**Creation of a data-driven algorithm for use in content creation**

Isaac Fagg

S4920008

Bournemouth University

BSc Games Design

**Abstract (0/200)**

I wish to request that my dissertation is assessed in accordance with marking option:

1: Lit Review 1400 words (30%); Dev Report 3000 words (55%);

* 1. **Introduction (315/1400) – 30 References**

As the gaming industry continues to expand any content that hopes to engage a wide audience must be able to accommodate for the varied preferences. However, designing and creating individualised game content would take up too much time and resources. Instead there has been a rising interest in automatic generation of game content to assist with producing

* 1. **Background/Context**

Context – Automatic Content Generation

Other Games

Opinions/Ratings

**1.3 Research**

**Procedural Generation**

**General Procedural Generation**

# Towards a Generic Framework for Automated Video Game Level Creation

A Multi-Faceted Surrogate Model for Search-based Procedural Content Generation

Evolving Levels using Grammatical Evolution

Mixed Initiative Content Creation

Procedural Level Design for Platform Games (Component based prodGen)

**Search-Based Procedural Content Generation**

The representation of game content is a central question when it comes to evolutionary computation (Togelius, 2010). Search-Based Procedural Content Generation (SBPCG), which is a type of procedural generation that tests the generated content on its fitness based on set parameters. This is used in conjunction with an evolutionary algorithm that changes what the parameters can be. When using SBPCG the representation of the content is very important as it determines how the algorithm analyses the content.

**Track Based Procedural Generation**

Interactive evolution for PCG of tracks in high-end racing games

Automatic Track Generation for High-End Racing Games Using Evolutionary Computation

TrackGen: An interactive track generator for TORCS and Speed-Dreams

**Personalised Procedural Generation**

Adapting Models of Visual Aesthetics for Personalized Content Creation

Towards Player-Driven Procedural Content Generation

**Player Behaviour**

**Player Modelling**

Player Modelling

Defining Personas in Games Using Metrics

Experience-Driven Procedural Content Generation

Measuring the experience of digital game enjoyment

Modelling Player Experience for Content Creation

Play-Persona: Modelling Player Behaviour in Computer Games

**Racing Games**

Making Racing Fun Through Player Modelling and Track Evolution

Towards automatic personalised content creation for racing games

Computational Intelligence in Racing Games

**Track Analysis**

Towards a Generic Method of Evaluating Game Levels

**Analytics - Include**

Learning Analytics for Serious Games

# Tracking Real-Time User Experience (TRUE): A comprehensive instrumentation solution for complex systems

**Evolutionary Algorithm**

**Other**

The big five personality dimensions and job performance

**1.5 Other**

Setout to do

What I did it in

Original Plan

Mirrored Research/Development

**2 Development and Implementation (1150/3000)**

Benefits/Pitfalls

Why?

Analytical – Work well/Didnt work well

**Version Control**

As I worked across 3 different devices (Home PC, Laptop and University Computers) I needed a way to reliably transfer data between them. I decided to use GitHub as it a popular cloud-based version-control system, which is available on all platforms. Commits were performed regularly during and after any significant work.

**Unity Engine**

I chose to use Unity Engine to develop the game as it was the platform, I was most familiar with as well as providing excellent tools such as Analytics and the ability to port to mobile devices.

2D

**Gameplay**

The car is operated by the user, who can use buttons to turn or brake. Originally, I had wanted to allow the player to control the car’s acceleration and speed, but found that it led to more complexity when

The interface currently implemented is very basic, and if I had gone through with releasing it onto an app store, I would have revisited it. I also never got to include audio to the game, as it was the last thing I had planned to add before releasing.

Checkpoints are placed around the track and are used to ensure the player is progressing around the track in the right direction. When the player crosses the finish line, they start a new lap.

Being able to place a certain number of checkpoints equally along the track was a problem. When making the path script, I had included a function that creates evenly spaced points along the track, where you entered the desire space between the points, and it would recreate the path with the points in the new locations. To get the desired number of points, I would have to figure out the length of the track and then divide it by the desired count. Except due to the length of the track being an estimate, the spacing would sometimes not return the wanted number. To fix this, I created another function which would iterate through multiple spacings until it returned a path with the desired number of points.

**Track Representation**

**Track**

Each track is stored as a sorted array of two-dimensional points, in order to keep file sizes to a minimum. Once the track points are needed, they enter a generator which calculates new control points in order to create a path.

The curve is then closed, so that the track can be completed multiple times. I decided to have the tracks be closed circuit rather than an infinite road, as it would yield better results in the evolution process.

The track is cut into segments, each segment is comprised of 3 points, the start, middle and end points. However as not every track will have a total point count divisible by 3 the segments start from each point and include the next 2 points after them. This results in each point being included in 3 different segments. [FIGURE].

Using vector maths, the distances and angles within the segment are calculated. With these measurements, it can be determined what direction the segment turns as well as its size and area. This is done for each point in the track to create an array of segments which are then used in the evolutionary algorithm.

**Track Generation (Pseudo-code)**

When making the random track generator, I needed to decide what type of random track to generate. Togelius (2007) used three different ways to make a random track: Straightforward, Random Walk and Radial. As they found in the paper, Random walk produced tracks that could be hard to drive on naturally and Radial method tracks made tracks that looked very similar (flowerlike).

I decided to go with a Straightforward process for initial generation as they produced tracks that were as easy or hard as they needed to be. In the paper they would generate a rectangle with rounded corners and then perform a mutation operation which would modify a control point by numbers drawn from a gaussian distribution.

CONVEX HULL/MIDPOINTS

Instead I decided to use a convex hull methodology, which generates random points and then encloses the points. Doing this produces more interesting starting shapes than a rectangle. Instead of modifying the existing points, I decided to add points in-between the current points. These new points were randomly placed within the bounds of their parent points.

To generate the random points, a randomly sized rectangle is created with a limit on each side (100<Side<300). Then 30 points are placed randomly inside that rectangle, from testing during development I found 30 to be a good number as having a higher number increases in more tracks looking like the bounding rectangle. 30 points allowed for enough variation in track design.

**Minimap/Icon**

[IMAGES] In order to make the minimap and menu icons for the map, a separate scene is used which generates the track with a solid black texture and then renders it to a useable image file. This process had little impact on performance as it is only used when a new track has passed the testing process.

**Track Testing**

**Player Tracking**

Each player can design their own profile with their own username and icon, this is not linked to the internal player profile used for tracking data. That profile is only created once they have read and passed the consent pages. For example, if someone below the age of 18 enters their age they will be able to access the game, but an account will not be made for them. I decided to let users under the participation age to play the base game with no data-tracking, so no player-based tracks, so that they are not encouraged to lie about their age in order to play the game.

Play Count – How many times did they play this track. Tracks played more than once have a higher fitness.

Recency – How long has it been since they last played this track.

Exit – Not Completed, Completed. Tracks that have been completed are factored in more.

**During the Race**

Performance – How did they compare to their other track performances. Average speed for the track. First time sets the baseline.

Problems – Massively increased lap/split times

**Ratings**

When it came to model the player’s behaviour for use in-game, Togelius (2007) used neural network-based controllers to figure out what each player would want from a track. However, I did not think I would have the time or skill in order to deliver a suitable neural network that could perform this function in my game. So, I decided to instead just to ask the user for their requests. This allows the algorithm to produce a track much quicker

Requests (How they are generated, what impact they have)

**Track Evolution (Pseudo-Code)**

**Population**

**‘Mating’ (Restrictions etc.)**

**Children (How many are made)**

I never got to test what the best number of children was for my generator, but in [PAPER] they generated [X] children before starting the selection process.

**Track Selection**

When initially designing the track generator I did not want a track to be able to overlap. To do this I made a testing process each generated track goes through before being accepted. The test involves using vector maths to see if the lines between points intersect, if there was an intersection then the track would be remade or lose a lot of fitness in the algorithm. This worked well for random tracks, but when generating child tracks there was a lot more intersections than normally found in random tracks. I could not figure out how to stop this from happening, so instead I modified the checking process to test the angle of the intersection rather than if there was a crossover. If the angle was too acute, which I found to be >20⁰, then the track would be rejected. This led to more interesting tracks being made by both the random generator and the algorithm.

Output Selection (From Children)

**TESTING / ANALYTICS**

What I would have done. Had I had time

**Future**

How the algorithm would work going forward

Population

**3 Self-Assessment of Learning (0/400)**

What did I get from this project?

New skills

New knowledge

How successful was the project?

**4. References**

**Procedural Generation**

Togelius, J. Yannakakis, G. Stanley, K and Browne, C., 2010. Search-Based Procedural Content Generation. Applications of Evolutionary Computation [online], 1, 141-150

Togelius, J. De Nardi, K. Lucas, S., 2007. Towards automatic personalised content creation for racing games [online]

**Player Behaviour**

**Content Analysis**

**Evolutionary Algorithm**