Revista Facultad de Ingeniería, Universidad de Antioquia, No.99, pp. 9-20, Apr-Jun 2021

Semi-automatic construction of video game design prototypes with MaruGen

Construcción semi-automática de prototipos de diseños de videojuegos con MaruGen

Italo Felipe Capasso-Ballesteros [1] **Fernando De la Rosa-Rosero** [1*]

[1]Grupo Imagine: Computación Visual, Departamento de Ingeniería de Sistemas y Computación, Facultad de Ingeniería, Universidad de los Andes. Carrera 1 # 18A - 12. C. P. 111711. Bogotá, Colombia.

CITE THIS ARTICLE AS:

I. F. Capasso and F. De la Rosa. "Semi-automatic construction of video game design prototypes with MaruGen", *Revista Facultad de Ingeniería Universidad de Antioquia*, no. 99, pp. 9-20, Apr-Jun 2021. [Online]. Available: https://www.doi.org/10.17533/udea.redin.20200369

ARTICLE INFO:

Received: October 28, 2019
Accepted: March 27, 2020
Available online: March 27, 2020

KEYWORDS:

Video game design; video games; automatic game design; game prototyping; procedural content generation

Diseño de videojuegos; videojuegos; diseño automático de juegos; prototipado de videojuegos; generación procedural de contenido

ABSTRACT: *Machinations Ruleset Generator* (MaruGen) is a semi-automatic system for the generation of mechanics, rules, spaces (environments), and missions for video games. The objective of this system is to offer an expression mechanism for the video game designer role based on the definition of rules, and the ability to explore the concepts of progression and emergence in video games by using a formal, usable, and defined tool to design games with innovative and complex elements, and behaviors defined from combinations of basic elements. Based on the expressed designs and with the participation of programmers and video game artists, MaruGen allows the generation of agile video game prototypes in the Unity game engine. These prototypes can be analyzed by the entire workgroup to look for games with diverse complexities that make them attractive to their users. MaruGen is based on the expression of rules on elements of interest in video games and the rewriting mechanism using L-Systems for the generation of procedural content. MaruGen was evaluated in the construction of the Cubic Explorer video game and tested by gamers and video game developers during the *Game Jam* Ludum Dare 38.

RESUMEN: *Machinations Ruleset Generator* (MaruGen) es un sistema semi-automático para la generación de mecánicas, reglas, espacios (ambientes) y misiones para videojuegos. El objetivo de este sistema es ofrecer un mecanismo de expresión al rol de diseñador de videojuegos, a partir de la definición de reglas y la capacidad de explorar los conceptos de progresión y sorpresa en los videojuegos usando una herramienta formal, usable y definida para diseñar juegos con elementos y comportamientos novedosos y complejos definidos a partir de combinaciones de elementos básicos. Basado en los diseños expresados y con la participación de los roles de programador y artistas de videojuegos, MaruGen permite generar prototipos ágiles de videojuegos en el motor de juegos Unity. Estos prototipos pueden ser analizados por el grupo completo de trabajo para buscar juegos con complejidades variadas que los hagan atractivos a sus usuarios. MaruGen se fundamenta en la expresión de reglas sobre elementos de interés en los videojuegos y el mecanismo de reescritura usando Sistemas-L para la generación de contenido procedural. MaruGen fue evaluado en la construcción del videojuego Cubic Explorer y probado por usuarios jugadores y usuarios desarrolladores de videojuegos durante el *Game Jam* Ludum Dare 38.

1. Introduction

The design and development of video games is, by its nature, a multidisciplinary and complex process, whose creation requires different talents and processes.

Developing a complete video game requires extensive communication between professionals from different areas and fields of knowledge, all necessary to achieve a successful final product.

In general terms, it can be considered that the main components of a video game are based on the elements of design, programming, and the generation of art and content. Each stage has its own specific processes, standards, and codes for the proper management of its

* Corresponding author: Fernando De la Rosa Rosero
E-mail: fde@uniandes.edu.co
ISSN 0120-6230
e-ISSN 2422-2844

field. Just as there are defined processes and rules, automation and the procedural generation of content have also been applied, which refer to automatic computational methods that create content either up front or on-the-fly; for example, the creation of mechanics, spaces, and characters in video games. This automatic generation of content is something that can already be considered ubiquitous in the video game development process, especially in the programming and content creation roles for a video game.

The main stage of the design of a video game is understood as the creation of mechanics, rules, missions, levels, and objectives to be fulfilled. This stage is one of the least developed ones, despite its great importance in the general conception of the video game. This conception, as well as its subsequent implementation and execution, permeates the whole process; thus, it is crucial that this process be well executed, for the successful completion of a video game project.

This work, Machinations Ruleset Generator (MaruGen) presents a semi-automatic tool for generating video game mechanics, rules and spaces (environments): Mechanics refers to the modes, possibilities, and restrictions that the player encounters while s/he is playing and which s/he can interact with to alter the state of the game. The rules are the elements and procedures that produce every possibility and constraint a game has. Spaces (or "environments" in [1]) refers to the worlds that the player should explore while s/he is playing. All of these concepts are defined in [1]. MaruGen uses the definition of generative grammar, based on the conceptual proposals of game designers, in an effort to formalize the field based on the concepts of mechanics, rules, and space design. This proposal's basis is found on systems based on different computing areas, such as the generation of grammars and rewriting of terms (in particular the use of L-Systems), and the procedural generation of mechanics, rules and spaces of video games.

2. Related work

In broad terms, video game design is understood as the process of creating the spaces (environments), the rules, and the content that make video games work. However, this description does not show how really broad and complicated the process of designing a video game can be. There are many classifications of the concept of game design; this is, mostly, because there are very precise definitions of what a video game is, with specific categories and elements. The author in [2], one of the early pioneers in game design, defines video games based on making clear distinctions between other types of activities and media, such as interactive games, puzzles, and competitions. Another more formal definition is given in [3], in which the games are separated into formal elements, dramatic elements, and dynamic systems, which make up a whole in the game.

However, there are more "fluid" and much less rigorous definitions; such as the one presented in [4], where ultimately defines games as means to create experiences; or the definition in [5] that defines it as an art form, with a series of rules and players pursuing a goal.

There are definitions, with both rigorous need and creative freedom, such as the MDA (Mechanical, Dynamic and Aesthetic) framework [6]. This framework establishes the proposal of rules and content, called mechanics that generate a series of dynamics; i.e., the interaction of these rules with the players when putting the system in motion, to finally fulfill an aesthetic about the player; in other words, the sensations and experiences of the player. Or the concept of the theory of fun [7], which specifies that all games are, in essence, learning tools. Thus, a game's fun is derived from learning, and boredom arrives when the game is mastered.

All these definitions seek, in a certain way, even if it is unintentional, to state a series of principles or tools that provide people who wish to study or design video games, a starting point for their operation. Something that, as an analogy, could be the equivalent of a music notation system (i.e., the scores) to create video games. However, all definitions tend to converge on some common themes. The first one is that each definition contains both a concrete, systemic, and tangible component (rules, systems, elements), and an abstract component, generally attributed to creativity and human genius (aesthetics, art, experience, theme). This is highlighted in [8], as a "formal and abstract" duality and its author called for creating formal and abstract tools in the design of video games. Additionally, this author managed to articulate the proposal, which unleashed many of the current video game creation schools in the 21^{st} century.

The author in [9] is one of the pioneers when it comes to formalizing the theory of video game design and looking for a precise and unambiguous grammar, which allows automating, formalizing and specifying, through machines, a possible language of the game, using concepts that he called "practical creativity", "game atoms", and, more recently, "ludemes", which results in creating a topology of the mechanics (rules and components) of a video game [10]. Something similar, although somewhat informal, has been defined in [11]; something the author calls "atoms," "cycles," and "arcs," which are similar to the concepts of feedback loops of the MDA framework [6].

Another endeavor to replicate the idea of formalizing video games originates from the concepts of "verbs" and "objects"; an equivalent of actions, mechanics, rules, and procedures within a game system proposed in [12] in an effort to achieve a grammar in games or alternative projects. These emerged and then lost momentum, such as "The 400 project" [13], a proposal to create 400 unique rules, related to video game design, or the book "Patterns in Game Design" [14], an attempt to reach the same concept of object-oriented programming patterns, applied to video games.

An interesting effort comes from the author in [15], who states that one of the fundamental components of game creation is the concept of the emergent game, that is, the interrelation that arises in the interaction between several individual components that are part of a system, evoked by the old saying "the whole is more than the sum of its parts". Another interesting concept is that of progressive game, known as the dynamic ability of a game to adapt progressively to the ability of the player. Starting from the concept that the properties of the emergence and progression, elements considered natural of good games, arise from a series of creative combinations of simple elements in games. And that these individual elements can be formalized and automated in game design.

The latter winds up in a conclusion made in [16], in which game design can start from a "recipe" or a "prescription" of the basic operation of a game, and from there it is a matter of "mixing and varying" the elements, based on simple and clear rules. The idea of creating a video game in this way is quite controversial, given the nature of games, being a product of an iterative effort that is a mixture of art and science, more alchemy than anything else [17]. However, this proposal seeks to eliminate this and solve the problem of iteration and progression with automation and a strict application of basic rules.

The author in [15] materializes this in a series of grammars and formal rules to design mechanics, rules, spaces, and missions for games called Machinations framework and Mission-Space framework, which have been formalized as concrete tools: Machinations Tool for Machinations and Ludoscope for Mission-Space. The Unexplored game [18], created with these tools as a base, is a concrete example.

With the combination of the system proposed in [15], the concept of "ludemes" proposed in [9], and the patterns in game design [14], it is possible to generate a "mathematics of game development", as defined in [10]. Thus, like the author in [9] mentions, if something is essentially mathematical, it can be formalized.

On the practical side, the procedural generation of content and rules groups existing techniques that support the development and design of video games in several aspects. First, the generation of visual, sound, and artistic elements within the video game (that is, sounds, music, animations, models, textures, etc.); and, secondly, the generation of spaces and levels of a video game. Different games show that the procedural generation has mostly focused on generating "superficial" elements of video games. The basic rules and components are usually already determined, and they trust that the combination of predefined elements and rules in a changing environment, and with changing content, will generate dynamics interesting enough for the player to play for a long time.

Additionally, to manage systems with emergent properties, it has been practical to use L-Systems, proposed in [19], where an L-System is a rewriting system based on a formal grammar defined by a set of symbols, an initial state of the system, and a set of rules that automatically generate new states of the system. In their origin, L-Systems offered a formal explanation to the behavior of certain multicellular organisms, such as algae, fungi, and plants. Moreover, its successful use has been seen when generating elements, models, and procedural art. For example, the Houdini Engine tool [20] uses an advanced L-System to generate all kinds of structures, such as bridges, buildings, and roads. On the other hand, the SpeedTree program [21] uses L-Systems to create all trees and plants. Many advances have also been made in the creation of organic procedural 3D models using L-Systems, something particular in systems such as LParser [22].

All of this shows the efforts and interest in formalizing and automating the design stage of video games, mainly in relation to their mechanics, rules, spaces (environments), and missions.

3. MaruGen – Semi-automatic prototyping proposal for video game designs

The automatic generation of the design of a video game has many different approaches and techniques. This problem encompasses many fields of knowledge: Artistic creation, (conceptual) design, procedural content, mechanics and rules, grammar, and rewriting. However, not all issues have been addressed in a general manner, and not all possible alternatives for generating content for video games have been explored. In particular, video game design is one of the fields where these ideas have been least explored. The call in [8] for the creation of formal and abstract tools for game design becomes important. Based on simple rules and patterns defined by the video game

designer, it is desirable for him/her to be able to create complex and emergent elements, an idea supported by the proposal of operation and basic arrangement of elements exposed in a previously defined grammar [15].

Our idea is to resort to emergent video game theory proposed in [15], in which the emergent property of games is caused by simple rules. This philosophy, combined with re-writing rules and L-Systems, can provide a generation of random, unexpected, and interesting mechanics, rules and spaces which, given the definition of an emergent game, and supported on a formal definition, never results in errors. The property of an emergent game is important in order to conceive novel games based on unexpected combinations of basic components.

Our proposal is MaruGen, a computational tool capable of assisting in the process of design and creation of rules, objectives, missions, and spaces of a video game. This semi-automatic tool, which is designed to be assisted by a game designer and other professionals, such as programmers and artists, can contribute to the generation of new and innovative video game mechanics and rules, based on the assumptions of the video game designer and defined as rules within the system. The emergent and progressive properties in video games are supported by the L-Systems that use the basic rules defined by the game designer.

4. MaruGen in the video game development process

The most common process of video game development in the industry is the iterative design; this is particularly common in its prototyping stage. Multiple design proposals are based on the development of software and the development of games as systems [6]. This development process is defined by cycles of the design, implementation, testing, and evaluation stages.

In this way, MaruGen is a tool that adapts to the iterative work of video game development and supports the stages of video game design and implementation Figure (1). MaruGen is intended for use by the video game designer, the one who defines the spaces, mechanics, rules, missions, and objectives of a given game. On the one hand, MaruGen is a visual and informative tool that allows the expression of the designer's abstract ideas in design elements and diagrams. On the other hand, MaruGen is a formal tool, capable of transforming its abstract diagrams into concrete representations, implemented by a programmer and completed with content by artists.

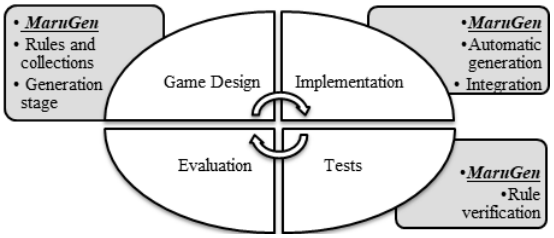

Figure 1 Representation of the iterative process of a video game and the role MaruGen plays in each stage

4.1 Videgame development support

More precisely, MaruGen supports the designer and other users in the process of producing a video game (Figure 2). In the first stage, MaruGen allows the designer to create the minimum rules and elements necessary to propose the basic design of a game. In the second stage, the programmers extend MaruGen's elements to generate the spaces and represent the defined mechanics of the MaruGen system. Since the diagrams generated by MaruGen are abstract, this implementation must be adjusted for the game that is being created in coordination with the game designers. The artists, meanwhile, are generating the contents of the game, 2D/3D models, audio, textures, etc. In the last stage, MaruGen generates spaces, game mechanics and rules in real time and integrates all the elements defined above. At this stage, the results of all the work are evaluated and, if necessary, the cycle is repeated in accordance with the concept of iterative development.

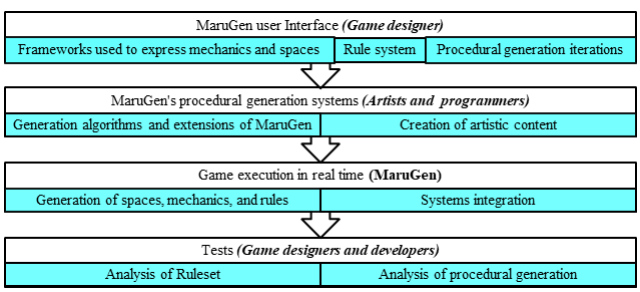

Figure 2 Video game development stages in MaruGen

4.2 Functioning

In the first stage of the development of a video game, MaruGen allows for the creation of mechanics and video game spaces. The Machinations framework was used as a basis to represent mechanics and behaviors in a video game, and the Mission-Space framework for the generation of spaces, game levels, and missions [15]. Both frameworks define a representation structure with known and documented nodes and arcs. These frameworks were used for their power of expressiveness, their

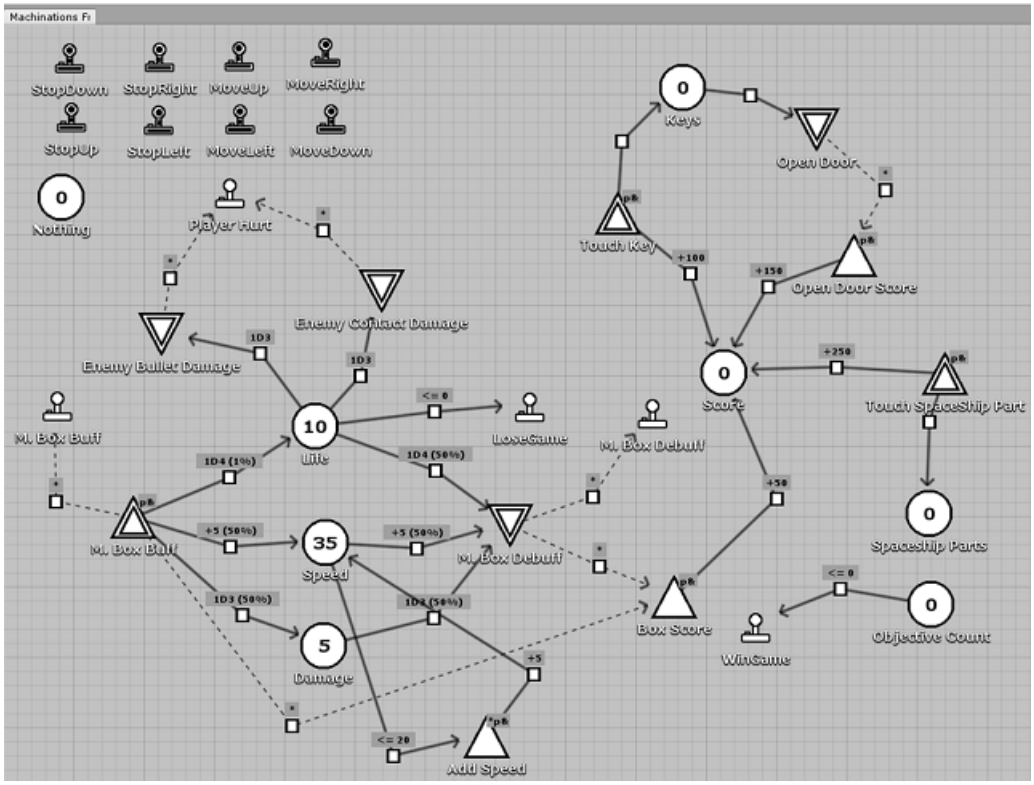

Figure 3 Graph constructed with machinations expressing the mechanics of a video game

mathematical foundation, their algorithmic development, and their validation in the development of several video games.

Nodes and arcs were reused from Machinations. The nodes define the properties of game elements or items; they can be of different types: Pool and its variant External Pool, Source, and Drain. Additionally, the External Event node was proposed in order to have the functionality of calling/executing an external method. All nodes have the property of being interactive, which allows any node to be activated by an event in the game engine. The arcs allow the connection of nodes with different semantics: flow, condition, and trigger.

From Mission-Space, the Place nodes that can contain mission elements and Lock nodes were reused. The arcs type Path and Valve were also used.

The designer expresses the mechanics and spaces of a video game from the creation of nodes and arcs, whose result is a graph in the respective framework (Figure 3).

In the second stage, this visual representation of the graph must be transformed into a concrete representation of a video game. MaruGen offers the programmer a series of classes that can be extended to implement the spaces and/or mechanics through code. The programmer user can specifically extend methods of creating nodes and/or arcs and methods that are executed before or after executing the graph reading process. The programmer can use procedural generation mechanisms to support the emergent property of the video game. The mechanism used in MaruGen is a term rewriting system based on L-Systems. This system requires an initial graph and uses the rules associated with levels, spaces, and mechanics. Also, in this system the type of replacements, the number of iterations, the order of appearance of each rule, and the different generation stages can be specified. The result corresponds to different graphs based on the same rules (Figure 4).

Depending on the game, the implementation is specific to its needs. With the joint work of a programmer, MaruGen allows extending a series of classes to generate the desired spaces and behaviors, based on the information in the graphs. Users who generate content (artists, musicians, creatives, etc.) also participate in this stage to associate these contents according to the elements that are modified in the different nodes and arcs of the graphs.

In the last stage, the Unity3D engine starts the video game, and the built graphs are executed. The execution of these graphs applying a procedural algorithm can generate variants of the same game, exploiting the emergent capacity of the video game. For example, for the same

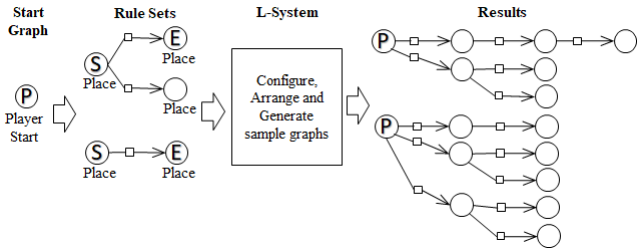

Figure 4 Application of L-Systems in the procedural generation of spaces and mechanics in MaruGen

Unity Editor	Base Graph Editor	Machinations Windows, Nodes, Arcs
		Mission-Space Windows, Nodes, Arcs
Base Graph (ScriptableObject)	Machinations	Nodes and Arcs
	Mission-Space	Nodes and Arcs
RuleSets (ScriptableObject)	BaseGraph L-Systems	ScriptableObject List (Base Graph)
Behaviours (Monobehaviour)	Mission Space Generator	ScriptableObject (RuleSet)
	External Node Event	ScriptableObject (BaseGraph, RuleSet)

Figure 5 MaruGen architecture integrated in the Unity3D game engine

video game, different spaces can be generated from different implementations of the same design graph. This depends on the type of game to be developed, and the generation algorithm implemented.

4.3 MaruGen architecture

MaruGen's architecture is defined under the Unity3D game engine (Figure 5), widely used by industry nowadays. MaruGen's interface uses the Unity editor extension system. On this system, the definition and manipulation of graphs and elements of the Machinations and Mission-Space frameworks are established. The graph system has a common base so that the system interface windows do not recognize the difference of each graph, except for the particular elements of each node and of each arc. Each specific framework, both Machinations and Mission-Space, extends on that common basis, and each specific node or arc has an analog in the interface system so that it is possible to manipulate each element of the system in a simple way, as well as to create new elements if necessary.

The graph system is supported under a native Unity system, called ScriptableObjects. ScriptableObjects are data containers that can be serialized by Unity. However, in order to allow working with other different tools and also to reduce dependency with Unity, these containers are exported in JSON format. These resources can be created within a Unity project and can later be edited.

There is a complementary component: a rewriting system, based on L-Systems. This system takes the ScriptableObjects that represent the different graphs created by the game designer and executes the respective algorithm to generate a new graph of the same type, such as another ScriptableObject within Unity. These procedural generation systems are also defined as ScriptableObjects within Unity.

All the above elements are designed to be executed in the game editor. The behaviors, however, are designed as scripts to be executed in real time, in the Unity game engine. A video game programmer can use these scripts to extend and implement procedural content generation, according to the description of the rules and the graphs presented. This is specific to each game, so it must necessarily be extended and completed by the programmer, in conjunction with the content generated by the artists and in coordination with the entire video game team.

5. Validation and results

In order to evaluate MaruGen, the game Cubic Explorer was designed and implemented using the Unity game engine. This game was featured in the Game Jam Ludum Dare 38 [23]. Cubic Explorer is a game where the player explores multiple cubic worlds in which s/he can find elements of different types (Figure 6): parts of a spaceship, enemies (soldiers or monsters), items (mysterious boxes), keys, padlocks, and doors.

The player has different weapons to defend himself and kill his/her enemies. The doors allow the player to change the cubic world, where s/he will find different configurations of elements. The keys allow the player to open doors that have a lock. The gathering of the spaceship parts and a kind of good boxes allows the player to earn points. However, successful enemy attacks or gathering other types of bad boxes cause the player to lose points.

The mechanisms for generating spaces, missions, and mechanics offered by MaruGen allow defining a combination of different possible objectives of the game (to obtain a minimum number of points, to destroy enemies and/or to collect all parts of the spaceship), with different behaviors for the elements of the game, and under different game conditions (playing time, increase/decrease of the player's health, and/or increase/decrease of the player's speed). All this favors the emergent feature of the Cubic Explorer game.

For the definition of the game space (cubic worlds and their elements), MaruGen allows the game designer to define some basic rules using the Mission-Space framework.

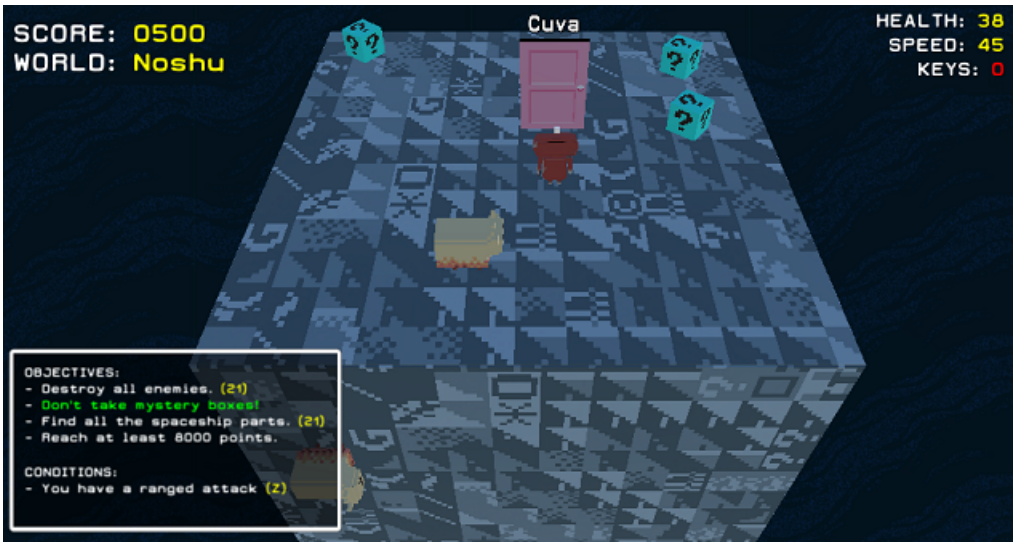

Figure 6 Noshu cubic world where the player is in the center and in front of the door that takes him/her to the Cuva cubic world. There are three mysterious boxes (good and/or bad) and a monster enemy. In the lower-left part, the objectives that the player must achieve are defined

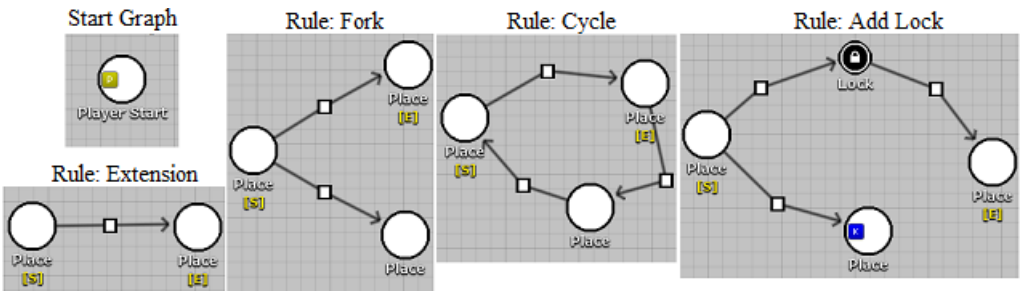

Figure 7 Individual Mission-Space rules (level structure)

Two sets of rules are defined using a grammar: rules about the connection of cubic worlds, and their access restriction by the use of keys and locks (Figure 7), and other rules related to the elements that appear in the cubic worlds (parts of the spaceship, the enemies, and the mysterious boxes) (Figure 8).

Each basic rule has associated elements and behaviors that the programmer user can modify inside the Unity game engine and, thus, be able to use it when generating the graph that defines the entire space of the video game and its elements (Figure 9).

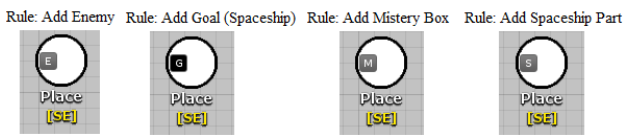

Figure 8 Individual Mission-Space rules (Enemies, parts of the spaceship and mysterious boxes)

Afterwords, MaruGen allows the designer to define

```
public class MissionSpaceSimpleGenerator : MissionSpaceBehaviour
{
    protected override void DoBeforeGraphGeneration(MissionSpaceGraph graph)
    {
        Before generating the graph
    }
    protected override void GenerateNode(Node node)
    {
        Generate a node
    }
    protected override void GeneratePath(Arc arc)
    {
        Generate a path
    }
    protected override void DoAfterGraphGeneration(MissionSpaceGraph graph)
    {
        After generating the graph
    }
}
```

Figure 9 Skeleton that allows extending the Mission-Space graphs as a tangible space in Unity in real time

an L-System that integrates the basic rules about the definition of cubic worlds and their elements in the game. Additionally, it includes the operating parameters of the L-System that define how to mix and apply the different rules (Figure 10). The result of the execution of this system is the generation of the space structure

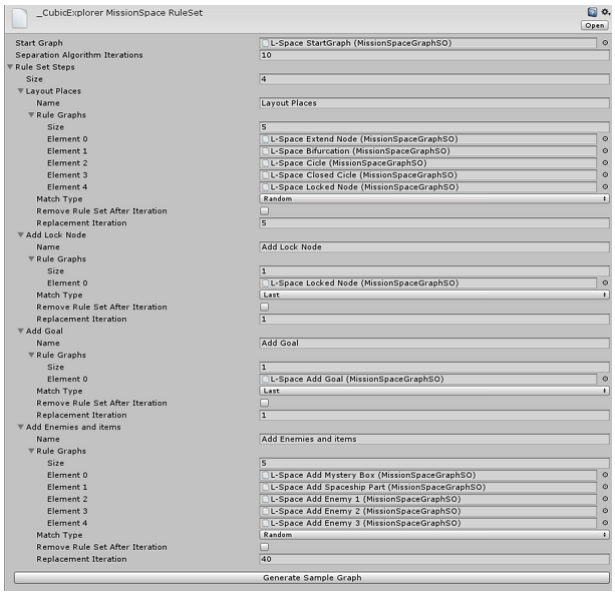

Figure 10 L-System that integrates different space-definition rules for the Cubic Explorer game

for the video game that complies with the basic rules of the designer (Figure 11). This result is emergent and novel because it can be generated during the game, it combines the rules consistently, and it takes advantage of the random properties of the L-System.

The designer can define the objectives of the game in MaruGen using rules expressed in the grammar of the Machinations framework. For Cubic Explorer, the possible objectives that the player can have are (Figure 12): to obtain a minimum score (number of points), to collect all parts of the spaceship, and to destroy the enemies.

Additionally, the designer can define some conditions on the player's status as (Figure 13): time limit of the game, increase/decrease of the player's health during the game, health level at the beginning of the game, player speed, and type of player attack.

MaruGen also allows the designer to integrate a composition of the rules on the objectives of the game, and the conditions of the player in an L-System, which generates the resulting graph defined in each execution of the game (Figure 14). This makes the game variable between different executions.

The designer can also define rules about the behavior of elements type enemy, mystery box, part of spacecraft, and key. All these elements have a common behavior to either pursue or evade the player, remain still, or patrol (Figure 15). Therefore, these elements are governed by common rules. An L-System allows mixing the different elements, each with its own behavior (Figure 16). This shows the variability of the elements in the game, even if common rules are used.

The general mechanics and rules of Cubic Explorer is defined by the combination of multiple resulting graphs generated by MaruGen: the graph of objectives and conditions of the game, and the graph of the elements of the game.

5.1 Main results

MaruGen was defined and integrated into the Unity3D game engine. MaruGen allowed to design, implement and test the Cubic Explorer video game [23] for three days during the Game Jam Ludum Dare 38. The main results of this participation were:

- Cubic Explorer was fully functional, which shows that MaruGen offers the necessary components to support a team composed of designers, programmers, and video game artists, and generate video game prototypes in an agile way.
- Cubic Explorer was evaluated by 36 users.
- Cubic Explorer obtained the 38th general position among 779 participating games corresponding to the top 5%. This position considers the following aspects: fun, innovation, theme, graphics, humor, and mood.
- Cubic Explorer obtained the 28th position in the aspect of innovation, which we consider is closely related to its mechanism for generating spaces, mechanics, and rules.
- Cubic Explorer obtained the 63rd position in the aspect of the game's theme, which we consider is closely related to the particularity of the spaces, the defined elements, and the mode of exploration offered by the game.
- Cubic Explorer obtained the 244th position in the fun aspect, which we consider is very related to the defined mechanics and rules, and the emergent feature of the game. This aspect is certainly a weakness of the final production of the game, but it is a good opportunity to ask ourselves questions to identify possible flaws: Were the Mission-Space rules or were the Machinations framework rules responsible for the low performance? Are there any bad patterns in the disposition of the rulesets? What singular piece of the game design should the designer change/remove? Instead of trying to look for any specific complex dynamic occurring in place, there is a possibility for just adding, removing or modifying one piece of the system and watch it unfold in a next iteration. The fundamental difference by using MaruGen is that the game designer can pinpoint an

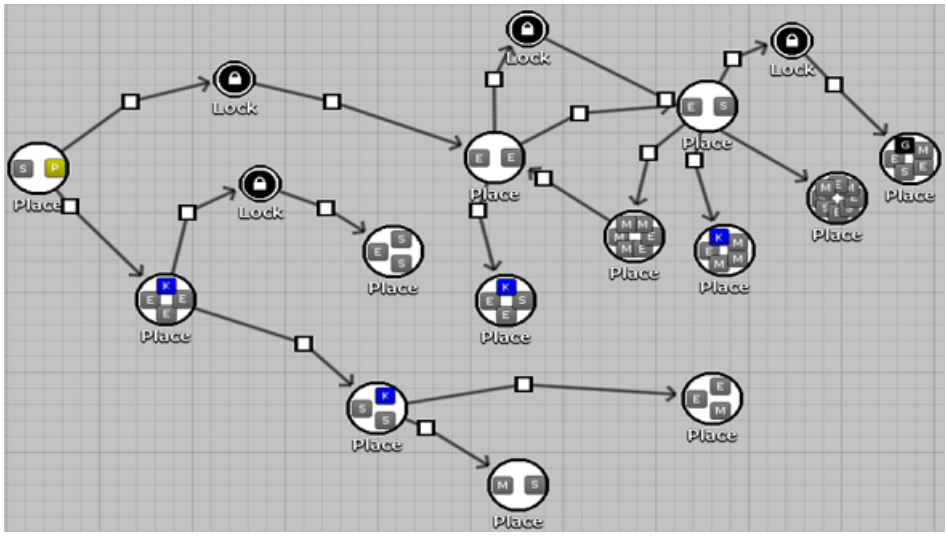

Figure 11 A possible result of the space generated by the L-System of the Cubic Explorer game

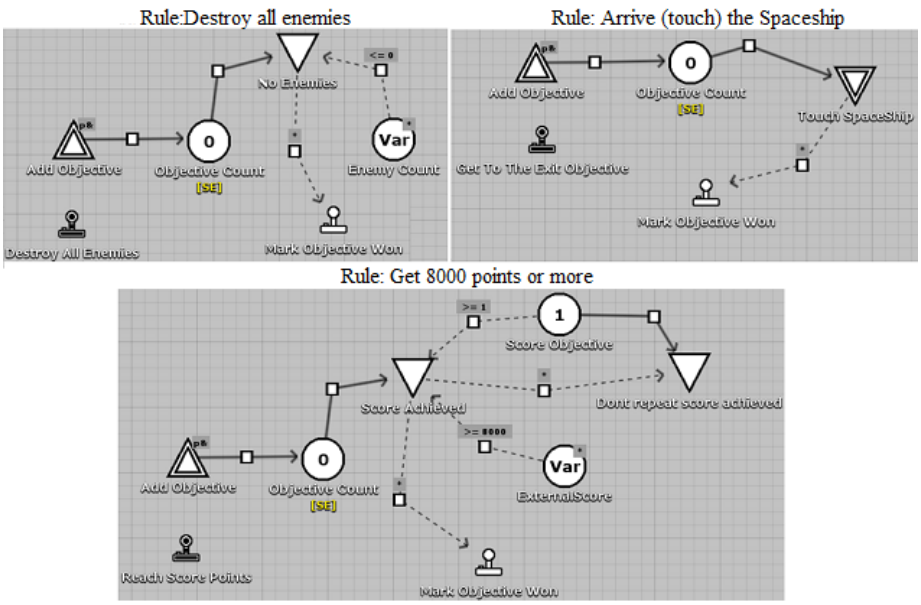

Figure 12 Collection of Machinations rules for the different objectives defined by the game designer

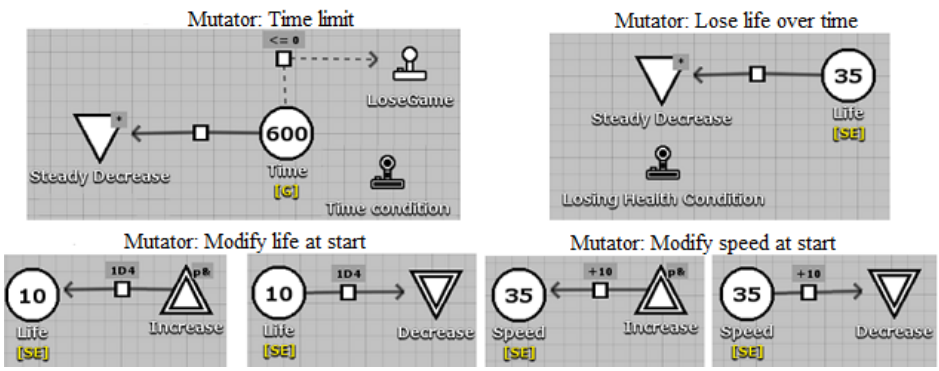

Figure 13 Collections of machinations rules that affect the player's status during the game Cubic Explorer

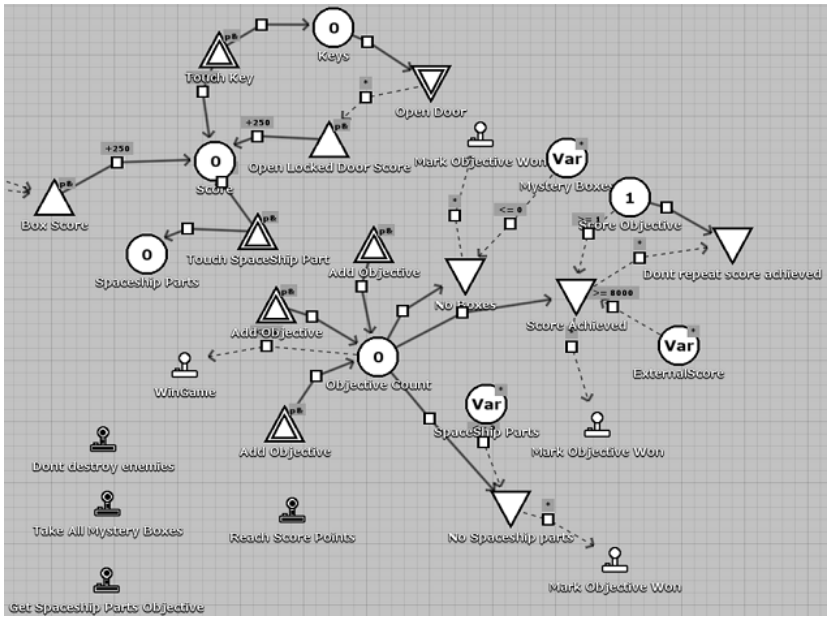

Figure 14 An example of a partial result of Machinations after running the L-System in Cubic Explorer

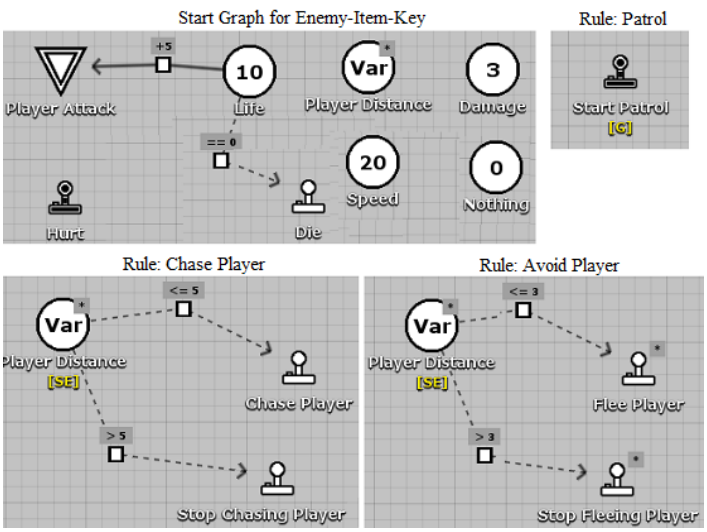

Figure 15 The elements type enemy, spaceship part, mystery box, and key have the same initial graph and set of rules

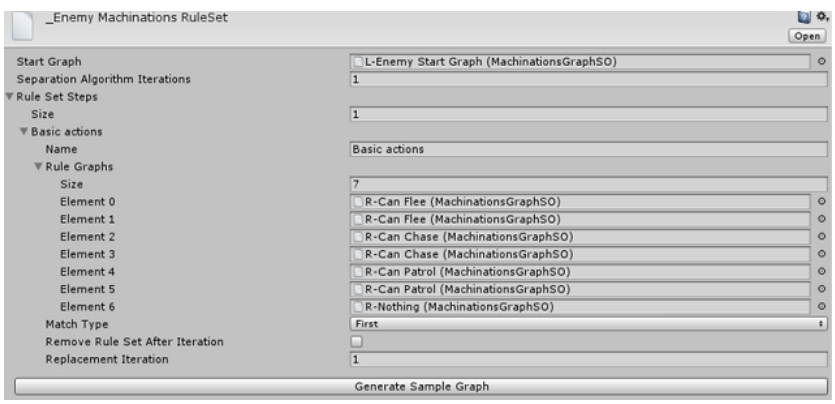

Figure 16 Enemies, mystery boxes, spaceship parts, and keys pass through the same rule filter

exact piece of the puzzle, and s/he can even provide design recommendations (e.g. *"for this type of game you should never use a rule like this, instead, you may use this one"*). This is a process similar to that used by Software Engineers and Architects when designing applications and using software engineering patterns (or anti-patterns).

- The evaluating users of Cubic Explorer made positive comments, and also commented on improvement possibilities.

6. Conclusions and future work

The creation and production of a video game implies a development cycle with different stages and with a very diverse work team (designers, programmers, artists, etc.). Due to how demanding the process is, in recent years, different computational tools have been built that allow and facilitate the development of elements at different stages of a video game.

MaruGen is a computational tool that proposes semi-automatic prototyping of video game designs from sets of basic rules defined by the video game designer. The basic idea is the generation of complex graphs from the expression of these basic rules. Thus, MaruGen uses a rewriting system, based on L-Systems, that allows generating well-defined complex graphs of different complexities and that promotes emergent and progressive properties in video games. This way, game designers can express designs, and they can also build and test game prototypes with their workgroup, easier and faster than applying the usual process. However, MaruGen does not intend to produce a completely satisfactory video game by itself, but rather the analysis of the produced results is made by the designers themselves to understand and improve the properties of the game in development. From a visual representation of the designs, it is required to complete the definition of the video game with the participation of programmers and artists. Additionally, MaruGen works on the Unity game engine, but it is possible to take the base concepts to other types of engines and frameworks. MaruGen reuses elements of the Machinations and Mission-Space frameworks to describe an abstract representation of the space, mechanics, and rules of a game, without having to specify or describe its operation in detail. MaruGen also allows the generation of groupings of rules, collections of basic concepts in games that, from the point of view of a game designer, can be reusable in other objects of the project or even in different projects.

The MaruGen tool was validated in the process of creating the Cubic Explorer video game, which participated in the Game Jam Ludum Dare 38, ranked in the top 5%, and obtaining good scores and comments from gamers and video game developers.

MaruGen seeks to be a computational tool to support the role of the video game designer in his/her process of construction and validation of mechanics and spaces. In that way, designers can discuss, analyze, share and potentially test different rulesets that can validate in their own projects in a concrete, direct way. A great potential result of this effort would be to generate compendiums and collections of "patterns", "anti-patterns", etc., which describe successful pieces of design that can be applied in a general design (or patterns just for specific purposes in certain styles of games).

Nevertheless, MaruGen is not a finished tool; there are several possibilities for improvement. One option is the adjustment and refinement of the proposed rewriting system. Likewise, a complete graph rewriting system can be considered. Another possibility of improvement is to make a better integration with the Unity game engine interface, especially a mode to visualize in real time the graphs of both Machinations and Mission-Space would be a help for verification and testing. It would also be very interesting to consider how to reduce the need for programmer intervention in some steps of video game development, and to facilitate the designer's expression of his/her design ideas in the Unity game engine. Finally, from the point of view of game design, it would be of great interest to be able to extend the concept of rules to rule patterns to increase the expressiveness of the designs. In particular, because of the result obtained in the fun aspect, it would be interesting to discuss patterns and rules using MaruGen to give some sense of "pattern" that can guarantee a certain reaction and dynamic in a game, that can be shared and "plugged in" by other game designers in the rule generation systems to compare and verify, eventually generating a compendium of patterns (or anti-patterns) of game design. This is probably something that does not exist and game designers usually only have as anecdotal proved/true evidence with no formal backup.

7. Declaration of competing interest

We declare that we have no significant competing interests including financial or non-financial, professional, or personal interests interfering with the full and objective presentation of the work described in this manuscript.

8. Acknowledgments

We would like to thank the IMAGINE research group and the Systems and Computing Engineering Department at

Universidad de los Andes (Bogotá, Colombia) for their support.

References

[1] A. Järvinen, "Games without frontiers: Theories and methods for game studies and design," Ph. D. dissertation, University of Tampere, Tampere, Finland, 2006.

[2] C. Crawford. (1984) The Art of Computer Game Design. [McGraw-Hill Osborne Media]. [Online]. Available: https://bit.ly/2T1Y9gW

[3] T. Fullerton, C. Swain, and S. Hoffman. (2008) Game design workshop – A playcentric approach to creating innovative games. [Elsevier & Morgan Kaufmann Publishers]. [Online]. Available: https://bit.ly/2T24hpt

[4] J. Schell. (2008) The art of game design – A book of lenses. [Elsevier & Morgan Kaufmann Publishers]. [Online]. Available: https://bit.ly/3c6OInQ

[5] G. Costikyan, "I have no words & i must design: Toward a critical vocabulary for games," in *Computer Games and Digital Cultures Conference Proceedings*, Tampere, Finland, 2002, pp. 9–33.

[6] R. Hunicke and M. LeBlanc and R. Zubek, "MDA: A formal approach to game design and game research," in *Proceedings of the Challenges in Games AI Workshop, Nineteenth National Conference of Artificial Intelligence*, Menlo Park, USA, 2004, pp. 1–5.

[7] R. Koster, *A Theory of Fun for Game Design*, 1st ed. O'Reilly Media, 2004.

[8] D. Church. (1999, Jul. 16) Formal abstract design tools. [Gamasutra The Art & Business of Making Games]. Accessed Sep. 28, 2019. [Online]. Available: https://bit.ly/32vegGM

[9] R. Koster. (2005) Game design atoms: Can game designs be diagrammed? [GDC Vault]. Accessed Sep. 28, 2019. [Online]. Available: https://bit.ly/2HXjsdg

[10] R. Koster. (2012) A theory of fun 10 years later. [GDC Vault]. Accessed Sep. 28, 2019. [Online]. Available: https://bit.ly/2wcYqVa

[11] D. Cook. (2012, Apr. 30) Loops and arcs. [Lostgarden]. Accessed Sep. 28, 2019. [Online]. Available: https://bit.ly/2PvSaii

[12] A. Anthropy and N. Clark, Eds., *A Game Design Vocabulary*, ser. Game Design/Usability. Addison Wesley, 2014.

[13] N. Falstein and H. Barwood. (2001) The 400 project. [Finite Arts]. Accessed Sep. 28, 2019. [Online]. Available: https://bit.ly/2VurulM

[14] S. Björk and J. Holopainen, Ed., *Patterns in Game Design*, ser. Game Development Series. Charles River Media, 2004.

[15] J. Dormans, "Engineering emergence: Applied theory for game design," Ph. D. dissertation, University of Amsterdam, Amsterdam, Netherlands, 2012.

[16] K. Burgun, *Clockwork Game Design*, 1st ed. CRC Press Taylor & Francis Group, 2015.

[17] D. Cook. (2007) The chemistry of game design. [Gamasutra The Art & Business of Making Games]. Accessed Sep. 28, 2019. [Online]. Available: https://bit.ly/2T84hV7

[18] J. Dormans. (2017) Unexplored. [Ludomotion]. Accessed Feb. 28, 2020. [Online]. Available: https://bit.ly/32I8aDj

[19] P. Prusinkiewicz and A. Lindenmayer. (1990) The algorithmic beauty of plants. [Springer-Verlag]. [Online]. Available: https://bit.ly/2wSRuNu

[20] SideFX. (2013) Houdini engine. [SideFX]. Accessed Oct. 8, 2019. [Online]. Available: https://bit.ly/2we8SM5

[21] SpeedTree. (2002) SpeedTree – 3D Vegetation modeling and middleware. [Interactive Data Visualization, Inc.]. Accessed Oct. 8, 2019. [Online]. Available: https://bit.ly/2Tp4XnA

[22] L. Lapré. About Lsystems and Lparser. [laurenslapre]. Accessed Oct. 8, 2019. [Online]. Available: https://bit.ly/3974SMb

[23] I. F. Capasso. (2017) Ludum Dare 38 Cubic Explorer. [Ldjam]. Accessed Oct. 8, 2019. [Online]. Available: https://bit.ly/2TijJN4