MaruGen: Una Herramienta para la Generación Semi-Automática de Diseño de Videojuegos

MaruGen: A Tool for Semi-Automatic Generation of Videogame Designs

Italo F. Capasso B. [0000-0002-2530-0048] y Fernando De la Rosa [0000-0002-9066-7225]

Universidad de los Andes, Bogotá, Colombia
{i-capass, fde}@uniandes.edu.co

***Abstract*:** The art of game design and development is by nature a complex and multidisciplinary process that requires multiple roles, processes and talents for its successful completion. One of the most crucial steps, game design (that is the creation of mechanics, rules, mission and objectives of a videogame) is one of the processes that has less formal development, despite its importance in the conception of a given game. MaruGen (Machinations Ruleset Generator) is a semi-automatic system that generates spaces and game mechanics, by using rulesets and grammar from previous attempt at designing formal grammars, in an effort to formalize the field with a computational tool. This system was tested by supporting the designer of a new videogame to participate in the Ludumdare GameJam.

***Resumen*:** El arte del diseño y desarrollo de juegos es por naturaleza un proceso complejo y multidisciplinario, que requiere múltiples roles, procesos y talentos para su realización exitosa. Una de las etapas críticas corresponde al diseño (creación de mecánicas y reglas, y definición de la misión y objetivos del videojuego) y es la etapa en que se tiene menos desarrollo formal a pesar de su importancia en la concepción de un juego. MaruGen (*Machinations Ruleset Generator*) es un sistema semiautomático que genera espacios y mecánicas de juego a partir de un conjunto de reglas y una gramática la cual se toma de intentos anteriores en el diseño formal de gramáticas, en un esfuerzo para formalizar el campo de videojuegos. Este sistema fue probado apoyando al diseñador de un nuevo videojuego con el cual se participó el *GameJam* Ludumdare.

Palabras Claves: Generación procedimental de contenido, diseño automático de juegos, sistema de reescritura de términos, Sistemas-L, Machinations, Micro-machinations, Mission/Space.

1 Introducción

Los videojuegos tienen problemas complejos y de naturaleza multidisciplinaria a resolver en términos de diferentes tipos de *assets* y recursos (imágenes, audios, código, etc.), verificaciones "esotéricas" sobre si una implementación es exitosa (p.ej. ¿Es el videojuego divertido?) y la generación de mundos consistentes, la estética y el sistema de reglas. Una parte importante en la mejora continua del desarrollo de videojuegos es la generación de contenido procedimental.

La generación procedimental de contenido (usualmente conocida como PCG [1]) se ha vuelto muy importante en la industria de videojuegos por muchos tipos de juegos y a diferentes escalas. Desde videojuegos pequeños, independientes y experimentales hasta los últimos de vanguardia con desarrollo a gran escala, la generación procedimental es actualmente una parte esencial en el desarrollo de un juego.

En las últimas dos décadas, muchos sistemas de generación de contenidos fueron creados para diferentes *assets* en los videojuegos. Entre los más comunes están audios, animaciones y modelos 2D y 3D. También hay un uso extensivo de contenido procedimental para crear elementos de un videojuego que se relacionan con el diseño de un juego, como elementos muy específicos, sus espacios, niveles y eventos.

La generación de sistemas de juego y reglas sin embargo es un reto más complicado. La idea de generar sistemas de reglas y de mecánicas para juegos ha sido relegada generalmente al rol del diseñador, quien crea las reglas, los prototipos de papel y digitales y realiza las pruebas de juego con jugadores de una forma interactiva. Esto es hecho generalmente en la etapa de preproducción del juego y se mantiene a través de todo el ciclo de desarrollo del juego independiente de la escala del juego. El diseñador tiene a su cargo una de las partes más manuales y basadas en su instinto en el proceso de crear un juego. Este proceso iterativo tiende a ser uno de los más críticos en el desarrollo de un videojuego, y muchos diseñadores nunca han utilizado ningún tipo de automatización. Aunque existen sistemas y herramientas de apoyo a la automatización del diseño, pocas de ellas se utilizan realmente en la industria en el proceso de diseño.

Con este propósito, nosotros proponemos MaruGen, una herramienta computacional, capaz de asistir en el proceso de diseño y creación de reglas, objetivos, misiones y espacios en un videojuego. Esta herramienta semi-automática y diseñada para ser asistida por un diseñador de juegos y otros roles, como programadores y artistas, puede contribuir en la generación de nuevas y novedosas mecánicas de videojuegos, basado en los supuestos del diseñador de videojuegos y definidos como reglas dentro del sistema. MaruGen construye a partir de la idea general que el diseñador propone soluciones de mecánicas y espacios estableciendo reglas sencillas, que extrapola en un juego más completo. Este artículo explica el sistema propuesto y su filosofía, que incluye las suposiciones de base y precisa cuando y como interviene el diseñador.

Este artículo se organiza en varias secciones: En la sección 2 se explica el estado del arte de varios métodos propuestos para el desarrollo de sistemas de diseño de juegos y los últimos intentos de herramientas automáticas de creación de diseños de juegos y paradigmas. En la sección 3 se introducen los fundamentos y referencias del sistema propuesto basados en los *frameworks* Machinations y Mission/Space (propuestos por Dormans [2]), y las ideas sobre el concepto de sistemas de reescritura de términos. En la sección 4 se enuncia el problema de interés y en la sección 5 presenta el concepto general de la herramienta propuesta y el proceso básico de implementación y evaluación. La sección 6 termina con las conclusiones y el trabajo futuro por realizar.

2 Estado del Arte en Generación Automática de Juegos

La generación automática de diseños de juego ha estado en desarrollo por dos décadas. Muchos aspectos de PCG se enfocan en aspectos específicos del desarrollo de juegos. Niveles, audios, modelos, *sprites* son parte del dominio de PCG. El campo de diseño de juegos ha sido un concepto difícil que tiene raíces en la creatividad natural requerida en su desarrollo. Generalmente esto ha sido aproximado de dos formas: Generación de reglas con gramáticas y representación de grafos.

2.1 Generación de Conjuntos de Reglas con Gramáticas Formales

La idea de usar gramáticas formales para generar una "gramática general" para un videojuego ha sido una solución muy popular y una de las primeras propuestas siendo el proyecto E.G.G.G [3] el que intenta modelar un lenguaje de juego para el desarrollo de Inteligencia Artificial (IA).

Probablemente, la primera implementación impactante conocida es la de Tongelius y Schmidhuber [4]. Este artículo implementa una idea de límites de tiempo y puntos, movimiento lógico, relaciones aleatorias con eventos de colisión, localización aleatoria de obstáculos y entidades en un espacio simple de juego. Todo esto unido con una propuesta de usar métodos combinados con una métrica "divertida" en la función de *fitness*.

Variations Forever [5] usa un conjunto rico de generación de reglas, asociado con términos simbólicos y generación de respuestas (ASP) para encontrar un espacio generativo de lógica para lograr una generación flexible del juego. Otra idea es la primera iteración de ANGELINA [6][7] que utiliza la misma idea de base de Togelius y Schmidhuber [4] utilizando un algoritmo coevolutivo para encontrar el mejor juego generado después de muchas generaciones.

En el trabajo de Browne y Faire [8] se creó un sistema denominado Ludi. Ludi utiliza una aproximación similar basada en un sistema de generación de reglas para juegos de tablero que generó el comercialmente conocido juego Yavalath y el lenguaje "*Video Game Description Language*" (VGDL). VGDL es un lenguaje formal de propósito general que intenta describir cualquier posible videojuego el cual fue creado como una herramienta de investigación buscando generar un juego de propósito general con IA [9].

Muchos de estos autores, trabajos y técnicas se encuentran compilados en el libro de Shaker *et al.* [1]. Este libro contiene las más recientes reflexiones sobre el estado del arte del PCG, proyectos realizados con estas técnicas y acerca del futuro del contenido procedural en videojuegos.

2.2 Representaciones de Grafos y Relaciones Semánticas

Otra idea en el campo de diseño automático de videojuegos en la dirección de generación de reglas y sistemas utilizando lenguajes formales proviene de las representaciones gráficas en forma de grafos y el uso de asociaciones semánticas de palabras con suposiciones de la vida real humana. Una de estas propuestas se encuentra en los *fra-*

meworks Machinations, Mission/Space y Micro-machinations [2]. Estos *frameworks* usan únicamente representaciones de grafos para describir sus sistemas. Y aunque se ajusta a una gramática, no intenta tener todos los requisitos de una gramática formal, y como tal es sujeta a otros tratamientos. Estos *frameworks* se explicarán en detalle en la sección 3.

Otro tipo de generadores de juego están basados en relaciones semánticas que se pueden representar como grafos. Ejemplos de estos están the Game-O-Matic, Micro Rhetorics [10] y el trabajo propuesto por Nelson y Mateas [11]. Estos generan mecánicas basadas en palabras aleatorias y luego usan Internet y APIs como ConceptNet y WordNet para generar conexiones y asociar estas con mecánicas de juegos.

Las diferencias de estos sistemas con la aproximación de gramáticas formales son: Primero, los juegos no tienden a enfocarse en las mecánicas en sí mismas sino en las relaciones entre ellas (p.ej. Machinations considera las relaciones de juegos como un sistema económico y Game-O-Matic ve las relaciones de semántica entre palabras). Segundo, no se requiere un lenguaje formal "estricto", únicamente su construcción básica y sus relaciones deben funcionar para lograr que el juego funcione de forma efectiva.

3 Fundamentos – Ideas e Implementaciones

A continuación, se presentan las ideas y teorías fundamentales en las que se basa nuestra herramienta de generación automática de juegos. Las ideas principales se basan en los *frameworks* propuestos por Dormans, el uso de modelos de transformación y el concepto de gramática formal generativa en el uso de sistemas de reescritura de términos. Estas ideas se presentan en el trabajo de J. Dormans [2].

3.1 *Frameworks* de Base (Machinations y Mission/Space)

***Framework* Machinations.** Este *framework* se centra en el desarrollo de un sistema gráfico de un lenguaje formal cuyo objetivo es representar, generar, y probar mecánicas de juego a partir de unas reglas básicas y limitaciones de un videojuego. Este sistema usa un lenguaje visual específico que intenta modelar las interacciones en un juego como transacciones y transformaciones sencillas. Este *framework* ha sido estudiado y utilizado para la generación de una herramienta de uso libre (Fig. 1). En iteraciones posteriores se llegó al ambiente de producción de juegos Micro-machinations [12] para un mejor uso práctico.

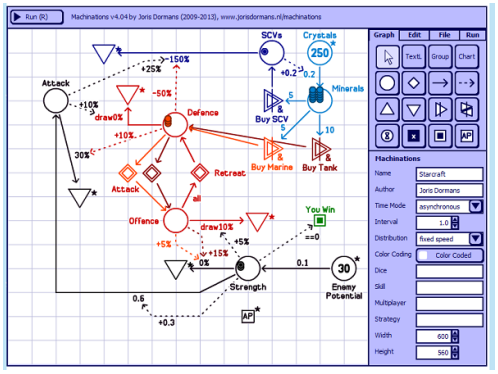

Fig. 1. Un pantallazo del *framework* Machinations de J. Dormans [2].

***Framework* Mission/Space.** Este *framework* define la estructura para que un nivel de un videojuego pueda jugarse [2]. El *framework* Mission se centra en el avance del jugador definiendo objetivos, obstáculos y lugares de avance a través de metáforas como candados, llaves y enemigos. El *framework* Space se centra en definir la estructura del nivel y su relación con la misión. Ambos *frameworks* están relacionados. Posteriormente se creó la aplicación Ludoscope (Fig. 2) que aprovecha las ideas de ambos *frameworks* en un sistema de diseño de juegos.

Estos *frameworks* fueron creados originalmente para un uso conceptual, permitiendo al diseñador crear la estructura de un juego, implementándolo manualmente. Sin embargo, Dormans propone su uso para la creación generativa de juegos debido a su naturaleza formal y orientada por una gramática, más allá de un uso académico.

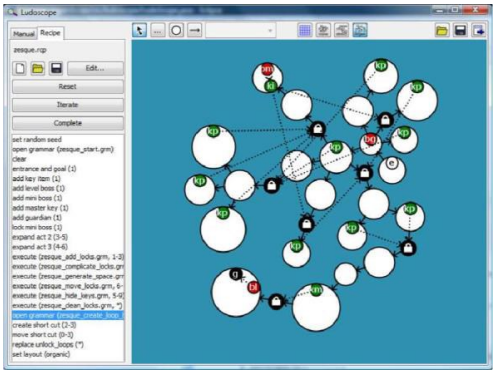

Fig. 2. Un pantallazo de la herramienta Ludoscope de J. Dormans [2].

3.2 Modelo de Transformaciones

El modelo de transformaciones es parte del estilo de ingeniería de software "*Model Driven Architecture*". La idea es modelar todos los elementos del diseño de la arquitectura de software. Esto implica que, usando procesos de abstracción, todas las ideas

harán parte de un modelo específico, y luego son transformadas en un nuevo modelo que concreta cada idea. Un ejemplo específico es UML, en el cual el diseñador modela su arquitectura y conceptos de un problema de diseño de software a través de diagramas y luego aplica un proceso de transformación para generar el código del modelo. Este proceso ya se puede realizar de forma automática en Ingeniería de *Software*.

Múltiples niveles de abstracción y transformación pueden ocurrir. Dormans propone que el modelo de transformación se aplique al diseño de juegos y que se transforme de forma generativa en código y en estructuras específicas del juego.

3.3 Gramáticas Formales Generativas, Sistemas de Reescritura de términos y Sistemas-L

Acorde a la idea de modelo de transformaciones, presentamos dos conceptos que tratan la idea de generación y transformación de sistemas. La propuesta original de gramáticas formales viene de Chomsky [13] las cuales se definen a partir de símbolos terminales, símbolos No-terminales, símbolos iniciales y un sistema de reglas de reescritura. Estas gramáticas se definen como una sucesión de símbolos no-terminales, generados por la aplicación de un número finito de veces de las reglas de la gramática.

Los sistemas de reescritura de términos definen una aproximación diferente en el manejo de las transformaciones. Estos se definen por símbolos y reglas de reescritura [12]. Se cuenta con un conjunto finito de símbolos que se consideran axiomas (es decir se asumen verdaderos) y un conjunto finito de reglas. Las reglas intentan reemplazar parte de los símbolos con otro conjunto, similar a la simplificación de igualdad en Algebra. Una diferencia con las gramáticas formales es que no hay símbolos iniciales y no hay diferencia entre los símbolos terminales y los símbolos no terminales. Una ventaja de estos sistemas es que sus transformaciones siempre son consistentes.

Teniendo en cuenta lo anterior, Dormans propone un sistema de reescritura para las reglas y mecánicas de un juego (uso del *framework* Machinations), un sistema de reescritura de grafos y un sistema de reescritura de formas que permite trabajar con el *framework* Mission/Space y generar niveles físicos [2]. Una primera aplicación de esta propuesta donde se implementó un prototipo básico se presenta en [14].

Existe un subconjunto de sistemas de reescritura de términos, llamados Sistemas Lyndenmayer, comúnmente llamados Sistemas-L o *L-Systems* [15]. Estos sistemas definen una gramática formal, pero tienen diferencias con las gramáticas de Chomsky: por un lado, los sistemas-L no tienen elementos terminales, lo cual define gramáticas generativas. Por otro lado, los sistemas-L aplican sus reglas de forma paralela. Adicionalmente los sistemas-L tienen otras características: Pueden ser con/sin contexto, determinísticos/estocásticos y paramétricos.

Aunque los sistemas-L son simples pueden crear modelos complejos a partir de pocas reglas sencillas debido a la propiedad emergente propia de estos sistemas [16] (Fig. 3).

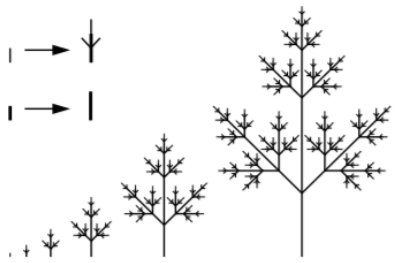

Fig. 3. Un ejemplo de un sistema-L[1]. Usando reglas simples se generan modelos complejos.

Los modelos teóricos anteriores junto con las técnicas de transformación de modelos de diseño de juegos son la base de la herramienta de generación automática de diseño de videojuegos propuesta en este artículo.

4 Problema de Estudio

El problema de generación automática de juegos es complejo, con diferentes propuestas y técnicas, dados los múltiples aspectos multidisciplinarios que se tratan en el proceso de un videojuego. Sin embargo, no todos los aspectos se trabajan en la generación procedimental de juegos. De hecho, el diseño de videojuegos es una de las áreas menos exploradas y con menos soluciones computacionales concretas en aplicaciones prácticas.

Teniendo en cuenta lo anterior, este artículo propone una solución a la generación automática de diseño, utilizando los *frameworks* propuestos por Dormans y sistemas de reescritura de términos (en particular, Sistemas-L) para la definición de mecánicas, misiones y espacios de un juego completo. Esto es un intento de generar un juego completo, a partir de la entrada del diseñador expresando las reglas de reescritura de todo el sistema, complementado con el aspecto de programación que genera el sistema físico (como el movimiento y la interacción, y construyendo el espacio mismo).

5 MaruGen – Solución Propuesta

La solución propuesta consiste en desarrollar una herramienta computacional para asistir al diseñador de juegos en la creación de reglas, objetivos, misiones y espacios en un videojuego, a partir de un conjunto básico de reglas simples. La herramienta se llama MaruGen (*Machinations Ruleset Generator*) la cual define una librería completa para el motor de juegos Unity3D. MaruGen utiliza sistemas-L como base para aplicar el proceso y métodos recomendados por Dormans de forma automática partiendo desde cero. La única asistencia del diseñador del juego es la creación de reglas básicas de reescritura para las mecánicas, misiones y contenido del juego, y para los desarrolladores del juego para programar los grafos resultantes de forma consistente.

[1] http://procedural-content-generation.wikia.com/wiki/L-Systems

MaruGen, como herramienta abstracta, está pensada para el uso de desarrolladores de videojuegos, que tienen como rol el diseñador de juegos. Esto es, aquél que define las reglas, espacios, mecánicas, misiones y objetivos de un juego dado.

Sin embargo, MaruGen no funciona solo como una herramienta abstracta, es decir, no es solo una herramienta visual e informativa. MaruGen también es una herramienta formal, capaz de convertir sus diagramas abstractos en representaciones concretas, implementadas por un programador y llenadas de contenido por artistas (Fig. 4).

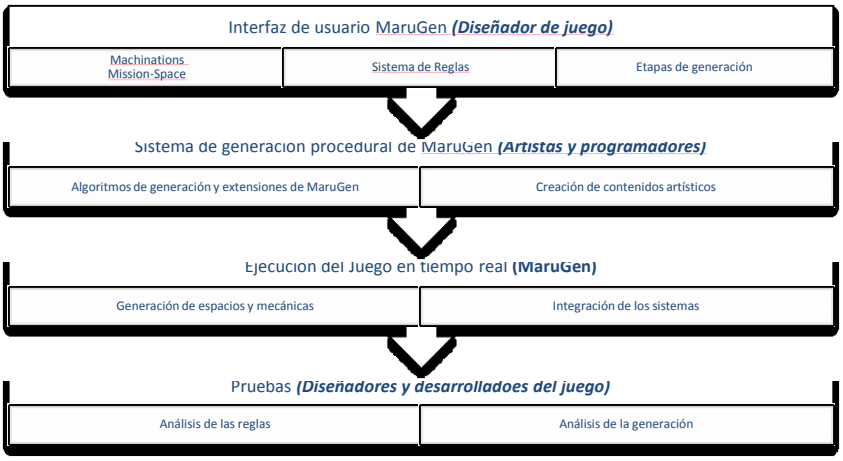

Fig. 4. – Etapas de creación de un juego utilizando la herramienta de MaruGen.

5.1 Herramienta Básica de Grafos en Unity3D

Para utilizar el sistema correctamente como un sistema-L generativo fue necesario primero construir la interface completa para crear, manipular y examinar la gramática en Unity3D. Unity tiene la capacidad de extender su interface de usuario, así que se trabajó en este aspecto para tener un sistema usable para manipular los grafos de base (Fig. 5). Las representaciones de estos grafos son codificadas en *ScriptableObject*, un contenedor especial de *assets* en Unity, que se puede exportar a formato JSON (diseñado para ser utilizado por fuera del motor Unity). Finalmente se utilizan una serie de *Behaviours* para especificar el tipo de generación y manejo en tiempo real del sistema (Fig. 6).

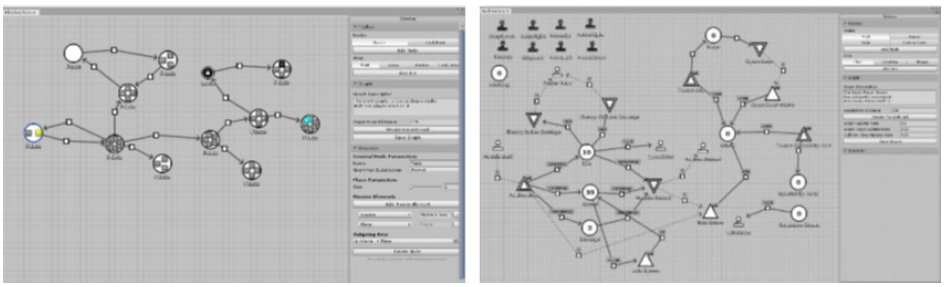

Fig. 5. Pantallazos de las herramientas de Grafos de MaruGen.
En la parte izquierda una representación de un grafo de Mission/Space.
En la parte derecha un grafo de Machinations.

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

Fig. 6. Una representación de la arquitectura de MaruGen en Unity.

Dado que estas representaciones de grafos son sólo entidades abstractas, un sistema para el programador del juego se consideró necesario para generar y extender los espacios necesarios. Un conjunto de clases que pueden ser extendidas y reemplazan métodos virtuales para crear un sistema de generación personalizado de acuerdo a las necesidades (Fig. 7).

El primer paso es lograr definir las reglas básicas como relaciones simples y personalizadas en grafos. A partir de este inicio, el diseñador puede lograr sistemas más ricos y complejos que pueden generar espacios y mecánicas más interesantes para el videojuego.

De hecho, el sistema trabaja por sí mismo de esta forma: se diseñan la(s) gramática(s) y se ejecuta(n) con la herramienta para ver los resultados obtenidos, teniendo en cuenta que el programador interviene para llenar los “espacios en blanco” en la generación del contenido y los comportamientos básicos del juego.

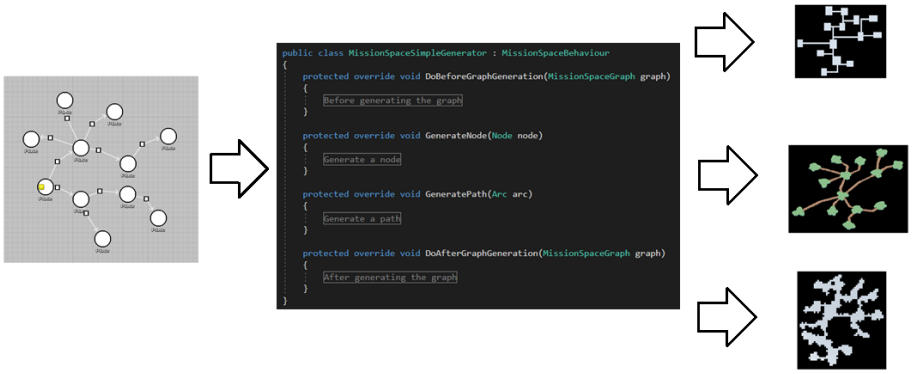

Fig. 7. Los desarrolladores de un juego pueden crear diferentes tipos de mundos y sistemas, usando diferentes algoritmos de generación procedimental a través de clases que se extienden y se aplican sobre los grafos generados.

5.2 Generador de Sistemas-L

Una vez el diseñador del juego completa las reglas básicas de mecánicas, misiones y espacios, el generador del juego genera una estructura del juego de manera aleatoria (Fig. 8). En este paso puede haber reglas especiales de reescritura diseñadas para que elementos como el ritmo, la dificultad y/o la variedad se tengan en cuenta al generar niveles y desafíos en el juego. Estas reglas pueden ser editadas y modificadas si se requiere.

Dentro del sistema generador de reglas, se puede especificar el tipo de reemplazo, el número de iteraciones, el orden de aparición de cada regla y las diferentes etapas de generación. Opcionalmente se puede generar un grafo dentro del editor para luego guardarlo y cargarlo posteriormente. O usarlos en tiempo real (a través de un *behaviour*) para generar un grafo nuevo cada vez que se requiera.

El sistema puede aplicar estas reglas aleatoriamente o a partir de parámetros definidos por el usuario diseñador/programador. La gramática resultante es "mejorada" usando un algoritmo de fuerzas en grafos, principalmente por problema de legibilidad, y también porque la espacialidad es importante en los grafos resultantes de Mission/Space.

5.3 Proyecto de Prueba y Generación de un Videojuego

Esta herramienta es útil si puede aplicarse en un "contexto real", esto es, generar un juego completo usando el sistema MaruGen. Algunos ejemplos y casos de prueba fueron generados para mostrar conceptos sencillos en el uso básico de la herramienta.

El juego Cubic Explorer fue creado completamente en un periodo de tres días en el contexto del *GameJam* Ludumdare para tener una experiencia completa con la herramienta (Fig. 9).

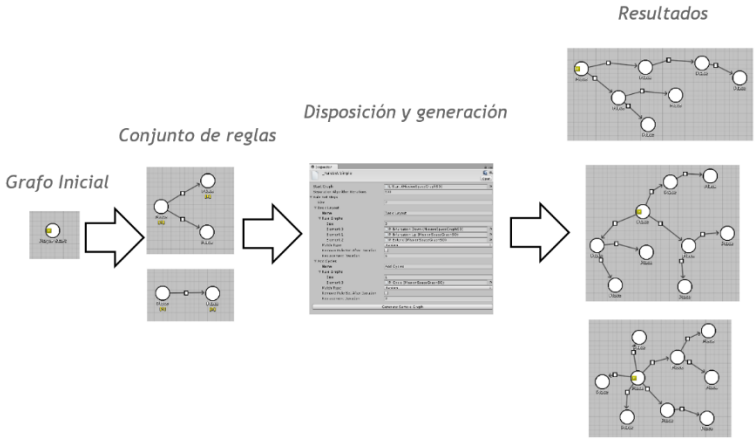

Fig. 8. El diagrama ilustra el proceso el funcionamiento de MaruGen a partir de un grafo inicial y un par de grafos de reglas simples, luego aplica reescritura manipulándolos con un script específico y como resultado genera representaciones complejas.

Fig. 9. Pantallazos del juego de prueba Cubic Explorer.

Algo a destacar de Cubic Explorer es que intenta usar los diagramas de manera particular. Los diagramas de Mission/Space no son usados en el sentido literal. En este juego, las rutas definidas en el diagrama se representan como nodos que definen puertas las cuales transportan al jugador a otras localizaciones tipo cubo. El generador del diagrama Machinations se usa para transmitir diferentes objetivos del juego, para dar diferentes comportamientos al personaje jugador (disparar en lugar de luchar con espadas, y una variedad de estadísticas) y definir diferentes comportamientos a enemigos y objetos del juego.

El juego completo junto con el código fuente se encuentra en el sitio web del evento Ludumdare[2].

6 Conclusiones y Trabajo Futuro

MaruGen es un sistema que intenta generar juegos semiautomáticos, a partir de crear una mecánica y/o reglas espaciales sencillas. Funciona sobre el motor de juegos Unity pero es posible extender los conceptos a otros *frameworks*.

Este artículo explora diferentes sistemas y conceptos existentes acerca del tema y analiza ideas de la tesis doctoral de J. Dormans que incluye gramáticas formales, sistemas de reescritura de términos y sus *frameworks* Machinations y Mission/Space.

MaruGen ofrece la posibilidad al diseñador de juegos de aprovechar la propiedad del juego emergente propuesta por Dormans [2]. Uno de los objetivos principales de este trabajo es el de proporcionar la capacidad de expresar conceptos abstractos en una herramienta computacional práctica que pueda utilizar el diseñador de juegos, en un intento de formalizar el rol del diseñador en la industria.

MaruGen no es un proyecto terminado. Hay varios aspectos de su concepción queque no se terminaron. Hay trabajo por realizarse en su usabilidad, en elaboración de especificaciones y estabilidad para hacer uso real de esta herramienta en la industria. También hay que revisar como otras cualidades de juego, como la "progresión", pueden integrase en el sistema. Además, usar el sistema en proyectos más grandes y profesionales es un excelente punto de partida para concretar su aplicación en casos reales.

Otro punto de interés en el trabajo futuro es trabajar en grupos o patrones de reglas. La idea es que diferentes patrones de reglas puedan ser aplicables en más de un juego. Esto es un paso hacia una aproximación de diseño de juegos basada en patrones.

Referencias

1. Shaker N., Togelius J. y Nelson M.J.: Procedural Content Generation in Games. Springer International Publishing (2016).
2. Dormans J.: Engineering Emergence (PhD dissertation). Universiteit van Amsterdam (2011).
3. Orwant J.: EGGG: Automated programming for game generation. IBM Systems Journal 39 (3.4), 782–794 (2000).
4. Togelius J. y Schmidhuber J.: An Experiment in Automatic Game Design. In: Proc. IEEE Symposium on Computational Intelligence and Games (2008).
5. Smith A.M. y Mateas M.: Variations Forever: Flexibly Generating Rulesets from a Sculptable Design Space of Mini-Games. In: Proc. IEEE Conference on Computational Intelligence and Games (2010).
6. Cook M. Colton S. y Gow J.: The ANGELINA Videogame Design System, Part I. IEEE Transactions on Computational Intelligence and AI in Games 9(2), 192 – 203 (2016).

[2] Revisar https://ldjam.com/events/ludum-dare/38/cubic-explorer y https://bitbucket.org/glitchypixel/cubic-explorer-ldjam-38-and-basic-marugen

7. Cook M.: The ANGELINA Videogame Design System, Part II. IEEE Transactions on Computational Intelligence and AI in Games PP (99), (2016).
8. Browne C. y Maire F.: Evolutionary Game Design. IEEE Transactions on Computational Intelligence and AI in Games 2(1), 1–16 (2010).
9. Ebner M., Levine J., Lucas S.M., Schaul T., Thompson T. y Togelius J.: Towards a Video Game Description Language. Artificial and Computational Intelligence in Games 6, 85–100 (2013).
10. Treanor M., Blackford B., Mateas M. y Bogost I.: Game-O-Matic: Generating Videogames that Represent Ideas. Procedural Content Generation in Games, (2012).
11. Nelson M.J. y Mateas. M.: Towards Automated Game Design. Artificial Intelligence and Human-Oriented Computing LNCS 4733, Springer (2007).
12. Van Rozen R. y Dormans J.: Adapting Game Mechanics with Micro-Machinations. Foundations of Digital Games LNCS 8255, (2014).
13. Chomsky N.: Three models for the description of language. IRE Transactions on Information Theory 2(3), 113–124 (1956).
14. Dormans J. y Bakkes S.: Generating Missions and Spaces for Adaptable Play Experiences. IEEE Transactions on Computational Intelligence and AI in Games 3 (3), 216–228 (2011).
15. Lyndemayer A. y Prusinkiewicz P.: The Algorithmic Beauty of Plants, Springer-Verlag, (1990).
16. Griswold R.: Designing with L-Systems, Part 1: String Rewriting Systems. (2004). http://www.cs.arizona.edu/patterns/weaving/webdocs/gre_ls01.pdf. [Ultimo acceso: Mayo 2017].