Ben Thompson

Supervisor: Thomas Oliver

Second supervisor: benjamin williams

F-Zero Procedural Race Track Generation for Believability

GDEV60001 Games development project

Contents

[Glossary 3](#_Toc167230179)

[Abstract 3](#_Toc167230180)

[Introduction 3](#_Toc167230181)

[Aims and Objectives 3](#_Toc167230182)

[Literature Review 3](#_Toc167230183)

[Procedural Content Generation 3](#_Toc167230184)

[Believability of Procedural Generation for Games 4](#_Toc167230185)

[Random vs Pseudo-Random Number Generation 4](#_Toc167230186)

[Random Noise Functions 5](#_Toc167230187)

[L-systems 5](#_Toc167230188)

[Cellular Automata 6](#_Toc167230189)

[Wave Function Collapse 6](#_Toc167230190)

[Machine Learning 6](#_Toc167230191)

[Generating Roads in Games 7](#_Toc167230192)

[Saving/Loading Tracks 8](#_Toc167230193)

[F-Zero Analysis 8](#_Toc167230194)

[F-Zero X: Procedural Track Generation 9](#_Toc167230195)

[F-Zero’s Gravity-Defying Movement 10](#_Toc167230196)

[Believability in F-Zero Track Design 11](#_Toc167230197)

[Research Methodologies 11](#_Toc167230198)

[*F-Zero X* Gameplay Representation 14](#_Toc167230199)

[Results and Findings 15](#_Toc167230200)

[Creating the Secondary Generation Rule Set 15](#_Toc167230201)

[Discussion and Analysis 18](#_Toc167230202)

[Analysis of Gameplay Recreation 20](#_Toc167230203)

[Analysis of Generation Algorithm 21](#_Toc167230204)

[Conclusion 22](#_Toc167230205)

[Recommendations 24](#_Toc167230206)

[References 25](#_Toc167230207)

[Appendices 28](#_Toc167230208)

[Appendix 1 – F-Zero X Expansion Kit Course Edit Footage 28](#_Toc167230209)

[Appendix 2 – F-Zero X Generated Track Data 28](#_Toc167230210)

[Appendix 2.1 – Generated Tracks 28](#_Toc167230211)

[Appendix 2.2 – Average Track Length 28](#_Toc167230212)

[Appendix 2.3 – Average Top Speed 29](#_Toc167230213)

[Appendix 3 – *F-Zero X* Track Data Analysis 29](#_Toc167230214)

[Appendix 3.1 – Average Lap Time Based on Track Length 30](#_Toc167230215)

[Appendix 3.2 – Distribution Average Time to Travel 1km compared to Element Amount 30](#_Toc167230216)

[Appendix 4 – Track Section Generation Rules 31](#_Toc167230217)

[Appendix 4.1 – Base *F-Zero X* Generation Rules 31](#_Toc167230218)

[Appendix 4.2 – Derived Generation Rules 31](#_Toc167230219)

[Appendix 5 – Track Generation Data 31](#_Toc167230220)

[Appendix 5.1 – Tracks Generated Using Base *F-Zero X* Generation Rules 31](#_Toc167230221)

[Appendix 5.2 – Tracks Generated Using Derived Generation Rules 31](#_Toc167230222)

[Appendix 6 – Track Generation Files 31](#_Toc167230223)

# Glossary

PCG – Procedural Content Generation

L-systems – Lindenmayer systems

# Abstract

# Introduction

Utilising PCG in the context of racing games is something that has been investigated extensively across many different sources, with research looking at different approaches and goals for what the procedurally generated output is trying to achieve. However, it has almost exclusively been focused on standard racing simulation scenarios, with tracks either being completely 2D or having very little / conforming to realistic variations in elevation. There is also a large focus on modelling tracks to the specifications of each individual player’s playstyle, and less research undertaken about taking a broader approach for generating believable and balanced race tracks. Little research has been undertaken to take the findings and push them into a different style of track generation, like the futuristic, anti-gravity style of the tracks that feature in the 3D iterations of the *F-Zero* series. The *F-Zero* games feature fast gameplay that has vehicles that can smoothly drive on any orientation, allowing for very interesting and unique track designs that involve verticality, rotations and inversions that wouldn’t be possible for realistic racing simulators. These games, particularly the *Nintendo 64* release *F-Zero X*, are widely known for their tight corners in the track design and harsh, unforgiving difficulty, which adds an extra layer to the investigation that has not been deeply explored before this.

# Aims and Objectives

The goals of the project are to produce a procedural track generator that reproduces tracks in the style of *F-Zero* and perform an investigation into how the different generation parameters affect the believability of the output of the procedural track generator. From this, the data from the sample of procedurally generated tracks will be used to ascertain the believability of the tracks, and which elements correlate with the largest effect on the believability of the tracks. The data values for the procedurally generated tracks will be measured against data collected on the tracks featured in the game *F-Zero X*, using the difference as an indication of how believable and accurate the output of the track generator is. Some of the parameters used in the generation of the tracks will be altered throughout the investigation, allowing for the variables that have the greatest effect at achieving believability for the tracks to be identified by comparing the coefficients of how different variables controlling the track generation relate to each other, and have the most impact on getting them as close to *F-Zero* tracks as possible.

# Literature Review

## Procedural Content Generation

Procedurally content generation is the concept of having the computer automatically generate content that appears in a game, instead of the alternative of needing a person to handcraft every portion of a game (Summerville, et al., 2018). It helps to speed up production of content and allows the scope of projects to increase further than what would usually be possible with team size, but with the trade-off of potentially not having all the content be as interesting. Large game productions, particularly massively multiplayer online games, have found the traditional by-hand game development processes to not be scalable to the huge player counts and project scope in modern game projects (Iosup, 2011). PCG has been used in various applications across a wide variety of different games, but the focus here will be on the generation of levels.

In its most basic form, PCG has been done by combining pre-defined design patterns to create a level (Dahlskog & Togelius, 2012), and then randomising specific elements to make the design feel more unique and interesting (Yu & Hull, 2011). Because the design patterns used to generate levels by these algorithms are manually designed, they don’t require checking to make sure they are within the constraints the level needs to be generated by like more automated generators would require, but by having certain elements that can be randomly changed in the algorithm it adds a small layer of interest to stop the output levels from quickly becoming stale to players.

### Believability of Procedural Generation for Games

The goal of believability in PCG for games is generating game content which the player can’t differentiate from content manually created by a game designer. Because game designers aim to create content that is of high-quality and that players find engaging and fun inside the constraints of the gameplay, the criteria that procedurally generated game content is aiming to make can be categorised based on player feedback in the same way. Because of the wide variety of content that can be made into games, a lot of the believability criteria will change depending on what the desired output of the PCG system is.

*StoryWorld* is a system for generating quests in a game, and procedurally writing a narrative that is affected by the player’s choices. It raises a barrier for PCG being in making the content generated distinct and uniquely interesting to players, rather than just feeling like variations on the same thing. To overcome the barrier, *StoryWorld* uses a system of storing character memories and having events influence character desires to generate the ongoing quests and narrative. This means that the content generated feel cohesive and connected as well as making sense within the game’s world, making it more believable for players (Prins, et al., 2023). The system is built in a way that means the PCG will emulate how a quest designer would approach creating new quests that fit the existing narrative of the world, thus the concept can be applied to PCG systems beyond narrative quest generation by imposing the rules a designer would follow on the system, improving player reaction to the content and making it seem more believable. To apply this to tracks in *F-Zero*, it would require analysis of how the tracks themselves are designed and structured to come up with a set of rules that need to be followed.

### Random vs Pseudo-Random Number Generation

Truly random generation isn’t possible to achieve on computers alone due to their logical nature so when it is required, an unpredictable physical process is needed to use as the basis to generate the values from. Methods that have been used include taking images of an array of lava lamps (Noll, et al., 1998) and sampling background radiation patterns (Images Scientific Instruments, 2007). For the purposes of generating content in video games, true random generation isn’t necessary and pseudo-random generation can be used. Pseudo-random generators use algorithms to generate the random values that they output, so there is a set pattern that they follow (Wen & Yu, 2019). The benefits of generating content pseudo-randomly is that they always output the same pattern when given the same seed input value, so the same content can be regenerated more than once if the initial seed value that was used is known. For a pseudo-random number generation algorithm, the pattern of values that is output should still seem random; tests such as looking at the Power Spectrum and the mean and variance of the output values can be run to evaluate whether there are any biases to specific values in the output of the procedural generation algorithm (Ko, 2015).

Seeding values can be generated through multiple different methods, a common form is using the system or CPU clock time as the seed, as it would produce a different random pattern whenever the project was launched, which is an approach many games take to. However, this can allow for exploits and manipulation of the random elements of the game by manipulating the system clock and launching the game at the correct time. For more secure implementations, getting stronger seeds becomes necessary without having to rely on random physical processes for true random number generation. A strong seed is one that is defined as being hard to obtain based on the output, as methods of reverse engineering seeds for pseudo-random number generators have been developed for specific generation algorithms (Bouillaguet, et al., 2020). Different methods of obtaining secure seeds have been researched, some of which involve sampling physical data sets such as using sensors to find a strong seed (Hong & Liu, 2015).

Some games, particularly those that have random/procedural generation as one of their core features, allow players to be able to input their own seed to initialise the generation, *Minecraft* (Mojang Studios, 2011)allows players to set the seed that is used to generate the game world. Exposing the seed is good for game projects that do not require security in the elements that the random number generation affects, communities can build around interesting outputs from the seed generation. In terms of the track generation project, keeping the seed exposed and allowing users to input their own seed is something that would be good to include, as there is no need for the initialisation values to be obscured and it allows for easy repeatability in the output using a pseudo-random number generation algorithm.

### Random Noise Functions

Random noise functions are often used as a way of procedurally generating elements in games. Using noise patterns originated in computer graphics as a cheap and inexpensive way to add realism to a 2D texture, as the texture can be generated procedurally and stored into memory (Lagae, et al., 2010). Perlin noise is a form of noise that is pseudo-random, using an algorithm to generate noise values that are related to each other so it forms a more natural, smoother pattern that is more useful for procedural generation. In 2 dimensions, it is defined as a grid of squares with a random unit vector at each intersection, using the dot product to create gradients around the point. Once interpolated, the values smoothly transition between each other, creating the distinct patterns of the noise. Noise has been used to generate terrain, as multiple noise patterns can be layered together to make more complex, intricate generation patterns (Archer, 2011).

Because of the multi-dimensional nature of noise generation, it can be applied to numerous different elements of generating *F-Zero* race tracks. 1-dimensional noise patterns can be used to control the height along the length of a track, as well as being able to utilise the values for other track elements like track rotation, width and tilt. A 2-dimensional noise pattern could be used as the layout of the track itself, by isolating a specific range of values to use as the track. The potential issue with the approach of drawing the track layout based on the noise output is that the track layout may not connect back together depending on the noise pattern generated, which wouldn’t meet the requirements of an *F-Zero* level.

### L-systems

L-systems are a concept that was originally created to model the repeating patterns in plant growth (Lindenmayer, 1968) and have long since been used as a procedural generation method specifically for generation of plant and tree meshes. L-systems take a specific string input, and then iterate over the string applying rules that create a larger and more complex system which can be represented by and reconstructed from a string of symbols. They are much more specialised in emulating repeatable branching patterns like those seen in trees and plants, but can also be reutilised for other concepts like emulating large road networks using relatively simple rules and patterns (Jormedal, 2013), as when iterated on they can become exponentially more complex. The concept of iterating over the string and using symbols to represent elements can be retooled into working on a linear system, where it could be applied to race track generation.

### Cellular Automata

A cellular automaton is a grid of a cells which each have different states, which are iterated through and change state relative to the surrounding cells based on mathematical rules, which comprise a single cell’s neighbourhood. The differences in the rules applied to the cells gives cellular automata a wide versatility in its potential uses, including but not limited to physics and biology. Perhaps most famously, Conway’s Game of Life is a cellular automaton with specific rules to create interesting patterns based on overpopulation and underpopulation rules (Gardner, 1970). Cellular automata have been applied to procedural level generation in games in various different aspect, but it is mainly focused on terrain and level generation like in the creation of cave systems (Johnson, et al., 2010). By altering the rules over time, they can be utilised to create distinct patterns such as maze levels (Adams & Louis, 2017).

### Wave Function Collapse

Wave function collapse is a PCG algorithm that generates content based on rules that can be either set by the designer or derived from an input. The algorithm pulls the tiles that it can use from the input, randomly selects a point on the grid to place a random tile down in, and then uses adjacency rules that have been derived from the positions of each tile on the input to propagate outwards, filling in the entire grid by removing the options available for each tile. It was originally developed as a program to procedurally generate images based on an input image (Gumin, 2016). The algorithm has been retooled to work for procedural generation in games, notably being used in the generation of terrain and levels (Stålberg, 2018). Because wave function collapse has the generation output follow rules based on what is around the object, it could be a useful approach for controlling what track sections get placed where in generating *F-Zero* tracks, as the rules can be defined by analysing tracks from the game.

### Machine Learning

Machine learning is an area of artificial intelligence in which an algorithm is made to be able to refine its output based on its existing output without any human input. There are many cases where machine learning has been utilised to generate procedural content for a wide variety of different game types, as they can iterate and refine the output set to keep improving the quality of the content that is generated. Studies have been conducted into how designers work with and interact with procedurally generated content by adding their own creations into the dataset and having them evaluate the generated designs to further refine and improve the output of the model (Kruse, et al., 2022).

Multiple studies have particularly looked at machine learning models can be used to procedurally generate race tracks in video games. Multiple different models have been used to compare the fun generated, making sure they meet the specific required criteria to generate as proper race tracks. A search-based algorithm returns the evaluated bestcontent to improve the quality of what is being used (Togelius, et al., 2011), often incorporating a way of simulating player behaviour to better quantify the results (Prasetya & Maulidevi, 2016). This idea is then extended further by modelling player behaviour and using that as the way to indicate the quality of tracks. Using this with an evolutionary method that can iterate on itself using the results from the dataset tailors the content more towards what the individual player wants and can improve specific player enjoyment (Loiacono, et al., 2011). In a commercial product which has a much broader player base and target audience, this could potentially be detrimental to believability of the output results as it may focus too much on a specific subset of player behaviours and preferences.

## Generating Roads in Games

Independent of the method used to procedurally generate the content, it needs to be handled in a way that allows for roads to be generated and represented in the game world. For roads that will need to curve, splines are a good method for representing the road data. Splines are a set of controls points that can be used to interpolate a smooth curve, with each section between two adjacent points on the spline containing a tangent point which control the interpolation that creates the curvature of each section. The simple nature of representing splines can be used to make easy changes to the overall shape and structure of the line. Applying extra parameters to the spline like width makes it very easy to represent using a mesh that can be used for roads (Griffin, 2018).

Splines are often used in game development so some game engines have dedicated spline tools implemented already (Unity Technologies, 2023). Depending on the project requirements, developers will also use third-party systems to handle the generation of more complex road systems, particularly when adding the visual polish to the spline content. *Houdini* (SideFX, 2023) is a versatile procedural content tool initially made for artists that can also be used by developers to take in splines and generate more complex road systems, allowing for automated smooth intersections and placement of visual elements (Santiago, 2019).

There are multiple different types of splines, which all have specific characteristics that can be used for generation. Catmull–Rom splines are made by having all the points making up the spline being on the spline line, with 2 additional points on either end of the curve. These types of splines are characterised by the smooth curves that they create compared to other splines. With the control points also lying on the spline, they allow for greater direct control of the spline path compared to other spline types (Yuksel, et al., 2011). B-splines are a type of spline that does not touch any of the control points that are used to define the shape, instead following internal knots that are created based on where the control points on the spline are. However, they also have limitations in the curves that they can convey, especially when mapping more complex mathematical functions like integral curves (Mainar, et al., 2001), but it is very unlikely these limitations would affect the track generation project if B-splines were used to generate the tracks. Without using the knots, the spline generates as a Bézier curve which has its shape outlined by the control points (Chen & Wang, 2003).

Procedural generation of roads and specifically race tracks has been featured in many different games. Some games, especially ones that are just including it as a side feature in the game take more simplistic approaches into generating their tracks. *TrackMania Turbo* (Ubisoft Nadeo, 2016), which represents its tracks as individual pieces that are connected in its track editor, which is one of the game’s main selling points. For the Random Track Generator option that it includes, sections are pieced together one after another, which continues until the defined length it has been given is reached. *DiRT 4* (Codemasters, 2017) employs a similar tile-based approach for its stages, generating new stages from random track sections that are placed in order of each other which can then be saved and shared with other players. Outside of the *Your Stage* mode, the game also uses this to generate the tracks it uses in the career mode, in lieu of manually-designed tracks. *Excitebike 64* (Left Field Productions, 2000) has the special *Desert* track, which creates a simple endless desert terrain of hills and jumps to race through, trying to get the quickest time to the campfire checkpoints.

*Gran Turismo 5* (Polyphony Digital, 2010)includes a procedural track generator in the form of its *Course Maker* mode, which generates a basic layout and then allows players to edit specific variables on the track, like how many sections there are in the track and the individual complexity of each of the sections. It however doesn’t give players any direct control of the generated output themselves. For the project, the random track generator will handle all of the creation of the tracks, with users only being able to edit specific variables that alter the generation. Taking elements of the approach used in *Gran Turismo 5* would prove useful to aiding the development of the project.

## Saving/Loading Tracks

To improve the user experience of working with the project, the saving and loading of tracks is a feature that needs to be considered. Being able to store tracks that are generated makes it much easier to be able to test the output of the procedural track generator and makes certain elements of the generation easily repeatable. Being able to load also means that if a user generates a track they really like and want others to play, they can share it effectively by simply sending the file to the other person, which they can then use to load it up in their own version of the project.

Multiple different file types can be useful to store data and could be applied to the tracks that are generated. Binary files are a way for storing data without making it easily editable for any user that finds the file location of the file, which works well for things such as saves. For storing more sensitive information, these saved files can have encryption methods applied to them to make it even harder to be tampered with and access the data stored inside (AlTuhafi, 2022). For simply storing generated tracks, this method isn’t needed as there is no real need to stop players from being able to alter the saved output as no data is being shared between different machines. File formats like XML and JSON can be used to directly store classes in a save file, serializing stored variables and functions into files which can then be read back in as the class type (Oh, et al., 2015). JSON files are a data format that extend from the JavaScript programming language, and compared to XML don’t use tags which result in smaller file size to hold the same data, making them a more efficient choice for storing class data.

The simplicity of how this data is stored relies on whether the random generation method being used is pseudo-random and is completely deterministic based on the seed that defines it (Wen & Yu, 2019). If the method used is deterministic, only the seed needs to be stored in the save file, along with the other editable variables that are used to affect the track generation so that can be properly regenerated from. However, if it isn’t then instead the data for each knot along the track spline would need to be saved instead, which also means that the track can’t be reloaded and have some of the parameters tweaked.

## F-Zero Analysis

*F-Zero* is a series of futuristic, fast-paced racing games in which the player races to win against the other opponent racers and achieve the fastest time on the courses. All the course designs in the series follow the convention of them being looped circuits allowing for multiple laps.

The first game in the series, *F-Zero* (Nintendo, 1992), started out as a 2D game emulating 3D without drawing any polygons on the screen, using the *Mode 7* graphics mode that was in the *Super Nintendo Entertainment System* to apply transform effects to the background layer so that it appears as a 3-dimensional plane. To accommodate this, all the designs of the tracks in this game and the subsequent games using the same gameplay style are flat circuits without any verticality to the terrain. The original *F-Zero*’s tracks really focused in on the hard, tight corners and track design, making it harder for players to complete the races without destroying their vehicle.

With the release of *F-Zero X* (Nintendo EAD, 1998)on *Nintendo 64*, the series moved into full 3D gameplay, and along with it added more complexity to the design of the tracks, as they could now move in all 3 axes and innovate on the gameplay in new ways not possible before. This was then further refined with its sequel *F-Zero GX* (Amusement Vision, 2003) with the generational hardware jump to the *Nintendo GameCube*. Once the series had moved into 3D with *X* and *GX*, there were more options that could be used to make the circuit designs more interesting outside of just adding in more tight corners. Elevation (Figure 1) and rotation (Figure 2) are ways that the map designs more interesting for players without impacting the map gameplay.

A blue and green wireframe of a train

Description automatically generated with medium confidence  
Figure 1. Full 3D model of Big Blue 2: Quick Turn from F-Zero X (Nintendo EAD, 1998), showing the elevation changes.

A video game screen shot

Description automatically generated  
Figure 2. Screenshot from F-Zero GX (Amusement Vision, 2003) showing the layout of the map Mute City: Twist Road.

### F-Zero X: Procedural Track Generation

The game *F-Zero X* includes a PCG system that generates race tracks, which is used in the *X Cup* to generate a set of 6 circuit to make up the tracks raced in the grand prix. Because a manual track editor was included in the Japan-exclusive expansion *F-Zero X Expansion Kit*’s *Course Edit* mode, it can be understood how basic circuits in the game are created and some conclusions into the workings of the procedural generation can be made.

Based on how they are constructed in *Course Edit* mode, tracks in *F-Zero X* are made using points in a 3D space, which are always connected sequentially with the final track section always connecting back to the start position to create the complete circuit (Appendix 1). The tilt angle of each individual point can also be edited to allow more dynamic track designs, and the player also has control of how wide the track is and what track types are used in each section. Based on the research into ways of representing race tracks that has already been conducted, it can be concluded that *F-Zero X* is very likely using splines as a way to represent race circuits.

After observing a set number of tracks be generated using the game’s PCG system (Appendix 2.1), there are a few patterns that can be drawn out from it. The tracks that are generated seem to mostly focus in on a specific gameplay element and not mix and match them very often. For example, the system generated a tunnel track that was just a complete circle, and another track that featured lots of banked turns but not any changes in elevation. The track system always places the recharging areas only at the very end of the track; this is a convention that most of the manually designed courses in the game also follow but there are exceptions. There are elements from the base game that the PCG system has not been observed to generate, such as intentionally placed jumps, trap elements and *dart zones* (Nintendo, 1999). Neither the recharge requirement nor the limitation on object types exists in *Course Edit* mode, showing that this is exclusively something put into the PCG system.

While the rules in place of circuit creation laid out in *Course Edit* mode combined with the constraints imposed on the PCG system mean that an impossible track won’t ever be generated, it seems it doesn’t go beyond that to make them believable to players. In one instance, a track generated with a long straightaway leading into a downhill obscuring a sharp corner. Because the cars will pick up a lot of speed on the straight, it becomes very hard to slow down and not fly off the end of the track. Because of this, all 30 racers on the track (one player and 29 AI opponents) flew off the side of the track and got a Game Over, with the player needing to restart from the beginning of the race.

As found from the data collected on the tracks generated by *F-Zero X*’s PCG system, the average course length is 01:21.271 (Appendix 2.2), compared to the average for the game’s main courses of 01:30:101 (Appendix 2.4). The average top speed that was attained in each of the procedurally generated tracks was 1019km/h (Appendix 2.3). This is noticeably slower compared to 1121km/h for the manually designed courses (Appendix 2.5), but the value was skewed by the track *Port Town: High Jump* which has top speed of 1606km/h, bringing the average up by around 50km/h.

### F-Zero’s Gravity-Defying Movement

In the original game the tracks are flat, so more standard physics simulation will likely have been used. For the 3D games however, to keep the vehicles always connected to the track, the gravity direction would need to change to go along the circuit. Building on top of the observations from the *Course Edit* mode that *F-Zero X* very likely stores and represents its tracks as splines, one approach could be to keep track of the forces acting on the player and keep updating the position based on the distance they are moving along the spline. This would have the drawback of needing different implementations for each of the track types in the game, as vehicles on the flat road tracks will act differently in relation to the track spline compared to the cylinder sections.

Another approach would be to make use of the barycentric coordinates of any point on a mesh face. Barycentric coordinates are used to define a specific position on a mesh face, representing it a ratio between the three vertices (Floater, 2015). By sending a line trace downwards to get the current face of the mesh, the hit point can be used to get the barycentric coordinates on the hit face, which can be used to interpolate the normal value at each vertex of the face to find the current orientation the player vehicle needs to be at. This means that in the context of *F-Zero*, the vehicles will be able to traverse any smooth terrain thrown at it, which would apply for any terrain type that is implemented into the game.

### Believability in F-Zero Track Design

Because the *F-Zero* series is known for its high difficulty and high-speed, making players hone their reflexes with tight corners and difficult tracks, what makes a track believable and considered to be well-designed for the game doesn’t always line up with what would be believable for other racing games. *F-Zero* tracks are designed in a way that pushes players to get as much speed as possible, and then challenges them with manoeuvring through the tighter track designs. Based on the comparison of the manually designed and procedurally generated circuits in *F-Zero X*, there is a point where the tight, challenging corners become too difficult for players and begins to feel unfairly designed.

The design requirements that all *F-Zero* tracks must meet is that they form a continuous loop that goes back to the start of the track, and that they include a section of the recharge road element somewhere along their length; in almost all cases this falls at the end of the track.

# Research Methodologies

The project will be built using the *Unity* game engine and will contain both the generation of tracks as well as having a recreation of *F-Zero* gameplay for the tracks to be judged in their effectiveness against. The benefits of using *Unity* as the game engine to build the project in is that it alleviates the need for handling rendering of the scene manually, as well as implementing collision checks.

In the game world, tracks will be defined by a Catmull-Rom spline that outlines each point of the track layout, which is then used as the basis for the procedural generation of the track mesh. Because tracks in *F-Zero X* (Nintendo EAD, 1998) are smooth, singular flowing shapes, having a single spline represent the entire track allows for easy creation of that style, one that would be very hard to replicate using nothing but pre-made track pieces that were placed next to each other to represent the track instead. Catmull-Rom splines have been chosen as the type of spline to use because of their smooth characteristics (Yuksel, et al., 2011), creating natural feeling curves and reducing the chance of tracks generating with harsh, tight angles that don’t feel very good to play through and would negatively affect the believability of the track generation output. They also travel directly through the points given to them, simplifying the process of building the track from the chosen section elements. *Unity*’s built-in *Spline* package (Unity Technologies, 2023) will be used to handle the spline generation, implementing the basic functionality of spline elements into the project. The package’s automatic spline tangent generation also creates the curve through the points as a Catmull-Rom spline, automatically generating the tangent values to represent the correct curve.

To represent the different types of tracks that are in *F-Zero X* (Nintendo EAD, 1998), there are two different track types that can be generated: cylinders and roads. Cylinders are wide, pipe-like track elements that the vehicle can drive smoothly around their entire circumference, their size will be defined by a given radius value. Roads are flat tracks that have a set width and can generate with barriers that stop the player vehicle from flying off the ends of the track. To generate the roads, two more versions of the spline will be used to create the width of the road and define the boundaries, for the mesh to generate between each of the two positions that these lay out. These two track types can be used to make up almost every single track in *F-Zero X*, the other track types that the game features such as the *Pipe* and *Half-Pipe* are used very infrequently and only for small parts of specific stages, so not including them in the project won’t affect the perceived believability of the generated tracks.

The generation method decided for the project will incorporate elements from multiple of the PCG techniques outlined in the research. The beginning track layout is picked out from a basic axiom layout, with each section of the track being represented by a unique character symbol that makes up a string, using an L-system approach to represent the tracks internally (Jormedal, 2013). Each of the axioms used will represent very basic and rudimentary track loops to use as the initial starting point of the track designs. To allow for more variety and less limitations on the tracks themselves, the generated sections are not forced to conform to the limitations of the initial layout and can morph into whatever shape the newly added sections dictate. The track elements are split into straights, left and right turns, sharp left and right turns, loops, and corkscrew elements, defining the representations of each character in the track string. Each of these are defined in the code with the rotation they cover, as well as minimum and maximum length values. This basic string that is randomly chosen is then iterated over multiple times, changing out and adding extra elements throughout the track string based on the preceding section, similar to the iterations performed in the Wave Function Collapse algorithm (Gumin, 2016). The rules for how track sections are derived from analysis of the human-designed tracks from the game *F-Zero X*, looking at what track elements follow on from each other (Appendix 4.1). To add some more variety to the potential results and gain a better range of results to compare the effect on believability, an extra set of rules will be created once the initial wave of tracks have been generated, based on highlighting trends that fit in better with the *F-Zero X* designs, as well as including certain track section rules that don’t exist in the original game’s tracks (Appendix 4.2). The tracks generated will always use the newest version of the implemented rules, but saved tracks will store which set of rules they make use of so that the exact same track is always regenerated.

Once the final track string has been created, the track sections are converted into the knots on the track spline, generating the basic layout of the track. The height variations at each section of the track are then generated and applied to the string based on user-controlled editor parameters. Each track section will also have some random variety in its length, allowing for more variation in the layouts of the generated tracks. If the randomly generated track elements are far apart (which would result in a large stretch for the last knot to connect back to the beginning of the track) extra track sections are added in to smooth the transition back to the beginning, making for a much more natural ending to tracks. The system for this will function in two separate phases, first adding sections to get the end of the track behind the start point should it be needed, and then a second phase where the track turns towards the start position and generates track sections moving towards it. To help smooth over the transition back to the start point of the track, a final point on the track spline will be added directly behind the start point, reducing potential generation issues occurring directly on the starting line of the track.

In the process of investigating the tracks for their generated believability, the height variance of each individual track section, how many times the track is iterated on and the chance that boost panels generate affect the believability of the output will be tweaked using exposed controls that users can edit in the game project. Changes to the rules and length of the sections will also be applied depending on the findings of the initial set of results. The parameters used by the procedural generator have to be changed throughout the investigation so that there is a wider range of variation in the outputs, which can be used to derive which variables change the believability of the track the most. The parameters will be changed to have them more in line with the data that has been collected from *F-Zero X*, so that the generated tracks increase their believability and fit better with the tracks that were designed for said game.

The data for the track generation will consider the “bumpiness” or verticality of the track. Each point on the track will have a height offset value applied to it that will work to make the generated tracks more varied and believable than just being flat, this defines how bumpy the track generation is. To control this, the procedural track generator will have exposed parameters for the minimum and maximum height offset values that are possible. Another parameter for the maximum height change possible over one section will also be implemented, to stop tracks from having very steep changes in terrain. Making sure the bumpiness of the track is right will be a crucial part of making sure the track is good to play, if the track has too many steep changes it will obscure the player’s view of the oncoming track. On the other hand, making the track too flat could make players feel the generated tracks are boring to play on, and might cause issues with self-collisions if everything is on a very similar plane.

The values that the believability will be compared against are values obtained from first-hand research by playing *F-Zero X*, obtaining averages for track and lap times, top speeds, as well as compiling information such as track layouts and lengths. Due to *Nintendo* not having released any official information online about how *F-Zero X* works, and there not being any single official source that compiles information about the game, the data must be obtained from testing the game first-hand. To get a good representation of data for the game, each of *F-Zero X*’s 24 tracks will be played, and the data for times, individual lap times, and max speeds for each lap will be compiled as mean averages for the usable data in the game. Each track will also be broken down into the sections that make them up, making use of the track sections that have already been defined as the elements of tracks used by the track generator, so that the output of the procedural content generator can be more accurately analysed and compared in relation to the tracks from *F-Zero X*.

The believability of the procedurally generated tracks will be defined by comparing trends in the analysis of *F-Zero X* tracks against the generated output. Elements such as the average lap length, top speed and the number of elements in an individual track will be used to determine how believable the generated tracks are. With the data pulled from the tracks in *F-Zero X*,the procedurally generated tracks will be compared against for how far removed they are from the observed trends in the data, particularly focusing in on the comparing the difference in relationships between the values using Pearson correlation coefficient (Edelman, et al., 2021). Repeating trends in the generation will also be picked out to see whether the patterns can be observed in *F-Zero X*’s track designs, and then be used to compare them against values such as the time taken to travel a certain distance.

Comparisons into the believability of the generated track output will also consider the buildup of the tracks themselves, showing the believability of the rules used and highlighting where this could further be refined to make more believable tracks. To have more variety in the tracks that are updating, during the process of the investigation a second ruleset will be implemented to improve the believability of the output by changing the odds that track elements are inserted into the track. There will also be the possibility of the investigation changing the axiom strings to better explore how that will affect the believability.

The generation of track elements will not account for whether the track is self-colliding. This would require checking each section of the track against the others for if there are any collisions, and moving sections that are colliding with each other, requiring the affected segments to be checked again until the entire track has been checked to not be colliding with itself. For the scope of the investigation, it is not something that has to be addressed as any affected tracks can just be regenerated again. In a commercial, publicly released project this would be something that would need to be addressed so that players can’t be given generated tracks that do not function correctly. For example, *TrackMania Turbo* (Ubisoft Nadeo, 2016) simply swaps the section it tries to place if it doesn’t fit, move above it if possible and even step backwards through the track to try and find a better solution as it goes along. This approach wouldn’t be very applicable to this specific project, as *TrackMania* *Turbo*’s random track generator only has its tracks generate from start to finish, without any looping for multiple laps.

### *F-Zero X* Gameplay Representation

For the gameplay representation of *F-Zero X* (Nintendo EAD, 1998), the vehicle will use line traces to determine its distance from the track, and then calculate the rotation it needs to be at using the barycentric co-ordinate position that is hit on the mesh face, and interpolating the normal values at each of the face’s vertices to get the direction it needs to be facing. The bonuses of this system are that the vehicle will drive seamlessly on both the flat tracks and cylinders, if a spline position-based approach was taken while ignoring any actual colliders on the object, it would require separate movement systems depending on what track type the vehicle is driving on. The vehicle itself will have a speed value which updates its position on the forward vector every single frame, allowing for the vehicle movement to seamlessly navigate the changes in height and rotation that might be presented on the track. Because *F-Zero X* has very tight feeling movement, this approach keeps the vehicle feeling snappy and responsive to the player’s control, which is something that would be much harder to replicate using *Unity*’s physics system directly through its forces, and then rotational torque to move the forces around based on the current track rotation that is needed.

To replicate the player controls of the gameplay in *F-Zero X*, a player will have access to most of the movement options that are present in the game, but instead mapped to keyboard controls for ease of use on PC, the platform the project is being created for. On top of the traditional movement options that would be found in most racing games, players will also be able to strafe left and right as they would in *F-Zero*, and also be able to drift which makes the vehicle’s turn radius much tighter for more precise and difficult sections of track. The noticeable gameplay nuances of *F-Zero X*’s vehicle handing, the vehicle slowing down as it goes into turns and players being able to hold their top boosting speed, will also be implemented into the project to get a deeper reflection of how vehicles in *F-Zero X* feels to control.

One of *F-Zero X*’s main gameplay systems is the vehicle energy, which is used as the health of the player. Energy can be used to boost and go faster, and whenever players collide with the barriers on the edge of the track, or any other racers, they lose some of their energy. Energy is gained back in the *Pit* elements on the track. These elements will be included in the project so that it more accurately reflects the gameplay of *F-Zero*, these features have been implemented into every game in the series so it is imperative that they are recreated for the project.

The actual values used for the driving speed will be created based on trying to be as close to the feel of *F-Zero X* as possible. There is not any information available for how the game’s driving system actually works, and how the values used in-game for elements such as the vehicle speed and track length actually correlate to each other in the game’s engine, so these values used in the project will be approximations based on first-hand comparisons for the vehicle controls in the project and *F-Zero X*. For the project one unit in the *Unity* game engine has been interpreted as representing 1 metre, so that the relationship between player speed and track length is proportional the same way it is in *F-Zero X*, in turn allowing for direct comparison of the data taken from the procedurally generated tracks in the project and the data from the manually designed tracks in *F-Zero X*.

A key consideration that must be considered when deriving results from and analysing the data is that the believability of tracks and the feeling of whether they fit with the tracks that have been designed for *F-Zero* is that ultimately is subjective in nature and will differ from person to person. Some people may agree with the results and findings from the data, but others may consider certain elements that haven’t been investigated more important for what defines an *F-Zero* track. To combat this, a wide range of testing has been done in the original game such that there is a wide breadth of data to compare against. This data can be used to compare against and look at the track designs objectively through quantified values, it removes the need to analyse each of the tracks in more subjective ways such as the feel of the player which is subjective to the individual playing the track and would introduce bias into the findings of the investigation.

# Results and Findings

All the data collected from the output of the procedural track generator has been compiled into a spreadsheet that separates the tracks to sections based on the parameters that were used to generate them (Appendix 5). Each of the tracks that have been generated have also been given arbitrary names to allow for slightly easier differentiation between each of them, but these haven’t been taken into any consideration when analysing the results; the names do not hold any significance to the investigation and were assigned based on noticeable features of the track design.

The first batch of tracks generated and tested by the procedural track generator used the default values for the track generation, which were implemented to be used as a simple baseline to begin the investigation from. Aside from the tracks that were rejected due to the track colliding with itself, some of the tracks had issues with being bumpy, obscuring the view of upcoming twists and turns from the player’s view. All the tracks generated based on the initial parameters of the project are much longer than those that appear in *F-Zero X*, which is particularly noticeable on the sections added to loop back to the beginning of the track. Compensating for this initial trend in the results, the number of iterations performed over the initial axiom track layout as well as the length of the axiom layout itself were in turn reduced to get a wider spread of data. The other exposed parameters for to observe an impact on how those changes affect the perceived believability of the generated tracks.

### Creating the Secondary Generation Rule Set

From the investigation, the second set of generation rules were developed based on the data as well as observed patterns in the track generation. Because corkscrews are defined as being 90° rotations of the track, there were times that a single rotation caused the track to sprawl out in another direction and significantly increase the length of the track, as there is a longer distance that needs to be covered to get back to the start point. It was also causing the flat tracks that were generating with rotations at the very end that were impossible for the vehicle to drive across. So, the rules were altered such that corkscrews were rarer but more likely to follow each other when they did generate, so that 180° rotations could help alleviate the issues that were being observed as patterns under the current rule system.

Another key detail that was to be addressed with the secondary rule set is the variety of the potential track layouts. As of right now, if any section combination isn’t present in a track in *F-Zero X*, it won’t be added into the track string. To encourage more variety in the output as well as making the generated tracks feel more unique, more combinations were added into the generation rule pool while keeping the major biases that exist in the original game’s rules. This way, the tracks can generate in a way that is different to *F-Zero X* but makes them more inline with the design philosophy of creating unique and memorable designs for each of the tracks. Finally, the new ruleset also reduces the chance that loops and corkscrews generate continuously when looping back to the beginning of the track, as there were many observed instances during the track generation of multiple loops being right next to each other. The finalised, secondary ruleset was then created based on the elements that needed to be addressed (Appendix 4) and implemented into the project, making sure that the old track save files would still generate on the old system to make sure the layouts were the same.

To analyse the rate of certain elements generating on the track, the generation rate of each track element per track has been calculated. This was done by totalling how many times a track section generates across the entire track data set for both of the rules, and then have it divided by the number of symbols that make up the tracks, compiling the data from both the generated output (Appendix 5), and the track section breakdown of each track from *F-Zero X* (Appendix 3). By getting the chance rates for an element to be used, the data across all tracks can be compared accurately, and won’t be skewed by the procedurally generated tracks having tracks much longer than those from *F-Zero X*. For this, the smoothing sections (represented by a B in the grammar) that are used to rotate the track to the correct orientation when looping back to the start point have been ignored, as they do not appear in the track structure breakdowns from *F-Zero X* and are only ever placed once per procedurally generated track.

Figure 3. Frequency of each track section being generated.

The collected frequency data shows that the initial generation rules are closer to the frequency values of each section that appears within *F-Zero X* for the straight and sharp turn elements, which is to be expected as they are controlled predominantly by the main axiom iterations. With the initial implementation of the looping sections, loops and corkscrews were much more common on the track designs, which the data reflects by showing a higher generation amount compared to the frequency of their utilisation within the track designs in *F-Zero X*. Both the initial and secondary data generation rules make use of left and right turns less than the tracks from the game, which would be a good place to improve the generation further with a third ruleset that has more incentivisation of their placement.

The size of the original axiom string and the number of times the axiom is iterated on has a direct impact on the length of the track that is being generated. The most used axiom iteration value is 1, as it was the value that had the length that got closest to covering the range of track lengths from the data of *F-Zero X*’s tracks.

Figure 4. Distribution of track length generations for the differing axiom iteration values.

By taking these values and comparing against the distribution of *F-Zero X*’s track lengths,

Comparing the distribution of track length against the top speed and the time per lap highlights which of the parameters got the generated tracks more in line with the correlation coefficient that is represented in the tracks from *F-Zero X*, thus creating tracks with a higher believability.

The speed of the laps is determined by the lengths of the tracks, but also by the rate at which boost panels are placed onto the track. This value is increased a lot on loops to reflect the increased placement rate that is observed in *F-Zero X*’s tracks.

The bumpiness values of the track that are controlled by

* Separate outputs into the changes in generation parameters and rules to show the differentiation, to be used in the discussion and analysis

# Discussion and Analysis

The parameter of the procedural track generation algorithm that was observed to have the most effect on the output and the differentiation between the tracks being generated and the tracks from *F-Zero X* is the iterations on the axiom applying the rules to add in extra sections into the track. Based on the algorithm presented for procedural generation of tracks, getting the track length to correlate with the data proved difficult. Partly this was due to the starting length of the axioms, as they directly contribute to how big most of the track is going to be generated. By changing the starting lengths of the axioms, the

A computer screen shot of a race

Description automatically generated

Figure 5. Track generated with no iterations on the axiom

Not performing any iterations on the track layouts at all kept the tracks on the shorter end of the length scale for *F-Zero X* track lengths, but it also produced very similar layouts that had issues smoothly looping back to the start point like in Figure 5. While the tracks were in line with the scale, they also had an 80% self-collision rate. The generated tracks also only follow the base axiom layouts with a single element added onto the end to loop back to the beginning, as none of the rules are applied to any portion of the track. All of the cylinder tracks could still be driven around, but certain positions around the cylinder would have to be avoided at certain times so that the player vehicle doesn’t clip into the other track and either skip a point in the track or get pushed off of it entirely.

One of the key factors to take into consideration with *F-Zero X*’s tracks and making the output of the track generation feeling believable, is to not have the generated tracks feel completely derivative of *F-Zero X* and instead have uniqueness and variety that appears within them. Each of the tracks that were designed for the game have sections or elements that they use to differentiate themselves from each other, beyond just the layout and visual changes present on each of the tracks. This reflects in the data collected about the frequency of the tracks, which shows how the rules and generation elements have differed from those that appear in the base game. The differences that appear from both rule sets can add more variety to the outputted tracks, especially in the case of the second rule set which implements more unique section combinations into the mix to add more variety.

One of the major elements of making the generated *F-Zero* tracks feel believable is the height variation values that are used between each section of the track. When the tracks are too flat, they can feel quite repetitive, but when they are too bumpy the player’s view of upcoming turns is often obscured, negatively impacting the gameplay experience on the generated tracks as observed in Figure 6.

A screenshot of a video game

Description automatically generated

Figure 6. Example of player's view of track path being obscured by a large change in height variation

The changes in the bumpiness controlled by the vertical offset parameters also had an observable impact on the self-collision rate of the procedurally generated tracks. Both the range of height variation values given and the clamped amount per section contribute to this, as a small range of possible variations limits the chance that a track will vertically avoid a self-intersection naturally. This proved to be more of an issue with the cylinder track elements as they are mesh that goes outwards in three dimensions. Cylinders have their radius defined as 20 metres, so any value lower than double that (to account for the full diameter of the mesh) increases the chances of self-collisions and thus decreases the believability of the track.

However, going too far in the opposite direction and pushing the height variation into very high values also can cause the same self-intersection issue, this time being caused by the sharp turns that can generate from the bumps in the road that it would potentially cause. The height variation values become a balancing act to find the sweet spot for believability in the generation, looking at the failure rate of self-intersections, the values that proved the highest success rate were a height variation minimum of 50, maximum of 200 and the maximum variation per section being 50; in combination with the secondary rules that have been implemented into the project a self-collision avoidance rate of 70%. Because of the height variation values being larger than the diameter of the cylinder, each of the cylinder tracks were still completable, leading to a final success rate of 70%.

After observing the height variation value changes, and how they don’t have much of an effect on the length of the track and therefore the lap times, they can be considered less important criteria for the believability of a track. They can negatively impact the track believability through obscuring player views of upcoming track elements and increasing the likelihood of self-intersections, but don’t contribute much to getting the tracks completely accurate to *F-Zero X*, which already has some tracks with little to no height variation in them.

The definition of the track sections themselves also plays a hand in the believability of the output. With each track being made up of elements that define a single section, the more sections that are defined the more nuance that can be added to the generated output. The generation of corkscrews in the output falls into this category, they were defined as being 90° rotations of the track to break it down for the generation, but in *F-Zero X* there are corkscrews that are smooth, singular rotations of 180°, and other angles. The types of turns could be broken down further too, implementing track features such as banked turns, and half loop inversions could be incorporated in the track design. For the purposes of the investigation, the track sections were not a limiting factor on the believability of the generation, as more detailed track section definitions would require more complexity in the generation system and would negatively affect the believability if not implemented and accounted for well.

The difference in the track section generation frequency between the generated output and the tracks from *F-Zero X* was most likely caused by the separate generation of the track loops once the main track has been generated from the axiom. Because it used a separate method for picking out the track sections that were following on from each other given that it had a specific end point to attempt to reach, it doesn’t follow the rules laid out by the main track generation algorithm. The generation system was built in this manner to allow for more variety in the track layouts and trying to impose as little limitations as possible for what the procedural track generator could output, but that came at the cost of having to work around potentially awkward loops being created. This also ties into the wide variety of track lengths present across all of the different axiom iteration counts; with their being so many potential directions a track can generate off in, the loop back to the beginning can become very long. This is especially evident in generated tracks that iterated on the axiom twice and sprawled outwards away from the start point with little looping, which then required all of that distance to be covered again to get back to the start and complete the loop. With this method being used, it makes the number of iterations on the axiom very significant to the believability and accuracy of the output track designs, as the tracks get exponentially longer the more iterations on the axiom that is created due to the generation rules adding in another section between each one on the track. The length of the starting axioms that are chosen also have this effect on the believability as their length dictates how big the pre-loop track design is going to become.

### Analysis of Gameplay Recreation

Outside of the gameplay abstractions that have already been outlined, the goal of the gameplay side of the project is to play and feel as accurate to *F-Zero* X as possible, so that the believability of the tracks can be more accurately ascertained. To measure how accurate the gameplay representation of *F-Zero X* is to the main game without incorporating subjective details of the gameplay like the gameplay feel, the average time taken to travel 1km across the tracks can be taken to measure the variance between the project’s gameplay recreation and that of the original game by utilising the data sets of the track lengths and the average lap times from the procedurally generated tracks (Appendix 5) and the data obtained from *F-Zero X* (Appendix 3).

The average time to travel 1km (defined using the average lap time divided by the track length) for the *F-Zero X* tracks is 4.423 seconds, while for the generated tracks it averages to 4.176 seconds. This difference is fairly negligible and shows that the gameplay recreation is accurate to *F-Zero X* in the terms of the vehicle movement speed. The extra variation can be attributed to the amount of boost sections that generate along a track; boost sections generate more often through loop elements, and sampling tracks that have multiple loops in close proximity to each other reveals a pattern of them having a lower time to travel 1km, as speed can be chained through the boost panels. Increasing the boost panel chance would allow players to go even faster around tracks, skewing the value and making the track themselves less believable.

A notable limitation with the method for approaching the player movement, by casting a ray down towards the track mesh and basing the rotation from an interpolated face normal value, stems from the very high speeds the player vehicle can move at in the project. Because the player can be moving multiple metres for every physics step in the engine, it is possible for players to potentially clip through the track on sharp elevation and rotation changes. The effects of this are limited to features that wouldn’t be believable for *F-Zero X* such as very sharp turns that go beyond 90°, but it is still a limitation that could be addressed by looking at other methods to move the player around the track. The spline-based movement approach that was outlined in the research would potentially be a remedy for the issues as it would use an offset value based on the distance along a spline to calculate all positions. Another way to tackle removing this limitation would be to break down the movement of the player into multiple steps instead of all at once, reducing the distance that is being moved all at once, but this would come at the cost of more ray casts and thus making the project less efficient.

### Analysis of Generation Algorithm

A major part of the generation algorithm that has caused issues during the investigation is the inability for it to detect self-collisions and adjust accordingly. This was outlined in the research methodologies as something that is not necessary, as for the purposes of the project the seed can simply be re-rolled and a new track generated. However, the failure rate due to self-collisions was very high, making collecting the data for the investigation a much longer process than it had to be. For applying the generation algorithm into a publicly available project that is used to generate content in a game, self-collision detection is an imperative feature that needs to be implemented to ensure the usability of the generated track output.

Another element of the track generation that the track generation element struggles with is looping the tracks back to beginning. The elements added in to smooth out the ending sections work to make a more natural path back to the start of the track, rather than just a hard path that cuts through everything. Upon the investigation of the track generation data, it was observed that the solution presented for the looping is not wholly effective at eliminating unnatural loops, as shown in *Figure 6*. Part of this result may be the more rigid parameters that the results are put under for the final section, with the turning and straight sections being forced to be separate (which can have results of multiple of the same generation element generating right next to each other).

A computer screen shot of a race

Description automatically generated

Figure 7. Multiple loop sections generated adjacently

While the rotations are always set accordingly, sometimes the automatic smoothing of the spline can cause the values to not line up correctly, resulting in some of the directions still being off and causing unintended generation effects. This can be observed in the loops in Figure 6, as the automatically generated tangents don’t have them perfectly rounded. This was caused by leaving the generating of the spline to the *Unity* spline package (Unity Technologies, 2023), as it just used the default tension value for Catmull-Rom splines to build the curves at each section. Taking control of this manually would allow for more believable track output as smoother, more naturally flowing curves could be created for the player. Part of the solution for alleviating the issues caused by the track generating its loop back to the initial start point is in the aforementioned self-collision checking, which should be able to detect that the final segment of the track is abnormal and needs to be altered. Another solution to this specific issue would be to enforce basic track shapes more strictly onto the design, so that the extra added in sections must end on a set point defined in the initial axiom layout so that the long looping sections aren’t needed to be generated as often.

# Conclusion

A track generator project that has an accurate recreation of *F-Zero X*’s gameplay system has been created, with tracks being able to be generated based on set parameters controlled by the user and rules that dictate how the track iterates on itself to create the track layout. To mirror the way tracks generate more accurately in *F-Zero* and to have adequate data to compare the believability of the project against, the game *F-Zero X* was investigated to collect data for each of the tracks in the game. Using the project that has been developed, a sample of tracks were generated with changing parameters to determine the effect they have on the believability of the generated output, and comparing the data to derive the parameters that affected the generation the most.

The generation model developed makes use of rules that follow *F-Zero X* design patterns while also adding some extra variety and uniqueness to them. The method for looping the track back to the beginning works to improve the believability but has been observed largely increasing the lengths of the tracks that are generated by going off into the wrong direction, creating sharp corners and intersections that reduce believability. The prevalence of self-collisions was also underestimated in the initial methodologies, it makes the tracks completely unbelievable and oftentimes impossible to drive across. For the purposes of the project and investigation another track can just be generated, but if the generation algorithm was to be utilised in a commercial game project, the self-collisions would need to be addressed.

With the generation algorithm presented in the investigation, the most important parameter that affects the believability of procedurally generated *F-Zero X* tracks is the number of iterations of rules applied to the base track layout, which directly controls the length of the tracks, which is one of the most important parameters. The believability of the length of the track can most easily be affected from the number of iterations performed, which is exposed as an editable parameter inside the project. However, it can also be affected by changing the length of the base track axioms for the base layout. Getting the height variation values correct is important in not having the track believability negatively affected, as it can go against the design considerations of the project by blocking players from being able to see upcoming track elements and increasing the odds of self-collisions. However, it isn’t one of the criteria that has much of a positive effect on the believability as the change in track length based on the height variation is marginal.

# Recommendations

To improve the output of the procedural generation such that it is equipped to deal with generating content inside of a commercial game, self-collision checks are something that could be implemented to further the project. To simplify the collision checks and avoid any checkpoints clipping into other elements of the tracks, each track section could be treated as an oriented bounding box that is larger in width and height than the track, which is then compared to see whether it is connected to any of the other track elements (excluding the ones that it is directly connected to). Another avenue that the project could be taken is looking into implementing track type changing throughout the track, adding the track jumps that are present in *F-Zero X*.

The presented research could be built on top of and taken further by implementing AI-controlled racers being able to race on the track, which would better emulate *F-Zero X*’s standard *Grand Prix* mode and not just the single-racer Time Attacks (Nintendo, 1999). By implementing a procedurally generated racing line to be followed by the computer, the enemy racers could be added to drive around the racers. The most basic form of this would be to have the racers simply try and follow the middle track spline to be used as the racing line, but it would be a very basic implementation that would only work well on the road track type and not the cylinder/pipe, which have the track spline detached from the points they are generating from. AI racing lines are something that has been investigated thoroughly, with some systems already having been researched and developed for procedurally creating ideal racing lines for standard racing tracks (Tan, et al., 2008), as well as solutions for optimising the results in multiple different dimensions (Xiong, 2010) which could be used as a basis to design a system for effective racing lines for *F-Zero*, and looking.

Based on the data from the racing lines, it could then be extrapolated to do the inverse, generating racing lines and creating tracks from that data. Once again, a simple approach would be to just use the racing line as the main track spline, but this would very likely lead to the ideal generated racing line changing from what was originally used to generate the track.

# References

Adams, C. & Louis, S., 2017. *Procedural Maze Level Generation with Evolutionary Cellular Automata.* s.l., IEEE, pp. 1-8.

AlTuhafi, A. W., 2022. Adaptation for Vigenère Cipher Method for Auto Binary File Ciphering. *ITM Web of Conferences,* 24 February, Volume 42, p. 1017.

Amusement Vision, 2003. *F-Zero GX.* s.l.:Nintendo.

Archer, T., 2011. *Procedurally Generating Terrain.* Duluth: s.n.

Bouillaguet, C., Martinez, F. & Sauvage, J., 2020. Practical Seed-Recovery for the PCG Pseudo-Random Number Generator. *IACR Transcactions on Symmetric Cryptology*, 2020(3), pp. 175-196.

Chen, Q. & Wang, G., 2003. A Class of Bézier-like Curves. *Computer Aided Geometric Design,* 20(1), pp. 29-39.

Codemasters, 2017. *DiRT 4.* s.l.:Codemasters.

Dahlskog, S. & Togelius, J., 2012. *Patterns and Procedural Content Generation: Revisiting Mario in World 1 Level 1.* s.l., ACM, pp. 1-8.

Edelman, D., Móri, T. F. & Székely, G. J., 2021. On relationships between the Pearson and the distance correlation coefficients. *Statistics & Probability Letters,* Volume 169, p. 108960.

Floater, M. S., 2015. Generalized Barycentric Coordinates and Applications. *Acta Numerica,* 27 April, Volume 24, pp. 161-214.

Gardner, M., 1970. Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American, Vol. 223, No. 4,* pp. 120-123.

Griffin, J., 2018. *Retrospective of Burnout Paradise: Laying the Foundation for a Generation of Open-World Racing Games.* Prague, 4C Conference.

Gumin, M., 2016. *Wave Function Collapse.* [Online]   
Available at: https://github.com/mxgmn/WaveFunctionCollapse  
[Accessed 22 November 2023].

Hendrikx, M., Meijer, S., van der Velden, J. & Iosup, A., 2013. Procedural Content Generation for Games: A Survey. *ACM Transactions on Multimedia Computing, Communications, and Applications, Volume 9, Issue 1,* pp. 1-22.

Hong, S. L. & Liu, C., 2015. Sensor-Based Random Number Generator Seeding. *IEEE Access*, Volume 3, pp. 562-568.

Images Scientific Instruments, 2007. *How Our Random Number Generators Work.* [Online]   
Available at: https://www.imagesco.com/psi/random-number-generator-how-it-works.html  
[Accessed 22 November 2023].

Iosup, A., 2011. POGGI: generating puzzle instances for online games on grid infrastructures. *Concurrency and Computation, Vol.23 (2),* pp. 158-171.

Jason Hayes, K. T., 2007. *The Creation of SAINTS ROW's Open World Cityscape: Stilwater.* San Francisco, GDC.

Johnson, L., Yannakakis, G. & Togelius, J., 2010. *Cellular automata for real-time generation of infinite cave levels.* s.l., ACM, pp. Article 10, pp. 1-4.

Jormedal, M., 2013. *Procedural Generation of Road Networks Using L-Systems,* s.l.: s.n.

Ko, M., 2015. *Math for Game Programmers: Parallel Random Number Generation.* San Francisco, GDC.

Kruse, J., Connor, A. M. & Marks, S., 2022. Evaluation of a Multi-agent "Human-in-the-loop" Game Design System. *ACM Transactions on Interactive Intelligent Systems, Volume 12, Issue 3,* pp. Article 19, pp. 1-26.

Lagae, A. et al., 2010. A Survey of Procedural Noise Functions. *Computer Graphics Forum, Volume 29 (8),* pp. 2579-2600.

Left Field Productions, 2000. *Excitebike 64.* s.l.:Nintendo.

Lindenmayer, A., 1968. Mathematical Models for Cellular Interactions in Development II. Simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology, Volume 18 (3),* pp. 300-315.

Loiacono, D., Cardamone, L. & Lanzi, P. L., 2011. Automatic Track Generation for High-End Racing Games Using Evolutionary Computation. *IEEE Transactions on Computational Intelligence and AI in Games, Volume 3 (3),* pp. 245-259.

Mainar, E., Peña, J. & Sánchez-Reyes, J., 2001. Shape Preserving Alternatives to the Rational Bézier Model. *Computer Aided Geometric Design,* 18(1), pp. 37-60.

Mojang Studios, 2011. *Minecraft.* s.l.:Mojang Studios, Xbox Game Studios.

Nintendo EAD, 1998. *F-Zero X.* s.l.:Nintendo.

Nintendo, 1992. *F-Zero.* s.l.:Nintendo.

Nintendo, 1999. *F-Zero X Instruction Booklet.* s.l.:s.n.

Noll, L. C., Mende, R. G. & Sisodiya, S., 1998. *Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of chaotic system.* United States of America, Patent No. US5732138A.

Oh, J., Kwon, J.-w., Park, H. & Moon, S.-M., 2015. *Migration of Web Applications with Seamless Execution.* s.l., ACM, pp. 173-185.

Polyphony Digital, 2010. *Gran Turismo 5.* s.l.:Sony Computer Entertainment.

Prasetya, H. A. & Maulidevi, N. U., 2016. Search-based Procedural Content Generation for Race Tracks in Video Games. *International Journal on Electrical Engineering and Informatics, Volume 8 (4),* pp. 774-786.

Prins, V. L., Prins, J., Preuss, M. & Gómez-Maureira, M. A., 2023. *StoryWorld: Procedural Quest Generation Rooted in Variety & Believability.* Lisbon, Association for Computing Machinery, pp. 1-4.

Santiago, D., 2019. *Procedurally Crafting Manhattan for 'Marvel's Spider-Man'.* San Francisco, GDC.

SideFX, 2023. *Game Development.* [Online]   
Available at: https://www.sidefx.com/industries/games/

Stålberg, O., 2018. *Wave Function Collapse in Bad North.* s.l., Breda University of Applied Sciences.

Tan, C. I., Chen, C.-M., Tai, W.-K. & Yen, S.-J., 2008. *An AI Tool: Generating Paths for Racing Games.* Kunming, IEEE, pp. 3132-3137.

Togelius, J., Yannakakis, G. N., Stanley, K. O. & Browne, C., 2011. Search-based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games, Volume 3 (3),* pp. 172-186.

Ubisoft Nadeo, 2016. *TrackMania Turbo.* s.l.:Ubisoft.

Unity Technologies, 2023. *About Splines | Splines | 2.5.1.* [Online]   
Available at: https://docs.unity3d.com/Packages/com.unity.splines@2.5/manual/index.html

Wen, Y. & Yu, W., 2019. Machine Learning-Resistant Pseudo-Random Number Generator. *Electronic Letters*, May, 55(9), pp. 515-517.

Xiong, Y., 2010. *Racing Line Optimisation,* s.l.: s.n.

Yu, D. & Hull, A., 2011. *The Full Spelunky on SPELUNKY XBLA.* s.l., GDC.

Yuksel, C., Schaefer, S. & Keyser, J., 2011. Parameterization and Applications of Catmull-Rom Curves. *Computer Aided Design,* 43(7), pp. 747-755.

# Appendices

## Appendix 1 – F-Zero X Expansion Kit Course Edit Footage

A link to YouTube footage of the *Course Edit* mode in *F-Zero X Expansion Kit* being used to construct a custom level.  
Available at: <https://www.youtube.com/watch?v=Gcv2PpQTfn8> [Accessed 13 November 2023].

## Appendix 2 – F-Zero X Generated Track Data

All data from the game was taken using the same vehicle and configuration to make sure they as constant as possible with each other. As *F-Zero X* has no save system for the procedurally generated tracks that it generates in the *X Cup* mode, no averages could be taken for the data of each individual track.

### Appendix 2.1 – Generated Tracks

A document of the generated tracks from the *X Cup* procedural track generation in *F-Zero X*.

### Appendix 2.2 – Average Track Length

A table of the average track length of the procedurally generated tracks.  
A screenshot of a data

Description automatically generated

### Appendix 2.3 – Average Top Speed

A table depicting the top speed of each of the tracks.A table with numbers and a number

Description automatically generated

## Appendix 3 – *F-Zero X* Track Data Analysis

A spreadsheet containing data calculated from playing through each of *F-Zero X*’s 24 tracks three times to determine averages for the data presented.

### Appendix 3.1 – Average Lap Time Based on Track Length

### Appendix 3.2 – Distribution Average Time to Travel 1km compared to Element Amount

Because the tracks in *F-Zero X* all have their lengths defined in kilometres, a relationship between how fast a player can normally travel on each track and how many different versions of each element have been used can be established.

## Appendix 4 – Track Section Generation Rules

### Appendix 4.1 – Base *F-Zero X* Generation Rules

The initial track generation rules that are derived from how tracks in *F-Zero X* are built up. The chart displays which of the defined track section elements follow on from each other. The data was gathered from all the 24 tracks available in the game.

A table with numbers and letters

Description automatically generated

These results are converted into percentages, which are then used as the probabilities that a track section is inserted into the generated track when iterating on each of the existing sections.

A table with numbers and percentages

Description automatically generated

### Appendix 4.2 – Derived Generation Rules

The derived track generation rules that were created after the first batch of tracks were generated, aiming to be more in line with the design rationale of *F-Zero X*, and including track combinations that don’t exist in the original game’s track designs.

A table with numbers and percentages

Description automatically generated

## Appendix 5 – Track Generation Data

### Appendix 5.1 – Tracks Generated Using Base *F-Zero X* Generation Rules

### Appendix 5.2 – Tracks Generated Using Derived Generation Rules

## Appendix 6 – Track Generation Files

Contains all the files that can be used to load each of the track files used in the data collection, as well as the result output used to inform the Track Generation Data.