
**Escuela Técnica de Ingeniería Informática y Telecomunicaciones**

***Arquitectura y Tecnología de Computadores***

PROYECTO DE FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

# Content generation in videogames

**Alumno:** Pedro Luis Trigueros Mondéjar
**Director:** Pedro Castillo Valdivieso
**Fecha:** 17 de agosto de 2015

D. Director del proyecto
Pedro Castillo Valdivieso
Arquitectura y Tecnología de Computadores
Universidad de Granada

CERTIFICA: Que la memoria titulada *Content generation in videogames* ha sido realizada por Pedro Luis Trigueros Mondéjar bajo mi dirección y constituye su Proyecto de Fin de Grado de Grado en ingeniería informática.

En Granada, a 17 de agosto de 2015

D. Pedro Castillo Valdivieso
Director del proyecto

## Resumen:

Los videojuegos es uno de los sectores emergentes en los últimos años a nivel mundial. El proyecto está basado en la generación procedural de contenidos para videojuegos, en este caso, los contenidos que se van a generar son ciudades, para ello se ha creado una biblioteca de generación de ciudades con tres algoritmos, la biblioteca ha sido creada para Autodesk 3D Maya, ya que se puede trasladar de una manera muy sencilla los resultados a Unity 3D uno de los motores de videojuegos más usado en los últimos años. Dos de los algoritmos son bastante competentes con los que hay actualmente en el mercado y otro, es un algoritmo que necesitaría grandes requisitos para su uso, pero da mejores resultados que los anteriores. Tras la creación de los algoritmos se hizo un análisis a posteriori, llegando a la conclusión de que todos los algoritmos tienen una eficiencia O($n$), también se hizo un estudio de la memoria llegando a la conclusión de que los dos algoritmos con mejores tiempos, también usaban pocoa memoria, por lo que se podrían generar ciudades muy grandes en cualquier juego. Por último esta biblioteca se utilizará para generar ciudades en videojuegos futuros y está libre en GitHub.

## Lista de palabras clave:

Videojuegos, generación procedural, computational intelligence in games, evolutionary game design.

*Dedicado a todos a los que nunca se les ha dedicado nada.*

# Agradecimientos

Principalmente a Pablo García Sánchez por haber hecho posible este proyecto, también a todos los que creyeron alguna vez en mí y a todos los que estuvieron siempre apoyándome.

# Índice general

# Índice de figuras

# Índice de tablas

# Introduccion

Han pasado más de sensenta años desde que Alexander S. Douglas, crease el primer juego computerizado "Nought and crosses", a través del cual una persona y una máquina se veían enfrentadas en un tablero del popular juego de tres en raya. Hoy en día podríamos empezar a hablar que los videojuegos se han convertido en uno de los mercados económicos más valorados por la sociedad, prueba de este auge desde la segunda mitad del S. XX y comienzos del S. XXI han sido la creación de nuevas e importantes empresas que se encargan del desarrollo de este tipo de productos, ejemplo de ello pueden ser empresas internacionales como "ATARI" o "NINTENDO", y nacionales como "PYRO STUDIOS".

Desde la incorporación de los videojuegos a la vida cotidiana, el ser humano a tratado de buscar no sólo el carácter lúdico de estos, sino también hallando la forma con la que poder trasmitir una serie de valores cívicos y conocimiento a la ciudadanía. Por ello se habla también de un segundo carácter que sería denominado como educativo. Por otro lado una de las características más llamativas del sector es su amplio repertorio para todos los públicos, idea inconcedible unas décadas atrás, ya que los videojuegos estaban ideados para un público en concreto, pero a raíz de los avances tecnológicos sufridos en esta industria, se ha producido una mayor difusión de los contenidos de los videojuegos, buscando que accesibles a todos el mundo. Por ende hoy en día, podemos encontrar videojuegos para niños, adolescentes, adultos y ancianos[13].

En los últimos años se han realizado grandes avances en la computación grafíca, dando lugar a la realidad virtual y una gran cantidad de sensores que detectan el exterior para reproducirlo dentro del propio juego. Consolas como "NINTENTO WII" o "XBOX ONE" ha intentado enforcar sus últimos lanzamientos en cámaras que detecten el movimiento de la persona. Estos avances han permitido el desarrollo de habilidades diversas, como motoras o creativas. Por otro lado, empresas como "OCULUS VR" han creado gafas de realidad virtual con las que el usuario puede establecer mayor contacto con el mundo virtual.

Otro de los grandes avances que se ha producido en los videojuegos en la última década es la mejora de la inteligencia artificial en estos. Se han implementado algoritmos como el A*, para mejorar la búsqueda de caminos en bots, también hemos podido ver un gran avance en algoritmos de aprendizaje automático con los que la máquina, aprende de nuestras acciones y sabe como contrarestarlas, pero nosotros nos centraremos en la generación procedural.

Hay diez principales tipos de investigaciones en el campo de la inteligencia artificial en videojuegos, estos se dividen según tres perspectivas, la primera el uso de la inteligencia artificial en el cada área (métodos de perspectiva computacional), por otra parte podemos analizar la relación del área respecto a lo que hace el jugador, humano,(perspectiva del jugador) y para finalizar, podemos analizar la zona en la que el jugador realiza acciones (perspectiva de la interacción jugador-juego), el lugar dónde se desarrolla el juego.[5]

Las investigaciones son las siguientes:

- Estudio de comportamiento de los NPC (Non-player character)
- Búsqueda y planificación
- Modelado de personajes
- Puntos de referencia en la inteligencia artificial de juegos
- Generación procedimental de contenido
- Narración computacional
- Creación de agentes creíbles
- Diseño de juego asistido con inteligencia artificial
- Inteligencia artificial general en juegos
- Inteligencia artificial en juegos comerciales

Las áreas de investigación tienen relaciones bastante fuertes entre ellas, el avance de alguna de ellas implica que se pueden mejorar las investigaciones con las que tienen relaciones.

## Objetivos del TFG

**Primer objetivo:** Consiste en la creación de una biblioteca que contenga algoritmos para la creación procedural de ciudades, cuya salida sea posible utilizarse en motores de

juegos como "UNITY 3D" o "UDK". Esta librería tendrá licencia GPL para permitir que cualquier persona pueda modificarla y utilizarla libremente sin nigún problema.

**Segundo objetivo:** Comparar los distintos algoritmos de generación procedural de ciudades de la biblioteca para ver cuál sería más óptimo para su utilización en un juego determinado, analizando los algoritmos a posteriori y obteniendo los superiores de las funciones.

**Tercer Objetivo:** Facilitar la utilización de la biblioteca a cualquier persona que no entienda de programación ni de desarrollo de software en general, ya que se desea que llegue a la mayor cantidad de usuarios posible.

### Estructura de la memoria

**Estado del arte:** En este capítulo se muestra información sobre las distintas áreas de investigación y las interacción entre estas, esta información ha sido extraída de [5]

**Generación procedural de contenidos:** En este capítulo se habla de los distintos tipos de generación procedural, los algoritmos usados para esta y la representación de los distintos espacios de búsqueda además de las funciones de evaluación, esta información ha sido sacada de [5][7][1][2][3][4][12][9].

**Motores de Juego:** En este capítulo se habla sobre lo que es un motor de juego y sus componentes, además de explicar como se elige este y se hace una elección para el proyecto, esta información ha sido sacada de [8][6][10].

# Capítulo 1

# Estado del arte

## 1.1. Los métodos de perspectiva computacional:

Los principales métodos usados en este campo son algoritmos evolutivos, aprendizaje reforzado, aprendizaje supervisado, aprendizaje no supervisado y búsquedas y planificación en árboles.

El aprendizaje supervisado se basa en un modelo de estancias en conjuntos de datos con el objetivo de evaluar las clases, los algoritmos más usados son ID3 o aprendizaje con árboles, redes neuronales y máquinas de soporte vectorial.

En aprendizaje no supervisado se usan búsquedas de patrones, es decir en bases de datos que no tienen valores que nos aporten información directa, los algoritmos más utilizados son el K-medias y auto-organización de mapas.

Los algoritmos evolutivos van dirigidos a la optimización estocástica de la generación de la base del juego, los más utilizados son colonias de hormigas y algoritmos genéticos.

En 1.1 se muestran las diferentes relaciones entre las investigaciones nombradas anteriormente y los algoritmos usados en esa investigación[5].

| | NPC | S&P | PM | PCG | CN | BA | AI-Ass. | GGAI | Com.AI | Total(Dominant) |
|---|---|---|---|---|---|---|---|---|---|---|
| Evolutionary Computation | ● | | ● | ● | | ❍ | ● | ❍ | | 6(4) |
| Reinforcement Learning | ● | | | | | ❍ | | ● | | 3(1) |
| Supervised Learning | ❍ | | ● | | | ● | | | ● | 4(2) |
| Unsupervised Learning | | | ● | | ❍ | | ❍ | | | 3(1) |
| Planning | | ● | | ❍ | ● | ● | ● | | ● | 6(5) |
| Tree Search | | ● | | | ❍ | ❍ | ❍ | ● | ● | 6(3) |
| Total(Dominant) | 3(2) | 2(2) | 3(3) | 2(1) | | 3(1) | 5(2) | 4(2) | 3(1) | 3(2) |

Tabla 1.1: Relaciones entre investigaciones, extraído de [5]

### Perspectiva del jugador:

La perspectiva del jugador se divide en tres dimensiones principales.

La primera dimensión se basa en "¿qué puede hacer el jugador con el juego?" y podemos dividir a su vez, esta dimensión en tres clases, El modelo, generar o evaluar y usuario.

- El modelo: Se puede generar gracias a las redes neuronales o a los algoritmos genéticos un modelo de juego[5].

- Generar o evaluar: Se basa en que hay que generar o evaluar en el juego, se pueden usar algoritmos para mejorar el juego mientras el jugador esta en interacción con este[5].

- Usuario: En esta dimensión se estudia la generación de personajes y el diseño de estos[5].

## 1.2. Interacción entre las investigaciones:

Hay varios tipos de influencias, las influencias fuertes y las influencias débiles, por otra parte hay algunas influencias que son bidireccionales, mientras otras son unidireccionales[5].

Uno de los principales nodos de influencia es el aprendizaje del comportamiento del NPC, este comportamiento es aprendido por la IA general del juego y por la interacción del jugador con el juego, por otra parte gracias a ese aprendizaje del comportamiento se genera contenido del juego y modelado de los personajes ya sean NPC o que se modifique el propio personaje que usa el jugador.

Por otra parte la planificación y búsquedas también ayudan a la generación de agentes e historias, además de contenido para el juego.

En la tabla 1.2 aparece la interacción entre las distintas investigaciones:

| | NPC | S&P | PM | AI Bench. | PCG | CN | BA | AI-Ass. | GGAI | Com.AI |
|---|---|---|---|---|---|---|---|---|---|---|
| NPC | - | | ❍ | ❍ | ❍ | | ❍ | ❍ | ● | |
| S&P | ❍ | - | | ❍ | ❍ | ● | ● | | ● | ❍ |
| PM | | | - | | ❍ | ❍ | ❍ | ★ | | ❍ |
| AI Bench. | ● | ❍ | ❍ | - | ❍ | | ❍ | | ● | ★ |
| PCG | ❍ | ★ | | ❍ | - | | | ❍ | ★ | ❍ |
| CN | | | | | ❍ | - | ★ | | | ❍ |
| BA | | | ❍ | ❍ | | ❍ | - | | | ★ |
| AI-Ass | | | | | ★ | ★ | ❍ | - | | ❍ |
| GGAI | ★ | ★ | ★ | ★ | ❍ | | | ★ | - | |
| Com.AI | ❍ | ❍ | ❍ | | ❍ | | | | | - |

Tabla 1.2: Relaciones entre investigaciones, extraído de [5]

## 1.3. Elección:

El proyecto estará basado en la "AI-ASSISTED GAME DESIGN" para el diseño del juego y principalmente basado en el diseño de ciudades o de mundos, según el juego, es decir como se ha mostrado anteriormente, algoritmos evolutivos para la generación de ciudades o mundos en videojuegos.

## 1.4. Mejora:

La mejora de mi proyecto ante el resto está basada principalmente en la facilidad de uso para que cualquier desarrollador pueda utilizarlo en su juego y en la rapidez de construcción de ciudades, ya que algoritmos complejos mencionados anteriormente, pueden dar mejores resultados, en un tiempo y con una complicadeza de uso mucho mayor.

# Capítulo 2

# Análisis del problema

La generación procedural de contenidos, se refiere a la creación de contenido del juego automáticamente, hay distintos aspectos que se pueden hacer con esta generación de contenidos[5]:

- Actuaciones de NPC
- Mapas y niveles
- Historias y diálogos
- Misiones
- Armas
- Personajes

Como se ha mencionado anteriormente, nos centraremos en la creación de contenido procedural de mapas y niveles, concretamente el proyecto irá centrado en la creación de mundos en 2D con ciudades que podrán verse en 3D al acercarte a estas.

Hay varias distinciones que hay que realizar para la generación de contenidos[7]:

- una de las más importantes, es si el juego es online u offline[7].
- Debemos de comprobar si el contenido generado es opcional o es necesario para el juego, ya que si es necesario debemos realizar algoritmos con mayor optimización que si es un contenido opcional[7].
- Debemos tener en cuenta la utilización de semillas aleatorias o de vectores con parámetros, ya que de ello dependerá el tipo de generación que queramos obtener[7].

- La elección de una generación estocástica o determinística dependerá de si queremos que el algoritmo nos genere una misma solución con distintos parámetros[7].
- Debemos de tener en cuenta si queremos generar de forma constructiva o generar y hacer un test en el que se mostraría la validez del resultado[7].

## 2.1. Algoritmos de generación de contenido procedural:

Estos algoritmos se bansan en la idea de generación y test de para generación procedural de contenidos, siendo el test es una función que acepta o rechaza el candidato y esto se almacena en un vector denominado fitness y también cuenta con un valor que se le ha asignado al nuevo candidato al generarse, como puede verse en la figura 2.1[1].

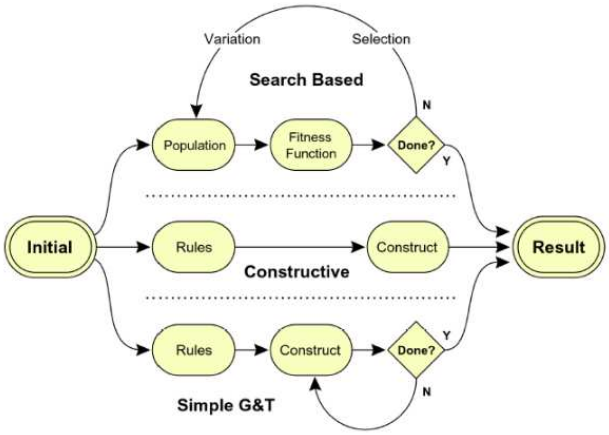


Figura 2.1: Generación de reglas, extraído de [1]

Para la generación procedural son necesarias unas reglas que vayan asociadas a las mecánicas de juego, por ejemplo[1]:

- **Hom and Marks** generaron unas reglas que eran representadas en el juego y estaban basadas en una función de evaluación de simulación estática y la teoría impulsada.
- **Browne** creó un sistema de diseño de reglas para direccionamiento usando programación genética, las reglas del juego estaban representadas en árboles, el lenguaje fue formulado para el proyecto.

En la figura 2.2 podemos ver como se generan las reglas del juego:

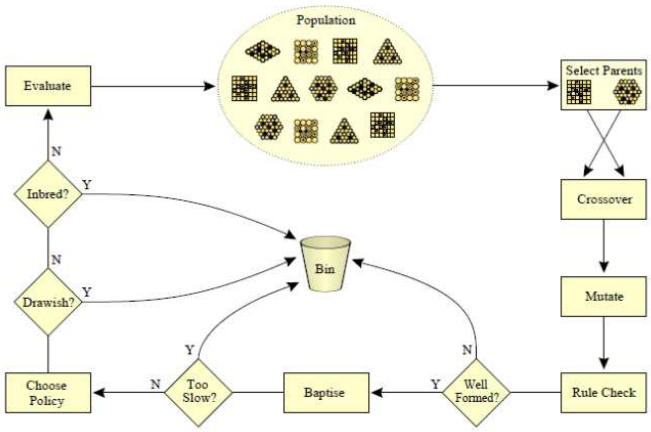


Figura 2.2: Generación de reglas, extraído de [1]

## 2.2. Representación del espacio de búsqueda:

Una distinción importante en la representación es entre la codificación directa o indirecta ya que una codificación directa implica una relativa simpleza computacional en el mapeo del genotipo-fenotipo, dado que en una computación compleja es necesario la creación del fenotipo desde el genotipo[4][12].

En nuestro caso, una forma de representar el terreno es en forma de expresiones de árboles.

## 2.3. Funciones de evaluación:

La evaluación es importante para ver las capacidades del algoritmo, para confirmar con garantía lo que puede crear y para comparar el contenido generado con otros contenidos[9].

Para diseñar la función de evaluación el diseñador debe primero decidir cuánto quiere optimizar para fortalecer el diseño.[9]

Tres tipos de evaluación son:

- Evaluación directa por funciones: en la que directamente una función saca el valor del fitness del contenido generado[9].
- Evaluación simulada por funciones: en el que el contenido se va manipulando y se va evaluando[9].

- Evaluación interactiva: mientras el juego está en ejecución se evalúa la interacción con el personaje y se van generando los contenidos[9].

Respecto a nuestro problema si realizamos una evaluación en cada punto que se determina mediante la consulta de árboles de expresión que van evolucionando y sustitimos las coordenadas de la posición actual por coordenadas de las constantes del árbol, tendríamos una buena evaluación para la generación de mapas, esta forma está basada en la **teoría de accesibilidad** en la cual se evalúa el terreno en función de la pendiente.

Para elegir las métricas apropiadas para la medición de la construcción es necesario esforzarse en la elección de medidas lo más lejanas posibles de los parámetros de entrada al sistema.

## 2.4. Alternativas a los algoritmos:

Las alternativas que hay actualmente para generación de contenido en videojuegos son las siguientes:

- Realización manual: en este caso, diseñadores se encargarán de realizar el contenido a mano, el costo de este trabajo es enorme y la empresa deberá de tener a varios trabajadores ocupando estos puestos de trabajo, pero la realización de este contenido es mucho mejor que la producida mediante algoritmos, aunque esta última no es la misma en todas las partidas.
- Realización aleatoria completa: en este caso, los algoritmos no controlan la aleatoriedad del contenido generado, por lo que se puede producir cualquier anomalía, pueden aparecer edificios flotando, partes de edificios separadas o incluso historias sin sentido en el caso de generación de historias.

# Capítulo 3

# Diseño de algoritmos

Un motor de juego, es una herramienta software diseñada para asistir a los desarrolladores en la tarea de creación de juegos y aplicaciones gráficas[8].

## 3.1. Arquitectura:

Un motor de juego está compuesto por una capa de tarjeta Hardware que representa el pc o consola sobre el que se ejecutará, una capa de drivers que son componentes Software que proporciona el sistema operativo o el proveedor de HArdware.

Por otra parte tiene el sistema operativo sobre el que está ejecutándose el juego todo el tiempo, la parte de SDKs que da soporte a la parte de programación del juego y la parte de estructuras de datos y algoritmos para dar soporte a la programación también.

Si nos centramos en la parte estética del juego, nos encontramos con una parte gráfica, una parte de animación de personajes y modelos y una parte de colisiones y física en la que se deforman los escenarios y da realismo al juego.

Sin embargo, si nos fijamos en la parte de la inteligencia artifical, nos encontramos una parte dedicada especialmente a eso, que está producida por el SDK.

Por último nos encontramos una serie de capas que ayudan a la depuración, renderizado, efectos visuales y gameplay para realizar pruebas.

En la 3.1 podemos ver como se organizan las distintas partes.

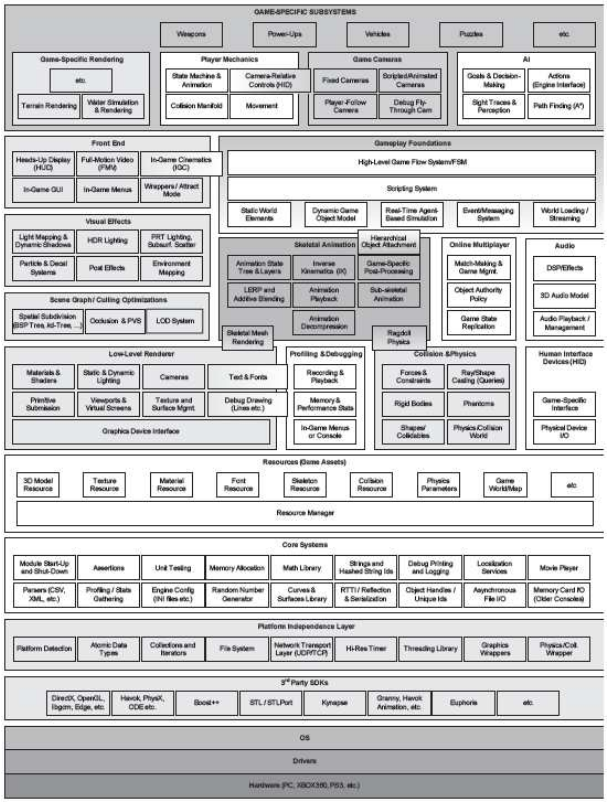


Figura 3.1: Arquitectura de un motor de juegos, extraído de [6]

## 3.2. Selección de motor:

Hay cientos de motores gráficos en el mercado y a la hora de elegir uno para el desarrollo de un proyecto hay que tener en cuenta los siguientes criterios[8]:

- Si la herramienta tiene soporte o no para 2D y 3D.
- Si la herramienta es multiplataforma.
- Los lenguajes de programación que soporta la herramienta.
- Si tiene un motor de físicas y herramientas de inteligencia artificial.
- Si incorpora herramientas más avanzadas como editores de terrenos.
- El tipo de licencia que utiliza.

En la tabla 3.1 están los tres motores de juegos más importantes del mercado.

Una vez que sabemos los criterios y los motores más usados del mercado, podemos disponernos a elegir un motor para nuestro proyecto.

Como el proyecto, tiene una parte en 2D, que es la de generación de planetas, necesitamos un motor que disponga de 2D, los cuales son los tres, sin embargo la parte

| Nombre | 2D/3D | IDE | Lenguaje | Plataforma | Motor Físico | Herramientas de IA | Características Avanzadas | Licencia |
|---|---|---|---|---|---|---|---|---|
| Unity3D | 2D y 3D | Mac y Windows | C#, Boo, JS | Mac, Windows, Linux, iOS, Android, PS3, XB360, PSP, Wii, PSVita, Flash, Web | PhysX | NavMesh y Path Finding | Forward y Deferred rendering, Occlusion Culling, light mapping, light probing, LOD discreto, edición de terrenos, sistema de partículas | Comercial. Versión básica gratuita (desktop y web) y versiones PRO y plataformas móviles y consolas con un pago único sin royalties |
| UDK/Unreal Engine 4 | 2D y 3D | Sí | Unreal Script | Mac, Windows, iOS, Android, PS3, XB360, PSVita, Wii, Flash | PhysX | Path Finding | Deferred rendering, occlusion culling, parallax mapping, Per Object Motion Blur | Comercial. Versión libre y pago de royalties sobre producto vendido al superar un umbral de facturación. La versión Unreal Engine 3 dispone de más funcionalidades y una licencia comercial más cara. |
| Corona SDK | 2D | No | LUA | iOS, Android, Nook, Kindle Fire | Box2D | No. Deben ser programadas desde cero en LUA | Interfaz simple de Lua para la programación de física | Comercial. Versión Indie para desarrollar para una única plataforma (iOS o Android) y versión Pro para todas las plataformas. |

Tabla 3.1: Lista de Motores obtenida en el artículo: [8]

de generación de ciudades, se hará en 3D, por lo que será necesario que disponga de 3D también,por lo que descartaremos el Corona SDK.

Por otra parte, necesitamos algo que se haga en PC, así que nos sirven los dos motores que nos quedan, pero si nos fijamos en los lenguajes de programación, Unity tiene una mayor variedad respecto a UDK que sólo soporta "Unreal script".

Si nos fijamos en la parte de motor de físicas, que es un motor que simula los movimientos físicos [10], vemos que es una parte irrelevante, ya que no vamos a mover nada, ni vamos a generar movimientos sismicos, ni de personajes.

Por otra parte si nos fijamos en la parte de la inteligencia artificial, vemos que Unity tiene una mayor variedad que UDK, lo que para el proyecto nos beneficia bastante para la programación de los algoritmos genéticos, lo mismo que pasa con las características avanzadas.

Si nos fijamos en el punto de licencias, Unity cuenta con una versión gratuita para la plataforma de PC, lo que nos viene perfecto, sin embargo, UDK para ofrecernos lo mismo que Unity requiere de una subcripción.

Una vez estudiado todas las características que nos proporcionan los motores que más se usan en el mercado, podemos decir que Unity es el que más nos beneficiará para el desarrollo del proyecto.

## 3.3. Selección de software de Modelado:

Como de motor hemos seleccionado Unity, nos centramos en los programas de modelado que son compatible con Unity y vemos que existen los siguientes:

- 3D Studio Max: este es un software bastante potente, acepta lenguajes de programación para generar algoritmos, pero con una licencia bastante cara, por lo que lo descartamos.
- Blender: software libre de modelado en 3D, es de los más utilizados, pero está capado en la geeración de algoritmos, por lo que no nos interesa.
- Autodesk Maya 3D: este es un software bastante completo, acepta su propio lenguaje de programación "Mel" pero también acepta algoritmos en Python, la licencia es bastante cara, pero al ser estudiante, Autodesk te regala una licencia de tres años, por lo que realizaremos los algoritmos para este software.

## 3.4. Algoritmos:

Con las herramientas mencionadas en los puntos anteriores, se han generado tres algoritmos que generan ciudades para la biblioteca, dos de ellos se han programado en Mel y otro en Python para comparar.

El primer algoritmo realizado en Mel y el tercero realizado en Python son muy similares, mientras que el segundo, realizado en MEl, es el algoritmo más complejo y al que dedicaremos más tiempo a su explicación.

### 3.4.1. Primer algoritmo:

Este algoritmo tiene una estructura bastante simple, está divido en tres partes, la primera comprueba si el GUI está creado y si no, crea uno nuevo donde leer los parámetros para generar las ciudades. La segunda parte es donde se genera la ciudad, en esta parte se utilizan los parámetros leídos anteriormente para recorrer una matriz e ir creando edificios aleatorios que se van reescalando para que todos tengan un mismo tamaño y la ciudad quede uniforme, por otro lado, la última parte realiza una limpieza de memoria para que no ocupe mucho espacio en memoria. En las figuras 3.2 y 3.3, podemos ver la GUI de este algoritmo y una ejecución.

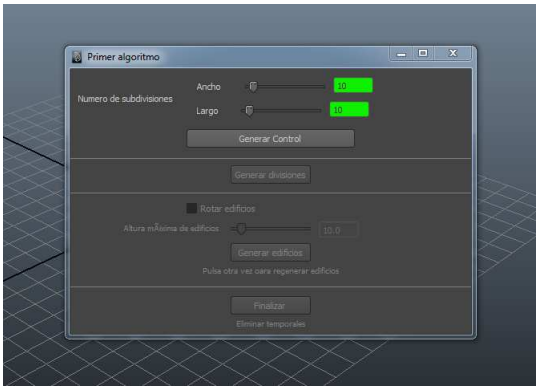


Figura 3.2: Interfaz gráfica del primer algoritmo.

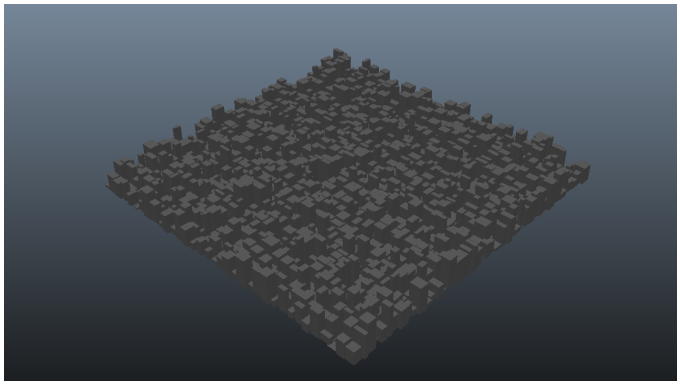

Figura 3.3: Ejecución del primer algoritmo

### 3.4.2. Segundo algoritmo:

Este algoritmo es el más sofisticado y complejo, consta de los siguientes pasos:

- En primer lugar comprobamos la versión de Autodesk Maya, ya que según la versión, utiliza unas aristas u otras.

- Tras el primer paso, comprobamos si no existe la ventana y si no existe, la crea, con dos botones que son princiales, en el primero genera la malla sobre la que se construirán los edificios, mientras que en la segunda se construyen los edificios.
- Para cada edificio, se genera un cuadrado principal, se separa aleatoriamente en plantas, siendo la distancia la misma en cada uno, se incluyen rectángulos para generar ventanas de forma seudo-aleatoria, se le coloca un cuadrado (o varios) en la parte superior y un cilindro con radio cero en uno de los extremos, se le inserta como antena si se ha seleccionado en el menú. Los edificios suelen ocupar un cuadrado de la malla, aunque si se desea se le puede cambiar el parámetro, para que todos tengan un tamaño parecido y no se quede estéticamente mal, se realiza como en los otros algoritmos un reescalado.

En las figuras 3.4 y 3.5 se muestra la interfaz y una ejecución realizada con el algoritmo.

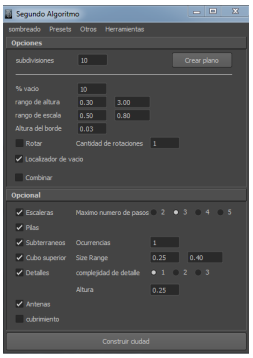


Figura 3.4: Interfaz gráfica del segundo algoritmo.

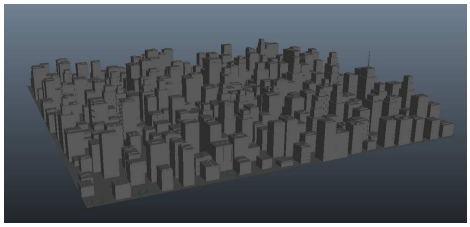

Figura 3.5: Ejecución del segundo algoritmo

### 3.4.3. Tercer algoritmo:

Este algoritmo es parecido al primero, tiene una estructura similar aunque varía la forma de generar las ciudades, ya que en este caso los edificios son leídos de un fihcero y se van insertando en la matriz con un random, de manera que si se modifica el fichero, cualquier persona podrá generar los edificios que vea necesarios en su juego, esto se puede realizar de una maenra bastante simple, mientras que en el primer algoritmo no ocurría eso. La primera y tercera parte si son iguales, dado que se comprueba si la GUI existe y en caso de que no exista se abre y se obtienen los valores introducidos de esta y en la tercera se limpia la memoria para eliminar temporales. En las figuras 3.6 y 3.7 podemos ver la GUI y una ejecución de este algoritmo.


Figura 3.6: Interfaz gráfica del tercer algoritmo.

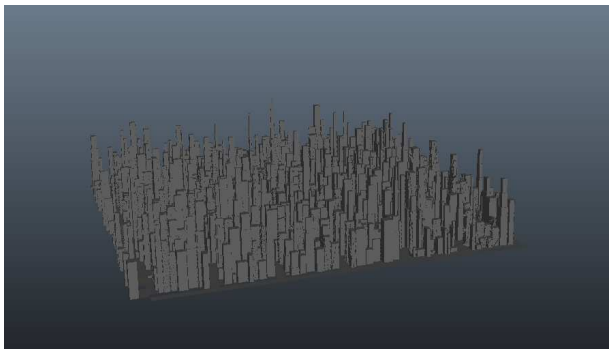

Figura 3.7: Ejecución del tercer algoritmo

# Capítulo 4

# Análisis de los algoritmos

En este capítulo nos centraremos en el análisis de los tres algoritmos explicados en el anterior, para ello haremos un anáisis a posteriori en un ordenador con las características mostradas en la tabla 4.1:

| | |
|---|---|
| Procesador | Intel Core i7-2600 3.4GHz |
| Memoria RAM | 32Gb DDR3 |
| Disco Duro | Samsung SSD 850 EVO 120Gb |
| Sistema operativo. | Windows 7 profesional 64-bits |
| Tarjeta gráfica | Nvidia GeForce GT-430 |

Tabla 4.1: Características del ordenador

## 4.1. Tiempos de ejecución:

Para los algoritmos hemos realizado distintas ejecuciones con valores de la forma:

- 10x10
- 20x20
- 30x30
- 40x40
- 50x50
- 60x60
- 70x70
- 80x80
- 90x90
- 100x100

Si nos damos cuenta, estamos realizando divisiones cada vez mayores para comprobar como aumenta el tiempo en cada algoritmo. Tras las distintas ejecuciones nos sale lo mostrado en 4.2

| Algoritmo | 10x10 | 20x20 | 30x30 | 40x40 | 50x50 | 60x60 | 70x70 | 80x80 | 90x90 | 100x100 |
|---|---|---|---|---|---|---|---|---|---|---|
| Primer algoritmo | 2.05 | 3.32 | 4.34 | 4.65 | 5.06 | 7.50 | 11.65 | 16.49 | 27.31 | 44.67 |
| Segundo algoritmo | 9.47 | 24.70 | 67.09 | 155.33 | 278.37 | 446,81 | 1144.14 | 1240.22 | 1500.64 | 1924.82 |
| Tercer algoritmo | 1.09 | 1.23 | 1.34 | 5.01 | 10.14 | 13.9 | 28.21 | 30.64 | 36.88 | 64.91 |

Tabla 4.2: Tiempo de ejecución (s) de los algoritmos para los distintas divisiones.

Como podemos ver, el tiempo en el segundo algoritmo es mayor, también se ha explicado anteriormente que es el más complejo, por otra parte, si representamos esos datos en gráficas de R nos daría lo mostrado en la figura 4.1.

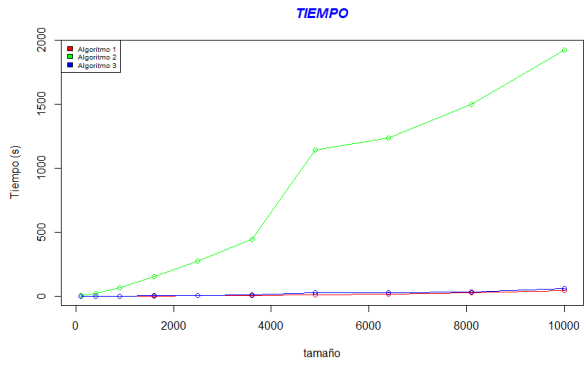


Figura 4.1: Representación gráfica en R de los tiempos de ejecución.

Con los datos representados, es trivial ver que los algoritmos, son de la forma $\mathbf{O}(n)$. Para mostrarlo mejor, en la figura 4.2 podemos ver las gráficas de los algoritmos además del gráfico de $O(n)$ y de $O(n^2)$.

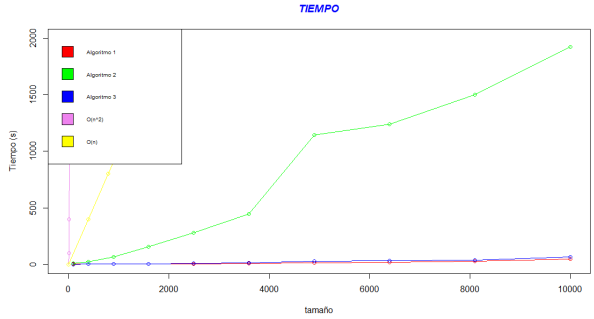


Figura 4.2: Comparación de tiempos con $O(n)$ y $O(n^2)$,realizada en R.

## 4.2. Memoria:

Para el estudio de la memoria utilizada por cada algoritmo hemos realizado la misma cantidad de ejecuciones con el mismo tamaño, dando los resultados indicados en 4.3

| Algoritmo | 10x10 | 20x20 | 30x30 | 40x40 | 50x50 | 60x60 | 70x70 | 80x80 | 90x90 | 100x100 |
|---|---|---|---|---|---|---|---|---|---|---|
| Primer algoritmo | 1.81 | 1.85 | 1.87 | 1.95 | 2.11 | 2.23 | 2.38 | 2.53 | 2.64 | 2.82 |
| Segundo algoritmo | 2.27 | 3.81 | 5.12 | 7.77 | 10.9 | 14.1 | 19.4 | 24.1 | 29.6 | 31.8 |
| Tercer algoritmo | 0.5 | 0.68 | 0.80 | 1.01 | 1.25 | 1.50 | 1.93 | 2.12 | 2.28 | 2.50 |

Tabla 4.3: Memoria RAM (Gb) utilizada en cada ejecución de los distintos algoritmos.

Por otro lado hemos analizado la memoria que utiliza cada uno de los algoritmos en las distintas ejecuciones para que el usuario que vaya a trabajar con la biblioteca, pueda elegir que algoritmo le conviene más. En la figura 4.3 podemos ver la cantidad de memoria RAM utilizada por cada algoritmo.

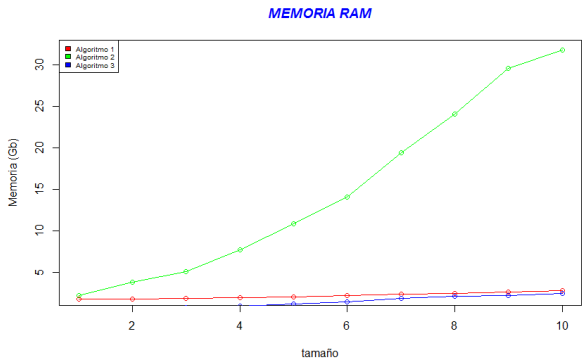


Figura 4.3: Memoria RAM (Gb) utilizada por los distintos algoritmos según ejecución, realizada en R.

Como podemos ver, la cantidad de memoria utilizada por el segundo algoritmo, es mucho mayor, deja al ordenador casi sin memoria en la última ejecución, mientras que el primero y el tercero usan una cantidad de memoria bastante razonable y que se puede usar en cualquier juego.

## 4.3. Conclusiones:

Las conclusiones obtenidas de los últimos puntos son:

Los algoritmos primero y tercero pueden ser utilizados en cualquier juego, realizando tamaños de ciudades tan grandes como se deseen, mientras que el segundo algoritmo,

sólo se puede utilizar en tamaños pequeños, si deseamos usarlo en cualquier juego, en el caso de querer ciudades grandes y con tanto detalle como este algoritmo nos las muestra, el juego tendría que tener calidad, y los requisitos mínimos para el funcionamiento de este serían bastante mayores.

# Capítulo 5

# Conclusiones y trabajo futuro

Los objetivos de este trabajo eran, la contrucción de una biblioteca para generación procedural de ciudades que fuese posible usar por cualquier persona, e importar las ciudades al motor de juegos Unity 3D para que sea posible usarlo en cualquier juego que se cree con este motor. Tras la realización de este trabajo, hemos llegado a la conclusión de que los algoritmos realizados son bastante competitivos con los que hay en el mercado, aunque el segundo algoritmo en ciudades muy grandes no se podría usar en cualquier juego, dado su coste.

El trabajo futuro que se puede añadir a este proyecto sería por una parte la construcción definitiva de algoritmos genéticos ya que los algoritmos actuales son de una única iteracción, y comprobar con costes cual de las ciudades que van saliendo con la mezcla de genes "edificios", para obtener ciudades aún más competitivas, por otra parte este algoritmo se usará para la generación de varios juegos que se encuentran actualmente en desarrollo y que esperemos que el año que viene uno de ellos se encuentre en la plataforma de videojuegos "STEAM", para su venta y distribución. Todo el trabajo realizado se encuentra en GitHub en [11].

# Bibliografía


[1] Lee C. McGuinness C. Ashlock, D. Search based procedural generation of maze-like levels. 2011. III, III, IX, 6, 7

[2] Maire F Browne, C. Evolutionary game design. ieee transactions on computational intelligence in ai and games. 2010. IX

[3] Pollack J.B Bucci, A. On identifying global optima in cooperative coevolution. 2005. IX

[4] Yannakakis G.N. Togelius J. Lanzi P.L Cardamone, L. Evolving interesting maps for a first person shooter. 2011. IX, 7

[5] IEEE Georgios N. Yannakakis, Member and IEEE Julian Togelius, Member. A panorama of artificial and computational intelligence in games. V, V, VIII, IX, IX, 1, 2, 3, 5

[6] Jason Gregory. *Game engine architecture*. III, IX, 10

[7] G. S. Hornby and J. B. Pollack. The advantages of generative grammatical encodings for physical design. 2001. IX, 5, 6

[8] Juan Vicente Martínez Pérez y Arturo Candela Moltó Jordi Llinares Pellicer. Gráficos a la máxima potencia: una comparativa entre motores de juegos. *3ciencias*, 2012. V, IX, 9, 10, 11

[9] Simon Colton Michael Cook and Jeremy Gow. Initial results from co-operative co-evolution for automated platformer design. IX, 7, 8

[10] Ian Millington. *Game physics development*. IX, 12

[11] Pedro Luis Trigueros Mondéjar. `https://github.com/Milpoder/PFG-ContentGeneration`, 2015. 21

[12] Gillian Smith Noor Shaker and Georgios N. Yannakakis. Evaluating content generators. IX, 7

[13] Simon Belli y Cristian López Raventós. *Breve historia de los videojuegos*. 2008. VII