# COMP3851: Future Power – Science and Engineering Challenge

Sol Scobie – c3302821

## Background

The Science and Engineering Challenge, SEC, is a nationwide STEM outreach program presented by the University of Newcastle [[4]](#SEC_Website). They run several events supporting STEM outcomes in education. Future Power is one of these events and the project was to take the original demo and create a digital browser-based version of the game. The original game is a wooden game board and “generators” that use microcontrollers to run a game based around teaching the basic idea of balancing a power grid so that the requested and generated power were not too far out of sink and causing the grid to overload and shut down.

A white board with several different colored objects on it

Description automatically generatedThe original game board required the transportation and setup of many unwieldy heavy boards, limiting the reach of the demo and increasing demand on staff. A digital demo alleviates this demand and allows for outreach beyond areas that the boards can be reasonably transported to and from. Embedding the demo in a browser tab using Unity’s WebGL compilation, working with low poly assets and the computationally simple nature of the game allows for low end hardware to run and experience the game reducing a further barrier to entry.

Fig 1: Original Game Board and Digital Reproduction

The project requested the use of the Unity game engine, having experience in other Game Object focused engines, much of the baseline knowledge was already present. Unity 2022.3 was selected as this is a long term support build of the engine, notably this caused issues later due to an unresolved bug in the engine UI since 2021. Unity as a tool has a large community of creators online documenting, tutorialising and sharing their creations online [[1]](#Future_Power_Resources_Playlist) [[3]](#Unity_Shaders_Bible). This community was an invaluable resource when adjusting from other more familiar tools like Unreal and S&box. As an example, many resources were sought out to learn the use of Unity’s Text Mesh Pro user interface building system, and many of the roadblocks in the project stemmed from understanding the methods of receiving input from a user. Alongside these resources, Unity’s documentation hosted directly on their website was an invaluable resource [[2]](#Unity_Scripting_API).

Unity itself is one of the archetypical modern Game Engines. It abstracts away the traditional “main loop” allowing users to operate in a GameObject focused paradigm where objects are given Components, such as Scripts, Camera’s, Listeners, Materials, etc. These Components drive the behaviour of GameObjects through their own update loops or one of the many rendering pipelines implemented in unity, this project utilised the Universal Rendering Pipeline, URP for its compatibility with Unity’s WebGL Build.

Working within Unity’s general setup, the Game Scene was divided into four main categories, the Main Camera and Player Controller. The Starting Menu for selection of which Set of Scenarios from the original game a player will tackle, labelled A or B. Scene Meshes, the collection of unscripted decorative elements, and Game Objects a folder of the interactive elements of the game such as the “Game Board” an object that controls the game state, monitors the players progress, saves and reports the player score and goals. The “Generator Controller” an object that holds references to the four “Generator Nodes” which take player input, calculate their current power output and report to the Game Board when queried. The “Consumer Controller” which holds a reference to seven “Consumer Groups” each of which holds a variable number of consumers that players may or may not be required to power based on the Scenario the Game Board is operating under.

Tools like Unity’s Lighting System, Shader Graph, Animator System, Text Mesh Pro and Unity’s Input System all presented unique Challenges and required external research primarily balanced between Community Creator resources, [[1]](#Future_Power_Resources_Playlist) the Unity Scripting API Documentation, [[2]](#Unity_Scripting_API) and the Unity Shader Bible [[3]](#Unity_Shaders_Bible) a book I had read the year before detailing and teaching the process of shader development with a focus on Unity itself.

To maximise performance on low end hardware all the scene lighting was “baked”, precalculated and saved to texture files to be referenced by the game when rendering. This is not a system I have engaged with extensively in other engines and is at time of writing a rough implementation based on a number of hastily implemented tutorials on Lighting, Skyboxes and Global Illumination.

Shader Graph is a system of graphics programming using “shader nodes” in place of traditional programming which compiles into equivalent HLSL code at runtime. I have used similar systems extensively in Blender and other game engines for texturing and assorted VFX. I had not anticipated issues such as Unity’s default URP shaders not allowing Roughness and Metallic textures to drive the strength of the effect and needed to build my own. Furthermore, Unity’s rendering model operates via a Smoothness value from 0-1, rather than a Roughness value from 0-1, so the exported roughness textures from Substance Painter required the Red channel of the Texture to be inverted by the shader before use else glossy objects would be Matte and Vice Versa.

Unity operates with OpenGL style Normal Maps, while many other engines make use of DirectX Normal Maps. Human error while texturing lead to some normal maps being exported in the wrong mode, this was corrected with a Boolean toggle in the Shaders created.

The Animator system was a familiar process to the equivalent in S&box however the Unity API and Creator resources had to be consulted multiple times to account for erroneous Root Motion placing objects away from their intended locations.

Text Mesh Pro became a core part of the project quickly, taking up a significant portion of research time as understanding the functionality of this often arcane feeling system became frustrating. TMP essentially was the entire Prototype, and remained a core aspect of player input and information presentation in the final version.

Unity has multiple input systems, frustratingly they are not named to be easily differentiable. The “New” input system is the one the project ended up working with and likely made the process harder. Once again community resources and the API were referenced, with particular focus on the exposure to the new concept of using Lambda Functions to assign callbacks to inputAction Events. [[1]](#Future_Power_Resources_Playlist) [[2]](#Unity_Scripting_API) [[3]](#Unity_Shaders_Bible)

## Aims

**The Digital Recreation of a simple game intended to stoke interest in STEM among young people and students.**

***Recreation of the mechanics of the original Future Power game boards*** – Done

Prototype of Game was completed in Sem1

Prototype Functionality was extended to meet the full scope of the game and requested features

Selection of Set A or B scenarios,

Reverse engineered scoring system as algorithm is unknown,

Reset Button to in one click reset an “Overloaded” Game Board

***Switching Input Systems to use event interrupts on player action in place of polling*** – Done

Work around Unity Engine Bugs in “Input System” UI that has existed since 2021

Raycasting for object click detection

Block Raycasting when clicking UI objects to avoid doubling up on inputs

***Creation of fully 3D Scene within which the game is played*** – Done

3D Scene Block Out was completed shortly before Sem 2 began and tweaked repeatedly

3D replication/alteration of Original Game Board

3D modelling room

Texturing Assets

Animating Levers, Buttons and Dials

Shader creation

Standard Opaque Environment Shader

Transparency Shader for Glass and Light Bulbs

Transparency Shader Leaves and Branches on “Tree Animate” model [[6]](#Tree_Animate)

Toggleable Emissive Shader for Generator Dials

Material Creation

Materials for all of the above Shaders for each prop that required one

Toggleable Emissive Material using standard Unity Shaders for consumer switches

*Attaching Game Actions to their 3D visual Representations* – Done

Clicking Generators, opens menu to change Generator Types,

Clicking “Change View” smoothly transitions the camera to the “Consumer” View of the board or the “Generator” view of the board

Levers display their current state to player visually. Blue levers indicate a “required” consumer, Green an active consumer, Red an inactive consumer

Levers are also animated based on this state and triggered to toggle by player click events

Dials make one full rotation when being adjusted from 0% to 100% power generation.

Dials glow green from their custom shader when clicked and held to show they are selected

Dials have had a notch added to show how far through rotating they are

Dials reset glow when click is no longer held and their rotation when the reset button is pressed

Prototype Ui Elements have been repurposed into camera space UI elements and “diegetic” UI elements

Camera Space, “Change View”, “Win Scenario”, “Win Breakdown” and “Reset Game to Try Different Scenario Set” UI elements

“Generator Nodes” show their individual target and current generation

“Game Box” shows total current and requested generation

“Consumer Nodes” show how much generation each individual requests when active

Stretch Goal: Create Sound Effects to pair with the Game – Ran Out of Time

Unexpected delays from the Input System Switch, and time it took to become comfortable working with Text Mesh Pro has resulted in the inability to pursue the necessary research and subsequent development necessary for the creation/ editing/ acquisition of appropriate sound files/background music for the game.

This feature was always considered a stretch goal by myself and my supervisor though being unable to attempt it is disappointing.

## Methods and Results

Development was essentially broken into three different sections of the Game. The Mechanics, primarily focused on Unity’s Scripting but needing to interact with the other two systems. Visuals, 3D models made/animated in Blender, Textured in Substance painter, Custom shaders built in Unity’s Shader Graph and Materials built from Custom and Built in Shaders. And the UI built with the Text Mesh Pro framework, often interacting with the scripting side of unity.

### *Mechanics*

Mechanically the game uses unscripted meshes, scripted meshes, a primary camera and player controller and the various UI elements either tied to world space, gameboard displays, start menu etc.

On game start many objects setup basic properties, e.g. initialising a List<>() of their children. The Gameboard loads no active Scenario and remains inert until the player selects Set A or Set B from the start Menu. These Sets hold Scenarios which are, Scriptable Objects a subtype of unity script often used for setting up data to be used, transferred or stored by other objects before or during runtime. Unity creators, like Sirenx, [[1]](#Future_Power_Resources_Playlist) advocate for their use for many of the use cases in Future Power.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedBecause of that “Sets” hold a List of Scenarios which are themselves individual Scriptable Objects, that hold a number of properties, such as the types of Generators Banned, the number of Consumers that must be provided power and if we require two renewable generators to win the scenario.

Fig X: Set and Scenario Game Object Editor

Other objects like Generators and Consumers reference their own scriptable objects to determine their behaviour and parameters.

The Unity Input System for this game is configured to listen for Mouse Clicks and the Shift Key. These trigger events onPress() and onRelease(), with the Mouse checking if it is hovering a UI element, before doing a Physics Raycast and searching for Objects with the tag “Clickable”, allowing the raycast to ignore irrelevant objects. Behaviour then changes based on the type of object, opening Generator Selections Menus

### *Visuals*

### *UI*

## Ethics

Future Power is intended to be used by minors, primarily students with an interest in STEM. To this end, the Future Power game, records no information about the user, sends no information to the organisation hosting the game or to myself as the creator. Any requirements for systems such as a “log in” would be facilitated by the host of the game and are thus outside the scope of this project.

The game does not include challenging or disturbing imagery.

Future Power makes use of assets not created by myself. It repurposes images created for the original game’s power generators as labelling that are owned by the SEC. Additionally, it makes use of “Rogland Clear Night” an HDRI by Greg Zaal on Polyhaven and “Tree Animate” a collection of three Animated and Textured Tree Models by Node\_λrt on Sketchfab [[5]](#Rogland_Clear_Night) [[6]](#Tree_Animate).

Both are licensed under Creative Commons [[7]](#CC0) [[8]](#CC4) and the Credits Button in the Start Menu User Interface of Future Power provides attribution to both creators including links to their work Fig X.

Fig. X: Future Power Credits Page

Technically CC0 does not require attribution [[7]](#CC0), it is my opinion, however, that not providing attribution to a creator whose work I am directly utilising to enhance my own development process would itself be inherently unethical and thus Greg Zaal is credited regardless of any legal requirements.

All other assets in Future Power that are not a tool directly provided by the Unity Game Engine including 3D models, Textures, Scripts, Materials, Shaders, Animations, etc. are all my own work.

## Project Management

Meetings during Semester two were consistently scheduled once a week, extenuating circumstances often extended the gap between these meetings to be once a fortnight.

When meetings were missed or otherwise significant progress was made shortly following a meeting I would communicate with my supervisor over discord presenting screenshots and recordings of new functionality, discussing aspects of the original game I was unclear as to the functioning of and discussing the best solution to unknown aspects of the original game’s code.

Feedback was helpful in finding the direction desired by my supervisor, however, not being a super technical person details and problem solving with issues encountered was often simply up to me to solve.

## References

[[1]](#Future_Power_Resources_Playlist) Future Power Resources Playlist: <https://www.youtube.com/playlist?list=PLqCGTs5q9fHw-pBzTihXnqnr_kVbD2BiE>

***Rather than linking to every Video I have attached a link to a playlist of the videos I saved and referred against while working.***

How To Deploy Your Unity Game With WebGL | *by BMo*

Unity Shader Graph – How to Update Shaders in Code | *by Rigor Mortis Tortoise*

Radial Menu in Unity Tutorial | *by gamesplusjames*

Unity 3d USING Lerp vs MoveTowards vs SmoothDamp (in 2 Minutes) | *by Royal Skies*

Unity Slerp Visualized – Like lerp, but rotund | *by Tarodev*

How to Animate Characters in Unity 3D | Animator Explained | *by iHeartGameDev*

Raycasts in Unity (made easy) | *by Game Dev Beginner*

Light Your World in Unity – 1 Minute Tutorial | *by HIYU*

Lerp smoothing is broken |*by Freya Holmér*

Shader Basics, … /… /… • Shaders for Game Devs[Part 1/2/3] | *by Freya Holmér*

How to rotate a vector | *by Freya Holmér*

Scriptable Objects: What are they? How do you use them? | *by Sirenix*

How to use Unity’s Input System | *by samyam*

Why You Should Use The New Input System In Unity + Overview | *by samyam*

Unity's New INPUT SYSTEM | Unity Beginner Tutorial 2024| *by SpeedTutor*

My Favourite way to Click on GameObjects | *by VR with Andrew*

Adding a Skybox in HDRP using HDRI in Unity | by Zii

How to replace DEFAULT SKYBOX in Unity - Custom Skybox with 360 TEXTURE | *by GDT Solutions ES*

[[2]](#Unity_Scripting_API) Unity – Scripting API: <https://docs.unity3d.com/2022.3/Documentation/ScriptReference/index.html>

[[3]](#Unity_Shaders_Bible) Espindola, F., Yeber, P. (2022). *The unity shaders Bible: A Linear Explanation of Shaders from Beginner to Advanced*.

[[4]](#SEC_Website) Science and Engineering Challenge – UON: <https://www.newcastle.edu.au/college/engineering-science-environment/education/science-and-engineering-challenge>

[[5]](#Rogland_Clear_Night) Greg Zaal (2024) Rogland Clear Night *used in Future Power with credit under CC0 Attribution license [7]*

[[6]](#Tree_Animate) Node\_λrt (2022) Tree Animate *used in Future Power with credit under CC Attribution license* *[8]* <https://sketchfab.com/3d-models/tree-animate-f0f9eb5e6c104bbb8e1f41c97019e6f2>

[[7]](#CC0) CC 0 1.0 Universal <https://creativecommons.org/publicdomain/zero/1.0/>

[[8]](#CC4) CC Attribution 4.0 International <https://creativecommons.org/licenses/by/4.0/>