INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
Campus Ciudad de México

Escuela de Graduados en Ingeniería y Arquitectura

Maestría en Ciencias Computacionales

Tesis:

SANDTRACE: MARCO DE TRABAJO PARA LA PERSONALIZACIÓN DE VIDEOJUEGOS

AUTOR: Carlos Fernando Mayer Llamosa

ASESOR: Dr. José Martín Molina Espinosa

México D.F., mayo de 2012

Contenido

Lista de Figuras

1. Introducción

Una forma de buscar ideas es simplemente observar (en silencio de preferencia) todo cuanto podamos a nuestro alrededor. En ocasiones resulta inspirador, en otras desalentador y en este caso en particular la observación concluyó en un poco de ambas. Cada día, (en México pero también en todo el mundo) se cometen actos sumamente reprochables; algunos terribles y dignos de un noticiero amarillista y otros tímidos y casi inexistentes. La relación de ellos con cada persona puede ser patente o apenas perceptible y se relaciona con infinidad de factores. Educación, tanto académica como en valores parecen ser un elemento faltante y muy necesario en la población de nuestro México contemporáneo.

El resultado de este déficit educativo posee un impacto de nivel nacional que no es tema del presente documento: Aquí la cuestión es hacer algo al respecto. Fue así como en un proceso inductivo involuntario[1], surgió la idea presentada en este documento; primero, como una aplicación reflejada en un caso de estudio (un videojuego de ética) y posteriormente como una abstracción general (en los capítulos siguientes se delimita el concepto de esta abstracción y el alcance de su generalidad) materializada como un sistema completo e independiente, capaz de permitir a un profesor personalizar y adaptar

[1] *Proceso Inductivo Involuntario:* Partir de un concepto muy particular a otro más general sin que éste sea el objetivo del proceso.

de acuerdo con sus necesidades didácticas, toda una serie de elementos, no sólo de ética, sino de cualquier materia, para cualquier videojuego existente o en desarrollo[2].

1.1. Motivación

La inspiración del trabajo de tesis surge a partir de la posibilidad de emplear un videojuego como medio (herramienta) para presentar dilemas éticos a los estudiantes y reflexionar sobre ellos posteriormente.

Considero de utilidad retomar algunas líneas de la misión del ITESM para dar seguimiento a la motivación: "*Es misión del Tecnológico de Monterrey formar personas íntegras y éticas, con una visión humanista y competitivas internacionalmente en su ámbito profesional (...)*". También es verdad que se trata de una institución conocida por estar siempre a la vanguardia en tecnologías. La idea entonces surge a raíz de una búsqueda de innovación que permita conjuntar las tecnologías a las que los alumnos tenemos acceso, con elementos de carácter pedagógico que refuercen o actúen en pos de la misión.

1.2. Antecedentes

De acuerdo con (Puentedura, 2005), a pesar de que jugar parece una actividad improductiva en principio, se trata de una de las actividades que caracteriza a los seres humanos, ya que desde los primeros días de nuestra existencia nos vemos envueltos en juegos de todo tipo e inclusive, existen registros que evidencian la participación de

[2] Es importante aclarar que para cumplir este punto, el videojuego deberá satisfacer ciertos criterios técnicos, definidos y establecidos en el presente documento, así como cumplir con (al menos una parte de) la temática que un profesor en particular busque cubrir.

nuestros ancestros en juegos de todo tipo. Siendo así, podemos concluir que los seres humanos jugamos desde hace literalmente millones de años.

1.2.1. Jugar, Juego, Videojuego

Jugar consiste en moverse con libertad dentro de una estructura más rígida (que la del propio juego), tal como una niña juega a ser mamá en el contexto de una familia donde ella es la hija y su papel es muy claro; un juego es propiamente un sistema en que los jugadores se ven envueltos en un conflicto artificial definido por reglas y que tiene un resultado cuantificable (Salen & Zimmerman, 2003). Tomando esta definición y mezclándola con la palabra video, que en latín significa "yo veo", obtenemos la palabra videojuego, que no es más que un *juego* cuyo principal elemento es visual y está representado en una pantalla (televisor, monitor de computadora, teléfono portátil, etc.).

1.2.2. Juego Serio (*Serious Gaming*)

Aunque podría sonar desalentador para jugadores experimentados, el término Juego Serio (*Serious Gaming*) no necesariamente implica la ausencia de entretenimiento. En realidad, un juego serio es aquél en que los objetivos van más allá de simplemente divertir al jugador. En el caso del presente proyecto, al buscar la conjunción de la educación y las nuevas tecnologías (videojuegos en concreto), resulta importante dar a conocer el concepto aquí descrito.

1.3. Propósito del estudio

Puesto que el problema buscará resolverse desde el área de las Ciencias de la Computación, la idea base para el estudio consiste en desarrollar un modelo de personalización aplicado como caso de estudio en un videojuego para plataformas

móviles de Apple® (sistemas iOS; iPhone, iPad), con elementos éticos manifestados en forma de dilemas[3] que inviten al jugador a una reflexión sobre los temas que allí se presenten. El aspecto principal no es el videojuego en sí, sino el motor que brindará la posibilidad de que un maestro o asesor establezca parámetros que permitan adecuar el juego a los propósitos de una materia (por ejemplo, en el caso de estudio de ética, secuencia de las escenas, cantidad de dilemas, etc.); la idea es mostrar el modelo aplicado en un juego.

1.3.1. Objetivos

- Proponer una abstracción del concepto de videojuego y de cada uno de los elementos que lo componen, así como el papel de ellos en la personalización
- Proponer una abstracción para la narrativa del videojuego, basada en los elementos que lo componen
- Proponer un sistema que permita, al ser incluido en el proceso de desarrollo, liberar juegos con la posibilidad de ser personalizados por un tercero

1.3.2. Metodología

Para poder hablar de una contribución a las Ciencias de la Computación, es necesario incluir elementos que distingan al proyecto de otros como él. La idea central es desarrollar un sistema que permita a profesores, por ejemplo, establecer parámetros de interés para propósitos de su materia, es decir, aprovechar las posibilidades de

[3] Del griego δί-λημμα, significa proposición doble y se entiende como un problema que presenta dos posibilidades siendo ninguna de ellas aceptable en su totalidad. Es un dilema ético cuando el problema involucra aspectos relacionados con el bien y el mal

personalización que el proyecto ofrece, para incluir videojuegos como herramientas educativas en la currícula. De este modo, el jugador seguirá la historia y los niveles correspondientes del juego, cumpliendo con los objetivos de entretenimiento que éste ofrezca, pero además ajustándose a una estructura planeada y establecida por un tercero; en este caso, un profesor.

Una vez desarrollado el sistema, se incluirá en un videojuego. Parte de la contribución de este proyecto, aunque no de manera esencial, es también el desarrollo del juego mismo.

Es importante hacer notar que el sistema debe ser incluido (como se menciona anteriormente) en el proceso de desarrollo del juego. Con esta premisa en mente, al final se tiene un juego que incorpora el sistema de personalización de parámetros y que muestra las posibilidades del mismo en un ambiente educativo.

1.4. Estructura del Documento

El presente texto consta de cuatro grandes bloques. El primero de ellos, del que esta sección es parte, busca familiarizar al lector con el tema y enfoque de este trabajo de tesis y presenta algunas definiciones generales, los objetivos y la metodología a seguir.

En un segundo bloque, se expone el estado del arte, en lo relacionado a actividades en donde el usuario afecta (toma decisiones) el curso de las mismas, la incorporación de la toma de decisiones en videojuegos, la importancia del elemento narrativo en ellos y un desglose de este último.

En el tercer bloque se proponen las abstracciones y arquitecturas necesarias, así como el diseño, análisis y desarrollo del sistema Sandtrace Framework, a fin de satisfacer el propósito del estudio.

Finalmente, en un cuarto bloque se presenta como caso de estudio, la implementación del Sandtrace Framework en un juego de video llamado Private Eye.

2. Estado del Arte: Videojuegos, Decisiones del Jugador

Por tratarse de un estudio relacionado con videojuegos, la revisión de literatura consistió principalmente en una investigación de campo, en la que probando los juegos, viendo videos sobre ellos y buscando en los foros (IGN sobretodo, www.ign.com), se hallaron varios títulos que abordan el tema de una manera similar. Sin embargo, conviene hacer un análisis más amplio y desglosado, ya que por un lado tenemos el papel que juega el usuario en la toma de decisiones, y por otro, la incorporación de estos mecanismos en un juego de video. A continuación se explicarán ambos elementos y posteriormente la fusión de ellos en un juego de video.

2.1. ¿Dónde toma el usuario decisiones?

El usuario puede tomar decisiones antes, durante o después de una actividad en particular, la cual puede ser o no un videojuego. A continuación se presenta un poco de historia sobre este tema, para concluir con la inclusión de la toma de decisiones en videojuegos.

2.1.1. Libros Interactivos

Este tipo de libros, innovadores en su tiempo, permitía una experiencia de lectura, de entretenimiento (e incluso de navegación marina) mucho más completa, al incluir elementos como piezas de papel móviles o giratorias, páginas con ilustraciones que parecían levantarse del libro, páginas impregnadas con olor, etcétera. Los primeros libros interactivos de los que se tiene conocimiento, eran llamados "*volvelles*", y se utilizaron

desde la edad media con propósitos diversos, por ejemplo, la localización de constelaciones. Sin embargo, los libros donde las ilustraciones se levantaban de las páginas, no aparecieron sino hasta el siglo XIX. De ahí en adelante, comenzaron a añadirse elementos varios, hasta que con la inclusión de electrónica fue posible añadir botones y efectos de sonido, entre otras aportaciones(University of Virginia, 2000).

2.1.2. Libros donde el lector decide el curso de la historia

Se trata de un caso particular de libros interactivos en los que, como su nombre indica, es el lector quien en puntos particulares del libro, puede decidir qué sucede con la historia, a través de navegar hacia una página determinada. Conocidos como "Escoje tu propia aventura" (*Choose your own adventure)*, se originaron en 1979, en una entonces nueva sección de Bantman Books. Sin embargo, sus orígenes se remontan a 1976, cuando Ed Packard llevó un manuscrito llamado *Sucargane Island* como propuesta a la compañía Vermont Crossroads Press, entonces presidida por R.A. Montgomery. El manuscrito fue comprado y renombrado a *The Adventures of You* (Las aventuras de ti). Posteriormente, *The Adventures of You* fueron llevadas a Bantman Books, en donde se invitó al escritor original (Ed Packard) y a otros que trabajaron con él, a colaborar con seis libros más, renombrando la serie a *Choose your own adventure* y dando con ello origen formal a este tipo de libros (Choose Your Own Adventure, 2012).

2.1.3. Programas de Radio

¿Quién de nosotros no ha escuchado alguna vez: "comuníquese al teléfono en cabina y vote por la canción de su preferencia"? Esta interacción con el usuario permite al último sentirse parte de un todo radiofónico en el que el curso de las acciones de la emisora de radio, se ve definitivamente influenciado por el contacto con el primero.

2.1.4. Video sobre pedido (*Video on Demand*)

Desde sus inicios en 1994, con el proyecto interactivo de televisión de Cambridge (iTV Trial)(ATM Ltd and Online Media, 1996) el video *on-demand* dio paso a una nueva era en el entretenimiento televisivo, en tanto que por primera vez, los usuarios meta podían tomar parte en la programación de series y películas, a través de votación por llamadas telefónicas. Un ejemplo actual de este tipo de servicio, es utilizado por canales de televisión de paga tales como TNT (Turner Network Television), parte de Time Warner Inc (Wikipedia, the free encyclopedia, 2012).

2.1.5. Reality TV

Aunque los primeros esbozos de este tipo de entretenimiento se remontan a la mitad del siglo XX, la televisión de realidad (*Reality TV*) tal como la conocemos hoy, se origión en 1992, cuando MTV sacó al aire por primera vez un show conocido como *The Real World* (El mundo real). De allí en adelante, varios programas adoptaron este esquema y siguen al aire al día de hoy. Lo interesante es que este tipo de programas incluyen al usuario (televidente) en la decisión del curso de la serie (pensemos por ejemplo en "La Academia" y en cómo son las votaciones telefónicas la base de eliminación de los concursantes) (History of Reality TV, 2012)

2.1.6. Videojuegos

Todos los juegos de video tienen como base una interacción con el usuario para lograr el entretenimiento, es decir, el jugador no permanece sentado sin realizar acciones (como en un programa de televisión o radio), sino que en todos los casos toma parte activa en el desarrollo del juego. Con esto en mente, es posible acotar entonces los juegos que

representan avances en áreas de interacción específicas y puntualmente, en la respuesta del curso de la historia del juego ante las acciones del usuario.

A continuación se enlistan algunos de los juegos que se destacan por la importancia que dan a la toma de decisiones del jugador.

2.2. Heavy Rain

Se trata de un juego de video que aborda un género llamado drama interactivo (*Interactive Drama*). En él, la habilidad y experiencia del jugador quedan en segundo lugar, dejando espacio para elementos como la intuición y el ingenio. La historia del juego se basa en los crímenes perpetrados por un homicida en serie, quien secuestra gente y la asesina cuatro días después. El juego comienza con el secuestro del hijo del personaje principal y aunque al inicio parece muy lento, conforme se desarrolla la trama el jugador se ve en la necesidad de tomar decisiones que afectan drásticamente el comportamiento y la secuencia del mismo. Cabe mencionar que las decisiones no son binarias (sí o no), sino que en ciertos momentos del juego se presentan varias opciones, pareciendo todas ellas "correctas" e "incorrectas" al mismo tiempo (IGN, 2010). En la figura 1 se muestra un ejemplo de la interfaz con varias opciones a seleccionar por parte del jugador.

Figura 1. Interfaz de opciones en Heavy Rain, para el Sony Playstation3 ®. Imagen tomada de (IGN, 2010)

Este juego, muestra un comportamiento interesante en el que, de alguna manera, la historia del primero se ve modificada y continúa, sin importar qué suceda. Por ejemplo, si el personaje principal muere, el juego no termina. La historia cambia (tal como sucedería en la vida real) y lleva a un desenlace distinto. Con base en las acciones que el jugador toma a lo largo del videojuego, varias historias diferentes pueden suscitarse. Es un título que no se termina al llegar al final del mismo, en tanto que si se vuelve a jugar, muy probablemente la historia sea distinta.

2.3. Fallout 3

Este videojuego de tinte retrofuturista toma lugar en lo que queda de la tierra después de una guerra nuclear, donde las ciudades fueron completamente destruidas y las personas viven en refugios subterráneos. Aunque el juego tiene sus objetivos propios, lo que interesa al tema de estudio es la manera en que las decisiones que el jugador toma se ven reflejadas en la interacción con otros personajes, los cuales pueden, por ejemplo, ofrecerse a ayudar, negar su ayuda e inclusive atacar al protagonista.

Además, en este juego las opciones no se presentan de manera explícita, sino que el control que se posee sobre el personaje principal permite realizar acciones tales como disparar a una persona inocente o hacer negocios con un criminal. Existe un indicador en el juego llamado *karma meter* (Medidor de Karma) que muestra que tan "bueno o malo" ha sido el personaje principal; sin embargo, la mejor manera de apreciar las consecuencias de los actos de éste es cuando existe la necesidad de interactuar con otros dentro del juego. Si ha sido "bueno", los "buenos" buscarán ayudarlo y le recibirán bien. Si ha sido "malo" los "buenos" lo desprecian y aunque es posible hacer negocios con "malos" ellos suelen ser traicioneros (IGN, 2010). En la figura 2 se muestra una captura de pantalla de este juego.

Figura 2. Captura de Pantalla de Fallout 3. Tomada de (IGN, 2010)

En todos los caso anteriores, y después de realizar un extenso estudio de campo, se concluye que uno de los elementos más importantes y de más peso en el juego, es la narrativa. La narrativa (*storytelling*) es la manera en que se cuenta una historia. La forma en que los componentes de ella se incluyen e interactúan con el jugador y con el ambiente de juego a lo largo del desarrollo de éste, van estrechamente relacionados con lo que el

jugador tiene en la mente mientras juega y con cómo actúa dentro del juego. A continuación más sobre este tema.

2.1. Narrativa

La narrativa es la especialidad literaria que se dedica a contar historias, eventos y sucesos, en diferentes modalidades, pero con tres factores en común: Un principio, un fin y una secuencia de las acciones en el tiempo (RENa, 2008). Aunque existe una vertiente de la narrativa relacionada específicamente con los videojuegos, es buena idea comenzar con una visión más general de este concepto.

Tomemos por tanto la estructura básica empleada en películas. Aunque en principio sabemos que los videojuegos **no** son películas, es importante comenzar sobre un terreno conocido para la mayoría de los lectores.

En una película (*screenplay*), la estructura básica consiste en tres escenas o actos principales, y debe incluir también al menos un cambio radical cerca de la mitad de toda la historia. Aunque pueden encontrarse en desorden, el primer acto en general contiene información relacionada con el lugar, los hechos y las bases de la historia que dan pie a todo el desarrollo de ésta y presenta el primer conflicto en la historia. El segundo acto consta precisamente del desarrollo, contiene el cambio radical antes mencionado y plantea uno o más conflictos a resolver. Finalmente, el tercer acto representa el desenlace a los problemas planteados en actos anteriores e incluye generalmente un conflicto final (Field, 1994).

Esta estructura se conoce como el paradigma de Syd Field (por supuesto a raíz de su autor) y se muestra en la figura 3.

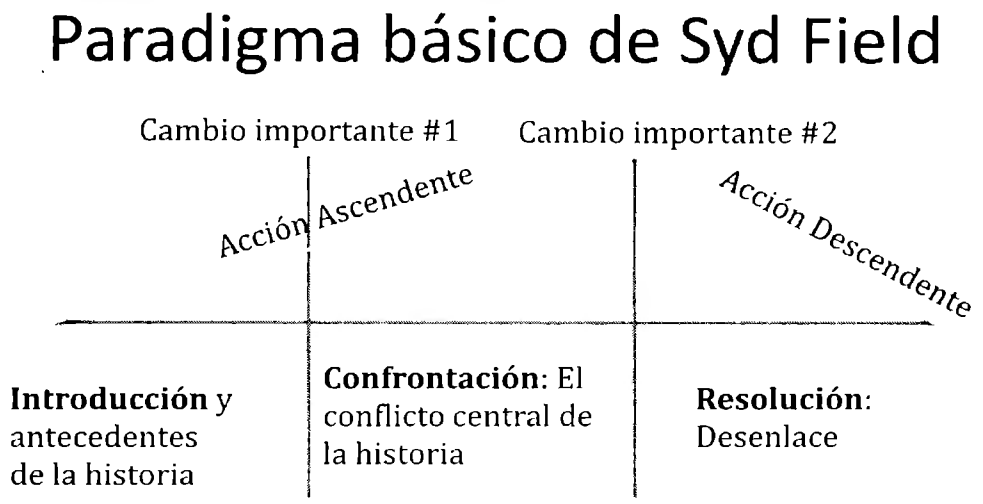


Figura 3. Estructura básica de una película, de acuerdo con Syd Field.

Con lo anterior en mente, podemos entonces comenzar a puntualizar de qué manera se pueden contar historias en los videojuegos. Existen cinco maneras principales de contar historias en un videojuego (Sheldon, 2004).

2.2. Historias en los Videojuegos

2.2.1. Lineal

Este tipo de sucesión de escenas es el más sencillo de comprender. Llanamente, una escena lleva a otra, que a su vez lleva a otra, de manera precisamente lineal, sin posibilidades de cambiar a otra sucesión de escenas distinta (en la figura 4 se muestra un diagrama explicativo).

Figura 4. Estructura Lineal

2.2.2. En ramas (*Branching*)

En este caso, la historia comienza con una escena inicial única pero posteriormente es posible "saltar" entre una y otra línea de juego, haciendo posible llegar a varios finales distintos a través de historias diferentes; sin embargo, posee limitaciones en tanto que no es posible "regresar" a escenas pasadas aunque sí "brincarse" escenas futuras (Véase figura 5).

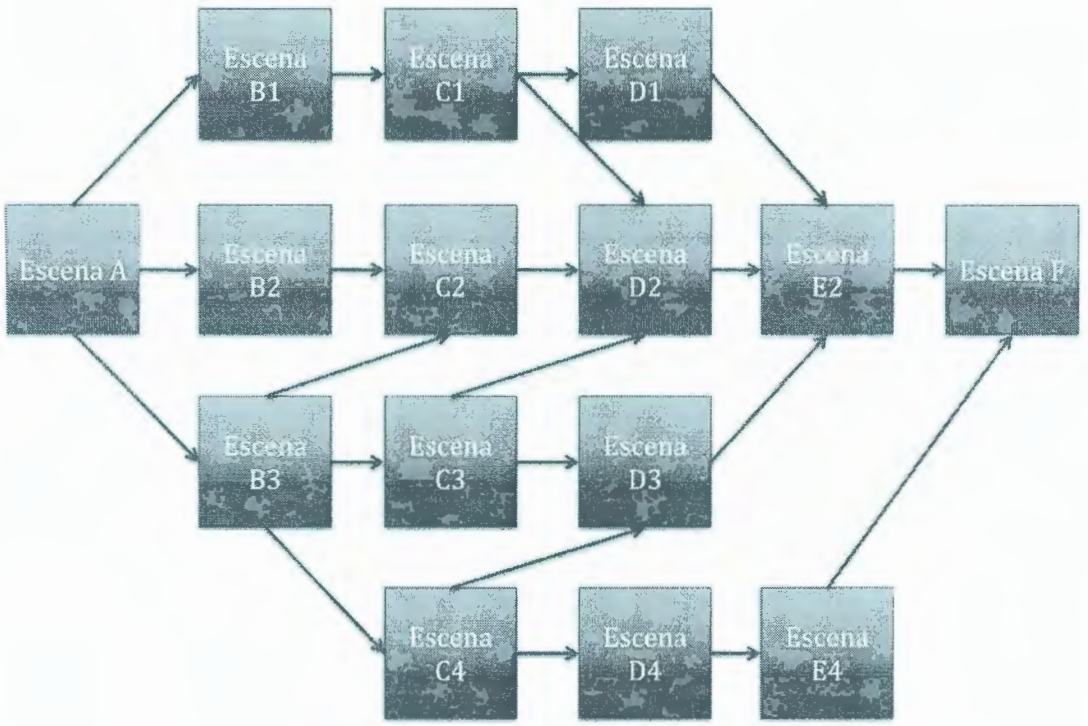


Figura 5. Estructura en Ramas (Branching).

2.2.3. En ramas, controlado (*Controlled Branching*)

Se trata de una especie de combinación entre narrativa lineal y narrativa en ramas. En resumen, el juego progresa siguiendo una línea preestablecida pero al mismo tiempo posee ramificaciones que están determinadas por las decisiones del jugador. Sin embargo, eventualmente y sin importar dichas decisiones, el juego regresa a la línea principal de la historia (figura 6).

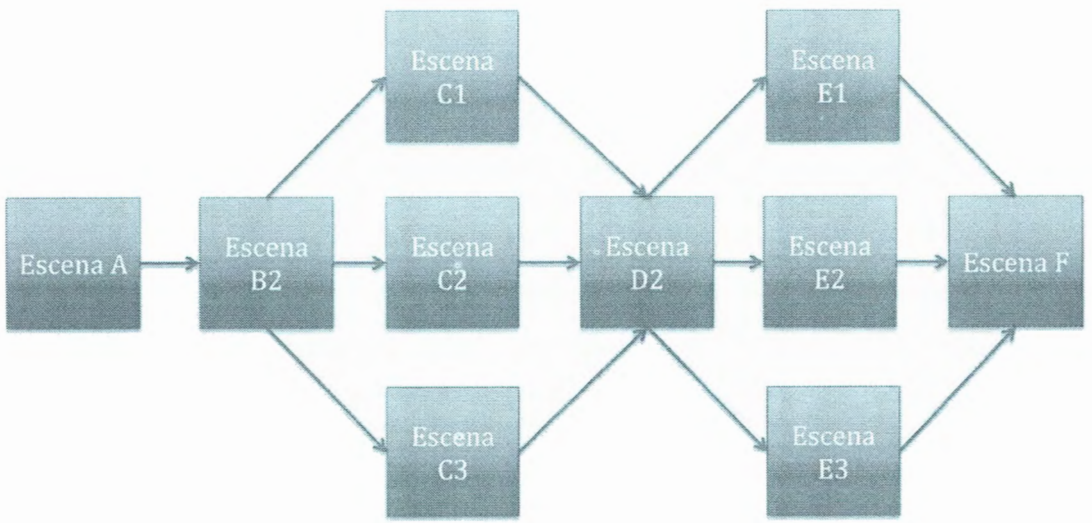


Figura 6. Estructura en Ramas, Controlada (*Controlled Branching*).

2.2.4. En red (*Web*)

Se trata de las historias más flexibles (y a la vez más difíciles de programar, en términos computacionales). Son básicamente un caso de narrativa en ramas con una gama de posibilidades aumentada (es decir, sin ninguna de las limitaciones de dicha técnica). Es

posible regresar a escenas pasados, cambiar la historia del juego una y otra vez, avanzar sin completar todas las tareas necesarias, etc. (Figura 7).

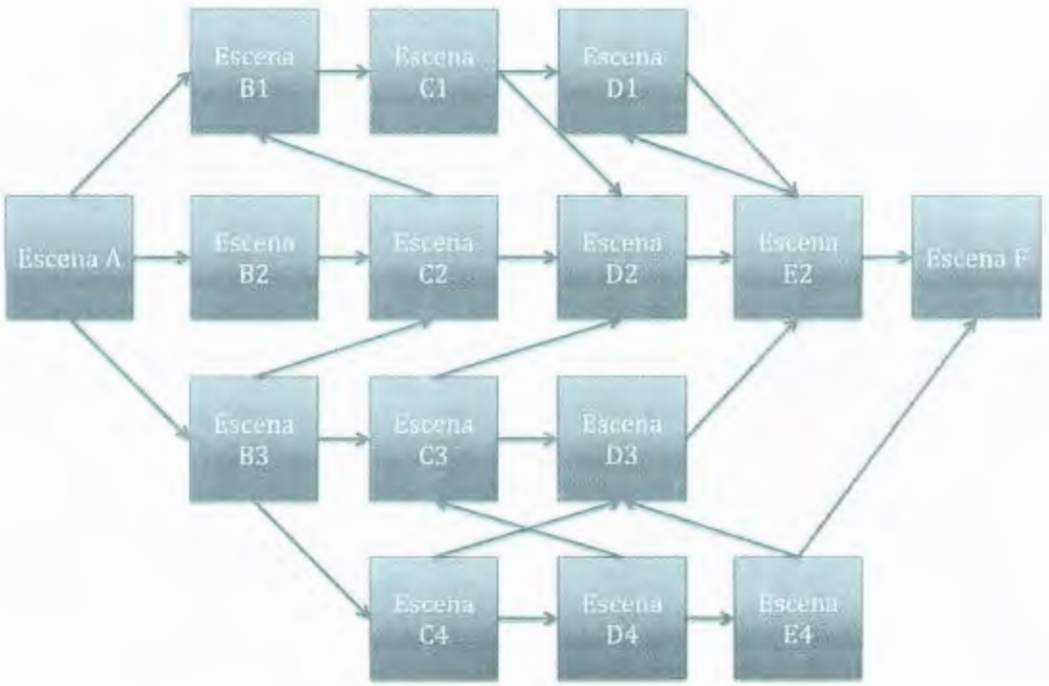


Figura 7. Estructura en Red (web).

2.2.5. Modular

Este tipo de narrativa, consta de una serie de bloques o "módulos" que contienen una pequeña tarea o historia en sí mismos y que pueden ser colocados uno tras otro sin que la historia pierda sentido. (Puentedura, 2005) nos menciona un ejemplo en el que se tienen dos bloques; uno con la misión de atravesar un sitio nadando y otro con la tarea de trepar una colina. Si el jugador decide primero atravesar nadando, entonces el trepar la colina debe ser la escena culminante y por tanto más complicada en ese nivel del juego. En

contraste, si se inclina por trepar primero, entonces atravesar nadando será más difícil y representará el clímax del nivel (figura 8).

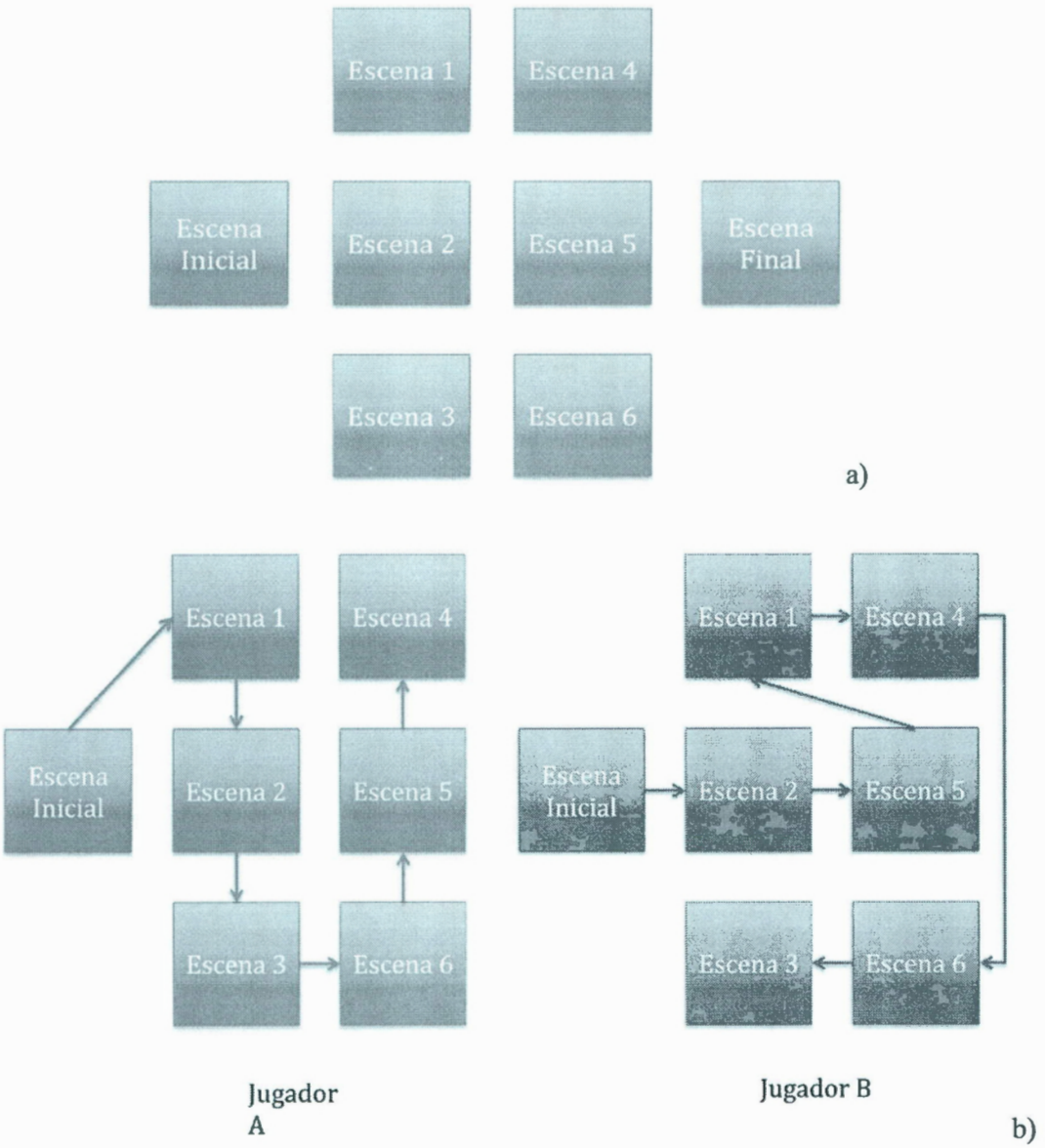


Figura 8. A) Serie de módulos o escenas, indicando el inicio y el final de la historia. B) Dos ejemplos de organización de la historia, jugada por personas distintas, en que las escenas no tienen por qué seguir un orden preestablecido pero la dinámica de juego permite que el grado de dificultad se ajuste al orden escogido por el jugador.

3. Sandtrace Framework

El Sandtrace Framework se basa en la conjunción de dos abstracciones: la de videojuego y la de narrativa. A continuación el desglose de esta idea.

3.1. Videojuego

Un videojuego, tal como se puntualizó en la introducción, no es otra cosa que un juego cuyo principal elemento es visual y se encuentra representado en una pantalla. Sin embargo, para fines del presente trabajo, se propone una definición complementaria, que retoma algunos elementos del trabajo de (Huesca, 2011) relacionados con la abstracción del concepto de videojuego, y los complementa con otros. Conceptos en común con el citado trabajo son el de escena y el de evento; sin embargo, la manera de abordarlos presenta diferencias importantes y por tanto en este documento se explica exclusivamente la propuesta del autor del mismo. De esta manera, un videojuego es un *conjunto de escenas, agrupadas en niveles, y que ellos en conjunto forman al juego*. Cada escena contiene a su vez eventos que pueden o no dar origen a un activo de juego (*asset*). Enseguida se detallan todos estos elementos.

3.1.1. Niveles

Si tomamos un juego y tratamos de dividirlo, partiendo de lo general a lo particular, el elemento más general de división después del juego en sí, serían los niveles.

Un nivel, tal como su nombre indica, se encuentra relacionado con el *progreso* de un juego. Si bien se asocian normalmente con la dificultad, pueden encontrarse además

ligados a otros elementos, tales como el acceso a opciones o lugares del juego antes inactivos, la revelación de información importante para la historia del juego, etcétera. En este sentido, un nivel es un elemento estructural abstracto en intangible, compuesto por una o más escenas, y asociado a un identificador numérico (nivel uno, nivel dos, etc). De este modo, el juego tiene en su primera subdivisión, un orden parcial en el que a cada nivel se asocian escenas que en su conjunto forman una historia de juego.

3.1.2. Escenas

Se proponen las escenas como la unidad fundamental de los videojuegos. Ellas representan pequeños universos en donde los personajes del juego **interactúan** con otros personajes o con elementos de su entorno. Cada interacción, acción o reacción, se encuentra asociada a un evento del juego. Por este motivo, de una manera conceptual pueden definirse las escenas como un conjunto de eventos del juego, que viven e influyen en esa escena, que modifican atributos, y originan o eliminan activos del juego (*assets*); las modificaciones de atributos y la creación o eliminación de activos se verán reflejados en escenas posteriores (para clarificar este punto, si en una escena, por ejemplo, un personaje destruye una casa, esa misma casa **no** puede "reaparecer mágicamente" en la escena o escenas siguientes). De esta manera, se evidencia la presencia de una narrativa subyacente entre escenas, responsable de (al menos) la coherencia de la historia.

3.1.3. Eventos

Un evento surge cuando de manera explícita se relaciona un elemento del juego con un elemento del programa tras él y dicha asociación aporta un valor o valores requeridos por el desarrollador y/o de importancia para el jugador. Por ejemplo (y para esclarecer un poco más esto último), supóngase un personaje, controlado por el jugador. El jugador

hace que el personaje entre a un edificio. Hasta este momento, el hecho de que el personaje haya entrado a un edificio es un **elemento del juego** solamente **susceptible** de ser evento. En el momento en que se relaciona explícitamente el tránsito del personaje al interior del edificio con el programa tras el juego, se crea un **evento**. Desde esta perspectiva el evento puede ser la relación del elemento del juego "tránsito al interior del edificio" con una variable del programa, de tipo booleana, por ejemplo: "Entró el personaje al edificio" cuyos valores serían: "Sí, No". Una vez expuesto el concepto de evento, se sugiere una clasificación de ellos, importante para secciones posteriores del presente documento.

3.1.3.1. Tipos de Eventos

3.1.3.1.1. Eventos Atómicos y no Atómicos

Un evento **atómico** consiste en aquel que se presenta una vez y no puede seguirse presentando durante la misma escena. Por ejemplo, encontrar la llave para salir de un edificio no puede volver a encontrarse una vez hallada.

En contraste, un evento **no atómico** puede presentarse varias veces. Por ejemplo, disparar un arma es un evento no atómico, en tanto que puede dispararse en varias ocasiones sin que por ello sea necesariamente un evento diferente.

3.1.3.1.2. Eventos Automáticos

Son aquellos en que la relación explícita entre el elemento del juego y el programa tras él no es visible para todos los involucrados. Por ejemplo, cuando el juego carga una escena, se genera un evento automático que indica que la escena se ha cargado (o no) exitosamente. El jugador o el profesor no se enteran jamás de este proceso.

3.1.3.1.3. Eventos Generadores de Activos

No todos los eventos generan activos. Un evento generador de activos es sencillamente aquél que al presentarse conlleva la creación de un activo de juego.

3.1.3.2. Consecuencias de los Eventos

Si bien un evento puede traer como consecuencia prácticamente cualquier cosa que el desarrollador decida, fungen ellos invariablemente **al menos** como predecesores de todos los activos de juego (*assets*). Esto significa que para que cualquier activo exista, debe presentarse un evento que lo genere. Es importante puntualizar, nuevamente, que este evento no es necesariamente visible para todos los involucrados. Por ejemplo, al cargar una escena con una ciudad dentro de ella, el evento generador es "cargar escena", mientras que el activo generado como consecuencia de este evento es la ciudad misma.

3.1.4. Activos de Juego (*assets*)

Son todos los seres o entes virtuales que coexisten en las escenas del juego. Se pueden subdividir en activos tangibles y en activos intangibles.

3.1.4.1. Activos Tangibles

Son aquellos cuya presencia e interacción es evidente en el juego. Comprenden modelos en 2D y 3D (tales como autos, edificios, terrenos, etc), *sprites*[4], avatares, (que no son más que la representación virtual del jugador en el mundo virtual) y otros similares.

[4] *Sprite*: En gráficas computacionales, se refiere a la integración visual de un elemento virtual, frecuentemente animado, independiente del entorno en el cual se integra. Por ejemplo, un modelo de un automóvil desplazándose en una carretera, siendo el sprite el automóvil y el entorno, la carretera.

3.1.4.2. Activos Intangibles

Son los activos difíciles de reconocer a primera vista en un juego; por ejemplo, ítems, dinero virtual, cámaras, luces, etc.

Todos los activos requieren un evento generador y un evento destructor. Sin embargo, no todos los eventos generan un activo. En el mundo virtual, los activos interactúan y se alinean con atributos preestablecidos que a fin de cuentas, escena con escena, contribuirán a dar forma a la historia del juego.

3.2. Narrativa

Tomando como base los diferentes tipos de narrativa en videojuegos mostrados en el capítulo anterior, se propone una estructura de orden parcial de las escenas, la cual se logra mezclando la organización en ramas y la organización en ramas controlada, en el que se establecen los niveles del juego y se asignan escenas susceptibles de ser colocadas por cada nivel. De esta manera, se garantiza la secuencia lógica de la historia del juego pero se abren las posibilidades a tantas líneas de juego como escenas haya por nivel. La figura 9 esquematiza esta propuesta y emplea un concepto llamado línea de tiempo (*timeline*). La línea de tiempo no es más que la dirección de la historia del juego, a través del tiempo. Sin embargo, se esquematiza enfáticamente porque en el Sandtrace Framework, todos los parámetros y opciones del juego están disponibles a lo largo de toda la línea de tiempo. Además, de este modo se satisface la definición de narrativa presentada anteriormente, en tanto que el modelo posee un inicio (escena inicial), un fin (escena final) y una secuencia de acciones en el tiempo (eventos y escenas a lo largo del *timeline*).

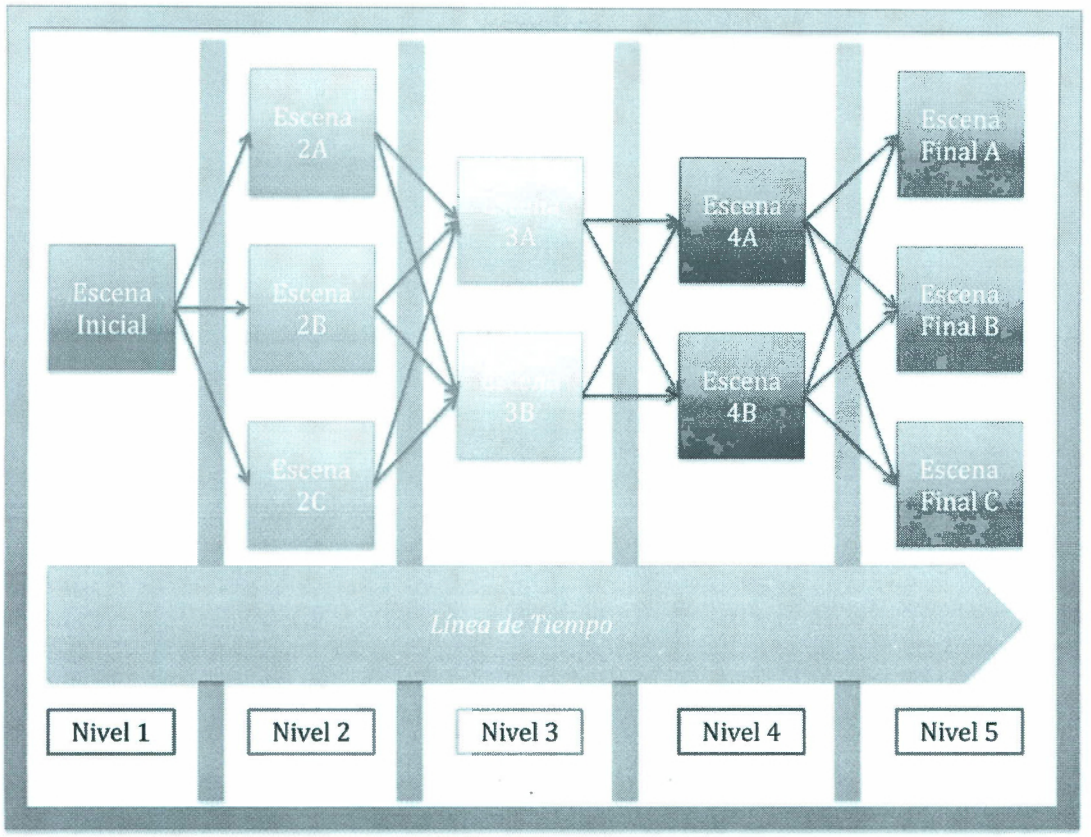


Figura 9. Flujo de escenas propuesto

Se tiene entonces un grupo de escenas de donde seleccionar y es posible elegir en qué nivel colocarlas. Existen evidentemente limitaciones en cuanto al nivel en que cada escena puede encontrarse, a fin de mantener una historia congruente. En este sentido, y tomando como base la figura 9, tenemos la siguiente organización:

- Niveles: Contienen una o más escenas; separados en la figura 9 por líneas rojas
- Escenas: Pueden colocarse en cada nivel, a juicio del personalizador. Se encuentran representadas en la figura 9 como bloques de colores.
- Línea de tiempo: Indica el flujo del juego.

Con lo anterior en mente se detalla la posibilidad de tránsito entre una escena del nivel *n* a **cualquier** escena del nivel *n+1*; el sistema desarrollado y expuesto en esta tesis decide, con base en parámetros explicados en lo sucesivo, y las acciones del jugador en la escena *n*, cuál de las escenas *n+1* es la más apropiada, y la regresará como sugerencia de escena siguiente a fin de que el desarrollador de videojuegos pueda cargarla.

3.3. Análisis

3.3.1. Descripción Global del Sistema

El Sandtrace Framework consta de una serie de clases en C# (API) que permiten a un desarrollador ajustar cualquiera de sus videojuegos (siempre que éste cumpla con el modelo de abstracción aquí propuesto) a fin de que parámetros de éste (tales como el orden de las escenas, el estado y características de eventos por escena y el valor de las acciones del jugador por cada escena), puedan ser personalizados por un tercero y se apliquen al juego, sin necesidad de compilarlo y publicarlo nuevamente con las nuevas opciones. El objetivo de este sistema es ofrecer a un tercero la posibilidad de personalizar elementos del juego, de acuerdo con sus objetivos particulares. Un ejemplo del campo de aplicación del sistema es la posibilidad de que un profesor (tercero) personalice elementos particulares de un juego para utilizarlo como herramienta educativa. Desde esta perspectiva se sugiere un ejemplo más:

Imaginemos la inclusión de un videojuego para asistir en un curso de ética. La implementación del sistema buscará permitir al profesor, después de que éste ha seleccionado un juego en donde existan elementos de carácter ético (tales como dilemas), elegir qué escenas desea que el alumno juegue, qué tipo de eventos puede el alumno jugar en cada escena y qué impacto tendrán en el curso de la historia del juego, de tal suerte

que las acciones del jugador, así como los parámetros establecidos por el educador, formen en conjunto la experiencia de un juego completamente personalizado y que satisfaga objetivos educativos propuestos por el profesor en cuestión.

3.3.2. Alcances

El proyecto consta de dos grandes bloques. El primero de ellos es la interfaz entre el profesor y los elementos personalizables del juego. El segundo consiste en la implementación en el juego del resultado de la personalización. Para fines del presente trabajo de tesis, se desarrolla en su totalidad el bloque de la implementación de parámetros en el juego y se presenta una parte demostrativa del bloque de personalización de parámetros. Esto implica que:

- La personalización de parámetros se realiza directamente sobre un archivo de texto XML, que es modificado ya sea desde una ventana de terminal o desde un procesador de texto en un ordenador cualquiera, siendo necesario subirlo a un servidor *web* posteriormente
- El Sandtrace consume el XML generado y publicado previamente desde un servidor *web*, lo abre y genera los objetos y estructuras de datos necesarios para su utilización por parte del desarrollador de videojuegos; así mismo, brinda funcionalidad relacionada con la secuencia de las escenas del juego y con la selección de las mismas

3.3.3. Requerimientos de Software

En este apartado se procede a la definición funcional y no funcional de las características que el sistema debe poseer.

3.3.3.1. Requerimientos Funcionales

El sistema permite:

1. Realizar de manera remota (*web*) la totalidad de la personalización
2. Seleccionar las escenas deseadas, por nivel
3. Seleccionar la cantidad de escenas por nivel
4. Asignar un **valor de entrada**, para cada escena
5. Definir un orden parcial de las escenas (una misma escena puede presentarse en más de un nivel, dependiendo del juego)
6. Seleccionar los eventos que se encontrarán presentes, por escena
7. Asignar una **calificación** a dichos eventos
8. Definir el **tipo** de evento
9. Evaluar las acciones del jugador, por cada escena
10. Seleccionar la escena siguiente más apropiada, con base en las acciones del jugador (eventos no automáticos), o con base en eventos automáticos

3.3.3.2. Requerimientos no Funcionales

Se enumeran los siguientes requerimientos no funcionales para el sistema:

1. La selección de parámetros se realiza desde una interfaz web
2. El sistema se codifica, en su parte nativa, utilizando C#; para la parte web, no es restrictivo este punto, aunque se sugiere perl o php
3. La comunicación entre las partes *web* y nativa se hace mediante un servicio XML
4. El sistema debe ofrecer escalabilidad para la inclusión de nuevos parámetros de personalización

5. El sistema debe funcionar al menos, aunque no únicamente en, plataformas iOS de Apple ® (iPad 2 en concreto)
6. La plataforma donde el sistema corre debe contar con acceso a Internet
7. El sistema será implementado **durante** el proceso de desarrollo de un videojuego, por parte del desarrollador
8. El sistema, una vez implementado, será utilizado por un tercero (profesor) para personalizar parámetros de juego
9. El usuario del juego no tendrá acceso a los parámetros de personalización
10. El personalizador del juego no tendrá acceso a las clases implementadas en el proceso de desarrollo
11. El desarrollador de videojuegos **debe** tener acceso a los parámetros de personalización

3.3.4. Conclusiones del análisis

Tras observar los requerimientos del sistema, se evidencia la necesidad de al menos dos grandes bloques. Uno de ellos comprende la parte de personalización y el otro de ellos, el resto del sistema. Tal como se especifica en los alcances, se desarrolla en su totalidad el Framework que permite la implementación de la personalización en el juego. Las implicaciones de ello son:

- El sistema es genérico y por tanto utilizable en cualquier ambiente de desarrollo compatible con C#

- El sistema está diseñado para utilizarse en etapas del desarrollo del videojuego y permite la personalización por parte de un tercero, una vez que el juego ha sido terminado
- El sistema propone e implementa un mecanismo de evaluación de escenas y un mecanismo de selección de escenas. De esta manera, con base en la evaluación de una, se regresa al desarrollador la siguiente "más conveniente" escena a cargar

3.4. Diseño

Con base en los resultados del análisis, se desglosa la propuesta de diseño del Sandtrace Framework.

3.4.1. Alto Nivel

Comenzamos por el diagrama de más alto nivel. La figura 10 esquematiza la solución propuesta, conteniendo un portal desde donde se realiza la selección de parámetros, y una aplicación nativa (videojuego) sobre la cual dichos parámetros son aplicados.

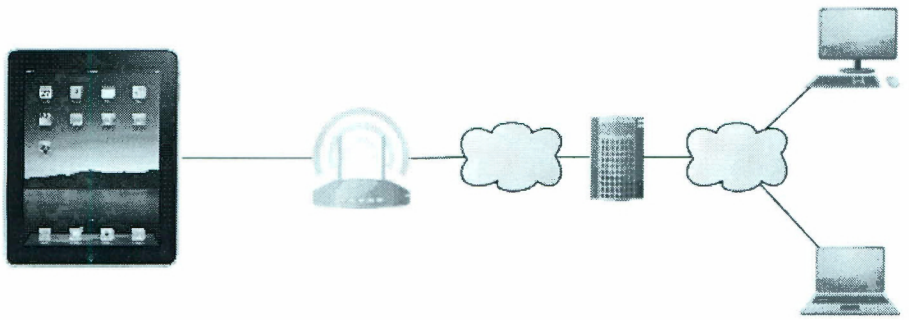


Figura 10. Desde un equipo terminal se accede a un servicio (portal) de personalización (derecha). El sistema (corriendo en el iPad) consume la información de personalización y la implementa en el juego

A partir de este punto, el documento se centra en el Sandtrace Framework, es decir, la parte que correrá de manera nativa. Se utiliza una muestra de lo que el servicio de personalización publicaría (aunque no corresponda la especificación en este punto, se aclara que se trata de un archivo XML con todos los parámetros personalizados). Por ello, se continua con el diseño del Framework siguiendo una metodología que va de lo general a lo particular.

3.4.2. Principales casos de uso

Dos son los casos de uso más importantes del sistema. El primero de ellos es *lanzar evento*, en donde el desarrollador informa al sistema que ha ocurrido un evento (figura 11).

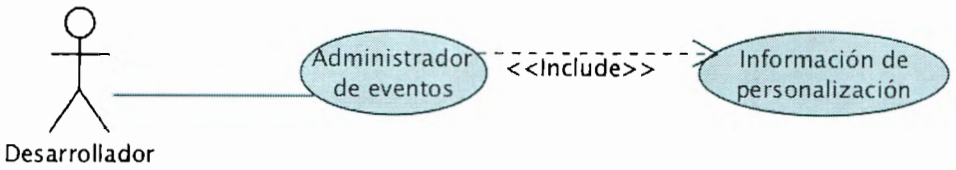


Figura 11. Lanzar evento

El segundo importante caso de uso es *solicitar escena*. En él, el desarrollador pregunta al sistema cuál es la siguiente mejor escena (figura 12).

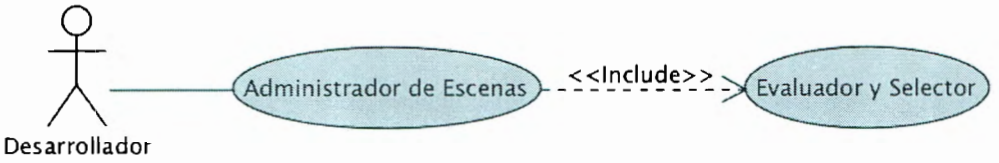


Figura 12. Solicitar escena

Ambos casos de uso requieren de un elemento en común, llamado *información de personalización* (figura 13).

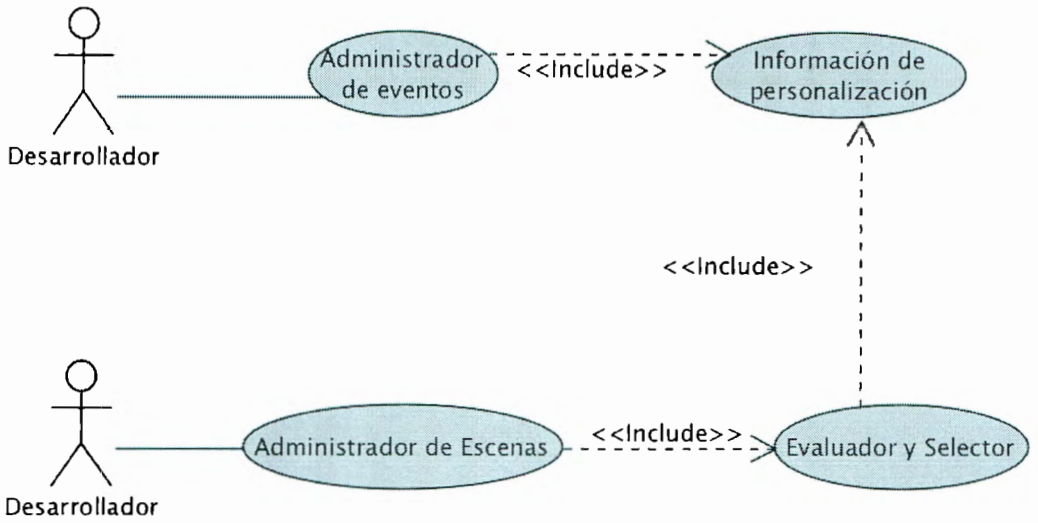


Figura 13. Relación entre los casos de uso

3.4.3. Diagrama de Componentes

El diagrama básico de la figura 14 muestra un acercamiento general a los principales componentes del sistema.

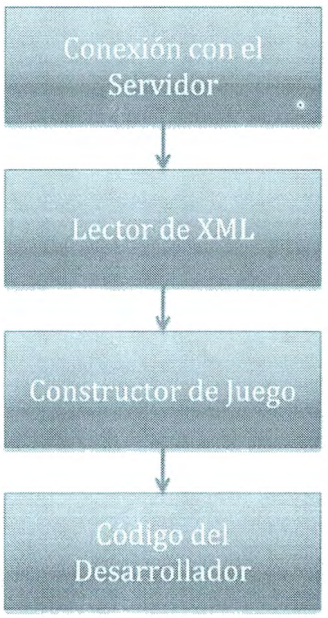


Figura 14. Diagrama fundamental del Sandtrace Framework

Podemos observar cuatro niveles. El más alto de ellos representa la comunicación con el servidor en donde se encuentra el archivo de personalización. Inmediatamente debajo, el módulo de lectura de ese archivo, el cual da pie al tercer bloque, encargado de la construcción de la abstracción de juego, para finalmente llegar al código en el que el desarrollador implementará la funcionalidad proporcionada por el sistema. Dicha funcionalidad se obtiene directamente de la abstracción de juego y por tanto depende solamente de ella.

Por tal motivo, se desglosa qué es lo que esta abstracción representa (ver figura 9). En el nivel más alto, tenemos un bloque azul que lo comprende todo y que llamaremos **juego**. El **juego** contiene, a su vez, a los **niveles**, delimitados en el diagrama con líneas rojas. Observamos que cada **nivel** consta de, *al menos* una **escena**. Finalmente, sabemos que cada escena contiene **eventos**.

Con todo lo anterior en mente, se propone formalmente el diagrama de componentes, representado en la figura 15.

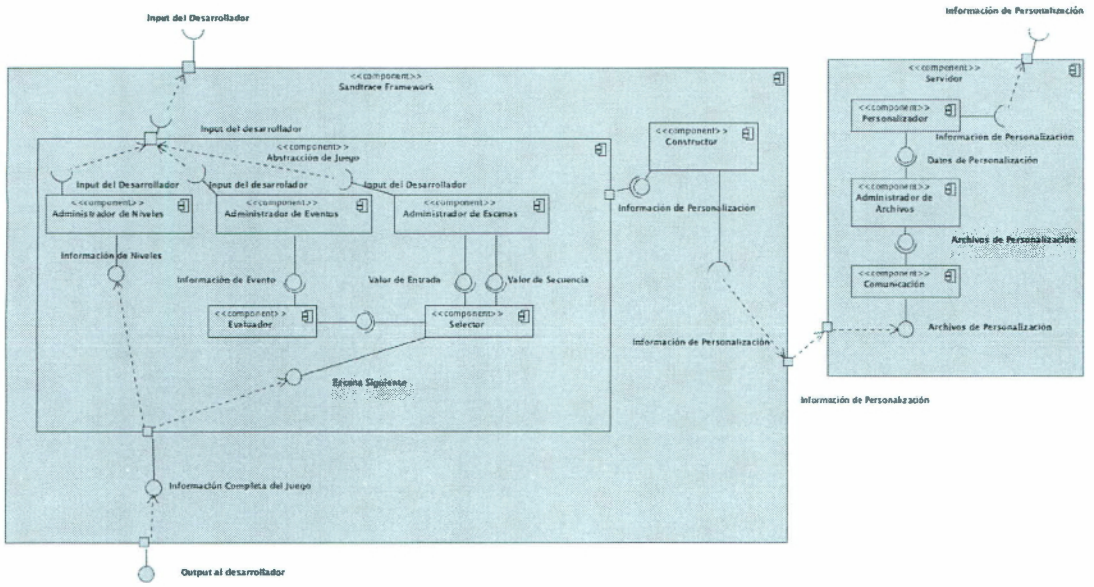


Figura 15. Diagrama de componentes

Del diagrama entonces observamos lo siguiente:

1. Un componente llamado "Servidor", sobre el cual no se entrará en detalle pero sabemos que permite la personalización remota
2. Un componente, llamado Sandtrace Framework, que representa el sistema completo
    a. Dentro de Sandtrace Framework tenemos al componente Constructor, encargado de crear la abstracción del juego
    b. Tenemos también al componente juego
        i. Dentro de juego tenemos al administrador de niveles
        ii. Dentro de juego tenemos también al administrador de eventos

iii. Dentro de juego tenemos también al administrador de escenas

iv. Un componente evaluador

v. Un componente selector

Todos los componentes mostrados, en conjunto, responderán para satisfacer los requerimientos del sistema. A continuación una explicación de cada componente.

3.4.3.1. Sandtrace Framework

Se trata del sistema completo. Permite aplicar en el juego, los parámetros de personalización recibidos desde el componente servidor. Requiere información del desarrollador y regresa información al desarrollador.

3.4.3.2. Constructor

Como su nombre indica, se encarga de leer los parámetros de personalización y con ellos construir una **abstracción** del juego, siguiendo la jerarquía juego→nivel→escena→evento.

3.4.3.3. Juego

Tras finalizado el trabajo del constructor, Juego contiene el total de la información requerida por el desarrollador para implementar el Sandtrace Framework en el desarrollo de su videojuego.

3.4.3.4. Administrador de Niveles

Controla la información relacionada con la **secuencia** del juego (nivel tras nivel).

3.4.3.5. Administrador de Escenas

Gestiona la información de la unidad fundamental del juego, es decir, las escenas.

3.4.3.6. Administrador de Eventos

Se encarga de gestionar la información de los eventos.

3.4.3.7. Evaluador

Con la información obtenida del administrador de eventos, evalúa una escena.

3.4.3.8. Selector

Con la información obtenida del evaluador y del administrador de escena, selecciona la siguiente mejor escena.

3.4.4. Diagrama de Clases

Para el Framework aquí presentado se desarrollan las siguientes clases (con la nomenclatura tal como se muestra):

1. Game
2. Level
3. Scene
4. _Event
5. getParams

Cada una de ellas contiene atributos y métodos que permiten la funcionalidad requerida. En la figura 16 se expone la relación que existe entre ellas.

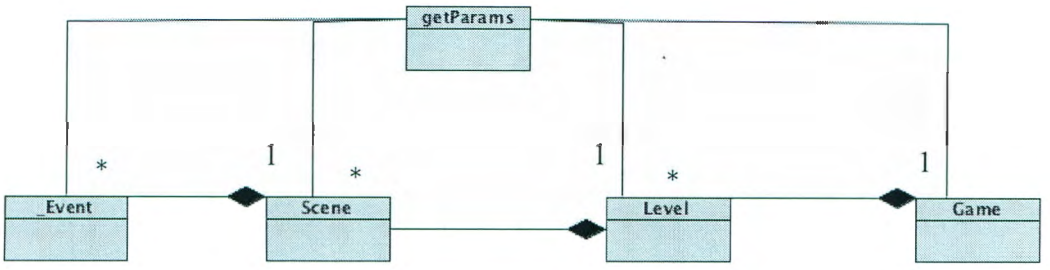


Figura 16. Diagrama de clases

De este modo, se representa la abstracción de videojuego propuesta (Juego→nivel→escena→evento) y la relación con una clase que obtiene la información del archivo XML de personalización y, con ella, construye dicha abstracción. Ya que el diagrama no contiene información a profundidad de las clases, se muestra enseguida qué atributos y métodos contienen, y la funcionalidad de cada uno de ellos.

3.4.4.1. Clase getParams

Comenzamos por aquí por ser esta clase la encargada de obtener toda la información de personalización (figura 17) y de construir con ella todos los objetos complejos Level que se almacenarán en la clase estática Game. Es por tanto un ***builder***[5]. En el anexo *diagrama de secuencia del builder* se encuentra precisamente la secuencia de construcción.

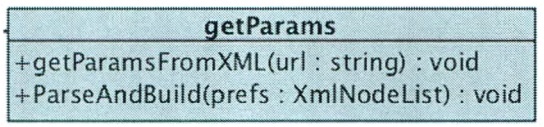


Figura 17. Clase getParams

[5] Se refiere al patrón de diseño de software conocido como *builder*. Se caracteriza por la construcción de objetos complejos.

Atributos: No contiene.

Métodos:

- getParamsFromXML: Se conecta con el servicio XML y obtiene el texto con los parámetros
- ParseAndBuild: Como su nombre indica, parsea el XML y construye la abstracción del Juego, es decir, recurre a las fábricas de las clases _Event, Scene y Level para construir y llenar con los datos del XML, instancias de ellas. Finalmente guarda todas las escenas en un diccionario en la clase estática Game.

3.4.4.2. Clase _Event

Se trata de la clase que permite la definición de los eventos (figura 18).

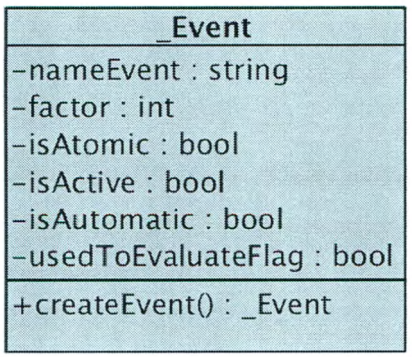


Figura 18. Clase _Event

Atributos:

- nameEvent: Nombre del evento
- factor: Valor numérico normalizado entre -10 y 10 que permite asignar una calificación a cada evento
- isAtomic: Define si el evento es atómico

- isActive: Define si el evento se encuentra activo (es susceptible de lanzarse)
- isAutomatic: Define si el evento es automático
- usedToEvaluateFlag: Determina si el evento ha sido ya utilizado para la evaluación de la escena en que se encuentra

Métodos:

- createEvent: Fábrica
- Getters y Setters para todos los atributos

3.4.4.3. Clase Scene

Contiene a los eventos y otros elementos (figura 19).

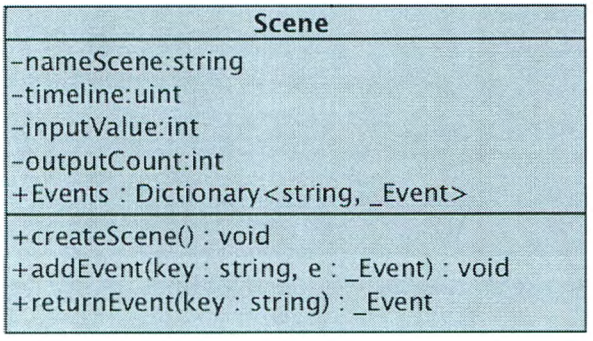


Figura 19. Clase Scene

Atributos:

- nameScene: Nombre de la escena
- timeline: Valor numérico entero sin signo (positivo) que **indica** el nivel en donde debe ir esa escena

- inputValue: Valor numérico utilizado para seleccionar esta de entre otras escenas, con base en la calificación de la escena anterior
- outputCount: Calificación de la escena actual
- Events: Diccionario que contiene todos los eventos de la escena, asociados al nombre de cada uno

Métodos:

- createScene: Fábrica
- addEvent: Añade un objeto _Event al diccionario
- returnEvent: Regresa un objeto _Event del diccionario
- Getters y Setters para todos los atributos

3.4.4.4. Clase Level

Como es de esperarse, esta clase contendrá a Scene, junto con atributos y métodos propios (figura 20).

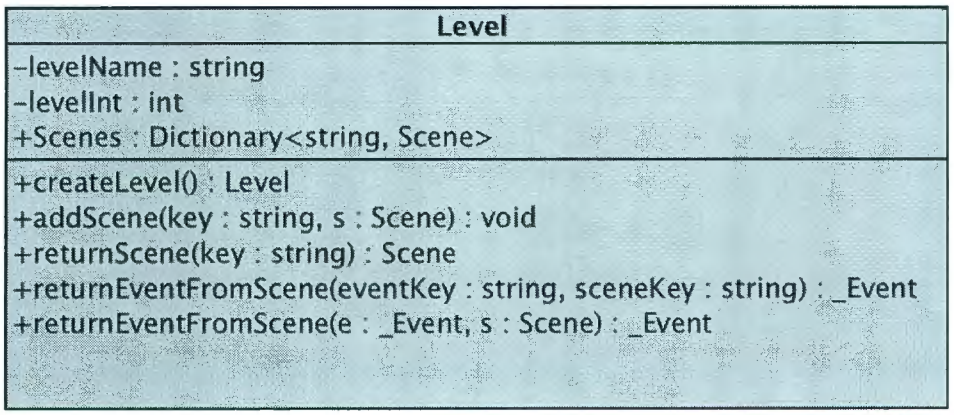


Figura 20. Clase Level

Atributos:

- levelName: Nombre del nivel
- levelInt: Número del nivel
- Scenes: Diccionario que contiene a las escenas de ese nivel

Métodos:

- createLevel: Fábrica
- addScene: Añade una objeto Scene al diccionario
- returnScene: Regresa un objeto Scene del diccionario
- returnEventFromScene: Regresa un objeto _Event del diccionario Events en el objeto Scene referenciados ambos por medio de cadenas
- returnEventFromScene: Regresa un objeto _Event del diccionario Events en el objeto Scene referenciados ambos por medio de objetos (un objeto _Event y un objeto Scene)
- Getters y Setters para todos los atributos

3.4.4.5. Clase Game

Es la clase más importante del Santrace Framework, ya que desde ella, el desarrollador puede acceder a toda la información de personalización, implementarla, lanzar eventos, evaluar escenas y obtener la mejor escena siguiente (figura 21). Se trata de una clase estática, a diferencia de las otras, en la que la información de todos los niveles (y por ende del resto de la personalización del juego) se almacenan en un diccionario público llamado Levels.

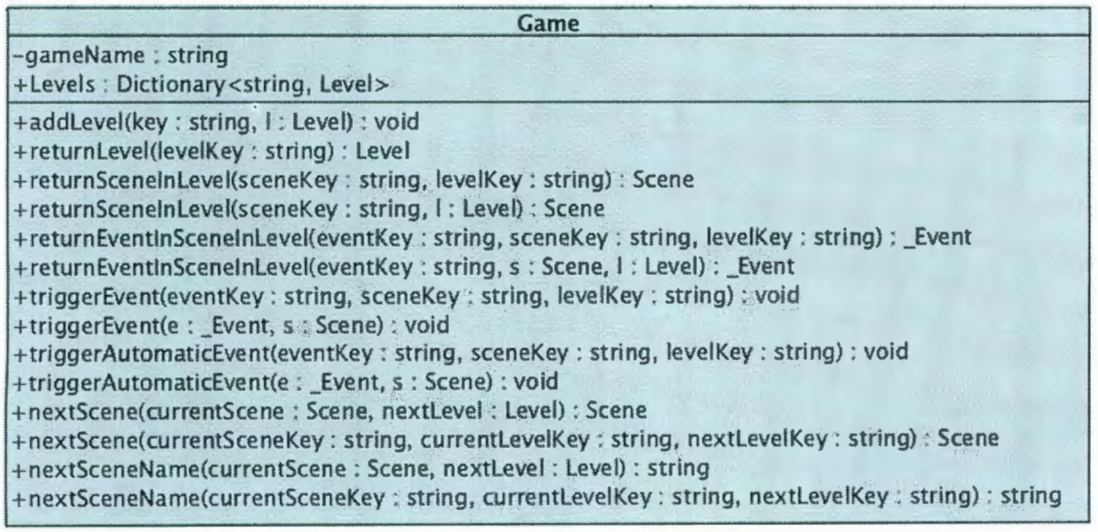


Figura 21. Clase Game: La más importante del Sandtrace Framework

Atributos:

- gameName: Cadena con el nombre del juego
- Levels: Diccioario con todos los objetos Level

Métodos:

- No contiene Fábrica, se utiliza como clase estática
- addLevel: Añade un objeto Level al diccionario
- returnLevel: Regresa un objeto Level del diccionario
- returnSceneInLevel: Regresa un objeto Scene de un nivel en particular, a través de objetos
- returnSceneInLevel: Regresa un objeto Scene de un nivel en particular, a través de cadenas
- returnEventInSceneInLevel: Regresa un objeto _Event de una escena particular de un nivel particular, a través de objetos

- returnEventInSceneInLevel: Regresa un objeto _Event de una escena particular de un nivel particular, a través de cadenas
- triggerEvent: Lanza un evento específico por medio de objetos
- triggerEvent: Lanza un evento específico por medio de cadenas
- triggerAutomaticEvent: Lanza un evento automático específico por medio de objetos
- triggerAutomaticEvent: Lanza un evento automático específico por medio de cadenas
-
- nextScene: Regresa, por medio de objetos, un objeto Scene con la siguiente mejor escena (con base en la calificación de la actual y el valor entrada de todas las posibles siguientes)
- nextScene: Regresa, por medio de cadenas, un objeto Scene con la siguiente mejor escena (con base en la calificación de la actual y el valor entrada de todas las posibles siguientes)
- nextSceneName: Regresa, por medio de objetos, una cadena con el nombre de la siguiente mejor escena (con base en la calificación de la actual y el valor entrada de todas las posibles siguientes)
- nextSceneName: Regresa, por medio de cadenas, una cadena con el nombre de la siguiente mejor escena (con base en la calificación de la actual y el valor entrada de todas las posibles siguientes)
- Getter y Setter para el único atributo privado

3.5. Mecanismo de evaluación y selección de escenas

Se abre un apartado especial para exponer un mecanismo de evaluación y selección de escenas, presente en el Sandtrace Framework.

Al revisar la sección del diagrama de clases, se observa que en las escenas hay atributos llamados *inputValue*, *outputCount*, y *timeline*. Asimismo, en cada evento hay un atributo llamado *factor*.

El mecanismo de evaluación consiste en tomar los factores de cada evento en una escena (que deberán encontrarse previamente normalizados para que la suma del total de ellos no exceda un valor de 10 ni sea menor a -10) y cada vez que un evento sea lanzado, añadir el valor del factor relacionado a él al atributo *outputCount*. De esta manera, al finalizar una escena, se tendrá un registro de los factores de todos los eventos lanzados. El personalizador asigna previamente un valor al atributo *inputValue*, definido por él en concordancia con sus objetivos. Al final, cuando se trate de pasar a la escena siguiente, el sistema busca el valor de *outputCount* de la escena actual, y lo compara con todos los valores de *inputValue* de las posibles escenas siguientes; Al final, se selecciona la escena cuyo *inputValue* es igual al *outputCount* de la escena previa, o es el **más cercano** de todos. Si hay dos o más escenas que se encuentren a la misma "distancia" numérica de la escena anterior, se seleccionará la primera de ellas, todo esto en concordancia con los valores de *timeline* de cada escena en cuestión, para corroborar la secuencia lógica de la historia del juego.

3.6. Implementación

El Sandtrace Framework está desarrollado en su totalidad en lenguaje C#.

Las cinco clases de las que consta deben ser incluidas en un proyecto y hacer referencia a ellas desde el código del usuario.

En este punto del documento se detalla la manera de incluir al Sandtrace Framework en un proyecto de desarrollo de un videojuego.

3.6.1. Prerrequisitos

Es indispensable la participación de un programador o desarrollador de videojuegos familiarizado con la programación orientada a objetos y con C#.

3.6.1.1. Videojuego

El videojuego debe poder ser representado de manera abstracta, en concordancia con la estructura expuesta anteriormente (Juego→niveles→escenas→eventos).

Es necesario contar con toda la información relacionada con los cuatro niveles jerárquicos de esta abstracción de videojuego durante el proceso de desarrollo del mismo (es decir, es necesario haber concluido la etapa de diseño de juego, conocer la historia y los eventos de éste, antes de implementar al Sandtrace Framework durante el proceso de desarrollo como tal).

El desarrollador debe tener acceso a los nombres (identificadores) de los eventos, escenas, niveles y juego, ya que ellos (las cadenas de caracteres que los conforman), serán de suma importancia y se utilizarán continuamente a lo largo de la implementación del Framework. Tal acceso puede ser la posesión (y comprensión) del archivo XML utilizado para la personalización, o la utilización de un ambiente de desarrollo similar al Microsoft Visual Studio®, desde el cual puedan "abrirse" en tiempo de depuración, los

objetos y clases para ver los valores que poseen sus atributos y de tal suerte, tener acceso a toda la información necesaria para su implementación en el desarrollo del juego.

3.6.2. Estructura del XML

Conocer la estructura del XML del que se alimentará el sistema es de vital importancia. Aquí se explica dicha estructura.

El archivo XML consta de un gran nodo raíz llamado "prefs". Dentro de este nodo, se encuentra el primer nivel de nodos hijo, llamados "level". Dentro de ellos tendremos los nodos de escena, llamados "scene" y finalmente, como hijo de "scene" aparece el nodo "events" cuyos hijos son nodos "event" y contienen la información relacionada con los eventos. Esta estructura en esencia es la misma abstracción de la definición de videojuego propuesta anteriormente, pero aterrizada utilizando XML. Cada nodo posee además de sus nodos hijo, atributos y valores propios. A continuación se desglosan, uno a uno, todos ellos.

3.6.2.1. Nodo raíz: prefs

La captura siguiente (figura 22) muestra cómo se ve el nodo raíz contraído, desde un editor de texto. Este nodo contiene toda la información del juego y su contraparte en el Framework es la clase Game.

Figura 22. Nodo raíz prefs

El primer nivel dentro de prefs es level. La figura 23 muestra esta relación.

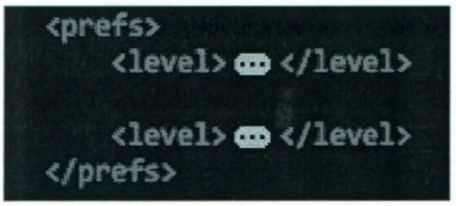


Figura 23. Contenido de prefs

3.6.2.2. Nodo level

Es el primer nivel de nodos hijo en prefs. Corresponde a la clase Level del Sandtrace Framework. La captura (figura 24) muestra el contenido del nodo level.

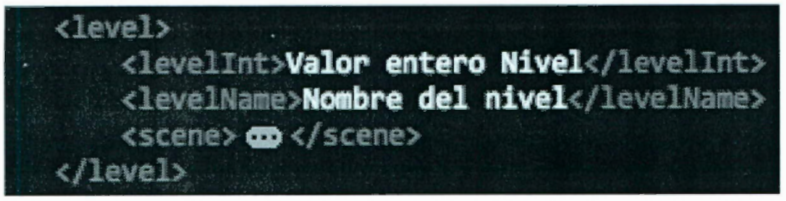


Figura 24. Contenido de level

Tal como la imagen sugiere, los campos "levelInt" y "levelName" poseen el número de nivel y el nombre del mismo. El nodo "scene" es el siguiente en la jerarquía del juego y contiene toda la información de cada escena.

3.6.2.3. Nodo scene

En este nodo se encuentra la información de lo que se expuso como la unidad fundamental del juego y corresponde a la clase Scene del Framework. La figura 25 presenta el contenido de scene.

```
<scene>
    <timeline>Valor entero del timeline (nivel al que pertenece)</timeline>
    <name>Nombre de la escena</name>
    <inputValue>Valor de entrada (-10 a 10)</inputValue>
    <events> ... </events>
</scene>
```

Figura 25. Contenido del nodo scene

Se observan los siguientes campos:

- Timeline: Tal como se explica en la imagen, representa el valor entero del nivel al que la escena pertenece
- Name: nombre de la escena
- inputValue: Valor numérico requerido por la escena para ser seleccionada como mejor escena siguiente
- events: nodo que contiene todos los eventos de la escena en cuestión

El nodo events, continuando con el siguiente nivel jerárquico, contiene, como su nombre indica, a todos los eventos de una escena.

3.6.2.4. Nodo events

Para cada escena hay un nodo events. Éste contiene a todos los eventos de una escena, tal como se muestra en la siguiente captura (figura 26).

```
<events>
    <event> ... </event>

    <event> ... </event>

    <event> ... </event>
</events>
```

Figura 26. Contenido de events

En cada nodo event del nodo events, se encuentra la información de personalización de cada evento.

3.6.2.5. Nodo event

Corresponde a la clase _Event del Sandtrace Framework. Contiene información propia de cada evento. A continuación su contenido (figura 27).

```
<event>
    <name>Nombre del Evento</name>
    <isActive>0 ó 1</isActive>
    <isAtomic>0 ó 1</isAtomic>
    <isAutomatic>0 ó 1</isAutomatic>
    <factor>Valor del factor (-10 a 10)</factor>
</event>
```

Figura 27. Contenido de cada nodo event

Campos:

- Name: Contiene una cadena con el nombre del evento
- isActive: Indica si el evento se encuentra o no activo (valor de 0 ó 1)
- isAtomic: Indica si el evento es o no atómico
- isAutomatic: Indica si el evento es o no automático
- factor: Valor numérico con que el evento en cuestión contribuye en la escena al ser lanzado

Al iniciarse el videojuego, toda la información contenida en el archivo XML será parseada y almacenada en objetos del Sandtrace Framework, y estará disponible para su

utilización durante el resto de la sesión de juego[6]. Aunque se hará mención en el manual de usuario, se hace notar que si se desea hacer la información persistente (para utilizarse en sesiones de juego subsecuentes, que **no** inicien desde cero), debe almacenarse el diccionario llamado "Levels", de la clase estática Game. En una sesión de juego posterior, se carga este diccionario y se apunta a él desde la clase Game para poder acceder a todos los parámetros de personalización del juego.

[6] Sesión de Juego: Periodo de tiempo desde que se ejecuta la aplicación del juego hasta que se cierra o se finaliza la misma.

4. Caso de Estudio: Private Eye

Partiendo de cero, pero con ayuda de estudiantes de licenciatura del ITESM CCM, de las carreras de Animación y Arte digital e Ingeniería en Tecnologías Computacionales, fue posible traer a la realidad las primeras dos escenas de un juego de video, llamado PrivateEye.

Para la creación del juego en sí, se utilizó una plataforma de desarrollo conocida como Unity®; el modelado de personajes y edificios se realizó en Autodesk Maya® y todo el código detrás del juego fue programado en los lenguajes javascript y C#.

Durante el desarrollo del juego (e inclusive una vez finalizado el mismo, puesto que se tiene total acceso al código) se incluyen las librerías del Sandtrace Framework y se implementa su funcionalidad para ser demostrada en el presente trabajo de tesis.

A continuación se describe el juego como tal, para posteriormente explicar las funciones que adquiere al utilizar el Sandtrace Framework.

4.1. Videojuego: Private Eye

4.1.1. Modelo Espacial del Juego

La historia del juego se desarrolla en una ciudad metropolitana retro-futurista.

Es una ciudad oscura, sucia y triste. Hay dos zonas sociales específicamente marcadas: la zona empresarial donde residen los egocéntricos ejecutivos, despilfarrando el dinero de

los impuestos, y la zona de pandillas, donde escasea el trabajo y la gente tiene que vivir del crimen para poder alimentar a su familia al siguiente día.

En el centro de la ciudad hay un pequeño parque; es la única zona verde y con alegría entre la triste atmósfera llena de humo y luces de neón. Es un lugar protegido, ya que alberga muchas especies de árboles y flora que están en peligro de extinción; pero aún más importante es el único lugar donde los niños pueden jugar y ser inocentes, hasta el momento que crezcan y tengan que decidir unirse a la burocracia o al círculo criminal. O tal vez no hacer nada de sus vidas, la cual sería la respuesta más sincera para contribuir al ciclo de corrupción.

A los lados del parque hay dos edificios residenciales grandes. Aquí se alberga la gente de clase media, quien tiene vista al parque para poder observar a sus hijos. En uno de estos edificios vive el personaje principal.

La zona de empresarial se compone de rascacielos altos y descuidados. Sin excepción alguna, en la fachada de cada edificio se exhibe en luces de neón el nombre de cada compañía. Como si trataran de atraer clientes de la misma manera que una trampa de luz atrae a las moscas.

Por el otro lado, la zona de pandillas se compone en su mayoría de edificios habitaciones. Estos están amontonados, como si el gobierno hubiese querido esconder a la gente pobre

en una esquina de la ciudad. Sin embargo en el espacio que sobra entre los edificios, se crea una intrincada red de callejones, donde ocurren la mayoría de los crímenes.

Hay varios anuncios flotantes a lo largo de la ciudad. Es común que sufran alguna descompostura y caigan en la ciudad bloqueando calles y provocando tráfico.
Enfrente del parque se encuentra la biblioteca pública de la ciudad. El detective puede entrar para aprender más acerca de su entorno; También hay un quiosco informativo en el parque; el mapa de la ciudad y una descripción de las diferentes zonas de la ciudad pueden ser obtenidos aquí.

4.1.2. Historia Principal del Juego

En este punto se describe la historia del juego hasta finalizado el primer nivel. Cabe aclarar que la descripción es narrativa y el juego desarrollado no contiene (en esta primera versión) todos los detalles descritos a continuación.

Un detective junto con un computológo, trabajan juntos para resolver un caso de asesinato serial que marcó sus vidas.

El personaje principal es precisamente el detective, de nombre Dan. Su hermano menor fue asesinado cuando Dan era aún pequeño, curiosamente ese mismo día, su padre desapareció. Debido a este hecho, la policía confirmó que el perpetrador del delito había sido nada más y nada menos que el padre de Dan.

Años más tarde, Dan se encontrará con el personaje secundario más importante del juego: Tommy. Computólogo distraído y ensimismado, posee una historia similar a la del

protagonista, en tanto que su familia entera fue enviada al otro mundo en manos de un homicida en serie.

Ambas historias convergen en un punto del juego en el que los personajes se unen para resolver un mismo caso.

La historia comienza cuando el detective recibe un correo anónimo (de Tommy), en el cual le cuenta que su padre no fue quien mató a su hermano. Por costumbre después de años de investigar casos poco interesantes, sale en busca del origen del mensaje. Al salir de su casa comienzan los sucesos que irán definiendo el curso de la historia; un taxi está a punto de atropellar a un niño entrando al parque. El detective enfrenta la opción de tratar de salvar el niño (en su mente, su forma de salvar a su hermanito), o no (quedando en shock por el trauma de su pasado). Al salvarle o no (la elección afectaría a su moral), el detective tendrá otra opción de perseguir al taxista o seguir con su misión principal.

Si elige perseguirlo, investiga el caso que le lleva a la zona pleito de la ciudad en donde lo puede detener. Al lograr detenerlo el detective entrará en un edificio contiguo para llamar a la policía. Es justo en este punto cuando se da percata de que es ese el lugar que estaba buscando originalmente (antes de desviar su atención y centrarla en el taxista). La señorita en la caja reconoce al conductor del taxi como el sujeto que trató de abusar de ella días antes. Una vez que es detenido el criminal, la encargada del lugar permite al detective tener total acceso a las computadoras y otros servicios del café Internet, sin costo alguno.

Si no elige investigar el caso, el detective tendría que encontrar el café Internet por sí solo. Al hallarlo, enfrentará a la encargada del establecimiento, una joven punk, quien no le quiere dejar pasar de buenas a primeras, por lo que será necesario convencerla o entrar de otra manera. Si se entra al café en contra de la voluntad de la dueña, ella llamará a la policía; El jugador tendrá que intentar escapar o no hacer nada y ser atrapado. Al obtener la información necesaria, el detective podrá avanzar al departamento del computólogo (es en este punto donde finaliza el primer nivel).

4.1.2.1. Personajes Jugables

4.1.2.1.1. Detective Dan

Personaje principal, su hermano menor fue asesinado cuando era un menor de edad. Su padre desapareció el mismo día.

Posee un pasado oscuro que lo llevó a convertirse en detective. Tiene una personalidad complicada, por veces serena pero en ocasiones muy explosiva.

Mientras se encontraba en una de sus investigaciones, se encuentra involucrado en una explosión, por el impacto pierde el conocimiento y el ojo; cuando recobra el conocimiento se encuentra en lo que aparenta ser un laboratorio, sin embargo nota algo extraño en su visión. Un científico aparece y le explica que lo encontró inconsciente y sin ojo, y como no logró encontrar voluntarios, decide probar con él su nueva creación: un ojo experimental.

Su meta es encontrar al asesino de su hermano con la ayuda de distintas pistas a través de los diferentes niveles.

Puede usar armas no mortales como macanas de luz. No usa armas de fuego (figura 28).

Figura 28. Detective Dan

4.1.2.1.2. Computólogo Tommy

Ingeniero en Sistemas, distraído y ensimismado, posee una historia similar a la del protagonista; casi para finalizar el primer nivel, da información valiosa al detective para continuar su búsqueda. Su familia y su mejor amigo fueron asesinados por el mismo sujeto.

Aparece en varios niveles para ayudarle al detective a resolver los casos.

Su meta es ayudar al detective a encontrar el asesino.

Tiene alto conocimiento de las computadoras, puede hackear sistemas para obtener información valiosa (figuras 29 y 30).

Figura 29. Concepto Computólogo Tommy

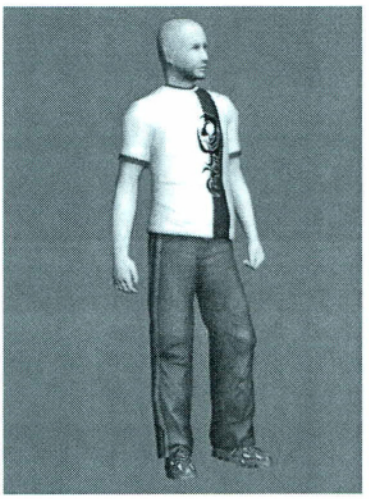


Figura 30. Modelo de Tommy

4.2. Interacción detallada con el Sandtrace Framework

Para mostrar la funcionalidad, aunque se completó el primer nivel del juego, se desarrollaron escenas de prueba adicionales que no forman parte de la historia del mismo. También se emplearon tres ejemplos de esquemas distintos de personalización, a fin de mostrar diferentes propiedades del sistema.

4.2.1. Ejemplo de Personalización A

El primer esquema de personalización contiene tres niveles (dos más la pantalla de inicio, considerada el nivel cero). En el anexo XML A se adjunta el archivo XML. La figura 31 representa la distribución de las escenas y niveles con base en este archivo de personalización.

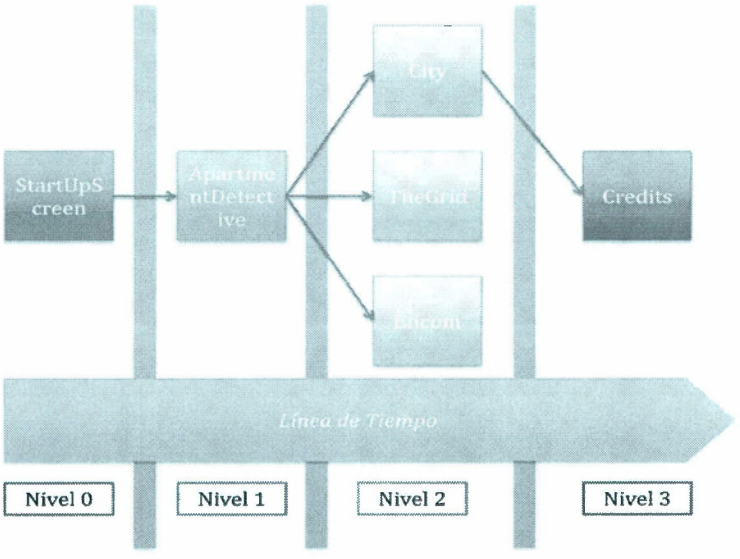


Figura 31. Distribución del Juego con base en el primer archivo de personalización (Los nombres de las escenas son puramente ilustrativos, no buscan hacer referencia ni reclaman propiedad intelectual de otros autores)

Se observa entonces la presencia de 4 niveles, de los cuales solamente 3 son interactivos y solamente 2 son jugables. El nivel cero contiene únicamente las pantallas de inicio,

donde se pregunta al usuario si desea continuar o si desea iniciar un juego nuevo (figura 32). En este caso, es al cargarse esta escena cuando se manda a llamar la clase que lee y construye la abstracción del juego a partir del archivo XML. Cabe mencionar que la información de una sesión de juego anterior no es persistente y si se desea que así sea es necesario almacenar la información de la clase estática Game y posteriormente recuperarla y vaciarla en la misma clase pero de la nueva sesión de juego.

Figura 32. Pantalla de inicio

Continuando con esta primera personalización y una vez que se comienza el juego (nivel 1), el usuario tendrá tres escenas posibles de las cuales solamente una será utilizada en el siguiente nivel (nivel 2). De acuerdo con el archivo de personalización, tales escenas tienen los valores de entrada como sigue:

- City: inputValue = 3

- TheGrid: inputValue = 1
- Encom: inputValue = 2

Dado que la escena ApartmentDetective (figura 33) es la única que se encuentra a continuación de StartUpScreen, se cargará ésta (a menos que se continúe una sesión de juego anterior). Jugando en la escena ApartmentDetective, hay 3 eventos importantes que tienen cabida en la selección de la escena siguiente. Ellos son:

- ReadMail (figura 34):
    - factor = 1
    - isActive = 1
    - isAutomatic = 0
    - isAtomic = 1
- TakeMap (figura 35):
    - factor = 1
    - isActive = 1
    - isAutomatic = 0
    - isAtomic = 1
- LookPaint (figura 36):
    - factor = 1
    - isActive = 1
    - isAutomatic = 0
    - isAtomic = 1

De esta manera, todos los eventos tienen el mismo "peso" en la decisión, pero el realizar uno, dos o los tres, afecta la escena siguiente.

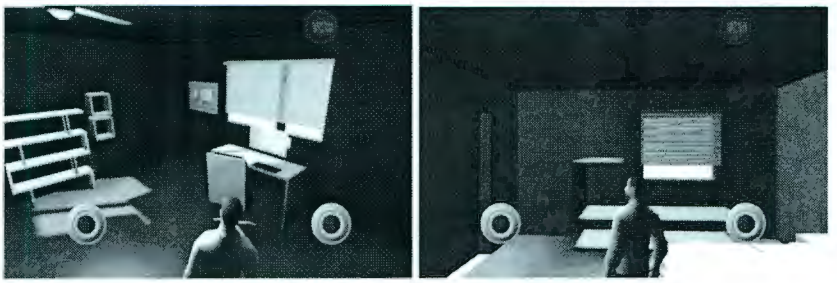


Figura 33. Departamento del Detective

Figura 34. Evento ReadMail

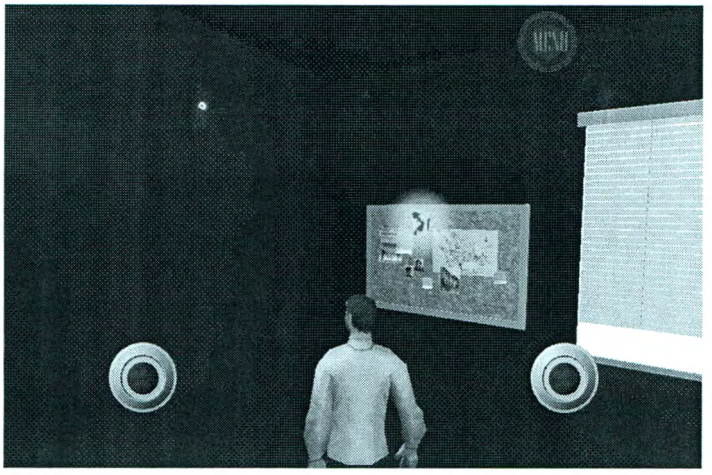


Figura 35. Evento ReadMap

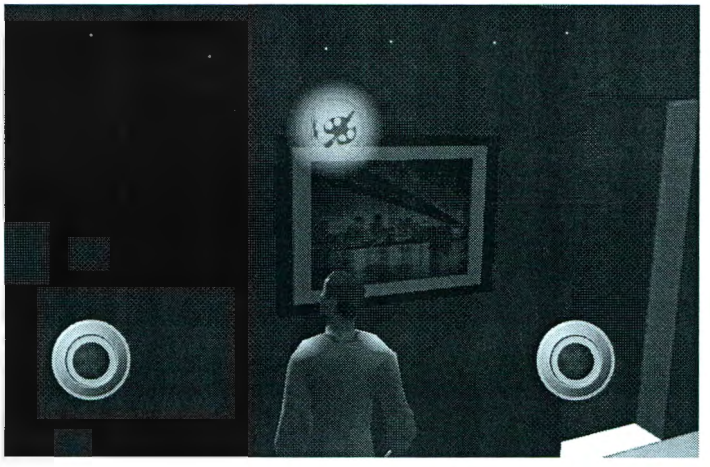


Figura 36. Evento LookPaint

Las posibilidades entonces son, que el usuario no provoque ninguno de los eventos arriba mencionados, o que provoque uno, dos o los tres. En este punto en particular, no es importante la "calidad" de los eventos sino la "cantidad" de ellos en la definición de la escena siguiente. De esta manera, si el jugador provoca ningún evento, o incluso uno de ellos, la escena siguiente será necesariamente TheGrid (figura 37).

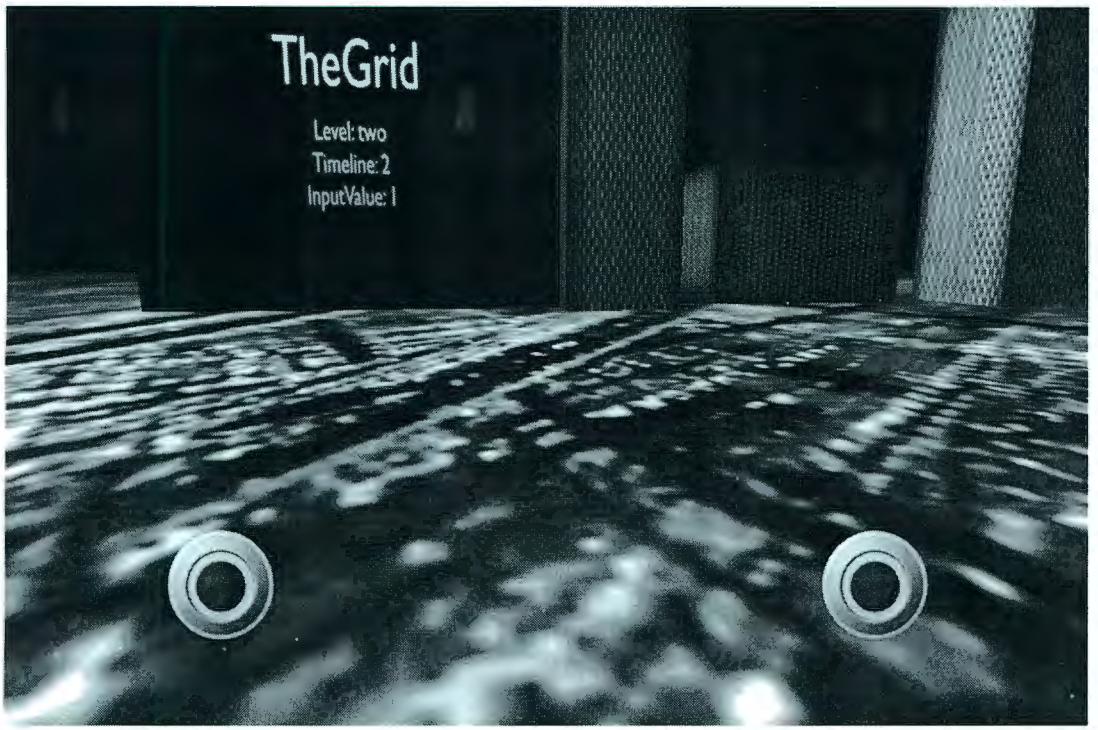


Figura 37. Escena de prueba "TheGrid". Obsérvese la textura que indica con qué valores se llega a ella y en qué nivel se encuentra

En contraste, si se realizan 2 eventos, se llegará a la escena de prueba llamada Encom (figura 38), y finalmente, si se llevan a cabo los 3 eventos, la escena siguiente sería la City (figura 39).

Finalmente, para llegar a la escena Credits, es necesario estar en la escena City y terminar el nivel. No existen parámetros de personalización que definan un comportamiento diferente en este caso.

Figura 38. Escena de prueba "Encom"

Figura 39. Escena "City"

4.2.2. Ejemplo de Personalización B

Este esquema es una modificación (reducción, básicamente), del esquema anterior. Tomamos entonces como referencia el diagrama de la figura 31, pero suponemos, en adición, que se desea llegar **directamente** de la pantalla de inicio, a la escena City, sin pasar por el departamento y sin lanzar los eventos que allí se encuentran.

En este caso, es necesario utilizar un archivo XML con algunas modificaciones (adjunto en el anexo XML B) en la escena ApartmentDetective (que no se jugará); en concreto, el tipo de eventos que ella contiene serán cambiados a automáticos. De esta manera:

- ReadMail:
  - factor = 1
  - isActive = 1
  - isAutomatic = 1
  - isAtomic = 1
- TakeMap:
  - factor = 1
  - isActive = 1
  - isAutomatic = 1
  - isAtomic = 1
- LookPaint:
  - factor = 1
  - isActive = 1

- isAutomatic = **1**
- isAtomic = 1

Posteriormente, el desarrollador debe asegurarse de que en su código, se mande a llamar la escena ApartmentDetective pero **no se cargue**, a fin de que los eventos automáticos sean lanzados y con el valor de ellos se mande a llamar la escena siguiente y sea, **ahora sí**, cargada. De esta manera, el jugador **nunca** se percata de que, desde la abstracción del videojuego, se abrió la escena ApartmentDetective, se evaluó con los eventos dentro de ella, todos de tipo automático y finalmente se llamó a la escena siguiente, con base en los valores de los eventos lanzados automáticamente. Es evidente que si se lanzan los 3 eventos del esquema del ejemplo anterior, la escena siguiente a la que el jugador tendrá acceso será necesariamente City, siendo esto lo deseado en nuestra suposición inicial.

A manera de pseudocódigo, esta secuencia podría verse así:

```
//Obtenemos el nombre de la escena siguiente, del nivel 1
string nextSceneName =Game.nextSceneName("StartUpScreen", "zero", "one");

//Con el nombre de la escena siguiente obtenemos el objeto Scene
Scene nextScene = Game.returnSceneInLevel("ApartmentDetective", "one");

//Lanzamos los eventos automáticos
Game.triggerAutomaticEvent("ReadMail", "ApartmentDetective", "one");
Game.triggerAutomaticEvent("TakeMap", "ApartmentDetective", "one");
Game.triggerAutomaticEvent("LookPaint", "ApartmentDetective", "one");

//Finalmente obtenemos la escena siguiente, del nivel 2
nextSceneName = Game.nextSceneName("ApartmentDetective", "one", "two");
```

De esta manera, lo que el jugador ve, inmediatamente después de la pantalla de inicio, es la escena City. El anexo **X** contiene el archivo XML necesario para esta funcionalidad.

4.2.3. Ejemplo de Personalización C

Finalmente, se propone un esquema un poco distinto de los dos anteriores, aunque sigue la secuencia básica de la figura 31. La diferencia radica en que en esta ocasión, los valores de los eventos en la escena ApartmentDetective son diferentes unos de otros, resultando ello en eventos más importantes o más deseables que otros. Los valores de los eventos son:

- ReadMail:
  - factor = 3
  - isActive = 1
  - isAutomatic = 0
  - isAtomic = 1
- TakeMap:
  - factor = 2
  - isActive = 1
  - isAutomatic = 0
  - isAtomic = 1
- LookPaint:
  - factor = 1
  - isActive = 1
  - isAutomatic = 0
  - isAtomic = 1

De esta manera, y comparando nuevamente con los valores de entrada de las tres posibles escenas siguientes, sabemos que:

1. Basta con el evento ReadMail para pasar directamente a la escena City. Sin embargo, el algoritmo provee la funcionalidad de "la escena más cercana", explicada anteriormente, de modo que no importa si se presentan el resto de los eventos, la escena City, al tener el inputValue más alto, seguirá siendo la siguiente
2. Si se ejecuta el evento TakeMap solamente, la escena siguiente será Encom; si se ejecuta tanto el evento TakeMap como el evento LookPaint, la escena siguiente entonces se convertiría en City
3. Si se presenta el evento LookPaint exclusivamente, la siguiente escena sería TheGrid, mientras que si se ejecuta LookPaint con cualquier otro evento, la escena nuevamente resultaría en City

Todo lo anterior presenta entonces muchas combinaciones y posibilidades de caminos a seguir, siempre consistentes con lo asentado en el archivo de personalización.

5. Conclusiones

En el presente trabajo se presenta un marco (SandTrace Framework) para la personalización de videojuegos. SandTrace brinda la posibilidad de definir un orden parcial en las escenas y seleccionar características y valores para ellas y los eventos que éstas contienen, todo ello sin ejecutar el juego (offgame). También permite la ejecución, durante el juego (ingame), de los parámetros antes definidos y la concordancia con éstos, bajo un esquema en el que una vez terminado el proceso de desarrollo del juego, se pueden cambiar los parámetros del mismo y ser reflejados en la aplicación del videojuego, sin necesidad de abrirlo, modificarlo o recompilarlo (a menos que se desee hacer cambios en el código por alguna circunstancia, como mantenimiento o actualización).

En concordancia con los objetivos propuestos para el presente documento, se concluye que el Sandtrace Framework representa una solución, tanto de diseño como de software, para la personalización de videojuegos, en la que se propone y se implementa una abstracción del modelo de videojuego, los elementos estructurales que lo componen y la interacción entre ellos. En este marco de trabajo se propone también una abstracción para la narrativa de un videojuego, en tanto que se definen los elementos característicos de ella y su equivalente en el modelo. Al acoplar ambas abstracciones e integrarlas en el sistema aquí expuesto, se obtiene una arquitectura que permite la personalización de los elementos, tanto de la abstracción del videojuego, como de la abstracción de narrativa, y que se verá reflejada en el videojuego mismo.

La funcionalidad del sistema se comprueba al incluirlo en el proceso de desarrollo de Private Eye, exponiendo en este documento tres escenarios que reflejan la mayoría de las posibilidades de personalización de videojuegos que el Sandtrace Framework ofrece. El primero de ellos muestra presenta una configuración en que los eventos de una escena (ambos parte de la abstracción de videojuego) no son tan importantes en la narrativa, como lo es la cantidad de ellos.

El segundo escenario muestra un control total sobre la narrativa, en tanto que permite sencillamente eliminar una escena que no se desea en la historia del juego.

Finalmente, el tercer escenario otorga un nivel de importancia diferente a cada evento, a partir del cual la narrativa se ve afectada.

Con base en todo lo anterior, es posible afirmar que el sistema satisface los requisitos con que fue diseñado y desarrollado y permite la personalización de un videojuego, en el marco del presente trabajo.

6. Trabajo a Futuro

Se consideran dos elementos importantes a considerar como trabajo por hacer:

1. El desarrollo del sistema generador de archivos XML online y una interfaz de usuario para el mismo. Si bien hasta el momento se cuenta con un servicio XML, la idea sería una interfaz gráfica de usuario muy fácil de usar, en la que el personalizador "tome y arrastre" bloques que representen las escenas, las coloque en espacios que representen los niveles y seleccione las propiedades de los eventos, bajo un esquema que se parezca a otros programas que haya utilizado anteriormente, a fin de hacer la interacción más intuitiva
2. La mejora del modelo de evaluación y selección de escenas, utilizando, por ejemplo, técnicas de sistemas inteligentes

7. Créditos

Es muy importante señalar (por eso el apartado especial) que en la elaboración del videojuego para el caso de estudio, se contó con el apoyo de un grupo de estudiantes, cuya participación se hace constar aquí. Ellos fueron:

Jesse Jacobs

Vianey Camacho

Anahí Vázquez

Alejandro Méndez

Emilio Verduzco

Eliss Mercado

Carlos Bonilla

8. Referencias

Choose Your Own Adventure. (2012). *History of CYOA*. Recuperado el 22 de marzo de 2012 desde www.cyoa.com: http://www.cyoa.com/pages/history-of-CYOA

ATM Ltd and Online Media. (1996). *Cambridge iTV trial*. Recuperado el 10 de enero de 2012 desde http://koo.corpus.cam.ac.uk/: http://koo.corpus.cam.ac.uk/projects/itv/

Blázquez, D. (2006). *EXEBlog. Desarrollo de Videojuegos y Más*. Recuperado el 4 de enero de 2012 desde exelweiss: http://www.exelweiss.com/blog/35/la-curva-de-aprendizaje/

etymonline. (2001). *Online etymology dictionary*. Recuperado el 25 de enero de 2011 desde etymonline.com: http://www.etymonline.com/index.php?term=video

Field, S. (1994). *Screenplay - The Foundations of Screenwriting* (3rd Edition ed.). (D. Publishing, Ed.)

History of Reality TV. (2012). Recuperado el 22 de marzo de 2012 desde www.reality-tv-online.com: http://www.reality-tv-online.com/articles/history-reality-tv.html

Huesca, G. (2011). *Arquitecturas de herramientas de autoría para ambientes edutainment*. Mexico.

IGN. (2010). *Fallout 3 Review*. Recuperado el 8 de febrero de 2011 desde ps3.ign.com: http://ps3.ign.com/articles/924/924345p1.html

IGN. (2010). *Heavy Rain: The Origami Killer*. Recuperado el 8 de febrero de 2011 pc.ign.com: http://pc.ign.com/objects/811/811232.html

Puentedura, R. (2005). Playing Games in Education - or Thank you Mario...But Our Princess is In Another University! *NMC Summer Conference.*

RENa. (2008). *Narrativa*. Recuperado el 12 de diciembre de 2011 desde www.rena.edu.ve: http://www.rena.edu.ve/TerceraEtapa/literatura/narrativa.html

Salen, K., & Zimmerman, E. (2003). Rules of Play: Game Design Fundamentals. *The MIT Press* .

Sheldon, L. (2004). *Character Development and Storytelling for Games.* (T. C. PTR, Ed.)

University of Virginia. (2000). *Pop Goes the Page: Movable and Mechanical Books from the Brenda Forman Collection*. Recuperado el 10 de agosto de 2011 lib.virginia.edu: http://www2.lib.virginia.edu/exhibits/popup/theme.html

Wikipedia, the free encyclopedia. (2012). *TNT (TV Channel)*. Recuperado el 12 de marzo de 2012 desde wikipedia.org: http://en.wikipedia.org/wiki/TNT_(TV_channel)

9. Anexos

9.1. Guía Rápida de Usuario

Para utilizar el Sandtrace Framework en la personalización es necesario:

1. Incluir las cinco clases que comprenden al Framework en el proyecto de desarrollo del videojuego (copiarlas en la carpeta donde se encuentren el resto de los archivos generados por el desarrollador)
2. Mandar llamar, en algún lugar del código del desarrollador pero justo al inicio del videojuego (la pantalla de inicio del juego puede o no ser tomada en cuenta como un nivel; esto queda a juicio del desarrollador. Sin embargo, si se toma en cuenta, se sugiere que le sea asignado el nivel cero "0"), al método getParamsFromXml de la clase estática y pública getParams, pasándole como parámetro un string que contenga la ruta (local o remota) del archivo de personalización, es decir:

```
string url = http://movil.ccm.itesm.mx/pecustomization/PlayerPrefsXML.xml;
getParams.getParamsFromXML(url);
```

3. Para cada evento del videojuego, lanzar el evento correspondiente en el Sandtrace Framework
4. Si se desea, solicitar la siguiente mejor escena, dependiendo del valor instantáneo de salida de la escena actual, cada vez que se lance un evento. De los puntos 3 y 4 de esta lista tendríamos:

```
//Evento del videojuego actionPerformed
Scene s;
string nextScene;
actionPerformed(){
```

```
        Game.triggerEvent(); //Lanzar el evento
        if(Game.getCurrentSceneCount() CONDICION){
                s =  Game.nextScene(); //Obtener la siguiente escena
                nextScene = Game.nextSceneName();    //O el nombre
        }
}
```

5. Una vez alcanzado el final de una escena, o el punto en que es posible pasar a la siguiente escena, dependiendo de lo que el desarrollador desee, solicitar el nombre o el objeto de la siguiente escena, para posteriormente cargarla:

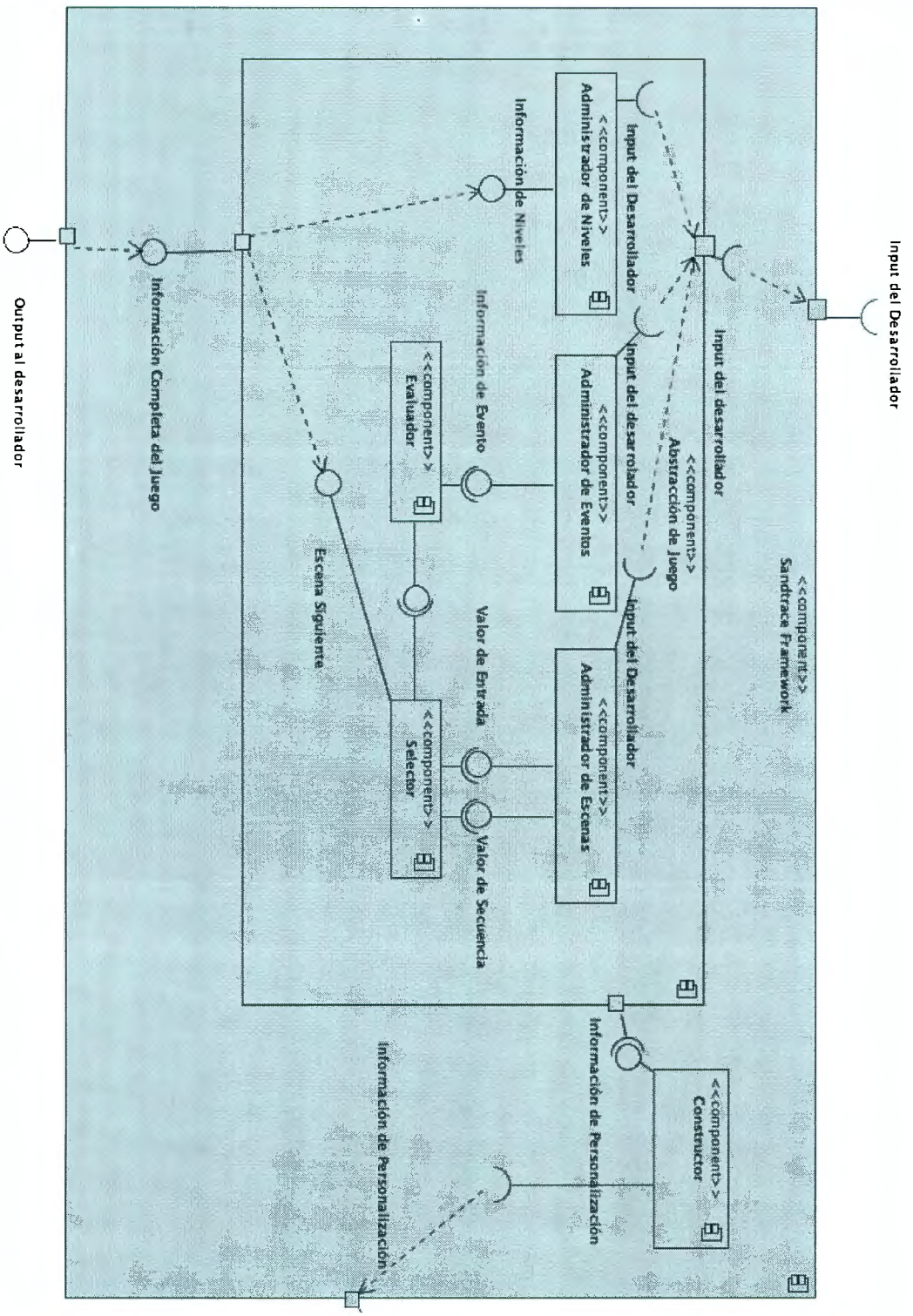
```
Scene s;
string nextScene;
{. . .}
s = Game.nextScene();            //obtener escena siguiente
nextScene = Game.nextSceneName();
Application.Loadlevel(nextScene); //Cargarla (Unity)
```

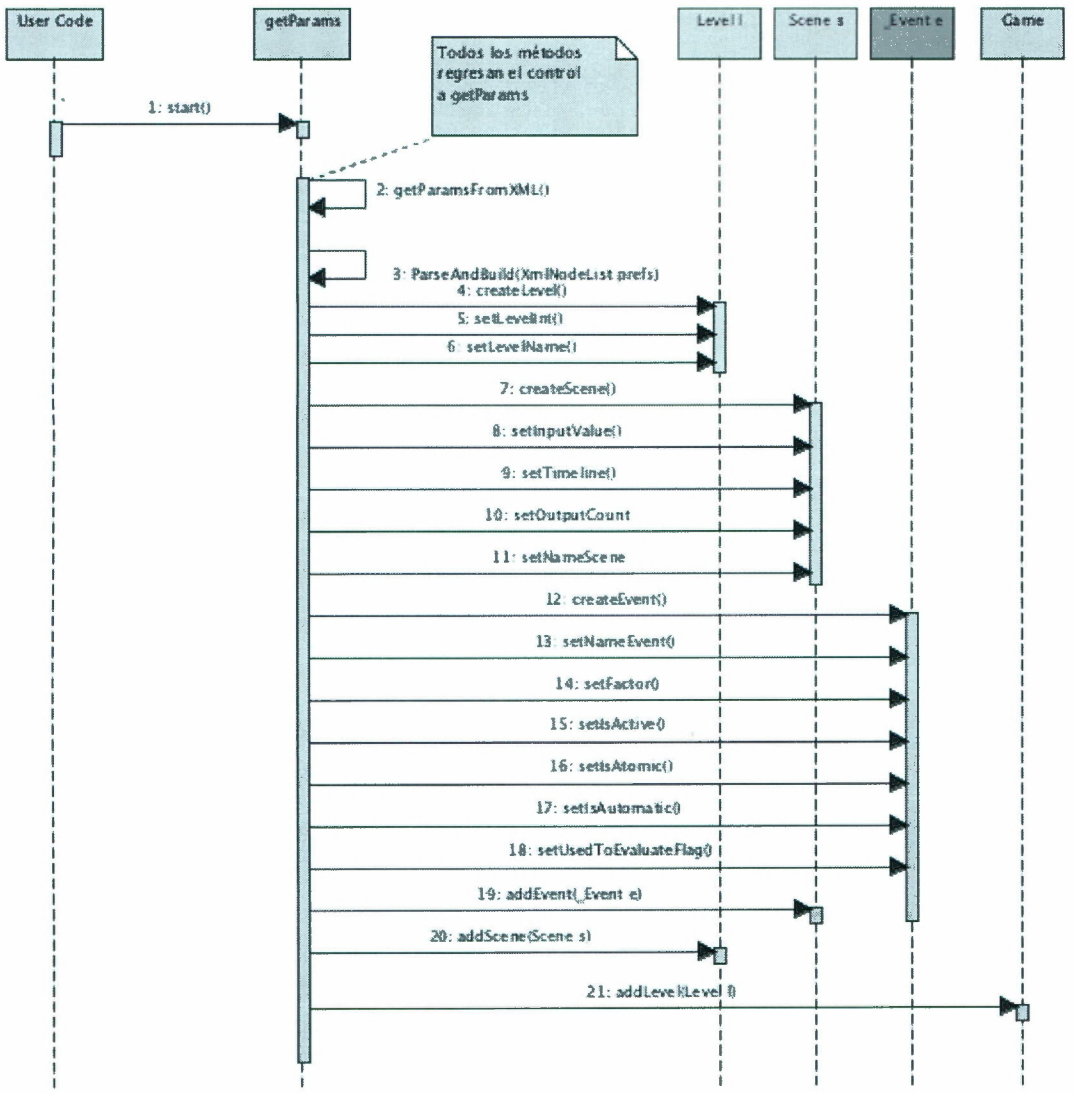
Algunos detalles importantes:

1. Es responsabilidad del desarrollador empalmar los eventos personalizados con los eventos del juego, es decir, cada vez que un evento personalizable tenga lugar en el juego, lanzar el evento correspondiente en el sistema aquí detallado. El Sandtrace Framework ofrece, precisamente, un marco de trabajo pero no limita ni obliga a quien lo use, a apegarse a él.
2. Todos los eventos de la clase _Event del Sandrace Framework, poseen un atributo llamado isActive. Este atributo no es manipulado ni empleado en otras clases del sistema sino que ha sido incluido para el uso del desarrollador. Es responsabilidad del mismo determinar cómo verificar si un evento se encuentra o no activo dentro de una escena y qué acciones tomar al respecto sobre el juego (en esencia, cómo implementar en el juego el que un tercero active o desactive eventos dentro de una escena).

3. Es tarea del desarrollador decidir (y realizar las adecuaciones pertinentes) si el usuario (jugador) podrá o no modificar, desde el juego, la URL del archivo de personalización.

9.2. Diagrama de componentes





9.4. Clase Game

```
using System;
using System.Diagnostics;
using System.Collections;
using System.Collections.Generic;

public static class Game {

        private static string GameName;

        public static Dictionary <string,Level> Levels
                = new Dictionary<string, Level>();



                                        //Properties
        public static string gameName{
```

```
            get{
                return GameName;
            }
            set{
                GameName = value;
            }
        }

        public static void addLevel(string key, Level l){
        //If you don't want to use dictionary methods.
            Levels.Add(key, l);
        }

        public static Level returnLevel(string levelKey){
        //Get level without accessing dictionary.
            Level l;
            Levels.TryGetValue(levelKey, out l);
            return l;
        }

        public static Scene returnSceneInLevel(string sceneKey, string levelKey){      //Get scene without creating
scene, nor level objects.
            Scene s;
            Level l;
            l = returnLevel(levelKey);
            s = l.returnScene(sceneKey);
            return s;
        }

                                                                                        //Get event without creating event,
scene or level objects.
        public static _Event returnEventInSceneInLevel(string eventKey, string sceneKey, string levelKey){
            _Event e;
            Scene s;
            s = returnSceneInLevel(sceneKey, levelKey);
            e = s.returnEvent(eventKey);
            return e;
        }

        /* * * * * * * * * * * * * * * *
         * SANDTRACE'S CRITICAL METHODS *
         * * * * * * * * * * * * * * * */

        public static void triggerEvent(string _event, string scene, string level){

            Level l;
            Scene s;
            _Event e;

            Game.Levels.TryGetValue(level, out l);
            l.Scenes.TryGetValue(scene, out s);
            s.Events.TryGetValue(_event, out e);
            if(!e.isAutomatic){
                if(!e.usedToEvaluateFlag)
                s.outputCount = s.outputCount + e.factor;

                if(e.isAtomic){
                    e.usedToEvaluateFlag = 1;
                }
            }
        }
```

```
public static void triggerEvent(_Event _event, Scene scene){
        if(!_event.isAutomatic){
                if(!_event.usedToEvaluateFlag)
                scene.outputCount = scene.outputCount + _event.factor;

                if(_event.isAtomic){
                        _event.usedToEvaluateFlag = 1;
                }
        }
}

public static void triggerAutomaticEvent(string _event, string scene, string level){

        Level l;
        Scene s;
        _Event e;

        Game.Levels.TryGetValue(level, out l);
        l.Scenes.TryGetValue(scene, out s);
        s.Events.TryGetValue(_event, out e);
        if(e.isAutomatic){
                if(!e.usedToEvaluateFlag)
                        s.outputCount = s.outputCount + e.factor;

                if(e.isAtomic){
                        e.usedToEvaluateFlag = 1;
                }
        }
}

public static void triggerAutomaticEvent(_Event _event, Scene scene){
        if(_event.isAutomatic){
                if(!_event.usedToEvaluateFlag)
                        scene.outputCount = scene.outputCount + _event.factor;

                if(_event.isAtomic){
                        _event.usedToEvaluateFlag = 1;
                }
        }
}

public static Scene nextScene(string currentSceneKey, string currentLevelKey, string nextLevelKey){
        Level nextLevel;
        Level currentLevel;
        Scene currentScene;
        Scene _nextScene;

        Game.Levels.TryGetValue(currentLevelKey, out currentLevel);
                //Get current level object from key
        Game.Levels.TryGetValue(nextLevelKey, out nextLevel);
                //Get next level object from key

        currentLevel.Scenes.TryGetValue(currentSceneKey, out currentScene);
        //Get current scene object from key

        _nextScene = nextScene(currentScene, nextLevel);
                        //Calls object-oriented nextScene method.

        return _nextScene;
                                                        //returns name
}
```

```
public static Scene nextScene(Scene currentScene, Level nextLevel){
//Gets objects as attributes, returns nextScene Object
        uint actualTimeline = currentScene.timeline;
                //Get current scene timeline value
        if(nextLevel.levelInt-actualTimeline ==1){
                        //Checks if nextLevel is ACTUALLY next level.
            int dif = 101;
                                                        //Set dif in 101
(considering max = 100)
            int lastDif = 101;
                                                        //For each loop keeps the
smalles difference so far
            int definitiveDif=101;
                                                        //Keeps the definitive dif for
debugging purposes.
            Scene nextClosestScene = Scene.createScene();
            foreach(KeyValuePair <string, Scene> pair in nextLevel.Scenes){
            //Foreach scene in next level...
                Scene s = pair.Value;
                                                        //get scene
                dif = s.inputValue-currentScene.outputCount;
                        //substract input value from output final value

                        if(Math.Abs(dif)<Math.Abs(lastDif)){
                                //If its smaller than the last one

                                lastDif = dif;
                                                        //Keep it
                                definitiveDif = Math.Abs(dif);

                                nextClosestScene = s;
                                                        //Save this scene
                        }
            }
            Debug.WriteLine("Diferencia definitiva: "+ definitiveDif);
            Debug.WriteLine("Next Scene: "+ nextClosestScene.nameScene);
            return nextClosestScene;
                                                //Return scene with smallest difference
        }
        else{
        Debug.WriteLine("Verify next level is ACTUALLY next level");
        Scene empty = Scene.createScene();

        return empty;
                                                //Return empty scene
        }
}

public static string nextSceneName(string currentSceneKey, string currentLevelKey, string nextLevelKey){
        Level nextLevel;
        Level currentLevel;
        Scene currentScene;
        Scene _nextScene;

        Game.Levels.TryGetValue(currentLevelKey, out currentLevel);
                //Get current level object from key
        Game.Levels.TryGetValue(nextLevelKey, out nextLevel);
                //Get next level object from key

        currentLevel.Scenes.TryGetValue(currentSceneKey, out currentScene);
        //Get current scene object from key
```

```
            _nextScene = nextScene(currentScene, nextLevel);
                        //Calls object-oriented nextScene method.

            string nextSceneName = _nextScene.nameScene;
                        //Gets name from returned object
            return nextSceneName;
                                                                    //returns name
        }


        public static string nextSceneName(Scene currentScene, Level nextLevel){
        //Very similar here...just that it returns a STRING
            uint actualTimeline = currentScene.timeline;

            if(nextLevel.levelInt-actualTimeline ==1){
                                    //Checks if nextLevel is ACTUALLY nextlevel
                int dif = 101;
                int lastDif = 101;
                int definitiveDif=101;
                Scene nextClosestScene = Scene.createScene();;
                foreach(KeyValuePair <string, Scene> pair in nextLevel.Scenes){
                    Scene s = pair.Value;
                    dif = s.inputValue-currentScene.outputCount;

                                if(Math.Abs(dif)<Math.Abs(lastDif)){


                                    lastDif = dif;

                                    definitiveDif = Math.Abs(dif);

                                    nextClosestScene = s;
                                }
                }
                Debug.WriteLine("Diferencia definitiva: "+ definitiveDif);
                Debug.WriteLine("Next Scene: "+ nextClosestScene.nameScene);
                return nextClosestScene.nameScene;
                                                    //Return string with next scene's name
            }else{
            Debug.WriteLine("Verify next level is ACTUALLY next level");
            Scene empty = Scene.createScene();
            empty.nameScene = "EmptyScene";
            return empty.nameScene;
                                                                    //Return empty
scene
            }
        }

        public static int getCurrentSceneCount(Scene currentScene){

            return currentScene.outputCount;

        }
}
```

9.5. Clase Level

```
using System;
using System.Collections;
using System.Collections.Generic;
```

```
public class Level {

        private string LevelName;
        private int LevelInt;
        public Dictionary<string,Scene> Scenes = new Dictionary<string,Scene>();

                public static Level createLevel(){
                Level l = new Level();
                return l;
        }

        public void addScene(string key, Scene s){
        //If you dont want to use dictionary methods.
                this.Scenes.Add(key,s);
        }

        public Scene returnScene(string key){
        //Get scenes without creating level objects.
                Scene sout;
                this.Scenes.TryGetValue(key, out sout);
                return sout;
        }

        public _Event returnEventFromScene(string eventKey, string sceneKey){        //Get EVENTS without
creating level, nor scene objects.
                Scene s = returnScene(sceneKey);
                _Event e = s.returnEvent(eventKey);
                return e;
        }


        public string levelName{
                get{
                        return LevelName;
                }
                set{
                        LevelName = value;
                }
        }

        public int levelInt{
                get{
                        return LevelInt;
                }
                set{
                        LevelInt = value;
                }
        }
}
```

9.6. Clase Scene

```
using System;
using System.Collections;
using System.Collections.Generic;

public class Scene {

        private string NameScene;
        private uint Timeline;
```

```
        private int InputValue;
        private int OutputCount;

        public Dictionary<string,_Event> Events =
        //Public Dictionary (For now, dont want to get in trouble).
                                        new Dictionary<string,_Event>();

                                                                //Class attributes properties.
        public static Scene createScene(){
                Scene s = new Scene();
                return s;
        }

        public void addEvent(string key, _Event e){
        //If you dont want to use dictionary methods.
                this.Events.Add(key,e);
        }

        public _Event returnEvent(string key){
        //Get events without creating scene objects.
                _Event eout;
                this.Events.TryGetValue(key, out eout);
                return eout;
        }

        public string nameScene{
                get{
                        return NameScene;
                }
                set{
                        NameScene = value;
                }
        }

        public uint timeline{
                get{
                        return Timeline;
                }
                set{
                        Timeline = value;
                }
        }

        public int inputValue{
                get{
                        return InputValue;
                }
                set{
                        InputValue = value;
                }
        }

        public int outputCount{
                get{
                        return OutputCount;
                }
                set{
                        OutputCount = value;
                }
        }

}
```

9.7. Clase _Event

```
using System;
using System.Collections;

public class _Event {


                        //Class Attributes
        private string NameEvent;
        private int Factor;
        private bool IsAtomic;
        private bool IsAutomatic;
        private bool IsActive;
        private bool UsedToEvaluateFlag;

        public static _Event createEvent(){
                _Event e = new _Event();
                return e;
        }


        public string nameEvent{                                        //Getters
and Setters
                get{
                        return NameEvent;
                }
                set{
                        NameEvent = value;
                }
        }
        public int factor{
                get{
                        return Factor;
                }
                set{
                        Factor = value;
                }
        }

        public bool isAtomic{
                get{
                        return IsAtomic;
                }
                set{
                        IsAtomic = value;
                }
        }
        public bool isAutomatic{
                get{
                        return IsAutomatic;
                }
                set{
                        IsAutomatic = value;
                }
        }
        public bool isActive{
                get{
                        return IsActive;
                }
```

```
            set{
                IsActive = value;
            }
        }
            public bool usedToEvaluateFlag{
            get{
                return UsedToEvaluateFlag;
            }
            set{
                UsedToEvaluateFlag = value;
            }
        }

}
```

9.8. Clase getParams

```
#region Namespaces
using System.Collections;
using System.Xml;
using System;
#endregion

public static class getParams{

        #region GlobalVariables

        public static XmlNodeList scenesList;

        public static XmlNodeList levelsList;
        public static XmlNodeList eventsList;

        public static XmlNodeList levelInt;
        public static XmlNodeList levelName;
        public static XmlNodeList timeline;

        public static XmlNodeList nameScene;

        public static XmlNodeList inputValue;

        public static XmlNodeList events;

        public static XmlNodeList eventName;

        public static XmlNodeList eventIsActive;

        public static XmlNodeList eventFactor;

        public static XmlNodeList eventIsAutomatic;
        public static XmlNodeList eventIsAtomic;

        #endregion
        // Use this for initialization
        public static void getParamsFromXML (string url) {


            XmlDocument CustomizationXML = new XmlDocument();
            CustomizationXML.Load(url);
```

```
            ParseAndBuild(CustomizationXML.SelectNodes("prefs"));
        }

        public static void ParseAndBuild(XmlNodeList prefs){

            levelsList = ((XmlElement)prefs[0]).GetElementsByTagName("level");

            #region Levels
            foreach (XmlElement nodo0 in levelsList){

                                                                    //Node list is received
                levelInt = nodo0.GetElementsByTagName("levelInt");
                levelName = nodo0.GetElementsByTagName("levelName");

                Level _level = Level.createLevel();
                _level.levelInt = (int)Convert.ToInt16(levelInt[0].InnerText);
                _level.levelName = levelName[0].InnerText;

                scenesList = nodo0.SelectNodes("scene");

                //Debug.Log("Elementos con tag scene: "+ scenesList.Count);

                #region Scenes

                foreach (XmlElement nodo in scenesList){

                                                                //First level child nodes
                    timeline = nodo.GetElementsByTagName("timeline");
            //Save timeline value
                    nameScene = nodo.GetElementsByTagName("name");
                //Save name
                    inputValue = nodo.GetElementsByTagName("inputValue");
            //Save input value
                    events = nodo.SelectNodes("events");
                //Open up events nodes
                                                        string NameScene =
nameScene[0].InnerText;
                    uint Timeline = (uint)Convert.ToInt16(timeline[0].InnerText);
                    int InputValue = (int)Convert.ToInt16(inputValue[0].InnerText);
                    int OutputCount = 0;

                    Scene _scene = Scene.createScene();

                    _scene.inputValue = InputValue;
                                    //Create This Scene.
                    _scene.timeline = Timeline;
                    _scene.outputCount = OutputCount;
                    _scene.nameScene = NameScene;

                                                                //Select events
                    #region Events


                            foreach(XmlElement nodo2 in events){

                                    if(nodo2.HasChildNodes){
                                        eventsList = ((XmlElement)
                                        events[0]).GetElementsByTagName("event");
    //Open up each event

                                                        foreach(XmlElement nodo3 in eventsList){
                //for every event:
```

```
                eventName = nodo3.GetElementsByTagName("name");
                                                    eventIsActive =
nodo3.GetElementsByTagName("isActive");
                                                    eventFactor =
nodo3.GetElementsByTagName("factor");
                                                    eventIsAtomic =
nodo3.GetElementsByTagName("isAtomic");
                                                    eventIsAutomatic =
nodo3.GetElementsByTagName("isAutomatic");

                                                    string EventName = eventName[0].InnerText;
                                                    int EventFactor =
(int)Convert.ToInt16(eventFactor[0].InnerText);
                                                    bool EventIsActive =
Convert.ToBoolean(Convert.ToInt16(eventIsActive[0].InnerText));
                                                    bool EventIsAtomic =
Convert.ToBoolean(Convert.ToInt16(eventIsAtomic[0].InnerText));
                                                    bool EventIsAutomatic =
Convert.ToBoolean(Convert.ToInt16(eventIsAutomatic[0].InnerText));


                                                    _Event _event = _Event.createEvent();

                                                    _event.nameEvent = EventName;
                                                    _event.factor = EventFactor;
                                                    _event.isActive = EventIsActive;
                                                    _event.isAtomic = EventIsAtomic;
                                                    _event.isAutomatic = EventIsAutomatic;
                                                    _event.usedToEvaluateFlag = false;
                                        //NONE has been yet used to evaluate

                                                    _scene.Events.Add(_event.nameEvent,_event);

                                                    }
                                                }
                                            }
                                        #endregion
                                        _level.Scenes.Add(_scene.nameScene,_scene);
                                    }
                                    #endregion
                                    Game.Levels.Add(_level.levelName,_level);
                                }
                                #endregion
                            }
                        }
```

9.9. XML A

```
<prefs>
        <level>
                <levelInt>0</levelInt>
                <levelName>zero</levelName>
                <scene>
                        <timeline>0</timeline>
                        <name>StartUpScreen</name>
                        <inputValue>0</inputValue>
                </scene>
        </level>
        <level>
                <levelInt>1</levelInt>
```

```
        <levelName>one</levelName>
        <scene>
                <timeline>1</timeline>
                <name>ApartmentDetective</name>
                <inputValue>0</inputValue>
                <events>
                        <event>
                                <name>ReadMail</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>0</isAutomatic>
                                <factor>1</factor>
                        </event>

                        <event>
                                <name>TakeMap</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>0</isAutomatic>
                                <factor>1</factor>
                        </event>

                        <event>
                                <name>LookPaint</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>0</isAutomatic>
                                <factor>1</factor>
                        </event>
                </events>
        </scene>
</level>

<level>
        <levelInt>2</levelInt>
        <levelName>two</levelName>
        <scene>
                <timeline>2</timeline>
                <name>City</name>
                <inputValue>3</inputValue>

                <events>
                        <event>
                                <name>StealMoney1</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>0</isAutomatic>
                                <factor>-1</factor>
                        </event>

                        <event>
                                <name>StealMoney2</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>0</isAutomatic>
                                <factor>-1</factor>
                        </event>

                        <event>
                                <name>StealMoney3</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
```

```
                                        <isAutomatic>0</isAutomatic>
                                        <factor>-1</factor>
                                    </event>

                                    <event>
                                        <name>StealMoney4</name>
                                        <isActive>1</isActive>
                                        <isAtomic>1</isAtomic>
                                        <isAutomatic>0</isAutomatic>
                                        <factor>-1</factor>
                                    </event>

                                    <event>
                                        <name>BackDoor</name>
                                        <isActive>1</isActive>
                                        <isAtomic>1</isAtomic>
                                        <isAutomatic>0</isAutomatic>
                                        <factor>-2</factor>
                                    </event>

                                    <event>
                                        <name>RegistryOrder</name>
                                        <isActive>1</isActive>
                                        <isAtomic>1</isAtomic>
                                        <isAutomatic>0</isAutomatic>
                                        <factor>2</factor>
                                    </event>
                                </events>

                            </scene>

                            <scene>
                                <timeline>2</timeline>
                                <name>Encom</name>
                                <inputValue>2</inputValue>
                                <events>

                                </events>
                            </scene>

                            <scene>
                                <timeline>2</timeline>
                                <name>TheGrid</name>
                                <inputValue>1</inputValue>

                                <events>

                                </events>
                            </scene>
                        </level>
                    </prefs>
```

9.10. XML B

```
<prefs>
                    <level>
                        <levelInt>0</levelInt>
                        <levelName>zero</levelName>
                        <scene>
                            <timeline>0</timeline>
                            <name>StartUpScreen</name>
```

```
                <inputValue>0</inputValue>
        </scene>
</level>
<level>
        <levelInt>1</levelInt>
        <levelName>one</levelName>
        <scene>
                <timeline>1</timeline>
                <name>ApartmentDetective</name>
                <inputValue>0</inputValue>
                <events>
                        <event>
                                <name>ReadMail</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>1</isAutomatic>
                                <factor>1</factor>
                        </event>

                        <event>
                                <name>TakeMap</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>1</isAutomatic>
                                <factor>1</factor>
                        </event>

                        <event>
                                <name>LookPaint</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>1</isAutomatic>
                                <factor>1</factor>
                        </event>
                </events>
        </scene>
</level>

<level>
        <levelInt>2</levelInt>
        <levelName>two</levelName>
        <scene>
                <timeline>2</timeline>
                <name>City</name>
                <inputValue>3</inputValue>

                <events>
                        <event>
                                <name>StealMoney1</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>0</isAutomatic>
                                <factor>-1</factor>
                        </event>

                        <event>
                                <name>StealMoney2</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>0</isAutomatic>
                                <factor>-1</factor>
                        </event>
```

```
                    <event>
                        <name>StealMoney3</name>
                        <isActive>1</isActive>
                        <isAtomic>1</isAtomic>
                        <isAutomatic>0</isAutomatic>
                        <factor>-1</factor>
                    </event>

                    <event>
                        <name>StealMoney4</name>
                        <isActive>1</isActive>
                        <isAtomic>1</isAtomic>
                        <isAutomatic>0</isAutomatic>
                        <factor>-1</factor>
                    </event>

                    <event>
                        <name>BackDoor</name>
                        <isActive>1</isActive>
                        <isAtomic>1</isAtomic>
                        <isAutomatic>0</isAutomatic>
                        <factor>-2</factor>
                    </event>

                    <event>
                        <name>RegistryOrder</name>
                        <isActive>1</isActive>
                        <isAtomic>1</isAtomic>
                        <isAutomatic>0</isAutomatic>
                        <factor>2</factor>
                    </event>
                </events>

            </scene>

            <scene>
                <timeline>2</timeline>
                <name>Encom</name>
                <inputValue>2</inputValue>
                <events>

                </events>
            </scene>

            <scene>
                <timeline>2</timeline>
                <name>TheGrid</name>
                <inputValue>1</inputValue>

                <events>

                </events>
            </scene>
        </level>
</prefs>
```

9.11. XML C

```
<prefs>
        <level>
```

```
        <levelInt>0</levelInt>
        <levelName>zero</levelName>
        <scene>
                <timeline>0</timeline>
                <name>StartUpScreen</name>
                <inputValue>0</inputValue>
        </scene>
</level>
<level>
        <levelInt>1</levelInt>
        <levelName>one</levelName>
        <scene>
                <timeline>1</timeline>
                <name>ApartmentDetective</name>
                <inputValue>0</inputValue>
                <events>
                        <event>
                                <name>ReadMail</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>0</isAutomatic>
                                <factor>3</factor>
                        </event>

                        <event>
                                <name>TakeMap</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>0</isAutomatic>
                                <factor>2</factor>
                        </event>

                        <event>
                                <name>LookPaint</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>0</isAutomatic>
                                <factor>1</factor>
                        </event>
                </events>
        </scene>
</level>

<level>
        <levelInt>2</levelInt>
        <levelName>two</levelName>
        <scene>
                <timeline>2</timeline>
                <name>City</name>
                <inputValue>3</inputValue>

                <events>
                        <event>
                                <name>StealMoney1</name>
                                <isActive>1</isActive>
                                <isAtomic>1</isAtomic>
                                <isAutomatic>0</isAutomatic>
                                <factor>-1</factor>
                        </event>

                        <event>
                                <name>StealMoney2</name>
```

```
                    <isActive>1</isActive>
                    <isAtomic>1</isAtomic>
                    <isAutomatic>0</isAutomatic>
                    <factor>-1</factor>
                </event>

                <event>
                    <name>StealMoney3</name>
                    <isActive>1</isActive>
                    <isAtomic>1</isAtomic>
                    <isAutomatic>0</isAutomatic>
                    <factor>-1</factor>
                </event>

                <event>
                    <name>StealMoney4</name>
                    <isActive>1</isActive>
                    <isAtomic>1</isAtomic>
                    <isAutomatic>0</isAutomatic>
                    <factor>-1</factor>
                </event>

                <event>
                    <name>BackDoor</name>
                    <isActive>1</isActive>
                    <isAtomic>1</isAtomic>
                    <isAutomatic>0</isAutomatic>
                    <factor>-2</factor>
                </event>

                <event>
                    <name>RegistryOrder</name>
                    <isActive>1</isActive>
                    <isAtomic>1</isAtomic>
                    <isAutomatic>0</isAutomatic>
                    <factor>2</factor>
                </event>
            </events>

        </scene>

        <scene>
            <timeline>2</timeline>
            <name>Encom</name>
            <inputValue>2</inputValue>
            <events>

            </events>
        </scene>

        <scene>
            <timeline>2</timeline>
            <name>TheGrid</name>
            <inputValue>1</inputValue>

            <events>

            </events>
        </scene>
    </level>
</prefs>
```