Emotional Response to Procedurally Generated Textures in a 3D Environment

Harri Renney - 15008005

April 10, 2018

In memory of Syed Zaeem

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The aim of this project is to investigate the Procedural generation of computer graphic textures to invoke emotional response from an individual.
The measured Electrodermal activity (EDA) of an individual will be used as a general indicator for their emotional arousal. Additionally the Electrodermal activity will be used to drive the learning process in the Procedural generation of textures. This will be done using the well known reinforcement learning technique called Q-Learning.
A 3D maze environment will be used to present the generated textures to the user. This is to create further immersion and give more influence to the textures in invoking an emotional response.
To conclude this project, a user study will be conducted to evaluate the effectiveness of the program's ability to increase emotional arousal in an individual.

1.1 Project Application

There are many possible applications of this software in the entertainment industry. The generated textures produced which create an emotional response in users can be taken and used in games or films where it is desirable to increase arousal in the audience. Additionally the entire program's functionality could be used in games that wish to implement the procedural texture generation process.

The purpose of the user study is to indicate the effectiveness of the program. If it is effective in increasing the emotional arousal in users, then it could be used in other applications.

Chapter 2

Background Concepts

In this section all the relevant background information that will need to be covered to fully understand this project will be described in brief but sufficient detail.

2.1 OpenGL

The OpenGL (Open Graphics Language) is an open source, cross platform API (Application Programming Interface) for 3D graphics rendering (Woo et al., 1999). Additionally it can be thought of as a standard or specification upheld by (Khronos, 2018), who are currently the group that manages and updates OpenGL officially.

OpenGL use in programming is made up of a collection of defined methods for interacting with a GPU (Graphics Processing Unit). It was intentionally designed to be hardware and OS (Operating System) independent, meaning it can be used across multiple platforms. However this streamlined design means it is solely a graphics rendering library and does not provide support for other related functions, like windowing and inputs. These must be acquired from other API's or programmed separately. The popular OpenGL APIs that provide this support are covered in appendix A.2.

2.1.1 Rendering Pipeline

OpenGL consists of a series of major operations it follows known as the "OpenGL Rendering Pipeline". This process is followed to convert data from the program into a final render image (Khronos, 2017).
The basic steps for this are outlined below:

1. **Vertex Processing**: Processes vertex data provided from application to a position in three dimensional space. (Vertex Shader)
2. **Primitive Assembly**: Collects a few single primitives into a sequence of primitives. Like lines, triangles etc.
3. **Rasterization**: Visible newly assembled primitives are divided into fragments. Fragments are little sections of the scene that are used to compute final pixel data.
4. **Fragment processing**: Each fragment is given data for producing the pixels colour within the fragment. (Fragment Shader)
5. **Frame Buffer**: All the information provided to render images on the screen loaded into frame buffer ready to load.

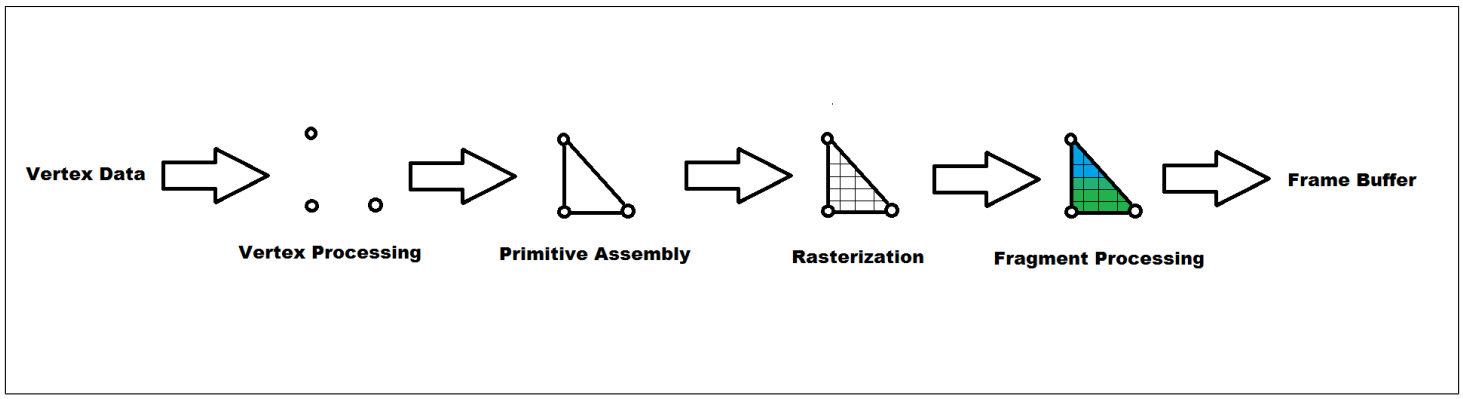

Figure 2.1
Graphics Pipeline

2.1.2 GLSL

GLSL (OpenGL Shader Language) is a shading language with C-like syntax for GPU programming (Rost et al., 2009). The application that uses OpenGL will compile the specified files into a shader program which is then loaded onto the GPU to process data as part of the rendering pipeline. How visuals appear in the final rendered image is decided in the shaders as they process the data received to the GPU.

There are two main shaders that are required to be explicitly defined and loaded onto the GPU in order for OpenGL to operate. They are the Vertex Shader, which is the part that receives the vertices and processes them. Then there is the Fragment Shader, which calculates the colour of the pixels in the space between vertices.

2.1.3 Relevance

OpenGL will be used in this project as it provides great low-level control and access to the GPU. This level of control is needed to implement procedural generation of textures.
Furthermore the supported C API provided by Khronos fits the C program stack that the rest of the system uses.

2.2 Electrodermal Activity

Electrodermal Activity (EDA) is the common term used for all electrical phenomena in skin, including all active and passive properties traced back to the skin.
In simple terms it is the degree of sweat observed in an individual. Sweating is a physiological reaction related to the autonomic nervous system. For this reason, EDA has for a long time been most frequently used as an indicator of arousal in psychophysiological research (Duffy et al., 1972).

The outdated term used for this same phenomena in skin is Galvanic Skin Response (GSR).
It is largely agreed upon that this term was not completely appropriate for a number of reasons:

1. It suggests skin is regarded as a galvanic element, which it is not.
2. It suggests GSR as always being provoked as some kind of a reflex.
3. Galvanic Skin Response was being used to cover not only EDA but all Electrodermal phenomena in general which was ambiguous.

In this report the more up to date term Electrodermal Activity will be used but the term galvanic skin response that might be used in other documents is likely referring to the same thing.

2.2.1 Electrodermal Activity Recording Methods

EDA recording is possible with relatively simple equipment, resulting in a variety of methodologies.
There have been attempts at standardizing techniques of Electrodermal recording as Fowles (Fowles et al., 1981) explains, but without any official standard agreed upon so far.
The most common used method is known as exosomatic DC recording. This is where a small direct DC voltage is applied across the skin using two electrodes and the current is kept constant. This means the resistance of the skin can be measured using Ohm's law $R = \frac{V}{I}$. The resistance can then be used to observe the EDA in the user (Boucsein, 2012).

2.2.2 Electrodermal Activity Signal

The EDA signal is the entire resistance measurement taken over a period of time.
There are two components in the resulting EDA signal (iMotions, 2016):

Skin Conductance Level (SCL) is the slow variation and change in the signal within tens of seconds to minutes. SCL is constantly changing within an individual and is dependant on things like hydration, skin dryness, autonomic regulation. SCL also differs naturally between individuals, and isn't informative for emotional observation as what causes the change isn't related to emotional response.

Skin Conductance Response (SCR) is the faster alterations seen in the signal as obvious EDA bursts/peaks. It is sensitive to specific emotional arousing stimulus events. These bursts occur between 1-5 seconds after the occurrence of the emotional stimuli.
To identify emotional responses accurately it is important to only be observing changes in the SCR specifically. To do this still requires identifying the SCL in the Electrodermal signal as well.

2.2.3 Relevance

EDA will be used in this project as it is a reliable indicator of any emotional responses in an individual. The physiological reaction it measures quickly takes place after the external stimuli is experienced. This means the stimuli causing the emotional response can be detected quickly for the program to act on.

Measuring EDA using the DC exosomatic method requires a simple bit of hardware and is a relatively straightforward process. The measured resistance values can represent an EDA signal which can be used to detect emotional responses. This can then be incorporated into the main program to drive the texture learning process.

2.3 Reinforcement Learning

Reinforcement learning (RL) is a machine learning methodology which establishes a mapping of situations/states to actions with the aim of reaching the goal optimally.
It does this by representing the system as an actor/agent which navigates through the states in an environment based on the actions it takes. The agent learns the problem by trial and error, updating its knowledge of the problem overtime as it makes right or wrong actions.

2.3.1 Markov Decision Process

To understand RL, the type of problem it is used to solve should be described first.
The problem type is known as a Markov Decision Process (MDP). This type of problem which is made up of states and actions. The states provide sufficient information to uniquely identify the current situation. This means knowledge of past states or actions are not needed to solve it, just the current one.

Abhijit Gosavi (Gosavi, 2011) covers the framework of the MDP as having the following elements:

1. **State of the system**: The set of parameters or information that describes the system. States will either have a discrete set or be continuous. Continuous states require some form of generalisation to represent.
2. **Actions**: Processes that transitions the system from its current state to a new one. Problems will either have a discrete set of actions it can take from each step or continuous actions that must be generalised.
3. **Transition probabilities**: Some environments are uncertain and actions in them unpredictable. In these cases there is probability involved to transition from current state i to another particular state j as a result of an action a.
4. **Immediate rewards**: The reward signal received by the system for transitioning from one state to another. it indicates the progression towards the goal state. This might be known before transitioning or only after the transition is made. Some problems will have immediate rewards in most/all transitions and some will have sparse reward, maybe even only present on the goal state!
5. **Policy**: The behaviour of the system. It defines how the system decides the next action to take in every state. The goal of the policy is to maximize reward in the long run.

There are a number of different techniques able to solve a MDP problem. RL is one of them.

2.3.2 Q-Learning

Q-Learning is an off-policy RL technique. It can build up a mapping of good actions through an environment without requiring a model for that environment. Further, Q-Learning can handle stochastic actions and rewards without needing to adapt.

The Q-value is the estimated reward a state is given by the Q-Learning algorithm. The algorithm draws on knowledge stored from past actions and states visited.
Using the Q-value the agent can estimate how good each action is. The agent's policy can use these estimates to decide on which action to take.

The process of updating the agents knowledge in discrete Q-Learning uses the following equation after visiting each new state:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \lambda\binom{max}{a}Q(s_t, a) - Q(s_t, a_t)]$$

The Q Function takes into account the following elements:

1. **Learning Rate** α: Determines the extent the newly acquired information influences old information. A value in range [0.0-1.0] where 0.0 would have no learning and 1.0 would consider new information completely.
2. **Discount Rate** λ: Determines the influence of future estimated reward on the current information stored. A value in range [0.0-1.0] where 0.0 would be short sighted only considering current reward and 1.0 would be long sighted.
3. $Q(s_t, a_t) \leftarrow Q(s_t, a_t)$: Update the current Q value with the current Q value plus the new information gathered.
4. r_{t+1}: The reward earned for transitioning to the next state.
5. $\binom{max}{a}Q(s_t, a)$: The max Q value estimated to be possible to achieve from next state.
6. $-q(s_t, a_t)$: The existing Q value subtracted from the reward and estimated max q value to find difference.

Problems which have discrete spaces that are not too large can have the Q-values for each state-action pair stored in memory. This becomes a problem when storing every possible state-action Q-value estimations. If a lookup table is used to store state-action pairs, the space needed grows at an unreasonable rate. This is a common problem and is known as the "curse of dimensionality" (Bellman, 2013). This makes many real Markov decision problems impractical to solve this way.

2.3.3 Value Function Approximation

In the real world, many Markov decision problems are not made up of a reasonable number of discrete states and actions. When the possible states and actions exceeds a reasonable number or is continuously precise, some form of generalisation needs to be incorporated. A solution to this is to use value function approximation.

Value function approximation (Sutton and Barto, 1998)[P.161] can be used as a parametrised function to calculate the Q-value instead of looking up the value from a stored table. The parametrised function takes values from features of the current state and works out the Q-value estimation from them. This is a form of generalisation.
Instead of storing Q-values, the learning takes place in updating a set of weights which influence the parametrised function. The goal is to learn the set of weights which supports the best state features.

There are many different function approximators that can be used to learn the weights:

- Linear Combination of features
- Neural Network
- Decision Tree
- Fourier/Wavelet bases

The simplest function approximation would be linear, essentially a weighted sum of all states:

$$Q(s) = W_0 + W_1 * f_1(s) + ... + W_n * f_n(s)$$

Linear approximation is guaranteed to converge on a local optimum. So if there is only one optimum in the search space it works well. If there are many optimum across the search space and a single global optimum, it isn't likely to find it.
In problems where there is only one optimum or finding the global isn't a necessity this works fine. If it is important to find the global then a non-linear method might be required.
An example of this would be an Artificial Neural network (ANN) which can discover a global optimum in a more complex problem space. (Sutton and Barto, 1998) [P.167-186]

2.3.4 Relevance

A considerable part of this project is in seeking to discover the unknown, theoretical values for the procedural texture generation process to construct emotionally arousing textures.
The quality function for this situation is not known. However there is a quality signal indicated by emotional responses detected by changes in the EDA.

The RL technique Q-Learning is elected for discovering the set of values for the procedural generation process. RL is chosen over other algorithms for the MDP problem as the reward in this situation is unknown until the action is taken. Bellman's value iteration equation (Bellman, 1957) for example cannot cope without knowing the rewards before hand.
As part of the Q-Learning agent, value function approximation is used to generalise the problem and overcome the near endless number of texture states.

2.4 Evolutionary Strategy

The Evolutionary Strategies (ES) are a sub-group of nature inspired search methods called Evolutionary Algorithms, of which the renown Genetic Algorithm belongs to (Rechenberg, 1973).
It works by following inspiration from parts of the theory of evolution. By making small variations to a set of solutions and keeping the best, progressively better solutions for the problem can be found.

2.4.1 General Outline

The basic abstracted idea of the ES is as follows.
To begin with, an initial population of parents is formed. Here a parent represents a possible solution to the problem.
Each parent generates a set of children to represent the offspring population. The process of generating children includes using evolutionary operators acting on copies of the parent, these new children make up the parents offspring. The evolutionary operators introduce slight variation to the children from the original parent.
Each child is then evaluated for its quality in solving the problem. The best from the offspring and optionally the parent too (depending on the selection operator used) is selected to be the parent in the next generation. This is repeated for each original parent until a new population of parents is formed.
This process is repeated with the idea that by using the evolutionary operators in each iteration to explore solutions, progressively better solutions in the parents can be found.

2.4.2 Evolutionary Operators

The evolutionary operators are the defined methods to be used on solutions to achieve the evolution inspired process.
The two basic selection operators first covered in Rechenberg's original paper (Rechenberg, 1973) are plus and comma selection:

Plus Selection $(\mu + \lambda)$ = The selection pool is made up of the newly generated solutions λ and the original parent μ. This means a better parent will be chosen over its children still. (Exploitative)
Comma Selection (μ, λ) = The selection pool is only the newly generated children λ of the parent μ. This means even if the parent is better, it is not considered for choice in next generation. (Explorative)

The other two genetic operators are universal for most evolutionary algorithms (Mitchell, 1998):

Mutation Operator = In Evolutionary Strategies this is a basic variation operator. It introduces the majority of genetic variation in children solutions. How this operator is specifically defined is still entirely problem-dependant in regard to the solution representation.
Recombination Operator = In Evolutionary Strategies this is where information from two or more parents is combined to form the new child. This would be instead of just taking a exact copy of one parent.

2.4.3 Relevance

The ES algorithm is used in the program as part of the Q-Learning agent's policy (Moriarty et al., 1999). It is used to search the huge problem space of potential future actions the Q-Learning agent can take. An ES explores close to the existing solution and adds other advantages, like self-adaptive mutation. This makes it more effective than other methods, including just randomly generating completely new actions.
The ES was chosen over the genetic algorithm as the solutions in the ES are encoded as a vector of real numbers. This matches the vector of floats required for generating the textures in the procedural process. Additionally with the right parameters it can be run with little impact on performance, which is important to maintain in this program for immersion.

2.5 Procedural Pattern/Texture Generation

The term procedural in computing can be thought of as a label for data that is produced through program code rather than read from a data structure.
This means a procedural texture is purely synthetic. It is generated from a program or model rather than read from a stored image in memory (Perlin, 1985). Procedural texture generation applies this method of algorithmically generating data for use in textures rather than reading one stored in memory. It is only concerned with how to present an image, typically with an RGB value for each pixel.

An RGB value is a collection of three different values which specify the amount of Red, Green and Blue combined into the resulting colour (Tkalcic and Tasic, 2003). Each value must be within a common range. Often used is [0-255], or [0.0-1.0] for more continuous precision. This format can be extended to RGBA which includes the original colours and an alpha channel,

which decides the degree of transparency to the colour.
There are other forms of representing colours and pixels but RGB is by far the most popular used.

There are two methods for procedurally generating textures (Ebert, 2003)[PAGE]:

Explicit methods: Where values are generated directly that make up textures. This means to generate textures in some fixed order previously then storing it in a conventional way to be used later. In procedural textures this would be done by generating the image in the program/application on the CPU, then storing it in an image buffer to be loaded later on the GPU.

Implicit methods: Where values are generated in reply to a query for a particular point, this means it evaluates each point on request. In procedural textures this would be done by generating values as each pixel is considered on the Fragment Shader.

Both of these methods generally work for most applications, but some are naturally better suited for certain situations.

In procedural textures, the image is defined by taking the texture or world coordinates as an input into some function $f(x)$. The function then processes the points in a particular way to output a colour for that position. (Ebert, 2003)
A simple example would be for generating a checkerboard. A grid of squares is calculated, then using the point's coordinates outputs the correct colour. Either a white or black pixel depending on which square it is calculated to be in. This is done for each pixel until a whole image of a black and white checkerboard squares is formed.

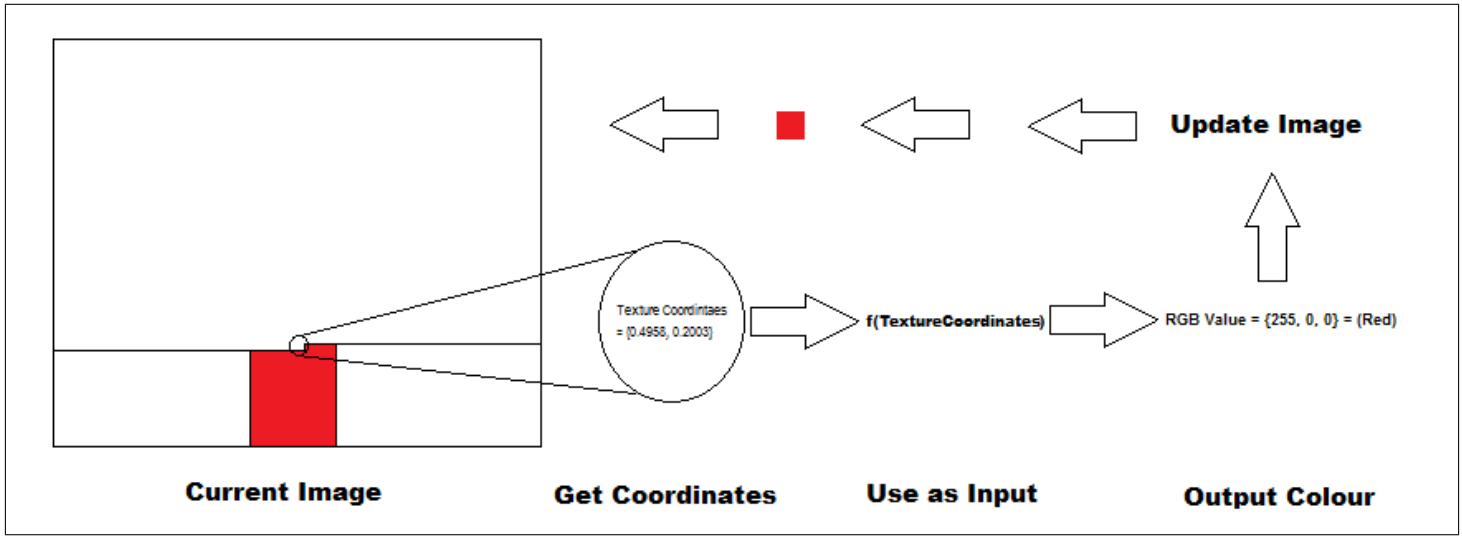

Figure 2.2
Procedural pattern generation calculates colour from texture coordinates

Within texture generation there are two areas to Procedural generation:

Pattern generation is where the texture pattern is defined. So the initial values of surfaces before any further processing is done.

Shading mode is where the program receives the initial pattern and processes it to simulate the behaviour of that surface in respect to lighting and reflection etc. A shading model can process any pattern it receives, it could be a procedural one just generated or a non-procedural stored image.

Complex pattern generation (Ebert, 2003)[PAGE] is based around the idea of building up small/simple patterns one into the next to form complex patterns. A common way is known as layering, where one texture placed on top of another to form final texture.
Function composition is a fundamental part of computer program in general so is no surprise to be found useful in pattern

generation either.

2.5.1 Noise

To generate convincing irregular procedural textures, an irregular primitive function is required. This is called Noise. (Ebert, 2003)[P67-89]
The Noise function is pseudo random is used to break up the regularity in patterns.
It needs to be pseudo random as a truly random function would change between frames, meaning the texture would not remain constant. A pseudo random function will produce the same "random" output with the same inputs. Therefore it can be controlled by using consistent inputs between frames.

As defined by the pioneers in procedural texturing and modelling, (Ebert, 2003)[P the Ideal Noise function properties:

- Noise is a repeatable Pseudorandom function of its inputs.
- Noise has a known range, between -1.0 to 1.0.
- Noise is band-limited, with a maximum frequency of about 1.
- Noise doesn't exhibit obvious periodic/regular patterns. Pseudorandom functions will always be periodic, but the period can be very large to satisfy this.
- The two other properties are not relevant to raise here.

A Noise function fed inputs of n dimensions will output a value for $n+1$ dimensions. So a 2D Noise function input with x and y coordinates will output a 3D value, perfect for creating texture of a material. By extending the Noise into an additional dimension from 2D to 3D the extra dimension can be used as time to animate a texture. This could apply to an ocean wave scene. (Perlin, 2002)
Lattice noises are the most popular implementations of Noise for procedural texture generation. If done correctly it satisfies the ideal Noise function requirements. The core to how Lattice Noise works is in a set of random values stored inside a "lattice". The lattice of stored static values are used for generating pseudorandom values.
The original idea for lattice Noise was thought of by (Perlin, 1985).

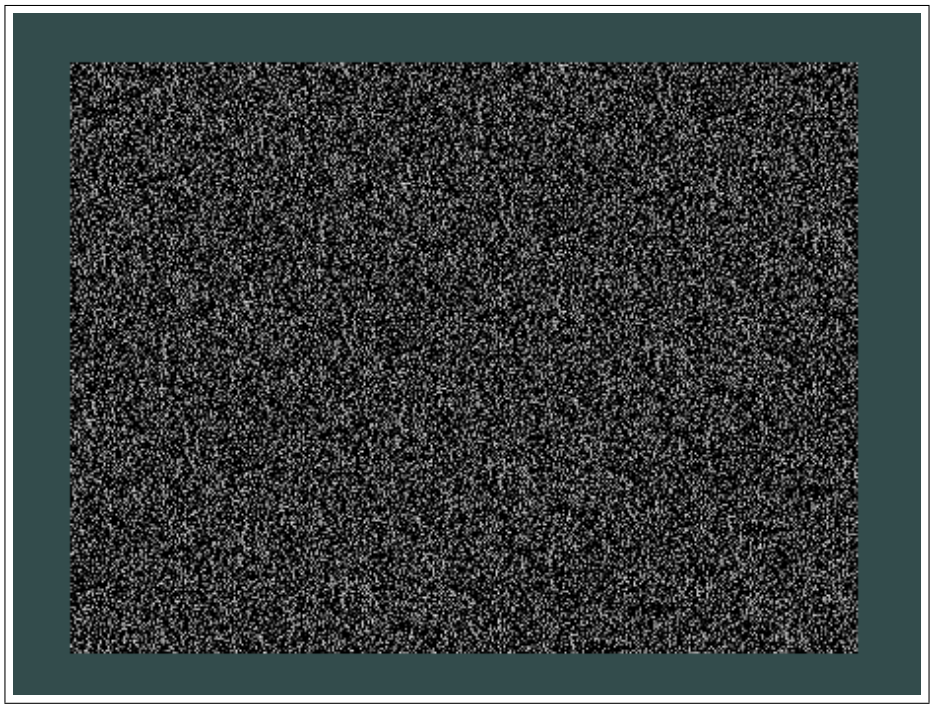

Figure 2.3
Lattice Noise

In lattice Noise, maxima and minima will occur at regularly spaced intervals. The recurrence of these maxima and minima can be noticeable to a user if no techniques are employed to avoid it. The subtle recurring pattern can be observed in figure 2.3.
Lattice Noise on its own will not fulfil the required property "Noise doesn't exhibit obvious regular patterns", however there are ways to avoid this. This is typically done by extending the Noise function to what's called gradient Noise, which is described by Ken Perlin in the same journal.

2.5.2 Turbulence

Turbulence works by making several calls to a Noise function and combining them together to make a stochastic spectral function with a fractal power spectrum, which is used for creating irregular patterns. (Ebert, 2003)[PAGERANGE]
The irregular, smoother randomness generated through turbulence can then be used as part of the input to any pattern generation function that allows a Noise input as a parameter.

Figure 2.4
Turbulence: Ten calls to Lattice noise combined

2.5.3 Relevance

The ultimate aim of the project is to generate new textures to progressively increase emotional arousal in the user. Therefore the Procedural generation of textures is fundamental to realising this.
The layering technique will be used to combine simple generated patterns to form complex textures. Additionally, irregularity will be introduced to the process using Noise functions to further the complexity in the textures.
With enough complexity in the texture generation process, a large range of patterns can be produced for the final texture.

Chapter 3

Related Work

What follows are descriptions of some of the existing projects in a similar domain and assesses their similarity to this project.

3.1 Stress makes art: Galvanic skin response and visual generation

The project goal for (jocelynzada, 2013) was to use EDA measurements to generate graphic visuals.
The EDA measurements were taken from the user through a prototyped Arduino device. The EDA measurements were then read from the device over the serial port and used as inputs for a graphics algorithm. The algorithm was written in python and generated graphical visuals in a windowed environment.
The Arduino device used a Exosomatic DC setup. This means a circuit with a constant current and variable voltage was used to read the skins resistance. This method of recording EDA is basic, however it proves sufficiently reliable in this project. It is not a safety critical task or one that requires a high degree of precision so works fine in this case.

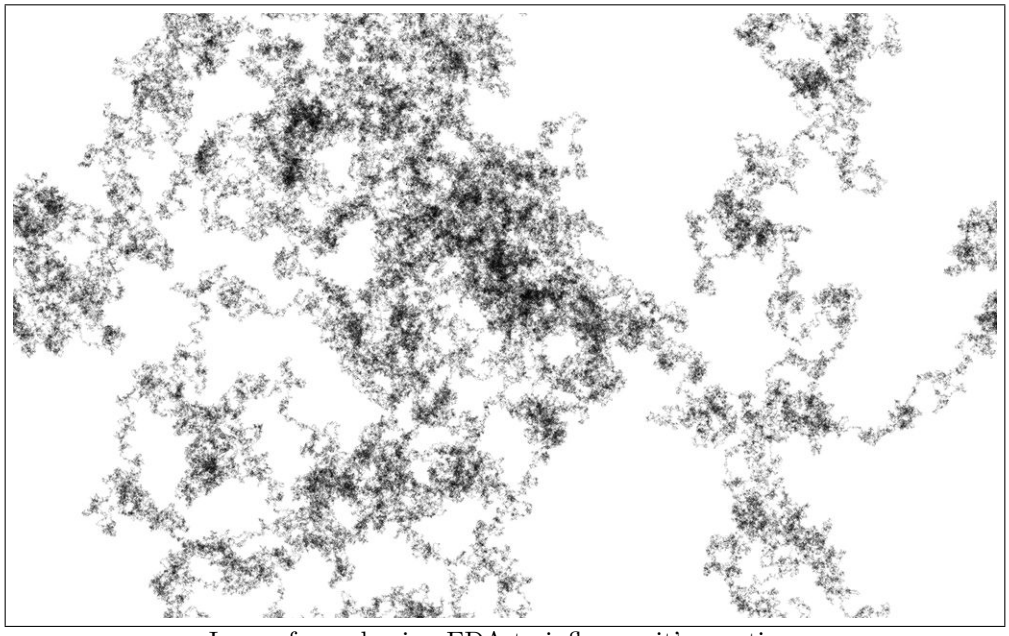

Image formed using EDA to influence it's creation.

3.2 Development of biofeedback mechanisms in a procedural environment using biometric sensors

In this project (Torres, 2013) a number of different methods of biofeedback recording were utilised in video games with the goal of creating further human-computer interaction.
A framework was developed capable of using a variety of different biofeedback models which can be drawn from for use in any accommodating program. The developed frame work was designed to be independent, therefore capable of being applied in other games with ease. In the study, a specifically tailored game environment was developed in order to test/assess the effectiveness of the various features of the biofeedback framework.
The collected biofeedback in this game environment was used to effect the game environment and physics. For instance, how fast the user was moving through the world. If an increase in physiological arousal was detected, the speed of movement would increase. In contrast if the user was detected to be in a relaxed state, they moved slower. The only visuals/graphics effected by the biofeedback was a case where bugs would appear on the surrounding walls if the user's observed stress level past some threshold. This was the only procedural graphics feature implemented in the game environment. Likely because the goal was in developing the biofeedback framework, therefore only this simply example feature was implemented.

Chapter 4

Requirements

In this chapter the the overall general requirements of the program will be identified. Each will then be explained in detail and the broken up into clear requirement cases. In this report's development sections, the requirement cases will be referred to when being satisfied.

Requirements

Requirement	Name
1	Explorable 3D environment
2	Electrodermal Activity Recording
3	Reinforcement Q-Learning to influence procedural texture generation
4	Evolutionary Strategy for the Q-Learning policy
5	Procedural texture generation with modifiable features
6	User study to evaluate program effectiveness

4.1 Requirement 1: Explorable 3D environment

An explorable 3D maze environment will be developed to present the procedurally generated textures to the user. The textures will be applied to the surfaces of the maze walls, floor and ceiling. With the user confined to this textured maze, a greater emotional reaction should be experienced by the user in response to the textures.

Other ways of presenting the textures were considered. An example of a simpler method would be to present the textures on a static object. However it is uncertain if this would be sufficiently engaging for the user to experience any emotional response.
Ideally it would be best to create a more complex 3D environment than a maze to immerse the user as much as possible in the textures. Spending too much time working on this would likely detract from the focus of the rest of the project.

The shader program must be compiled from any source code provided. This is so the Procedural generation Fragment Shader can be used in the shader program.
The functionality of a first-person virtual camera system will be needed for realistic and engaging navigation through the maze (Haigh-Hutchinson, 2009). The camera must be restricted to remain within the maze by using collision detection.

Requirement Cases

Case 1 (C1.3D): Compiles the shader program from written source code.
Case 2 (C2.3D): Provides a first-person virtual camera system.
Case 3 (C3.3D): Provides functions for quickly forming custom maze layouts.
Case 4 (C4.3D): Supports collision detection between the camera and the maze walls.

4.2 Requirement 2: Electrodermal Activity Recording

The EDA of the user will be used to indicate if any emotional response is incited by the textures.
A device capable of measuring EDA will need to be developed. This will require embedded software to be written in order for EDA measurements to be taken and sent to the main program. Additionally the main program will need to be able to interact with the EDA device to receive measurements.

Emotional responses to external stimuli make clear changes in the EDA of the individual, therefore a high level of precision in the measurements will not be required.

The target platform for the program will be on the Windows OS. The program will need to be able to establish a serial connection. Then process the values it reads from the port to produce the human resistance used to observe changes in the user's EDA.

Requirement Cases

Use Case 5 (UC5.EDA): Supports the Windows OS.
Use Case 6 (UC6.EDA): Establishes a serial connection to a system port
Use Case 7 (UC7.EDA): Read measurements from the connected port.
Use Case 8 (UC8.EDA): Processes measurements to produce the correct human resistance values.

4.3 Requirement 3: Reinforcement Q-Learning to influence procedural texture generation

The process of producing new feature sets for texture generation will be done using a RL technique. Using this, it might be possible to learn texture features which increase emotional arousal in the user. The EDA will be used as the observable reward signal to drive the learning process.

The renown Q-Learning algorithm will be used to discover the texture features. The number of possible texture images is considered endless, therefore it must be generalised somehow. Linear value approximation provides generalisation and Q-Learning can naturally be extended to incorporate this.
The number of possible actions that can be taken in each state is equally as large. Therefore an effective way of searching it must be employed. The ES will be used as part of the policy to search the huge number of possible actions.

Requirement Cases

Use Case 9 (UC9.RL): Provides control over the Q-Learning agents parameters for tuning.
Use Case 10 (UC10.RL): Uses Linear value approximation for generalisation.
Use Case 11 (UC11.RL): Uses an ES in the Q-Learning policy.

4.4 Requirement 4: Evolutionary Strategy for the Q-Learning policy

The Q-Learning agent's state is represented by real number values and therefore is considered a continuous search space. For this reason an algorithm that can effectively search a continuous problem space is needed.
The ES algorithm is capable of search with reliable results, provided it can estimate the quality of the actions accurately. The Q-Learning agent provides the estimation for the quality of the actions for the ES.

The ES class will be made abstract. This is so it can be inherited by the Q-Learning class for use in exploring the possible actions as part of its policy (Moriarty et al., 1999). The fitness function for the ES will be made as a pure virtual function. This is so it can be defined for the specific problem by the inheriting class.

Requirement Cases

Use Case 12 (UC12.ES): Provides control over the ES's parameters.
Use Case 13 (UC13.ES): Developed as an abstract class.
Use Case 14 (UC14.ES): Fitness function is programmed as a pure virtual function.

4.5 Requirement 5: Procedural texture generation with modifiable features

A method for generating textures in real-time needs to be developed.
Procedural textures are generated using the texture coordinates as inputs to generate the pattern. Then Irregularity is introduced to the pattern using Pseudorandom Noise functions.
This process should be controllable through a collection of modifiable texture features. A feature could be the presence of a certain pattern, or the degree of the primary colour red in the final texture. By progressively changing the features, new complex textures can be generated.

The generation process needs to be optimized, it shouldn't effect the performance of the program enough to effect the user interaction. A smooth experience is required to maintain the level of immersion created by the maze environment.

Requirement Cases

Use Case 15 (UC15.PTG): Uses the implicit method for procedural generation. Therefore must be programmed in the Fragment Shader on the GPU.
Use Case 16 (UC16.PTG): Provides a method for loading variables from the main program on the CPU to the Fragment Shader on the GPU.
Use Case 17 (UC17.PTG): Collection of functions for generating basic patterns from the input texture coordinates.
Use Case 18 (UC18.PTG): Support for introducing irregularity into the generation process of some of the basic patterns using gradient Noise.
Use Case 19 (UC19.PTG): Method for layering multiple patterns together to form complex pattern.
Use Case 20 (UC20.PTG): Set of modifiable features to influence presence of certain features in the final texture.

4.6 Requirement 6: User study to evaluate program effectiveness

Although is is empirically accepted that visuals have an effect on an individual's emotional response (Pitchforth, 2010), whether visuals can be generated to progressively increase emotional arousal is questionable.
A brief investigation into the effectiveness of this program will be made as part of a user study.
In the user study two sets of sample data will be gathered from a population of users. The first sample set of a natural period, the second during a period interacting with the program. A student t-test will then be made on the results. If there is a statistically significant difference between the two periods, the program's effectiveness in reaching its goal will be supported.
A second program will be developed for observing the EDA measurements in real-time during the measurement periods. This will be used for recognising any problems that arise during measurement periods.

Originally the project was focused on the emotional response of stress. However, identifying specific emotions using EDA is a highly debated and controversial topic as described in (Duffy et al., 1972). For this reason the project aim was revised to be more general and consider all emotional responses.

Requirement Cases

Use Case 21 (UC21.US): A set of mean EDA samples taken from a population during a natural baseline period.
Use Case 22 (UC22.US): A set of mean EDA samples to be taken from a population during a period interacting with the program.
Use Case 23 (UC23.US): An EDA recording program developed for observing changes in EDA in real-time.
Use Case 24 (UC24.US): Conduct a Student T-Test on the two sample sets to show evidence for any statistically significant difference.

Chapter 5

Design

In this chapter the entire program is broken down into the essential major modules needed to achieve the goal of the project. Each module's contents will be summarised and the relations between them defined to form the entire program.

Some of the third party APIs like OpenGL take a purely C style functional programming structure. Modules designed to work closely with them will also follow this structure. For this reason the program is not considered purely object-oriented. UML is typically used in designing purely object oriented systems. However it still supports use for functional-based modelling. This is described in detail by (Douglass, 2009). The UML digram will use the stereotype <<File>> for C style modules and <<Class>> for classes.

The diagram 5.1 and upcoming descriptions of the modules only include the important functions and variables. Assume basic functionality like getters and setters in the classes.

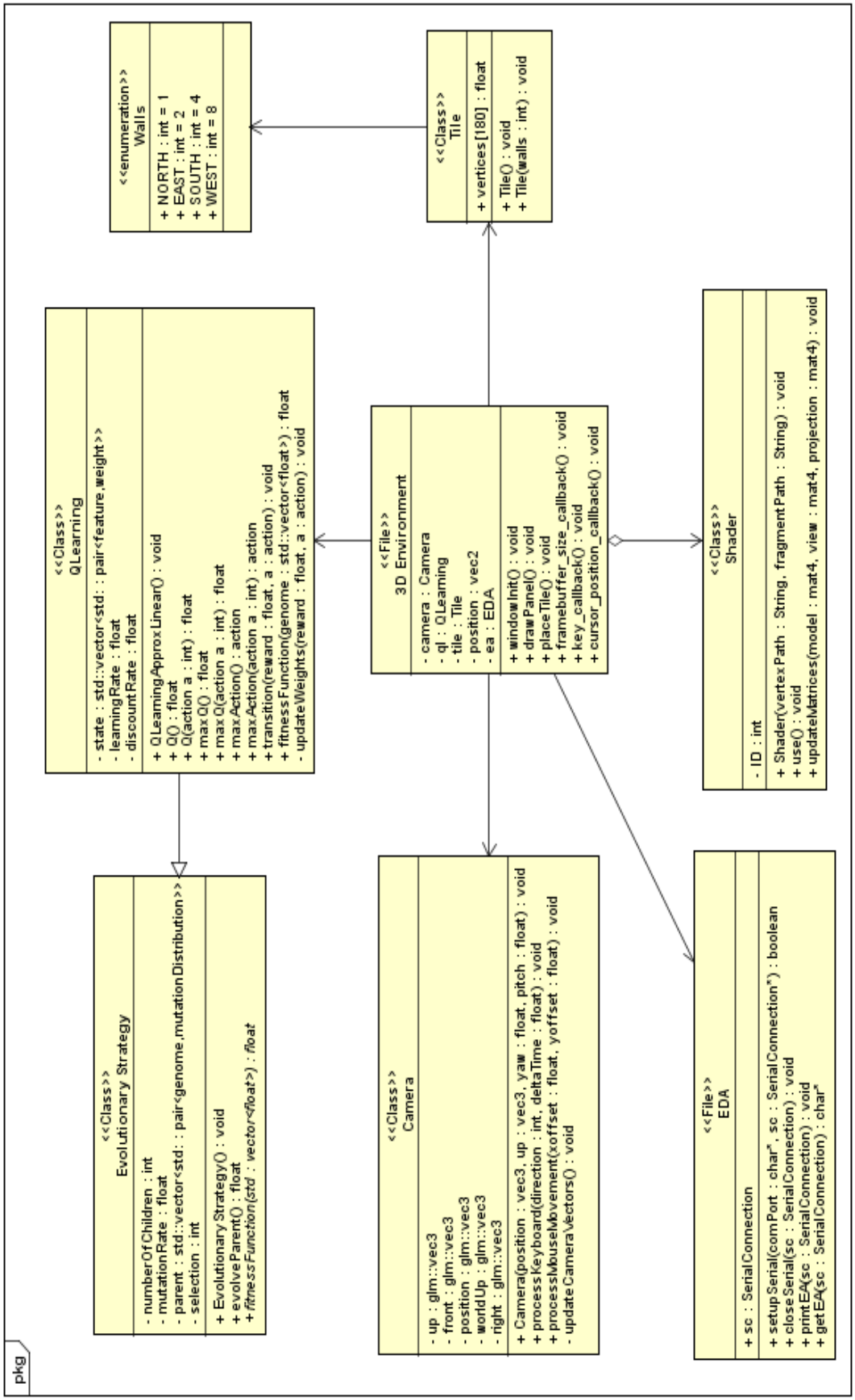

Figure 5.1
UML File Diagram of the whole program to illustrate modules and their relationships.

5.1 3D Environment

This module establishes the main flow of the whole program. The majority of the code to create a window and render a 3D environment to it is written here.

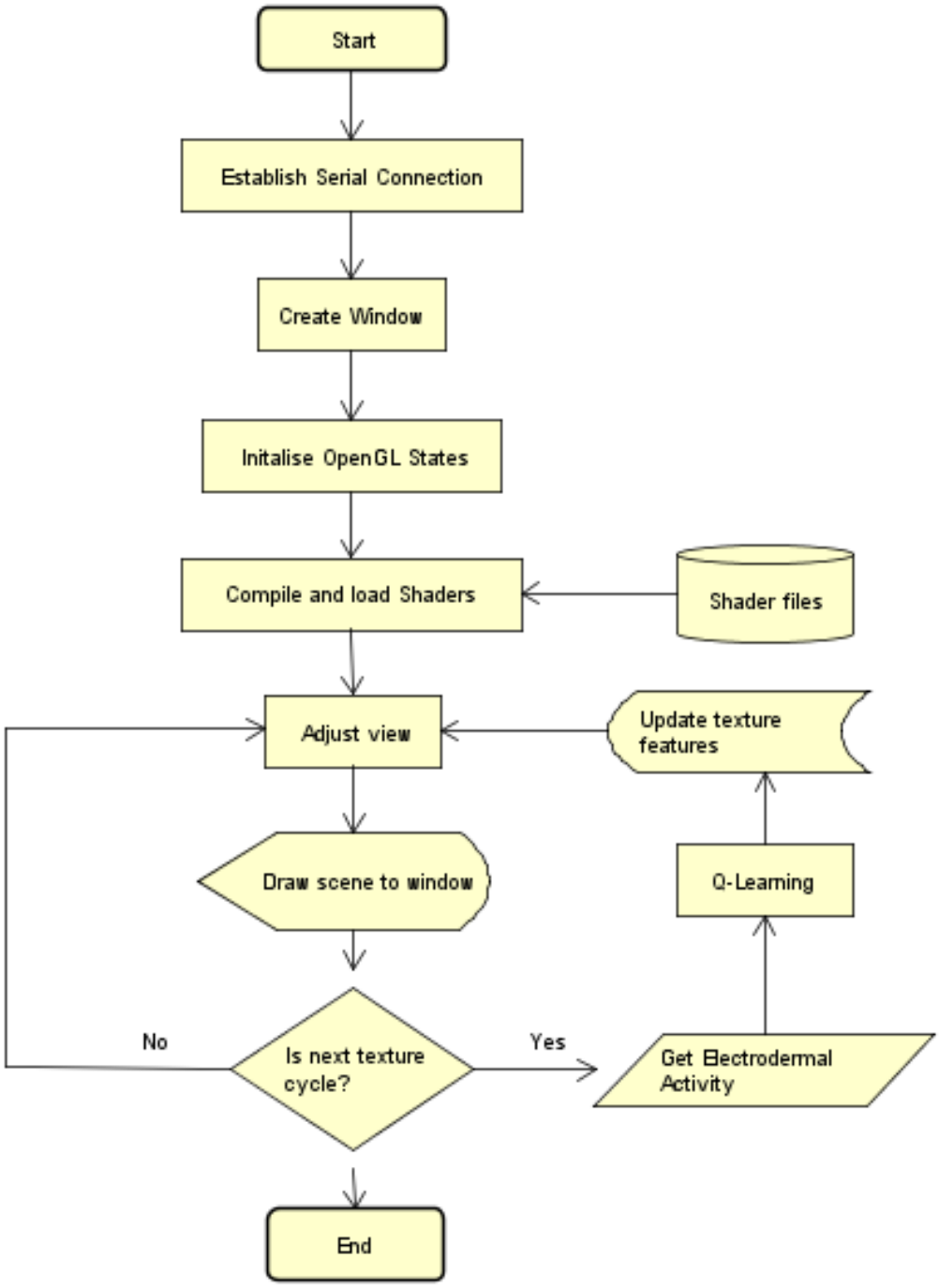

Figure 5.2
Flow diagram showing abstract step-by-step process through the main 3D Environment program

Variables

Camera camera
Decides the scene rendered to window. Uses current position to calculate the view matrix which decides where objects move in relation to camera.

QLearning ql
Used to intelligently generate a new set of changes to be made to the current set of procedural texture features.

Tile tile
Used to construct a specific tile to next load onto GPU to render to the scene.

vec2 position
Used to keep track of the users position in the 3D world. The main purpose for this is for collision detection with walls. However has future potential for other uses.

Functions

void windowInit()
Prepares a window and context for user interaction and rendering of scene. Using mainly GLFW API calls.

void drawPanel(Tile tile, unsigned int texture, int vertice)
Draws a panel of a tile to the scene.

void placeTile(Tile tile, glm::vec3 translation, int numberTiles, glm::mat4 modelMatrix, int VBO, Shader shaderProgram, unsigned int texture)*
Draws a tile to the scene in a place specified within the 3D world.

void framebuffer_size_callback(GLFWwindow window, int width, int height)*
This callback function is executed on detection of a resize window event. In the function is code to handle anything affected by the window resize.

void key_callback(GLFWwindow window, int key, int scancode, int action, int mods)*
This callback function is executed on detection of a keyboard input event. In the function is a switch case with code for handling specific key inputs.

void cursor_position_callback(GLFWwindow window, double xpos, double ypos)*
This callback function is executed on detection of cursor position change. In the function is code to change the camera position depending on cursor position.

5.2 Reinforcement Learning

This module is used to create a RL agent. Using the Q-Learning algorithm with linear approximation to learn.
The ES is inherited to explore actions in the agent's policy.

Algorithm 1 Q-Learning Value Approximation

1: Initialise Weights $w = w_0, w_1, ..., w_n$
2: **for** each episode/run **do**
3: $s \leftarrow$ Initial State
4: $a \leftarrow$ Action decided by policy (ES)
5: Take action a, observe reward r and next state is s'
6: Update weights w
7: $w \leftarrow w + \alpha(r + \lambda \binom{max}{a'} Q(s', a') - Q(s, a)) \overleftarrow{\nabla}_w Q(s, a)$
8: $s \leftarrow s'$
9: **end for**

Variables

std::vector<std::pair<feature, weight>> state
Defines the current state of the agent. In value approximation a state is defined by a collection of features in some range and weights which decide their importance.

float learningRate
The learning rate is a real number in range 0.0 - 1.0 which decides the influence new information has on the agents weightings.

discountRate
The discount rate is a real number in range 0.0 - 1.0 which decides the influence the estimated future Q values for the next possible states has on the the agents weightings.

Functions

void QLearningAppoxLinear()
Constructs a Q-Learning agent.

float Q()
Gets the Q value estimate of the current state using a weighted sum of features.

float Q(action a)
Gets the Q value estimate of the state the action a takes the agent to.

float maxQ()
Gets the Q value of the estimated best state from current state.

float maxQ(action a)
Gets the Q vaue of the estimated best state from the state the action a takes the agent to.

action maxAction()
Gets the estimated best action from current state.

action maxAction(action a)
Gets the estimated best action from the state the action a takes the agent to.

void transition(float reward, std::vector<float> action)
Transitions the agent from the current state to the state action a takes the agent to. The immediate reward from this next state must be passed.

float fitnessFunction(std::vector<float> genome) override
This class inherits a class for deciding the next action to take. This defines it's policy/behaviour. In this project the ES is used to search for the next action, then using the Q-Learning value estimate for deciding the best one to pick from.

void updateWeights(float reward, action a)
Calculates the changes to the weights using the immediate reward from the state and estimated future values then updates the agent's weights accordingly.

5.3 Evolutionary Strategy

An abstract class used to employ the basic ES algorithm in the class that inherits it. The fitness function must be defined by the inheriting class to operate.

Algorithm 2 Evolutionary Strategy with Comma Selection

```
1: p ← Parent
2: b ← best child
3: for each child do
4:     c ← Current child
5:     for each gene of c do
6:         g ← get corresponding gene from p
7:         g ← Mutate(g)
8:     end for
9:     if fitness(c) > fitness(b) then
10:        b ← c
11:    end if
12: end for
13: p ← b
```

Variables

int numberOfChildren
Defines the number of children each new generation from a parent consists of. A higher number of children allows more diversity but requires longer to compute.

float mutationRate
A value in the range of 0.0 - 1.0 which decides the chance for each gene if mutation occurs.

int selection
Decides the type of selection operator used to decide the pool of solutions to pick the next parent from. 0 sets to comma selection and 1 for plus selection.

std::vector<std::pair<genome, mutationDistribution>> parent
The current parent, typically the best solution from last generation that survived. Inheriting class gets the parent's genome when it needs the current best solution.

Functions

EvolutionaryStrategy(int numberOfGenes, int numberOfChildren, float mutationRate, float mutationDistribution)
Constructs ES method with specified passed parameters.

float evolveParent()
Using the current parent, creates a generation of offspring with slight variation introduced. Then selection of best solution for new parent done according to defined selection operator.

virtual float fitnessFunction(std::vector<float> genome) = 0
The declared fitness function. When the ES method is inherited for use in another class, this function must be defined according to the current situation that decides fitness.

5.4 Tile

This simple class allows the quick creation of custom maze tiles in the form of arrays of vertices which can be rendered by OpenGL in the scene. By passing the desired walls identified by an enumerated type to the constructor the tile is formed as array of vertices and can then be easily loaded onto the GPU to be rendered.

Variables

float vertices[180]
Static array to hold vertices to form custom tile. 180 is maximum vertices to form all walls.

Functions

void Tile(int Walls)
Constructs an array of vertices that represents a desired tile. The specified walls passed into this constructor populates the vertices array with correct values to form the desired tile.
This dynamic approach to acquiring the vertices for each type of tile is more practical than storing each tile type uniquely.

5.5 EDA

This file is used as a wrapper for easily establishing a serial connection with a port and reading from it. It essentially abstracts the boilerplate code away from the main program code for clarity and readability.

Variables

SerialConnection sc Structure which holds all the important data to be stored to establish and maintain an connection to a serial port.

Functions

boolean setupSerial(char port, SerialConnection* sc)*
This is the entry function for establishing a serial connection with the specified port.

void closeSerial(SerialConnection sc)
This function closes/frees the serial connection object, therefore can be thought of as physically freeing the port safely for other programs to then use.

void printEA(SerialConnection sc)
Receives the buffered bytes from the serial port and prints them to the console. Used for debugging the serial ports output.

char getEA(SerialConnection sc)*
Receives the buffered bytes from the serial port and returns them in a char/byte buffer for use in program.

5.6 Camera

This module is designed to return a view matrix in accordance to the camera's position. The camera object receives inputs of camera movement and Eular angles which are then calculated into a 4x4 matrix to be used as a view matrix.
A detailed guide to programming the camera class can be found on `www.LearnOpenGL.com` (de Vries, 2014).

Variables

glm::vec3 Position
glm::vec3 Front
glm::vec3 Up
glm::vec3 Right
glm::vec3 WorldUp
All these variables are used to define the camera position and direction it faces. Along with Eular angles this is all that is needed to calculate a 4x4 view matrix to move scene to point of view of the camera.

float Yaw
float Pitch
The Eular angles for providing precise alterations to the view from the current direction of the camera.

Functions

Camera(glm::vec3 position, glm::vec3 up, float yaw, float pitch)
Creates a camera object with a position and direction calculated from the passed parameters.

void processKeyboard(int direction, float deltaTime)
Uses direction from passed parameter to change camera position appropriately.

void processMouseMovement(float xoffset, float yoffset, GLboolean constrainPitch)
Uses x and y values passed parameters to change camera angle appropriately. constrainPitch set to stop pitch exceeding 90 degree angles which would flip the screen direction.

void updateCameraVectors()
Using any newly acquired camera positioning and angle. Calculates and updates the rest of the camera values ready for view matrix retrieval.

5.7 Shader

This file is used mainly as a wrapper for compiling files into shaders, then creating shader programs which can be loaded onto the GPU for use at any time.

Variables

unsigned int ID
Simply an ID allocated to each shader object to show in memory where the shader program is stored for use.

Functions

Shader(const GLchar vertexPath, const GLchar* fragmentPath)*
Compiles both text files into shaders which are then used to create a shader program which is identified by the ID kept in the object.

void use()
Loads the shader the function is called onto the GPU for use thenceforth.

void updateMatrices(glm::mat4 modelMatrix, glm::mat4 viewMatrix, glm::mat4 projectionMatrix)
Updates the matrices on the shader program to passed parameters. This will be called every frame to update the scene appropriately.

5.8 Procedural Generating Shader

The Procedural generation of textures will take place in the Fragment Shader, making this an implicit procedural method (See **Implicit method**).

General flow:

1. For each pixel:
    - Calculate Pseudorandom value from Noise.
    - Calculate value for all patterns.
    - Calculate the influence each pattern has on final texture.
    - Combine patterns together according to their influence.
    - Calculate degree of primary colours in pixel.
    - Set the final value for pixel colour.
2. Output collection of pixels as displayable image.

Variables

in vec2 TexCoord
Input to this shader of the current texture coordinate. Passed from the previous stage in rendering pipeline.

out vec4 colour
Output of this shader, a value for the colour of the current pixel being considered.

uniform int tileID
The current tile being considered ID. This can be used in seeding randomness between different tiles.

uniform float time
The time can be used to seed randomness to create real time variation inbetween frames to animate them.

Functions

float mapToRange(float inputValue, float inputStart, float inputEnd, float outputStart, float outputEnd)
Used to map a value of one input range to its equivalent in another output range.
Example: RGB [0-255] to [0.0-1.0] range.

float snoise(vec2 v)
A Pseudorandom gradient Noise function known as Simplex Noise. Generates a random float from texture coordinates to produce irregularity in pattern.

float snoiseTurbulence(float x, float y, float initialSize)
Creates turbulence by combining a number of simplex Noise calls according to initial size.

Each of the simple pattern generating functions will ideally follow this format in the Fragment Shader:

vec3 f(xCoord, yCoord, noise, ...)

Where:

xCoord & yCoord = Used to generate the initial pattern. Typically drawn from texture coordinates in range 0-1.
noise = The noise value which is previously generated is used to influence the randomness within the pattern.
... = Any additional inputs that influence the pattern in some way. A common one is the scale of the pattern, for example how many squares on a checker board.

This format follows the conventional format for pattern generation functions. They are self contained and provide the necessary parameters for control. This makes pattern combination more manageable.

Chapter 6

Hardware Development

This chapter describes the development process for a device which is needed to meet the project **requirement 2**. This means it must be able to measure Electrodermal activity in a person and transmit this data to the system which the main program is running on.

6.1 EDA Sensor Module

As described in the background concepts (**Section** 2.2.1), EDA can be measured with the exosmatic DC method. By creating a small direct voltage circuit across the skin with two electrodes, the current can be kept constant and the resistance measured.
As it is a popular and reliable method, there are cheap and easily accessible EDA sensor modules available. If setup correctly, the sensor module will output a relative resistance value which can be received by a system running the main program.

Seeed Studio (seeed, 2014) produce a EDA sensor module which is used in this project.
The following table is the Grove EDA sensor module's specification:

Grove - GSR_Sensor V1.2

Parameter	Value/Range
Operating voltage	3.3V/5V
Sensitivity	Adjustable via a potentiometer
Input Signal	Resistance, NOT Conductivity
Output Signal	Voltage, analog reading
Finger contact material	Nickel

An Arduino (Appendix A.1) was connected with the module to manage the measurements and send them to the connected system.

The provided grove documentation defines how the serial port reading must be processed to obtain the resistance:

$$\text{Human Resistance}(\Omega) = \frac{(1024 + 2 * Serial\_Port\_Reading) * 10000}{512 - Serial\_Port\_Reading}$$

6.1.1 Calibration

Before the Grove sensor module can be used for resistance measurements, it needs calibration. The module must be physically adjusted in order to calibrate the output of the sensor module for use in the resistance equation. The maximum output possible should not exceed 511, even though it is capable of up to 1023.

The calibration process is conducted by first setting the sensor module up to read the raw signal output without the electrodes connected. An adjustable resistor on the module can be manipulated with a screwdriver. Whilst observing the raw signal output, the resistor is adjusted until the reading is given at 512. Once this calibration is made, the electrodes can be attached and the output values will then satisfy the human resistance equation.

6.2 Embedded Program

Software to establish a serial connection and take EDA measurements must be flashed onto the Arduino Microcontroller.
During the setup stage, this software first establishes a serial connection to a system over a USB port.
During the main loop the Microcontroller reads off the analogue pin which is connected to the output of the sensor module, this will be the resistance signal. The Microcontroller then sends it over the established serial connection to the computer system.
During each measurement interval, multiple readings are made and averaged over 5ms. This is done to avoid any anomalous values being sent from the device.

6.2.1 Pseudo Code

Algorithm 3 Microcontroller Program: Measure Electrodermal Activity

```
averageEA ← 0
sensorValue ← 0
EA ← AnaloguePin0

function SETUP
    serial ← establishSerial
end function

function LOOP
    sum ← 0
    for 0 to 10 do
        sensorValue ← read(EA)
        sum ← sum + sensorValue
        delay(5)
    end for
    averageEA = sum/10
    serial.send(averageEA)
end function
return
```

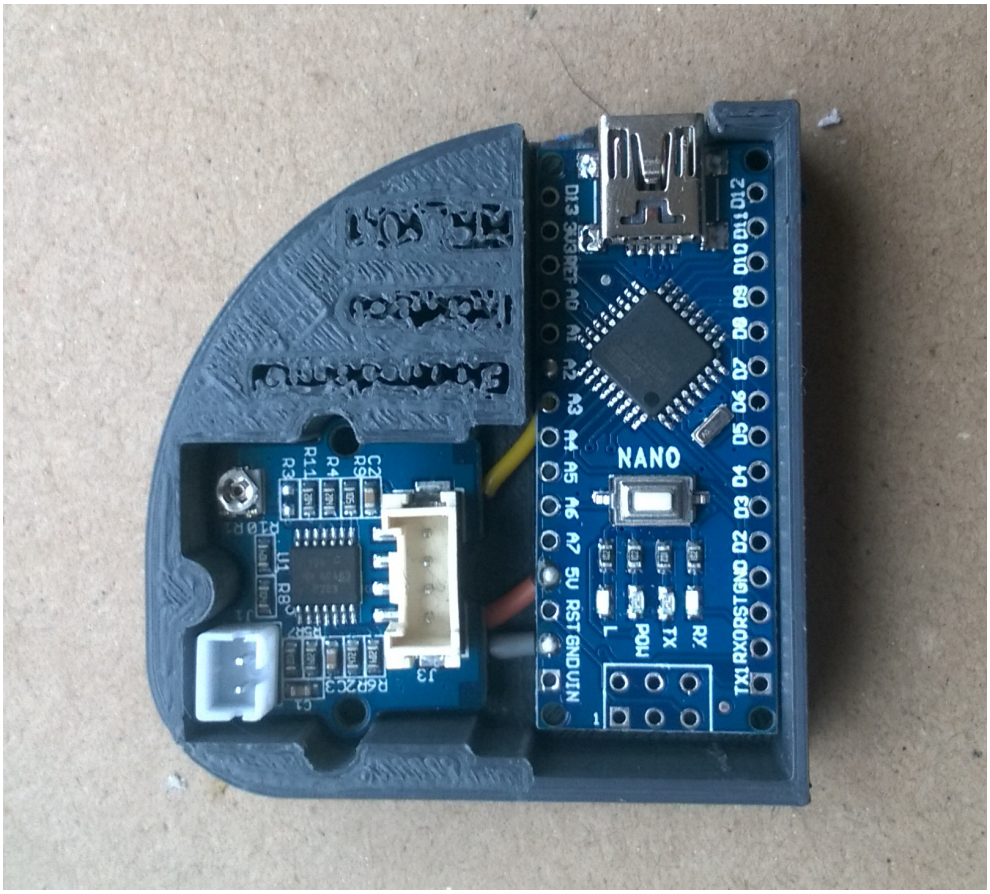

Figure 6.1
The prototyped device used to record EDA measurements

Chapter 7

Software Development

This chapter covers the the development process of the program. The design is followed closely to produce all components identified in figure 5.1.
The program requirements will be completed within iterations or sprints of program development. When specific requirements are met, they will be referred to by case code.

The program will be produced over a planned three iterations. In each iteration certain requirements are met.

Iteration 1: Aims to satisfy requirements 1 & 2. A fully functioning and customizable 3D maze environment will be established as the foundation of the program. The EDA recording module will be developed to provide the EDA functionality for the program.

Iteration 2: Aims to satisfy requirements 3 & 4. The Q-Learning module is developed along with the ES module which is extended to be used in the Q-Learning module.

Iteration 3: Aims to satisfy requirement 5 & preparation for requirement 6. A procedural Fragment Shader should be created with modifiable features which controls the final generated texture. Finally all the functionality should be brought together in preparation for the program's use in the user study.

7.1 Iteration 1

7.1.1 3D Environment Development

The 3D maze environment is rendered within a window using the OpenGL API. The rendered scene is controlled using conventional logical methods and algorithms commonly employed in graphics programming.

The fundamental structure and methods are established first. This is done by developing a basic 3D graphics program to render a static 3D object to the window.
First, the initialisation stage is executed. This is were all the preparation is made to begin rendering scenes:

- The OpenGL pipeline is configured. The pipeline state is adjusted using a set of modifiable attributes.
- Shader programs are complied and loaded onto the GPU from source code (C1.3D).
- The structure for reading vertex array objects is defined.

Once the initialisation stage is complete, the program enters a rendering loop. This is were the world scene is drawn, then rendered to the window every frame.
Any changes made to the scene happen here. This largely consists of translating objects in the world to new positions. In the case of this static rendering program, not much happens in the rendering loop apart from loading the same set of vertices to the GPU to be render to the window. However, all is in place for the next steps in realising a complex 3D environment.

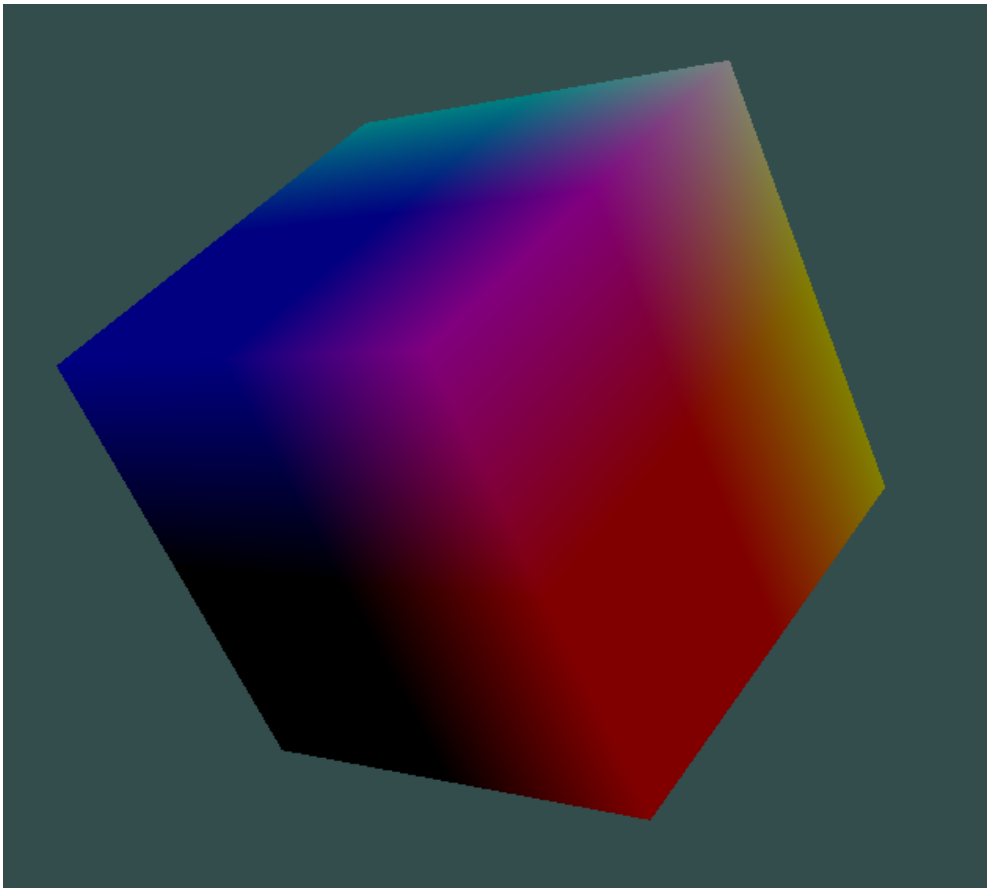

Figure 7.1
A static 3D cube rendered in a window using OpenGL

Camera Development

In this program the virtual camera system takes a first-person perspective. The first step to achieving this camera system is to develop the camera module. The camera module works by taking keyboard and cursor inputs, then calculating the changes to be made to the scene. (van Oosten, 2011)
Contrary to what may be expected, rather than calculating the camera's movement through a static scene of objects, the objects move around the camera view. This is done to give the illusion of camera movement through the world. The complex reasoning is described in the official OpenGL handbook (Woo et al., 1999). Once the calculation is completed, a new view matrix is formed. The view matrix is used to move the objects in the scene in such a way as to look like the camera has moved to a new position and angle according to the inputs.

The next step for camera movement is to provide a method for reading keyboard and cursor inputs. This is required to calculate the view matrix in the camera module. This is done using callback functions provided by the GLFW API. The callback functions are defined to handle specific keyboard inputs and cursor movements to update the view matrix. These callback functions are then registered during the initialisation stage.
This results in perceived camera movement each frame as inputs are detected and the view matrix is updated to transform objects in the scene to new positions (C2.3D).

Maze Development

A structured way for forming custom mazes needs to be established. It is impractical to manually draw each object into the world, it would result in huge amounts of redundant code and require much more time to design. Instead the program uses a class for automatically forming tiles and functions to place them in the world.

The tile module is developed purely for constructing an array of vertices which represents a cube with surfaces in the desired directions. The cube surfaces drawn are decided by the cardinal directions passed as parameters to the constructor. Convenient functions are then provided that enables the placement of these generated maze tiles in specific positions within the world (See function placeTile). This enables fast formation of custom mazes which are manageable (C3.3D).

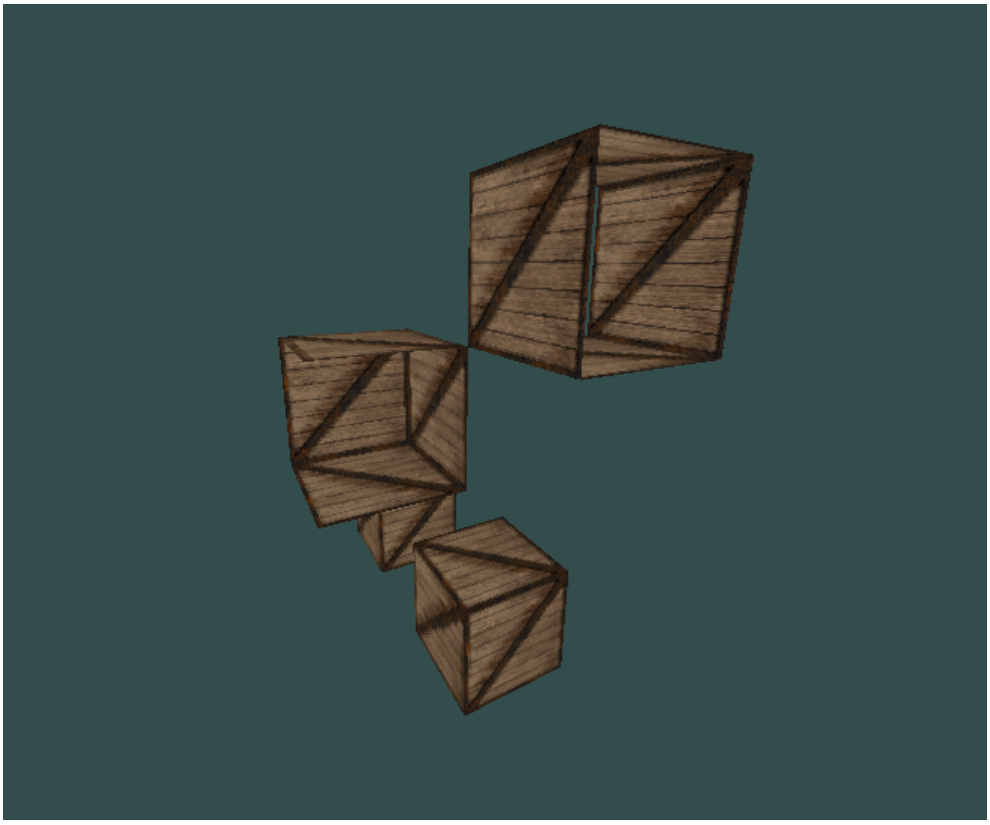

Figure 7.2
Generated tiles placed around the 3D environment using supported functions.

Collision Detection

The final functionality required for **Requirement 1** is collision detection for the camera position.
The camera should not be able to move to a position outside of any of the tiles. Essentially this is achieved by restricting the camera position to remain inside the boundaries of the tiles (Van Den Bergen, 2003). If the next movement would take the camera position beyond the tile wall, the movement in that direction is not applied (C4.3D).
Collision detection is considered separate logic from the rendering process. However it still draws on knowledge of the tile boundaries from the tile vertex arrays, just like OpenGL needs to for rendering.

Summary

The foundations for the rest of the program is in place. The core 3D environment allows the user to move through a custom built maze and is restricted to movement inside it. The rest of the modules can be developed independently and then introduced in turn into the program to achieve the rest of the project goals.

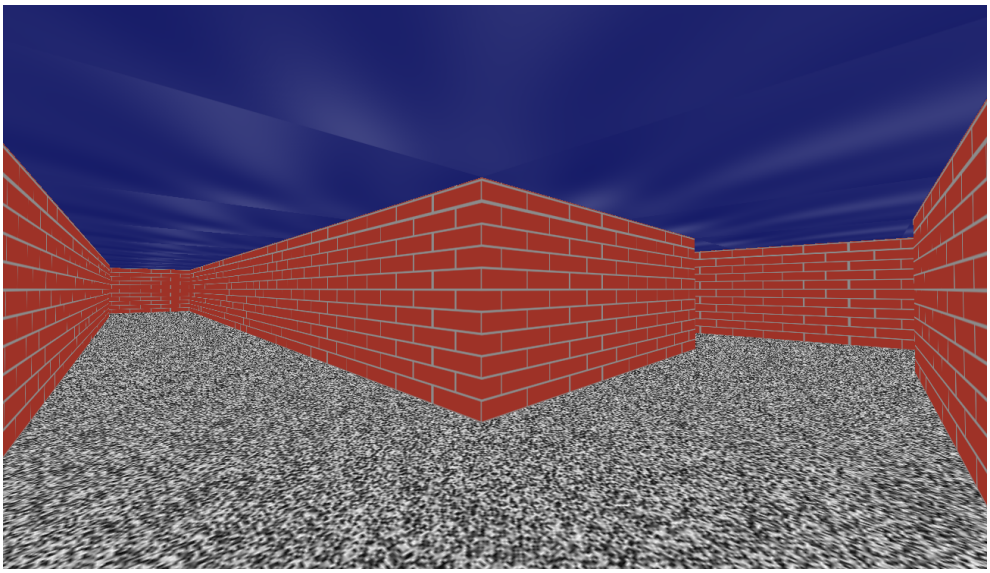

Figure 7.3
The 3D environment with textured walls

7.1.2 Electrodermal Activity Development

The development of the EDA module is used to interact with the EDA recording device. This requires establishing a connection with a serial port on the system the program is running on (UC6.EDA). As this program is run on a windows OS, the win32 API (Axelson, 2015) is used in order to establish the serial connection (UC5.EDA).
The module then allows the retrieval of EDA measurements from the device (UC7.EDA), the measurements first undergo processing to calculate the human resistance. The resistance values can then be read into the main program and used as desired (UC8.EDA).
With this module, the program can interact with the EDA device to retrieve correct resistance measurements. These can then be used for the learning process by the Q-Learning agent.

7.2 Iteration 2

The design of the AI modules were created with a standalone/independent structure to allow the control of them on their creation as passed parameters (UC9.RL),(UC12.RL).
This is done so the modules can easily be included in any program with a high level of control provided. The modifiable parameters and abstract functions enables tweaking of the process to suite the particular problem for the best performance.

As according to the design, the ES is an abstract class that is inherited by the Q-Learning agent (UC13.ES) as part of it's policy to decide the next best action. For this reason, naturally the ES module will be created first and tested independently before creating the Q-Learning module, and using it with the ES employed as its policy.

7.2.1 Evolutionary Strategy Development

The ES is developed to provide user control to the following parts:

- The selection operator.
- The number of genes.
- The number of children in each parent's offspring.
- Mutation rate, the probability the mutation operator executes on a gene.
- Mutation distribution, the range the mutation can extend to.
- The fitness function, requires override of pure virtual function.

The ES works by creating a copy of the current parent, this is referred to as a child. The child has random changes made to it by visiting each float in the vector and applying the mutation operator with a probability decided by the mutation rate. The mutation operator then works by making a change to the float it acts on according to a Gaussian normal distribution (Gauss, 1809) with a default mean 0 and standard deviation decided by the mutation distribution. The child is then checked against the best solution so far according to the fitness function. If it is considered better, the child becomes the new best solution. This process is repeated for the number of children set. Once all children have been generated and compared, the new parent is set to the best child generated.

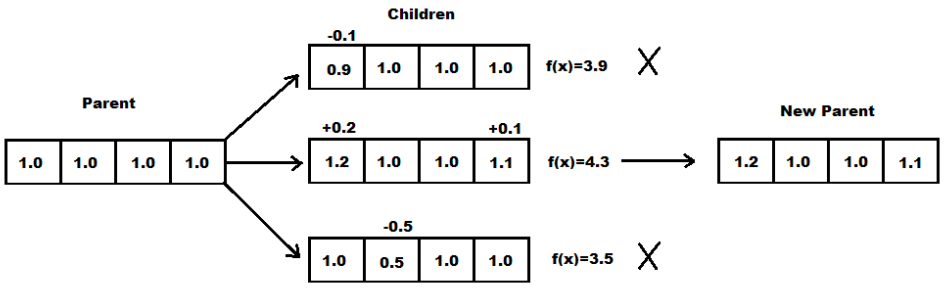

Figure 7.4
Explanation of Evolutionary strategy using the comma separation.

7.2.2 Q-Learning Linear Approximation Development

The Q-Learning agent is developed to provide user control to the following parts:

- The number of features to represent the state.
- The learning rate, the degree of influence new information has on weights.
- The discount rate, the degree of influence the future estimated reward has on the weights.
- The Q function used for the inherited ES fitness function.

The Q-Learning agent works by finding a new set of actions to take from the current state to meet some goal. The ES is used to find the next action to take, therefore acting as part of the agent's policy (UC11.RL). The fitness function for this problem is unknown. Instead, the agent's Q value estimate is used to predict the fitness of each action (UC14.ES).
All the weights default initialise to 0 on creation. Therefore the first action taken is purely random as the agent has had no experience in the problem yet. The predicted "best" action is then taken with the change observed in the EDA used as the reward.
The reward is used to update the weights according to the Q-Learning linear function approximation equation (Algorithm 1). A positive reward will reinforce weights of the increased states and weaken the decreased states. If the reward is negative, this applies in the opposite direction (UC10.RL).

Ideally after making a few actions, the weights will be changed to prioritize which features are important and avoid unimportant or harmful features. Then when future best actions are estimated, it picks the better ones it has learnt from past its experiences.

7.3 Iteration 3

In this iteration the focus is on producing the procedural texture generation functionality to apply to the maze environment. The majority of this is done in the Fragment Shader and written in GLSL (UC15.PTG).

The Fragment Shader is part of the shader program which is loaded onto the GPU. For that reason, the shader program is separate and has no direct access to the important data from the main program. However the shader will require data for a few things, including the values for the texture features which will shape the generated procedural texture.
To do this in the main program, one of the shader module functions are used to load any required data onto the GPU for the shader program to access (UC16.PTG).

7.3.1 Pattern Functions

The process of developing the procedural texture patterns is largely based around using various math functions. They typically take 2 floating point inputs from the texture coordinates in order to output an appropriate float for the colour.
Computer generated images are often constructed from geometric models. Composed of a number of lines, curves and polygons. Mathematical functions can represent these geometric models in images. Provided the function is structured to take a 2-dimensional domain to a colour. (Karczmarczuk, 2002)

A definition of this could be expressed as:

$$Image = Point \rightarrow Colour \quad \text{(Elliott, 2003)}$$

In figure 7.5 the sin function with an input of x has its output used for the colour at each pixel. In the function the x axis is first converted to radians, then applied through the sin function and the returned value used for all three RGB values for the pixel. This results in the dark and light strip pattern on the 2D surface.

The number of times the sin pattern repeats across the texture is controlled by simply multiplying the input coordinates. This works as it is a periodic function which will repeat past some boundary.
Periodic Functions are important as they allow recurrence within patterns, this allows the control of scale in the final generated texture. The most well used Periodic Functions are sin, cos and perhaps mod (Altman and Kidron, 2016).

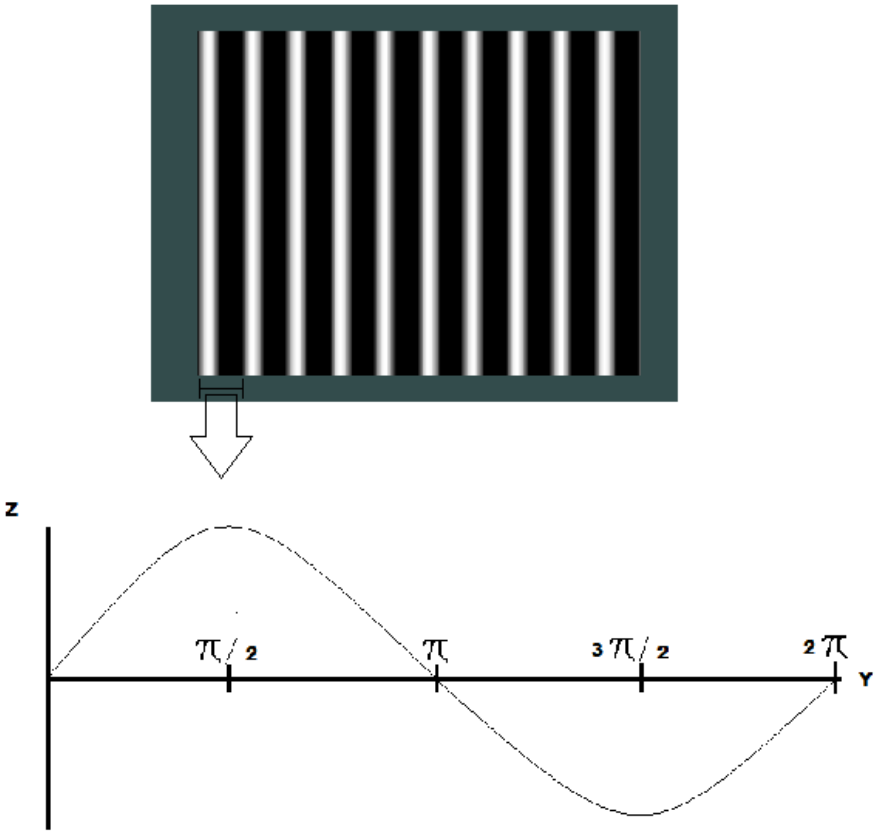

Figure 7.5
The periodic function sin to take the input of the x axis as the output for the pixel colour.

Simple patterns like these can be combined to create more complex patterns. In this project to procedurally generate the complex texture pattern made up of the texture features, the layering technique (See Section: 2.5) will be used.
In the shader a collection of simple patterns will be generated (UC17.PTG). Some of the patterns' generation process uses a gradient Noise value passed as a parameter to introduce randomness in the resulting pattern (UC18.PTG). Each generated pattern then has it's influence calculated. The influence is how dominant that pattern is when it comes to layering it in the final texture. Each pattern is then layered one over the other according to their influence/strength to result in a new complex pattern.

The following layering equation is used to calculate the new complex pattern from the simple patterns and their associated feature (UC19.PTG):

$$out = \frac{(p_0 * f_0) + ... + (p_n * f_n)}{n}$$

Where:
p = pattern [0.0 - 1.0]
f = feature [0.0 - 1.0]
n = Number of features

7.3.2 Texture Features

The following table describes the texture features the Fragment Shader will use (UC20.PTG):

FragmentShaderFeatures

Name	description	Notes
featureRed	Influences the degree of primary colour red in texture.	
featureGreen	Influences the degree of primary colour green in texture.	
featureBlue	Influences the degree of primary colour blue in texture.	
featureBrightness	Influences the level of brightness in the texture colours.	This is a boundary feature. It has no effect until it exceeds a value then is triggered.
featureAnimate	Influences the rate the texture animates in real time.	Only certain generated patterns support animation between frames.
featureRing	Presence of a ring pattern in texture.	Supports animation. Uses Noise.
featureHeart	Presence of a heart pattern in texture.	
featureMarble	Presence of a marble pattern in texture.	Uses Noise.
featureVStripe	Presence of a vertical stripe pattern in texture.	Supports animation. Uses Noise.
featureDiamond	Presence of a diamond pattern in texture.	
featureBrick	Presence of a brick pattern in texture.	
featureCheckerboard	Presence of a checkerboard pattern in texture.	Supports animation. Uses Noise.
featureStar	Presence of a star pattern in texture.	
featureXor	Presence of a Xor pattern in texture.	

7.3.3 Electrodermal Activity Recording Suite

To gather meaningful information from the users during the user study, their EDA needs to be recorded during a period before and while they use the program. Rather than just logging the data in a file, a GUI was created to visualise the EDA readings over the period on a line graph in real time. This allows much more control within the user study as issues can immediately be detected and resolved.

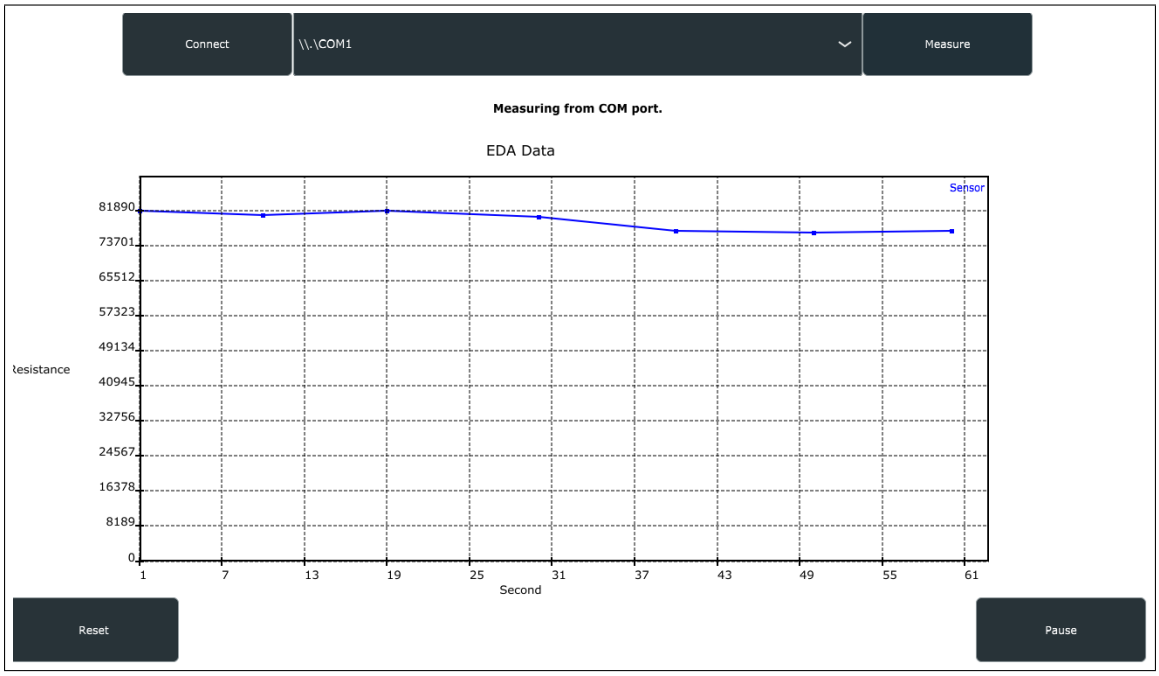

Figure 7.6
JUCE framework used to create a GUI for plotting EDA measurements.

The EDA Suite was developed using the JUCE C++ framework (UC23.US). JUCE provides GUI functionality for simple window applications. This was used to provide an interface to establish a serial connection and begin plotting the EDA measurements received to a line graph. The application was also given the functionality to pause and reset the recording session. This is useful as it gives further control during the user study recording periods.

7.3.4 Bringing it together

The Q-Learning agent needs to be employed in the program so that it contains a vector of features which represent the texture features. These features will then be used in the Fragment Shader to procedurally generate the textures. The agent must be constructed with a feature size equal to that of the number of texture features decided on.

Every texture update cycle the agent chooses an estimated best action for a new set of features. The features are uploaded to the GPU and used to generate a new complex texture. The change in EDA is recorded and used as a reward to learn the effectiveness of the action that was taken.
The idea is that the better texture features discovered by the agent are reinforced by the reward indicated by any emotional response.

The Irrklang API was used to implement audio in the program for further immersion (Appendix A.3).

Chapter 8

Testing

In this chapter, the testing will be conducted on the software produced after each development iteration.
Once testing of the whole system has been successful, the requirements will have been met. Excluding requirement 6, as this requires the user study to be performed.
Any flaws or weaknesses found should be noted and then corrected when the system is revisited before moving onto the next iteration. This is repeated until no more software and hardware bugs are present. It is crucial the program successfully covers all the testing before its use in the user study.

8.1 Test Cases

Three examples of the test cases made are covered in this section.
The rest of the test cases for the program's major functionality can be found in the appendix B.

Test Case 1: Collision detection

ID	1
Title	Collision detection
Pre-conditions	Generate environment with walls for all 4 cardinal directions
Test Steps	1. Move camera position to pass North wall. 2. Move camera position to pass East wall. 3. Move camera position to pass South wall. 4. Move camera position to pass West wall.
Expected Results	On reaching each wall, the camera is restricted to continue moving in that direction.
Result	✓

Test Case 3: Electrodermal activity measurement device

ID	3
Title	Electrodermal activity measurement device
Pre-conditions	1. Embedded program flashed onto device. 2. Individual is wearing electrodes. 3. Serial terminal setup to read from correct port.
Test Steps	1. Observe initial measurements in serial terminal. 2. Instruct individual to take a deep, prolonged breath. 3. Observe new measurements.
Expected Results	Check readings are within an expected/reasonable range $[1000 - 100,000\Omega]$ (Fish et al., 2003). Then observe at the time the individual takes a deep breath, the resistance drops notably as physiological reaction to taking a deep breath takes place.
Result	✓

Test Case 8: Shader program creation

ID	8
Title	Shader program creation
Pre-conditions	1. Source code for Vertex Shader written. 2. Source code for Fragment Shader written.
Test Steps	Start the program and wait for setup phase to complete.
Expected Results	Debug information to console indicates successful creation of the shader program.
Result	✓

8.2 Reinforcement Learning Example Test

It is important that the Q-Learning agent is confirmed to be working correctly before use in the main program.
To confirm the ES and Q-Learning modules are working as expected they should be tested against a simple problem to show if they can solve them in a reasonable number of iterations.
This test is aimed at confirming the modules are functioning as expected since they are developed from scratch. This is not a test to measure the performance of these different techniques.
The ES module must be tested first to confirm it is functioning correctly before the Q-Learning module. This is because the Q-Learning agent is dependant on the ES to function.

The example problem definition
The solution is represented by a vector of floats. The quality of the solution is equal to the sum of the even indexed items minus the sum of the odd indexed items.

Ideally for this problem, every iteration the even indexed floats would increase and the odd indexed floats would decrease. For testing against the example problem a quality/fitness of 25 must be achieved to be considered satisfactory.

ES Tested
The ES will use the following parameters to solve the example problem:
Selection = Comma Selection
numberOfGenes = 14
numberOfChildren = 100
mutationRate = 0.1
mutationDistribution = 0.1

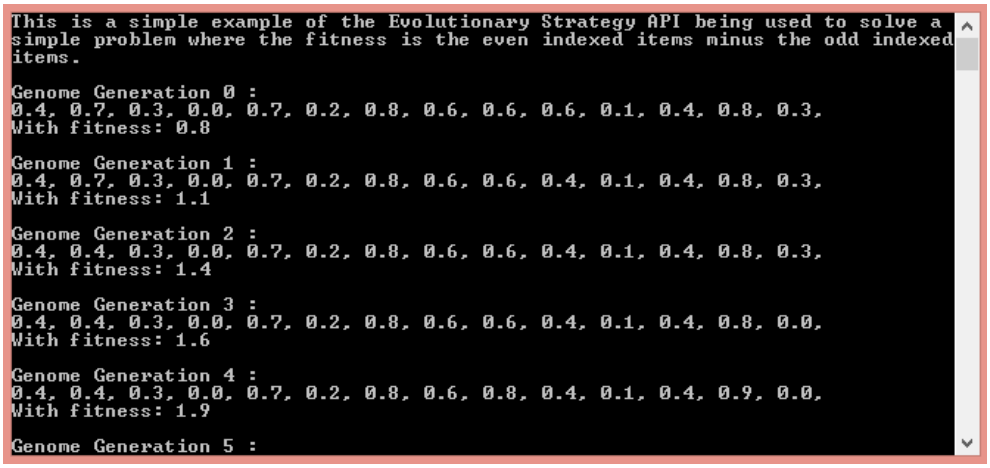

Figure 8.1
The Evolutionary strategy solving a simple mathematical problem.

Observe in figure 8.1, the ES begins with a vector of numberOfGenes default initialized between 0.0 and 1.0. It creates a set of new solutions, calculates their fitness and then picks best for the parent of next generation. In this simple example case the fitness function is accurately known so the fitness function is completely accurate and the best solution from the children is always correctly chosen.
The parent isn't in the considered selection process as the comma selection is used.

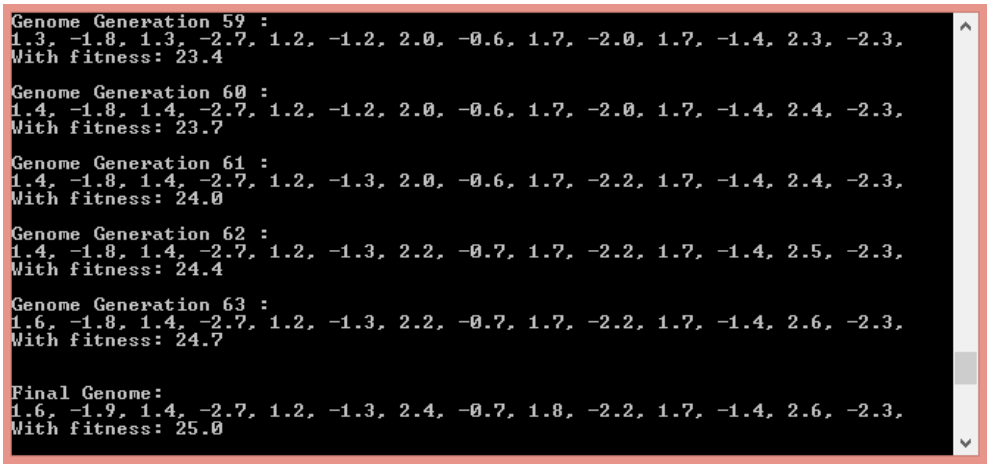

Figure 8.2
The Evolutionary strategy solving a simple mathematical problem.

This repeats every iteration, creating progressively better solutions until satisfying the test with a fitnesses of 25 as observed in figure 8.2. This is only achieved so easily as the quality/fitness function is known. If the problem was unknown, then a way of estimating better solutions is needed.
That is where Q-Learning becomes very useful, it can solve the problem without knowing the problem definition.

Q-Learning Tested
The Q-Learning agent will use the following parameters to solve the example problem:
numberOfFeatures = 14
learningRate = 0.2
discountRate = 0.7
(The same ES parameters used as in the standalone ES test.)

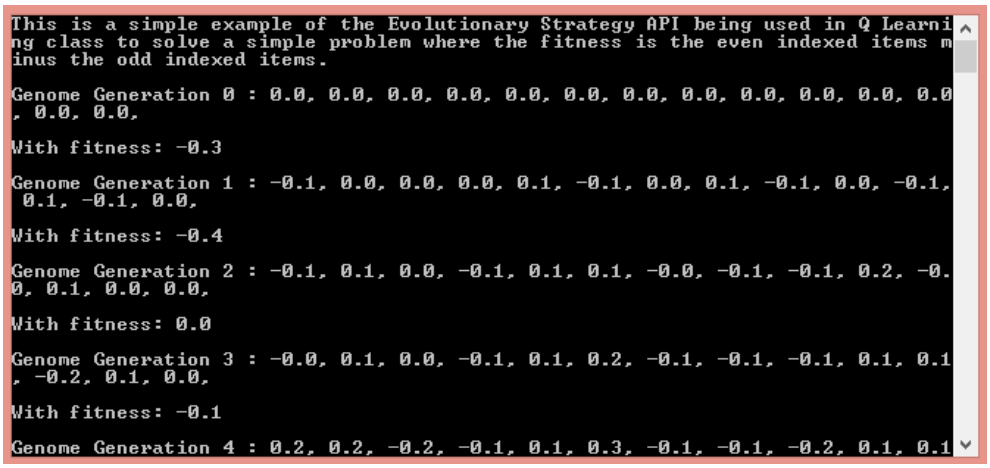

Figure 8.3
The Q-Learning agent solving a simple mathematical problem.

Observe in figure 8.3, the Q-Learning agent begins with a vector of numberOfFeatures initialised to 0.0 by default. It searches for a new best estimated action from its current state, then makes a transition to that state. The agent observes the reward for the new state, then updates the weights according to the linear approximation weight update equation.
After the first action is taken, the agent does not make the correct changes to satisfy the problem. Therefore it results in a negative fitness and the weights are adjusted accordingly. This continues to repeat until through trial and error the weights adjust to favour actions which increase indexed items, and decrease odd indexed items.

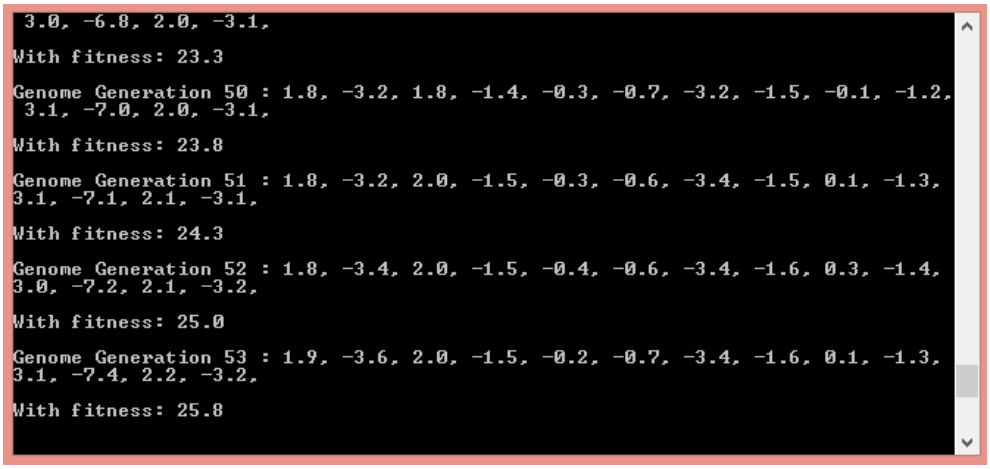

Figure 8.4
The Q-Learning agent solving a simple mathematical problem.

After the process has been repeating for 53 iterations as shown in figure 8.4, the fitness of 25 is satisfied. The fitness is continually increasing through correct actions determined by a set of learned weights.
The main advantage to using a RL is not in the performance. Q-Learning is able to solve problems using only a reward signal after an action is taken. The evolutionary strategy could not do this alone without trying each action it finds first, which is impractical in many situations.

Chapter 9

User Study Strategy

In order to carry out the user study, EDA samples must be gathered from a population. Each user's natural EDA and their EDA whilst interacting with the program are recorded. The mean is then calculated for each period and used in each sample set.
These samples are collected in order to conduct a paired samples student t-test on the two sample sets. This is to discover if the program creates a statistically significant increase in the emotional response as it is designed to, or fails to do so.

9.1 Setup

It is important to engage the user in the program to get the maximum emotional response possible through the procedurally generated textures. Not from any other interfering external stimuli. Therefore a number of test conditions to create further immersion will be set.

Set of experiment conditions and environment:

- Dimly lit room.
- User advised to fully focus on the program from start until finish.
- Quiet test location.
- User advised not to speak.
- The electrodes will be securely fitted onto the index and middle finger, palm side.

If any of these conditions are breached and the study leader decides this impacts the results, the current session will be reset provided the volunteer still gives their verbal consent.

It is possible to either use the same Q-Learning agent across the study between users or to create a new one for each user. In this user study, a new agent will be created for each user. This means it will have no past experiences and be tailored towards the current user interacting with the program. This was chosen instead as past agent experiences could interfere with the current user session. If this user study was successful, it could be worth further investigation in a future user study to discover which method is more effective.

9.2 Sampling Method

The sampling method outlines how the two sets of samples will be obtained from the user study. This involves the method of measurements, and then what calculations need to be done to produce the final sample data items.

9.2.1 Measurement Strategy

During each sampling period, each user will have a collection of EDA measurements taken. The measurements will be taken in 10 second intervals and measure the resistance(Ω) across the skins surface. 10 Second intervals provides suffcicient time for any emotional response to the external stimuli to take place, which is usually 1-5 seconds (See **Skin Conductance Response**).
Typically this should result in about 30 measurements being taken during both periods as each period should take 5 minutes to conduct.

The mean will be calculated from the collection of resistance values. The two means will then be added to a first baseline sample set and a second stimulus sample set respectively. Once a sufficient sample size of 10 is collected, the student t-test can be conducted.

9.2.2 Sample 1: Electrodermal Activity Baseline Period

The EDA baseline of each user will be taken during a baseline period. This is to find the natural physiological state of the user before they are effected by any major external stimulus (UC21.US).
In a natural baseline recording, no stimuli are presented. The user should be comfortably seated in a relaxed position, with their eyes closed if they wish. They should not be subject to the program at this point and given time to get used to the situation.
An ideal baseline period should be at least 2-4 minutes and conducted at the beginning of the recording session. However since the baseline period will be used as a sample for the t test, a baseline period of 5 minutes is chosen as this matches the approximate time to navigate through the maze program during the stimulus period. Additionally a longer baseline period is beneficial as it gives time for the individual to get more comfortable and relaxed for the experiment ahead.

The natural baseline period will be conducted as follows:

1. After the user has been briefed on the test they will be seated.
2. The electrodes will be fitted to the index and middle finger to record EDA.
3. The user will confirm once they are comfortable and feel relaxed.
4. The baseline period will be recorded for 5 minutes.

Notes:

- The user is welcomed to suggest anything which will make them feel more relaxed and the tester will do his best to provide within the test guidelines.
- It will be suggested to the user to close their eyes if they wish to.

9.2.3 Sample 2: Electrodermal Activity Stimulus Period

The EDA stimulus period will be taken during a period in which the user interacts with the program. By navigating their way from the default starting position to the end point indicate by a set of white doors (UC22.US).

The EDA Stimulus Period will be conducted as follows:

1. The user remains seated after the baseline period has ended.
2. The system is prepared for recording the use of the main program for the stimulus period. This should be done quickly and without agitating the user.
3. The user is informed they must make their way through the maze to find the exit point, marked by white doors.
4. The user is left to navigate their way through the maze by the experimenter.
5. Once the user reaches the exit indicated by the white doors. The experiment is concluded and the data is stored in a sample folder.

Notes:

- If for whatever reason the user cannot reach the exit after a reasonable 20 mins, the experiment will be cancelled.
- It must be stressed, if the user asks to leave at any time for whatever reason they are allowed to do so without any objections form the experimenter.
- As the user explores the maze, any unique events such as a stand out comment from the user should be recorded as a timestamped note.
- The maze is expected to take approximately 5-6 minutes to navigate through.

Chapter 10

User Study Analysis

The two sets of sample data taken during the baseline period and stimulus period have been collected. Now they will undergo a Student's paired samples t-test to show if their is a reliable increase in emotional arousal from a user's natural state to when they're interacting with the program.

10.1 Experiment Sample Results

The EDA measurements are taken as resistance(Ω). If the resistance increases between two intervals, the user's degree of sweat is lower. This means they are likely experiencing a decrease in emotional arousal. In contrast a decrease in resistance indicates the user is experiencing an increase in emotional arousal.

The following table shows the results for the mean resistance for each user during the baseline period (UC21.US), then for the stimulus period (UC22.US).

Experiment Sample Results

Sample	EDA Baseline Samples S_1 (Ω)	EDA Stimulus Samples S_2 (Ω)
1	95038	80463
2	77688	63144
3	63432	40461
4	41262	37923
5	444031	261289
6	55253	55829
7	87189	78332
8	193312	133720
9	158233	159224
10	76060	63662
Mean	129150	97405

10.2 Student's T-Test

The Student's T-Test will be used in this user study to test the difference between the samples (UC24.US). Essentially it is used to determine if there is a statistically significant difference between two samples. This is the confidence that the change in the results is caused by the changed factor and not by random chance.
The Student's T-Test is a well established statistical hypothesis test method, first covered by William Gosset in his 1908 paper (Student, 1908).

Depending on the source of the samples gathered, there are different types of t-tests that can be conducted. For this user

study the two samples are obtained from the same population of users. The one sample during the baseline period, the second during the stimulus period. Therefore the paired samples variant of the t-test is used.
This increases statistical power as there is no random variation introduced from using different unrelated groups.

The t-score/value is the ratio between the two sample sets. A t-value of 3 would indicate the sets are three times as different from each other.
More importantly, the t-value has a p-value. The p-value is the probability the difference occurred by chance. The value represents a chance between 0% to 100% and typically written as a decimal equivalent.
The p value is used as the evidence against a null hypothesis. The smaller the p-value, the stronger the evidence that the null hypothesis should be rejected. If the calculated P-value is less than the specified alpha level (Typically 0.05) then the null hypothesis is rejected in place of the alternative hypothesis which means there is a statistically significantly difference.

10.2.1 Hypothesis

The t-test works by first defining a null hypothesis and an alternative hypothesis. Both hypotheses are defined in such a way to be mutually exclusive.
The hypotheses are claims made about the difference between the samples' means. If the null hypothesis is shown to be false, then the alternative hypothesis can be accepted.
Not only must the samples' means disagree with the null hypothesis to be considered disproved, but the difference must also be shown to be significantly different, using the calculated p-value.

The following one-tailed hypotheses are defined:

Null hypothesis
The mean resistance in the user during the baseline period is equal to or statistically less than the mean resistance during the stimulus period.

$$H_0\colon\ S_1 - S_2 \leqslant 0$$

Alternative hypothesis
The mean resistance in the user during the baseline period is statistically greater than the mean resistance during the stimulus period.

$$H_1\colon\ S_1 - S_2 > 0$$

When using a one-tailed test, the test is for the possibility of the relationship in one direction and completely disregarding the possibility of a relationship in the other direction.
All the statistical significance in a one-tailed test is allotted in the one direction of interest.
The one tailed t-test is used in this investigation as the interest is in discovering if the program creates a significant increase in emotional arousal in the user. Considering the program is designed to increase emotional arousal, the opposite case that it is decreasing arousal is irrelevant. If a future investigation was conducted to discover if the program effects emotional arousal in either way, a two tailed test would be made.
The direction of the change needs to be shown first to establish that the significant difference is in the correct direction.

Alpha level
The alpha level is the probability of incorrectly concluding that the null hypothesis is false. Therefore if the p-value is less than or equal to the alpha level, the null hypothesis can be confidently rejected in place of the alternative hypothesis. In this case the resistance in the baseline period will be considered significantly greater than that in the stimulus period.

The α level is decided by the level of confidence desired from the evidence to reject the null hypothesis in favour of the alternative hypothesis. The confidence level should be high in the evidence. A commonly desired level of confidence is 95% meaning an alpha level of 0.05 is chosen for a one-tailed test.

$$ConfidenceLevel = 95\%$$

$$\alpha = 100.0 - 0.95 = 0.05$$

In this user study an alpha level of 0.05 will be used. If the p-value is equal to or less than the alpha level, the study will support the goal of the program which is to increase emotional arousal in the user.

10.2.2 Analysis of Sample Data

Part of the one tailed t-test is to show the direction of the calculated difference in the t-value. This needs to be done because if the difference is in the opposite direction to the alternative hypothesis, then the null hypothesis remains satisfied. Therefore the first part to the t-test is to find the direction of any difference.
The direction should oppose the null hypothesis and support the alternative. This will mean when the t-value is then calculated, the difference it indicates is in the direction of the alternative hypothesis.

$$\text{if}(S_1 - S_2 > 0) \text{ then t-value positive for current test}$$

$$129150 - 97405 > 0 \text{ therefore the t-value will be positive.}$$

The first sample's mean is greater than the second sample's mean. This suggests the direction of the difference opposes the null hypothesis and supports the alternative hypothesis. The t-value that will be calculated will therefore show significant difference in the same direction. This is indicated by keeping the t-value positive. If the direction was the opposite way, the t-value would be made negative. (Ruxton and Neuhäuser, 2010)

Now a number of steps must be taken on the results to find the t and p values. Calculating the p-value is the key to discovering if there is significant difference between the two samples.

To begin with, the mean between the difference of every sample must be calculated:

$$\text{Mean Difference: } \overline{d} = \frac{(S_1 - S_2)}{N} = 31745.072$$

Next the mean difference can be used to find the sample standard deviation:

$$\text{Standard Deviation: } s_d = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{d})^2} = \sqrt{\frac{1}{10-1}\sum_{i=1}^{1} 0(x_i - 31745.072)^2} = 55841.48337$$

From the sample standard deviation, the standard error can be found:

$$\text{Standard Error: } SE(\overline{d}) = \frac{s_d}{\sqrt{n}} = \frac{55841.48337}{\sqrt{10}} = 17658.62754$$

The mean difference is used again, along with the now calculated standard error to find the sought after t-value:

$$\text{T-Value } T = \frac{\overline{d}}{SE(\overline{d})} = \frac{31745.072}{17658.62754} = 1.797708906$$

The calculated t-value can be used to see if its equivalent p-value is less than the alpha level, and therefore surpasses the desired confidence level.
This is done using a lookup table, which is typically used as calculating the p-value accurately requires doing a definite integral calculation.

t Table

cum. prob	$t_{.50}$	$t_{.75}$	$t_{.80}$	$t_{.85}$	$t_{.90}$	$t_{.95}$	$t_{.975}$	$t_{.99}$	$t_{.995}$	$t_{.999}$	$t_{.9995}$
one-tail	0.50	0.25	0.20	0.15	0.10	0.05	0.025	0.01	0.005	0.001	0.0005
two-tails	1.00	0.50	0.40	0.30	0.20	0.10	0.05	0.02	0.01	0.002	0.001
df											
1	0.000	1.000	1.376	1.963	3.078	6.314	12.71	31.82	63.66	318.31	636.62
2	0.000	0.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	22.327	31.599
3	0.000	0.765	0.978	1.250	1.638	2.353	3.182	4.541	5.841	10.215	12.924
4	0.000	0.741	0.941	1.190	1.533	2.132	2.776	3.747	4.604	7.173	8.610
5	0.000	0.727	0.920	1.156	1.476	2.015	2.571	3.365	4.032	5.893	6.869
6	0.000	0.718	0.906	1.134	1.440	1.943	2.447	3.143	3.707	5.208	5.959
7	0.000	0.711	0.896	1.119	1.415	1.895	2.365	2.998	3.499	4.785	5.408
8	0.000	0.706	0.889	1.108	1.397	1.860	2.306	2.896	3.355	4.501	5.041
9	0.000	0.703	0.883	1.100	1.383	1.833	2.262	2.821	3.250	4.297	4.781
10	0.000	0.700	0.879	1.093	1.372	1.812	2.228	2.764	3.169	4.144	4.587
11	0.000	0.697	0.876	1.088	1.363	1.796	2.201	2.718	3.106	4.025	4.437
12	0.000	0.695	0.873	1.083	1.356	1.782	2.179	2.681	3.055	3.930	4.318
13	0.000	0.694	0.870	1.079	1.350	1.771	2.160	2.650	3.012	3.852	4.221
14	0.000	0.692	0.868	1.076	1.345	1.761	2.145	2.624	2.977	3.787	4.140
15	0.000	0.691	0.866	1.074	1.341	1.753	2.131	2.602	2.947	3.733	4.073
16	0.000	0.690	0.865	1.071	1.337	1.746	2.120	2.583	2.921	3.686	4.015
17	0.000	0.689	0.863	1.069	1.333	1.740	2.110	2.567	2.898	3.646	3.965
18	0.000	0.688	0.862	1.067	1.330	1.734	2.101	2.552	2.878	3.610	3.922
19	0.000	0.688	0.861	1.066	1.328	1.729	2.093	2.539	2.861	3.579	3.883
20	0.000	0.687	0.860	1.064	1.325	1.725	2.086	2.528	2.845	3.552	3.850
21	0.000	0.686	0.859	1.063	1.323	1.721	2.080	2.518	2.831	3.527	3.819
22	0.000	0.686	0.858	1.061	1.321	1.717	2.074	2.508	2.819	3.505	3.792
23	0.000	0.685	0.858	1.060	1.319	1.714	2.069	2.500	2.807	3.485	3.768
24	0.000	0.685	0.857	1.059	1.318	1.711	2.064	2.492	2.797	3.467	3.745
25	0.000	0.684	0.856	1.058	1.316	1.708	2.060	2.485	2.787	3.450	3.725
26	0.000	0.684	0.856	1.058	1.315	1.706	2.056	2.479	2.779	3.435	3.707
27	0.000	0.684	0.855	1.057	1.314	1.703	2.052	2.473	2.771	3.421	3.690
28	0.000	0.683	0.855	1.056	1.313	1.701	2.048	2.467	2.763	3.408	3.674
29	0.000	0.683	0.854	1.055	1.311	1.699	2.045	2.462	2.756	3.396	3.659
30	0.000	0.683	0.854	1.055	1.310	1.697	2.042	2.457	2.750	3.385	3.646
40	0.000	0.681	0.851	1.050	1.303	1.684	2.021	2.423	2.704	3.307	3.551
60	0.000	0.679	0.848	1.045	1.296	1.671	2.000	2.390	2.660	3.232	3.460
80	0.000	0.678	0.846	1.043	1.292	1.664	1.990	2.374	2.639	3.195	3.416
100	0.000	0.677	0.845	1.042	1.290	1.660	1.984	2.364	2.626	3.174	3.390
1000	0.000	0.675	0.842	1.037	1.282	1.646	1.962	2.330	2.581	3.098	3.300
z	0.000	0.674	0.842	1.036	1.282	1.645	1.960	2.326	2.576	3.090	3.291
	0%	50%	60%	70%	80%	90%	95%	98%	99%	99.8%	99.9%
	Confidence Level										

Figure 10.1
T-Table used to find critical value for the tested alpha level (Gerstman, t Table).

The row used is decided by the degree's of freedom, this is equal to N-1. The column considered is equal to the desired confidence level. The t-value in this position is compared to the result's t-value. If the result's t-value is greater than the table's t-value, then the confidence level is met/exceeded. The p-value for it is at least that of the that rows p-value for that tailed test.

Using the table, an alpha level of 0.05 in a one tailed test for 9 degrees of freedom has a t-value of 1.833.:

$$1.833 > 1.798 \text{ - Therefore, the p-value is greater than the desired } 0.05$$

The t-value is less than the table's 1.833. This means the t-value's p-value, is somewhere greater than 0.05.

To confirm this using the spreadsheet tool, the specific p-value can be calculated accurately as:

$$\text{P-Value} = TTEST(S_1, S_2, 1, 1) = 0.05288673969$$

Now the p-value can be compared to the alpha level to determine if their is evidence to reject the null hypothesis:

$$\text{P-Value} \leqslant \alpha \text{ - Reject null hypothesis}$$

$$0.05288673969 > 0.05$$

The p-value is greater than the defined alpha level. This means there is not enough confidence in the evidence and the null hypothesis cannot be rejected (UC24.US).

10.3 User Study Conclusion

From this user study alone, there is not sufficient support for the program's ability to increase arousal in the user. Although the difference in the samples' means indicates there is, it is not considered significantly different. This means it is too likely the change occurs through random chance to be accepted.

The level of confidence in the results is still very good and close to the desired confidence level. The P-value is worked out to be 0.0529 which means there is only a 5.29% chance the difference is caused by randomness. So the results could still be considered to support the alternative hypothesis to a degree.

The user study was far from ideal. There are a number of areas it could be improved in:

1. A much larger sample size would be better. With only 10 samples, the population is far under-represented. This should be at least 30 samples.
2. The physical environment could be improved and further conditions made. An example would be somehow guaranteeing users have not drank caffeinated drinks or exercised prior to the study.
3. A more accurate way of observing the emotional response from the EDA measurements.

The greatest flaw in the user study was in how the EDA measurements were taken. The change observed in the entire EDA signal was used. This included the SCL signal, which is the natural change in the user, irrespective of any emotional response. The desired EDA part to use is solely the SCR signal, which indicates change in emotional arousal.
In the future the next best area to focus would be in extending the EDA measurement ability. By developing the EDA module to include functionality to filter out the SCL signal and identify the SCR signal. The emotional response would be far more accurately recognised.

The aim of the program to increase emotional arousal is suggested to an extent by the results. The issue is whether this was caused by the procedurally generated textures. The emotional responses seen in the stimulus period would likely have been caused as the user is now interacting with a 3D environment program, and not specifically because of the generated textures. To more accurately differentiate any emotional responses from the procedurally generated textures, a baseline period with variable stimuli should be taken.
This would be a period in which the user interacts with the maze environment but static, stored textures are used. After this period, the stimulus period with generated textures is conducted and the results between these two periods are compared.

Chapter 11

Conclusion

This concluding chapter will cover a personal reflection on what was learnt, what the project achieved and areas that could be improved in the future.

11.1 Personal Reflection

The research process was straightforward for many of the project's relevant areas. However the more niche aspects were more difficult to find resources for. An example of this would be in using the ES for the policy in the Q-Learning agent. Both independently were described in detail yet resources covering their use together were hard to come by, and even then they were abstract. This required recognising how to go about combining their use together correctly from seperate resources.

Many technical areas were covered over the whole project. Using the renown OpenGL API with well established methods for creating a 3D environment shows the ability to use practical tools to create software for a certain situation.
Additionally, using documents that cover abstract ideas like Q-Learning and then implementing them in functioning code displays a degree of intuition.
Covering so many different areas was a large issue as it left little time to fully develop certain areas.

Taking on what might be considered an ambitious project humbles the author. They quickly realise it is hard to reach their intended goal especially if insufficient preparation is made. There were a number of pitfalls throughout the process especially concerning the EDA recording method in the user study. However, it does seem to arm one for bigger things ahead in future project were these issues can be avoided.

11.2 Future Improvements

User Study
The user study didn't surpass the confidence level to support the effectiveness of the program. In the future it would be ideal to conduct another user study.
The sample size should be much larger than 10. As many users as realistically possible, however a sample size of 30 would be sufficient. Additionally more control with the users prior and during the test would be desirable.
A stimulus baseline period where the user interacts with the program without generated textures for comparison.

EDA Module
The EDA measurements used in this project didn't identify emotional responses specifically. The signal wasn't filtered and therefore included natural bodily changes in the EDA signal that are unrelated to emotional response. This is likely what caused the decrease in the p-value in the user study, introducing a random influence on the results.
Developing the EDA module further to include functionality covering filtering of the types of signals. This would allow a much more accurate reward signal for the program and the measurements during user studies would be reliable.

Q-Learning Module
In the Q-Learning module, linear function approximation was used to generalise states when assessing their value. Linear function approximation is essentially a local hill climber when optimizing for the problem. For better optimisation over a complex problem space a non-linear approximation method should be used.
A good substitute for a non-linear method would be an artificial neural network (ANN). In an ANN the learning in the weights would be handled by the back propagation process. As ANNs are a blackbox, it would be possible to implement a well established API for an ANN as part of the Q-Function. This could create a more sophisticated learning process that benefits the generation of increasingly arousing textures.

11.3 Concluding Thoughts

This project is built up on a number of well established methods and techniques. At it's core, what would normally be a typical graphical application is incorporated with AI techniques to introduce unconventional learning in the texture generation.
One of the main achievements from this is in the process of gathering EDA information to recognise emotional response, then using AI to train a set of values to create further emotional response in the generated textures. This covers the basis for what could be a framework developed for use in other applications. The framework would be aimed at gathering biofeedback information from potentially numerous sources (not just EDA), then learning to increase the emotional arousal from the user in further generated content. This content could be in any area like audio, 3D models, gameplay features, etc. This would be great to allow anybody who wishes to include a way of identifying emotional responses in users for use in any of the procedurally generated content in their applications. With a generic framework, the developer would only need to be concerned with how they use the trained values for their generated content. The rest would be handled by the framework.

Another thing that has become apparent is in the availability of EDA recording devices. Although there are expensive, professional level devices and software available for industry and medical use. There is not much choice in cheaper devices for EDA recording.
If somebody wishes to take EDA measurements for a small project, they will be looking at prototyping a device they must create themselves using a development board. This can be streamlined and useful for certain projects, however it takes time to develop and can detract from the project's goal.
Ideally a cheap alternative to the higher end plug and play EDA devices should be developed and distributed. Many people are not interested in hardware and just want to plug the device into a USB port and use the measurements received on that port in their program.

Finally the procedural texture generation process in the Fragment Shader introduced an interesting idea in the texture features. The process uses a set of real numbers to influence the presence of patterns, colours and animation rates to produce a huge variety of different resulting textures.
It would seem an effective way of producing new textures from a manageable set of parameters, this could be explored further and greatly improved. How the texture generation uses the texture features could be a whole project in itself to pursue after this one.

Appendices

Appendix A

Further Relevant Concepts

A.1 Arduino

The Arduino IDE and designed micro-controller boards allow quick prototyping of devices which don't require any high level functionality or management like an embedded OS would provide. The requirement of the device in this project is comparatively simple.
Arduino supports the interfaces and small processing power needed to measure EDA and send it to the system running the main program.

A.2 OpenGL Related API's

GLFW (Graphics Library Framework) is a open source, multi-platform library for use with OpenGL for windowing functionality.
Therefore it provides programmers with the ability to create and manage windows the application will run within. This includes input events through keyboard, mouse, etc. (GLFW, 2018)

GLAD (GL Loader-Generator) is an OpenGL Loading Library.
GLAD is a library purely built around supporting OpenGL extensions for the target platform/OS. Required for efficient runtime mechanisms, which is important for OpenGL to perform well. (GLAD, 2018)

A.3 Irrklang

Irrklang is a cross platform sound library. It is considered to provide high level abstraction and therefore is known as a sound engine. The audio engine provides quick audio functionality in a state based system. Sound engines can be created and then audio files loaded into them and played through sequential functions or in dedicated thread controlled through setting the state of the engine in the main thread.

Irrklang is the third party API used in this program for providing the sound functionality. A third party, high level sound engine was opted for in place of more complex/sophisticated low level API. This is because it would detract from the focus of the project to spend time on audio when an accessible sound engine can be used to quickly implement it.
In future development of the program, a more complete audio module could be developed to incorporate the learning process in it as well.

Appendix B

Complete Test Cases

Test Case 1: Collision detection

ID	1
Title	Collision detection
Pre-conditions	Generate environment with walls for all 4 cardinal directions
Test Steps	1. Move camera position to pass North wall. 2. Move camera position to pass East wall. 3. Move camera position to pass South wall. 4. Move camera position to pass West wall.
Expected Results	On reaching each wall, the camera is restricted to continue moving in that direction.
Result	✓

Test Case 3: Electrodermal activity measurement device

ID	3
Title	Electrodermal activity measurement device
Pre-conditions	1. Embedded program flashed onto device. 2. Individual is wearing electrodes. 3. Serial terminal setup to read from correct port.
Test Steps	1. Observe initial measurements in serial terminal. 2. Instruct individual to take a deep, prolonged breath. 3. Observe new measurements.
Expected Results	Check readings are within an expected/reasonable range $[1000 - 100,000\Omega]$ (Fish et al., 2003). Then observe at the time the individual takes a deep breath, the resistance drops notably as physiological reaction to taking a deep breath takes place.
Result	✓

Test Case 4: Procedural generation using pattern features

ID	4
Title	Procedural generation using pattern features
Pre-conditions	Modify program for a debug mode where the texture features can be directly controlled with inputs from the user.
Test Steps	Make controlled changes to each pattern feature then observe the new generated texture.
Expected Results	Setting each pattern feature should show the expected pattern in generated texture.
Result	✓

Test Case 5: Procedural generation using colour features

ID	5
Title	Procedural generation using colour features
Pre-conditions	Modify program for a debug mode where the texture features can be directly controlled with inputs from the user.
Test Steps	Make controlled changes to each colour feature then observe the new generated texture.
Expected Results	Setting each colour feature should show the expected colour change to the pattern in the generated texture.
Result	✓

Test Case 6: Procedural generation using animation feature

ID	6
Title	Procedural generation using animation feature
Pre-conditions	Modify program for a debug mode where the texture features can be directly controlled with inputs from the user.
Test Steps	Make controlled changes to the animation feature then observe the new generated texture.
Expected Results	Setting the animation feature should show the expected animation rate in the pattern to change in the generated texture.
Result	✓

Test Case 7: Establish serial connection

ID	7
Title	Establish serial connection
Pre-conditions	1. Serial device connected to system port N. 2. Port number N is passed as argument from command line.
Test Steps	Start the program and wait for setup phase to complete.
Expected Results	Debug information to console indicates successful established connection.
Result	✓

Test Case 8: Shader program creation

ID	8
Title	Shader program creation
Pre-conditions	1. Source code for Vertex Shader written. 2. Source code for Fragment Shader written.
Test Steps	Start the program and wait for setup phase to complete.
Expected Results	Debug information to console indicates successful creation of the shader program.
Result	✓

Test Case 9: Q-Learning agent learning

ID	9
Title	Q-Learning agent learning
Pre-conditions	1. Source code for Vertex Shader written. 2. Source code for Fragment Shader written.
Test Steps	Start the program and wait for setup phase to complete.
Expected Results	Debug information to console indicates successful creation of the shader program.
Result	✓

Test Case 10: Tile generation

ID	10
Title	Tile generation
Pre-conditions	1. Generate each of the possible tiles equally spaced in the 3D environment. 2. Set debug mode on for full freedom.
Test Steps	Visit each tile and check correctly generated.
Expected Results	Each of the rendered tiles has expected panels on it.
Result	✓

Appendix C

Ethical Documents

Participant Information Sheet (PIS)

Study title
Emotional Response to Procedurally Generated Textures in a 3D Environment

Invitation and brief summary
The focus of the user study is to gather information on the effectiveness of an interactive program to increase emotional arousal in the user.
The participant's biofeedback measurement will be recorded and used in an AI learning technique and be stored for analysis.
The session should take approximately 15-20 minutes to complete.

Involvement
Taking part in this user study will require your Electrodermal activity to be recorded. Electrodermal activity is the electrical properties of skin which changes as a person experiences physical or emotional stimuli. This will require wearing a pair of electrodes which are fitted onto two adjacent fingers. This is completely safe and there is no risk of electrical harm.
The Electrodermal activity will be recorder during a baseline period of 5 minutes. During this period you will be instructed to remain seated in a relaxed position and try not to talk.
After this period, your measurement will be recorded as you interact with the program. This will involve navigating your way through a 3D maze in which new textures will be generated every 10 seconds. This period typically takes 5-6 minutes to complete. During this period you will be instructed to fully focus on reaching the exit of the maze, indicated by white doors.
The electrodermal activity recorded will be help anonymously meaning there will be no link between you and the collection of measurements made for each session.

Further supporting information
The instructor may make additional instructions and requests from you during the session.
All instructions made by the researcher are completely optional to follow. However if the researcher believes your behaviour will impact the reliability of the results, he may decide to restart or end the session with you.
If any external factor disrupts the session and the researcher wishes to restart, provided you still give your verbal consent.
You may ask to end the session at any time or leave without needing to give any reason.

Possible benefits in taking part
Taking part in this user study will give participants a first hand experience in exposure to biometrics used in interactive programs. In the future this level of interaction between user and system may become more powerful in everyday life.
Additionally if participants wish, they can discuss in more detail how the program applies AI learning techniques to the process. This would be very useful for anybody working in a similar field.

Possible disadvantages and risks of taking part
The only health risk that is involved is in participants with Epilepsy. It is possible for the program to generate animations which might discretely flash. Although it isn't to a level that should be of concern, the participant will be informed in case they are extremely sensitive.

Version control
Participant Information Sheet Date: 10/03/2018
Participant Information Sheet Version: V 1.0
If the consent form date and version do not match this PIS. Inform the researcher and request the correct documents.

Figure C.1

CONSENT FORM

Title of Project: Emotional Response to Procedurally Generated Textures in a 3D Environment

Name of Researcher: Harri Renney

Please initial box

1. I confirm that I have read the information sheet dated 10/03/2018 (V 1.0) for the above study. I have had the opportunity to consider the information, ask questions and have had these answered satisfactorily. ☐
2. I understand that my participation is voluntary and that I am free to withdraw at any time without giving any reason, without my medical care or legal rights being affected. ☐
3. (If appropriate) I understand that the information collected about me will be used to support other research in the future, and may be shared anonymously with other researchers. ☐
4. I agree to take part in the above study. ☐

_____	_____	_____
Name of Participant	Date	Signature
_____	_____	_____
Name of Person taking consent	Date	Signature

Copy of consent form can be provided on request.

Figure C.2
The consent form that must be signed by each volunteer in the user study.

Emotional Response to Proceduraly Generated Textures User Study Schedule

Lead Experimenter Details: Harri Renney Harri2.Renney@live.uwe.ac.uk 07711845730

Name	Date	Scheduled Start	Scheduled Finish	Actual Start	Actual End	Signature	Notes

*By providing your signature on this form you are giving your informed consent to the expriment process outlined in the information sheet and consent form.

Figure C.3
The schedule sheet signed by users to agree on a time slot for the session.

Glossary

API An Application Programming Interface is a set of programming tools used for building software applications. 9, 10, 24, 27, 36, 37, 39, 44, 58, 60

EDA Electrodermal Activity is the common term used for all electrical phenomena in skin, including all active and passive properties traced back to the skin. 3, 8, 10, 11, 13, 19, 22, 23, 33, 34, 36, 39, 40, 43, 44, 49, 50, 52, 56–58, 60

ES An Evolutionary strategy is a nature inspired evolutionary algorithm that mimics the process from the Theory of Evolution. Very similar to the genetic algorithm. 13, 14, 22, 23, 27–29, 36, 39, 40, 46, 47, 57

Fragment Shader A programmable shader that equates to the fragment processing stage in the OpenGL rendering pipeline. 9, 10, 15, 21, 23, 31, 32, 36, 41, 42, 44, 46, 58, 63

GLSL The OpenGL Shader Language is a shading language with C like syntax for GPU programming with OpenGL. 10, 41

GPU Graphics Processing Unit used in the creation of images to a frame buffer to output to a display. 9, 10, 15, 23, 27, 29, 31, 36, 37, 41, 44

GSR The outdated term used to describe electrical phenomena in skin. More up to date term is Electrodermal activity(EDA). 10

JUCE JUCE is an open source, cross platform application framework for C++. It provides tools for GUI's and audio applications. 44

MDP A Markov Decision process is a framework used to help make decisions on a stochastic environment. 11–13

microcontroller A computer that occupies a single integrated circuit. This means it must include a CPU, memory and input/output interfaces. 34

Noise In this project noise is the term for the random number generator of computer graphics. 16–18, 23, 31, 32, 42, 43

OpenGL The Open Graphics Language is an open source, cross platform API for 3D graphics rendering. 9, 10, 24, 29, 36, 38, 60

Periodic Function A function that repeats its values over regular intervals. Most known examples are in trigonometry functions like sine. 41

procedural generation A method of creating data algorithmically rather than retrieving it from stored memory. 8, 10, 15, 18, 21, 31

pseudorandom A process that appears to be random but is not. The pseudorandom sequence exhibits randomness but can be controlled to reproduce the same sequences (Usually with an input). 16, 23, 31, 32

Q-Learning An off-policy reinforcement learning technique. Typically used to find optimal solutions to Markov decision processes. 8, 12–14, 22, 27, 28, 39, 40, 44, 46–49, 57, 58

RL Reinforcement Learning is an approach to machine learning that is inspired by behaviourist psychology. Based around using trial and error to learn best actions in certain states. 11–13, 22, 27, 48

SCL The Skin Conductance Level is the part of the Electrodermal activity signal which is influenced by slow variations from factors like hydration, skin dryness, etc. 11, 56

SCR The Skin Conductance Response is the part of the Electrodermal activity signal which is influenced by any emotionally arousing stimuli. 11, 56

Student's T-Test The Student's T-Test is used in statistical hypothesis analysis to determine if there is significant difference between two samples.. 52

Vertex Shader A programmable shader that equates to the vertex processing stage in the OpenGL rendering pipeline. 9, 10, 46, 63

Bibliography

Altman, R. and Kidron, I. (2016), 'Constructing knowledge about the trigonometric functions and their geometric meaning on the unit circle', *International Journal of Mathematical Education in Science and Technology* **47**(7), 1048–1060.

Axelson, J. (2015), *USB complete: the developer's guide*, Lakeview research LLC.

Bellman, R. (1957), 'A markovian decision process', *Journal of Mathematics and Mechanics* pp. 679–684.

Bellman, R. (2013), *Dynamic programming*, Courier Corporation.

Boucsein, W. (2012), *Electrodermal activity*, Springer Science & Business Media.

de Vries, J. (2014), 'Camera'. Available from: https://learnopengl.com/Getting-started/Camera [Online; accessed 09-01-2018].

Douglass, B. P. (2009), Uml for the c programming language., Technical report, Rational Software.

Duffy, E., Greenfield, N. and Sternach, R. (1972), 'Handbook of psychophysiology', *Handbook of Psychophisiology* .

Ebert, D. S. (2003), *Texturing & modeling: a procedural approach*, Morgan Kaufmann.

Elliott, C. (2003), Functional images, *in* 'The Fun of Programming', "Cornerstones of Computing" series, Palgrave.
URL: *http://conal.net/papers/functional-images/*

Fish, R., Geddes, L. and Babbs, C. (2003), *Medical and Bioengineering Aspects of Electrical Injuries*, Lawyers & Judges Publishing Company.
URL: *https://books.google.co.uk/books?id=luNUiqJHdDwC*

Fowles, D. C., Christie, M. J., Edelberg, R., Grings, W. W., Lykken, D. T. and Venables, P. H. (1981), 'Publication recommendations for electrodermal measurements', *Psychophysiology* **18**(3), 232–239.

Gauss, C. F. (1809), 'Theoria motus corporum coelestum', *Werke* .

Gerstman, B. B. (t Table), 'Camera'. Available from: http://www.sjsu.edu/faculty/gerstman/StatPrimer/t-table.pdf [Online; accessed 27-02-2018].

GLAD (2018), 'Glad'. Available from: http://glad.dav1d.de [Online; accessed 09-01-2018].

GLFW (2018), 'Glfw'. Available from: http://www.glfw.org [Online; accessed 09-01-2018].

Gosavi, A. (2011), 'A tutorial for reinforcement learning', *Department of Engineering Management and Systems Engineering* .

Haigh-Hutchinson, M. (2009), *Real time cameras: A guide for game designers and developers*, Morgan Kaufmann Publishers Inc.

iMotions (2016), *Galvanic Skin Response: The Complete Pocket Guide*, iMotions.

jocelynzada (2013), 'Stress makes art: Galvanic skin response and visual generation'. Available from: http://www.instructables.com/id/Stress-Makes-Art-Galvanic-Skin-Response-and-Visual [Online; accessed 09-01-2018].

Karczmarczuk, J. (2002), Functional approach to texture generation, *in* 'International Symposium on Practical Aspects of Declarative Languages', Springer, pp. 225–242.

Khronos (2017), 'Rendering pipeline overview'. Available from: https://www.khronos.org/opengl/wiki/Rendering[Online; accessed 09-01-2018].

Khronos (2018), 'Khronos-group'. Available from: https://www.khronos.org [Online; accessed 09-01-2018].

Mitchell, M. (1998), *An introduction to genetic algorithms*, MIT press.

Moriarty, D. E., Schultz, A. C. and Grefenstette, J. J. (1999), 'Evolutionary algorithms for reinforcement learning', *J. Artif. Intell. Res.(JAIR)* **11**, 241–276.

Perlin, K. (1985), 'An image synthesizer', *ACM Siggraph Computer Graphics* **19**(3), 287–296.

Perlin, K. (2002), Improving noise, *in* 'ACM Transactions on Graphics (TOG)', Vol. 21, ACM, pp. 681–682.

Pitchforth, A. (2010), *Emotional Response to Auditory and Visual Stimuli*, Loma Linda University.

Rechenberg, I. (1973), 'Evolution strategy: Optimization of technical systems by means of biological evolution', *Fromman-Holzboog, Stuttgart* **104**, 15–16.

Rost, R. J., Licea-Kane, B., Ginsburg, D., Kessenich, J., Lichtenbelt, B., Malan, H. and Weiblen, M. (2009), *OpenGL shading language*, Pearson Education.

Ruxton, G. D. and Neuhäuser, M. (2010), 'When should we use one-tailed hypothesis testing?', *Methods in Ecology and Evolution* **1**(2), 114–117.

seeed (2014), 'Grove - gsr sensor'. Available from: http://wiki.seeed.cc/Grove-GSR[Online; accessed 09-01-2018].

Student (1908), 'The probable error of a mean', *Biometrika* pp. 1–25.

Sutton, R. S. and Barto, A. G. (1998), *Reinforcement learning: An introduction*, Vol. 1, MIT press Cambridge.

Tkalcic, M. and Tasic, J. F. (2003), *Colour spaces: perceptual, historical and applicational background*, Vol. 1, IEEE.

Torres, V. P. (2013), 'Development of biofeedback mechanisms in a procedural environment using biometric sensors', *arXiv preprint arXiv:1310.2102* .

Van Den Bergen, G. (2003), *Collision detection in interactive 3D environments*, CRC Press.

van Oosten, J. (2011), *Understanding the View Matrix*, 3D Game Engine Programming.

Woo, M., Neider, J., Davis, T. and Shreiner, D. (1999), *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*, Addison-Wesley Longman Publishing Co., Inc.