Universidad de las Ciencias Informáticas

Facultad 5

Título: “Generación de entornos de interiores en tiempo de ejecución”

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores:

Carlos Manuel Valdivia Marín.

Yudel Martínez Hernández

Tutor:

Ing. Omar Correa Madrigal

Declaración de autoría.

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ___ días del mes de _____del año _____.

_____ _____

Yudel Martínez Hernández. Carlos Manuel Valdivia Marín.

Autor. Autor.

_____ _____

Ing. Omar Correa Madrigal. Ing. Yenifer Del Valle Guevara.

Tutor. Tutor Asesor.

Agradecimientos

Le agradezco a mi mamá que es lo que más quiero en este mundo por todo el cariño y amor que me ha brindado a lo largo de mi vida y por estar presente en todo momento, por su preocupación incondicional y por la buena educación que siempre me ha proporcionado. A mi papa que siempre ha sido mi mejor amigo, mi confidente, gracias por darme tu amor paternal y por confiar siempre en mí, por ser mí conciencia, mi pepe grillo, por darme tus infinitos consejos que tanto me han ayudado a superar las dificultades que me ha impuesto la vida. Les agradezco también a mi tía y a mi abuela , mis dos viejitas lindas que tanto se han preocupado por mí , por mi bienestar, gracias por estar siempre tan pendientes de mi educación y de mi vida, por todo el cariño y amor que me han dado ,y por enseñarme lo grande e importante que es la familia. También quisiera agradecerles a mi abuela Flora y a mi abuelo Yeyo, que sin ser mis verdaderos abuelos los quiero como tal, por estar siempre a mi lado desde los comienzos de mi vida, gracias por quererme como a un nieto más. A mis compañeros, que me han ayudado a superar las dificultades a lo largo de la carrera y por los buenos momentos que compartimos juntos, en especial a Yudel, mi compañero de tesis, por los largos debates profesionales y por su excelente trabajo a lo largo de la tesis.

Carlos Manuel.

Agradecimientos

Me gustaría agradecerles por su ayuda en la realización de este trabajo y por la ayuda brindada a lo largo de los 5 años a todos mis compañeros de aula o apartamento, la primera vez que escribí esto trate de poner todos los nombres y todos los días recordaba a alguien más por lo que desistí y simplemente diré que a todos ellos muchísimas gracias. A Carlos que trabajó muy duro y además no me dejó hacerlo a mi manera que obviamente no era nada buena. También deseo agradecerle a todos los profesores por intentar transmitirme sus conocimientos, en especial a la profe Lida. Un agradecimiento especial a nuestro tutor Omar por la ayuda que nos brindó. A mis 5 súper amigos en Santiago por todos los momentos vividos. Y sobre todas las cosas a mi hermanito, mi tía, mi mamá y mi abuela que son lo más grande que tengo en este momento. Y a mi abuelo que gracias a él estoy aquí por su dedicación y determinación a convertirme en alguien.

Yudel.

Dedicatoria

A mis padres por brindarme su cariño y amor incondicional.

Carlos Manuel

A mi abuelo por notar que los buenos no nacen, se hacen y a Adisleidis que pronto será una estrella.

Yudel

Resumen:

Como resultado de la investigación realizada, en el siguiente trabajo se propone una biblioteca de generación de entornos de interiores en tiempo de ejecución utilizando para ello un sistema de gramática de grafo. Para el uso del mismo el usuario debe definir en un fichero los símbolos y las reglas de producción a utilizar para confeccionar la gramática. Este proceso fue automatizado mediante la creación de un editor de gramáticas el cual exporta el fichero que utiliza el generador para leer la gramática, esta también constituye una forma de garantizar la integridad de dichos datos. Con esta gramática el generador crea de forma infinita un grafo donde cada nodo representa un símbolo de la gramática, pero el grafo no crece símbolo a símbolo sino de regla de producción en regla de producción lo cual permite que los elementos definidos como sucesión en la gramática también constituirán una sucesión en el grafo. Para la visualización de estos elementos se confeccionó un sistema genérico que permite la independencia del motor gráfico utilizado, lo cual demuestra su flexibilidad, este sistema de visualización se encarga de ubicar los elementos en el entorno mediante una estructura de capas las cuales son contenedoras de los modelos tridimensionales o contenidos de la escena. Este sistema de capas posee una estrategia que conjuga el reemplazo de los contenidos en las capas, con la correcta traslación y rotación de los mismos en el sentido del desplazamiento del usuario, calculando con precisión las próximas ubicaciones de las capas brindando en consecuencia una mayor sensación de realismo.

Índice

Introducción

En el amplio mundo de los videojuegos se han desarrollado numerosas técnicas para que los juegos cada vez sean más entretenidos e interactivos para los jugadores, esto se debe a la gran variedad de modalidades de videojuegos que existen en el mundo actual y el gran auge que ha alcanzado esta rama de la realidad virtual en estos momentos.

Los videojuegos se han hecho muy populares porque además de brindar un amplio entretenimiento donde podemos despejar nuestras mentes, recrearnos y ampliar nuestras habilidades, pueden ser empleados con otros fines como tratamientos para diversas enfermedades, simuladores, entre otras aplicaciones que pueden resultar muy ventajosas.

Es por esta razón que la Universidad de Ciencias Informática (UCI) se ha propuesto adentrarse en esta gran industria que hoy representa un fuerte exponente en la economía mundial. Para ello se han creado numerosos proyectos en la facultad 5 perteneciente con el objetivo de explotar estas potencialidades.

Específicamente en el proyecto Juegos CNEURO que se dedica a la realización de juegos de rehabilitación, se desea conformar un juego que contenga entornos infinitos y con una gran variedad de contenidos, logrando que el usuario tenga la sensación de que está transitando por diferentes lugares del entorno.

Lo anterior hizo plantearse la siguiente interrogante. ¿Cómo podríamos lograr escenarios virtuales de interiores que sean variables y a su vez generados en tiempo de ejecución, para videojuegos? La investigación emprendida para lograr este objetivo se enmarca en las estructuras y técnicas utilizadas para la generación de entornos virtuales, más específicamente en las técnicas de generación de entornos interiores en tiempo de ejecución.

El objetivo del trabajo de diploma es elaborar una biblioteca para la Generación de Entornos Virtuales de interiores en tiempo de ejecución para videojuegos.

Con la culminación del mismo se debe lograr que los juegos realizados en el proyecto CNEURO sean más entretenidos debido al permanente cambio y combinación de los elementos del entorno(Adams, 2002), además se disminuiría el consumo de recursos tanto de software como de hardware que exigen este tipo

de entornos los cuales se tornan extensos y en consecuencia abarcan mayor espacio en la memoria de la máquina.

Con el objetivo de dar cumplimiento a lo anteriormente planteado se trazaron una serie de tareas investigativas las cuales se muestran a continuación:

- Identificar y sintetizar las principales características y técnicas de implementación de la generación de entornos virtuales de interiores para determinar cuál incluir en la solución.
- Definir la arquitectura que asumirá la biblioteca de generación de entornos de interiores.
- Desarrollar la biblioteca de generación de entornos de interiores.

En la investigación de la Generación de Entornos Virtuales de Interiores Infinitos se utilizarán algunos métodos científicos, centrados principalmente en los teóricos, los cuales servirán de guía a lo largo del desarrollo de este tema para solucionar el problema planteado.

Una de las formas en que se llevara a cabo dicha investigación es mediante el análisis de teorías y la búsqueda de la esencia de la problemática a resolver, por lo que se ha decidido utilizar el método ***Analítico – Sintético***. Este permitirá realizar una serie de estudios que aporten los conocimientos necesarios para introducir a fondo el tema referente a la Generación de Entornos Virtuales, para así determinar los algoritmos, técnicas y teorías más utilizadas y mejor argumentadas a nivel global.

Es conveniente emplear el método ***Inductivo – deductivo*** para luego de Inducir una serie de conocimientos referentes a la Generación de Entornos infinito en tiempo real poder arribar a razonamientos que conlleven a la deducción de conocimientos que puedan ser aplicables al problema a tratar en particular.

Otro de los métodos utilizados es el ***Análisis histórico lógico*** el cual permitirá analizar y estudiar la trayectoria y evolución de los Generadores de Entornos infinitos a través de la historia, para así poder contar con una noción de cuan desarrollado está el tema a nivel global.

Capítulo 1: Fundamentación teórica.

Introducción

Las técnicas de generación de entornos virtuales representan una gran ventaja para la realización de Videojuegos ya que automatiza la construcción de entonos virtuales apoyado en los gráficos por computadora. Esta técnica se ha utilizado en sus diferentes variantes por un gran número de aplicaciones como son los Sistemas Geoespaciales 3D, en visualización de clima y en ambientes topológicos y en diversos videojuegos, de los cuales podemos mencionar los SIMS, el World of Warcraft, Wolfenstein, Doom entre muchos otros, dando la posibilidad de crear grandes entornos, con una gran diversidad de contenidos y ambientes.

Esta técnica se ha utilizado en varios géneros de juegos como por ejemplo, los juegos de estrategia, como Civilización II. Al comenzar cada partida de este juego se generan nuevos entornos de forma aleatoria dándole al jugador una novedosa experiencia.

Para hacer esto los entornos se dividen en baldosas cuadradas donde cada una de ellas está compuesta de un determinado tipo de terreno tales como las praderas, los bosques o el mar. El tipo de terreno de cada baldosa se genera aleatoriamente de una forma realista y cuando estos se fusionan logran un nuevo entorno.(Madrigal, et al., 2009).Sin la aplicación de este tipo de técnicas, juegos como civilización II y otros similares se harían monótonos y se perdería el interés en ellos, ya que los jugadores encontrarían en todo momento el mismo entorno una y otra vez. Con el uso de esta técnica se logra construir un entorno de exteriores pues lo que se generan son bosques, praderas etc. pero no siempre el entorno estará constituido por esta clase de elementos en ocasiones también el desarrollo del juego será en el interior de un edificio o una casa por lo cual se dividirán las diferentes técnicas de generación de entornos en dos grupos para su mejor comprensión, la generación de entornos de exteriores y la generación de entornos de interiores(Alvarado., 2007)(Madrigal, et al., 2009). Ambas técnicas tienen como objetivo hacer más diversas las escenas que conforman un juego.

La generación de entornos virtuales se puede clasificar según el espacio de tiempo en que se realiza: en Pre procesamiento y en Tiempo de Ejecución(Alvarado., 2007). Ambas brindan grandes ventajas pero es necesario analizar cuál es la más conveniente en cada momento.

La generación de entorno en pre procesamiento consiste en generar entornos una vez que la aplicación sea ejecutada, pero después de esto, el entorno creado se mantendrá invariable hasta que la aplicación sea nuevamente ejecutada(Alvarado., 2007). Este tipo de generación es muy utilizado en juegos de estrategia donde una vez comenzada la partida es conveniente que el mapa creado no varíe, aunque sí, que al iniciar una nueva, brinde la posibilidad al jugador de explorar otro mapa diferente al anterior(Madrigal, et al., 2009).En el caso de la generación en tiempo de ejecución el entorno creado se irá transformando constantemente a medida que se valla transitando por el mismo, dando la sensación de que este es infinito. Este tipo de generación es muy utilizada en juegos donde el objetivo a cumplir no depende directamente del entorno, sino de otros factores como por ejemplo en el videojuego terapéutico Meteorix para el tratamiento de la ambliopía, donde se utiliza un entorno espacial que se genera de forma infinita, hasta que el jugador cumpla con requisitos médicos establecidos mediante un algoritmo.

Figura 1 Entorno de exteriores (juego SIMCity**).**

Figura 2 Ejemplo de entorno de interiores 1.

Figura 3 Ejemplo de entorno de interiores 2.

Figura 4 Ejemplo de entorno de interiores 3.

1.1. Generación de entornos exteriores.

1.1.1. Mapas de altura

Una de las variantes de generación de entornos de exteriores más utilizadas actualmente en los videojuegos es la generación de terrenos basados en Mapas de Alturas (Height Map)(Polack, 2003), donde existen varios algoritmos para desarrollar esta técnica, aunque algunos son más óptimos que otros. Entre estos podemos encontrar al algoritmo de fuerza bruta, al de técnica fractal de terreno, los de formación por fallas y el de desplazamiento del punto medio.

Un mapa de altura es un arreglo bidimensional de valores de altura organizados en una rejilla regular. En cada par (x, y) de la rejilla se almacena el valor de z (altura). Este valor coincide con la coloración en escala de grises del pixel (x , y) de la imagen que se desee, el cual se acota en el intervalo entre 0 que sería la altura más baja (color negro) del terreno y 255 la más alta (color blanco). Está imagen puede ser creada en cualquier editor que permita la creación de imágenes en escala de grises. El método es genérico y permite generar un número interminable de terrenos, solo es necesario proporcionar la imagen en escala de grises y las cotas de altura asociadas a los valores 0 y 255 respectivamente(Pubb, 2003).

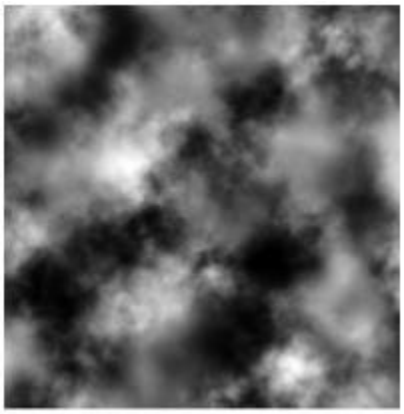

Figura 5 Mapa de altura con escala de grises.

Figura 6 Representación en OGRE del mapa de altura de la imagen en escala de grises **(Madrigal, Gutiérrez, & González, Generación de entornos virtuales en tiempo de ejecución., 2009)**.

Por otra parte el algoritmo de formación por fallas que utiliza la *Técnica de Generación fractal de terrenos* genera mapas de alturas potencias de 2 y con iguales dimensiones, por ejemplo 1024x1024, no se puede generar un mapa de alturas de 512x1024. Un fractal está definido como una forma similar a sí misma, a diferentes escalas. Probablemente el ejemplo más simple de forma fractal sea la curva de Koch, en la cual, una forma geométrica es subdividida en segmentos donde cada uno de ellos es remplazado con la forma inicial y así sucesivamente. Este algoritmo trabaja generando fallas en el terreno, en su mayoría para terrenos medianamente altos, con varias llanuras, paisajes suaves y sin grandes alturas. Un Fractal

puede ser generado a partir de procesos iterativos o recursivos. Se utilizan mayormente en la generación de árboles y terrenos.

Estas técnicas son óptimas según las características del terreno, para grandes entornos, consumiría muchos recursos utilizar estos algoritmos, cargar un terreno mediante un mapa de altura, o incluso la generación de un terreno en tiempo real para entornos muy extensos sería muy costoso.

1.1.2. Frustum Filling (Rellenado del volumen de visión)

Una de las estrategias más utilizadas en la Generación de entornos en tiempo real es el frustum filling (Rellenado del volumen de visión) ya que esta logra economizar los recursos de la PC obteniendo mejores resultados visuales. (Greuter, 2001)

La misión de esta técnica esta en rellenar constantemente el frustum con los contenidos que se vallan generando en el entorno, los cuales al salir del mismo, ya sea por la rotación o translación de la cámara serán eliminados, llevándose a cabo el mismo proceso nuevamente.

Esta estrategia puede ser utilizada en conjunto con cualquier otra técnica de generación, ya sea de exteriores o de interiores. Con la aplicación del frustum filling se han logrado generar diversos entornos como ciudades, paisajes y entornos espaciales.

Una vía para implementar el rellenado del volumen de visión es dividir la escena en celdas cuadradas mediante una rejilla de dos dimensiones. Cada celda representa un lugar donde se ubica un contenido (Greuter, 2001).Las celdas son organizadas en forma de cuadrado alrededor de la cámara, ubicada ésta, en el centro del mismo. Al trasladar la cámara, este cuadrado conformado por celdas es trasladado también, permaneciendo la cámara siempre en el centro.

Para su aplicación esta estrategia tiene que integrase con el sistema de visualización del motor gráfico que se esté utilizando, ya que antes de realizar el frustum culling es necesario realizar el frustum filling, lo cual supone un inconveniente porque modificar dicho sistema no es una tarea trivial. (Madrigal., 2010)

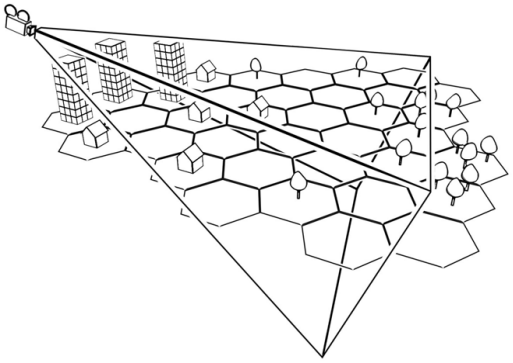

Figura 7 Aplicación del algoritmo frustum filling**(Greuter, 2001)**.

1.2. Generación de Entornos Interiores.

La técnica de generación para entornos de interiores es muy utilizada en juegos en primera persona y se caracteriza por crear escena de espacios cerrados como edificios y otro tipo de construcciones que contengan elementos como pasillos, cuartos etc., donde los jugadores puedan desplazarse libremente por el interior de las mismas, interactuando con estas y colisionando con puertas, paredes u objetos que interfieran en su camino.

Figura 8 Generación de entornos de interiores en el juego Wolfensteins.

Figura 9 Generación de entornos de interiores en el juego Doom.

1.2.1. Lazy Generation (Generación tardía)

La técnica de generación de interiores “Lazy Generation” se desarrolla en tiempo real y es construida de forma procedural, es decir algorítmicamente. Para lo cual no se utilizan modelos prediseñados, sino que se construyen las geometrías en tiempo de ejecución polígono a polígono .Este enfoque está destinado a ser utilizado en aplicaciones 3D como videojuegos, maquetas virtuales y simuladores. La intención es crear una aplicación que provea un entorno mucho más extenso que la memoria disponible y mucho más grande que los entornos que los diseñadores serian capaz de crear manualmente en una cantidad razonable de tiempo(Hahn, et al., 2006). Este tipo de trabajo está inspirado principalmente porque existen muchos juegos en los cuales los edificios no pueden ser atravesados libremente por el jugador. Para suprimir esta limitante se utiliza esta técnica comenzando su aplicación en el momento en que se intenta acceder al edificio y así se va creando el entorno interior a medida que vayamos caminando por él.

Esta estrategia utiliza solo la porción de memoria que el modelo que se está creando necesita, permitiendo ahorrar los recursos de la PC. Otra de las ventajas que se le atribuye es que todos los espacios generados son persistentes aunque se borren de la memoria pues se lleva un registro de los mismos.

Las construcciones se generan mediante el fraccionamiento de las regiones temporales en regiones más pequeñas de diversos tipos, las cuales son conectadas por portales. Se definen dos tipos principales de regiones que conforman el interior de los edificios: Las regiones temporales que no son más que las regiones del espacio donde la generación puede ocurrir, cuya implementación está representada por cajas alineadas al eje y las regiones de construcción (Hahn, et al., 2006), las cuales representan el producto visible final del generador y contienen las geometrías necesarias para la representación y la detección de colisiones, así como los objetos visibles que pueden ser colocados en el edificio.

Esta técnica de generación no depende de la trayectoria seguida, la idea es hacer que las regiones temporales siempre generen los mismos resultados independientemente de cualquier otra región.

En cada etapa de la generación, una región temporal será dividida en regiones temporales más pequeñas hasta convertirse en una región de construcción. Este proceso se aplicara en las regiones temporales hasta que solo sean visibles las regiones de construcción. Cada región temporal a ser generada no utilizara la información de otras ya existentes, trayendo como resultado que la generación sea coherente y que las regiones puedan ser generadas en cualquier orden, no afectando así el resultado final. Luego que cada región temporal sea dividida, se crean los nuevos portales para conectar las nuevas regiones. Es importante señalar que estos portales solo pueden ser conectados a las regiones recién creadas, ya que si se conectan a cualquier otra región los resultados no serían independientes del orden de generación de las regiones.

En esta estrategia es necesario utilizar puntos para limitar la generación de regiones. Dado un conjunto de regiones temporales y un punto, la generación sólo se llevará a cabo en la región que contiene el punto, esta se dividirá en regiones más pequeñas y el proceso continuará con la región que contenga el punto y solo se detendrá cuando este se encuentre dentro de una región de construcción.

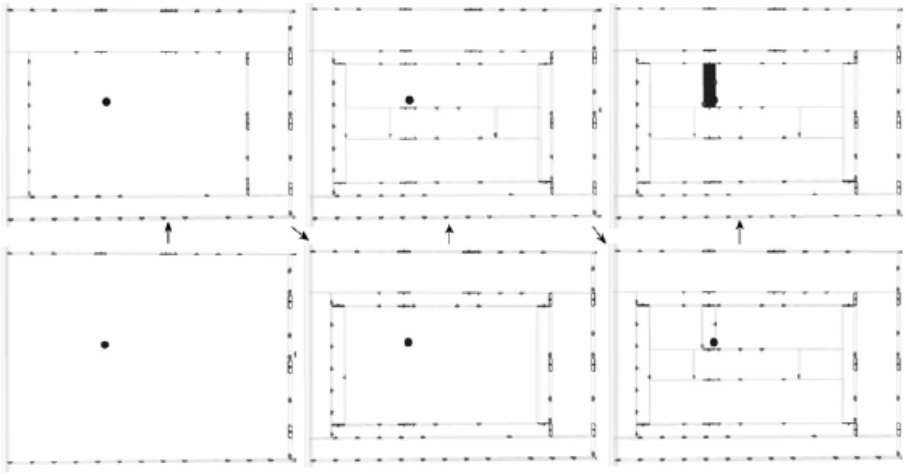

Figura 10 Generación de regiones por puntos**(Hahn, et al., 2006)**.

La variación de los contenidos se logra mediante la aplicación de un generador de números aleatorios donde estos darán siempre la misma secuencia de números al azar para una semilla dada. Cada región del edificio siempre utilizará la misma semilla por lo que se obtendrán la misma secuencia de números y por ende el resultado de la generación siempre será idéntico.

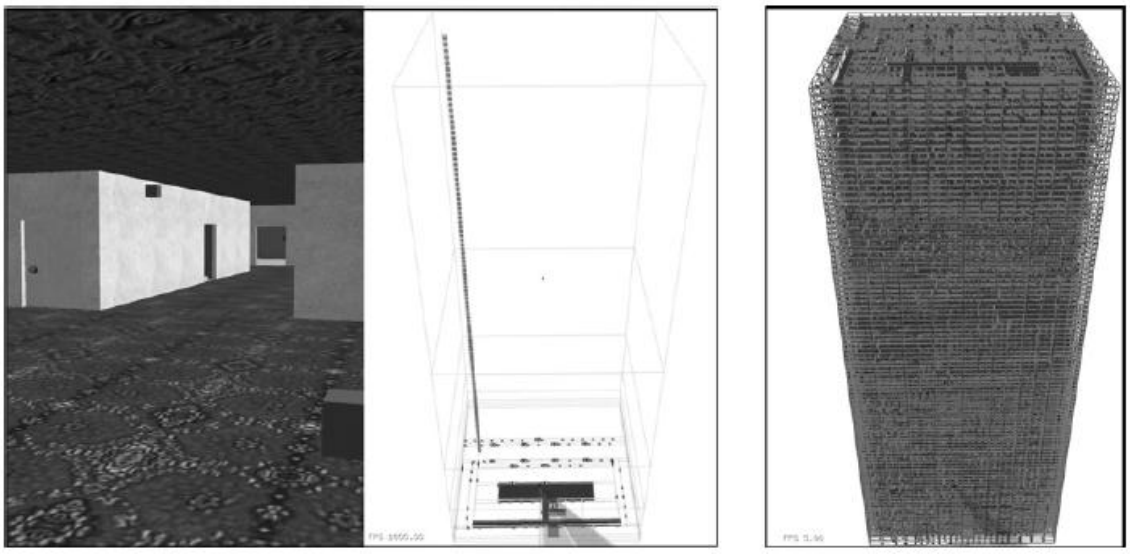

Figura 11 Resultados de "Lazy Generation"**(Hahn, et al., 2006)**.

Se puede decir que esta técnica posee como ventajas la velocidad, la aleatoriedad y la persistencia del entorno pero es complicada de implementar y no soporta la construcción de entornos a partir de modelos prefabricados.

1.2.2. Generación de Interiores basadas en gramáticas

Esta técnica primeramente genera las topologías del juego, que no son más que la secuencia ordenada de elementos que conformaran el entorno y luego son sustituidos por las geometrías que representan las diferentes topologías. Esto antes mencionado tiene una gran ventaja ya que no solo se queda dentro de las fronteras de la generación de terrenos sino que permite generar diferentes contenidos de ambientación , ya sean vidas , bonos , armas , adversarios etc. que son colocarlos en algún lugar del entorno. Dentro de los juegos que utiliza este tipo de técnica de generación podemos encontrar DarckForce y StartCraft. Un aspecto importante a tener en cuenta en este tipo de generación es la aleatoriedad de sus contenidos, tanto de ambientación como de creación de terrenos, la cual le da al juego un mayor grado de diversidad.

Para generar las topologías del juego se utilizan gramáticas, siendo la gramática libre de contexto las aplicadas más frecuentemente en este tipo de casos. Una gramática en general es un cuádruplo **G** = {**N**, **Σ**, **P**, **S**} donde:

N: Conjunto de símbolos no terminales.

Σ: Conjunto de símbolos terminales.

P: Conjunto de Reglas de Producción.

S: Axioma o símbolo distinguido.(Rozenberg, 1997)

El lenguaje generado por una gramática es el resultado de la aplicación de un número finito de reglas de producción en donde todos los elementos del mismo son terminales.

Las gramáticas libres de contexto permiten la expansión de sus no terminales sin estar sujeto a los elementos que la rodean, es decir el lado izquierdo de cada regla de producción solo admite un símbolo

no terminal, el cual puede derivar en su parte derecha cualquier combinación de símbolos terminales y no terminales, de la forma *A->aBc* donde *A* € N

Y *aBc* € (N ∪ Σ)*, algo que no ocurre con la regla: *xBy -> P* en donde *B* solo puede ser substituida por *P* si y solo si se encuentra entre *x* y *y*. (Adams, 2002)

Para la generación de interiores con gramáticas libres de contexto se puede utilizar la gramática de cadena, la cual es mayormente usada en la generación de contenidos de ambientación aunque también se puede usar para generar los terrenos del entorno. Este tipo de gramática solo admite que sus símbolos terminales tengan a lo sumo dos conexiones. A continuación se muestra un ejemplo de gramática de cadena aplicado a la generación de entornos de interiores:

Secciones -> cuarto – pasillo

Secciones -> cuarto – Secciones

Como se observa el lenguaje generado por esta gramática es limitado y cada elemento que la conforma solo tendrá acceso a otros dos elementos como máximo.

Existe otro variante dentro de la gramática libre de contexto que permite generar entornos de interiores, conocida como gramática de grafo, en las cuales un símbolo puede estar conectado con cuantos se desee, dependiendo de la regla de producción que sea aplicada. Los grafos proporcionan una manera natural de modelar las complejas redes que se encuentran en los juegos por lo que, esta técnica es muy utilizada para la generación de terrenos en interiores.

1.2.3. Gramática de Grafo

A continuación se expone un ejemplo de este tipo de gramática aplicado a la generación de entornos de interiores:

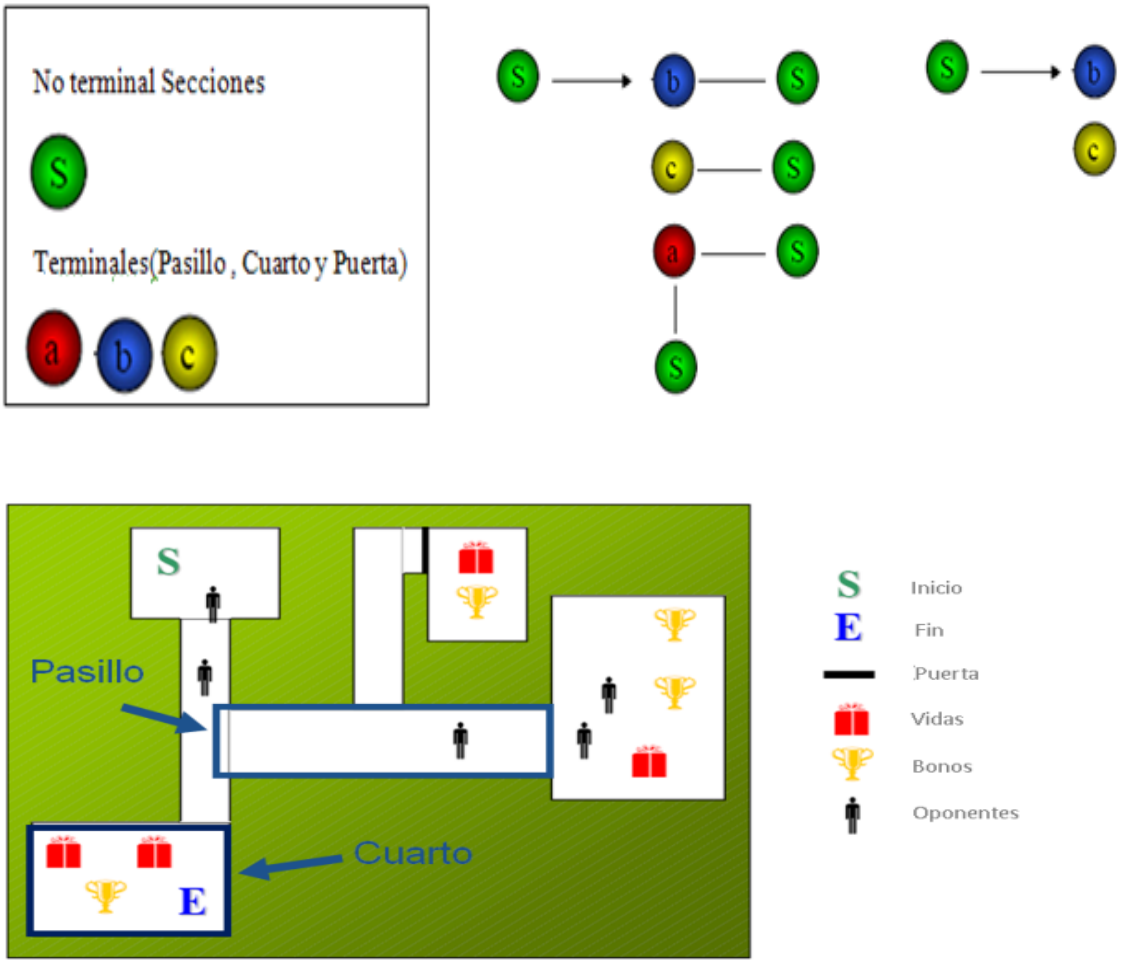

Figura 12 Gramática de grafo.

Para desarrollar un generador de interiores basado en gramáticas es importante tener en cuenta la aleatoriedad con la que se aplican las reglas de producción, las cuales son las encargadas de construir la topología de la aplicación a realizar. La aleatoriedad de las reglas de producción puede lograrse de diversas maneras, una de ellas es asignándole probabilidades a las misma, es decir si se tiene que un símbolo no terminal *A* es parte izquierda de varias reglas no sabríamos cuál aplicar en cada momento, por lo que a estas se le asigna una probabilidad de ocurrencia que bien puede ser de 0 a 1 o de 0 a 100. Este procedimiento se repite con cada grupo de reglas que sean derivadas por un mismo no terminal y así

asegurar la diversidad del entorno(Adams, 2002). En esta estrategia como es lógico las reglas que tengan mayor probabilidad son las más propensas a aplicarse. A continuación exponemos un ejemplo de lo anteriormente explicado:

GRUPO_OPONENTES –30 -> oponente

GRUPO_OPONENTES –50 -> oponente oponente

GRUPO_OPONENTES –20 -> oponente oponente oponente

MUNICIONES –10 -> balas

MUNICIONES –40 -> misiles

MUNICIONES –50 -> granadas

En el ejemplo se encuentran dos grupos de reglas de producción a las cuales se les han asignado probabilidades de ocurrencia de hasta 100.

Los sistemas de Gramáticas de Grafo(Adams, 2002)(Rozenberg, 1997) deben determinar primeramente que regla de producción han de aplicar, luego utilizan algún método de re emplazamiento de nodos para sustituir en el grafo los nodos de la parte izquierda de la regla de producción con los de la parte derecha y de esta forma hacer crecer el grafo y a su vez el entorno. Esta idea parte de la concepción de que cada regla de producción está escrita de la forma p: D → I donde D es la parte derecha de la regla de producción y I la parte izquierda. Debemos considerar a la vez q D y I son un sub grafo cada uno de ellos, en este proceso se realizarán constantes búsquedas de partes izquierdas para ser sustituidas por partes derechas en lo que se denomina la aplicación de una regla de producción. Y que la aplicación de un conjunto de reglas de producción nos llevará a la construcción del grafo principal.

Una gramática de grafo nos permite un buen control sobre el tamaño y la dificultad del entorno generado, dado que un grafo es una forma más natural de representar la red que compone el mundo donde se encuentra el jugador que puede desplazarse en varias direcciones.

El sistema de gramática de grafo es el encargado de todas las transformaciones que ocurrirán sobre el grafo, debe ser totalmente independiente, solo contendrá información de los nodos y las aristas. Pero si debe brindarnos la posibilidad de asociar a las reglas de producción una probabilidad, dichas aristas deben ser vistas como una abstracción pues solo señalan las conexiones entre nodos.

En la confección de un sistema de gramática de grafo, las gramáticas a utilizar pueden ser libres de contexto o dependiente de contexto, dado que en algún momento necesitaremos encontrar una coincidencia dentro del grafo con los nodos que constituyen la parte izquierda de una regla de producción a aplicar , aquí si la gramática es libre de contexto entonces solo se necesitara buscar el nodo correspondiente a la parte izquierda en el grafo y luego aplicar la regla de producción que puede ser por ejemplo sustituirlo por otros 2 nodos terminales , en caso de utilizar una gramática dependiente del contexto en lugar de 1 se tendría que buscar un conjunto de nodos ,enlazados formando un sub grafo, dado que es un grafo y por tanto no existe un nodo inicio , o fin , o un orden específico de los mismos esta operación de buscar un sub grafo adquiere una complejidad algorítmica alta. Ahora bien según el tipo de gramática que utilicemos será el sistema de re emplazamiento de nodos a utilizar.

1.2.3.1 Sistema Algebraico de reemplazamiento de Nodos

Si nos encontramos en ocasión de sustituir un sub grafo aplicamos el método algebraico el cual plantea que cuando en el grafo principal encontramos una coincidencia de la parte izquierda de la regla de producción y este nodo solo aparece en el lado izquierdo de la producción este es eliminado del grafo principal, si este nodo solo aparece a la derecha entonces se agregara al grafo principal y si aparece en ambos lados de la producción se mantiene en su estado inicial. En esta técnica las conexiones de los nodos reemplazados se eliminan, quedando solo los enlaces que están conectados inicialmente con el nodo recursivo, es decir con el que aparece a ambos lados de la producción. Este enfoque utiliza una gramática sensible al contexto lo que le da una mayor posibilidad de generar un entorno más grande y que conserva sus conexiones.

1.2.3.2Sistema Algorítmico de re emplazamiento de Nodos

Este método es el que se aplica al utilizar gramáticas libres de contexto(Rozenberg, 1997). Las cuales contarán con un único nodo en la parte izquierda de la regla. Al encontrar una coincidencia de la parte

izquierda de la regla de producción en el grafo principal, todas las aristas que están conectando dicho nodo a el resto del grafo son eliminadas y se crean nuevas aristas que conecten los nodos añadidos al aplicar la regla de producción al resto del grafo, basándose en reglas de conexión predefinidas. Por ejemplo supongamos que tenemos la siguiente regla de producción (figura #13) y (figura #14) es nuestro grafo principal.

Figura 13 Regla de producción.

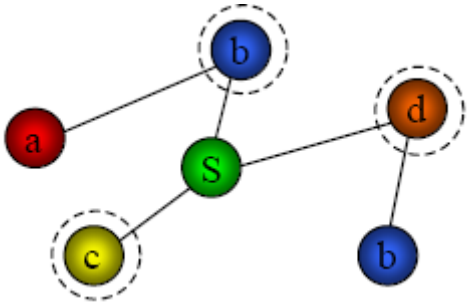

Figura 14 Grafo inicial.

Teniendo las siguientes reglas de conexión predeterminadas

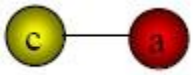

Figura 15 Regla de conexión 1.

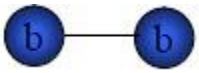

Figura 16 Regla de conexión 2.

Al aplicar la regla y sustituir el nodo S por su equivalente el grafo quedaría como sigue:

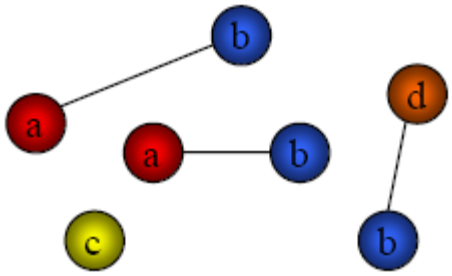

Figura 17 Representación del grafo luego de producción.

Siguiendo el algoritmo el próximo paso sería conectar los nodos según las reglas, en este caso conectaríamos el nodo a con todos los nodos c que fueran vecinos del nodo que eliminamos en este caso S. Y el nodo b añadido con todos los nodos b que eran vecinos del nodo eliminado. Al terminar este proceso el grafo queda de la siguiente forma.

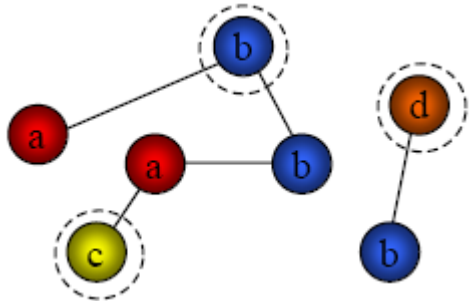

Figura 18 Representacióndel grafo luego de conectar los nodos.

Como es posible observarla desventaja de este método radica en que: si una arista que se encontraba conectada al nodo eliminado no está contemplada su conexión con alguno de los nodos agregados entre las reglas entonces esa parte del grafo quedaría desconectada como sucedió en el ejemplo con el nodo d que no poseía ninguna regla que lo conectara a los nodos añadidos.

1.3. Aleatoriedad.

Para generar números aleatorios siempre se debe definir una semilla de generación (random seed), si esta semilla no es variada la secuencia de números generados será siempre la misma(Stewart, 2004), hacer cambios constantes de esta puede aproximar la generación al tipo de aleatoriedad que se evidencia

en el mundo real. Partiendo de esto, algunos autores (Haahr., 1999) dividen los generadores de números aleatorios en dos categorías:

Generador de números aleatorios verdaderos.

Generador de números aleatorios simulados.

El primer caso se refiere a aquellos generadores que producen siempre secuencias diferentes de números aleatorios, estos nunca reproducen una secuencia de números aleatorios generados anteriormente. Por otra parte, un generador de números aleatorios simulado, devuelve siempre una misma secuencia de números aleatorios dado a que su semilla se mantiene invariable. En la generación de entornos esta última clasificación es la más usada, debido a que se desea mantener por lo general, una coherencia en los entornos tanto en la forma de los contenidos como en su ubicación espacial. La generación de secuencias de números completamente diferentes es en ocasiones compleja si para ello se toman semillas de generación similares.

1.4. Memoria a corto plazo.

En términos psicológicos la memoria a corto plazo o “Memoria Operativa” es el sistema que utilizan los seres humanos para almacenar información acerca del entorno con el cual interactúan. Está limitada a, aproximadamente, 7±2 elementos durante 10 s. Las funciones generales de este sistema de memoria abarcan la retención de información, el apoyo en el aprendizaje de nuevo conocimiento, la comprensión del ambiente ,la formulación de metas inmediatas y la resolución de problemas.(Miller, 1956)(Taylor, 2003)(Casals, 2005)

El almacenamiento temporal de cierta cantidad de información, se hace necesario en un sistema de generación de entornos virtuales, pues en ocasiones pueden existir datos que solo son necesarios durante un periodo de tiempo y luego son desechados, contribuyendo con la reutilización y el aprovechamiento de los recursos de memoria con que se dispone. Cuando se construyen terrenos con gran variedad en su relieve, la cámara se puede trasladar en una dirección u otra, pero el usuario recordará aquellos

elementos que pudo observar anteriormente cuando se desplazaba a través del entorno, lo lógico sería, que al volver al mismo lugar pueda encontrarse con los mismos elementos. En este caso, la memoria a corto plazo, desempeña un papel fundamental para almacenar durante un período de tiempo información de todos los contenidos anteriormente vistos, en la posición donde estaban ubicados y con las mismas características(Madrigal, et al., 2009).

1.5. Volumen envolvente: Bounding volume.

El volumen envolvente (bounding volume) en gráficos por computadora y en geometría computacional, es un volumen de encierro para un conjunto de objetos, que contiene completamente la unión de los objetos en el conjunto(Gilabert, et al.). Se utiliza para mejorar la eficiencia de las operaciones geométricas simples. Normalmente, se aplican para la detección de colisiones y en el proceso de selección de visibilidad.

Están extendidos a las simulaciones físicas, videojuegos y otras áreas de la geometría computacional. Los algoritmos de detección de colisiones son un componente básico de los videojuegos 3D, sin ellos los personajes podrían atravesar las paredes y otros obstáculos, además la detección de colisiones con volumen envolvente es mucho más rápida que con el objeto en sí, debido a que es una geometría mucho más simple(Gilabert, et al.).

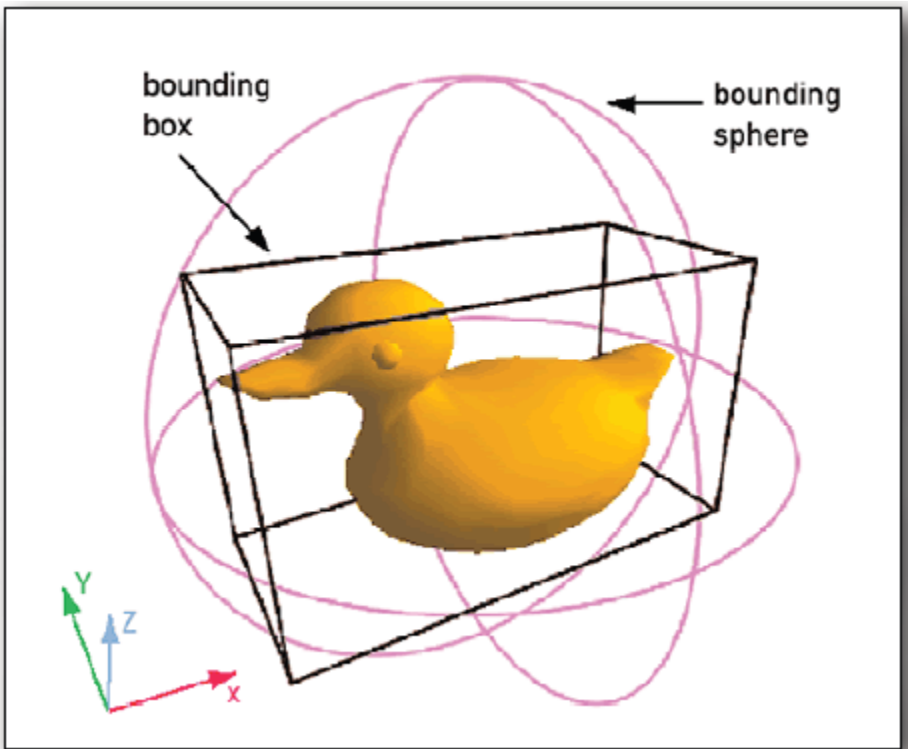

Figura 19 Representación del bounding box y del bounding sphere.

1.5.1. Tipos de volúmenes envolventes.

La elección del tipo de delimitación de volumen para una aplicación está determinada por una variedad de factores como el coste computacional de calcular un volumen de encierro de un determinado objeto, el costo de su actualización en las aplicaciones en las que los objetos pueden moverse o cambiar de forma o tamaño. El costo de la determinación de las intersecciones, y la precisión deseada en la detección de las colisiones. Es común utilizar varios tipos de volúmenes para un objeto, un volumen simple para pruebas rápidas, pero en conjunción con una mayor precisión, dada por un volumen más complejo(Gilabert, et al.).
Un ejemplo de volumen de encierro puede ser una esfera. En los gráficos 2-D, sería un círculo. La envolvente esférica está representada por centro y radio. Es muy rápida en la prueba de colisiones: dos esferas se entrecruzan cuando la distancia entre sus centros no supera la suma de sus radios. Esto las hace apropiadas para objetos que pueden moverse en cualquier número de dimensiones.

Una delimitación cilíndrica es un cilindro que contiene el objeto. En la mayoría de las aplicaciones el eje del cilindro está alineado con la dirección vertical de la escena. Los cilindros son apropiados para los objetos 3-D que sólo pueden girar alrededor de un eje vertical, pero no en otros ejes. En los videojuegos, los volúmenes envolventes cilíndricos se utilizan con frecuencia como delimitación de los volúmenes de personas de pie(Gilabert, et al.).

Una caja envolvente (bounding box) es un cubo, o en 2-D un rectángulo, que contiene el objeto. Son preferibles a otras formas de delimitación cuando la detección de la colisión tiene que ser bastante precisa. Estos tienen una variante denominada OBB o (oriented bounding box) los cuales se diferencian en que sufren en tiempo de ejecución las mismas transformaciones que los objetos a los que engloban por lo que representan un aumento de costo pero también brindan mayor precisión en las operaciones(P.D'Amato, 2009).

También es determinante en la elección del volumen envolvente a utilizar la geometría a la que envolverá y el entorno en el que se utiliza pues no sería óptimo por ejemplo utilizar una esfera para envolver un carro pues sería imprecisa su colisión con el suelo, este es un caso donde es mejor utilizar un cubo como envoltura.

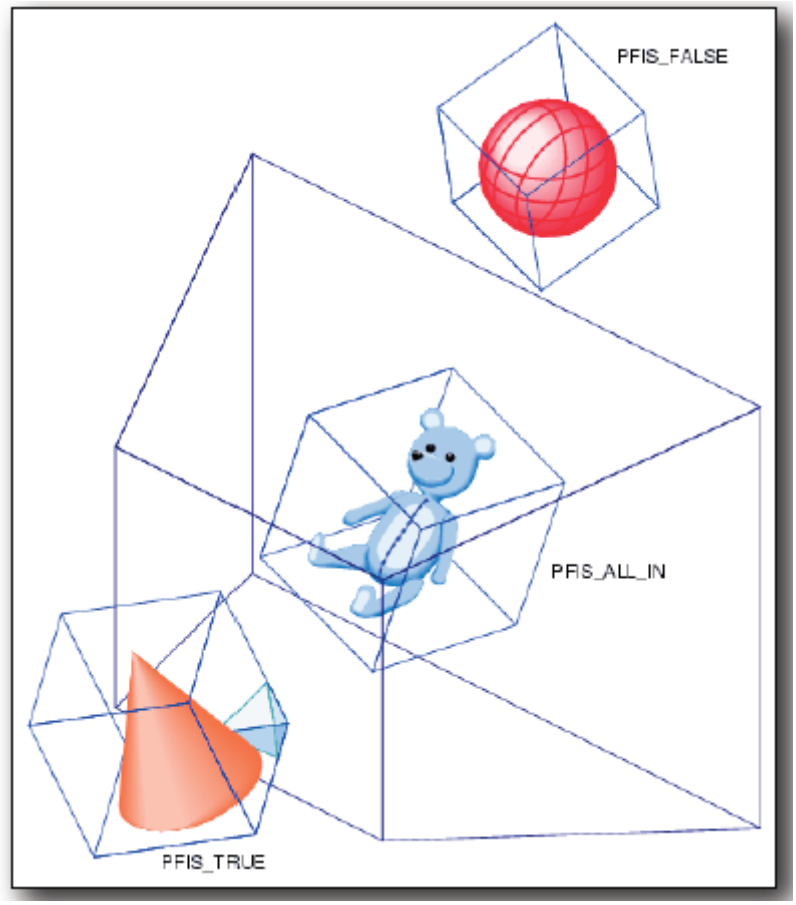

Figura20 Tipos de bounding box

1.6. Bibliotecas estudiadas.

Se realizó un estudio de una biblioteca elaborada en la Universidad de las Ciencias Informáticas llamada Generador de Entornos Virtuales en tiempo de ejecución(Madrigal, et al., 2009), la misma está diseñada para guiar la creación de entornos infinitos para videojuegos, es una herramienta genérica basada en capas, donde el usuario tiene la posibilidad de definir sus propios contenidos y la forma en que se comportar los mismos, es decir, como rotan , se desplazan y la manera en que se cargar los modelos a utilizar, pues cada motor gráfico realiza esta acción de forma diferente. Esta biblioteca también posee un

generador de terrenos prefabricados basado en una estrategia de gestión de fragmentos y una memoria a corto plazo que permite “recordar” durante un periodo de tiempo contenidos generados anteriormente. La principal fortaleza de esta biblioteca radica en la flexibilidad de su arquitectura que permite la interacción con cualquier motor gráfico, además de facilitar la creación de generadores de entornos de diversos tipos.

Capítulo 2: Solución Propuesta.

2.1. Configuración de la Gramática.

La generación de interiores basada en gramáticas de grafos se caracteriza por definir reglas de producción a partir de las cuales se podrá generar el entorno. La gramática a utilizar es libre de contexto, lo que la hace más flexible. No se utilizará una gramática dependiente del contexto buscando mayor flexibilidad y velocidad en la aplicación final. Se acordó también que solo contendrá un No-Terminal, el cual podrá derivar cualquier combinación de terminales y no terminales en su parte derecha. Esto supone una ventaja, y es que todos los terminales podrán estar conectados entre sí, siempre respetando la coherencia definida por el usuario en las reglas de producción.

Abstractamente los terminales representan los contenidos topológicos que se colocarán en la escena, por lo cual al escribir la gramática es necesario especificar la cantidad de conexiones que contiene cada terminal, estas conexiones pueden ser de 1 hasta 4, siendo obligatorio declarar un terminal de una conexión para garantizar el final de la generación y otro de más de una para garantizar la generación infinita.

Las reglas de producción en su parte izquierda siempre tendrán el No-Terminal de la gramática, el cual derivará en la parte derecha de la regla, compuesta por la combinación de Terminales y No-Terminales. Este paso tiene sus peculiaridades dado que la parte derecha puede estar compuesta de solo terminales, aunque para que esto se cumpla, deben culminar obligatoriamente con un terminal de una sola conexión, siendo esta, una regla de producción no recursiva, las cuales al ser aplicadas son las que ponen punto final a la generación . Las reglas de producción recursivas al contrario de las no recursivas, deben terminar con un terminal de más de una conexión. Estas reglas son muy importantes para el presente trabajo porque son un paso importante en el cumplimiento de una de las metas del mismo que es generar interiores de forma infinita.

Todas las reglas de producción deben cumplir con la siguiente sintaxis:

Todos los terminales pertenecientes a una regla de producción que no se encuentren al final de la misma deben ser de dos conexiones. Esta restricción permite que al ser lineal la regla de producción, cada terminal tenga sus conexiones cubiertas.

Como se había comentado, las reglas de producción recursivas son aquellas que terminan con un terminal de más de una conexión. Dichas conexiones son satisfechas con No-Terminales que agrega el generador de forma implícita ahorrándole ese trabajo al usuario que las declara. Este tipo de reglas de producción siempre va a contener símbolos No-Terminales y estos solo aparecerán al final de la regla.

Las reglas de producción no recursivas terminan con un terminal de una conexión lo que implica que estas no contienen símbolos No-Terminales en su parte derecha.

Un ejemplo de gramática de grafo libre de contexto se propone a continuación:

No Terminal:

Secciones

Terminales:

Cuarto1 (4 conexiones)

Cuarto2 (2 conexiones)

Pasillo1 (2 conexiones)

Cuarto3 (1 conexión)

Reglas de producción:

Secciones -> Cuarto2 – Pasillo1

Secciones -> Cuarto2 – Cuarto1

Secciones -> Cuarto2 – Cuarto3

Secciones -> Cuarto3

Secciones -> Cuarto1

Como se puede observar en el ejemplo, existe un solo no terminal en la gramática y los terminales tienen especificado el número de conexiones con que cuenta cada uno. En el ejemplo el terminal Cuarto3 posee una sola conexión y los otros más de una, por lo que cumple las restricciones planteadas. Entre las reglas de producción es posible clasificar como no recursiva a (Secciones -> Cuarto3), el resto son reglas recursivas.

2.2. Herramienta “Editor de Gramática”

Como se pudo observar la gramática a utilizar por el generador debe ser definida por el usuario con sumo cuidado, ya que este tiene que tener presente una serie de restricciones y características que no pueden ser pasadas por alto. Por este motivo se debe crear una herramienta que ayude al usuario a lidiar con dichas gramáticas de una forma segura, previendo los errores más comunes en los que estos suelen incurrir.

La herramienta debe detectar errores, notificarlos e impedir el uso de la gramática en el generador hasta que no esté completamente limpia de errores. Existen diferentes tipos de errores que la gramática deberá detectar, entre los cuales se encontrarán: sintácticos, semánticos y advertencias, este último no impedirá la salva del fichero a utilizar, solo nos alertaría sobre terminales declarados que no se están utilizando.

Este editor es el encargado de salvar el fichero que leerá la biblioteca de generación de interiores.

Sintaxis de la Gramática

Para que la herramienta a desarrollar detecte los errores de forma correcta, es necesario que la gramática que toma como entrada el editor siga ciertas reglas sintácticas. La gramática se divide en

bloques, los cuales deben estar en un determinado orden de aparición. Estos bloques son encabezados por etiquetas o palabras reservadas, las cuales indican el tratamiento que recibirá cada uno de estos.

En general la gramática va estar constituida por 5 bloques. El primero de estos, encabezado por la palabra reservada (**inicio**), indica el comienzo de la definición de la gramática, así como el bloque representado por la palabra reservada (**fin**), será el encargado de dar a conocer la culminación de la misma.

El bloque que le sigue al bloque inicio es el responsable de declarar el símbolo no terminal de la gramática, el cual está encabezado por la etiqueta (**no-terminal**:). Seguido de esta, se declara dicho símbolo, el cual debe constituir una cadena valida, luego se coloca un punto y coma (;) al final del mismo, indicando la culminación del bloque. Una cadena es válida si comienza con una letra o el símbolo (_) y es seguida por letras, números o símbolos (_). Este concepto también es válido para la declaración de los símbolos terminales.

Ejemplo del bloque de No-Terminales:

no-terminal:

Secciones;

Una vez declarado el bloque de los no terminales, se pasa a declarar el de los terminales. Este bloque está definido en su inicio por la palabra reservada (**terminales**:). Seguido se comienzan a especificarlos símbolos terminales, los cuales al igual que los no terminales deben constituir cadenas válidas. Después de esto se indican la cantidad de conexiones que posee cada símbolo terminal, representado por el símbolo (^) y luego se indican las direcciones hacia donde se encuentran sus conexiones, estas pueden ser de cuatro tipo (Up, Down, Left, Right). En este bloque se pueden declarar cuantos símbolos terminales se requiera, todos seguidos de la cantidad de conexiones y terminados en punto y coma (;)

Ejemplo del bloque de terminales:

terminales:

cuarto1 ^ 4 – Left Right Up Down;

cuanto2 ^ 1 - Up;

pasillo ^ 2 – Left Right;

Por último, el bloque de las reglas de producción, representado por la etiqueta (**reglas_produccion**), es donde se declaran todas las reglas existentes en la gramática. La sintaxis de cada una de ellas se expone a continuación:

- La parte derecha de la regla está separada de la parte izquierda por medio de una flecha (->) la cual significa derivación.
- En la parte izquierda los símbolos terminales se separan por medio del símbolo (-), el cual simboliza las conexiones entre ellos.

Ejemplo del bloque de Reglas de Producción:

reglas_produccion:

Secciones -> cuarto1;

Secciones -> cuarto2;

Secciones -> pasillo;

Secciones ->pasillo - cuarto1;

Secciones -> pasillo – cuarto2;

Ejemplo de definición de una gramática valida

inicio

no_terminal:

Secciones;

terminales:

cuarto1 ^ 4 – Left Right Up Down;

cuanto2 ^ 1 - Up;

pasillo ^ 2 – Left Right;

reglas_produccion:

Secciones -> cuarto1;

Secciones -> cuarto2;

Secciones -> pasillo1;

Secciones -> pasillo1 - cuarto1;

Secciones -> pasillo1 – cuarto2;

Final

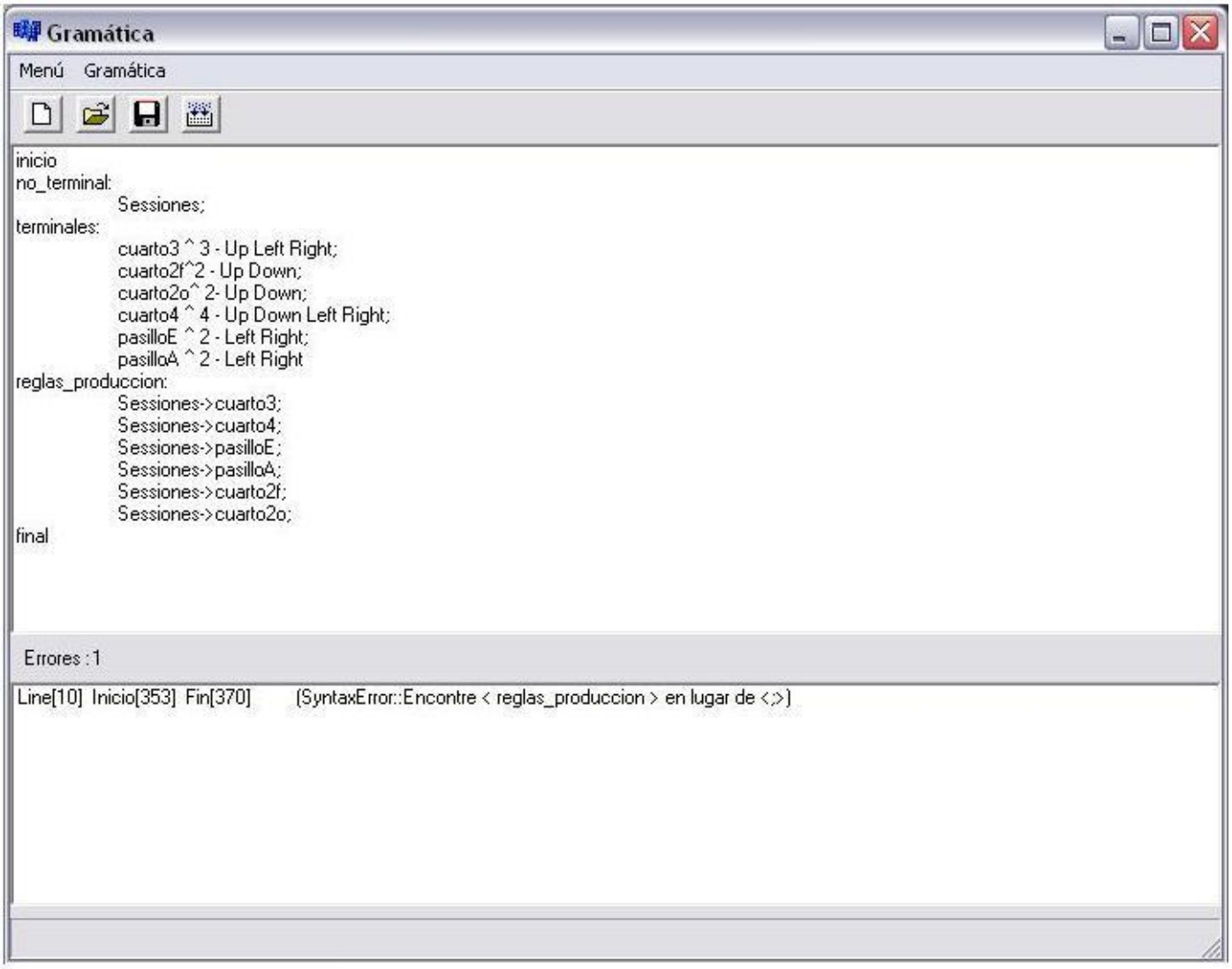

Figura 21 Imagen de la herramienta de configuración de gramáticas

2.3. Creación en el generador de las reglas de producción y los símbolos.

El primer paso a realizar debe ser la lectura del fichero con la gramática, de donde se obtendrán los símbolos terminales con sus cantidades de conexiones, las reglas de producción con la información de si son recursivas o no y el axioma, todo basado en las cadenas de caracteres obtenidas del fichero. Dado que las gramáticas solo poseerán un no terminal entonces las reglas de producción solo almacenarán la

parte derecha de las mismas y su tipo de recursividad. Sería recomendable almacenar estas divididas según el tipo de recursividad para imprimir velocidad al proceso de obtención aleatoria de una regla de algún tipo (recursiva o no recursiva) pues de tenerlas todas en una misma lista habría que recorrer en cada momento dicha lista e ir preguntando por las reglas del tipo que se esté buscando para almacenarlas y luego devolver una de ellas de forma aleatoria, proceso el cual resultaría un poco lento. Para lograr este objetivo proponemos la creación de una clase, que almacene ambas listas y además el no terminal al que estas pertenecen, de esta forma, si en algún momento el sistema necesita cambiar a gramáticas con más de un no terminal puede utilizar una colección de elementos de este tipo diferenciados cada uno por el no terminal al que corresponden.

2.4.Aplicando la primera Regla.

Al finalizar la lectura del fichero de la gramática de forma automática se procede a crear el grafo, proceso que debe comenzar con la elección al azar de una regla de producción para ubicarla directamente en el grafo , esta regla de producción debe ser de tipo recursiva pues debemos garantizar que el entorno seguirá creciendo. En caso contrario se obtendría un entorno limitado a los elementos de una única regla de producción, sin posibilidades de continuar la expansión.

Como el grafo comenzará desde el vacío, al menos para la primera regla de producción el proceso debe de ser similar al siguiente. Primeramente se crearán nodos los cuales almacenarán los símbolos correspondientes a la parte derecha de la regla de producción que estamos aplicando, acto seguido se crearán conexiones entre todos los nodos añadidos formando así una especie de cadena con los mismos, luego como hemos escogido una regla recursiva debemos agregar a los nodos que aún les falten conexiones por llenar la cantidad suficiente de no terminales hasta lograr el número de conexiones que ellos admiten. Al terminar este proceso ya el grafo estaría inicializado , pero de intentar utilizarlo en este momento corremos el riesgo de solo poder visualizar el nodo inicial rodeado del vacío, para evitar esto y asegurar que el nodo inicial este rodeado de adyacentes que no sean no terminales deben aplicarse reglas de producción sobre estos. Ahora explicaremos de forma más detallada cada uno de estos pasos.

2.4.1 Como adicionar un nodo.

El proceso de adicionar un nodo al grafo solamente necesitará de un símbolo, estos símbolos son limitados ya que provienen de la gramática por lo que un nodo a pesar de ser básicamente un símbolo

realmente contendrá un símbolo como información además de poseer un identificador, lo cual nos permitirá por ejemplo tener 2 nodos del tipo equivalente al símbolo “cuarto” los cuales no serían iguales debido a que tendrían diferentes identificadores.

Los nodos generados se almacenarán en el grafo, que estará equipado con una lista de nodos y una lista de adyacencia. Cada vez que se solicita adicionar un nodo dado un símbolo, internamente se creará un nodo con el símbolo como información, se le asignará un identificador y se le debe crear una lista de adyacencia inicialmente vacía.

2.4.2 Como conectar dos nodos.

La próxima acción a realizar es conectar los nodos que se crearon, como fue visto en la investigación la creación de una arista entre dos nodos es algo simbólico que solo representará la unión entre dichos nodos y por ende la capacidad de viajar directamente entre ellos esto lo que nos representa es que cuando nos encontremos en el juego existirá una puerta que al atravesarla nos ubicará en el nodo al que estamos conectados. Por tanto, la creación de una arista solo consta de 2 acciones básicas primero la comprobación de que esta arista no exista, y luego agregamos la arista que no es más que el registro del identificador del nodo al que se está conectando en la lista de adyacencia perteneciente al nodo en cuestión, como estamos en presencia de un grafo no dirigido, también debemos añadir una conexión en sentido contrario.

2.4.3 Adicionar nodos recursivos.

Debe ser una funcionalidad que nos permita en cualquier momento, agregarle como adyacentes a un nodo la cantidad de no terminales suficientes para completar el máximo de conexiones que este admite. La misma debe utilizar el identificador del nodo al que añadirá los no terminales como adyacentes y el símbolo que contiene dicho nodo para ser capaz de conocer la cantidad de conexiones del mismo. Esta técnica nos permitirá que los no terminales siempre se encuentren en la zona exterior del grafo haciendo más fácil la navegación y expansión del mismo.

Aquí existen dos situaciones para la primera regla que aplicamos, ya que puede ser de un elemento o de varios, en cualquiera de los casos el proceso de llenar las conexiones restantes solo agrega no terminales

hasta la cantidad de conexiones admitidas por el símbolo. Por ejemplo de utilizar una gramática como la siguiente:

No_terminal: NT;

Terminales: A - 4, B - 2, C - 1;

Reglas:

NT → B;

NT → B – C;

NT → A;

NT→ B – A;

Primero que todo notar que en esta gramática las reglas recursivas serían la primera, la tercera y la cuarta , supongamos que al solicitar una de ellas para ser la primera en aplicarse el resultado fue la tercera regla , pues bien en este caso como consta de un solo elemento y A tiene cuatro conexiones posibles entonces el resultado de agregarle los nodos adyacentes correspondientes tendría una representación gráfica aproximada a la siguiente, donde podemos ver el símbolo acompañado de todos los no terminales que lo rodean.

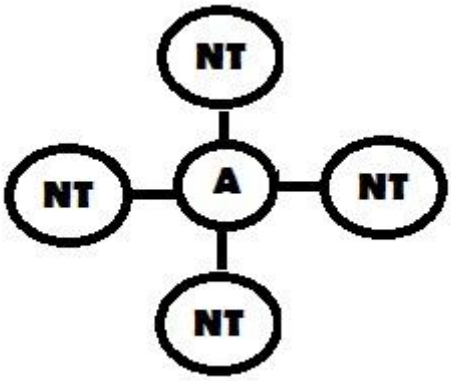

Figura 22 Símbolo de cuatro conexiones con sus no terminales adyacentes.

Suponiendo que en lugar de la tercera aplicáramos la cuarta regla que posee dos elementos el grafo finalizaría como sigue.(Ver Figura. 22)

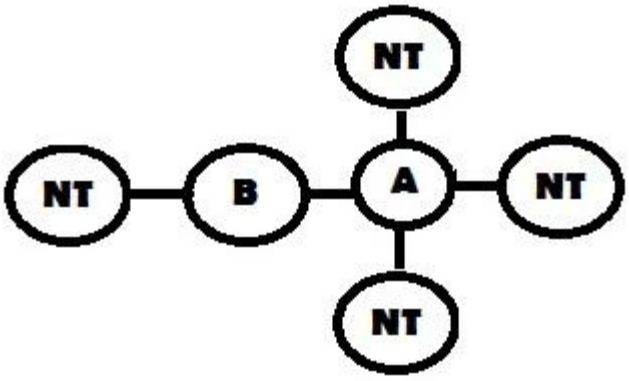

Figura 23 Regla de producción de más de un terminal con sus no terminales adyacentes.

Aquí podemos ver como al terminar la transformación cada nodo tiene llenas todas sus posibles conexiones ya sea con no terminales que luego pudieran volverse en terminales o con terminales a los que se encuentra conectado.

Para asegurarnos de tener algún lugar hacia el cual movernos físicamente desde el nodo inicial, ahora debemos aplicar reglas de producción sobre cada uno de los no terminales generados, pues de no hacerlo

al intentar visualizar el entorno solo tendríamos por ejemplo el cuarto inicial y un vacío tras sus puertas pues no existiría ningún nodo adyacente que sea un terminal el cual pudiéramos visualizar.

2.5. Aplicar Reglas de Producción.

Para aplicar una regla de producción es necesario conocer el identificador del nodo al que vamos a sustituir y la parte derecha de la regla por la que lo vamos a sustituir. Para realizar este procedimiento primeramente adicionaremos al grafo un nodo por cada símbolo de la parte derecha de la regla estos nodos deben contener a dichos símbolos, luego conectamos los nodos agregados entre ellos. Esta serie de nodos conectados entre sí, ahora debe ser enlazada al grafo, para ello la conectaremos al adyacente del no terminal sobre el cual estamos aplicando la regla y luego debemos borrar el no terminal con todas las conexiones que se relacionan con él. En cierta forma esta es una combinación de los enfoques algebraico y algorítmico ya que tenemos una gramática libre de contexto donde en vez de utilizar reglas de conexión prescritas vamos a conectar un nodo de los nodos agregados a un nodo que pertenece al grafo con lo cual nos aseguramos de que todos los nodos agregados ya están conectados al grafo mediante la arista creada.

2.6.Expansión del grafo.

Para lograr un entorno infinito en tiempo de ejecución con una gramática de grafo, es necesario que en la medida en que exista un desplazamiento en el entorno, el grafo continúe creciendo y en el sentido en que nos estamos moviendo, pues no tendría sentido hacer crecer el grafo en un sentido si el desplazamiento se está realizando en otro.

Esta funcionalidad es muy importante pero a la vez sencilla pues solo tomará la posición hacia la que nos dirijamos y generará los adyacentes correspondientes para que se pueda continuar el desplazamiento. La misma deberá ser capaz de determinar cuándo debe ser aplicada una regla recursiva o no recursiva en caso de que se desease comenzar a acotar el entorno y en qué momento una parte del grafo puede ser borrada.

Para evitar que el grafo crezca en gran medida, cuando se haya recorrido una parte del mismo se borrarán los nodos más antiguos ya que un usuario del juego podría recordar quizás las últimas 3

habitaciones en las que estuvo pero muy difícilmente recuerde las 10 habitaciones que visitó anteriormente.

2.7. Generador por capas.

Con el objetivo de que el generador pueda interactuar con cualquier motor gráfico se creará una sistema conformado por tres estructuras básicas, las capas que contendrán las copias de los contenidos que forman parte del entorno, los contenidos que contendrán los modelos tri-dimencionales que se visualizarán en el entorno y una estructura controladora que almacenará las capas y los contenidos originales denominada fábrica cuya responsabilidad será creará las copias de los contenidos para asignárselas a las capas, evitando la permanente carga de los modelos.

Se utilizará una estrategia con 5 capas que se van trasladando y rotando en torno a la dirección en la que se desplace el usuario y se reemplazarán los contenidos de las mismas según los símbolos presentes en el grafo, se utilizarán 5 debido a que un nodo solo podrá tener hasta 4 adyacentes que al ser visualizados conjuntamente con el nodo suman 5 elementos. La rotación de las capas se realiza en función de las direcciones en las que se encuentran las conexiones que poseen los contenidos, permitiendo un correcto enlace de los mismos.

Palabras claves: Capas, Contenidos, Conexiones.

2.8.*Detección del Momento de Generación.*

Siempre que el usuario se traslade en el entorno será necesario determinar el momento de cambio de un modelo a otro, que es cuando se generarán nuevos elementos en el grafo, se trasladarán las capas y cambiarán sus contenidos. Para realizar esta acción es necesario conocer las coordenadas de un punto de referencia el cual puede representar la posición de la cámara o de un determinado avatar. De los cálculos referentes al cambio de modelo se encargará un subsistema llamado Bounding Box.

La biblioteca utilizará un Oriented Bounding Box (OBB) o Bounding Box Orientado al Objeto. El OBB es muy preciso, debido a que la envoltura solo se calcula una vez, luego se le aplican las mismas

transformaciones que al objeto, quedando de forma idéntica, pero con una orientación diferente. Esto nos permitirá que al rotar el modelo en cualquiera de sus ejes la envoltura se mantenga ajustada y relativamente orientada.

Además, para la generación de los contenidos en la escena, se necesita contar con la información de las dimensiones de la caja que envuelven los objetos, para así poder colocar los adyacentes a este sin que se solapen entre ellos.

2.9. Especificaciones del generador de entorno de interiores.

Uno de los detalles que hay que tener presente antes de usar el generador de entornos de interiores son las coordenadas del motor gráfico a utilizar. Esto está dado porque la biblioteca establece sus propios ejes de coordenadas, lo cual afecta la forma en que se trasladan y rotan los contenidos que conforman la escena. La biblioteca establece que la coordenada Z aumenta hacia fuera de la pantalla y disminuye hacia adentro, la coordenada Y aumenta en dirección hacia arriba, disminuyendo en sentido contrario y la ordenada X crece de izquierda a derecha, disminuyendo de derecha a izquierda. Dado que cada motor gráfico define su sistema de coordenadas, el usuario deberá definir una transformación, encargada de llevar las coordenadas del motor que utiliza a las coordenadas de la biblioteca.

Otra de las características que tiene la biblioteca de generación de entorno de interiores es que trabaja internamente con sus propios vectores y matrices. La mayoría de los motores gráficos definen su propia matemática, por lo que se hace necesario convertir los vectores y matrices brindado por este en aquellos que soporta el generador. Esta no debe ser una tarea que le traiga grandes complicaciones al programador.

Esta versión del generador tiene una restricción en cuanto al modelado y diseño de las geometrías que conformaran el entorno. Los modelos 3D que forman parte del generador deben tener a lo sumo cuatro conexiones, es decir, desde el punto de vista del diseño no pueden exceder las cuatro puertas o portales. Además, estas conexiones deben estar situadas exactamente en el medio de las caras que conforman dicho modelo. Esta característica debe ser respetada para obtener un entorno perfectamente acoplado, ya que garantiza que los diversos modelos que participan en la escena queden correctamente conectados.

2.10. Proceso de desarrollo.

2.10.1. Lenguaje de programación.

Se utilizará como lenguaje de programación el C++ estándar, por ser un lenguaje robusto, libre, multiplataforma con soporte para la Programación Orientada a Objetos. Se utilizarán sus estándares para construir un componente libre en cuanto a Sistema Operativo y compatible con varios motores gráficos pues este lenguaje creado a mediados del año 1980 es el más utilizado en la actualidad en el área de gráficos por computadora. Como IDE de desarrollo se utilizará Visual Studio por las facilidades que brinda para la programación en este lenguaje.

2.11. Visual Paradigm.

Visual Paradigm es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Este software ayuda a un modelado UML más rápido, contribuyendo a construir aplicaciones de calidad con un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Esta herramienta CASE también proporciona abundantes tutoriales, demostraciones interactivas y proyectos UML. Por ser una herramienta libre y con muchas facilidades en su uso se utilizará para la realización del modelado del proyecto.

Capítulo 3: Ingeniería del Sistema.

3.1. Modelo de Dominio.

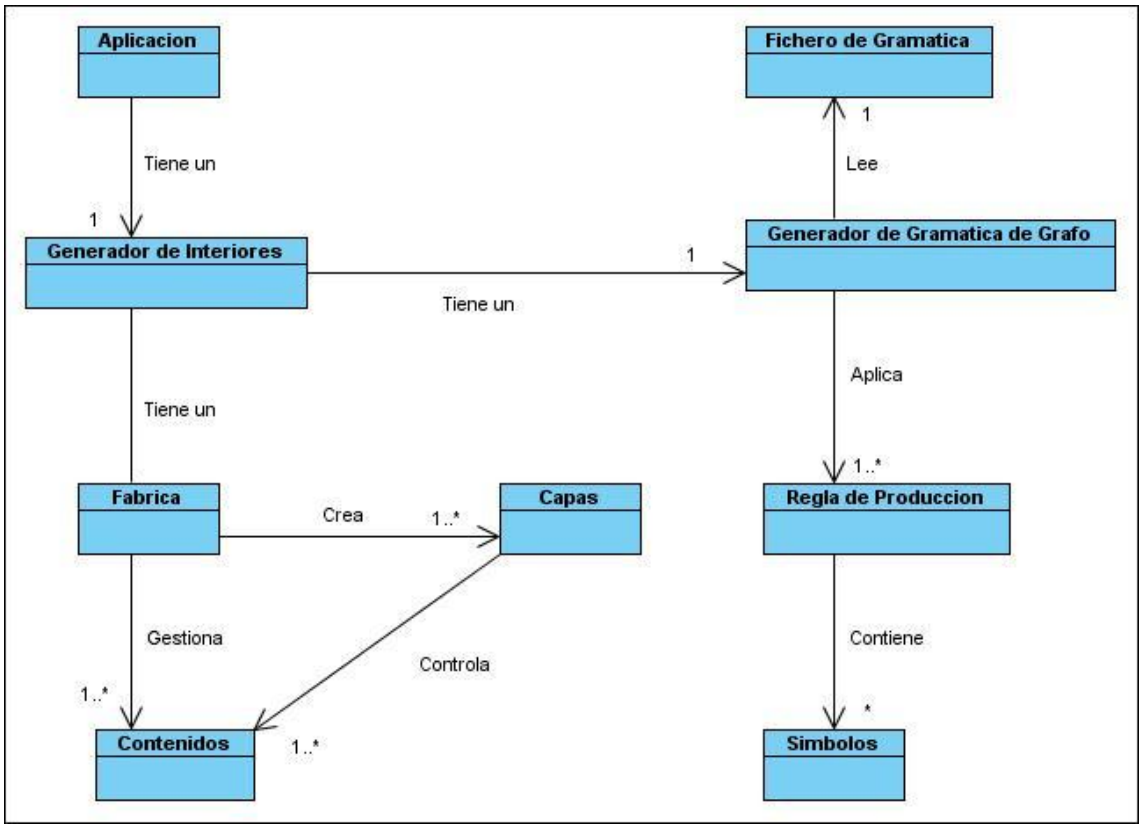

Figura 24 Modelo de Dominio.

3.2. Glosario de términos del Dominio

Capas: Se encargan de gestionar los contenidos de la escena vinculados al generador de interiores permitiendo trasladarlos, rotarlos y reemplazarlos.

Contenidos: Representan los modelos 3D que utiliza el generador para construir su entorno.

Símbolos: Son la representación abstracta de los contenidos a utilizar en el entorno.

Reglas de Producción: Se definen como los dos sub grafos denominados parte Derecha y parte Izquierda, donde esta última es reemplazable por la primera. Están conformadas por símbolos y se encargan de darle un orden a la generación.

Fábrica: Es la encargada de cargar, almacenar y replicar los contenidos originales, para así utilizar sus copias en la escena, permitiendo además crear las capas encargadas de gestionar los mismos.

Fichero de gramática: Contenedor de la gramática por la cual se rige el generador. En él se encuentra la declaración de los símbolos y las reglas de producción.

Generador de Gramática de Grafo: Contiene las principales funcionalidades para construir el entorno mediante la aplicación de las reglas de producción.

Generador de Interiores: Constituye la interfaz controladora encargada de fusionar los elementos abstractos de la gramática con los modelos físicos que conforman el entorno.

3.3. Requisitos no funcionales

- **Requisitos de Software**

  La biblioteca puede ser compilada en Windows XP, Ubuntu 8.4 y versiones superiores a estas. Además, para su utilización se requiere del uso de algún motor gráfico como Scene Tool Kit(STK) u Object-Oriented Graphics Rendering Engine(OGRE).

- **Requisitos de Hardware:**

Se requiere de una PC con prestaciones mínimas de:

Procesador: Pentium 4.

Memoria RAM: 512Mb.

- **Restricciones en el diseño y la implementación**

La biblioteca debe implementarse en el lenguaje C++ utilizando sus bibliotecas estándares y debe brindar la posibilidad de ser integrada con cualquier motor gráfico.

- **Requisitos de Seguridad**

Integridad de los datos provenientes del fichero de la gramática.

- **Requisitos de Usabilidad**

La biblioteca de generación de entornos de Interiores exige ser usada por usuarios con un básico conocimiento de gráfico por computadora y de programación.

3.4. Requisitos funcionales

R1 Inicializar entorno.

R1.2 Leer el fichero de la gramática.

R2 Generar entorno.

R2.1 – Aplicar Regla de producción.

R2.2 – Organizar Contenidos.

R3 Gestionar Contenido.

R3.1 Almacenar contenidos.

R3.2 Crear contenido copia.

R3.3 Eliminar contenido copia.

3.5. Descripción de los casos de uso

Descripción del caso de uso "Inicializar Entorno"

Caso de Uso:	Inicializar Entorno
Actor:	Aplicación
Resumen:	El caso de uso se inicia cuando la aplicación indica inicializar el entorno. El sistema lee la gramática, crea el grafo inicial, los contenidos y las capas a utilizar en el entorno.
Referencia:	R1, R1.2
CU asociados:	
Precondiciones:	El fichero de la gramática tiene que estar creado.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. La aplicación ordena inicializar el entorno.	1.1- Ir a la sección "Leer el fichero de la gramática". 1.2. Se crean las capas a utilizar en el entorno.

	1.3. Se cargan los modelos 3D con los cuales se crean los contenidos originales del entorno.
Sección "Leer el fichero de la gramática"	
Acción del Actor	**Respuesta del Sistema**
	1.1.1. Crear los símbolos terminales y no terminales que conformaran el grafo, obtenidos a partir de la gramática definida por el usuario. 1.1.2. Crear las reglas de producción que guiaran la generación, a partir de la información brindada por el usuario en el fichero de la gramática. 1.1.3. Crear el Grafo de símbolos.
Flujos Alternos	
Acción del Actor.	**Respuesta del Sistema**
Fichero de la gramática no existente	Se procede a lanzar un error de aviso y automáticamente se sale de la aplicación.

Pos condiciones:	Los contenidos originales se encuentran cargados y el grafo creado.
Prioridad:	Crítico.

Descripción del caso de uso "Generar Entorno"

Caso de Uso:	Generar Entorno
Actor:	Aplicación
Resumen:	El caso de uso se inicia cuando el sistema indica comenzar la generación. La biblioteca aplica una regla de producción de las leídas en la gramática y organiza las capas y contenidos dependiendo del resultado de la misma. Este procedimiento es repetido cada vez que ocurre algún desplazamiento dentro del entorno de interiores.
Referencia:	R2, R2.1, R2.2
CU asociados:	
Precondiciones:	Sistema inicializado.
Flujo Normal de Eventos	

Acción del Actor	**Respuesta del Sistema**
1. La aplicación indica generar el entorno.	1.1. Ir a sección “Aplicar Regla de producción”. 1.2. Ir a sección “Organizar Contenidos”.
Sección “Aplicar Regla de producción”	
Acción del Actor	**Respuesta del Sistema**
	1.1.1. Seleccionar un nodo no terminal del grafo. 1.1.2. Seleccionar una regla de producción de forma aleatoria. 1.1.3. Crear un sub grafo en función de la regla de producción seleccionada. 1.1.4. Sustituir el nodo no terminal por el nuevo sub grafo.
Sección “Organizar Contenidos”	
Acción del Actor	**Respuesta del Sistema**
	1.2.1. Se le asignan los contenidos a las capas dependiendo de la regla de producción aplicada. 1.2.2. Se rotan las capas teniendo en cuenta la orientación que tienen las

	conexiones de sus contenidos y se trasladan en dependencia de la dirección hacia la que se desplace el usuario.
Flujos Alternos	
Acción del Actor.	**Respuesta del Sistema**
Pos condiciones:	Generación de un entorno de interiores infinito.
Prioridad:	Crítico.

Descripción del caso de uso "Gestionar Contenidos"

Caso de Uso:	Gestionar Contenidos
Actor:	Aplicación
Resumen:	El caso de uso se inicia al cargar los contenidos originales del entorno , al adicionar copias de estos a las capas o al eliminarlos
Referencia:	R3, R3.1, R3.2, R3.3
CU asociados:	
Precondiciones:	Sistema inicializado

<table>
<tr><th colspan="2">Flujo Normal de Eventos</th></tr>
<tr><td>Acción del Actor</td><td>Respuesta del Sistema</td></tr>
<tr><td></td><td>1. El sistema cuenta con varias opciones sobre los contenidos.

a) Si se desea cargar y crear los contenidos originales, ir a la sección “Almacenar Contenidos”.

b) Si se desea crear copias a los contenidos originales, ir a la sección “Crear contenido copia”.

c) Si se desea eliminar contenidos, ir a la sección “Eliminar Contenidos”.</td></tr>
<tr><td colspan="2">Sección “Almacenar contenidos”</td></tr>
<tr><td>Acción del Actor</td><td>Respuesta del Sistema</td></tr>
<tr><td></td><td>2.1. Se crean los contenidos originales del entorno.

2.2. Se crean las capas que contendrán los contenidos del entorno</td></tr>
<tr><td colspan="2">Sección “Crear contenido copia”</td></tr>
</table>

Acción del Actor	**Respuesta del Sistema**
3. La aplicación envía una identificación del contenido para adicionar en la capa.	3.1. Se crea una copia de un contenido inicialmente cargado 3.2. Adiciona el contenido a una capa ya existente.
Sección "Eliminar contenido copia"	
Acción del Actor	**Respuesta del Sistema**
4. La aplicación selecciona el contenido para ser eliminado de la capa.	4.1. Se elimina el contenido que posee la capa.
Flujos Alternos	
Acción del Actor.	**Respuesta del Sistema**
Pos condiciones:	Contenidos cargados y creados, copiados y eliminados correctamente.
Prioridad:	Crítico.

3.6. Diagrama de casos de uso del sistema

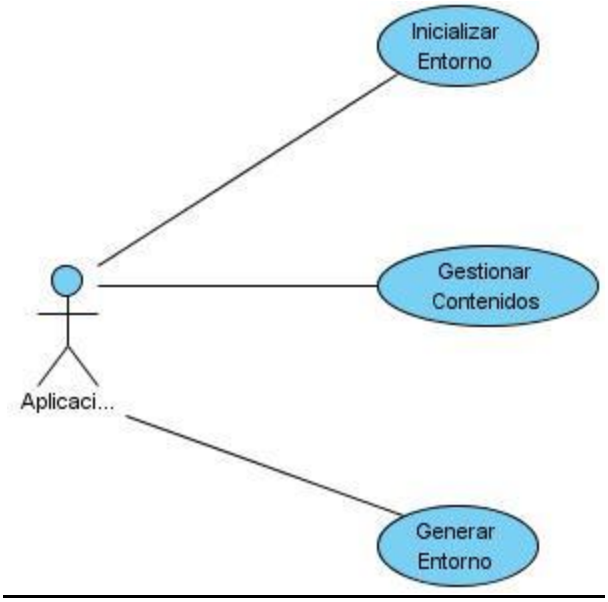

Figura 25 Diagrama de caso de uso del sistema.

3.7. Diagrama de clases del diseño

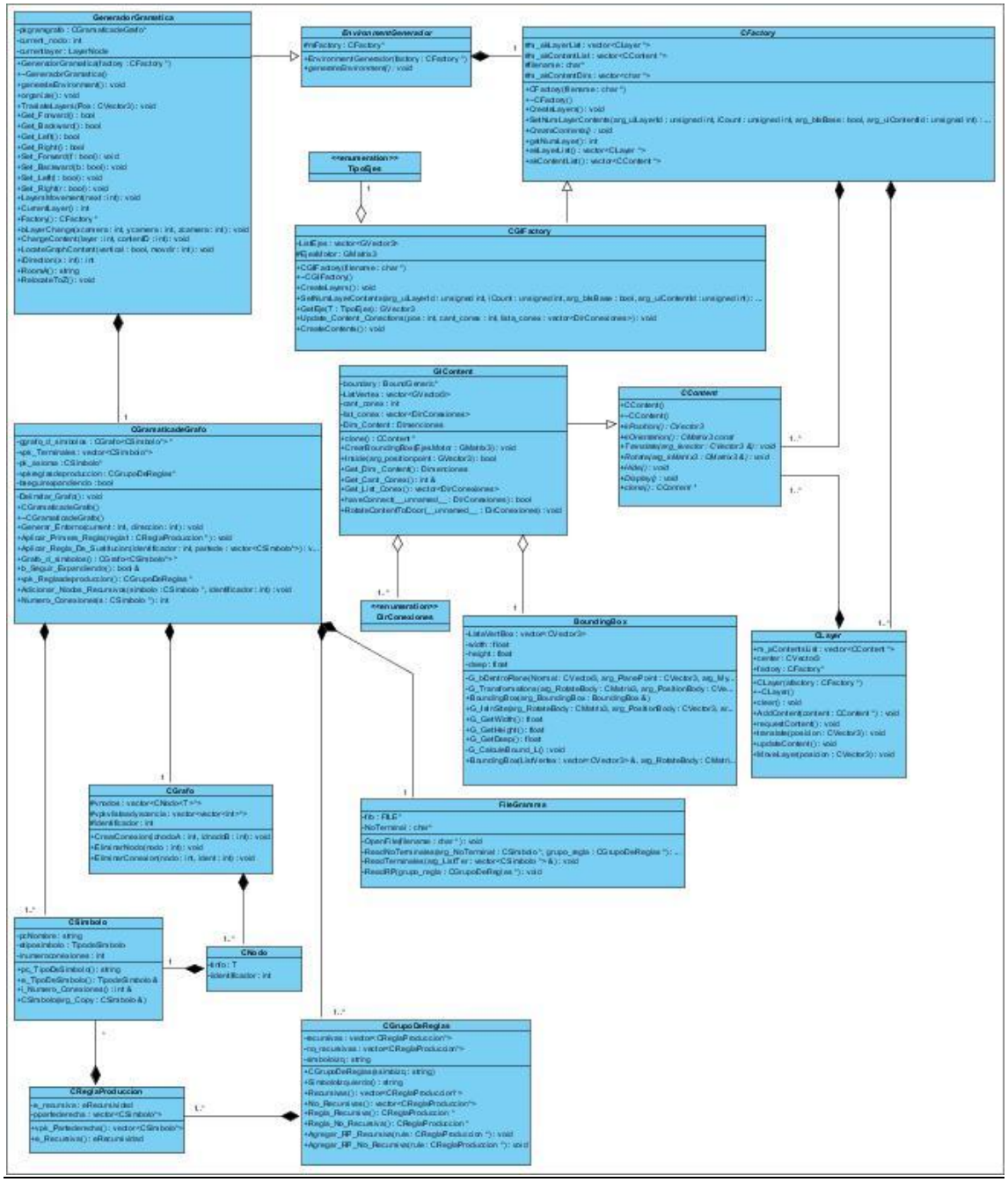

Figura 26 Diagrama de clases del diseño.

3.8. Realización de los casos de uso

Almacenar Contenidos

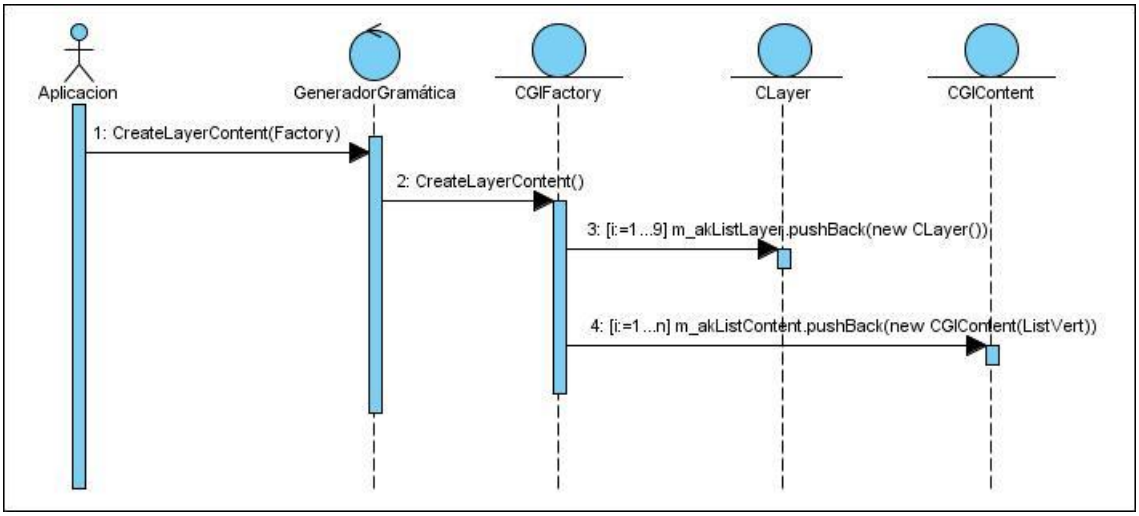

Figura 27 Diagrama de secuencia del caso de uso Almacenar contenidos.

Crear contenidos copias

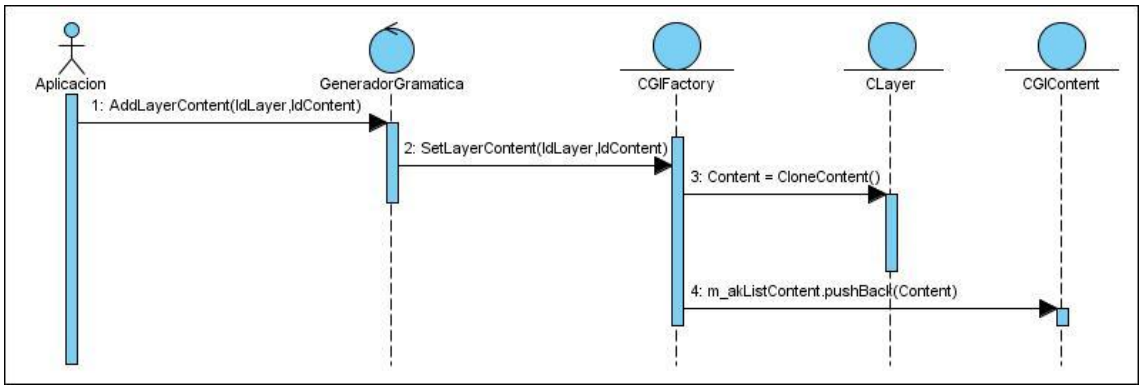

Figura 28 Diagrama de secuencia del caso de uso Crear contenidos copias

Eliminar contenidos copias

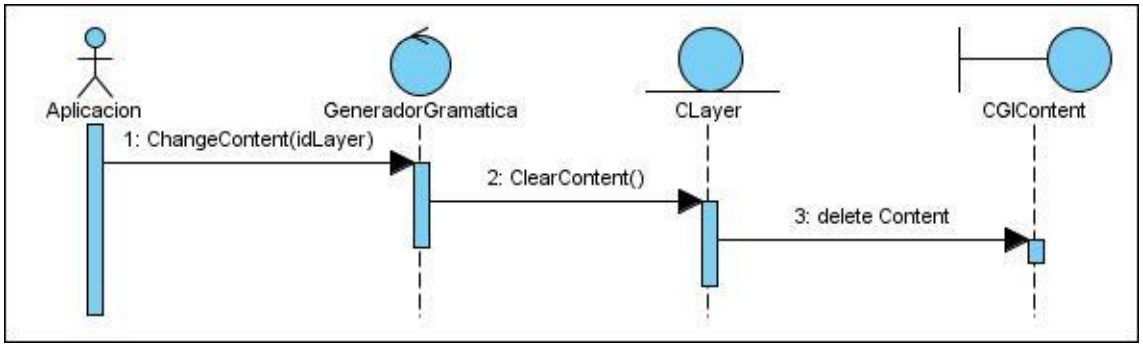

Figura 29 Diagrama de secuencia del caso de uso Eliminar contenidos copias.

Inicializar Entorno

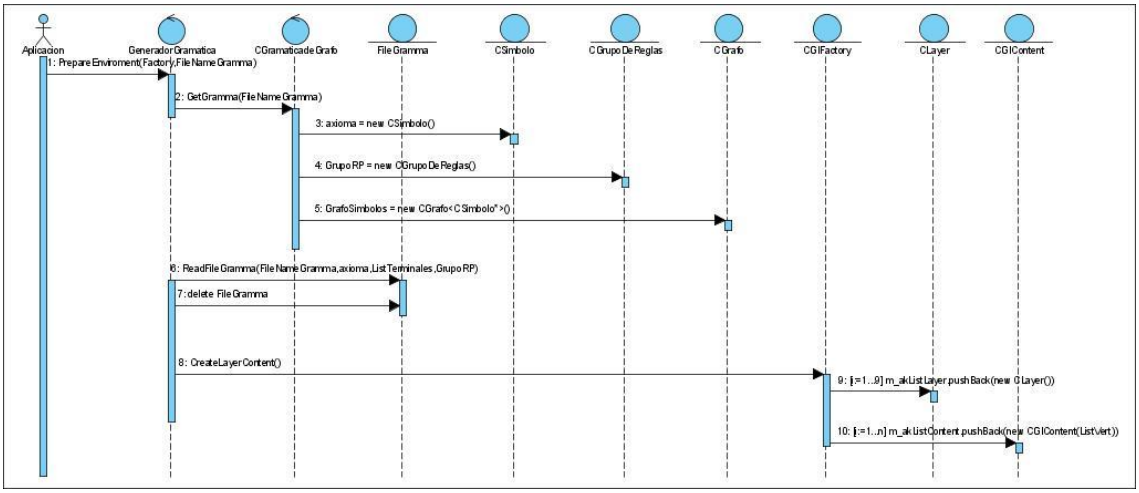

Figura 30 Diagrama de secuencia del caso de uso Inicializar entorno.

Generar **Entorno**

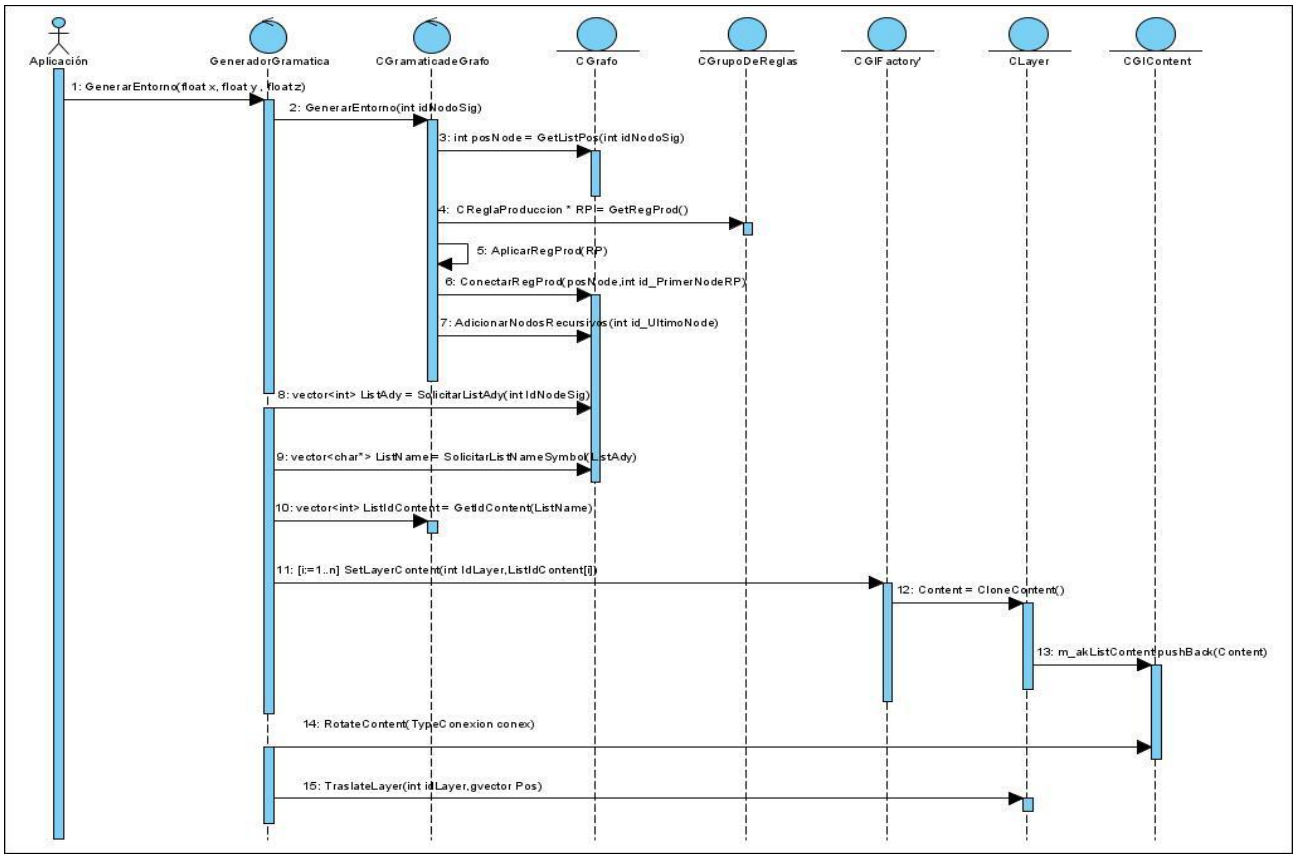

Figura 31 Diagrama de secuencia del caso de uso Generar entorno.

3.9. Patrones de diseño

"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma"

Christopher Alexander

En el presente trabajo se optó por la utilización de patrones ya que nos brindan numerosas ventajas a la hora de construir un software, ahorrando así tiempo y experimentos inútiles, facilitando la comunicación interna entre clases y mejorando la calidad del diseño y la implementación.

Cuando se programa con un enfoque orientado a objeto muchas veces aplicamos patrones generales de software para asignación de responsabilidades *(***GRASP***) que son considerados, más que patrones* propiamente dichos, una serie de "buenas prácticas" recomendable en el diseño de software. Entre los patrones utilizados en el desarrollo del generador de interiores clasificados en esta categoría podemos encontrar:

El Experto en información, el cual se usa en todas las clases de la aplicación ya que es el principio básico de asignación de responsabilidades, el cual nos indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. Así mismo también se utilizan otros patrones como el creador encargado de instanciar nuevos objetos y clases, siempre y cuando cuente con la información necesaria para ello y el polimorfismo, utilizado para redefinir funcionalidades, permitiendo así, asignar el mismo servicio a diferentes objetos. Por último también encontramos el patrón controlador el cual sirve de intermediario entre una determinada interfaz y el algoritmo que la implementa. La puesta en práctica de todos estos patrones tributa a la obtención de un diseño con una mayor cohesión y una disminución del acoplamiento.

Además de la aplicación de los patrones GRASP también se aplicaron en este trabajo *patrones creacionales GOF, los cuales* proponen soluciones para problemas de diseño en los que sólo varía la forma en que se crean los objetos y se usan clases abstractas para encapsular conocimientos sobre clases concretas.

En particular se utilizó el patrón creacional Factory Method, encargado de definir una interfaz para crear objetos, permitiendo que sus subclases decidan qué clase instanciar. Este patrón se usa cuando una clase no puede anticipar qué objetos va a crear o cuando una clase quiere que sus subclases indiquen qué objetos crea.

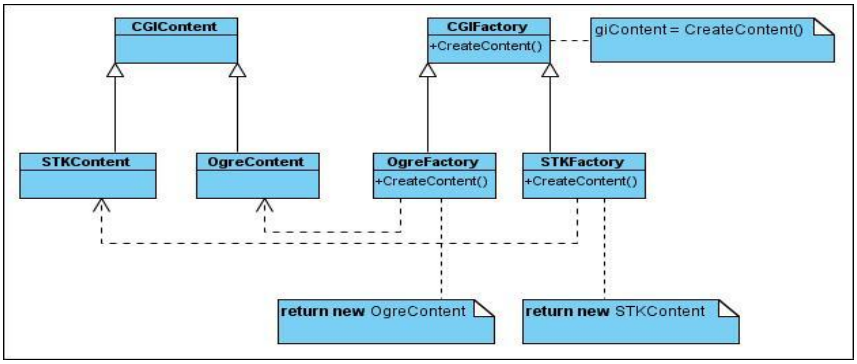

Figura 32 Utilización del patrón Factory Method

3.10. Diagrama de componentes

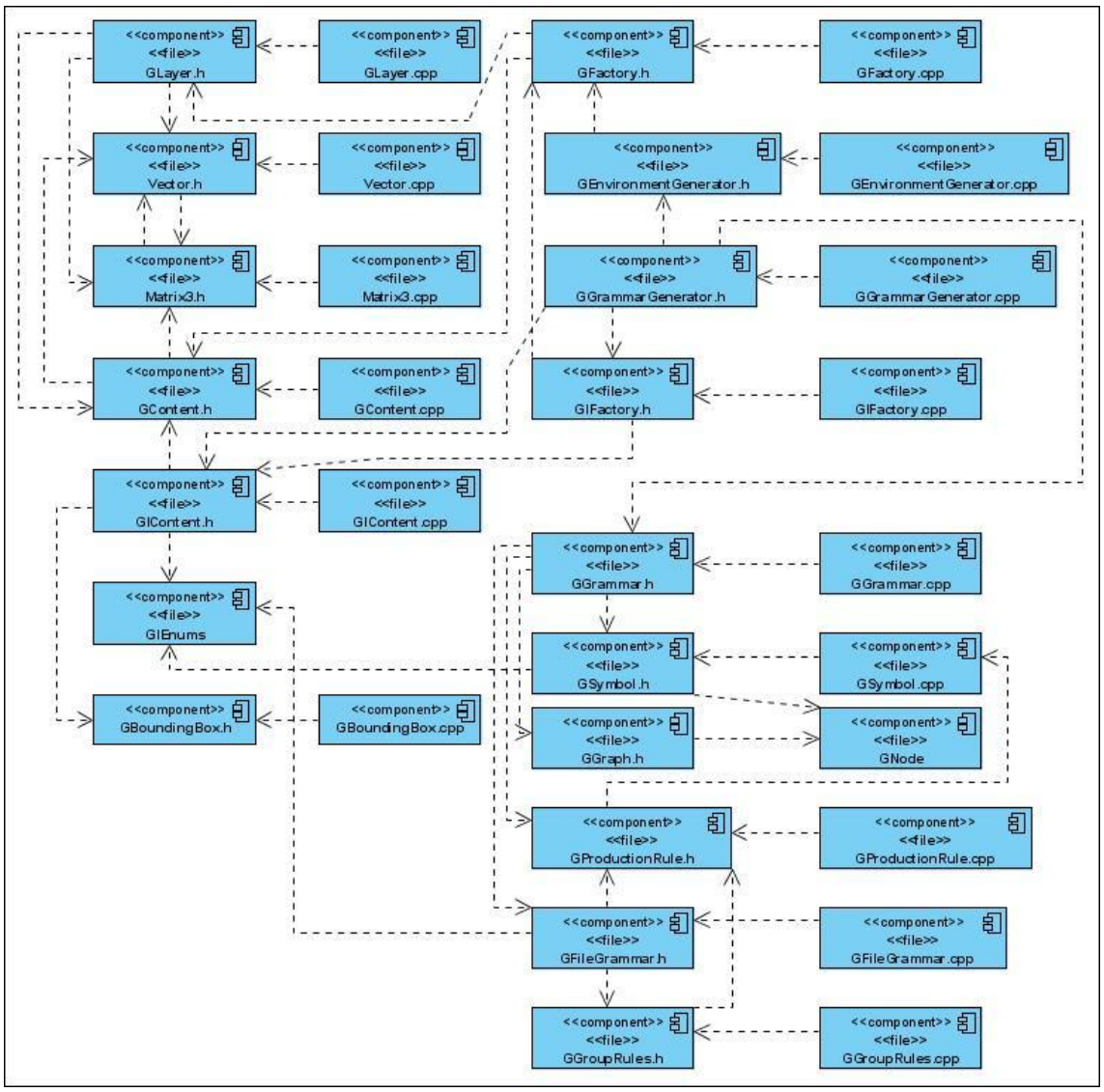

Figura 33 Diagrama de componentes

3.11. Resultados Obtenidos:

El componente elaborado presenta una estructura flexible, que le permite ser integrado con cualquier motor gráfico con soporte para tres dimensiones, lo cual es una de sus principales potencialidades. Fue probado con diferentes tipos de gramáticas y modelos observándose estabilidad en el funcionamiento, independientemente de los modelos o las gramáticas utilizadas. Se ha apreciado que el generador (una vez comenzada la generación comienza a disminuir su velocidad cuando arriba aproximadamente a la sexta habitación)(es más lento luego de comenzar la generación cuando se está arribando aproximadamente a la sexta habitación) pues hasta este momento no se han borrado ninguno de los nodos generados en el principio hacia las que podía trasladarse el usuario, a partir de aquí comienza a funcionar la memoria a corto plazo borrando los elementos que quedan retrasados en el entorno , logrando así simplificar las búsquedas en el grafo por lo que aumenta la velocidad con que se comporta la aplicación.

Para que pueda utilizarse en los diferentes motores gráficos solo deben ser definidas las funcionalidades básicas de rotar los objetos, trasladarlos y cargar los modelos. Además debe asignarse a una estructura creada las direcciones en que se encuentran los ejes del motor gráfico.

También pudo apreciarse que la sensación de realismo del generador será más eficiente con modelos grandes y espaciosos con elementos como mesas, sillas y que además posean puertas reales en las conexiones.

Recomendaciones:

A lo largo del proceso de desarrollo del presente trabajo se realizaron acotaciones para una primera iteración del mismo, además de que no se implementaron ideas que pudieran ser utilizadas para mejorar el sistema, estas se expresan como recomendaciones para futuras iteraciones del mismo.

1. Posibilitar el uso de modelos de más de 4 conexiones.

2. Posibilitar la creación del entorno en tiempo de pre procesamiento utilizando una cota del tamaño del entorno y de esta forma obtener un entorno completo e invariable para cada ejecución, esta podría ser una muy buena opción para un juego de shooter.

3. Crear un generador de contenidos de escena que ubique tantos elementos decorativos, como elementos del juego ya sean vidas u oponentes.

4. Crear una interfaz en QT o cualquier otra herramienta libre para el editor de gramáticas elaborado. Dicho proceso no debe ser complicado pues el editor fue realizado en C++ utilizando solo sus bibliotecas estándares.

5. Realizar una aplicación en la STK versión GREEN pues existían problemas en la versión 8.2 “Newton” de dicha biblioteca por los cuales no pudo probarse totalmente su funcionamiento en la misma.

Conclusiones:

Se comprobó mediante el estudio de los diferentes métodos y técnicas existentes para la generación de entornos que estos constituyen una potente estrategia a utilizar en el desarrollo de videojuegos y por tanto una solución a la generación de entornos virtuales de interiores, infinitos y en tiempo de ejecución, pero dichos resultados pueden ir en detrimento del realismo, si no se cuenta con un buen diseño y buenos componentes de generación aleatoria.

Teniendo en cuenta los resultados arrojados por el presente trabajo se arribó a la conclusión de que para entornos de interiores extensos es más factible utilizar un generador que construya el entorno por partes, cargando en memoria solo los elementos de la escena que necesitan ser visualizados, en lugar de hacerlo de la forma tradicional, cargando el entorno completo en memoria. Esto permite simplificar la cantidad de polígonos a visualizar en el entorno además de ahorrar tiempo y esfuerzo en el diseño de los elementos que formarán parte del mismo.

Bibliografía

Adams, David. 2002.*Automatic Generation of Dungeons for Computer Games.* 2002.

Alvarado., Lidia Ortega. 2007.*Avances en Sistemas de Información.* 2007.

Casals, Pere. 2005.*Taller de memoria: ejercicios prácticos.* s.l. : Horsori Editorial, 2005.

Dawson, Michael. 2004.*Begining C++ Programming.* s.l. : Stacy L. Hiquet., 2004.

Dymchenko, Lev. 2004.*Polygonal Technology: Weaknesses and Alternative.* 2004.

Ehrig, Hartmut, Prange, Ulrike and Ehrig, Karsten. 2006.*Tutorial on Fundamentals on algebraic graph transformation.* Berlin : s.n., 2006.

Gilabert, Damián de la Rosa and García, Jose Manuel Pereira.*Características topológicas y geométricas de objetos 3D.*

Granados, Gonzalo Tirado. 2007.*Introducción a OGRE 3D.* 2007.

Greuter, Jeremy Parker Stefan. 2001.*Real-time Procedural Generation of Pseudo Infinite Cities. .* Melbourne , Victoria , Australia : s.n., 2001.

Haahr., Mads. 1999.*Introduction to randomness and random number.* 1999.

Hahn, Evan, Bose, Prosenjit and Whitehead, Anthony. 2006.*Persistent Realtime Building Interior Generation.* Carleton : Association for Computing Machinery, Inc., 2006.

Lipp, Markus, Wonkay, Peter and Wimmer, Michael. 2006.*Interactive Visual Editing of Grammars for Procedural Architecture.* Arizona : s.n., 2006.

Lozano, Miguel and Calderón, Carlos. 2003.*Entornos virtuales 3D clásicos e inteligentes.* Valencia : s.n., 2003.

Madrigal, Omar Correa, Gutiérrez, Julio Oscar and González, Giraldo Andrés. 2009.*Generación de entornos virtuales en tiempo de ejecución.* Habana : UCI (Polo de Realidad Virtual), 2009.

Madrigal., Omar Correa. 2010.*Informe final de Proyecto.* Ciudad de La Habana : s.n., 2010.

Miller, George A. 1956.*The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information.* s.l. : Psychological Review, 1956.

P.D'Amato, Juan. 2009.*Estrategias de Optimización .* 2009.

Polack, Trent. 2003.*Focus On 3D Terrain Programming.* Ohio : Premier Press, 2003.

Pubb, Greg. 2003.*Terrain Engine usin C++ y Directx 9. .* Massachusetts. : Charles River Media., 2003.

Rozenberg, Grzegorz. 1997.*Handbook of Graph Grammars and computing by graph transformation.* Singapore : World Scientific Publishing Co. Pte. Ltd., 1997.

Stewart, Nigel. 2004.*Beyond the horizon.Computer generated,three-dimensional, infinite virtual worlds without repetition in real time.* Australia : s.n., 2004.

Taylor, Ann. 2003.*Introducción a la psicología: una visión científico humanista.* s.l. : Pearson Educación, 2003.

Teixeira, Bruno and Lamothe, Andre. 2002.*Game Programming all in one.* s.l. : Stacy L. Hiquet, 2002.

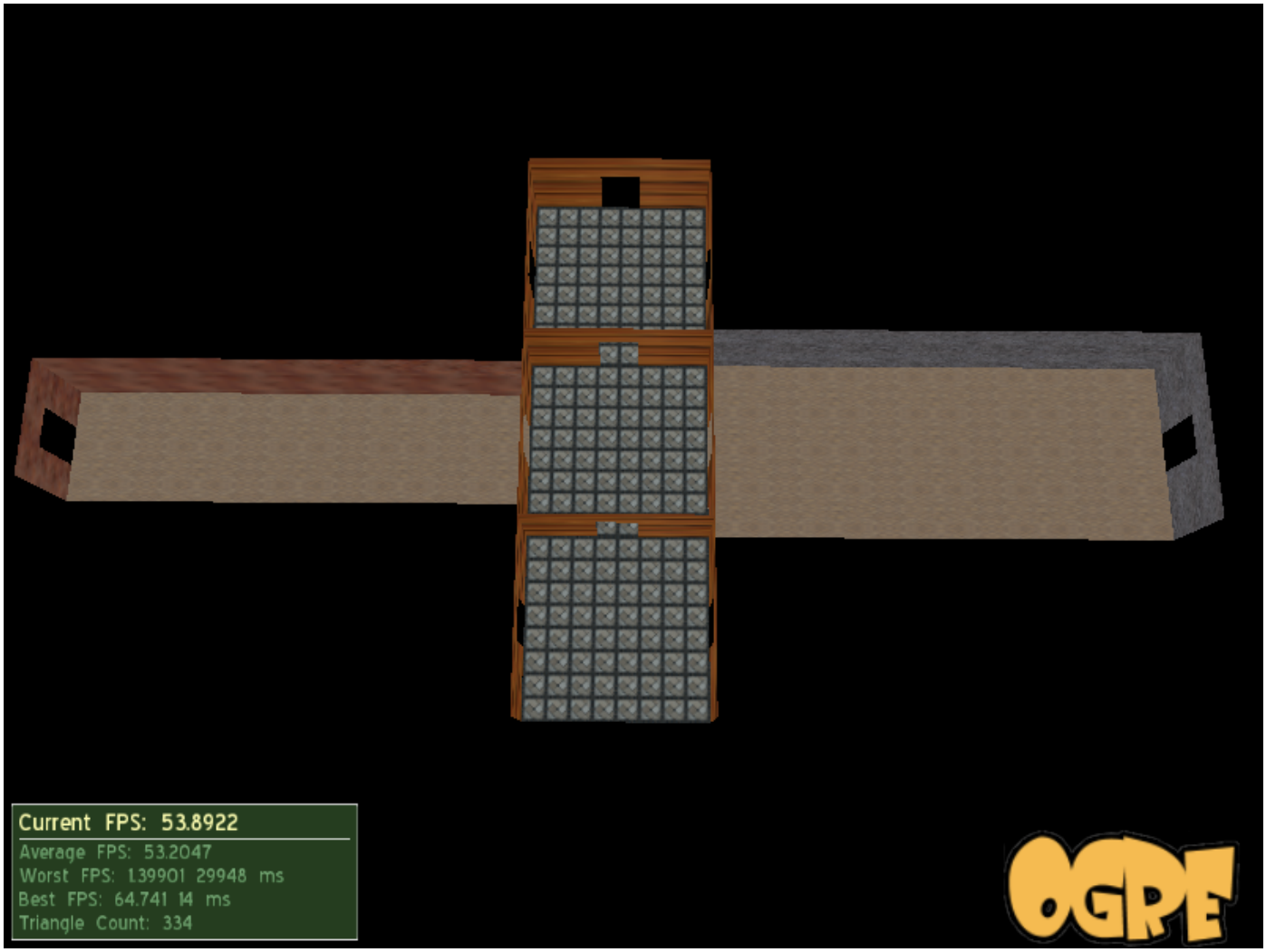

Figura 34 Vista exterior del generador con cuarto de cuatro conexiones en el centro conectado con otros cuartos (verticales en la imagen) y pasillos que se aprecian en la horizontal.

Figura 35 Vista interior del generador mirando hacia el pasillo, esta imagen y la anterior se obtuvieron utilizando en el generador la siguiente gramática.

No_Terminal:

Secciones

Terminales:

Cuarto1 (4 conexiones)

Pasillo1 (2 conexiones)

Pasillo2 (2 conexiones)

Reglas de producción:

Secciones ->Pasillo1 – Cuarto1

Secciones ->Pasillo2 – Cuarto1

Secciones -> Cuarto1

Figura 36 Cuarto de 3 conexiones conectado con tres cuartos de dos conexiones

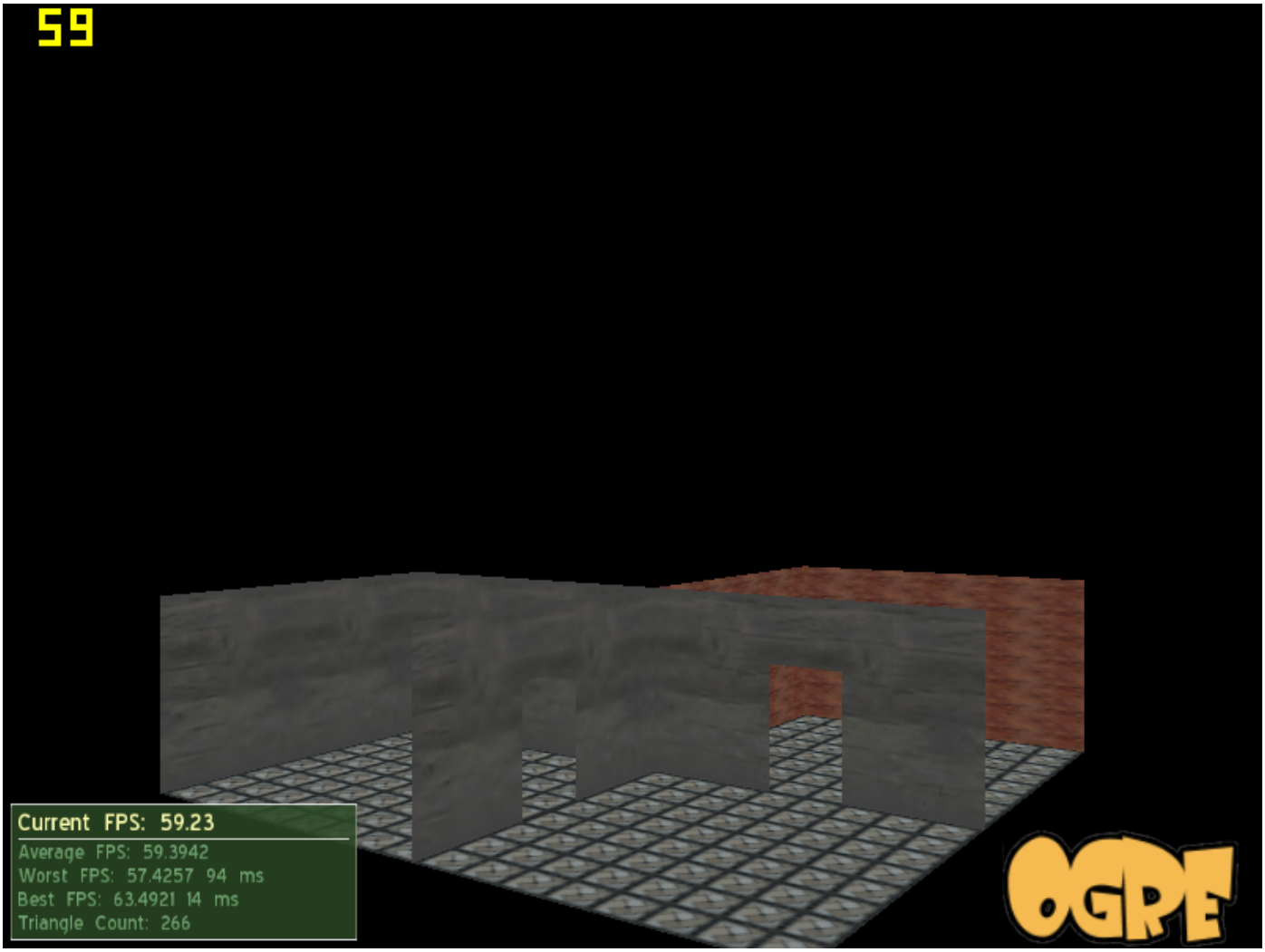

Figura 37 Cuarto de dos conexiones conectado con un cuarto de dos conexiones y uno de tres conexiones. El objetivo principal de esta imagen y la anterior es mostrar como el generador ubica los contenidos en las direcciones de las puertas. Estas imágenes fueron generadas utilizando la siguiente gramática.

No_Terminal:

Secciones

Terminales:

Cuarto3 (3 conexiones)

Cuarto2o (2 conexiones)

Reglas de producción:

Secciones ->Cuarto2o – Cuarto3

Figura 38 Vista entorno donde se aprecian las distintas direcciones en las que se puede trasladar el usuario.

Glosario de términos.

Algoritmo formación por fallas: Algoritmo que pertenece a la técnica de generación fractal de terreno. Proceso que genera fallas en el terreno, obtiene como resultado un mapa de alturas que representa terrenos con medianas alturas.

Ambliopía: Es la pérdida de la capacidad de un ojo para ver los detalles y es la causa más común de problemas de visión en los niños. Ocurre cuando el cerebro y los ojos no funcionan juntos apropiadamente. En las personas con ambliopía, el cerebro favorece a uno de los ojos. El ojo preferido tiene visión normal, pero debido a que el cerebro ignora al otro ojo, la capacidad de visión de la persona no se desarrolla normalmente. Entre las edades de 5 y 10 años, el cerebro detiene su crecimiento y la afección se vuelve permanente.

Bounding volume: Cuerpo de poca complejidad que se utiliza para encerrar otros cuerpos de mayor complejidad, con el objetivo de minimizar cálculos respecto a este último, por ejemplo, determinación de si es visible, o detección de colisiones contra este.

Biblioteca: Conjunto de subprogramas utilizados para desarrollar software. Las bibliotecas contienen código y datos, que proporcionan servicios a programas independientes, es decir, pasan a formar parte de estos. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular.

Capas: elementos contenedores donde se ubican los contenidos del entorno.

Conexiones: espacios que comunican a los contenidos de la escena por ejemplo puertas o aberturas que representen las mismas.

Contenidos: contienen las geometrías utilizadas además del número de conexiones y las direcciones en las que estas se encuentran.

Cono de visión (frustum): Representa el volumen de visión de la cámara utilizando una proyección en perspectiva, este volumen es similar a una pirámide truncada, está formado por los 6 planos que definen los límites de la cámara: cercano (near), lejano (far), izquierdo (left), derecho (right), arriba (top) y abajo (bottom).

Entorno virtual: Simulación de mundo o entorno, denominado virtual, en el que el hombre interacciona con la máquina en un entorno artificial semejante a la vida real.

Frustum culling: Técnica de selección de objetos visibles determinando si están o no en el interior del frustum de la cámara.

Realidad Virtual: Representación de escenas o imágenes de objetos producidos por un sistema informático, que da la sensación de su existencia real.

Real time: En gráficos por computadoras, las aplicaciones en tiempo real son aquellas que van generando los fotogramas a medida que son necesarios durante la propia corrida de la aplicación. El tiempo real se consigue cuando el tiempo de respuesta de la aplicación es lo suficientemente corto como para que la percepción del fenómeno se corresponda con fidelidad al sistema real.

Topología: Rama de las matemáticas que estudia las propiedades de las figuras con independencia de su tamaño o forma.

Índice de Figuras

Índice de figuras