|  |
| --- |
| **Procedurally Generated,**  **Environmentally Responsive Trees**  Stylianos Zachariou  BSc (Hons) Computer Games Applications Development, 2023 |

|  |
| --- |
| School of Design and Informatics  Abertay University |

# Table of Contents

[Table of Contents i](#_Toc134829817)

[Table of Figures ii](#_Toc134829818)

[Acknowledgements iii](#_Toc134829819)

[Abstract iv](#_Toc134829820)

[Abbreviations, Symbols and Notation vi](#_Toc134829821)

[Chapter 1 Introduction 1](#_Toc134829822)

[Chapter 2 Literature Review 3](#_Toc134829823)

[Chapter 3 Methodology 9](#_Toc134829824)

[Chapter 4 Results 25](#_Toc134829825)

[Chapter 5 Discussion 31](#_Toc134829826)

[Chapter 6 Conclusion 37](#_Toc134829827)

[List of References 40](#_Toc134829828)

[Appendices 42](#_Toc134829829)

# Table of Figures

# Acknowledgements

WRITE WATEVER

# Abstract

**Context:**

Trees are a common feature used in game environments; however usually unresponsive, and static. The contrast between the ever-changing shape of trees in real-life and stagnant pre-made models, highlights the false reality of games, therefore, disturbing player immersion. Dynamic trees, growing according to their surrounding environmental factors, could be implemented to increase a game’s verisimilitude as well as player experience.

**Aim:**

The aim of this project is to explore and compare generative algorithms that can be used for the procedural generation of realistic, visually appealing, environmentally responsive trees, in real-time.

**Method:**

Using C++ and the Unreal Engine, an application was constructed, featuring implementations of the space colonization and shortest path algorithms, to procedurally generate environmentally responsive tree structures. A virtual habitat with various environmental factors was developed to act as a setting for tree growth. Surrounding components such as the sun, wind, temperature and soil conditions were customizable and heavily influenced each algorithms finalised tree structures.

**Results:**

The resulting trees were required to be responsive to the environment, realistic, visually appealing and to grow in real-time with a low performance impact. Therefore, the algorithms were compared and evaluated using quantitative and qualitative metrics such as performance calculations and visual realism. Both algorithms performed equally, with each algorithm being visually preferred depending on the surrounding environmental factors.

**Conclusion:**

Although there is room for improvement concerning visual detail, environmental responsiveness and scalability, this project proved the possibility of applying procedurally generated, responsive trees to video game environments without having a high impact on performance.

# Abbreviations, Symbols and Notation

UE5 – Unreal Engine 5

SCA – Space Colonization Algorithm

SPA – Shortest Path Algorithm

FPS – Frames Per Second

# Chapter 1 Introduction

Environmental design is a crucial part of a game’s development process and can directly affect the player’s experience. As game developers strive to increase immersion, interactable and evolving game environments are becoming the new norm. A dynamic game world can not only be more visually appealing, but also reduce monotony and player boredom. In practice, a lot of resources are needed to create such environments, since artists require a significant amount of time to build models and produce animations.

In an attempt to confront this issue, researchers are trying to introduce artificial intelligence, to the development process, for the efficient creation of content with little human interaction. A recent example of this is the use of generative adversarial networks for the creation of new DOOM levels (Giacomello, et al., 2018). Consisting of a generator and a discriminator, GAN are able to create high quality content based on the training material but are still not very reliable. Even though the study’s results were very promising, a lot more research and resources are needed to build and train a model that can successfully compete with human developers. Moreover, AI started being applied to games for content-drafting in recent times, and a lot more advancements are needed to make the results reliable. Real-time generation is even more complex and would require a far more improved network to be able to produce realistic, evolving content. A more typical approach for the creation of video game material is procedural generation.

Procedural generation is widely used in the games industry, mainly for the efficient drafting or on-the-fly creation of content (Short & Adams, 2017). While content drafting can be of great use when producing materials later to be polished by hand, real-time generation is best suited for evolving game worlds. An example of real-time procedural generation are Minecraft’s “infinite” worlds (Mojang Studios, 2011). Using sophisticated generative algorithms, the world is constructed in real-time and can be expanded for up to three billion blocks. The randomness introduced by procedurally generating the map is also sought after, making each world completely unique and providing players with a distinct experience each time. Even though usually adopted for map generation, real-time procedural generation could also be used for in-game procedural modelling.

In the real world nothing is static; given enough time everything eventually evolves or deteriorates. On-the-fly procedural modelling can assist in the replication of this effect, therefore increasing a game’s realism. Trees are a common attribute used in game scenes although usually represented with still, pre-build models. In order to increase immersion, generative algorithms could be used to create evolving tree models.

This project will compare multiple generative algorithms, adapted for the procedural modelling of environmentally responsive tree structures, and explore the feasibility of adapting them in commercial video games.

In real-life tree growth is heavily depended on the environment with some affecting factors being the sun’s position, wind direction, temperature, and soil quality. Since this project is attempting to make trees in games more realistic, the generative algorithms will be challenged to construct the models in response to a simulated customizable environment.

In the following chapters, generative algorithms will be researched and implemented for the procedural development of environmentally responsive trees in real-time. The algorithms will be evaluated based on their visual appeal, realism, and performance effect, resulting in an ideal model, able to procedurally generate trees in any game environment.

# Chapter 2 Literature Review

For the effective implementation of procedurally generated, environmentally responsive trees, extensive research was required, to explore suitable branching algorithms. Alongside efficient real-time generation, it is necessary for the chosen algorithms to achieve realistic growth, therefore, further biological investigation was necessary for the recognition of key tree attributes and affecting environmental factors. In this chapter, three algorithms, commonly used for the procedural content-drafting of trees, will be analysed, and further explored in terms of biological factors affecting trees and effective evaluation methods.

**2.2 The Algorithms**

A picture containing cartoon, clipart, carmine, illustration

Description automatically generatedSCAs are noticeably suitable for the procedural generation of trees, due to their often resulting organic branching structures (Fu, et al., 2023). As the name suggests, these algorithms, starting from a single root point, can successfully expand and cover a large amount of a given volume (Runions, et al., 2005). As described in Runions et al. (2007), the process starts with the initial node being placed where the tree is required to grow. A volume is chosen, close to the root node, and filled with randomly placed attraction points. The colonization begins with vectors being drawn from the original node to nearby attraction points. Those vectors are normalised and added up, with a new node placed in the attraction direction. Each attraction point also has its own decimation radius and will be deleted if a node is detected entering. With the recursion of the SCA and the addition of interpolation while new nodes are added, the tree can seem to be growing in real-time.

Figure 1 - The different stages included in one iteration of the space colonization algorithm. (Runions, et al., 2005)

SPAs can also be employed for the successful procedural generation of realistic-looking trees in real-time. Similar to the SCA, a root node and a volume should be initialized at the required tree location. An irregular graph is created in the volume, with random edge weights and multiple random endpoints. Using the algorithm, the shortest path from each endpoint to the root is found and a mesh is created representing that path. The process is repeated, and a tree structure is constructed (Xu & Mould, 2012). In previous implementations, like XU & Mould (2012), of the SPA, Dijkstra’s algorithm was used due to the performance flexibility content-drafting provides. However, because of the imperative attention to performance in this project, the A\* might provide a more suitable solution, due to its higher efficiency (Wayadhi, et al., 2021). Although satisfactory results can be achieved using this technique, guiding vectors can be utilized, adding more control to the tree’s shape (Xu & Mould, 2015). Guiding vectors will alter the edge weights, therefore, influencing the path between endpoints and root point. Further, guiding vectors are a key to enforcing a controllable amount of naturalistic randomness, while ensuring branch dispersion.

A picture containing flower, art

Description automatically generated

Figure 2 - Branching structures created with the use of guiding vectors and the path-finding algorithm. (Xu & Mould, 2015)

Initially introduced by Lindenmayer in 1968, L-systems is a mathematical theory of simple multicellular organism development, which was later applied to plant growth. (Prusinkiewicz & Lindenmayer, 1990) The main concept of this theory is the recursive replacement of an object’s components using a specific ruleset. For the rewriting to take place, the intended structure must be represented using a string genotype. Each rule defines the replacement of certain symbols with the general formula being: *(symbols) -> (new symbols)*.

A picture containing sketch, drawing, tree

Description automatically generatedThe more iterations of a ruleset on a genotype, the more complex structures will be produced. Theoretically, this process can be repeated an infinite number of times, therefore creating fractals (Prusinkiewics, 1986). Because of their fractal-like recursive structures, trees could be replicated with the employment of simple genotypes and rulesets.

Figure 3 - Tree structures created using different iterations of the L-system with the rule of: (F) -> (F[-F]F[+F][F]). (Shaker, et al., 2016)

Although widely used, L-systems greatly affect performance because of their fractal-like behavioural pattern (Stepney & Verlan, 2018). Since this might threaten the real-time aim of the project, the SPA and SCA will be favoured over it due to their prospect of better performance with minor modification.

**2.3 The Tree and its Environment**

A significant component of the project is the surrounding habitat and its effects on the generated trees. In Fowells & Means, 1990, some major environmental factors affecting tree shape or growth rates were identified and will be used as generation parameters in the application. Firstly, the average temperature, as well as the daily fluctuation in temperature, have an impact on the chemical reactions of trees which directly correlate to the rate of growth (Fowells & Means, 1990). Each type of tree has a range of temperatures in which it can survive and an optimal temperature in which it thrives. The average temperature will be an environmental factor affecting tree generation in the project, however, since the day-night cycle will not be simulated, the daily fluctuation in temperature will not be considered. Moreover, the book recognises light as an influential element. The light’s intensity and quality regulate photosynthesis, while the direction controls the tree’s shape, due to a process called “phototropism” (Kendrik & Kronenberg, 1994). The sun’s position will be used as a parameter for the procedurally generated trees, affecting the amount of tree branches created towards the direction of the light. Moisture and soil quality are also established as major environmental factors impacting a tree’s rate of growth (Fowells & Means, 1990). Soil quality is a broad term including texture, depth, acidity and structure. In the application, both moisture and soil quality will be implemented with the simplification of soil quality to the single parameter of soil acidity. Lastly, although minor, constantly strong wind is mentioned to affect a tree’s shape with most of its branches growing towards the wind’s direction. Therefore, wind will be the last element added to the simulation, affecting the trees’ new branch growth.

The optimal values of these environmental factors greatly vary between different types of trees, thus, for this project the English Oak (“Quercus robur”) will be used as a template for the generated trees. In Gilman & Watson, 1994, common morphological and behavioural patterns of English Oaks are identified. This species of trees is found in temperate regions; hence, the cardinal temperature range is 4oC - 41oC, with the optimal being 25oC - 30oC (Gilman & Watson, 1994). To simulate this effect, the generated trees will grow the fastest when the temperature is optimal, with the rate being reduced while the surrounding temperature moves further from the 25oC - 30oC range. If the temperature is not in the cardinal range, the tree will not be able to develop. As most trees, the English Oak’s crown is affected by the light’s direction, with more branches growing in positions with increased exposure to the sun. This will be applied to the procedurally generated trees by increasing the probability of nodes being created towards the direction of the light. Moreover, even though the English Oak can adapt to various levels of soil acidities (4.5ph-8ph), a certain amount of moisture is required for it to thrive, specifically, occasionally wet to dry soil. In the project, if the soil conditions are not within the defined boundaries, the trees will grow much slower, with extreme cases completely stopping growth. The English Oak is known to be windfirm, meaning it can withstand high velocity winds without breaking, however, similarly to most trees, the crown can be affected by strong continuous winds in one direction (The Royal Horticultural Society, 2022). In most cases, there will be an increased number of branches growing in the direction of the wind. Depending on the wind direction and power, the generative algorithms will have a higher probability of spawning new nodes in the direction of the wind, therefore morphing the tree in the wind’s direction. Moreover, a range of impacting wind will be enacted of 4-10 Beaufort, rendering only high winds as effective (National Weather Service, 2007). According to the Royal Horticultural Society, the English Oak needs 20-50 years to fully develop, with a minimum height and spread of 12m and 8m respectively. Usually, a wide trunk and drooping branches with a reddish brown colour can be observed by a fully mature English Oak tree. In an attempt to mimic the size and shape, an area will be defined where the tree would be available to grow into. The area will slowly grow throughout the simulation to a maximum amount of 18m height and 15m spread. The trunk and branches will be correctly textured and slowly grow thicker in a similar rate to the procedurally generated tree’s, real-life counterpart.

**2.4 Visual Representation**

Branch node creation based on real-life data is only a part of procedurally generating realistic trees. To increase realism, the generated trees have to be represented using a uniform, textured and correctly lit mesh. In the Unreal Engine’s documentation, the “UProceduralMeshComponent” was found as a suitable solution for the creation of the required mesh (Epic Games, 2022). The component takes in vertex positions, UV coordinates and normal vectors to create a procedural mesh. Before the creation of the mesh, a circle will be calculated at a certain radius using each nodes position and rotation as the centre. Equidistant points will be created on the circumference of the circles and will be used as the vertex positions. Lastly, the UV coordinates will be calculated, between each pair of circles, using the cylinder formula. As time progresses, and the number of branches increases, the radius of circles surrounding older nodes will increase, therefore, rendering older branches thicker.

**2.5 Evaluation**

After completion, the project requires evaluation to determine whether the initial aim was accomplished. Realism and visual appeal are qualitive measures and opinions may vary between different people; thus, a survey will be constructed and distributed for the collection of ratings for the procedurally generated, environmentally responsive trees. In order to maximize the effectiveness of the survey, Hamed (2016) was closely studied. The key advice stated in the article is: the questions should be unambiguous, simple, with the avoidance of technical language (Hamed, 2016). Therefore, the survey will consist of multiple images and videos of generated trees using all three algorithms and variations of the environmental factors’ values, paired with the simple questions: “How realistic do you find this tree?” and “How visually appealing do you find this tree?”. The user will be able to choose an answer between five numerical ratings, with number one signifying the least satisfaction, gradually increasing up to five, signifying the highest satisfaction. Moreover, the size of the survey has an impact on engagement, so the survey will have an estimated completion time of ten minutes maximum. The data collected will later be used to give each algorithm a realism rating, which will be used as part of the overall evaluation.

**2.6 Conclusion**

To conclude, even though three algorithms were researched, only two will be implemented, due to their prospective low performance effect and high visual appeal, the SPA and SCA. The two algorithms will be modified to generate trees resembling the English Oak’s short trunk and thick crown, while also realistically dynamic to their surrounding environment. The trees will be displayed using the UE5’s procedural mesh component, with manual vertex and UV coordinate calculations. The realism and visual appeal of each algorithm will be later evaluated using a short survey consisting of various photographic material.

# Chapter 3 Methodology

The procedural generation of trees is often employed in game development, for the drafting of models, later to be polished and added to environments by artists. This chapter specifies how generative algorithms, commonly used for the creation of branching structures, are adapted, and evaluated for the creation of tree models, responsive to their surrounding habitat, in real time.

After consideration of multiple engines and coding libraries, the UE5 was chosen, for the implementation of the two generative algorithms, due to its modifiability, high quality visuals and useful built-in functionality.

**3.1 Tree Seed**

The tree seed is the base class for the generative algorithms. It includes a scene component, for the placement of the trees in the scene, and variables, customisable from the editor, affecting the trees’ mesh, environmental sensitivity, and feature attributes. Moreover, the class contains functionality, essential to both algorithms, such as mesh calculations and tree crown adaptations based on environmental factors.

**3.1.1 The Environment**

The environmental factors, identified in the literature review, were applied to the simulation with the creation of the “EnvironmentSettings” class, inheriting from the UE5’s “WorldSettings”. The inheritance allows all trees in a level to be affected by the same environmental conditions, due to each level having its own, unique world setting values.

Since the algorithms are attempting to replicate the growth of the same tree species (English Oak), both should be identically affected by their habitat. Therefore, the “ApplyEnvironment” function was constructed, in the “TreeSeed” base class, for the adjustment of tree attributes in relation to the level’s environment settings.

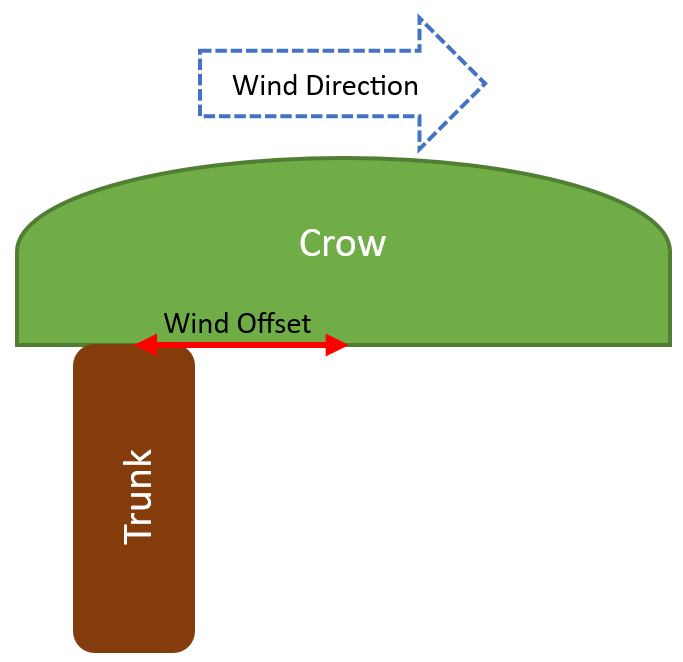
Each tree’s crown position and rotation are affected by the wind’s direction, power, and light’s position. As mentioned in chapter 2, the English Oak is a relatively wind resistant tree species, thus, the wind’s direction only influences the crown if its power is more than 25mph (6 Beaufort). The maximum affecting wind power is set to 63mph, (10 Beaufort), since stronger winds normally uproot trees and would have caused improbable results. Using the wind’s values, an offset vector is calculated, with the maximum distance being equal to the crown’s radius (Figure 4). The wind offset is stored and later used by each algorithm during the node spawning process.

Figure 4 - Wind offset's effect on the resulting tree shape

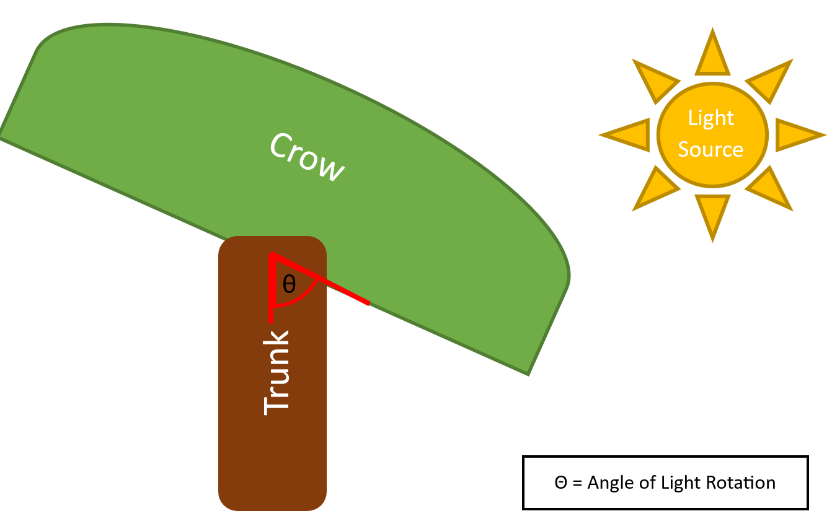
Realistically, the light is required to act as an attractive force on the tree’s crown. Hence, a rotator is calculated in accordance with the direction vector between the tree and light’s position (Figure 5).

Figure 5 - Light rotation's effect on the resulting tree shape

To prevent exaggeration, a pitch angle limit is applied to the rotator, in order to restrict the crown’s downward growth.

The remaining environmental factors, established in the previous chapter (soil PH, moisture, and temperature), have an effect on the tree’s rate of growth instead of the crown. The rate of growth determines the speed with which a tree’s branches grow and is depended on the closeness of the environmental factors’ values to their cardinal ranges. Initially, the rate is set to its maximum value and gradually decremented by each factor with a non-optimal value. If any of the factors has an extreme value which would normally prevent a tree’s development, the rate of growth is set to 0, preventing any new branches to grow.

Table 1 - Environmental factor values and effect on trees.

|  |  |  |  |
| --- | --- | --- | --- |
| ENVIRONMENTAL FACTORS AFFECTING TREE GROWTH | | | |
| Environmental Factor | Affecting Value Range | Optimal Value Range | Effect On Trees |
| Wind | 25mph - 63mph | - | Offset between the centre of the crown and trunk. |
| Light | - | - | Rotation of crown towards the source. |
| Temperature | 4 oC - 41 oC | 25oC - 30oC | Affects rate of growth, halting growth if value is not in “Affecting Value Range” |
| Soil Moisture | 10% - 65% | 20% - 45% |
| Soil PH | 2.5ph – 11ph | 4.5ph - 8ph |

**3.1.2 Visual Representation**

Due to the complex nature of branching structures, the manual creation of the trees’ visual representation was required. Allowing the generation of separate mesh sections for the construction of a composite visual, the UE5’s procedural mesh component was added to the Tree Seed base class.

Both algorithms’ resulting trees consist of multiple nodes, each holding information of their transform, direction vector and location of their parent node (previous node on the branch). For each newly grown node, new vertices and UV coordinates have to be identified. The process begins with the definition of a circle around the required node. The circle’s radius is calculated based on the number of the node’s children and is rotated using its direction vector. The growth in radius is an additional factor enforcing naturalistic tree generation, with a logarithmic growth rate.

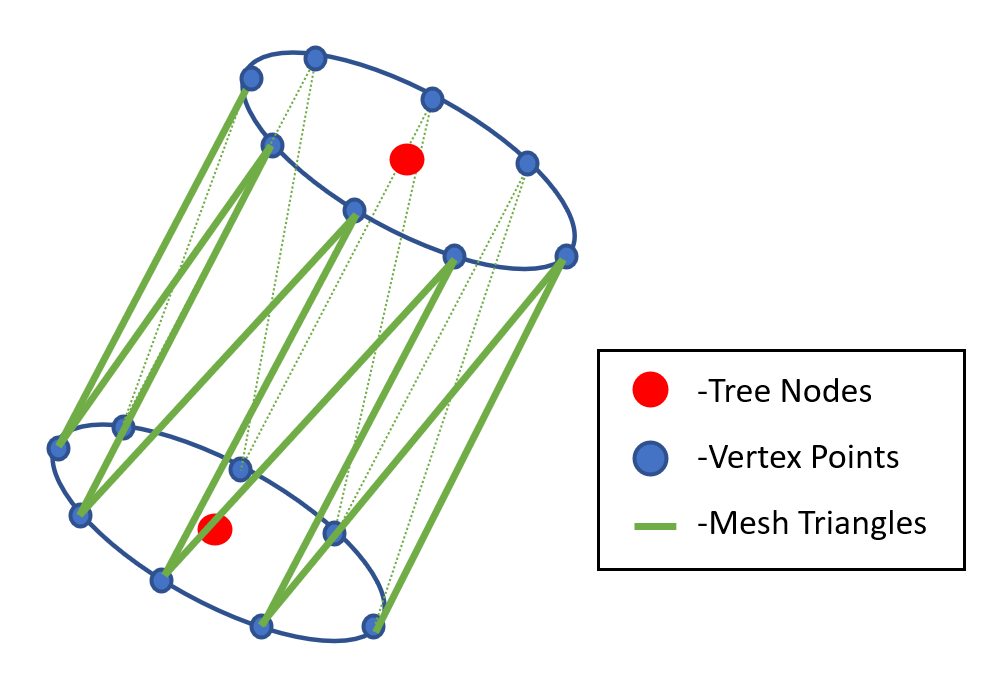
Subsequently, equidistant points are determined on the circumference of the two dimensional shape, with the aid of the “CalculateNodeMeshVerticesAndUV” function, found in the “TreeSeed” class, with their three dimensional coordinates stored in the array of vertices. Moreover, the function calculates each location’s corresponding UV coordinates and stores them in the appropriate array. This method is also repeated for the node’s parent with the only exception being the circle’s rotation, which is rotated identically to the child’s circle (Figure 6).

Figure 6 - Mesh creation method

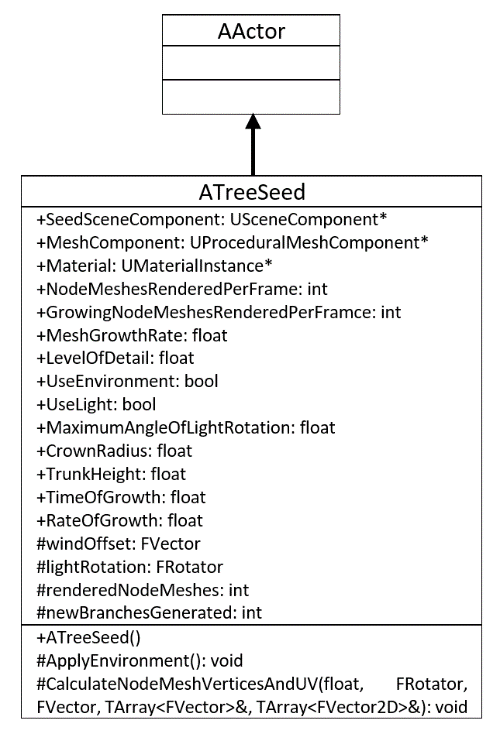
The similarity in rotation ensures the consistency of the required branch thickness and the depletion of visual bugs created from abrupt changes in direction. After all the vertices have been established, weaving instructions for the creation of mesh triangles are necessary. Therefore, the “CreateGridMeshTriangles” function, provided by the UE5’s kismet procedural mesh library, is purposed for the population of the triangles array containing vertex indices. The three arrays are then passed to the procedural mesh component’s “CreateMeshSection”, which generates and renders a new mesh section, representing the newly grown branch section.

Since branches grow randomly, often with sharp changes in direction, a spherical mesh section is also rendered for every new branch section. The sphere is created using the new node’s location and transform and adopts a radius equal to the node’s circle of vertices. The spheres assist in the formation of a seamless resulting visual component representing the generated trees.

The above described procedure, of composing a tree’s mesh, is encapsulated in the “CreateMesh” function. This function loops through the array of all spawned node’s and develops each node’s mesh sections. Already rendered node’s have a stored mesh section index, which can be used to identify and update their corresponding vertices. To prevent unnecessary calculations, the function is only called after the creation of new nodes.

While striving to increase realism, the importance of the slow evolution of the mesh, became apparent. Hence, the “GrowBranches” function was created for the progressive growth of new branches. This function loops through all newly generated nodes, initially added to the growing node array, and calculates a growth progress percentage based on time and the pre-determined “rate of growth” attribute. The mesh section creation process is then repeated, however, instead of the growing node’s translation used for the first circle of vertices, a location between the node and its parent is determined using the progress percentage. Once the growth progress is complete, the node is removed from the growing node array and added to the final node array which is later used in the “CreateMesh” function.

**A screen shot of a computer

Description automatically generated with low confidence3.1.3 UML Diagram**

**3.2 Space Colonization**

As discussed in the literature review, the space colonization algorithm (SCA) was an undeniable choice for the procedural generation of trees, due to its ability of producing branching structures by mimicking actual plant behaviour.

**3.2.1 The Nodes**

Firstly, some guidance is required to control the shape of the growing branching structure. Therefore, the “AttractionNode” class was created, inheriting from the UE5’s actor class, allowing it to be placeable in the world. This primitive class is compromised by a scene component and a collider, both important for the nodes’ placement and detection.

Moreover, the SCA’s growth pattern demands multiple simultaneous, actively developing branches, deeming the construction of a new “TreeNode” class necessary. Objects of this class are not only responsible for carrying out essential growing location calculations but also represent already grown branches. This class includes a scene component and a collider, similarly to the “AttractionNode” class, along with two distinctly sized, spherical, trigger colliders. In order to determine the position of the next node, all surrounding influences should be considered, thus, the larger collider is purposed as a detector. The detection sphere is set to generate overlap events, with the construction and assignation of new custom overlap begin and end functions. The “OnOverlapBegin” function is automatically executed when a begin overlap event is observed and is initially responsible for examining the colliding actor. If the actor is an object of the “AttractionNode” class, it is added to the attraction influences array, while if it’s an object of the “TreeNode” class, it is added to the detraction influences array. The orginal SCA only requires the use of attraction points, however, with additional experimentation, it was discovered that with the awareness of tree nodes as detraction points, the clamped spawning of tree nodes could be prevented.

To further restrict node clamping, the second, smaller sphere collider was purposed as an attraction node “kill” radius. Each tree node’s combined influences, commonly lead to a specific location, which when reached can cause a blockage. Hence, with the destruction of attraction nodes, close in proximity to tree nodes, continuous branch growth is ensured, with the altercation of external influences, affecting new node location calculations.

Being responsible for determining the next node’s location, the “TreeNode” class includes the “CalculateNextTreeNodePosition” function, which performs the necessary calculations and stores the resulting location in an accessible variable. Since the tree’s trunk and crown are generated with two different rulesets, two distinct node spawning calculation methods were implemented.

Tree nodes comprising the trunk, need to be able to grow without the impact of any external influences, accordingly, the nodes’ direction vectors are used to devise new spawning locations. A node’s direction vector is determined directly after it was spawned with the help of the, “TreeNode” class’s, “CalculateCurrentDirection” function. Receiving the location of the node’s parent as a parameter, the function calculates the direction vector using a basic vector subtraction. Since this direction vector should symbolize the direction in which the next node must spawn in, a random vector is also added, providing some lifelike randomness to the simulation.

Without it being the sole affecting factor, the reserved direction vector is also used by growing crown branches. New crown nodes are generated by considering all external influences and combining them into a single average vector. Because of the two types of influences, an average attraction and detraction vector are initialized, with values based on the product of additions between all attraction and detraction influences’ directions, respectively. The node’s direction vector is also included in the average attraction value, with an equal significance as any other attraction node. The vectors are then normalized, the detraction value is subtracted from the attraction value, and the results are saved in a general average vector.

These growing techniques are distinguished by the “CalculateNextTreeNodePosition” function’s “useDirection” boolean parameter, provided by the SCA. As both approaches result in a single vector variable, the resulting position is found by multiplying the vector’s value to the tree’s branch length attribute, also passed as a function parameter, and adding it to the current node’s position.

With the implementation of the above described functionality, the “TreeNode” class’s need for certain limiting factors, for the prevention of uncontrollable tree node growth, became clear. Subsequently, particular confining factors were added: the maximum number of children and surrounding tree nodes halting new node spawning, and an expiration time. The maximum number of children attribute restricts unstoppable node spawning from largely stimulated nodes, while the maximum number of surrounding nodes ensures the avoidance of crowded tree node areas. The time limit applied to tree nodes, optimises the algorithm by removing old, stale nodes from new branch location calculations.

**3.2.2 The Algorithm**

Even though the two node classes incorporate most of the required methods for the SCA, a new “SpaceColonizationTreeSeed” class, inheriting from “TreeSeed”, was created to combine all functionality and control the pacing of the algorithm.

Objects of this class can be placed in the scene, wherever needed, to generate a tree, optionally using the surrounding environmental factors. Firstly, the simulation starts in the “SpaceColonizationTreeSeed” class’s “BeginPlay” function, called automatically by the engine at the beginning of the application. This function sets the foundation for the algorithm by calling the “ApplyEnvironment” method, detecting the surrounding habitat’s conditions, and accordingly regulating some of the tree’s attributes. These attributes are then used for the spawning of the crown’s attraction points, in the “CreateAttractionPoints” function. As the English Oak has an indisputable semi-spherical crown, all attraction points are aimed to spawn in random locations inside the required shape, with a pre-determined radius. This is done by initially choosing random positions in a cube, with edges equal to the sphere’s diameter, and saving the ones in the required semi-sphere, while the other positions are discarded and re-calculated (Figure 7). Depending on the tree attributes, based on the environment, a transform is constructed to alter each attraction point’s position, affecting the final crown’s placement.

A picture containing sky, outdoor, grass, cloud

Description automatically generated

Figure 7 - Attraction node crown created by the SCA

After the completion of the attraction point semi-sphere, the “BeginPlay” function spawns the first tree node using the “SpawnNewNode” function, indicating the beginning of simulation. This function takes a parent “TreeNode” pointer as a parameter, however, since the first node has no parent, a null pointer is passed instead. Receiving the empty pointer, the function responds accordingly, creating a parentless node, saving it in the growing node queue.

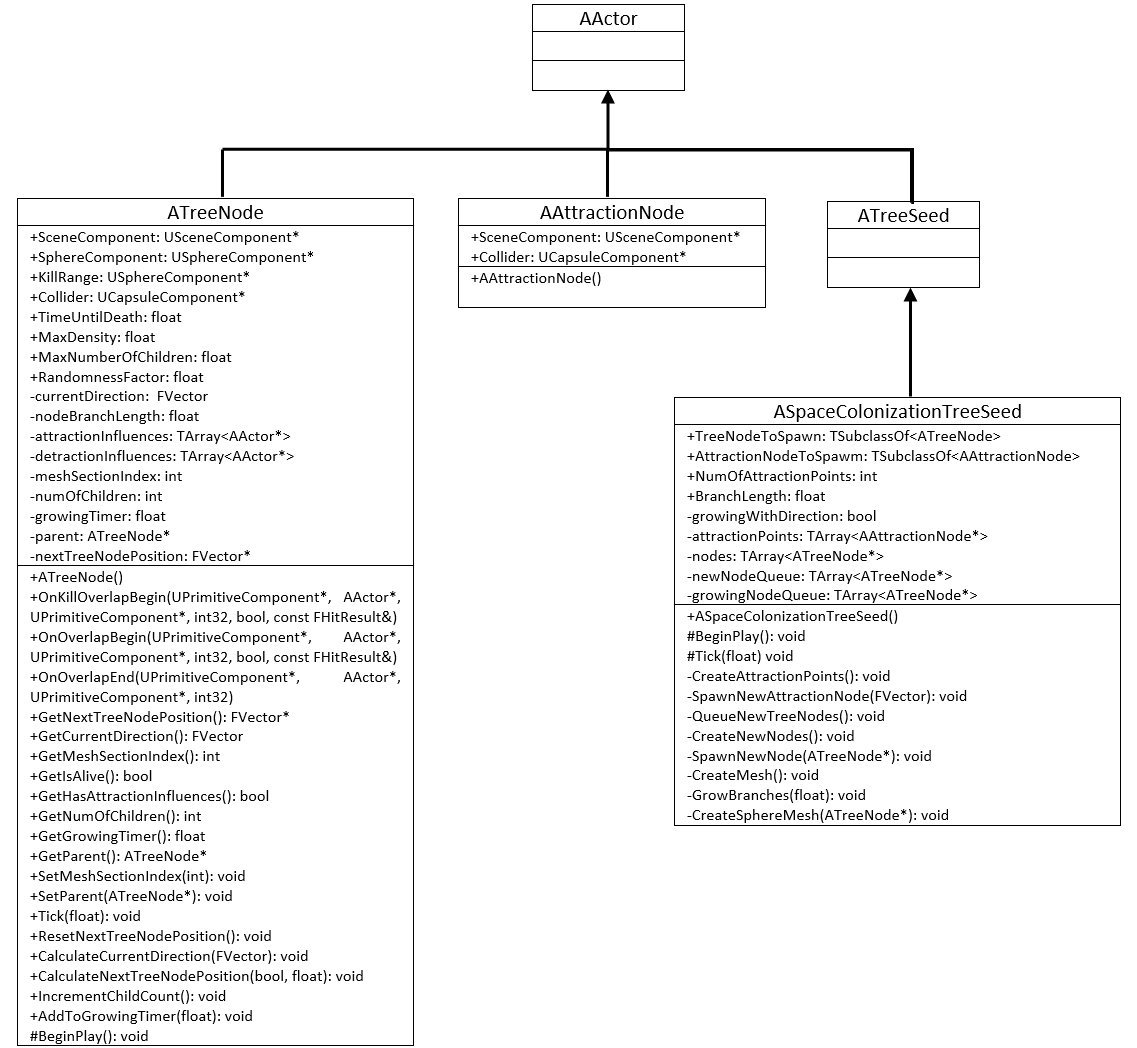
Since only one node is initially created, a branch is not able to grow yet, therefore, the node is moved to the final nodes array. The lack of growing nodes prompts the class’s “Tick” function, which is called every frame, to call the “QueueNewTreeNodes” method. Aiming to queue new tree nodes for spawning, this function loops through the final nodes array and inquires the validity of each active node’s next tree node position pointer. If the pointer is not null, the parent tree node is added to the new node queue array. While the new node queue is not empty, the “Tick” function is required to call the “CreateNewNodes” function, responsible for spawning the queued nodes. Therefore, the “SpawnNewNode” is called three time per-frame, until all queued nodes are spawned. Additionally, the “SpaceColonizationTreeSeed” class is responsible for checking whether the attraction point crown has been reached by the tree nodes, saving the information with boolean data structure. This boolean is passed as a parameter to each tree node, when spawned, to regulate the next tree node location calculation technique.

With the detection of newly spawned tree nodes, the “GrowBranches” function is called, by the “Tick” function, and slowly grows branches, starting from parent nodes to the new nodes. When all new branches are fully grown, the whole procedural mesh is updated by the “CreateMesh” function, offering a visual representation of the tree’s current state to the user.

The above described procedure, compromises one step of the SCA, which is repeated until the maximum time of growth is reached.



Figure 8 - A completed SCA tree without the effect of environmental factors.

**3.2.3 UML Diagram**

**3.3 Shortest Path**

The SPA (SPA) was the second algorithm chosen for the procedural generation of environmentally responsive trees, due to its high efficiency, aiding in real-time generation, and path branching ability.

**3.3.1 Guiding Vector**

In order to employ the SPA, a graph was required, for supplying an infrastructure where the algorithm would be allowed to operate. As discussed in the literature review, the graph should be consisting of guiding vectors for the realistic rendition of branching structures, therefore, the “GuidingVector” class was created, inheriting from the UE5’s actor class.

The class consists of a scene component, providing transformation abilities, with a sphere and capsule colliders, for detection and communication with adjacent guiding vectors. Moreover, the class includes essential functionality, such as the partial creation of the graph and algorithm specific score calculations.

While constructing the graph, connections should be established between neighbouring nodes. Thus, the “DetectConnections” function was introduced to the “GuidingVector” class along with an initially empty, connections array. When called, the function uses the object’s sphere component to detect any overlapping nodes, adding them to the connections array. Depending on the radius of the sphere collider, this function may occasionally be unable to locate any surrounding objects, therefore, leaving certain nodes stranded. This might cause further issues if the secluded node is later appointed as an endpoint, which would render the completion of the algorithm impossible. To prevent node isolations, a while loop was added to the function, which keeps increasing the radius of the sphere component until an overlap actor is detected. The incrementation in size is gradual as to not create excessive or unexpected connections.

The fundamental difference between a regular node graph and the guiding vector graph, used in this simulation, is that guiding vectors point towards a specific direction which provides a more beneficial score, when followed by the SPA. In this implementation, guiding vectors are crucial for influencing the growth of branches, hence, their direction was made to point outwards, aiding in the spreading of the crown, although still including some naturalistic randomness. The “SetThisAsConnectionsParent” function, was constructed in the “GuidingVector” class, and is a key component on determining each guiding vector’s direction. The function loops through all the node’s connection and sets it as their parent, converting the regular graph nodes into guiding vectors pointing away from the node. Furthermore, if required, it can be used by specifically chosen nodes to purposely manipulate the direction of guiding vectors.

The SPA used for this application is the A\* algorithm. This algorithm calculates and compares each node’s F score using the goal location. Since the calculated scores are node specific, a “CalculateFGScores” function was constructed in the “GuidingVector” class. The function takes the end location position as a parameter, as its essential for determining the score, and initially calculates the node’s G score. The G score represents the distance of the whole path completed to reach the current node and is later to be used during the F score calculation. The distance between this and the preceding guiding vector is calculated, with the amount halved if the previous guiding vector was pointing towards this direction, therefore applying the guiding bias. The current G score is then measured by adding the distance and the previous guiding vector’s G score. The F score is then calculated with a single equation, adding the G score with the node’s distance to the goal location. The F score is then saved to be later used by the algorithm.

**3.3.2 The Algorithm**

For the execution and organisation of the SPA, a separate “ShortestPathTreeSeed” class was created, inheriting from the “TreeSeed” base class. The inheritance not only allows the placement of the class’s instances in the scene, but also provides essential environmental adaptation and visual representation rendering capabilities.

Starting from the “BeginPlay” function, immediately called at the beginning of the application, the surrounding environmental factors are identified and used to alter the appropriate growing attributes, with the exploitation of the “ApplyEnvironment” function. The adjusted attributes will be used for the corresponding spawning of the guiding vectors, however, being the goal position of the SPA, the first guiding vector is placed without any external influences to ensure its connection to both, trunk and crown. Therefore, a transform is constructed, only using a z-axis translation equal to the trunk’s height, to calculate the initial guiding vector’s location. The position vector is later passed, as a parameter, to the “SpawnGuidingVector” function, which spawns a new object of the “GuidingVector” class at the required location.

There are two functions constituting the spawning of all the simulation’s guiding vectors, “CreateTrunk” and “SpawnAllGuidingVectors”. The “CreateTrunk” is firstly executed, in order to create a vertical line of nodes, below the initial guiding vector, that will, when connected, formulate the tree’s trunk. It starts by spawning a single node at the base of the trunk and establishing it as an end point, making use of the guiding vector’s “SetEndpoint” function. Thereafter, while iterating through positions between the trunk’s base and peak, new nodes are spawned at pre-determined intervals, with the addition of a minute random offset. Although minimal, the offset provides a sense of naturalistic growth to the trunk without the production of extreme or unrealistic results.

In order to complete the irregular graph’s foundation, the “SpawnAlllGuidingVectors” function is then called. As described for the previous algorithm’s attraction node crown construction, a semi-spherical shape should be instituted in an attempt to mimic the tree’s real-life counterpart’s (the English Oak) shape. Therefore, a specific number of random positions in a cubic volume, with edges equal to the required sphere’s diameter, are selected. The nodes located in the desired semi-spherical top half are spawned and saved, while the rests’ positions are re-calculated. During the creation of the nodes, a transform is also produced using all environmentally adapted attributes, causing the crown to realistically respond to its surrounding habitat.

After their initialization, the “BeginPlay” function loops through all nodes, calling each’s “DetectConnections” function, weaving them into a collective irregular graph. As discussed, in order to convert nodes to guiding vectors, parents need to be established. Therefore, the algorithm’s “SetParentsGuidingVectors” function calls all existing nodes’ “SetThisAsConnectionsParent” function, converting each node’s neighbours to guiding vectors. This process starts with the base crown node, dictating the crown’s middle guiding vector directions to ensure the outward spreading of branches, while guiding vectors’ directions A picture containing outdoor, sky, grass, cloud

Description automatically generatedfurther away from the centre are unpredictable.

Figure 9 - Guiding vectors created at the beginning of the simulation by the SPA.

The final necessary preparation stage is the random election of endpoints. Endpoints mark the beginning of each path, or the end of each branch, needed to be traced to the initial crown base node by the SPA. The “ChooseEndpoints” function was created for the arbitrary picking of a pre-determined number of endpoints. The function includes the choosing of random indices, each representing a guiding vector, converting the corresponding node to an endpoint. The endpoint’s index is then saved in a separate array, marking the end of the algorithm’s preparation.

Prior to any further implementation of the SPA, two new essential arrays were initialized, the visited and unvisited node arrays. At the end of the “BeginPlay” function, the first end point is added to the unvisited nodes array, inciting the launch of the simulation. The “ShortestPathTreeSeed” class’s “Tick” function, then starts to automatically execute every frame by the engine. This function is responsible for progressing and, when required, reset the A\* algorithm, while also rendering and updating the tree’s visual component.

Initially, the “Tick” function confirms the existence of end points. If none are found, the algorithm is completed, however, if the endpoint indices array is populated, the “StepAStarAlgorithm” function is called. This function aims to move the SPA by one step each frame and returns a boolean representing the state of completion of the algorithm (True: Path completed, False: Path not yet completed). The process begins with the iteration of the unvisited node array, with the guiding vector owning the lowest F score declared as the current node. The current node is then validated and diverted to the visited node array. If the current node is not the algorithm’s goal node, all its undiscovered connections are obtained and added to the unvisited node array, while also calculating their scores using the “CalculateFGScores” function. This allows each step of the algorithm to explore more guiding vectors, drawing a path to the target position. Nevertheless, if the function has reached the required goal node, the path needs to be saved. With this implementation of the SPA, it is essential for branches to be saved in a separated data structure, in order for the mesh functions to slowly render them. A simple structure was created named “Branch”, consisting of a guiding vector array. When a path is completed by the “StepAStarAlgorithm” function, a new branch object is created with its “nodes” array being populated by the guiding vectors constituting the path. The order in which the nodes are placed in each branch’s array is reversed, therefore, starting with the goal node and ending with the endpoint node, the only exception being the trunk path. The first completed path is the trunk, which is required to visually grow in the opposite direction as all other branches; instead of starting from the crown, it should start from the endpoint. Thus, the stored branch’s array order is switched before adding it to the “growingTreeNodes” array.

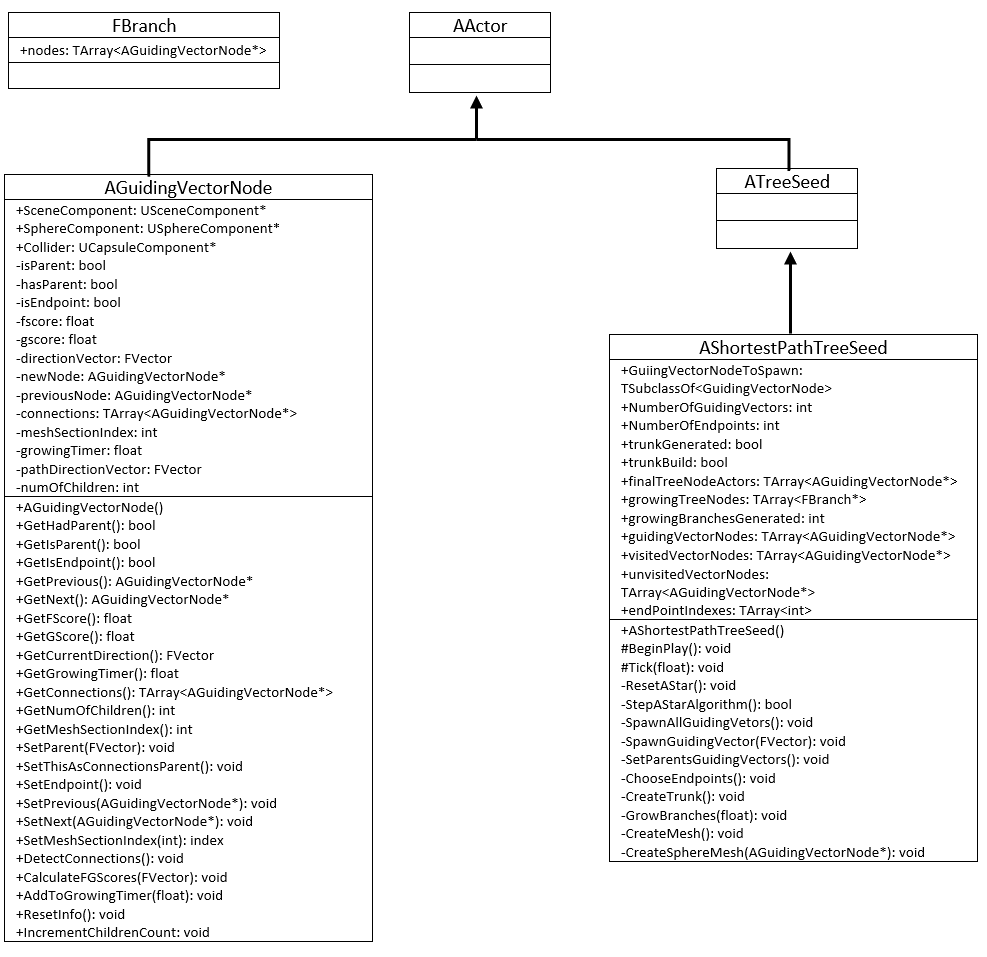
When the SPA is completed the “ResetAStar” function is called, in order to reset all essential arrays and guiding vectors to their original state, allowing the commencing of the SPA’s next iteration.

The final essential element of the simulation is the mesh generation. Hence, with the detection of newly generated branches, the “GrowBranches” function is called, to slowly progress the visual growth of the already developed branches. When all new branches are fully grown, the “CreateMesh” function is executed, updating the visual component and offering a complex mesh, representing the final procedurally generated tree.



Figure 10 - A completed SPA tree without the effect of environmental factors.

**3.3.3 UML Diagram**

****

**3.4 Optimizations**

Although successful in procedural tree generation, both algorithms required certain optimization limitations to reduce their effect on performance and increase the possibility of achieving real-time growth.

Initially, the SCA had an incremental effect on performance as the simulation would progress due to the additive creation of new nodes. The algorithm would iterate through all existing tree nodes and spawn all requested successors immediately, leading to high numbers of new actors each frame. A maximum number of spawned nodes per frame was set for the elimination of the performance bottleneck, encouraging stable new branch creation.

Moreover, the first implementation of the SPA had major performance issues during the creation of new branches since in each frame the algorithm was required to completely path-find between the endpoint and the target node. This was resolved by changing the algorithm’s single frame aim from the completion of the path to the stepping of the pathfinding process.

After the algorithm optimizations were applied, the SPA and SCA were successfully time efficient, however the application’s framerate remained at a lower rate than desired. The mesh construction process was determined as the primary affecting factor; therefore, optimization restrictions were necessary. The two main limitation attributes added to the mesh creation process were the maximum number of growing nodes and updated nodes. Originally, the mesh growing and updating functions would loop through all nodes each frame when rendering the tree’s visual representation. With the implementation of the two attributes, the gradual growing and updating of the mesh was applied, increasing the application’s framerate with a stable refreshment rate.

Lastly, the mesh updating function is only called when new nodes are created, instead of each frame, removing unnecessary calculations and improving performance.

**3.5 Evaluation**

The aim of this project was the adaptation of branching algorithms for the in-game procedural generation of environmentally responsive trees. To therefore, accurately compare each algorithm’s response to the envisioned functionality, both qualitative and quantitative evaluation methods were employed.

Procedurally generated content is usually manually edited by professionals before being added to a game. These algorithms, however, are implemented as to automatically produce content, without any human interference, hence requiring immediate naturalistic and aesthetically pleasing results. Since realism and visual appeal are qualitative variables, an anonymous survey was conducted for the gathering of appeal and realism ratings. This survey consisted of videos, which the participants were asked to rate by answering the following questions:

*“How visually appealing do you find the above tree?”*

*“How realistic do you find the above tree?”*

There are five number rating choices for each of the questions, number one signifying the worst (dissatisfaction) and five signifying the best (satisfaction). Because of the diverse growth conditions and results the algorithms can produce, eight videos of distinctive trees were chosen, grown by both algorithms in the following habitats:

* Negligent environmental conditions, having no effect on the trees. (Figure)
* Strong continuous wind in a specific direction. (Figure)
* Light only reaching the trees from a certain side. (Figure)
* Light only reaching the trees from a certain side and strong continuous wind in a specific direction. (Figure)

With the additional goal of simulating procedural tree generation in real-time, quantitative performance calculations were essential for the fair evaluation of the two algorithms. Therefore, using the UE5’s timing capabilities, each algorithm’s delta time will be recorded, measuring the amount of time spent per frame. Moreover, the scalability of the procedural generation algorithms will also be tested by recording the framerate of the application while generating various amounts of trees simultaneously.

Finally, the two algorithms will be compared by combining the results of both evaluation methods, quantitative and qualitative, aiming to determine the most balanced procedural generation method.

**3.6 Summary**

Although very distinct, common factors were identified between the SCA and SPA simulations, which enabled an object oriented approach while implementing the project. By inheriting from a common base class, consisting of environmental and mesh calculation methods, each algorithm specific class was able to further expand functionality and successfully procedurally generating trees. Through various optimization techniques, the two algorithms were able to simulate tree growth in real-time, confirming the possibility of in game adaptations. Due to the complexity of this projects aim, both qualitative and quantitative evaluation methods were used, to accurately assess each algorithms performance effect and visual appeal, with the goal of identifying the most well-balanced method for procedurally generating environmentally responsive trees.

# Chapter 4 Results

As mentioned in the previous chapter, due to the complexity of this project’s aim, both qualitative and quantitative evaluation methods were required for the accurate comparison of the implemented algorithms. Therefore, using extensive performance calculations and an anonymous survey, all necessary data were acquired.

**4.1 Quantitative Data**

In order to assess the effect on performance, the possibility of in-game adaptability and scalability of each algorithm, time measurements were essential.

The two algorithms were tasked to generate one tree, without any affecting environmental factors. The delta times of the first two thousand frames, consisting of both trunk and crown generation, were then measured and exported to a text file. This method was repeated five times for each algorithm, with repetition and total averages being calculated. The delta seconds were then converted to frames per second for the easier understanding of values. For the clearer perception of data, a line graph was also constructed, since the raw data consisted of twenty thousand numbers (Figure 11).

Figure 11 - Framerate during single tree generation by each algorithm. (Line graph)

**ADD EVOLUTION OF TREES IN INTEVALS OF ABOVE 2000FRAMES**

Figure 11 displays the two algorithms’ performance in the first two thousand frames of the tree generation. As can be seen, the SCA had a higher overall FPS fluctuation, while the SPA had less value variation. The SCA can be identified as the best performing algorithm with a higher range of FPS than the SPA. Moreover, there is a clear negative correlation between the two algorithms’ performance and the simulation’s progression, which is stabilised during the final five hundred measurement frames. A box and whiskers graph can also be plotted to better visualise and compare the general performance of both algorithms (Figure 12).

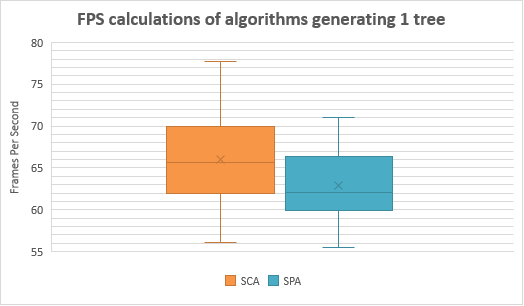


Figure 12 - Framerate during single tree generation by each algorithm. (Box and whiskers graph)

The box and whiskers graph clearly shows that the SCA’s and SPA’s results were in the ranges of 56.26-77.68 FPS and 55.54-71.09 FPS respectively. Furthermore, the SCA had an average of 65.99 FPS and median of 65.62 FPS, which is higher than the SPA’s average, 62.96 FPS, and median, 62.07 FPS.

For evaluating scalability and the possibility of multiple of these trees being adapted to a game’s background environment, the same timing process was repeated for both algorithms while attempting to generate multiple trees simultaneously. As before, for each number of generated trees the simulation was repeated five times, with the first two thousand frames’ delta seconds being stored and exported. Each algorithm’s separate results were plotted on a line graph to observe the difference in performance with the addition of more trees (Figure 13 & Figure 14). The collective results were also plotted on a box and whiskers graph for the simplification of the immense amount of data (Figure 15).

Figure 13 - Framerate during multi-tree generation by the SCA. (Line graph)

Figure 14 - Framerate during multi-tree generation by the SPA. (Line graph)

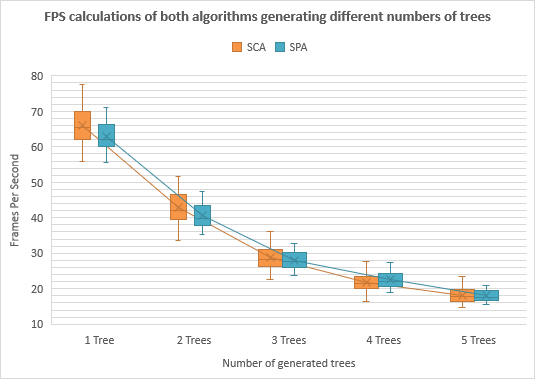


Figure 15 - Framerate during multi-tree generation by both algorithms. (Box and whiskers graph)

In Figure 13 and Figure 14, the negative correlation between the applications’ FPS and the progression of the simulation can still be seen, although fewer fluctuating values were recorded as the number of generated trees increase. Also, from all three graphs, an exponential decay pattern can be seen concerning the FPS, since the values decrease with a decelerating pattern, with a higher effect on performance between one and two trees, than four and five trees.

In comparison, the SPA has mostly higher maximum, mean and median FPS values, in higher ranges than the SCA. Additionally, as each algorithm is tasked to generate more trees, the difference of average and median FPS between the two algorithms declines because of the more abrupt drop in FPS by the SCA. This causes the algorithms to perform almost identically when procedurally generating more than 3 trees.

All time measurements were performed under the same conditions on the same machine. For the purpose of this study a Razer Blade 14 was used with an AMD Ryzen 9 5900HX processor with Radeon Graphics, 16GB of memory and an NVIDIA GeForce RTX 3070 graphics card.

**4.2 Qualitative Data**

Since realism and visual appeal is equally important as the performance section of the aim, a survey was constructed, as described in chapter three, and was distributed using Google Forms.

The survey consisted of eight videos, consisting of the two algorithms growing trees in 4 distinct environments. The participants were initially shown an image of a real life English Oak and were asked to use it as a reference when answering each question. The questions simply asked realism and visual appeal ratings for each tree and were optional, in-case any participant didn’t want to answer.

In total, eighty four participants completed the survey, with their answers automatically plotted to question-specific bar charts by the application. The raw results can be found in **APPENDIX**.

The results were also displayed in two separate tables for the easier depiction of patterns (Table 2 & Table 3).

Table 2 - Visual appeal ratings gathered by the anonymous survey.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| VISUAL APPEAL RATINGS | | | | | | | | | | |
| Affecting Environmental Factors | Space Colonization  Algorithm Ratings | | | | | Shortest Path Algorithm Ratings | | | | |
| 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| None | 6 | 17 | 20 | 24 | 17 | 9 | 16 | 23 | 24 | 12 |
| Wind | 10 | 22 | 22 | 17 | 13 | 12 | 22 | 22 | 15 | 12 |
| Light | 10 | 15 | 20 | 25 | 14 | 7 | 15 | 28 | 20 | 14 |
| Wind and Light | 14 | 14 | 20 | 18 | 18 | 13 | 17 | 20 | 21 | 13 |
| Total | 40 | 68 | 82 | 84 | 62 | 41 | 70 | 93 | 80 | 51 |

Table 3 - Realism ratings gathered by the anonymous survey.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REALISM RATINGS | | | | | | | | | | |
| Affecting Environmental Factors | Space Colonization  Algorithm Ratings | | | | | Shortest Path Algorithm Ratings | | | | |
| 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| None | 5 | 18 | 27 | 14 | 13 | 8 | 16 | 25 | 18 | 10 |
| Wind | 10 | 24 | 18 | 12 | 12 | 9 | 22 | 23 | 14 | 9 |
| Light | 9 | 9 | 23 | 23 | 12 | 6 | 12 | 26 | 21 | 12 |
| Wind and Light | 9 | 14 | 18 | 22 | 14 | 11 | 13 | 26 | 17 | 10 |
| Total | 33 | 65 | 86 | 71 | 51 | 34 | 63 | 100 | 70 | 41 |

Overall, the results for trees made by the SCA showed high satisfaction in, both, realism, and visual appeal. The SPA’s ratings on the other hand are mostly consisted of one, two and three number ratings, therefore signifying mediocre to lower satisfaction, in both visual appeal and realism categories.

Using the above demonstrated results, the qualitative evaluation of the project will be completed, proving which algorithm is visually prevalent.

# Chapter 5 Discussion

**5.1 Introduction**

The aim of this project is to establish an algorithm that is able to successfully procedurally generate, environmentally responsive trees, in real-time, with the possibility of in-game adaptations. To evaluate the implemented algorithms, their comparison, based on qualitative and quantitative measurements, is required.

Due to the substantial amount of raw data, the creation of tables and graphs was essential for the clearer understanding of the results. Using the created data representations, patterns can be identified to determine the effectiveness of each algorithm in response to the aim.

**5.2 Quantitative Evaluation**

The collected quantitative data portray each algorithm’s performance effect when simultaneously simulating various numbers of trees. As real-time generation is essential, a higher or equal framerate to the standardized (60FPS) would be preferred.

As can be seen in Figure 12, when generating a single tree, the two algorithms were able to maintain satisfactory mean and median FPS throughout the first two thousand frames of the simulation. Even though their minimum values are slightly under the optimal, the fact that the lower-quartiles are higher than the goal framerate, with greatly higher maximum and higher-quartile results, provides promising future, in-game adaptation possibilities.

The fluctuation in framerate is seen by both algorithms, however, more significant during the SCA’s procedural generation. The variation in values shown in Figure 12 can be broken down into two factors by considering the Figure 11. From this figure, the general fluctuation in FPS can be seen, which is continuous throughout the simulation, but also the deviation caused by the negative correlation between the framerate and time. Even though slight perpetual fluctuations were expected, since different performance effects apply at various stages of each algorithm, the SPA’s results were more ideal as the variation in values is shorter and more predictable. The decline in framerate as each simulation progressed was also anticipated due to the incrementing number of branches and calculations executed by each algorithm. Although having a much higher initial framerate, the SCA’s values had a much sharper drop than the SPA’s, therefore resulting in analogous performances during the final frames. This shows that simultaneous calculations of multiple tree node positions using the SCA is much more time consuming than the incremental concurrent calculations completed by the SPA. During the final frame measurements, the declines’ stabilization signifies the two algorithms reaching their optimization limits, therefore, preserving steady framerates for the rest of the generation. The stable ending FPS recorded for both algorithms consist of values adjacent to the 60FPS aim. Considering that both algorithm’s optimization limits were not set to their optimal values and that the testing was concluded on a laptop computer, the two algorithms show great success in their ability of procedurally generating a single tree in real-time without majorly affecting the application’s performance.

Since single tree simulations performed efficiently in real-time, the scalability of each project was also evaluated. Although not stated in the aim, each algorithm’s ability of procedurally generating multiple simultaneous trees can have an effect on whether a developer would choose to implement it in a game. Therefore, as to further evaluate each algorithm’s performance, the same time measurements were carried out while the algorithms were tasked to simulate multi-tree growth.

Even though distinct characteristics could again be recognised in Figure 13 an Figure 14 with better peak performances by the SCA but more stable framerate by the SPA, certain similarities on the effect of multiple tree generation on each algorithm’s performance became apparent. Firstly, there was an expected, yet significant, decline in framerate by each additional tree. The decrease followed an exponential decay pattern, which can be seen clearly in the comparison box and whiskers graph (Figure 15). Although both algorithms face this decline, the SCA had a higher drop rate than the SPA, rendering the two algorithms equally performing while generating more than three trees.

Moreover, the general fluctuation in framerate, throughout both algorithms’ simulations, is gradually reduced as the number of trees is incremented. This is caused by the application executing multiple methods concurrently, therefore stacking together each procedural generation’s fluctuating performance to create a steadier total performance. The stabilization in performances can also be seen from Figure 15, by the decreasing range between each algorithm’s maximum and minimum values as more trees are added to the application.

A minor performance effect can also be seen at the starting frames of the algorithms’ three, four and five tree simulations. This is due to the concatenation of the set-up functions of all trees, which becomes more noticeable during multi-tree generation.

As are currently implemented, the two algorithms can’t be used for the concurrent generation of multiple trees without greatly affecting the applications performance. On the other hand, since scalability was not part of the aim, the algorithms were constructed for generating single trees, thus when tasked with multi-tree generation, multiple instances of the algorithms are running at the same time. As the algorithms are able to procedurally generate a tree in real-time, there are optimizations and techniques that can allow scalable tree growth using only a single instance of the algorithms, which will cause overall slower results but better performance.

Overall, both algorithms had negligent effect on performance when generating a single tree, therefore achieving the real-time aspect of the aim. With a much higher fluctuation in framerate, the SCA performed generally better, always having higher mean and maximum FPS values. However comparatively slower, the SPA had a much more stable performance through the simulation, with a less steep framerate drop with the incrementation of simultaneously generated trees.

**5.3 Qualitative Evaluation**

As described in chapter four, qualitative values were collected using an anonymous survey, in order to evaluate the visual appeal and realism of the procedurally generated trees. The survey was shared and collected 84 separate responses which provided enough visual ratings for both algorithms’ procedural generations in various environmental conditions.

The participants’ preference was made clear, with the SCA algorithm receiving the most number four and five ratings, signifying high and extremely high satisfaction, on both realism and visual appeal categories. On the other hand, the SPA received the most number three and one ratings, signifying mediocre or no satisfaction.

Although the SCA is depicted as the general visually superior algorithm, the ratings can be examined in depth to locate the preferred method of generation for each habitat. The first trees, requiring participant ratings were grown by the two algorithms without the effect of any external environmental factors. The results show that the SCA managed to score a much higher number of successful visual appeal rating, however, the SPA closely overtook the algorithm in positive realism ratings. Similarly, when asked to rate the trees generated in an environment with constant powerful winds in a specific direction, the participants offered higher visual appeal ratings to the SCA and more positive realism ratings to the SPA. This shows a distinction by the participants between realism and visual appeal, meaning that a realistic-looking tree is not necessarily a visually appealing tree.

The four remaining trees, including two trees grown in an environment with stationary light and two trees grown in an environment with a combination of constant winds and stationary light, received unambiguously positive results for the SCA for both, visual appeal and realism categories. The SPA collected more mediocre to negative satisfaction ratings for those trees, which also swayed the total score and depicted the SCA as the visually preferred algorithm.

Individual algorithm environmental comparisons could also be useful in identifying each algorithm’s optimal environmental conditions, as decided by the participants. The most highly visual appeal rated tree created using the SCA is clearly the one grown without the consideration of any environmental factors, with forty one positive ratings. Similarly, the SPA’s most visually appealing tree, as elected by the participants with thirty six positive ratings, is the one grown without the effects of the surrounding environment. On the other hand, the least visually appealing trees grown by both algorithms were in the windy environment, with substantial amounts of negative ratings. The participants’ disapproval of trees grown in the windy environment is also clear by the realism ratings, showing an unambiguous number of negative results.

The separation of visual appeal and realism is further proved by the difference of most visually appealing trees and most realistic trees as chosen by the participants. The SPA’s highest rated in realism tree is the one grown in an environment with constant wind and stationary lighting while the SCA’s is the tree grown in an environment which only includes a stationary light.

Generally, the SCA is the visually superior algorithm, determined by the summing of all votes. However, the SPA could be more ideal in cases where realistic results are required.

The qualitative evaluation was completed for the visual assessment of each algorithm’s generated trees. With the addition of all ratings, the results of five hundred and ten positive and four hundred and fourteen negative show the successful response by both algorithms to the visually appealing and realistic aspect of the aim.

**5.5 Conclusion**

Taking into consideration both quantitative and qualitative evaluations, both algorithms successfully responded to the aim of this project with overwhelmingly positive results. Both algorithms were able to create a single tree in real-time with average framerates higher than the 60FPS industry standard. The incrementation of trees per simulation caused both algorithms to perform poorly, since they were not created with scalability into consideration, however the successful generation of one tree depicts promising adaptations for simultaneous multi-tree generations. Generally, sustaining higher framerates than the SPA, the SCA provided more promising, yet fluctuating, performance results. On the other hand, the SPA, although having slightly lower performance, was more stable and exhibited more favourable scalability possibility due its more gradual drop in framerate.

Both algorithms also produced exceptional visual results, with the survey participants rating the SCA’s trees as the most visually appealing and realistic. However, the SPA surpassed the SCA in realism ratings concerning trees grown in certain habitats. The two algorithms also receive much higher numbers of positive ratings, therefore showing high participant satisfaction.

This project successfully implemented two algorithms for the procedural generation of realistic, visually appealing, and environmentally responsive trees, in real time. Both algorithms were successfully efficient, producing high quality results which can be added in game environments to create evolving and immersive worlds. Although a specific algorithm cannot be deemed as superior, each algorithm has its own thriving conditions and use cases.

# Chapter 6 Conclusion

As the games industry is continuously applying new technologies to improve realism and add details to products, consumers are becoming more demanding. Evolving environments have now become an expected standard, therefore, it is important to keep inventing new ways of creating lively scenes for the players to experience. This project aimed to construct an in-game tree growing model, applying the everchanging effects of procedural generation to a frequently used game environment component.

By expanding on the principles of content-drafting, two branching structure algorithms were chosen and adapted for the real-time growth of trees. The algorithms were also tasked to adapt the generation based on the surrounding environmental factors in an attempt to produce realistic and visually appealing results. The SCA and SPA successfully managed the real-time procedural generation of single trees with an average application framerate higher than the industry standard. The survey also showcased the general satisfaction of the trees’ visual representation, scoring high appeal and realism ratings for both algorithms. Henceforth, achieving similar simulation framerates while creating distinct visual results, an ideal algorithm can be chosen depending on the stylistic preference of the environment.

**6.1 Implications**

By achieving the real-time generation of environmentally responsive trees, which are both visually appealing and realistic, the methods used in this project can aid environmental designers while also providing audiences with more immersive game experience. This project provides a more time efficient approach than the current industry-standard, content drafting of trees, by offering real-time procedural generation methods. The two algorithms’ results are not only visually appealing and realistic but also environmentally responsive, therefore removing the need for human interaction.

**6.2 Future Work**

Even though successful results were produced by the implement methods, this project can be further expanded to procedurally generate improved tree models.

**6.2.1 Environmental Response**

Currently, the algorithms detect their surrounding environmental factors and adapt the corresponding growing attributes at the beginning of the application. A steady environment is assumed; hence, the environmental adaptation of attributes is never repeated. This acts as a limitation for the implemented algorithms since any change in the game environment wouldn’t be reflected on the trees’ shape or rate of growth.

The addition of a periodical environmental detection system could resolve this issue and reinforce the algorithms responsiveness to the game’s environmental influences.

**6.2.2 Visual Representation**

Although gaining high satisfaction ratings by the survey participants on both visual appeal and realism categories, the generated trees’ visual representation can be greatly improved.

Firstly, a higher resolution tree texture could be applied to the tree, instead of the current single-colour material. Each branch’s colour could also be determined by its hight, allowing the possibility of a darker or greener lower trunk section.

Additionally, with the employment of shaders, leaves or flowers could be added to branches, which would create increasingly visually appealing results. The shaders could also be adapted by the environment to generate realistic results for the surrounding habitat (e.g., lower population of leaves appearing in windy environments).

**6.2.3 Scalability**

Since single tree procedural generation could be achieved by the two algorithms, there is a possibility for achieving scalable results with little adaptation. As the current optimization causes the gradual generation of trees, the same technique could be applied during multi-tree generations.

The two algorithms could be re-implemented as manager classes, holding all functionality and pointers to the simultaneously growing trees. The progressive iteration of all trees could be performed by the classes, with one tree’s procedural generation step performed each frame. This model could provide infinite scalability, however, increasingly slow growth as the number of trees is incremented.

**6.2.4 Conclusion**

To conclude, this project successfully implemented two algorithms for the procedural generation of environmentally responsive, visually appealing and realistic trees, in real time. Offering room for improvement, this project stands as a template for the future of real-time procedural environmental design.

# List of References

Epic Games, 2022. *UProceduralMeshComponent.* [Online]   
Available at: https://docs.unrealengine.com/5.0/en-US/API/Plugins/ProceduralMeshComponent/UProceduralMeshComponent/  
[Accessed 25 November 2022].

Fowells, H. A. & Means, J. E., 1990. The Tree and its Environment. In: *Silvics of North America.* Washington DC: Department of Agriculture, Forest Service, pp. 1-11.

Fu, Y. et al., 2023. *Skeleton extraction and pruning point identification of jujube tree for dormant pruning using space colonization algorithm.* s.l.:s.n.

Giacomello, E., Pier, L. L. & Daniele, L., 2018. *Doom level generation using generative adversarial networks..* 2018 IEEE Games, Entertainment, Media Conference (GEM) (pp. 316-323), IEEE.

Gilman, E. F. & Watson, D. G., 1994. *Quercus Robur, English Oak.* s.l.:University of Florida.

Hamed, T., 2016. How to Design and Create an Effective. *International Journal of Academic Research in Management (IJARM),* Volume 5, pp. 37-41.

Kendrik, E. R. & Kronenberg, M. H. G., 1994. *Photomorphogenesis in plants.* 2nd ed. s.l.:Kluer Academics.

Mojang Studios, 2011. *Minecraft.* Stockholm: Mojang Studios.

National Weather Service, 2007. *Estimating Wind Speed.* [Online]   
Available at: https://www.weather.gov/pqr/wind  
[Accessed 6 May 2023].

Prusinkiewics, P., 1986. Graphical Applications of L-Systems. *Proceedings of Graphics Interface*, pp. 247-253.

Runions, A., Lane, B. & Prusinkiewics, P., 2007. *Modeling Trees with a Space Colonization Algorithm.* Prague, Proceedings of the Eurographics Workshop on Natural Phenomena.

Runions, A., Lane, B. & Prusinkiewicz, P., 2005. Modeling and visualization of leaf venation patterns. *ACM SIGGRAPH 2005 Papers,* pp. 702-711.

Short, T. X. & Adams, T., 2017. *Procedural Generation in Game Design.* s.l.:Tayler & Francis Group.

Stepney, S. & Verlan, S., 2018. *Unconventional Computation and Natural Computation.* France, Springer.

The Royal Horticultural Society, 2022. *Quercus robur.* [Online]   
Available at: https://www.rhs.org.uk/plants/14294/quercus-robur-common-oak-aac/details  
[Accessed 25 November 2022].

Wayadhi, R. M., Ginting, S. H. N. & Syahputra, D., 2021. Greedy, A-Star and Dijkstra's Algorithms in Finding the Shortest Path. *International Journal of Advances in Data and Information Systems,* 2(1), pp. 45-52.

Xu, L. & Mould, D., 2012. A procedural method for irregular tree models. *Computer & Graphics,* Volume 36, pp. 1036-1047.

Xu, L. & Mould, D., 2015. Procedural Tree Modeling with Guiding Vectors. *Computer Graphics Forum,* 34(7), pp. 47-56.

# Appendices