PRIFYSGOL
BANGOR
UNIVERSITY

School of Computer Science and Electronic Engineering
College of Environmental Sciences and Engineering

AI-Assisted Texturing for Unity Game Development

Maximilian Clarke

Submitted in partial satisfaction of the requirements for the
Degree of Bachelor of Science
in Computer Science

Supervisor Dr. Llyr Ap Cennydd

May, 2023

Acknowledgements

> *AI is neither good nor evil. It's a tool. It's a technology for us to use.*
>
> — **Oren Etzioni**

Dr. Llyr Ap Cennydd (Supervisor)

Statement of Originality

The work presented in this thesis/dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Student:

Maximilian Clarke

Statement of Availability

I hereby acknowledge the availability of any part of this thesis/dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein. I further give permission for a copy of this work to be deposited with the Bangor University Institutional Digital Repository, and/or in any other repository authorised for use by Bangor University and where necessary have gained the required permissions for the use of third party material. I acknowledge that Bangor University may make the title and a summary of this thesis/dissertation freely available.

Student:

Maximilian Clarke

Abstract

In the past decade, artificial intelligence, perhaps beyond all other forms of technological development, has made leaps and bounds in terms of its advancement, availability, reliability, variety of application, and ease of use. One of the more recent applications of AI has been the generation of completely unique images through text prompts - among the first AI text-to-image models made widely available was Stable Diffusion which was publicly released online in August 2022 and is free to use.

One area that has seen very limited use of creative AI models such as Stable Diffusion is in games development. For new game developers especially, the process of finding or creating textures for their games can be a very gruelling process, especially if these textures need only to serve as placeholders until a time when the developers have more time and resources to improve them - these developers may need to either search the internet for adequate royalty-free textures that serve both their intended visual and dimensional requirements or painstakingly create their own textures using various external software, or else, leave their game objects blank.

In this project a plugin has been developed that aims to introduce a new option for those who may find themselves in the aforementioned situation. The plugin will be developed for use within Unity and will consist of a tool that displays a window similar to the interface that can be seen on the Stable Diffusion webpage, this tool will allow the user to type anything they wish into a text box, and the tool will use the Stable Diffusion model to generate completely unique images using that text-prompt, the user can then import these images into Unity to use as textures in their game, without having to leave the Unity application at all.

Contents

List of Figures

Chapter 1

Introduction

The means of developing games has become increasingly accessible in recent years to even the most unqualified of aspiring game developers, thanks to improvements in general hardware and the availability of game engines such as Unity and Unreal. Texturing in engines such as these, however, may prove to be difficult for unexperienced users or those who do not have the ability or time to create their own textures, therefore, by using state of the art AI text-image models in tandem with the capabilities of these game engines, it may become a foreseeable reality in future where the developer never need leave their application when accessing suitable textures for their game models.

In this initial section, the key aim for the project will be presented in detail, followed by any objectives thereof, for which the aim will require. And finally, a dissertation overview will be provided detailing where in this paper each of the objectives will be undertaken.

1.1 Aims

The overall aim for this project is to develop a Unity-based plugin that will invoke a local build of Stable Diffusion to produce images based on a user-prompt, this prompt would be typed into a text-input box provided in a simple UI window within Unity, an assortment of parameters will also be included in the UI for the user to tweak their desired output, the finalised image will be displayed in an output section of the UI, as well as an import button so that the user can easily transfer the output image of their choice directly into

their Unity assets. From this, it is hoped that further development may be encouraged in areas such as this in future, to provide more sophisticated, yet easy to use AI-assisted tools for game developers.

1.2 Objectives

In order to achieve the aim of this project, the following objectives will need to be considered:

- Conduct a thorough Literature Review that explores the backgrounds of Stable Diffusion and Unity, what they are, how they came about, how they work, and what they are used for.

- Investigate whether the integration of both the Unity Engine and Stable Diffusion is feasible, and if so, how this can be facilitated.

- Inquire into related work that similarly endeavours to use AI to simplify the means of using textures in the games development process.

- Develop within Unity, a plugin that provides an efficient and accessible solution to the problem.

- Carry out a suitable user-feedback evaluation of the plugin with a game developer.

1.3 Dissertation Overview

Each of the objectives will be covered over four distinct chapters in this dissertation. There will be a Background chapter focusing on the Literature Review, which will provide research on both Stable Diffusion and Unity Engine, explaining the history, associated technology, and contemporary use of each, as well as investigating their potential mutual integration, and finally concluding the chapter with a section looking into related work. Following this, the Design chapter will layout the necessary research and overall design considerations for the plugin. Next, the Implementation chapter will detail the

development of the plugin, as prescribed by the Design chapter. Lastly, the Evaluation chapter will present a user test, showing the overall performance and functionality of the plugin through the experience of a game developer, allowing for crucial feedback to be deliberated, and to use for improvement of the project in future.

Chapter 2

Background

In this chapter, a background analysis of Stable Diffusion and Unity Engine is done so that a better understanding of the capabilities and potential of each can be developed, as well as a look into whether they can functionally work in unison as this project requires. It will also review related works that similarly attempt to provide AI-assisted texture implementation tools for game developers.

2.1 Background Analysis of Unity Engine & Stable Diffusion

To ensure a thorough understanding of the primary tools that will be used for this project, it will be necessary to analyse key contextual information about each from relevant sources.

2.1.1 Unity Engine

The Unity Engine is a cross-platform game engine that allows game developers to build high-quality 2D and 3D games. It has become one of the most widely used game engines in the world thanks to the fact that is a professional-grade engine for game development - as of 2019, it was the second most utilised game engine on Steam, being used for 13.2% of all titles, behind Unreal Engine at 25.6%[8] - while also being one of the most accessible engines for new developers, often proclaimed as the best game development tool for beginners[5].

Unity began as an online collaboration project between Nicholas Francis and Joachim Ante in 2002. Francis, who lived in Copenhagen at the time, was developing his own game engine, when he asked for help building an open source shader-compiler through an OpenGL forum. His post received a response from Ante, who was a student in Berlin, and they soon were building the shader-compiler part-time together. Eventually they met in person and agreed to combine efforts and start a game studio, as well as creating a robust, licensable tech infrastructure through the merging of both of their own game engines, which would later become the Unity Engine. Soon they would involve David Helgason, whom Francis had worked with prior since they were in high school. After 3 years of development, and a €25,000 investment, the team released the first version of Unity in 2005, targeted primarily for Mac OS X. Over 15 years later, Unity is now broadly used by Windows and Mac users alike in developing some of the foremost games on the market, with immensely popular titles such as Rust, Subnautica, Beat Saber, Cuphead, and many more, built using the Unity Engine. As of 2020, Unity boasts 2 billion active users and 1.5 million creators using their engine monthly[1].

One of Unity's most powerful properties is the way it simplifies the development workflow, whilst retaining the possibility of great complexity within projects. This is evident in the visual editor, which allows developers to create and modify game objects, scenes, and environments using a visual interface and inspector, without the need for coding. Yet, this editor seamlessly integrates with Unity's script functions, allowing the user to fully control the functionality of their games. For scripts, Unity supports a wide range of programming languages including C#, JavaScript, and Boo, allowing for optimal compatibility and great flexibility for developers. Figure 2.1 shows the Unity Editor's scene view.

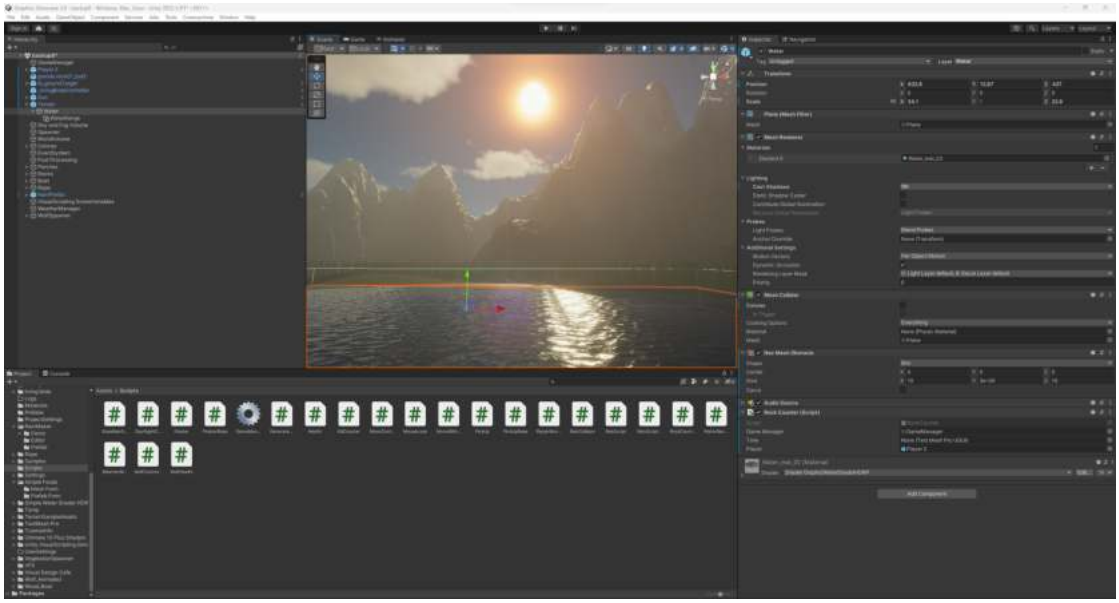


Figure 2.1: The Unity Visual Editor

Unity's accessibility is among the key reasons for its immense success. As well as having a comprehensive and straightforward user-interface, Unity has published an abundance of official learning resources, including in-depth lessons, video tutorials and starter projects on their website, this is so they can introduce the basics of game development to even the most unexperienced of users.

Overall, due to its extensive popularity, as well as its flexibility, compatibility and stability, Unity will be an ideal environment for the development of an experimental tool, such as this project. Since this project will be aimed mostly at unprofessional developers or small studios, it would be suitable to develop within Unity since it is often the engine of choice for these users. Unity also crucially supports the creation of plugins that allow the user to alter the engine interface and functionality extensively, it also provides informative documentation and tutorials on this subject that will be very useful throughout development. This ultimately, is the method through which Stable Diffusion functionality will be introduced to the Unity developer environment. Figure 2.1 shows the typical Unity Editor environment.

2.1.2 Stable Diffusion

Stable Diffusion is deep-learning, text-to-image, latent diffusion model, developed by RunwayML and the CompVis group of Ludwig Maximilian University of Munich, while being guided and funded by Stability AI. It was publicly released online in August 2022, showcasing its ability to generate completely unique images based on user text-input.

It employs a diffusion-denoising algorithm that allows for the generation of images from scratch or the modification of existing images. The model incorporates the user text-prompts that describe elements to be included or omitted from the output, these elements are the result of a machine-learning algorithm that is trained on a large dataset of images. By adjusting various parameters such as sampling types, output image dimensions, seed values, and inference steps, users can guide the image synthesis process. Stable Diffusion also supports image modification techniques such as the addition of noise to existing images, and also the ’inpainting’ and ’outpainting’ of images, whereby some aspects of these images can be modified by restoration or expansion beyond image boundaries respectively. The model’s capabilities can be further enhanced through the use of front-end implementations, which provide additional features such as emphasis markers and negative prompts. Figure 2.2 shows a diagram of the Stable Diffusion algorithmic process.

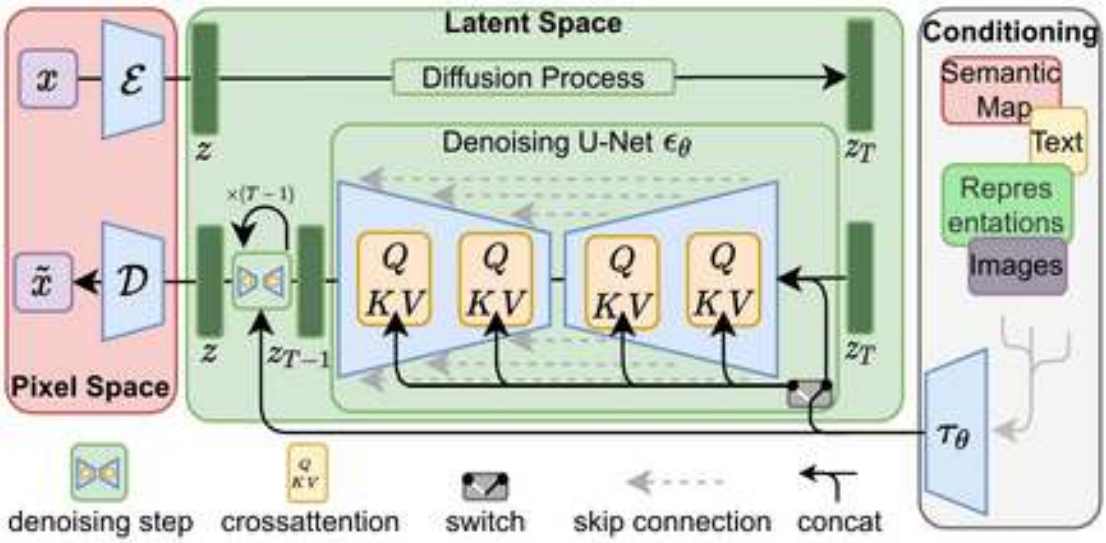


Figure 2.2: Diagram of the latent diffusion architecture used by Stable Diffusion

The success of Stable Diffusion has led to the development of a vast diversified range of similar diffusion-based text-to-image generators, such as DALL·E, Midjourney, and NightCafe that all offer their own 'style' of the same model, some being more advanced and well-trained than others. That said, Stable Diffusion would be the most suitable diffusion-model for this project, more so than other services, since it is one of the only open-source text-to-image models that can be downloaded locally, making it ideal for the modification of code if necessary, for quicker and more reliable processing speeds, and for final packaging and distribution.

2.2 Unity & Stable Diffusion Interoperability

Due to Stable Diffusion being open-source, in culmination with Unity's extensible architecture, mutual integration should be possible.

To demonstrate an example of how a potential functional plugin may operate:

- A plugin script calls on the local build of Stable Diffusion.
- Accesses the parameters of that build, including the text-input and the modifiers.
- Sends the parameters that have been completed by the user within the Unity plugin to the Stable Diffusion build.
- Stable Diffusion build accepts the parameters and processes the request.
- Image(s) are generated and the full files are sent to the plugin interface to be viewed by the user.
- The user then has the ability to edit the image(s) using some tools within the window.
- The user then is able to import the desired image(s) into their project.

2.3 Related Work

In order to develop a sufficient understanding of the problem this project is attempting to solve, it is important to analyse other solutions to similar issues, so that any strengths and weaknesses pertained to them can be learned from and incorporated into this project where possible.

2.3.1 Dream Textures - *Carson Katri*

Dream Textures is a free to use, open source plugin for Blender that uses Stable Diffusion's text-to-image functionality in order to generate unique textures for models. In this plugin, Stable Diffusion is hosted locally and can, therefore, be used offline, and, in terms of processing speed, is only dependant on machine hardware, rather than server reliability. It also has a multitude of features that allow for harmonious implementation of textures onto models, such as a feature called 'Texture Projection', which uses the depth of a model to create a suitable texture for it. Another of its features that may merit a positive experience from the user, is the ability to toggle seamless textures, which will be particularly useful for larger models, whereby the edges of the textures at maximum resolution will be noticeable. It also has an upscaling feature which allows the Stable Diffusion AI to automatically increase the resolution of a texture up to four times, allowing for much better visual quality and fidelity[6]. Figure 2.3 shows the website where Dream Textures can be downloaded.

Whilst Dream Textures achieves its goal of providing high quality AI-generated textures for game developers to use for their models, a potential drawback could be that it can only be utilised within the Blender application, while popular, it is usually only used by experienced game developers who are able to create models for themselves, rather than using built-in options that engines such as Unity provide. This may be an issue for the large number of Unity users who are unexperienced or beginners, and will most likely only use either the basic built-in models provided by Unity, or models available for download on the Unity Asset Store. This criticism is remedied somewhat due

to the plugin being free to download, however, it is still mostly limited in its potential reach of users due to the platform it is made for. Figure 2.3 shows the Dream Textures webpage where users can download the software.

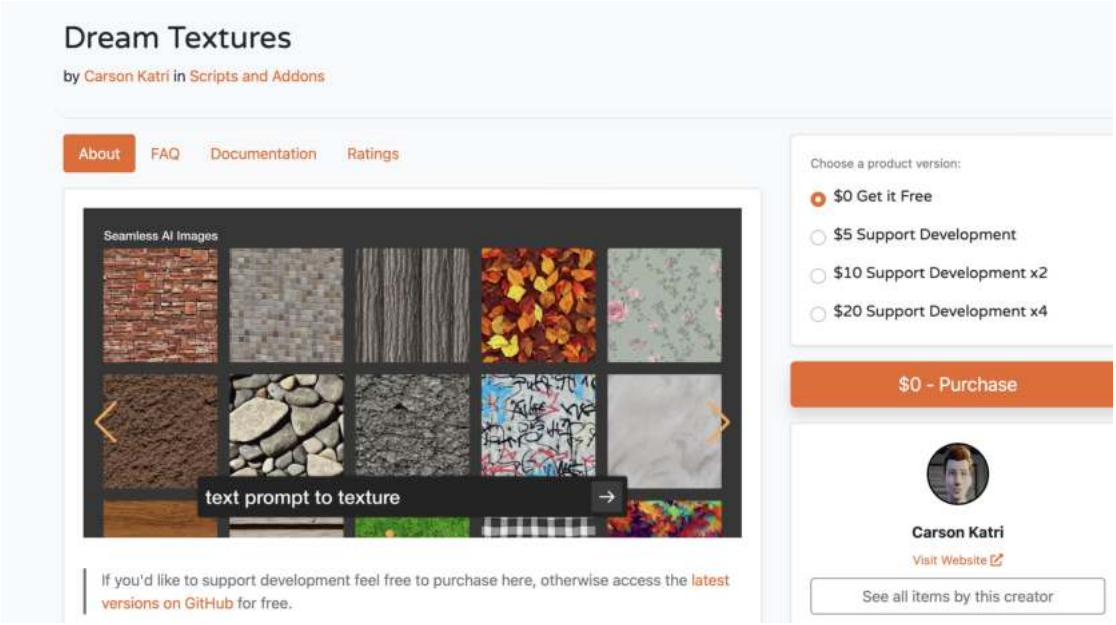


Figure 2.3: The Dream Textures Webpage

Overall, by analysing the strengths and weakness of this plugin, it is likely feasible to incorporate the functionality of seamless textures in the project to enhance the quality of repeated textures. It may also be useful to be wary of making the plugin overly technical and complicated, since it is a primary goal of this project to cater more to the casual users of Unity.

2.3.2 Pixela.ai - *Samyam*

Pixela.ai is a free online repository that stores AI generated textures produced by Stable Diffusion. It allows the user to search through a large number of pre-generated textures, for any that may be suitable to their needs. It includes features such as the ability to favourite textures so that the user can find a specific texture again without needing to download it, and it has a 'New Random Textures' button that shuffles the textures that are shown by default when clicked, allowing for the user to view random textures they may like to use if they are unsure of the type of texture they want[7].

The website relies on the community to upload their own images that they generate with Stable Diffusion so that others may use them, this allows for an almost unlimited number of possible variations of images available to the every user, limited only by the number of users who choose to upload their images. The fact that Pixela.ai is completely free to use, will be beneficial to all users since they all have access to the same content. There is also the benefit of using these textures without concern over copyright infringement since all the textures are AI-generated and unique, and therefore not owned by anyone. Figure 2.4 shows the Pixela.ai website where users can access all the AI-generated textures.

However, a clear disadvantage to Pixela.ai is that it has to be used outside of the game engine, therefore it is not an efficient way of acquiring textures in the sense that external programs have to be used. The textures are also not directly editable, since they are pre-generated, there is no way to modify the AI parameters or undergo post-generation editing within Pixela.ai itself, whereas this can be done with Stable Diffusion extensions.

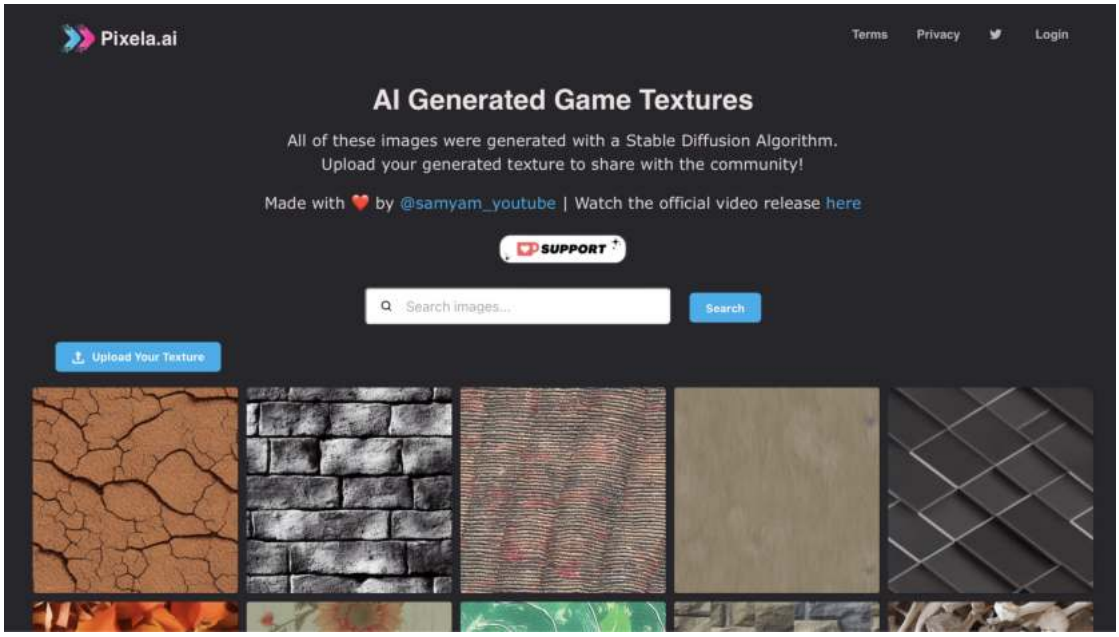


Figure 2.4: The Pixela.ai Webpage

Through the analysis of a tool such as Pixela.ai, it can be inferred that aspects such as allowing a repository for pre-generated textures may be beneficial even for a Unity-based plugin; if a user can save textures that are generated

within the plugin to an online database, and access to that repository is given to all users, it may be useful either for those whose machines are not capable of running a GPU-intensive program such as Stable Diffusion, or those who simply require quick access to basic AI-generated textures, to be able to import textures generated by others. The negative aspects of Pixela.ai would invariably be omitted due to the fact that a live version of Stable Diffusion will be used by the plugin, and the plugin will be Unity-based.

2.4 Summary

To summarise this chapter, a background analysis of both Unity and Stable Diffusion has been completed, it is clear that Unity is the most favourable option for game engine and developer environment in the case of this project, due to its widespread popularity amongst a vast range of different users, including professional developers, as well as beginners, this is beneficial since the plugin has an opportunity to reach a large and diverse portion of all game developers. It is also important that Unity has extensible architecture, which will facilitate smoother integration of external programs such as Stable Diffusion as well as allowing for the possibility of increasing the features of the plugin.

The background analysis has also shown that Stable Diffusion is the most ideal text-to-image AI model mainly due to the fact that it is open-source and, therefore, can be downloaded in bulk and used locally, eliminating the need for reliance on servers, and allowing the ability to use the plugin offline. Due to the potential interoperability between Unity and Stable Diffusion, this chapter has also provided a prospective process that the plugin could follow. This chapter has also reviewed comparable solutions that also aim to give game developers AI-based tools for texturing by weighing the strengths and weaknesses of each of these solutions, in order to learn from and improve the concept of this project.

The following chapter will cover the Design process that will be integral to the development of this project. the operational process that was suggested in

section 2.2 will be considered and refined in accordance with any technical requirements.

Chapter 3

Design

The research has shown that generating unique textures with AI can be incredibly useful for all kinds of game developers. The Design chapter lays out the fundamental and potential requirements, the options and parameters the plugin will utilise, the GUI designs, as well as a simple prototype.

3.1 Requirements

Development of the Stable Diffusion plugin for Unity must focus on delivering an efficient, user friendly interface. It is clear from the research and analysis in the Background chapter that an integrated AI texture generator within the Unity application would benefit the vast majority of Unity users, from the professionals to the beginners. By persevering with this goal, it will allow users to enjoy a much more efficient and productive workflow when developing their games, since they will not even need to leave the Unity application to acquire the textures they need. This plugin will aim to be as user-friendly as possible to allow beginner developers to find the textures they need using the plugin, with minimal external assistance.

3.1.1 MOSCOW Requirements

The necessary as well as the potential features and requirements will be provided using the MoSCoW method. It will detail what the project Must include, Should include, Could include, and Will not include.

In order to deliver on the fundamental goals of the project, the Stable Diffusion plugin for Unity MUST:

- Allow the user to send a text-input prompt to a local build of Stable Diffusion.
- Allow the user to generate AI textures without the need to leave the Unity application.
- Allow the user to alter the Stable Diffusion seed.
- Allow the user to alter the Stable Diffusion steps.
- Allow the user to import the generated images into their Unity Assets.

These requirements must be met, otherwise the project will not be successful in its objective.

The following requirements show what the project should include. It SHOULD:

- Allow the user to toggle seamless textures.
- Allow for the editing of an initial image by Stable Diffusion to produce alterations to the original.
- Allow the user to alter size of the textures.

These requirements, while not completely necessary to deliver a functional plugin, they would significantly improve the user experience and should be included if possible.

The next requirements present what the project could include. It COULD:

- Remember previously generated textures.
- Include advanced editing tools after generation.
- Include an online database for community sharing of textures.
- Allow for the altering of the Img2Img strength.
- Allow the user to change the number of images generated.
- Allow the user to change the sampling method.

These requirements could possibly be included, but only if the other features function optimally.

The final set of requirements are aspects that will not be developed for this dissertation project. This project WILL NOT:

- Include a PBR texture generator.
- Include a ’Texture Projection’ function.
- Include an upscaling feature.

These requirements will not be included in this project, either because they would require too much use of external applications, or they would not offer any important functionalities to the plugin.

Now that the requirements for this project have been clearly set, both the necessary and possible UI parameters need to be defined and explained clearly.

3.2 Stable Diffusion Plugin UI Parameters

This plugin will require a number of options in the user-interface to be available to the user so that they can change the texture output to what they need. All of these parameters were referenced in the MoSCoW requirements in section 3.1.1.

3.2.1 Seed Parameter

The seed parameter will control the number that Stable Diffusion uses to initialise the image generation. It will be set to -1 by default so that the seed will be random every time Stable Diffusion runs, but the user has the ability to select what seed number they want to use. If the user inputs a specific seed value such as 50, as long as the seed number is 50 Stable Diffusion will produce the same image, given all other parameters remain the same.

3.2.2 Steps Parameter

The number of steps denotes how many times the Stable Diffusion algorithm iterates, every step removes more noise from the image, and so essentially the more steps the AI takes the more ’guesses’ it makes about what the

image ’should’ look like based on the user-input, therefore it is normal to see an increase in quality the higher the step value, until any changes become almost unnoticeable around a value of 100. Increasing the step value would directly increase the performance cost due to the the algorithm processing time increasing linearly with the step number.

3.2.3 Seamless Parameter

When the user enables the seamlessness parameter, the edges of the output image will be rearranged so that opposite sides appear to be continuations of each other. This will be useful for users who need to texture large objects such as the ground or sky, this parameter will ensure that there is not visible tiling occurring in the texture.

3.2.4 Size Parameter

The size parameter will give the user a number of image size options to choose from, these will be 64x64, 128x128, 256x256, 512x512, and 1024x1024. The higher the image size the higher the computational cost, and therefore the longer the processing time.

3.2.5 Img2Img Parameter

The Img2Img parameter will be a slider which will only be used when another image is being used with a prompt to generate additional textures on that existing image. The parameter controls the strength of the blending of the new texture created by the prompt and the old texture that is being used.

3.2.6 Image No. Parameter

The image number parameter will control how many images Stable Diffusion will produce from a single prompt, both images will be set to different seeds and so will always be different. The higher the image number, the higher the computational cost due to the fact that Stable Diffusion has to perform the algorithm the number of times requested by the image number.

3.2.7 Sampling Method Parameter

The method parameter will inform Stable Diffusion what sampling method the user wants it to use. The sampling method being used will change how Stable Diffusion handles its sample data, for example, some methods may improve precision of the algorithm in matching the output image with the prompt whilst others may prioritise speed more so than quality. Giving this option to the user will allow the user to modify their textures further.

3.3 Paper UI

To develop an initial idea for what the user interface of the plugin will look like, a rough sketch was made showing the interface and all the essential parameters and buttons. With a UI similar to the one presented in Figure 3.1, the plugin will meet its Must requirements, as explained in section 3.1.1.

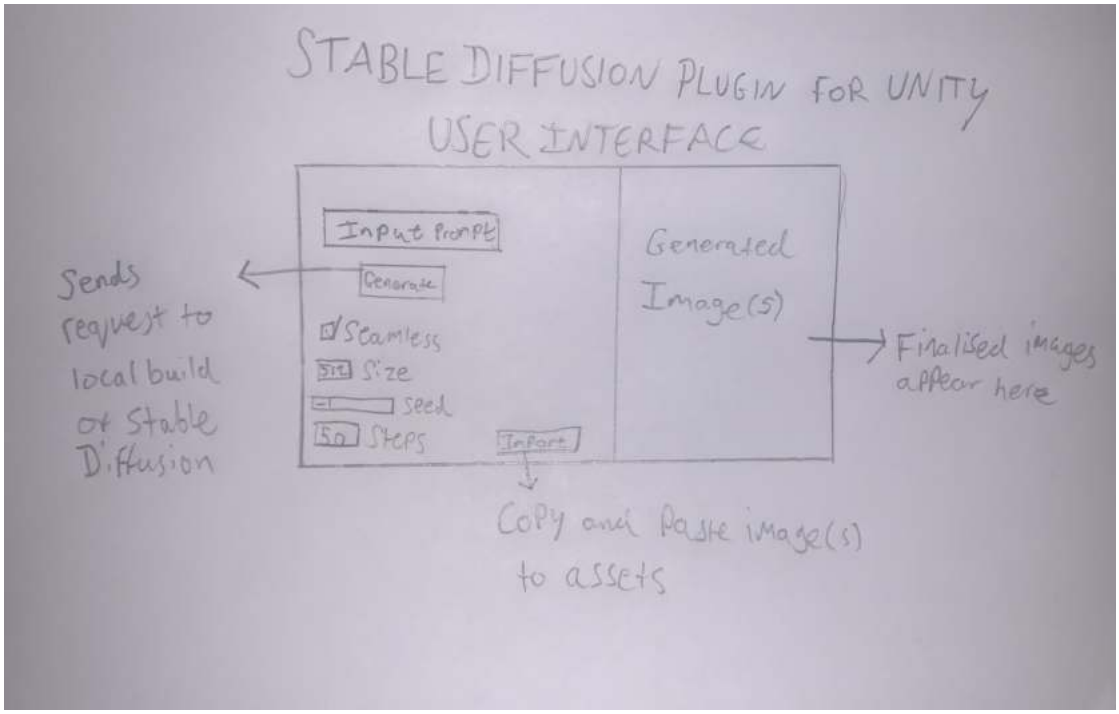


Figure 3.1: A rough sketch of the plugin's UI

Some aspects concerning this design may need to be considered, such as whether it is 'user-friendly' enough for unexperienced Unity users. One way the UI can be simplified and thus, made easier to understand for users in question, is by including some of the non-crucial and complicated options in

an ’advanced settings’ dropdown menu, thereby leaving the UI less cluttered with technical terms that may be difficult to understand by the average user, whilst still giving the option for more advanced parameters to those who wish to use them. The fact that there may be more parameters added to the final design, makes this a more relevant issue to consider.

3.4 Computer UI

The next UI design was made in Photoshop and was made in the style of the Unity Editor interface. The same parameters are present but the more technical or less relevant ones are hidden behind the ’advanced’ dropdown menu.

Figure 3.2: A computer illustration of the plugin’s UI

Figure 3.2 shows some relevant changes with respect to Figure 3.1, including the use of an ’advanced’ dropdown menu, in order to declutter the UI while retaining full parameter functionality. The computer UI also demonstrates what the plugin may look like with four generated images instead of one, with functionality that allows the selection of a specific image, for which the user

can then choose to import that single image, or multiple, depending on how many are highlighted.

3.5 Running Stable Diffusion Locally

This project will require a build of Stable Diffusion to be downloaded and hosted on a local server so that Unity can send web requests in order to interact with it. The build of Stable Diffusion that will be used will be InvokeAI[3], since it is able to run a web program with Stable Diffusion on the localHost server. The plugin will be written in C# within the script of the plugin UI, the plugin will use the HTTP POST function to send the necessary data to the localHost server, Stable Diffusion will accept this data and generate the image, the plugin UI script will then have a function that will retrieve the image from the server and then display it within the UI window of the plugin.

3.6 Prototype

A prototype version of the plugin has been developed using Unity and Microsoft Visual Studio 2022. The purpose of the prototype is to create the initial basic plugin for the Stable Diffusion functionality to be added to. Figure 3.3 shows the user interface that was developed in the prototype.

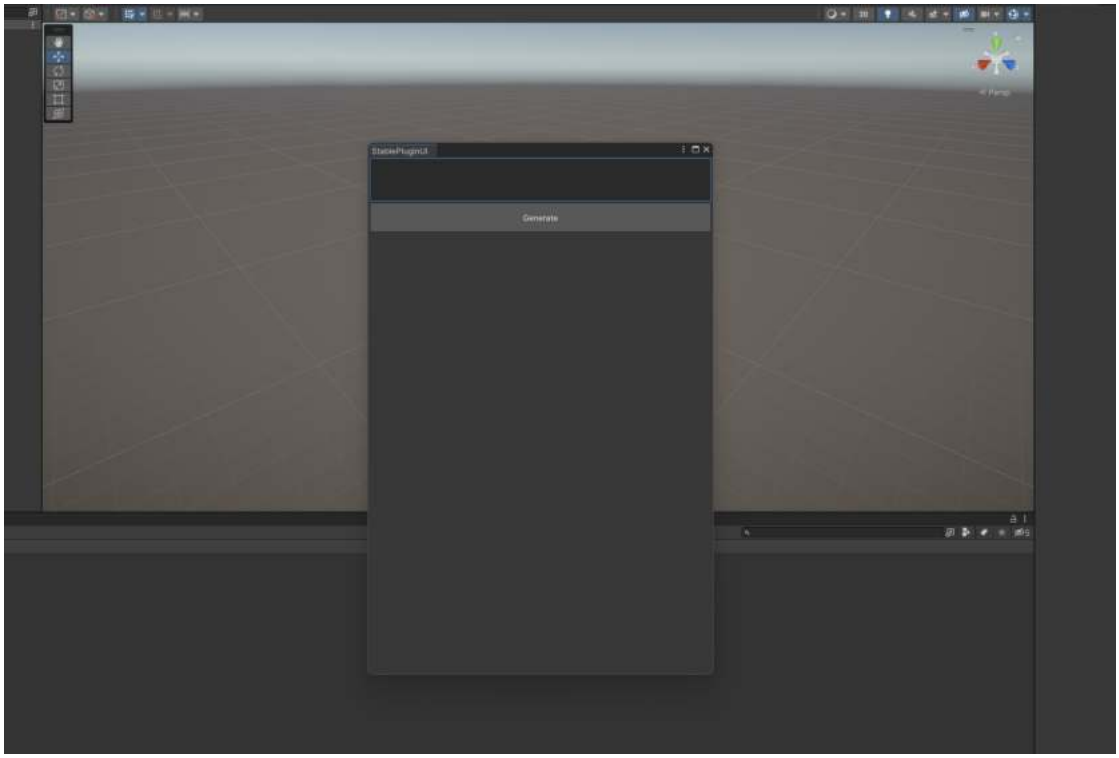


Figure 3.3: The plugin UI window for the prototype

The prototype includes a simple window that can be accessed from Tools/StablePluginUI in the Unity Editor. There is a text box which will be used for the user-prompt, there is also a functioning 'Generate' button that will be used to send the parameter and prompt data to the Stable Diffusion local build.

An aspect of the prototype that differs from the initial designs, is the decision to make the window longer vertically rather than horizontally because it was noticeably obstructing some of the scene editor more so than if it was horizontal, in the final implementation, the generated images will appear beneath the prompt-box.

3.7 Summary

To summarise the Design chapter, the key requirements have been defined using the MoSCoW method in section 3.1.1, from this the project will be able to aim to achieve the most necessary features primarily so that it will be successful in its goal. This chapter has also outlined and defined the individual

parameters that can be included in order to give the user greater control over the result of the plugin. The design for the plugin UI has also been looked at, with designs such as those shown in Figures 3.1 and 3.2, this gives the project a good starting point for which to improve the plugin's visuals as well as some functionality. This chapter has discussed the potential method for hosting a local build of Stable Diffusion on a server, allowing for interaction between it and Unity. Finally, a basic prototype was developed to experiment and improve on the previously given designs, and to also develop the basic structure of the plugin.

To deliver a successful final version, the implementation must include a functioning UI script that can send and receive requests from localHost so that Unity and Stable Diffusion can easily exchange data using HTTP POST, it must also have included options in the UI for each of the necessary parameters for the user to interact with, implementation of a display for the image needs to also be included so that the user can see the image once it is downloaded, and finally, a functioning UI button must be included that can import the generated image into the user's assets.

Chapter 4

Implementation

The final implementation was carried out mostly in Unity, since most of the functionality for this project will be contained in a single Unity UI window. The programming was done in C# using Microsoft Visual Studio 2022.

4.1 Plugin UI

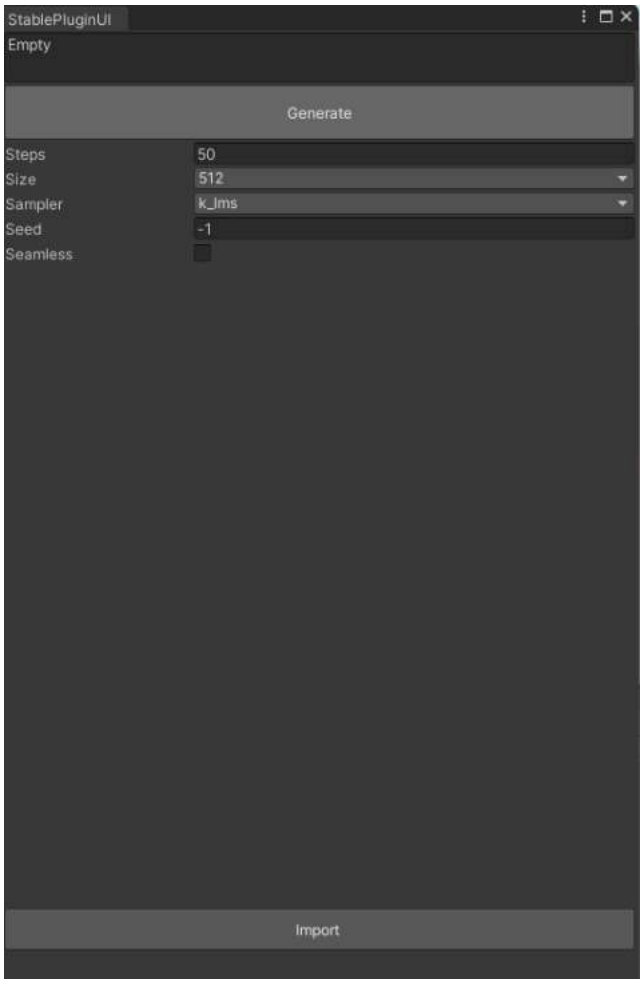


Figure 4.1: The final UI window design

The plugin's UI, shown in Figure 4.1, has been implemented predominantly using Unity's EditorGUILayout class, this allowed the UI to include the buttons and parameters required by the Design chapter. The UI includes 5 sections:

- The first section is the text-input at the top of the UI, where the user can input their prompt that will be sent to the local server where Stable Diffusion is hosted.

- The second section below the input box is the 'Generate' button where the user can click to submit their prompt to Stable Diffusion.

- The third section is where the parameter modifiers are located, there are 5 parameters included in this finalised version:

  - There is an integer-input box where the user can specify the number of steps they wish Stable Diffusion to use.
  - There is a dropdown menu displaying 6 image size presets, each option is displayed as a single number because the width and height of every image will be identical, the options are 32, 64, 128, 256, 512, and 1024 - the values are in terms of pixel count.
  - The third parameter is to allow the user to change the sampling method Stable Diffusion utilises, the default will be k_lms which is the standard method used by Stable Diffusion. By giving the user the option to change the sampling method, it will increase the potential differentiation of output images, and will give them more control over the style of each texture they generate.
  - The fourth parameter is another integer-input box where the user can input any number to change the seeding that Stable Diffusion uses - set to -1 by default which will automatically use a random seed.
  - The fifth parameter is the toggle option to make Stable Diffusion generate a seamless texture, this will help the user to apply textures with the intention of smooth repetition.

- The fourth section is the area that will display the generated image that has been received by the plugin, sent by the localHost.

- The fifth section is the 'Import' button that, when clicked, allows the user to save the generated texture into their assets - it will save them into a folder named 'Textures', if there is no folder named this in the user's assets, the plugin will make one.

4.2 Stable Diffusion on LocalHost

One of the more crucial tasks for the plugin was the hosting of a local build of Stable Diffusion for Unity to access. This was achieved by downloading a version of the Stable Diffusion software that has been configured for tasks such as displaying the AI functionality on web servers[4], called InvokeAI. With the InvokeAI package downloaded, Stable Diffusion will need a model containing a large collection of images to use for its algorithm, this 7GB package was downloaded from a site called Hugging Face[2]. Using a program called Anaconda, InvokeAI was linked with the model package and was eventually able to start running successfully on LocalHost.

Figure 4.2: Stable Diffusion interface displayed on the localHost server

4.3 StablePluginUI Script

The StablePluginUI script is where most of the functionality of this project lies, each segment of code can be broken up into six sections.

The full list of figures containing the complete scripts for the project can be seen in Appendix A.

4.3.1 ShowWindow Class & Initialisations

As Figure A.1 shows, the plugin script began with a class made to create a window that can be selected in the Unity Editor under Tools/StablePluginUI. Following this are the variable initialisations, which define most of the elements that are used in the HTTP POST requests, as well as other functions.

4.3.2 GUI Class

The next segment of code as shown in Figure A.2 provides the values that determines the layout of the UI window, there are variables for both the 'Generate' and 'Import' buttons, the parameter options, as well as the output area for the generated texture.

4.3.3 Texture Import Class

Figure A.3 shows the class that controls the 'Import' button's functionality. The class has a function that, when clicked, it looks for a folder named 'Textures' in the user's assets folder, if it doesn't exist, it will create one. The class then copies the file received by Stable Diffusion and then pastes it into the new folder.

4.3.4 Generate Class

In Figure A.4, the process of sending the user prompt to Stable Diffusion takes place, this is done by compiling the parameter options selected by the user into a single string, and converting it to JSON. Next a WebRequest is made and this JSON data is sent by the HTTP POST method. The local server where Stable Diffusion is hosted will receive this data and begin generating the user's request. The following section is for receiving the HTTPWebResponse that Stable Diffusion will send back to Unity containing the image, the plugin then receives this response and displays the image file in the window.

4.3.5 Preferences Load & Save Classes

The code shown in Figure A.5 defines the save and load options that the user selected previously, this prevents the parameters being reset every time the plugin is reopened. It uses Unity's native EditorPrefs.Get and EditorPrefs.Set classes in order to do this.

4.4 JsonToC Script

The JsonToC script shown in Figure B.1(see Appendix B), is a simple method for serialising and deserialising the parameter options into JSON, allowing for the user data to be submitted to the localHost server.

4.5 Summary

The final implementation overall was mostly successful. All the Must requirements and some of the Should and Could requirements, as stated in section 3.1.1, were met. Once the plugin is fully set up, the user does not have to leave the Unity application at all when looking for textures, the plugin uses a fully localised and offline version of Stable Diffusion to generate its textures, it includes fully functional parameters for step count, texture size, sampler method, seed, and a seamlessness toggle. The plugin UI also includes a fully functional 'Import' button for the user to save any texture they like to their assets folder to then be used for their models. There is one minor issue relating to the texture size, in that the preview image changes size depending on the option selected, so smaller textures would be much smaller as a preview and vice-versa, which should not happen since it is a preview. Besides this, however, the Stable Diffusion Plugin for Unity project has been successful in terms of its implementation.

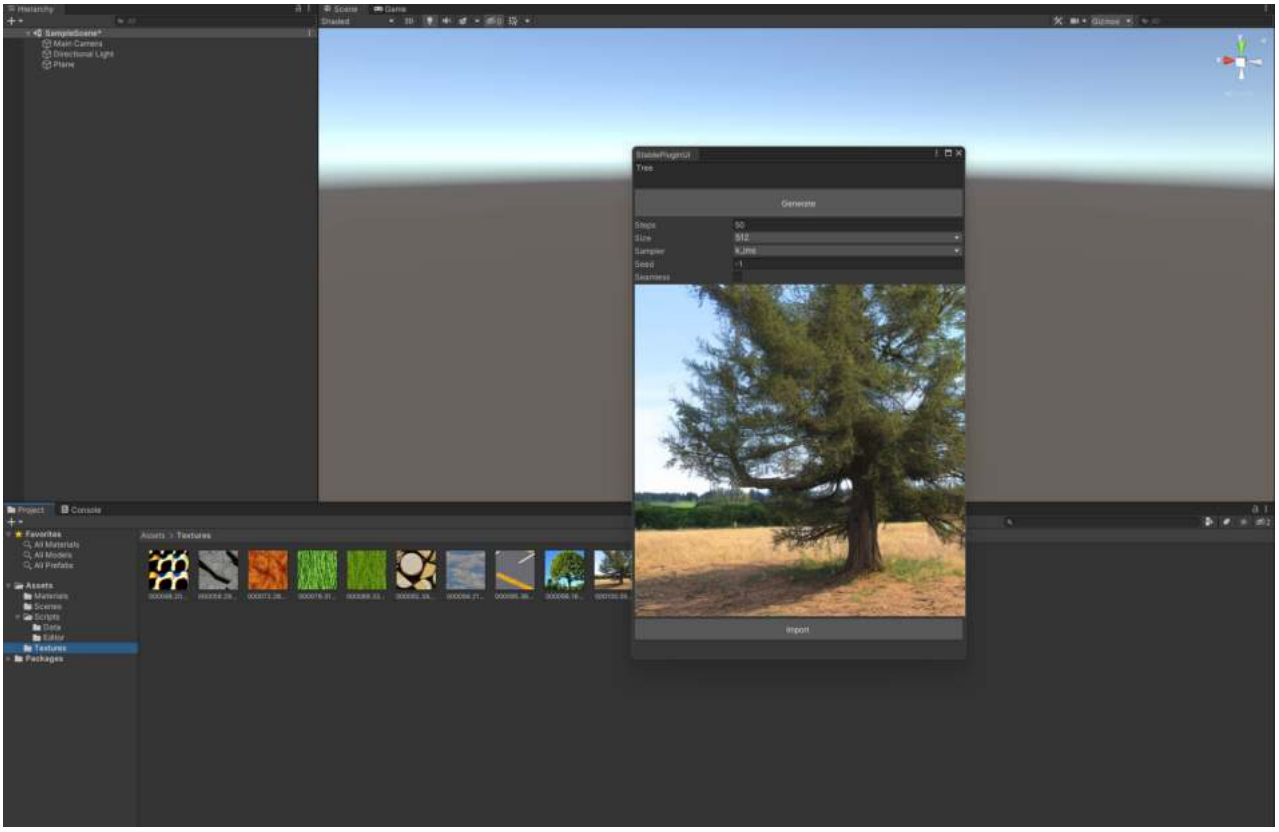


Figure 4.3: An AI generated texture of a tree

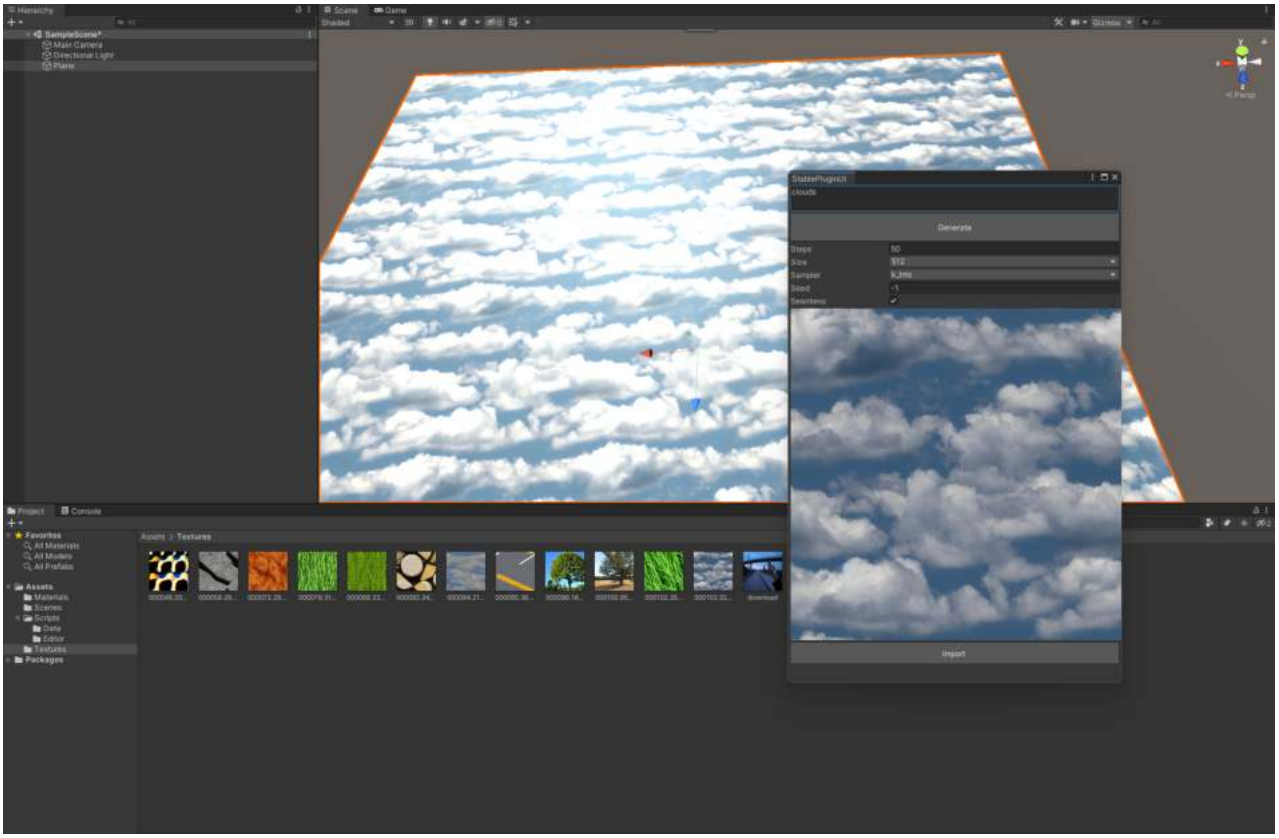


Figure 4.4: A seamless AI generated texture of clouds applied to a plane object

Chapter 5

Evaluation

The Stable Diffusion Plugin for Unity has been fully tested and reviewed by a friend of the researcher, who is a computer scientist and aspirational game designer. They were asked to use the plugin and experiment with all the parameters as much as they like, to evaluate the plugin based on how user friendly it seemed to them, how useful they felt it was, whether they thought the generated images were good enough to be used in game design, and if they thought the options that the parameters gave produced a noticeable divergence in texture output. Finally the test user was asked to rate the entire experience, summarising any positive aspects of the plugin and also anything they feel could be improved on in future.

5.1 Functionality

The test user was pleased with all the features, noting that the ability to acquire new textures without having to leave Unity was a very positive aspect. They showed approval over the included parameters saying that they allowed for a great amount of variability for the generated textures, hence giving more options to game developers. Although they mentioned that there are a few parameters that the online version had that were missing from this plugin, such as using an initial image, therefore, not having included parameters such as these detracts from the user experience slightly.

5.2 Usability

In terms of the plugins overall user-friendliness and intelligibility, the test user commented that the user interface was suitably simple, which given the target audience was a positive feature. They said the lack of too many parameters was beneficial for those Unity users who may not fully understand all the technical aspects of tools such as Stable Diffusion, so a simple UI would be easier to handle and learn, and hence, be more attractive for most people.

5.3 Textures

Overall, the test user was content with the textures that the plugin was able to provide. They said that the parameters allowed for a lot of flexibility and so its always possible for any user to find an image that they will be happy with. They were also pleased with the options for changing step number, sampler method and texture size, so that if the user wanted they could produce much higher quality images given the best parameters, albeit there would be a significant waiting period to generate a single texture. When the test user was asked whether they thought the textures produced by the plugin were suitable enough for use in Unity game development, they responded saying in terms of using the texture for just the albedo in a game object, they are suitable due to the fact that the images are high quality, however, they did say that since the plugin only generates 2D textures, they would be significantly inferior compared to textures that include PBR maps, as well as 2D images for example.

5.4 Summary

The User Evaluation in summary was very beneficial in terms of listening to conclusive feedback on the plugin. The test user was happy with most aspects of the plugin, especially its ease of use and the quality of the output. It is clear from this feedback that the project has delivered on its aim of being accessible to most Unity users, whether they are experienced or unexperienced in game development. Some things that the test user wished were included, however,

were more parameters such as the Initial Image functionality that Stable Diffusion allows online, this would definitely be beneficial to include in a plugin like this because it would allow the user to reuse textures they particularly like and make secondary editions of those textures using another prompt. Another feature they wanted to see was an option to produce materials for these generated textures, if this could be done then the textures could be used in much higher quality games. Overall, the review from the test user was mostly positive, and some essential feedback was obtained.

Chapter 6

Conclusion

The full development of the Stable Diffusion Plugin for Unity has been successful in its key aims and introduces an efficient and simple way for game developers of all abilities to find textures. The 5 Must goals that were defined in section 3.1.1 have all been met, the plugin accesses a local build of Stable Diffusion, and so can be used without any need to connect to the internet, and without any limitations due to internet speeds. The plugin never requires the user to leave the Unity environment to find their textures, which is a significant improvement for most users due to them having to either find suitable textures online, or create their own. And the plugin includes five parameters, three that were required in section 3.1.1 and two additional, all these parameters allow the user to alter their experience to how they want it to be.

Despite this, however, as stated in the user test in Chapter 5, this plugin is lacking in terms of the number of parameters that is offered, compared to Stable Diffusion online, and therefore, leaves a more basic tool overall. The plugin also lacks any other functionality in terms of PBR maps or similar, while most Unity textures will have additional texture maps creating a more realistic texture for higher quality games.

6.1 Revisiting Objectives

Looking back at the initial research and objectives, it is clear that this project has overall been a success in what it aimed to achieve. The background research undertaken in Chapter 2 has proven to be very useful in acquiring the

necessary knowledge for the development. It provided in-depth information about the nature of both Unity and Stable Diffusion and how they could be used in tandem to create a tool such as this, it also highlighted efforts made by others to solve similar problems that this project aimed to address, it was very useful to review the successful and detrimental aspects of these works, allowing the development of this plugin to focus on either utilising or improving on them. The evaluation provided in Chapter 5 was also equally helpful since it provided a thorough breakdown of all the details that a normal user would notice and the developer may miss, they provided very constructive feedback on the positive features of the plugin that give the user a good experience, and also on the features that could have been included that would have significantly improved that experience.

6.2 PESTLE Analysis

In order to provide a complete evaluation of this project, the PESTLE analysis will help to achieve this. The PESTLE analysis stands for Political, Economic, Sociocultural, Technological, Legal and Environmental factors, for a project such as this, the only relevant factors to consider for this analysis are the economic, technological and legal. In terms of any economic and technological factors affecting this plugin, the user will require some expensive hardware to be able to run Stable Diffusion locally, it is recommended to have at least 6GB of VRAM in a computer. Some users of Unity may fall short of this requirement and so will not be able to use this plugin. The only legal concerns for this project would be if there are any copyright laws made in future that target AI image generators, this may affect the freedom of use for the users of this plugin.

6.3 Future Work

Having obtained useful feedback from the user test in Chapter 5, it is clear that a project such as this is capable of improving greatly. Some features that could be included in future would be the full inclusion of default Stable Diffusion parameters into the plugin, this would allow much more user customisation

of the textures that are generated, and it would allow the full capabilities of Stable Diffusion to be included in the plugin. A criticism that was made in the user test was the absence of any visual enhancement features, such as the inclusion of PBR materials or texture maps, in future work it may be possible to invoke another program from the plugin that can automatically take an initial image and generate PBR materials and detailed texture maps from simple 2D images, this would greatly increase the quality of the textures generated by the plugin, and would broaden the appeal of the plugin to developers working on higher quality games. Another possible feature that could be implemented in future, would be the option of applying newly generated textures to an object of the user's choice without having to leave the StablePluginUI window. Finally, an option to consider in future would be to undergo the requirements to release this plugin on the Unity Asset Store, this would allow the plugin to be available to most users.

References

[1] S. Hollister, Ed., *Unity's IPO filing shows how big a threat it poses to Epic and the Unreal Engine* 24th Aug. 2020. [Online]. Available: https://www.theverge.com/2020/8/24/21399611/unity-ipo-game-engine-unreal-competitor-epic-app-store-revenue-profit (visited on 26th Apr. 2023) (p. 5).

[2] 'Hugging face.' (), [Online]. Available: https://huggingface.co/CompVis/stable-diffusion-v-1-4-original (visited on 24th Apr. 2023) (p. 25).

[3] Istein. 'Invokeai github.' (), [Online]. Available: https://github.com/invoke-ai/InvokeAI (visited on 23rd Apr. 2023) (p. 20).

[4] Istein. 'Invokeai web server.' (), [Online]. Available: https://github.com/invoke-ai/InvokeAI/blob/9b28c65e4b0526956fd90ad86e2118536e4eaa9b/docs/features/WEB.md (visited on 24th Apr. 2023) (p. 25).

[5] Juegoadmin, Ed., *Best Game Engines for Game Development in 2023*, Game Development 11th Jan. 2023. [Online]. Available: https://www.juegostudio.com/blog/best-game-engines (visited on 6th May 2023) (p. 4).

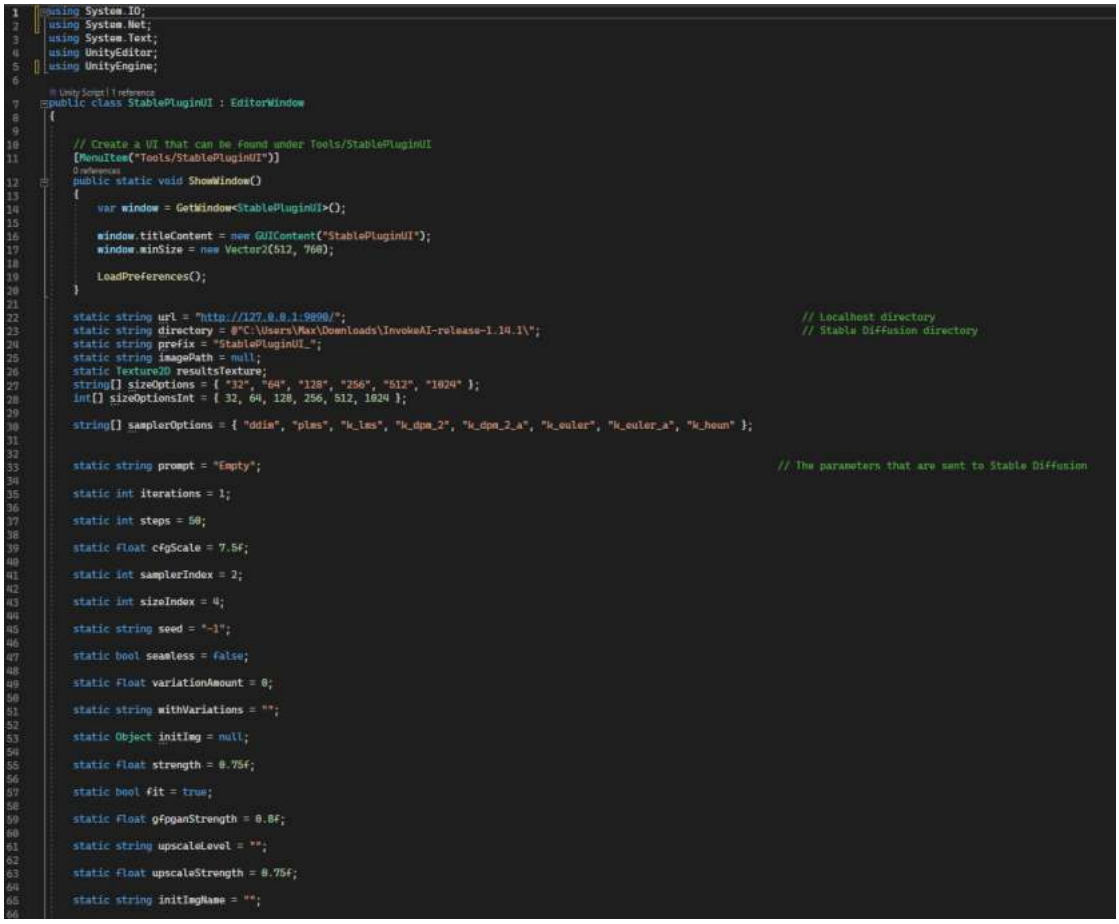
[6] C. Katri. 'Dream textures.' (), [Online]. Available: https://www.blendermarket.com/products/dream-textures (visited on 12th May 2023) (p. 9).

[7] Samyam. 'Pixela.ai.' (), [Online]. Available: https://pixela.ai (visited on 13th May 2023) (p. 10).

[8] M. Toftedahl, Ed., *Which are the most commonly used Game Engines?* Production 30th Sep. 2019. [Online]. Available: https://www.gamedeveloper.com/production/which-are-the-most-commonly-used-game-engines-#close-modal (visited on 28th Apr. 2023) (p. 4).

Appendix A

StablePluginUI Script



```
using System.IO;
using System.Net;
using System.Text;
using UnityEditor;
using UnityEngine;

Unity Script | 1 reference
public class StablePluginUI : EditorWindow
{

    // Create a UI that can be found under Tools/StablePluginUI
    [MenuItem("Tools/StablePluginUI")]
    0 references
    public static void ShowWindow()
    {
        var window = GetWindow<StablePluginUI>();

        window.titleContent = new GUIContent("StablePluginUI");
        window.minSize = new Vector2(512, 768);

        LoadPreferences();
    }

    static string url = "http://127.0.0.1:9090/";                                   // Localhost directory
    static string directory = @"C:\Users\Max\Downloads\InvokeAI-release-1.14.1\";    // Stable Diffusion directory
    static string prefix = "StablePluginUI_";
    static string imagePath = null;
    static Texture2D resultsTexture;
    string[] sizeOptions = { "32", "64", "128", "256", "512", "1024" };
    int[] sizeOptionsInt = { 32, 64, 128, 256, 512, 1024 };

    string[] samplerOptions = { "ddim", "plms", "k_lms", "k_dpm_2", "k_dpm_2_a", "k_euler", "k_euler_a", "k_heun" };


    static string prompt = "Empty";                                                 // The parameters that are sent to Stable Diffusion

    static int iterations = 1;

    static int steps = 50;

    static float cfgScale = 7.5f;

    static int samplerIndex = 2;

    static int sizeIndex = 4;

    static string seed = "-1";

    static bool seamless = false;

    static float variationAmount = 0;

    static string withVariations = "";

    static Object initImg = null;

    static float strength = 0.75f;

    static bool fit = true;

    static float gfpganStrength = 0.8f;

    static string upscaleLevel = "";

    static float upscaleStrength = 0.75f;

    static string initImgName = "";
```

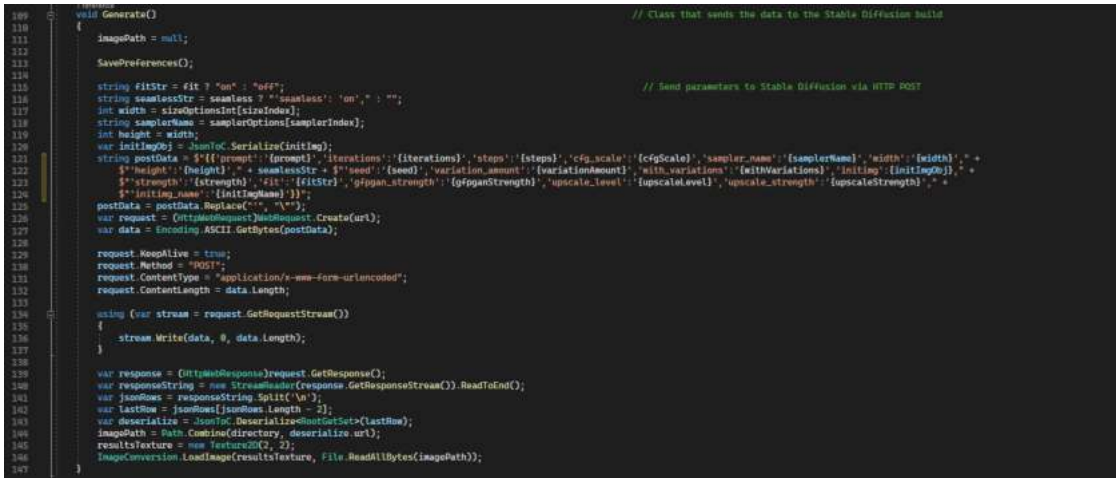
Figure A.1: ShowWindow Class & Initialisations

```
    // Create a UI window for the plugin
    0 Unity Message | 0 references
    void OnGUI()
    {
        prompt = EditorGUILayout.TextArea(prompt, GUILayout.Height(40));

        if (GUILayout.Button("Generate", GUILayout.Height(40)))
        {
            Generate();
        }
        steps = EditorGUILayout.IntField("Steps", steps);
        sizeIndex = EditorGUILayout.Popup("Size", sizeIndex, sizeOptions);
        samplerIndex = EditorGUILayout.Popup("Sampler", samplerIndex, samplerOptions);
        seed = EditorGUILayout.TextField("Seed", seed);
        seamless = EditorGUILayout.Toggle("Seamless", seamless);

        // Results
        EditorGUILayout.LabelField(new GUIContent(resultsTexture), GUILayout.Width(EditorGUIUtility.currentViewWidth), GUILayout.Height(EditorGUIUtility.currentViewWidth));

        if (GUILayout.Button("Import", GUILayout.Height(34)))
        {
            ImportTexture();
        }
    }
```

Figure A.2: GUI Class

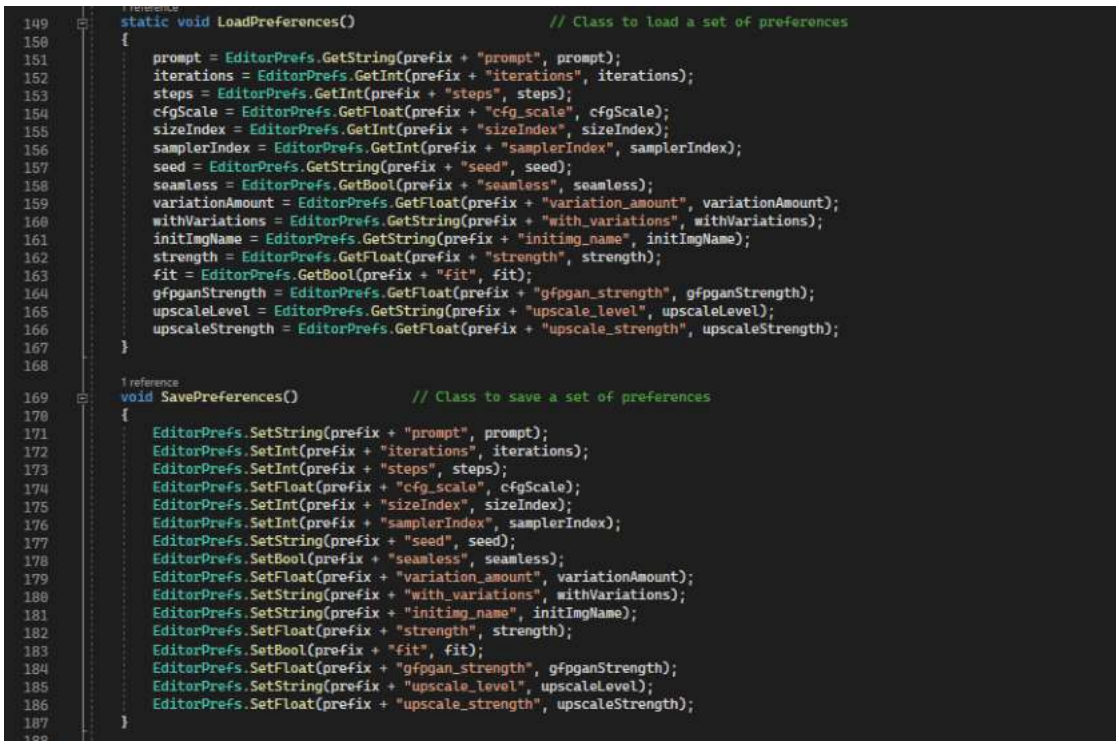
```
void ImportTexture()
{
    // Copy the generated image to Assets folder
    if (File.Exists(imagePath))
    {
        if (!Directory.Exists("Assets/Textures"))                    // Searches for Texture folder in Assets, if it doesn't exist, create it
        {
            Directory.CreateDirectory("Assets/Textures");
        }
        File.Copy(imagePath, "Assets/Textures/" + Path.GetFileName(imagePath), true);
        AssetDatabase.Refresh();
    }
}
```

Figure A.3: Texture Import Class



```
void Generate()                                                      // Class that sends the data to the Stable Diffusion build
{
    imagePath = null;

    SavePreferences();

    string fitStr = fit ? "on" : "off";                                                  // Send parameters to Stable Diffusion via HTTP POST
    string seamlessStr = seamless ? "'seamless': 'on'," : "";
    int width = sizeOptionsInt[sizeIndex];
    string samplerName = samplerOptions[samplerIndex];
    int height = width;
    var initImgObj = JsonToC.Serialize(initImg);
    string postData = $"{{'prompt':'{prompt}','iterations':'{iterations}','steps':'{steps}','cfg_scale':'{cfgScale}','sampler_name':'{samplerName}','width':'{width}'," +
        $"'height':'{height}'," + seamlessStr + $"'seed':'{seed}','variation_amount':'{variationAmount}','with_variations':'{withVariations}','initimg':{initImgObj}," +
        $"'strength':{strength},'fit':'{fitStr}','gfpgan_strength':'{gfpganStrength}','upscale_level':'{upscaleLevel}','upscale_strength':'{upscaleStrength}'," +
        $"'initimg_name':'{initImgName}'}}";
    postData = postData.Replace("'", "\"");
    var request = (HttpWebRequest)WebRequest.Create(url);
    var data = Encoding.ASCII.GetBytes(postData);

    request.KeepAlive = true;
    request.Method = "POST";
    request.ContentType = "application/x-www-form-urlencoded";
    request.ContentLength = data.Length;

    using (var stream = request.GetRequestStream())
    {
        stream.Write(data, 0, data.Length);
    }

    var response = (HttpWebResponse)request.GetResponse();
    var responseString = new StreamReader(response.GetResponseStream()).ReadToEnd();
    var jsonRows = responseString.Split('\n');
    var lastRow = jsonRows[jsonRows.Length - 2];
    var deserialize = JsonToC.Deserialize<RootGetSet>(lastRow);
    imagePath = Path.Combine(directory, deserialize.url);
    resultsTexture = new Texture2D(2, 2);
    ImageConversion.LoadImage(resultsTexture, File.ReadAllBytes(imagePath));
}
```

Figure A.4: Generate Class



```
static void LoadPreferences()                        // Class to load a set of preferences
{
    prompt = EditorPrefs.GetString(prefix + "prompt", prompt);
    iterations = EditorPrefs.GetInt(prefix + "iterations", iterations);
    steps = EditorPrefs.GetInt(prefix + "steps", steps);
    cfgScale = EditorPrefs.GetFloat(prefix + "cfg_scale", cfgScale);
    sizeIndex = EditorPrefs.GetInt(prefix + "sizeIndex", sizeIndex);
    samplerIndex = EditorPrefs.GetInt(prefix + "samplerIndex", samplerIndex);
    seed = EditorPrefs.GetString(prefix + "seed", seed);
    seamless = EditorPrefs.GetBool(prefix + "seamless", seamless);
    variationAmount = EditorPrefs.GetFloat(prefix + "variation_amount", variationAmount);
    withVariations = EditorPrefs.GetString(prefix + "with_variations", withVariations);
    initImgName = EditorPrefs.GetString(prefix + "initimg_name", initImgName);
    strength = EditorPrefs.GetFloat(prefix + "strength", strength);
    fit = EditorPrefs.GetBool(prefix + "fit", fit);
    gfpganStrength = EditorPrefs.GetFloat(prefix + "gfpgan_strength", gfpganStrength);
    upscaleLevel = EditorPrefs.GetString(prefix + "upscale_level", upscaleLevel);
    upscaleStrength = EditorPrefs.GetFloat(prefix + "upscale_strength", upscaleStrength);
}

1 reference
void SavePreferences()              // Class to save a set of preferences
{
    EditorPrefs.SetString(prefix + "prompt", prompt);
    EditorPrefs.SetInt(prefix + "iterations", iterations);
    EditorPrefs.SetInt(prefix + "steps", steps);
    EditorPrefs.SetFloat(prefix + "cfg_scale", cfgScale);
    EditorPrefs.SetInt(prefix + "sizeIndex", sizeIndex);
    EditorPrefs.SetInt(prefix + "samplerIndex", samplerIndex);
    EditorPrefs.SetString(prefix + "seed", seed);
    EditorPrefs.SetBool(prefix + "seamless", seamless);
    EditorPrefs.SetFloat(prefix + "variation_amount", variationAmount);
    EditorPrefs.SetString(prefix + "with_variations", withVariations);
    EditorPrefs.SetString(prefix + "initimg_name", initImgName);
    EditorPrefs.SetFloat(prefix + "strength", strength);
    EditorPrefs.SetBool(prefix + "fit", fit);
    EditorPrefs.SetFloat(prefix + "gfpgan_strength", gfpganStrength);
    EditorPrefs.SetString(prefix + "upscale_level", upscaleLevel);
    EditorPrefs.SetFloat(prefix + "upscale_strength", upscaleStrength);
}
```

Figure A.5: Preferences Load/Save Classes

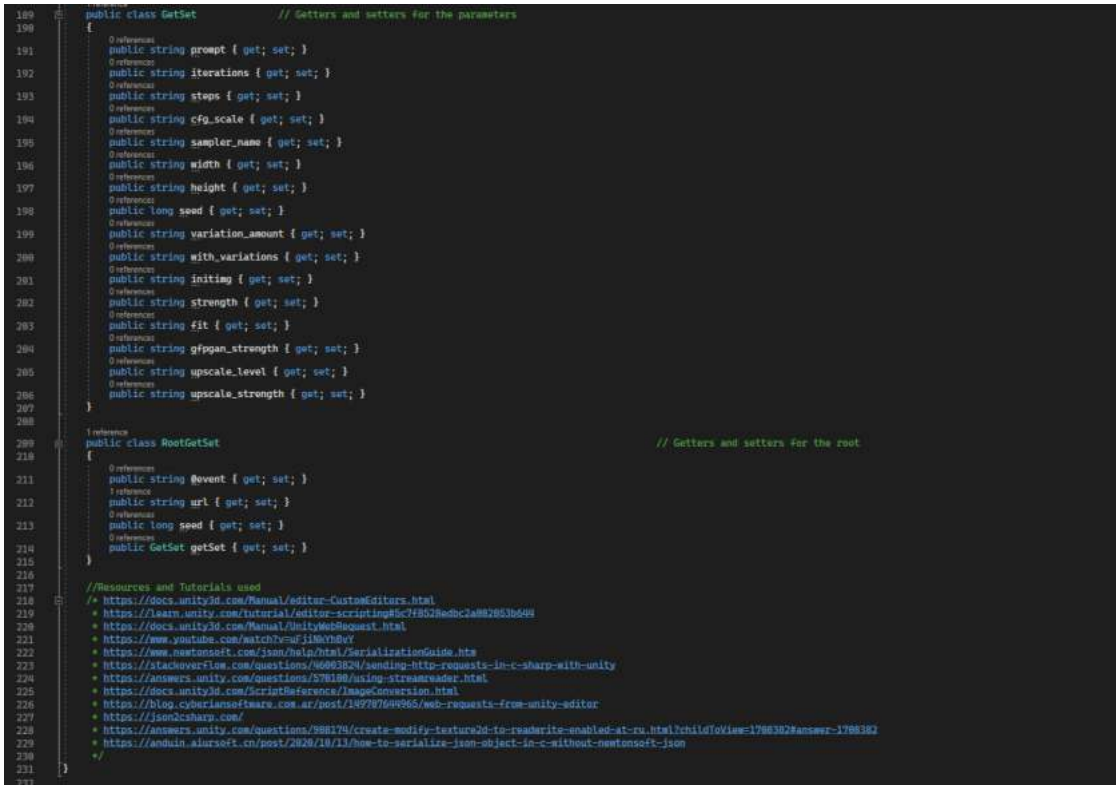


Figure A.6: Getter Setter Classes & References

Appendix B

JsonToC Script

```
using System.Runtime.Serialization.Json;
using System.IO;
using System.Text;

2 references
public static class JsonToC
{
    //Deserialise an object from a json string
    1 reference
    public static T Deserialize<T>(string body)
    {
        using (var stream = new MemoryStream())
        using (var writer = new StreamWriter(stream))
        {
            writer.Write(body);
            writer.Flush();
            stream.Position = 0;
            return (T)new DataContractJsonSerializer(typeof(T)).ReadObject(stream);
        }
    }


    //Serialize an object to json string

    1 reference
    public static string Serialize<T>(T item)
    {
        using (MemoryStream ms = new MemoryStream())
        {
            new DataContractJsonSerializer(typeof(T)).WriteObject(ms, item);
            return Encoding.Default.GetString(ms.ToArray());
        }


    }
}
```

Figure B.1: JsonToC Script

Appendix C

Poster

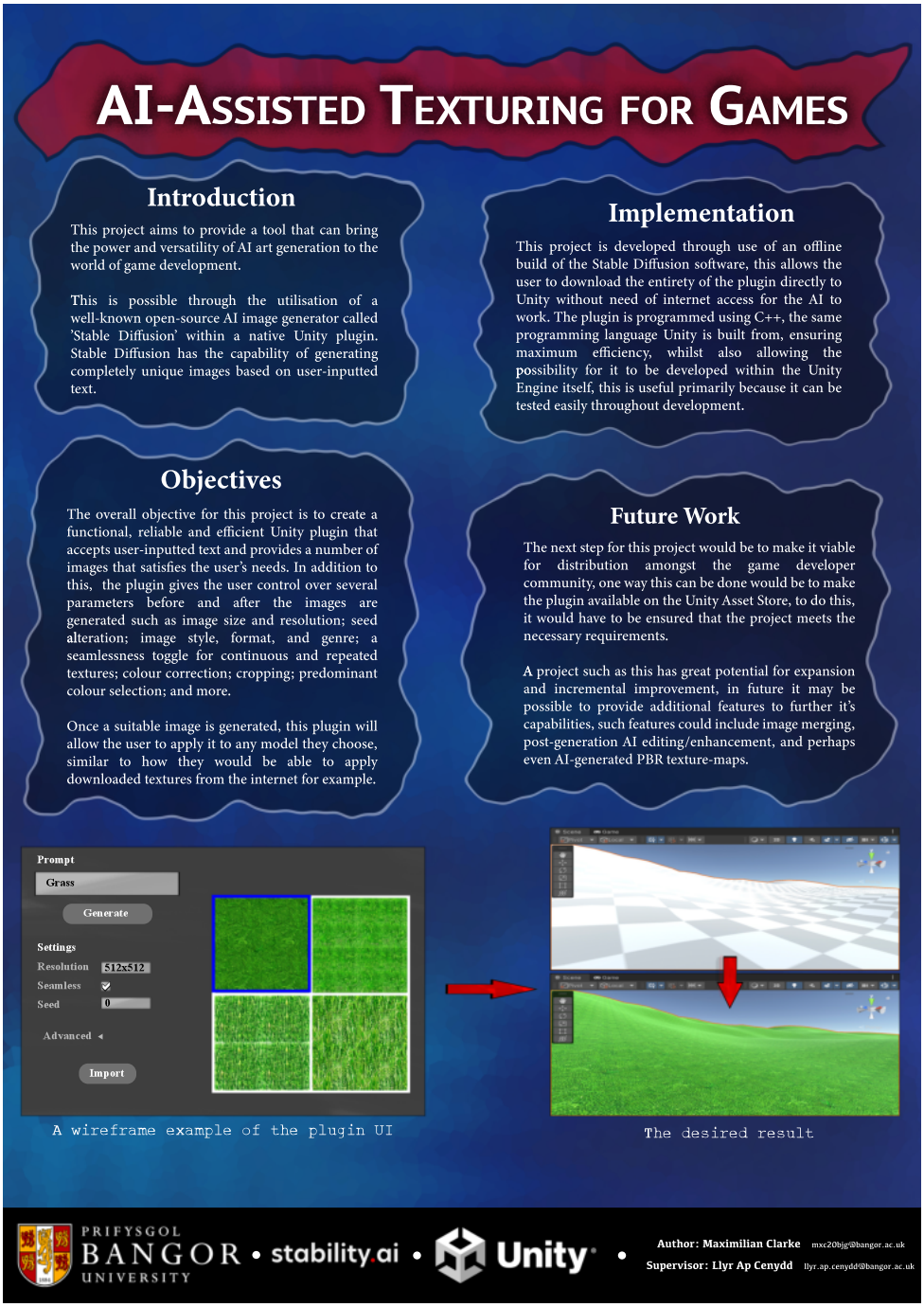


Figure C.1: Poster used for this dissertation at the expo