U.PORTO

# Node-based Native Solution to Procedural Game Level Generation

**Guilherme Silva Gama | UP202100735**

Master in Multimedia at Universidade do Porto

Advisor: António Fernando Vasconcelos Cunha Castro Coelho (PhD)

Co-Advisor: Diana Quitéria Teixeira de Sousa (MSc)

June of 2023

# Node-based Native Solution to Procedural Game Level Generation

**Guilherme Silva Gama | UP202100735**

Master in Multimedia at Universidade do Porto

Advisor: António Fernando Vasconcelos Cunha Castro Coelho (PhD)

Co-Advisor: Diana Quitéria Teixeira de Sousa (MSc)

*“A good idea is something that does not solve just one single problem, but rather can solve multiple problems at once.”* — **Shigeru Miyamoto**

# Resumo

A Geração Procedural de Conteúdo (PCG) aplicada ao domínio do desenvolvimento de jogos tem se tornado um tópico proeminente com um número crescente de implementações e aplicações. Soluções de PCG standalone e plugin, regidas por interfaces baseadas em nós e outras abordagens de alto nível, enfrentam limitações em termos de integração, interatividade e responsividade quando inseridas no processo de desenvolvimento de jogos. Essas limitações afetam a experiência do utilizador e inibem o verdadeiro potencial que estes sistemas podem oferecer.

Adotando uma metodologia de Action-Research, realizou-se um estudo preliminar com entrevistas a especialistas da área. A avaliação da relevância e da interface mais adequada para a solução proposta foi concretizada através de uma série de protótipos visuais. Posteriormente, foi implementado um protótipo funcional e conduzido um estudo de caso para uma amostra mais ampla, incluíndo especialistas e desenvolvedores de jogos. Os participantes realizaram uma série de exercícios orientados, operando com o protótipo. Após a conclusão dos exercícios propostos, os participantes avaliaram a relevância da solução e da experiência do utilizador através de um questionário.

No desenvolvimento de uma metodologia nativa de PCG baseado em nós, integrado no motor de jogo, identificámos limitações e concluímos que existem diversos desafios ainda por superar no que diz respeito a uma implementação completa de um sistema complexo e amplo.

**Palavras-chave:** Geração Procedural, Desenvolvimento de jogos, Interface baseada em nós, Experiência de utilizador

# Abstract

Procedural Content Generation (PCG) applied to game development has become a prominent topic with increasing implementations and use cases. However, existing standalone and plugin PCG solutions, which use Node-based interfaces and other high-level approaches, face integration, interactivity, and responsiveness limitations within the game development pipeline. These limitations hinder the overall user experience and restrain the true potential of PCG systems.

Adopting an Action-Research methodology, a preliminary interview was conducted with experts in the field. The evaluation of the solution's relevance and the identification of the most suitable interface approach was carried out using a series of visual prototypes. Subsequently, a functional prototype was implemented, and a case study was conducted using a broader sample, joining PCG experts and game developers. Participants engaged in a series of guided exercises, operating with the implemented solution. After completing the exercises, the solution's relevance and user experience was evaluated through a questionnaire.

In developing a native node-based PCG methodology integrated into the game engine, we identified limitations. We concluded that several challenges are yet to be overcome regarding fully implementing a complex and extensive system.

**Keywords:** Procedural Content Generation, Game Development, Node-based Interfaces, User Experience

# Acknowledgements

I would like to express my deepest appreciation to everyone who helped and contributed to the completion of this thesis work and the culmination of my Master's degree.

To my esteemed supervisor, Prof. António Coelho, for whom I am immensely grateful for believing in my ambitious vision for this thesis project. Your guidance and support have been instrumental in bringing this project to its successful completion.

To Diana Sousa, my co-supervisor, I am profoundly grateful for the countless hours dedicated to this dissertation. Your relentless patience and meticulous attention to detail demonstrated your genuine care for my academic growth and success. Your careful observations and suggestions helped shape this work to its fullest potential.

To João Jacob, Martinus Suijkerbuijk, Nélson Rodrigues, and Pedro Silva for their expertise, guidance, and invaluable feedback during this research. Your contributions have greatly influenced the development and strength of this project.

To Pedro Silva, once again, I want to express my deepest gratitude. Your exceptional PhD work served as the core foundation for the implementation of this project, without it, this work would not have achieved the heights it reached.

To my friends, Afonso, Fábio, João, Lucas, Rita, Tiago, and all others, I am sincerely grateful for the joy, help, and patience you had to endure my frustrations. Above all, thank you for the memories we created together.

Last but not least, I am immensely grateful to my beloved parents, my caring sister, my sweet girlfriend Inês, and my dear grandma. Their love, support, and unwavering presence have brought immeasurable joy and strength to my journey. In the toughest moments, their guidance and understanding have been vital. I am forever blessed to have their love and support in my life.

# Table of Contents

# List of Figures

# List of Tables

# Abbreviations and Symbols

| | |
|---|---|
| FURPS | Functionality, Usability, Reliability, Performance, and Supportability |
| NPC | Non-playable Character |
| PCG | Procedural Content Generation |
| UI | User Interface |
| USS | Unity Style Sheet |
| UX | User Experience |
| UXML | Unity Extensible Markup Language |
| VPL | Visual Programming Language |

# 1 Introduction

Video games have evolved from being a mere form of entertainment to becoming an integral part of multiple domains including education, commerce, and healthcare. The Game Development industry has witnessed remarkable expansion, driven by the increasing complexity and intricacy of the levels created. This growth highlights the increasing importance placed on the field, as developers aim to design immersive and stimulating gaming experiences.

Designing levels for these intricate worlds is a challenging process, as it requires a unique set of strategies and techniques to tackle all the requirements for each game. However, specific metrics such as variety, cohesion, and clarity can be considered essential components for any well-designed level. This has led to the development of advanced modelling techniques that allow for a high-level approach to creating game worlds in a manageable and controllable manner. These techniques are inherently procedural, which means that they use algorithms to generate data as opposed to a manual creation process. Also known as Procedural Content Generation, these algorithms have clear advantages over a manual approach in level design. The advantages range from the potential of nearly unlimited content variation to a higher game designer's autonomy. However, it is crucial to acknowledge that there are also certain drawbacks and limitations inherent in their usage.

## 1.1 Problem Specification

Procedural Content Generation (PCG) provides a valuable approach to level design, facilitating rapid experimentation through the creation of flexible systems. By leveraging PCG, complex levels can be rapidly regenerated with minimal effort. This approach enables level designers and developers to iterate as needed, gaining a deeper understanding of the level's fundamental elements.

However, despite the benefits provided by the current PCG systems, they present several challenges once integrated into the complete game development pipeline. Since most PCG systems are standalone software, substantial friction between the developer and the generated level is depicted. This reduced interactivity is caused by the large reload times and constant switching between software during each iteration cycle, making it difficult for designers and developers to implement an iterative level design approach effectively.

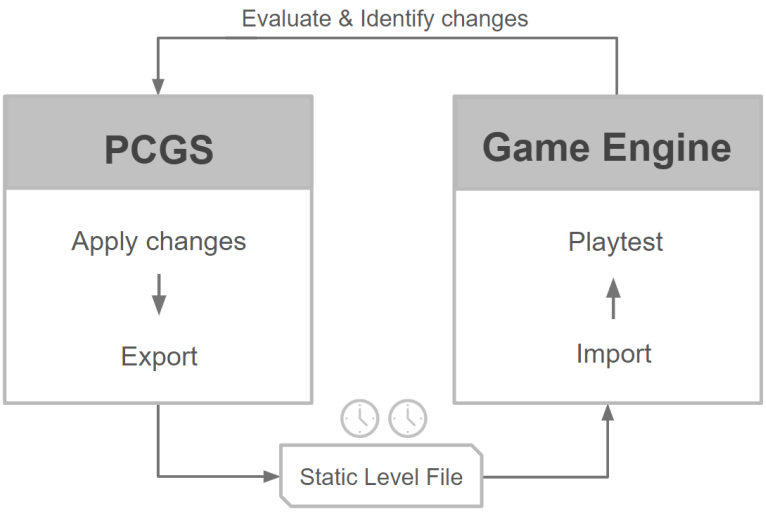



*Figure 1 –Usage cycle of current PCG systems.*

Although PCG systems offer plugin software as a solution to this issue, it does not fully resolve it, introduces complexity, and demands additional computational resources.

The limited ease of use of PCG systems is another noticeable problem. These systems are undoubtedly complex software and require a high level of technical expertise to be used effectively, which can be daunting for level designers and

developers who are already familiar with the game engine's logic and structure. It is usually necessary to relearn naming conventions, symbols, and workflows to use the system effectively. Additionally, the provided plugin solutions require a previous setup on both software ends that involves unintuitive manual directory paths and component naming specifications to work correctly.

In summary, current limitations of interactivity and ease of use present significant obstacles to using PCG systems in game development. Addressing these issues is crucial for maximising the benefits of PCG systems and streamlining the game development process.

## 1.2 Objectives

The main objective of this thesis is to design and develop a node-based PCG methodology natively integrated into a game engine. This objective aims to contribute to the field of study by addressing the limitations of the current PCG workflow and providing game developers with an optimized process for generating content procedurally. The focus is on developing a robust and intuitive framework that enhances the user experience and streamlines its development process.

Additionally, the following goals must be addressed to reach the main objective:

1. Identify the fundamental metrics of User-Experience in a PCG system;
2. Apply the best evaluation process of a PCG system User-Experience;
3. Understand the most relevant User-Interface approaches for a node-based PCG systems.

The above objectives seek to ensure a solid foundation for developing a native node-based PCG system and improving its respective interface workflows.

## 1.3 Research Questions

This project aims to answer the central research question: “Does developing a PCG system natively in a Game Engine provide significant advantages for the Level Design process?”. However, it also raises additional questions such as:

1. What are the key interactivity and user experience metrics that hold the most significance in a PCG system?
2. What are the potential approaches for designing the interface of a PCG system?

## 1.4 Methodology

To address the study’s specified objectives, this research adopts the Action-Research Methodology, which follows a cyclical process involving planning, action, observation, and reflection. This methodology facilitates continuous iteration through the stages of design, implementation, and evaluation, allowing for the gradual refinement and improvement of the proposed solution. The iterative nature of this approach allows for the collection of feedback, making necessary adjustments, and enhancing the overall effectiveness of the PCG system under investigation.

In order to practically apply the Action-Research Methodology, two distinct studies were planned and conducted (Figure 2):

Study I involved a semi-structured online interview conducted with a sample group of experts in the field of Game Development who have experience in working with PCG systems. The focus of this study was to evaluate a series of low-fidelity prototypes and gather feedback on the User Experience (UX) of the initial approach for the PCG system’s User Interface (UI).

Study II consisted of a digital questionnaire that was provided after the implementation of the native functional prototype. This study aimed to measure the user experience of the system and validate or corroborate the benefits associated with the proposed solution. The questionnaire allowed users to provide feedback on their

experience with the integrated PCG system, for further assessment of the solution's usability, functionality, and overall effectiveness.

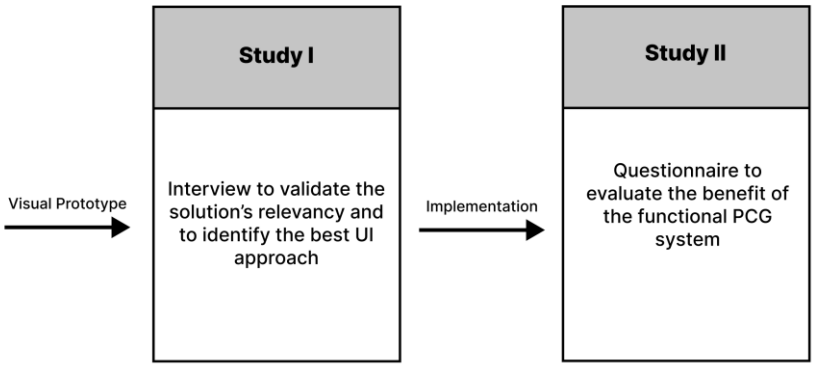



*Figure 2 - The order and context of the two conducted Studies*

## 1.5 Dissertation Structure

This document contains a total of eight chapters, along with a section of references and annexes. Following this order, the subsequent paragraphs provide an overview of the content covered in each chapter:

Chapter 1 (Introduction) contextualizes the thesis domain and motivation, highlighting the addressed problem, objectives, and research questions.

Chapter 2 (State of the Art) offers theoretical contextualization on the central subjects and performs a comparative analysis of current external PC systems. It concludes by discussing the possible technologies for the solution's development.

Chapter 3 (Methodologies) outlines the action-research methodology applied in the project's context, including the task roadmap, and the structure of each case study.

Chapter 4 delves into the process of software requirement analysis and design. It covers the analysis and documentation of technological backbones such as the target Game Engine and the PCG base library. Additionally, it enumerates functional and non-functional use cases using the FURPS model and presents the class diagrams of the designed structure.

Chapter 5 (Implementation) chronologically addresses the implementation of the specified use cases based on the initial requirements and designs.

Chapter 6 (Results and Discussion) presents and discusses the results of the research instruments, providing detailed insights into each conducted study.

Chapter 7 (Conclusions) condenses all the significant conclusions from the work and studies, considering the encountered obstacles, difficulties, and feedback collected.

Finally, chapter 8 (Future Work) takes into account and enumerates potential future tasks for subsequent iterations.

# 2 State of The Art

## 2.1 Level Design

Level design is a crucial game development component, often regarded as an art form in its own right. According to (Adrian & Ana Luisa, 2013), “The level design is an art which consists of creating the combination of challenge, competition, and interaction that players call fun and involves a careful and deliberate development of the game space (...)”. To fully comprehend such complex and multilayered process, it is necessary to define Game Level and what it encompasses.

### 2.1.1 Taxonomy and Challenges

Levels are defined play spaces where players advance by overcoming obstacles, interacting with non-playable characters (NPCs), and collecting items. They are segments of a larger game world and thus inherit its problems and challenges in an atomic scope. These spaces comprise many interconnected components, and it is important to consider the bigger picture when designing them. Therefore, designing a level should take into account all of its elements, their global significance, and how they behave within the context of the game as a whole.

According to Jong (2008) in his book "The How’s and Whys of Level Design", the process of level design is centred on six pillars:

Firstly, "Optimization and Polish" is a crucial aspect of level design, requiring the level to be optimized for smooth performance and playability. Additionally, a certain level of polish is essential to maintain the player's suspension of disbelief and avoid frustrating gameplay.

Secondly, "Gameplay" plays a central role in keeping players actively engaged and immersed in the game. Ensuring enjoyable gameplay is vital to retain the players' interest and motivation throughout their gaming experience.

Thirdly, "Immersion" is key to creating a captivating and believable game world. The level must be designed to fully envelop the player, enhancing their overall gaming experience.

Fourthly, the "Visuals" of the level are of utmost importance in attracting the audience and setting the desired atmosphere. The presentation and aesthetics contribute significantly to the overall appeal of the game.

Fifthly, "Functional Design" is integral to level design, as it aligns the level's narrative and overarching theme with the game mechanics. All elements within the level must appear to belong and serve logical purposes, enhancing the overall gameplay experience.

Finally, "The Combination" of all these pillars is essential to achieve a harmonious balance and ensure a high-quality level design. By successfully integrating and balancing these elements, game developers can create compelling and immersive gaming experiences for their players.

### 2.1.2 Gameplay-Centred Approaches

Level design is a vital component of game development, as it contributes to shaping the player's experience and entertainment level while exploring it. As a level designer, it is crucial to put him/herself in the player's perspective and comprehend their point of view to identify what key elements shape the experience. According to Galuzin (2011), there are three leading approaches to planning a game level based on a gameplay perspective:

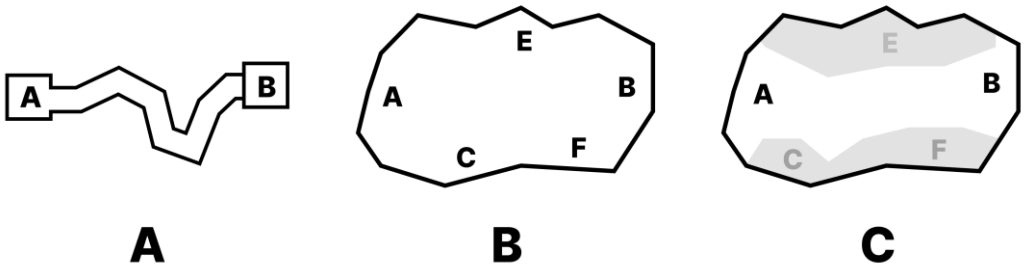



*Figure 3 - Gameplay centred Level Design Approaches.*

A. **Linear**: This approach is characterised by a straightforward, hard-locked path the player must follow to progress from point A to point B. This type of level design is particularly suited for story-driven games, emphasising a linear progression of events. Examples of such games include the Crash Bandicoot and Super Mario Bros series.

B. **Open World**: Unlike the linear approach, open-world levels give the player complete freedom to explore the vast, detailed universe without restrictions or physical barriers. This level design requires significant planning and thought, but the outcome is frequently an immersive and highly replayable game. Examples of such games include the Grand Theft Auto and Elder Scrolls series.

C. **Mixed**: The third approach is a blend of the first two, allowing the player to traverse a planned and soft-locked path while exploring the level and completing it as they choose. This approach gives the player more freedom of choice while maintaining partial control of the chronological order of the game events. Titles like Dark Souls and The Legend of Zelda series effectively use this approach.

### 2.1.3 Objects

Interactivity is a defining facet of games, that sets them apart from static virtual worlds. To craft an engaging experience, it is crucial to comprehend the desired interactivity and the means to achieve it effectively.

Based on Jong (2008) and considering the following the jungle-level setting in "Shadow of The Tomb Raider" (Figure 4) as an example. To design such level, the designer must be aware of its overall scope, such as how the jungle will look and the

path the player will navigate. On a more granular level, he must plan which objects from the environment will react with the player, with the Non-playable Characters (NPCs with other objects, and if the game engine can simulate these degrees of interaction complexity.

*Figure 4 – Gameplay of "Shadow of The Tomb Raider" jungle level.*

According to Short Tanya and Adams Tarn (2017), a set of generic properties can describe the objects that make up a game level. These properties include:

(1) **Topological**: Properties based on the underlying structure of the content, independent of its appearance. For the game logic, these properties may include the presence of loops and cycles or the branching of the player's path. For the narrative structure, these properties may include the story's length or the shape of the narrative arc.

(2) **Experimental**: Properties which describe how the player interacts with and experiences its content. It includes the pace of the player's movement through the space, the level of difficulty, and the desired number of solutions/strategies the player can achieve.

(3) **Aesthetic**: Properties that describe the generated content's graphic and audio qualities. These include using a specific colour palette, the proportion of warm and cool colours, and the tempo of the background music.

(4) **Semantic**: Focuses on the objects' significance, their contextual relevance within the game, and the methods used to represent and communicate these elements to the player.

### 2.1.4 Semantics

The previous chapter discussed the importance of objects and how they shape the level's experience. However, to achieve a fully immersive experience, it is necessary to delve deeper into semantics applied to the game objects and their universe.

Semantics, as defined in linguistics, refers to the study of meaning in language and communication. It encompasses the meaning of words, phrases, sentences, and even actions, focusing on understanding the purpose behind their inherited phenomena (Kearns, 2017; Löbner, 2013).

Analogously to object semantics, game world semantics explores the meaning behind game elements, including objects, interactions, and events. They play a crucial role in creating behaviorally cohesive and believable worlds by establishing clear and consistent relationships between objects and interactions. Moreover, semantics can help players understand and make sense of the world and their actions within it.

In Tutenel et al. (2008), three levels of world semantics in games were specified:

(1) **Object semantics**: Referring to an object's physical and functional properties that are specific to it. Physical properties include material, dimensions, and mass. On the other hand, functional properties comprehend the behaviours of interaction in which they can impact other objects, as introduced in the concept of "smart objects"
(2) **Object relationships**: Describing relationships between instances of objects (e.g., due to proximity) or between object classes. Inheritance is a key type of relationship that can be expressed in a taxonomy to classify and group objects with similar properties. Other relationship types include ownership, causality, aggregation, and inclusion, commonly found in ontologies.
(3) **World semantics**: Designating the highest level of specification in the game environment, affecting all entities. Factors such as daylight, weather, seasons, and contextual information (e.g., demographic and economic) globally impact game development, particularly in games' business and urban simulation genre.

## 2.2 Procedural Content Generation

Procedural Content Generation (PCG) is the algorithmic generation of data as opposed to a manual input process. The first attempts at PCG can be traced back to the mathematician Benoît Mandelbrot's introduction of the term "Fractal" in his work. Fractals are a class of shapes that repeatedly produce seemingly complex and unexpected models when following a set pattern. So, it is achieved through an iterative and recursive approach to a defined equation, and the result has strong atomic level similarities to the singular rule shape but produces a distinguished result in its overall composition.

### 2.2.1 Origins

PCG algorithms, applied to game development, started to take shape in 1980, with the classic dungeon crawler Rogue, one of the first examples of this type of dynamic content generation. In Rogue, the maps were randomly generated, providing a unique experience for each game playthrough. Later, this concept gave birth and rise to the "Roguelike" genre of games, characterised using randomised levels and permanent player death.

With the advancement of technology and the growing need for increasingly complex and comprehensive virtual worlds, the area of PCG has grown to include a diverse set of approaches and tools. Today, PCG is widely used in the game development industry to generate all forms of content, including terrains, maps, levels, narratives, puzzles, and characters.

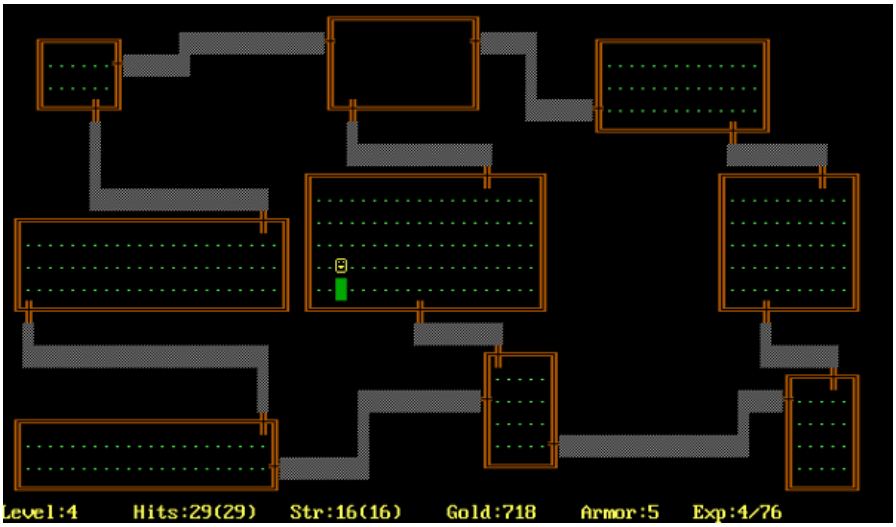



*Figure 5 – A dungeon from Rogue (1980).*

### 2.2.2 Advantages and Disadvantages

PCG is a powerful tool that can help developers create replayable and enjoyable games. However, it is important to note that it is not a one-size-fits-all solution, and its effectiveness depends on the specific use case. In this subchapter, we will explore the general advantages and disadvantages of PCG in game development.

**Advantages:**

(1) Increased replayability: PCG allows for the generation of different levels, enemies, and items each time the game is played, providing a new experience for the player.
(2) Reduced development time and costs: With PCG, developers can automate creating content, allowing for faster and cheaper development.
(3) Greater control over the game's difficulty: PCG can create adaptive difficulty levels that adjust to the player's skill level, making the game more accessible to a wider range of players.
(4) Greater Designer’s autonomy: PCG allows artists and designers to create their own stories and experiences, giving them control over the whole level development process.

**Disadvantages:**

(1) Quality assurance: PCGs can generate unexpected results and are usually challenging to debug, making it a significant risk for game development if manual validation and fine-tuning processes are not employed.

(2) Reduced authored experience: PCGs can be less suitable for games that rely heavily on an authored experience, as it can be difficult to control the generated content at a granular level.

### 2.2.3 Use-cases

Regarding implementing PCG in game development, it is imperative to carefully consider the scope of its involvement in achieving the game's objectives. The amount of time required for generating most of the game content through procedural methods cannot be overlooked and significantly impacts project planning. As changes to project direction are common in game development, they may result in substantial implications for the codebase (Short Tanya & Adams Tarn, 2017). The preceding work highlights that implementations in the field of PCG can be categorised into:

**Integral -** In the best-known examples, the decision to incorporate PCG is part of the game's basic Design. Most games that heavily rely on PCG also rely completely on it to be the kind of games they are. Rogue and Spelunky are two games that heavily rely on randomness for their basic gameplay.

**Drafting Content** - PCG is, in many cases, utilised in game improvement to deliver much content that is then cleaned later. The normal illustration of this is an open-world game, for example, Skyrim, which has a huge explorable guide. Major segments of this are created utilising procedural strategies and are then changed and cleaned by hand later. Different models incorporate riddle games where a generator can create many instances of resolvable riddles, and afterwards, a human physically chooses or sorts the additional interesting ones.

**Modal** - Some games have a modest PCG need yet have a unique gameplay mechanic that uses procedurally generated material. This outstanding piece of ancillary content to the main game story can be referred to as "infinity mode."

**Segmented** - To employ PCG, specific game sections may be divided apart from the rest of the game's Design. For example, procedural music might still be used in a linear, hand-crafted game. Interesting random components could be present in a particular

graphical effect. A particular chamber in a single gaming area may provide unique procedural guidelines.

### 2.2.4 Interior Levels

Within the realm of PCG, Interior Level Generation has its specific challenges and requirements, mostly based on the architectural intricacy of designing spaces. Several academic attempts to tackle this challenge have been made with vastly different approaches.

**Case 1: Office Building PCGS**

Hahn et al. (2006) developed a system for generating persistent game interior architecture. Their system focuses on generating huge rectangular office buildings through a space partitioning method based on architectural principles. This system was designed to generate in real-time by only creating subregions of a building that contain rooms visible to the player. Achieved using a persistent random seed, it allows sections of the building to be discarded and later re-generated as needed. The system's limitations, however, are that it generates rooms and hallways with limited architectural detail.

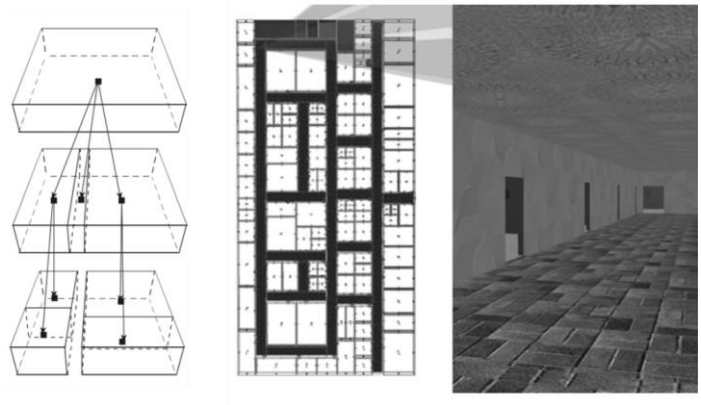


*Figure 6 – (Left) A small generation tree. An overhead (Middle) view of a completely generated floor. (Right) The building's first-person view of its generated contents.*

**Case 2: House Layout PCGS**

Merrell et al. (2010) created a PCG system to generate more realistic house layouts based on the process used by human architects. Their approach consisted of two stages, first generating an architectural program from high-level requirements through training a network on real architectural programs and second generating a layout from the

architectural program through an optimisation system based on a function measuring room accessibility, area, aspect ratio, and shape. This system can generate houses with multiple storeys and in multiple styles, resulting in a more realistic representation of interior spaces compared to other PCG systems.


*Figure 7 - A bubble diagram (left), generated by a Bayesian network trained on real-world data. A set of generated floor plans (middle). A 3D model (right), generated from the floor plans.*

## 2.3 Node-based Interfaces

Node-based interfaces fall under the domain of Visual Programming Languages (VPLs). To understand the significance and principles of node-based interfaces, it is necessary first to define what VPLs are and what they encompass.

### 2.3.1 Visual Programming Language

Visual Programming Languages (VPLs), as defined in (Ates et al., 2006), are programming languages that provide the means to specify and execute programs in two or more dimensions. This approach differs from textual programming, where the programmer edits a one-dimensional stream of characters. With VPLs, the user interacts with a two-dimensional representation, making it easier to understand and visualise the program's structure and flow.

VPLs are also referred to as flow-based interfaces, as they display complex processing structures as a flow of information. This higher interpretation capability is believed to result from cognitive psychology, as visual information can be processed with two hemispheres of the human brain in parallel rather than just one (Preidel & Borrmann, 2016). Due to VPLs' features and potential have been widely introduced as a higher-level addition to programming languages to lower the entry barrier for new programmers

(Shin et al., 2014). Most VPLs belong to one of two categories: programs that reside entirely in the virtual world or programs with a physical counterpart but with a relatively abstract connection between hardware and software.

### 2.3.2 Graph Structure

As stated previously, node-based interfaces are a subset of the domain of VPLs, and their Design is constructed around a graph.

According to Singh (2014), graphs are discrete mathematical structures that model the pairwise relations between objects. Graphs provide a convenient representation of various mathematical objects, consisting of two sets: a set of vertices (also referred to as nodes) and a set of edges. The restrictions imposed on the edges can vary depending on the problem, with directed edges used in some situations and undirected edges in others. This flexibility and versatility make graphs a useful tool for solving real-life problems.

Translating onto a software representation, nodes correspond to procedures that execute a series of program instructions that perform transformations, analyses, or filters on the incoming data conveyed through edges. The ability to link nodes together allows complex tasks to be broken down into atomic units that are easier to understand.

Referencing the work of Pedro Silva in "Procedural Content Graphs" (Silva, 2015), the implementation of nodes includes inputs and outputs represented by derived classes from base classes. These inputs and outputs are interconnected through pointers, allowing nodes to reference and retrieve data from other nodes. During execution, a node retrieves its inputs by following the pointers stored in its inputs, performs its operation on the retrieved inputs, and generates corresponding outputs.

## 2.4 Existing Solutions

### 2.4.1 Houdini

Houdini, created by SideFX, is a standalone 3D animation and visual effects software that features numerous PCG systems. The software is known for its robust and efficient node-based interface, which allows users to easily create, edit, and manage complex 3D assets (Holub et al., 2020; Naiman et al., 2017). It is widely adopted by professional game, film, and television studios, establishing its significant presence in the industry.

Houdini's node-based interface operates by organising the creation process into a series of interconnected nodes, each corresponding to a specific operation or set of operations. The nodes can be linked to create a graph representing the functional data flow of operations required to generate the final output. It allows users to visualize the logical steps involved in the whole graph easily.

From simple asset production to intricate simulations and visual effects, Houdini is capable of a multitude of tasks. The software's flexibility and robustness make it a popular choice for game development, where it is used to generate terrain, environments, characters, and other game assets. So, Houdini's node-based interface also makes it an ideal tool for film and TV visual effects, where it is used to create complex simulations and effects that would be difficult or impossible to achieve using other methods.

In addition to its main application, Houdini also contains a plugin integration software entitled Houdini Engine. This middleware software allows users to make changes directly inside the game engine (compatible with Unity and Unreal Engine) and generate procedural results without leaving the engine.

### 2.4.2 Blender

Blender is an open-source 3D computer graphics software, developed by the Blender Foundation. This software encompasses an extensive range of features that allow for 3D modelling, animation, compositing, and simulation. With its lightweight versatility and continuous support, Blender has established itself as a preferred software tool for many indie studios and independent artists. Additionally, the software benefits from a large, collaborative user community that provides support, resources, and knowledge.

Blender recently introduced a new tool for modifying geometry entitled Blender Geometry Nodes. This node-based procedural content generation system provides a flexible and intuitive way of generating, modifying and transforming 3D geometry. With Blender Geometry Nodes, users can create complex and dynamic shapes by connecting various nodes in a visual graph interface. Exposed graph parameters allow for quick adjustments and subsequent 3D models' customization by influencing the node network. Blender Geometry Nodes have sparked much interest and popularity within its community, as it allows for a procedural 3D modelling process, which can lead to more complex and flexible results than using traditional modelling methods.

Viga Entertainment, a software development company, has developed a plugin integration software called "Livelink for Blender" that bridges the technological gap between Blender and the Unreal Engine. Then, it allows Blender Geometry Nodes to be

used within the Unreal Engine, making it easier for game developers to import and use the generated 3D models and assets within their games.

### 2.4.3 Sceelix

Sceelix is a standalone procedural generation software for automating 2D/3D content creation using algorithms, rules, and mathematical models. Developed by Pedro Silva in an academic context (Silva, 2015), this software offers a unique approach to content generation that does not focus on a single type of content or structure. Sceelix can produce various types of content, including terrains, vegetation, roads, cities, props, game objects and many more, thus making the content generation process simpler, fluid, efficient, and complete.

The nature of Sceelix's node-based language is unique in that it focuses on high-level operations, such as creation, loading, modification, and export, which encapsulate a lot of complexity, such as best fit algorithms, constraint-based programming, or growth simulations. This complexity can be easily implemented using the underlying programming language, C#. In addition, small mathematical expressions, such as addition, multiplication, or trigonometric functions, are performed on the node parameters themselves, reducing the need for clutter in the graphs. Furthermore, with features such as encapsulation, which allows the reuse of full graphs as nodes within other graphs, the visual language becomes easier to navigate, understand, and manage.

Sceelix's team has developed a Unity plugin allowing direct communication with the Unity Editor. This integration allows data generated in the Sceelix Designer to be directly transmitted to an open Unity Scene or a prefab, providing a seamless integration of content generated in Sceelix into Unity’s game engine.

## 2.5 Summary

The State of the Art chapter examines Level Design, Procedural Content Generation, and Node-based Interfaces. The Level Design section discusses the development process and key pillars: optimization, gameplay, immersion, visuals, and functional design. PCG explores its origins, advantages, and different usages in games, while highlighting challenges in generating interior levels. The Node-based Interfaces section focuses on Visual Programming Languages (VPLs) and their use of graphs. The chapter concludes with a review of existing PCG solutions like Houdini, Blender, and Sceelix, all featuring plugins to address the problem.

**Table 1 - Comparison summary of identified PCG systems.**

| PCG SYSTEM | HOUDINI | BLENDER GEOMETRY NODES | SCEELIX |
|---|---|---|---|
| PRICING | Free for non-commercial use, starts at $269/year for an Indie license | Free | Free |
| GAME ENGINE PLUGIN | Supports Unity & Unreal Engine (requires paid license) | Paid 3rd party Unreal plugin | Supports Unity Engine |
| SOURCE CODE | C++ and Python | Python | C# |
| LICENSE | Closed-source | Open-source | Open-source |
| SIMULATION NODES | ✓ | ✓ | |
| TOTAL NUM OF NODES | More than 300 nodes, can create custom nodes using VEX language | Over 150 nodes, can create custom nodes using Python language | Over 60 nodes, can create custom nodes using C# language |

# 3 Methodology

A Gantt chart was used to plan and visualize tasks in this master's thesis. The project was divided into six tasks. The first task involved conducting a literature review on Game Level Design, Procedural Content Generation (PCG), and Node-based interfaces. The second task focused on creating visual prototypes using Figma to explore interface approaches for the proposed PCG solution. Study I (Task 3) involved interviews with PCG specialists to gather feedback on the prototypes and optimal user interface. Task 4 included designing the software architecture and implementing the solution in C#. Study II (Task 5) involved hands-on experimentation and user feedback through a questionnaire. The final task was dedicated to ongoing documentation of the research processes and results.

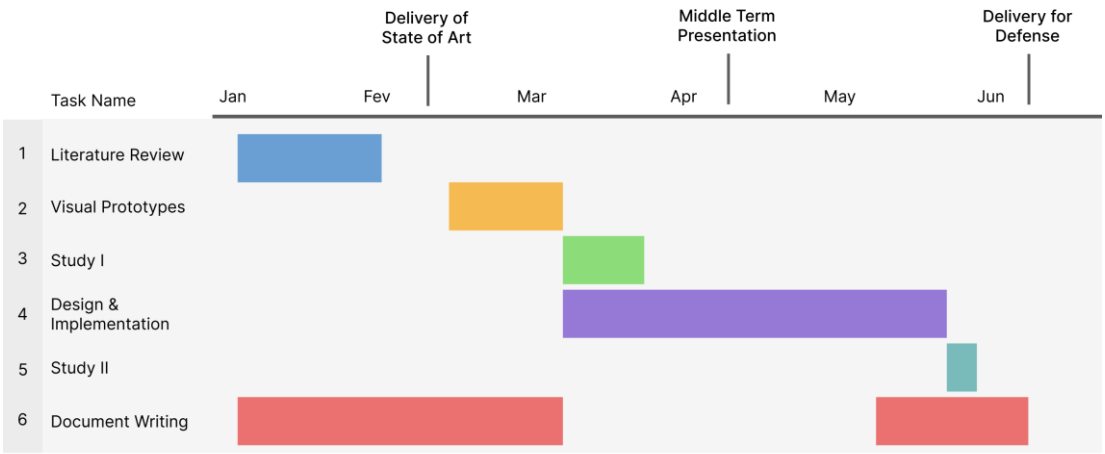



*Figure 8 - Gantt Chart of the dissertation project's tasks.*

## 3.1 Research Methodology

The chosen methodology for this research is the Action-Research Methodology, which follows a cyclical process of planning, acting, observing, and reflecting. It is a widely used approach that emphasizes active involvement and collaboration with stakeholders throughout the research process (Stringer et al., 2014)

In this methodology, the research is conducted in iterative cycles, involving the Design, implementation, and evaluation phases. Each cycle builds upon the insights gained from the previous one, leading to continuous improvement and refinement of the PCG system being developed.

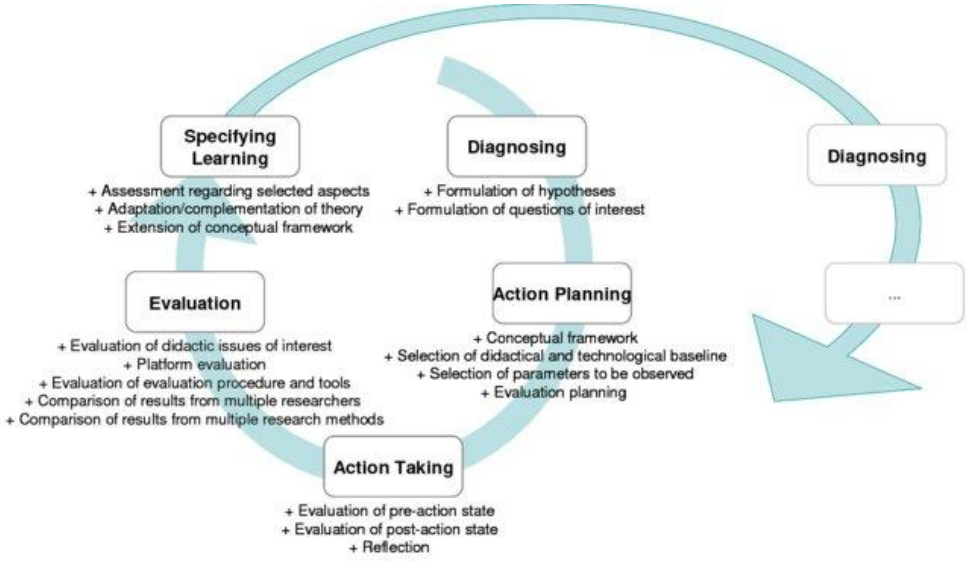



*Figure 10 - Action-Research Cycle. In FIGL, Kathrin et al. (2005).*

The iterative nature of the Action-Research Methodology is particularly relevant in the context of this project. By involving game developers as stakeholders, this approach facilitates the identification of design flaws at an early stage and the incorporation of feedback-driven features in subsequent iterations (Brydon-Miller et al., 2003). It allows for a more comprehensive understanding of the needs and preferences of the end-users, ensuring that the user interface of the PCG system meets their expectations.

Furthermore, the Action-Research Methodology helps bridge the gap between theory and practice by actively involving the target users in the research process. In this case, game developers are not only the participants but also key collaborators, providing valuable insights and expertise to inform the development of the PCG system (Coghlan

& Bannick, 2015). This participatory approach raises a sense of ownership among the target users and increases the likelihood of the system's successful implementation and adoption within the game development community.

By adopting the Action-Research Methodology, this project aims to create a PCG system that is not only theoretically grounded but also practical and user-centred. This methodology's iterative and collaborative nature enabled the inclusion of real user perspectives, leading to a more refined and effective solution for game developers.

## 3.2 Research Instruments

### 3.2.1 Study I - Interview

The first study, Study 1, consisted of individual semi-conducted interviews with a sample group of PCG experts. The interviews aimed to identify the system's focus and assess the potential benefits of a native PCG solution based on their experiences. The interviews, ranging from 20-30 minutes, were conducted in person or remotely via Zoom.

Participants were gathered through academic recommendations and community Discord Channels. The study spanned over two weeks and aimed to gather in-depth qualitative data. The interviews were audio recorded for accurate data analysis, with participants providing initial GPD permission consent.

Concurrently with the interviews, a guiding presentation was created to clarify the central research problem, and the proposed solution, and showcase two distinct visual interface approaches (Figure 11).

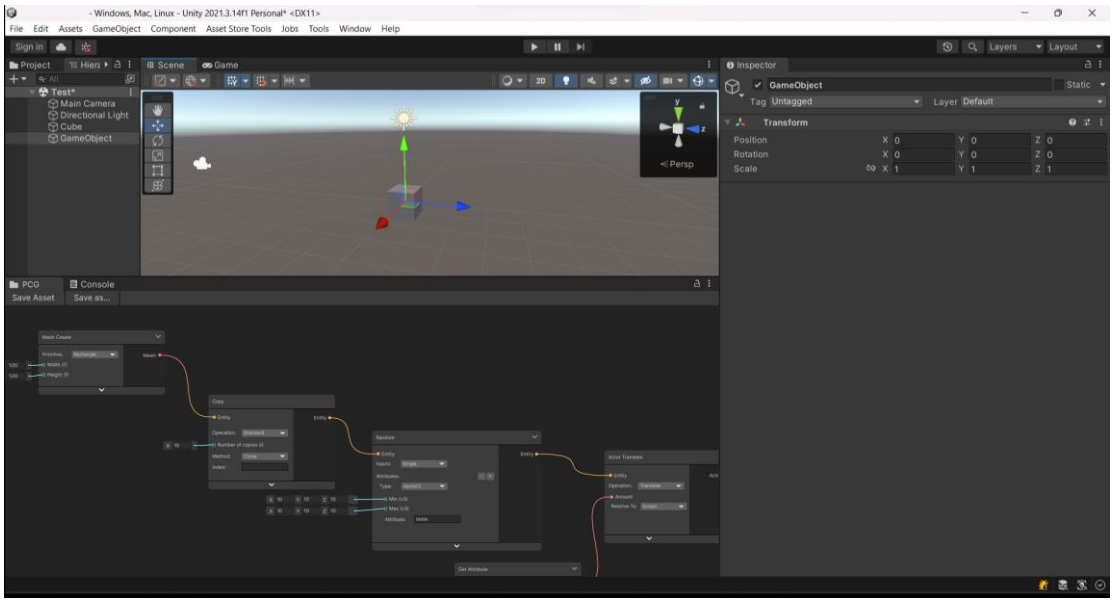


*Figure 11 - PCG Solution portrayed in Unity engine*

The interview began with an introduction, explaining the purpose of the interview. Participants were invited to introduce themselves and briefly describe their game development experience. Questions then explored participants' familiarity with PCG software, such as Houdini or Blender Geometry nodes, and its application context.

Key questions concerned the existing technological gap between PCG software and the development pipeline. Participants shared their perspectives on a potential native engine solution using a node-based approach and its benefits, specifically in which phase of game development it would be most useful.

UI feedback questions involved presenting visual representations of different UI approaches (Figure 12) for the node-based PCG solution. Participants were asked to provide feedback on the most suitable interface approach and express their preferences regarding attributes and global parameters handling within the selected approach.

The interview concluded with a summary of the key points discussed, allowing participants to provide additional comments or insights.

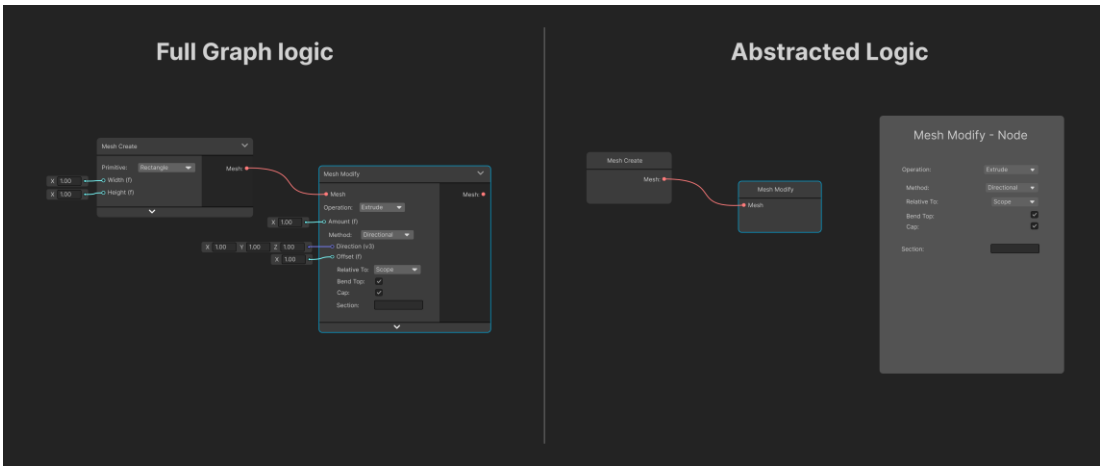



*Figure 12 - Visual prototype containing two distinct UI approaches*

### 3.2.2 Study II – User experience questionnaire

The second study was conducted to assess the user experience of the native PCG system through the usage of the System Usability Scale (SUS) questionnaire (Appendix 10.1). This widely recognized questionnaire is designed to measure the usability of a software system and gather feedback on its effectiveness, intuitiveness, and ease of use.

Participants, who were game developers, engaged with the software by following a quick start guide (preview in Figures 13 and 14, full content in Appendix 10.2) that provided an overview of working with the PCG. The guide familiarized them with the theoretical foundations of the generated data types and demonstrated the various essential interface operations and windows.

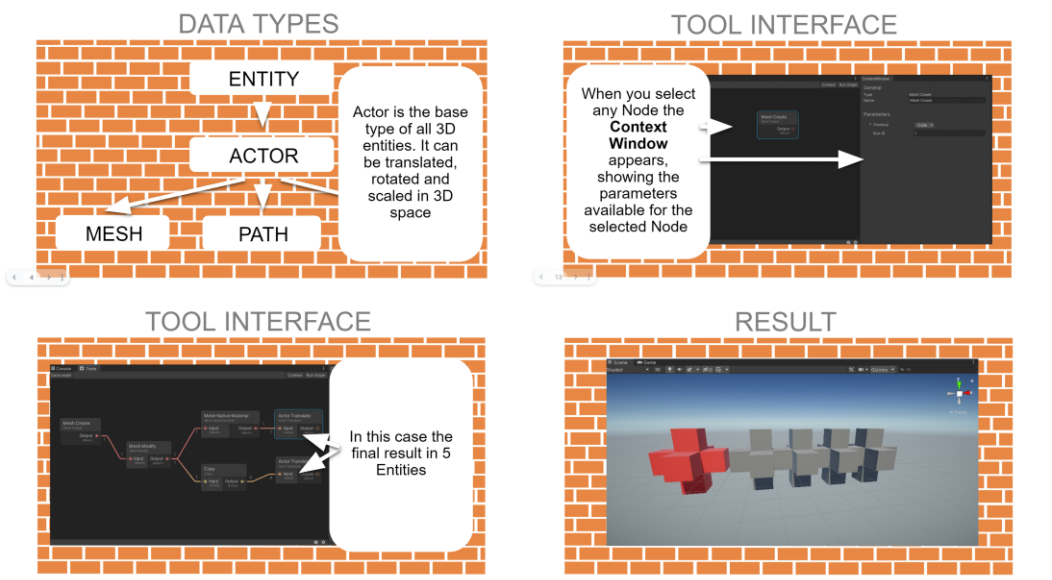



*Figure 13 - Various frames of the Quickstart Guide*

After completing the guide, participants were presented with a series of three practical exercises accompanied by video tutorials. These exercises focused on essential procedural concepts of the PCG, such as geometry creation, translation, copy, and randomisation, as well as the usage of custom expressions as parameters.

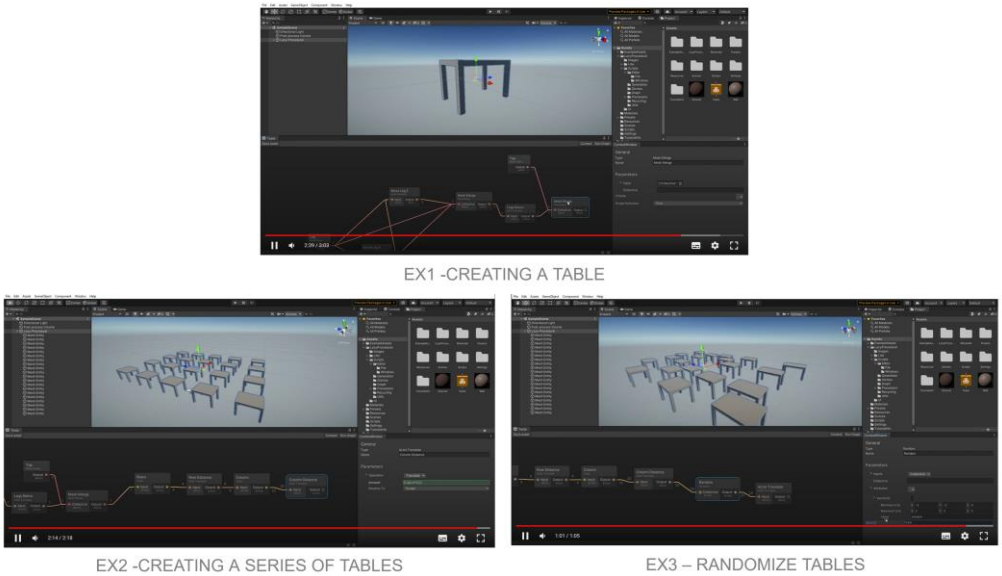



*Figure 14 - The three PCG exercises contained in the Quickstart Guide*

After completing the exercises, participants were presented with the SUS questionnaire. The questionnaire consisted of 10 statements, and participants rated their level of agreement with each statement on a Likert scale ranging from 1 (Strongly Disagree) to 5 (Strongly Agree). The statements covered usability aspects, including learnability, efficiency, user satisfaction, and error prevention.

Additionally, the SUS questionnaire included an overall score that participants could assign using the Likert scale. This score provided a summary evaluation of the overall usability of the PCG system. Furthermore, an open-ended question was included to allow participants to provide more detailed feedback and offer insights on currently implemented features that could be further tested and improved, as well as suggestions for new features that could enhance the user experience.

By combining quantitative ratings from the SUS questionnaire with qualitative feedback from the open-ended question, a comprehensive evaluation of the system's usability and user experience was obtained. The data collected from the questionnaire served as a valuable measure of the tool's effectiveness and provided meaningful insights for refining and enhancing the native PCG solution.

This questionnaire format played an essential role in measuring the user experience of the implemented PCG system and visualising the user’s solution direction for further improvements/refinements.

## 3.3 Summary

This chapter introduces the central research methodology and instruments used throughout the project’s research. The chosen methodology is the Action-Research Methodology, emphasizing active stakeholder involvement and collaboration. Two studies were conducted: Study I involved interviews with PCG experts to gather feedback on prototypes and user interface preferences, while Study II used a user experience questionnaire to assess the usability of the native PCG system. The iterative and collaborative approach ensures the practical and user-centred development of the PCG solution.

# 4 Analysis & Design

## 4.1 Solution Domain

### 4.1.1 Unity Engine

The Unity Engine provides a solid foundation for implementing additional custom tools/functionalities using the C# programming language. The namespaces ***UnityEngine*** and ***UnityEditor*** namespaces offer essential classes and functionalities for runtime and editor-mode development, respectively. For this PCG system, the *UnityEditor* namespace becomes particularly relevant as it focuses on editor-mode development.

Unity's *UIElements* namespace is employed to build the detailed interfaces of this system. *UIElements* is a modern UI framework that uses a structure similar to HTML and CSS. It allows for the creation of dynamic and responsive UIs by utilizing a combination of UXML (UI XML) and USS (UI Style Sheet) files.

UXML is an XML-based language that describes the structure and hierarchy of UI elements, defining their layout and interconnections. It provides a declarative approach to UI creation, allowing developers to specify the composition of the interface visually.

On the other hand, USS is a style sheet language used to define UI elements' visual appearance and style. It provides a set of rules and properties that determine how the UI elements are rendered, allowing for consistency and customization across the interface.

The *GraphView* library within Unity's UIElements namespace is especially relevant for this solution as it provides a powerful framework for creating node-based interfaces and interactions. The GraphView structure consists of various classes, including:

- **GraphView**: The GraphView class is the main container for the node-based interface. It handles the rendering and interaction of nodes, connections, and other graphical elements within the graph.
- **Node**: The Node class represents individual nodes within the graph. It contains methods for rendering the node, handling user input, and managing connections to other nodes.
- **Port**: The Port class represents input or output ports on nodes. It enables the connection of nodes by providing connection points for links.
- **Edge**: The Edge class represents connections between nodes. It handles the rendering and management of links between nodes.

### 4.1.2 Sceelix

Sceelix is divided into several components that work together to enable procedural generation. The **Sceelix Core** component provides the fundamental functionality for managing graphs, nodes, and their connections. It forms the backbone of the software and handles the execution and evaluation of the graph. **Sceelix Designer** is the graphical user interface (GUI) component that allows the users to create and edit graphs visually.

#### 4.1.2.1 Graph Concepts in Sceelix

**Entities**: In Sceelix, entities represent the objects or elements that the user wants to generate procedurally. Entities can be anything from buildings and landscapes to characters or textures. A graph represents each entity and contains nodes that define its characteristics and properties.

**Ports**: Ports are the connection points within nodes that allow data flow between nodes. They represent the inputs or outputs of a node and serve as the means to transfer information or values from one node to another. Ports enable the creation of a data flow network within the graph.

**Flow**: The flow in Sceelix refers to the order in which nodes are executed within the graph. It defines the sequence of operations and ensures that the dependencies between nodes are properly resolved. The flow concept ensures that nodes are evaluated in the correct order, considering any data dependencies.

**Parameters**: Parameters in Sceelix are used to control and customize the generation process. They allow users to define variables that can be adjusted to influence the outcome of the procedural generation. Parameters can be linked to nodes, enabling users to modify their values dynamically and observe the impact on the generated content.

#### 4.1.2.2 Node Overview in Sceelix

Nodes are the building blocks of Sceelix graphs and represent individual operations or functions within the procedural generation process. Each node has a specific purpose and contributes to the overall generation process. Some common types of nodes include:

**Generator Nodes**: These nodes are responsible for generating content, such as shapes, textures, or patterns. They define the core elements of the procedural system and play a crucial role in shaping the final output.

**Modifier Nodes**: Modifier nodes alter or transform existing content generated by other nodes. They provide a means to modify the properties or characteristics of entities or introduce variations to the generated content.

**Utility Nodes**: Utility nodes perform auxiliary functions and provide additional capabilities to the procedural system. They can include nodes for noise generation, mathematical calculations, or other specialized operations.

**Control Nodes**: Control nodes enable the creation of branching logic and conditional operations within the graph. They allow users to control the data flow and introduce decision-making processes during generation.

## 4.2 Functional & Non-Functional Requirements

This subsection presents the functional and non-functional requirements based on the FURPS classification system. FURPS stands for Functionality, Usability, Reliability, Performance, and Supportability.

Starting with the enumeration of functional requirements within the scope of functionality (letter F), actors and their interrelationships are depicted using a use case diagram. For each use case, its function and purpose are described in greater detail, taking the form of user stories. Finally, the non-functional requirements are outlined, encompassing the remaining fields (letters URPS). These requirements identify physical, implementation, interface, and design constraints that the solution demands.

### 4.2.1 Functional Requirements

A possible definition of “functional requirement" is the realisation of a need that a solution (software) should meet. These requirements are essential in the project design as they specify all the solution's functionalities.

Typically, when using the agile Scrum methodology, the term "use case" is adopted in gathering functional requirements. Subsequently, the solution's functionalities to be implemented are represented through a use case diagram (Figure 15). This diagram not only enumerates the established use cases but also specifies the actors to whom each case is intended and how they relate to each other.

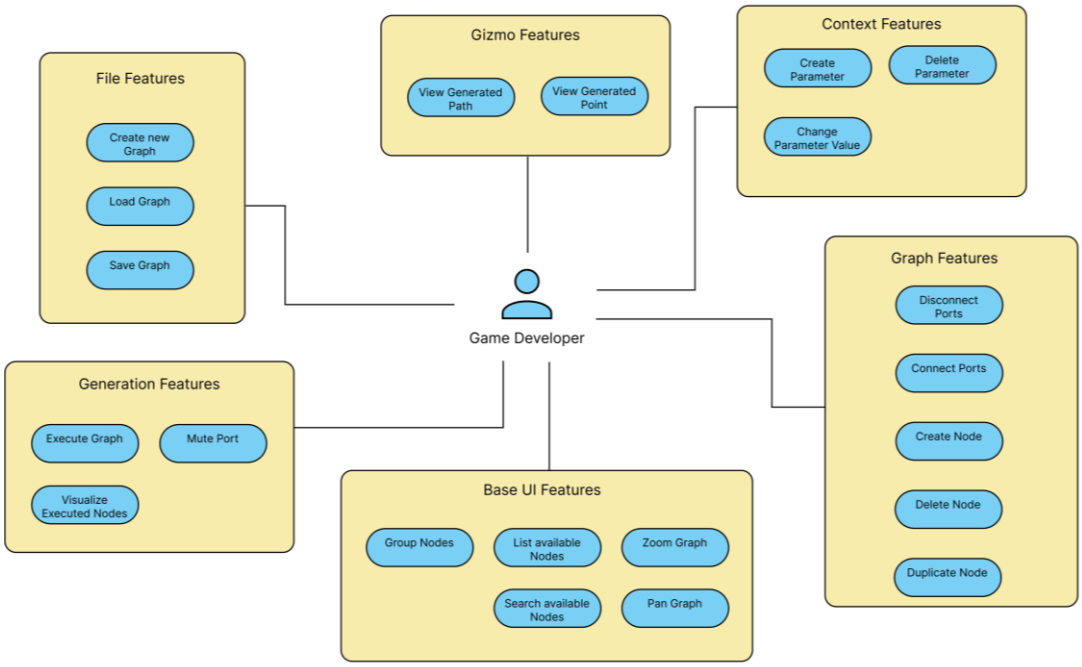



*Figure 15 - Use Case diagram of the project.*

The use case diagram reveals that the use cases can be categorized into six distinct groups:

- **File Features:** These features involve creating, loading, and saving graph files. They encompass use cases associated with graph persistence and file visualisation, as the graph file serves as the foundation for all interactions within the tool.
- **UI Features:** This category includes features that pertain to common user interface interactions, such as zooming and panning the graph, creating node groups, and browsing and searching for available nodes to add.
- **Graph Features:** These features are centred around modifying the graph's structure. Use cases in this group include creating or deleting a node, establishing or removing links between node ports, and duplicating nodes.
- **Context Features:** This group encompasses features related to a side helper window that displays information about the selected node and its parameters. Use cases in this category involve visualizing and modifying node parameters, converting a node into an expression, and adding or removing subparameters within an addable parameter list.

- **Generation Features:** These features are associated with generating the content of the created graph. Use cases within this group involve executing the generation process visualising the processed nodes, and muting individual node ports.
- **Gizmo Features:** This group covers use cases related to the debug visualisation of generated data that lacks an inherent visual representation, such as paths and points.

### 4.2.2 Non-Functional Requirements

The remaining constraints on the development of the solution, imposed by the software and hardware context, which do not represent any tangible functionality, are classified as non-functional requirements.

Continuing with the remaining letters and symbols of the FURPS classification system, the non-functional requirements of each category are presented accordingly.

#### 4.2.2.1 Usability

The PCG tool aims to provide a user-friendly and intuitive interface to enhance the usability and accessibility of procedural content generation. The user interface (UI) design should align with the overall default UI ecosystem of Unity while incorporating elements from the Sceelix library to ensure a consistent and cohesive user experience. The actions required to create and modify procedural graphs should be clear and well-guided, allowing users to navigate and interact with the tool easily. The PCG tool should provide informative tooltips, contextual help, and documentation to assist users in understanding the functionality and purpose of each node, port, and parameter.

#### 4.2.2.2 Reliability

The PCG tool must exhibit high reliability to ensure consistent and accurate content generation. It should handle changes to the graph structure and parameter values changes, without data corruption or unexpected behaviour. The tool should prevent the creation of infinite loops within the graph by enforcing safeguards and validation mechanisms to detect and prohibit cyclic connections. Furthermore, it should include mechanisms to handle errors and exceptions gracefully, providing informative error

messages to users when issues occur during graph generation or data processing. The PCG tool should prioritise stability and robustness, minimising crashes, freezes, and other unexpected failures.

#### 4.2.2.3 Performance

The PCG tool should be optimised for performance to enable efficient and responsive content generation. It should aim for fast reload times when modifying the graph structure or adjusting parameter values, allowing users to iterate quickly during content creation. The tool should leverage caching and parallelisation techniques, where applicable, to expedite content generation and improve overall performance.

#### 4.2.2.4 Supportability

To ensure the long-term support and maintenance of the PCG tool, it should adhere to industry-standard practices and guidelines. The tool should be compatible with the recommended specifications of the Unity engine, ensuring optimal performance and compatibility with the target platform. It should specifically support the Unity LTS versions (2020.3 and 2021.3), as these are widely adopted and known for their stability. The PCG tool should also consider the compatibility of the libraries it depends on, as not all C# libraries are compatible with Unity. It should prioritise using libraries that are well-maintained and actively supported by the developer community to minimise compatibility issues and ensure ongoing support.

## 4.3 Software Architecture

As described above, this solution relies on the underlying structures of both Unity and the Sceelix libraries. Unity serves as the foundation for the user interface (referred to as the "front-end"), while the Sceelix libraries handle the generation and processing of data (referred to as the "back-end"). The construction of this solution is centred around integrating and utilising these two components.

Figure 16 provides a class diagram showcasing the overall code architecture of the PCG system, offering a visual representation of how the central components are organised and interconnected.

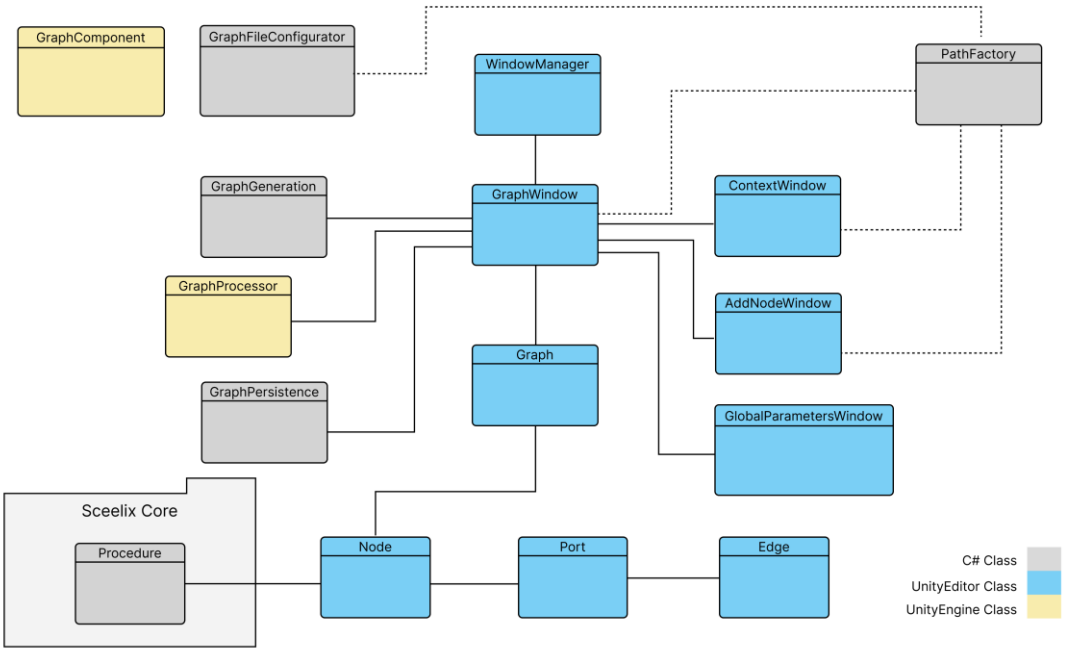



*Figure 16 - Class Diagram of the overall solution's structure*

Starting from the **WindowManager** class, this class is responsible for creating and managing all the editor windows of this system, especially creating **GraphWindow**. This class is the container of all the visual components within the node-based editor with their respective layout. This window has various child windows, that serve as auxiliary panels to this system's interface (such as the **ContextWindow**, **AddNodeWindow**, and **GlobalParametersWindow**). However, the central window that holds the major and

central graph interactions is the **Graph** class (that inherits from the Unity **GraphView.Graph** base class). This class contains a list of Node instances and all the information, which in turn have the necessary visual information of its position, ports, and connections. Each **Node** class contains a **Procedure** class which is the Sceelix data necessary for the execution/generation of the PCG.

## 4.4 Visual Identity

The visual identity of a software tool plays a crucial role in its success, as it serves as the face and representation of the tangible solution. In the context of the PCG tool developed in this project, a well-designed visual identity is necessary to enhance its visibility, establish a strong brand presence, and make it more approachable to users. This chapter delves into the process of designing the visual identity of the PCG tool, highlighting the importance of creating a clear and compelling brand that aligns with the visual aesthetics of the game engine.

### 4.4.1 Name and Logo

The central name chosen for the PCG tool is "Lazy Builder," which embodies the concept of effortless level construction. The name itself conveys the idea of building without excessive effort or complexity. By selecting this name, the intention is to attract users who seek a user-friendly and efficient tool for creating game levels.

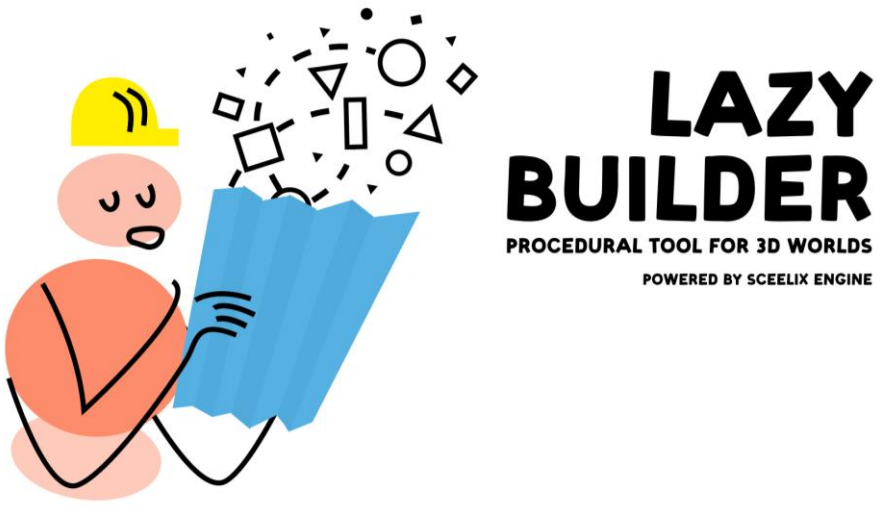



*Figure 17 - PCG tool main logo design.*

The "Lazy Builder" logo serves as a visual representation of this concept. It features a construction worker avatar holding a blueprint sheet, symbolizing the node-based editor and signifying the planning and organization involved in the level construction process. This combination of elements effectively communicates the tool's core functionality and highlights its ease of use.

The "Zyzol" font exhibits a playful yet professional style, evoking a sense of warmth and approachability. Additionally, the color palette employed in the visual identity revolves around orange hues. The incorporation of orange colors within the visual identity establishes a strong connection to building bricks, a concept reinforcing the tool's core purpose.

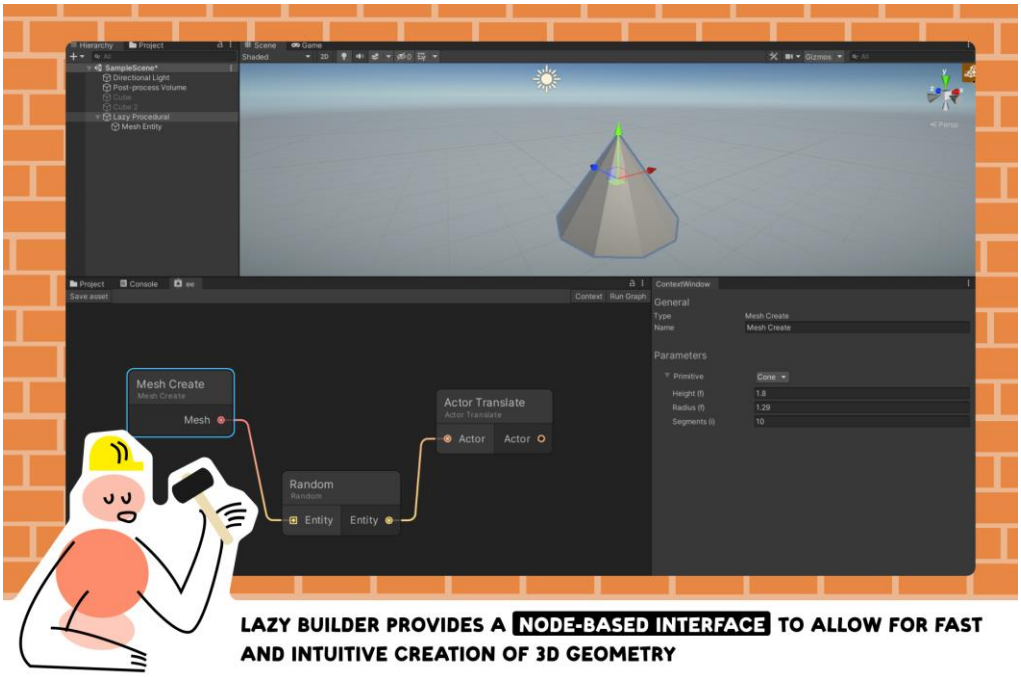



*Figure 18 - PCG tool description image.*

### 4.4.2 Visual Interface

The visual interface of the PCG tool draws inspiration from two key sources: the structure of Sceelix's Editor and the popular node-based tool in Unity Engine known as Shader Graph. By incorporating elements from these established tools, the PCG tool's interface benefits from familiarity and aligns with standard practices, facilitating user adoption and ease of use.

While the structure of Sceelix's Editor provides a foundation for the PCG tool's interface, it has been further refined and optimized to meet the specific requirements of level design. This ensures that users familiar with Sceelix will find commonalities and recognize certain features, allowing for a smooth transition and integration into their workflow.

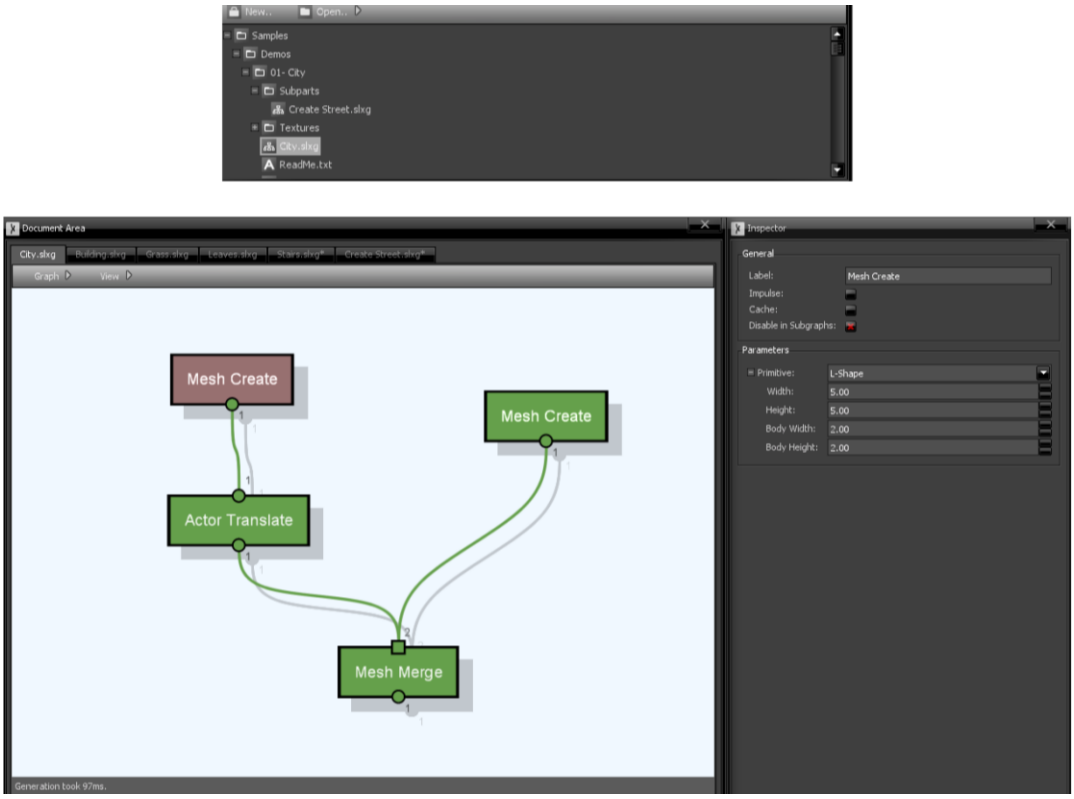



*Figure 19 – Sceelix's Nod-based (Bottom) and Project (Top) windows.*

Additionally, the PCG tool's interface takes cues from the Shader Graph, a widely used and respected node-based tool within the Unity Engine ecosystem. By adopting similar interface elements and design patterns, the PCG tool benefits from the intuitive and user-friendly nature of Shader Graph. This familiarity allows users already experienced with Shader Graph or other node-based tools in Unity Engine to quickly grasp the PCG tool's interface and leverage their existing knowledge.

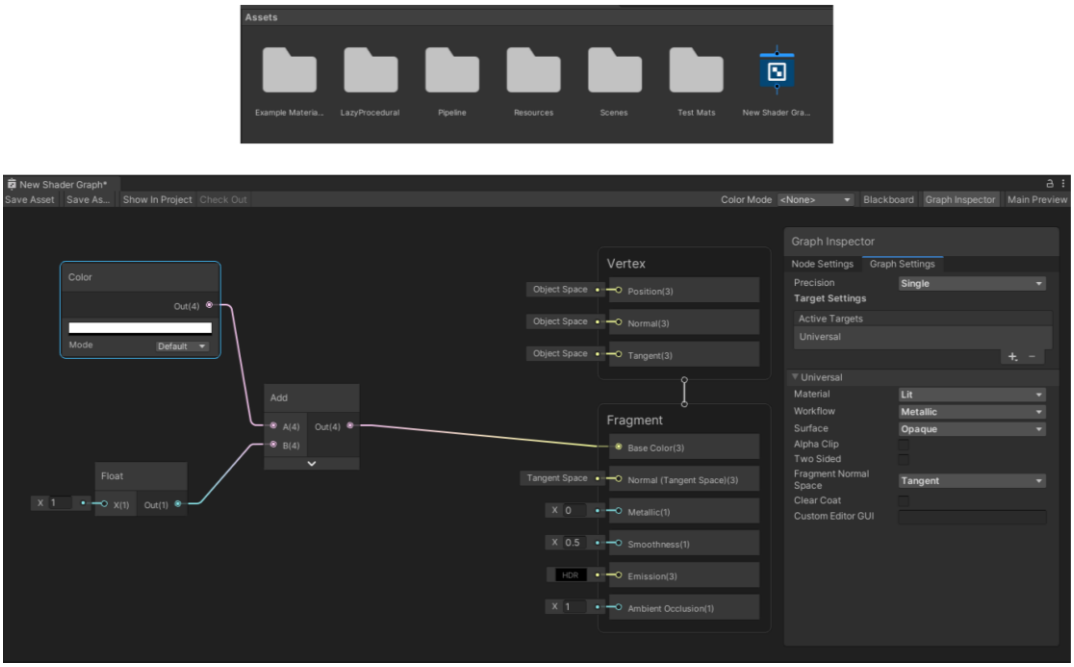



*Figure 20 – Unity Shader Graph's Node-based (Bottom) and Project (Top) windows.*

The graph icons used in the PCG tool's interface have been carefully designed to resemble a variant of the Shader Graph, incorporating the brand's colour palette. This deliberate choice not only maintains visual consistency with the overall visual identity but also ensures that the interface feels cohesive and aligned with the tool's purpose.

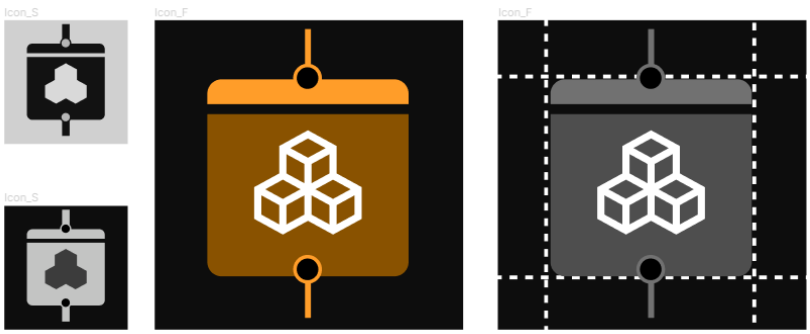


*Figure 21 – Lazy Builder file icons.*

By blending elements from Sceelix's Editor and taking inspiration from Shader Graph, the PCG tool's visual interface strikes a balance between familiarity and innovation. Users will find a comfortable and intuitive environment that encourages creativity and productivity while being visually consistent with industry-standard practices. This cohesive and thoughtfully designed visual interface enhances the overall user experience and contributes to the successful adoption and usage of the PCG tool.

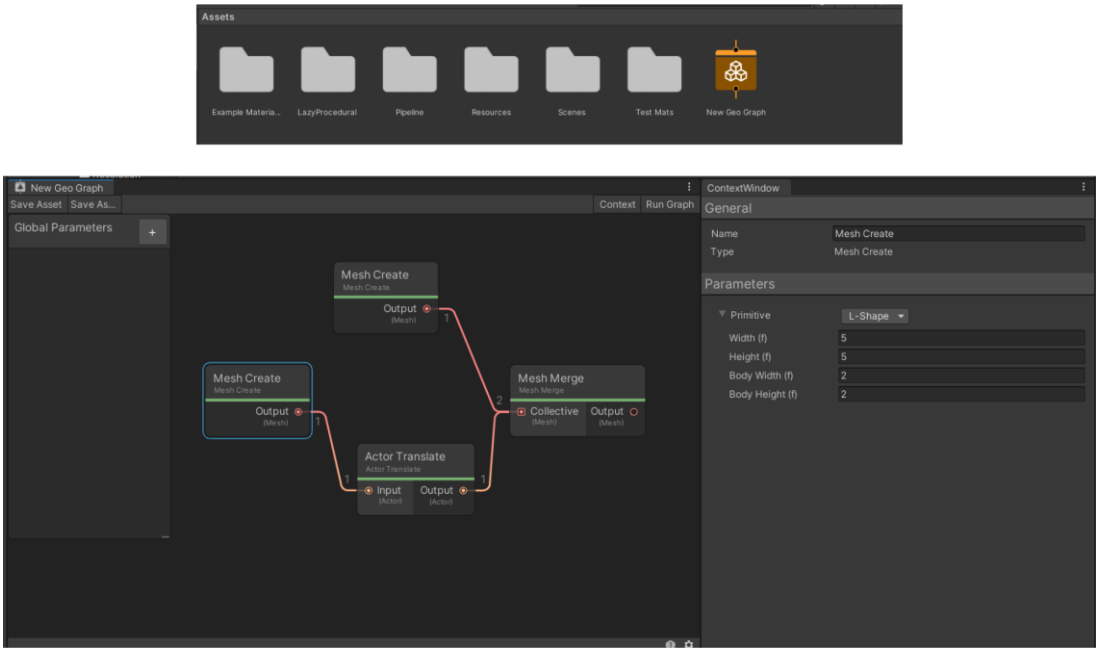



*Figure 22 – Lazy Builder's Node-based (Bottom) and Project (Top) windows.*

## 4.5 Summary

This chapter provides an overview of the solution domain, focusing on the Unity Engine and Sceelix components. Unity Engine offers essential functionalities for development, including UIElements for dynamic UI creation. Sceelix enables procedural generation with graph-based entities, ports, flow, and parameters. Nodes play a vital role in the generation process, categorized as generators, modifiers, utility, and control nodes. The chapter outlines functional and non-functional requirements based on the FURPS+ classification system, covering usability, reliability, performance, and supportability. The software architecture integrates Unity and Sceelix components, facilitating interactions within the node-based editor. Finally, the chapter explores the visual identity of the PCG tool, encompassing the name, logo, icons, and visual interface design, which aim to enhance brand recognition, user experience, and alignment with industry standards.

# 5 Implementation

This chapter presents the implemented use cases chronologically, grouped by their respective feature subset. Each subchapter provides insights and explanations on the implementation process, aligned with the initial design made.

## 5.1 Base UI & Graph Features

The first set of tasks focused on establishing the foundation for the base and graph-specific UI interactions. A functional prototype containing placeholder content was developed with the essential graph data structures (Nodes, Ports, and Edges). The base UI features such as zooming and panning, and the graph functionalities of Node creation/deletion, Edge creation/deletion, and Node duplication were implemented.

To enhance user error prevention, a validation function was implemented during the edge connection process. This exact implementation disables Ports that either have incompatible data types or could form an infinite closed loop.

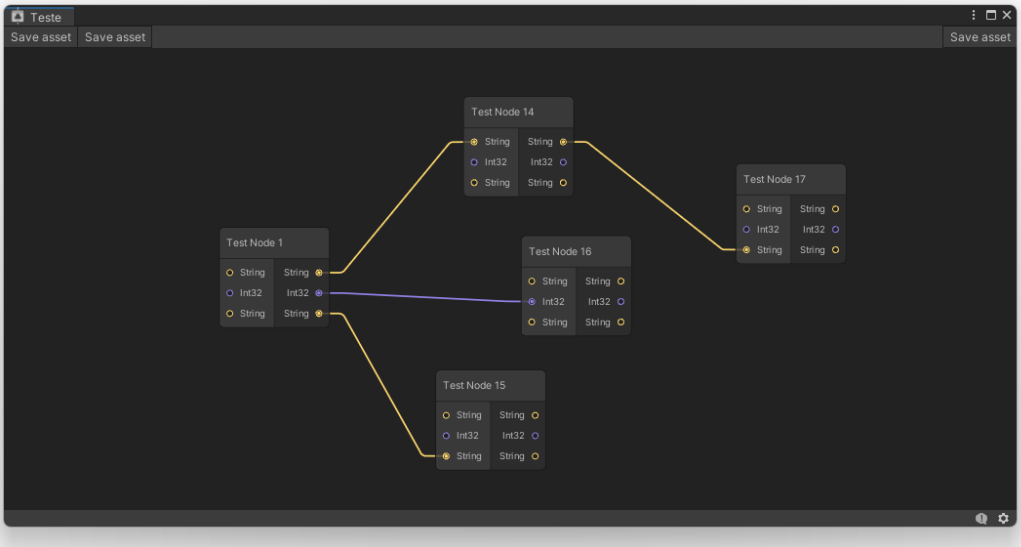



*Figure 23 - Node-based interface with placeholder data.*

## 5.2 Generation Features

The second set of tasks encompasses all features related to the complete procedural generation process, including the PCG algorithms to transform the graph network into geometry data and subsequently its materialization into Unity objects.

### 5.2.1 Sceelix Integration

Before proceeding with the generation classes' development, the Sceelix Core library was integrated into the Unity Engine. The source code was added to the project, and during this process, compatibility issues arose concerning external library dependencies and code compatibility. While Sceelix was developed using C# and .NET, it was initially designed as standalone software using the .NET Framework, a subset of .NET. Consequently, certain adjustments were made to the source code to ensure the successful compilation of Sceelix within the Unity environment.

### 5.2.2 Graph Processing

For the graph processing, it is portrayed in Sceelix's documentation that every Procedure (attached to a Node in a 1:1 proportion) can be executed individually, resulting in a list of Entity instances. In a case of a Node that contains inputs, it is necessary to execute beforehand the Procedures of all inputs Nodes and attach the result to the current Procedure. Only then the current Node's Procedure can be performed correctly.

When integrating Sceelix, a specific Procedure type entitled IndependentGraph Procedure was discovered. This Procedure enables the appending of Graph data to encapsulate the generation process. Similar to the implemented UI class, this Graph comprises a list of Sceelix Nodes, each containing Ports and their respective connections. To utilise this Procedure type, Sceelix Nodes need to be created from each Procedure and connected.

However, challenges arise when dealing with this Procedure type, particularly in a scenario that frequently adjusts Procedure parameter values. The Sceelix Nodes created from each Procedure do not reflect the modifications made to the source Procedure, nor can they be directly modified. Consequently, the Graph needs to be reconstructed for every minor modification, leading to a significant delay between the parameter value change and the Graph execution.

Considering that responsiveness and rapid feedback are the central points of this native solution, this approach was discarded and a new generation process needed to be implemented. There were four essential rules for this generation process:

1. Execute all valid Procedures.
2. Execute Procedures with inputs only after the respective Inputs have been executed.
3. Store the outputs of the final Procedure, as they represent the result of the Graph's execution.
4. Ensure that this implementation achieves faster execution compared to IndependentGraph’s approach.

For very simple graph scenarios a Depth-first search algorithm (DFS) can comply with the established rules. This algorithm involves traversing each branch individually from its root Node, executing its Procedure and appending the result to the input of the next Node’s Procedure. However, when the graph contains multiple branches that merge into a single one, a limitation arises with the DFS approach.

In the provided diagram (Figure 24), the DFS algorithm would traverse the nodes in the order "A, B, C, D, E, F, D, G, D". Since the Procedure of Node D requires input data that is not yet available during the first and second visits, executing it becomes impossible. To address this, a simple validation can be implemented to prevent execution and exit the branch in case of missing inputs.

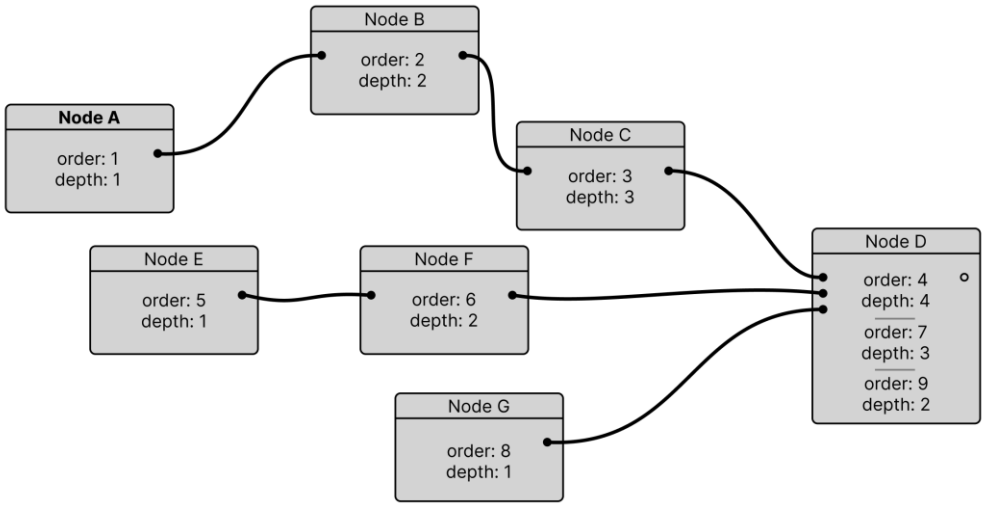



*Figure 24 - Example portraying the flow of a Depth-first Search algorithm.*

Considering the specified fourth rule and the fact that the DFS algorithm has a time complexity of O(N+E), where N represents the number of Nodes and E of Edges, further improvements can be made. Instead of running this algorithm every time, on every parameter value change, a matrix of Node execution orders can be calculated only when the graph structure changes. The first-pass algorithm groups and sorts Nodes based on their maximum visited depth, resulting in a matrix-like 1-(A, E, G); 2-(B, F); 3-(C); 4-(D), as shown in Figure 25. Subsequently, the second pass, triggered by a parameter value change, iterates linearly (with a time complexity of O(N)) over all Nodes and appends the results of the last execution order group. This optimization reduces the computational effort required for the Graph execution.

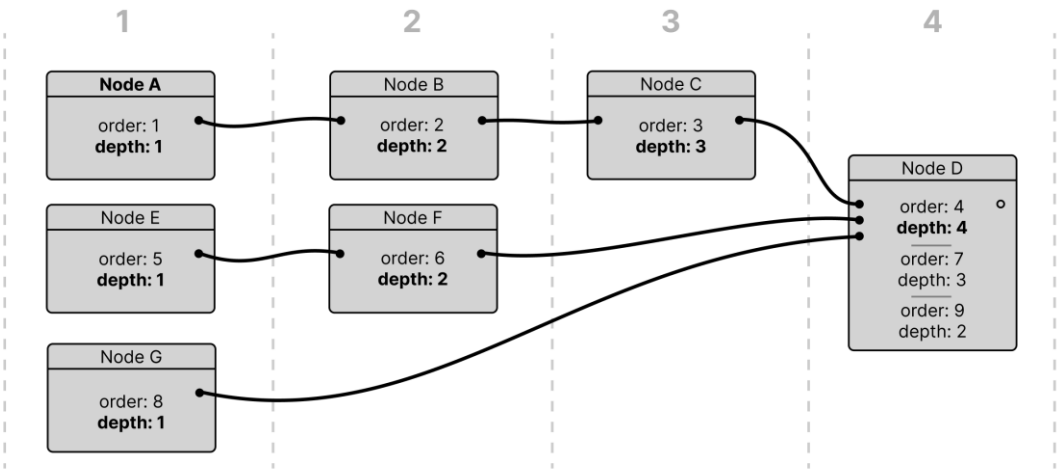



*Figure 25 - Example portraying the process of creating a matrix of Node execution orders.*

### 5.2.3 Content Population

After the graph is processed and a list of Entities is obtained, it is necessary to materialise the results into tangible Unity objects and geometry data. This graph population process initiates by identifying the Entity type and directing it to the respective Populator. Table 2 provides an overview of the implemented Populators for different data types.

**Table 2 - Implemented Entity Populators.**

| ENTITY TYPE | MATERIALISED OUTPUT |
|---|---|
| ENTITY | Empty Gameobject |
| MESH ENTITY | Gameobject with MeshRenderer and MeshFilter components |
| PATH ENTITY | Gameobject with the custom Path component |
| POINT ENTITY | Gameobject with custom Point component placed on its specified position |
| SURFACE ENTITY | Gameobject with Terrain component |
| GAMEOBJECT ENTITY | Gameobject containing a cloned instance of the specified source Gameobject |

Considering that generation performance is highly important for a rapid response to parameter adjustments, it becomes impractical for graph Populators to constantly create and destroy Gameobjects for every minor change.

To address this issue, a GameObject recycling system was implemented. This system stores all previously created objects, and, during a new graph regeneration, it attempts to find the best-fitting existing objects one by one. The selection is based on the minimum number of components that need to be added or removed. When a perfect component match is not made, the method creates and removes the necessary components to align with the new requirements; while generating any missing Gameobjects. Finally, any surplus objects that are no longer recycled are respectively destroyed.

## 5.3 Context Features

A context window and its interactions were implemented to facilitate the visualisation and editing of Node parameters. This window dynamically displays the parameters based on the selected node. A recursive UI builder method was implemented because these Node parameters can have subparameters, which can further contain additional subparameters.

When a parameter contains subparameters, a foldout is created to group all the child parameters. The parameter types encompass a wide range, including primitive types such as floats, integers, booleans, and strings as well as more complex types such as GameObjects, Textures, Colors, and Materials.

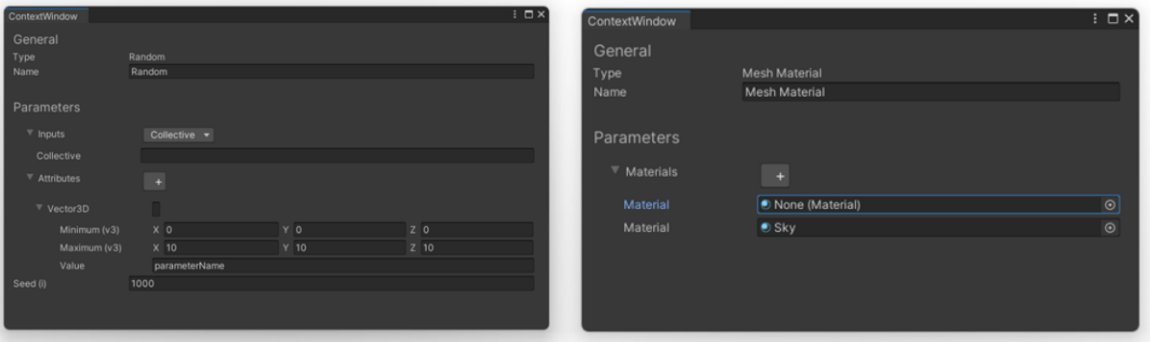



*Figure 26 - Example of two different Context Window exposing their Node's parameters.*

## 5.4 File Features

To preserve the graph's structure and progress, a custom file type named ".GeoGraph" was created. This file format serves as a container for all the essential graph data, which is serialised into JSON (JavaScript Object Notation). Using JSON allows for lightweight and efficient storage/retrieval of its information. By implementing this custom file type, users can save and load their graph projects, enabling seamless continuity and persistence of their work.

## 5.5 Gizmo Features

As previously mentioned, the procedural graph can produce Path and Point Entities that do not have a direct counterpart in Unity's components. These data serve as intermediary representations for placing or generating geometry data. To address this, custom components were developed for both types of Entities. These components utilise Gizmos to visually represent their respective spatial data (see Figure 27).

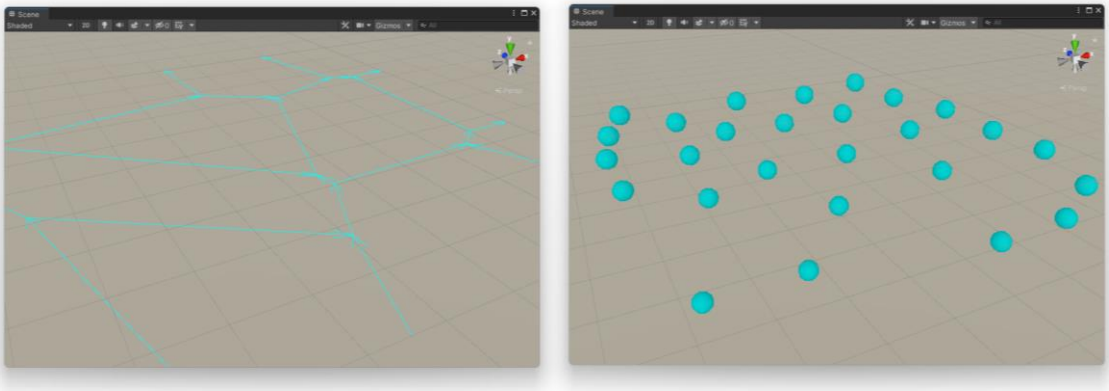


*Figure 27 – Implemented Path and Point Gizmo components.*

## 5.6 Summary

This chapter explains the overall implementation process of the use cases in chronological order, grouped by their respective feature subset. It covers the establishment of the base UI and Graph features, the integration of the Sceelix Core library, the generation process and its optimization, the content population, the Node context window, the file persistence, and gizmo features for additional visual data representation. All the designed functional requirements were successfully implemented, and the overall non-functional requirements were also met.

# 6 Results and Discussion

## 6.1 Study I

As referred to in section 3.2.1, study I involved conducting semi-structured interviews in PCG. Four specialists were interviewed, and their responses were systematized through a careful analysis of the audio-recorded data. By comparing the backgrounds of each interviewee and summarizing their responses to the two central questions, the overall answers were organized into Table 4.

**Table 3 - Study I Interviewees' overall answers systematized.**

| Question | Opinions | | | |
|---|---|---|---|---|
| | Intrv 1 | Intrv 2 | Intrv 3 | Intrv 4 |
| Participant Experience | Programmer (simulations/games) | Programmer (computer graphics/games) | Artist (computer graphics/games) | Programmer & Technical Artist (games/simulations) |
| Game Engines | Unreal & Unity | Unity | Unreal | Unreal |
| Used existing PCG software | Houdini (its team) | Houdini, City Engine, Sceelix, SpeedTree, | Blender Geo Nodes, Houdini | Blender Geo Nodes, Sceelix |
| Created custom PCG implementations | Procedural Grammars (L Systems) | Procedural modelling for runtime parametric usage | Super formula and L Systems | Unreal Mesh Components for procedural mesh manipulation |
| Relate or agree w/ the stated thesis central problem | Agrees (specially for artists with low level of programming knowledge) | Agrees that this problem is related with the overhead of communication | Agrees (specially during runtime mode to see the PCG changes) | Agrees (specially for runtime procedural generation) |
| Full graph or context abstracted UI approach? | Context free (can simplify and help reduce the clutter of the code) | Context free (can be easier to maintain complex graphs and dense nodes) | Both approaches (Context free can be easier to maintain complex graphs and more efficient to experienced users) | Both approaches ideally for tackling scenarios. Nodes would have parameters but could also be collapsed for a minimal view |

### 6.1.1 Central Question I

Regarding the first central question, experts unanimously acknowledged the existence of the identified research problem and agreed that a native solution would effectively respond to it. Notably, participants with backgrounds in art or technical art emphasized the proposed solution’s potential, particularly for runtime procedural content generation, as it mitigates the requirement for programming skills to modify geometry data during runtime since standalone solutions cannot provide such control. Additionally, one interviewee highlighted the absence of a PCG system that can effectively utilize the native assets of game engines, which they consider a fundamental gap currently in the industry.

### 6.1.2 Central Question II

Regarding the second question, all experts agreed that as the graph size scales up, it becomes challenging to manage and visually cluttered. Some participants even humorously referred to the phenomenon as "spaghetti code," drawing a parallel to Unreal Engine's Blueprint system. However, two interviewees acknowledged the benefits of the full node approach, especially for users with limited knowledge of procedural tools and workflows or when working with minimal and straightforward graphs. The first interviewee noted that understanding graph dependencies can be challenging with a context-free approach since one needs to delve into the node's content to ascertain them. The second interviewee attributed the context-free interface in their PCG implementation (Sceelix) to numerous options within a single node, leading to visual clutter. The third interviewee expressed concerns about the UI space required for writing expressions in a full graph approach, which they considered disruptive.

### 6.1.3 Open-ended Question

The interviewees provided diverse responses regarding the open-ended question about fundamental user experience features that would benefit by being implemented. However, several recommendations emerged as common themes among them, including the following:

- “It would be useful to have a system that could convert from Houdini and other popular PCG systems to the internal ones.”
- “What I would like to see is an error report based on the Nodes and not on a global console.”
- “In Unreal Blueprints allows creating a certain point of control to organize spatially the connection that would be a useful feature to have.”
- “On a project that uses versioning it is a must that graphs are serialized into readable text.”

### 6.1.4 Discussion

The insights gathered from the study indicated a consensus among the specialists regarding the reality of the identified problem and the necessity to address it. Most participants favoured a context-free user interface approach, considering its usability and accessibility. Additionally, the challenges associated with managing complex graphs were acknowledged, with the metaphor of "spaghetti code" humorously highlighting the confusion and visual clutter that can arise.

## 6.2 Study II

The second study was conducted later to the completion of the solution’s implementation, as referenced in Section 3.2.2. This study involved ten participants with diverse backgrounds in the game development field. While most participants completed the exercises in person during the "Game Dev Meet" event, where game developers gather to share projects and establish connections, some participants opted to complete the exercises remotely. Remote participants were required to install the PCG system in advance and follow the tutorial guide and exercises through an online version. To ensure consistency and gather honest feedback, no additional assistance was provided apart from the content presented in the guide. However, if participants encountered obstacles

or errors that hindered their progress, limited support was given to address those issues. The study's data-gathering process consisted of three distinct formats:

1. Quantitative questions: Participant backgrounds and the System Usability Scale (SUS).
2. Written feedback: Open-ended questions where participants provided justifications and highlighted software features and issues.
3. Observational notes: Documentation of user actions and behaviours during in-person exercises.

The following subchapters will discuss the results of each format accordingly.

### 6.2.1 Quantitative Questions

This subchapter delves into the quantitative questions used to gather data in Study 2. The first set of questions focuses on gathering background and experience information from the participants in the game development field.

The next set of questions revolves around the System Usability Scale (SUS), a widely used questionnaire designed to assess the usability of a system. These ten questions aim to evaluate various usability aspects, including learnability, efficiency, and satisfaction with the PCG system. Participants were asked to rate their agreement with each statement on a Likert scale ranging from 1 (Strongly Disagree) to 5 (Strongly Agree).

Lastly, the participants were asked to provide an overall rating for the PCG system. This question aimed to capture their overall impression and satisfaction with the system. In the following sections of this chapter, we will address each question individually and provide detailed commentary on the results obtained from the quantitative data.

### Question 0.1 - Age

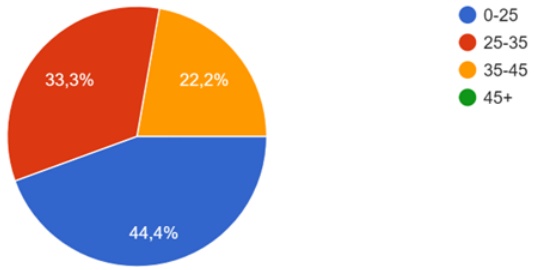



*Figure 28 -Data obtained for question 0.1 Age*

Based on the age distribution of the participants (Figure 28), the majority (44.4%) fall within the age range of less than 25 years. The second largest group consists of participants aged between 25 to 35 years, accounting for 33.3% of the respondents. Participants aged from 35 to 45 years old make up 22.2% of the sample, while there were no respondents above the age of 45. These results suggest that the study gathered a relatively younger sample audience, which could have impacted the interpretation of the findings regarding generational perspectives and preferences in the game development field.

### Question 0.2 - Current Academic Degree

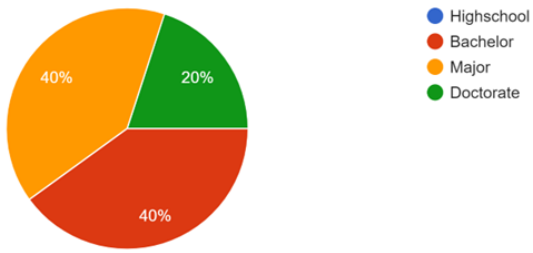



*Figure 29 - Data obtained for question 0.2 Current Academic Degree*

For the second contextual question, 40% of the interviewees answered that they have a Bachelor's degree, while another 40% have a Master's degree, and 20% of the participants hold a Doctorate. There were no respondents with a high school education. These findings suggest that the study attracted participants with various levels of academic knowledge, including both undergraduate and postgraduate qualifications. This academic diversity should contribute to a more nuanced understanding of the results, including perspectives from different educational backgrounds.

**Question 0.3 - Experience in Game Development**

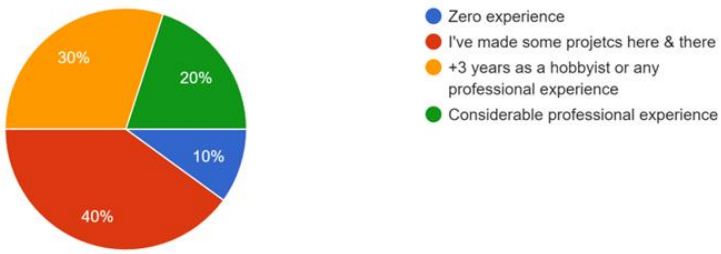



*Figure 30 - Data obtained for question 0.3 Experience in Game Development*

The results show that the participants' experience in game development varies. 10% of the respondents reported having no experience in game development. 40% mentioned having worked on a few projects occasionally. Another 30% indicated having more than three years of experience either as a hobbyist or professional. The remaining 20% reported having considerable professional experience in game development. These results suggest diverse experience levels among the participants, which could provide valuable insights from different technical perspectives.

### Question 0.4 - Experience with Procedural Tools

“Do you have any experience with Procedural Modelling Software (such as Houdini, Blender Geo Nodes, etc...) or have you implemented any custom Procedural Systems?”

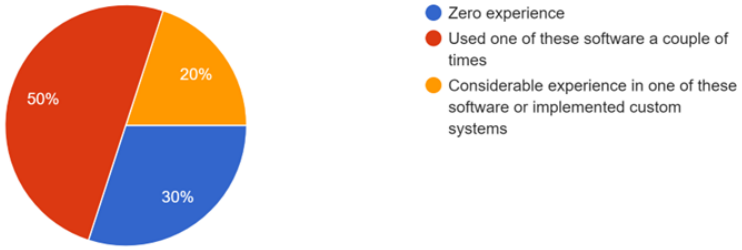



*Figure 31 Data obtained for question 0.4 Experience with Procedural Tools*

The results indicate that 30% of the participants reported having no experience with procedural modelling software or implementing custom procedural systems. 50% mentioned having used one of these software tools a couple of times, indicating moderate familiarity. Interestingly, 20% of the participants reported having considerable experience with one of these software tools or having implemented custom procedural systems, suggesting a higher level of expertise in this area. These findings suggest a mix of participants with varying degrees of experience with PCG systems, which contributes to a more comprehensive understanding of the system's intuition in different degrees of familiarity with similar systems.

### Question 1 - Future Usage

“I think that I would like to use Lazy Builder frequently.”

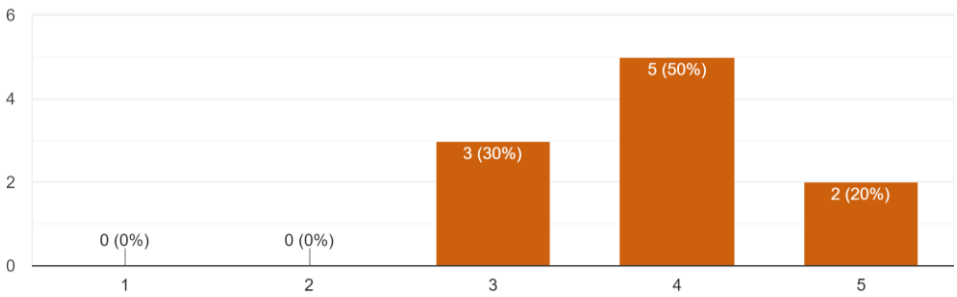



*Figure 32 - Data obtained for question 1 Future Usage*

The results of SUS Question 1 indicate that 30% of the participants felt neutral about their likelihood of using Lazy Builder frequently. However, 50% agreed or strongly agreed that they would like to use it frequently, indicating a positive inclination towards future usage. Interestingly, 20% strongly agreed with the statement, suggesting high interest in incorporating Lazy Builder into their workflow. These findings indicate a generally positive perception of the system's potential for future usage among the participants. The high percentage of agreement and strong agreement suggests that Lazy Builder can potentially be a valuable tool in the participants' game development processes.

### Question 2 - Complexity

“I found Lazy Builder unnecessarily complex”

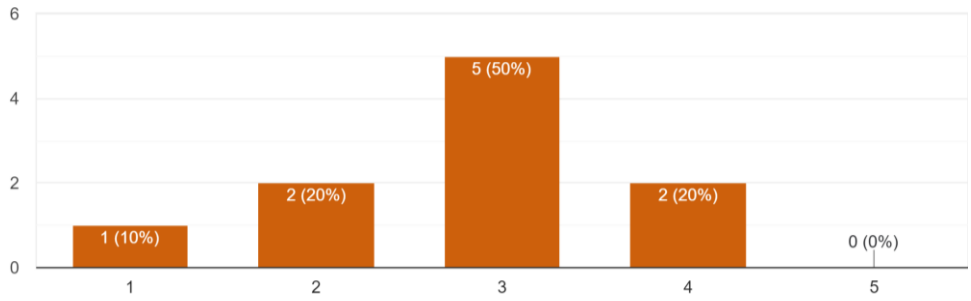



*Figure 33 - Data obtained for question 2 Complexity*

The results of Question 2 indicate that 10% of the participants strongly disagreed and 20% disagreed that Lazy Builder was unnecessarily complex. On the other hand, 50% of the participants felt neutral about the system’s complexity. Additionally, 20% agreed that Lazy Builder was unnecessarily complex. Interestingly, none of the participants strongly agreed with this statement. These results suggest that there is a mixed perception among the participants regarding the complexity of Lazy Builder. While a portion of the participants found it to be complex, the majority had a neutral view. This indicates that there might be room for improvement in simplifying the user experience and reducing complexity.

### Question 3 - Ease of use

“I thought Lazy Builder was easy to use”

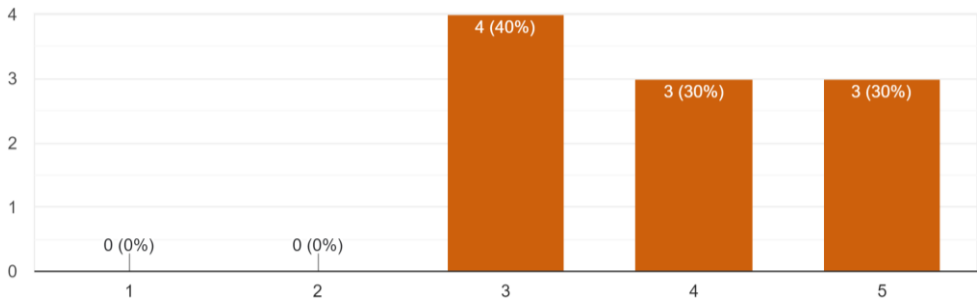



*Figure 34 - Data obtained for question 3 Ease of use*

The results of Question 3 show that none of the participants strongly disagreed or disagreed that Lazy Builder was easy to use. 40% of the participants felt neutral about the ease of use, while 30% agreed and 30% strongly agreed that Lazy Builder was easy to use. These results indicate a positive perception of the system's usability, with a significant portion of participants finding it easy or very easy to use. However, a notable proportion of participants also had a neutral stance. This suggests that while Lazy Builder is generally considered user-friendly, there may still be areas for improvement to enhance the overall usability.

### Question 4 - Support necessity

“I think that I would need the support of a technical person to be able to use Lazy Builder.”

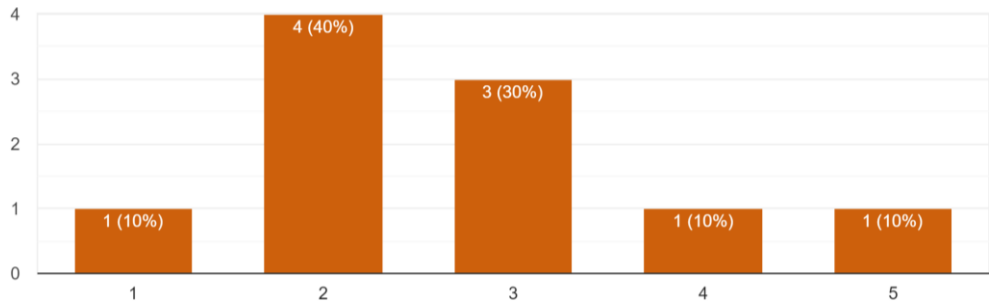



*Figure 35 - Data obtained for question 4 Support necessity*

The results of SUS Question 4 indicate that 10% of participants strongly disagreed, 40% disagreed, and 30% felt neutral about the need for technical support to use Lazy Builder. On the other hand, only 10% agreed and 10% strongly agreed that they would require technical assistance. These results suggest that a majority of participants felt that they did not necessarily need the support of a technical person to use Lazy Builder. However, a significant portion still expressed uncertainty or need for assistance. This highlights the importance of providing clear and accessible user documentation and better UI clarity in this system.

### Question 5 - Functionalities

“I found the various functions in Lazy Builder were well integrated.”

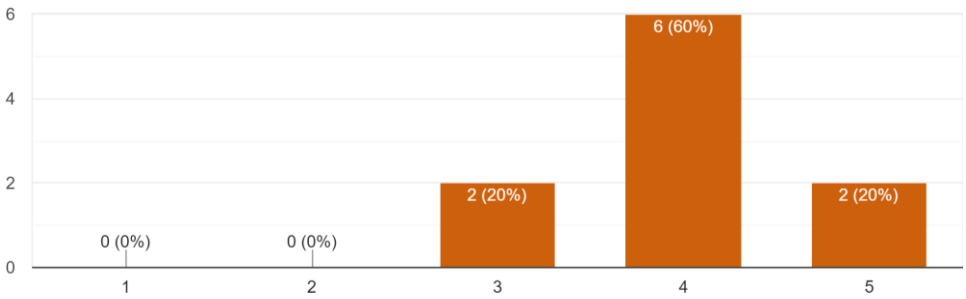



*Figure 36 - Data obtained for question 5 Functionalities*

The results of SUS Question 5 indicate that none of the participants strongly disagreed or disagreed that the various functions in Lazy Builder were well integrated. 20% of participants felt neutral, while 60% agreed that the functionalities were well integrated. Additionally, 20% strongly agreed with this statement. These results suggest that the participants found the functions in Lazy Builder to be effectively integrated, which is a positive outcome.

### Question 6 - Consistency

“I thought there was too much inconsistency in Lazy Builder.”

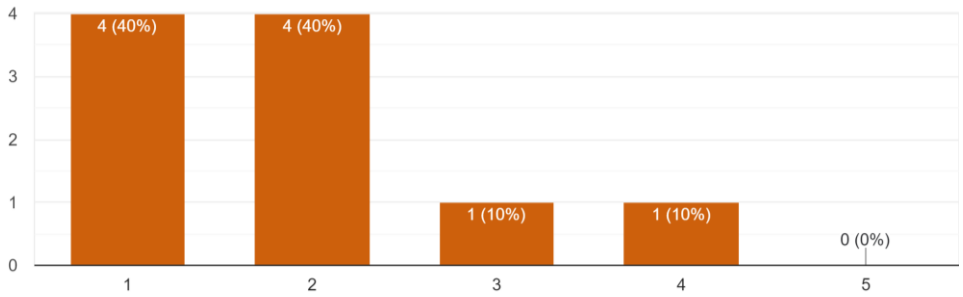



*Figure 37 - Data obtained for question 6 Consistency*

The results of Question 6 show that 40% of the participants both strongly disagreed and disagreed that there was too much inconsistency in Lazy Builder. Another 10% agreed, while 10% felt neutral on this statement. Notably, none of the participants strongly agreed that there was too much inconsistency. These results indicate that most participants did not perceive significant issues with inconsistency in the system.

### Question 7 - Learning Curve

“I would imagine that most people would learn to use Lazy Builder very quickly.”

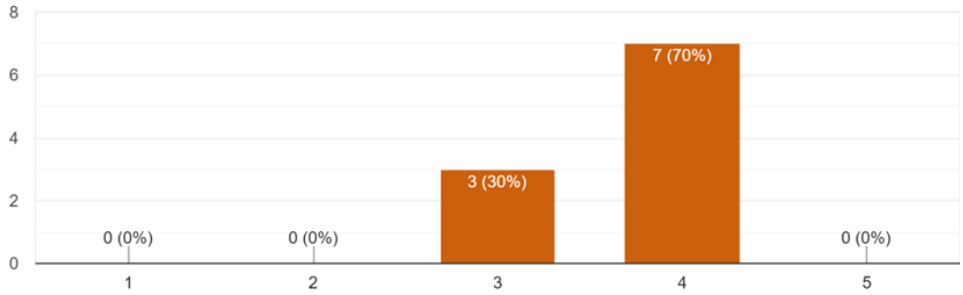



*Figure 38 - Data obtained for question 7 Learning Curve*

The results of Question 7 indicate that 70% of the participants agreed that most people would learn to use Lazy Builder very quickly. Meanwhile, 30% felt neutral on this statement, and none of the participants strongly disagreed or disagreed. These results suggest that most participants perceived the system as having a relatively low learning curve, implying that it was intuitive and easy to grasp for new users.

### Question 8 - Speed of Interactivity

“I found Lazy Builder very cumbersome to use.”

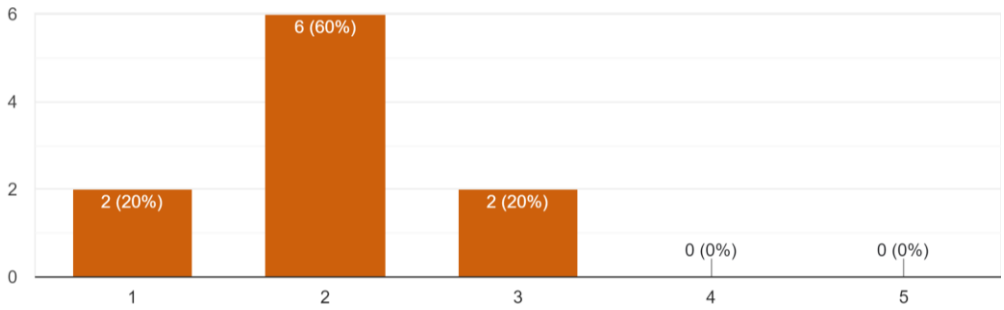



*Figure 39 - Data obtained for question 8 Speed of Interactivity*

Based on the results of SUS Question 8, 20% of the participants strongly disagreed that they found Lazy Builder cumbersome to use, and 60% disagreed. Finally, another 20% felt neutral. None of the participants agreed or strongly agreed. These results suggest that most participants did not perceive Lazy Builder as cumbersome, indicating that the system was generally considered user-friendly and not overly difficult to interact with.

### Question 9 - Confidence in use

“I felt very confident using Lazy Builder.”

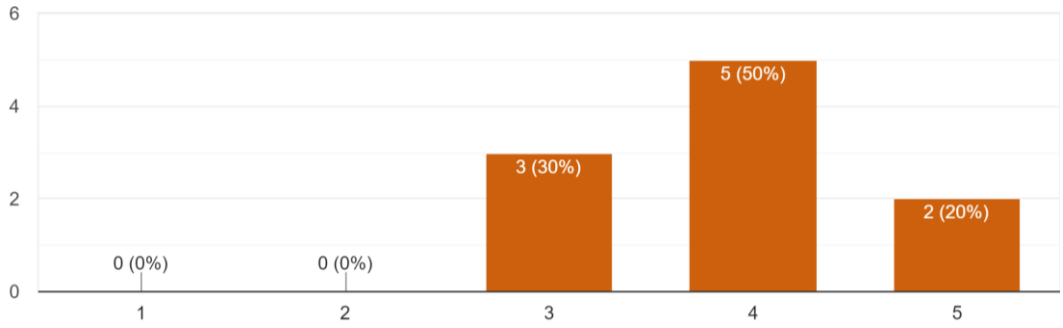



*Figure 40 - Data obtained for question 9 Confidence in use*

Based on the results of Question 9, none of the participants strongly disagreed or disagreed that they felt confident using Lazy Builder. 30% of the participants felt neutral, while 50% agreed and 20% strongly agreed with the statement. These results point out that the majority of the participants had a positive perception of their confidence in using Lazy Builder, suggesting that the system inspired a sense of competence and technical assurance in its users.

### Question 10 - Learning entry barrier

“I needed to learn a lot of things before I could get going with Lazy Builder.”

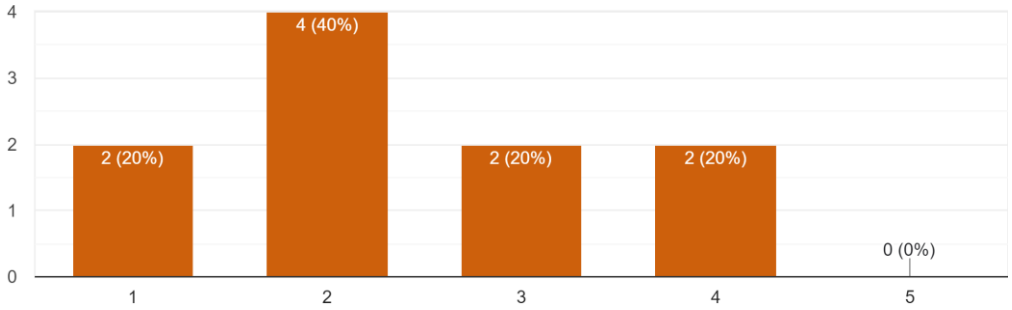



*Figure 41 - Data obtained for question 10 Learning entry barrier*

Based on the results of SUS Question 10, 20% of the participants strongly disagreed, 40% disagreed, 20% felt neutral, and 20% agreed that they needed to learn a lot of things before they could get going with Lazy Builder. These results suggest that a significant percentage of the participants perceived a moderate to high learning entry barrier with the system. It indicates that some participants may have found the initial learning curve or the amount of knowledge required to start using Lazy Builder to be relatively high. This feedback highlights the importance of providing clear and accessible learning resources and tutorials to help users overcome this entry barrier and quickly get started with the PCG system.

### Final Question - Overall Interface Experience

Considering all the contact you've made with tool and the key points addressed earlier. How can you describe the overall User-Experience of this node-based tool?

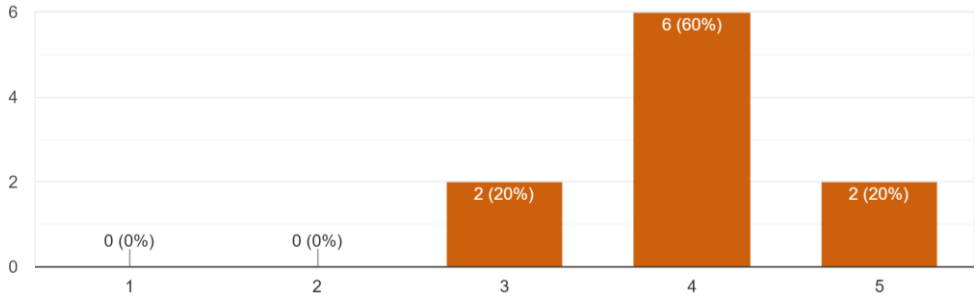



*Figure 42 - Data obtained for the final question Overall Interface Experience*

Based on the results of the final question on the overall interface experience, 20% of the participants rated the user experience as considerably good (5), indicating a positive evaluation. The majority, 60%, rated it as good (4), further highlighting a generally favourable user experience. No participants rated it as considerably bad (1) or bad (2). However, 20% of the participants provided a neutral (3) rating, suggesting that there is room for improvement and fine-tuning to enhance the overall user experience of the node-based tool.

### 6.2.2 Written Feedback

At the end of the questionnaire, participants were given the opportunity to provide a brief justification for their overall scores and share suggestions for improvement. Some participants provided detailed feedback on current features and proposed new ones to enhance the tool's usability. Answers from the open feedback question have been categorized into: justifications, new feature suggestions, and bug reporting.

**Justifications/Comments**

- "The tool needs documentation. Each function could have a small description of what it does."
- "I find the tool to be on the right track, but there are quite a few bugs that affect the experience."
- "I really liked the visuals of the graph and the speed of execution, which provided a smooth experience."

**New Feature Suggestions**

- "If there was the possibility of doing Undo, that is, Ctrl+Z, it would help to improve the experience."
- "Maybe turning the expressions used into something node-based like the input of Values in the respective parameters."
- "There should be a way to open the 'Create Node' window in the interface via a button."
- "An easier way to convert parameters to expressions/fixed values and delete parameters."
- "A way to change the seed on several random nodes without manually changing each one."
- "Try to have more than one colour scheme."
- "Some pre-made examples to use or start with."

**Bug Reporting**

Various bugs were reported regarding the Context window, Node search window, Node Parameter folding, Node duplication, and Graph persistence.

### 6.2.3 Observational Notes

The feedback in this section, gathered from participants' oral comments and non-verbal actions during the exercises, complements the written feedback provided earlier. These observations capture additional insights that were not explicitly mentioned. This feedback was collected from users who participated in presential sessions of this study. Based on participant feedback and observations during the study sessions, the following patterns were identified:

- 5 participants did not use the search window in the Add Node window, opting to search for nodes within the categories manually.
- 4 participants, despite being instructed otherwise, attempted to open the "Add Node" window using the right mouse click.
- 3 users leaned closer to the screen when entering an expression parameter, suggesting difficulties in reading the text.
- 3 users were confused about an error still being displayed in the console after its resolution.
- 3 users reported difficulties distinguishing between the node type and node name fields.
- 2 users attempted to use the standard right mouse click "Copy" and "Paste" functions, which lead to confusion.
- 2 users experienced confusion when connecting multiple edges to a single port, causing the identification numbers in each port to become cluttered and unreadable.
- 2 users noted that the Add Node window contents did not fill the entire window size, leading to confusion.
- 1 user pointed out that the visual order of Node Name and Node Type in the Context window differed from that displayed in the Node block itself.
- 1 user suggested the inclusion of a node for executing custom scripts.

- 1 user was confused when not all search results were displayed in the “Add Node” window since the previous search term remained in the search bar.

### 6.2.4 Discussion

Study II aimed to evaluate the usability of the PCG system through a combination of quantitative questions, written feedback, and observational notes. A total of ten participants took part in the study, providing valuable insights into their experiences with the tool.

The quantitative questionnaire followed the SUS structure and assessed various aspects of the perceived usability of the PCG system. Overall, the results showed positive feedback with participants generally finding the tool useful and easy to use.

Participants provided additional justifications and comments regarding their overall score in the written feedback section. Some participants highlighted the need for improved documentation, while others reported specific bugs or suggested enhancements for some existing features. New feature suggestions included undoing actions, improving the node search function, and providing pre-made examples. Participants also raised concerns about the learning curve associated with node-based tools.

Observational notes were gathered to support the written feedback and provide additional insights into user behaviour and areas of confusion. Some participants struggled with certain interactions, such as opening the Add Node window, distinguishing between node types and names, and reading expression parameters.

## 6.3 Game Jam Usage

This section presents a usage case that emerged during the course of the research, providing a more complete usage scenario with tangible results. It is important to note that this study was not initially planned as a structured case study but rather as an opportunity to explore more of the practical application of the implemented PCG tool.

### 6.3.1 Game Description

A team of game developers, consisting of one programmer and one designer, participated in the Instituto Superior de Engenharia do Porto (ISEP) Level Up Game Jam of 2023. During this event, participants come together to create games within a limited timeframe (for this case 48h), with the added challenge of incorporating a specific theme into their game design.

The Game Jam's theme, "Death is a new beginning," inspired the team to create a 2.5D game entitled "00:11" with strong reference to games such as Alto's Journey and Tiny Wings.

In "00:11," players take on the role of an explorer named Sunny, embarking on a challenging adventure in a mysterious world. The game features treacherous mountains and valleys with obstacles that must be traversed, all while being pursued by a wall of light. Sunny possesses the ability to harness the power of orbs, which can be to increase its movement speed. When the player is caught by it, triggering the game over state, the player discovers that the main game scene was part of a near-death experience in heaven, that offers the game's character a chance to reunite with their ancestors.


*Figure 43 - Gameplay of the "00:11" Game Jam title,Main Game scene (Top), Game Over scene (Bottom).*

### 6.3.2 Game Implementation

From a technical and development standpoint, the team employed Sceelix as a hybrid procedural approach to generate each mountain segment of the game level. The Surfaces data type was utilized to generate Unity terrains, with each segment being nearly one-dimensional, having a width of 1 pixel.

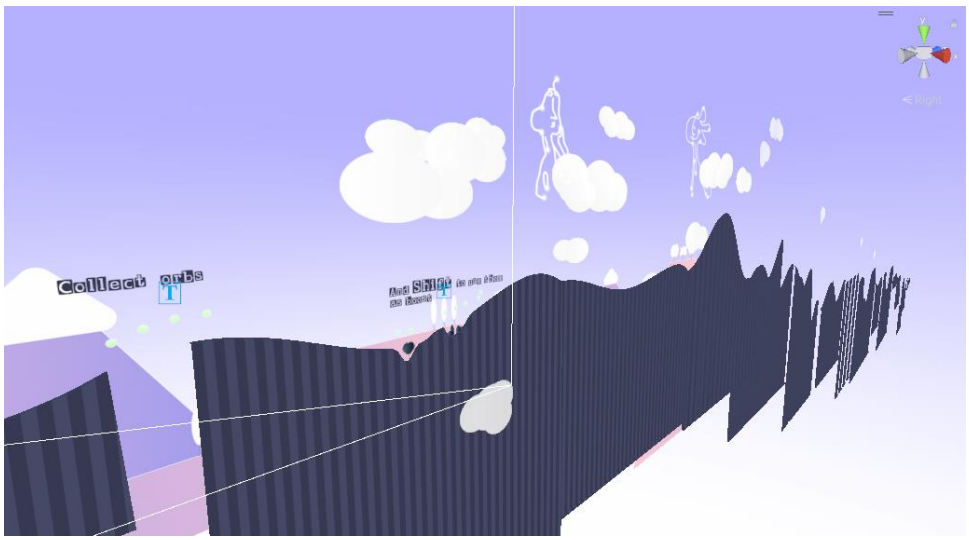


*Figure 44 - Mountains created for the “00:11” Game Jam title.*

Sceelix parameters such as scale, roughness, frequency, and seed were adjusted to produce different results for various mountain sections, influencing their level of difficulty. Although the primary focus was not on generating the entire map, the team created individual sections using this approach, allowing for multiple iterations tested via gameplay.

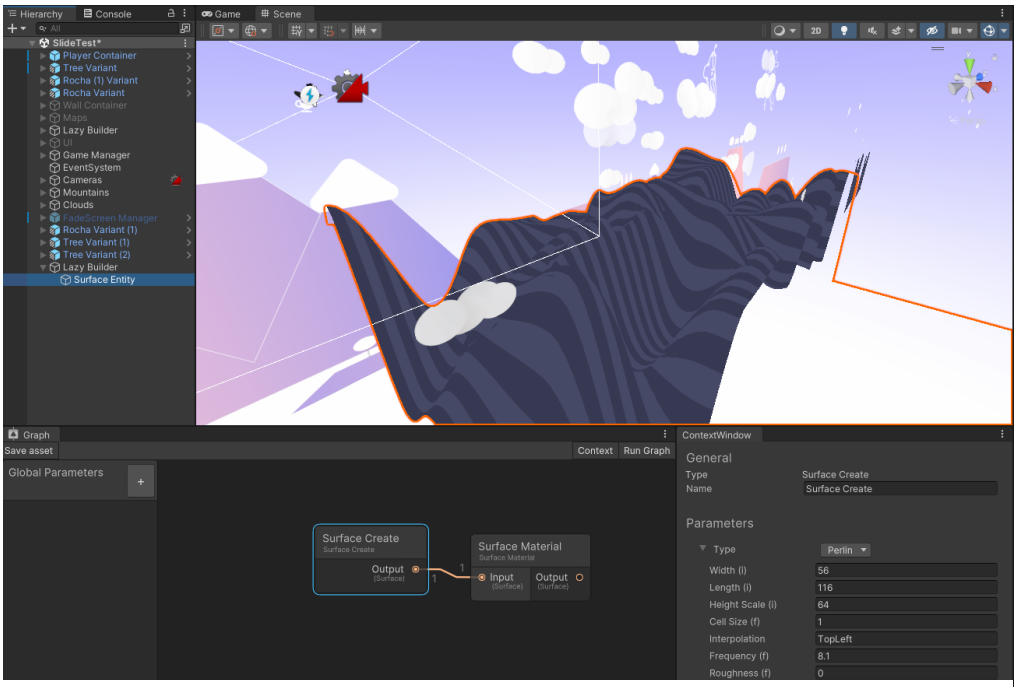



*Figure 45 - Construction of Perlin noise based terrains for”00:11”.*

### 6.3.3 Discussion

The team provided oral feedback on the overall experience of using the native PCG tool. While acknowledging the presence of a few usability bugs that hindered the tool's full utilization of the original features and the robustness of Sceelix, it highlighted the need for various additional features when dealing with surfaces. Consequently, some workarounds and manual operations were employed to achieve the desired final map. However, it is important to note that despite these challenges, the PCG tool enabled the team to develop a functional game featuring smooth and custom sliding mountains, a task that would have been considerably more difficult with a PCG system built entirely from scratch.

# 7 Conclusions

This research aimed to address the limitations faced by current PCG systems in game development, specifically regarding integration, interactivity, and ease of use. A thorough exploration of state of the art in Level Design, PCG, and Node-based Interfaces provided the foundation for understanding the challenges and existing solutions.

The first study involved interviews with PCG experts who gathered general feedback and in-depth advice on the preferable UI approach for this node-based interface. The insights obtained from this study confirmed the relevance of the identified problem and the presented native solution. The experts' preferences gravitated towards the context-free user interface approach, where each Node's information is only displayed upon its selection. Challenges associated with managing large-scale graphs were also acknowledged, emphasizing the importance of addressing its visual organization to reduce visual clutter and maintain readability.

A comprehensive analysis and design process covered various aspects, including the structure of Unity Engine, and Sceelix library, which led to a steered implementation of the identified solution requirements. By taking advantage of Unity Engine's functionalities and using the Sceelix library as the Procedural Generation backbone, the implementation fulfilled both functional and non-functional requirements, ensuring the system's usability, reliability, performance, and supportability.

Study II focused on evaluating the usability of the native PCG system through a combination of quantitative questions, written feedback, and observational notes. The results showed positive feedback overall, with participants finding the tool useful and easy to use. However, the written feedback and observational notes highlighted areas for improvement, such as documentation, bug fixes, and feature enhancements.

As for the usage case of the Game Jam, despite some usability issues and the need for additional features when dealing with terrains, the PCG tool allowed the team to develop a functional game through an iterative hybrid procedural approach. The tool's efficiency surpassed building a custom PCG algorithm from scratch, enabling the level of concretization within the jam's timeframe.

The obtained results indicate that the developed native PCG methodology presents a substantial benefit to the level design process. The positive feedback from participants in Study II indicates that the system was generally useful and easy to use, fulfilling the objective of providing a fluid user experience for game developers. The iterative and user-centred methodology employed throughout the research contributed to the system's effective development and refinement.

Answering to the central research question "Does developing a PCG system natively in a Game Engine provide significant advantages for the Level Design process?". Within the limited amount of time to develop a functional prototype and to collect feedback from both specialists and game developers, results indicate that it provides a seamless experience with minimal friction between the user and the playable level.

In response to the question "What are the key interactivity and user experience metrics that hold the most significance in a PCG?", a variety of metrics were identified based on participant feedback. While a definitive answer could not be determined, certain metrics were consistently emphasized by the participants, including visual clearance to manage graph visual clutter, contextual documentation to ease the learning curve of the PCG system, and precise error reporting for easy identification and understanding of errors.

Answering the last question "What are the potential approaches for designing the interface of a node-based PCG system?". Based on existing PCG solutions and other node-based systems the two main UI approaches are the full graph and the abstracted approach. But due to the limited time frame of design and implementation of the prototype, other more experimental UI approaches were not explored.

The implementation and research of this project faced several limitations due to the complex nature of PCG systems and the project's duration. One limitation pertained to the robustness of the functional prototype, which did not encompass all data types and operations and required further testing.

Two significant features of PCG, namely graph data encapsulation (allowing for subgraphs and reusing dense graph operations) and runtime generation, were not integrated into the prototype. This necessitated restructuring to create a separation between the graph structure and its visual editor component.

Moreover, the development of this PCG system could not be applied to a more specific usage case, such as dungeons, islands, or urban environments, as originally planned. Collaborating with the "Edscape" project to design educational escape rooms was intended, but due to the extensive time dedicated to the PCG integration, a more in-depth usage case could not be explored.

In conclusion, this research contributed to addressing the current limitations of PCG systems in the game development pipeline by proposing a successful solution.

# 8 Future Work

This research lays the groundwork for further exploration and development in the field of PCG systems integrated into game engines. However, it is important to acknowledge that several challenges still need to be overcome in fully implementing a complex and extensive PCG system, to fully answer this research problem.

The feedback received from participants in Study II has provided valuable directions for future improvements, such as enhancing documentation, addressing bugs, and implementing suggested features. The identified learning curve associated with node-based tools should also be addressed through improved onboarding and user support.

The development of the PCG system is still ongoing and its current focus is on incorporating the suggested improvements and fixing the identified issues and expanding the system's capabilities. Its codebase is open-source (on GitHub) to help future contributors to be part of this project and the game development community has picked some interest in the project.

The subsequent research work on this project should focus on conducting more extensive user testing with a broader set of participants. Through continuous iteration and refinement of this native system, its full potential can be materialized, maximizing the benefits of PCG and elevating the overall game development process to new heights.

# 9 References

Adrian, D. F. H., & Ana Luisa, S. G. C. (2013). An approach to level design using procedural content generation and difficulty curves. *IEEE Conference on Computatonal Intelligence and Games, CIG*. https://doi.org/10.1109/CIG.2013.6633640

Ates, K., Kukluk, J., Holder, L., Cook, D., & Zhang, K. (2006). Graph grammar induction on structural data for visual programming. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, 232–239. https://doi.org/10.1109/ICTAI.2006.61

Brydon-Miller, M., Greenwood, D., & Maguire, P. (2003). Why action research? *Action Research*, *1*(1), 9–28.

Coghlan, D., & Bannick, T. (2015). Doing action research in your own organization. *Action Learning: Research and Practice*, *12*(2), 237–241. https://doi.org/10.1080/14767333.2015.1049453

Galuzin, A. (2011). *Ultimate level design guide by Alex*. https://www.goodreads.com/book/show/20944485-ultimate-level-design-guide

Hahn, E., Bose, P., & Whitehead, A. (2006). Persistent realtime building interior generation. *Proceedings - Sandbox Symposium 2006: ACM SIGGRAPH Video Game Symposium, Sandbox '06*, 179–186. https://doi.org/10.1145/1183316.1183342

Holub, O., Moiseienko Mykhailo, & Moiseienko Natalia. (2020). *Fluid Flow Modelling in Houdini*. 1–9.

Jong, S. de. (2008). *The Hows and Whys of Level Design*.

Kearns, K. (2017). *Semantics* (2nd ed., Vol. 1). https://books.google.pt/books?id=WJNKEAAAQBAJ

Löbner, S. (2013). Understanding Semantics. In *Understanding Semantics, Second edition: Vol. Vol 1* (2nd ed.). Taylor and Francis. https://doi.org/10.4324/9780203528334/UNDERSTANDING-SEMANTICS-SEBASTIAN-LOEBNER

Merrell, P., Schkufza, E., & Koltun, V. (2010). Computer-generated residential building layouts. *ACM Transactions on Graphics*, *29*(6). https://doi.org/10.1145/1866158.1866203

Naiman, J. P., Borkiewicz, K., & Christensen, A. J. (2017). Houdini for Astrophysical Visualization. *Publications of the Astronomical Society of the Pacific*, *129*(975), 058008. https://doi.org/10.1088/1538-3873/AA51B3

Preidel, C., & Borrmann, A. (2016). Towards code compliance checking on the basis of a visual programming language. *Journal of Information Technology in Construction (ITcon)*, 1–20. http://www.itcon.org/2016/25

Shin, J., Siegwart, R., & Magnenat, S. (2014). *Visual Programming Language for Thymio II Robot*. 2–5. https://doi.org/10.3929/ethz-a-010144554

Short Tanya, & Adams Tarn. (2017). *Procedural Generation in Game Design - Google Livros*. https://books.google.pt/books? id=Rj4PEAAAQBAJ

Silva, P. A. B. da. (2015). *Improving Expressiveness, Integration and Manageability in Procedural Content Generation*. https://repositorio-aberto.up.pt/handle/10216/99064

Singh, R. P. (2014). Application of Graph Theory in Computer Science and Engineering. *International Journal of Computer Applications*, *104*(1), 1–4.

Tutenel, T., Bidarra, R., Smelik, R. M., & de Kraker, K. J. D. (2008). The role of semantics in games and simulations. *Computers in Entertainment (CIE)*, *6*(4). https://doi.org/10.1145/1461999.1462009

# 10 Appendices

## 10.1 Appendix 1

# User Experience Assessment

**Before beginning this questionnaire please try to complete the following tutorial exercises:**
https://docs.google.com/presentation/d/1rUGbXzxXKXIvSs6e_vvyMhGzZoylVPB5/

This questionnaire is part of a thesis project for the **Master in Multimedia at FEUP.**

The thesis, entitled "**Node-based Native Solution to Procedural Level Generation**", aims to provide a native engine solution for procedural level generation in the Unity Engine.

Your feedback will help evaluate the tool's clarity, interactivity, usability, and overall user experience. It is an opportunity to share your thoughts on the features, challenges, and suggestions for improvement.

Following GDPR guidelines, this anonymous questionnaire data will only be used for research purposes. If you agree in sharing the information prompted below please proceed with the questionnaire.

*Thank you for taking the time to provide your insights!*

---

* Indicates a mandatory question

# Age

*Please choose only one option*

- 0-25
- 25-35
- 35-45
- 45+

# Current academic degree

*Please choose only one option*

- Highschool
- Bachelor
- Major
- Doctorate

# Experience in Game Development

*Please choose only one option*

- Zero experience
- I've made some projects here & there
- +3 years as a hobbyist or any professional experience
-

Considerable professional experience

# Experience with Procedural Tools

Do you have any experience with Procedural Modelling Software (such as Houdini, Blender Geo Nodes, etc...) or have you implemented any custom Procedural Systems?

*Please choose only one option.*

- ○ Zero experience
- ○ Used one of these software a couple of times
- ○ Considerable experience in one of these software or implemented custom systems

---

## Rating System

Your honest and unbiased feedback is essential in assessing the usability/user experience of this tool. Please keep in mind the next responses should reflect your initial impressions, independent of the additional instructional materials provided.

Please rate the following statements regarding your experience using "Lazy Builder" on a scale from 1 to 5, as specified below:

**1 Strongly Disagree | 2 Disagree | 3 Neutral | 4 Agree | 5 Strongly Agree**

# Future Usage *

I think that I would like to use Lazy Builder frequently

*Please choose only one option*

1

2

3

4

Strongly Agree

# Complexity *

I found Lazy Builder unnecessarily complex

*Please choose only one option*

| | |
|---|---|
| | Strongly Disagree |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| | Strongly Agree |

# Ease of use *

I thought Lazy Builder was easy to use

*Please choose only one option*

Strongly Disagree

1

2

3

4

5

Strongly Agree

# Support necessity *

I think that I would need the support of a technical person to be able to use Lazy Builder

*Please choose only one option*

Strongly Disagree

1

2

3

4

5

Strongly Agree

# Functionalities *

I found the various functions in Lazy Builder were well integrated

*Please choose only one option*

Strongly Disagree

1

2

3

4

5

Strongly Agree

# Consistency *

I thought there was too much inconsistency in Lazy Builder

*Please choose only one option*

Strongly Disagree

1

2

3

4

5

Strongly Agree

# Learning Curve *

I would imagine that most people would learn to use "Lazy Builder" very quickly

*Please choose only one option*

Strongly Disagree

1

2

3

4

5

Strongly Agree

# Speed of Interactivity *

I found "Lazy Builder" very cumbersome to use

*Please choose only one option*

Strongly Disagree

1

2

3

4

5

Strongly Agree

# Confidence of use *

I felt very confident using Lazy Builder

*Please choose only one option*

Strongly Disagree

1

2

3

4

5

Strongly Agree

# Learning entry barrier *

I needed to learn a lot of things before I could get going with Lazy Builder

*Please choose only one option*

Strongly Disagree

1

2

3

4

5

Strongly Agree

# Final Questions

The last 2 final questions aim to better understand your overall opinion on the previous answers. Express your thoughts below!

## Overall Interface Experience *

Considering all the contact you've made with tool and the key points addressed earlier How can you describe the overall User-Experience of this node-based tool?

*Please choose only one option*

Considerably Bad

1

2

3

4

5

Considerably Good

# Problems, Improvements & New Features

Can you provide a brief justification behind the above overall score?

Suggestions that you can point out:

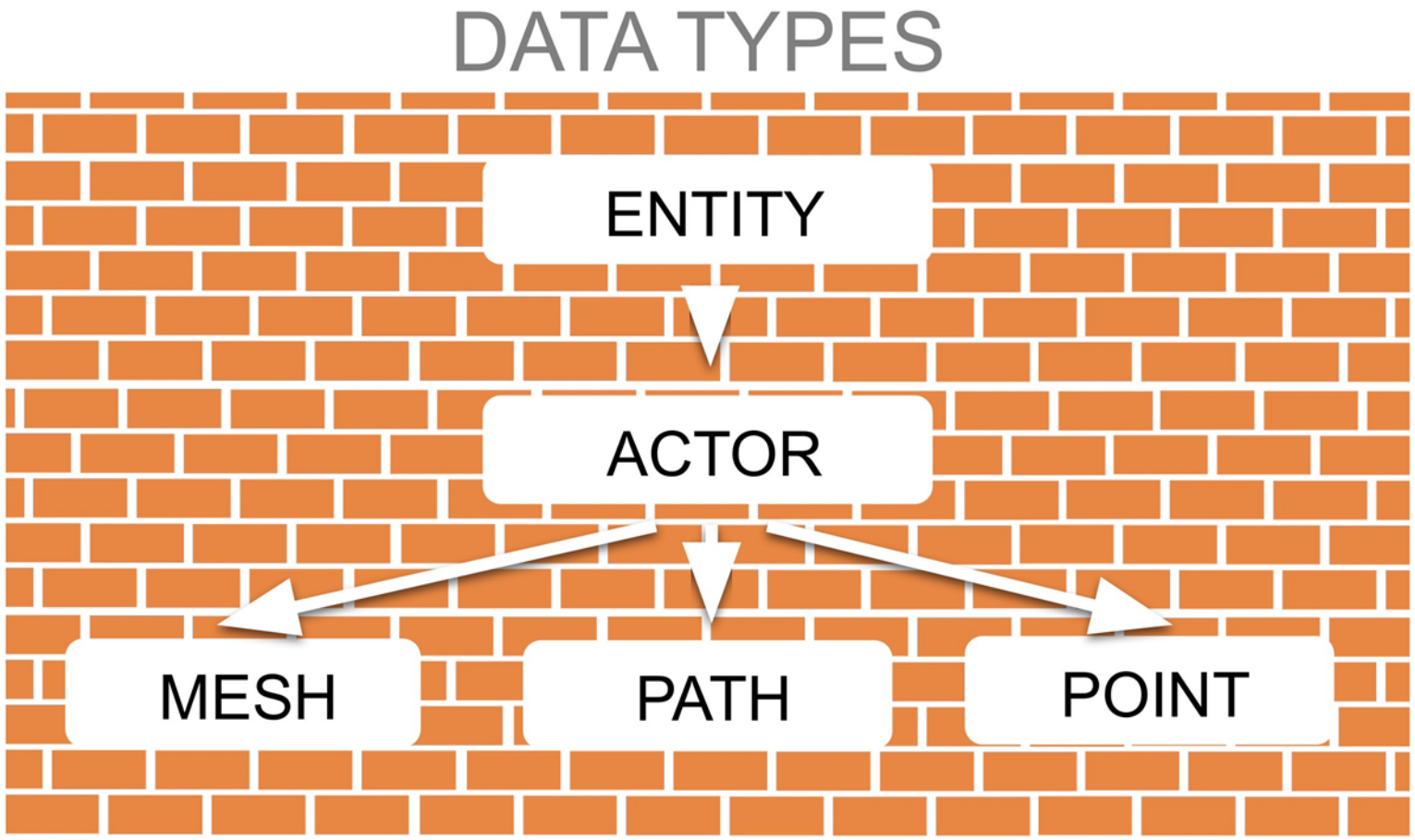
- Current implemented features do you think could be further tested and improved
- New features that could be added and would greatly benefit the overall tool usage

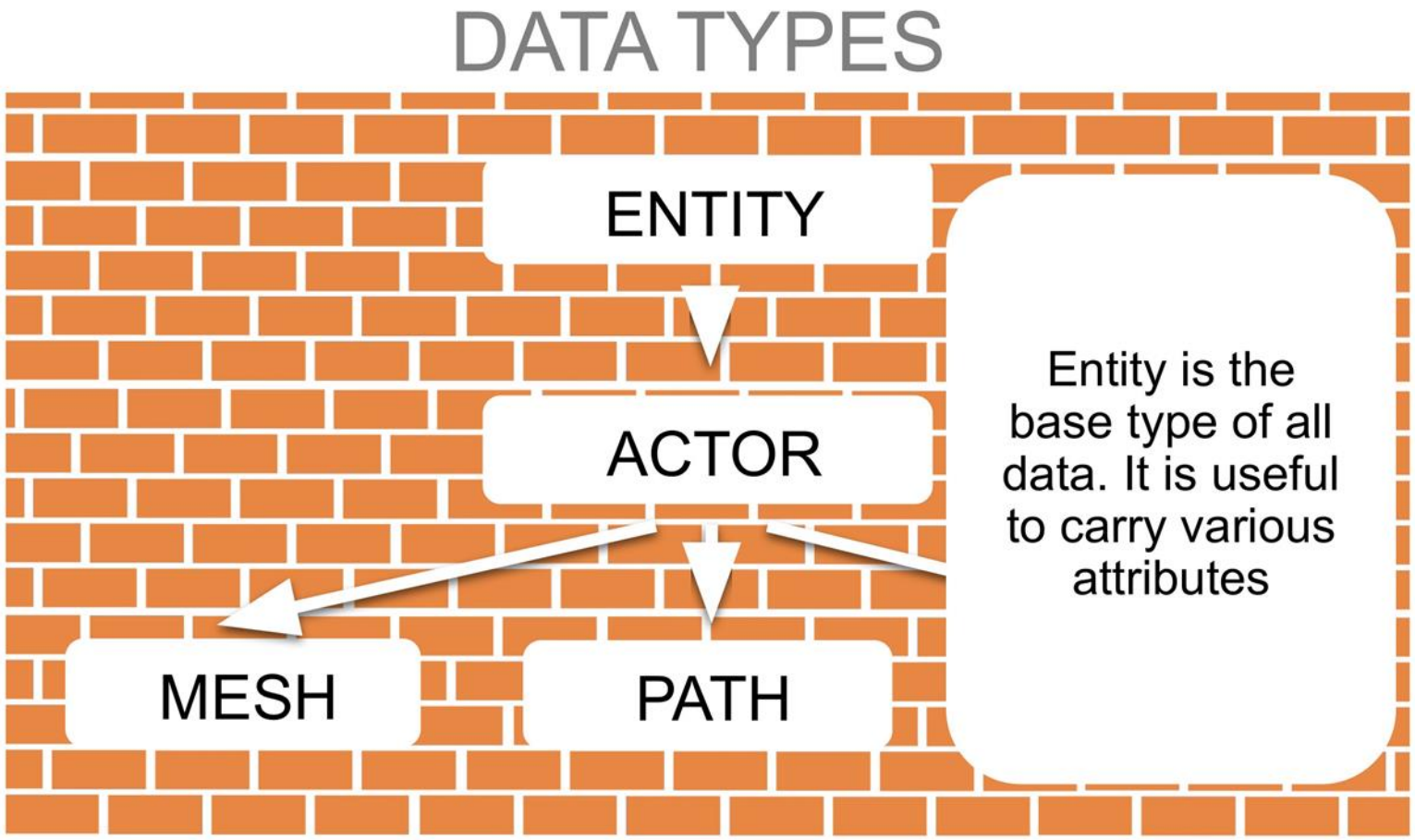
This content was created using Google Forms

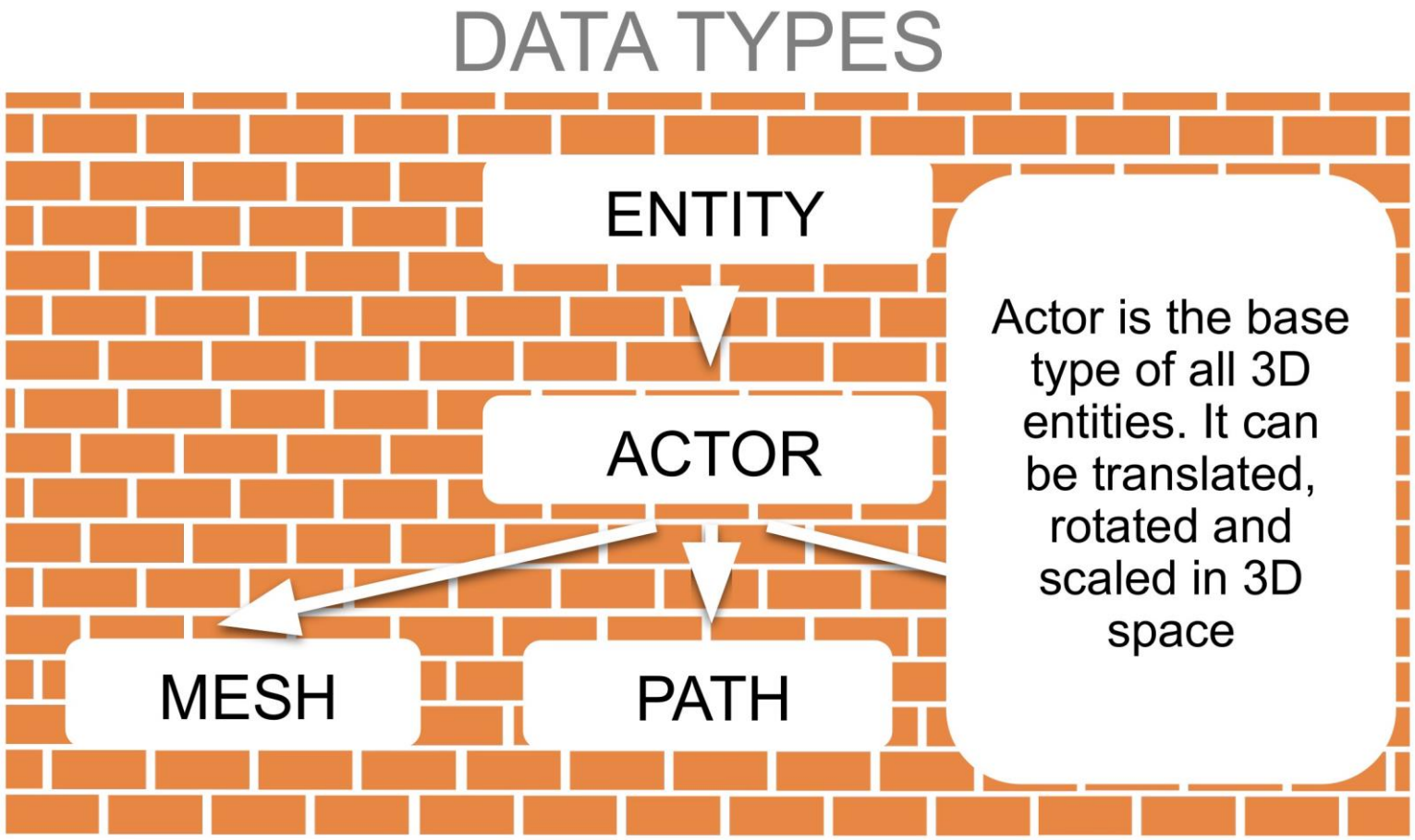
## 10.2 Study II Presentation

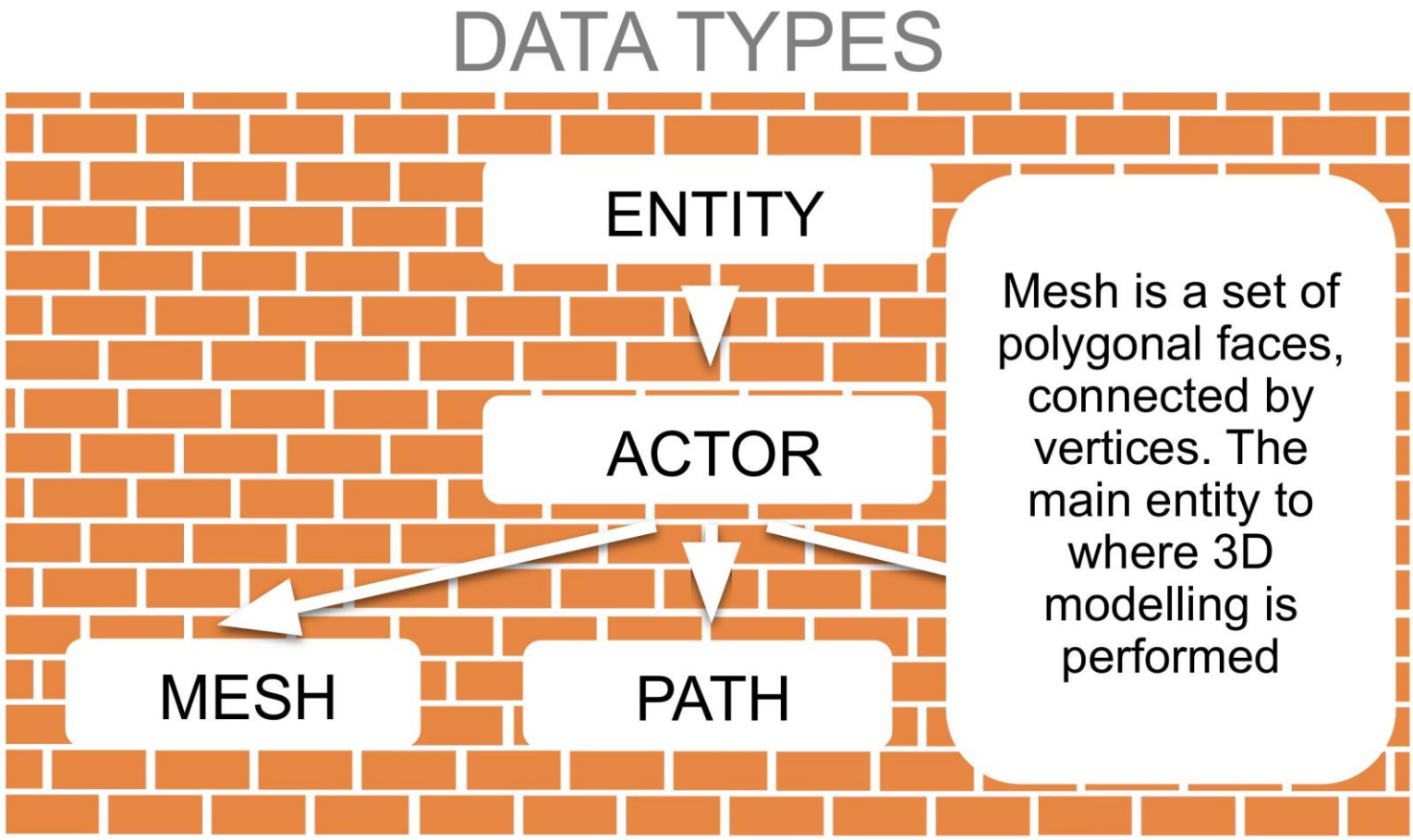
LAZY BUILDER

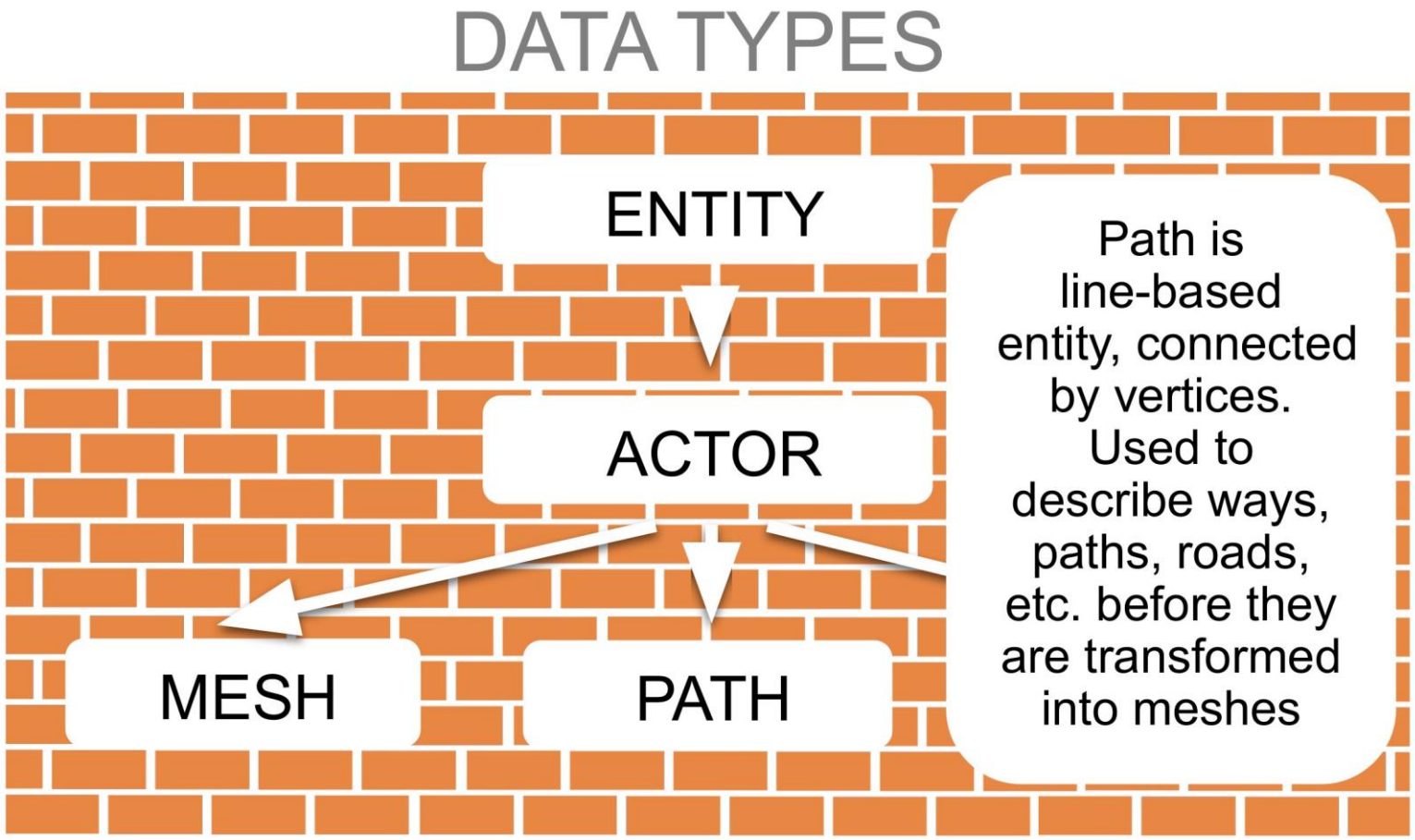
HOW TO USE IT

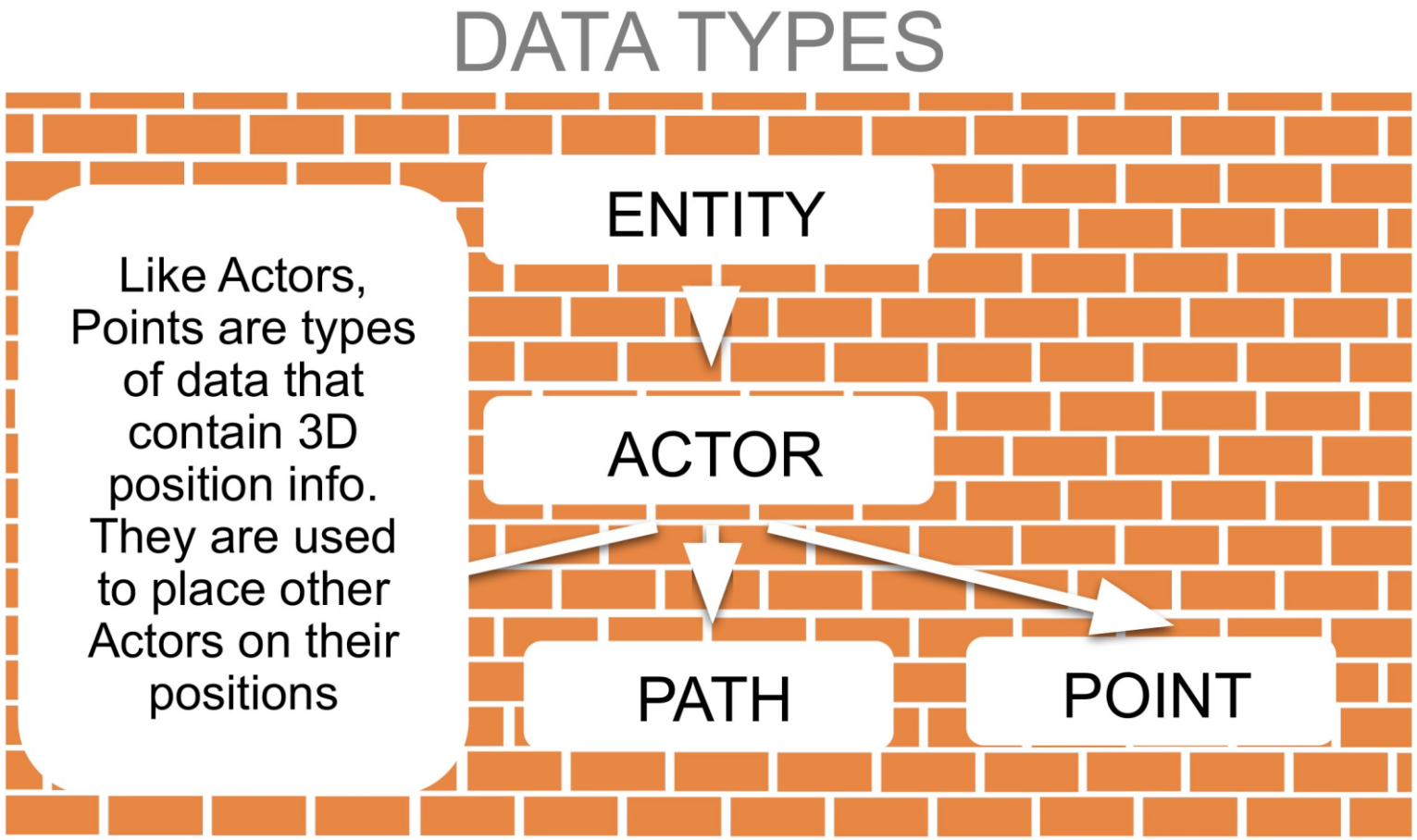

DATA TYPES
ENTITY
ACTOR
MESH
PATH
POINT


DATA TYPES
ENTITY
ACTOR
MESH
PATH
Entity is the base type of all data. It is useful to carry various attributes

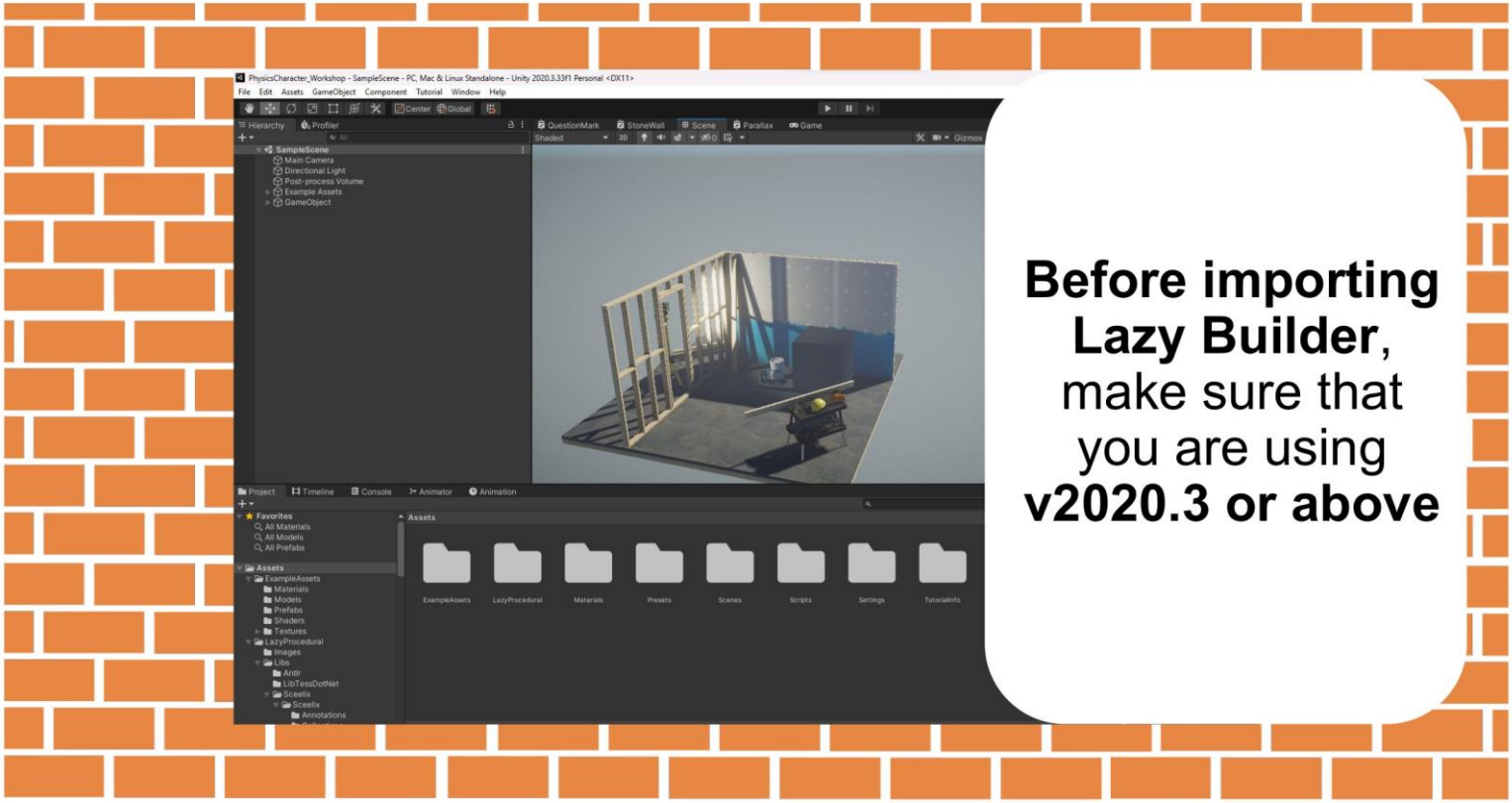

DATA TYPES
ENTITY
ACTOR
MESH
PATH
Actor is the base type of all 3D entities. It can be translated, rotated and scaled in 3D space


DATA TYPES
ENTITY
ACTOR
MESH
PATH
Mesh is a set of polygonal faces, connected by vertices. The main entity to where 3D modelling is performed

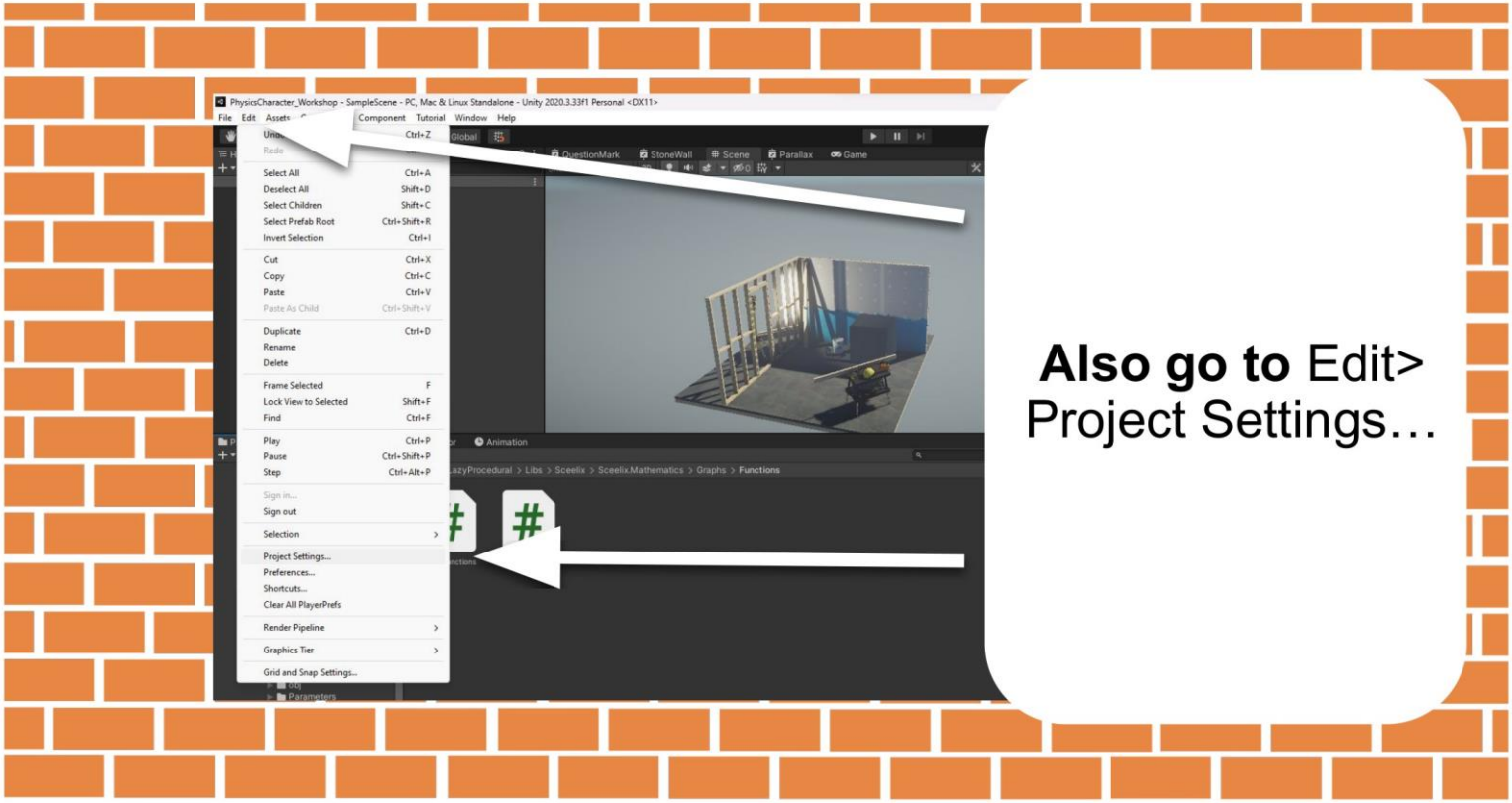

DATA TYPES
ENTITY
ACTOR
MESH
PATH
Path is line-based entity, connected by vertices. Used to describe ways, paths, roads, etc. before they are transformed into meshes


DATA TYPES
ENTITY
Like Actors, Points are types of data that contain 3D position info. They are used to place other Actors on their positions
ACTOR
PATH
POINT

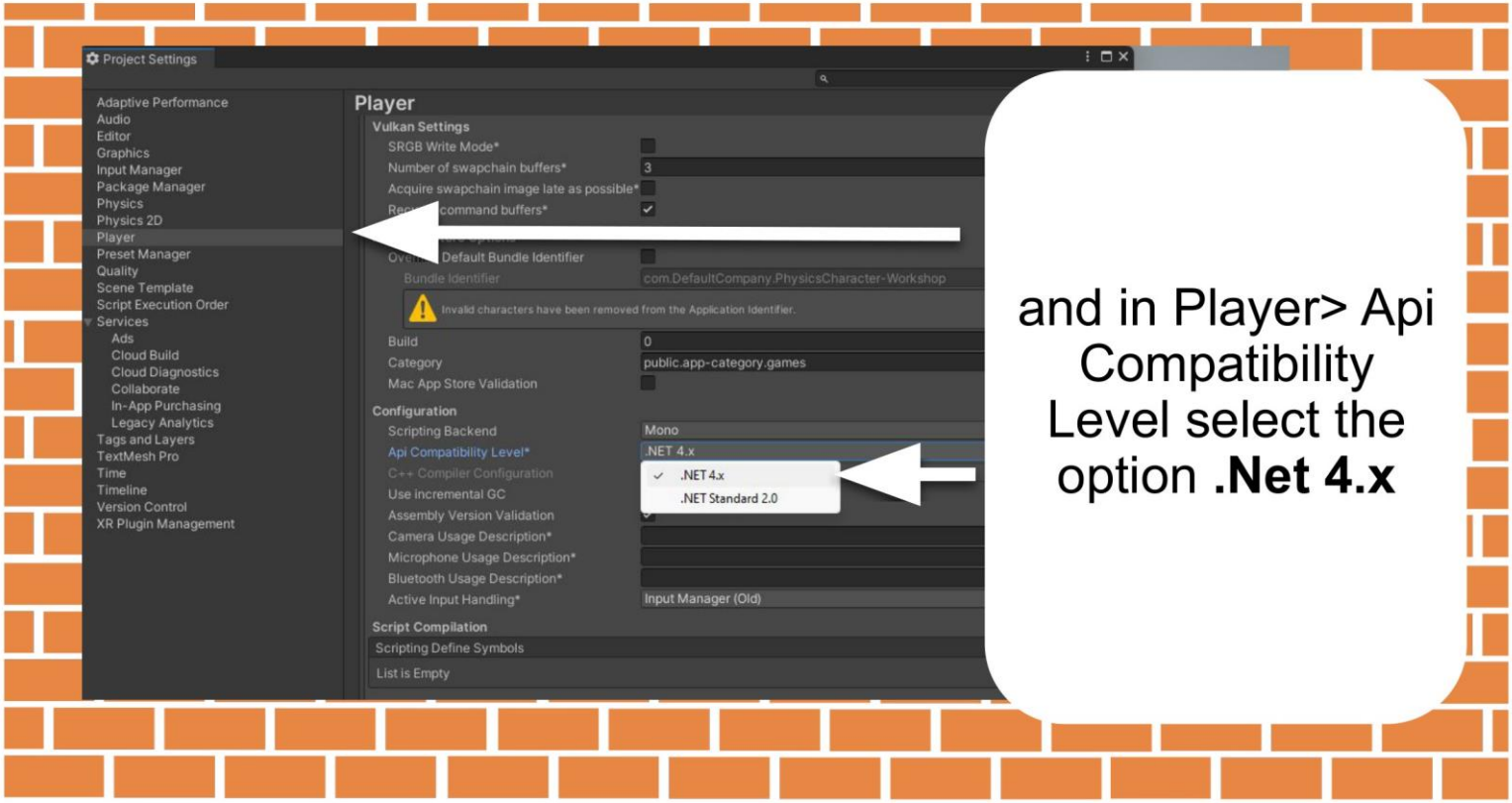
# BEFORE START

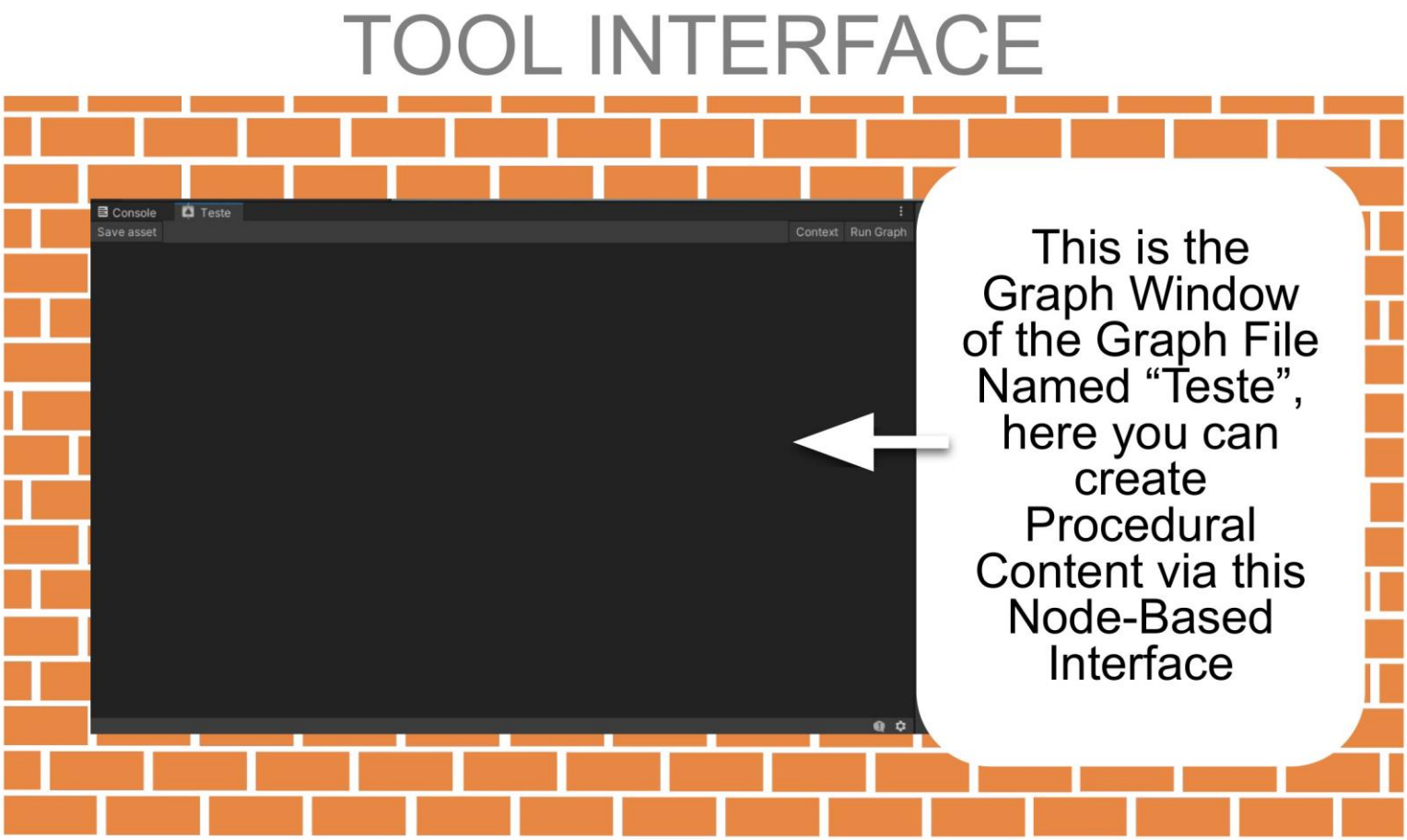


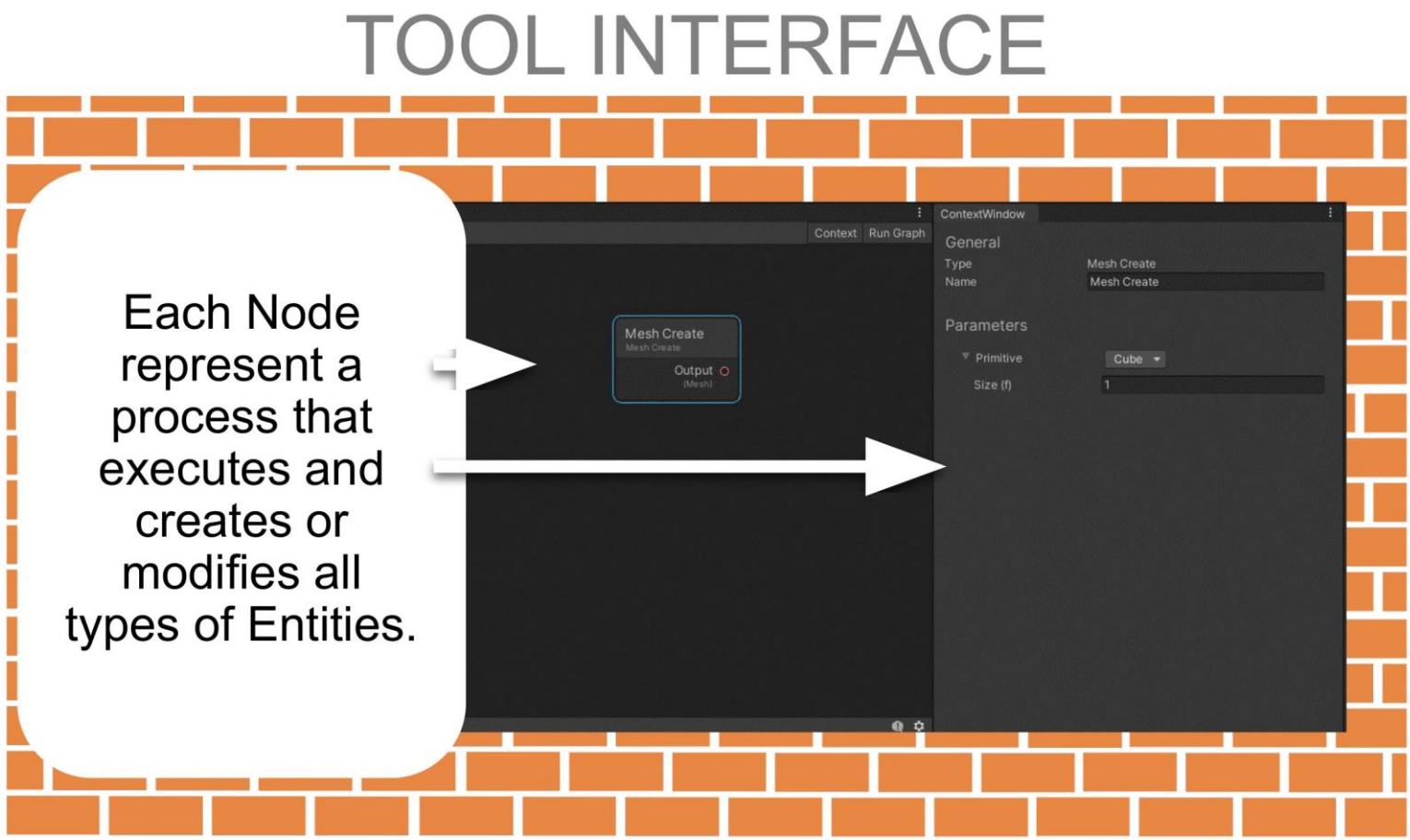
# BEFORE START

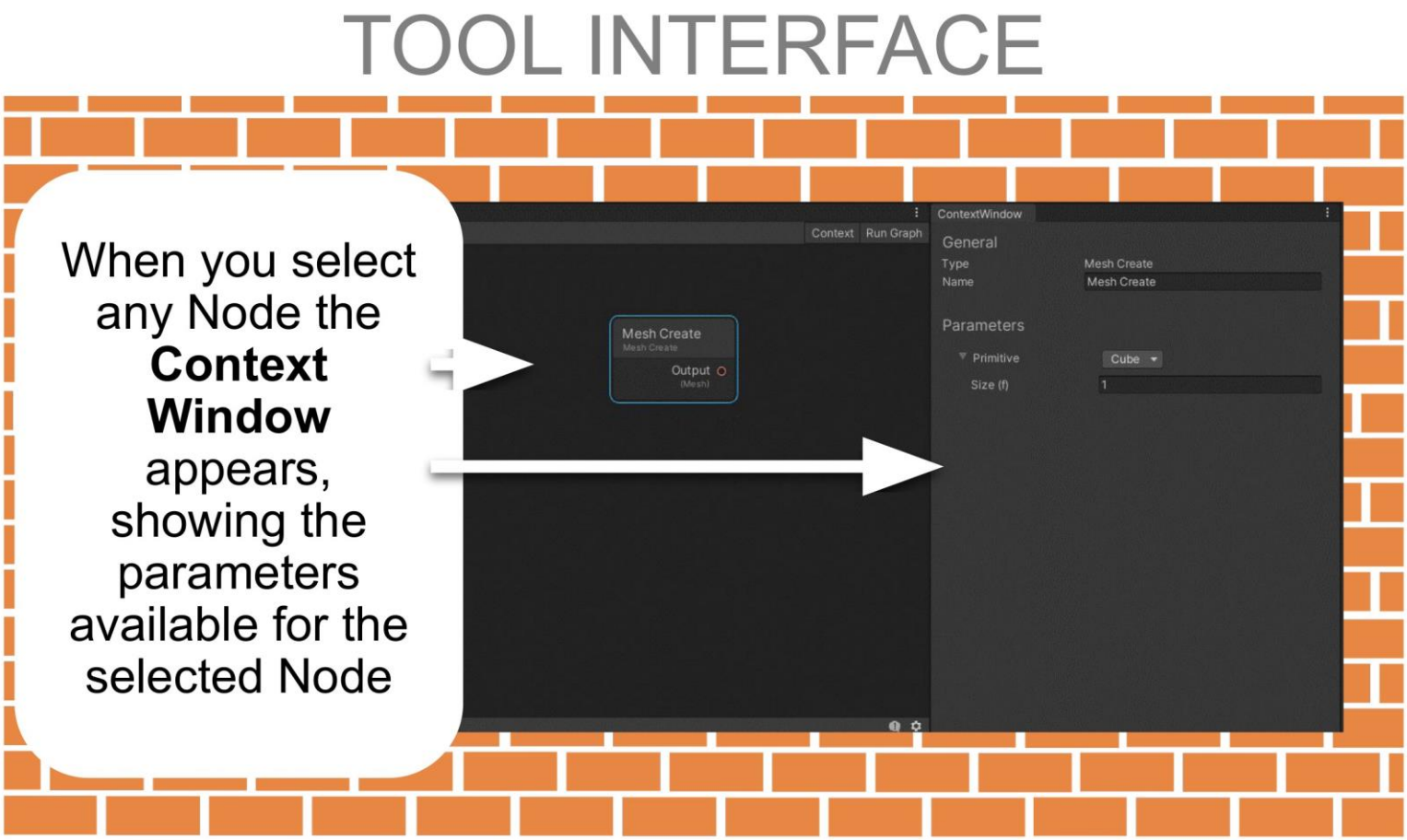


# BEFORE START

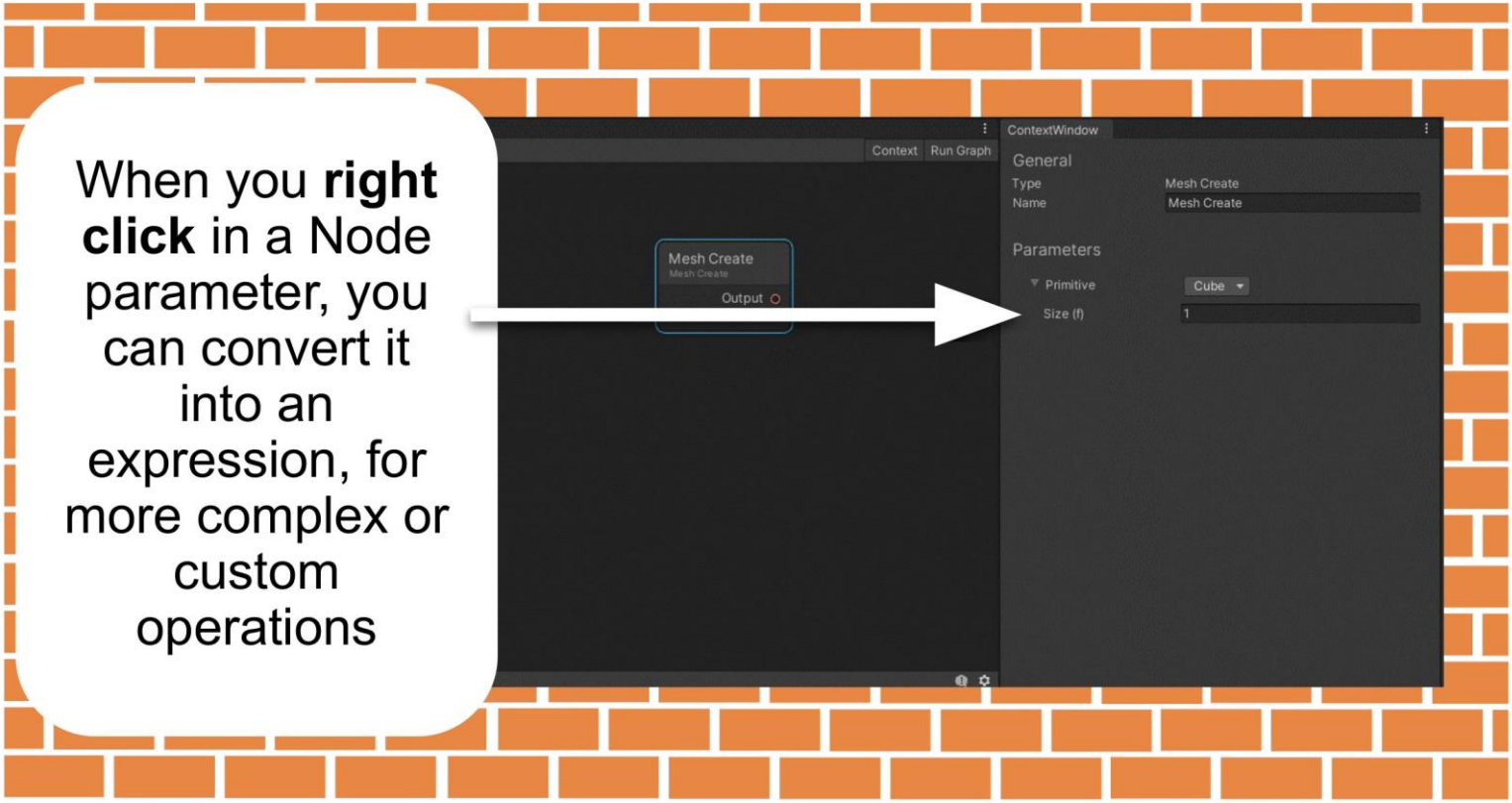



TOOL INTERFACE
Console
Teste
Save asset
Context
Run Graph
This is the Graph Window of the Graph File Named “Teste”, here you can create Procedural Content via this Node-Based Interface

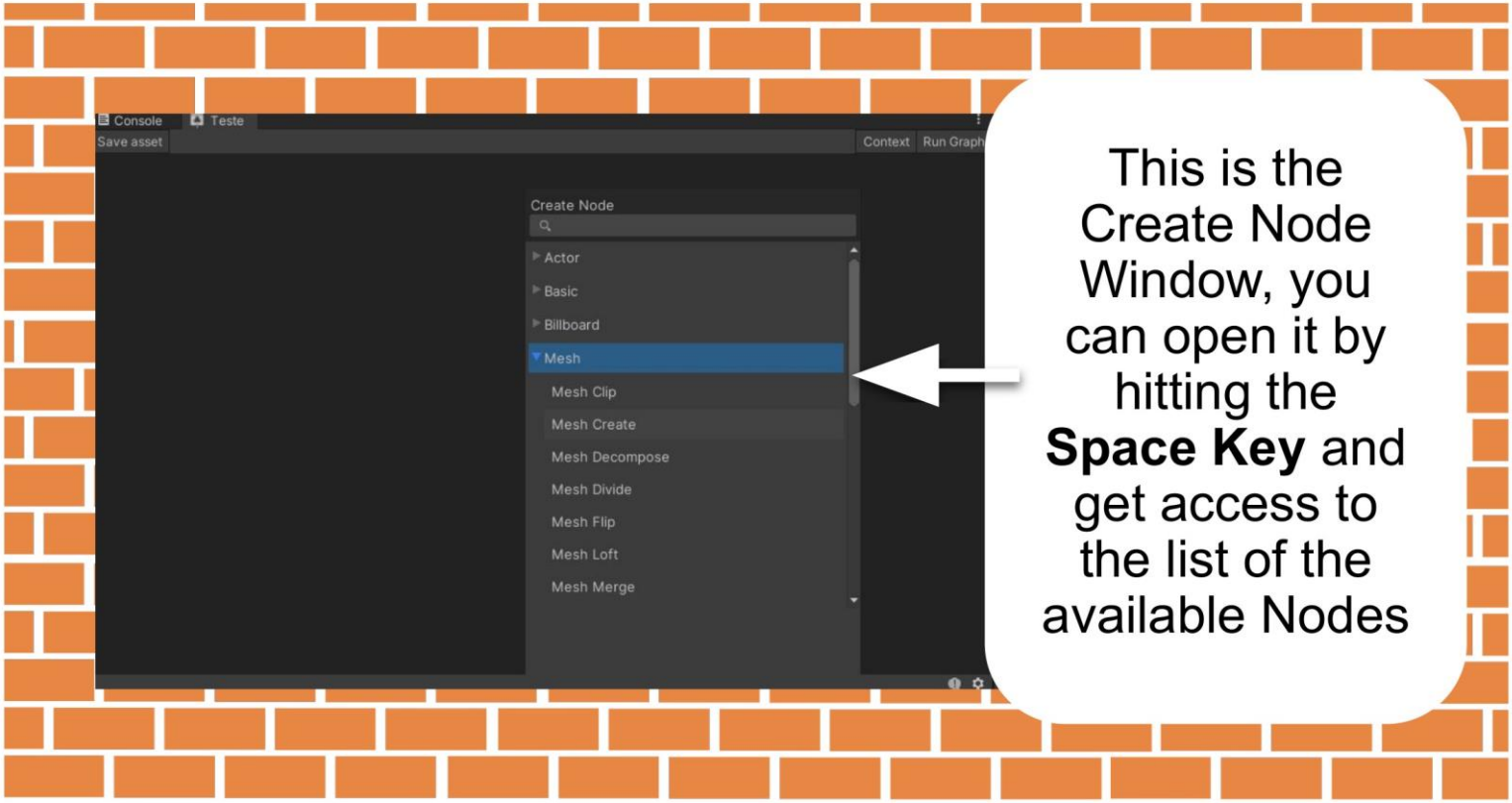

TOOL INTERFACE
Each Node represent a process that executes and creates or modifies all types of Entities.
Context
Run Graph
Mesh Create
Mesh Create
Output
(Mesh)
ContextWindow
General
Type
Mesh Create
Name
Mesh Create
Parameters
Primitive
Cube
Size (f)
1


TOOL INTERFACE
When you select any Node the **Context Window** appears, showing the parameters available for the selected Node
Context
Run Graph
Mesh Create
Mesh Create
Output
(Mesh)
ContextWindow
General
Type
Mesh Create
Name
Mesh Create
Parameters
Primitive
Cube
Size (f)
1

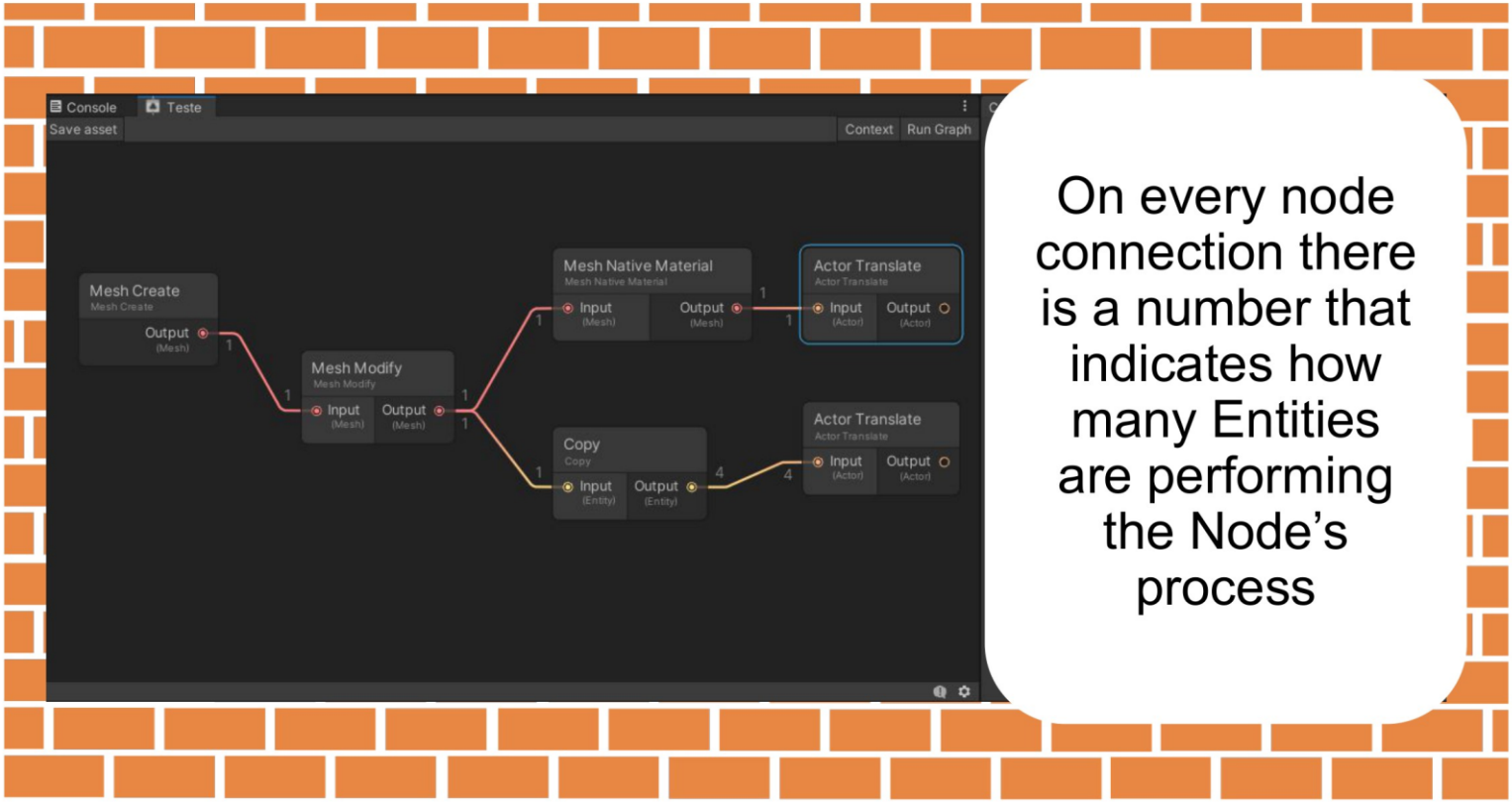
# TOOL INTERFACE

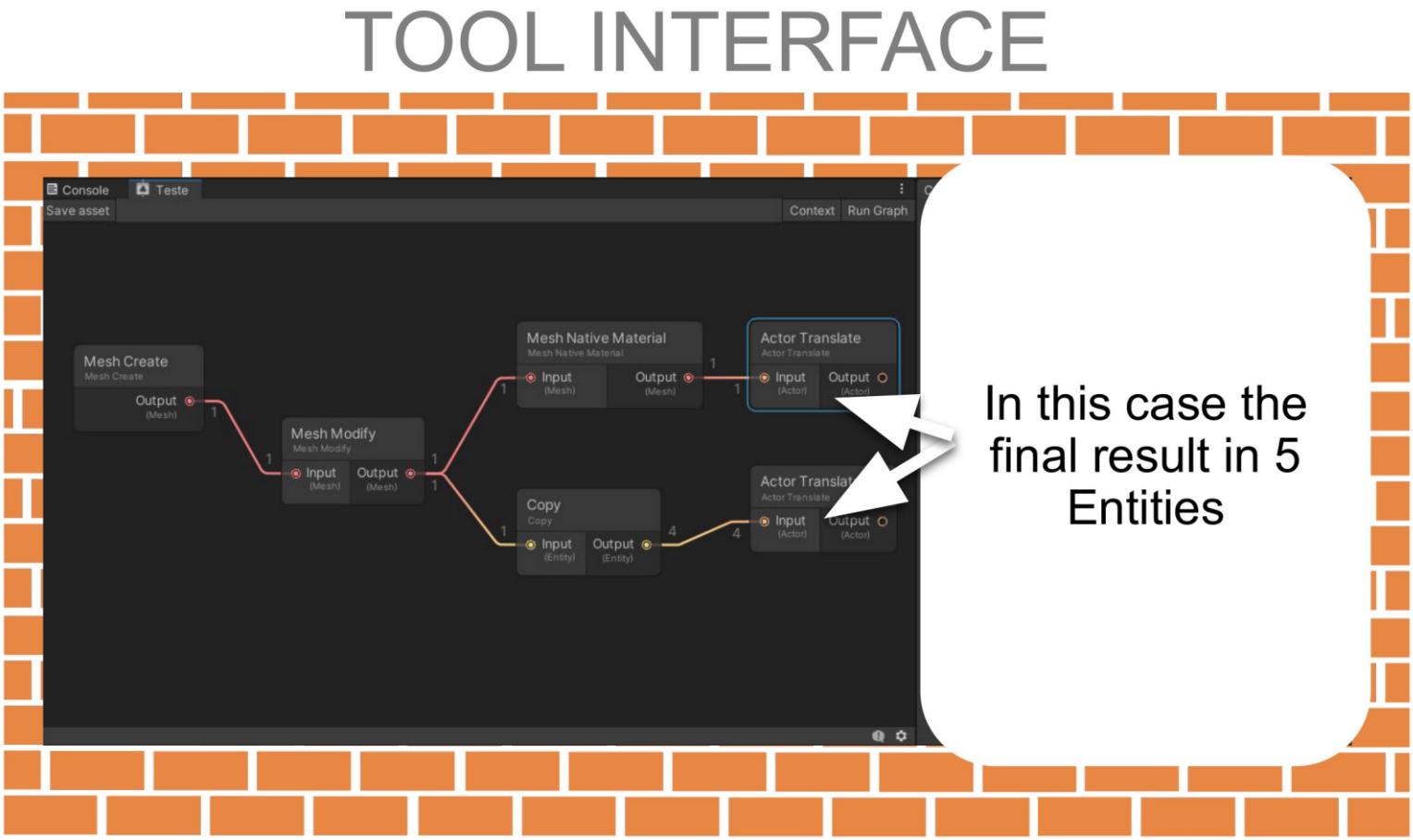


# TOOL INTERFACE

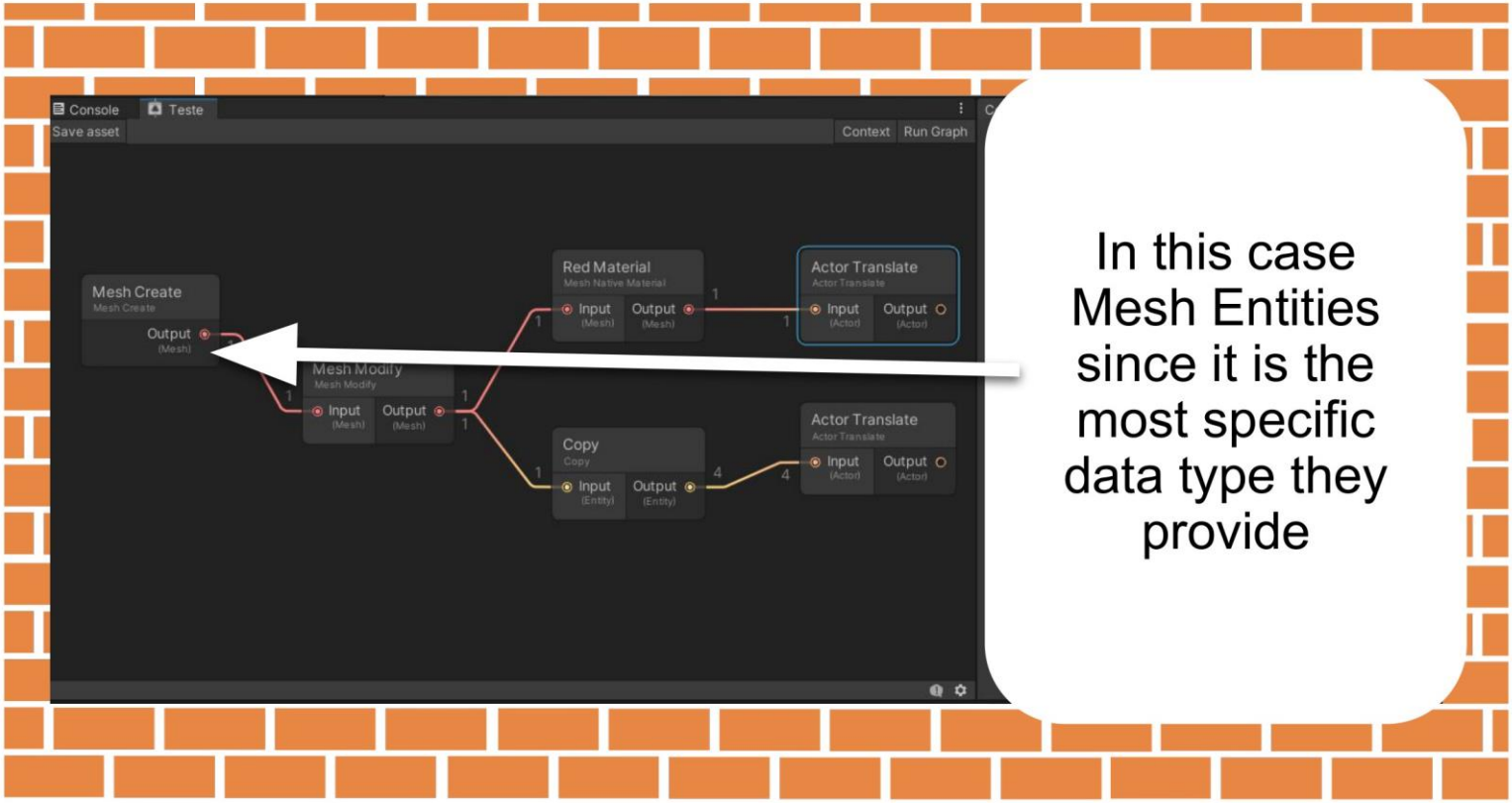


# TOOL INTERFACE

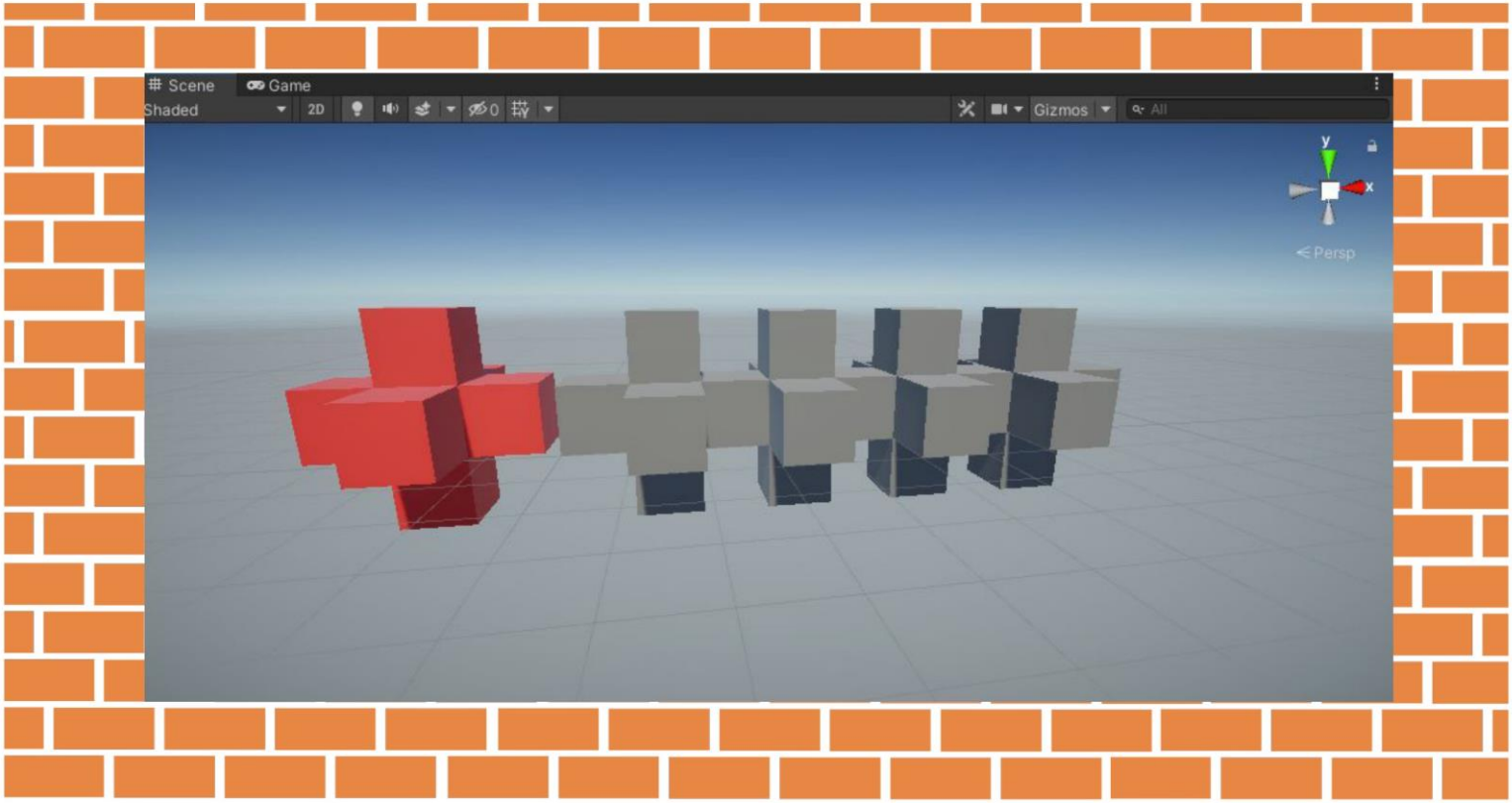


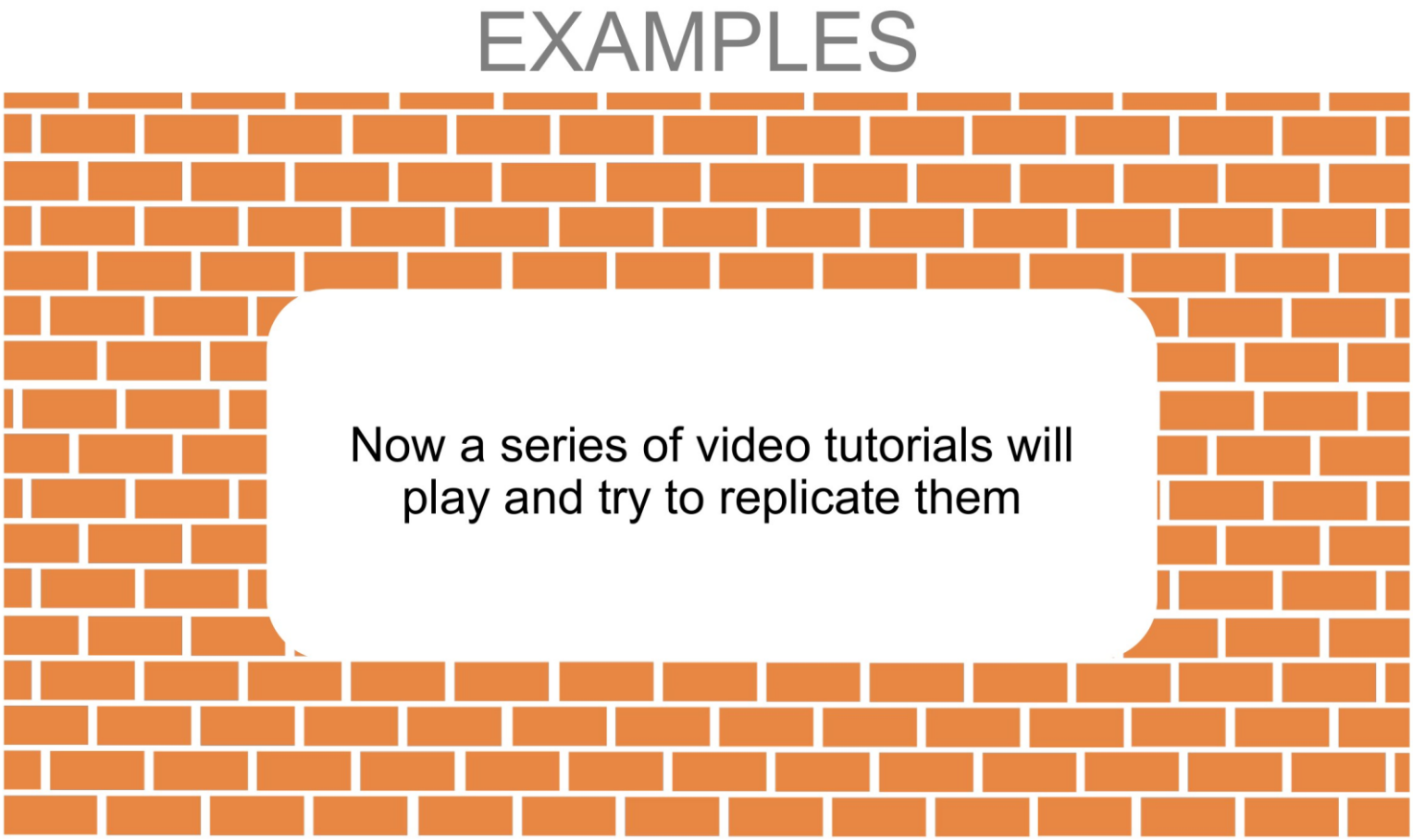

TOOL INTERFACE
Console
Teste
Save asset
Context
Run Graph
Mesh Create
Mesh Create
Output
(Mesh)
Mesh Modify
Mesh Modify
Input
(Mesh)
Output
(Mesh)
Mesh Native Material
Mesh Native Material
Input
(Mesh)
Output
(Mesh)
Actor Translate
Actor Translate
Input
(Actor)
Output
(Actor)
Copy
Copy
Input
(Entity)
Output
(Entity)
In this case the final result in 5 Entities

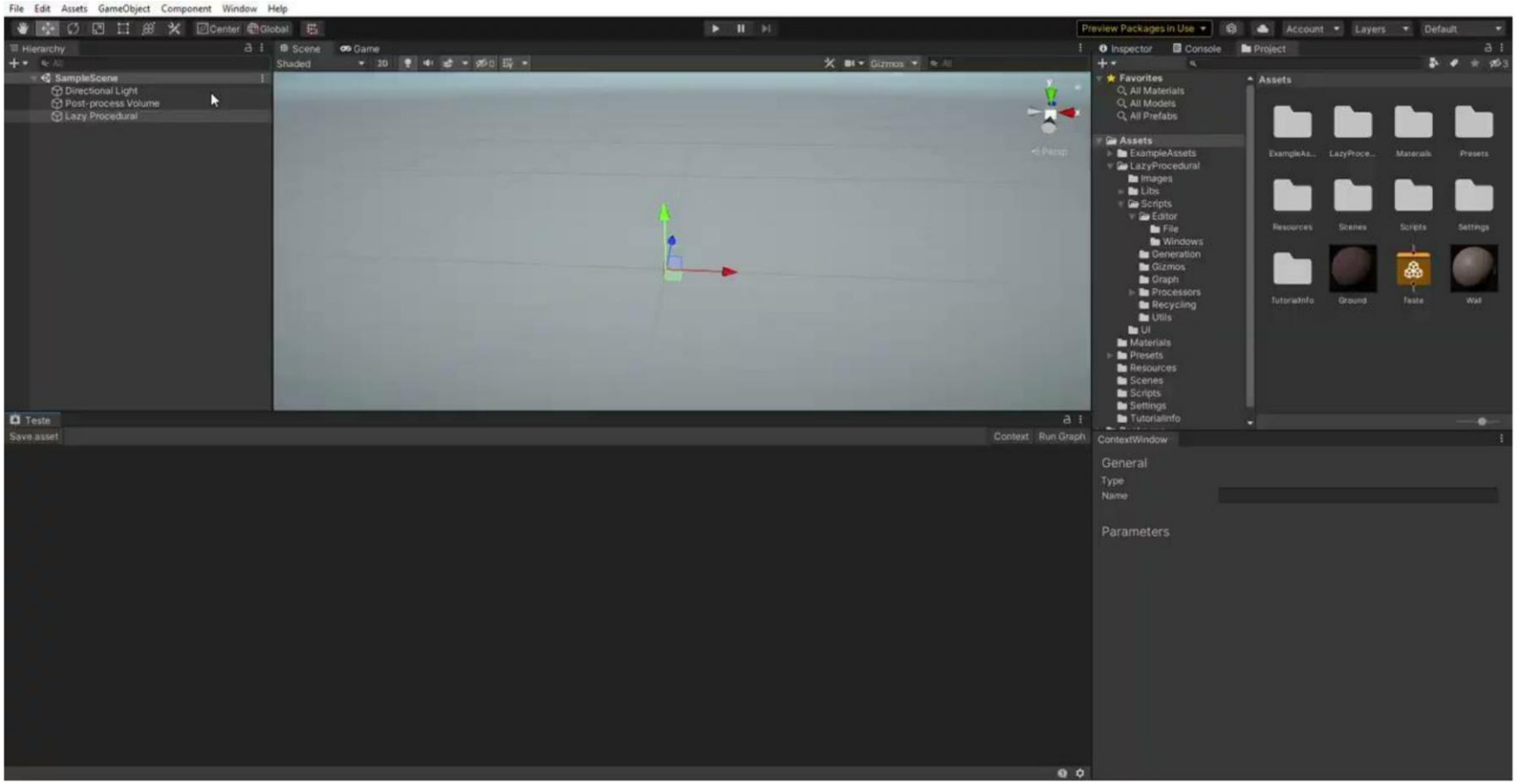
# TOOL INTERFACE



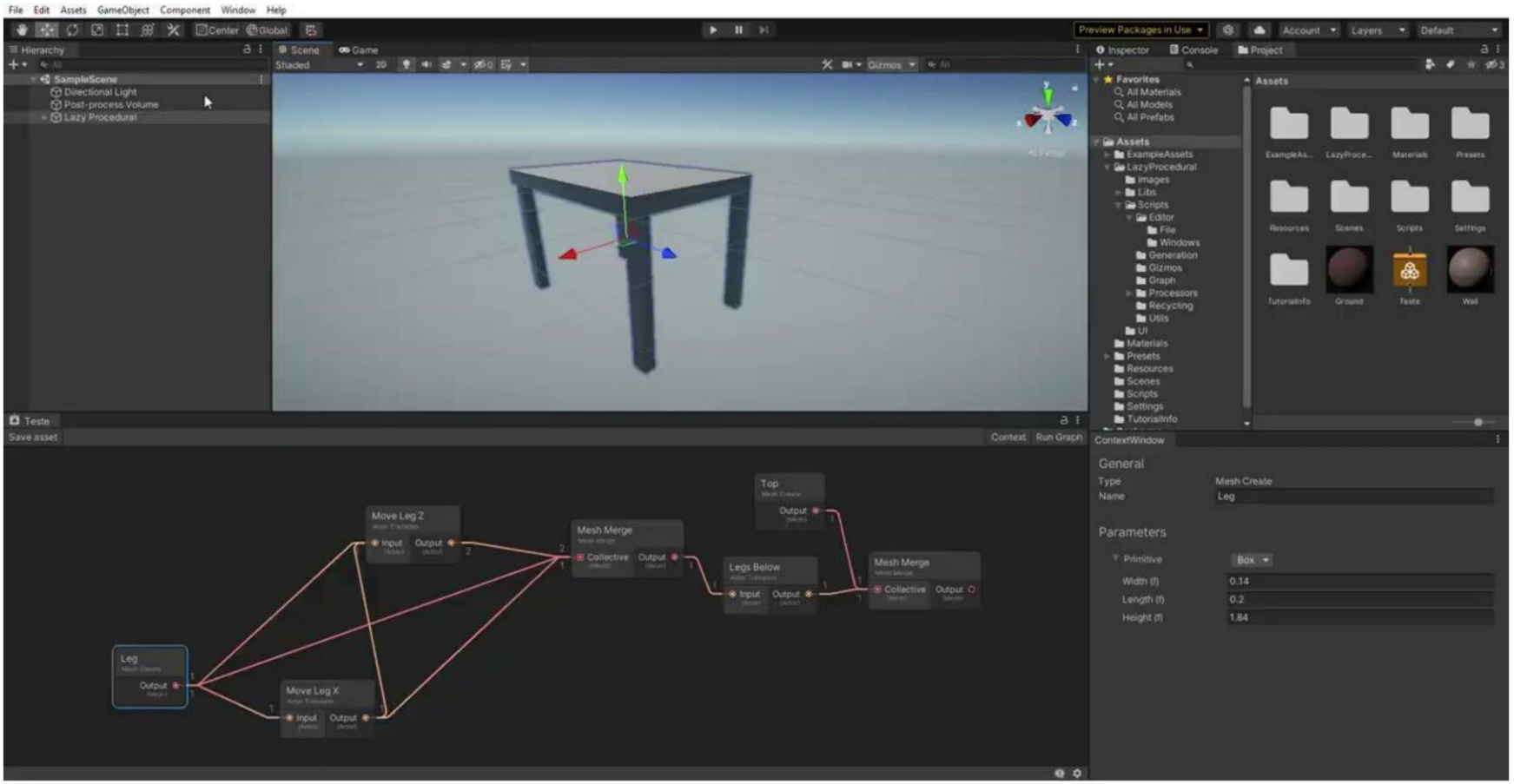
RESULT



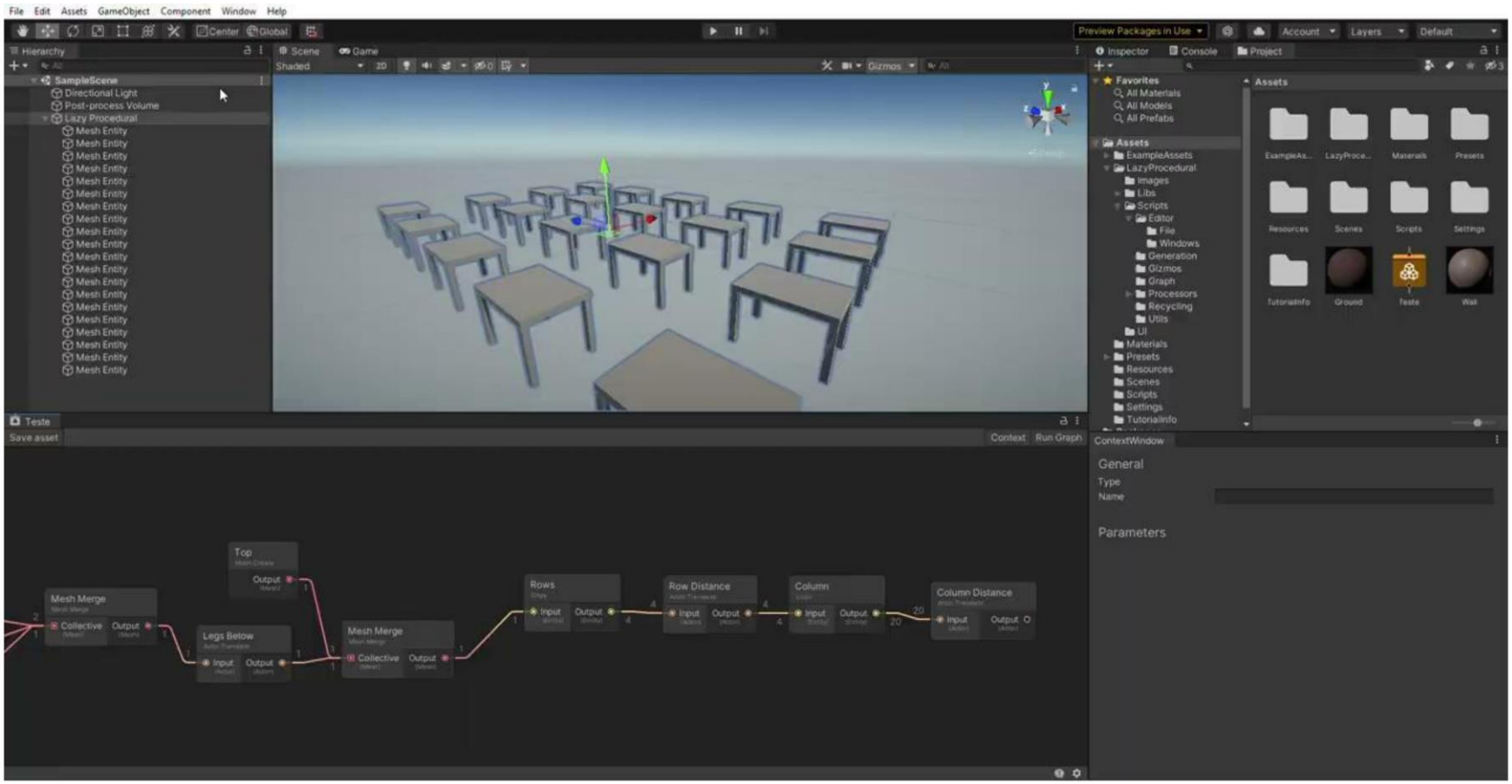

EXAMPLES
Now a series of video tutorials will play and try to replicate them



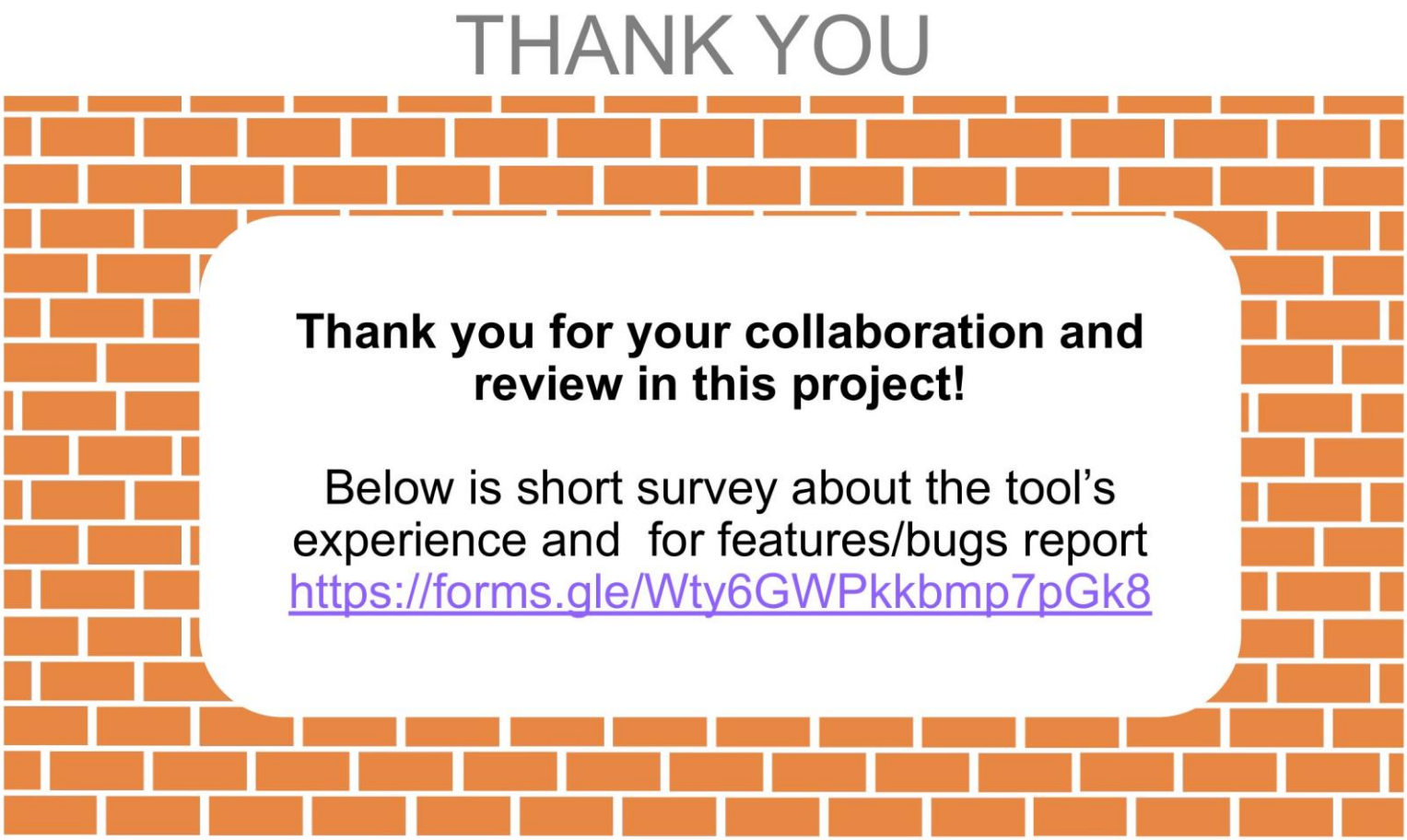

EX1 -CREATING A TABLE




EX2 -CREATING A SERIES OF TABLES




EX3 – RANDOMIZE TABLES


THANK YOU
Thank you for your collaboration and review in this project!
Below is short survey about the tool's experience and for features/bugs report
https://forms.gle/Wty6GWPkkbmp7pGk8