Un Modelo Conceptual para el Soporte de Ecosistemas 2D basados en Simulación Física para Superficies

José Pascual Azorín Vicente

UNIVERSIDAD
POLITECNICA
DE VALENCIA

Trabajo de Fin de Máster

Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Director

Dr. Javier Jaén Martínez

Alejandro Catalá Bolós

Julio de 2012, Valencia

Agradecimientos

A mis padres y hermanas, por su apoyo.

A mis sobrinos Raúl y Nerea, por ser como son.

A Alejandro Catalá, por su paciencia infinita, sus consejos y su esfuerzo.

A Javier Jaén, por sus consejos y apoyo.

A todos los miembros del grupo de investigación ISSI, por su atención y colaboración.

INDICE GENERAL

INDICE DE FIGURAS

Capitulo 1. Introducción

Este capítulo se centra exclusivamente en la presentación de las principales motivaciones detrás de este trabajo de fin de máster y de sus objetivos directores. De esta forma, en primer lugar se describe cuál es la motivación de este trabajo. En segundo lugar se presentan los objetivos del trabajo. En tercer y último lugar, se presenta brevemente cada uno de los capítulos restantes.

1.1 Motivación

La invención del ordenador permitió abordar de forma automatizada una serie de tareas tediosas de cómputo que, hasta entonces, las personas debían realizar de manera manual o semiautomatizada. El ámbito de aplicación de los ordenadores se expandió con relativa rapidez a otros dominios de la vida, pasando de ser utilizado exclusivamente en ambientes industriales a ser utilizado en los hogares. En este último ámbito, en muchas ocasiones el ordenador se ha utilizado más como una herramienta de entrenamiento o de actividades lúdicas que como una herramienta de trabajo personal. A medida que los ordenadores personales se hicieron más populares y asequibles, se comenzó a pensar en la forma en la que los ordenadores podrían ayudar a los seres humanos en el aprendizaje, en su enriquecimiento personal, y en el desarrollo de diversas habilidades por medio de la interacción con las máquinas, adoptando por lo general una visión lúdica de las actividades a realizar por sus usuarios con el objetivo de que resultasen más atractivas y motivadoras. Entre muchas de las habilidades que resultan de interés se encuentra la creatividad, el pensamiento computacional y las técnicas de resolución de problemas. La creatividad consiste en la generación de nuevas ideas o conceptos, o de nuevas asociaciones entre ideas y conceptos conocidos, que habitualmente producen soluciones originales a problemas ya existentes o nuevos produciendo así nuevas oportunidades para el individuo y para la sociedad por medio de la innovación [Ale00]. El *pensamiento computacional* consiste en proponer métodos de resolución de problemas que pueden usarse para resolver algorítmicamente problemas complicados y que se usa a menudo para realizar grandes mejoras en la eficiencia de los procesos [Wing06]. En esencia consiste en entender en términos computacionales los procesos informalmente establecidos que nos rodean. De esa forma, se formalizan y se estructuran dichos procesos para mejorar su comprensión y eficiencia. El desarrollo de este pensamiento permite una mayor abstracción útil para los individuos en el desarrollo de sus actividades. Por último, las habilidades relacionadas con el “problem solving” o resolución de problemas [Dun45] se centran en mejorar la forma en que se abordan los detalles de los problemas para alcanzar una solución. La resolución de problemas puede incluir operaciones

matemáticas o sistemáticas y puede ser un indicador de las habilidades de pensamiento crítico de una persona.

Para mejorar el proceso de aprendizaje de estas habilidades, es de vital importancia tomar en consideración los postulados de las teorías de aprendizaje de corte constructivista, tales como el aprendizaje experiencial [Kol94] [Dew63] y el aprendizaje situado [Bro89] . En estas teorías "aprender es hacer" y se considera también la reflexión y la discusión como procesos necesarios de suma importancia más allá de la acción para aprender de forma efectiva. También el aprendizaje social es de interés, considerando que el conocimiento emerge de la interacción y la comunicación entre individuos [Vyg78]. Además, para mejorar el aprendizaje es importante incrementar los niveles de motivación de los participantes en los procesos de aprendizaje, por lo que este factor se considera transversal a la mayoría de las teorías y/o aproximaciones [Csi88].

Teniendo en cuenta todo lo anterior, se podría decir que para promover experiencias óptimas de aprendizaje debemos proveer un entorno que fomente la motivación, que permita la interacción entre pares, se centre en actividades que exijan la realización y construcción de artefactos y, finalmente, que por medio de procesos de discusión y refinamiento se puedan mejorar dichos artefactos. Una de las estrategias más efectivas que parecen admitir esos supuestos para el fomento del aprendizaje es el uso del juego tal y como ya planteaba Clark Abt en los años 60 [Abt70]. Estos juegos, todavía de índole no tecnológica, acuñados bajo el término general de "juegos serios" por el hecho de que no tratan sólo el divertimiento sino que trabajan principalmente el aprendizaje, permiten la participación activa, la implicación y el compromiso en las tareas, que se considera tan importante en las teorías constructivistas [Bru60][Pia67] y experienciales [Kol94] [Dew63] , así como la interacción social que permite la inclusión de todos los individuos persiguiendo objetivos comunes mejorando la experiencia de aprendizaje. Además, Abt va más allá, y considera no sólo el jugar a juegos serios como medio para aprender, sino que considera también la etapa de creación del juego como parte del propio juego, haciendo énfasis en la relevante importancia que esta etapa tiene en el aprendizaje en relación al resto de fases del juego.

Todo lo dicho anteriormente motiva que persigamos la provisión futura de un entorno de aprendizaje en el que se juegue a la creación de juegos, facilitando así habilidades mencionadas al principio de este capítulo tales como la creatividad, el pensamiento computacional, o las técnicas de resolución de problemas, aunando en cierta manera muchas de las ideas y aproximaciones al aprendizaje mencionadas anteriormente.

Entre los sistemas existentes relacionados con la problemática que pretendemos abordar podemos destacar algunos atendiendo a las capacidades constructivas que ofrecen:

1) Sistemas que permiten crear entornos de juegos a partir únicamente de elementos predefinidos, como por ejemplo StoryTelling Alice, AlgoBlock, IncreTable, The Augmented Knight's Castle, The Tangible Programming Space, Quetzal y Tern.
2) Sistemas que permiten crear entornos de juego mediante elementos creados por el usuario, como por ejemplo Agentsheets, Scratch, Topobo, LogoBlocks.
3) Sistemas que solo permiten jugar en un único entorno de juego con elementos predefinidos, como por ejemplo Cleogo, Turtan, Teaching Table, TViews Table RPG (TTRPG).
4) Sistemas que permiten crear otros sistemas: Raptor y STARS.

Estos mismos sistemas también podríamos disponerlos atendiendo a otro criterio de clasificación, tal y como puede ser el objetivo principal para el que fueron diseñados:

1) Sistemas orientados al aprendizaje.
    a. Sistemas para enseñar matemáticas básicas a niños: Teaching Table.
    b. Sistemas que ayudan a los niños a mejorar la comprensión del equilibrio, del centro de masas, la coordinación, el movimiento relativo y los múltiples grados de libertad: Topobo.
    c. Sistemas para enseñar lenguajes de programación: StoryTelling Alice, The Tangible Programming Space, AlgoBlock, LogoBlocks, Quetzal y Tern, Agentsheets, Scratch, Cleogo, Turtan.
2) Sistemas orientados al entretenimiento.
    a. Sistemas que permiten jugar a juegos: TViews Table RPG (TTRPG), IncreTable, The Augmented Knight's Castle.
    b. Sistemas que permiten realizar bocetos de juegos: Raptor.
    c. Sistemas que permiten desarrollar juegos avanzados: STARS.

Todos estos trabajos serán descritos en detalle en el siguiente capítulo, y esencialmente permiten de una u otra forma la creación de juegos así como jugar a los juegos creados. Como veremos, las tecnologías que soportan las actividades asociadas en cada uno de estos sistemas varían notablemente. Algunos de los sistemas se basan en PC y enfocan su interacción en el uso de teclado y ratón como periféricos de entrada. Otros adoptan cierta interfaz que soporta la interacción tangible mediante elementos y bloques físicos, y otros utilizan cierto tipo de interacción natural basado en pantallas táctiles. De todo el espectro de tecnologías existentes, consideramos especialmente interesantes las superficies interactivas.

Las superficies interactivas son pantallas que sirven no solo de visualización (*output*) sino que permiten la entrada de comandos por medio de los dedos o elementos tangibles específicamente diseñados para ello (*inputs*). Este soporte tecnológico puede ayudar a alcanzar un aprendizaje de carácter más social y activo de acuerdo a las teorías revisadas anteriormente ya que fomenta características como la colaboración, la cooperación y una comunicación cara a cara entre sus usuarios, al mismo tiempo

que incrementan el nivel de motivación de los usuarios [Hor08]. Es por ello, que la aplicación de las ideas expuestas por Abt sobre juegos serios por medio de las superficies interactivas parece apropiado. Más si cabe cuando no existe todavía un sistema que permita crear nuestros propios juegos en 2D y jugar a dichos juegos utilizando para ello una superficie interactiva. De acuerdo a sus ideas sobre juegos serios, así como las características de interacción sociales y colaborativas de las superficies interactivas, sería adecuado soportar tanto la creación como la ejecución de los juegos creados. Como un paso necesario previo a la construcción de una plataforma que soportara ese tipo de actividad creativa, es necesaria la creación de un modelo que nos permita soportar la creación y ejecución de nuestros propios juegos en 2D que incorporen ya determinado comportamiento físico sobre dicha superficie interactiva. En este aspecto conceptual esencial se va a centrar el presente trabajo fin de máster.

Dicho modelo debería cubrir los siguientes requisitos:

- Soportar la creación y ejecución de juegos caracterizados por simular físicamente entidades bidimensionales y permitir la reutilización de conceptos ya definidos.
- Definir y modificar tipos de entidades y poder crear varias entidades de un tipo de entidad previamente definido, de tal forma que soporte la especificación de propiedades y acciones sobre dichos tipos de entidades. Estos determinarán la estructura estática de las entidades.
- Poder definir los eventos que queremos detectar en el juego y poder definir atributos para dichos eventos. Estos eventos permitirán en un futuro gobernar la simulación del juego por medio de modelos de comportamiento como pueden ser las reglas.
- Disponer de varios tipos de datos predefinidos para asignarlos a las propiedades, atributos de eventos o parámetros formales.
- Poder definir escenarios de juego con las entidades que queramos que participen en el juego y poder definir reglas para cada escenario que nos permitan ejecutar dicho escenario.
- Dichas entidades deben tener una estructura visual y comportarse físicamente. Por tanto, es necesario soportar la definición de los “esqueletos” de las entidades mediantes componentes geométricos unidos por articulaciones y poder dotarle de apariencia visual.
- En los juegos creados los usuarios deben poder interactuar de forma simple con las entidades del juego sobre la superficie interactiva.

1.2 Objetivos

El objetivo principal de este trabajo es proponer y validar un modelo conceptual que soporte tanto la creación como la ejecución de juegos 2D con un comportamiento

físico y que sirva de soporte tecnológico adecuado para que en un futuro soporte aprendizaje como el descrito por Abt mediante el uso de superficies táctiles.

Este objetivo forma parte de un objetivo más general que hemos comentado anteriormente y que no abordamos en este trabajo fin de máster, y que consistiría en la construcción de un sistema que nos permita la creación y ejecución de nuestros propios juegos en 2D sobre una superficie táctil.

En nuestra propuesta, y tras el estudio del estado del arte, que presentaremos en el capítulo 2, propondremos un modelo que cumpla los requisitos anteriormente enunciados y nos permita crear y ejecutar juegos 2D en superficies interactivas. Éstas no son más que superficies multitáctiles que funcionan a la vez como pantallas de visualización bidimensional. El uso de este tipo de plataforma tecnológica de base permite fácilmente proveer una interacción y reflexión colaborativa, y utilizar objetos tangibles como medio de interacción.

1.3 Estructura del documento

Este trabajo se ha organizado en cinco capítulos. A continuación se describe brevemente el contenido de cada uno de los capítulos restantes:

•Capítulo 2. Estado del arte

El capitulo 2 presenta varios sistemas que están relacionados de alguna manera con la creación y ejecución de juegos. También se presenta una tabla comparativa de las características de los diferentes sistemas descritos.

•Capítulo 3. Modelo para la construcción de ecosistemas.

El capítulo 3 presenta el modelo conceptual que se ha propuesto para la creación de juegos 2D con un comportamiento físico y el modelo que se utiliza para llevar a cabo la ejecución de dichos juegos 2D. A continuación se presentan un par de casos de estudio, el Arkanoid y el Pong, que se han implementado utilizando el modelo propuesto anteriormente y que nos permitirán validar dicho modelo. Por último se comentan las limitaciones del modelo propuesto y las conclusiones que se han alcanzado al final de este capítulo.

•Capítulo 4. Conclusiones y trabajos futuros.

El capitulo 4 se comentan las conclusiones generales resultantes de la realización de este trabajo y posibles trabajos futuros relacionados con este trabajo.

•Capítulo 5. Anexos.

El capitulo 5 contiene extractos de código e imágenes de los ejemplos implementados.

Capitulo 2. Estado del Arte

En este capítulo presentamos una serie de trabajos relacionados con entornos que permiten la creación y ejecución de juegos. Clasificaremos los trabajos presentados según el grado de creación que nos permitan, es decir, si sólo nos permiten crear juegos con elementos predefinidos o nos permiten crear juegos con elementos creados por el usuario, o si no nos permiten crear juegos y sólo nos permiten jugar en un único entorno de juego con elementos predefinidos.

A continuación, pasamos a explicar cada una de las categorías en que hemos clasificado los trabajos, así como los trabajos que se encuentran dentro de estas categorías.

2.1 Sistemas de creación de juegos con elementos predefinidos

Los sistemas incluidos en esta categoría permiten crear entornos de juegos a partir únicamente de elementos predefinidos. En estos sistemas, los usuarios no pueden crear sus propias entidades, solo pueden seleccionar las entidades predefinidas del sistema que quieran incorporar al escenario y definir su comportamiento.

2.1.1 AlgoBlock

AlgoBlock [Suzu95] es una herramienta educativa en la que los usuarios utilizan bloques físicos que se conectan entre sí para programar el movimiento de un submarino virtual a través de un laberinto. Cada bloque físico representa una instrucción determinada que el submarino puede realizar (ir hacia adelante, ir a la derecha, ir a la izquierda, rotar, etc.) y ofrece el control del parámetro asociado a esta instrucción mediante diferentes botones o sensores integrados en la misma pieza. El resultado de la ejecución del programa se muestra en una pantalla CRT en forma de un submarino animado. AlgoBlock está dirigido al aprendizaje de lenguajes de programación y permite a los estudiantes mejorar sus habilidades de resolución de problemas mediante trabajos de programación colaborativa. Al trabajar con herramientas tangibles que pueden compartirse en un espacio colaborativo, AlgoBlock permite la interacción física y la colaboración.

2.1.2 The Tangible Programming Space

Es un sistema que permite a los niños la creación, edición y simulación de entornos virtuales de forma colaborativa [Fern06] . Los niños pueden crear colaborativamente mundos virtuales insertando objetos y asignando propiedades a los objetos del mundo virtual utilizando para ello objetos físicos.

La configuración física del sistema consta de una alfombra grande y blanca con una rejilla de etiquetas de posiciones identificables por debajo, un juego de cartas de

programación de material plástico, varios bloques de creación tangibles con un lector RFID que está conectado de forma inalámbrica al software del ordenador, y una pantalla que muestra el sistema que se está construyendo.

Los bloques de creación conectan el sistema físico con su representación virtual. Cuando los usuarios interactúan con el sistema, añaden objetos y comportamientos a la representación en pantalla colocando cartas encima de los bloques de creación. En la pantalla de visualización se muestra un rectángulo en rojo que nos indica la posición de los bloques creadores en la alfombra. La posición de los bloques de creación y el identificador de las cartas colocadas en ellos se comunican inalámbricamente al software que se ejecuta en el ordenador host.

Hay dos tipos de cartas de programación. Las cartas con imágenes se usan para colocar nuevos objetos en sitios específicos de la pantalla, mientras que las de comportamiento se usan para especificar la funcionalidad de los objetos que ya se han añadido. Para añadir una nueva imagen en un lugar especifico, la carta de imagen se coloca encima del bloque de creación. Los comportamientos se añaden a objetos existentes colocando primero el bloque de creación en una posición en la que hay un objeto y poniendo encima después una carta de comportamiento. Las cartas de comportamiento constan de una serie de comportamientos para el movimiento, las colisiones, la interacción del usuario, y para cambiar las propiedades de los objetos. Las construcciones hechas con el sistema se almacenarían solo en forma digital. Incluso aunque un objeto físico pueda usarse para almacenar lógicamente el código de una construcción especifica, solo puede accederse al código en si mismo mediante la pantalla del ordenador.

En un principio, uno de los problemas a los que se enfrentó este sistema fue que la parte física y la parte virtual se comportaran como “espejos” el uno del otro. Por ejemplo, se esperaría que fuera posible construir un bosque con muchos árboles, aunque solo tuviéramos una imagen de un árbol. Finalmente se optó por que las dos formas de representación (la física y la virtual) se complementaran la una a la otra mediantes los diferentes tipos de recursos e información que es posible proporcionar en cada uno de ellos.

2.1.3 StoryTelling Alice

StoryTelling Alice [Kelle06][Kelle07a][Kelle07b] es un lenguaje de programación creado por Caitlin Kelleher con el objetivo de hacer más atractiva la programación para chicas adolescentes (de entre 11 a 15 años). Este objetivo está motivado por las bajas tasas de mujeres estudiantes de ingeniería informática en EEUU. StoryTelling Alice está basado en el entorno de programación Alice que permite a los programadores noveles crear programas que controlan el movimiento de objetos en un mundo virtual 3D. StoryTelling Alice está pensado especialmente para que las chicas puedan contar sus propias historias mediante animaciones 3D, programando los escenarios y el

comportamiento de los personajes y objetos que aparecen en sus animaciones. La pantalla principal de StoryTelling Alice muestra el escenario 3D, una lista con los objetos que hay en el mundo construido, y las propiedades, métodos y funciones del objeto seleccionado. El editor de métodos, permite la especificación del comportamiento en base a la instrucción realizando las selecciones oportunas en las vistas anteriores. Alice proporciona dos formas de ayuda para que los usuarios aprendan a programar:

1) Alice usa un método de construcción de programas de arrastrar y soltar que impide que los usuarios cometan errores de sintaxis.
2) En Alice, los programas crean animaciones para que los usuarios puedan ver sus errores y corregirlos fácilmente.

Para crear un programa en Alice, los usuarios seleccionan de una galería de objetos 3D los objetos que van a usar en su programa. Todos los objetos realizan el mismo conjunto de animaciones básicas como moverse, girar y redimensionar. Las animaciones de Alice se presentan como fichas graficas que los usuarios pueden arrastrar y soltar en sus programas.

Alice permite a los usuarios crear sus propios procedimientos. Los métodos creados pueden tomar parámetros (números, strings, objetos, colores, etc.).

Se realizaron estudios [Kelle07b] para comparar el aprendizaje, comportamiento y actitud de chicas que comienzan a programar con StoryTelling Alice y con Alice (Generic Alice) mediante una serie de talleres de cuatro horas. En el estudio participaron 88 chicas scout, 45 se asignaron a un grupo de control que usó Generic Alice y 43 se asignaron a un grupo experimental que usó Storytelling Alice. La media de edad de las participantes era de 12,6 años y todas las participantes excepto cuatro estaban entre los cursos 5º y 9º. 76 participantes asistían a la escuela pública y 12 a la privada. Los participantes tuvieron dos horas y quince minutos para completar el tutorial y crear un programa con la versión de Alice que se le había asignado. Después, los participantes realizaron un cuestionario de programación y completaron un examen. A continuación, los participantes tuvieron 30 minutos para probar la otra versión de Alice. Al final del taller se pidió a los participantes que eligieran uno de los dos para llevarse a casa. Finalmente, se les pidió a los participantes que seleccionaran uno de los programas que crearon para compartirlo con los otros. Los resultados del estudio sugirieron que los participantes que usaron Generic Alice y Storytelling Alice tuvieron el mismo éxito a la hora de aprender conceptos de programación. Sin embargo, los participantes que usaron Storytelling Alice mostraron mayores pruebas de compromiso con la programación; pasaron un mayor porcentaje de tiempo programando, y fue más probable que usaran tiempo extra para continuar programando, y expresaron una mayor interés en el uso futuro de Alice que los

participantes que usaron Generic Alice. Los usuarios encontraron a los dos igual de entretenidos.

2.1.4 Quetzal y Tern

Quetzal y Tern [Horn07] son dos lenguajes de programación tangibles que utilizan objetos físicos sin partes electrónicas ni suministro de energía para representar distintos elementos de programación, órdenes y estructuras de control de flujo con las que poder construir un programa mediante la conexión entre sí de dichos bloques físicos. Estos lenguajes se diseñaron para enseñar a programar en clase a niños entre los últimos años de educación primaria y secundaria. Las principales ventajas de estos lenguajes son que utilizan elementos durables y de bajo coste, que permiten trabajar en ambientes sin conexión, y que fomentan la colaboración entre los niños.

El lenguaje de programación Quetzal permite controlar robots de LEGO Mindstorms. Consta de fichas de plástico interconectadas que representan estructuras de control de flujo, acciones y parámetros. Las sentencias del lenguaje se conectan juntas para formar cadenas de flujo de control. Algunas sentencias también aceptan parámetros que pueden incluir constantes o lecturas de sensores. Las fichas de parámetros son fichas de plástico con formas específicas que representan su tipo de datos. Éstas pueden insertarse en slots en la cara superior de las sentencias.

El lenguaje de programación Tern permite controlar robots virtuales en la pantalla del ordenador. En Tern, los programadores conectan bloques de madera con forma de piezas de puzle para formar cadenas de flujo de control. Estos programas controlan robots virtuales simples en un mundo de cuadricula en la pantalla del ordenador. Los múltiples robots pueden interactuar en el mismo mundo, y equipos de estudiantes pueden colaborar para resolver problemas como recoger objetos o navegar a través de un laberinto. El profesor puede proyectar el mundo de cuadrícula en una pared de la clase para que todos los estudiantes puedan participar en una actividad compartida.

Del mismo modo que en Quetzal, en Tern los programas pueden incluir bucles, bifurcaciones y valores de parámetros. El lenguaje Tern también incluye la capacidad de crear subrutinas llamadas skills. Las subrutinas se definen usando un bloque especial Start Skill y pueden invocarse en cualquier parte de la cadena de control de flujo. En Tern las sentencias JUMP y LAND permiten introducir bucles en el flujo de control del programa. Estas sentencias se conectan con un cable enrollado que representa el flujo de ejecución a medida que se mueve de la sentencia JUMP a la sentencia LAND.

Los lenguajes se compilan usando una estación portable de escaneo y mediante tecnología fiable de visión por computador se obtiene una imagen, con la que el compilador convierte un programa directamente en código de maquina virtual (en el caso de Tern) o en un lenguaje intermedio basado en texto como NQC (en el caso de

Quetzal) (Not Quite C (NQC) es un lenguaje de programación basado principalmente en el lenguaje C pero con limitaciones especificas, como el número máximo de subrutinas y variables permitidas). Cuando ocurre un error de sintaxis, el compilador muestra una imagen del programa original, un mensaje de error y una flecha que muestra la posición del problema. En Tern no hay errores de sintaxis del lenguaje.

2.1.5 IncreTable

Incretable [Leit08] es un juego para una mesa táctil inspirado en el juego The Incredible Machine. En este juego, los usuarios pueden combinar piezas del mundo real y virtual para resolver puzles en el juego. Las acciones del juego incluyen colocar fichas de dominó con lápices digitales, controlar un coche virtual modificando el terreno virtual o controlar robots reales para que caigan sobre dominós reales o virtuales.

Incretable proporciona interacción multimodal basada en un visualizador de proyección multimodal, lápices digitales, una cámara detectora de la profundidad y objetos físicos y robots personalizados. Los objetos del mundo real provocan la participación activa del usuario en la creación de contenido en el que se difuminan los límites entre el mundo real y virtual.

El objetivo general de IncreTable es ordenar una colección dada de objetos tanto virtuales como reales de un modo complejo para solucionar un puzle. Cada nivel presenta un puzle que requiere interacción multimodal produciendo la interactividad del usuario.

Algunos de los elementos que se utilizan en IncreTable son:

1)Mesa táctil. Compuesta por una pantalla de proyección por detrás, un segundo proyector por arriba y una cámara de detección de profundidad.

2)Lápiz digital. La entrada de usuario directa se implementa mediante el uso de lápices digitales.

3)Portales. Son interfaces entre el mundo real y el virtual.

4)Robot. Se mueven con marcadores aumentados que se proyectan en la pantalla de proyección de atrás. El robot tiene cinco sensores de brillo para calcular el desplazamiento relativo entre el robot y la imagen del marcador, y el robot se programa para que siga la imagen del marcador mediante control. De este modo, podemos mover el robot simplemente moviendo el marcador. En el juego, los robots siguen rutas predefinidas o son controlados por los jugadores mediante una entrada de lápiz o gamepad.

En cada nivel del juego los usuarios tienen que jugar con objetos reales y digitales. En uno de los niveles el objetivo es construir una rampa con objetos reales. En este nivel

se proyecta un coche virtual desde arriba en los objetos sobre la mesa. A los objetos físicos sobre la mesa se les localiza mediante una cámara de profundidad. Cuando comienza el nivel, el coche se desliza por la rampa. La velocidad del coche y la dirección se definen mediante la posición y altura de la rampa real. El coche tiene que cruzar el área bajo la torre física (portal) que se activa y cae sobre las fichas de dominó que se pueden ver delante. Otro portal físico detecta si todas las fichas han caído y le dice al juego si el nivel se ha completado con éxito.

Otros escenarios incluyen colocar fichas de dominó para lograr ciertos objetivos en el nivel. En IncreTable, los usuarios no solo juegan con fichas reales de dominó sino también con fichas virtuales. Usando una interfaz de lápiz inalámbrica, los jugadores pueden dibujar un camino sobre la superficie de la mesa para colocar las fichas de dominó digitales (proyectadas). Como el sistema se basa en un motor de física incluso las fichas digitales pueden caerse mediante un manejo torpe.

Los jugadores pueden seleccionar entre diferentes acciones, poner fichas de dominó, reposicionar, o borrar fichas de dominó usando una barra de herramientas tangible. Al mismo tiempo, otros usuarios pueden comenzar a colocar fichas de dominó reales directamente en la superficie de proyección de la mesa, creando una experiencia de realidad mezclada muy interesante. Mientras juegan, los usuarios pueden moverse libremente alrededor de la mesa. IncreTable no tiene un modo dedicado para colocar las fichas de dominó. De ahí que ocurra a menudo que las fichas reales o virtuales comienzan a caerse antes de que los usuarios comiencen la reacción en cadena, obligando a los usuarios a concentrarse y trabajar juntos incluso más aun.

2.1.6 The Augmented Knight's Castle

Es un entorno de juguetes aumentado [Lamp07] formado por edificios y figuras de Playmobil de la Edad Media en el cual se enriquece el juego de los niños con música de fondo, efectos de sonido, comentarios verbales de los juguetes y diferentes formas de comunicación visual y táctil en respuesta al juego de los niños. Además, se pueden integrar en el juego experiencias de aprendizaje interactivo, como por ejemplo enseñar a los niños canciones, poemas o hechos ocurridos en la Edad Media.

Se introducen en el juego teléfonos móviles y "juguetes inteligentes" equipados con sensores y lectores de RFID para mejorar el juego y producir una mayor interacción. Los niños pueden usar los dispositivos móviles para tocar partes del juego como parte de un escenario de aprendizaje, una historia que se desarrolla o simplemente como parte del juego libre. Se persiguen dos enfoques al añadir dispositivos móviles al escenario de juego: en primer lugar, los teléfonos móviles y en segundo lugar, dispositivos móviles con sensores y actuadores integrados en los juguetes que los convierten en "juguetes inteligentes". Además del juego enriquecido y las experiencias de aprendizaje, los entornos de juego aumentados pueden ayudar a facilitar el

desarrollo de habilidades sociales en niños con desordenes sociales o enfermedades mentales como el autismo y incluso ayudar a detectarlas.

Se usa tecnología de identificación por Radio frecuencia (RFID) para identificar los juguetes automáticamente y de forma discreta en el conjunto de juguetes, y para detectar la posición de los objetos en el escenario de juego. Las etiquetas RFID de tamaños diferentes se adjuntan o incorporan a las piezas del escenario de juego para identificarlas únicamente. Para solucionar el problema de la orientación de las etiquetas en los campos de las antenas, se etiquetan la mayoría de los objetos con varias etiquetas de orientación diferente (por ejemplo, la parte trasera y la parte de debajo de las figuras) para leer al menos una de las etiquetas en un campo de antena. El identificador de 64 bits que se almacena en la etiqueta RFID se usa como clave para mapear los objetos del juego con información virtual como el nombre, imágenes, historias o sonidos. Las antenas RFID se pegan a los edificios de juguete o a los diferentes tipos de elementos de suelo, para detectar la presencia de figuras que estén cerca.

Para poder observar la proximidad de objetos de juguete más grandes que se mueven sobre el escenario de juego durante el juego (por ejemplo, un carruaje), se incorporan lectores móviles RFID que se comunican con un computador mediante BlueTooth.

Los lectores RFID, un multiplexor, y los módulos de lectura RFID se conectan a una computadora (estación base) donde se filtran las observaciones de las etiquetas (se eliminan las lecturas de falsos negativos) y se agregan a un middleware RFID. El middleware también proporciona una interfaz abstracta al hardware RFID para intercambiar fácilmente el hardware de diferentes fabricantes sin cambiar el software.

Para integrar los teléfonos móviles en el juego, se incorpora al teléfono móvil un módulo lector de RFID construido a medida, lo que permite apuntar y tocar otros objetos del escenario de juego para obtener información multimedia (audio, texto, imágenes o video) relacionada con el objeto tocado. También los niños pueden tocar las figuras y teléfonos móviles de los otros niños para comenzar una lucha entre las figuras del juego.

Se han construido dispositivos móviles embebidos en juguetes para que haya una integración perfecta e intuitiva con el entorno de juego. Este tipo de juguetes se llaman “juguetes inteligentes” y abarcan uno o más roles en el juego, dependiendo de su rol en la vida real: por ejemplo, una espada se usa para luchar pero también puede usarse para examinar objetos. Al menos debería incluirse un juguete en el juego que permita seleccionar una figura y mostrar información de ella (o reproducir el correspondiente audio y video). A estos “juguetes inteligentes” se les puede adjuntar sensores de luz y aceleración 3D y un micrófono. Estos sensores aportan un contexto al juego que puede añadirse a la interacción “apuntar y tocar” haciendo el juego más

atractivo. Los micrófonos permiten detectar el nivel actual de ruido causado por los niños y actuar en consecuencia. Además, los sensores de aceleración pueden usarse para realizar reconocimiento de gestos como sacudir una botella mágica o un movimiento circular con una varita mágica. Se pueden incorporar módulos de vibración a una varita mágica y otros objetos para proporcionar una respuesta táctil. También se pueden usar LEDs en una botella mágica para dar una respuesta visual a los niños para indicar que la poción mágica de la botella está activa.

Los datos RFID filtrados y totales de todos los lectores se almacenan como información de objetos en un modelo de posición simbólico del juego que se enriquece con información adicional de objetos. El sistema de monitorización de objetos (SMO) se encarga de gestionar el modelo de objetos y proporcionar una abstracción de detalles específicos RFID. Además ofrece un modelo de programación para que el programador del juego pueda definir la generación de eventos y acciones basadas en la información del modelo de objetos. La parte estática del modelo de objetos que representa las posiciones de interés (que se monitorizan mediantes lectores RFID) es predefinida. Por ejemplo, si la figura de juguete del caballero dorado se pone cerca del árbol encantado (es decir, si el lector RFID detecta la etiqueta en la figura de juguete), el SMO recupera información relacionada como el nombre y las descripciones, y crea un objeto nodo para el caballero dorado que se pone en el modelo de objetos como un nodo hijo del árbol encantado. Una vez que se quita la figura de su posición física en el árbol encantado, se quita el nodo del caballero dorado. El modelo de objetos por tanto cambia continuamente cuando los niños están jugando y representa la situación de juego actual como la detecta el hardware RFID en casi tiempo real.

Basándose en el juego o situación de aprendizaje actual y escenario, la lógica de juego determina qué efectos de sonido, música de fondo o comentarios verbales se reproducen o qué actuadores se activan como respuesta a una acción o aleatoriamente. En la implementación actual, la lógica del juego se realiza con una arquitectura software del SMO basada en una maquina de estados que permite reaccionar a situaciones más complejas en un tiempo determinado. Se usa una maquina de estados para modelar los procesos de juego a medida que avanza el tiempo, donde un estado representa una cierta situación de juego. Las transiciones de estados se disparan mediante condiciones que el programador del juego puede formular mediante operadores lógicos y un lenguaje de consulta simple para acceder a la información del modelo de objetos. Los mismos estados pueden realizar acciones como reproducir sonidos o activar actuadores. El conjunto de todas las definiciones de maquinas de estados crea la reproducción de audio de sonidos, música y comentarios verbales y la respuesta visual y táctil. Por ejemplo, se podría disparar un evento cuando el dragón rojo sale de la torre del dragón o la música de fondo cambia cuando de idílica a sonido de batalla cuando al menos tres figuras de caballeros del dragón se ponen en la llanura delante del castillo del rey. De forma similar los escenarios de

aprendizaje se modelan con maquinas de estados que reaccionan a una cierta configuración de las figuras de juguete.

2.2 Sistemas de creación de juegos con elementos propios

Los sistemas incluidos en esta categoría permiten crear entornos de juego mediante elementos creados por el usuario.

2.2.1 LogoBlocks

LogoBlocks [Bege96] es un lenguaje de programación gráfico, desarrollado por Andrew Begel del grupo de Epistemología y aprendizaje del MIT, para programar el "Programmable brick". El "Programmable brick" es un pequeño ordenador portátil que una persona puede agregar a una creación LEGO para controlar motores y leer entradas de sensores. LogoBlocks se creó como alternativa gráfica al lenguaje BrickLogo, que es una variante de Logo desarrollada para usarse con el "Programmable brick". En vez de escribir un programa textual con sus construcciones sintácticas, los usuarios pueden poner bloques que representan trozos de un programa en la pantalla. Gracias a la programación gráfica, al usar una representación gráfica de los objetos, se puede mostrar más concretamente la orientación del objeto (doble clic en el contenedor del objeto para ver qué hay dentro), se elimina sintaxis molesta y se visualiza mejor la ruta que el programa está siguiendo. La programación gráfica facilita la compartición de programas y la explorabilidad de los programas (mirando a la imagen de un programa, un usuario podría deducir más fácilmente su significado). En los lenguajes gráficos se usan indicaciones visuales. Las conexiones entre los objetos pueden hacerse explicitas mediante el diseño y la representación gráfica de las construcciones. Los procedimientos que toman parámetros podrían tener agujeros que se rellenen con los parámetros que tengan la forma adecuada.

Los inconvenientes de la programación gráfica son: la frustración de programadores sofisticados que quieren expresar concisamente una afirmación que podría representarse mejor usando texto y el estado real de la pantalla (no se pueden tener más de 50 primitivas visuales en la pantalla al mismo tiempo). Para que los iconos y los gráficos sean comprensibles necesitan ser los suficientemente grandes para que puedan verse o tener una etiqueta de texto. Algunos lenguajes también describen llamadas a funciones entre clúster de gráficos. Si hay demasiadas funciones en una página, el código se vuelve lioso y difícil de seguir. Otro problema es la difícil extensibilidad (los lenguajes gráficos tienden a estar limitados por el diseño del autor sin pensar en la adición de características adicionales).

En LogoBlocks, los usuarios pueden coger bloques de una paleta en la parte izquierda de la pantalla y ponerlos en el área principal de trabajo. Cada bloque tiene un color y forma diferente.

Existen tipos diferentes de bloques:

- Bloques de acciones, pueden controlar los motores y “hacer” operaciones como wait y repeat. (Son aproximadamente rectangulares pero tienen bordes redondeados)

- Bloques de sensores, permiten obtener información del mundo real para el programa. Los bloques de sensores tienen forma oval. 2 Tipos: digitales (on/off) y analógicos (valores de 0 a 255).

- Bloques de variables, permiten a los usuarios conectar números a funciones que los necesitan, como en onfor, wait y repeat.

-Variables numéricas. Tienen forma de flechas. Los números pueden cambiarse de 0 a 255 pulsando control + clic.

- Bloques de procedimientos. Para implementar abstracción de procedimientos. Tienen forma de carpetas con texturas de animales.

Todos los bloques se identifican por una forma única y una descripción textual que se dibuja dentro de cada uno. Cada bloque tiene varios significados predefinidos que el usuario puede seleccionar.

Indicaciones visuales

1) Cualquier bloque que requiere una variable tiene un corte con la forma de la variable que encajará allí.
2) Flujo de programa. Ver un programa ordenado en múltiples bloques de colores en una buena manera de ver el flujo de instrucciones. Otra ventaja es que se puede visualizar fácilmente la programación paralela.
3) Para solucionar el problema del límite “Deutsch” de la legibilidad de pantalla, se añadieron múltiples páginas al espacio de trabajo. Los bloques están en una página particular, así que podríamos tener una página para una definición de procedimiento particular y llamar al procedimiento de otra página.

2.2.2 AgentSheets

AgentSheets [Repe00] es una herramienta basada en agentes que permite a sus usuarios crear simulaciones y juegos interactivos en 2D y publicarlas como applets de Java en la web mediante una interfaz amigable de drag and drop (arrastrar y soltar). AgentSheets combina tecnología Java, tecnología de agentes programables de usuario final y tecnología de hojas de cálculo.

En AgentSheets, los usuarios pueden crear simulaciones especificando el comportamiento de agentes en un mundo en 2D basado en una cuadricula similar a una hoja de cálculo. Una celda de la cuadricula puede contener cualquier número de agentes apilados y los usuarios interactúan con los agentes mediante manipulación directa. Los agentes son rutinas software que se ejecutan en segundo plano y realizan una acción cuando ocurre un evento especificado. Los usuarios diseñan el aspecto de

los agentes dibujando iconos y crean programas usando reglas de reescritura graficas en las cuales los usuarios seleccionan condiciones (configuraciones de iconos en el mundo o relativos a otro agente) y acciones que muestran al sistema lo que debería pasar bajo estas condiciones moviendo los agentes a sus nuevas posiciones. Las condiciones y las acciones permiten a los agentes realizar una variedad de operaciones que incluyen computación de formulas similares a las hojas de cálculo, reacción a los clics de ratón y a las teclas, detectar otros agentes, envío de mensajes a agentes, reproducir sonidos e instrumentos MIDI, hablar, recopilar información de páginas web, moverse a nuevas posiciones de la cuadricula, y cambiar la apariencia. Además, AgentSheets proporciona herramientas para crear analogías entre agentes. Por ejemplo, si un usuario quiere que un tren siga unas vías de tren de la misma forma que un coche sigue una carretera, puede usar la herramienta de analogía para especificar esto fácilmente. El uso de analogías proporciona una forma fácil de reusar código.

AgentSheets utiliza el lenguaje de programación de usuario final Visual AgenTalk©. Visual AgenTalk es un entorno de programación basado en un enfoque que los diseñadores del sistema llamaron “Programación táctil” que se centra en permitir a los usuarios manipular código en múltiples contextos para ayudar a su comprensión, en la construcción de programas más complejos y en poder compartir código entre programadores. Los diseñadores de AgenTalk creen que los usuarios deben poder soltar trozos de código (comandos o sentencias condicionales) en tres contextos: el editor de programa, el mundo de programación (el mundo basado en una rejilla en el que se ejecuta el programa), y el mundo de colaboración. Permitir a los usuarios soltar código en el mundo de programación permite a los usuarios probar el comportamiento de trozos individuales de código sin ejecutar el programa entero. Esto da a los usuarios un modo de explorar y comprender código que no crearon. Visual AgenTalk también permite a los usuarios compartir código fácilmente con otros usuarios a través de la web.

2.2.3 Scratch

Scratch [Resn09] es un lenguaje de programación grafico desarrollado por el MIT que permite a los niños programar historias interactivas, juegos, animaciones y simulaciones y compartir tus proyectos con otros. Scratch está basado en el lenguaje de programación gráfico Logoblocks. Los usuarios construyen los programas arrastrando y soltando elementos de código, lo que elimina la posibilidad de cometer errores de sintaxis. La gramática que utiliza Scratch está basada en una colección de “bloques de programación” gráficos que los niños unen entre sí para crear programas. Los conectores de los bloques sugieren la manera en la que deberían encajar las piezas. Los bloques se crean para que encajen solamente de forma que tengan sentido sintácticamente. Las estructuras de control (como forever y repeat) tienen forma de C para sugerir que los bloques deberían colocarse dentro de ellas. Los bloques que imprimen valores se les da una forma acorde con los tipos de valores que devolverán:

oval para números y hexagonal para booleanos. Los bloques condicionales (como el if y el repeat-until) tienen huecos con forma de hexágono para indicar que se requiere un booleano. Las posibles operaciones que se pueden realizar están prefijadas y los parámetros de las operaciones parámetros están predefinidos.

La pantalla principal de Scratch consta de un área donde se visualiza la animación 2D, un área para seleccionar el tipo de bloque que queremos utilizar en nuestra animación (movimiento, apariencia, sonido, control, operadores, variables, etc.), un área donde se muestran todos los bloques del tipo seleccionado, un área de scripts donde se arrastran los bloques que componen el programa, un área de creación de vestimentas, un área creación de sonidos y un área de creación de sprites.

Scratch se diseñó para ser altamente interactivo. Simplemente clicando en una pila de bloques, comienza a ejecutarse su código inmediatamente. Se pueden incluso hacer cambios a una pila mientras se está ejecutando, por lo que es fácil experimentar con nuevas ideas incrementalmente y iterativamente. Si se quieren hilos paralelos, simplemente se crean múltiples pilas de bloques. Scratch soporta muchos tipos diferentes de proyectos (historias, juegos, animaciones, simulaciones), por lo que gente con una amplia variedad de intereses puede trabajar en proyectos que les parezcan atractivos. También facilita la personalización de sus proyectos mediante la importación de clips musicales, voces grabadas y la creación de gráficos.

Existe una comunidad de usuarios de Scratch on-line. Dado que Scratch es una aplicación de escritorio monousuario, para promover la discusión y la creatividad, la comunidad on-line provee la infraestructura necesaria para ese fin, de tal forma que los usuarios pueden colaborar, apoyarse, criticarse mutuamente y contribuir al trabajo de otros usuarios de forma deslocalizada. La interfaz de usuario de Scratch permite fácilmente la conexión con la comunidad on-line para una fácil y rápida compartición de proyectos con sitio web.

2.2.4 ShadowStory

ShadowStory [Lu11] es un sistema de narración de historias inspirado en las marionetas de sombras tradicionales chinas que permite a los niños usar un tablet PC para crear personajes digitales animados y otros accesorios al estilo de las marionetas y con herramientas (virtuales) similares; y interpretar historias en directo en una pantalla de proyección controlando los personajes con movimientos simples del cuerpo mediante sensores de orientación manuales e inalámbricos.

Las marionetas de sombra son figuras planas articuladas hechas con cuero semitransparente. Se manipulan desde atrás y contra una pantalla iluminada por detrás usando una serie de palos. Los escenarios, o los telones de fondo están hechos también de cuero y están pegados a la pantalla. La trama de la historia se representa

mediante movimientos de las marionetas, narraciones y conversaciones, así como libretos cantados por interpretes, parecidos a los de las operas chinas.

ShadowStory incluye dos modos de interacción: el modo “diseño” en el que se crean los elementos de la historia; y el modo “interpretación” en el que se puede interpretar la historia en público. También hay una librería de videos de obras de marionetas para que los niños las vean cuando quieran.

En el modo diseño, los niños usan un tablet PC con un lápiz como entrada para crear tres tipos de elementos de historia: personajes, accesorios y telones de fondo. Para crear un personaje, el sistema proporciona una plantilla articulada que consta de las partes del cuerpo requeridas: cabeza, pecho, barriga, antebrazo izquierdo/derecho, brazo izquierdo/derecho, mano izquierda/derecha y pierna izquierda/derecha y un elemento opcional del personaje. Los niños pueden crear estas partes individualmente usando una herramienta “cuchillo” y una herramienta “brocha” para tallar o pintar en ella. Parecidos a los cuchillos físicos que se usan al crear marionetas reales, se proporcionan dos tipos de “cuchillos”: uno para tallar el contorno y uno para tallar patrones interiores. La herramienta brocha permite a los niños pintar con varios colores y grosores. Además varias herramientas de “stamp” permiten a los niños imprimir patrones decorativos tradicionales sin esfuerzo. Después de crear las partes individuales los niños pueden guardar el personaje articulado completo en la librería para usarlo en la interpretación. Para crear accesorios que no son articulados, el sistema proporciona un fondo vacio gris que representa un trozo de cuero, como el que usan los artistas de marionetas, en el cual los niños pueden usar las mismas herramientas de cuchillo y brocha para crear objetos movibles como por ejemplo animales. Del mismo modo pueden crear escenarios que permanecen estáticos durante la obra. Además de los elementos creados por los niños, el sistema incluye una librería de personajes, accesorios, y escenarios, cuyo contenido es generalmente familiar para los niños chinos. Estos pueden usarse directamente en sus obras o servir de inspiración para sus propias creaciones. Los elementos auditivos como cantos, narración o música no se incluyen en el modo diseño.

Después de crear todos los elementos de la historia, los niños pueden cambiar al modo Interpretación para representar sus historias. En primer lugar, deberían organizar el escenario con los elementos de acuerdo a las historias. Pueden introducir elementos salvados en el escenario seleccionando de la librería de personajes, accesorios o fondos. Cada personaje o accesorio añadido al escenario se asigna automáticamente a un par de sensores manuales, mientras que un telón introducido en el escenario reemplaza al anterior. Una vez que el escenario está preparado, los niños pueden pulsar el botón “perform” para activar los sensores manuales e interpretar sus historias. La interfaz de interpretación se muestra también en una pantalla de

proyección visible para todos los intérpretes y públicos, permitiéndoles participar con otros de manera similar a las obras de marionetas reales.

Manipular una marioneta real requiere mucha habilidad de control de las manos y tocar directamente la marioneta con los palos. Para simplificar el mecanismo de control en nuestro sistema, y para liberar a los niños de estar junto al ordenador, optamos por usar sensores manuales inalámbricos. Para cada personaje, el niño que controla tiene un par de sensores de orientación 3D, uno en cada mano. En vez de la manipulación directa de las partes de la marioneta, el niño puede mover el personaje a la izquierda, derecha, arriba y abajo inclinando el primer sensor en la dirección correspondiente. Inclinar el segundo sensor a la izquierda o a la derecha produce que el personaje doble el cuerpo en la correspondiente dirección, como si se inclinara o mirara hacia arriba. Otras articulaciones del personaje se mueven como corresponde para que concuerden con los movimientos principales, por ejemplo los brazos y piernas se balancean naturalmente a medida que el personaje camina. De forma similar, el niño puede mover o rotar un accesorio con un par de sensores. Como un niño solo puede controlar un personaje o accesorio, casi todas las historias deben ser interpretadas por varios niños colaborativamente.

Los niños pueden narrar, doblar o hacer efectos de sonido mientras interpretan. Cuando acaba la historia, pueden apretar el botón de Stop para acabar.

2.2.5 Topobo

Topobo [Park08] es un sistema 3D de ensamblado constructivo con memoria cinética embebida. El objetivo de Topobo es ayudar a los niños a mejorar la comprensión del equilibrio, del centro de masas, la coordinación, el movimiento relativo y los múltiples grados de libertad.

Permite a los niños grabar y reproducir el movimiento físico. Conectando una combinación de componentes pasivos (estáticos) y activos (motorizados), los niños pueden montar rápidamente formas dinámicas como animales y esqueletos. Apretando, tirando, retorciendo y estirando los componentes pueden animar esas formas.

Topobo se basa en la programación por demostración. Al pulsar el botón de grabación de un componente activo, se puede indicar al componente activo como queremos que se comporte. Al cambiar al modo de reproducción el componente activo repite el comportamiento que se ha grabado anteriormente.

Topobo permite la interacción física de varios niños para construir una forma, con lo que fomenta la colaboración entre ellos.

Topobo se utilizó en cinco casos de estudio:

1) En un programa extraescolar de enriquecimiento realizado por profesores con 18 estudiantes de 13 a 15 años durante 3 meses con sesiones temáticas y juego libre.
2) En aulas de 4º y 7º grado de ciencias realizado por profesores de ciencias para 36 estudiantes de 9-10 años y 12-13 años durante 8 meses con lecciones orientadas a objetivos y juego libre.
3) En un centro de robótica realizado por pedagogos para 32 estudiantes de 4-6 años y 8-14 años durante 5 meses en sesiones guiadas.
4) En un museo de ciencias realizado por exhibidores y programadores para más de 200 personas durante 4 meses con actividades en el suelo, demostraciones y conversaciones internas.
5) En un curso de arquitectura con 12 estudiantes de arquitectura (centrado en uno específicamente) de 24 a 29 años durante 8 meses para el trabajo de diseño en la tesis.

En todos los contextos, los educadores que trabajaron con Topobo lo consideraron una herramienta útil o interesante. Sin embargo, como kit de construcción destacó en contextos que permitían un periodo mayor de uso. En general, los niños más pequeños necesitan más tiempo con el sistema que los mayores, y las interacciones cortas (con un usuario de cualquier edad) demandan actividades más limitadas. Quizás el mensaje más destacado de los propios educadores es que los educadores necesitan experiencia previa con el sistema para ganar confianza en su habilidad para enseñarlo, y les hubiera gustado tener materiales de soporte a la enseñanza más completos.

Los comentarios de los educadores y el uso de Topobo demostraron que desean las mismas cosas hacia las que los tangibles están ya trabajando: una estructura de programación y control más transparente, la habilidad de jugar físicamente con ideas de matemáticas y ciencias, y la habilidad de poner en las manos de la gente comportamientos dinámicos y simulaciones que son una parte cada vez más importante de la enseñanza científica. Los hallazgos y conclusiones de los casos de estudios anteriormente citados no se reportaron con datos.

2.3 Sistemas que sólo permiten jugar con elementos predefinidos

Los sistemas incluidos en esta categoría solo permiten jugar en un único entorno de juego con elementos predefinidos.

2.3.1 Cleogo

Cleogo [Cock98] es un entorno de programación en grupo basado en el lenguaje de programación Logo que permite a varios usuarios (cada uno en su propio ordenador y conectados entre ellos a través de Internet) colaborar en tiempo real en el desarrollo de programas simultáneamente y ver su ejecución. Cleogo utiliza una interfaz gráfica

de usuario de tipo Windows (es decir, con ventanas, iconos, menús y un puntero de ratón) para programar el movimiento de una tortuga.

El objetivo de Cleogo es animar a los niños a la resolución colaborativa de problemas. Cada usuario de Cleogo tiene su propia pantalla, teclado y ratón, y comparte el control simultáneo de todos los mecanismos de interfaz. No hay límites integrados en el número de usuarios simultáneos, pero cuatro es un máximo realista antes de que la degradación de la respuesta del sistema comience a afectar a la colaboración. Los usuarios pueden estar en la misma habitación o distribuidos físicamente por Internet, en cuyo caso se necesita una canal de audio adicional para poder comunicarse mediante la voz. Las acciones de programación de un usuario se comunican inmediatamente a los otros usuarios.

No hay restricciones en las acciones que cada usuario puede hacer en cada momento, y no hay controles de acceso para determinar qué partes de la interfaz pueden manipular los usuarios. Si dos usuarios intentan establecer un valor diferente para un parámetro de forma simultánea, o si un usuario intenta mover la tortuga hacia adelante mientras otro la hace moverse para atrás, no hay políticas de software para resolver el conflicto. La resolución de los conflictos se deja a los usuarios mediante protocolos sociales. El asunto clave para el software es asegurar que todos los usuarios son conscientes de las acciones de los otros. En Cleogo, esto se consigue mediante “telepointers” que continuamente muestran la posición del cursor de cada usuario a través de todas las ventanas del usuario. Los telepointer también realizan un papel crítico al apoyar a las referencias de los usuarios en las que expresiones como “esto”, ”eso” y “ponlo allí” requieren una actividad gestual para clarificar el contexto de la sentencia.

Para desarrollar los programas, los usuarios pueden utilizar cualquier mezcla de tres paradigmas de programación: un lenguaje de manipulación directa para programación por demostración, un lenguaje con iconos y un lenguaje estándar basado en texto. Los usuarios pueden seleccionar el paradigma que se adecue a su tarea o a su nivel de habilidad. Las acciones de programación expresadas como entrada en cualquiera de los paradigmas producen las expresiones de salida correspondientes en los otros dos paradigmas.

El entorno de programación basado en texto soporta un dialecto estándar de Logo, sin ordenes de procesamiento de listas. Las líneas de programa se escriben en una ventana de entrada de texto en la parte baja de la pantalla y se ejecutan cuando el usuario clica el botón “Do It” o cuando pulsa la tecla return. El historial de las órdenes previamente ejecutadas, que puede haberse expresado en cualquiera de los paradigmas, se muestra en una listbox desplazable. Los usuarios pueden volver a ejecutar líneas o secuencias de líneas en la lista del historial seleccionando y clicando.

Existe otra ventana que proporciona una representación usando iconos de todos los elementos del lenguaje de Cleogo. Los mecanismo icónicos para construcciones de Cleogo incluyen mecanismos para definir nuevos procedimientos con parámetros y valores de muestra de parámetros, construcciones para crear bucles repeat, construcciones para generar sentencias condicionales, una "calculadora" para generar expresiones, mecanismos para llamar a procedimientos de movimiento de la tortuga, e iconos que permiten al usuario procedimientos definidos por el usuario con valores de parámetros. Los valores de parámetros para cualquier procedimiento se establecen mediante un deslizador que se encuentra junto a los iconos de procedimientos. Los procedimientos definidos por el usuario pueden mostrar un número arbitrario de sliders de parámetros, pero para guardar procedimientos del estado real de la pantalla definidos dentro del entorno de programación con iconos, se limita a dos parámetros.

En la ventana de programación por manipulación directa, los usuarios generan ordenes de Logo arrastrando diferentes segmentos de la tortuga con el ratón. Las acciones de programación correspondientes que producen idénticos movimientos de la tortuga se muestran simultáneamente en las ventanas de programación de texto y con iconos. Arrastrar la cabeza de la tortuga provoca que la tortuga gire inmediatamente. Arrastrar su cuerpo provoca movimiento hacia adelante en línea recta o hacia atrás, y clicar la cola de la tortuga cambia de Pen-up (levantar el lápiz) a Pen-down (bajar el lápiz).

2.3.2 TurTan

TurTan [Gall08] [Juli09] es un lenguaje de programación tangible que utiliza una interfaz de mesa táctil con objetos tangibles que representan instrucciones virtuales de un programa. Está inspirado en Logo, y por lo tanto se diseñó para realizar geometrías con una tortuga. Como en el lenguaje Logo original, uno de los objetivos de diseño de Turtan es la enseñanza de conceptos de programación, principalmente a niños.

Turtan se ejecuta en una mesa táctil, que se ha construido con un proyector y una cámara bajo la mesa. La cámara captura toda la actividad de la superficie, identificando y detectando las posiciones de los objetos y de los dedos, mientras que el proyector se ocupa de la representación visual del sistema. Todo el sistema se ejecuta en un solo PC como dos procesos separados: uno para el seguimiento visual y el otro para la aplicación principal Turtan. Este último se encarga del diseño de la interacción, la lógica de aplicación y la representación visual que se dibuja permanentemente en la superficie de la mesa. Turtan comienza con una pantalla negra con la imagen de una pequeña tortuga en mitad de la superficie y un conjunto de objetos de formas diferentes y con iconos diferentes dispuestos aleatoriamente por toda la pantalla. Cuando se pone un tangible en la mesa, se muestra una respuesta visual debajo del bloque informando al usuario de que se ha reconocido el bloque y la tortuga realiza la instrucción asociada. La tortuga ejecuta cada instrucción relativa respecto a su propia posición, creando instantáneamente un vector grafico de salida que se imprime en un lienzo virtual, y que es manipulable mediante gestos. El usuario puede cambiar la

orientación, la posición o ampliar este lienzo con un dedo para translaciones, o dos dedos para escalar y rotar.

Los programas de Turtan se componen de una secuencia de instrucciones tangibles. Cada instrucción tiene un parámetro variable que se determina mediante la rotación que el usuario aplica a su correspondiente tangible. Pueden usarse varios tangibles del mismo tipo en el programa. Los tipos de instrucciones, manteniendo las operaciones básicas de Logo, son los siguientes:

Moverse sin pintar. La tortuga se mueve hacia adelante y hacia atrás sin pintar una línea.

Moverse pintando. La tortuga se mueve hacia adelante y hacia atrás dejando un trazo de línea coloreado.

Rotar. La tortuga gira hacia la derecha o la izquierda.

Escalar. La tortuga escala su tamaño y también pinta sus distancias de trazo siguientes en una escala diferente.

Cambiar color. Cambia el color de la línea.

Repetir. La tortuga repite todas las acciones desde la primera instrucción tantas veces como se indique.

Comenzar. Pone la tortuga en el centro.

TurTan integra de manera natural la interacción intuitiva basada en toques, gestos y uso de tangibles con la visualización en tiempo real de la salida del programa que los usuarios colaborativamente están construyendo de manera explorativa.

2.3.3 Teaching Table

Teaching Table [Khan07] [Khan06] es una mesa táctil interactiva desarrollada con el objetivo de enseñar matemáticas básicas a niños de 3 a 5 años mediante la realización de actividades físicas.

El niño tiene que poner sobre la mesa táctil unos objetos etiquetados (en forma de bloques numerados, formas geométricas, etc.) para resolver los problemas que se le plantean. Un sistema de ayuda da consejos informativos al niño cada vez que comete un error. Actualmente, la Teaching Table puede ser usada por solo un niño a la vez, por lo que es monousuario, y por tanto, no es colaborativa.

Teaching Table utiliza tecnología de detección electromagnética (EM) para obtener una localización precisa de los objetos físicos etiquetados a medida que se mueven por la superficie de la mesa. La detección se implementa mediante una superficie de detección formada por una rejilla de cables de antena, y un conjunto de etiquetas

actuables electromagnéticamente que pueden detectarse en la superficie de la rejilla. Una pantalla LCD proporciona una salida gráfica que coincide con la colocación y movimiento de los objetos en la rejilla de detección, permitiendo una respuesta visual en tiempo real para las interacciones de un niño con las piezas físicas. Se incorporan también unos altavoces en la configuración de la mesa para proporcionar un canal de respuesta de audio adicional.

El software esta implementado en Java y consta de tres módulos: módulo de detección de objetos (maneja la comunicación con el hardware de posicionamiento), módulo de actividad (controla las actividades educacionales y la salida visual) y herramientas de evaluación (permite realizar tareas administrativas y evaluación de los niños).

El módulo de detección de la posición lee la etiqueta ID y la información de posición mediante el puerto serie y la transmite al módulo de actividad, que actualiza su estado actual. El sistema responde entonces a las interacciones del niño con la respuesta audiovisual adecuada. La reproducción de las diferentes actividades de aprendizaje puede controlarse desde una consola de escritorio separada basada en PC, que permite al profesor monitorizar el progreso del niño.

El sistema se diseñó para integrarse en el entorno de clase, y proporciona actividades de aprendizaje para niños y herramientas de evaluación para profesores que se ajustan al plan de estudios de preescolar. Algunas de las categorías de actividades que se pueden realizar con la Teaching Table son:

Categoría 1: Desarrollo de una comprensión de los números.

Temas tratados: Contar de memoria; identificar números, etc.

Actividad de ejemplo: Poner los números 1-10 in en orden en la superficie de la mesa.

Categoría 2: Crear y duplicar patrones simples

Temas tratados: Repetir patrones; completar patrones incompletos, etc.

Actividad de ejemplo: Recrear un patrón (visualizado en la pantalla) poniendo bloques en los lugares apropiados.

Categoría 3: Ordenación y clasificación de objetos

Temas tratados: Identificar/ordenar los objetos parecidos (todos los rojos juntos, todos los cuadrados juntos) etc.

Actividad de ejemplo: Poner todos los bloques rojos juntos, y todos los bloques azules juntos.

Categoría 4: Desarrollar un sentido del espacio y una comprensión de las formas geométricas básicas.

Temas tratados: Reconocer, describir y comparar las formas geométricas básicas, etc.

Actividad de ejemplo: Similar a la categoría 1, con la única diferencia de que se usan formas geométricas en vez de bloques de números.

Categoría 5: Aprender a usar varios medios de medida estándar y no estándar.

Temas tratados: Clasificación, Ordenación, etc.

Actividad de ejemplo: Encontrar el bloque más grande de un conjunto dado de bloques.

La potencia de la Teaching Table yace en su naturaleza genérica, que permite actividades que ayudan al desarrollo de una variedad de diferentes habilidades. Aunque la investigación actual se ha centrado en el desarrollo de habilidades matemáticas, se podrían extender las actividades para cubrir conceptos en otras áreas, como el lenguaje y la alfabetización, la ciencia, y el desarrollo creativo. También se ha pensado en desarrollar actividades colaborativas para que pequeños grupos de niños interactúen juntos con la mesa.

2.3.4 TViews Table RPG (TTRPG)

TViews Table RPG [Maza07] es una implementación del juego de rol "Dungeons and Dragons" sobre la mesa táctil Tviews. TViews es una mesa táctil interactiva que puede seguir simultáneamente la posición de múltiples objetos etiquetados en tiempo real a medida que se mueven por su superficie, proporcionando una visualización gráfica coincidente y simultanea.

En un juego de rol, los jugadores adoptan el rol de personajes ficticios creados por ellos mismos y trabajan juntos para contar una historia dentro de un sistema dado de reglas. En los juegos de rol tradicionales que se juegan en persona y en grupos pequeños, un director de juego guía la actividad y ayuda a conducir y dar forma al desarrollo de la historia, proponiendo retos que los jugadores deben superar. La construcción de la historia ocurre de manera improvisada, a medida que los participantes determinan las acciones de sus personajes dentro del marco del juego, y tejen colectivamente los fragmentos de la historia en un todo coherente.

Para coordinar el juego y llevar la trama hacia adelante el director de juego puede ajustar varios elementos del juego mediante la GMI (GameMaster Interface) que se ejecuta en un ordenador adyacente a la mesa Tviews. EL GMI permite al director de juego controlar las estadísticas y el comportamiento de varias entidades del juego como personajes no jugadores (PNJs) y objetos mágicos o inanimados.

TTRPG se implementó para hasta tres jugadores que se sientan alrededor de la mesa táctil y manipulan objetos tangibles (peones de personajes) que representan a sus personajes (guerrero, mago o pícaro) y a otros objetos del juego como una herramienta de selección (para validar las elecciones de menú) y un circulo de opciones para recorrer los menús. Los peones de los personajes pueden moverse por los mapas que se muestran en la mesa, y pueden colocarse en los lugares apropiados para disparar las acciones deseadas. Los jugadores pueden manipular un círculo tangible de opciones para activar diferentes menús durante el juego y recorrer las opciones. El sistema detecta la posición y orientación del círculo de opciones en la mesa. Cualquier jugador puede usar el círculo de opciones en su turno para acceder a un menú u opción para su personaje colocando el círculo de opciones en el área directamente delante de él. Esto provoca la aparición de un menú grafico, mostrando las opciones relevantes en el contexto relacionadas con la actual posición del personaje. El jugador que tiene el control del círculo de opciones utiliza una herramienta de selección. Cuando un jugador marca su opción de menú deseada, debe validar su opción con la herramienta de selección. Esto se realiza colocándola cerca del centro del tablero. Una vez que la herramienta de selección se pone en la mesa, se marca con un círculo rojo. Esto indica al jugador que el tablero está esperando validar la opción de menú. El jugador puede validar la opción de menú deseada arrastrando la herramienta de selección a uno de los hotspots del tablero, localizados en las dos esquinas entre los tres jugadores.

El juego en el entorno de la mesa consta de tres modos diferentes: selección de personajes, juego libre y lucha. Cuando el director de juego comienza el programa, se muestra la pantalla de selección de personajes, los jugadores eligen el peón que quieren usar para representar su personaje. Usando su peón, seleccionan las opciones de personaje que se muestran frente a ellos. Después de que los jugadores han seleccionado sus personajes, se les envía a la primera habitación de la mazmorra. En este momento, los jugadores están en juego libre y deben mover sus personajes a las posiciones iniciales de la habitación que indican las áreas de tres colores. Los jugadores inician el modo Lucha diciéndole al director de juego que quieren atacar a uno de los orcos. Un jugador puede iniciar el modo de lucha espontáneamente, o puede discutir la decisión con otros jugadores.

2.4 Sistemas de creación de otros sistemas

Los sistemas incluidos en esta categoría permiten crear otros sistemas de juegos y están orientados al diseñador de juegos.

2.4.1 Raptor

Raptor [Smit09] [Smit10] es una herramienta que nos permite realizar bocetos de juegos de ordenador. La realización de bocetos permite a los jugadores experimentar las ideas del juego y explorar si el juego que se está desarrollando será entretenido sin

necesidad de construir un prototipo totalmente funcional. Debido al alto coste de desarrollo de juegos, el ámbito de uso de Raptor es en el diseño temprano (donde la realización de bocetos es más beneficiosa). Raptor puede usarse para realizar bocetos de un amplio rango de tipos de juego, como juegos de rol fantásticos, juegos de disparos multijugador o juegos de carreras, pero no funciona bien en juegos con grandes cantidades de movimiento vertical u oclusión desde arriba debido a que los testeadores y los diseñadores tienen diferentes puntos de vista sobre el juego. Los diseñadores tienen una vista desde arriba 2D[1], mientras que los testeadores ven el juego en el juego en la forma tradicional 3D.

Los diseñadores del juego realizan el boceto en una mesa táctil y un testeador ejecuta el juego sobre un PC (que está conectado a la mesa táctil por medio de la red) interaccionando de forma habitual con un mando de juegos. La mesa táctil permite la incorporación de forma dinámica de cambios en el prototipo mientras el testeador está probando el juego en el PC.

La interfaz de la mesa táctil cubre la superficie de una mesa alrededor de la cual los múltiples usuarios pueden trabajar al mismo tiempo y mejora la de una interfaz de escritorio ya que proporciona:

Transparencia de las herramientas, en la que los bocetos pueden crearse usando gestos y accesorios físicos en vez de apuntar y clicar.

Mejora la colaboración, ya que el diseño de la mesa permite una comunicación íntima, y en la que la entrada multitouch de la mesa permite que más de una persona interactúe con el boceto a la vez.

Proceso de diseño igualitario, en el que no se favorece a los programadores, y los roles pueden establecerse y cambiarse de forma fluida.

La mesa táctil permite realizar bocetos de juego de forma colaborativa y más fácilmente que las herramientas tradicionales basadas en ordenadores de escritorio. Con Raptor, los diseñadores utilizan gestos para crear y manipular mundos virtuales. Los gestos se utilizan también para conectar controladores de juego a objetos virtuales en el mundo de juego.

Para comenzar a crear un prototipo de un juego, los usuarios crean un mundo virtual. A los diseñadores se les presenta un terreno vacio en la superficie de la mesa. El terreno puede navegarse usando gestos panning y zooming. Para dar forma al terreno, usan gestos que imitan la manipulación de la arena en una caja de arena. Para crear una colina, usan un gesto de “cavar”. Del mismo modo, para crear un valle se usa un gesto de “extender”. Para añadir objetos a la escena, colocan un objeto físico en la mesa táctil. El objeto virtual se añade al escenario en la misma posición en que se puso

[1] Dos dimensiones

el objeto físico. Para eliminar objetos virtuales se usa un borrador. Los jugadores pueden arrastrar los objetos virtuales por el escenario con los dedos.

Raptor permite a los diseñadores prototipar rápidamente la entrada del juego mediante una interacción de realidad mezclada con la mesa táctil. Por ejemplo, al poner sobre la mesa táctil, un mando de una consola Xbox, se muestran un anillo de pines que rodea el controlador y cuyos pines representan los diferentes canales de entrada y salida que el controlador proporciona.

Para ver el mundo de juego desde otro ángulo distinto de la vista desde arriba de la mesa, los diseñadores pueden conectar una pantalla mediante la LAN. La posición de la vista y el ángulo se controlan poniendo pequeños cámaras de plástico sobre la superficie de la mesa. Para mover la posición de la cámara, el diseñador simplemente pone la cámara sobre la mesa y la rota hasta tener el ángulo deseado. Los diseñadores pueden también pulsar un objeto interactivo con la cámara de plástico para crear una vista "persecución". Cuando una cámara está en modo persecución, seguirá de cerca detrás del objeto interactivo a medida que se mueve por el mundo de juego.

Los bocetos de Raptor se basan en el prototipado "Wizard-of-Oz", que permite a los usuarios experimentar la interacción con un sistema real interactivo incluso antes de que exista. En un prototipo Wizard-of-Oz, los diseñadores, en vez de tener que programar nuevas ideas de juegos, pueden llevarlas a cabo utilizando la mesa táctil para manipular rápidamente el juego mientras las testeadores prueban el juego sentados frente a una pantalla tradicional. Los diseñadores ven la interacción del testeador y pueden modificar las respuestas a la entrada del usuario. La técnica de prototipado "Wizard-of-Oz" es particularmente útil cuando el comportamiento del sistema aún no está bien definido. En estos casos, el desarrollador puede experimentar con ideas diferentes simplemente cambiando las respuestas a la entrada del usuario. Sin embargo, la técnica se limita a sistemas en los que la rápida respuesta a la entrada del usuario no es crítica para dar tiempo al diseñador a modificar la respuesta a la entrada del usuario.

2.4.2 STARS

STARS [Mage03] es una plataforma para desarrollar juegos de mesa aumentados por computador, que integran dispositivos móviles con una mesa interactiva. El objetivo de STARS es aumentar los juegos de mesa tradicionales con funcionalidad de computación pero sin sacrificar las dinámicas de interacción centradas en humanos de los juegos tradicionales de mesa táctil.

STARS consta de un marco de interacción que dinámicamente acopla dispositivos de entrada y salida móviles y estacionarios con el juego de mesa.

El rango de tipos de juegos soportados en STAR incluye juegos de mesa corrientes como el Monopoly, pero el sistema se experimenta mejor con juegos más complejos

de estrategia o rol. En los juegos aumentados, el ordenador controla la lógica del juego. La lógica del juego puede hacer cumplir las reglas del juego, computar simulaciones complejas en el mundo del juego, realizar un seguimiento de los inventarios de los jugadores y realizar otras tareas rutinarias.

Algunos de los componentes de la plataforma STARS son:

1) Mesa. Proporciona el tablero de juego que se muestra en la superficie. Proporciona una interfaz tangible con piezas de juego ordinarias cuya posición se sigue mediante una cámara aérea. Actualmente se usa la interacTable, que es un componente Roomware© [Stre01] que caracteriza una superficie sensible al tacto usada para gestos y menús. Se usa un visualizador de plasma embebido en la mesa y una cámara montada sobre la mesa para detección adicional. Este enfoque tiene la ventaja de proporcionar una imagen de alta calidad en la superficie de la mesa, sin importar las condiciones de luz.

2) Visualizador. Se usa para mostrar información pública relevante del juego que cada jugador puede ver en cualquier momento. En él se pueden reproducir videos o animaciones que son disparadas por un jugador o por la lógica de la aplicación software STARS.

3) Personal Digital Assistants (PDAs). Cada jugador puede traer una PDA para conectarla con los otros componentes STARS. La PDA se puede usar para tomar notas privadas y para datos privados administrativos o para canales de comunicación privada que permiten a la lógica del juego STARS y a los jugadores enviar mensajes privados a otros jugadores.

4) Dispositivos de audio. STARS proporciona un módulo de generación y de reconocimiento del habla con múltiples canales de entrada y salida basados en la API Microsoft Speech. Dependiendo del tipo de juego, los jugadores llevan auriculares para recibir mensajes privados generados por la lógica de juego de STARS o por otros jugadores. Un altavoz público se usa para emitir muestras de audio ambiente o música para subrayar la acción de la mesa de juego. Se pueden usar auriculares para órdenes verbales a la lógica del juego. El reconocimiento del habla está basado actualmente en una lista de ordenes XML que permite variaciones del lenguaje natural.

Para diferentes necesidades de interacción, unos medios y modalidades son más adecuados que otros. En STARS, el ordenador ayuda a encontrar la composición óptima de modos.

En cuanto a la interacción del usuario, el gestor de interacción es el componente software más importante de STARS. Mapea las peticiones de interacción desde el nivel más alto de la lógica del juego al nivel más bajo de los servicios de la interfaz disponibles para un modo. Al hacer esto, tiene en cuenta pistas dentro de las

peticiones de interacción sobre las características que el modo debería tener. Proporciona también una interfaz de programación flexible y de alto nivel que permite a las aplicaciones de juegos de STARS formular peticiones de interacción que son fáciles de usar e independientes de modo y al mismo tiempo flexibles permitiendo especificar diferentes requisitos cada modo implicado.

Los componentes lógicos del sistema se distribuyen entre cuatro capas de abstracción decreciente. La primera capa y la más abstracta es la capa de aplicación. Implementa funcionalidad específica de cada aplicación de juegos que se ejecuta en la plataforma STARS. Incluye la lógica del juego, es decir, el conjunto de reglas necesarias para definir un juego, así como el material gráfico y los recursos de interacción de la base de datos de recursos. Toda funcionalidad adicional no cubierta en una capa más profunda debe implementarse en esta capa.

En la capa del motor de juego se localizan ciertos conjuntos de herramientas comunes para la mayor parte de los juegos de tablero para permitir una mayor abstracción en la capa de aplicación. Por ejemplo, el tablero de juego se representa con soporte de scripting básico y plantillas de definición de objetos. También, al construir sobre la definición de usuarios presente en las sucesivas capas de motor principal, los jugadores participantes se administran aquí con propiedades comunes a la mayoría de juegos para mesas táctiles como contadores de puntuación o un inventario de los objetos de juego portados.

La capa del motor principal incluye funcionalidad principal relacionada con la configuración hardware de STARS que no está restringida a las aplicaciones de juegos, sino que tiene un rango más amplio. Además del gestor de sesión y el soporte de registro, el gestor de interacción es el principal componente de esta capa. Esta indirectamente relacionada con el gestor de usuarios, que proporciona las asociaciones entre usuarios y dispositivos. El gestor de usuarios se consulta cuando una aplicación lanza una interacción privada para un usuario específico y devuelve los dispositivos asociados con el usuario. Estas asociaciones se establecen actualmente de forma explícita.

En la capa de dispositivos se administran los componentes hardware de STARS. Esto incluye integración dinámica, enumeración de servicios y abstracción de características. Para cada dispositivo, existe una envoltura que implementa los servicios de interfaz disponibles que se llaman mediante el gestor de interacción. Los dispositivos se registran en el bróker de dispositivos que es el responsable de administrar la comunicación entre un dispositivo y STARS, u otras aplicaciones en nuestro entorno de computación ubicuo.

2.5 Tabla Comparativa

En este capítulo se realiza una comparación más sistemática de los trabajos anteriormente descritos, en base a una serie de características y dimensiones.

Antes de comentar dichas características una por una, hemos de clarificar que a partir de este punto utilizaremos el término ecosistema, o ecosistema de juegos para referirnos a un juego compuesto por varios elementos capaces de mostrar un comportamiento determinado. Más adelante daremos una definición más precisa de este término.

A continuación, definimos cada una de dichas características de los sistemas:

1) Objetivo del sistema. Representa la función primaria para la que se creó el sistema. Por ejemplo, aprendizaje de lenguajes de programación, entretenimiento, etc.
   En la tabla comparativa que se presenta más adelante se utiliza la siguiente leyenda:
   A = Sistemas cuyo objetivo es el aprendizaje
   E = Sistemas cuyo objetivo es el entretenimiento
   S = Sistemas cuyo objetivo es el sketching de juegos
2) Herramientas de autoría. Indica si el sistema permite la edición de ecosistemas, y en caso afirmativo, si el sistema soporta la creación de nuestras propias entidades o únicamente nos permite agregar al ecosistema entidades predefinidas. En este último caso, el sistema seria sólo de consumo de contenidos. Esto es un aspecto importante dado que bajo la perspectiva de Abt [Abt70], la tarea de especial relevancia es la creación de artefactos para el juego. Consumir sólo contenidos predefinidos es útil como medio de transmisión de conocimientos y habilidades. Sin embargo, la existencia de herramientas de autoría que soporten la creación de entidades propias puede impulsar la adecuación a una mayor variedad de actividades tal y como reclaman McFarlane et al. [McF02] y Gros [Gro04] [Gro07] en sus respectivos estudios con software predefinido.
3) Público objetivo. Indica el público para el que está pensado el sistema, es decir, el público al que va dirigido el sistema.
4) Estudio/Experimento. Indica si se realizó un estudio o experimento del sistema y qué tipo de evaluación se realizó. Además de construir un sistema de acuerdo a una serie de teorías cognitivas y/o sociales, es interesante que estos sean evaluados en cierto grado para validar que algunos de los supuestos sobre los que se sustentan o alguno de los objetivos para los cuales fueron construidos se soportan.
5) Forma de interacción social. Indica cómo se relacionan entre sí los usuarios del sistema.

6) Colaboración. Indica si el sistema es colaborativo, es decir, si los usuarios del sistema pueden colaborar en tiempo real, presencialmente o en red, en la creación o simulación del ecosistema.

Las tres siguientes propiedades se refieren a las capacidades de simulación de ecosistemas que tienen los distintos sistemas:

7) Ecosistema Virtual/Tangible. Indica si el ecosistema construido está formado únicamente por elementos virtuales, únicamente por elementos tangibles, o es un ecosistema formado por elementos virtuales y elementos tangibles. En el caso de que sea un ecosistema formado por entidades virtuales se indica si dichas entidades son entidades 2D o 3D.
8) Especificación de comportamiento. Indica el tipo de modelo que se utiliza para indicar cómo queremos que se comporte el ecosistema en la simulación. Es decir, si para realizar la simulación se utilizan procedimientos, reglas de reescritura, etc.
9) Soporte tecnológico simulación. Indica la tecnología que utiliza el sistema para la simulación del ecosistema. Es decir, si para la simulación se utilizará un PC de escritorio o una mesa táctil, o si se usará RFID, Bluetooth, visión por computador, etc.

Las tres siguientes propiedades se refieren a las capacidades de edición de ecosistemas que tienen los distintos sistemas:

10) Soporte tecnológico editor. Indica la tecnología que utiliza el sistema para la edición del ecosistema, es decir, para crear entidades propias y añadirlas al ecosistema o añadir entidades predefinidas al ecosistema.
11) Tipo de modelo de ecosistema. Indica qué tipo de entidades componen el ecosistema. Por ejemplo, si las entidades del ecosistema son agentes como en Agentsheets, o son bloques físicos como en Algoblock, Quetzal/Tern, y Topobo o sprites como en Scratch.
12) Construcción del comportamiento. Indica la forma en la que se edita el ecosistema: cómo se crean las entidades del ecosistema, ya sean nuestras propias entidades o entidades predefinidas, y cómo se añaden al mundo o ecosistema. Por ejemplo, se puede editar el ecosistema mediante la conexión de bloques físicos, arrastrando elementos de una paleta al espacio de trabajo, colocando tangibles en una mesa táctil o programando una maquina de estados.

	AlgoBlock	LogoBlocks	Cleogo	AgentSheets	The Tangible Programming Space	Teaching Table	StoryTelling Alice	Scratch
Objetivo del sistema	A	A	A	A	A	A	A	A
Herramientas de autoría	Si. Entidades predefinidas	Si. Entidades predefinidas	No	Si. Entidades propias	Si. Entidades predefinidas	No	Si. Entidades predefinidas	Si. Entidades propias
Público al que va dirigido	Estudiantes	Programadores del "Programmable Brick"	Estudiantes	Cualquiera	Niños (de 6 a 12 años)	Niños de 3 a 5 años	Programadores novatos	Niños
Estudio/Experimento	Si. Grupo de 3 niños de 12 años	Proof-of-concept	Proof-of-concept	Proof-of-concept	Proof-of-concept	Proof-of-concept	Si. 200 chicas de Middle School (entre 11 y 15)	Proof-of-concept
Forma de interacción social	Presencial	-	En red	-	Presencial	-	-	-
Colaboración	Si	No	Si	No	Si	No	No	No
Simulación/Interpretación								
Ecosistema Virtual/Tangible	Tangibles	Virtual 2D	Virtual 2D	Virtual 2D	Virtual 2D	Tangibles	Virtual 3D	Virtual 2D
Especificación de comportamiento	Procedimientos	Procedimientos	Procedimientos	Reglas de reescritura gráficas y métodos	Objetos y comportamientos prefijados	Procedimientos	Procedimientos	Procedimientos (Operaciones prefijadas y parámetros predefinidos)
Soporte tecnológico simulación	Desktop PC	Desktop PC, "Programmable Brick"	Desktop PC	Desktop PC, Java Applets	Desktop PC, pantalla de proyección	Tabletop (Tecnología de detección electromagnética)	Desktop PC	Desktop PC

Edición								
Soporte tecnológico editor	-	Desktop PC	-	Desktop PC, Herramientas Java, hojas de cálculo y agentes programables	RFID	-	Desktop PC	Desktop PC
Tipo de modelo de ecosistema	Bloques	Robots (motores y sensores)	Tortuga y órdenes	Agentes	Objetos y comportamientos	Objetos etiquetados	Objetos, escenarios y eventos	Objetos(Sprites)
Construcción del comportamiento	Conexión bloques físicos	Drag and drop de bloques de la paleta al espacio de trabajo	3 paradigmas de programación (texto, iconos, manipulación directa)	Drag and drop de elementos gráficos	Cartas de comportamiento y bloques creadores	Módulo Java de actividad (controla las actividades educacionales y la salida visual)	Drag and drop de fichas	Drag and drop de piezas de puzzle

A = Aprendizaje E = Entretenimiento S = Sketching de juegos

	Quetzal y Tern	TViews Table RPG	TurTan	IncreTable	Topobo	Raptor	STARS	Knight's Castle
Objetivo del sistema	A	E	A	E	A	S	E	E
Herramientas de autoría	Si. Entidades predefinidas	Si. Entidades predefinidas	No	Si. Entidades predefinidas	Si. Entidades predefinidas	Si	No	Si. Entidades predefinidas
Público al que va dirigido	Estudiantes	3 jugadores de RPG (aprox. desde 14 años en adelante)	Niños y no programadores	Cualquiera	Niños	Diseñadores y programadores de videojuegos	Programadores de juegos	Niños
Naturaleza del Estudio/Experimento	9 niños de 1º y 2º curso	Proof-of-concept	Proof-of-concept	Proof-of-concept	Cinco casos de estudio	Proof-of-concept	Proof-of-concept	Proof-of-concept
Forma de interacción social	Presencial	Presencial	Presencial	Presencial	Presencial	Presencial	Presencial	Presencial
Colaboración	Si	Si	Si	Si	Si	Si	Si	Si
Simulación/Interpretación								
Ecosistema Virtual/Tangible	Tangibles	Virtual 2D y Tangibles	Virtual 2D y Tangibles	Virtual 3D y Tangibles	Tangibles	Virtual 3D y Tangibles		Tangibles
Especificación de comportamiento	Procedimientos	Procedimientos	Procedimientos	Procedimientos	Programación por demostración	Procedimientos	Procedimientos	Maquina de estados
Soporte tecnológico simulación	Estación portable de escaneo (Visión por computador)	TViews tabletop	Tabletop	Tabletop, Lápiz digital, Portales, Robot	Componentes pasivos(estáticos) y activos (motorizados)	Tabletop, Desktop PC	Tabletop, PDA's, dispositivos de audio	RFID, Bluetooth, teléfonos móviles

Edición								
Soporte tecnológico editor	-	Desktop PC	-	Tabletop, Lápiz digital	-	Tabletop	-	-
Tipo de modelo de ecosistema	Bloques	Personajes jugadores, Personajes no jugadores, escenarios, objetos.	Tortuga y tangibles con órdenes (instrucciones, modificadores y tarjetas)	Objetos físicos o digitales, robots y portales	Bloques físicos	Personajes, objetos, cámaras y escenarios		Personajes y edificios
Construcción del comportamiento	Conexión bloques	Tres modos diferentes: selección de personajes juego libre y lucha.	Drag and drop de tangibles en la mesa táctil	Colocación de tangibles en el tabletop	Conexión bloques físicos	Colocación de tangibles en el tabletop		Programación de la máquina de estados

A = Aprendizaje E = Entretenimiento S = Sketching de juegos

	AlgoBlock	The Tangible Programming Space	StoryTelling Alice	Quetzal y Tern	Incretable	Knight's Castle	LogoBlocks	AgentSheets	Scratch	Topobo	Cleogo	Turtan	Teaching Table	TViews Table RPG	Raptor	STARS
Objetivo del sistema																
Aprendizaje	X	X	X	X			X	X	X	X	X	X	X			
Entretenimiento					X	X								X		X
Sketching de juegos															X	
Herramientas de autoría																
Permite edición y creación de entidades							X	X	X	X						
Permite edición pero no creación de entidades	X	X	X	X	X	X										
No permite edición											X	X	X	X		
Permite construir otros sistemas															X	X
Colaboración																
Colaborativo (presencial)	X	X			X	X			X	X		X		X	X	X
Colaborativo (en tiempo real y en red)											X					
No colaborativo			X	X			X	X					X			
Ecosistema Virtual/Tangible																
Virtual 2D		X					X	X	X		X					
Virtual 3D			X													
Virtual y tangibles					X							X		X	X	
Tangibles	X			X		X				X			X			
Especificación del comportamiento																
Procedimientos	X	X	X	X	X	X	X		X	X	X	X		X	X	X
Programación por													X			

demostración																
Reglas de reescritura gráficas								X								
Maquina de estados						X										
Soporte tecnológico																
Tabletop					X							X	X	X	X	X
Desktop PC	X	X	X			X	X	X	X		X				X	

A modo de conclusión de la comparación entre sistemas realizada, podemos dividir los sistemas comparados en cuatro tipos:

1) Sistemas que nos permiten crear nuestras propias entidades y añadirlas al ecosistema, además de soportar la programación del comportamiento por parte del usuario final. Ejemplos de estos sistemas son Scratch, Agentsheets, Topobo y Logoblocks.
2) Sistemas que no nos permiten crear las componentes o entidades a utilizar en el ecosistema, aunque nos permiten añadir entidades predefinidas a nuestro ecosistema y programar el comportamiento. Ejemplos de estos sistemas son: Algoblock, The Tangible Programming Space, Storytelling Alice, Quetzal y Tern, Incretable, The Augmented Knight's Castle.
3) Sistemas que no permiten la edición de ecosistemas. Ejemplos de estos sistemas son: Cleogo, Teaching table, Turtan, TViews Table RPG.
4) Sistemas que no entran en las categorías anteriores se caracterizan por que no están destinadas al usuario final, sino que sirven para construir otros sistemas por parte de usuarios avanzados o profesionales. Ejemplos de estos sistemas son: STARS, Raptor.

De acuerdo a la lista de características de los sistemas, el sistema que pretendemos construir debería ser un sistema cuyo objetivo fuera el aprendizaje, que tuviera herramientas de autoría y permitiese crear nuestras propias entidades, un sistema dirigido a estudiantes, que permita una interacción social presencial y sea colaborativo, un sistema cuyas entidades sean virtuales 2D y permita el uso de tangibles, que utilice procedimientos para indicar el comportamiento del sistema en simulación. Dicho sistema debería utilizar una mesa táctil para la simulación y la edición y debería permitirnos realizar la edición del ecosistema mediante la colocación de tangibles en la mesa táctil.

Capitulo 3. Modelo para la construcción de ecosistemas

3.1 Introducción

En este capítulo describiremos el modelo propuesto que nos permita la creación y simulación de ecosistemas con el objetivo final de soportar determinadas actividades de aprendizaje creativo sobre superficies interactivas.

Necesitamos un modelo que soporte la creación y simulación de ecosistemas y que se adhiera a las ideas de Clark Abt [Abt70], en las que se utilizan los juegos serios para soportar aprendizaje creativo. Abt considera el jugar a juegos no sólo como medio para aprender, sino que considera también la etapa de creación del juego como parte del propio juego, y hace énfasis en la relevante importancia que esta etapa tiene en el aprendizaje en relación al resto de fases del juego.

El modelo propuesto nos permitirá crear y simular ecosistemas y servirá de base para la futura implementación del sistema general. Hemos de tener en cuenta que se desea que el sistema construido cumpla los postulados de las teorías de aprendizaje constructivista, tales como el aprendizaje experiencial [Kol94] [Dew63] , y el aprendizaje situado [Bro89] , en las cuales "aprender es hacer" y que consideran también la reflexión y la discusión como procesos necesarios de suma importancia más allá de la acción para aprender de forma efectiva. De la misma forma, también se desea que el sistema construido cumpla los postulados del aprendizaje social que considera que el conocimiento emerge de la interacción y la comunicación entre individuos [Vyg78]. Además, hemos de tener en cuenta que queremos que el sistema funcione en superficies interactivas, ya que dichas superficies permiten un aprendizaje más social debido a que fomentan características como la colaboración, la cooperación y una comunicación cara a cara entre sus usuarios, al mismo tiempo que incrementan el nivel de motivación de los usuarios [Hor08].

En el resto del capítulo se describen los detalles del modelo propuesto.

3.2 Modelo de Ecosistema

En primer lugar, definimos ecosistema como un conjunto de entidades interactivas, basadas en un modelo simple de relaciones acción-reacción y causa-efecto, que permiten exhibir comportamiento, y que interaccionan entre ellas y con las acciones de los participantes en el juego desarrollado.

Otra definición de ecosistema podría ser, el lugar donde varias entidades habitan, interactuando entre ellas, y llevando a cabo sus acciones con el propósito de alcanzar sus objetivos.

Un ecosistema se compone de entidades, debido a que las entidades nos permiten tener diferentes elementos con comportamiento, y esto nos permite cumplir lo indicado por [Abt70] referido en el punto 3.1, de poder crear elementos del juego y simularlos posteriormente. Además, las entidades son una abstracción adecuada ya que permiten la especificación de comportamiento físico centrada en las entidades, pudiendo centrarnos en la definición de entidades individualmente.

Un ecosistema se describe en base a escenarios, en los que participan entidades, y dichas entidades se construyen en base a definiciones de tipos existentes en el ecosistema, que tienen dos aspectos: la descripción física y la descripción lógica, que vienen a controlar el comportamiento en la simulación en base a físicas o en base a reglas, respectivamente.

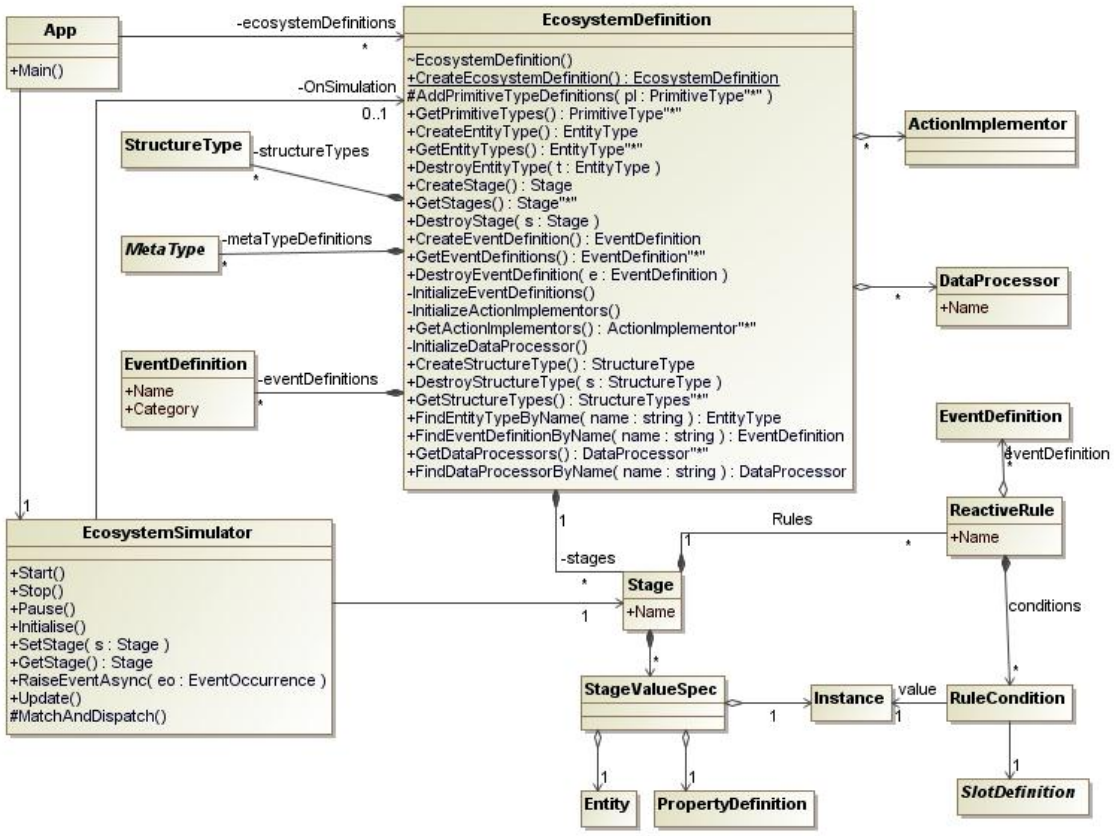

Figura 1. Clases principales del modelo

En la Figura 1, la clase *App* representa la clase principal de la aplicación, en la que podemos tener varias definiciones de ecosistemas (representadas en la clase *EcosystemDefinition*) y un simulador de ecosistema (representado en la clase *EcosystemSimulator*). Una definición de ecosistema contendrá definiciones de eventos

(*EventDefinition*), definiciones de tipos de datos (*Metatype*), definiciones de tipos de estructura (*StructureType*), y de escenarios de simulación del ecosistema (*Stage*).

Definimos evento como un suceso que queremos capturar cuando se produzca en nuestro ecosistema. En cuanto a la definición de tipo de estructura, se definirá más adelante.

La clase *Stage* representa los escenarios en los cuales se puede realizar la simulación del ecosistema. Estos escenarios podrán tener definidas varias reglas (*ReactiveRule*).

La definición de las reglas (*ReactiveRule*) implica definir con qué evento se va a disparar la regla, la población fuente que puede disparar dicha regla y la población destino sobre la que se aplicará la operación o acción del consecuente de la regla.

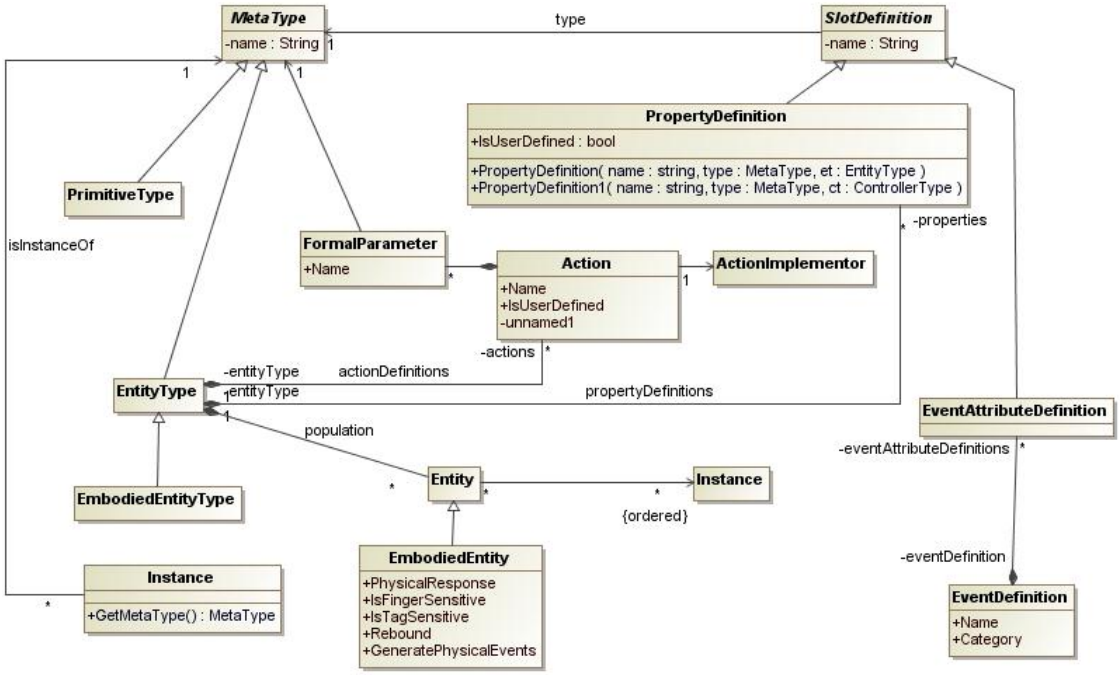

Figura 2. Modelo para la definición de tipos de datos

La clase *MetaType* da cabida a los diferentes tipos de datos que podemos tener en el ecosistema. *MetaType* se especializa en tres subclases: la clase *PrimitiveType* permite la representación de tipos de datos primitivos, como por ejemplo un entero, un booleano, una cadena de caracteres, etc.; la clase *EntityType* permite representar a un tipo de entidad.

Un tipo de entidad (*EntityType*) se caracteriza por estar compuesto de un conjunto de propiedades (*PropertyDefinition*), que pueden ser predefinidas o definidas por el usuario, y de un conjunto de acciones (*Action*) cada una de las cuales puede tener parámetros formales (*FormalParameter*) y que se implementa en las instancias de la clase *ActionImplementor*.

La clase *Entity* representa una instancia de un tipo de entidad. La clase *Instance* representa el concepto genérico de instancia de un objeto *Metatype*, es decir, una instancia de un tipo primitivo o de un tipo de entidad.

Mediante esta clase *Instance* y mediante su especialización, el modelo permite, por tanto, no solo la definición de instancias, sino que dichas instancias sean conformes a la descripción de los tipos de entidad que se han definido anteriormente.

Como hemos dicho antes, un evento es un suceso que queremos capturar cuando se produzca en nuestro ecosistema. Los eventos pueden tener atributos, que son parámetros que contienen información relevante del evento que se ha producido. En el modelo, las clases *EventDefinition* y *EventAttributeDefinition* nos permiten la especificación de tipos de eventos y la definición de atributos para los eventos especificados, respectivamente.

3.2.1 Comportamiento basado en leyes físicas

En la Figura 2 podemos ver que como subclase de *EntityType* tenemos la subclase *EmbodiedEntityType*, que representa a los tipos de entidad cuyas instancias pueden simularse en base al motor de físicas. En nuestro caso, utilizaremos el motor de físicas Farseer[2]. A su vez, la clase *Entity* se especializa en la subclase *EmbodiedEntity* que representa a las entidades que pueden ser simuladas en base al motor de físicas. Por tanto, estas subclases tendrán una representación visual en simulación y un comportamiento físico.

Los tipos de entidad que pueden ser simulados en base al motor de físicas (*EmbodiedEntityType*) tienen asociado un tipo de estructura (*StructureType*) que contiene la representación visual del tipo de entidad mediante formas geométricas básicas unidas por articulaciones y unas vestimentas (*ProtoCostume*) que representan las texturas que se le aplican a cada componente del tipo de estructura y que determinan el aspecto visual del tipo de entidad. El tipo de estructura podría verse como una especie de “esqueleto” del tipo de entidad.

El tipo de estructura (*StructureType*) se define mediante una serie de formas geométricas o componentes (*StructuralComponent*) unidos mediante articulaciones (StructuralJoint). La clase *StructuralComponent* tiene las siguientes subclases: *RectangleStructuralComponent* que representa un componente rectangular, *EllipseStructuralComponent*, que representa un componente con forma elíptica, y *PolygonStructuralComponent*, que representa un componente con forma poligonal. Cada *StructuralComponent* se relaciona con un objeto *Body* en el motor de físicas Farseer.

[2] Farseer Physics Engine: http://farseerphysics.codeplex.com

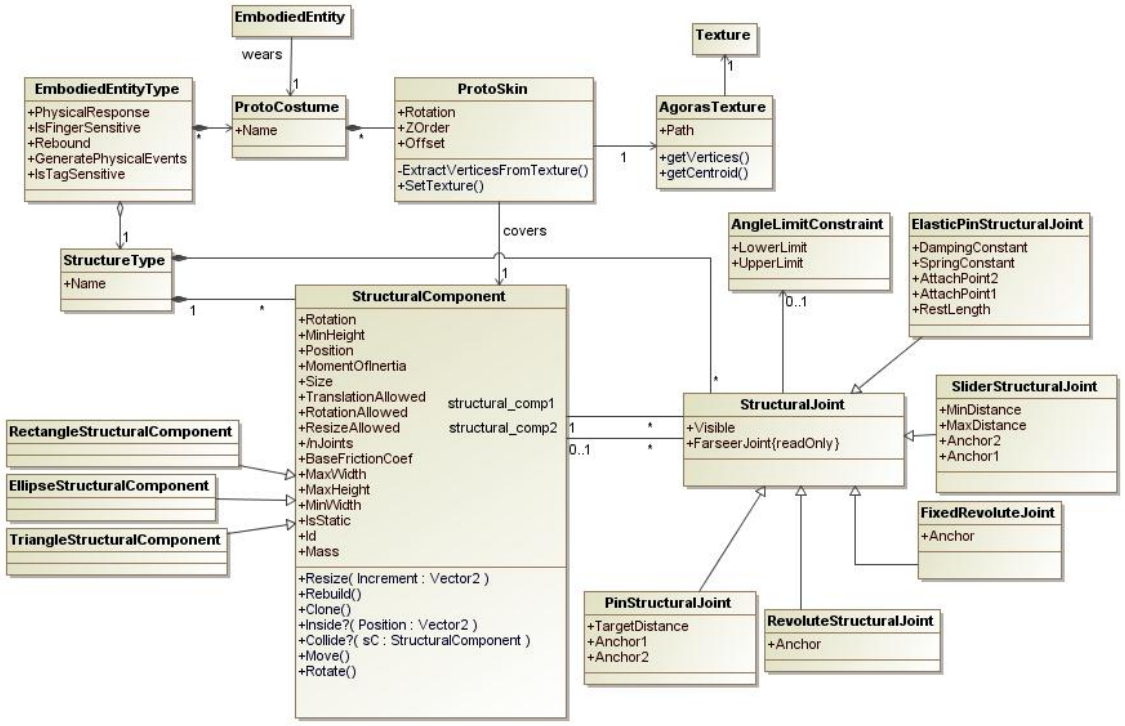

Figura 3. Modelo para la definición de los tipos de entidad

La clase *StructuralComponent* contiene atributos que determinan su comportamiento físico, como por ejemplo:

1) IsStatic. Determina si el componente es estático.
2) Rotation. Determina la rotación del componente.
3) Mass. Determina la masa del componente.
4) Position. Determina la posición del componente.
5) MomentOfInertia. Determina el momento de inercia del componente.
6) Size. Determina el tamaño del componente.
7) RotationAllowed. Determina si el componente permite la rotación.
8) TranslationAllowed. Determina si el componente permite la translación.
9) ResizeAllowed. Determina si el componente permite el reescalado.
10) BaseFrictionCoef. Representa el coeficiente base de fricción.

Las articulaciones (*StructuralJoint*) se utilizan para unir dos *StructuralComponent* y pueden ser de varios tipos (*PinStructuralJoint*, *RevoluteStructuralJoint* o *ElasticStructuralJoint*) según cómo deseemos que se comporte la articulación. Las articulaciones pueden tener asociado un ángulo máximo o mínimo entre el componente y la articulación (*AngleLimitConstraint*). Cada *StructuralJoint* se relaciona con un objeto *Joint* en el motor de físicas Farseer.

La clase *StructuralJoint* tiene el atributo Visible, que determina si es visible o no la articulación y el atributo de solo lectura *FarseerJoint* que representa la articulación del motor de físicas Farseer con la que está relacionado.

En la Figura 4 tenemos dos ejemplo de tipos de estructura simples, uno con forma circular y otro con forma rectangular.

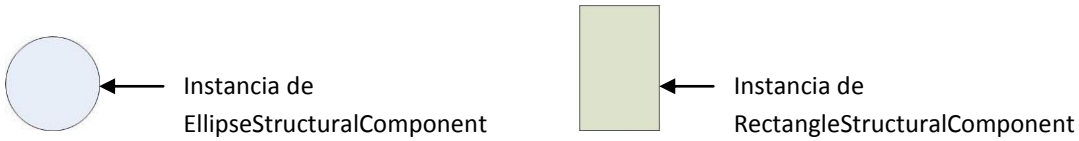

Figura 4. Instancias de tipos de estructura circular y rectangular

En la Figura 5 vemos otro ejemplo de tipo de estructura un poco más complejo que representa una plataforma con un trampolín.

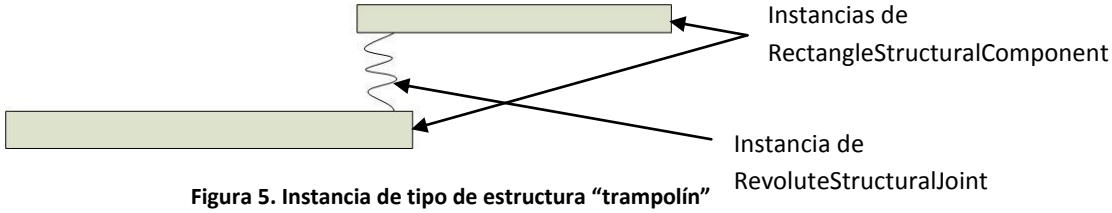

Figura 5. Instancia de tipo de estructura "trampolín"

El ejemplo de la Figura 5 estaría formado por dos *StructuralComponent* rectangulares (*RectangleStructuralComponent*) unidos por una articulación de tipo muelle (*RevoluteStructuralJoint*).

La Figura 6 representa un tipo de estructura compleja "marioneta" compuesto por seis *StructuralComponent* (un *EllipseStructuralComponent* para representar el circulo de la cabeza y cuatro *RectangleStructuralComponent* para representar los brazos y las piernas y otro *RectangleStructuralComponent* para representar el torso) y cinco *PinStructuralJoint* (uno para conectar la cabeza con el torso, dos para conectar los brazos con el torso y otros dos para conectar las piernas con el torso).

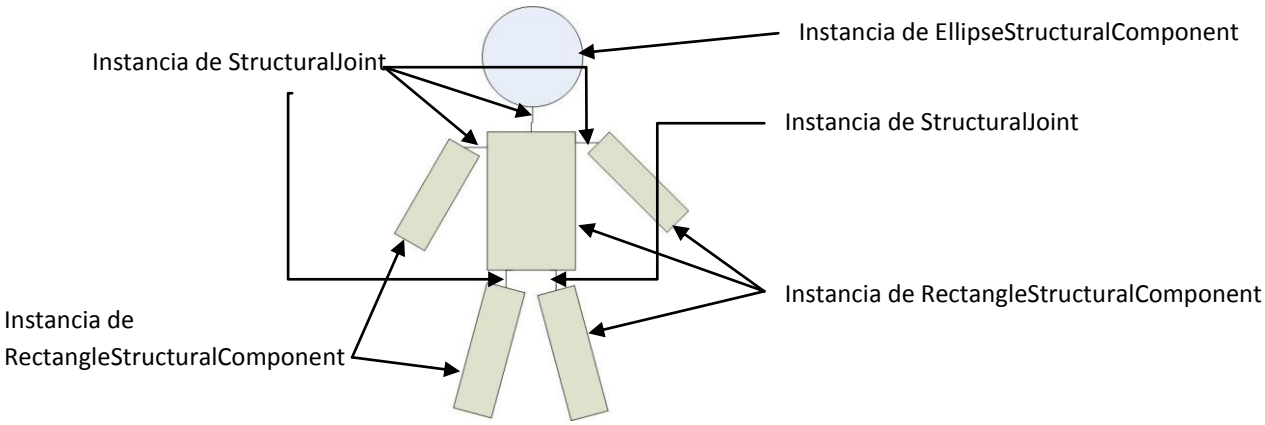

Figura 6. Instancia del tipo de estructura "marioneta"

Las vestimentas a nivel de tipo de entidad (*ProtoCostume*) se definen asociando una textura (*Protoskin*) a cada componente (*StructuralComponent*) del tipo de estructura.

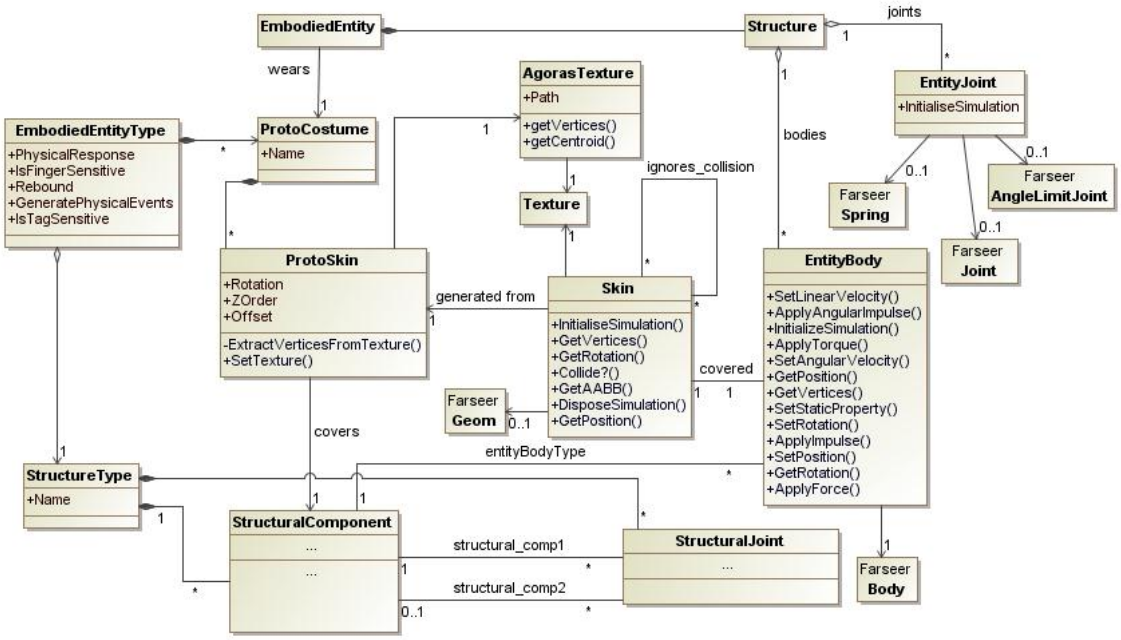

Figura 7. Modelo para la definición de las entidades en simulación

En la Figura 7 podemos ver el modelo anterior con algunas clases adicionales que nos permiten soportar la instanciación en tiempo de ejecución. A nivel de las entidades, la estructura de la entidad se mantiene en la clase *Structure* que está formada por componentes *EntityBody* unidos por articulaciones *EntityJoint*.

Los *EntityBody* y *EntityJoint* son los equivalentes a un *StructuralComponent* y un *StructuralJoint* respectivamente pero a nivel de entidad. Los componentes *EntityBody* pueden tener asociada una textura *Skin* que es el equivalente de *ProtoSkin* pero a nivel de entidad.

Hemos visto que necesitamos una clase *StructureType* que represente la estructura de los tipos de entidad y una clase *Structure* que represente la estructura de las entidades. La razón de tener dos clases distintas para las estructuras de las entidades y de los tipos de entidad es poder dar soporte para creación y soporte para simulación.

Esta misma "dualidad" ocurre con las clases *StructuralComponent* y *EntityBody*, que representan a un componente a nivel de tipo de entidad y a nivel de entidad, respectivamente, y *StructuralJoint* y *EntityJoint*, que representan a una articulación a nivel de tipo de entidad y a nivel de entidad, respectivamente.

Un ejemplo básico que evidencia y explica la necesidad de esta dualidad resulta en el supuesto que creemos un tipo de entidad con un tipo de estructura compuesto por un solo componente con forma triangular para representar una nave espacial en un juego. Supongamos que en la simulación utilizamos dos naves (ver Figura 8) que son instancias de este tipo de entidad. Debido a que en la simulación cada nave tendrá una

posición diferente, necesitamos que la posición se guarde a nivel de entidad y que, por lo tanto, cada nave esté representada por un *EntityBody* diferente.

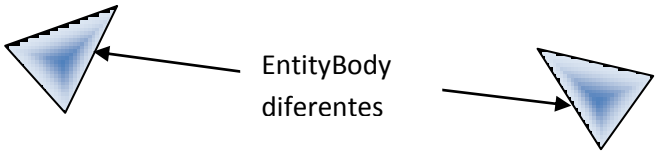

Figura 8. Ejemplo de juego de naves espaciales

Otro ejemplo para explicar la necesidad de esta dualidad se produce si tenemos dos naves espaciales en la simulación, y queremos cambiar la vestimenta de una de las naves respecto a la del tipo de entidad. Para ello necesitaremos que exista una vestimenta en el tipo de entidad y otra vestimenta distinta a nivel de entidad, para que al realizar los cambios en la vestimenta de la entidad no afecte a la vestimenta de la otra nave. No obstante, estas vestimentas en tiempo de ejecución se derivan de las vestimentas elegidas sobre la estructura correspondiente.

La Figura 9 representa un ejemplo de instancia de una entidad con el tipo de estructura "Marioneta". Esta instancia de entidad estará compuesta por seis *EntityBody* (uno para representar la cabeza y cuatro para representar los brazos y las piernas y otro para representar el torso) y cinco *EntityJoint* (uno para conectar la cabeza con el torso, dos para conectar los brazos con el torso y otros dos para conectar las piernas con el torso).

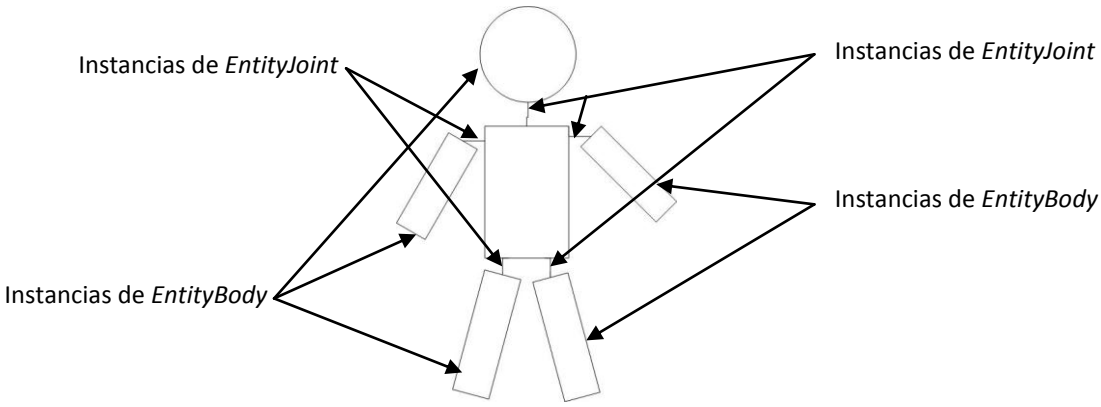

Figura 9. Instancia de una entidad con el tipo de estructura "marioneta"

La Figura 10 representa un ejemplo de un *protocostume* para el tipo de entidad de la figura 8. Para la cabeza se ha aplicado un skin con las facciones de la cara (Figura 11 (a)), y para las manos (Figura 11(b)), piernas (Figura 11(c)) y torso (Figura 11(d)) se han aplicado otros skins.

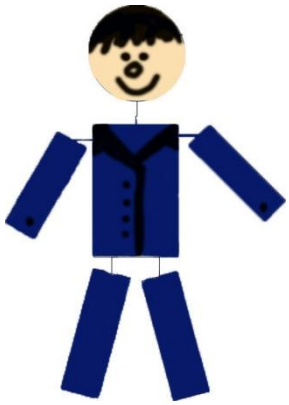

Figura 10. Instancia de un protocostume con el tipo de estructura "marioneta"

(a)Skin1 **(b)Skin2** **(c)Skin3** **(d)Skin4**

Figura 11. Ejemplos de skins

3.3 Modelo de ejecución y simulación

Este apartado trata del modelo de ejecución y simulación que utiliza nuestro simulador y en él se explican los distintos módulos que componen la arquitectura del simulador.

3.3.1 Arquitectura del simulador

Existen tres módulos principales que intervienen y posibilitan el proceso de la simulación:

1) El módulo de definición del ecosistema se encarga de crear las instancias de las clases de nuestro modelo a partir de una definición del ecosistema que se encuentra en un fichero de especificación .eco. La funcionalidad de este módulo se implementa principalmente en la clase *EcosystemDefinition*.
2) El módulo de simulación contiene una referencia al motor de físicas Farseer y se encarga de traducir los elementos de nuestro modelo con un comportamiento físico, en geometrías con las que trabaja el motor de físicas Farseer. Además, este módulo proporciona una API para manipular y simular los elementos de nuestro ecosistema. Dicha funcionalidad se implementa en la clase *EcosystemView*.
3) El módulo de orquestación se encarga de realizar la coordinación u orquestación para capturar los eventos y con ello hacer progresar el sistema. Para ello, utiliza la funcionalidad que le proporcionan los módulos anteriores junto con un procesador de matching de eventos que se encarga de controlar el disparo de las reglas reactivas definidas en el escenario. Dicha funcionalidad se implementa principalmente en la clase *EcosystemSimulator*.

En la Figura 12 podemos ver un esquema resumen de la arquitectura del simulador con los módulos involucrados en el proceso de simulación.

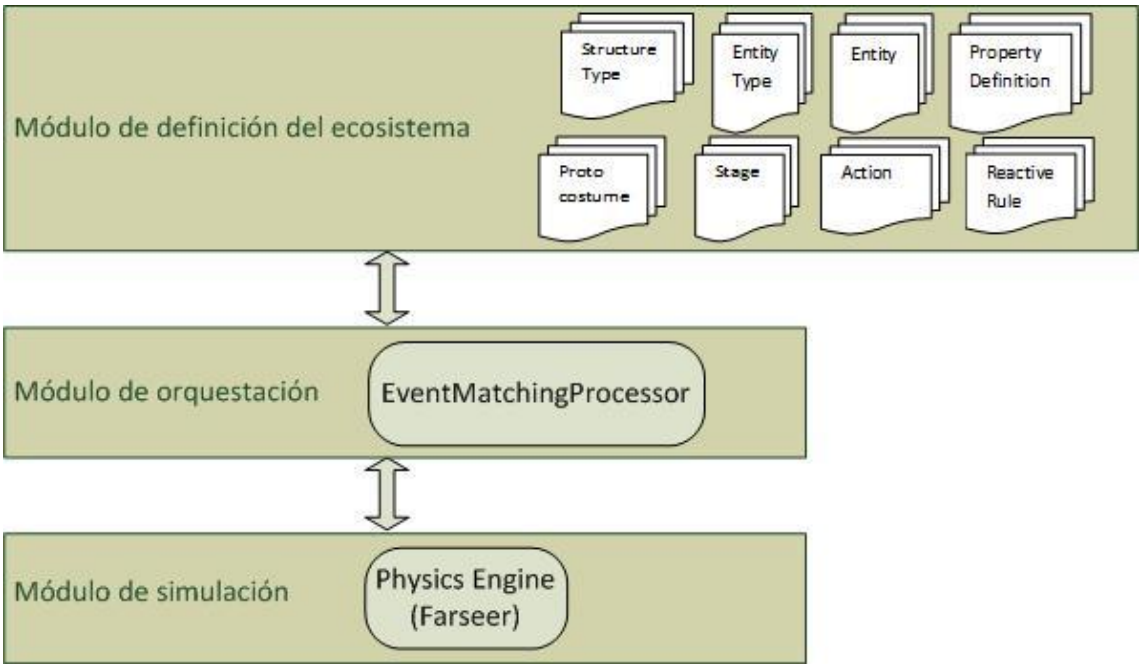

Figura 12. Arquitectura del simulador

A continuación, pasamos a comentar en profundidad cada una de las partes del simulador.

3.3.1.1 Módulo de definición del ecosistema

El módulo de definición del ecosistema nos permite realizar la instanciación del modelo partiendo de la definición del ecosistema que se encuentra en un fichero de especificación .eco.

En este módulo se crean varios objetos poblando parte del meta-modelo para soportar tipos primitivos que usaremos en la definición del ecosistema. Los tipos primitivos básicos que el sistema soporta son: *Float* (números decimales con coma flotante), *Integer* (números enteros), *String* (cadenas de caracteres alfanuméricas), *Bool* (valores booleanos), *Vector2* (vectores de dos dimensiones), *GeneralEntityType* (tipo de entidad genérico) y *GeneralMetaType* (metatipo general).

GeneralEntityType es un tipo de datos genérico especial que representa a un tipo de entidad genérico. Se utiliza para poder definir atributos de eventos, propiedades y parámetros formales de acciones que pueden tomar como valor cualquier tipo de entidad.

GeneralMetaType es un tipo de datos especial que representa a un tipo de datos genérico. Se utiliza para poder definir atributos de eventos, propiedades y parámetros formales de acciones que pueden tomar como valor cualquier tipo de entidad o cualquiera de los otros tipos primitivos.

Este módulo contiene métodos para gestionar los *EntityTypes*, *EventDefinitions*, *ActionImplementors*, *DataProcessors*, *Stages* y *StructureTypes* del ecosistema.

Se crea un tipo de entidad especial, *StageType*, y su correspondiente entidad para representar al propio escenario de simulación del ecosistema. Esto se hace para poder añadir al escenario acciones y propiedades necesarias para la simulación. Existen unas propiedades predefinidas que sólo son propias del escenario. En la primera versión del prototipo implementado se ha considerado la propiedad *Gravity*, que permite controlar el vector gravedad del escenario.

De la misma forma, existen también una serie de acciones predefinidas que sólo son propias del escenario. La Figura 13 muestra el diagrama de instancias que muestra las acciones predefinidas de los escenarios y sus parámetros.

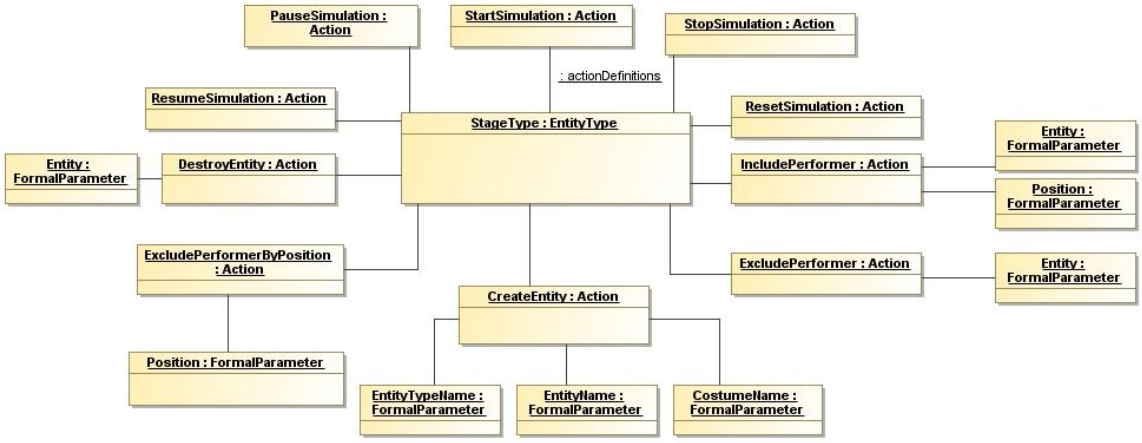

Figura 13. Acciones predefinidas de los escenarios

A continuación, presentamos cada una de estas acciones predefinidas y explicamos su función ayudándonos de pseudocódigo.

- StartSimulation: Inicia la simulación del ecosistema.

```
StartSimulation ()

BEGIN

 SIMULATOR.Run();

END;
```

- StopSimulation: Finaliza la simulación del ecosistema.

```
StopSimulation ()

BEGIN

 SIMULATOR.Stop();
```

```
 SIMULATOR.ResetEcosystem();

 SIMULATOR.Stop();

END;
```

Antes de parar el simulador, paramos y reseteamos el módulo de simulación y después limpiamos la cola del procesador de matching de eventos de ocurrencias de eventos y coincidencias.

- ResetSimulation: Reinicia la simulación del ecosistema.

```
ResetSimulation ()

BEGIN

 SIMULATOR.Stop();

 SIMULATOR.Run();

END;
```

- PauseSimulation: Pausa la simulación del ecosistema.

```
PauseSimulation ()

BEGIN

SIMULATOR.Pause();

END;
```

- ResumeSimulation: Continua la simulación del ecosistema después de una pausa.

```
ResumeSimulation ()

BEGIN

SIMULATOR.Resume();

END;
```

- CreateEntity: Crea una entidad de un determinado tipo y vestida con un determinado *ProtoCostume* y lo añade al ecosistema.

```
CreateEntity (EmbodiedEntityType type)

BEGIN

EmbodiedEntity newEntity = SIMULATOR.CreateInstance(type,name);

 SIMULATOR.Performers.Add(newEntity);
```

```
 THROW EVENTOCCURRENCE (InstanceCreated, newEntity);

 THROW EVENTOCCURRENCE (PerformerIncluded, newEntity);

END;
```

Esta acción lanza dos ocurrencias de evento, InstanceCreated y PerformerIncluded, que informan de que se ha creado una instancia de un tipo de entidad y de que se ha añadido una entidad a la simulación.

- DestroyEntity: Elimina del ecosistema una determinada entidad.

```
DestroyEntity (EmbodiedEntity entity)

BEGIN

SIMULATOR.DestroyInstance(entity);

 THROW EVENTOCCURRENCE (InstanceDestroyed, entity);

END;
```

Esta acción lanza la ocurrencia de evento InstanceDestroyed que informa de que se ha eliminado una entidad de la simulación.

- IncludePerformer: Añade una entidad ya creada a la simulación actual en una posición determinada.

```
IncludePerformer (Entity includedEntity)

BEGIN

SIMULATOR.Performers.Add(includedEntity);

 THROW EVENTOCCURRENCE (PerformerIncluded, includedEntity);

END;
```

Esta acción lanza la ocurrencia de evento PerformerIncluded que informa de que se ha añadido una entidad a la simulación.

- ExcludePerformer: Elimina una entidad determinada de la simulación sin eliminarla del ecosistema.

```
ExcludePerformer (Entity excludedEntity)

BEGIN

 SIMULATOR.Performers.Remove(excludedEntity);

THROW EVENTOCCURRENCE (PerformerExcluded, excludedEntity);

END;
```

Esta acción lanza la ocurrencia de evento PerformerExcluded que informa de que se ha eliminado una entidad a la simulación.

- ExcludePerformerByPosition: Elimina una entidad de la simulación sin eliminarla del ecosistema. Esta será aquella que se encuentre en una posición determinada.

```
ExcludePerformerByPosition (Vector2 position)

BEGIN

 Entity excludedEntity = GetEntityAtPosition(position);

 SIMULATOR.Performers.Remove(excludedEntity);

THROW EVENTOCCURRENCE (PerformerExcluded, excludedEntity);

END;
```

Esta acción lanza la ocurrencia de evento PerformerExcluded que informa de que se ha eliminado una entidad a la simulación.

El módulo de definición del ecosistema se encarga de crear los eventos predefinidos del sistema. La Figura 14 muestra el diagrama de instancias con los eventos predefinidos que automáticamente se introducen en la definición del ecosistema así como sus atributos.

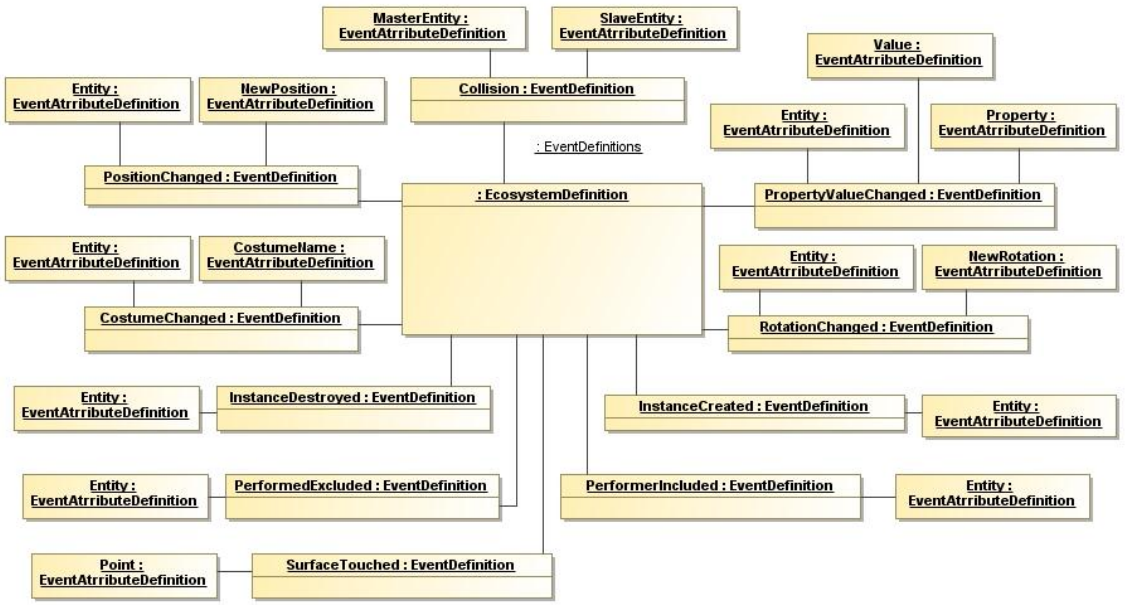

Figura 14. Definiciones de eventos predefinidas

A continuación, explicamos en qué consisten cada una de estas definiciones de eventos:

1. Collision: Colisión entre entidades
2. PropertyValueChanged: Cambio del valor de una propiedad
3. PositionChanged: Cambio de la posición de una entidad
4. RotationChanged: Cambio de la rotación de una entidad.
5. CostumeChanged: Cambio de la vestimenta de una entidad.
6. InstanceCreated: Creación de una entidad de un determinado tipo de entidad.
7. InstanceDestroyed: Se destruye una entidad determinada.
8. PerformerIncluded: Una entidad ha sido incluida en la simulación.
9. PerformedExcluded: Una entidad ha sido excluida de la simulación.

10. SurfaceTouched: El usuario pone el dedo sobre la Surface.

Para cada EntityType se añaden unas propiedades predefinidas. La Figura 15 muestra un diagrama de instancias que ilustra las propiedades predefinidas de los tipos de entidad y sus tipos.

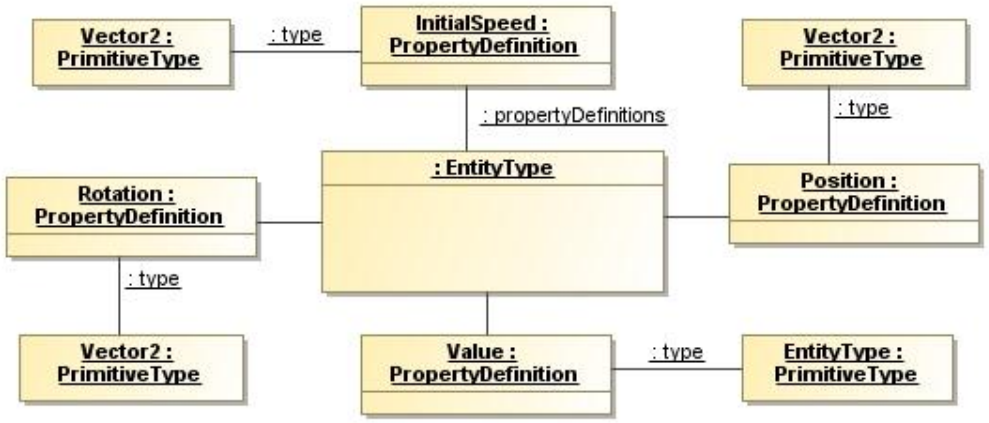

Figura 15. Propiedades predefinidas de los tipos de entidad

A continuación, explicamos cada una de estas propiedades predefinidas:

1. InitialSpeed: Velocidad inicial de la entidad.
2. Position: Posición de la entidad.
3. Rotation: Rotación actual de entidad.
4. Value: Instancia concreta de la entidad.

De estas propiedades, la propiedad *Value* es *Readonly*, es decir, el usuario no puede asignarle un valor. Su significado es de especial relevancia tal y como se explica a continuación. En esencia, las reglas utilizadas en nuestro sistema hacen uso de definiciones de propiedades de entidades o atributos de tipos de eventos. Estos elementos pueden utilizarse para componer las precondiciones y las postcondiciones de las reglas. Sin embargo, con el propósito de poder involucrar fácilmente a las instancias de entidades directamente en la especificación de las reglas, necesitamos exponer, de acuerdo a lo anterior, una propiedad que proporcione el valor de la instancia misma. De esta forma podríamos comprobar si un atributo o parámetro de tipo entidad es igual a una instancia de entidad concreta, simplificando la complejidad de nuestro modelo.

Para cada *EntityType* se añaden también una serie de acciones predefinidas. La Figura 16 muestra un diagrama de instancias que muestra las acciones predefinidas de los tipos de entidad y sus parámetros.

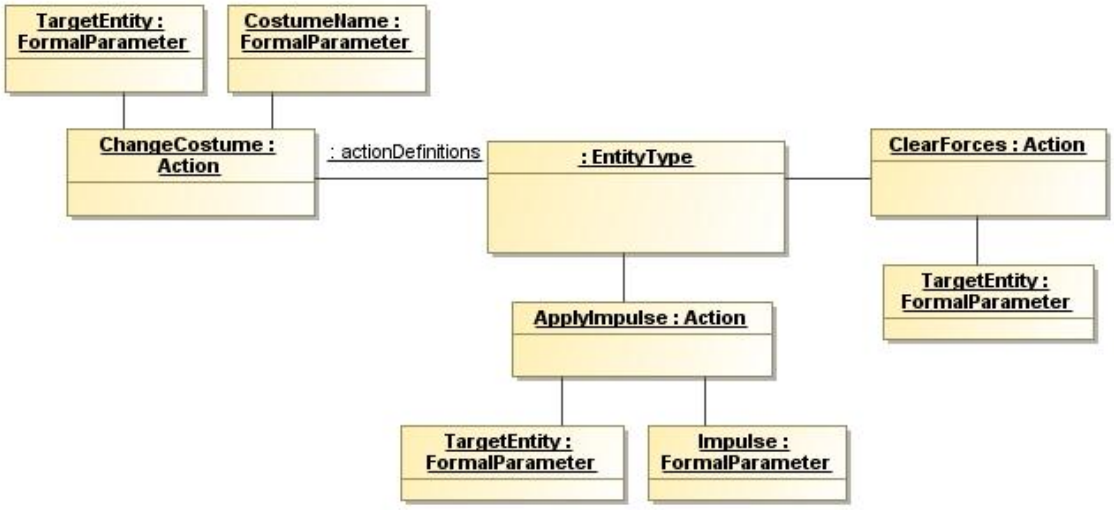

Figura 16. Acciones predefinidas de los tipos de entidad

- ChangeCostume: Cambia la vestimenta activa de una entidad.

```
ChangeCostume ([Target]EmbodiedEntity eInstance, ProtoCostume costume)

PRE:  ((Cast  (eInstance.MetaType,  EmbodiedEntityType)).ProtoCostumes[costume]
is not null)

BEGIN

eInstance.wears = costume;

THROW EVENTOCCURRENCE (CostumeChanged, eInstance, costume));

END;
```

En esta acción se comprueba que se cumple la precondición de que la *protocostume* que se quiere poner a una entidad no sea *null*. En caso de que no sea *null*, se le pone esa *protocostume* a la entidad y se lanza un evento de CostumeChanged.

- ApplyImpulse: Aplica una aceleración a una entidad para aumentar su velocidad.

```
ApplyImpulse ([Target]EmbodiedEntity eInstance, Vector2 direction)

BEGIN

FOREACH EntityBody b in Body eInstance.Structure.bodies

DO

b.ApplyImpulse(direction);

DONE;

END;
```

En esta acción para cada *EntityBody* de la entidad se le aplica un impulso en la dirección indicada.

- ClearForces: Deja de aplicar todo tipo de fuerzas que se estuvieran aplicando sobre una entidad.

```
ClearForces([Target]EmbodiedEntity eInstance)

BEGIN

FOREACH EntityBody b in Body eInstance.Structure.bodies

DO

b.ClearForce();

DONE;

END;
```

En esta acción para cada *EntityBody* de la entidad se dejan de aplicar las fuerzas que se estuvieran aplicando sobre él.

3.3.1.2 Módulo de simulación

El módulo de simulación proporciona una API para manipular y simular los elementos de nuestro ecosistema. Además, contiene una referencia al motor de físicas Farseer y se encarga de transformar los elementos de nuestro modelo con un comportamiento físico, en geometrías (*Geom*) con las que trabaja Farseer, y de gestionar los eventos físicos utilizando para ello una cola de eventos físicos (*PhysicalEvent*).

Para realizar la transformación de los elementos de nuestro modelo con comportamiento físico (*EmbodiedEntity*), en geometrías entendibles por Farseer, cada uno de los skins que cubre un *EntityBody* de una *EmbodiedEntity* se transforma en los objetos de Farseer *Geom* y *Texture*, que contienen información de los vértices de la geometría y de la textura respectivamente.

Necesitamos que cada vez que el motor de físicas *Farseer* detecte la ocurrencia de un evento en el sistema, ya sea de colisión de entidades (*CollisionEvent*), cambios en la posición de las entidades (*PositionChangedEvent*), cambios en la rotación de las entidades (*RotationChangeEvent*), o de salida o entrada de las entidades de los límites de la ventana (*OutOfBoundsEvent*), y se lo comunique al módulo de simulación.

La Figura 17 muestra la parte de nuestro modelo que modela los eventos físicos anteriormente mencionados. Los atributos *Index* son índices de una lista de las geometrías (*Geom*) de Farseer que componen el ecosistema, que nos permiten conocer qué skin de la entidad referenciada ha generado el evento.

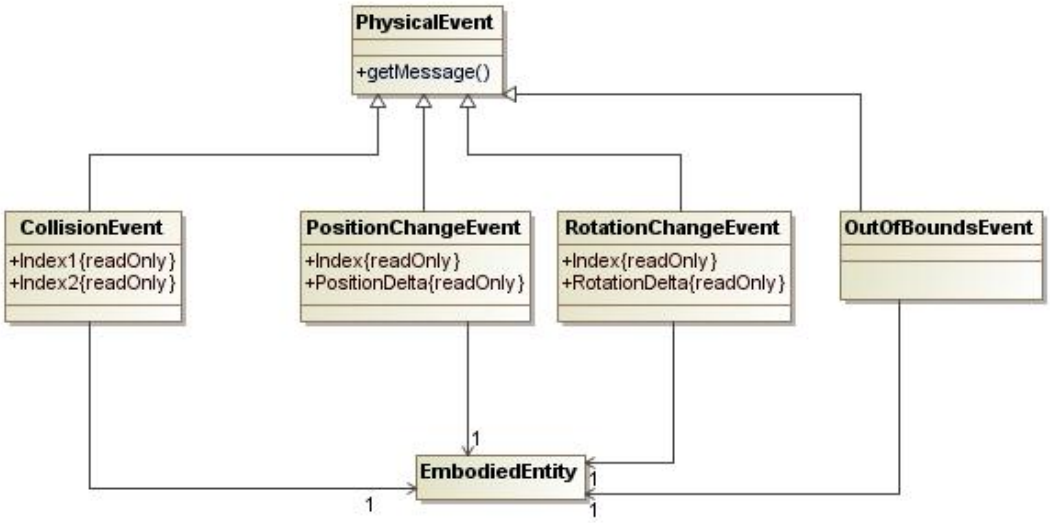

Figura 17. Eventos físicos

Los métodos que proporciona el módulo de simulación para la gestión de estos eventos físicos son:

void EcosystemView_OnSkinCollision(int Id1, int Id2). Captura los eventos de colisión de los skins y encola el *PhysicalEvent* correspondiente. (Id1 es el id de la primera Geom y Id2 el de la segunda).

void EcosystemView_OnPositionChanged(int Id, Vector2 Increment). Captura los eventos de cambio de posición de los skins y encola el *PhysicalEvent* correspondiente. (Id es el id de la Geom y Increment es el valor en que ha cambiado la posición).

void EcosystemView_OnRotationChanged(int Id, float Increment). Captura los eventos de cambio de rotación de los skins y encola el *PhysicalEvent* correspondiente. (Id es el id de la Geom y Increment es el valor en que ha cambiado la rotación).

void EcosystemView_OnSituationChanged(int Id, bool Out). Captura los eventos de cambio de situación de los skins (salida o entrada de la ventana) y encola el *PhysicalEvent* correspondiente. (*Id* es el id de la Geom y *Out* indica el nuevo estado del skin, true si ha salido y false si ha entrado).

Además de las funcionalidades comentadas anteriormente necesitamos otras funcionalidades como poder añadir y quitar entidades de la simulación, cambiar el skin de las entidades, iniciar o parar la simulación, mover y parar las entidades que se están simulando, o aplicar un impulso a una entidad. Para ello el módulo de simulación proporciona una API que nos permite:

1) Cambiar el skin de una entidad

void SetEntityProtocostume(Entity e, ProtoCostume p). Cambia el skin de una entidad (*e* es la entidad y *p* el *protocostume* a partir del cual se creara el skin).

2) Añadir y quitar entidades de la simulación

void AddEntity(EntityType et, Vector2 basePosition). Añade una entidad al ecosistema creada a partir de un tipo de entidad (*et* es la entidad y *basePosition* es la posición base donde se añadirá la entidad).

void RemoveEntity(Entity e). Elimina la entidad seleccionada.

void RemoveEntityByPosition(Vector2 Position). Elimina la entidad que está en la posición deseada.

void RemoveEntityByPosition(Vector2 Position, out Entity removedEntity). Elimina la entidad que está en la posición deseada. (*Position* es la posición en la que se encuentra la entidad y *removedEntity* es la entidad que se ha eliminado).

3) Resetear el Ecosistema y parar o iniciar la simulación

void ResetEcosystem(). Resetea el ecosistema para dejarlo en el estado inicial. Detiene la simulación, elimina todas las entidades, resetea el simulador de físicas y limpia la cola de eventos.

void Run(). Activa la simulación y lanza el evento OnRun().

void Stop(). Desactiva la simulación y lanza el evento OnStop().

4) Cambiar la gravedad del simulador.

void SetGravity(Vector2 grav). Aplica al mundo la gravedad deseada.

5) Mover, obtener y parar las entidades y sus componentes.

void MoveEntityByPosition(Vector2 EntityPosition, Vector2 PositionIncrement). Mueve la entidad que se encuentra en la posición seleccionada. (*EntityPosition* es la posición en la que está la entidad y *PositionIncrement* es el incremento de posición que se quiere aplicar a la entidad).

void MoveEntity(Entity e, Vector2 PositionIncrement). Mueve una entidad (*e* es la entidad a mover y *PositionIncrement* es el incremento de posición que se quiere aplicar a la entidad).

void MoveEntityComponentTo(Vector2 Position, Entity e, int Index). Mueve un componente de una entidad. (*Position* es la nueva posición del componente, *e* es la entidad e *Index* es el índice del componente).

void GetEntityAtPosition(Vector2 Position). Obtiene la entidad que está en la posición seleccionada.

void GetEntityComponentAtPosition(Vector2 Position, out Entity e, out int index). Obtiene una entidad y el índice del componente (el índice del skin que se encuentra en dicha posición en la lista de skins de la entidad) que hay en la posición seleccionada. *Position* es la posición y *e* es la entidad en la posición, null si no hay ninguno y *index* es el índice del componente, -1 si no hay ninguno

void StopEntityAtPosition(Vector2 Position). Para la entidad que está en la posición seleccionada. Position es la posición en la que está la entidad

void StopEntityComponentAtPosition(Vector2 Position). Detiene el componente de una entidad que está en la posición seleccionada.

void StopEntity(Entity e). Detiene una entidad.

void StopEntityComponent(Entity e, int index). Para el componente seleccionado de la entidad seleccionada (*index* es el índice del *EntityBody* que se desea parar en la lista de EntityBodies de la entidad).

6) Aplicar impulso a entidades y componentes

void ApplyImpulseToEntity(Entity e, Vector2 impulse). Aplica un impulso a todos los componentes de una entidad.

void ApplyImpuseToComponent(Entity e, int ComponentIndex, Vector2 impulse). Aplica un impulso a uno de los componentes de una entidad. (*e* es la entidad, *ComponentIndex* es el índice del skin al que se quiere aplicar un impulso en la lista de skins de la entidad y *impulse* es el impulso a aplicar).

void ApplyAngularImpulseToEntity(Entity e, float amount). Aplica un impulso angular a una entidad. (*e* es la entidad y *amount* la cantidad de impulso a aplicar).

void ApplyAngularImpulseToComponent(Entity e, int Index, float amount). Aplica un impulso angular a un componente de una entidad. (*Index* es el índice del *EntityBody* al que se desea aplicar un impulso angular en la lista de EntityBodies de la entidad).

7) Mostrar y ocultar mensajes en pantalla.

void ShowMessage(String message). Muestra un mensaje en la pantalla.

void HideMessage(). Esconde el mensaje.

3.3.1.3 Módulo de orquestación

El módulo de orquestación se encarga de realizar la coordinación para capturar los eventos y de esa forma hacer progresar el sistema. Para ello, utiliza llamadas a la API del módulo de simulación y al módulo de definición del ecosistema.

La parte de coordinación que realiza este módulo es una de las partes más importantes del proceso general de la simulación. Dicho proceso consta de las siguientes partes:

1) El simulador carga la definición del ecosistema desde un fichero .eco. mediante el módulo de definición del ecosistema e inicializa las propiedades y acciones predefinidas de las entidades del escenario actual.
2) En segundo lugar, se inicializa el módulo de simulación en el cual las entidades con comportamiento físico se convierten en geometrías manejables por el motor de físicas Farseer. Cuando se produce un evento en Farseer, se le comunica al módulo de simulación que lo encola en una cola de eventos físicos.
3) El simulador comprueba periódicamente si se ha producido un evento en la simulación comprobando la cola de eventos físicos del módulo de simulación y en ese caso, enruta el evento a un procesador de matching de eventos (*EventMatchingProcessor*) que se encarga de comprobar para cada ocurrencia de evento pendiente, si existe una regla reactiva del escenario actual que se acople a dicha ocurrencia de evento, que cumpla las condiciones de la regla, y que cumpla la precondición de la misma. Si es así, se indica al simulador que existen nuevas coincidencias disponibles y se realiza la acción o actualización de la propiedad de la entidad que se indique en la regla disparada.

En la Figura 18 muestra un esquema de las componentes involucradas en el proceso general de simulación comentado en el párrafo anterior.

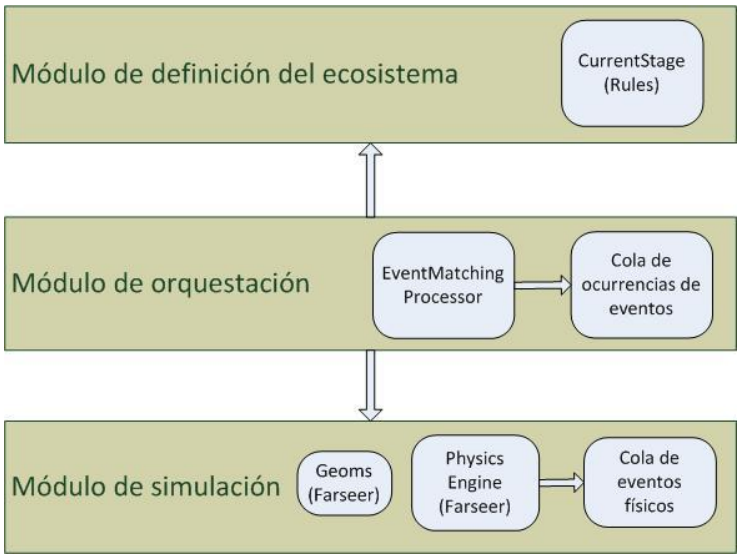

Figura 18. Proceso general de la simulación

3.4 Casos de estudio

En este apartado exponemos dos casos de estudio de juegos, el Arkanoid y el Pong, que se han implementado utilizando para ello el modelo propuesto. Dichos casos de estudio nos permitirán validar el modelo propuesto.

3.4.1 Arkanoid

Arkanoid es un juego de los años 80 de tipo arcade cuyo objetivo es destruir todos los ladrillos de la pantalla de juego utilizando para ello una paleta que únicamente se puede mover en el eje horizontal y con la que golpearemos una bola que a su vez golpeará los ladrillos hasta destruirlos y eliminarlos del escenario. Se dispone de una serie de vidas que se consumen cada vez que la bola sale por la parte inferior de la pantalla y que al agotarse hacen que acabe la partida. Una vez se destruyen todos los ladrillos, se pasa de nivel. No todos los ladrillos se rompen después del primer golpe; algunos de ellos deben golpearse 2 o más veces. En la Figura 19 podemos ver una imagen del Arkanoid original.

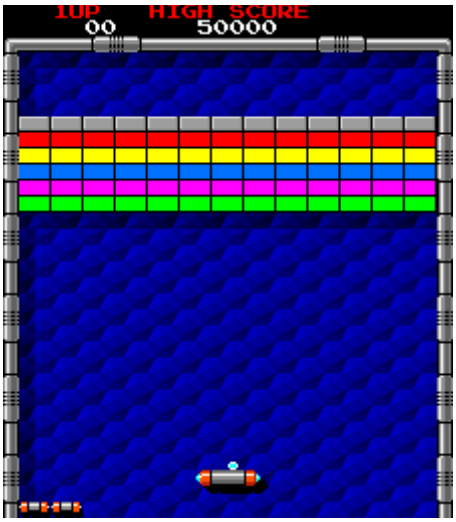

Figura 19. Arkanoid original

A diferencia del juego original en el que solo podía jugar un jugador, en la versión que hemos implementado pueden jugar dos jugadores, de forma que los dos jugadores se reparten el eje horizontal entre los dos. Cada jugador suma un punto por cada ladrillo que golpea ya sea directamente o utilizando los rebotes de la bola en las paredes y en otros ladrillos para golpearlos. En la implementación que hemos hecho el color de los ladrillos indica el número de golpes que les queda para ser destruidos: los ladrillos de color rojo indican que les quedan dos golpes y los de color verde indican que les queda un golpe. Al golpear un ladrillo de color rojo cambia su color a verde para indicar que le queda un solo golpe para ser destruido.

3.4.1.1 Representación para el escenario del ARKANOID

En nuestra versión de Arkanoid hemos modelado los componentes del juego en términos de tipos de entidades, tipos de estructuras, y posibles vestimentas de dichos tipos de entidad, tal y como se muestra en la Figura 20.

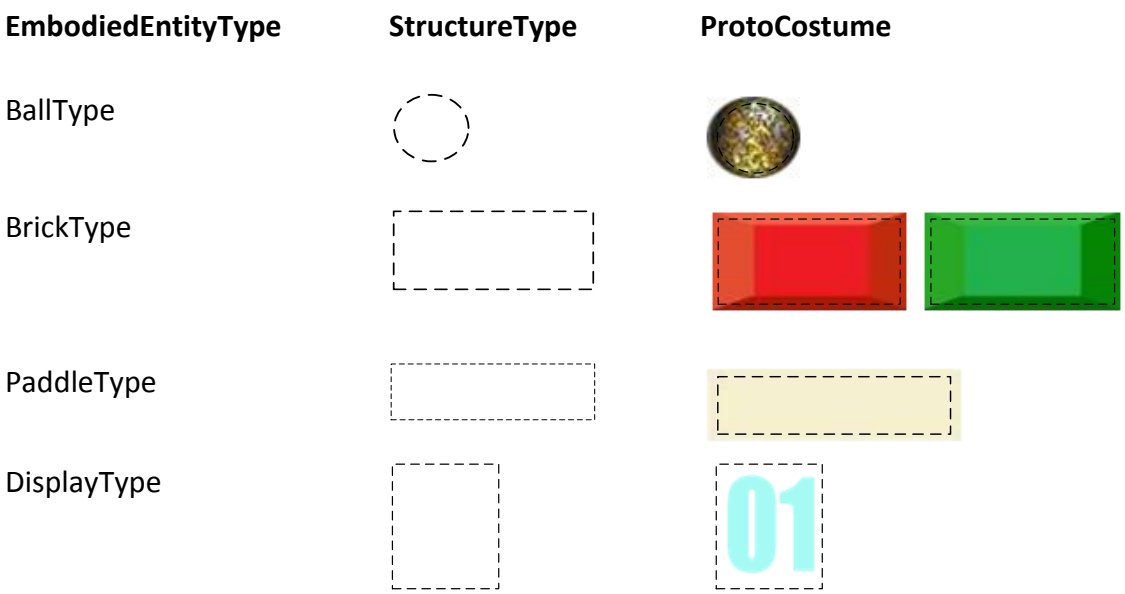

Figura 20. Modelado de Arkanoid

BallType es el tipo de entidad que usaremos para la bola. Este tipo de entidad contendrá la propiedad de tipo entero *LastPaddleHit* que usaremos para indicar la última paleta que ha golpeado la bola y de esta forma saber a quién sumar los puntos por golpear un ladrillo. Si la bola golpea a la paleta 1 la propiedad tomará el valor 1 y si golpea a la paleta 2 tomará el valor 2. Este tipo de entidad utilizará una única vestimenta con forma redonda.

BrickType es el tipo de entidad que usaremos para los ladrillos. Tiene dos vestimentas posibles: una roja y otra verde. Este tipo de entidad contendrá la propiedad de tipo entero *NumHits*, que usaremos para guardar el número de veces que un ladrillo debe ser golpeado por la bola antes de ser destruido.

PaddleType es el tipo de entidad que usaremos para las paletas. Utiliza una única vestimenta con forma rectangular.

DisplayType es el tipo de entidad que usaremos para los marcadores de puntuación. Este tipo de entidad tiene 11 vestimentas en las que cada una de las vestimentas es una imagen de un número de 0 a 10. Inicialmente se le pone a cada marcador la vestimenta del 0. Este tipo de entidad contendrá la propiedad de tipo entero *Count* que guarda la puntuación del jugador en forma numérica.

En la Figura 21 podemos ver un sketch del escenario que queremos modelar.

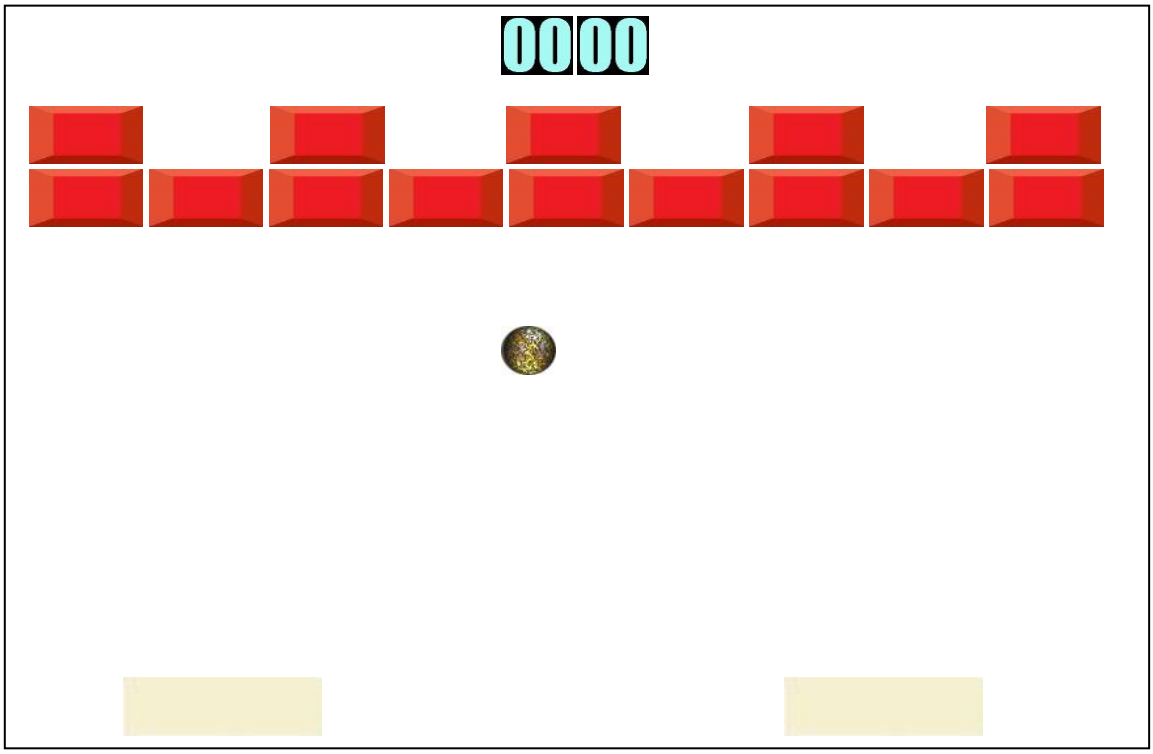

Figura 21. Sketch de Arkanoid

La definición del ecosistema que utiliza nuestra versión de Arkanoid se almacena en un fichero con la extensión .eco. Dicho fichero contiene la definición de todos los elementos necesarios anteriormente descritos utilizando para ello un formato específico. En la Figura 22 vemos como se inicia la definición del ecosistema para nuestra versión de Arkanoid.

```
ECOSYSTEM NAME Arkanoid

…

END_ECOSYSTEM
```

Figura 22. Definición de ecosistema

La Figura 23 muestra la parte del fichero .eco que se encarga de definir los tipos de estructura. En dicha figura vemos cómo se define el tipo de estructura BallStructure indicando la ruta de un fichero .st que contiene la definición de dicho tipo de estructura. El formato del fichero .st se explica en el Anexo.

```
STRUCTURE_TYPES
      STRUCTURE_TYPE NAME BallStructure PATH
      "Content/Simulator/Arkanoid/BallStructure.st" END_STRUCTURE_TYPE
      …
END_STRUCTURE_TYPES
```

Figura 23. Definición de tipos de estructura

Los tipos de entidad también se definen en el fichero .eco que contiene la definición de nuestro ecosistema. Dicha definición de tipos de entidad se realiza de la siguiente manera:

1. Se asignan valores a unos atributos predefinidos de cada tipo de entidad. Estos atributos pueden ser los siguientes, y su valor debe ser del tipo lógico (true/false). Si no se especifican, tomarán su valor por defecto.
    - PHYSICAL_RESPONSE: Indica si la entidad debe responder a eventos físicos tales como la gravedad, rebotes, etc. Si se establece a falso, será una entidad inerte que no interaccionará con ninguna otra. Por defecto vale true.
    - REBOUND: Para una entidad A, indica si, al producirse una colisión con otra entidad B, debe producirse el efecto de que A rebota contra B. Por defecto vale false.
    - IS_FINGER_SENSITIVE: Indica si la entidad debe reaccionar al contacto de un dedo para cambiar su posición acorde con la de éste. Por defecto vale false.
    - IS_TAG_SENSITIVE: Indica si la entidad debe reaccionar al contacto de un tangible cambiando su posición conforme lo haga éste. Por defecto vale false.
2. Se asocia al tipo de entidad un tipo de estructura ya definido.
3. Se crean y al mismo tiempo se asignan las diferentes vestimentas que podrá vestir una entidad de este tipo durante la simulación. Cada vestimenta lleva asociado un nombre y la ruta de un fichero .pc donde dicha vestimenta es definida.
4. Se crean propiedades propias de este tipo de entidad. Cada propiedad deberá tener un nombre y un tipo. Estos tipos pueden ser:
    - Integer: Número entero.
    - Float: Número real en coma flotante.
    - String: Cadena de caracteres.
    - Vector2: Vector de dos componentes reales en coma flotante representado como dos números consecutivos.
5. Se crean las acciones propias de este tipo de entidad. Cada acción deberá tener asociado un nombre y, si corresponde, una serie de parámetros. Cada parámetro deberá tener un nombre y un tipo. Los tipos son los mismos que para las propiedades.

Un ejemplo de definición de un tipo de entidad puede verse en la Figura 24. En este ejemplo se define el tipo de entidad *PaddleType*. En primer lugar, se les da valor a los atributos predefinidos: se pone el atributo PHYSICAL_RESPONSE a true ya que queremos que la paleta responda a eventos físicos y se pone a true el atributo IS_FINGER_SENSITIVE ya que queremos que los jugadores puedan manejar la paleta con los dedos. En segundo lugar, se le asigna a este tipo de entidad el tipo de estructura *PaddleStructure* y por último se definen las vestimentas que podrá utilizar

este tipo de entidad, en nuestro caso una vestimenta llamada *PaddleCostume*, indicando la ruta del fichero .pc que contiene la definición de dicha vestimenta. El formato del fichero .pc se explica en el Anexo.

```
ENTITY_TYPES
      ENTITY_TYPE PaddleType
            PHYSICAL_RESPONSE true
            IS_FINGER_SENSITIVE true
            IS_TAG_SENSITIVE true
            STRUCTURE_TYPE PaddleStructure
            PROTO_COSTUMES
                  PROTO_COSTUME PaddleCostume PATH
                  "Content/Simulator/Arkanoid/PaddleCostume.pc"
                  END_PROTO_COSTUME
            END_PROTO_COSTUMES
      END_ENTITY_TYPE

…

END ENTITY TYPES
```

Figura 24. Definición de tipos de entidad

Debemos definir también las distintas entidades que formaran parte del ecosistema. Cada entidad debe tener un nombre y un tipo de entidad ya definido asociado. Además, se debe indicar qué vestimenta visten en el momento inicial de la simulación.

Para nuestro ecosistema necesitaremos definir las siguientes entidades:

1) 15 entidades del tipo de entidad *BrickType* con la vestimenta *RedBrickCostume*.
2) 1 entidad de tipo de entidad *BallType* con la vestimenta *BallCostume*.
3) 2 entidades de tipo de entidad *PaddleType*, Paddle1 y Paddle2, con la vestimenta PaddleCostume.
4) 2 entidades de Display1 de tipo de entidad *DisplayType*: Display1 y Display2. Representan los marcadores de puntuación del jugador 1 y del jugador 2 respectivamente. Tienen como vestimenta inicial la correspondiente al número 0 y tienen la propiedad *Count* a 0.

En la Figura 25 se muestra un ejemplo de definición de entidades en Arkanoid. En este ejemplo se define la entidad *Ball* que es del tipo de entidad *BallType* y utiliza la vestimenta *BallCostume*.

```
ENTITIES
      ENTITY
            NAME Ball TYPE BallType COSTUME BallCostume
      END_ENTITY
…

END_ENTITIES
```

Figura 25. Definición de entidades

El siguiente paso es definir los distintos escenarios que puede tener el ecosistema e indicar cuál debe ser el escenario inicial. Una vez realizado esto la definición del ecosistema estará finalizada.

Para cada escenario hay que definir los siguientes elementos en el fichero .eco:

1. Nombre.
2. Atributos sobre los bordes del escenario. Esto es, indicar si el escenario (que ocupará toda la superficie) debe tener bordes (para evitar que las entidades se salgan de la mesa) o no. Dichos atributos son los siguientes y reciben un valor lógico (true/false) que por defecto (si no se le indica lo contrario) está establecido a false:
    - TOP_BORDER
    - BOTTOM_BORDER
    - LEFT_BORDER
    - RIGHT_BORDER
3. Participación de las distintas entidades.
4. Reglas reactivas.

El punto 3 consiste en definir las entidades que pueden participar en el escenario y en asignar un valor a cada una de las propiedades (predefinidas o definidas por el usuario) de las entidades. Dicho valor es el que tomarán inicialmente las propiedades si el escenario en el que se encuentran es lanzado a simulación. Las propiedades predefinidas son las siguientes:

- Position: Posición que ocupará la entidad en el escenario (teniendo en cuenta que el punto central de la superficie es el centro de coordenadas cartesiano). Debe ser del tipo Vector2. Por ejemplo, para situar una entidad en el origen debe especificarse como valor 0,0 0,0 (es decir, dos elementos en coma flotante correspondientes a las coordenadas en el eje de las abscisas X y de las ordenadas Y, respectivamente).
- InitialSpeed: Impulso inicial aplicado sobre la entidad. Es del tipo Vector2.
- Rotation: Real en coma flotante que indica la orientación inicial de la entidad en radianes.
- Gravity: Vector2 que indica la intensidad de la fuerza de gravedad que se aplica sobre una entidad.

En la Figura 26 se ejemplifica la definición del escenario ArkanoidGameStage. Como no queremos que la pelota rebote en el borde inferior y sí en el resto de bordes, ponemos el atributo BOTTOM_BORDER a false y activamos el resto de atributos de bordes. A continuación, en la sección PARTICIPATIONS definimos la participación de la entidad *Ball* de tipo *EntityBall* en el escenario y inicializamos la propiedad position a (0,-100), la propiedad *InitialSpeed* a (-500,-500) y la propiedad *LastPaddleHit* a 0.

```
STAGES
      STAGE
      NAME ArkanoidGameStage
      TOP_BORDER true
      BOTTOM_BORDER false
      LEFT_BORDER true
      RIGHT_BORDER true
      PARTICIPATIONS
            ENTITY BallType::Ball
            PROPERTY NAME Position TYPE Vector2 VALUE 0,0 -100,0
            END_PROPERTY
            PROPERTY NAME InitialSpeed TYPE Vector2 VALUE -500,0 500,0
            END_PROPERTY
            PROPERTY NAME LastPaddleHit TYPE Integer VALUE 0 END_PROPERTY
            END_ENTITY

            …

            END_PARTICIPATIONS

            … [RULES DEFINITION]

      END_STAGE
END_STAGES

CURRENT_STAGE ArkanoidGameStage
```

Figura 26. Definición de escenarios

Por último, después de la sección de participaciones se definen las reglas reactivas que se usan en ese escenario. Cada una de las reglas tiene asociada una prioridad (si no, por defecto, es 1) de manera que una regla se ejecuta antes que otra cuanto mayor sea su probabilidad. También tiene asociado un evento, que es el que lanza la evaluación de la regla. Además, un origen y un objetivo, que pueden ser o bien una entidad o un tipo de entidad (si se quiere hacer referencia a cualquier entidad de ese tipo); y una operación (que puede ser la modificación de una propiedad del objetivo o la ejecución de una acción). Adicionalmente, pueden tener una precondición que determinará si la regla debe ser ejecutada o no; un filtro-precondición que determina, siempre que el objetivo sea un tipo de entidad, qué entidades de dicho tipo se verán afectadas; y la especificación de la operación que se desea realizar si ésta es asignar un valor a una propiedad de la entidad o entidades objetivo.

A continuación, comentamos las reglas expresadas en pseudocódigo que se han definido en la implementación del Arkanoid.

En nuestra versión de Arkanoid, necesitamos saber la última paleta que ha golpeado la bola para saber a qué jugador sumar los puntos por golpear un ladrillo con la bola. Para ello utilizamos la propiedad *LastPaddleHit* del tipo de entidad *BallType* que debe valer 1 si la ultima paleta que ha golpeado la bola es la paleta 1 (Paddle1) y debe valer 2 si la ultima paleta que ha golpeado la bola es la paleta 2 (Paddle2). Las siguientes

reglas se encargan de asignar el valor de esta propiedad en función de la paleta que haya golpeado la bola.

PRIORITY: 10
IF S: Paddle1 THROWS an EVENT E: Collision
AND E.SlaveEntity = Ball
THEN WITH T: Ball
PERFORM O: T.LastPaddleHit = 1

PRIORITY: 10
IF S: Paddle2 THROWS an EVENT E: Collision
AND E.SlaveEntity = Ball
THEN WITH T: Ball
PERFORM O: T.LastPaddleHit = 2

En nuestra versión de Arkanoid, los ladrillos con la vestimenta de color rojo se destruyen después de dos golpes: si reciben un golpe cambian su vestimenta por otra de color verde y si reciben otro golpe más se eliminan del escenario. Para implementar esto, necesitamos una regla que en caso de colisión de la bola con un ladrillo, reduzcan en 1 la propiedad *NumHits* del ladrillo:

PRIORITY: 15
IF S: ANY BrickType THROWS an EVENT E: Collision
AND S.NumHits > 0 AND E.SlaveEntity=Ball
THEN WITH T: ANY BrickType
SO THAT E.Entity = T.Value
PERFORM O: T.NumHits = T.NumHits -1

Necesitamos dos reglas más que, en caso de colisión de la bola con un ladrillo, incrementen la propiedad *Count* del marcador del último jugador que haya golpeado la bola, es decir, en caso de que la propiedad *LastPaddleHit* valga 1, entonces incrementaremos en 1 la propiedad *Count* del Display1, y en caso de que valga 2 incrementaremos la propiedad *Count* del Display2.

PRIORITY: 15
IF S: Ball THROWS an EVENT E: Collision
AND InstanceOf(E.SlaveEntity, “BrickType”) and S. LastPaddleHit=1
THEN WITH T: Display1
PERFORM O: T.Count = T.Count + 1

PRIORITY: 15
IF S: Ball THROWS an EVENT E: Collision
AND InstanceOf(E.SlaveEntity, “BrickType”) and S. LastPaddleHit=2

THEN WITH T: Display2
PERFORM O: T.Count = T.Count + 1

Al alcanzar la propiedad *NumHits* de un ladrillo el valor 1, queremos que cambie la vestimenta roja del ladrillo la verde, para ello usaremos la siguiente regla:

PRIORITY: 5
IF S: ANY BrickType THROWS an EVENT E: PropertyValueChanged
AND E.Property = NumHits AND NumHits=1
THEN WITH T: ANY BrickType
SO THAT E.Entity = T.Value
PERFORM O: T.ChangeCostume("GreenBrickCostume")

También necesitamos una regla que elimine el ladrillo del escenario si el *NumHits* del ladrillo es igual a 0. Esto lo conseguimos con la siguiente regla:

PRIORITY: 10
IF S: ANY BrickType THROWS an EVENT E: PropertyValueChanged
AND E.Property = "NumHits" AND S.NumHits = 0
THEN WITH T: ArkanoidGameStage
PERFORM O: T.ExcludePerformer(S)

Para que cuando cambie el valor de la propiedad *Count* de los marcadores, cambie la vestimenta acorde al nuevo valor establecido, se utiliza la siguiente regla:

PRIORITY: 10
IF S: ANY DisplayType THROWS an EVENT E: PropertyValueChanged
AND E.Property = "Count"
THEN WITH T: ANY DisplayType
SO THAT E.Entity = T.Value
PERFORM O: T.ChangeCostume("DisplayCostume" CONCATENATED T.Count)

En la implementación de Arkanoid que hemos hecho pueden jugar dos jugadores que se reparten el eje horizontal entre los dos, por lo tanto necesitamos reglas que limiten el movimiento en el eje X de las dos paletas.

El rango de movimiento en el eje X de la paleta 1 (Paddle1) irá de -462 a – 50 y el de la paleta 2 (Paddle2) irá de 50 a 462.

Las siguientes dos regla limitan a -50 el máximo valor y a -462 el mínimo valor que puede tomar la coordenada X de la posición de la paleta 1.

PRIORITY: 15
IF S: Paddle1 THROWS an EVENT E: PropertyValueChanged
AND E.Property = Position AND S.Position.X > -50

THEN WITH T: Paddle1
PERFORM O: T.Position.X = -50

PRIORITY: 15
IF S: Paddle1 THROWS an EVENT E: PropertyValueChanged
AND E.Property = Position AND S.Position.X < -462
THEN WITH T: Paddle1
PERFORM O: T.Position.X = -462

Las siguientes dos regla limitan a 50 el mínimo valor y a 462 el máximo valor que puede tomar la coordenada X de la posición de la paleta 2.

PRIORITY: 15
IF S: Paddle2 THROWS an EVENT E: PropertyValueChanged
AND E.Property = Position AND S.Position.X < 50
THEN WITH T: Paddle2
PERFORM O: T.Position = 50

PRIORITY: 15
IF S: Paddle2 THROWS an EVENT E: PropertyValueChanged
AND E.Property = Position AND S.Position.X > 462
THEN WITH T: Paddle2
PERFORM O: T.Position = 462

Si la pelota se sale por la parte inferior de la pantalla (la parte que cubren las paletas), entonces la pelota debe volver al centro de la superficie, esto se consigue con la regla:

PRIORITY: 15
IF S: Ball THROWS an EVENT E: OutOfBounds
AND E.Property = Position AND E.Position.Y = -384
THEN WITH T: Ball
PERFORM O: T.Position = 0,0 0,0

Queremos que el juego finalice cuando algún jugador llegue a 10 puntos, para lo que usamos la regla siguiente:

PRIORITY: 10
IF S: ANY DisplayType THROWS an EVENT E: PropertyValueChanged
AND S.Count = 10
THEN WITH T: ArkanoidGameStage
PERFORM O: T.PauseSimulation

Una muestra de cómo se vería en simulación la implementación del Arkanoid realizada puede verse en la Figura 27.

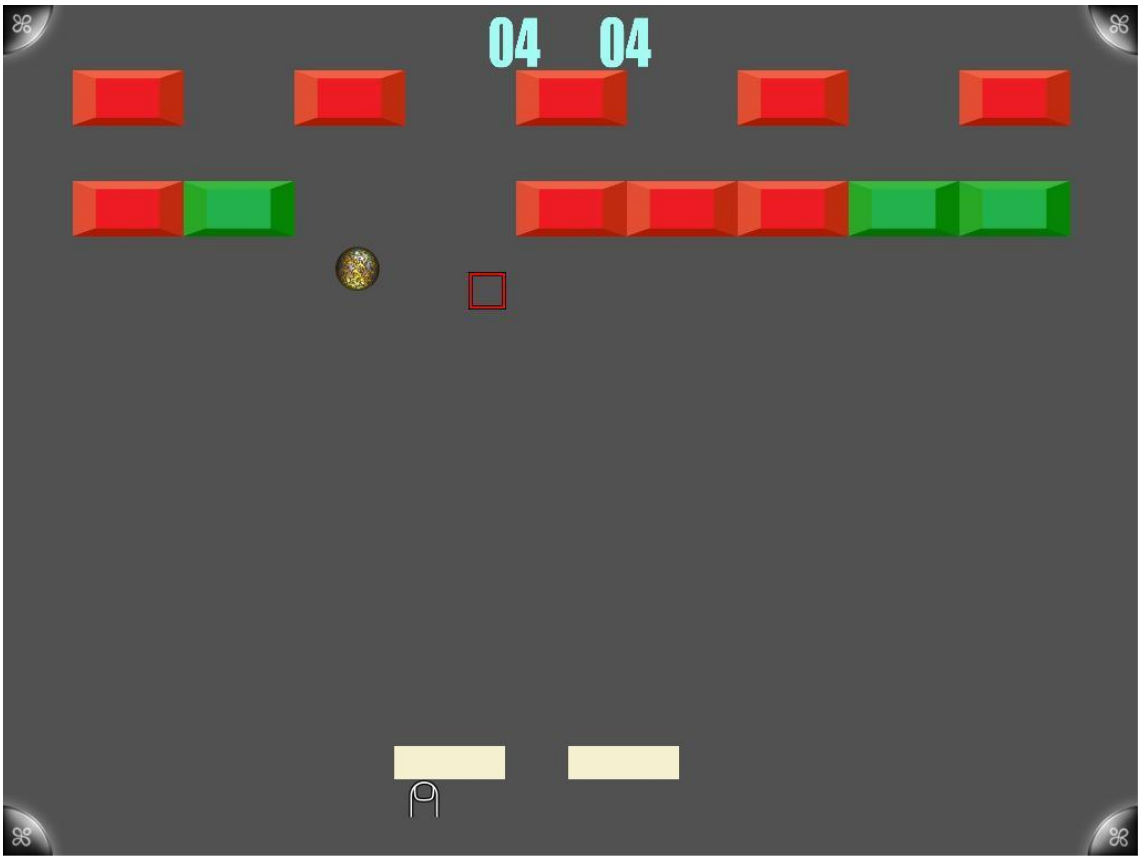

Figura 27. Screenshot de la implementación de Arkanoid

3.4.2 Pong

El Pong fue uno de los primeros juegos de tipo arcade en los años 70. Es un juego de deportes en 2D que simula el tenis de mesa o ping-pong. El jugador controla una paleta del juego moviéndola verticalmente a través de la parte izquierda de la pantalla y puede competir contra un oponente controlado por el ordenador u otro jugador que controla una segunda paleta en el lado opuesto. Los jugadores usan las paletas para golpear la bola de un lado a otro. El objetivo del juego es que un jugador gane más puntos que su oponente; los puntos se consiguen cuando uno de los jugadores no consigue devolver la bola al otro jugador. En la Figura 28 podemos ver una imagen del Pong original.

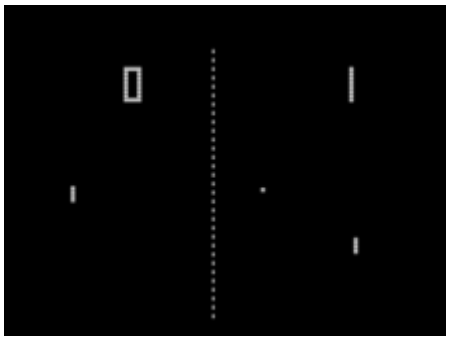

Figura 28. Pong original

A diferencia del juego original en el que las paletas se movían mediante un joystick, en la versión que hemos implementado los jugadores pueden mover las paletas con los dedos. Además, la partida acaba cuando uno de los dos jugadores llega a 10 puntos.

3.4.2.1 Representación para el escenario Pong

En nuestra versión de Pong necesitaremos una bola, dos paletas para golpear la bola (una para cada jugador), dos marcadores para contar los puntos de cada jugador y dos "porterías" invisibles cuya función es detectar que una paleta no ha podido golpear a la bola y que, por tanto, uno de los dos jugadores ha anotado un punto.

En la Figura 29 podemos ver un sketch del escenario que queremos modelar.

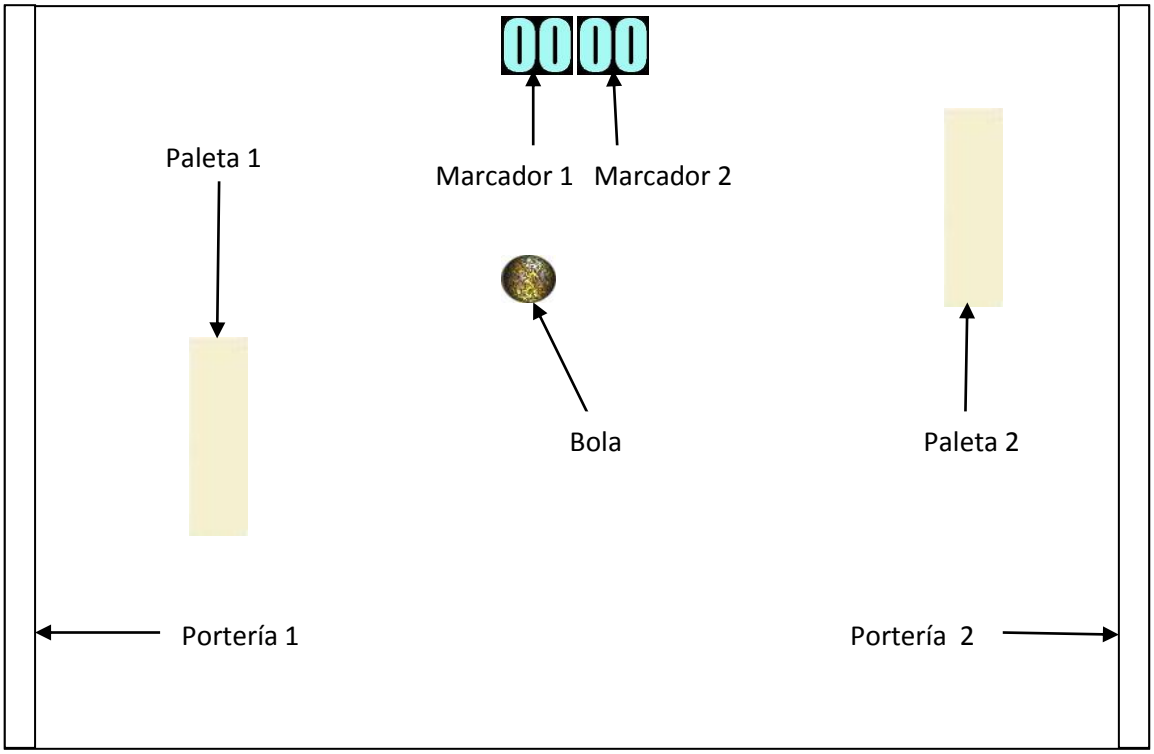

Figura 29. Sketch de Pong

Para ello necesitaremos los siguientes tipos de entidad:

El tipo de entidad *BallType* nos permite representar la bola. Tendrá el atributo REBOUND a true y una única vestimenta con una forma redonda.

El tipo de entidad *PaddleType* nos permite representar las paletas. Tendrá los atributos IS_FINGER_SENSITIVE e IS_TAG_SENSITIVE a true y una única vestimenta con forma rectangular.

El tipo de entidad *DisplayType* nos permite representar los marcadores de puntuación. Tendrá en atributo PHYSICAL_RESPONSE a false ya que no queremos que los marcadores reaccionen ante otras entidades. Este tipo de entidad tiene 11 vestimentas diferentes en las que cada una de las vestimentas es una imagen de un

número de 0 a 10, que representan los distintos valores que podrán tomar los marcadores (al llegar a 10 el juego terminará). Inicialmente se le pone a cada marcador la vestimenta del 0. Este tipo de entidad contendrá la propiedad de tipo entero Count que guarda la puntuación del jugador en forma numérica.

El tipo de entidad *GoalType* nos permite representar las porterías y tendrá una única vestimenta con una forma rectangular y con un color muy difuminado para que no se aprecie mucho.

A continuación, se definen las entidades que antes se comentaban con sus tipos correspondientes. Sólo indicar que las entidades marcadores deben tener como vestimenta inicial la correspondiente al número 0. Tendremos las entidades Bola, Paleta 1, Paleta 2, Marcador 1, Marcador2, Portería 1 y Portería 2.

El siguiente paso es definir los distintos escenarios que puede tener el ecosistema y, después, se indicará cuál debe ser el escenario inicial. Una vez realizado esto la definición del ecosistema estará finalizada. En nuestro ejemplo, el único escenario que tenemos se llama *Table* y tiene los cuatro bordes activos.

Pasamos a describir ahora las distintas participaciones de las entidades en el escenario. En el caso del Pong, las participaciones de cada entidad en el escenario Table se establecen como sigue:

- Bola
  - Position: 0,0 0,0 (en el centro de la superficie).
  - InitialSpeed: -500,0 -500,0 (una velocidad con dirección arbitraria).
- Paleta 1
  - Position: -480,0 0,0 (a la izquierda de la superficie).
- Paleta 2
  - Position: 480,0 -300,0 (a la derecha de la superficie).
- Marcador 1
  - Position: -50,0 350,0 (en la parte de arriba de la superficie, un poco más cerca de la paleta 1).
  - Count: 0 (propiedad definida por nosotros).
- Marcador 2
  - Position: 50,0 350,0 (en la parte de arriba de la superficie, un poco más cerca de la paleta 2).
  - Count: 0 (propiedad definida por nosotros).
- Portería 1
  - Position: -540,0 0,0 (a la izquierda de la superficie, detrás de la paleta 1).
- Portería 2
  - Position: 540,0 0,0 (a la derecha de la superficie, detrás de la paleta 2).

Las reglas definidas en el ejemplo del Pong son las siguientes expresadas en pseudocódigo:

- GameOver: Cuando algún jugador llegue a 10 puntos, el juego se detiene.

PRIORITY: 10
IF S: ANY DisplayType THROWS an EVENT E: PropertyValueChanged
AND S.Count = 10
THEN WITH T: Table
PERFORM O: T.PauseSimulation

- Score1Increment: Cuando la bola toca la portería del jugador 2, el contador del jugador 1 se incrementa en 1.

PRIORITY: 1
IF S: ANY BallType THROWS an EVENT E: Collision
AND E.SlaveEntity = Goal2
THEN WITH T: Display1
PERFORM O: T.Count = T.Count + 1

- Score2Increment: Cuando la bola toca la portería del jugador 1, el contador del jugador 2 se incrementa en 1.

PRIORITY: 1
IF S: ANY BallType THROWS an EVENT E: Collision
AND E.SlaveEntity = Goal2
THEN WITH T: Display2
PERFORM O: T.Count = T.Count + 1

- ChangeCostume: El marcador que ve incrementado su contador, cambia su vestimenta acorde al nuevo valor establecido.

PRIORITY: 0
IF S: ANY DisplayType THROWS an EVENT E: PropertyValueChanged
AND E.Property = Count
THEN WITH T: ANY DisplayType
SO THAT E.Entity = T.Value
PERFORM O: T.ChangeCostume(“DisplayCostume” CONCATENATED T.Count)

- ResetBallPosition: Cuando la bola choca contra una portería, vuelve al centro de la superficie.

PRIORITY: 15
IF S: ANY DisplayType THROWS an EVENT E: PropertyValueChanged
THEN WITH T: Ball
PERFORM O: T.Position = 0,0 0,0

3.5 Limitaciones

En este apartado comentamos algunas limitaciones del modelo propuesto así como posibles soluciones a dichas limitaciones. Concretamente, comentaremos limitaciones relacionadas con el modelo de interacción del usuario, con la representación de variables en la superficie interactiva, con la posibilidad de segmentar los juegos implementados y con la posibilidad de definir entidades no monolíticas en un juego.

3.5.1 Modelo de interacción del usuario

Una de las limitaciones del modelo es que no se trata íntegramente la parte de la interacción de usuario, es decir, no podemos definir maneras avanzadas de interactuar con los elementos del ecosistema. Con el modelo actual solo podemos hacer que una entidad del ecosistema siga el rastro de un contacto producido por un usuario sobre la superficie interactiva, o que detecte cuando el usuario toca una entidad del ecosistema. Aunque esto nos permitiría la construcción de reglas para gestionar y enriquecer la interacción, cubriendo interacciones básicas, sería de interés soportar mecanismos más avanzados y complejos en lo concerniente a la interacción de usuario. Por ejemplo, con el modelo actual no podríamos definir un determinado comportamiento para una entidad del ecosistema cuando, por ejemplo, un usuario trazara una circunferencia que rodease dicha entidad, ya que actualmente no podemos detectar ese gesto.

Una posible solución a este problema sería modificar el modulo de simulación del modelo de ejecución y simulación, añadiendo una nueva capa en el middleware que contendría una implementación de un modelo de interacción avanzado que detectara las interacciones avanzadas que queremos y nos permitiera asignar un comportamiento a dichas interacciones. Esto nos permitiría crear un framework de interacciones con las interacciones más útiles y asignar comportamientos genéricos independientes del ecosistema. Por ejemplo, dentro del framework de interacciones podríamos tener una interacción que fuera “trazar una cruz sobre una entidad” a la que le podríamos asignar el comportamiento “eliminar dicha entidad del escenario”.

3.5.2 Representación de variables

Otra limitación que presenta el modelo propuesto es que no existe una forma de representar variables numéricas y alfanuméricas en la superficie interactiva. Es por esto que en los marcadores de los juegos implementados se han utilizado skins para visualizar la puntuación de cada jugador. El problema de esta forma de visualización es que si queremos modificar los juegos para que acaben cuando un jugador llegue a los 100 puntos, hemos de crear el resto de skins que nos faltan hasta llegar a 100. Esto no haría falta si dispusiéramos de una forma de representar en pantalla la propiedad *Count* que tenemos en las entidades que sirven de marcador.

Una posible solución a este problema sería añadir a la clase *EntityType* una nueva clase especializada llamada *DisplayEntityType* que nos permitiera representar en un

visualizador el valor de una propiedad de una entidad en la simulación. Esta clase estaría asociada con una propiedad de una entidad participante en el escenario de simulación y tendría atributos como por ejemplo la localización del visualizador en la superficie interactiva, su color y su tamaño.

3.5.3 Segmentación de los juegos

Llamamos segmentación de un juego al proceso de gestionar y regular el desarrollo de la experiencia de juego durante el diseño de un juego. La segmentación del juego describe como se descompone un juego en unidades más pequeñas de juego.

Se identifican dos modos principales en los que se suele segmentar el juego: temporal y espacial. El primer modo es la segmentación temporal, lo que quiere decir limitar, sincronizar y/o coordinar la actividad del jugador a lo largo del tiempo. Este tipo de segmentación toma dos subtipos distintos. El primero se denomina segmentación temporal mediante coordinación y regula quien juega en qué momento. El segundo se denomina segmentación con el tiempo como recurso y especifica límites de tiempo o periodos de juego. El primero trata sobre coordinación, mientras que el segundo usa el tiempo como un recurso.

La segmentación temporal mediante coordinación se refiere a cómo un juego regula las acciones de un jugador en un juego y cómo estas acciones ocurren a lo largo del tiempo. La forma más tradicional de coordinación es aquella en que los jugadores toman turnos. En muchos juegos, en cualquier momento dado, solo un jugador puede realizar acciones en el juego, mientras que los otros jugadores esperan a su turno para jugar. En los juegos que usan rondas, los jugadores deciden independientemente sus acciones y resuelven después las consecuencias de esas acciones simultáneamente. Una vez que las acciones se han resuelto, empieza una nueva ronda y los jugadores deciden otra vez sus acciones. Esta forma de segmentación regula las acciones de los jugadores a lo largo del tiempo, pero no restringe el tiempo que sus movimientos pueden durar.

Con nuestro modelo resulta complicado implementar juegos que requieran turnos ya que, aunque se podrían simular los turnos mediante reglas, no tenemos implementado un concepto específico para los turnos. La solución a este problema sería incorporar al modelo e implementar explícitamente conceptos como jugador, turno y/o tirada de un jugador.

Otra variedad de segmentación temporal es aquella que utiliza el tiempo como recurso. En estos tipos de juegos el propio juego establece su duración total o la de sus subperíodos. La división temporal no tiene porqué ser constante. El tiempo puede considerarse como un recurso en el sentido de que ciertas acciones, reglas, o eventos pueden modificar la duración del juego. Otra forma de segmentación temporal ocurre

cuando al jugador se le asigna un tiempo para completar una cierta tarea, cumplir un objetivo o simplemente hacerlo lo mejor que pueda.

En nuestro modelo no se contempla este tipo de segmentación, por lo que no podríamos segmentar los juegos mediante periodos de tiempo fijo que definan la duración del juego. Sin embargo, podría implementarse mediante la inclusión en el framework de un concepto de temporización.

La segunda forma de segmentación es la espacial, en la cual el espacio virtual del juego se descompone en sublocalizaciones. La segmentación espacial resulta de la división del mundo de juego en diferentes espacios que también dividen el juego. En estos casos, el mundo del juego se presenta no como un todo continuo sino como subespacios distintos a los que se navega por separado y que pueden tener sus propias reglas especiales. Cada subespacio puede ser mayor que lo que se puede visualizarse en la pantalla; lo que importa es si se distinguen como localizaciones separadas y si hay restricciones de juego o diferencias entre cada localización. Algunos términos que se usan para describir formas particulares de segmentación espacial incluyen *niveles*, *mapas* e incluso *mundos*. Un *nivel* es un subespacio reconocible del mundo de juego o una localización virtual diferenciada que contiene tareas que deben cumplirse para que los jugadores puedan avanzar.

Nuestro modelo nos permite soportar la segmentación espacial, ya que nos permite descomponer el espacio virtual del juego utilizando escenarios diferentes y usar reglas para pasar de un escenario al otro.

La mayoría de los juegos incluyen múltiples formas de segmentación que están interrelacionadas u ocurren al mismo tiempo. Aunque es raro que un juego tenga una sola forma de segmentación, a menudo una forma de segmentación es más destacable que las demás porque tiene un mayor impacto en el juego. También es posible que un juego use varios modos de segmentación en momentos diferentes, es decir, un juego puede estar temporalmente segmentado en una sección, mientras que el resto del juego podría estar segmentado espacialmente.

3.5.4 Entidades no monolíticas (marionetas)

Tal y como se ha ejemplificado en este capítulo, con el modelo propuesto podemos definir entidades que no sean monolíticas, es decir, que estén compuestas por varios componentes unidos por articulaciones. Esto permite construir entidades que simulen a marionetas en un juego de marionetas, que sería útil para crear entornos centrados en actividades de *storytelling*. El problema que tendríamos es que en la simulación necesitaríamos añadir al modelo puntos de anclaje, como los que utilizan las personas que mueven las marionetas reales, que al moverlos el usuario movieran partes de la marioneta simulada. Además, a estos puntos de anclaje no les debería de afectar la gravedad del escenario ya que el usuario sólo los utilizaría para mover la marioneta en

la simulación. Tampoco deberían verse en la simulación del escenario, con lo cual deberían tener una semántica distinta del resto de componentes de entidades que intervienen en la simulación.

Una posible solución a este problema sería añadir un nuevo atributo a la clase *StructuralComponent* y a la clase *StructuralJoint*. Dicho atributo booleano en la primera clase nos permitiría diferenciar entre un componente normal y un punto de anclaje de las marionetas. Cuando este atributo estuviera a true, el componente sería un punto de anclaje y tendría un aspecto visual distinto del de los componentes normales y no le afectaría la gravedad del escenario. Del mismo modo, un nuevo atributo booleano en la clase *StructuralJoint* nos permitiría diferenciar entre una articulación normal y un hilo que une un punto de anclaje a la parte de la marioneta que queremos mover. Cuando este atributo estuviera a true, la articulación se vería como un hilo que une puntos de anclaje y componentes normales y tendría un aspecto visual distinto del de las articulaciones normales.

3.6 Conclusión

En este capítulo, hemos definido en primer lugar una serie de conceptos necesarios para comprender el modelo propuesto. Posteriormente, se ha presentado dicho modelo, que nos permitirá la creación y simulación de ecosistemas con el objetivo final de soportar determinadas actividades de creación sobre superficies interactivas.

Después de explicar el modelo propuesto, hemos comentado la arquitectura del simulador, que está compuesta de tres módulos principales que se encargan de llevar a cabo el proceso de la simulación: el módulo de definición del ecosistema, el módulo de simulación y el módulo de orquestación.

A continuación, hemos presentado dos casos de estudio de implementaciones de juegos clásicos, el Arkanoid y el Pong, lo que nos ha permitido validar el modelo conceptual propuesto.

Por último, hemos visto las limitaciones que presenta el modelo propuesto y así como posibles soluciones a estas limitaciones.

Las principales ventajas del modelo presentado son que soporta la definición y simulación de una gran variedad de ecosistemas y que nos permite la reutilización de conceptos ya definidos como los tipos de entidad y los tipos de estructura, lo que nos ahorra tiempo a la hora de crear los ecosistemas.

Una vez creado un tipo de entidad, podemos crear varias entidades del tipo que acabamos de crear y modificar a nivel de entidad las propiedades que queramos. También nos permite fácilmente extender los eventos que queremos detectar en la simulación y sus correspondientes atributos, pudiendo enriquecer la variedad de ecosistemas implementables en la plataforma en el futuro.

Por tanto, el modelo presentado es extensible ya que además de los eventos se permite con un esfuerzo relativo la extensión de los tipos de datos predefinidos, articulaciones, componentes de las entidades, etc. Esto nos permite modificar de forma ágil el modelo para soportar nuevas funcionalidades futuras.

Resulta relativamente fácil añadir funcionalidades más complejas al simulador. Dado el diseño en capas del simulador se podría añadir fácilmente un framework que nos permitiera definir interacciones avanzadas o conceptos como turno y tirada que nos permitieran crear juegos que utilicen este tipo de segmentación temporal como se ha mencionado en la sección de limitaciones.

Capitulo 4. Conclusiones y trabajos futuros

En este trabajo fin de máster hemos propuesto y validado un modelo conceptual que soporta tanto la creación como la ejecución de juegos 2D con un comportamiento físico.

Después de realizar un estudio de los sistemas existentes relacionados con el que pretendemos construir y de realizar una comparativa de las diferentes características de dichos sistemas, hemos comprobado que no existe todavía un sistema que permita crear nuestros propios juegos en 2D y jugar a dichos juegos utilizando para ello una superficie interactiva.

El modelo conceptual propuesto permite soportar la creación y ejecución de juegos caracterizados por simular físicamente entidades bidimensionales y permite la reutilización de conceptos ya definidos. Además, permite definir y modificar tipos de entidades, y crear varias entidades de un tipo de entidad previamente definido, de tal forma que soporta la especificación de propiedades y acciones sobre dichos tipos de entidades. También, permite definir los eventos que queramos detectar en el juego y definir atributos para dichos eventos. Dicho modelo dispone de varios tipos de datos predefinidos para asignarlos a las propiedades, atributos de eventos o parámetros formales. Además, nos permite definir escenarios de juego con las entidades que queremos que participen en el juego y definir reglas para cada escenario que nos permitan ejecutar dicho escenario. Las entidades definidas tienen una estructura visual y tienen un comportamiento físico. En los juegos creados los usuarios pueden interactuar de forma simple con las entidades del juego sobre la superficie interactiva.

El modelo propuesto tiene una serie de limitaciones. La primera de ellas es que que no se trata íntegramente la parte de la interacción de usuario, es decir, no podemos definir maneras avanzadas de interactuar con los elementos del ecosistema. La segunda limitación del modelo es que no existe una forma de representar variables numéricas y alfanuméricas en la superficie interactiva. Otra limitación es que el modelo no contempla la posibilidad de segmentar los juegos mediante periodos de tiempo fijo que definan la duración del juego. La ultima limitación es que para implementar un juego de marionetas en la simulación necesitaríamos añadir al modelo puntos de anclaje, como los que utilizan las personas que mueven las marionetas reales, que al moverlos el usuario movieran partes de la marioneta simulada. Todas estas limitaciones quedan como trabajo futuro a tratar, extendiendo el modelo y su implementación para abordar todas ellas.

Además, como hemos comentado anteriormente, el modelo conceptual que hemos propuesto sería un paso previo para cumplir un objetivo más amplio, que consistiría en la construcción de un sistema que permita a los usuarios la creación y ejecución de sus propios juegos en 2D sobre una superficie táctil. Dicho sistema nos permitiría crear y editar nuestros propios juegos usando para ello una interfaz táctil. Como un paso intermedio, el modelo y el simulador presentado en este trabajo fin de máster ha sido utilizado para componer e implementar un editor de estructuras físicas simulables para la realización de ciertas tareas creativas. Con ese entorno se realizó un experimento con adolescentes de 16 años a los que se les pidió que utilizaran la plataforma para construir de manera creativa entidades a partir de los elementos básicos de construcción que la plataforma provee. Comparando con otro entorno no tecnológico basado en bloques de construcción de madera, se midieron una serie de variables típicamente vinculadas a la creatividad, y resultando favorable el uso de la plataforma digital en términos de originalidad. Además, en términos de facilitación de la interacción, el uso de la plataforma digital demostró que el grado de cooperación de los participantes en las tareas de creación fue significativamente mayor. Esta evidencia permite soportar la idea que una plataforma como la que se quiere construir puede ser útil dado que el grado de interacción social y discusión entre pares es importante en términos de desarrollo del pensamiento crítico y aprendizaje.

Parte de este trabajo fin de máster ha contribuido a la escritura de dos artículos de investigación que se citan más abajo. El primero se trata de una conferencia internacional especializada en *virtual learning*. El artículo fue galardonado con el "Excellence Award". El segundo trata una versión extendida publicada en un *special issue* de una revista internacional.

Alejandro Catalá, Fernando García, José Azorín, Javier Jaén, José Antonio Mocholí. Exploring Direct Communication and Manipulation on Interactive Surfaces to Foster Novelty in a Creative Learning Environment. Proceedings of the International Conference on Virtual Learning (ICVL'11)- Distinguished with the *Excellence Award*.

Alejandro Catalá, Fernando García, José Azorín, Javier Jaén, José Antonio Mocholí. Exploring Direct Communication and Manipulation on Interactive Surfaces to Foster Novelty in a Creative Learning Environment. International Journal of Computer Science Research and Application 2012, Vol. 02, Issue. 01 (Special Issue. Extended Selected Papers ICVL2011), pp. 15-24. ISSN 2012-9572 (20% acceptance rate).

Capitulo 5. Anexos

5.1 Anexos del Arkanoid

5.1.1 Estructura de los ficheros .st

5.1.1.1 Tipo de estructura para la pelota: BallStructure.st

```
STRUCTURE TYPE
BallStructure

COMPONENTS 1
ELLIPSE
# Id
0
# layer
0
# static {-1 = static; 0..n= fricCoef }
0
# position
0 0
# radius
20 20
# rotation
0

JOINTS 0
```

5.1.1.2 Tipo de estructura para las paletas: PaddleStructure.st

```
STRUCTURE TYPE
PaddleStructure

COMPONENTS 1
RECTANGLE
# Id
0
# layer
0
# static {-1 = static; 0..n= fricCoef }
-1
# position
0 0
# dimensions x and y
100 30
# rotation
0

JOINTS 0
```

5.1.1.3 Tipo de estructura para las porterías: BrickStructure.st

```
STRUCTURE TYPE
BrickStructure

COMPONENTS 1
RECTANGLE
# Id
0
# layer
0
# static {-1 = static; 0..n= fricCoef }
-1
# position
0 0
# dimensions x and y
100 50
```

```
# rotation
0

JOINTS 0
```

5.1.1.4 Tipo de estructura para los marcadores: DisplayStructure.st

```
STRUCTURE TYPE
DisplayStructure

COMPONENTS 1
RECTANGLE
# Id
0
# layer
0
# static {-1 = static; 0..n= fricCoef }
-1
# position
0 0
# dimensions x and y
50 50
# rotation
0

JOINTS 0
```

Las líneas precedidas por un # indican comentarios. Cada tipo de estructura tiene un nombre y es formado por una serie de componentes. Éstos pueden ser elipses (ELLIPSE), rectángulos (RECTANGLE) o triángulos (TRIANGLE). Si una estructura tiene diversos componentes estos pueden ir unidos por *joints*. Para todos los componentes hay que dar valor, obligatoriamente, a los siguientes atributos en orden descendente (uno en cada línea):

1) Identificador que lo diferencia de los otros componentes.
2) Capa.
3) Si, o bien es estático (-1), o bien tiene un coeficiente de rozamiento con el "suelo" (un valor de 0 a n).
4) Posición relativa al eje de coordenadas local a la estructura.
5) Dimensiones x e y (si es una elipse, son los radios en x y en y).
6) Rotación.

En el caso del Arkanoid, tan sólo cabe destacar que todas las estructuras tienen un solo componente y que, salvo la pelota, todos son estáticos pues no se moverán debido a la acción de ninguna fuerza física. Las paletas podrán moverse con los dedos o con marcadores, pero ni la gravedad ni el contacto con otras entidades las hará cambiar su posición.

5.1.2 Estructura de los ficheros .pc

5.1.2.1 Vestimenta para las paletas: PaddleCostume.pc

```
PROTOCOSTUME
PaddleCostume

STRUCTURE PaddleStructure

SKINS 1
Content\Simulator\RectangleSkin.png
```

```
# id
0
# offset
0 0
# scale
1 1
# rotation
0
```

5.1.2.2 Vestimenta para los ladrillos: GreenBrickCostume.pc

```
PROTOCOSTUME
GreenBrickCostume

STRUCTURE BrickStructure

SKINS 1
Content\Simulator\GreenBrickSkin.png

# id
0
# offset
0 0
# scale
1 1
# rotation
0
```

5.1.2.3 Vestimenta para los ladrillos: RedBrickCostume.pc

```
PROTOCOSTUME
RedBrickCostume

STRUCTURE BrickStructure

SKINS 1
Content\Simulator\RedBrickSkin.png

# id
0
# offset
0 0
# scale
1 1
# rotation
0
```

5.1.2.4 Vestimentas para los marcadores: DisplayCostume0.pc – DisplayCostume10.pc

```
PROTOCOSTUME
DisplayCostume0

STRUCTURE DisplayStructure

SKINS 1
Content\Simulator\DisplaySkin00.png

# id
0
# offset
0 0
# scale
1 1
# rotation
0
PROTOCOSTUME
DisplayCostume1

STRUCTURE DisplayStructure

SKINS 1
Content\Simulator\DisplaySkin01.png

# id
```

```
0
# offset
0 0
# scale
1 1
# rotation
0
```

```
PROTOCOSTUME
DisplayCostume2

STRUCTURE DisplayStructure

SKINS 1
Content\Simulator\DisplaySkin02.png

# id
0
# offset
0 0
# scale
1 1
# rotation
0
```

```
PROTOCOSTUME
DisplayCostume3

STRUCTURE DisplayStructure

SKINS 1
Content\Simulator\DisplaySkin03.png

# id
0
# offset
0 0
# scale
1 1
# rotation
0
```

```
PROTOCOSTUME
DisplayCostume4

STRUCTURE DisplayStructure

SKINS 1
Content\Simulator\DisplaySkin04.png

# id
0
# offset
0 0
# scale
1 1
# rotation
0
PROTOCOSTUME
DisplayCostume5

STRUCTURE DisplayStructure

SKINS 1
Content\Simulator\DisplaySkin05.png

# id
0
# offset
```

```
0 0
# scale
1 1
# rotation
0
```

```
PROTOCOSTUME
DisplayCostume6

STRUCTURE DisplayStructure

SKINS 1
Content\Simulator\DisplaySkin06.png

# id
0
# offset
0 0
# scale
1 1
# rotation
0
```

```
PROTOCOSTUME
DisplayCostume7

STRUCTURE DisplayStructure

SKINS 1
Content\Simulator\DisplaySkin07.png

# id
0
# offset
0 0
# scale
1 1
# rotation
0
```

```
PROTOCOSTUME
DisplayCostume8

STRUCTURE DisplayStructure

SKINS 1
Content\Simulator\DisplaySkin08.png

# id
0
# offset
0 0
# scale
1 1
# rotation
0
PROTOCOSTUME
DisplayCostume9

STRUCTURE DisplayStructure

SKINS 1
Content\Simulator\DisplaySkin09.png

# id
0
# offset
0 0
# scale
```

```
1 1
# rotation
0
```

```
PROTOCOSTUME
DisplayCostume10

STRUCTURE DisplayStructure

SKINS 1
Content\Simulator\DisplaySkin10.png

# id
0
# offset
0 0
# scale
1 1
# rotation
0
```

5.1.2.5 Vestimenta para la pelota: BallCostume.pc

```
PROTOCOSTUME
BallCostume

STRUCTURE BallStructure

SKINS 1
Content\Simulator\BallSkin.png

# id
0
# offset
0 0
# scale
1 1
# rotation
0
```

Las líneas precedidas por un # indican comentarios. Cada vestimenta tiene un nombre y está asociado a un tipo de estructura. Además, es formado por una serie de imágenes llamadas pieles (*skins*). Cada *skin* tiene unos atributos a los que hay que dar valor, obligatoriamente, en orden descendente (uno en cada línea):

7) Identificador del componente de tipo de estructura al que viste.
8) Desplazamiento (*offset*) respecto del centro del componente al que viste.
9) Factor de escala respecto al centro del componente al que viste. Un valor real para el eje x y otro para el eje y.
10) Rotación respecto al componente.

5.1.3 Imágenes para los *skins*

5.1.3.1 Piel para la pelota

BallSkin.png

5.1.3.2 Pieles para los marcadores

DisplaySkin00.png DisplaySkin01.png DisplaySkin02.png DisplaySkin03.png

DisplaySkin04.png DisplaySkin05.png DisplaySkin06.png DisplaySkin07.png

DisplaySkin08.png DisplaySkin09.png DisplaySkin10.png

5.1.3.3 Piel para las paletas

RectangleSkin.png

5.1.3.4 Pieles para los ladrillos

RedBrickSkin.png GreenBrickSkin.png

Capitulo 6. Bibliografía

[Abt70] Abt, C. 1970. Serious Games. Viking Press, New York.

[Ale00] Aleinikov A., Kackmeister S., Koenig R. (Eds.). “Creating creativity: 101 definitions". Midland, MI: Alden B. Dow Creativity Center, Northwoods University, 2000.

[Bege96] Andrew Begel. “LogoBlocks: A Graphical Programming Language for Interacting with the World”. 1996.

[Bro89] Brown J. S., Collins A., Dugiud P. “Situated cognition and the culture of learning. Educational Researcher vol. 18, no. 1, pp. 32-42, 1989.

[Bru60] Bruner J. “The Process of Education”. Cambridge, MA: Harvard University Press, 1960.

[Cock98] Andy Cockburn, Andrew Bryant. “Cleogo: Collaborative and Multi-Metaphor Programming for Kids”. APCHI '98 Proceedings of the Third Asian Pacific Computer and Human Interaction. 1998.

[Csi88] Csikszentmihalyi M., Csikszentmihalyi I., “Optimal Experience. Psychological Studies of Flow in Consciousness”. Cambridge University Press, 1988.

[Dew63] Dewey J. “Experience and Education”. New York: Collier, 1963.

[Dun45] Duncker, Karl; Lees, Lynne S.(Trans). “On problem-solving”. Psychological Monographs, Vol 58(5), 1945, i-113.

[Fern06] Ylva Fernaeus & Jakob Tholander. “Finding design qualities in a tangible programming space”. CHI '06 Proceedings of the SIGCHI conference on Human Factors in computing systems. 2006.

[Gro04] Gros B., Grup F9. “Pantallas, juegos y alfabetización digital”. Comunicación y Pedagogía, vol. 208, pp. 25-27, 2004.

[Gro07] Gros B. “The Design of Learning Environments Using Videogames in Formal Education”. DIGITEL, pp. 19-24, The First IEEE International Workshop on Digital Game and Intelligent Toy Enhanced Learning (DIGITEL'07), 2007.

[Hor08] Hornecker E. “I don’t understand it either, but it is cool”- Visitor Interactions with a Multi-touch Table in a Museum. IEEE Tabletops and Interactives Surfaces, IEEE

International Workshop on Horizontal Interactive Human-Computer Systems, pp. 121-129, 2008.

[Horn07] Michael S. Horn, Robert J.K. Jacob. “Designing Tangible Programming Languages for Classroom Use”. TEI '07 Proceedings of the 1st international conference on Tangible and embedded interaction. 2007.

[Gall08] Daniel Gallardo, Carles F. Julià and Sergi Jordà. “TurTan: a Tangible Programming Language for Creative Exploration”. Music Technology Group, Universitat Pompeu Fabra. In Proceedings of Tabletop. 2008

[Juli09] Carles F. Julià, Daniel Gallardo, and Sergi Jordà. “TurTan: Un Lenguaje de Programación Tangible Para el Aprendizaje”.

[Khan06] Madhur Khandelwal. “Teaching Table.A tangible mentor for pre-kindergarten math education”. Tesina de master.

[Khan07] Madhur Khandelwal, Ali Mazalek. “Teaching table: a tangible mentor for pre-k math education”. TEI '07 Proceedings of the 1st international conference on Tangible and embedded interaction. 2007.

[Kelle06] Caitlin Kelleher. “Motivating Programming: using storytelling to make computer programming attractive to middle school girls”. Doctoral Thesis. November, 2006.

[Kelle07a] Caitlin Kelleher,Randy Pausch. “Using storytelling to motivate programming”. Magazine Communications of the ACM - Creating a science of games CACM Volume 50 Issue 7, July 2007.

[Kelle07b] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. Storytelling Alice Motivates Middle School Girls to Learn Computer Programming”. CHI 2007 Proceedings • Programming By & With End-Users. April 28-May 3, 2007.

[Kol94] Kolb D. A. “Experiential Learning: Experience as the source of learning and development”. Englewood CLiffs, NJ: Prentice-Hall, 1984.

[Lamp07] Matthias Lampe and Steve Hinske. “The Augmented Knight’s Castle – Integrating Mobile and Pervasive Computing Technologies into Traditional Toy Environments”. In: Carsten Magerkurth, Carsten Roecker (Eds.): Concepts and technologies for Pervasive Games - A Reader for Pervasive Gaming Research. Vol. 1, Shaker Verlag, Aachen, Germany, 2007.

[Leit08] Jakob Leitner, Michael Haller, Kyungdahm Yun, Woontack Woo, Maki Sugimoto, Masahiko Inami. “IncreTable, a mixed reality tabletop game experience”. ACE '08 Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology. 2008.

[Lu11] Fei Lu, Feng Tian, Yingying Jiang, Xiang Cao, Wencan Luo, Guang Li, Xiaolong Zhang, Guozhong Dai and Hongan Wang. “ShadowStory: Creative and Collaborative Digital Storytelling Inspired by Cultural Heritage”. Proceeding CHI '11 Proceedings of the 2011 annual conference on Human factors in computing systems.

[Mage03] C. Magerkurth, R. Stenzel, Dr. Dr. N. Streitz, E. Neuhold. “A Multimodal Interaction Framework for Pervasive Game Applications”. Workshop at Artificial Intelligence in Mobile System 2003(AIMS 2003), Seattle, USA, Oct. 12, 2003.

[Maza07] Ali Mazalek, Basil Mironer, Dana Van Devender. “The TViews Table Role-Playing Game”. 4th International Symposium on Pervasive Gaming Applications, PerGames 2007.

[McF02] McFarlane A., Sparrowhawk A., Heald Y. “Report on the educational use of games”. Teem: Teachers Evaluating Educational Multimedia, 2002.

[Park08] Amanda J. Parkes, Hayes Solos Raffle, Hiroshi Ishii. “Topobo in the wild: longitudinal evaluations of educators appropriating a tangible interface”. CHI '08 Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems. 2008.

[Pia67] Piaget J. “Six psychological studies”. New York: Vintage books, 1967.

[Repe00] Alexander Repenning. “AgentSheets®: an Interactive Simulation Environment with End-User Programmable Agents”.Journal of Artificial Societies and Social Simulation vol. 3, no. 3, 2000.

[Resn09] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond,Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for all. Communications of the acm, november 2009, vol. 52, no. 11.

[Smit10] J. David Smith and T. C. Nicholas Graham. “Raptor: Sketching Games with a Tabletop Computer”. Futureplay '10 Proceedings of the International Academic Conference on the Future of Game Design and Technology. 2010.

[Smit09] John David Smith. “Raptor: Sketching Video Games With a Tabletop Computer”. Thesis. 2009.

[Stre01] Streitz, N.A., Tandler, P., Müller-Tomfelde, C., Konomi, S. “Roomware: Towards the Next Generation of Human-Computer Interaction based on an Integrated Design of Real and Virtual Worlds”. In: J. A. Carroll (Ed.): Human-Computer Interaction in the New Millennium, Addison Wesley, 553-578, 2001.

[Suzu95] Hideyuki Suzuki, Hiroshi Kato. “Interaction-level support for collaborative learning: AlgoBlock—an open programming language”. CSCL '95 The first international conference on Computer support for collaborative learning. 1995.

[Vyg78] Vygotsky L.S. “Mind in Society: The development of Higher Psychological Processes”. Harvard University Press, 1978.

[Wing06] Jeannette M. Wing. Magazine Communications of the ACM - Self managed systems. Volume 49 Issue 3, March 2006.