selected.

Game engine analysis

Selecting the appropriate game engine is crucial. While both engines provide excellent opportunities for developing a good DJ simulation game, the engine of choice is Unity 2023 LTS. Based on the analysis conducted within the literature review, a table weighing the pros and cons of each engine has been created. Using a table easily illustrates the greater advantages of Unity over Unreal Engine for this application.

	Unity 2023 LTS	**Unreal Engine 5**
Performance	Due to lesser (unnecessary for this use case) graphical fidelity, this engine provides better performance	The high-fidelity graphics are very computationally heavy.
Scripting	C# and MonoBehaviour – provides all necessary	C++ and Blueprints – provides plenty of additional functionality. However, it is

	functionality for this project. The scripts compile quickly on play.	not needed for this project. C# in-engine compilation is not straightforward.
Stability	LTS version offered – provides excellent stability and no unexpected updates.	No LTS version. Very unstable on macOS.
Animation/simulation systems	Robust suite of animation tools. Everything needed for a realistic crowd/NPC simulation.	Overwhelming suite of tools. Many of which are not needed for this application.
Environmental sensing	Great interaction between colliders, triggers and MonoBehaviour scripts.	Great implementation of Blueprints.

Looking at the data in this table, it is can be observed that Unity 2023 LTS should be the engine of choice. It fulfils all requirements, without much extra bloat. Being an LTS release, it should not cause down-time or issues related to unexpected behaviours or updates.

Secondary features analysis

Unity provides the GetSpectrumData function [14]. This in-engine function provides a solution for reactive lighting. Other than this, either manual keyframing of waveforms would be necessary, or a highly inaccurate estimated down-beat calculation based on the track's BPM (BPM/60). Using manual keyframing is extremely time consuming and means changes in music will not be updated within the engine automatically. Using the BPM down-beat calculation is an extremely flawed solution, since it assumes the drum track is of a constant, never-ending 4/4-time signature duration, and estimates the music's downbeat based on the BPM/60 calculation, intending to estimate exact time markers of down-beats. Not much music exists composed in this monotonic way. Furthermore, due to audio latency, this calculation would quickly become desynchronised. Therefore, GetSpectrumData is the best choice, allowing for automated real-time audio waveform data processing, with drag and drop analysis based on file importing. This method, while difficult to initially develop, ensures that all critical music analysis can be done without much more development efforts, and it is done extremely accurately based on each individual music file's waveform data.

Since Unity provides a robust Animator state machine [12], this is the best solution to use. It is built into the engine, and allows for Boolean conditional scripting, that calculates what animation the NPC should perform based on Boolean conditions. External animation rigging could have been used via Mixamo or After Effects, however that would be a tedious manual process, would not integrate within Unity's scripting and would not be directly controllable by the engine. Therefore, no external tools are necessary beyond the Animator state machine provided by Unity.

Due to the game revolving around music, and specifically track selection by genre, it is critical that the player is provided with a diverse library of music. For legal purposes (which will be later discussed), the music must be copyright free. Some of the music can be composed by me, using my strong experience with music production. However, producing a good quality takes time, therefore original content can be combined with the multitude of royalty-free music libraries accessible via YouTube, SoundCloud and other sources.

So that the game can be fun, it must allow the player to “master a problem mentally” [15]. This problem can be presented via a timed points system. This provides the player with a time limit to achieve the best possible performance. This mechanic is a good choice since it closely models the way a real-life DJ has to “master [the] problem mentally” [15]. It is a great visual indicator for the player to see how well they are performing. Combining the fun factor and effective visual feedback of this mechanic makes it a great choice for this project, while invoking a sense of urgency and providing the player with a clear target, purpose and objective.

Project specific functional & non-functional requirements

- Based on the conducted research and analysis, the following functional and non-functional requirements have been identified:
- Powerful, high-performance game engine
- Realistic and reactive crowd behavioural simulation animator & state machine
- Reactive environmental lighting elements
- Real-time music analysis algorithm
- Diverse, multi-genre, royalty-free music library.
- Simulated DJ hardware in accordance with industry standards
- Game logic that assesses the player’s performance and returns feedback via points
- A convincing & immersive map
- Straight forward visual cues on preferred crowd music genres

Requirements prioritisation

Not all of these requirements will be able to be fully met. The MoSCoW system is a great way to quickly organise priorities of requirements. Based on the analysis of this project, it is important to focus on core gameplay and simulation features. The game must have a purpose, and a means of executing that purpose. Using a MoSCoW analysis table allows the development process to focus on critical features first, and then implement less important features later. This means that the final product will not be missing any of its core features.

MUST HAVE	SHOULD HAVE	COULD HAVE	WON’T HAVE
Diverse music library	Audio-reactive environment	Vinyl emulated slow-down/speed-up	Multiplayer
Controllable DJ decks	NPC animation	Controller support	Music importing
Reactive crowd	NPC engagement logic (leaving/ entering the room)	Tutorial	
Points system		Multiple maps	
NPC balancing manager		DJ mixer emulation	

SWOT analysis

Conducting a SWOT analysis greatly helps ensuring there is no unexpected issues arising or any other factors that can result in project downtime. Considering how complex of a system this project is, avoiding downtime and development-halting issues is a must.

Strengths	Weaknesses	Opportunities	Threats
• Utilising version control. • Knowledge of niche.	• Time constraints • Lack of knowledge	• Documentation • Tutorials	• Software issues – engine crashes, bugs. • Lack of implementation. • Time limitations. • Data loss

Risk and Contingency planning

Based on the SWOT analysis above, several risks and threats have been identified. The project's success is heavily dependent on the resolving and mitigation of these issues, to ensure no developmental downtime occurs.

Using a legacy Long Term Support (LTS) branch of Unity has been an intentional decision in order to mitigate software issues. When beginning this project, Unity 6 Preview has been available for use. However, upon testing and exploration, it has been found that it offers no real advantages for the requirements of this project. Instead, it contains many undocumented issues and is much less stable, causing frequent crashes. Using a 2023 LTS version of Unity is an excellent balance, where it is recent enough to still be supported, and mature enough to have rich documentation and bug/crash fixing patches.

A major threat for this project has been a lack of implementation. This means that certain requirements outlined for DJ Simulator require solutions and implementations that are not well documented and do not provide much support. Real-time audio analysis is an example of a requirement that may suffer from the threat of lack of implementation. There are not many resources available that cover advanced levels of audio analysis within Unity. However, using my extensive experience in audio sciences and theory, I have been able to adapt formulae from real-life audio applications into DJ Simulator.

A very frequent threat for large-scale projects is the lack of time, or poor time management. This can cause serious problems to the quality of the final product. In order to mitigate this, the project has been begun early. Furthermore, using a high-end MacBook Pro laptop means that development can be made anywhere, thus providing much more opportunities to work on the project.

When working within the digital realm, data loss is always a threat. It can cause severe down-time and loss of progress. In order to mitigate this, GitHub version control has been utilised alongside a backup on two independent SSD locations. This means that should any data loss occur; a backup will be readily available.

Development methodology analysis

As discussed earlier, two development methodologies have been considered for this project: Waterfall and Agile. It is important to consider which methodology allows for the most efficient usage of development time. Agile is the ideal development methodology for this project. This is because of the nature of this project being feedback heavy and iterative cycles being

essential on this scope of work. Due to the frequent interim demos, drop-in sessions and showcases, following an Agile methodology, such as the Scrum framework is beneficial since it enables continuous improvement and encourages rapid and flexible response to change [17]. Considering the experimental and research nature of this project, going for a static approach like Waterfall would result in time being wasted, and features not being implemented based on feedback. There are many situations where the static nature of the Waterfall methodology would result in missed opportunities in improvement and feedback [18].

One of the leading Agile-based frameworks that is favoured by game developers [18] is the Scrum framework. While this is a team-based framework, it can be adapted for solo development [19] by adapting the roles to be fulfilled by one person. This modified solo Scrum methodology can then be planned out in a project management platform such as Microsoft Project, Monday or Trello. In this case, Trello has been used due to its easy and free online application. Microsoft Project is not available on macOS, and on Windows requires extensive setup. Monday does not have a free package that is useful for this project.

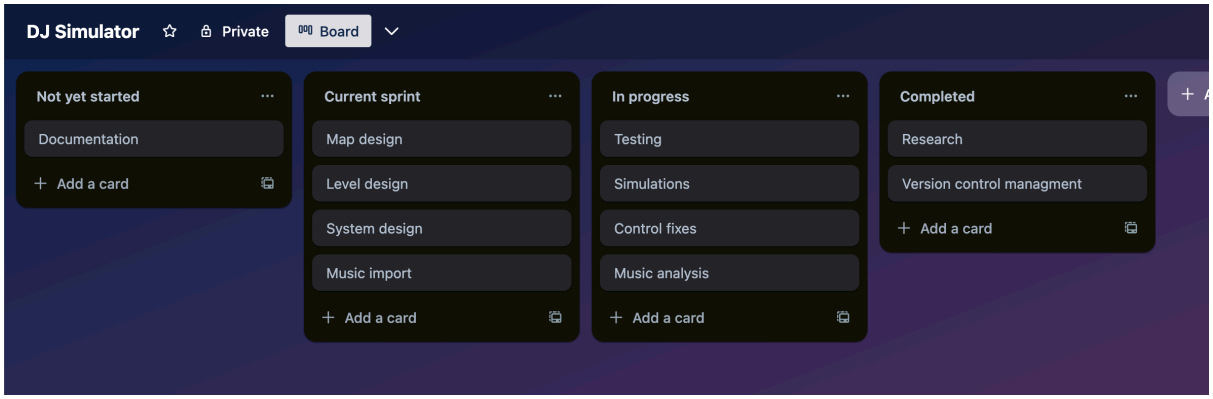


Figure 5 – an extract of the solo Scrum Agile project plan via Trello.

Now that the project contains a clear Scrum vision mapped out on Trello, the iterative development can begin. Feedback can be collected and improvements can be made. When research breakthroughs are made, Scrums can be organised, and the findings can then be added into the future development cycles, and then implemented into the project.

Conceptual Game Design

Product overview

The project is titled DJ Simulator as a homage to the vast array of simulation games, such as Microsoft Flight Simulator or Goat Simulator. The target age is PEGI 7. No themes of violence, gore or foul language are expected to be present in the game. The reason for a rating of ages 7 or higher is due to the depiction of nightlife themes. Being able to keep the age rating low enables the product to be accessible to a wider audience. Being a PEGI 7 game gives young kids/teenagers who are too young to experience nightlife and venue DJing a chance to immerse themselves into the DJing niche. There is no necessity for any adult content, and steps were taken in order to keep it this way, such as a lack of profanity within music, the avoidance of suggestive portrayal of in-game characters, and the omittance of any alcohol or drug related stereotypes that are frequently associated with nightclubs and nightlife. Omitting these controversial topics of nightlife has been done in such a way that does not hinder the immersion of the game, yet enables the age rating to be kept as low as possible.

Game outline



Figure 6 – a simple diagram explaining how to succeed at the game.

The player is spawned into the club, and the club is then populated with a randomly generated crowd, consisting of colour-coded NPCs that enjoy House, Techno or Jungle music. The player then has 5 minutes to play music that aligns with the genre that will satisfy the crowd the most.

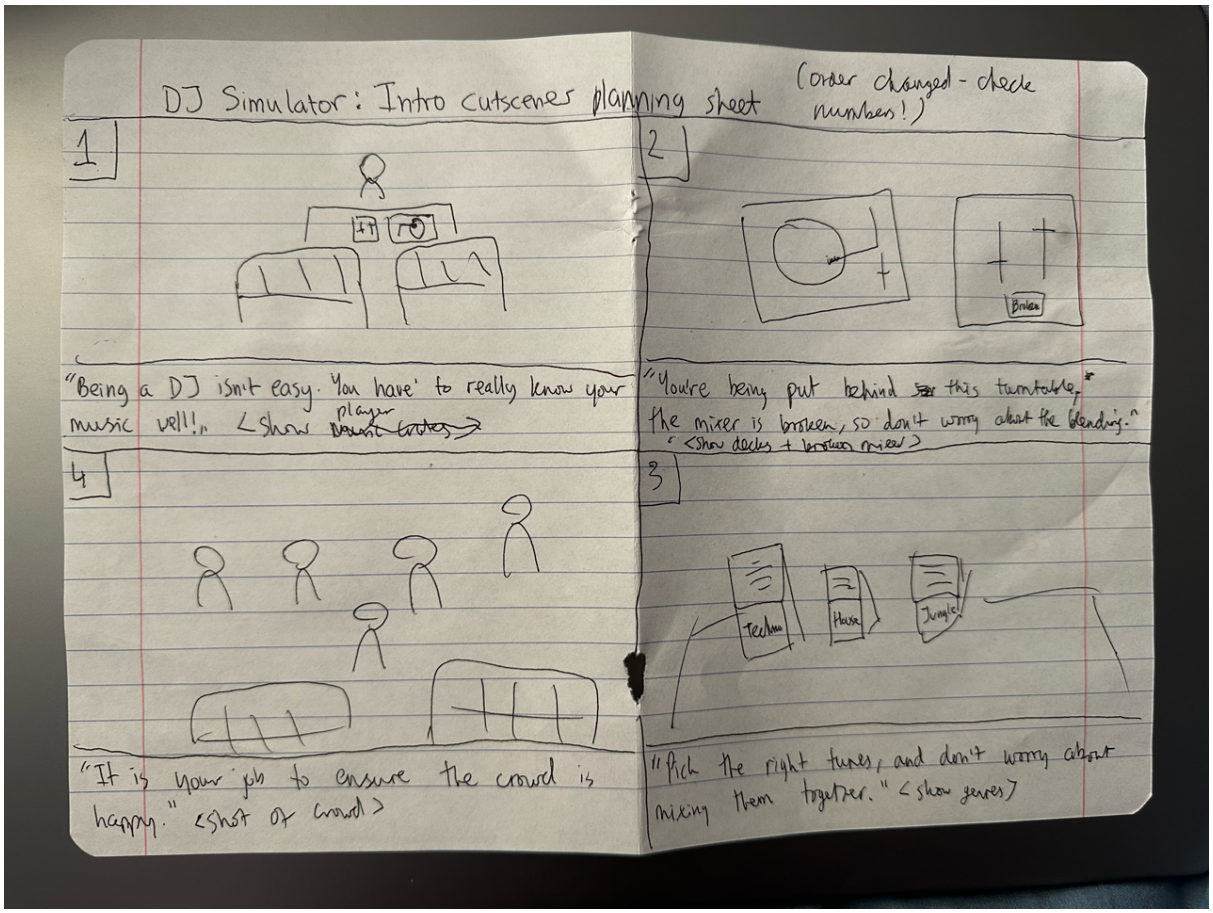


Figure 7 – storyboard planning paper.

Due to the near-sandbox simulator qualities of the game, no major story has been implemented into the game. A simple, 4 frame cutscene plays at the start of the game in order to provide context and purpose to the player. Other than that, it is assumed that the player already exhibits a level of interest into the DJing niche, and no further gripping story enticement is necessary. After the brief context-setting cutscene introduction, the player is thrown straight into the game. Considering how simple the story is, a simple storyboard has been designed on paper in a 4-panel format, since this illustrated the vision in the most direct and detailed way possible.

Gameplay

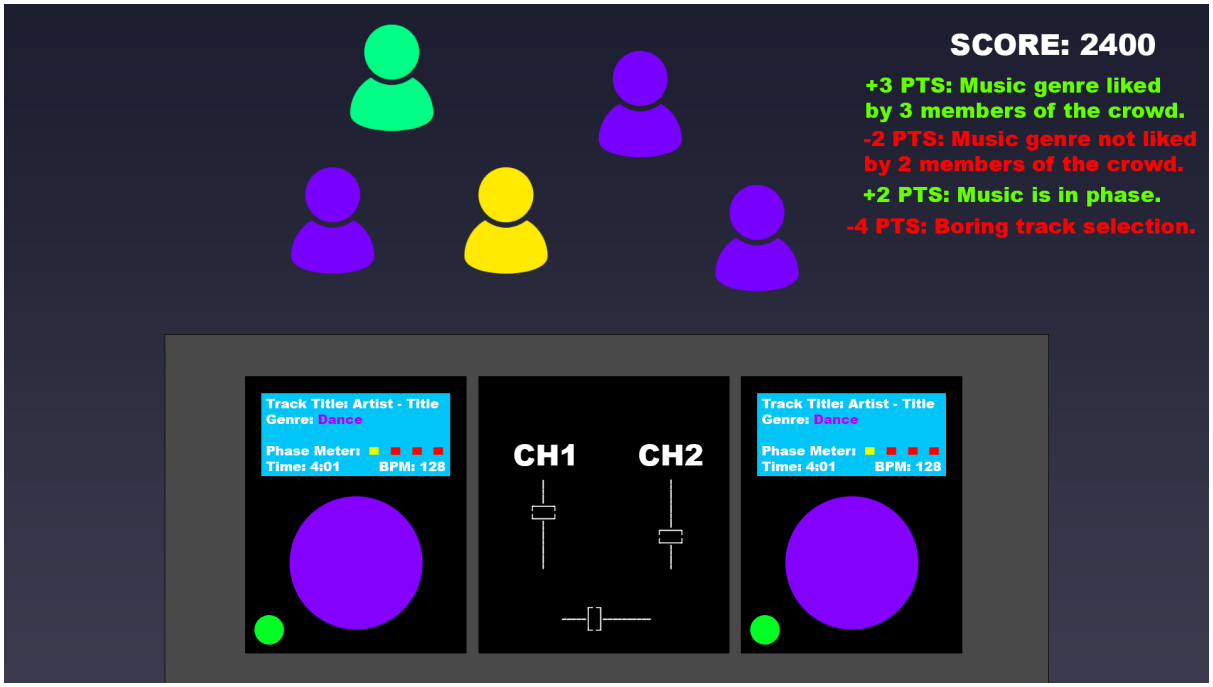


Figure 8 – early concept diagram.

The player's view throughout the game will be of the DJ decks and of the crowd. The crowd will be colour coded based on the music that they like. The tracks that are being loaded onto the decks will also be colour coded based on their genre. The aim of the game is to score as many points as possible during a timed DJ set. Points are awarded by playing music that the crowd enjoys (see *Figure 6.*).

There is only one, time-limited round. However, based on feedback, a few different play modes have been implemented. The main game mode (which will be mostly the focus within this report) shows a tally of the current crowd population, as well as the score list. The hard game mode is more realistic since it removes the tally of the crowd population. This is more realistic with real-life DJing scenarios, where the DJ would not have such a tool, and would be required to analyse the crowd manually. Finally, the free-play mode removes the time limitation, so that the player can practise, explore and play without competitive game objective constraints.

When spawned in, the player must analyse the crowd and then select music from one of the three music genre crates. Once selected, the player must then load the track onto the deck, play it, and observe the crowd's reaction and the feedback provided by the points list. Occasionally, to encourage adaptability and creativity, the crowd will reset. The player must then repeat the process. When the timer ends, the player will be provided with their score, showcasing how well the player managed to impress the crowd.

The game contains no enemies, unless the timer can be considered one. There are no hazards or power-ups. These features are normally reserved for action/adventure games, and most simulator games do not require such features. For example, Microsoft's hugely successful Flight Simulator series, which can be considered a benchmark for simulation games, does contain any of these features for the sake of realism.

Level design

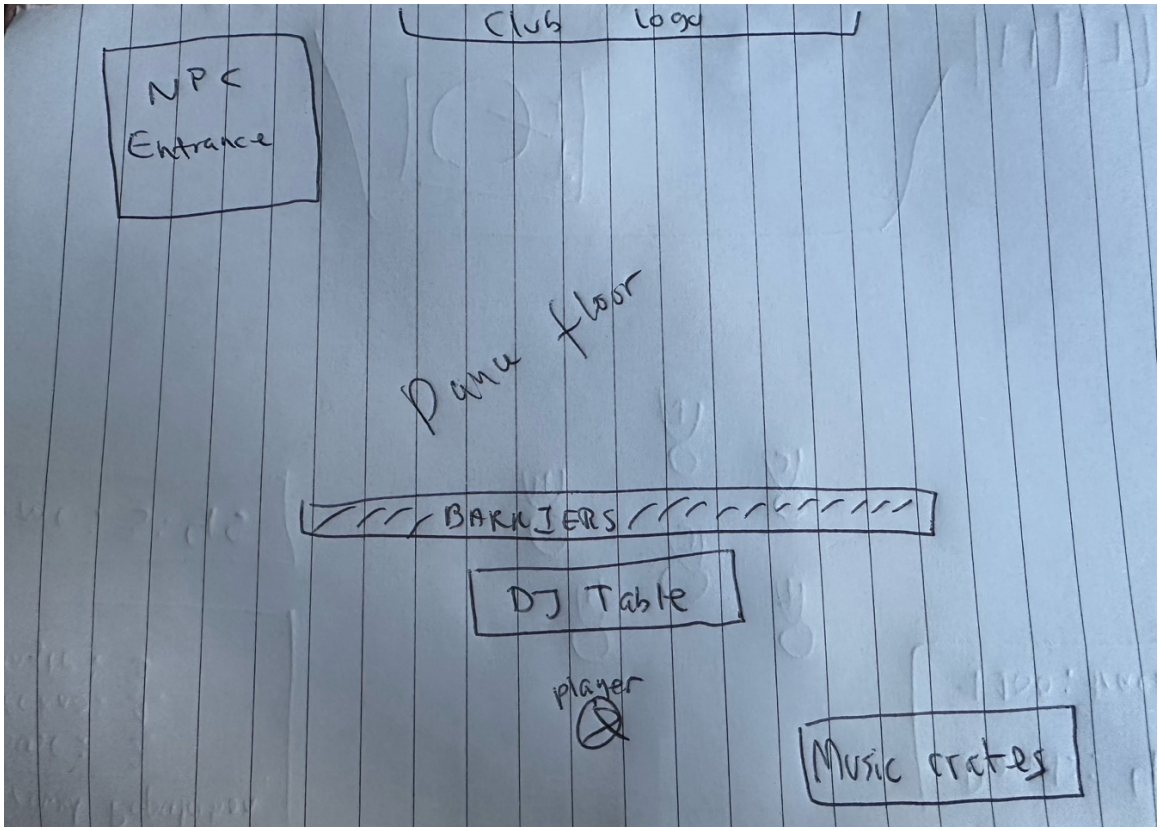


Figure 9 – rough sketch of the map.

The game world is designed to model a run-down, low-budget nightclub, which is in line with the experiences most beginner DJs have at venues [7]. The choice of minimal level design solidifies the low-budget, run-down nature of the club, and ensures that the player is not distracted from analysing the crowd. As expected, based on the contextual initialisation the player receives from the introductory cutscene, the player will find their DJ decks along with the crowd in front of them, and the music crates next to them. The player can also see the entrance, where NPCs will enter and join the crowd.

All of the DJ tools are designed in a way to be close, visible and accessible to the player as to not hinder the player's performance within a timed setting. Likewise, the crowd is made to be strongly spaced out from any other objects on the map, so that the player can easily monitor and assess the crowd.

Club logos and decorations are played occasionally around the map in order to appeal to the low-budget aesthetic of the game world. It gently enhances the detail while not distracting the player from the action. Similarly, barriers are placed to ensure that the player cannot free roam beyond the intended map boundaries, and because barriers typically appeal to the mise-en-scene of most nightclubs. These features ensure that the level is functional, immersive but not too bust or cluttered. There is also a rotating disco ball present, which also increases immersion due to being typical nightclub furniture and is used as a diffusion surface for the club's lighting system.

Character design

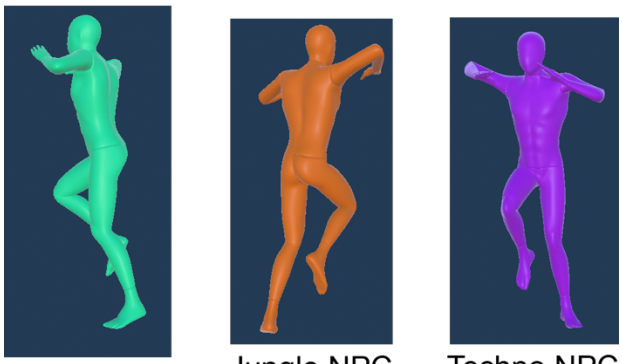


Figure 10 – design of in-game NPCs.

As with many other design elements within the game, the NPCs are designed in a way that is functional, but not distracting. Utilising ambiguous models ensures no provocative or offensive imagery, and also promotes inclusivity by not alienating in-game characters to any specific demographic. The NPCs are colour coded based on the genre they enjoy the most. Being such a simple design means that the player can instantly see what genre the NPC enjoys. This is an example of efficient and functional visual cues.

The player never will see their own player model due to the first-person perspective of the game. Therefore, to not introduce distractions, no visible player modelling exists within the game, and even shadows for the player are removed.

Graphical User Interface design

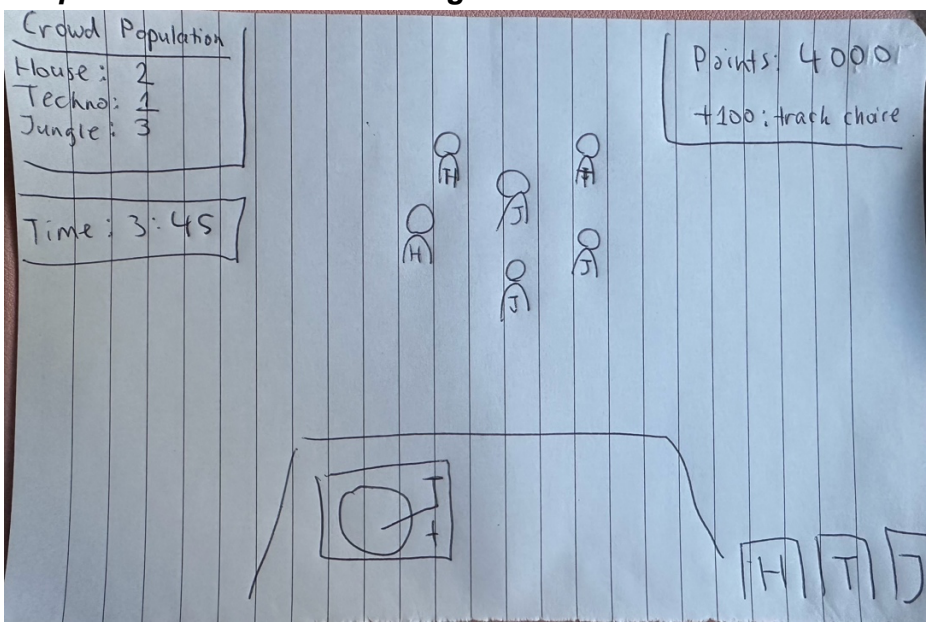


Figure 11 – rough plan of visual cues.

DJ Simulator must be able to provide a large amount of data in visual cues for the player. A sketch has been made purposefully for how the GUI can be used to provide and assist with visual cues *(see Figure 11)*.

The Points List is there to show the player what the game has rewarded the player for. It serves as a points breakdown and feedback system informing the player of any mistakes that they may be making that can cost them points. This is a very useful performance indicator and teaching tool showcasing how the player can improve and gain more points.

The Crowd Population tally (not available in hard mode) is an assist for the player, designed to help navigate the crowd. Dependant on the situation, the crowd may sometimes be very tightly packed together, which can make it difficult to assess the situation. It also serves its purpose from an accessibility perspective, allowing colour-blind players to use the Crowd Population tally instead of relying on colours they cannot see.

Music library planning

ID NUMBER	GAMEOBJECT NAME	GENRE TAG	TITLE - ARTIST	LICENSE TYPE
1	Techno001	TECHNO	Populationcull - Justin Tuijl	Royalty free
2	Techno002	TECHNO	Dia de Dios - LightBlazeMC	Original content
3	Techno003	TECHNO	Don't Stop (Tempo Remix) - Akudama	Royalty free
4	Techno004	TECHNO	DASQ – Techno Power	Royalty free
5	House001	HOUSE	LightBlazeMC - Turok	Original content
6	Jungle001	JUNGLE	Sappheiros - Universe	Creative Commons (BY 3.0)
7	Jungle002	JUNGLE	Ripple – On Your Mind	NCS – Copyright free
8	House002	HOUSE	Vidaloca - Housess	Royalty free
9	House003	HOUSE	Jorja Smith x Preditah - On My Mind (Imacci Remix)	Original content

Figure 12 – table holding data about music tracks within the game.

It is crucial that DJ Simulator contains a vast library of royalty-free multi-genre music. In order to keep track of this, a table has been made that keeps holds information about: the index of the music within the list array of AudioSources, which GameObject the audio file (and its derived UI elements, such as album art) is attached to, the internal genre tag assigned to the track, track title & artist, and also licensing terms. This allows for good organisation within the game's music handling logic, and it also allows for an overview of any copyright issues that may arise.

Meeting requirements via design

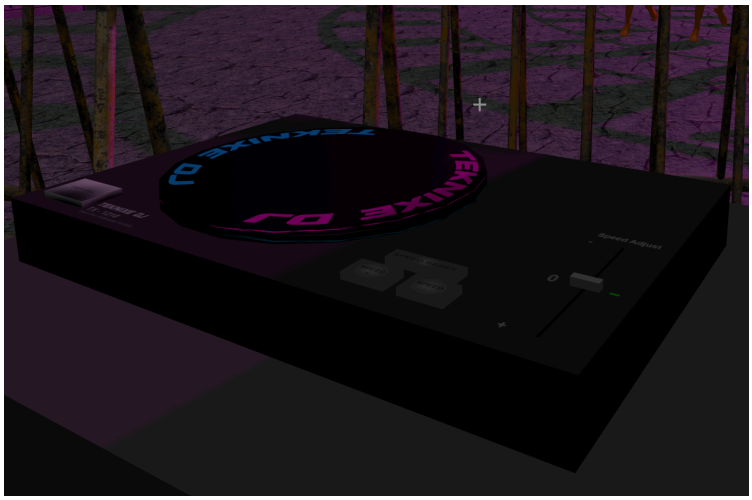


Figure 13 – in-game prototype of the DJ decks.

As covered by the analysis of Vinyl Reality, it is important that the systems modelled within DJ Simulator follow the niche conventions of DJing. Hence, many DJ system design choices have followed the conventions set by industry standard hardware, such as the in-game decks which resemble the Technics SL-1210. This meets the goal of simulating DJ hardware in accordance with industry standards.

Straight forward visual cues are an important requirement identified within earlier analysis. As prior discussed, most design elements are made in a way that has enough fidelity to convey the run-down, low-budget nature of the environment, however not in excess as to cause distraction and confusion for the player seeking visual cues.

To conclude, it is important that DJ Simulator has clear and unobstructed visual cues. The player cannot be distracted by unnecessary visual elements, since this would ruin immersion and cause a frustrating artificial difficulty spike. To ensure that is not the case, DJ Simulator employs a near-minimalist design language, where no unnecessary detail is present. This meaningful design results in a frustration free, accessible experience. It also ensures no computational power is wasted, and full advantage is taken of Unity's high efficiency and performance.

Technical Game Design

Class structure

Originally, the Game Manager was intended to perform the flawed BPM/60 audio analysis calculations. However, once this algorithm was replaced by the much superior GetSpectrumData, it was clear that a modular approach, with separate C# scripting files is cleaner, easier to manage and more versatile. Not relying on a single Game Manager instance means that various intertwined logic can be executed without complex parallelization necessary.

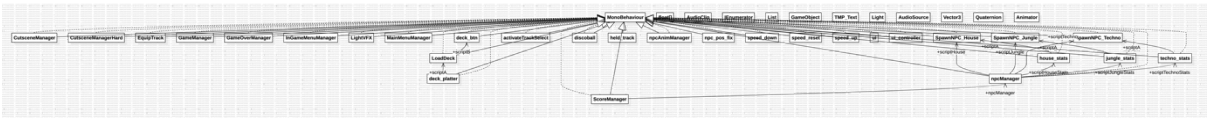


Figure 14 – UML class diagram.

The modular approach can be observed within the UML class diagram *(see Figure 12)*. There is very little relationships between classes, and each class is designed to compute a very specific purpose. Whilst there are little relationships between classes, there is a large amount of shared data, such as variables, being parsed across classes. The majority of the shared data is related to the game logic sensing the player's actions (and data related to it), such as: the music title currently equipped, the music genre currently equipped, current track playing, crowd population numbers, etc.

Most major computations shall have their own manager script, such as: CutsceneManager, GameOverManager, InGameMenuManager, MainMenuManager, npcAnimManager, npcManager and ScoreManager. Using individual managers for major computations is advantageous here due to the fact that logic can be easily tested, debugged and expanded on based on necessity. For example, if there is an issue with the scoring system, the

ScoreManager can quickly be analysed, improved and hot-swapped back into the game without re-compiling or affecting other logic within this complex simulation system.

Animation state machines

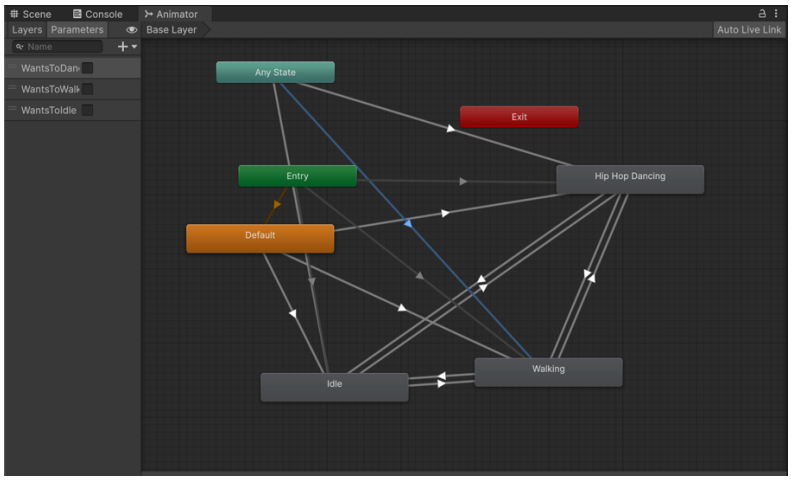


Figure 15 – Animator state machine for NPC logic.

In order to realistically simulate NPC crowd behaviours, Unity's Animator state machine will be used. This allows for an approach that somewhat resembles the neural Boolean logic a human brain might do when deciding on whether they should dance, stand or walk. Using a state machine for this application provides a neural-like simulation and invokes the feeling that the NPCs are truly 'thinking' about what they are doing, and are anticipating a physical response to the player's actions. This state machine is the implementation of the stimuli-based Boolean logic that the AI uses. As it was mentioned before, this approach is enough to convey realistic behavioural patterns for crowd members, without the need for more complex algorithms such as Dijkstra's algorithm. If certain very specific conditions are true, the NPC will perform the specific animation, and thus provide visual feedback to the player based on their actions.

The state machine mostly handles animation performance and transitions. The state machine is controlled by C# scripting within the npcAnimManager class. Utilising first-party solutions means that everything falls under the MonoBehaviour namespace, and animation can easily work with the game's logic computations done within C# scripting.

Component interfacing

Due to the independent, modular-like nature of the game's scripting logic, it is important that data can be interfaced across the different classes and functions. This is why a decision has been made to keep everything possible in-engine. Doing so means that every segment of the game's logic is a derivative of MonoBehaviour, and thus can be easily referenced across classes via a public static call, or a GameObject reference, if necessary. Since everything is kept within the confines of Unity, all physics calculations and any other calculations made by the engine can easily be found and referenced.

Implementation

Early prototype stage

Due to the large number of complex algorithms within the game, it is important for an early prototype to exist, in order to assess how Unity is handling the audio data computations that are essentials for the gameplay. Because of this, an early prototype was developed which contained the majority of the essential audio calculation logic.

```
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
RaycastHit hit;

if (Physics.Raycast(ray, out hit))
{
    if (hit.transform.gameObject == targetObject)
    {
        if (targetAudioSource != null)
        {
            // If the audio hasn't played yet, play it for the first time
            if (!hasPlayed)
            {
                targetAudioSource.Play();
                hasPlayed = true; // Mark as played
                isPlaying = true;
            }
            // If the audio is currently playing, pause it
            else if (targetAudioSource.isPlaying)
            {
                targetAudioSource.Pause();
                isPlaying = false;
            }
            // If the audio is paused, resume playback
            else if (!targetAudioSource.isPlaying && targetAudioSource.time > 0f)
            {
                targetAudioSource.UnPause();
                isPlaying = true;
            }
        }
```

Figure 16 – deck start/stop logic. (deck_btn.cs)

This prototype contains the logic for the DJ Deck's Start/Stop button. It is essential that the audio behaves as it would on real-life DJ equipment. The AudioSource should not automatically play music and also it should be able to pause and resume playback, instead of stopping it entirely. Utilising this script ensures that true play and pause can be achieved, without looping or restarting. This is a core behaviour of DJ hardware, and must be modelled exactly true-to-life.

```
1 reference
public void PlayDeck()
{
    // Check if there are any audio clips assigned
    if (audioClips.Count > 0)
    {
        currentClipIndex = held_track.TrackID;
        scriptB.hasPlayed = false;
        // Play the first clip
        PlayClip(currentClipIndex);
    }
    else
    {
        Debug.LogWarning("No audio clips assigned in the list!");
    }
}

// Method to play a specific clip based on index
1 reference
public void PlayClip(int index)
{
    if (index >= 0 && index < audioClips.Count)
    {
        currentClipIndex = held_track.TrackID;
        audioSource.clip = audioClips[index];
        //audioSource.Play();
        Debug.Log("current clip index: " + currentClipIndex);
    }
    else
    {
        Debug.LogError("Invalid clip index: " + index);
    }
}

// Method to add a new AudioClip to the list at runtime
0 references
public void AddClip(AudioClip newClip)
{
    if (newClip != null)
    {
        audioClips.Add(newClip);
        Debug.Log("Added new clip: " + newClip.name);
    }
    else
    {
        Debug.LogWarning("Cannot add null AudioClip!");
    }
}
```

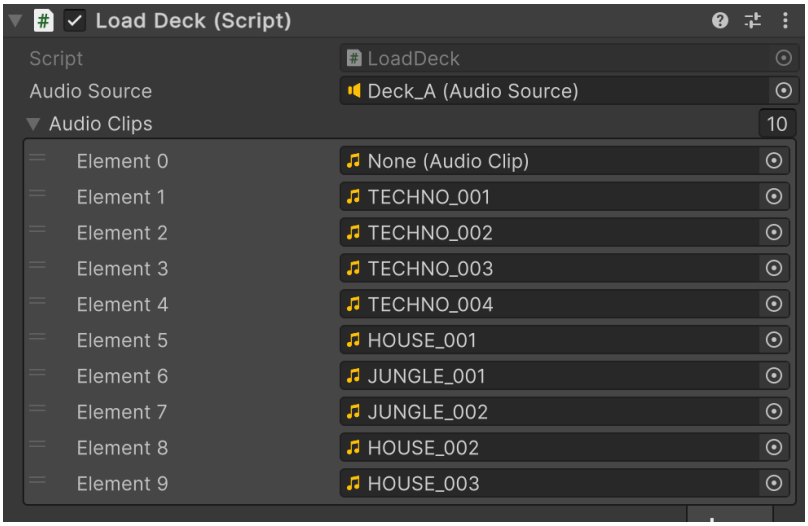


Figure 17 – hot-swappable AudioSource list for the in-game music library, along with the in-engine list of tracks to be loaded at runtime. (LoadDeck.cs)

On top of being able to play, pause and stop music reliably, a solution for music library indexing, loading and management is necessary. Utilising a simple AudioSource is not adequate due to the large amount of music data. Instead, a list array will be used. This is a

great solution since it will allow the upcoming genre and spectrum data analysis algorithms to refer to specific tracks by index, making it much easier to find and reference the necessary track. This implementation compiles the list on runtime, meaning that any roster changes in the music list can be done very rapidly, and the game's music library management logic will adapt itself on runtime.

```
void Update()
{
    if (audioSource.isPlaying)
    {
        audioSource.GetSpectrumData(samples, 0, FFTWindow.Rectangular); // Get audio spectrum data
        float intensity = GetAverage(samples) * colorMultiplier; // Calculate intensity based on spectrum data
        lightColor = 0;
        Color newColor = new Color(intensity, lightColor, intensity);
        lightToChange.color = newColor;
    }
}

// Calculate the average of the spectrum data
1 reference
private float GetAverage(float[] data)
{
    float sum = 0;
    for (int i = 0; i < data.Length; i++)
    {
        sum += data[i];
    }
    return sum / data.Length;
}
```

Figure 18 – code responsible for real-time audio spectrum data analysis. (LightVFX.cs)

Since real-time audio analysis and reactive lighting are very important requirements set for this project, it is important to approach this functionality early within the project's developmental cycle. This implementation samples 64-byte chunks of audio spectrum data, and then performs an averaging calculation upon it, and finally takes this data and outputs a float value that can be used as light intensity. The main challenge here is processing raw binary audio data (which is huge) and converting it into one simple float value that is representative of the music's volume (energy) levels. Sampling smaller chunks of audio binary data and then further reducing it into a single float value via the process of mathematical sampling and averaging enables the function to provide usable data for lighting intensity, and any other environmental variables that can be controlled by a numerical value.

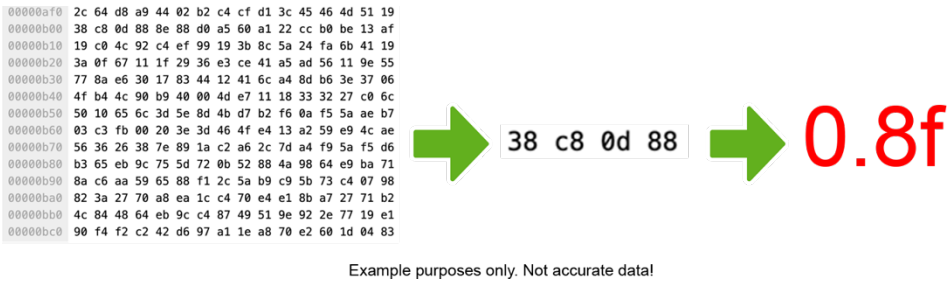


Figure 19 – diagram explaining the audio spectrum data analysis pipeline.

On the example diagram above *(see Figure 17)*, it can be observed how the script loads a music file as raw binary data, then samples small chunks of binary data into the *samples* array. After this, the data in the array is subjected to an averaging mathematical calculation, and a single float value that represents the chunk of spectrum data is provided by the function. This float value is much more usable and malleable for objects within Unity that take float/integer parameters.

After successfully extracting usable float data from the audio file's raw binary data, it can then be attached to a light source's intensity parameter. This results in a light source that reacts to

audio spectrum peak data in real-time; regardless of the file, or if any changes are made to it in between various complies and builds.

```
targetAudioSource.pitch += pitchStep;

// Clamp the pitch between the minimum and maximum limits
targetAudioSource.pitch = Mathf.Clamp(targetAudioSource.pitch, minPitch, maxPitch);
```

Figure 20 – pitching algorithm. (speed_down.cs, speed_up.cs)

Finally for this prototype, a simple pitch up and down feature has been added. This resembles the style of pitch manipulation found on the Technics SL-1210s. While this feature does not provide any points rewards for the player, including it enhances immersion and provides expected functionality from a DJ vinyl turntable.

Interim prototype stage

Having the majority of audio analysis implemented in the early prototype now means that gameplay loop features need to be implemented, such as the points system, audio sensing logic, and NPC population. For now, the animation of NPCs is not important, and neither are menus or UI, as the game must first be able to sense incoming data based on player actions.

```
0 references
void Start()
{
    HeldTrack = "None";
    TrackID = 0;
    genreTag = "None";
}
```

Figure 21 – data being held for music tracks. (held_track.cs)

The game relies on variable data from *held_track.cs*. The sole purpose of this function is to gather reported data into one place and let all other functions reference the variables within *held_track.cs* for a consistant and accurate reporting of the current track's genre data.

```
held_track.HeldTrack = TrackName;
held_track.TrackID = setTrackID;
held_track.genreTag = SetgenreTag;
```

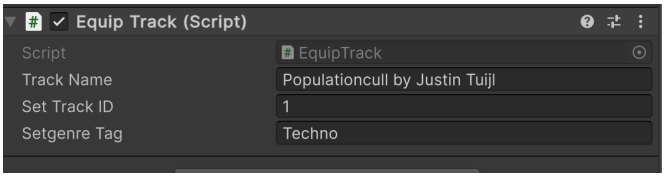


Figure 22 – EquipTrack.cs gathering data via inspector from each GameObject of music data.

The *held_track.cs* function gathers its data from the *EquipTrack.cs* serialized public variables, which are visible to the inspector. Each music track has its own GameObject with an *EquipTrack.cs* script attached to it. When the player selects the GameObject with the script attached to it, it will set the data within *held_track.cs* to the corresponding name, ID and genre data. This solution offers a simple way for the game to understand what type of music the player is playing, and how the game world should react to it.



```
IEnumerator AwardGenrePointsCoroutine()
{
    while (deck_btn.isPlaying)
    {
        if (canAwardGenre == true)
        {
            canAwardGenre = false;
            score = score + 250;
            //Debug.Log("Points awarded! Total points: " + score);
            scoreText.text = "Score: " + score.ToString();
            pointsListText.text = "You played a popular genre!";

            yield return new WaitForSeconds(8f);
            canAwardGenre = true;
        }

        else
        {
            yield return null;
        }

    }
}

// Start is called before the first frame update
0 references
void Start()
{
    scoreText.text = "Score: " + score.ToString();
    pointsListText.text = "Do some cool stuff to get points!";

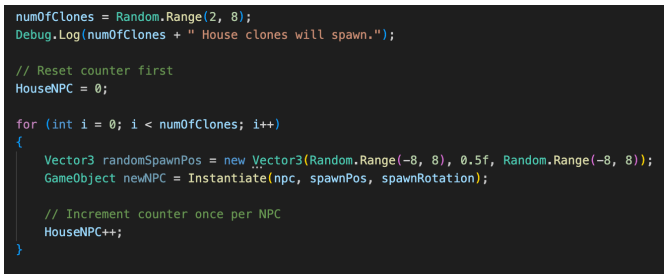
    StartCoroutine(AwardPlayTrackPointsCoroutine());
    StartCoroutine(AwardGenrePointsCoroutine());
}
// Update is called once per frame
0 references
void Update()
{
    StartCoroutine(AwardPlayTrackPointsCoroutine());

    if (npcManager.popularGenre == "Techno" && held_track.genreTag == "Techno")
    {
        StartCoroutine(AwardGenrePointsCoroutine());
    }

    if (npcManager.popularGenre == "House" && held_track.genreTag == "House")
    {
        StartCoroutine(AwardGenrePointsCoroutine());
    }
    if (npcManager.popularGenre == "Jungle" && held_track.genreTag == "Jungle")
    {
        StartCoroutine(AwardGenrePointsCoroutine());
    }
}
```

Figure 23 – co-routine responsible for awarding the player points. (ScoreManager.cs)

The scoring system relies heavily on data from *EquipTrack.cs* and *held_track.cs* to check if the genre being played is the one which is the most popular right now. To do this, a comparison is made between the most popular genre (based on NPC population) and the currently played genre, and if a match exists, the co-routine will award points for however long this condition is true.



```
numOfClones = Random.Range(2, 8);
Debug.Log(numOfClones + " House clones will spawn.");

// Reset counter first
HouseNPC = 0;

for (int i = 0; i < numOfClones; i++)
{
    Vector3 randomSpawnPos = new Vector3(Random.Range(-8, 8), 0.5f, Random.Range(-8, 8));
    GameObject newNPC = Instantiate(npc, spawnPos, spawnRotation);

    // Increment counter once per NPC
    HouseNPC++;
}
```

Figure 24 – spawning logic for NPC clones. (SpawnNPC_House.cs, SpawnNPC_Techno.cs, SpawnNPC_Jungle.cs)

At this point, the game needs to handle NPC spawning logic. For now, no animations are necessary, therefore randomised instantiation and spawning is acceptable. The game will spawn a randomised number of each NPC type, based on a pre-fab of each NPC type, with appropriate scripts attached to it.

```
// Check for duplicate NPCs
Debug.Log("Checking for duplicate NPCs...");
if (a == b || a == c || b == c)
{
    Debug.Log("Duplicate NPCs found.");
    scriptTechnoStats.techno_stat_text.SetText("Techno: Adjusting NPC numbers...");
    scriptHouseStats.house_stat_text.SetText("House: Adjusting NPC numbers...");
    scriptJungleStats.jungle_stat_text.SetText("Jungle: Adjusting NPC numbers...");

    GameObject[] npcs = GameObject.FindGameObjectsWithTag("npc");

    foreach (GameObject npc in npcs)
    {
        Destroy(npc);
    }

    // Clear the list of clones
    scriptTechno.TechnoNPC = 0;
    scriptHouse.HouseNPC = 0;
    scriptJungle.JungleNPC = 0;
    Debug.Log("Duplicate NPCs cleared.");
    StartCoroutine(crowdResetUIPrompt());
    scriptHouse.generateNPC();
    scriptJungle.generateNPC();
    scriptTechno.generateNPC();
}
```

Figure 25 – NPC duplicate checking. (npcManager.cs)

The game will also check to make sure there is no equal number of NPCs. There must be only one genre that is the most popular so that points can be awarded correctly. There is a script within *npcManager.cs* that will check for duplicates, and reset the dancefloor when necessary. This auto-balancing system ensures that the game is fair and balanced at all times.

Final prototype stage

For the final stage of development, menus, cutscenes and NPC animations must be present. At this stage, the game must now be immersive and realistic.

```
// Initial state: walking animation but not moving
isWalking = true;
isMoving = false;

// Wait for a random time before starting to move
float waitBeforeMovingTime = Random.Range(1f, 13f);
Debug.Log("NPC will display walking animation for " + waitBeforeMovingTime + " seconds before moving");
yield return new WaitForSeconds(waitBeforeMovingTime);

// Start actual movement
isMoving = true;
Debug.Log("NPC started moving");

// Move for a random duration
float movementDuration = Random.Range(2f, 8f);
Debug.Log("NPC will move for " + movementDuration + " seconds");
yield return new WaitForSeconds(movementDuration);

// Stop moving and walking
isMoving = false;
isWalking = false;
Debug.Log("NPC stopped moving");

//transform.rotation = Quaternion.Euler(0, 180, 0);
transform.rotation = Quaternion.Euler(0, Random.Range(150, 180), 0);
isMoving = true;
isWalking = true;

float movementDuration2 = Random.Range(1f, 9f);
yield return new WaitForSeconds(movementDuration);
isMoving = false;
isWalking = false;
//transform.rotation = Quaternion.Euler(0, Random.Range(100, 200), 0);
```

Figure 26 – logic for NPC walk-in. (npcAnimManager.cs)

The NPCs will now follow the Boolean rules of the Animator state machine. When spawning in, they will walk a randomised distance at a randomised angle in order to distribute and disperse across the dancefloor. Pauses, duration and speed changes have been randomised in order to give the NPCs natural unique characteristics in order to increase behavioural realism.

```
if (isWalking)
{
    // Walking animation takes priority
    anim.SetBool("WantsToDance", false);
    anim.SetBool("WantsToIdle", false);
    anim.SetBool("WantsToWalk", true);

    // Only move the NPC if isMoving is true
    if (isMoving)
    {
        transform.position += transform.forward * Random.Range(0.5f, 4f) * Time.deltaTime;
    }
}
else if ((npcType == held_track.genreTag) && deck_btn.isPlaying)
{
    // Dancing when preferred genre is playing
    anim.SetBool("WantsToDance", true);
    anim.SetBool("WantsToIdle", false);
    anim.SetBool("WantsToWalk", false);
}
else
{
    // Default to idle
    anim.SetBool("WantsToDance", false);
    anim.SetBool("WantsToIdle", true);
    anim.SetBool("WantsToWalk", false);
}
```

Figure 27 – Animator state machine Boolean logic. (npcAnimManager.cs)

Once the NPCs are settled in their location, the script will decide whether the NPC does or does not like the music being played. If the music being played is not of the NPCs favourite genre, they will idle. This will show they player that they are not enjoying themselves. Otherwise, if the music being played is the NPCs favourite genre, they will dance, indicating that they are 'satisfied'.



Figure 28 – the main menu.

A timer script has been added that counts down from 5 minutes, and on end displays the player's accumulated score, and then returns to main menu. The main menu and in-game menu were also implemented. They use simple scene management logic to change the scene based on button input. The in-game menu shows on screen when the P key is pressed. The in-game menu lets the player return to the main menu. Additionally, level variations for hard mode and free play were added. Hard mode has no crowd population tally visible on screen, and free play mode has no timer, making it an endless, practise game mode.

Testing and Evaluation

Consistent testing has been crucial for this project. Utilising the modular scripting approach means that all mini-systems within various scripts must be tested properly and any anomalies accounted for.

Black box testing

Test condition	Expected result	Actual result	Notes
Select a track from a crate.	Track selection UI appears. Track gets equipped and the title is shown in the UI.	As expected.	
Play an equipped track on the decks.	The correct song plays and the scoring and crowd reacts.	As expected.	
Play a popular genre	A large amount of points is awarded.	As expected.	
Play an un-popular genre	A small amount of points is awarded.	As expected.	
Stop music playback.	Music stops, crowd stops, points stop.	As expected.	
Change equipped track.	New music plays, points and crowd react accordingly based on track popularity.	As expected.	
Hold an un-popular track while a popular one is playing	No effect on score is expected.	As expected.	
Hold a popular track while an un-popular one is playing	Player should not be awarded extra points.	Player is being awarded extra points as if they were playing a popular track.	This may be caused by the game reading the held track genre and not the playing track genre. A now playing track genre variable needs to be made.
Timer runs out	Game is over and score is displayed accurately.	As expected.	

Figure 29 – table capturing results from black box testing.

Based on the testing conducted, the game logic behaves as expected. Crowd AI and the environment react appropriately to changes being made by the player. However, a logical flaw has been found. A variable holding the genre of the actual track playing does not exist, therefore the scoring system awards the player based on the track they are holding, not the one they are playing. Further sensing logic must be added that is able to detect the genre of the music being played, as opposed to the music that the player is holding (if it is different). This needs to be fixed so that the game can report the player's score accurately. This should be easily done within the existing scripting logic.

Functional testing

Since black box testing has been done for the overall game logic, for the NPC spawning, a functional testing method has been used. The game has been started 10 times, and the number of spawned NPCs at start has been noted down.

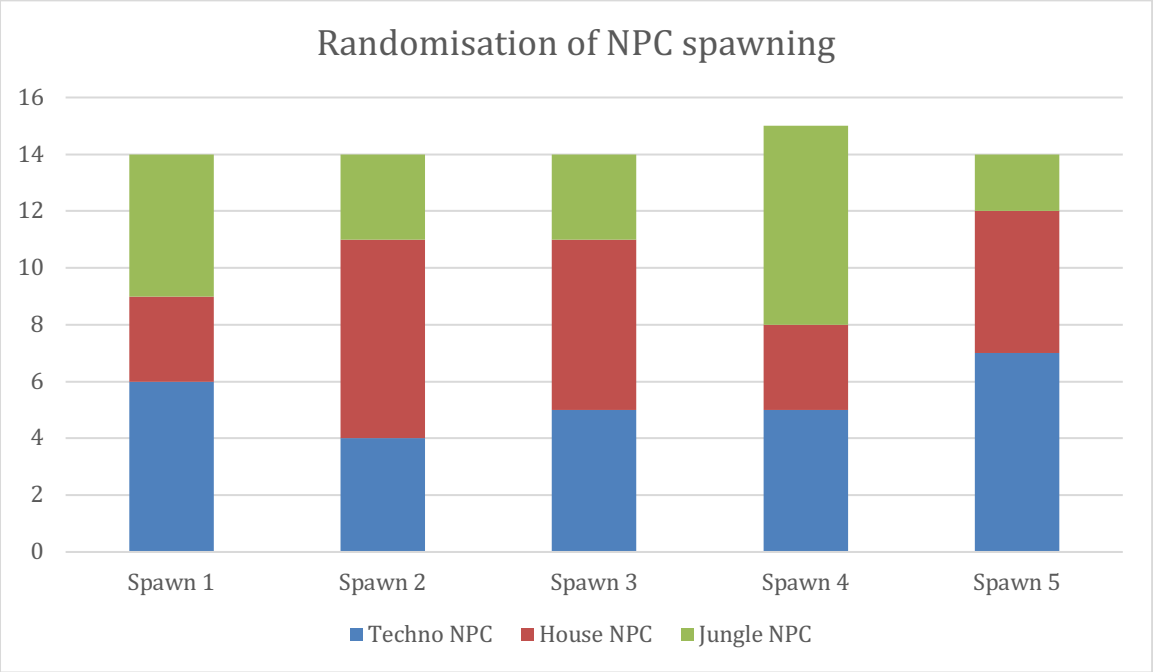


Figure 30 – table capturing distribution of NPC types at spawn, based on 5 playthroughs.

The chart above proves that the auto-balancing algorithm works, and spawns in a diverse and varied amount of NPC on every playthrough. There are no clear patterns of a certain NPC type spawning in more often than expected.

Player feedback

Player feedback is a very important part of development. Using the Scrum development methodology framework provides excellent opportunities for implementing provided feedback into the game. A simple Microsoft Forms questionnaire has been issued to players after their playthrough. The provided data has then been plotted onto a diagram to visualise opportunities for improvement.

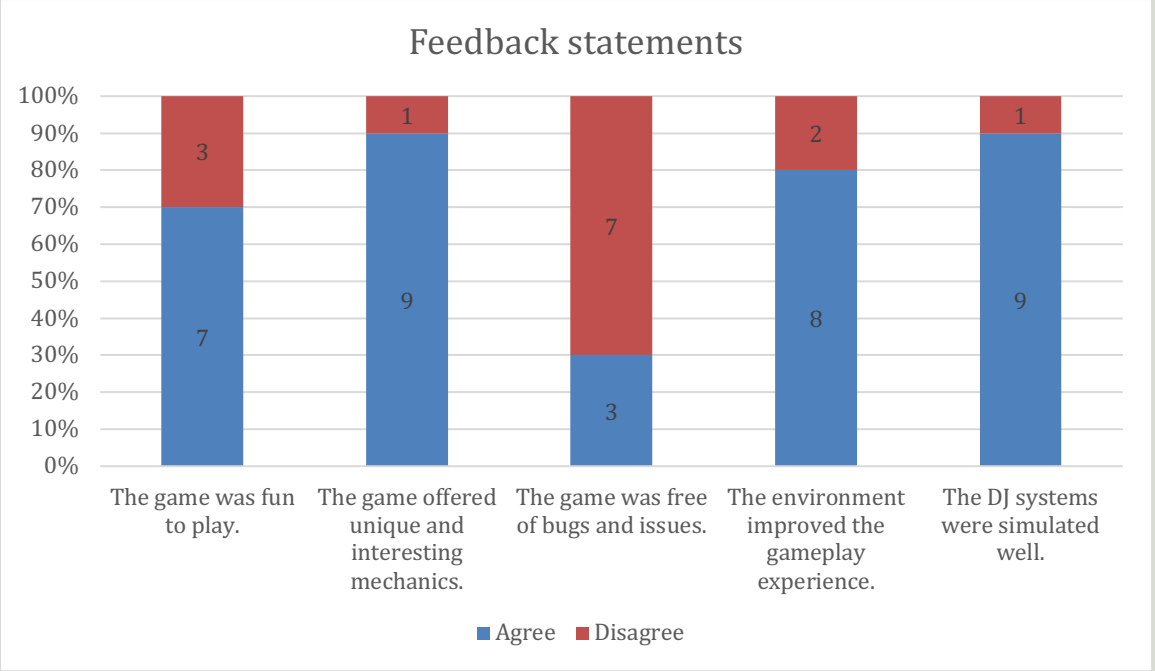


Figure 31 – feedback collected via form based on playthroughs of 10 players.

Based on the observations from this chart, it can be agreed that the majority of expectations and requirements are met. One very clear opportunity for improvement is debugging. This is most likely due to the points awarding bug that has been captured during the black box testing phase. If that bug was to be fixed, it can be assumed that the feedback for the statement would rise, resulting in overall positive feedback from players.

Key observations

Based on the testing and evaluations conducted on the final build of the game, it is clear that the game has achieved its purpose of developing a unique and faithful DJ simulation experience. The key development highlights and features that development was emphasised on is well received and well praised. Following an iterative solo Scrum framework has allowed for many opportunities to capture bugs and anomalies, hence there being one (albeit major) bug. Once resolved, it is expected that the game will receive universally positive feedback, with an average overall rating of approximately 8.5/10. This calculation has been based on the data portrayed in *Figure 31*, if the outlier was to be ignored.

Legal, Social, Security and Ethical Issues

Copyright & licencing issues

Being a game with a large amount of music content, copyright issues are the main point of concern for this project. There are very strict copyright laws in the UK [20] and the world. Digital content can be taken down, and legal action can be pursued [21] if said content is found to be in violation of copyright laws. Therefore, it is critical that DJ Simulator conforms to copyright laws in its entirety. To address this, all music within DJ Simulator is either original content, or from a royalty-free source *(see Figure 12)*.

The Unity game engine has undergone major criticism recently for wanting to charge studios every time a person installed a game using Unity's code [22]. Based on the extreme backlash, this decision has been amended. Therefore, should there be a public release of DJ Simulator,

it would not be affected by this problem. Aside from that, Unity's licensing model should not be an issue for DJ Simulator; unless it went on to become a large-scale project. However, in its current state that is not an issue.

Most of the artwork and in-game content are original. It has been made within fully-licensed for commercial use tools such as Adobe Photoshop, Autodesk Maya, Mixamo and Logic Pro X. Opting for mostly original content means that it is not difficult to obtain rights and licences. The external assets and textures used follow the Standard Unity Asset Store EULA and/or Creative Commons licenses, and are referenced in the technical appendices section of the report.

As previously discussed, any references to explicit imagery, that is frequently associated with nightclubs and nightlife in general, has been removed. References to drugs, alcohol and suggestive content have been completely avoided. This has been done in order to expand the potential player base to younger audiences and also as to not alienate audiences based on misrepresentation. The game is expected to receive a PEGI rating of 7+ due to the portrayal of nightlife.

Data protection & security

The game does not collect or store any user data whatsoever. This means that there is no need for encryption or consent of sensitive data, such as passwords or personal identification characteristics. The game has been designed in a way where it does not read or write any data beyond the engine's internal files and libraries. This has been done as to avoid any security vulnerabilities.

Ethical & social impact & implications

There are no loot-box, gambling or intentionally addictive mechanics within the game. The game is designed to be ambiguous with its portrayal of views and should not have any content that suggests or puts forward any opinionated views. The game offers accessibility via its approachable and clear audio-visual cues. May of the actions within the game have duplicate cues (i.e. the game provides information in more than one way). For example, crowd enjoyment can be inferred via the crowd population tally, the points list and the crowd behaviours themselves. Despite being a music-based game, the UI elements are descriptive enough that the game could be played with no audio, allowing people that are hard of hearing to still enjoy this music game.

As already mentioned, any reference to protected characteristics has been avoided. The crowd models are deliberately vague as to not distract the player, and also to avoid the conveying of any stereotypes, or alienating the player by lack of representation. Despite nightclubs being notorious places of drug and alcohol usage and potential violence, the portrayal of such actions is completely unnecessary for this project, and thus has been avoided. Vagueness in those aspects means that the game is open and inviting to players of any age group, or any protected characteristic class. This means that any person with an interest in music and the DJing scene can play this game. This is useful due to the fact that there are DJs with massive differences in age, all across the world. Therefore, making a DJ simulation game that appeals to only a certain age group would cause a huge loss of opportunity in a largely untapped market.

Critical Review and Conclusion

Summary of achievements

Overall, the majority of set targets have been achieved. A convincing simulation of tasks & experiences of a typical beginner DJ, realistic crowd behaviours and a low-budget nightclub environment that feels 'alive' and responsive to the player's actions have been achieved. Testing has shown that these elements are functional within the game. A greater depth of simulation for the DJ hardware would have been beneficial, however due to the intricacy of said systems, that was not possible with the time and resources at hand.

Personal experiences

This project has been very enlightening on what can be achieved with good knowledge of tools. I initially have not been convinced that it would be possible to achieve such levels of reliable real-time audio analysis and simulation. However, upon learning the importance of studying documentation and research papers, I found the majority of the tasks and requirements set very approachable and achievable. A great approach that I have learnt for major developmental challenges was to use the Solo Scrum framework to my advantage and breakdown and iterate through problems.

I am glad that I followed the advice of starting in the summer. Not doing so would have certainly resulted in far more stress and a much lower level and quality of completion. With that being said, in the future, I would allocate more time towards blog writing as this has quickly developed a backlog and increased workload due to the blog not being updated weekly.

Conclusion

There is still a lot that can be achieved for this project. New maps, new music, more gameplay modes and scenarios are just a few potential ventures. Having such unique ideas and concepts means that it is worth taking the game further and expanding it before potentially marketing it.

This project has been an excellent showcase of the many valuable lessons I have learned whilst embarking on this degree. I consider this project to be a success, which I would not have deemed possible if not for the years of preparation and knowledge being shared by all of the teachers I have had. With that being said, I would like to thank all of the faculty staff that has taught me over the years and contributed their knowledge to make this project possible.

Technical Appendices

External assets sources

https://assetstore.unity.com/packages/3d/environments/urban/rusty-barricade-with-pbr-materials-2k-4k-217901

https://www.mixamo.com/

https://typodermicfonts.com/

https://assetstore.unity.com/packages/2d/textures-materials/wood/wood-pattern-material-170794

https://www.sketchuptextureclub.com/textures/architecture/paving-outdoor/pavers-stone/blocks-regular/pavers-stone-regular-blocks-texture-seamless-06290

https://www.the3rdsequence.com/texturedb/texture/195/white+cracked+concrete/

https://www.textures.com/browse/brown/32111

References

[1] Ler digital studio (2025). Beat Saber - VR rhythm game. [online] Beatsaber.com. Available at: https://beatsaber.com/documentation/environment-specific-lighting-attributes-and-explanations/index.html [Accessed 22 Apr. 2025].

[2] Sunil Khobragade (2024). Bringing Stories to Life in Casual Gaming Experiences [online] Linkedin.com. Available at: https://www.linkedin.com/pulse/visual-storytelling-casual-game-development-sunil-khobragade-xjvhf/ [Accessed 22 Apr. 2025].

[3] Venue Lights. (2024). Nightclub Lighting | Venue Lights. [online] Available at: https://venuelights.co.uk/lights/venue-lighting/nightclub-lighting/ [Accessed 22 Apr. 2025].

[4] Vinyl Reality. (2021). Vinyl Reality - DJ in VR. [online] Available at: https://www.vinyl-reality.com/ [Accessed 22 Apr. 2025].

[5] Vinyl-reality.com. (2019). audio_analysis [Vinyl Reality Wiki]. [online] Available at: https://vinyl-reality.com/wiki/doku.php?id=audio_analysis [Accessed 22 Apr. 2025].

[6] Technics (n.d.). SL-1200 – Heritage - Technics. [online] www.technics.com. Available at: https://www.technics.com/global/home/sl1200/heritage.html.

[7] www.djgym.co.uk. (n.d.). How Do DJs Get Their First DJ Gig: A Guide For New DJs. [online] Available at: https://www.djgym.co.uk/post/how-do-djs-get-their-first-dj-gig-a-guide-for-new-djs.

[8] Simoes, R. (2024). A Complete Guide To The DJ Club Standard Setup For The Industry In 2024 - If You’re In Search Of DJ Gigs, Here’s What You Should Know - Learningtodj.com. [online] Learning to DJ. Available at: https://learningtodj.com/blog/the-club-standard-dj-setup-in-2024/.

[9] Fandom.com. (2025). GTA Online Nightclubs. [online] Available at: https://gta.fandom.com/wiki/Nightclubs [Accessed 22 Apr. 2025].

[10] Unity Technologies (2024). Unity - Scripting API: MonoBehaviour. [online] Unity3d.com. Available at: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/MonoBehaviour.html.

[11] Epic Games Developer. (2025). Designing Visuals, Rendering, and Graphics | Unreal Engine 4.27 Documentation | Epic Developer Community. [online] Available at:

https://dev.epicgames.com/documentation/en-us/unreal-engine/designing-visuals-rendering-and-graphics?application_version=4.27.

[12] Unreal Engine. (2024). Animation. [online] Available at: https://www.unrealengine.com/en-US/uses/animation.

[13] Unity Technologies (2025). Unity - Scripting API: AudioSource.GetSpectrumData. [online] Unity3d.com. Available at: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/AudioSource.GetSpectrumData.html [Accessed 22 Apr. 2025].

[14] Unity Technologies (2025). Unity - Manual: Animation State Machines. [online] docs.unity3d.com. Available at: https://docs.unity3d.com/Manual/AnimationStateMachines.html.

[15] Raph Koster (2005). A theory of fun for game design. Scottsdale, Az: Paraglyph Press.

[16] ProductPlan (2023). Agile vs. Waterfall | Pros, Cons, and Key Differences. [online] www.productplan.com. Available at: https://www.productplan.com/learn/agile-vs-waterfall/.

[17] Wikipedia Contributors (2018). Agile software development. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Agile_software_development.

[18] Starloop Studios. (n.d.). Best Agile Practices in Game Development | Starloop Studios Blog. [online] Available at: https://starloopstudios.com/best-agile-practices-in-game-development/.

[19] Lucidchart (2017). Scrum for One: Adapting Agile Scrum Methodology for Individuals. [online] Lucidchart. Available at: https://www.lucidchart.com/blog/scrum-for-one.

[20] UK Music. (n.d.). Copyright. [online] Available at: https://www.ukmusic.org/policy-campaigns/copyright/.

[21] musiciansunion.org.uk. (2022). Copyright, Moral Rights and Performers' Rights Musicians Must Know About. [online] Available at: https://musiciansunion.org.uk/legal-money/workplace-rights-and-legislation/copyright-moral-rights-and-performers-rights.

[22] Unity: Gaming boss quits after furious pricing backlash. (2023). BBC News. [online] 10 Oct. Available at: https://www.bbc.co.uk/news/technology-67063855.