CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Generación Procedimental de Tramas Argumentales

Autor: Borja Molina Rueda

Director: Antonio Latorre de la Fuente

MADRID, JULIO 2016

Contenido

Resumen

Español

En el siguiente documento describiremos las técnicas de generación procedimental de contenido utilizadas para construir un sistema de creación de tramas argumentales con restricciones en videojuegos. Concretamente, nuestro programa desarrolla misiones o *quests* para juegos adscritos al género RPG (*Role-Playing Game*).

La herramienta implementada hace uso de un planificador de orden parcial o POP para buscar dentro de un espacio inicial definido por el usuario de la aplicación una solución que satisfaga el objetivo de la misión, además de usar el mismo planificador para hallar caminos conflictivos que el jugador pudiera escoger e invalidar el plan encontrado y buscar planes de contingencia alternativos.

English

The following document will describe the procedural content generation techniques used to build a system of restricted plotlines on video games. Specifically, our program develops missions or quests for RPG (Role-Playing Game) games.

The implemented tool uses a partial order planner or POP to search a solution to fulfill the objective of the mission within an initial space defined by the application. It also uses the same planner to find conflicting paths the player could choose and invalidate the plan to found and seek alternative contingency plans.

Introducción

Definición de PCG en videojuegos

La generación procedimental de contenido (PCG por sus siglas en ingles) en videojuegos se define como la creación automática del contenido de un juego haciendo uso para ello de distintos algoritmos [1]. Esto se contrapone con las técnicas clásicas de generación de contenido donde la tarea recae en los distintos artistas y diseñadores que conforman el equipo desarrollador que producen a mano los diferentes recursos que utiliza el videojuego.

Este crecimiento del uso de PCG dentro de la industria del videojuego se debe a ciertas ventajas que estos algoritmos pueden aportar que de otra forma serían imposibles:

- El contenido desarrollado de esta manera puede ser creado en tiempo real, liberando recursos de la máquina como la memoria usada a cambio de depender de otros en mayor medida como puede ser más capacidad de procesamiento. Claro exponente de esto es *Elite*, simulador espacial publicado en la década de los 80 que con la tecnología de la época conseguía poner en juego varias galaxias.
- Ante el reciente auge de los servicios de venta digital de videojuegos y por ende el abaratamiento de la distribución de estos al saltarse intermediarios han surgido compañías independientes con menor presupuesto (denominadas en el sector compañías *indies*). Estas técnicas han permitido grandes resultados a la hora de

generar material jugable con una menor cantidad de personas involucradas en el desarrollo. Uno de los ejemplos más actuales es *No Man's Sky* desarrollado por *Hello Games* para PC y PS4, que con un presupuesto limitado han sido capaces de generar la friolera de 18 trillones de planetas con distinta fauna y flora cada uno.

– Siguiendo con la idea del contenido generado en tiempo real, otra de las posibles ventajas es la adaptación de este contenido al usuario analizando su estilo de juego y creando los posibles gráficos, sonidos, enemigos, misiones, … en concordancia al perfil del jugador que se haya calculado. EL *Left4Dead* de *Valve* hace uso de un algoritmo que ajusta el juego al ritmo de los jugadores [14].

Clasificación de PCG en videojuegos

Según la clasificación propuesta por M. Hendrikx et al. [2] podemos dividir las técnicas de PCG en videojuegos en seis categorías distintas ateniéndose al tipo de recurso generado:

1. **Game Bits**: llamados así por ser considerados como la unidad mínima de contenido, nos encontramos con distintos elementos como sonidos, texturas, construcciones o vegetación. Buenos ejemplos de esto son los videojuegos *Borderlands* (Gearbox Software, 2009) con su generación procedimental de armas o *.kkriefer* (.theprodukkt, 2004) que consigue ser un videojuego con gráficos 3D ocupando tan solo 96 KB.

Ilustración 1:Imagen de Borderlands, Gearbox Software (2009)

2. **Game Space**: es el entorno donde se desarrolla el juego, entendiendo por esto desde la construcción del mapa de una mazmorra hasta la generación de la orografía de un terreno por donde el jugador va a navegar. En el plano comercial se

ha prodigado mucho este tipo de técnicas en generación de mazmorras como en *Diablo* (Blizzard, 1996) o *Spelunky* (Derek Yu, 2009).

3. **Game Systems**: esta categoría engloba estructuras más complicadas formadas por subelementos que se relacionan entre sí. Suele englobar comportamientos de agentes con su entorno como ciudadanos de una ciudad o animales en un ecosistema, como nuestro grupo de enanos en *Dwarf Fortress* o el sistema económico de los distintos planetas en *Elite*.
4. **Game Scenarios**: no confundir con la categoría Game Space. Aquí se engloban todo el contenido relacionado con los eventos, como pueden ser los puzles o la narrativa. Es menos habitual que las expuestas anteriormente porque requiere de técnicas de procesamiento del lenguaje natural entre otras. Uno de los pocos ejemplos que tenemos en la industria de esto es en *Left4Dead* que controla el ritmo del juego con la aparición de enemigos y otros eventos. En el mundo de la investigación se puede hablar de *Façade* [7] que utiliza una serie de "eventos de la trama" para guiar la narrativa.
5. **Game Design**: este es el caso más general de contenido ya que habla de los elementos que componen la descripción del juego en sí. Las reglas, el tema o la ambientación forman parte de este apartado. A día de hoy no existen juegos comerciales que hagan uso de este tipo de contenido.
6. **Contenido Derivado**: por último, hablamos de los elementos más allá del juego o el metajuego propiamente dicho. Contenido como un vídeo del *gameplay* de la partida jugada o las listas de puntuación se adscriben a esta descripción. Con el florecimiento de las redes sociales y los productos transmedia este contenido está a la orden del día. Recientemente se ha podido observar un mayor interés por este contenido, como el añadido del botón *Share* en los mandos de la consola de *Sony PlayStation 4* que permite compartir contenido directamente del juego a tus redes sociales.

Este trabajo se engloba en la cuarta categoría, ya que utilizamos la estructura de misión clásica de los RPG para crear un contenido tanto jugablemente correcto y completo. Estudiando todas las posibles acciones que pueda realizar el jugador, como narrativamente interesante, usando para ello el estudio de los *Tropes*, patrones aplicados en una gran cantidad de obras narrativas y que guiarán al algoritmo para buscar estructuras argumentalmente interesantes.

Trabajos Previos

Historia del PCG en videojuegos

El PCG en los videojuegos nació debido a la necesidad de comprimir los datos ante las limitadas capacidades de almacenamiento de los dispositivos de juegos [3].

Aunque aún hay discusión sobre ello, la mayoría de autores señalan a *Akalabeth* como el primer videojuego que hizo uso de este conjunto de técnicas. Este videojuego de 1980 creado por Richard Garriott usaba un número proporcionado para el usuario como semilla

para decidir parámetros del juego como el valor inicial de los atributos o el diseño de las mazmorras [4]. Ese mismo año llegaría *Rogue*, un RPG donde todos los niveles eran generados procedimentalmente, lo que permitió el añadido de "muerte permanente" como una característica principal en la mecánica de juego: si nuestro protagonista reduce a 0 la variable que representa su salud el juego se resetea de nuevo perdiendo el progreso obtenido hasta el momento. Esto es posible gracias a que cada partida es distinta y no se hastía al jugador teniendo que repetir ninguna sección, uno de los grandes logros conseguidos por las técnicas de PCG aplicadas en videojuegos. Este juego sería pionero creando un subgénero entero de los RPG denominado en su honor *roguelike*.

Otro gran exponente de videojuego que hace uso de las ventajas de estos algoritmos en esta época es *Elite*, un simulador espacial desarrollado por Ian Bell y David Braben donde pilotamos una nave que surca un espacio generado procedimentalmente compuesto por 8 galaxias con 256 sistemas solares cada uno que tienen a su vez entre 1 y 12 planetas [3] lo que nos da entre 2048 y 24576 planetas distintos, sin duda unas cifras que requerirían de una gran cantidad de tiempo y medios de haberse generado con las técnicas más tradicionales.

Durante la década de los 90 hubo muy pocos ejemplos de videojuegos comerciales que hicieran uso de algoritmos procedimentales en la creación de su contenido, que quedó relegada a subgéneros muy concretos como la estrategia o los mencionados roguelikes. Una de las excepciones más famosas fue *Diablo*, desarrollado por la compañía *Blizzard* en 1996 para PC [5] donde estas técnicas son usadas tanto para crear las mazmorras donde el juego tiene lugar cómo los atributos de los ítems que el jugador consigue durante la partida.

Desde el 2000 y con énfasis en los últimos años estamos siendo testigos de un aumento considerable de videojuegos que utilizan este tipo de técnicas para crear algunos de sus elementos, desde la generación de escenarios donde se desarrolla el juego en *Spelunky* hasta la generación de armas para el sistema de combate como en *Borderlands* [2] pasando por el tema que nos atañe en este trabajo con la generación de misiones en el videojuego *The Elders Scrolls V: Skyrim* con su sistema *Radiant Story* de creación de quests dinámicas [6].

Fuera de la industria comercial es interesante hablar de *Façade*, videojuego desarrollado por Michael Mateas y Andrew Stern en 2005 [7]. El jugador toma el papel de amigo invitado al piso de una pareja treintañera, donde a lo largo de la noche se sucederán varios descubrimientos y situaciones dramáticas. Sus autores lo definen como drama interactivo y actualmente es uno de los mayores exponentes de lo que se puede conseguir con PCG más complicado, incluyendo *Game System* y *Game Scenarios* de la clasificación expuesta en el apartado anterior.

Narrativa Interactiva

P. Weyhrauch define el drama interactivo [8] como *«La presentación por ordenador de mundos altamente ricos en contenido, los cuales están poblados por personajes complejos y dinámicos, y también por el usuario, cuya experiencia es formada en este mundo por un destino dramático»*. Si analizamos esta descripción, podemos observar dos piezas fundamentales para el desarrollo de historias interactivas: por un lado tenemos las relaciones entre el usuario y las entidades que definen su entorno y por otro una serie de eventos por los que el jugador tiene transitar hasta llegar a una conclusión dramática para provocar en este una serie de emociones.

Como hemos visto anteriormente, en los videojuegos comerciales no se ha prodigado este tipo de técnicas avanzadas de PCG, así que tendremos que movernos al mundo de la investigación para ver las aportaciones más interesantes al tema.

Hemos mencionado que uno de los componentes necesarios para construir un drama interactivo son los elementos interactivos que pueblan el mundo. Los sistemas multiagentes pueden ayudarnos ya que su enfoque es muy cercano al problema que se quiere tratar. B. Mac Namee [9] define ciertas características deseables que estos sistemas deberían cumplir. Algunos ejemplos son los siguientes:

- Los agentes deben mostrar un comportamiento creíble ante un amplio rango de situaciones.
- El sistema de agentes debe poder ejecutar en tiempo real.
- El sistema no tiene que tener unos requerimientos de memoria no realistas.
- El sistema debe ser relativamente fácil de usar para usuarios no programadores.

Tratando más acerca de los agentes en sí, encontramos varios trabajos interesantes. Una de las tendencias actuales es aprovechar avances en la rama de la psicología para intentar implementar agentes que simulen un comportamiento derivado de un arquetipo de personalidad [10] usando para ello un modelo muy estudiado denominado OCEAN [11] que divide esta en 5 parámetros: apertura al cambio, responsabilidad, extraversión, afabilidad e inestabilidad emocional. Incluso la autora S. Bura propone en su presentación en la GDC una extensión de 6 subdivisiones por cada categoría del OCEAN, haciendo un total de 20 parámetros. Otro modelo usado para desarrollar agentes verosímiles es tanto el modelo OCC como el PAD [12], que intentan modelar las emociones humanas. El primero divide las emociones en 22 categorías, mientras que el segundo modela las emociones como un espacio tridimensional y los efectos de una acción en el estado emocional como un vector.

Se pretende conseguir con todos estos modelos tanto que el jugador tenga mayor predisposición a empatizar con los NPC (*Non-Player Characters*, personajes no jugables controlados por el programa) como que las reacciones de aquellos sean más creíbles.

El otro componente de la narrativa interactiva son los eventos dramáticos y cómo conseguir una narración en un medio interactivo haciendo uso de la capacidad de elección del jugador. Aquí podemos ver aproximaciones muy distintas.

Uno de los pocos ejemplos en videojuegos comerciales es la entidad conocida como *El Director* [14] en el videojuego *Left4Dead* la cual controla el flujo de ítems y enemigos dependiendo del ritmo de la partida y el comportamiento de los jugadores. Esto es el nivel más básico y realmente no nos metemos aún en estructuras narrativas, solo manejamos la jugabilidad en sí.

F. Peinado propone utilizar casos diseñados por humanos como base de conocimiento para crear nuestras propias soluciones [15] haciendo uso para ello de un CBR (*case based reasoning*). Otros autores hacen uso de una aproximación más parecida a la nuestra como en el caso de M. Riedl et al. [16] donde se implementa un POP (*Partial-Order Planner*) para poder estudiar todos los posibles caminos hasta un objetivo, introduciendo la idea de *Objetivos del autor*, metas que se pretenden alcanzar no para conseguir el objetivo final, sino para dar dramatismo a la historia generada.

Desarrollo

Diseño

Este proyecto consiste en la realización de una herramienta capaz de generar misiones para un videojuego de corte RPG. Definimos una misión cómo una serie de acciones parcialmente ordenadas que tras su consecución nos permite llegar a un objetivo concreto. La aplicación está desarrollada con dos objetivos fundamentales en mente.

Ámbito generalista

Esta herramienta se desarrolla con la intención de servir de ayuda a cualquier diseñador de videojuegos que quiera crear quests en su obra y, por ende, su implementación ha sido realizada desacoplando dos partes: la primera genera la misión y la traduce a un formato XML, que puede ser consumida por la segunda parte que lo interpreta y genera dichas misiones en el juego correspondiente. Para poder generar misiones, se han establecido una serie de elementos comunes a todos los juegos del género (personajes, objetos, escenarios, …) mientras que el diseñador puede establecer la descripción de su universo mediante las distintas etiquetas y reglas de las que se hablaran en profundidad en apartados posteriores.

Generación estática de la trama

La creación de las tramas no se realiza en tiempo real, si no que éstas ya están generadas una vez el juego se inicia. Ciertas ventajas como la adaptabilidad del contenido al usuario analizando su comportamiento no pueden ser posibles bajo este formato. Esta decisión se ha tomado tras analizar una amplia variedad de herramientas para generar contenido para RPGs (la mayoría de videojuegos comerciales de este género dan acceso libre a herramientas para desarrollar modificaciones a su producto) y llegar a la conclusión que normalmente este tipo de contenido es el único que se puede generar con esas herramientas.

Para conseguir la sensación de que la trama se adapta a las decisiones del jugador se ha optado por un enfoque muy cercano al de M. Riedl et al. [16] en el cual además de calcular una posible misión que cumpla la condición de poder ser realizable en el espacio de hechos y reglas que hemos creado, se crean planes de contingencia en los arcos en los cuales el jugador puede tomar una decisión distinta de la establecida y dar al traste con el plan original.

Elección del lenguaje y del entorno

A la hora de empezar el desarrollo del proyecto se analizan varios lenguajes de programación distintos en los cuales poder implementar el programa. Después de barajar las distintas opciones, nos decantamos por C# [17], un lenguaje desarrollado por *Microsoft* orientado a objetos y estáticamente tipado, por varias razones: es un estándar dentro de los lenguajes utilizados para el desarrollo de videojuegos ya que Unity [18], uno de los motores más extendidos para la creación de videojuegos, hace uso de ello. Sumado esto a la gran documentación oficial que se puede encontrar en la web [17] y la facilidad de desenvolvernos con el lenguaje ya que sigue el paradigma orientado a objetos con el que estamos familiarizados gracias al enfoque de nuestra formación en lenguajes similares nos decidimos por su uso. Para el desarrollo nos inclinamos por la solución oficial que proporciona Microsoft: *Microsoft Visual Studio*, un entorno de desarrollo integrado que además de aportar todas las facilidades para trabajar con el lenguaje escogido, nos ayudara a otras tareas posteriores como la creación de esquemas XML.

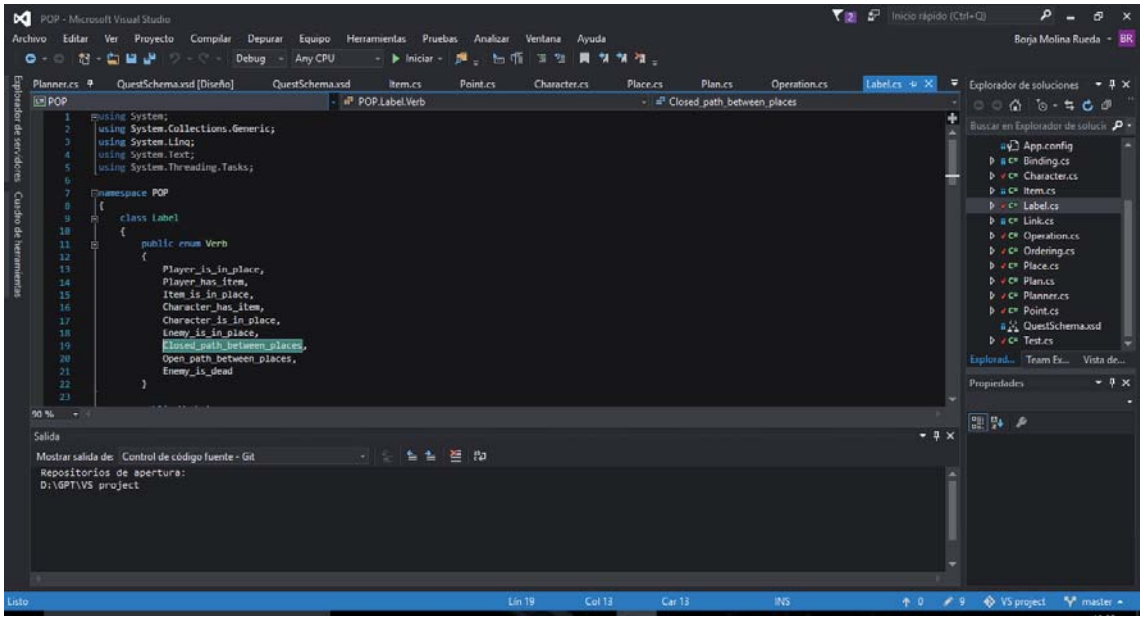

Ilustración 2: Muestra de interfaz de Microsoft VS

Algoritmo de planificación de orden parcial

A continuación pasamos a describir el algoritmo desarrollado en la herramienta para la generación de tramas. Se trata de una simplificación de UCPOP [19] una implementación clásica de un planificador de orden parcial. A diferencia de los planificadores anteriores,

como STRIPS que únicamente son capaces de generar planes completamente ordenados donde cada acción precede y es precedida exclusivamente por otra hasta llegar al estado final, nuestro algoritmo nos permite realizar estructuras más complejas donde se pueden representar varias operaciones que entre ellas no están ordenadas (a la hora de ejecutar el plan, pueden realizarse en cualquier orden temporal siempre que se cumplan el resto de restricciones).

Representación de los elementos

Para poder describir el planificador implementado, primero se debe explicar los elementos que componen un plan:

Etiqueta

Unidad mínima de información en nuestro sistema, consta de cuatro parámetros: un tipo que describe la etiqueta (el jugador está en un lugar, el personaje tiene un objeto), dos variables para aportar información (personaje, lugar, objeto, …) y un parámetro binario que indica si la etiqueta tiene valor afirmativo o negativo.

Tipo	**Descripción**
Player_is_at_place	El jugador está en un escenario.
Character_is_at_place	El personaje no jugador está en un escenario.
Enemy_is_at_place	El enemigo está en un escenario
Item_is_at_place	El objeto está en un escenario.
Open_path_between_places	El camino entro dos escenarios está abierto.
Player_has_item	El jugador tiene en su poder un objeto.
Character_has_item	El personaje no jugador tiene en su poder un objeto.
Enemy_is_dead	El enemigo está muerto.

Tabla 1: Ejemplos de tipos de etiqueta

Operación

También denominado regla o acción, describe una acción y vienen definidas por dos series de etiquetas. La primera son las precondiciones necesarias para poder ejecutar la operación, y la segunda describen el efecto que tiene realizar dicha acción.

Precondiciones	**Operación**	**Efecto**
Player_is_at_place(Forest,,true) Player_is_at_place(Desert,,false) Open_path_between_places(Forest,Desert,true)	El jugador se mueve desde el bosque hasta el desierto	Player_is_at_place(Forest,,false) Player_is_at_place(Desert,,true)
Player_is_at_place(Forest,,true) Item_is_at_place(Sword,Forest,true) Player_has_item(Sword,,false)	El jugador coge la espada en el bosque	Item_is_at_place(Sword,Forest,false) Player_has_item(Sword,,true)
Player_is_at_place(Desert,,true) Character_is_at_place(Alice,Desert,true) Player_has_item(Sword,,true)	El jugador entrega la espada a Alice en el desierto	Player_has_item(Sword,,false) Character_has_item(Sword,,true)

Player_is_at_place(Forest,,true) Enemy_is_at_place(Orc,Forest,true) Player_has_item(Sword,,true) Enemy_is_dead(Orc,,false)	El jugador mata al enemigo Orco en el bosque con la espada	Enemy_is_at_place(Orc,Forest,false) Enemy_is_dead(Orc,,true)

Tabla 2: Ejemplos de operaciones

Conexión causal

Se describe como una relación entre dos operaciones y una etiqueta, siendo esta última consecuencia de una de las reglas (denominada proveedora) y precondición de la otra, llamada receptora.

Ordenamiento

Restricción temporal entre dos operaciones, describiendo que la primera debe realizarse antes de la segunda.

Plan

Un plan en nuestro sistema se define por tres conjuntos. El primero engloba operaciones que hay que llevar a cabo para realizar con éxito dicho plan. El segundo contiene una serie de ordenamientos y por último una lista de conexiones causales entre las operaciones.

Planificador

Estructura que contiene los elementos necesarios para calcular el plan. Consta del plan que se está resolviendo, una agenda de objetivos que faltan por cumplir para que el plan sea realizable y una lista de operaciones que se pueden usar para completar dicho plan.

Descripción del algoritmo

La implementación del algoritmo es básicamente la descrita en UCPOP con ligeras modificaciones:

1. **Inicialización**: el planificador arranca con el plan inicial, que consta de dos operaciones: inicio, que no tiene precondiciones y como efecto describe el estado inicial; y final, que no tiene efecto y como precondiciones tiene los objetivos que se buscan en forma de etiquetas. La agenda se inicializa a su vez con estas precondiciones del final y se añaden todas las operaciones al planificador que puedan ser usadas para resolver el problema.
2. **Verificar consistencia del plan**: en este paso se comprueba que ninguna operación añadida colisiona con las conexiones causales que ya tenemos. Se dice que una operación colisiona con una conexión causal cuando uno de los efectos de la acción es contraria a la etiqueta de la conexión. Es decir, las dos etiquetas son del mismo tipo y contienen la misma información exceptuando que sus valores afirmativos son distintos (si una es afirmativa la otra es negativa y viceversa). Si esto ocurre, se busca una posible solución:
    a. **Promocionar**: se añade un ordenamiento que restrinja temporalmente la acción a posteriori de la receptora de la conexión.

b. **Democionar**: igual que el anterior, pero en este caso se obliga a que la operación sea realizada antes de la proveedora definida en la conexión.

Si ninguna solución es aplicable entonces el algoritmo falla.

3. **Terminación**: si la agenda no contiene elementos, significa que ya se ha encontrado un plan consistente que resuelve el problema y el algoritmo se detiene. En caso contrario se sigue con el paso 4.
4. **Selección de la operación**: se escoge la etiqueta que encabeza la agenda del planificador, y se busca entre las operaciones ya insertadas en el plan una que tenga como efecto dicha etiqueta. De no encontrarse se pasa a examinar entre las operaciones posibles del planificador y si tampoco se halla ninguna entonces el algoritmo falla. En caso contrario y se encuentre una, se añade a la lista de acciones del plan y se vuelve al paso 2.

Análisis en profundidad

El algoritmo posee ciertas características destacables que lo diferencia de otras aproximaciones.

Uno de los primeros detalles que puede llamar la atención es que dada la descripción de nuestras etiquetas se puede observar que la formalización que se ha escogido para representar el mundo está más cerca del lenguaje STRIPS que del ADL [20], básicamente esto significa que no tenemos operadores condicionales, ya que nuestras etiquetas son descritas explícitamente. Esta decisión fue tomada buscando la simpleza a la hora de implementar el algoritmo. En última instancia, esto provoca que tengamos más definiciones de etiquetas y reglas y a cambio nuestro algoritmo se simplifica debido a la omisión del paso de unión y separación que se da en las implementaciones clásicas.

Otra propiedad interesante es su comportamiento, que ante un mismo input siempre es idéntico y dependiente del orden en el que se hagan las declaraciones de las operaciones debido a una búsqueda determinista de las reglas y de la etiqueta objetivo en el paso 4. Esto último tiene grandes repercusiones, haciendo que nuestro algoritmo sea solvente, pero no completo.

Visualización del plan

Para representar el plan trazado de una forma más directa y sencilla y ser capaces de tratar los resultados cómodamente se dispone de un software de visualización de grafos denominado *Graphviz*. Este programa es de código abierto y completamente gratuito, además de usar como entrada documentos de texto plano en formato DOT, un lenguaje muy simple y estandarizado para representar grafos.

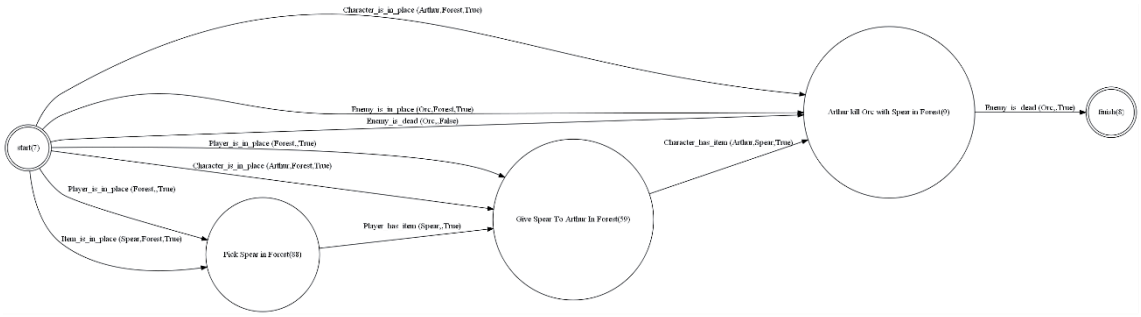

Ilustración 3: Representación visual de un plan realizado con el software Graphviz

Algoritmo de evaluación de tramas de contingencia

Descripción del algoritmo

Una vez tenemos un plan que satisface los objetivos propuestos, dada la naturaleza interactiva del medio en el que se trabaja se debe analizar dicho plan para buscar los puntos en los que el jugador puede tomar decisiones distintas de las previstas y que invaliden la solución hallada. Para ello hemos dividido el trabajo en varios pasos:

1. **Búsqueda de conflictos**: gracias a la descripción que hace un planificador de orden parcial de un plan con las conexiones causales resulta muy sencillo visualizar en que puntos puede el jugador tomar una decisión que invalide dicho plan. En este paso buscamos en cada conexión casual si existe una operación tal que un efecto de esta contradice la etiqueta de la conexión (similar a las colisiones descritas en el paso 2 del algoritmo del planificador).
2. **Calculo del plan parcial**: una vez hallada una operación que colisiona con el plan original, creamos una nueva copia de este sin los nodos posteriores a la colisión, y añadimos la acción encontrada.
3. **Replanificación**: en el paso anterior hemos obtenido un plan parcialmente resuelto. Volvemos a ejecutar nuestro algoritmo de planificación para dicho plan, calculando así una solución completa que funciona para el caso en el que el jugador escoja realizar la acción que colisionaba con el plan original. Se vuelve al paso 1 para buscar otras posibles operaciones conflictivas.

El algoritmo debe ejecutarse recursivamente también para los nuevos planes deducidos, ya que estos a su vez pueden contener nuevas acciones conflictivas. Si en el paso de replanificación no se encuentra un plan que solvente el problema, toda la quest debe ser rechazado, ya que existe una acción que el jugador puede ejecutar y que impida la consumación de la misión.

Representación de la quest en lenguaje XML

Obtenida la quest en forma de plan con tramas de contingencia escogemos representarla con un lenguaje estandarizado como es XML para poder facilitar la tarea de construir un programa capaz de consumir nuestro formato y traducirlo a alguna especificación que el juego con el que se desea utilizar el generador pueda utilizar.

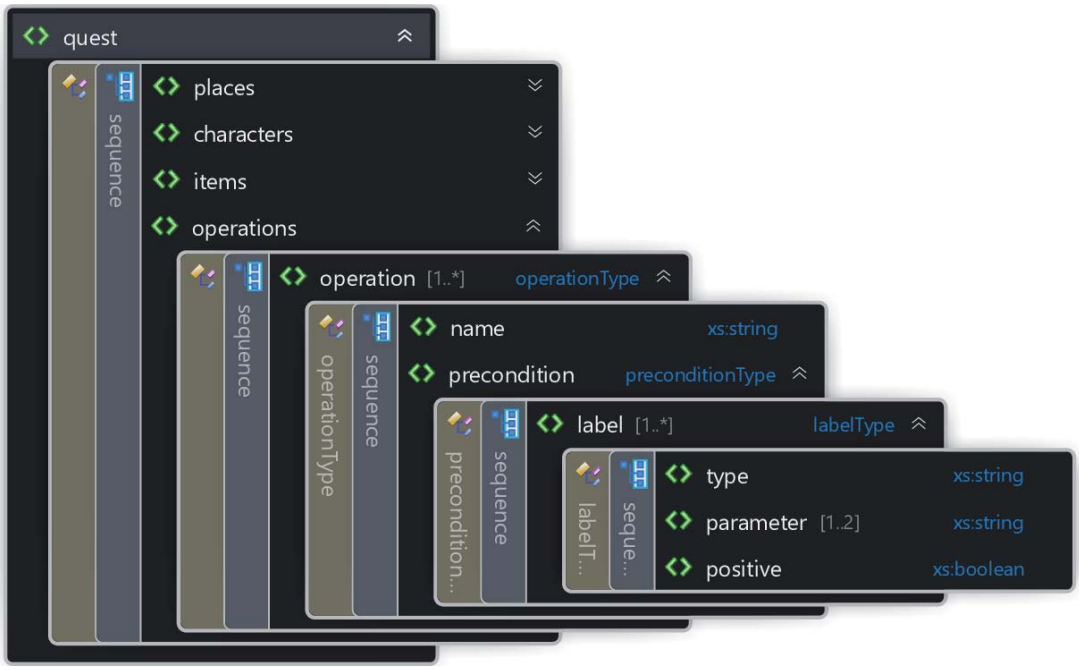

Ilustración 4: Esquema XML de una quest visualizado en Microsoft VS

Se observa en la ilustración 4 se ha dividido nuestro esquema XML en cuatro componentes: lugares, personajes, objetos y operaciones. Las tres primeras se han añadido pensando en la generación dinámica de elementos en el universo del juego y se comentara más adelante su uso en la sección de futuros trabajos.

Dada nuestra descripción de las operaciones siguiendo el lenguaje STRIPS, tan solo con dar el nombre de la regla se puede describir completamente una operación (por ejemplo, la acción "Dar la espada a Alice en el bosque" es distinta de "Dar la espada a Bob en el desierto" y tendrán por ende nombres distintos). Aun así, se ha decidido añadir también las precondiciones de cada uno para facilitar la tarea de traducción de nuestro formato a uno que el videojuego objetivo pueda consumir.

Si se analiza uno puede darse cuenta que en el output no se distingue entre distintos planes y simplemente se añaden las operaciones de todas las tramas (original y de contingencia) y con las precondiciones de estas. Esto se debe a los resultados estudiados en el siguiente apartado, donde se explicará que el plan original no es por definición mejor (ni a la hora de medir su calidad narrativa, ni su capacidad de diversión, …) que los de contingencia, y simplemente se analiza las precondiciones para indicar al jugador que operaciones son posibles como siguiente paso, siendo entonces probable que se muestren operaciones de varios planes a la vez.

Resultados

Entorno de juego Flare

Con el objetivo de poder probar nuestro sistema se busca un videojuego capaz de aceptar modificaciones para desarrollar allí las misiones generadas y poder testearlas. Se escoge

el software Flare por encima de otras respuestas comerciales (*Creation Kit* de The Elders Scrolls V: Skyrim [21] o el *REDKit* de *The Witcher 2: Assassins of Kings* [22]) ya que al igual que Graphviz este es un software gratuito y de código libre.

Flare dispone de todas las piezas básicas de todo RPG: personajes, enemigos, mapas, objetos y botines, … que podemos describir en unos documentos de texto plano que se incluyen en el engine y que el entorno lee cuando arrancamos una partida. El programa incluye además recursos como imágenes para los diálogos o texturas para los escenarios reduciendo así la tarea de crear nosotros mismos estas piezas pudiendo concentrarnos en probar nuestro algoritmo.

Resultados de las pruebas en el entorno

Se crean distintos universos de prueba para probar con el engine Flare, definiendo las distintas etiquetas, operaciones y demás del mundo en el que se buscara una posible quest para ser jugada. El primer objetivo se consigue en cada prueba realizada, ya que nuestras misiones son jugables e imposibles de dejar al jugador realizar ninguna acción que las convierta en irrealizables. Por otro lado nos encontramos con un problema que no se había advertido hasta esta fase del desarrollo. El plan que se convierte en quest no es ni el más obvio, ni el más divertido. Cuando un diseñador humano diseña una quest no solo lo hace para que esta tenga ciertas características como si es entretenida o si tiene variedad de situaciones, además se suele buscar que la única manera de completar la misión sea la descrita por él. En el caso de nuestro algoritmo esto no se restringe y da resultados extraños como que una trama de contingencia sea una forma de resolver la historia más lógica que el propio plan original y este es uno de los motivos por el cual en el documento XML no distinguimos entre distintos planes original y de contingencia.

Futuros trabajos

Generación de múltiples quests

Durante este proyecto se han enfocado todos los esfuerzos en la generación de una quest que cumpliera con ciertas características como puede ser la completitud de la misma. La adaptación necesaria para escalar nuestro algoritmo y poder realizar múltiples quests no solo requiere de un mayor esfuerzo cuantitativo, sino que el problema es cualitativamente más complejo, ya que hay que tener en cuenta factores como el no poder contar con un estado inicial único si se quiere que las misiones sean independientes o parcialmente dependientes en el orden en que se pueden realizar (esta característica se suele dar en la mayoría de videojuegos comerciales, así que es una propiedad exigible por el usuario medio). Parte se ha resuelto debido a la generación de tramas de contingencia, y esa puede ser una posible línea para seguir el proyecto por esta rama.

Generación de nuevos elementos

El algoritmo planteado en este texto parte de la base de que el desarrollador que va a hacer uso de sus funciones tiene diseñado todo el universo y sus elementos: personajes, escenarios, objetos, … Pero, ¿Y si se quiere dejar algunos elementos en el tintero? Esto añadiría grados de libertad posibilitando la creación de una capa superior de software anterior

a la del planificador que generaría dichos elementos bajo ciertas premisas y necesidades. Estas condiciones para la creación de los nuevos elementos podrían estar ligadas al tema tratado en el siguiente punto.

Tramas argumentalmente interesantes

En el documento se ha tratado el aspecto jugable de las misiones, priorizando que fueran completamente resolubles y que el jugador no pudiera atascarse en ellas. Pero yendo un paso más allá se puede generar las quest buscando además otras condiciones: mécanicas jugables interesantes, variedad de acciones, … Entre todas ellas destaca la búsqueda de una narrativa más sofisticada que implique al usuario y resulte atrayente. Sería interesante poder utilizar tropos para ello [23] o también denominado espacios comunes, definidos cómo patrones que se dan en la mayoría de medios de expresión (literatura, teatro, cine, …) y que se pueden usar como recurso, además de existir una gran base de datos que intenta recopilar la gran mayoría de estos [23]. La dificultad más grande a la hora de intentar hacer uso de ello es que los tropos son de muy diversa índole y tenemos desde arquetipos de personajes hasta arcos argumentales o los distintos conflictos de la historia sin ninguna separación explicita, lo cual conllevaría una gran labor para poder producir una base de datos útil con estos elementos.

Conclusiones

El programa aquí expuesto es un buen primer paso para adentrarnos en la generación procedimental de tramas y el storytelling dentro de los videojuegos, pero como se ha podido observar, aún quedan muchos pasos que realizar para tener una pieza de software competente. Con lo visto hasta ahora se concluye que para poder realizar un sistema eficaz de generación de tramas se ha de llegar a una solución de compromiso entre los elementos jugables, narrativos y simulacionistas. Mientras que nuestra solución es muy competente si hablamos de simulación, ya que tiene en cuenta todos los elementos necesarios que hacen falta antes de poner en marcha la quest para que esta sea realizable, no cumple los estándares de diversión y entretenimiento que se le exigen a todo juego y no busca de ninguna forma una trama argumental que sea narrativamente interesante, dejándolo en una sucesión de acciones que no tienen por qué implicar al jugador emocionalmente.

Bibliografía

[1] J. Togelius, E. Kastbjerg, D. Schedl and G. Yannakakis, "What is procedural content generation?", Proceedings of the 2nd International Workshop on Procedural Content Generation in Games - PCGames '11, 2011.

[2] M. Hendrikx, S. Meijer, J. Van Der Velden and A. Iosup, "Procedural content generation for games", ACM Trans. Multimedia Comput. Commun. Appl., vol. 9, no. 1, pp. 1-22, 2013.

[3] D. Aversa, "Procedural Contents Generation in Modern Videogames", 2015. [Online]. Available: http://www.davideaversa.it/wp-content/uploads/2015/06/Procedural-Contents-Generation.pdf.

[4] J. Maher, "» Akalabeth The Digital Antiquarian", Filfre.net, 2016. [Online]. Available: http://www.filfre.net/2011/12/akalabeth. [Accessed: 22- Apr- 2016].

[5] "Blizzard Entertainment:Classic Games", Us.blizzard.com, 2016. [Online]. Available: http://us.blizzard.com/en-us/games/legacy/. [Accessed: 22- Apr- 2016].
[6] "Category:Radiant Story - Creation Kit", Creationkit.com, 2016. [Online]. Available: http://www.creationkit.com/Category:Radiant_Story. [Accessed: 22- Apr- 2016].
[7] M. Mateas and A. Stern, "Façade: An Experiment in Building a FullyRealized Interactive Drama," in Game Developers Conference, 2003.
[8] P. Weyhrauch, "Guiding Interactive Drama", Ph.D. Dissertation, Carnegie Mellon University, 1997.
[9] B. Mac Namee, "Proactive Persisten Agents: Using Situational Intelligence to Create SUpport Characters in Character-Centric Computer Games", Ph.D. Dissertation, Univeristy of Dublin, Trinity College, 2004.
[10] D. Távara and A. Meier, "Agents with Personality for Videogames", Articulated Motion and Deformable Objects, 2006.
[11] L. Goldberg, "The structure of phenotypic personality traits.", American Psychologist, vol. 48, no. 1, pp. 26-34, 1993.
[12] Stéphane Bura, "Emotional AI for Expanding Worlds" in Game Developers Conference, 2012.
[13] L. Peña, J. Peña and S. Ossowski, "Representing Emotion and Mood States for Virtual Agents", Multiagent System Technologies, pp. 181-188, 2011.
[14] "The Director", Left 4 Dead Wiki, 2008. [Online]. Available: http://left4dead.wikia.com/wiki/The_Director. [Accessed: 22- Apr- 2016].
[15] F. Peinado, "Interactive Digital Storytelling: Automatic Direction of Virtual Environments", Upgrade, no., pp. 42-46, 2006.
[16] Mark O. Riedl, David Thue, and Vadim Bulitko. "Game AI as Storytelling". In Pedro Gonzales Calero (Ed.) Artificial Intelligence for Computer Games. Springer Verlag, 2011.
[17] "C#", *Msdn.microsoft.com*, 2016. [Online]. Available: https://msdn.microsoft.com/es-es/library/kx37x362.aspx. [Accessed: 01- Jul- 2016].
[18] Unity - Game Engine", *Unity3d.com*, 2016. [Online]. Available: https://unity3d.com/es. [Accessed: 01- Jul- 2016].
[19] J.S. Penbertby and D. Weld, "UCPOP: A Sound, Complete, Partial Order Planner for ADL", Proceedings Third International Conference on Principles of Knowledge Representations and Reasoning, pp. 103-114, 1992
[20] E. Pednault, "ADL: exploring the middle ground between STRIPS and the situation calculus", Proceedings of the first international conference on Principles of knowledge representation and reasoning, pp. 324-332, 1989
[21] "Creation Kit", *Creationkit.com*, 2016. [Online]. Available: http://www.creationkit.com/. [Accessed: 01- Jul- 2016].
[22] "REDKit", 2016. [Online]. Available: http://redkit.cdprojektred.com/en/home. [Accessed: 01- Jul- 2016].
[23] Tropes - TV Tropes", *TV Tropes*, 2016. [Online]. Available: http://tvtropes.org/pmwiki/pmwiki.php/Main/Tropes. [Accessed: 02- Jul- 2016].

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Mon Jul 04 19:07:48 CEST 2016
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)