EXTENDIENDO UADVENTURE CON FUNCIONALIDADES DE GEOPOSICIONAMIENTO

VÍCTOR MANUEL PÉREZ COLADO

MÁSTER EN INGENIERÍA INFORMÁTICA, FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Ingeniería Informática

Febrero 2017

Calificación: Sobresaliente 10

Directores:

Baltasar Fernández Manjón
Iván Martínez Ortiz

Autorización de Difusión

VÍCTOR MANUEL PÉREZ COLADO

27 de febrero de 2017

El/la abajo firmante, matriculado/a en el Máster en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Extendiendo uAdventure con funcionalidades de geoposicionamiento", realizado durante el curso académico 2016-2017 bajo la dirección de Baltasar Fernández Manjón e Iván Martinez Ortiz en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

D. Víctor M. Pérez Colado D. Baltasar Fernández Manjón D. Iván Martínez Ortiz

Resumen

El crecimiento de las tecnologías móviles ha creado nuevas posibilidades para el desarrollo de videojuegos. En este ámbito, sacar partido de los sensores existentes en los terminales puede dar lugar a nuevas mecánicas que integren todavía más los juegos con la realidad. Este es el caso de los juegos geoposicionados. El gran éxito de videojuegos como Pokémon GO ha suscitado un gran interés social en dicho campo. Dentro del ámbito educativo, los juegos serios no pueden ser una excepción en este movimiento. Por ello, las nuevas generaciones de juegos serios pueden basarse en ideas del campo del geoposicionamiento (y de la realidad aumentada) para proporcionar enseñanzas más interactivas y auténticas en las que se mezclen elementos reales y elementos de juego en el proceso de aprendizaje. La plataforma de creación de juegos denominada uAdventure, que aspira a convertirse en el sucesor de eAdventure tras el declive de Java, ha dado el salto definitivo a los dispositivos móviles a través de una reimplementación basada en el motor de videojuegos Unity3D. En este proyecto, además de hacer una reingeniería que ha permitido mejorar y pulir uAdventure, se han incorporado nuevas funcionalidades orientadas a sacar provecho del GPS incorporado en los dispositivos móviles. De esta forma se han creado nuevas mecánicas de juego tales como el acceso a zonas geoposicionadas, la visualización de elementos espaciales o el acercamiento a elementos en el espacio, dando lugar a nuevas interacciones conectadas directamente con el entorno que favorecen un aprendizaje a través de la experimentación. Más concretamente, se han desarrollado una serie de *plugins* para la plataforma uAdventure que permiten mezclar nuevos elementos geoposicionados y elementos del modelo de uAdventure en un nuevo tipo de escena de tipo mapa. Este mapa permitirá a los jugadores ser representados en el entorno de juego y convertir el entorno real en un elemento virtual. Además, dado que el geoposicionamiento puede fallar en interiores, se incorporan soporte de códigos QR que permite realizar posicionamiento de bajo coste en interiores. Para la implementación de los *plugins*, la arquitectura de uAdventure se ha mejorado drásticamente en términos de extensibilidad, pudiendo incorporar nuevas mecánicas sin necesidad de ser un gran experto en la plataforma. Finalmente, el proyecto cubre la incorporación del geoposicionamiento en las analíticas de aprendizaje, así como la ejecución de un caso de prueba con usuarios que ha permitido una validación inicial del desarrollo realizado y de las nuevas posibilidades de las analíticas resultantes teniendo en cuenta el geoposicionamiento.

Palabras clave

eAdventure, uAdventure, Herramienta de Autoría, Unity, Geoposicionamiento, GPS, QR, Juegos Serios, Pokémon GO, Learning Analytics, xAPI

Abstract

Mobile technologies evolution has created new possibilities for game development. Taking advantage of the mobile device sensors it is possible to develop new game mechanics that better integrate games with reality. This is the case of geolocated games. The big success of videogames like Pokémon GO has aroused a big social interest in that field. Serious games cannot be an exception for this movement. New generations of serious games can take advantage of geolocation (and also augmented reality) to create more interactive and realistic learning experiences that mix up real and game elements in the learning process. The serious games authoring tool called uAdventure, that aims to become the successor of eAdventure after all the present Java issues (e.g. deployment), has made the final step towards mobile devices by being reimplemented on top of Unity3D videogame engine. In this project, not only a reengineering that has improved and polished uAdventure has been done but it has also included new functionalities oriented to take advantage of the GPS already present in mobile devices. This way, new game mechanics have been developed such as entering geopositioned zones or especial element visualizations, creating new kinds of interactions directly connected with the environment to allow for a more experience-based learning. That way a new series of plugins for uAdventure have been developed, allowing to mix up new geopositioned elements with already existing uAdventure model elements in a brand new map scene. This map will allow players to be represented inside the game and transform the real world into a virtual interactive element. In addition, due to the possible fails of indoors geopositioning, QR code support is provided as a low-cost alternative way to implement indoors positioning. In order to develop the plugins, uAdventure's architecture has been drastically improved in terms of extensibility, allowing it to incorporate new mechanics without being an expert in the uA platform. Finally, the project has not only provided the support for geopositioning in uA linked with the learning analytics capabilities, but also has done an experiment with users to initially validate the development made and the new possibilities of the resulting analytics based on geopositioning.

Keywords

eAdventure, uAdventure, Authoring Tools, Unity3D, Geopositioning, GPS, QR, Serious Games, Pokémon GO, Learning Analytics, xAPI

Índice de contenidos

Índice de figuras

Agradecimientos

En primer lugar, quiero agradecer a Baltasar Fernández Manjón la oportunidad de unirme al grupo de investigación e-UCM y brindarme la oportunidad de realizar este proyecto en el que intervienen mi pasión, los videojuegos, y una buena motivación social, la educación, donde poder aportar mi granito de arena a mejorar la sociedad.

También quiero agradecer a Iván Martínez Ortiz todo el esfuerzo, seguimiento, apoyo e ideas que me ha dado para la realización de este proyecto, y por haber sido uno de los primeros profesores que me enseñó el camino hacia ser el ingeniero que hoy soy.

Agradecer también a Federico Peinado que siempre me haya brindado todo el apoyo y oportunidades que ha podido, siendo casi un profesor-padre para mí, pues siempre se ha implicado con su pasión por los videojuegos en todas las ideas que he tenido.

Agradecer a todos los compañeros del Aula 16 que me permitan pasar ratos increíbles que hacen parecer mi trabajo una utopía. Sin vosotros habría terminado el proyecto mucho antes.

Agradecer a mi hermano, Iván, que me haya ayudado siempre que lo he necesitado.

Agradecer a mi familia y amigos todo el apoyo y comprensión en los momentos de más estrés, en los que me han allanado el camino y me han dado todo el cariño que he necesitado. Sin vosotros NADA sería posible.

Y, por último, quiero agradecerle a Cristina que haga de mi vida un lugar donde sentir que soy un súper héroe capaz de lograr todo lo que me propongo y apoyarme siempre en mis propósitos.

Para Alison,

Porque eres una hermana maravillosa y sé que algún día encontrarás tu lugar y harás cosas increíbles.

Capítulo 1. Introducción

Los juegos serios son aquellos juegos que no tienen como principal y único propósito el entretenimiento [1]. Por tanto, el ámbito de los juegos serios es muy amplio y diverso en cuanto a género, tipo, contenido y propósito. No obstante, uno de los propósitos principales para los que se han venido utilizando los juegos serios es el educativo, tanto en enseñanzas regladas como en formación y capacitación en el ámbito empresarial [2]. En este contexto, la creación de juegos serios se ha venido realizando utilizando diferentes herramientas y metodologías de desarrollo. Una aproximación ha consistido en el uso de herramientas de autoría que facilitan la creación de juegos serios sin necesidad de programar. En este sentido, la plataforma uAdventure, que es un elemento esencial de este trabajo, es una herramienta de autoría de juegos serios [3]. La plataforma uAdventure se centra en la autoría de juegos de aventura gráfica "*point and click*" que es un género particularmente adecuado para el aprendizaje, además de proporcionar un buen equilibrio entre aplicabilidad y coste [4], [5].

Cabe destacar que la plataforma uAdventure es una evolución tecnológica y reimplementación de la plataforma eAdventure[1] (que lleva casi diez años en desarrollo) sobre el entorno Unity3D[2] que se encuentra todavía en fase de desarrollo [6]. Esta transformación ha sido necesaria para permitir la continuación del proyecto y para dar un soporte adecuado a los dispositivos móviles. Este proyecto, parte de esta esta versión inicial de la plataforma uAdventure con el objetivo de aprovechar las ventajas que aporta con la integración de tecnologías móviles.

Los dispositivos móviles proveen principalmente de dos ventajas especialmente relevantes para la creación de juegos serios: la conectividad y una capa de sensores disponibles de forma casi estándar en todos los terminales. Dichos sensores, en general orientados a la obtención de información de medios físicos, abren la posibilidad a la creación de nuevos formatos de videojuegos, basados en la generalmente conocida realidad aumentada (RA) [7]. Los sensores se dividen en ópticos, permitiendo acceso a imágenes captadas desde el dispositivo; físicos, siendo estos sensores o sistemas capaces de obtener información del contexto físico en el que se encuentra el dispositivo; o electrónicos, que permiten comunicarse con diversos

[1] http://e-adventure.e-ucm.es

[2] https://unity3d.com

protocolos de rango cercano con otros dispositivos con el objetivo de extraer información de ellos (e.g. NFC, Bluetooh).

uAdventure se fundamenta en los principios de ubicuidad y movilidad. Por ello, los juegos desarrollados en uAdventure deben poder beneficiarse de la contextualización del jugador a partir de dicha capa de sensores disponibles en el dispositivo. En la actualidad, estas capacidades se explotan mínimamente aprovechando exclusivamente las capacidades multiplataforma de uAdventure, en particular el soporte de dispositivos móviles. Sin embargo, la contextualización es un concepto más amplio que puede contemplar también el hecho de proveer al juego la información sobre la situación física en la que se está ejecutando y que permite el desarrollo de tipos de juego ubicuo que se nutren de información del mundo real de origen gráfico o de posicionamiento. Por tanto, **el principal objetivo de este trabajo es el de incluir en uAdventure aspectos de geoposicionamiento para que se puedan explotar en la creación de nuevos tipos de juegos.**

1.1. Juegos ubicuos

Los juegos ubicuos, del inglés *pervasive games* o también *ubiquitous games*, son la categoría de juegos que incluyen información del mundo real en su experiencia de juego [8], [9]. El género, aunque bastante amplio, destaca por el subgénero de juegos que se extienden en base a la situación espacial. Dentro del subgénero de situación espacial se sitúan tanto los juegos geoposicionados, como los juegos basados en realidad aumentada. Además de las posibilidades de incluir el posicionamiento, los juegos ubicuos pueden extenderse teniendo en cuenta el tiempo, en el que el juego se comporta de forma diferente en base al momento en el que se juegue, y estos aspectos se pueden complementar con otros más sociales, donde las acciones de otros jugadores participan el desarrollo de la experiencia del jugador principal.

Gracias a esta fuerte conexión del juego con la realidad, al desarrollarse juegos serios ubicuos estos resultan más eficaces, al promover una enseñanza continua fuera de los centros de estudio que se basa en la experiencia y en la interacción con el entorno [10]. Esta eficacia reside en tres aspectos fundamentales. El primero que se ha detectado, es el incremento en la motivación que generan los juegos de realidad aumentada. Esta motivación genera interés y el interés mejora el aprendizaje. El segundo es la transformación del medio del conocimiento al propio escenario, impulsando la deducción por parte del jugador obteniendo un conocimiento

conexo con la realidad. El tercero es que, al ser un juego serio, los elementos que intervienen pueden formar parte de la metodología real de estudio por lo que el juego se acerca a la simulación.

1.1.1. Juegos basados en geoposicionamiento

Los juegos basados en geoposicionamiento son aquellos que utilizan como fuente de datos la información de sistemas de geolocalización como GPS o GLONASS. Entre sus mecánicas, este género de videojuegos suele nutrirse de un mapa aumentado con elementos relevantes para el juego que se mezclan con la información real del mapa. De esta forma, se forma una realidad mixta, similar en algunos aspectos a la realidad aumentada, en la que se requerirá de la movilidad del jugador para el desarrollo y avance del juego.

Un ejemplo claro son los videojuegos de Niantic[3]. Su primer éxito, Ingress[4], fue avalado por Google y tuvo mucho éxito como uno de los pioneros del geoposicionamiento como método de juego. Su segundo y mayor éxito fue Pokémon GO[5], que fue avalado por Nintendo. El éxito de Pokémon GO (que mezcla geoposicionamiento y realidad aumentada) ha sido reconocido a nivel mundial [11] y ha dado un gran impulso a la aceptación de los juegos ubicuos. Precisamente la acogida de este juego, especialmente entre el público más joven, es una de las motivaciones por los cuales extender uAdventure para dar soporte a este tipo de juegos y aprovechar la experiencia y la motivación que ha generado en el público y mejorar también con ello la salud de los jóvenes incrementando su actividad física [12].

1.1.2. Realidad aumentada

La realidad aumentada, en su definición, es la mezcla de elementos reales tangibles con elementos virtuales a través de dispositivos electrónicos [7], [13], [14]. Por lo general, se considera realidad aumentada a la mezcla de imágenes obtenidas por una cámara con elementos virtuales. En su relación con el marco educativo, la realidad aumentada permite a los estudiantes beneficiarse de la interacción con el mundo real en el proceso educativo. Gracias a la realidad aumentada, los juegos serios pueden utilizar elementos virtuales que orienten al jugador para adquirir de forma contextualizada e interesante los conocimientos ya existentes en el entorno en

[3] https://www.nianticlabs.com/

[4] https://www.ingress.com/

[5] http://www.pokemongo.com/

que se encuentran. No obstante, este aspecto de la realidad aumentada queda fuera de los objetivos de este trabajo.

Capítulo 2. Introduction

Serious games are a kind of game that doesn't have as main and only purpose entertainment [1]. Therefore, the serious game domain is very wide and diverse in genre, type, content and purpose. Nevertheless, one of the main purposes they have been used is the educational one, from class lessons to business environments [2]. In that context, the creation of serious games has been implemented by using many different tools and with various development methodologies. One approach is the use of authoring tools that simplify the creation of serious game without the need of dealing with programming. In that sense, uAdventure platform, that is a tool for this kind of job as it is a serious game authoring tool [3]. uAdventure platform allows for the authoring of "point and click" graphic adventure games that is a genre particularly appropriate for the learning. In addition, this genre, provides a very good balance between applicability and cost [4], [5].

It's important to note that uAdventure platform is a technological evolution and reimplementation of the eAdventure[6] platform (that has been running for almost ten years) over the Unity3D engine and that is still at the development stage [6]. This reimplementation has been necessary to simplify the maintenance of the project and to give full support to mobile devices taking advantage of the new possibilities that mobile technologies could bring to games. This project, starts with the initial beta version of uAdventure platform.

Mobile devices provide mainly two kind of advantages especially relevant for the creation of serious games: mobile connectivity and a sensor layer included almost in every device. Those sensors, in general oriented to the obtaining physical data, open the possibility to the creation of new formats of videogames based on the geolocation or in the augmented reality (AR) [7]. The sensors are divided in optical, when they provide optical information captured from the device; physical, by providing information about physical context of the device; or electronic, that are capable of communicate with other devices by using closed range protocols with the idea of extract information from them such as NFC or Bluetooth.

uAdventure is based in the principles of ubiquity and mobility. Because of that, games developed inside uAdventure must be able to take advantage of the contextualization of the player based on this layer of sensors already present in the device. At the beginning of this

[6] http://e-adventure.e-ucm.es

project, these capabilities are minimally exploited in uAdventure, taking advantage only of the multiplatform and connectivity capabilities. However, the contextualization is a very wide concept that can contemplate also the creation of new game mechanics based on physical information of the device motion or optical data from the surrounding environment, that allows the creation of ubiquitous games that use this real world contextual information. Therefore, **the main objective of this project is to include inside uAdventure aspects of geopositioning to make possible to create new kind of location-based games.**

2.1. Ubiquitous games

Ubiquitous games, also known as, pervasive games are a game category that include real world information in the gaming experience [8], [9]. The genre is very wide and diverse that includes the subgenre of games that are based of spatial information. Inside of this subgenre are comprised both geolocated games and augmented reality games. In addition to the possibilities of including the location, ubiquitous games can be extended considering time, acting differently depending on the moment the user plays it (e.g. day and night cycles) and can also be complemented with other social information, where the actions of other players participate in the experience of the main player.

Thanks to this strong connection of the game with the reality when the game is played it is frequently more effective and engaging. One of the key characteristics is that those games promote a more authentic learning outside of the learning centers and more based on the experimentation and interaction with the actual environment [10]. This effectiveness is generated by three fundamental aspects. The first being detected is the increase of the motivation when playing AR games. This motivation generates interest and the interest improves the learning process. The second aspect is the environment where the knowledge is provided, using real world as the game scenario, boosting deduction from player and resulting in a more authentic knowledge. The third and last one, is that, because it's a serious game, the elements that are present in the game can be part of an actual methodology being the game closer to simulations.

2.1.1. Games based in geopositioning

The games based in geolocation are those that use as source of data the information from geolocation systems such as GPS or GLONASS. This genre of videogames includes new game

mechanics that are usually based on a map augmented with game elements that are mixed with the real information in the map. This way, mobility of the player is required for the progress in the game.

A clear example of success in this genre are the videogames from Niantic[7]. Its first success, Ingress[8], was supported by Google and had a very big impact as one of the pioneers' games to use geopositioning as a game mechanic. Niantic second and biggest success has been Pokémon GO[9] that has been supported by Nintendo. The success of Pokémon GO (that mixes up both geopositioning maps and AR) has been confirmed worldwide [11] and has given a big impulse to the acceptation of ubiquitous games. Precisely, the good reception of Pokémon GO, especially among the youth people, is one of the motivations for extending uAdventure to include support for this kind of games. This will allow to take advantage of the experience and motivation already generated into these games and improve also the health of the young people by increasing their physical activity [12].

2.1.2. Augmented Reality

Augmented Reality (AR) by its definition, is the mix of real physical elements with virtual elements by using electronic devices. Generally, it is considered AR the mix up of optical information from a camera with computer generated visual elements [7], [13], [14]. Related to the educative frame, AR can help students to benefit from the interaction with real world in the learning process. Thanks to AR, SG can use visual elements that orientate the player to acquire in a contextualized and interesting way the knowledge already existing in the environment they are. Nevertheless, this aspect of AR is not in the scope of this current project but it is describing as a future work.

[7] https://www.nianticlabs.com/

[8] https://www.ingress.com/

[9] http://www.pokemongo.com/

Capítulo 3. Punto de partida: eAdventure y uAdventure

eAdventure, y su evolución en Unity3D, uAdventure, son plataformas que tienen como uno de sus objetivos principales simplificar la creación de los juegos serios sin necesidad de programar, de este modo también se mejora la creación de los juegos ya que se posibilita que los educadores participen con un papel activo durante el desarrollo, incluso permitiendo que puedan desarrollar juegos por sí mismos.

La democratización de la tecnología móvil ha permitido que el número de usuarios móviles se ha multiplicado hasta superar a los usuarios de ordenadores clásicos. El resto del capítulo proporciona una breve introducción sobre la herramienta eAdventure, centrándose en las en las innovaciones que introdujo en el momento de su creación, los problemas que ha adquirido en el tiempo y sus transformaciones hasta llegar a su nueva reimplementación bautizada como uAdventure. Cabe destacar que la "u" de uAdventure ha sido escogida para destacar que la plataforma tiene como un objetivo principal abordar el aprendizaje ubicuo (u-Learning [15]) y para representar la relación con Unity3D.

3.1. eAdventure

eAdventure es una plataforma para el desarrollo de juegos serios, en formato de aventura gráfica, desarrollado sobre Java SE por el grupo de investigación e-UCM, de la Facultad de Informática de la Universidad Complutense de Madrid. En esencia, eAdventure provee una herramienta de creación de aventuras gráficas para personas sin ningún tipo de conocimiento de programación. El objetivo es que pueda ser utilizada por usuarios como profesores o expertos en un dominio concreto (e.g. medicina) para desarrollar juegos serios con un presupuesto reducido y que puedan ser aplicados en distintos entornos educativos [3].

El formato de aventura gráfica es, según Van Eck [4], un formato muy adecuado para la enseñanza. Esto se debe a que permite mantener un equilibrio entre el entretenimiento y el aprendizaje. Si un juego serio pretende ser una herramienta educativa de relevancia, este debe proporcionar una enseñanza de calidad y que permita profundizar en el conocimiento. Por ello, a través de la narrativa, el jugador puede experimentar y poner en práctica los conocimientos adquiridos, a la vez que los refuerza y extiende. Sin embargo, para que un juego serio sea eficaz es necesario mantener el interés del jugador durante el desarrollo del juego. Por ello, se debe mantener un equilibrio entre entretenimiento y enseñanza. Este equilibrio, en el caso de

eAdventure, se alcanza a través de la trama del juego, los recursos y mecánicas que se proporcionan. Al estar destinado a usuarios sin ningún conocimiento de programación, el desarrollo de mecánicas propias resultaría demasiado complejo mientras que, desarrollar una buena trama no requiere un gran conocimiento de informática (pero si un buen guion o historia). Además, el género de aventura gráfica ha sido uno de los más exitosos de la historia del videojuego, con títulos como The Secret of Monkey Island[10], Broken Sword[11], la saga Blackwell[12] o el reciente Broken Age[13].

El presupuesto del juego, además, influye en el formato, ya que para poder desarrollar mecánicas elaboradas es necesario tener una formación especializada (e.g. programación, gráficos) que supone un alto coste a la hora de formar o contratar personal. Dado que las aventuras gráficas siguen básicamente el mismo formato y no requieren de mecánicas nuevas para especializar los juegos, es un formato idóneo para el desarrollo de una plataforma de autoría. Además, eAdventure puede ser utilizada para la modificación y mejora de videojuegos ya existentes creados con la misma plataforma, permitiendo la creación de juegos especializados a partir de juegos genéricos o incluso versiones distintas de los juegos. Para ello, eAdventure puede importar juegos ya existentes y modificarlos. Además, incluye herramientas propiamente orientadas a la educación como son la capacidad de personalización del juego en base al usuario, o el soporte de distintos tipos de evaluación [16] (o "*assestment profiles*").

Finalmente, la propiedad clave para su éxito es la integración con las plataformas ya existentes de e-Learning. Por ello, el juego resultante debe ser flexible y multiplataforma, y a su vez, debería implementar estándares ya existentes en el e-Learning para simplificar su despliegue en sistemas de gestión de e-learning (i.e. LMS) que ya incluyen soporte a dichos estándares como, por ejemplo, Moodle [17]. Debido a estos requisitos, en ese momento se escogió Java SE ya que puede ser ejecutado en los principales sistemas operativos y en navegadores web, siendo este último detalle crucial para su integración con LMS. Por ejemplo, de este modo si se implementa con el estándar SCORM [18] el juego puede lanzarse desde el LMS y además puede

[10] https://es.wikipedia.org/wiki/Monkey_Island

[11] https://es.wikipedia.org/wiki/Broken_Sword

[12] https://es.wikipedia.org/wiki/Blackwell_(serie)

[13] https://es.wikipedia.org/wiki/Broken_Age

servir como herramienta de evaluación ya que este estándar soporta la comunicación para devolver resultados del juego al LMS.

Sin embargo, el envejecimiento de Java [19] y las guerras comerciales han provocado que la plataforma Java no sea tan atractiva para desarrollar juegos. No obstante debido a los problemas de seguridad [20] la mayoría de navegadores han dejado de darle soporte frente a tecnologías web nativas. Este factor se ha convertido en el principal problema para continuar con Java SE como principal soporte a través de la JVM.

Por otro lado, Java SE ha supuesto otro impedimento en la expansión hacia tecnologías móviles. Tanto Android como iOS no cuentan con implementaciones de la JVM dejando fuera del entorno a eAdventure. Por ello, junto con el factor web, eAdventure ha tratado de adaptarse a nuevas plataformas móviles en diversos proyectos.

El primer proyecto de adaptación, eAdventure para Android [14] (que fue un proyecto interno del grupo e-UCM) consistió en la adaptación mediante libGDX[15] (librería para el desarrollo de juegos multiplataforma) de eAdventure para hacerlo funcionar en Android. Para ello, se reimplementaron las bases del intérprete en dicho motor, añadiendo nuevas funcionalidades específicas para la plataforma como un repositorio de juegos que permite buscar y descargar juegos existentes y, además, nuevas funciones interesantes para la aplicación de lo que se va a implementar en este proyecto relacionadas con el control mediante GPS y códigos QR en la ejecución. Pese al éxito desde el punto de vista técnico, su impacto desde el punto de vista de la comunidad de usuarios de eAdventure fue limitada ya que no había una herramienta de edición integrada que permitiera añadir dichas características sobre los juegos de una forma sencilla. Por ello, y por motivos de obsolescencia de la arquitectura, el proyecto se reinició con el objetivo de solventar estos problemas que eAdventure tenía de raíz.

El reinicio de eAdventure sobre libGDX permitiría la realización de nuevos formatos de juegos compatibles con Android, sobre la cual podrían implementarse nuevas mecánicas, incluyendo ricas animaciones y efectos gráficos. El proyecto, que fue rebautizado como Mokap [21]fue desarrollado por Dan Cristian Rotaru y Antonio Calvo Morata en conjunto con el grupo de e-UCM y resultó un éxito relativo dentro de Android debido a que el alcance de Mokap era

[14] https://github.com/e-ucm/eadventure-legacy-android

[15] https://libgdx.badlogicgames.com/

reducido (no tenía todas las características disponibles in eAdventure) con objeto proporcionar una experiencia más rica desde el punto de la usabilidad. Aun así, Mokap ha contado con juegos reales aplicados como es el caso del juego para entrenar a niños con implante coclear denominado Listen With Lemmur, desarrollado en colaboración con el University College de Londres [22] .

Finalmente, pese a ambos proyectos, eAdventure no terminó de reiniciarse hasta el año 2016, cuando Iván José Pérez Colado y Piotr Marzal elaboraron proyectos integrados, partiendo de una perspectiva diferente, es decir, utilizando Unity3D como base para la herramienta.

3.2. uAdventure

uAdventure nace como solución para permitir crear juegos serios ejecutables en los nuevos dispositivos móviles, así como para posibilitar que juegos creados en eAdventure se puedan actualizar y ejecutarse también en dispositivos móviles.

Su desarrollo comenzó en 2016 mediante dos proyectos con el objetivo de reimplementar los dos módulos principales que componen eAdventure: la herramienta de autoría y el intérprete para los juegos generados con la herramienta de autoría. En primer lugar, Iván José Pérez Colado desarrolló un intérprete para los juegos de creados con eAdventure [6]. Este primero proyecto se abordó dando soporte a juegos existentes para eAdventure con una complejidad incremental (desde el punto de las características de eAdventure utilizadas). Paralelamente, Piotr Marzal centró sus esfuerzos en el desarrollo de una reimplementación del editor de eAdventure sobre el editor de Unity [23]. Pese a ser un proyecto de fin de grado, la implementación final del editor abarcaba prácticamente todas las posibilidades de edición existentes en eAdventure, aunque contaba con diversos problemas de estabilidad.

Si bien el proyecto que cubría la parte de intérprete y ejecución fue una reimplemetación completa, el editor de eAdventure sobre Unity era una reimplementación de la interfaz sobre Unity GUI, pero se sostenía sobre una “traducción” de la parte del núcleo del modelo que le mantenía desacoplado de Unity y heredando la filosofía de Java para su arquitectura. En las últimas fases de ambos proyectos se logró una integración mínima de ambos, capaz de coexistir en el mismo entorno, aunque de muy baja estabilidad. Por ello, como parte de este proyecto se ha colaborado con Iván José Pérez Colado para solventar los errores de la herramienta y para

adquirir el conocimiento necesario para poder realizar no sólo mejoras puntuales si no también modificaciones arquitectónicas con soltura.

La motivación de uAdventure es clara: suplir los problemas heredados de Java cubriendo además las nuevas necesidades de portabilidad. Esta portabilidad es la que permite el avance hacia dispositivos móviles, dando el salto al m-Learning [24], y mezclando el propósito original de eAdventure en el e-Learning [3] dando lugar a una plataforma de aprendizaje ubicuo en el paradigma del u-Learning [15]. Adicionalmente, Unity como motor de juegos evita tener que volver implementar algunas de los componentes de eAdventure ya que se encuentran incluidos directamente en Unity3D [25] (el paper referenciado forma parte de este trabajo de fin de máster).

Adicionalmente y como parte más novedosa uAdventure tiene en cuenta aspectos más novedosos para facilitar la evaluación y seguimiento de los estudiantes dentro del juego utilizando analíticas del aprendizaje (*Learning Analytics*) [26], que se basa en aprovechar las técnicas de aprendizaje máquina y *big data* para realizar un seguimiento de las actividades de los estudiantes y poder mejorar el proceso educativo. En este sentido, el diseño e implementación de uAdventure permite realizar un seguimiento en tiempo real de todas las interacciones del estudiante con todos los elementos del juego, decisiones tomadas y progreso a lo largo del juego. Este seguimiento permite realizar un análisis de las interacciones del alumno tanto para evaluar su rendimiento individual, como para compararlo frente a sus compañeros. El soporte en uAdventure de las analíticas de aprendizaje se complementa con el trabajo realizado en el proyecto europeo del H2020 RAGE [27], donde el grupo e-UCM es responsable del desarrollo de una plataforma para facilitar la aplicación de las analíticas de aprendizaje aplicadas a los juegos serios. Para ello se utiliza el estándar de comunicaciones en base a la experiencia llamado Experience API (xAPI) [26]. Su perfil orientado a la educación ha sido especificado por Ángel Serrano Laguna en conjunto con el grupo e-UCM y ADL [28]. Para realizar la traza de las actividades, RAGE proporciona, entre otras, una herramienta sobre Unity que permite el envío de trazas en dicho formato, orientado a la experiencia, hacia la infraestructura de servidor de RAGE para su almacenamiento, procesamiento y visualización. Parte de los resultados del presente trabajo servirán como casos de estudio para el proyecto RAGE.

De manera resumida, uAdventure ha permitido solucionar los problemas del envejecimiento de Java e introducir nuevas características novedosas como el soporte a las

analíticas del aprendizaje. Asimismo, desde el punto de vista de las funcionalidades de autoría se han aportado novedades que mejoran su facilidad de uso como, por ejemplo, nuevos editores para diálogos, efectos y condiciones que los representan de manera más visual. No obstante, la versión actual de uAdventure ha arrastrado algunos elementos de la antigua herramienta eAdventure que ya no son necesarios. Por tanto, uAdventure requiere una evolución y adaptación desde el punto de vista de su arquitectura, con objeto de aprovechar de manera más efectiva las características que ofrece la plataforma Unity3D. Parte de este trabajo ha consistido en descubrir estos elementos innecesarios y abordar su simplificación, migración e integración dentro de los componentes nativos de Unity3D. Finalmente, cabe destacar que uAdventure mantendrá la compatibilidad con juegos creados con eAdventure y también permitirá guardar los juegos en el mismo formato (con las evoluciones necesarias para dar cabida a las nuevas características de uAdventure) facilitando la migración de los juegos a la nueva plataforma.

Por otro lado, con el salto a Unity3D se da la posibilidad de que accedan a uAdventure nuevos usuarios con perfiles más avanzados de programación y conocimiento en videojuegos. Por ello, para que uAdventure pueda tener éxito y, a su vez, ser mejorable por la comunidad, su código debe ser fácilmente extensible mediante métodos que sigan la filosofía de atributos de Unity [29]. A raíz de ello, *plugins* para uAdventure podrían ser desarrollados para cada una de las capas que conforman uAdventure, es decir: editor, núcleo e intérprete. Por ello, en este proyecto, ha sido necesario reformar uAdventure en este sentido y, ya contando con dichas funcionalidades de extensibilidad, se han probado y se han usado para implementar como casos de prueba los nuevos *plugins* de geo-posicionamiento e identificación de códigos QR.

Si bien uAdventure deberá seguir evolucionando para mejorar su integración Unity y aprovechar las nuevas funcionalidades que pueda ir ofreciendo, es importante que la idea original de eAdventure como herramienta de autoría para no-programadores y como solución de bajo coste siga manteniéndose como principio fundamental de todo el proceso de transición.

Capítulo 4. Estado del Arte

Como se ha descrito en la introducción el principal objetivo de este trabajo es incluir en uAdventure aspectos de geoposicionamiento para que se puedan explotar en la creación de nuevos tipos de juegos. Esto requiere integrar nuevas mecánicas de juego que se adecuen a este tipo de información y a las nuevas interacciones que permite [30]–[32]. Para ello, se analizan en este estado del arte diferentes productos de mercado, así como las ventajas e inconvenientes de sus distintos enfoques. El desarrollo realizado en uAdventure toma como base este análisis buscando identificar buenas prácticas, mejorar la experiencia de usuario y evitar en la medida de lo posible problemas ya identificados en otros sistemas.

El ámbito que se va a explorar abarca principalmente las herramientas de autoría de juegos serios geoposicionados y algunos juegos serios basados en geoposicionamiento.

4.1. MITAR

El proyecto MITAR [10] es una herramienta para la creación de juegos serios geoposicionados que se basa fundamentalmente en la idea de utilizar ubicaciones reales que son enriquecidas con el juego serio. Para ello, el diseñador puede añadir a los lugares reales personas u objetos que permitan mezclar la realidad con datos ficticios para así incorporar la experiencia de lo real en ellos.

Este aspecto es muy positivo ya que el aprendizaje se mejora por la cercanía, dando lugar a una mejor comprensión y síntesis. Además, la cercanía genera un aumento de la implicación ya que se utilizan entornos que son familiares para los jugadores.

Para jugar, se utilizaban PDAs con GPS aunque posteriormente se intentó lanzar una versión para Android. La interacción con los personajes y objetos da lugar a conversaciones que el jugador utilizará para recabar información. Asimismo, el jugador tiene la capacidad de realizar mediciones sobre el entorno utilizando una herramienta en el interior del juego. El juego, además, se puede desarrollar en diversos momentos del tiempo, ya que el jugador puede utilizar portales temporales para explorar el lugar en el que se encuentra en momentos del pasado o futuro. Finalmente, para progresar en el juego se lanzan disparadores al interactuar con dichos elementos.

MITAR además, permite crear juegos en los que las personas se dividen en roles y cada rol experimenta diferentes partes del juego, fomentando la cooperación y el juego en equipo.

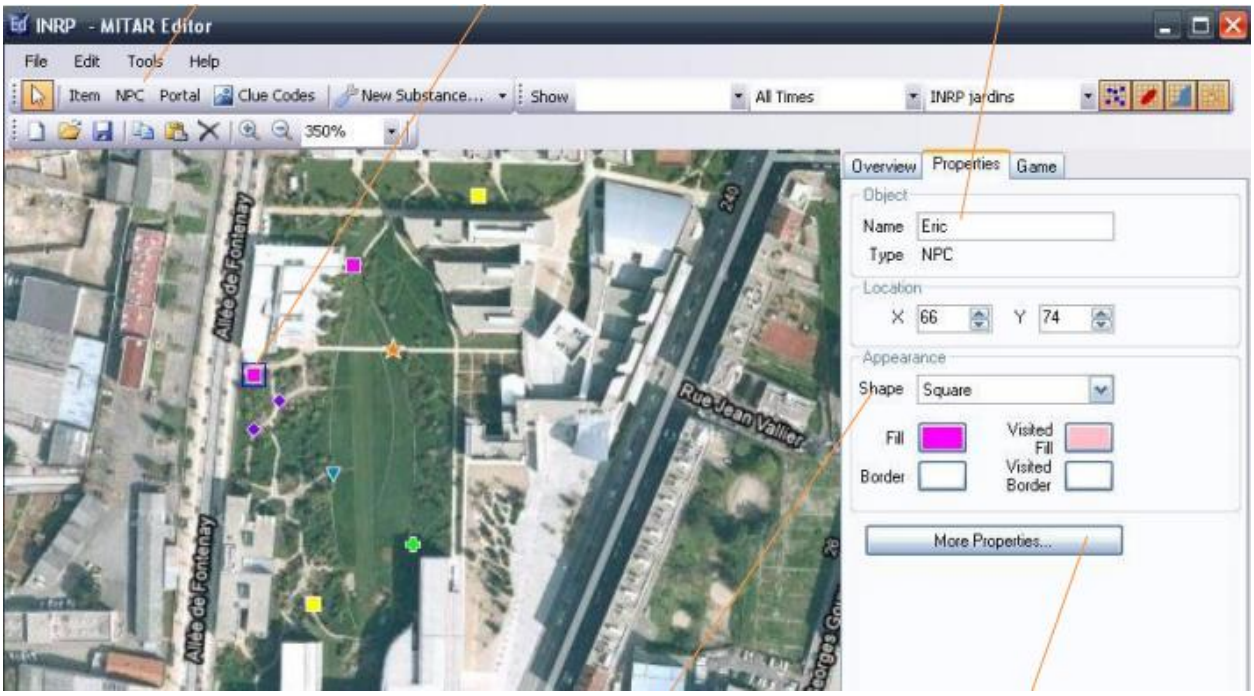

Fig. 4.1 MITAR Editor. Vista de NPC.

Si bien las funcionalidades que ofrecía la herramienta son interesantes, en su primera versión, el editor de MITAR, llamado MITAR Editor (Fig. 4.1), era extremadamente complejo por lo que se simplificó creando MITAR GameEditor [33], un editor de tipo arrastrar y soltar que puede ser usado incluso con bajos conocimientos informáticos. Tras esta simplificación, se llevó a cabo un experimento en el que estudiantes diseñaban y desarrollaban un juego serio que el resto de la clase debería jugar y, gracias a ello, se logró un entendimiento mucho más profundo de la materia debido a que los estudiantes debían conocer previamente lo que iban a integrar en sus juegos.

4.2. DiscoveryAgents

La herramienta de DiscoveryAgents[16] permite la creación de juegos geoposicionados con RA. Los juegos que se diseñen sobre esta plataforma pasan a formar parte del conjunto de juegos ya creados en DiscoveryAgents con el nombre de misiones.

Cada misión consiste en una sucesión de retos a elegir entre los diez tipos diferentes, que requieren interacción por parte del usuario para ser resueltos. Para iniciar los retos, el usuario

[16] https://discoveryagents.net

podrá utilizar tanto GPS como códigos QR. Antes de cada reto, al usuario se le proporcionará una información y una imagen que introducirán al reto. Una vez resuelto, el usuario recibirá puntos que afectarán a su puntuación final en el juego, así como una nueva información para continuar con la misión.

Para el jugador, acceder a los juegos es tan sencillo como descargarse una aplicación para su *smartphone* y acceder a la misión. Para el editor, la labor de diseño también es muy sencilla ya que la web incorpora ayuda guiada para los primeros pasos como se puede observar en la Fig. 4.2. Además, para el control de las ubicaciones, cuenta con una aplicación que permite registrar posiciones para ser utilizadas a posteriori durante la edición.

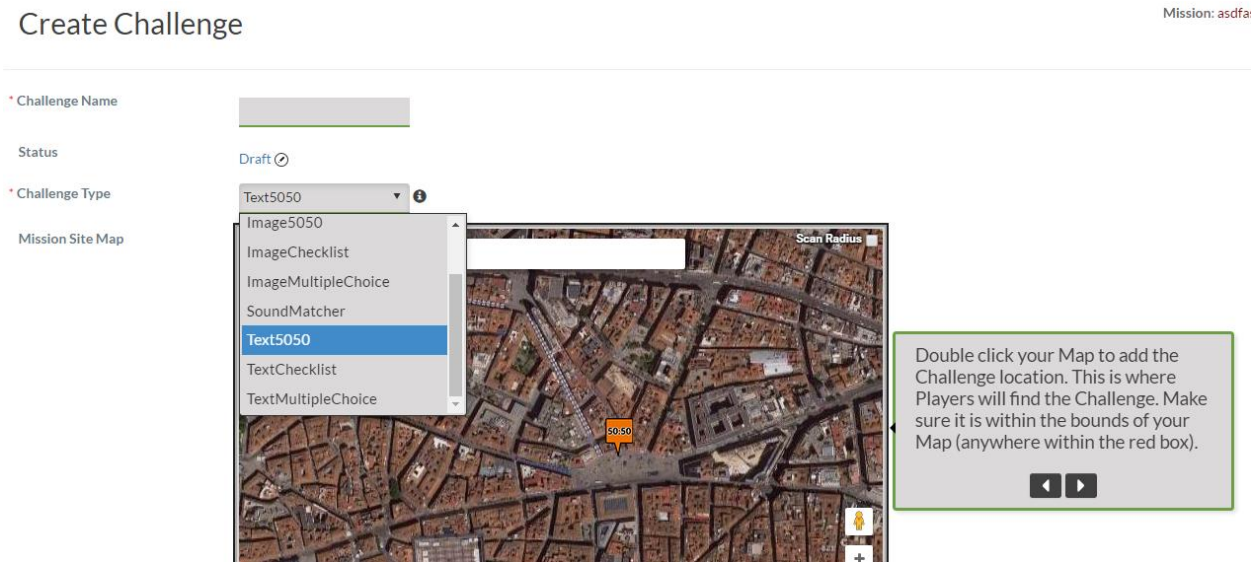

Fig. 4.2 DiscoveryAgents Mission Builder. Creando un reto de 50:50.

Como puntos negativos cabe destacar la imposibilidad de encadenar múltiples retos uno tras otro de manera que el conjunto forme un único reto, así como de elaborar una trama que requiera que el jugador haga uso de sus conclusiones para progresar en el juego más allá de la simple obtención de puntos. Además, no existen maneras de poder revisar la actividad de los jugadores ni sus respuestas por lo que el único método de evaluación de cara a la revisión por parte del profesor es la puntuación que reporte haber conseguido el jugador.

DiscoveryAgents además ha realizado una investigación [34] de su efectividad con Calgary Parks[17] con el objetivo de estudiar las diferentes experiencias de los usuarios que visitan el parque utilizando un juego diseñado para la ocasión. Para comparar dividieron el conjunto de jugadores en tres grupos de los cuales sólo uno de ellos utilizaría la app durante la visita. Entre los resultados se destacó un incremento del 10% en la felicidad y la diversión, y un 5% en la emoción. Además, se demostró que tanto los grupos que utilizaron la app como los que siguieron un mentor obtuvieron los mismos resultados de calificación, en contra a los grupos que visitaron el parque por libre.

4.3. ActionBound

El servicio que proporciona ActionBound[18] es muy similar al descrito anteriormente. La funcionalidad principal es la creación de gimkanas compuestas por diferentes pruebas. La metodología que siguen los usuarios es similar al caso anterior, los jugadores acceden a través de una aplicación que da acceso a todos los juegos y los editores utilizan la página web para editar las pruebas y visualizar los resultados.

En ActionBound, de cara al jugador, el juego se estructura en niveles que engloban pruebas que el jugador deberá ir completando para conseguir puntos y progresar. Sin embargo, el posicionamiento es un elemento más que se añade como método de progreso en el juego, no como requisito iniciador de pruebas. De esta forma, los niveles están pensados para acercar al jugador a la zona en la que se producen las pruebas que ocurrirán a continuación. Además, los niveles pueden seguir un orden fijo o flexible, siendo este último el que permite al jugador seleccionar el nivel que desea realizar a continuación. El editor por su parte es muy sencillo, pues muestra todas las pruebas en una vista principal, como se puede observar en la Fig. 4.3, que permite revisar en resumen la gimkana completa y permite añadir nuevas pruebas siguiendo formularios paso a paso.

[17] http://www.calgary.ca/CSPS/PARKS/Pages/home.aspx

[18] https://en.actionbound.com/

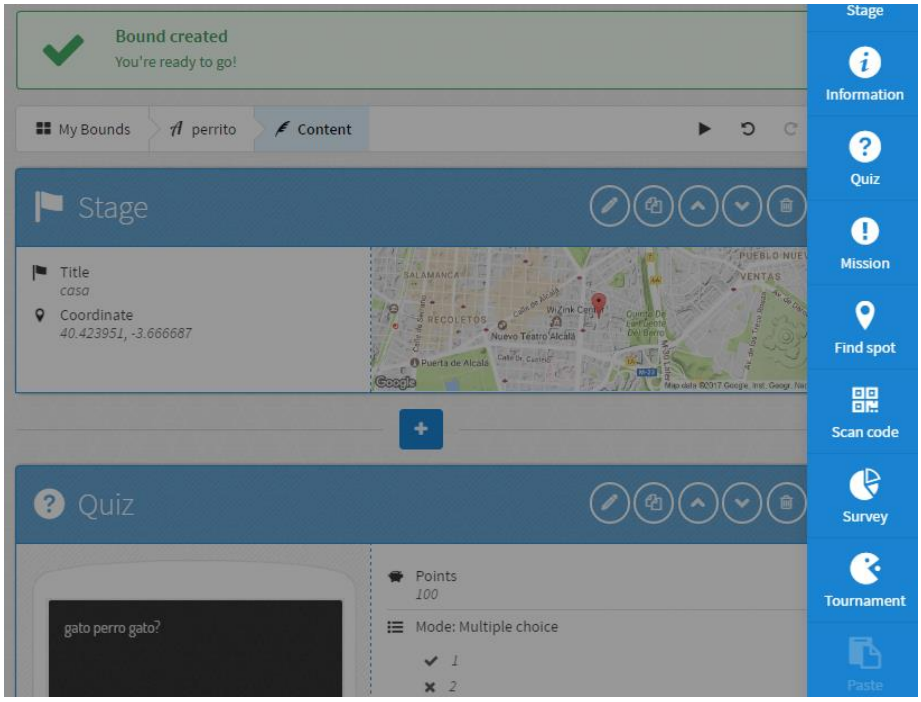

Fig. 4.3 ActionBound. Vista de la misión y panel de retos.

Al igual que en el caso anterior algunas de las pruebas son puntuadas. Dichas pruebas podrán bloquear al jugador de tal forma que no se le permita continuar hasta superarlas. Existen pruebas de tipo búsqueda, en la que el jugador deberá localizar una ubicación o un código QR, de tipo creativo en las que el jugador podrá añadir una respuesta que no tendrá una solución fijada o de tipo test en las que el jugador recibirá una puntuación en base a la exactitud de su respuesta. Estas últimas pueden ser temporizadas.

Cabe destacar la capacidad de ActionBound para la realización de actividades grupales, en las que los jugadores pueden comparar su puntuación final permitiendo aprovechar la competitividad para la obtención de mejores resultados. Además, para el creador de la gimkana se proporciona un panel en el que pueden visualizarse todos los resultados con información interesante como los puntos conseguidos, el tiempo tardado y los resultados de cada prueba como puede observarse en la Fig. 4.4.

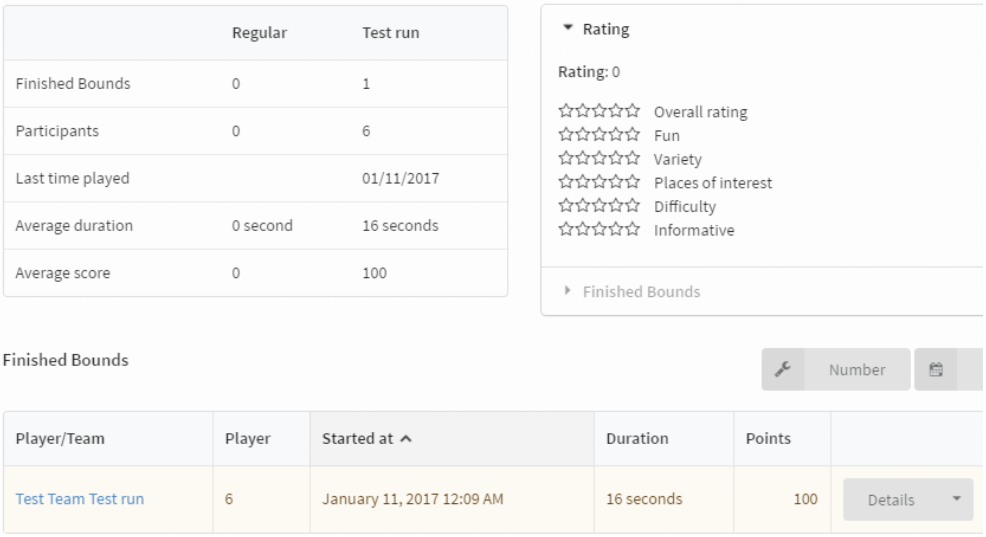

Fig. 4.4 ActionBound. Panel de revisión de resultados.

En cuanto a los puntos negativos cabe destacar la poca personalización y caracterización temática de las pruebas, salvo la inclusión de imágenes en las propias pruebas, empeorando así la inmersión en el juego. Además, los mapas están poco integrados con el juego por lo que la experiencia de realidad aumentada es baja. Las pruebas, por su parte, no incluyen información previa salvo la pregunta, aunque existe la posibilidad de incluir información por libre entre las pruebas que cumpla esta función o incluso haga las veces de diálogos.

En general, ActionBound provee de un servicio sencillo, claro y relativamente flexible para la jugabilidad que se ofrece, logrando una gran popularidad en Alemania (y Europa) donde ha llegado a realizar una publicación científica de un experimento sobre el sistema solar [35].

4.4. GooseChase

En GooseChase[19] se ofrece un servicio similar al anterior pero más simplificado en cuanto a pruebas y estilo de juego. En GooseChase, el jugador deberá completar misiones para obtener puntos. Estas misiones, pueden ser realizadas en el orden que se desee, aunque pueden ser presentadas a gusto del diseñador del juego y pueden consistir en tres únicos tipos: pruebas en las que hay que enviar una foto o un video, pruebas de enviar una respuesta textual o pruebas

[19] https://www.goosechase.com/

en las que hay que alcanzar una ubicación. De los tres tipos, sólo las dos últimas pueden ser evaluadas.

En el caso de las pruebas geoposicionadas, cabe destacar la personalización del radio de aceptación para alcanzar la posición, aunque no se permite la definición de zonas. Además, en contra a lo general, no existe la posibilidad de la utilización de códigos QR lo cual limita el uso en interiores.

Al contrario que en los casos anteriores, dado que se permite el envío tanto de fotos como de vídeos como respuestas, se promueve la creatividad e integración con el entorno de los jugadores en la resolución de las pruebas, que deberán ser valoradas posteriormente.

A destacar en este servicio, lo más valioso es el sistema de control de los jugadores ya que el organizador puede definir participantes y grupos a su gusto. Además, el organizador puede seguir en tiempo real el progreso y actividad en las pruebas según son completadas, así como visualizar el ranking y los archivos multimedia enviados por los jugadores. Por contra, los jugadores no tienen acceso al ranking, por lo que obtener beneficios de la competitividad es más limitado.

En el lado negativo, aparte de la ausencia de QRs, los jugadores no progresan durante el juego, sino que se limitan a realizar las pruebas en el orden que desean. Además, reciben poco conocimiento ya que la única información que recibe el jugador es un pequeño párrafo previo a cada misión, sin existir la opción de añadir textos al completar misiones o de forma libre entre cada misión. Estos hechos combinados, limitan la experiencia de juego y el desarrollo de tramas que envuelvan al jugador en un aprendizaje más profundo.

4.5. Eventzee

En el caso de Evenzee[20] nos encontramos en un punto intermedio a las herramientas anteriormente descritas, en el que destaca la simplicidad, utilizando el mismo formato de aplicación para todos los juegos existentes y web para diseñar y editar juegos. Sin embargo, incluye algunas funcionalidades que no se encuentran en las anteriores.

El formato de las misiones es la evaluación en base a puntos. Sin embargo, no existen pruebas de texto por lo que todas las pruebas son de fotografiar o encontrar códigos QR o

[20] https://www.eventzeeapp.com/

ubicaciones. Para el segundo caso, cabe destacar el hecho de que se previene el uso malintencionado de los QR por parte de los jugadores ya que pueden limitarse estos al rango de un GPS para ser válidos.

Pese a sus ventajas, también cabe destacar la poca variedad de retos, así como la poca información que el jugador obtiene de los mismos. Por contra, la filosofía de Eventzee es guiar al jugador para que obtenga pistas o información a través de realizar pruebas reales que den acceso a los QR. Por ello, el modelo de Eventzee se acerca más al de una aplicación de apoyo.

4.6. Scavify

El servicio que ofrece Scavify[21] es el más simplificado de todos. Ofrece una app para los jugadores y un editor web que provee de la creación de juegos con un conjunto de pruebas sin orden. Completar pruebas entregará puntos al jugador que afectarán a la puntuación final.

Entre sus pruebas cabe destacar la simplicidad, más que una pregunta y una forma de resolverla. Las formas de resolver son cuatro: entrada de texto, escanear un QR, adjuntar una foto o alcanzar una ubicación. El jugador deberá completar tantas tareas como sean posibles.

Destaca de cara al organizador el rico panel de analíticas que entrega, basadas en *machine learning* y gráficos.

En el lado negativo, destaca la pobre información que el usuario obtiene, siendo una simple frase por cada prueba. Por ello, en su filosofía se vende la idea de enriquecer y hacer más interesante el acercamiento a la información en el mundo real ya disponible, la resolución en equipo y realizar un aprendizaje interactivo.

4.7. GPSHunts

El servicio propuesto por HuntFun.co.uk llamado GPSHunts[22] ofrece un sistema de pruebas en un mapa geo-posicionado. El jugador deberá completar retos iniciados a través de GPS en el que podrá enviar fotos, videos o responder preguntas.

Cabe destacar como punto positivo, el hecho de permitir participar por equipos a través de un único móvil, así como el uso de temporizadores para la realización de las pruebas.

---

[21] https://www.scavify.com/

[22] http://www.gpshunts.co.uk/

Por contra, es un servicio poco intuitivo y poco cómodo. Las interacciones se realizan a través de la web, sin contar con aplicaciones nativas. Además, las analíticas muestran muy pocos detalles. En general, el servicio ofrecido es orientado a ser individualizado ya que las pruebas deben ser empaquetadas en categorías cerradas.

4.8. MooveTeam

En el caso de MooveTeam[23], el servicio ofrecido es un acercamiento más individual a la producción de actividades que impulsen la movilidad con el uso de GPS y códigos QR. Esta empresa española ofrece la creación individualizada de dichos juegos entre los que destacan cazas de tesoro, juegos de misterio, tours o hasta habitaciones de escape y ha sido participante del Mobile World Congress de Shangai [36].

En el lado positivo, se puede observar la riqueza de las soluciones, en las que los juegos promueven una trama interesante de cara al jugador. Sin embargo, el lado negativo es que no se ofrece una herramienta de autoría ya sea como software para sobremesa o vía web, lo que lo encuadra en el tipo de juegos específicos, alejándose de las herramientas de creación de juegos.

4.9. CtOS Enabler

Desarrollado bajo el programa Emprende UCM, el proyecto CtOS Enabler se destina a la activación de *smart cities*, tomando como fuente de información principal la ubicación de la persona [37]. CtOS Enabler permite que cada consumidor pueda definir sus zonas de influencias para que usuarios conectados a través de CtOS Enabler reciban la interacción con dicha zona, pudiendo recibir cualquier tipo de información relevante, tales como ofertas, información histórica, turística o eventos cercanos entre otras.

Para su funcionamiento, CtOS Enabler se provee como un servicio web configurable a través de un sitio web que es capaz de administrar las zonas de influencia (Fig. 4.5). Si bien CtOS Enabler no se encarga más que de almacenar una información representativa para la zona, es el cliente que se conecta con CtOS Enabler el encargado de sacar provecho de la interacción. Para ello, CtOS Enabler provee de una librería para Java, aunque es potencialmente utilizable en cualquier plataforma a través de su API.

[23] http://mooveteam.com/

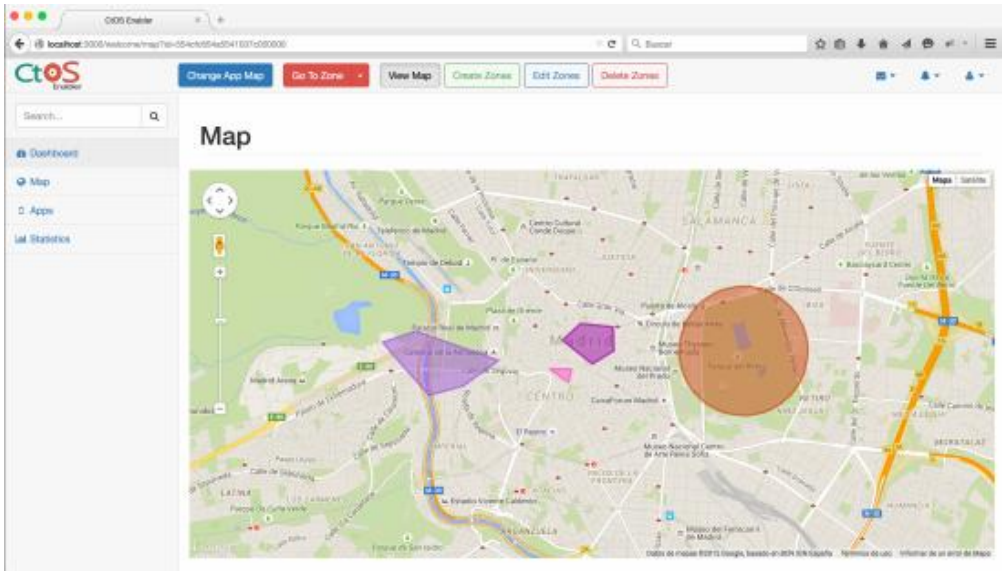

Fig. 4.5 Mapa de ejemplo construido con CtOS Enabler.

El proyecto tuvo una fuerte repercusión mediática y destacó por la realización de un tour cultural en el escorial llamado Post Fata Resurgo [38]. El tour, llamado para la ocasión gymkana histórica, fue un evento en relación con la saga de videojuegos Assasins Creed[24] que movió a los jugadores a contrarreloj por la ciudad para la recopilación de pistas que se resolverán en una ubicación final.

[24] https://es.wikipedia.org/wiki/Assassin's_Creed

Capítulo 5. Metodología

En este capítulo, en base a lo expuesto en los capítulos anteriores, se exponen los objetivos que abarcará este proyecto y las aportaciones que se llevarán a cabo para materializarlos. En esta línea, los objetivos planean no sólo el desarrollo de funcionalidades sino la experimentación y prueba de las mismas.

5.1. Funcionalidades sobre uAdventure sobre geo-posicionamiento

El **primer objetivo** consiste en el desarrollo de funcionalidades de geo-posicionamiento sobre el uAdventure. Como base para llevar a cabo el desarrollo el proyecto, uAdventure supone un punto de partida fundamental debido a su código abierto y a su orientación de juegos serios. Por su diseño, uAdventure da soporte a funciones para el desarrollo de aventuras gráficas. No obstante, aunque las escenas sean diferentes, es posible reutilizar gran parte de contenido de uAdventure sobre las nuevas mecánicas de geo-posicionamiento. Para su integración con mecánicas de juego geo-posicionadas se enumeran los siguientes tres sub-objetivos.

5.1.1. Nueva escena con forma de mapa

El **primer sub-objetivo** del desarrollo es la creación de un nuevo tipo de escena basada en la integración de elementos de juego sobre un mapa aumentado en el que el jugador sea representado. Uno de los motivos del éxito de Ingress y Pokémon GO es la capacidad de inmersión del jugador en el juego mediante el mapa de juego. Este elemento puede encontrarse también en varios de los ejemplos expuestos en el estado del arte como MITAR y DiscoveryAgents. La visión del mapa permite al jugador situarse dentro del juego, a la vez que recibe información básica y acceso a configuración. Esta escena con forma de mapa era una de las mecánicas inexistentes en la versión de eAdventure para Android.

La escena de mapa sirve de contenedor para los elementos de juego, así como para asistir al jugador en el proceso de juego con navegación y ayudas visuales. Dado que las nuevas mecánicas de geo-posicionamiento sobre uAdventure no pretenden cambiar drásticamente la forma de juego, el modelo de diseño de los elementos que conforman el mapa es híbrido entre las nuevas funcionalidades y lo que ya existe en las escenas de uAdventure. De esta forma, el mapa puede almacenar elementos diseñados específicamente para el mapa (que se especifican más adelante), como áreas, rutas o puntos de interés, así como los propios elementos de uAdventure,

es decir, personajes, objetos y atrezos. Todos ellos se almacenan como referencias que contendrán datos sobre posición y/o representación.

El mapa en sí mismo, en la versión inicial, contará con una forma de visualización básica aérea bidimensional, similar a lo que proporciona uAdventure. Sin embargo, al ser una nueva escena, esta deberá diseñarse para poder acercarse a entornos más familiares al jugador de juegos exitosos, por lo que deberá diseñarse para tener soporte de vista tridimensional.

5.1.2. Elementos geoposicionados

El **segundo sub-objetivo** es incorporar en el mapa elementos geoposicionados además de contener los elementos de uAdventure. Estos elementos se caracterizan por representar zonas en el mapa y tener formas características de interacción. Estos elementos se dividen en tres tipos: puntos, rutas y áreas. Además, todos ellos cuentan con un área de influencia, que les permite definir el rango sobre el cual se considera que el jugador está en contacto con ellos. Todas las acciones sobre elementos geoposicionados, al igual que las acciones ya existentes en uAdventure ejecutan un efecto siempre y cuando se cumplan las condiciones. Sin embargo, el usuario no requiere de un menú para ejecutarlas, sino que utiliza su contextualización como forma de interacción.

Las acciones que se podrán realizar son cuatro:

- Entrar en la zona: Esta acción se ejecuta al entrar en el rango de influencia. Opcionalmente, podrá requerirse que el jugador esté fuera del área en el momento de comenzar la escena para evitar bucles.
- Salir de la zona: De forma similar al comportamiento de entrar, al salir también habrá acciones, y estas, a su vez podrán requerir que el jugador se encuentre dentro a priori.
- Mirar: Esta acción implica que el jugador observe físicamente la dirección requerida para ejecutarse. Existirán dos tipos de direcciones siendo la primera, una dirección prefijada y la segunda, la dirección hacia el centro del elemento. Para ello, se utilizará la brújula del dispositivo. Además, el jugador deberá estar dentro del área de influencia.
- Inspeccionar: Esta acción, a diferencia de las anteriores, se ejecutará cuando el jugador interactúe con el mapa a través de los métodos clásicos, permitiendo mostrar información del área.

Por su parte, las referencias a elementos de uAdventure también juegan un papel con el mapa teniendo su área de influencia para poder ejecutar acciones y pudiendo permanecer ocultas hasta que el jugador se encuentre en el radio de interacción, simulando de esta forma la mecánica de juego de Pokémon GO.

5.1.3. Navegación en el mapa

El **tercer sub-objetivo** que cierra el geo-posicionamiento basado en mapas permitirá al jugador navegar a través de los mapas. Para evitar que el jugador pueda perder el objetivo de juego se proporcionará control de la navegación en base a direcciones que podrá ser ampliado a rutas en futuras versiones. La navegación debe permitir configurarse para poder secuenciar una serie de pasos que podrán ser procesados o bien por orden o por cercanía del jugador al elemento. Además, algunos de los elementos deberán poder bloquear la navegación hasta que sean completados al interactuar con los elementos del juego. Para ello, el navegador podrá ser controlado sin necesidad de proporcionar una dirección, permitiendo seleccionar entre los elementos actuales si se desea avanzar, atrasar o saltar directamente a algun elemento especifico.

5.2. Códigos QR

El **segundo objetivo** del proyecto permite utilizar códigos QR para abarcar el geoposicionamiento desde otro acercamiento orientado a interiores. Los códigos QR jugarán un papel fundamental para el posicionamiento en interiores dado que el GPS no es una herramienta fiable en estos casos y por tanto podría ocasionar bloqueos en el juego. Si bien esta funcionalidad ya se implementó en eAdventure para Android, no se completó su integración con la plataforma. Por ello, se añadirán editores que permitan controlar los efectos de la detección de un código QR y las condiciones bajo las cuales el mismo podría ser escaneado. Además, el editor permitirá guardar e imprimir las imágenes de los QR directamente.

El escaneo, por su parte, será utilizado mediante un escáner integrado, dado que evita romper la experiencia de juego, así como permite su funcionalidad en todas las plataformas posibles sin tener que depender de programas de detección de terceros.

5.3. Extensibilidad de uAdventure

Dentro de este trabajo será necesario extender uAdventure para añadir nuevas funcionalidades. Por ello, el **tercero objetivo** de este proyecto es plantear una reingeniería del código para permitir que la plataforma sea extensible. Esta reingeniería no sólo pretende mejorar la curva de aprendizaje en futuros desarrolladores, sino que es un paso fundamental para el ciclo de vida de uAdventure al hacer mucho más mantenible el código. En su estado actual, uAdventure tiene un código orientado a implementar las funcionalidades de eAdventure, pero no se orienta a ser una plataforma sobre la que desarrollar o extender.

Por ello, será necesario que sea posible extender el proyecto en diversas partes fundamentales, tanto en la capa del núcleo, en la capa de edición y en la capa de ejecución. Los paquetes que se implementen en este proyecto y, en general, todos aquellos que se enganchen a los puntos de extensibilidad que se proponen (de ahora en adelante llamados *plugins*) serán capaces de integrarse con uAdventure y desacoplarse con ninguna o mínimas modificaciones del proyecto original fomentando el autodescubrimiento.

La capa de núcleo se compone de tres partes: modelo, persistencia y carga. La parte de modelo deberá poder contener los nuevos elementos que requieran los *plugins*. Junto a ella, la parte de persistencia y carga, que también deberá poder ser extendida para permitir que los nuevos elementos se persistan junto al modelo XML. Por último, dado que en este caso se crea un nuevo elemento principal, la vista de mapa, se deberá permitir que el juego de paso a elementos que no sean las escenas clásicas.

En la capa de ejecución, deberán reflejarse la extensión realizada en el modelo permitiendo ejecutar las nuevas escenas o efectos que se hayan añadido. De esta forma, tanto el elemento principal como los efectos añadidos deberán acoplarse en el momento adecuado con el ciclo de vida natural de elementos principales y efectos.

Los puntos fundamentales en el editor son la vista principal, que deberá poder incluir nuevos apartados en base a contenidos nuevos que añadan los *plugins*, y las vistas de efectos, que también deberán poder extenderse en base a los nuevos efectos que incluyan los *plugins*.

Cabe destacar en general que, en la parte de edición y, en medida que se permita, en la parte de ejecución, se seguirá la filosofía de Unity, en la cual se prima el autodescubrimiento y el automatismo a través de herramientas propias del lenguaje como las etiquetas o atributos, o incluso, la identificación física de archivos en lugares determinados.

5.4. Analíticas del aprendizaje sobre las nuevas mecánicas

El **cuarto objetivo** de este proyecto es integrar analíticas del aprendizaje con las nuevas mecánicas de geoposicionamiento. Una de las novedades de uAdventure fue la integración de analíticas del aprendizaje a través del proyecto europeo del H2020 RAGE en el que participa e-UCM. En esta línea, dado que las mecánicas de geoposicionamiento son diferentes de la casuística general trazada será necesario incorporar tres nuevas trazas.

La **primera traza** es el seguimiento del jugador por el mapa, realizándose periódicamente cada cinco segundos. La **segunda traza** es la interacción con los diferentes elementos geoposicionados a través de sus cuatro nuevas acciones (entrar, salir, mirar e interactuar). Por último, la **tercera traza** cubrirá el descubrimiento de elementos nativos en el mapa, siempre que estos estén ocultos, así como la interacción a través de mapa con ellos.

5.5. Prueba de concepto

Para probar la extensión de uAdventure con funcionalidades de geo-posicionamiento, el **quinto objetivo** de este proyecto plantea el desarrollo de un experimento o caso de estudio, en el que se contraste la validez de las mejoras propuestas y permita vislumbrar como se podrían utilizar para la creación de nuevos experimentos que aprovechen dichas características de geo-posicionamiento. Por tanto, se pretende crear un juego simple pero que permita poner en práctica las mecánicas de juego de geo-posicionamiento y contrastar la eficacia en base a la experiencia de usuario y los datos obtenidos a través de las analíticas de aprendizaje.

El experimento a desarrollar con uAdventure mediante las nuevas capacidades del editor consistirá en una la realización de una guía gamificada que permita al usuario conocer las instalaciones deportivas del campus de Moncloa de la UCM. No es sólo un paseo guiado ya que se incorporará además la utilización de organismos necesarios para algunos de los procesos universitarios como los registros. A través de esta guía gamificada, se pretende que el usuario conozca la oferta deportiva, la situación de las diferentes instalaciones y los lugares donde deberá entregar documentación en el caso de que se requiera para un procedimiento oficial.

El juego puede empezar en cualquier punto del campus de Moncloa, pero preferiblemente desde el metro de ciudad universitaria ya que es el epicentro de la actividad y el principal nexo de comunicación. El juego comienza mostrando una breve introducción a la actividad, en la que, en forma de diálogo con un personaje que hará de asistente, se proporciona el objetivo principal

y se introducen los controles y la interacción con el juego. **En este apartado se prueba la integración de escenas de mapas y personajes nativos de uAdventure, así como conversaciones**.

Como ejemplo de toma de contacto se le pedirá al jugador que mire hacia el edificio de estudiantes. Una vez se encuentre mirando en dicha dirección, se informará de sus principales funciones. Si, por el contrario, el usuario mirara en dirección contraria hacia medicina, se mostrará otro mensaje que bromeará acerca de "no irse de cañas a la cafetería de medicina". **En este apartado se prueba la acción de mirar de los elementos geoposicionados**.

A continuación, el jugador deberá caminar hasta cualquiera de los dos principales centros deportivos ya sea el del paraninfo o la zona sur de deportes. Durante el trayecto se hará alusión al carril bici si el usuario está *suficientemente* cerca del mismo. Para ello, el mapa mostrará el carril bici junto con un icono que lo señalará. Una flecha indicará al usuario hacia las instalaciones deportivas y una vez que el jugador las alcance (detectado en función del GPS del dispositivo móvil), se lanza automáticamente una nueva escena en la que aparecerá la información básica de las instalaciones. En ella aparecerá un entrenador que será el encargado de proporcionar la información y, a continuación, invitará al jugador a caminar por las instalaciones para que haga una toma de contacto y descubra algunos de los deportes que se practican allí a través de la recogida de las diferentes pelotas con las que se juega a dichos deportes. Según el centro deportivo al que se vaya, se mostrarán unas pelotas u otras, pero la mecánica del juego será la misma. Dichas pelotas aparecerán ocultas y el jugador sólo dispondrá de una información básica sobre cuáles pelotas le quedan por encontrar. Para encontrarlas, cuando el jugador se encuentre suficientemente cerca del campo apropiado, la pelota se revelará permitiendo ser recogida. **En este apartado se prueban la navegación por cercanía, la acción de entrar a un elemento geo-posicionado y la integración de objetos nativos, así como su descubrimiento en el mapa.**

Una vez se haya completado la recogida de pelotas, el jugador volverá a la oficina principal donde se le contarán las actividades disponibles en dicho centro y los precios más significativos por sesión, si se tiene el bono correspondiente y que es gratis si se dispone del carné anual.

A continuación, se guiará al jugador a la piscina más cercana utilizando una flecha en la vista principal de mapa que indicará la dirección hacia la que se encuentra la piscina. En el caso del paraninfo, se guiará hasta la piscina del complejo deportivo Nuestra Señora de la Almudena.

En ambas piscinas, un QR identificará el lugar donde comprar la entrada a la misma y se hará alusión a la otra piscina, difiriendo el caso de que sea invierno o verano (la piscina del sur es para verano, descubierta y la del norte es para invierno cubierta). Al detectar el QR, se mostrará una escena donde se mostrarán dichos diálogos. **En este apartado se probará la navegación paso a paso y el escáner de códigos QR.**

Al acabar la tarea en la piscina se pedirá al jugador que rellene una instancia de reserva de una pista que deberá entregar en cualquiera de los registros de la universidad. El juego hará algunas indicaciones en forma de diálogo y se dará por rellena (sin llegar a solicitar información real) y, a continuación, se podrá seleccionar el registro al que se quiere ir, pudiendo elegir simplemente, el más cercano de todos.

Finalmente, el usuario será guiado (nuevamente a través del mapa y una flecha, en base a su posición GPS) hasta la instalación, donde habrá un QR junto a la oficina del registro que deberá identificar y que dará por concluida la actividad.

En base al tiempo que se pruebe de juego, las tareas de las piscinas podrán ser opcionales, siendo meras menciones de su ubicación y funcionalidades.

Para medir la eficacia, se realizarán encuestas de conocimiento, así como valoraciones de la experiencia. Durante todo el proceso, además, se recolectarán analíticas del aprendizaje.

5.6. Estructuración del desarrollo

El desarrollo de este proyecto constará de cuatro iteraciones con temática bien diferenciada en la cual se llevarán a cabo los objetivos de este proyecto. Estas cuatro iteraciones consisten en: i) la creación sobre Unity de elementos de temática geo-posicionada y QR, ii) la rearquitectura de uAdventure para la mejora de la extensibilidad, iii) la integración de los elementos de i en uAdventure, iv) el desarrollo del caso de prueba.

En la primera parte, se crearán o adaptarán sobre Unity elementos que darán soporte a las herramientas y componentes de juego que se conectarán con uAdventure. Estas componentes son fundamentalmente los mapas dinámicos y los elementos que estos contendrán. Ambos deberán poder representarse en las partes de edición y ejecución, aunque la parte de ejecución será mucho más rica a nivel visual, mientras que la parte de edición será más pobre ya que será más importante que sea sencilla de manipular y editar. Además, se harán prototipos de reconocimiento de códigos QR.

En la segunda parte, se revisará la estructura actual y se plantearán los pasos a seguir para poder integrar los nuevos elementos en forma de *plugins* autodescubiertos. Además, se revisarán y corregirán todos los errores posibles que se encuentren en uAdventure.

En la tercera parte, se procederá a integrar los elementos que existen en uAdventure en los mapas y, de manera opuesta, integrar estos en el ciclo de vida de uAdventure. Además, se realizarán los nuevos efectos y se construirá un modelo compatible con uAdventure y que sea capaz no sólo de persistirse y reconstruirse, sino también de ser ejecutado en el momento apropiado.

Finalmente se implementará y ejecutará el caso de prueba descrito con anterioridad, realizando las integraciones con los sistemas de analíticas de aprendizaje que se consideren necesarias, utilizando y extendiendo el tracker ya existente en uAdventure.

Capítulo 6. Desarrollo

En este capítulo se expone el desarrollo realizado siguiendo la metodología en base a prototipos, partiendo de un análisis rico y con una mejora iterativa de lo desarrollado en los apartados anteriores. En muchos de los casos, el desarrollo se ha realizado teniendo en cuenta aspectos de calidad software presentes en el estándar ISO/IEC 9126, centrándose especialmente en la mantenibilidad, desde el punto de vista de la extensibilidad, la calidad del uso y la fiabilidad [39], [40]. Para lograrlo, todo el desarrollo se ha realizado basándose en patrones de diseño [41] tales como factorías, observables, adaptadores, etc. Como describe el último apartado del capítulo 4, el desarrollo se estructura en cuatro partes que se expondrán en los próximos apartados de este capítulo.

6.1. Desarrollo de componentes para Unity

Dentro de las componentes de Unity se han desarrollado dos componentes que permiten la gestión de mapas, así como los elementos que los contienen. Además, se ha integrado un lector de QR. En los siguientes sub apartados se analizan los diferentes elementos.

6.1.1. El elemento mapa

En base a los objetivos del proyecto se deberá desarrollar una escena que permita interactuar con los elementos de juego. En esta escena, el contenedor principal es un mapa que mantiene el jugador y permite visualizar su posición real sobre el mismo. Asimismo, en el mapa deberán mostrarse los diferentes elementos que se hayan establecido durante el diseño del juego.

En cuanto a su complejidad gráfica, el requisito básico es que se debe permitir que se muestren los trozos de un mapa existente, como por ejemplo Google Maps[25], en un zoom confortable y que el jugador quede en el centro del mapa de manera perfectamente visible. Sin embargo, y para incrementar la cercanía con juegos como Ingress o Pokémon GO, es muy interesante que el mapa pueda contener estructuras reales, tales como edificios, parques, lagos o monumentos. En esta línea, el mapa deberá pertenecer al entorno 3D para que pueda ser extendido en el futuro para poder incorporar dichos elementos de representación tridimensional.

[25] https://www.google.es/maps

En una investigación inicial, se buscaron componentes ya implementadas que permitieran la construcción integrada en Unity de dicho mapa, liberando parte de las tareas de programación y reduciendo el esfuerzo hacia la adaptación y las próximas tareas. Debido precisamente a la tendencia de mercado causada por Pokémon GO, el desarrollador independiente Baran Kahyaoğlu[26] desarrolló a finales de 2016 un mapa sobre Unity con características similares a las que se requieren en este proyecto al que bautizó como Mapzen GO[27]. El nombre de Mapzen pertenece a una compañía de mismo nombre que mantiene un servicio web de información vectorial topológica[28], que incluye todo tipo de elementos como construcciones y elementos naturales. Debido a que Mapzen GO se basa principalmente para la construcción del mapa en dicho servicio, de ahí tomo prestado su nombre. Por sus características iniciales y por su licencia MIT, se tomará dicho proyecto como base para la elaboración del *plugin* de uAdventure.

La versión inicial de Mapzen GO había sido desarrollada en un corto período de tiempo y contaba con una arquitectura bastante potente, que, sin embargo, no se ajustaba perfectamente a las necesidades de este proyecto. La arquitectura original, aunque compleja, cuenta con una estructura basada en *plugins* que se ejecutan de manera jerárquica. Cada *plugin* se ejecuta una vez para cada trozo de mapa, también llamado *tile*, y añade una pequeña parte de funcionalidad a éste. De esa forma, los *plugins*, que actúan de decoradores (patrón *decorator*), van sumando tipos de elementos y construcciones que forman el *tile* final.

El elemento principal de la arquitectura es el TileManager. Esta clase se encarga de definir los metadatos que se transmiten a los *tiles* que se generarán, en base a algunos parámetros como la posición del jugador, el zoom, la latitud y longitud. Para ello Mapzen GO utiliza una transformación definida por osgeo[29], dentro de lo que se conoce como Tile Map Service (TMS) [42], para la transformación de coordenadas a metros o píxeles definida en EPSG:3857 [43]. Dicha definición es un tipo de marcador esférico (para la superficie terrestre) que utilizan mapas como Google Maps u OpenStreetMap[30] (OSM). Una vez se convierten la latitud y longitud a metros, existe una conversión directa a píxeles en base al zoom. A mayor es el zoom, se

[26] http://www.barankahyaoglu.com/

[27] https://github.com/brnkhy/MapzenGo

[28] https://mapzen.com/

[29] http://wiki.osgeo.org/wiki/Main_Page

[30] https://www.openstreetmap.org

requieren mayor cantidad de píxeles para cubrir la superficie completa de la tierra y por lo tanto mayor es la cantidad de detalle que contiene el mapa que utilice dicho zoom.

Una vez que TileManager calcula la posición en metros correspondiente a la posición actual del jugador utilizando TMS, debe localizar el *tile* que contiene dicha posición y, para ello, existen dos sistemas diferentes: el definido en TMS, que crece de abajo hacia arriba y de izquierda a derecha y el definido por Google y utilizado por OSM, que invierte la coordenada vertical. Dado que se utilizan los servicios de OSM, se utiliza el segundo método de división y a continuación se incluye esta información de meta dato en el *tile* que se procederá a crear. Además del *tile* central, el TileManager extenderá en un radio calculado por método manhattan[31], los *tiles* alrededor del *tile* central, en base a un parámetro rango.

Una vez se han generado los metadatos, utilizando UniRx[32] se realiza una carga de la primera capa de información vectorial procedente de Mapzen y, una vez se finaliza la carga, se llama a la capa de *plugins* que interpretan la información y generan las estructuras visuales correspondientes. Esta capa trata a todos los *plugins* por igual, pero se pueden identificar dos tipos claros: las factorías que generan elementos tridimensionales y los *plugins* que añaden información nueva procedente de otras fuentes al mismo *tile*.

Las factorías heredan de la clase *plugin* y definen una *query* única que permite, dentro de los elementos *json* recibidos desde Mapzen, encontrar los elementos que se quieren instanciar. De esta forma, existen diferentes factorías para "*buildings*", "*landrouses*", "*roads*" y otros elementos significativos definidos en el estándar geojson [44] para la identificación de elementos. Todos ellos, al definir su *query*, inmediatamente reciben uno a uno los datos que deben transformar en elementos tridimensionales y utilizan la librería de triangulación[33] de Christian Woltering distribuida con licencia MIT para la transformación de los puntos en triángulos y posteriormente en mallas tridimensionales.

Por otro lado, la otra cara de *plugins* que añade nuevas funcionalidades sólo expone dos ejemplos. El primero y más importante es el que, a partir de la información del *tile*, obtiene una imagen proveniente de un servicio de *tiling* de mapas basados en píxeles (en contra a lo que hace

[31] https://es.wikipedia.org/wiki/Geometr%C3%ADa_del_taxista

[32] https://github.com/neuecc/UniRx

[33] http://triangle.codeplex.com/

Mapzen, que hace *tiling* de información vectorial) y crea un elemento 3D que contenga dicha imagen en el lugar donde debe aparecer el *tile*. El otro ejemplo de funcionalidad de ejemplo incluida en los *plugins* adicionales se encarga de poner objetos personalizados en el mapa, siendo un mero ejemplo de colocación de esferas.

TileManager, además, cuenta con dos clases herederas que mejoran su funcionalidad. La primera, es DynamicTileManager. Este nuevo mánager es capaz de controlar la carga de tiles en base a los movimientos que haga el jugador y, de ser configurado para ello, reposiciona los *tiles* para mantener al jugador centrado. Por otro lado, heredando de ésta se encuentra CachedDynamicTileManager que además se encarga de almacenar la información obtenida de Mapzen en ficheros de caché que podrán ser cargados en lanzamientos posteriores de la aplicación.

Finalmente, Mapzen GO incluye un apartado más de control del jugador. Una vez cargado, el avatar del jugador puede moverse libremente utilizando las posiciones del personaje y, utilizando un *collider* que simboliza el *tile* central del mapa permite detectar cuando el jugador se ha desplazado del centro y se deben cargar nuevos *tiles* y/o reposicionar el centro del mapa. El control en la demostración que provee MapzenGO se produce mediante entrada de teclado. Finalmente, una vez el jugador se aleje demasiado de los *tiles*, para economizar memoria, se destruirán aquellos que no estén lo suficientemente cerca de este.

Tras el análisis inicial de la estructura y funcionamiento de Mapzen GO se identifican los siguientes puntos que deberán modificarse:

1. Pese a su estructura de *plugins*, TileManager tiene una vinculación muy fuerte sobre los datos vectoriales de Mapzen, que le impiden poder cargar directamente los *tiles* basados en píxeles hasta que finaliza la carga de los datos de Mapzen. Dado que en la versión inicial del mapa no se incluirán los mapas vectoriales debe ser posible desacoplar Mapzen.
2. Los elementos del mapa, pese a representar su posición en Unity en base a distancias reales no tienen soporte a escala ni a posicionamiento relativo, por lo que, dado que será necesario adaptar el mapa a la escena de uAdventure, será necesario transformar las coordenadas globales a coordenadas locales. Además, para poder implementar una futura carga dinámica de *tiles* de zoom superior que

permita cubrir temporalmente el área hasta que los *tiles* inferiores sean cargados, los *tiles* deben de cubrir áreas diferentes en base al zoom al que se destinen.

3. Por último, pese a que el jugador se puede controlar, no existe ninguna relación entre su posición en el mapa y sus coordenadas ni, de forma contraria, una forma de posicionar al jugador en las coordenadas que se desee.

A continuación, se expone el análisis, diseño e implementación de las soluciones de los puntos anteriores.

El **primer punto** requiere de la rearquitectura completa de los *plugins* de Mapzen GO de tal forma que se desacoplen los elementos, pero conserven las dependencias entre las factorías y los datos de Mapzen existentes. Por ello, será necesario que los *plugins* incorporen en su interfaz la lista de elementos de los que dependen para poder ejecutarse. Por otro lado, será necesario tener un nuevo tipo de *plugin* que recopile los comportamientos relacionados con Mapzen GO dentro de TileManager y los incorpore en su interior, ejecutándose sin ninguna dependencia.

En este sentido, la arquitectura pasa a tener niveles de *plugins*, un nivel de *plugin* en el que se cargan elementos no dependientes y, en medida que se cumplan las dependencias se cargan los *plugins* de niveles inferiores. En la capa superior se encuentran dos *plugins*, el *plugin* de Mapzen GO y el *plugin* de imágenes de *tiles* de OSM. En la siguiente capa se encuentran todos los *plugins*/factorías que dependen de los datos obtenidos por el *plugin* de Mapzen.

Además, es necesario que se modifique el elemento de base para la paralelización de los diferentes *plugins* a través de la librería UniRx, de tal forma que, una vez un *plugin* finalice se revisen y arranquen los *plugins* que dependen de él.

En base a esto, el diseño se reforma en capas definidas por las dependencias. La primera capa incluiría la clase TileManager. Bajo esta capa, la clase MapzenPlugin y la clase TileImagePlugin. Finalmente, bajo esta capa, todas las factorías que sean convenientes. Con vistas a futuro, uAdventure requerirá de la incorporación de un *plugin* propio para incluir elementos de juego. Este plugin, a su vez, estará dividido en dos capas, una capa de *plugin* de filtrado y control y una capa de *plugins* que toman los elementos filtrados y los crean. En las Fig. 6.1 y Fig. 6.2 se encuentran los diagramas de clases que comparan ambas arquitecturas.

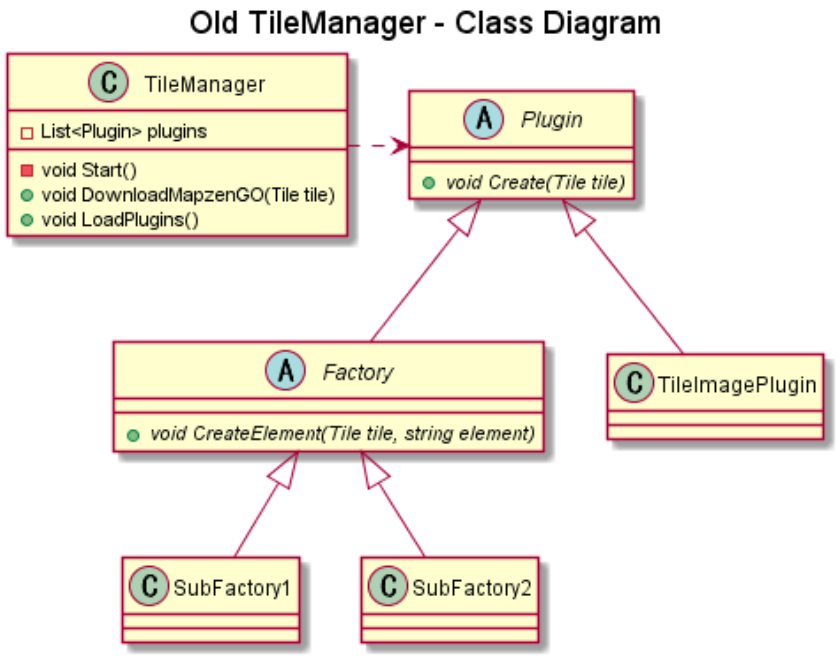

Fig. 6.1 Arquitectura antigua de plugins en Mapzen GO

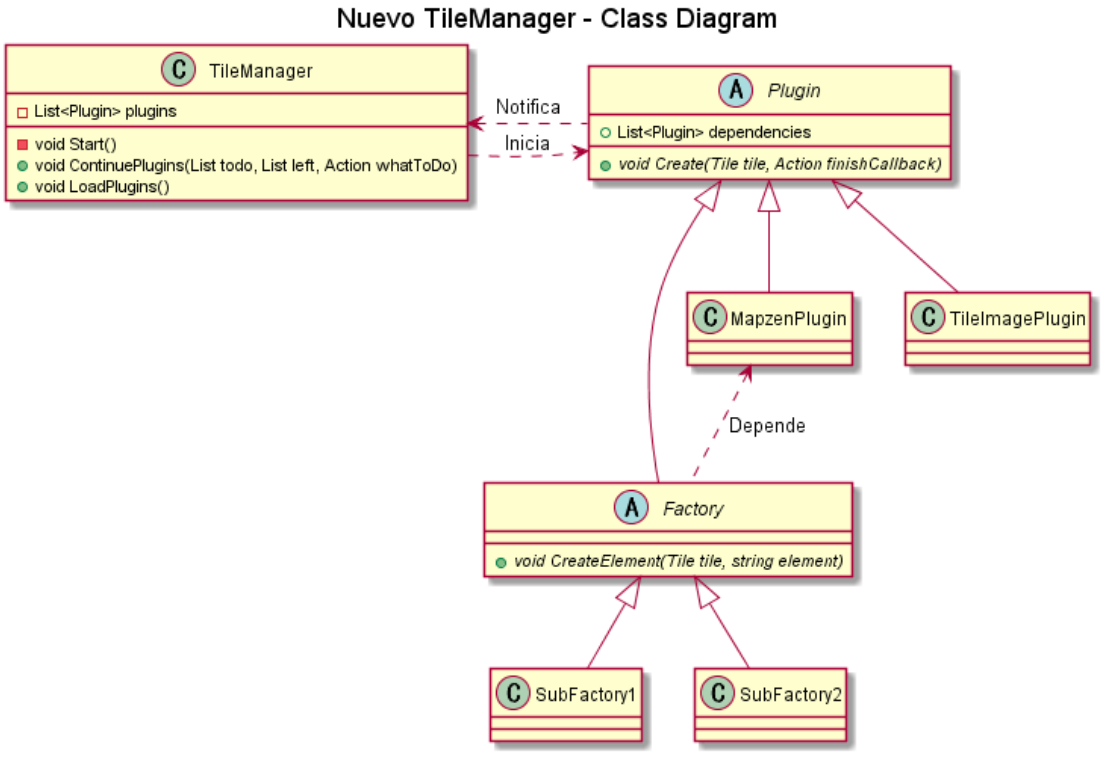

Fig. 6.2 Nueva arquitectura de plugins que incorpora dependencias y MapzenPlugin.

Los diagramas anteriores muestran cómo se ha mutado TileManager para eliminar la carga de MapzenGO y se ha incorporado un método ContinuePlugins que permite continuar con la carga de *plugins* cuando se notifique la finalización. Por su parte, la clase Plugin incluye ahora una lista de dependencias que TileManager utilizará para iniciar dicho *plugin* cuando todas se satisfagan. Además, puede observarse que las dependencias de Factory incluyen a MapzenPlugin, la nueva clase que incorpora toda la funcionalidad antes ubicada en TileManager.

Para resolver el **segundo punto**, a saber, que los elementos mantengan siempre la misma escala y éstas se correspondan con medidas reales de manera uniforme, se revisó que todo el código se rigiera bajo esta norma. Para ello, se analizaron todas las partes de código en las que se generan y posicionan elementos en el mapa, tales como *tiles* y construcciones y se revisó que todos los elementos se rigieran por posiciones locales en lugar de posiciones globales. Para ello, los elementos utilizarán la siguiente fórmula para calcular su posición en el ámbito de Unity.

Posición Local (Unity uds.) = Posición Global (m) – Posición del Centro del Mapa (m)

Utilizando la formula, los elementos se posicionan de forma independiente al zoom y su posición en Unity es una representación directa de los metros que abarca el elemento. Dado que el sistema de coordenadas globales requiere al menos un *double* para tener la precisión suficiente, es posible que los elementos muy lejanos al centro se vean afectados por la precisión *float* que proporciona Unity en los posicionamientos. No obstante, este caso no se dará, ya que el centro del mapa puede reubicarse afectando de forma directa a la posición de todos los elementos dentro de Unity, reduciendo la cantidad de memoria necesaria para alcanzar una alta precisión. Además, la cantidad de elementos visibles alrededor del jugador es limitada, por lo que, al centrar el mapa en base al jugador, nunca se perderá precisión.

El **tercer y último punto** a resolver en el mapa es el manejo del jugador, tanto su control por coordenadas, como la posibilidad de extraer coordenadas del mismo. Una vez que sea permisible dicho control al activarse la ubicación, el controlador podrá mover el personaje conforme el jugador se mueva en el mundo real. Por otro lado, una vez se puedan extraer las coordenadas del mismo se podrán realizar las acciones en base a ubicaciones que requieren los elementos geo-posicionados. La principal ventaja del acercamiento que se basa en el uso de un personaje o avatar es que permite que el movimiento del jugador sea dinámico y visual, de la

misma forma que se realiza en Pokémon GO, utilizando animaciones de pasos que se distribuyen en el tiempo, frente al teletransporte del personaje. Además de moverse, el avatar puede responder a la dirección hacia la que mira el jugador, siendo una respuesta directa a los movimientos y permitiendo que la acción de "mirar hacia" (*lookAt*), sea más evidente.

Para realizar esto, es necesario incluir una nueva clase en el jugador actual que se encargue de proporcionar métodos para el movimiento en base a coordenadas, que realice la conversión a coordenadas reales, y utilice la clase de Unity ThirdPersonController para realizar el movimiento del avatar. Al finalizar, su actualización, la nueva posición del jugador con el movimiento que se haya avanzado en el tiempo se transforma a coordenadas, que serán accesibles de forma pública. Paralelamente, en base al estado de la brújula el avatar presentará una rotación siempre y cuando no se encuentre en movimiento en ese instante. Finalmente, de manera pública se podrá consultar si el jugador está mirando en una determinada dirección o hacia un determinado punto. El diagrama de la Fig. 6.3 presenta la arquitectura mencionada.

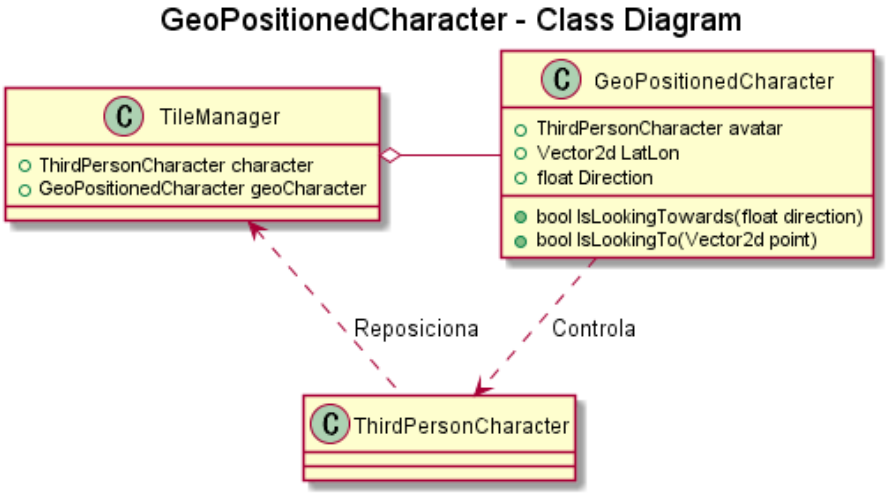

Fig. 6.3 Incorporación del GeoPositionedCharacter al sistema

Pese a que existirán los *plugins* de Mapzen y las factorías, en la primera versión de la implementación no se utilizarán por dos motivos: evitar sobrecargar los dispositivos más antiguos y asegurar que la incorporación de dichos elementos se realiza de la forma correcta, evitando que las formas y tamaños interfieran en las interacciones con el juego. Sin embargo, podrían ser perfectamente utilizables una vez se hagan los primeros experimentos y se pruebe el sistema completo.

6.1.2. Elemento de mapa como editor de Unity

En este subapartado se expone el análisis, diseño e implementación de un elemento compatible con el editor de Unity que muestre un mapa y permita interactuar con él para añadir formas geométricas y otros elementos gráficos, que será utilizado posteriormente como base para las extensiones de uAdventure de geoposicionamiento. Como proveedor gráfico se establece un elemento común al otro mapa que permitirá compartir los tiles entre editor y ejecución y utilizar la misma herramienta de carga. Dichos tiles pueden proveer de varios servicios, destacando principalmente por su velocidad de carga el de OpenStreetMap.

Este proveedor gráfico, llamado TileProvider, extrae su funcionalidad básica del interior de TileImagePlugin incluido en Mapzen GO, desacoplándose de éste en una clase independiente. El procedimiento para la carga se realiza mediante UniRx, que encapsula las conexiones web nativas de Unity para hacerlas más intuitivas y responsivas, evitando las corrutinas y utilizando *callbacks*. De esta forma, TileProvider da un paso más en la encapsulación y provee un método LoadTile que devolverá la textura con el tile siempre que la descarga sea correcta. Además, TileProvider hace de sistema de cacheado, manteniendo en memoria los *tiles* recientes para evitar descargas innecesarias.

Una vez establecida la fuente de información gráfica se construyó la clase GUIMap, compatible con el ciclo de vida de la GUI de Unity. El ciclo de vida de la GUI de Unity se compone por llamadas al método OnGUI conforme se suceden eventos[34]. Estos eventos son, en general, de tres tipos: de interacción del usuario, de pintado y de *layouting*. Para maximizar la compatibilidad en los diferentes entornos de GUI se evita el evento de *layouting* que permite al elemento tomar las dimensiones de forma automática a través del uso de GUILayout. Para todos los eventos, entonces, GUIMap recibe le rectángulo o área que debe de utilizar.

Dentro del evento de pintado, GUIMap realiza un clip de la interfaz en base al rectángulo recibido, que recortará todo aquello que se encuentre fuera de dicho clip. La primera parte del pintado consiste en el pintado de *tiles* del mapa. Para ello, se adquieren los tiles necesarios a través de TileProvider. Dado que el rectángulo del mapa es dinámico, pero está definido en píxeles, y existe una conversión directa entre píxeles y coordenadas, es posible calcular los *tiles* que serán necesarios para cubrir la superficie total del mapa. Sin embargo, el área del rectángulo

[34] https://docs.unity3d.com/ScriptReference/Event.html

se encuentra en coordenadas relativas al centro, por lo que, para que los píxeles del rectángulo sean válidos es necesario convertirlos a píxeles globales a través de la siguiente fórmula.

$$PixelCentral = LatLonToPixel\ (\ CoordenadaCentral\ )$$
$$TileInicial = PixelToTile\ (\ PixelCentral - [AnchoRectángulo\ /\ 2, AltoRectángulo\ /\ 2]\ )$$
$$TileFinal = PixelToTile\ (\ PixelCentral + [AnchoRectángulo\ /\ 2, AltoRectángulo\ /\ 2]\ )$$

Una vez definidos los *tiles* inicial y final, dado que ambos son vectores bidimensionales, se itera sobre la coordenada x e y para dibujarlos en el espacio del mapa. Dado que los *tiles* del borde no se pintarán de forma completa es necesario hacer un dibujado parcial a través de coordenadas en la textura, conocidos comúnmente como coordenadas "uv". Puesto que los píxeles de la GUI crecen desde la esquina superior izquierda mientras que las coordenadas "uv" lo hacen desde abajo a la izquierda, en el proceso se convierten las coordenadas. Este proceso de cálculo de "uv" sólo requiere de dos parámetros para realizarse: el área que ocuparía la textura original y el área que se pretende dibujar. Para simplificar este cálculo, la clase ExtensionRect añade nuevas funciones a Rect que permiten transformar ambos parámetros en el valor apropiado. El proceso se realiza en cuatro pasos: se obtienen los píxeles inicial y final que ocupa el *tile* (según TMS) y se convierten a píxeles relativos restando el pixel central, se genera un rectángulo con dichos píxeles, se realiza la intersección del rectángulo con el área total que dispone GUIMap y, finalmente, se realiza la transformación a coordenadas "uv" del área intersecada con el área del rectángulo original. El resultado permite dibujar a través de la función GUI.DrawTextureWithTexCoords el *tile* de una forma más eficiente, evitando dibujar píxeles que no se verán. La Fig. 6.5 muestra un ejemplo de mapa contenido en una interfaz de Unity y la Fig. 6.4 muestra la relación de las clases GUIMap, TileProvider y ExtensionRect. Además del pintado de *tiles*, una vez se definen los elementos del mapa, éstos se pintarán a continuación.

El mapa puede contener dos tipos de elementos: geometrías geoposicionadas y recursos gráficos geoposicionados. El primer grupo permite definir áreas de influencia en base a geometrías que asientan sus puntos en coordenadas. Estas geometrías geoposicionadas se encuentran definidas en diversos estándares como GeoJSON o GML [45] (Geo XML). Dado que

el modelo de uAdventure se serializa en XML y existen equivalencias entre ambos estándares, se utilizará GML como definición para las geometrías, con algunos extras funcionales.

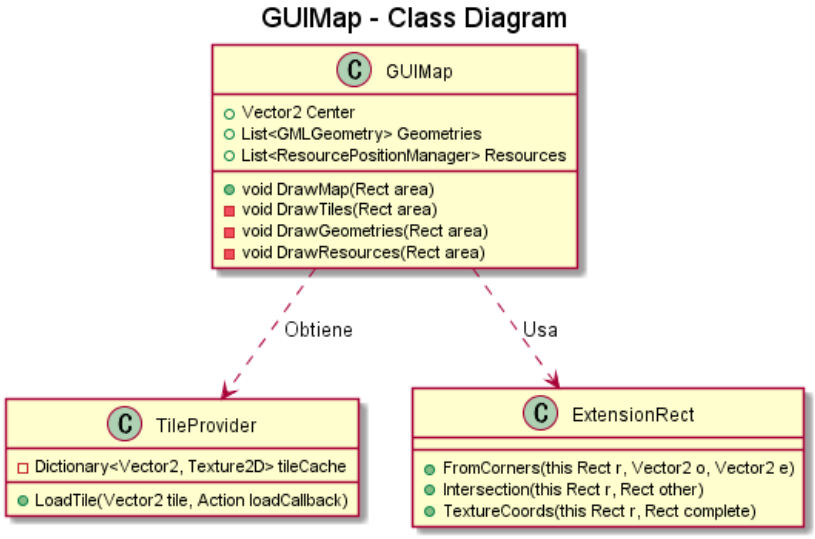

Fig. 6.4 Diagrama de clases de GUIMap TileProvider y ExtensionRect.

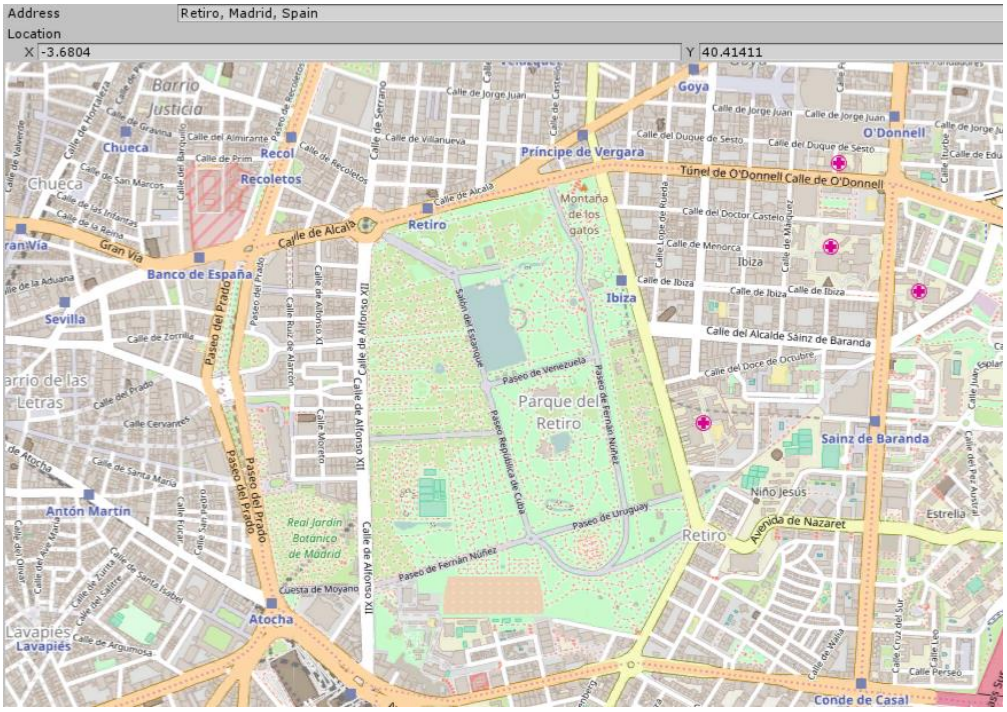

Fig. 6.5 Mapa integrado en el editor de Unity, centrado en el parque del retiro.

Dentro de GML se definen tres tipos de geometrías geoposicionadas: puntos, caminos y polígonos. El primero se identifica con la etiqueta "*point*" y se caracteriza por contener una única

coordenada. Asimismo, los caminos se identifican por la etiqueta "*lineString*" y por contener un conjunto ilimitado de puntos. Por último, los polígonos difieren levemente. Su etiqueta es "*polygon*" pero en su interior, el área puede definirse de múltiples formas. En este caso se ha utilizado la más sencilla: definir el polígono por el anillo exterior de puntos. Además de estas geometrías, GML contempla curvas, superficies o cajas, pero por su complejidad, no se han implementado en esta versión. Sin embargo, para cubrir estas necesidades se ha añadido un parámetro extra a todas las geometrías: el radio de influencia.

Para representarlas, la clase GMLGeometry contiene la información que las define, siendo sus atributos el tipo de geometría, la lista de puntos y el radio de influencia. En su interior, GUIMap mantiene una lista pública de todas las GMLGeometry que debe manipular. Utilizando esas referencias, al finalizar el renderizado de los tiles procede a renderizar las geometrías. Para ello, utiliza lo métodos para el dibujado de líneas, arcos y el pintado de polígonos convexos de la clase Handles de Unity. Los puntos se ven representados por círculos completos de borde negro e interior azul. Los caminos, antes de representar cada punto dibujan una línea negra entre los puntos. Por último, los polígonos primero dibujan el relleno, después una línea que une todos sus puntos que, a diferencia de los caminos, conecta también al punto inicial y final, y, finalmente, dibuja todos sus puntos. Para el rellenado del interior, Handles provee de un método para el dibujado de polígonos convexos. Sin embargo, los polígonos definidos por las geometrías pueden ser tanto cóncavos como convexos, por lo que, para solucionar este problema, se utiliza la propiedad que dice que todo polígono cóncavo puede dividirse en un número finito de polígonos convexos que, en conjunto, cubren el área original. Existen algoritmos óptimos en tiempos polinómicos para realizar esta tarea dividiendo en triángulos, a lo que se llama triangulación [46]. La triangulación permite dividir un polígono en polígonos convexos dado que no existe ninguna forma de crear un triángulo cóncavo. Por ello, se utiliza la librería de triangulación para dividir el polígono en triángulos que se pintarán de forma individual con la función DrawConvexPolygon.

Una vez se han representado las geometrías se dibuja el área de influencia de la misma, desde la cual se detectaría al jugador. En el caso de los puntos, el cálculo del área de influencia es trivial ya que consiste en transformar los metros a píxeles (con la conversión proporcionada por TMS) y dibujar un círculo alrededor del punto al radio de influencia. En el caso de polígonos y rectas, la solución no es tan evidente. Pese a que para polígonos convexos la solución se limita

a redibujar las aristas a la distancia del radio de influencia y unirlas mediante arcos tangentes, el caso de los polígonos cóncavos es mucho más complejo de solucionar. Por ello, se optó por la utilización de la librería externa Clipper[35] de Angus Johnson que se distribuye bajo licencia Boost. Esta librería permite dos funcionalidades con un alto grado de optimización: realizar operaciones sobre las áreas de polígonos como la intersección, la unión, etc., y, además, permite la extensión (o retracción) de polígonos en base a un *offset*. Gracias a ello, con una simple función se pueden calcular de forma óptima las áreas de influencia de caminos y polígonos cóncavos y convexos. La Fig. 6.6 muestra los tres tipos de geometrías y sus radios de influencia de 10 metros. Un punto aparece sobre la estatua de Alfonso VI, un camino recorre el parque con varios puntos y un polígono describe el área de reunión.

Fig. 6.6 Representación de los tres tipos de figuras (punto, camino y polígono), junto con su área de influencia.

Además de las figuras geométricas geoposicionadas, el mapa también puede contener recursos gráficos basados en texturas. Dentro de uAdventure, los elementos de las escenas utilizan recursos gráficos para dibujarse. Al poder situar recursos gráficos en el mapa se permite utilizar elementos de uAdventure en el nuevo tipo de escenas. Dentro de los objetivos que se han propuesto en el proyecto, los objetos de uAdventure pueden colocarse en el mapa de tres formas

[35] http://www.angusj.com/delphi/clipper.php

diferentes: la primera y más evidente es el posicionamiento del centro en una coordenada; la segunda, requiere de un radio y una distancia con respecto al centro del mapa, de tal forma que el elemento siempre permanecerá en el mismo lugar independientemente que el jugador se mueva; la tercera y última, es mediante el último posicionamiento relativo a la pantalla y, aunque similares, se diferencia del segundo tipo en que este no se ve influido por el mapa, por lo que cambios de escala, zoom o movimientos de la cámara no afectarán a su percepción.

Para representar dichos elementos en el mapa con una interfaz común, permitiendo que se pueda extender añadiendo nuevos tipos, se utiliza la interfaz ResourcePositionManager, que contiene tres métodos: Configure, Draw y Repositionate. El mapa utilizará Draw para dibujarlos, mientras que Repositionate permitirá modificar su posición más adelante. Configure, aunque no se utilizará hasta que se integre el mapa con uAdventure, permitirá configurar el PositionManager de forma genérica a través de un diccionario de parámetros y valores.

El esquema final de la arquitectura puede verse en la Fig. 6.7, donde aparecen tanto GMLGeometries como los diferentes PositionManagers.

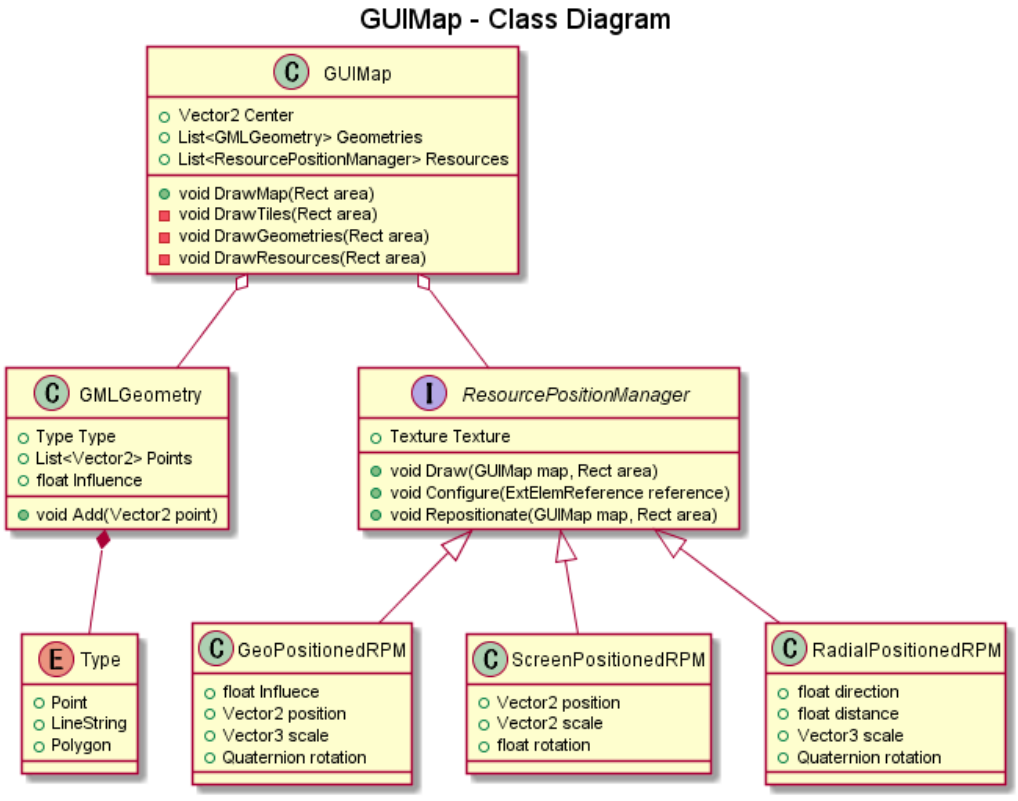

Fig. 6.7 Arquitectura de elementos que aparecen en el mapa.

Las tres implementaciones de PositionManager son GeoPositioned, ScreenPositioned RadialPositioned que realizan respectivamente los tipos de posicionamiento descritos. A continuación, se describen los diferentes dibujados de los elementos, mientras que, los reposicionados serán tratados más adelante cuando se trate la interacción con el mapa. GeoPositioned cuenta con cuatro atributos, el rango de influencia, la posición, que representa las coordenadas y además la escala y la rotación de elemento. En su método Draw se limita a dibujar la textura, utilizando la dimensión de píxeles reales (obtenidos por TMS) por lo que se ve afectado por el zoom del mapa, posicionando el centro de la imagen en las coordenadas asociadas a position. RadialPositioned por su parte, dibuja la textura igual que GeoPositioned, pero, dado que no cuenta con una posición, utiliza el centro actual de GUIMap, una dirección y un radio para situar el objeto. Para ello, utiliza la siguiente fórmula.

*PixelCentral = PixelCentralMapa + [Cos (dirección), Sen (dirección)] * distancia*

Finalmente, ScreenPositioned posiciona el elemento en base a su posición sobre la pantalla. Para ello, toma como referencia una dimensión de pantalla de 800x600 y genera una nueva posición escalándola en base a las dimensiones del rectángulo del mapa.

GUIMap, además de poder ser visualizado (en su método DrawMap) permite la interacción con los elementos de su interior. En su interacción más básica, provee de controles de deslizamiento del mapa, moviendo su centro en base a la delta del ratón, cambios de zoom, utilizando la rueda del raton y, además, si se ha realizado clic izquierdo sobre el mapa, al finalizar el dibujado el mapa retornará *true* al elemento que solicitó su dibujado, por lo que, de manera externa, pueden realizarse acciones. Para que las acciones puedan utilizar la información del ratón sobre el mapa, previo al retorno del valor *true* se realiza el cálculo de las coordenadas sobre las que se encuentra el pixel que se encuentra el ratón y se almacena en la variable pública de GUIMap, GeoMousePosition.

Además de este control, las figuras geométricas que estén en su interior pueden ser manipuladas. La manipulación de las figuras permite interacción diferente en base a la figura geométrica que se seleccione. Para seleccionar la figura, la clase GMLGeometry proporciona un método público Inside que retornará true siempre que se considere que el ratón está en el interior

de la figura o suficientemente cerca de ella. En el caso de los puntos, siempre que se encuentre a un pequeño rango se considerará seleccionado dicho punto. Para el caso de líneas se calcula que el ratón esté cerca del punto o de la línea utilizando el producto escalar y el producto vectorial de manera sencilla. Para calcular si la coordenada del ratón está dentro del polígono es sencillo para polígonos convexos, pero, dado que pueden ser cóncavos, se utiliza un método basado en el número de aristas con las que intersecta el eje x [47]. De ser impar, el punto es interior y de ser par, exterior.

Una vez seleccionada la figura, si se desplaza el ratón sin soltar el botón izquierdo se considerará que se ha arrastrado la figura y esta se desplazará por el mapa. Además, si el ratón se pulsa suficientemente cerca de un punto, en lugar de desplazar la figura, será el punto el que se desplazará, permitiendo alterar los puntos y las formas de los caminos y polígonos.

Por otro lado, la selección de recursos gráficos no ha sido necesaria para esta versión, sino que cuando se desea mover, se llama al método Repositionate en el PositionManager y este adquirirá la posición actual del ratón en el mapa. Dado que cada PositionManager actúa diferente, Repositionate no devuelve un valor, sino que almacena la información necesaria en el diccionario de parámetros que se recibió a través de Configure. Gracias a la arquitectura flexible que proporciona el uso de un diccionario genérico pueden integrarse nuevos PositionManagers que podrían posicionarse de formas diferentes sin tener que alterar los PositionManagers existentes.

El mapa GUIMap, por sí sólo, provee de funcionalidad suficiente para el control y posicionamiento de todos los elementos que se crearán más adelante. Sin embargo, pese a que el mapa puede centrarse a través de la latitud y longitud, este tipo de control no es intuitivo para los usuarios más comunes. Por ello, se incluye además del método de entrada de latitud y longitud, un campo de texto que permite acceder a un servicio de búsqueda inversa de posiciones (*reverse geocoding*). A través de direcciones, nombres de lugares importantes, ciudades además de muchos otros tipos de palabras clave, el usuario puede escribir el lugar que desea centrar y, tras una pequeña pausa de pocos segundos en la que se realiza la consulta, el usuario puede seleccionar directamente la dirección en la que desea centrar el mapa. Por ejemplo, al buscar “Retiro” el usuario podrá ver como primera opción el Parque del Retiro de Madrid y centrar su mapa en él, mejorando drásticamente la facilidad de uso e interacción con el mapa.

Dado que Unity no incluye desplegables se implementó una clase desplegable (DropDown) que es compatible tanto con GUI como GUILayout, en la cual se utiliza un elemento TextField. Siempre que TextField tenga el foco del usuario se mostrará un *scroll* en el espacio inferior que ocultará los posibles elementos de GUI que pudiera haber bajo él. Para detectar el foco se utilizan los métodos SetNextControlName y GetNameOfFocusedControl de GUI para dicho propósito. Dado que el *scroll* aparecerá sobre diversos elementos de GUI es necesario que su pintado se haga más adelante. Es importante mencionar que, si el pintado se hace más adelante, puede ocurrir que los elementos que se dibujen en el momento intermedio capturen el foco del usuario y, para evitarlo, es necesario capturar el evento actual siempre que el ratón esté dentro del rectángulo que se utilizará para el área de *scroll*. Por ello, la clase DropDown requiere de dos llamadas: Begin, que dibuja el campo de texto, controla el foco, captura el evento si es necesario y retorna el valor actual del campo de texto, y End, que dibuja el *scroll* tras haber pintado toda la GUI y, si se selecciona un elemento dentro del scroll, retorna true. Para dibujar el *scroll* una vez se dibuja el TextField, se crea un rectángulo aprovechando el atributo público *height* que permite definir la altura del mismo, que se posicionará justo debajo del rectángulo utilizado en TextField. En el caso de usar GUILayout, el método GetLastRect de GUILayoutUtility permite conocer dicho rectángulo. Siempre que el evento no haya sido capturado ya por otra componente se capturará el evento usando Use durante el método Begin. Si el evento se capturara, éste se devuelve a su estado original cuando se haga la llamada a End, justo antes de dibujar el *scroll*, lo que permitirá a los elementos del interior del *scroll* poder capturar el evento. Para hacer el *scroll* se utiliza BeginScrollView y en su interior, se pintan tantos botones como elementos posibles hay para seleccionar. Para evitar que tengan aspecto de botón, se utiliza un estilo personalizado de GUISkin, que funciona para todo el DropDown, modificando el color de fondo y las barras de control del *scroll*. La Fig. 6.8 muestra un DropDown con elementos en funcionamiento.

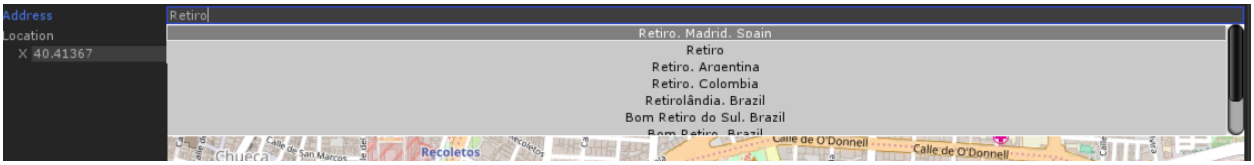

Fig. 6.8 Dropdown sobre un campo Vector2 y un GUIMap

Utilizando la clase DropDown que permite crear elementos desplegables y la API de búsqueda inversa (*reverse geocoding)* incluida en Mapzen GO. Para cargar los lugares en base a

un texto, la clase SearchPlace utiliza uno de los servicios de Mapzen para dicho propósito[36]. Sin embargo, dado que dicho servicio tiene un máximo tiempo de utilización de cinco veces por minuto, si se hacen tantas llamadas como cambios de texto se producen se agotará dicho límite mientras el usuario escribe. Para evitarlo, se utiliza una variable temporal que se resetea al introducir un nuevo carácter y que, al llegar a tres segundos sin cambio, procede a realizar la búsqueda. Una vez SearchPlace haya terminado la búsqueda, se extraen los nombres y coordenadas asociadas almacenadas en la variable "dataCache" y se introduce en el DropDown la lista de nombres. Si se selecciona alguno de ellos, el método End de DropDown retornará true y a través de Value podremos conocer el valor seleccionado (ya que el campo de texto no se habrá actualizado aún). A través de un diccionario inverso, se recuperan las coordenadas y finalmente se centra el mapa en dicha posición.

Para simplificar el uso de SearchPlace y el DropDown, la clase PlaceSearcher encapsula dicho comportamiento, heredando de DropDown, pero incorporando el funcionamiento descrito antes tras las llamadas a Begin y End y ofreciendo una variable pública nueva con el resultado de las coordenadas seleccionadas. El diagrama de la Fig. 6.9 muestra la relación entre una GUI cualquiera, PlaceSearcher y GUIMap.

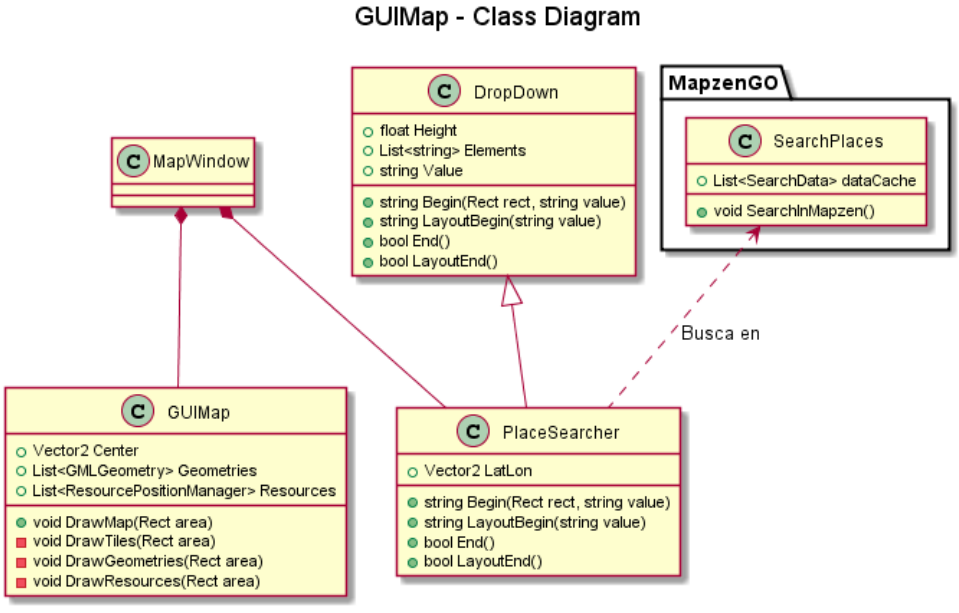

Fig. 6.9 Diagrama de clases de GUI usando GUIMap y PlaceSearcher.

[36] https://mapzen.com/documentation/search/reverse/

En su conjunto, GUIMap y PlaceSearcher ofrecen la funcionalidad necesaria para los *plugins* de la capa de edición que se vayan a implementar para uAdventure en las próximas fases del desarrollo. A continuación, se expone la parte de creación y lectura de códigos QR.

6.1.3. Lectura de QR en ejecución

Además del geoposicionamiento, en este proyecto se ha desarrollado posicionamiento en interiores. Este posicionamiento se hace en esta primera fase a través de códigos QR [48] aunque en el futuro se podrá implementar posicionamiento en interiores siguiendo la misma línea a través de Beacons [49] o Wifi [50].

Para la lectura del QR es necesario el uso de input visual sobre el cual realizar la búsqueda. Dentro de Unity la opción más viable es el uso de WebCamTexture, destinado a la captura de imágenes en tiempo real a través de una cámara en el terminal. Si bien el uso de WebCamTexture no está comprobado para todos los dispositivos será el acercamiento que se realice por ahora. Otros tipos de cámaras que se pueden utilizar son las del *plugin* de pago de Vuforia para Unity, por ejemplo.

Sobre el objeto WebCamTexture se extrae la textura real de tipo mapa de bits sobre la cual se procederá a realizar la búsqueda del QR. Para ello se utiliza la popular librería XZing[37] para el escaneo de diversos códigos de barras, BIDIs o QRs. Dado que XZing no es soportado de forma nativa en Unity ni en todas sus plataformas, se ha utilizado una variante de XZing.NET[38] de Cedric Poon capaz de soportar el decodificado en multiplataforma, incluyendo Android e iOS.

Para su uso, se ha añadido la clase QRReader que maneja las texturas y conecta la información proveniente de la cámara con el BarcodeReader de XZing. Para iniciar el lector y pararlo se utilizan los métodos púbicos Enable y Disable. Durante Enable, se inicia la cámara y se espera medio segundo para asegurarse de que ya se recibe información visual. Para hacer la búsqueda del QR se utiliza un *thread* secundario. Dado que Unity no permite acceder a elementos de UnityEngine desde *threads*, para que el *thread* se conecte con Unity, en el método Update del QRReader se extrae la matriz de colores de la imagen actual y se almacena en la variable accesible para el *thread.* Éste, a continuación, lee la información de la cámara desde dicha variable cada 200 milisegundos para evitar la sobrecarga del juego y si detectara algun QR,

[37] https://github.com/zxing/zxing

[38] https://github.com/cedricpoon/ZXing.Net

almacenaría en un *string* el resultado. Finalmente, desde Update de nuevo se lee el *string* de salida y, si ha sido modificado, se notificaría a los objetos que lo soliciten. Para hacer este estilo de notificaciones de tipo Observable se utiliza un UnityEvent[39] que funciona de manera similar a un delegado, notificando a los objetos que se hayan registrado en él. Para deshabilitar el QRReader, en Disable se deshabilita el *thread* sin llegar a detenerlo, evitando que utilice los recursos en el escaneo, permaneciendo en un bucle de pausa constante.

6.1.4. Creación de QR en editor

Para la creación de QRs en el lado del editor, se ha integrado la librería QRCoder[40] de Raffael Herrman compatible con Unity en la parte de editor, dado que la implementación de XZing sobre Unity elimina las funcionalidades de creación. Dicha librería permite la creación de texturas de Unity de tipo Texture2D que pueden utilizarse desde cualquier parte del programa, tanto en editor como en ejecución.

Sin embargo, QRCoder contaba con un pequeño fallo de cara a la compatibilidad con el editor. Al generar los QRs, la coordenada Y aparecía invertida por lo que a la hora de generar los QR éstos variaban de forma entre la imagen del editor y las imágenes exportadas. Esto, pese a no ser un problema de cara a la ejecución dado que los códigos QR permiten este tipo de lectura en inverso o espejo, podía dar a confusión del usuario por lo que se corrigió dicha funcionalidad en el método de generación de códigos QR.

Más adelante se utilizarán tanto los módulos de ejecución como de edición para la creación de *plugins* para uAdventure.

6.2. Modificaciones de uAdventure y extensibilidad: uAdventure *plugins*

En este apartado del capítulo del desarrollo se exponen las nuevas modificaciones arquitectónicas de uAdventure que mejoran su extensibilidad, además de las mejoras y correcciones que se han hecho al software original a lo largo del proyecto.

Nada más comenzar esta segunda fase del desarrollo se detectó un error grave que imposibilitaba la integración de diversos elementos, incluidos elementos de sistema e incluso, el uso de MapzenGO. Este problema se ha heredado de las ventajas que supone Java en el

[39] https://docs.unity3d.com/Manual/UnityEvents.html

[40] https://github.com/codebude/QRCoder

desarrollo. Principalmente, en Java, todos los ficheros de una carpeta se convierten automáticamente en miembros del paquete con dicho nombre. Sin embargo, en C# no existe el concepto de paquete y las carpetas sólo suponen una forma organizativa del código para el programador. Como sustituto a dicho elemento, C# incluye espacios de nombres (*namespaces*)[41] que permiten al programador acotar el rango de uso de una clase a dicho espacio. Para poder acceder a los elementos de un namespace, al igual que java los importa, C# enumera los espacios de nombres que desea utilizar en un fichero.

Unity no es un entorno enfocado a la programación por lo que es común que elementos como los espacios de nombres pasen desapercibidos para los programadores de la plataforma que desean realizar soluciones para un proyecto de pequeño o mediano tamaño. Sin embargo, debido a que uAdventure es una plataforma de gran envergadura, con elementos muy genéricos como efectos, animaciones o acciones, es muy común que éstos sobrescriban a otros elementos empeorando las capacidades de integración.

Este hecho se hizo inmediatamente notorio al integrar UniRx, que se utiliza en MapzenGO para accesos remotos. Dado que uAdventure cuenta con la clase Action, pero las funciones lambda por defecto son en C# llamadas Action, resultó en un caos dentro del código que imposibilitaba la compilación. Otros elementos nativos de Unity como la clase Animation quedaban inaccesibles debido a dicho factor.

Por ello, la primera y gran medida que se implementó en uAdventure fue la integración de espacios de nombres. Para ello se usaron cinco grandes grupos, todos ellos miembros del namespace uAdventure. Estos son: Core, Runner, Tracker, GameEmulator y Editor. Éstos, se caracterizan por contener los siguientes elementos:

- Core: Cuenta con el modelo y su controlador principal, así como el control del estado. Además, incluye todo lo necesario para la serialización y deserialización.
- Runner: Engloba los elementos que principalmente forman parte de la ejecución de los juegos. Estos son, en general MonoBehaviours como la clase Game, SceneMB, ObjectMB, etc., así como el ResourceManager para cargar *assets*.
- Tracker: Contiene todo lo necesario para la conexión con Learning Analytics, incluyendo principalmente el Tracker de RAGE.

[41] https://msdn.microsoft.com/es-es/library/z2kcy19k.aspx

- GameEmulator: Incluye los elementos que conforman el emulador de juegos.
- Editor: Abarca todos los elementos de edición, creación de elementos y gestión de assets. Además, incluye los DataControl para acceso al modelo y todas las ventanas que ejercen como controladores del modelo.

Si bien la inclusión de espacios de nombres supone un cambio mínimo en el proyecto, supuso una ardua tarea de modificación de código dado que se tuvieron que editar todos los archivos existentes en el proyecto para incluir el espacio de nombre adecuado. La cifra de archivos de código modificados ascendió a 569. El diagrama de la Fig. 6.10 muestra a alto nivel la relación entre los espacios de nombres. En la figura, además de identificarse los espacios de nombres se observan las diferentes relaciones. Core es el elemento principal del que se nutren tanto Runner como Editor. Runner, además, requiere la dependencia con Tracker dado que lo utiliza para el envío de LAs. Por último, GameEmulator es un manejador de Runner. Si bien todos los paquetes son en general bastante independientes existen algunas referencias cruzadas puntuales que se deberán solucionar en futuras versiones del código.

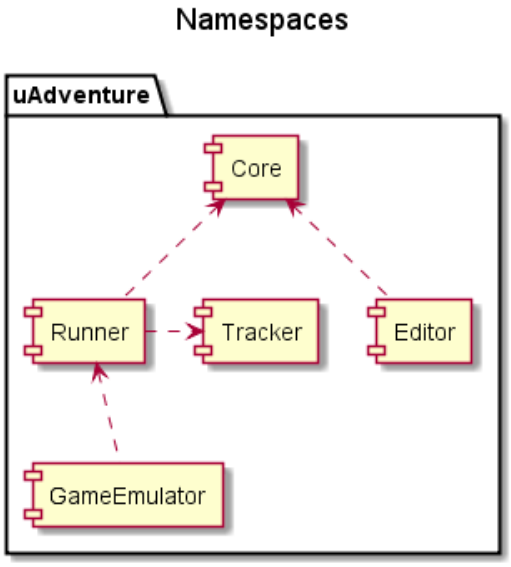

Fig. 6.10 Espacios de nombres de uAdventure y sus relaciones

Tras la incorporación de los espacios de nombres, se hizo una revisión general de la funcionalidad del proyecto para verificar que todo funcionara correctamente. El cambio de espacio de nombres no generó errores a priori, pero sí se detectaron los siguientes errores que se solventaron.

6.2.1. Reparación de errores de uAdventure

En primer lugar, se podía trabajar con proyectos existentes, pero, sin embargo, fallaba la creación de los mismos. Esto se debía a que controller nunca realizaba la creación de los ficheros utilizando newFile. En su lugar, la llamada a LoadOrCreateProject llamaba a Init sin introducir la ruta de destino y este fallaba en la carga. La primera modificación utilizó newFile para la creación del fichero, lo que derivó en un nuevo error dado que newFile utilizaba el diálogo de selección de ficheros en su interior. Este error se daba si no se seleccionaba una ruta o la ruta ya existía. Para solucionarlo, se extrajo toda la funcionalidad del selector de ficheros a la ventana de creación de la aventura, donde se podría cortar el flujo de carga si se cancelaba el proceso. Finalmente, dado que LoadOrCreateProject es un método de uso poco intuitivo se decidió dividir en CreateProject y LoadProject, lo que permite utilizar con certeza LoadProject en el caso de la carga de ficheros y evitar la creación de aventuras al intentar cargar ficheros en ubicaciones inexistentes.

Por otro lado, por la forma en la que está hecho uAdventure puede no necesitar ni cargar ni crear un proyecto, ya que en su interior contiene una carpeta *currentgame* que mantiene el modelo actual. Por ello, durante los guardados se ha eliminado la necesidad de utilizar el directorio externo, mejorando la capacidad de integración con sistemas de control de versiones que mantengan el proyecto de Unity únicamente.

Además de estos errores, existía un error crítico al tratar de reutilizar un recurso que ya se encuentre en la carpeta *currentgame*. Dado que, en el proceso de importación del recurso seleccionado se sobreescribe cualquier recurso existente en la carpeta *currentgame* con el nuevo recurso, si el archivo de destino que se pretende escribir se encuentra abierto, el proceso fallará. Dado que el origen y el destino es el mismo se produce dicho caso. Para evitarlo, sólo se copian los ficheros siempre que su origen y destino no sea igual.

6.2.2. Mejoras para el editor principal

Antes de entrar en detalle sobre las características que se han implementado de extensibilidad, en este apartado se van a desarrollar las mejoras que se han realizado a la encapsulación y facilidad de uso de la ventana principal de Unity. Inmediatamente después de haber realizado la encapsulación de todos los elementos, la ventana principal se pudo hacer extensible con mayor facilidad.

La clase EditorWindowBase es el contenedor principal de todos los editores existentes para elementos de uAdventure. Estos elementos pueden ser escenas, objetos, personajes, entre otros. Por ello, la ventana principal realiza su flujo principal, en el que dibuja el menú de la izquierda y, a continuación, en función de la ventana que se encuentre seleccionada, se dibuja en la ventana principal el contenido del editor seleccionado. Una vez se selecciona el editor correspondiente, existe una clase individual para cada editor encargada de manipular el modelo correspondiente que hereda de la clase BaseWindow, permitiendo que el editor se comporte de forma polimórfica mostrando el editor seleccionado sin necesidad de conocer qué tipo de editor es. Pese a tener un buen planteamiento arquitectónico, la ventana principal mantenía una alta cohesión con todos los posibles editores interiores que podía manipular. Principalmente, esto se dividía en dos errores de diseño que incrementaban la cohesión del modelo.

El primer error residía en que, pese a que existían múltiples botones laterales para el control de los editores, todos ellos eran manipulados por la ventana principal, que se encargaba de adquirir todos los recursos gráficos y textos que debían aparecer en el botón. Además, para seleccionar la ventana de editor adecuada a cada botón, todos los botones se dibujaban de forma secuencial, diferenciando el resultado individual de pulsar cada botón.

El segundo error residía en que, al seleccionar algunos de los editores, éstos podían extenderse para mostrar brevemente, en un submenú, los elementos que podían seleccionarse para ser editados. Debido a esto, una parte del modelo era conocida por la ventana principal para poder ser mostrada en dicho lugar. Por ello, cada botón requería de un funcionamiento independiente, no generalizado para la recopilación de información del modelo correspondiente al editor. Además, en cada submenú existían botones para la creación, duplicación y destrucción de los elementos presentes en el submenú. Dado que, para controlar los elementos era necesario manipular a través del Controller el modelo adecuado, para cada uno de los menús existían métodos que cumplían dichas funciones para la manipulación de los elementos.

Para solucionar ambos errores de diseño se realizó un único cambio que permitía, por un lado, eliminar toda referencia a los recursos y textos que aparecen en los botones de cada menú y, por otro lado, eliminar todo el control que se realiza sobre el modelo en la ventana principal. Este cambio es sencillo: integrar tanto el botón como el panel inferior al botón dentro de las ventanas de cada editor. Dado que cada editor ya existía encapsulado en LayoutWindow (que hereda de BaseWindow) se decidió extender dicha clase en la nueva clase

EditorWindowExtension, clase que más adelante se utilizará como pilar principal para hacer extensible la ventana principal. EditorWindowExtension tiene dos partes fundamentales: el dibujado del menú y el dibujado de su ventana principal. Dado que, el dibujado de la ventana principal ya se encuentra correctamente encapsulado en LayoutWindow, es el método OnGUI el encargado de asumir dicha labor en la interfaz. Para el dibujado del menú correspondiente a dicho editor, el editor principal llamará a DrawLeftPanelContent cuando se encuentre dibujado todos los menús del lateral izquierdo. Pese a que esto abre las puertas a que cada editor pueda dibujar el menú rompiendo el estilo principal de la ventana, esto también supone una ventaja ya que no todas las ventanas tienen que adaptarse al mismo formato. Finalmente, para cambiar el editor en EditorWindowBase al seleccionar uno de los editores, EditorWindowExtension cuenta con un delegado OnRequestMainView que permite que EditorWindowBase sea notificado cuando el editor solicite acceder a la vista principal.

Una vez establecida la interfaz entre la ventana principal y sus extensiones, el paso siguiente fue trasladar todo el dibujado del menú a cada ventana. Uno de los errores que tenía el diseño anterior era la repetición de código en todas las partes que aparecía el menú. Debido a que esto es una mala práctica, utilizando herencia se ha realizado una jerarquía de clases que se encarga de proporcionar diferentes clases que realizan el dibujado del botón y del menú, así como de notificar de forma automática el acceso al delegado. Esta jerarquía se divide en tres capas más especializadas.

La primera capa más genérica es donde reside EditorWindowExtension, que permite a la extensión dibujar lo que quiera en el panel con total libertad. Justo bajo ella y siguiendo el patrón anterior existe la clase ButtonMenuEditorWindowExtension. Como su nombre indica, esta clase añade la funcionalidad de la división Botón-Menú, controlando que, al seleccionarse el botón, se notifique al delegado de manera automática, tomando el control de la ventana principal en ese momento. Para ello, proporciona un control de métodos abstractos que deberán implementarse en las clases inferiores para el dibujado del botón y del menú. Estos métodos abstractos son: dos, DrawButton y DrawMenu, aunque a estos dos, se añade un método OnButton que notificará a la jerarquía inferior cuando el botón se ha pulsado. Para mejorar el aspecto visual, ButtonMenuEditorWindowExtension añade un control de animaciones que permitirá que los menús aparezcan de manera progresiva como ya se hacía antiguamente en eAdventure. Para ello, se utiliza el método BeginFadeGroup y EndFadeGroup, siempre que la propiedad protegida

UseAnimations esté activada. El elemento inferior deberá utilizar la variable protegida "extended" para controlar cuando desea mostrar y ocultar su menú. A continuación, en la jerarquía se encuentran dos casos de herencia diferentes en base a la necesidad de mostrar o no controles de menú.

El caso más sencillo, en el que no sea necesario realizar ninguna gestión de menú, la clase DefaultButtonEditorWindowExtension implementa tanto el dibujado del botón como el dibujado del menú, siendo este último vacío. Además, la clase deja sin implementar el método OnButton, por lo que, las clases que lo hereden puede utilizar dicho control para la inicialización de su ventana principal, carga de modelo, etc.

El otro caso de ventana de editor se encarga de facilitar las labores de dibujado de menú, que, por lo general, en todos los editores que existían en uAdventure se limitaba al manejo de un listado con posibilidades de añadir, quitar, seleccionar y manipular los elementos del interior, cambiando el nombre y reordenándolos. Dado que, Unity incluye una clase ReorderableList que permite una gestión de una lista de elementos en su interior, con botones de añadir y eliminar, es más apropiado utilizarla frente a utilizar los controles que se implementaron en la primera versión de uAdventure. Por ello, esta clase se ha llamado ReorderableListEditorWindowExtension. En general, pese a que permite herencia en casi todas sus partes, la clase proporciona métodos abstractos y virtuales a la clase heredera para recibir los eventos que se produzcan en el listado. Estos eventos son OnAdd, OnUpdateList, OnAddOption, OnSelect, OnRemove, OnReorder y OnAddDropdown, siendo este último, el único virtual dado que su implementación por defecto mostrara en un menú contextual, los valores almacenados en Options. Por otro lado, la lista requiere del dibujado de la cabecera, el pie y cada uno de los elementos. Para ello, los métodos DrawHeader, DrawFooter y DrawElement incluyen implementaciones por defecto para dichos eventos que pueden ser sobreescritas ya que los tres métodos son virtuales. Por defecto, dado que la lista pretende simplificar el trabajo, DrawElement incluirá el nombre de los elementos que estén en la variable Elements, permitiendo además editar el elemento seleccionado, que, de ser modificado, llamará al método OnElementNameChanged.

Una vez se estableció la jerarquía, fue posible una encapsulación del código referente a cada editor en su editor correspondiente. Dentro de los editores que existían, los que hicieron uso de la ReorderableList fueron Scenes, Cutscenes, Books, Items, Atrezzos, Characters y

Conversations, siendo Cutscenes el único que hacía uso real de las posibilidades de añadir con diferentes opciones. Por otro lado, los que hicieron uso de DefaultButton fueron Player, Advanced Features y Assestment Profiles. El diagrama de la Fig. 6.11 muestra la nueva arquitectura de EditorWindowBase junto con sus extensiones.

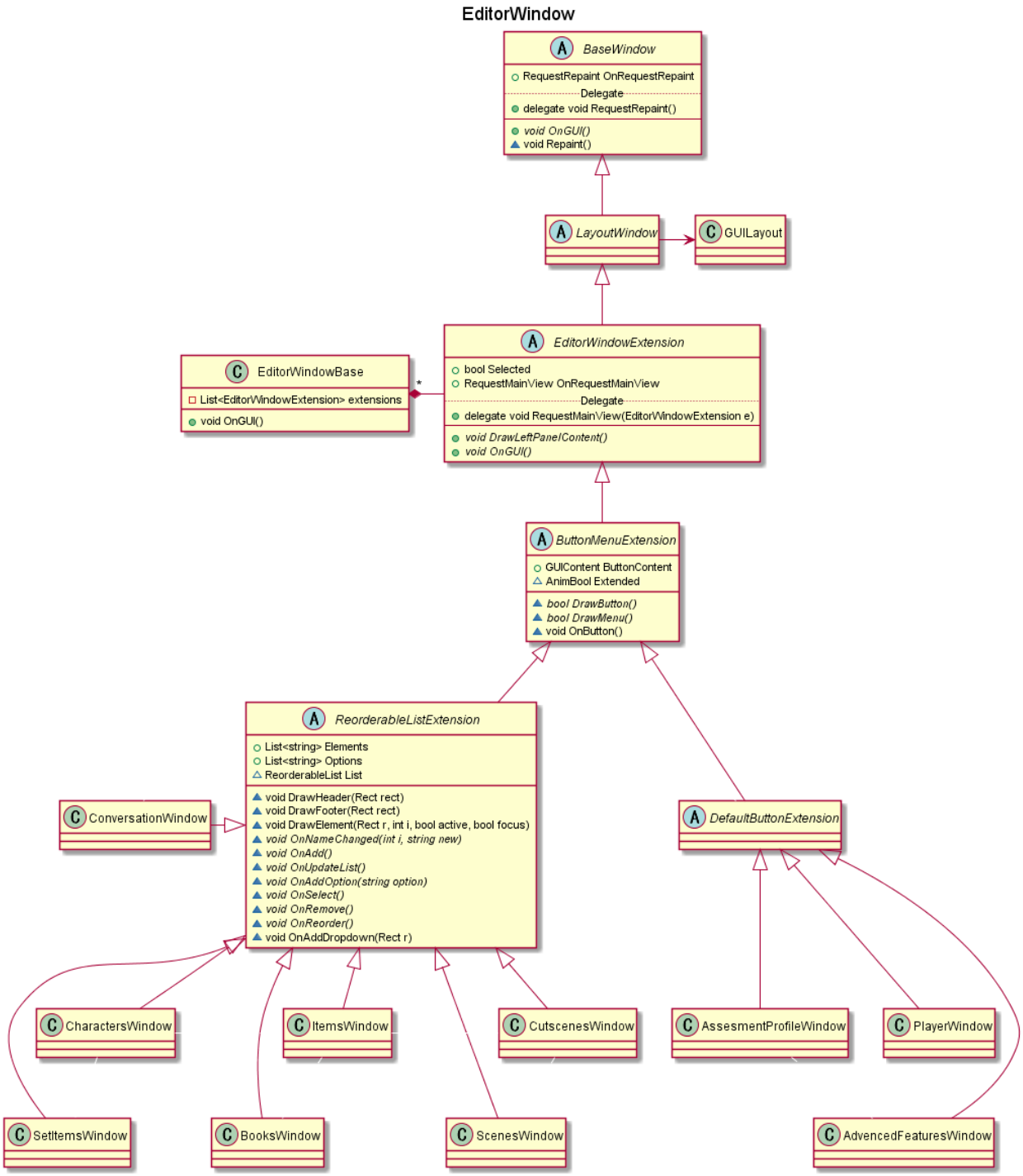

Fig. 6.11 Diagrama de extensiones de EditorWindowBase

Finalmente, puesto todo en común, la ventana principal de editor se encarga de manejar el ciclo de vida de las extensiones, dónde recibe los eventos de OnRequestMainView y marca el editor adecuado como Selected, deseleccionando todos los demás en el proceso.

Además de todas las modificaciones de encapsulación, se modificaron todas las ventanas para poder ser redimensionables, actualizando su rectángulo de dibujado en cada iteración y reimplementando EditorWindowBase para expandirse al máximo tamaño posible.

A continuación, se expone el siguiente paso a la encapsulación, la auto-extensibilidad del editor utilizando atributos de C#.

6.2.3. Extensibilidad del editor principal

En el apartado anterior se expone la creación de extensiones que permiten la extensibilidad en el editor principal. Sin embargo, para el uso de dichas extensiones era necesario incluir al menos una línea en la clase EditorWindowBase, haciendo necesario para nuevos desarrolladores conocer el punto exacto para hacer efectiva la incorporación de su plugin.

Siguiendo la filosofía de Unity que promueve el autodescubrimiento de los editores se implementó un atributo para clases de C# que permite que dicha clase pueda ser identificada por el editor principal como extensión. Este atributo, llamado EditorWindowExtensionAttribute se caracteriza por dos parámetros. El primero es la prioridad del editor, haciendo que los números menores aparezcan más arriba que números mayores. El segundo parámetro es un parámetro flexible en dimensión que permite enumerar las clases del modelo a las que va destinado dicho editor. De esta forma, si fuera necesario sería posible conocer si existe un editor disponible para una clase o, en el caso contrario, notificar que se ha encontrado un elemento que no puede editar ninguna extensión. Además, pueden filtrarse las extensiones por clases que editan. Ninguna de las dos funcionalidades anteriores está disponible actualmente, pero desde el punto de vista del diseño es razonable que las extensiones contengan dicho parámetro de cara al futuro.

Para el uso del atributo se ha evitado que EditorWindowBase tenga que realizar la búsqueda y carga de extensiones encapsulando dicha búsqueda y creación en la clase EditorWindowExtensionFactory. Además, el hecho de que exista una factoría permitiría el cambio de sistema de carga de una manera más sencilla.

EditorWindowExtensionFactory se encarga de la carga de los elementos y creación de instancias de los mismos. Públicamente provee un método GetAvaliableExtensions que buscará e instanciará todas las extensiones posibles y las devolverá ordenadas por prioridad. También se ha

implementado el método GetExtensionFor, que permite encontrar una extensión capaz de manipular el tipo facilitado. Para realizar la búsqueda se utiliza Assembly obtenido a través de AppDomain, filtrando todas las clases utilizando Linq utilizando el método IsDefined que permite detectar si una clase está definida por un atributo. Esta forma de búsqueda se utiliza generalmente en todas las extensiones de uAdventure que utilizan atributos de C#.

Una vez se realiza la búsqueda de todas las clases que son definidas con dicho atributo, se extraen las prioridades y tipos disponibles y se almacenan en una caché sobre la cual se realizarán las búsquedas en los métodos GetAvaliableExtensions y GetExtensionFor.

El diagrama de la Fig. 6.12 muestra el diagrama de clases del diseño propuesto anteriormente. Como puede verse en el diagrama, la clase EditorWindowBase obtiene las extensiones a través de EditorWindowExtensionFactory, que busca todas las clases que tienen EditorWindowExtensionAttribute y genera instancias usando Activator.

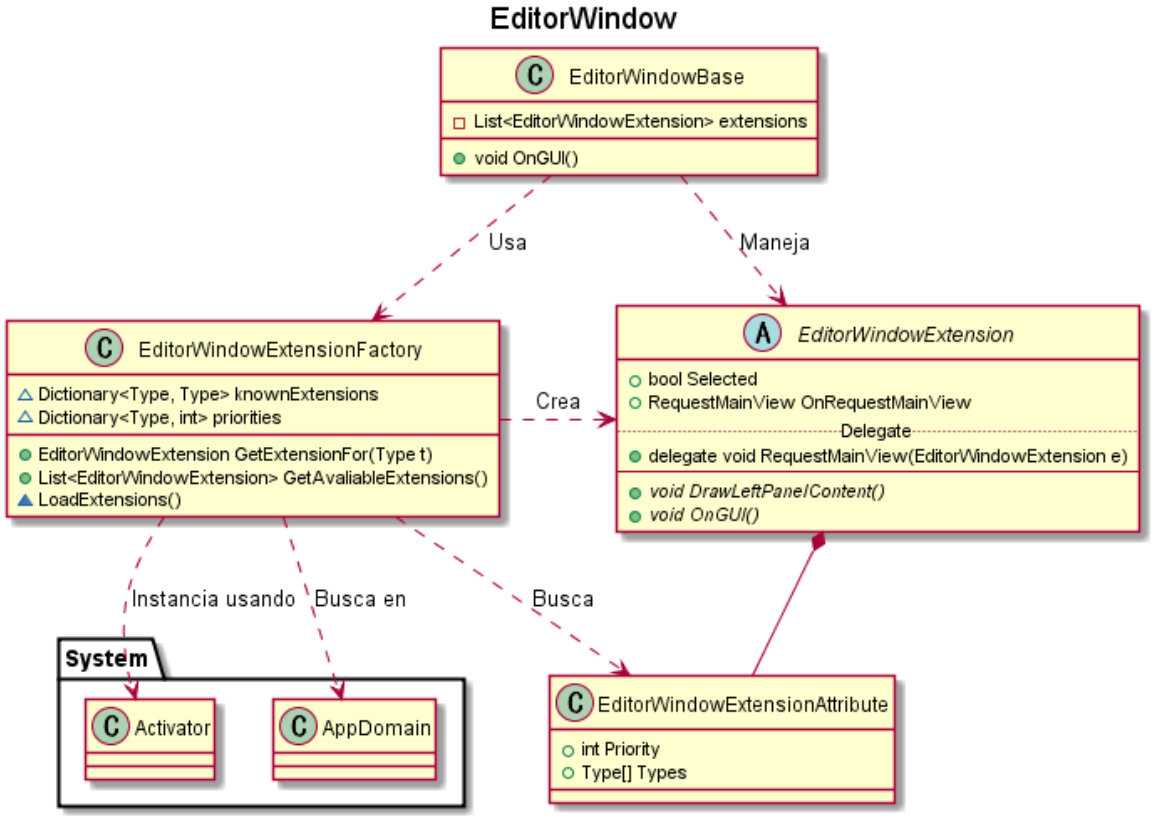

Fig. 6.12 Diagrama de clases de EditorWindowBase con autodescubrimiento de extensiones.

En el siguiente apartado, se expone la extensibilidad en el modelo, necesaria para que las nuevas ventanas del editor puedan almacenar y consultar su modelo.

6.2.4. Extensibilidad del modelo

En este apartado se exponen los cambios que se han hecho al modelo para que soporte extensibilidad en el mismo, pudiendo almacenar y buscar elementos y tipos a través del modelo, manteniendo el orden y permitiendo la persistencia y carga de los elementos de forma dinámica.

En uAdventure el modelo que se plantea es bien conocido. Por ello, el modelo se basa en una estructura de elementos conocidos que mantienen a los elementos del nivel inferior. Esto se debe a que el desarrollo de eAdventure se plantea con un objetivo claro, que es el desarrollo de aventuras gráficas y, en dicho género, el conjunto de elementos que pueden componer el juego es limitado a los elementos que se han modelado en eAdventure. Dado que, uAdventure se plantea en base a eAdventure y ninguno de los dos planeaba ser extendido para poder desarrollar nuevos formatos de juegos como los que se podrán realizar a través de geoposicionamiento, que pueden romper el esquema de la aventura gráfica, es necesario que el modelo sea extensible y flexible para incorporar los nuevos elementos. Esta extensibilidad podría darse en diferentes capas del modelo más profundas, como la capacidad de extender los elementos que puede contener la escena, pero, dado que dicho nivel puede ser mucho más complejo se ha optado por realizar en esta versión una extensibilidad simplificada en la cual sólo se puede extender el modelo a nivel de capítulo. Aunque se ha mantenido la estructura original para evitar tener que hacer modificaciones mayores al controlador, se han añadido dos métodos que permitirán el cambio progresivo del control del modelo hacia el nuevo formato. Si ahora mismo se accede a las escenas utilizando Chapter.getScenes() el nuevo sistema permite utilizar Chapter.getObjects<Scene>(), de una forma mucho más genérica.

En base a esto, tanto Chapter como ChapterDataControl se han extendido para dar soporte a las siguientes funciones genéricas:

- GetObjects<T>(): La función utiliza un template que permite introducir el tipo de los elementos que se espera recibir. Si el modelo no tuviera dichos elementos, se creará una nueva lista que se retornará. La lista retornada puede ser manipulada en el exterior afectando al modelo original por lo que se eliminan los métodos Add, Remove, etc.
- GetObjects(Type t): Similar al anterior, en lugar de utilizar el template recibirá el tipo por parámetro, retornando el mismo tipo de listado.

- GetObjects(): Este listado permite recibir todos los elementos que se encuentren el capítulo, aunque, en este caso, no será posible manipularlo externamente ya que surge de la combinación de todas las listas del interior.
- FindRefference(string id): Este método permite encontrar el objeto que tenga dicho id. Para ello, se busca en aquellos elementos que implementen HasId.

Por su parte ChapterDataControl añade los mismos métodos, permitiendo que las nuevas extensiones puedan almacenar su modelo a través del DataControl. Dado que, el sistema de DataControls está destinado a ser reemplazado con el sistema de AssetDatabase de Unity, no se ha puesto énfasis en los DataControl por ahora.

Para serializar el modelo, en lugar de utilizar los listados individuales y utilizar los *subparsers* se genera el listado de todos los objetos con GetObjects y, a continuación, se serializan todos los objetos de forma individual. Dado que no se conoce el tipo a serializar y, es posible, que el *writer* del Chapter no tenga conocimiento de su existencia, se ha rediseñado el sistema completo de serialización y deserialización de XML para que tenga soporte a autoextensibilidad con el objetivo de permitir la serialización de todos aquellos elementos que se encuentren en el modelo extensible sin la necesidad de conocer el tipo del elemento que se pretende serializar. El nuevo sistema es expuesto en el siguiente subapartado.

6.2.5. Nuevos sistemas de serialización y deserialización

En el apartado anterior se ha introducido a la extensibilidad del modelo. Dado que, un modelo extensible debe poder ser persistido y cargado surge la necesidad de poder serializar los elementos del modelo sin conocer qué clase se encarga de escribirlo, y deserializar contenidos en el XML sin necesidad de conocer qué clase es capaz de transformar los nodos de XML en objetos del modelo. Dado que el sistema debe ser retro-compatible, es recomendable evitar utilizar sistemas de serialización automática pues no siempre se comportarán generando la estructura que el antiguo modelo espera ni deserializar correctamente, ya que la estructura es diferente. Sin embargo, el nuevo sistema puede permitir serialización híbrida, en la que los antiguos DOMWriters y SubParsers se encarguen de partes del modelo antiguo y el resto de elementos puedan ser serializados a través de otros sistemas automatizados.

Para serializar un objeto de manera genérica se ha propuesto una arquitectura en la que un objeto intermediario llamado DOMWriterUtility se encarga de serializar un objeto cualquiera,

permitiendo liberar de dicha labor a la clase interesada en serializar el objeto. Dado que la serialización debe ser extensible, el objeto intermediario utiliza clases secundarias especializadas en la serialización de los objetos de tipos concretos. Estas clases secundarias, llamadas DOMWriters implementan la interfaz IDOMWriter que requiere el método DOMWrite el cual recibe el nodo padre al que se adherirán los hijos y el objeto a serializar. Además, el método DOMWrite puede recibir una serie de parámetros que permiten que el serializador se comporte de diferentes formas. Estos parámetros, que implementan la clase IDOMWriterParameter, pueden ser de cuatro tipos: DontCreateNodeParameter, que solicita al DOMWriter que no cree nodo, sino que incorpore la información al nodo padre, NodeNameParameter, que permite que el serializador utilice el nombre del parámetro para su nuevo nodo, ChapterTargetParameter, que permite saber al serializador cuál es la escena que se debe ejecutar primero en el Chapter y InvalidParameter, que permite avisar al serializador de que la aventura es inválida. Los dos primeros parámetros van destinados al serializador actual, pero los dos últimos se propagan a través de toda la serialización. Para simplificar el uso de los dos primeros parámetros, la clase ParametrizedDOMWriter (que hereda de IDOMWriter) automatiza el uso de dichos parámetros ofreciendo dos métodos abstractos, DefaultName, para saber cuál sería el nombre por defecto de no recibir el parámetro NodeNameParameter y FillNode, cuyo objetivo pretende rellenar el nodo que se envíe, pudiendo decidir si crear el nodo o enviar el nodo padre si se recibe el método DontCreateNodeParameter.

Además de conectar con las clases especialistas, DOMWriterUtility permite una serialización flexible de listas y de diccionarios. Para serializar las listas, la recorrerá y adjuntará todos los nuevos nodos al nodo padre. Para los diccionarios, creará nodos intermedios cuyo nombre se basará en la clave del diccionario y en su interior, serializará el contenido del objeto.

Para conectar con las clases especialistas y otorgar una mayor flexibilidad, DOMWriterUtility se nutre de atributos de C#. Por ello, todas las clases que quieran formar parte del sistema de forma automática deben utilizar el atributo DOMWriter, el cual recibirá además los tipos de objetos para los cuales el serializador está preparado. Esto sigue la misma línea de la filosofía Unity en la que los elementos pueden ser descubiertos automáticamente a través de sus atributos. Para ello, DOMWriterUtility se nutre de Assembly y busca el él las clases que incluyen dicho atributo. Tras ello, guarda en su interior una caché con todos los serializadores conocidos. Una vez se solicita que DOMWriterUtility serialice un objeto, buscará en la caché y

utilizará el serializador correspondiente de existir. La Fig. 6.13 muestra la arquitectura de DOMWriterUtility que se ha expuesto.

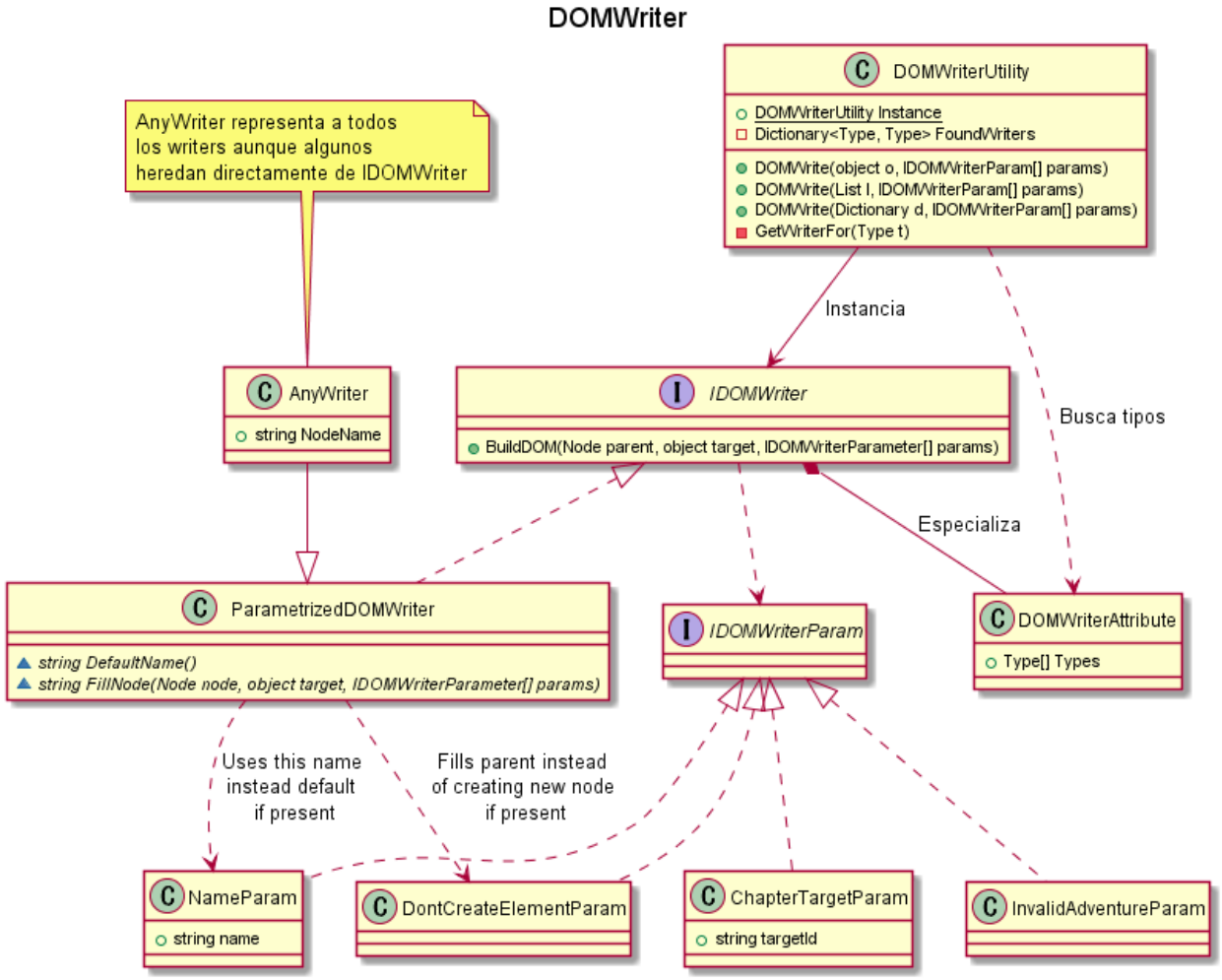

Fig. 6.13 Diagrama de clases de DOMWriter

El proceso de transición fue complejo dado que fue necesario que modificar todos los serializadores antiguos para adaptarse al nuevo sistema de serialización. Los DOMWriter antiguos en lugar de añadir los nodos que creaban al nodo padre, creaban un nuevo nodo y lo retornaban al terminar. Este sistema, aunque válido, no era compatible ya que existen casos en los que la serialización de un único objeto puede generar múltiples nodos, como, por ejemplo, un efecto. En la mayoría de los casos, el cambio ha sido sencillo y siguiendo un patrón general. Sin embargo, todos los DOMWriters se han adaptado para que, cuando haya elementos por debajo de la jerarquía del objeto actual (como condiciones o efectos), éstos se serialicen utilizando también

DOMWriterUtility. De esta forma, para serializar un objeto ya nunca es necesario conocer qué DOMWriter es necesario para dicha labor, eliminando la fuerte cohesión existente entre los serializadores y potenciando la futura extensibilidad de todos los elementos sin tener que modificar sus serializadores.

Los serializadores que se han modificado para formar parte del sistema de DOMWriterUtility son todos excepto AdaptationDOMWriter, AssestmentDOMWriter, ExpectedGameIODOMWriter y ResourcesDOMWriter, haciendo un total de 15 clases adaptadas al nuevo sistema.

Una vez completado el sistema serializador, el siguiente paso es realizar el proceso contrario y construir un sistema deserializador. Dado que ambos se asientan en los mismos principios de encapsulación, flexibilidad y extensibilidad, ambos tienen arquitecturas muy similares a alto nivel. Sin embargo, cuentan con diferencias significativas.

En el caso del serializador, su objetivo es reemplazar a los SubParser. Éstos, reciben el nodo a deserializar y retornan el objeto ya deserializado. Para realizar esta labor, la clase que desea obtener el objeto anteriormente debía conocer el SubParser adecuado encargado de dicha serialización. En la nueva arquitectura, solo será necesario enviar el nodo a DOMParserUtility, que retornará el objeto apropiado según el nodo proporcionado. Además de este funcionamiento, si la clase padre estuviera esperando un tipo concreto, podría solicitar a DOMParserUtility que buscara un deserializador adecuado a dicho tipo. Por otro lado, si se enviara una lista de nodos (XMLNodeList), DOMParserUtility retornaría una lista nueva con todos los objetos que haya encontrado en los diferentes nodos del interior pudiendo solicitarse también que éstos pertenezcan a un tipo concreto. Dado que, este proceso debe ser genérico, se utilizan clases especializadas para el deserializado de los elementos en las que se delega la deserialización. Estas clases deserializadoras deben implementar la clase IDOMParser, que reemplaza a los antiguos SubParser. Para ello, IDOMParser incorpora un método DOMParse que recibe un XMLElement con el nodo a deserializar y retorna un *object* con el objeto deserializado. Además, el método puede recibir parámetros como un *array* de *object*. Esos parámetros pueden utilizarse para cualquier tarea y deben propagarse si se hacen deserializados anidados. Dado que, el capítulo (Chapter) se utiliza en varios DOMParsers como los efectos y las condiciones, para el control de las referencias a variables y *flags*, el capítulo es uno de los elementos que se transfieren a través de estos parámetros.

Dado que DOMWriterUtility debe ser extensible se ha realizado una arquitectura muy similar a la de DOMParserUtility, en la que se utiliza la etiqueta DOMParser para identificar a los potenciales deserializadores. Sin embargo, la etiqueta DOMParser es significativamente diferente pues en su interior debe almacenar, por un lado y al igual que DOMWriter, los tipos que es capaz de manejar y, por otro lado, y mucho más importante, los nombres de los nodos que es capaz de manejar. De esta forma, si DOMParserUtility no recibe un tipo esperado, utilizará el nombre del nodo para encontrar el deserializador adecuado al nodo. Este procedimiento es el más general, ya que permite un uso autoextensible y se utiliza, por ejemplo, en la deserialización de los elementos del nodo del capítulo.

Para realizar el auto descubrimiento de las clases que participan en la deserialización automatizada se utiliza Assembly, buscando las clases que incluyen el atributo DOMWriter y añadiendo a dos cachés diferentes, una que organiza por nombres de nodos y otra por tipos, los diferentes deserializadores encontrados. El diagrama de la Fig. 6.14 muestra la arquitectura de DOMWriter expuesta.

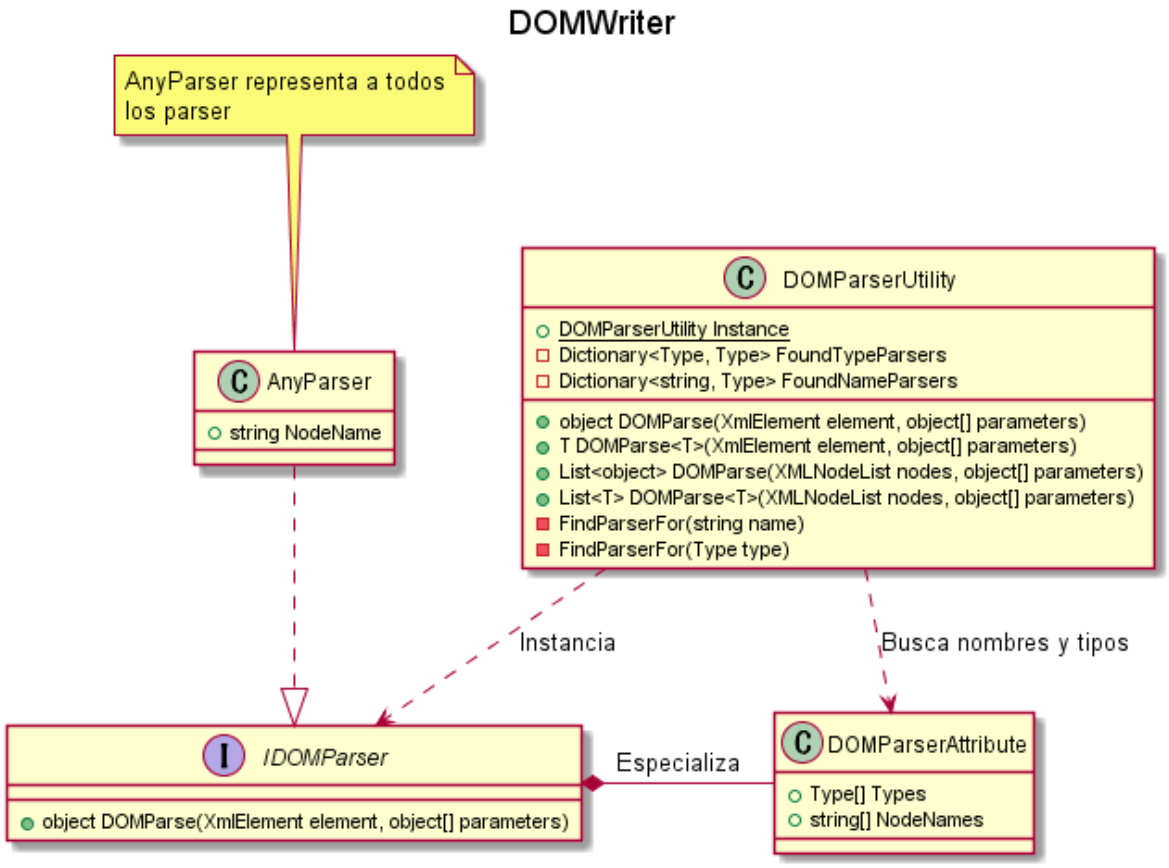

Fig. 6.14 Diagrama de clases de DOMParser

Para la adaptación al sistema de DOMParsers, dado que los SubParser pueden comportarse de maneras muy diversas, en la mayoría de los casos se han realizado clases adaptadoras al nuevo sistema, que encapsulan el SubParser en lugar de tener que modificar el funcionamiento original de la clase. Para ello, se han creado 15 nuevos DOMParsers, quedando fuera de la adaptación AdaptationSubParser, AssestmentSubParser, ExpectedGameIOSubParser y ResourcesSubParser.

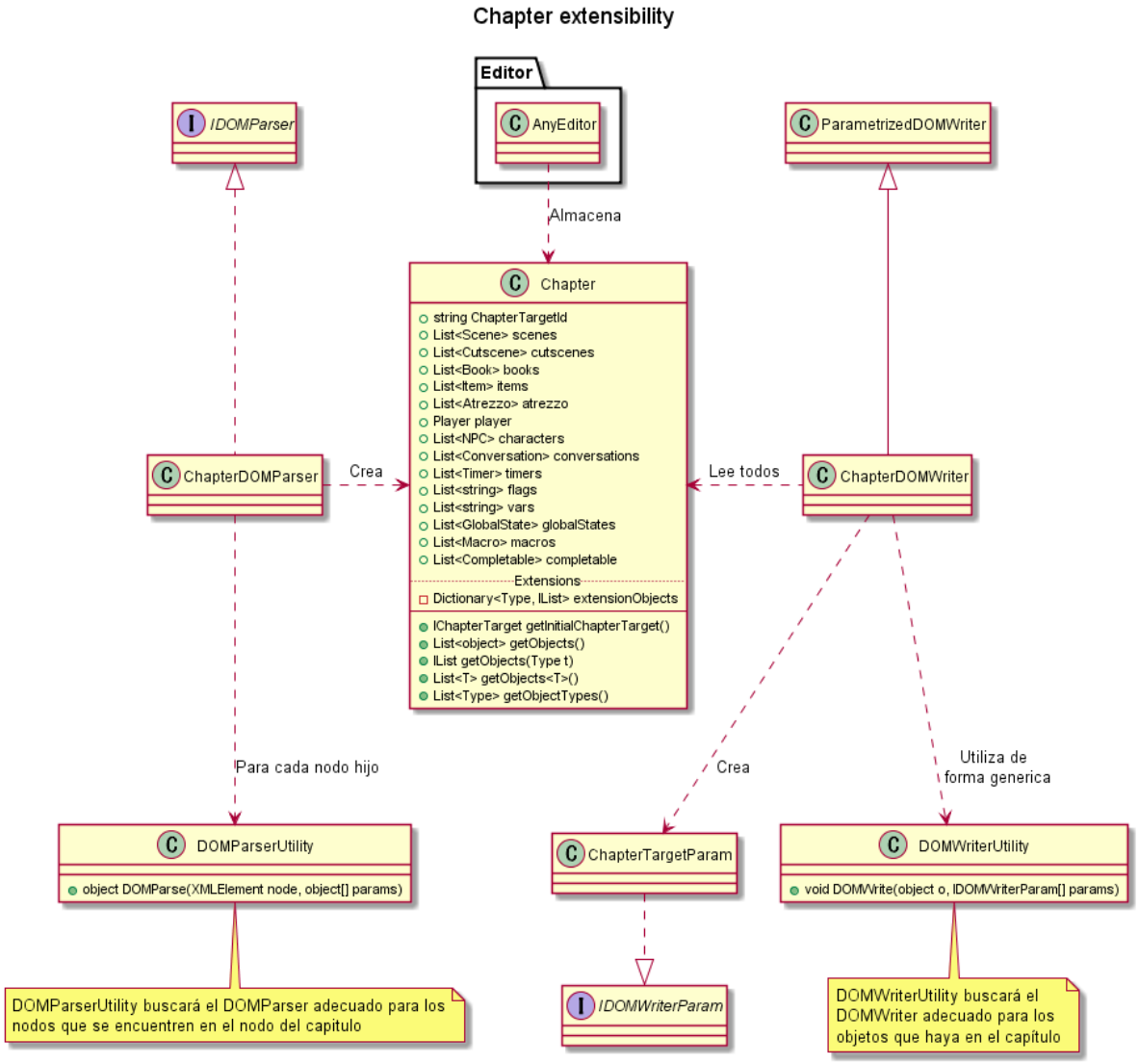

Fig. 6.15 Diagrama de clases de la extensibilidad de Chapter conectada con sus parsers.

El nuevo sistema de deserialización y serialización se planteó con la necesidad de permitir una serialización y deserialización genérica del modelo extensible en chapter. Para cerrar el subcapítulo de la extensibilidad de chapter, el diagrama de la Fig. 6.15 muestra el

diagrama de clases que representa el funcionamiento actual de la serialización y deserialización de Chapter, así como de la funcionalidad que se le incorpora. Como puede observarse, Chapter mantiene todas las listas antiguas dado que el sistema de DataControl se nutre de ellas para su funcionamiento y, dicho sistema, no será revisado hasta futuras versiones, donde se traslade su funcionalidad al sistema de manejo de assets de Unity. Además de esto, el diagrama muestra los métodos genéricos de acceso al modelo que utilizan los nuevos serializadores y deserializadores de chapter para su funcionamiento y que han permitido una reducción drástica del código.

Como pequeña conclusión al cambio del sistema, el nuevo serializador para chapter ha pasado de ocupar 151 a tan sólo 108. En el caso del deserializador, el cambio es más fuerte todavía, donde se ha pasado de 238 a 89. Esta reducción de líneas demuestra que la arquitectura es más potente y eficiente de cara al programador, destacando una programación más genérica evitando la repetición de código para tareas similares.

6.2.6. Extensibilidad en la ejecución principal

Hasta ahora se han expuesto las posibilidades de extender el modelo utilizando como contenedor el capítulo. Gracias a los editores personalizados es posible añadir elementos al capítulo y manipularlos y, sumado al serializador genérico es posible que éstos sean salvados y cargados de memoria persistente. Sin embargo, actualmente no existe forma de lograr que los nuevos elementos del modelo formen parte del flujo de ejecución del programa dado que, en el modelo actual, las escenas de diferentes tipos son las únicas que reciben dicho flujo y, en su interior, solo es posible almacenar los elementos del modelo antiguo.

Para ello, a continuación, se analizan las diferentes formas en las que una escena se convierte en el principal elemento de la ejecución pues serán los puntos donde se deberá soportar que nuevos elementos puedan ser asignados también. Para que una escena se convierta en el elemento principal pueden ocurrir tres cosas. La primera, es que ésta sea el objetivo (*target*) del capítulo y, por ello, sea el objeto que se utilice como iniciador del juego. La segunda es que, en alguna de las salidas (*exits*) de una escena se encuentre referenciada una nueva escena. El jugador, al interactuar con la salida iniciará la transición hacia la nueva escena. La tercera y última es que se ejecute algun efecto de cambio de escena que iniciará la transición hacia la misma.

En común, todas ellas destacan por que no requieren nada más que un nombre o id para el almacenamiento y posterior recuperación del elemento. Sin embargo, en dicho lugar, desde

cualquier editor, sólo es posible almacenar *ids* de escenas. Por ello, se ha creado el concepto superior de IChapterTarget, que es una interfaz que deben cumplir todos los objetos que pretendan ser utilizados como elemento principal de la escena. Para su uso, GeneralScene, que es la clase padre de todas las escenas existentes, hereda de IChapterTarget y en todos los editores en los que se recuperaba la lista de *ids* de escenas para proporcionar como opciones al usuario, se han remplazado los *ids* por los obtenidos en base a IChapterTargets. Más adelante, las nuevas escenas de tipo MapScene que se implementarán también utilizarán dicha interfaz para poder convertirse en el elemento principal de la escena.

Por otra parte, en la capa de ejecución (Runner), para poder centrar la ejecución en los nuevos modelos es necesario rediseñar el funcionamiento de la clase que hace de intermediario entre el modelo y la ejecución principal. Dicha clase es la clase Game, que se sitúa entre la carga del modelo y las escenas en ejecución, controlando el flujo de ejecución e interacción y convirtiendo los elementos principales del modelo en elementos en ejecución. En este proceso, la clase Game busca la escena apropiada, la instancia y almacena su referencia. Sin embargo, dado que ahora puede que el objetivo (*target*) no sea una escena, será necesario utilizar una nueva interfaz común a todos los objetivos IRunnerChapterTarget que se analizará a continuación

En el modelo anterior, Game se comunica con todas las escenas mediante diversos mensajes básicos ubicados en SceneMB. El primer evento es el envío del elemento del modelo que el elemento en ejecución va a utilizar. Tras ello, Unity iniciará el objeto y en ese momento se transformará el modelo en elementos en ejecución. El segundo evento se utiliza cuando se producen cambios a nivel global. Los cambios de flags y variables pueden modificar los contenidos de la escena y, por ello, dichos cambios deben notificarse a la escena para que esta pueda para refrescar su contenido. Dicho evento, es llamado RenderScene, que, aunque implica la renderización completa de la escena se utiliza con el propósito anteriormente mencionado. En tercer lugar, las escenas reciben la interacción del usuario desde la clase Game que hace de intermediario para poder cortar el flujo de interacción durante la ejecución de efectos y menús. Para ello, la escena en ejecución hereda de la interfaz Interactuable, que la provee de los mensajes apropiados para dichos eventos. Por último, cuando se hacen transiciones entre escenas, el método Destroy juega un papel fundamental dado que permite que la escena se destruya tras un tiempo, permitiendo así realizar la carga de la nueva escena, procesar la transición y finalmente permitir que la escena se destruya una vez completada.

El objetivo del nuevo modelo para la ejecución es permitir materializar otros elementos del modelo, además de las escenas, que den cabida a nuevas mecánicas incompatibles con las escenas anteriores. Si bien más adelante se hablará del caso de introducir escenas geoposicionadas que utilicen un mapa como base, la arquitectura debe ser flexible como para que se permitan introducir todo tipo de mecánicas, dando pie a la creación de cualquier tipo de mecánica o minijuego. Por ejemplo, se puede crear un minijuego tipo Super Mario y que este se ejecute en un determinado punto del juego y, al terminar, salte la ejecución a la ejecución clásica de uAdventure.

Al analizar la interfaz que utiliza la clase Game con las diferentes escenas, se puede observar claramente que no existe ninguna limitación en cuanto a la forma que tendrá la escena y, por otro lado, la comunicación es suficientemente flexible como para permitir cualquier tipo de juego. En el caso de las escenas actuales, SceneMB tiene control absoluto sobre lo que renderiza y sus características, así como lo que desea hacer con la interacción del usuario. Sin embargo, una de las potenciales limitaciones es el flujo de interacción del usuario, pues, Game sólo utiliza la interfaz Interactuable a través del botón izquierdo del ratón y esto puede resultar escaso para algunos tipos de juego en los que se utilicen otras fuentes como interacción. A pesar de ello, debido a la forma de funcionar de Unity, la interacción puede interpretarse desde cualquier elemento de forma libre y, dado que Game permite que conocer si hay un elemento ocupando la interacción con el método isSomethingRunning y que, además, cualquier elemento Interactuable tome control de la interacción a través de Execute, es posible adaptar el funcionamiento de Game a cualquier tipo de mecánica. Por ejemplo, si un elemento utilizara la tecla “a” para su interacción en una determinada mecánica, éste al llamaría a Execute y tomaría control de la interacción hasta que, decidiera liberarla en su método Interacted.

Según éste análisis, entonces, solo será necesario introducir en IRunnerChapterTarget los eventos analizados StateUpdated (anteriormente llamado RenderScene), Interacted (proveniente de heredar la clase Interactuable), Destroy y un atributo Data donde establecer el elemento del modelo que se pretende crear. Además de la interfaz, es necesario proporcionar un sistema que permita que Game pueda instanciar el elemento apropiado con el que se comunicará en base al elemento del modelo que se haya seleccionado como objetivo. Este elemento IChapterTargetFactory se encargará de la transformación de los elementos IChapterTarget, residente en el modelo, en un IRunnerChapterTarget, residente en ejecución. Para ello, contiene

un método Instantiate que recibe como parámetro el IChapterTarget a materializar. Se consideran factorías dado que crean elementos en ejecución en base a una configuración de entrada. Sin embargo, pese a tener una factoría individual para cada target que se encargue de ponerlo en marcha, es necesario un nivel por encima que permita el autodescubrimiento y selección de la factoría apropiada para dicha labor.

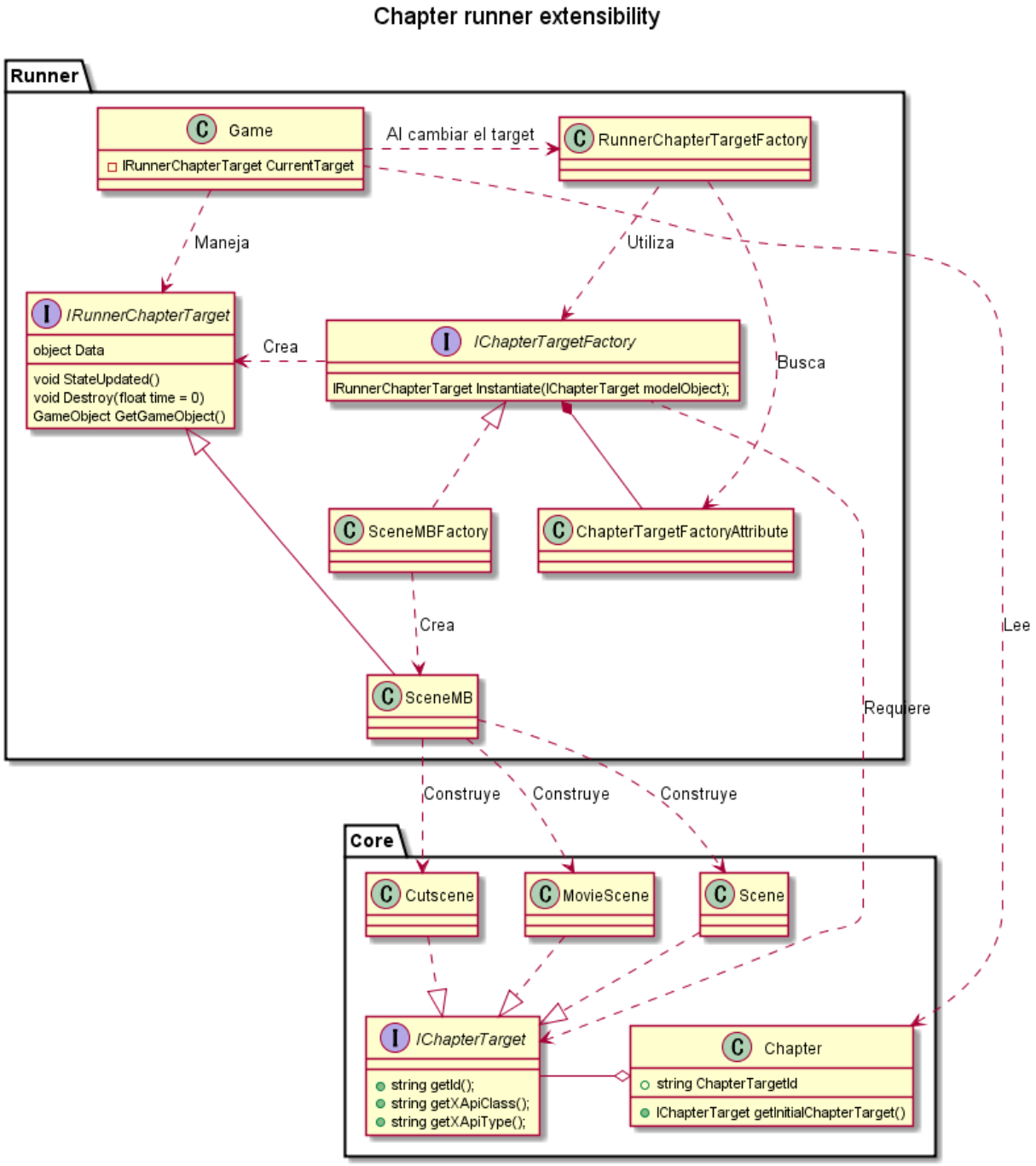

Fig. 6.16 Diagrama de clases del nuevo sistema de ChapterTargets

Esta clase se llama RunnerChapterTargetFactory y, de manera similar a las factorías/clases con autodescubrimiento anteriores utiliza el atributo de C# ChapterTargetFactoryAttribute para la búsqueda de las factorías especializadas. En su interior, ChapterTargetFactoryAttribute almacena el tipo del IChapterTarget que es capaz de convertir en IRunnerChapterTarget y, dicho atributo, será utilizado por RunnerChapterTargetFactory para realizar la búsqueda de la factoría apropiada para el target. El diagrama de la Fig. 6.16 muestra la arquitectura expuesta anteriormente. Como puede verse, se han añadido las clases expuestas anteriormente e, implementando las interfaces anteriores, en el lado del modelo se encuentran las posibles Scenes que heredan de IChapterTarget, y, en el lado de la ejecución, SceneMB implementa ahora IRunnerChapterTarget que es instanciada a través de la nueva clase SceneMBFactory completando el hueco entre Game y SceneMB.

Más adelante, en el capítulo en el que se introducen las MapScenes, se ampliará este diagrama para incluir los elementos que conecten a Game con este nuevo tipo de escenas.

En el próximo apartado se expone extensibilidad en efectos, el sistema de programación integrado en uAdventure.

6.2.7. Extensibilidad al sistema de efectos

Si bien uAdventure provee de muchas facilidades para las mecánicas de juego más habituales en los juegos de tipo aventura gráfica, para poder dar flexibilidad en los juegos pudiendo generar comportamientos personalizados y controlar el estado global del juego, uAdventure provee de un sistema de programación visual que se basa en efectos. Los efectos son instrucciones que realizan tareas complejas de alto nivel, que se configuran en tiempo de edición y se materializan en ejecución, cuando se lance el efecto. Los efectos pueden lanzarse desde muchos puntos, tales como carga de escenas, salidas, interacción con objetos o personajes o incluso durante las conversaciones.

Dado que un efecto, por su naturaleza, se comporta de manera genérica siguiendo un patrón similar al patrón de diseño *command*, es el punto más apropiado para que un desarrollador de *plugins* para uAdventure pueda introducir controles para sus mecánicas. Por ejemplo, más adelante se expondrán las mecánicas que se han introducido para los *plugins* de geo-posicionamiento y códigos QR, siendo algunas de ellas, control de la navegación o el lanzamiento de escenas limitadas a un área GPS. Si bien el modelo es lo suficientemente generalista como para poder ser extensible, en la práctica, la clase EffectHolder es la única capaz

de ejecutar efectos y, para ello, utiliza un enorme *switch* que diferencia el tipo del efecto. Esto también ocurre tanto en la serialización como en la deserialización donde existe un enorme *switch* en ambos casos. Sin embargo, en la capa de edición, dado que se parte de un editor de secuencias personalizado adaptado del proyecto de la UCM IsoUnity [51] realizado por los hermanos Iván José Pérez Colado (autor de uAdventure Runner) y Víctor Manuel Pérez Colado, cuenta con autodescubrimiento de editores apropiados para nuevos tipos de efectos, siempre que éstos hereden de la clase EffectEditor. Debido a ello, esta parte no será necesario extenderla pues es lo suficientemente extensible en el estado actual como para poder introducir nuevos editores de efectos.

El diseño de los efectos parte de una clase genérica AbstractEffect de la cual heredan todos los posibles efectos. En AbstractEffect sólo se almacena el tipo del efecto como número entero, siendo éste un elemento heredado de la arquitectura antigua de eAdventure, mientras que, en las clases herederas de AbstractEffect se almacenarán todos los contenidos de la configuración del efecto. Por otro lado, en su ejecución, EffectHolder diferenciará entre los efectos por su tipo y realizará la acción apropiada.

Siguiendo este diseño, pero utilizando un modelo un poco más genérico se introduce un nuevo tipo "*custom effect*" para todos los nuevos efectos que serán tratados a través del nuevo sistema de ejecución de efectos. Para que entren en el nuevo sistema, todos los nuevos efectos deberán utilizar el nuevo tipo.

Aprovechando los antiguos switch para detectar el caso de "*custom effect*" se introduce en el diseño antiguo los nuevos elementos de extensibilidad. Estos puntos son, la conexión de los efectos con la clase DOMParserUtility y DOMWriterUtility para la serialización y deserialización automatizada de los mismos y, por otro lado, la ejecución personalizada de los efectos dentro de EffectHolder. El diagrama de clases de la Fig. 6.17 muestra las conexiones con ambos elementos para la serialización y deserialización de efectos personalizados.

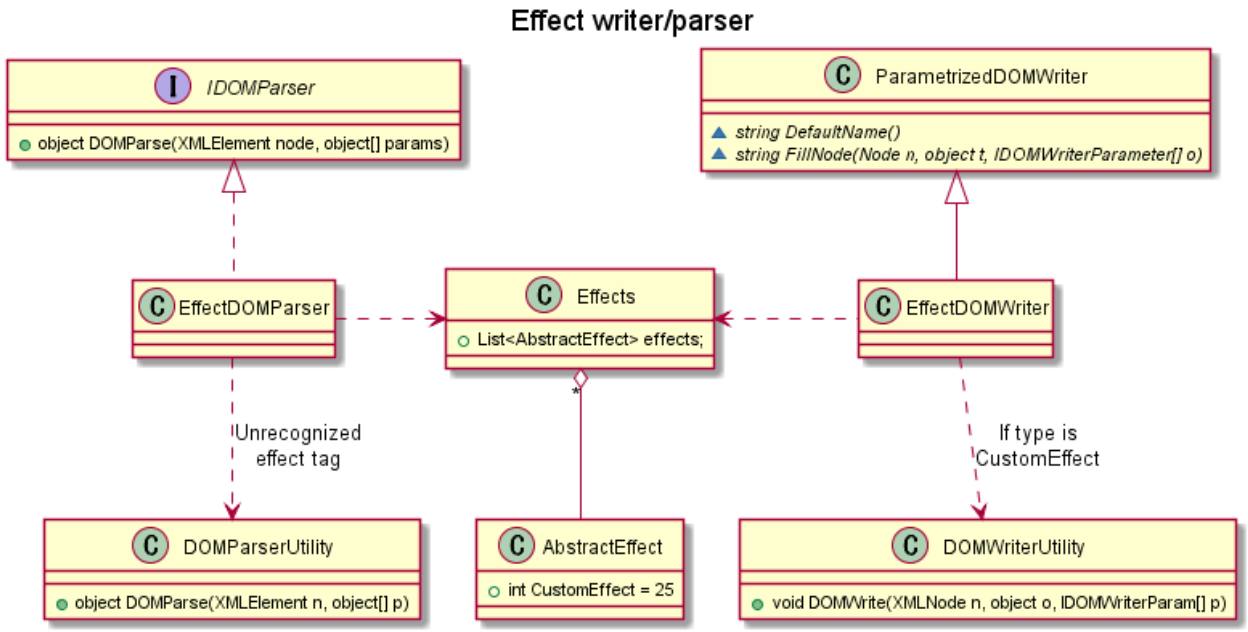

Fig. 6.17 Diagrama de clases de la serialización y deserialización de efectos.

Para este último punto, es necesario introducir un nuevo elemento en la arquitectura que permita al desarrollador establecer un ejecutor siguiendo el patrón command para su efecto personalizado. Esta clase ejecutora es llamada CustomEffectRunner e implementa la interfaz Sequence, al igual que el propio EffectHolder, para seguir la línea de ejecución del efecto. Dado que el CustomEffectRunner necesita el efecto para conocer la configuración que hay en su interior, el atributo Effect permitirá establecer dicho parámetro. Sin embargo, EffectHolder no conoce todos los efectos personalizados que existen ni los ejecutores asociados a ellos y no sería conveniente que EffectHolder tuviera referencias a ellos ya que derivaría en problemas de encapsulación. Para evitarlo, y siguiendo la línea del desarrollo anterior, se utiliza autodescubrimiento basado en atributos de C# a través de una clase intermediaria. Esta clase, llamada CustomEffectRunnerFactory, analiza el código en busca del atributo CustomEffectRunnerAttribute para asociar las clases a los tipos que ejecutan. Para ello, el atributo almacena los tipos de los efectos que es capaz de ejecutar. Una vez se detecta un efecto de tipo "*custom effect*" se utiliza CustomEffectRunnerFactory para crear un ejecutor apropiado que se encargue de poner en marcha el efecto. El diagrama de clases de la figura ER muestra la arquitectura expuesta para la ejecución de efectos personalizados.

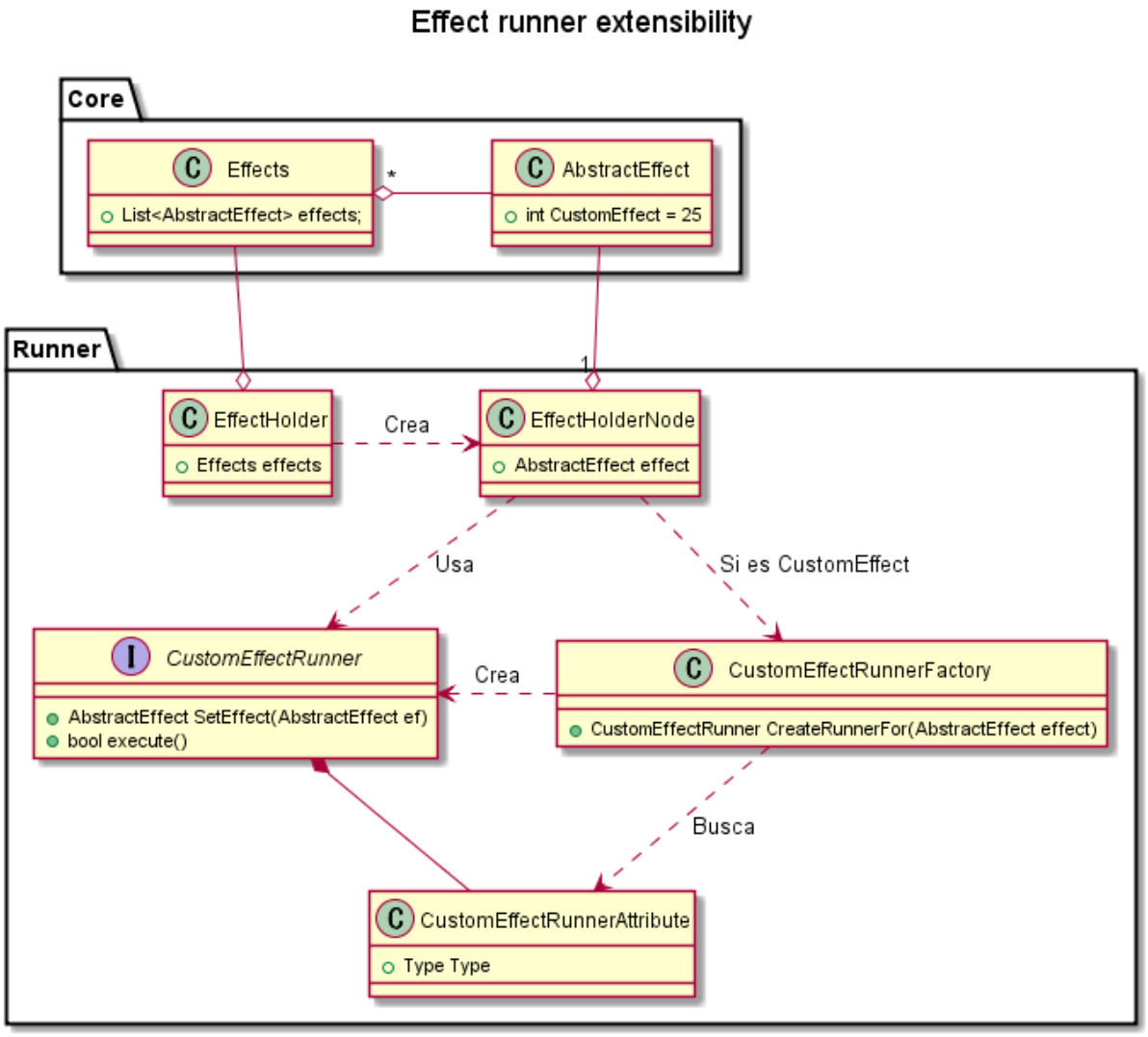

Fig. 6.18 Diagrama de clases de CustomEffectRunner.

6.3. Extensiones de uAdventure

En este apartado se expone el uso directo que se ha dado a todas las funcionalidades de extensibilidad introducidas en el apartado anterior, relacionando los elementos independientes desarrollados en el apartado 6.1 con uAdventure a través de los métodos propuestos en 6.2. Con ello se ponen en práctica todos los formatos propuestos de extensibilidad generando paquetes independientes a modo de *plugins*. Primero se exponen las extensiones para geoposicionamiento en el subapartado 6.3.1 y a continuación se exponen las extensiones de códigos QR en el apartado 6.3.2.

6.3.1. Extensiones de Geoposicionamiento

En este subapartado se exponen los diferentes *plugins* de uAdventure de geoposicionamiento que utilizan tanto el mapa desarrollado GUIMap para el editor de Unity como el mapa modificado de MapzenGO para la ejecución.

El modelo capaz de relacionar los elementos de los mapas con el modelo de uAdventure establece la clase MapScene en la que se almacenan todas las propiedades del mapa, así como sus contendidos, que heredan de la clase MapElement. MapScene, dado que además puede convertirse en el elemento principal de la ejecución, hereda de IChapterTarget. Dentro de los objetivos del proyecto, se debe poder establecer elementos geoposicionados definidos por zonas, así como la inclusión en ellos de las nuevas acciones relacionadas con geoposicionamiento. Para ello, se establece la clase GeoElement que almacena la geometría GMLGeometry (que es compatible con GUIMap) y las posibles acciones geoposicionadas, que se definirán a través de las clases GeoActions. Para cubrir las acciones planteadas se han creado cuatro clases que heredan de GeoAction. EnterAction define la acción de entrar en la influencia del objeto y cuenta con un atributo que permite diferenciar el caso en el que ya se esté dentro al crear el elemento. ExitAction define la acción de salir de la influencia y es opuesta a EnterAction. LookToAction permite controlar la dirección en la que mira el jugador, pudiendo establecer en sus atributos que no sea necesario estar en la influencia del objeto y que la dirección en la que se mira sea siempre el centro del objeto. Por último, la acción InspectAction para la interacción directa del usuario tiene un atributo que permite limitar la interacción al rango de influencia del objeto. Todas las acciones, contendrán efectos (Effects) y condiciones (Conditions) que permiten configurar las consecuencias de la acción y las condiciones en las que se puede realizar.

Para relacionar los elementos del modelo con la MapScene se utiliza la clase MapElement, destinada a almacenar una referencia junto con una serie de atributos de contexto. Existen dos tipos de clase que heredan de MapElement, diferentes en función del contexto que almacenan, que son GeoReference para GeoElements y ExtElemReference para elementos de uAdventure. Los elementos geoposicionados ya pertenecen al mapa y por ello su contexto es definido automáticamente por su GMLGeometry, mientras que, para los elementos de uAdventure, si es necesario un contexto avanzado que permite posicionarlos en la escena de mapa.

En las escenas nativas de uAdventure el contexto que utilizan las referencias a elementos almacenan la posición y la capa del elemento. Por el contrario, en las escenas geoposicionadas es necesario un modelo más complejo que cubra los tres tipos posibles de posicionamientos (global, radial y en pantalla). Por ello, los ExtElemReferences mantienen parámetros para los diferentes contextos de forma flexible utilizando un diccionario de *string* a modo de contenedor genérico que almacena *objects*. Para poder utilizar el mismo sistema de información para la representación de forma inconexa con el tipo de mapa en el que se representará y, además, generalizar la edición, almacenamiento y posterior recuperación de los parámetros, se utilizan descriptores que, de manera limpia, encapsulada y extensible, mantienen una meta descripción de las variables de contexto requeridas para cada tipo de posicionamiento. En esta clase intermediaria llamada Descriptor se mantiene un conjunto de parámetros definidos con la clase ParameterDescription que almacena el tipo y propiedades del parámetro asociado a la variable de contexto, pudiendo establecer valores por defecto, máximos y mínimos. Además de los parámetros, el descriptor proporciona un nombre para el editor para el tipo de posicionamiento. Existen tres tipos de descriptores: WorldPositionedDesc, ScreenPositionedDesc, y RadialPositionedDesc.

El primer descriptor, WorldPositionedDescriptor, que va destinado a describir los parámetros de WorldPositionManager, tiene cinco parámetros. Estos son: Position, que almacena las coordenadas del centro del objeto, Scale, que almacena la escala, Rotation, que permite rotar el objeto, InteractionRange que permite establecer la distancia en metros desde la cual el jugador podrá interactuar con el elemento y, en cooperación con esta, RevealOnRange que permite al usuario establecer que el objeto sólo aparezca si el jugador está en el rango de interacción. Para ello, se utilizará una animación similar a la que realizan los Pokémon en el juego Pokémon GO, imitando dicha mecánica en la que se insta al jugador a deducir en base al entorno la posible ubicación del elemento. El segundo descriptor, ScreenPositionedDescriptor, que va destinado a describir los parámetros de ScreenPositionManager, tiene tres parámetros. En este caso, solamente son necesarias las coordenadas x e y de la pantalla, la escala bidimensional del elemento y la rotación en el eje Z. El tercer y último descriptor, RadialPositionedDescriptor, que va destinado a describir los parámetros de RadialPositionManager, tiene cuatro parámetros. Estos parámetros son, el radio y la distancia necesarios para situar el objeto y la rotación y escala del mismo. Más adelante, en las Fig. 6.21 y Fig. 6.25 aparece el uso del modelo expuesto para los ámbitos de edición y ejecución.

Editor de GeoElements

Para la manipulación de los GeoElements se introduce una nueva extensión del editor que controlará tanto su geometría e influencia como las descripciones y acciones. La extensión GeoElementWindow se conecta con el editor base de uAdventure y hereda del tipo de editor ReorderableListExtension para poder mostrar la lista de elementos. Dicho editor cuenta con tres apartados destinados a modificar tres aspectos del elemento.

El primer apartado sirve para manipular la geometría del elemento a través de la interacción con el mapa. Para ello se utiliza un botón con dos estados que permite iniciar y finalizar la edición, y, a medida que el usuario haga clic en el mapa se irán añadiendo nuevos puntos a la figura. **El segundo apartado es capaz de editar los campos** de área de influencia, id, nombre, descripción y documentación, es decir, la información del elemento. **El tercer y último permite la creación y manipulación de las acciones geoposicionadas** y, para ello, se utiliza una ReorderableList. Para su creación con posibilidades de extensibilidad se utiliza una factoría GeoActionsFactory en la cual se contienen todas las posibles acciones y se retornan con el método AvaliableActions que retorna una lista de tipos y nombres de acción para el editor. Para adquirir el nombre, todas las acciones cuentan con una propiedad Name. Una vez el usuario ha seleccionado la acción se utiliza CreateActionFor pasando como argumento el tipo de la acción o el nombre y la factoría retornará un clon de la acción deseada. Este editor destinado a las acciones geoposicionadas puede verse en la Fig. 6.19.

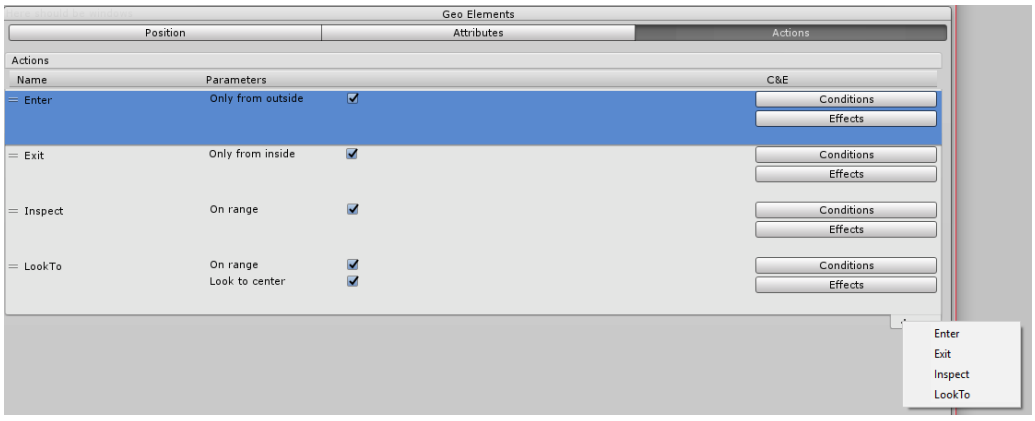

Fig. 6.19 Editor para GeoActions sobre GeoElements

Editor de MapScenes

Para manipular las MapScenes se añade un *plugin* al editor principal de tipo ReorderableListExtension llamado MapSceneWindow. En él, una única vista permite añadir referencias al mapa y poder visualizar el resultado final de la escena. MapSceneWindow contiene tanto un GUIMap en el que se representan los elementos como un PlaceSearcher que facilita el uso del mapa. Estos elementos, que son referencias, se manipulan utilizando una ReorderableList que, en el momento de añadir recopila todos los posibles objetos en el modelo de uAdventure y genera un menú desplegable en el que se puede seleccionar el objeto que se desee añadir. Para ello, se cargan desde ChapterDataControl todos los Item, NPC y Atrezzos, así como los GeoElement. En el momento de la adición, los tres primeros pasarán a ser ExtElemReference y el GeoElement pasará a GeoReference. Cada vez que se añade o quita un elemento a las referencias, para poder visualizarlos, se recargan y cachean los elementos referenciados del modelo y se introducen convenientemente en el GUIMap.

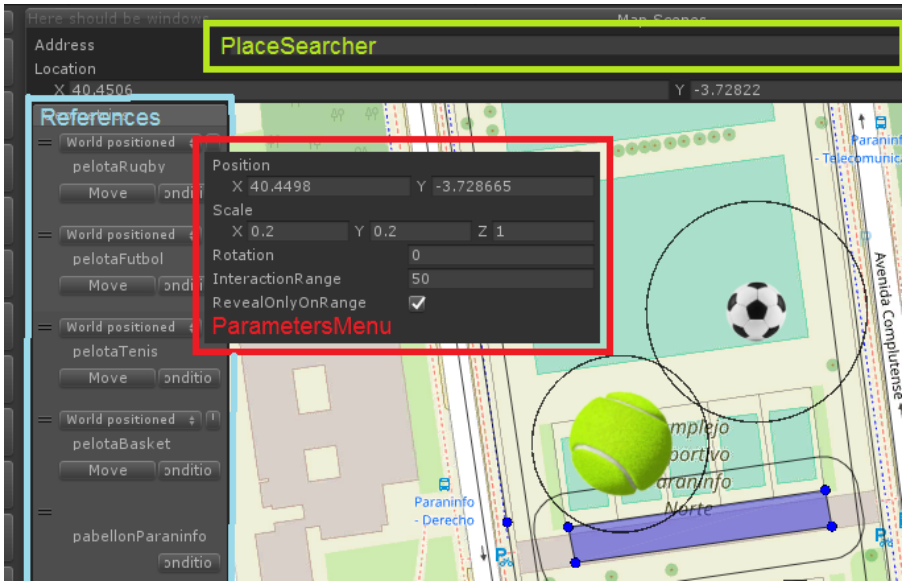

Fig. 6.20 Editor MapSceneWindow para escenas geoposicionadas.

Por un lado, se introducen en el mapa las GMLGeometries de los GeoElement y, por otro lado, se convierten los Descriptor de las ExtElemReference en PositionManager adecuados para el posicionado del recurso según el tipo de descriptor. Esta conversión se realiza en la clase DescriptorFactory, que permite al editor conocer a todos los posibles Descriptores y poder obtener instancias de cada uno de ellos en el momento en que el usuario desee que un elemento cambie de tipo de posicionamiento. Esta factoría incluye dos métodos:

GetAllPossibleDescriptors, que retorna una lista de nombres y tipos y, CreateDescriptor que recibe un tipo y retorna una instancia del mismo.

Para la edición genérica de los parámetros de cualquier descriptor se introduce el editor ParametersMenu que aparece en forma de "dropdown" (Fig. 6.20), recibiendo el descriptor y un diccionario donde se almacenarán los valores, genera de forma dinámica los campos necesarios para editar los parámetros. Esto simplificará además la futura extensibilidad de los posicionadores, que no requieren de editores personalizados para la configuración de sus parámetros.

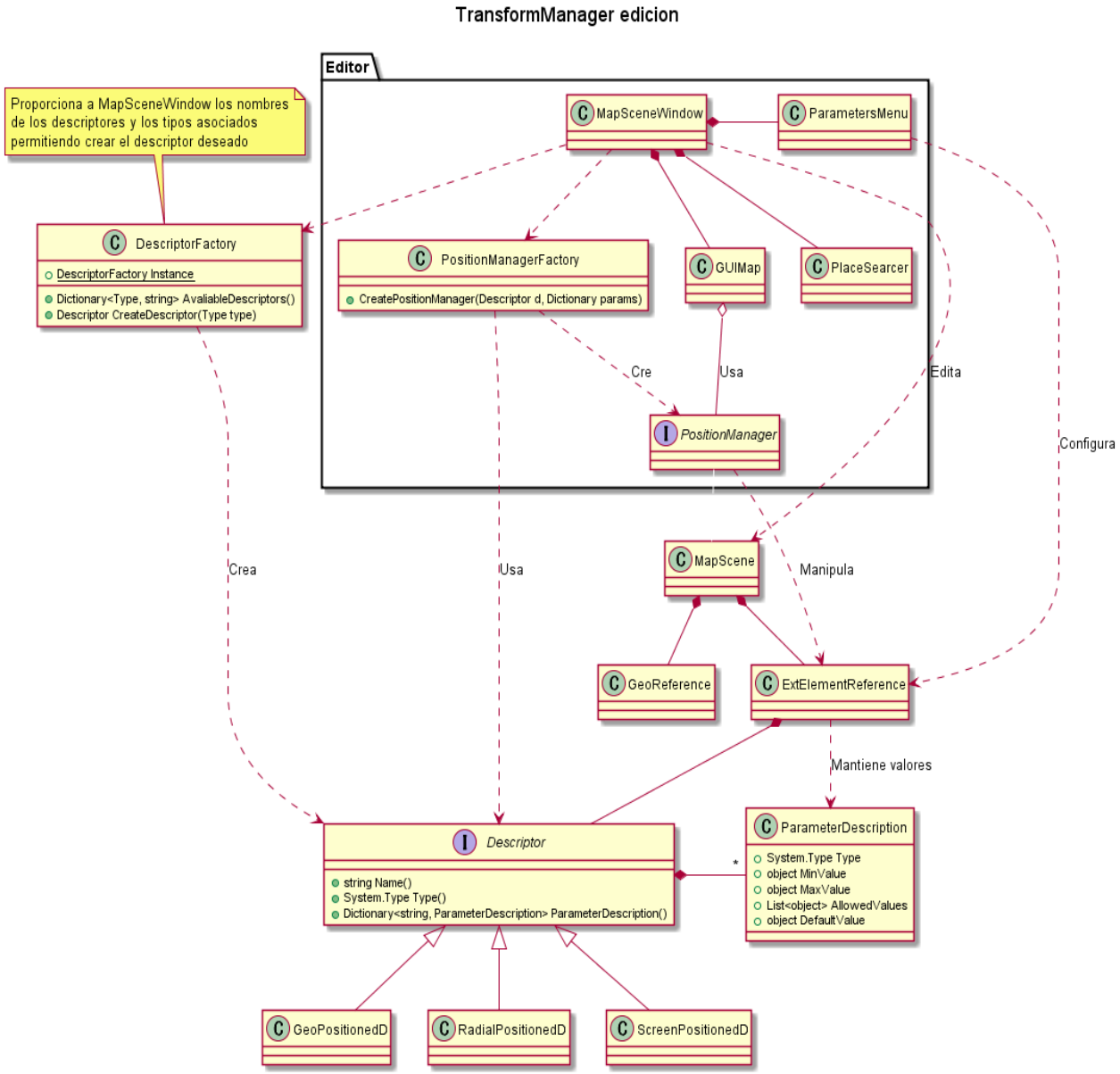

Fig. 6.21 Diagrama de clases de MapSceneWindow y los Descriptores

Finalmente, para transformar el Descriptor en un PositionManager se utiliza la factoría PositionManagerFactory que, en base a un Descriptor y un diccionario de parámetros, busca el PositionManager adecuado al descriptor y retorna una nueva instancia conectada a dicho diccionario. Por ello, los PositionManager cuentan con un nuevo atributo público llamado Type que permite asociarse a un descriptor y que la factoría utilizará para la creación.

En la lista de elementos se muestran de forma genérica a todas las referencias las condiciones de aparición del elemento. Las ExtElemReferences cuentan además con un elemento desplegable Popup que permite seleccionar entre los diferentes descriptores y un botón que permite configurar los parámetros abriendo la ventana ParameteresMenu.

El diagrama de clases de la Fig. 6.21 muestra la clase MapSceneWindow y su relación con los descriptores, así como la factoría de descriptores y el editor de parámetros de éstos. Como se observa en la figura, la clase MapSceneWindow, se encarga de coordinar toda la edición del modelo, el mapa y los elementos que aparecen en el mismo.

Serializadores de MapScene y GeoElements

En la capa de serialización y deserialización se han añadido nuevas clases para la persistencia de los nuevos elementos MapScene y GeoElement.

Las clases serializadoras y deserializadoras de GeoElement utilizan para la serialización de la geometría el estándar de XML para geoposicionamiento, GML. En éste se establece la estructura que se ha seguido para el almacenamiento de las posibles figuras geométricas y las coordenadas asociadas a sus vértices. Para serializar y deserializar las GeoActions se han añadido nuevas clases GeoActionWriter y GeoActionParser especializadas en las acciones existentes. Para conectarse con ambas, GeoElementWriter utiliza DOMWriterUtility al que facilita la lista de acciones a serializar y, de manera análoga GeoElementParser realiza el proceso de carga de acciones a través de DOMParserUtility. Gracias al uso de dichos sistemas, si se incluyeran nuevas acciones, el sistema dinámico vincularía sus *writers* y *parsers* de forma automática.

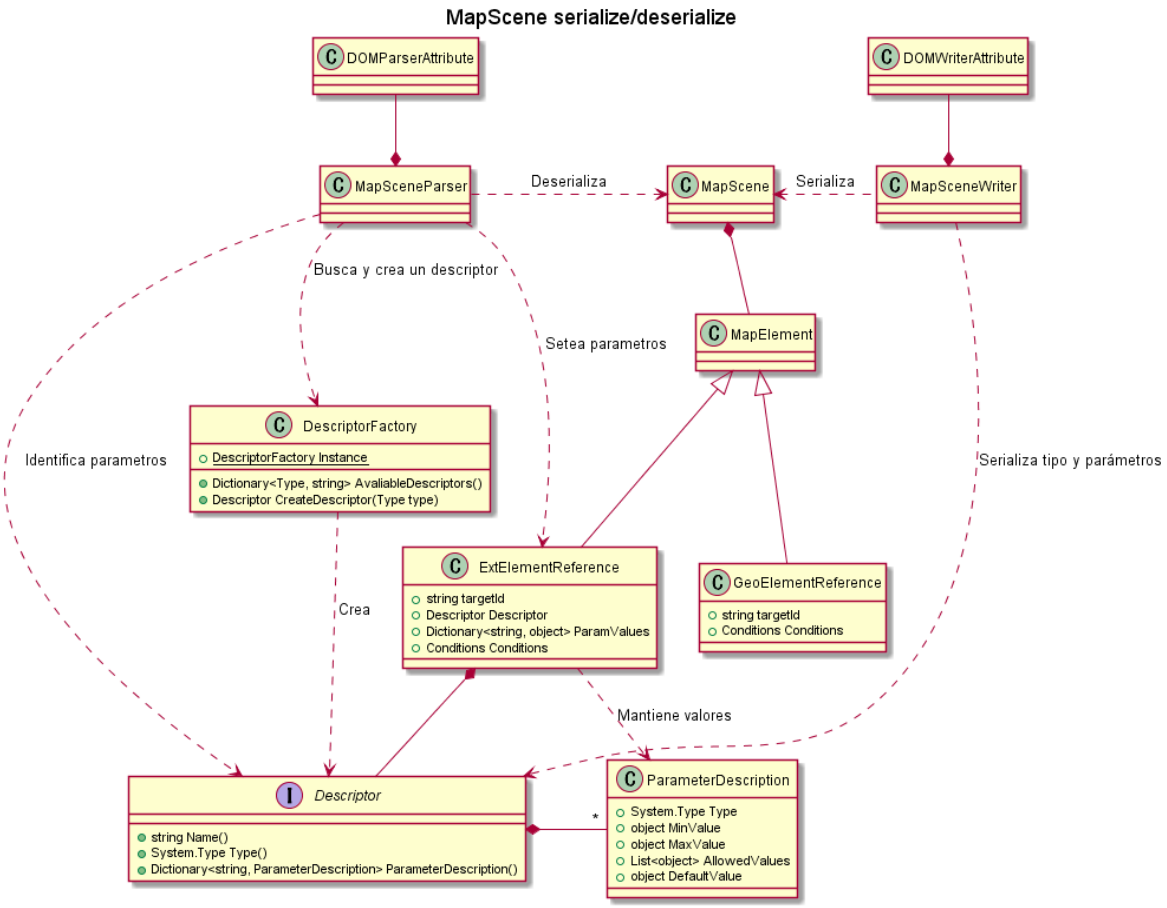

Fig. 6.22 Diagrama de clases de serialización y deserialización de MapScenes

Las clases serializadoras y deserializadoras de MapScene representadas en la Fig. 6.22, además de persistir y cargar los atributos de MapScene, realizan la serialización y deserialización de los MapElements. Para realizar la serialización de las GeoReferences, dado que no es necesario añadir el contexto, sólo se almacena el *id*. Sin embargo, para serializar las ExtElemReference, además del *id* se serializa tanto su descriptor, del cual se almacena el tipo como *string*, como los valores de los parámetros asociados al descriptor. En este punto los descriptores son especialmente valiosos ya que, como se muestra en la Fig. 6.23 permiten interpretar los parámetros del diccionario genérico y transformarlos en valores para el XML, asociando los tipos de manera dinámica en el proceso.

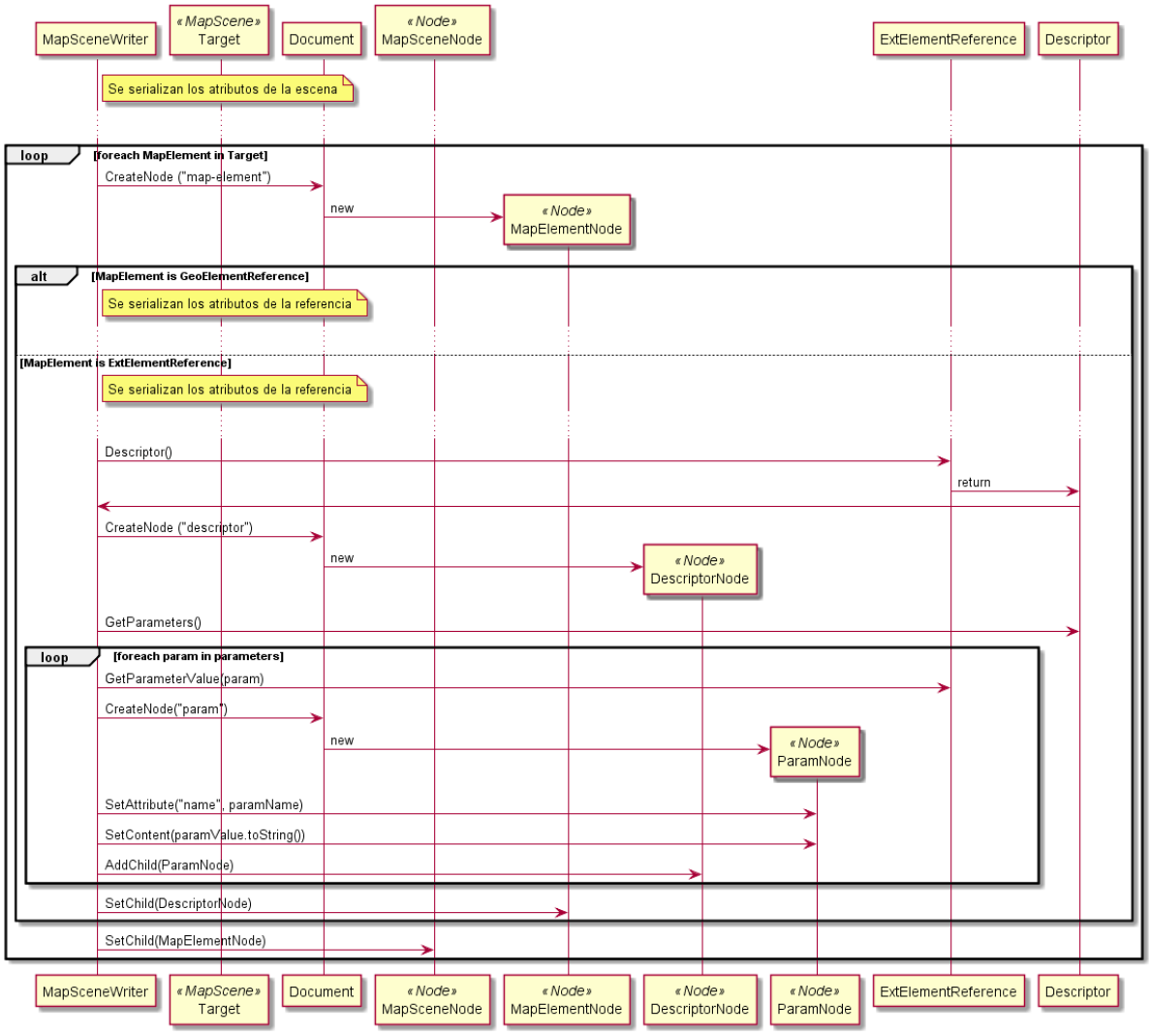

Fig. 6.23 Diagrama de interacción de la serialización del MapScene, destacando la interacción con el descriptor.

Por otro lado, para la deserialización, como se muestra en la Fig. 6.24 una vez se cargan los atributos de las referencias, en el caso de los elementos geoposicionados, se crean los descriptores utilizando la factoría DescriptorFactory, que realiza la creación del descriptor asociado al tipo cargado desde el XML. Una vez se tiene un descriptor, se itera sobre sus parámetros para así realizar la búsqueda y parseo del parámetro dentro del nodo XML asociado al descriptor, recuperando el nodo que contiene el valor del parámetro.

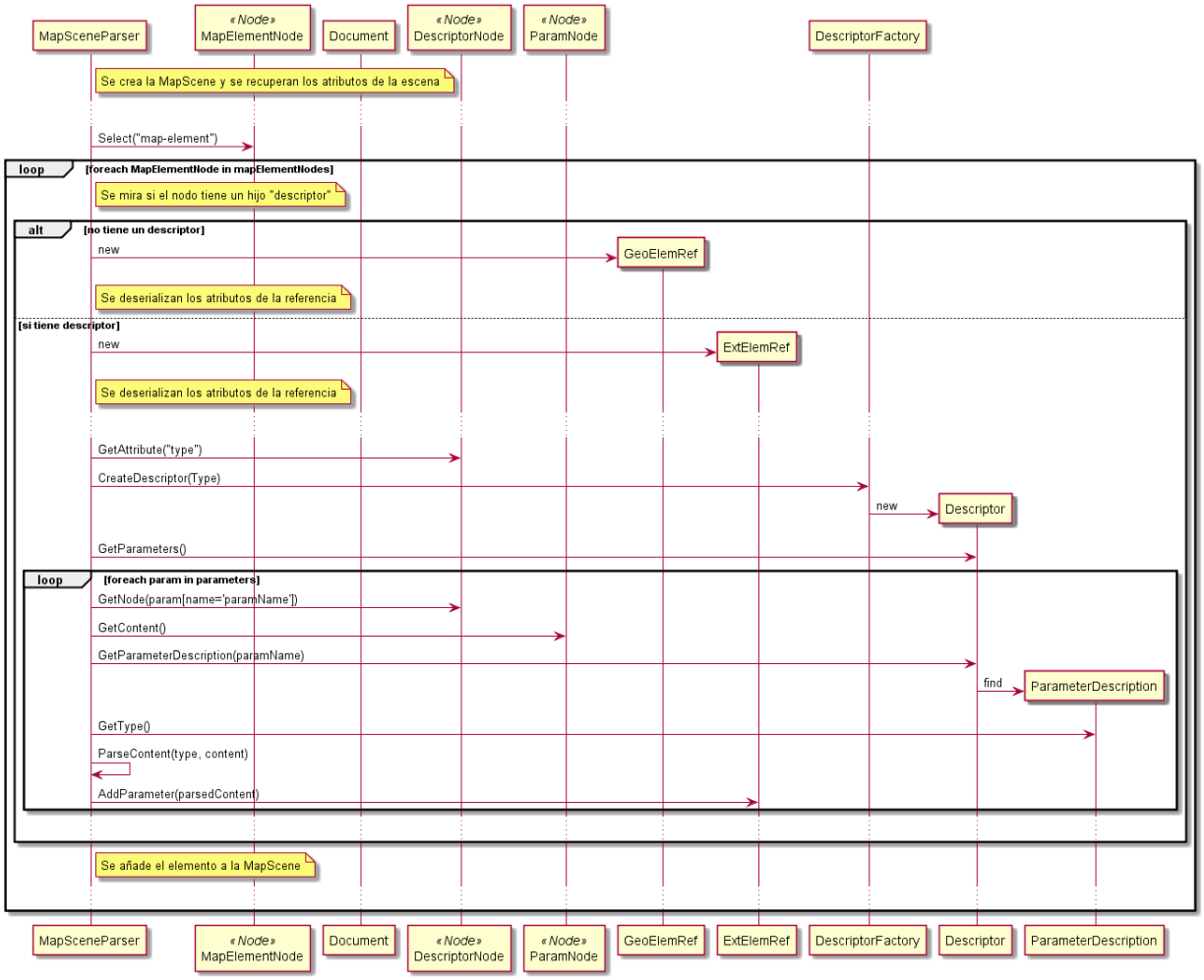

Fig. 6.24 Diagrama de interacción de la deserialización de MapScene, destacando la interacción con el descriptor

Ejecución de MapScenes

Para la parte de ejecución de las MapScene, dado que las MapScenes son IChapterTarget, se ha añadido una clase MapSceneMB que implementa IRunnerChapterTarget y permite que Game le entregue el flujo principal de la ejecución. Para la creación de MapSceneMB se ha añadido además un *plugin* MapSceneMBFactory a la factoría RunnerChapterTargetFactory que crea MapSceneMB al recibir una MapScene.

En su interior, la MapSceneMB contiene un TileManager de MapzenGO al que se le han añadido *plugins* para la conversión de MapElement a elementos tangibles en ejecución. El primer plugin es el uAdventurePlugin y permite por un lado el filtrado de los elementos en base a las condiciones y, por otro lado, el mantenimiento del listado de los elementos que han sido creados.

Este listado permite a los *plugins* que crean los elementos que, en el caso de que un elemento pueda aparecer en dos tiles diferentes, si ya ha sido creado en un tile, no se cree en los demás tiles que el elemento aparezca. Para la creación de los elementos, los plugins que se utilizan heredan de MapElementFactory que permite simplificar la creación de elementos pues realiza un bucle en el que se analizan los posibles elementos a crear y los asigna al tile correspondiente. Para el análisis provee dos métodos abstractos: Belongs, que permite conocer si el elemento que se va a crear pertenece al *tile* y Create que finalmente crea el elemento. Los dos plugins que heredan de MapElementFactory son GeoElementFactory y ExtElemFactory.

Por su parte, GeoElementFactory se encarga de la creación de GeoElementMB a partir de GeoElements. Para ello, además de instanciarlo crea una geometría utilizando la librería Triangulation.NET y la asigna como malla. En el caso de ser un punto, en lugar de crear una malla creará una esfera. Por otro lado, de ser un camino, lo extenderá previamente utilizando Clipper para que tenga un grosor visible.

Por otro lado, ExtElemFactory crea elementos de uAdventure en base a ExtElemReference. Sin embargo, dado que los elementos de uAdventure tienen autogestión del posicionamiento al heredar de Representable, es necesario establecer una clase intermedia GeoWrapper que aplique los posicionamientos que existen al Representable. En el momento de crearse, la clase GeoWrapper utiliza la factoría TransformManagerFactory, que funciona de una forma muy similar a PositionManagerFactory, pero en este caso genera un TransformManager, que es una clase especializada en manejar la propiedad Transform del GeoWrapper para que el elemento se vea finalmente como el posicionamiento deseado. Por ello, al igual que PositionManagers, existen tres TransformManager encargados de cada uno de los tipos de posicionamiento. Para su funcionamiento GeoPositionedTransformManager además de situar el elemento en las coordenadas corrige la desviación que causa de forma automática el Representable. Además, si la propiedad ShowOnRange está activa, mantendrá el objeto invisible hasta que el jugador esté cerca, iniciando además un lanzador de partículas que causa el efecto de descubrimiento (Fig. 6.26). Por su parte ScreenPositionedTransformManager es bastante más sencillo dado que la única labor que cumple es la de actualizar el elemento conforme la cámara se mueve por el mapa. El diagrama de clases de la Fig. 6.25 muestra el diseño que se ha descrito.

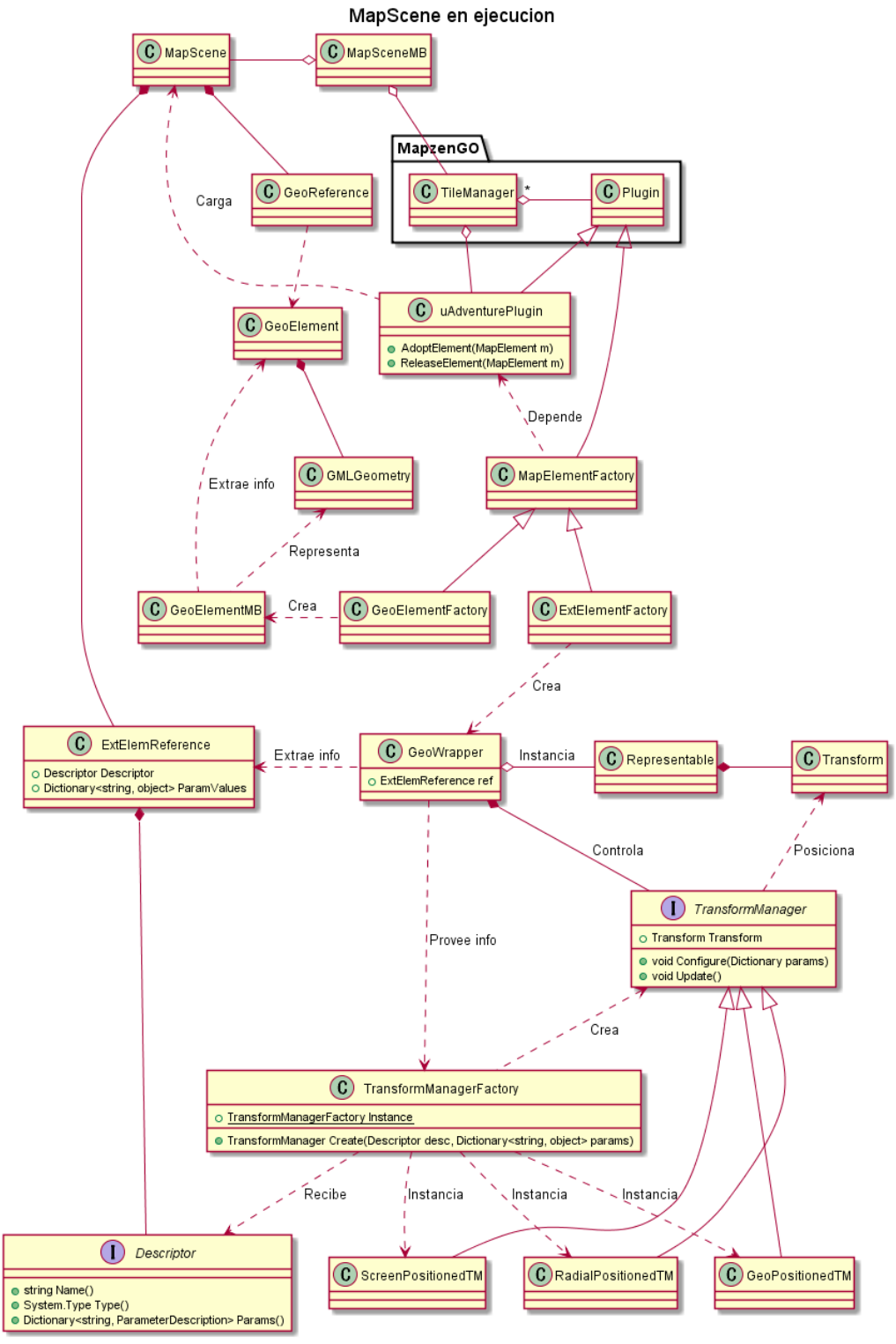

Fig. 6.25 Diagrama de clases de MapSceneMB, TransformManager y los plugins del mapa.

La figura Fig. 6.26 muestra un elemento de cada tipo en ejecución, así como un elemento que acaba de ser descubierto por el jugador al estar dentro de su rango de interacción.

Fig. 6.26 Elementos variados sobre el mapa con un elemento descubierto.

Además de la carga de los elementos a través de TileManager, MapSceneMB se encarga de asegurarse que el GPS se inicie y de que, una vez iniciado, el jugador se mueva de manera correspondiente. Para ello, la clase GPSController añade una pequeña capa de usabilidad al GPS de Unity, permitiendo iniciarlo con control de errores y ofreciendo dos métodos públicos IsStarted, que permite conocer si el GPS está arrancado o arrancando, e IsLocationValid que permite conocer si la localización actual cumple con los requisitos de precisión adecuados.

Para el control del jugador, siempre que se reciban nuevos datos de ubicación se actualizará su posición. En el caso de que el cambio de posición fuera mayor que 150 metros se teletransportará al jugador en lugar de permitirle que realice la animación de movimiento.

Para el control de las GeoActions de los GeoElement se incluyen las cuatro clases GeoActionManager especializadas en cada acción. La interfaz GeoActionManager permite a GeoElementMB delegar la responsabilidad de la detección y ejecución de la acción a través de un método Update. La clase AbstractGeoActionManager simplifica la ejecución de acciones, controlando que Game no esté ejecutando nada en dicho momento, comprobando las condiciones de la acción y dejando un método abstracto CustomChecks para cada tipo de acción. Siempre que todas las condiciones se cumplan, AbstractGeoActionManager se encargará también de ejecutar la acción. De este AbstractGeoActionManager heredan EnterGeoActionManager, ExitGeoActionManager, LookToGeoActionManager e InspectGeoActionManager, que comprueban en su método CustomCheck que se cumplan las condiciones de cada tipo específico de acción. Para la creación de forma simplificada de las GeoActionManagers se utiliza la clase

GeoActionManagerFactory que, a través del método Type del GeoActionManager permite reconocer el tipo de acción asociado al *manager* y crear un nuevo *manager* en consecuencia.

El diagrama de la Fig. 6.27 muestra la arquitectura para las acciones expresada anteriormente. En él se puede observar cómo los manager detectan si deben ejecutarse en base al jugador y cómo el manejador abstracto es el encargado de comprobar las condiciones y de ejecutar el efecto.

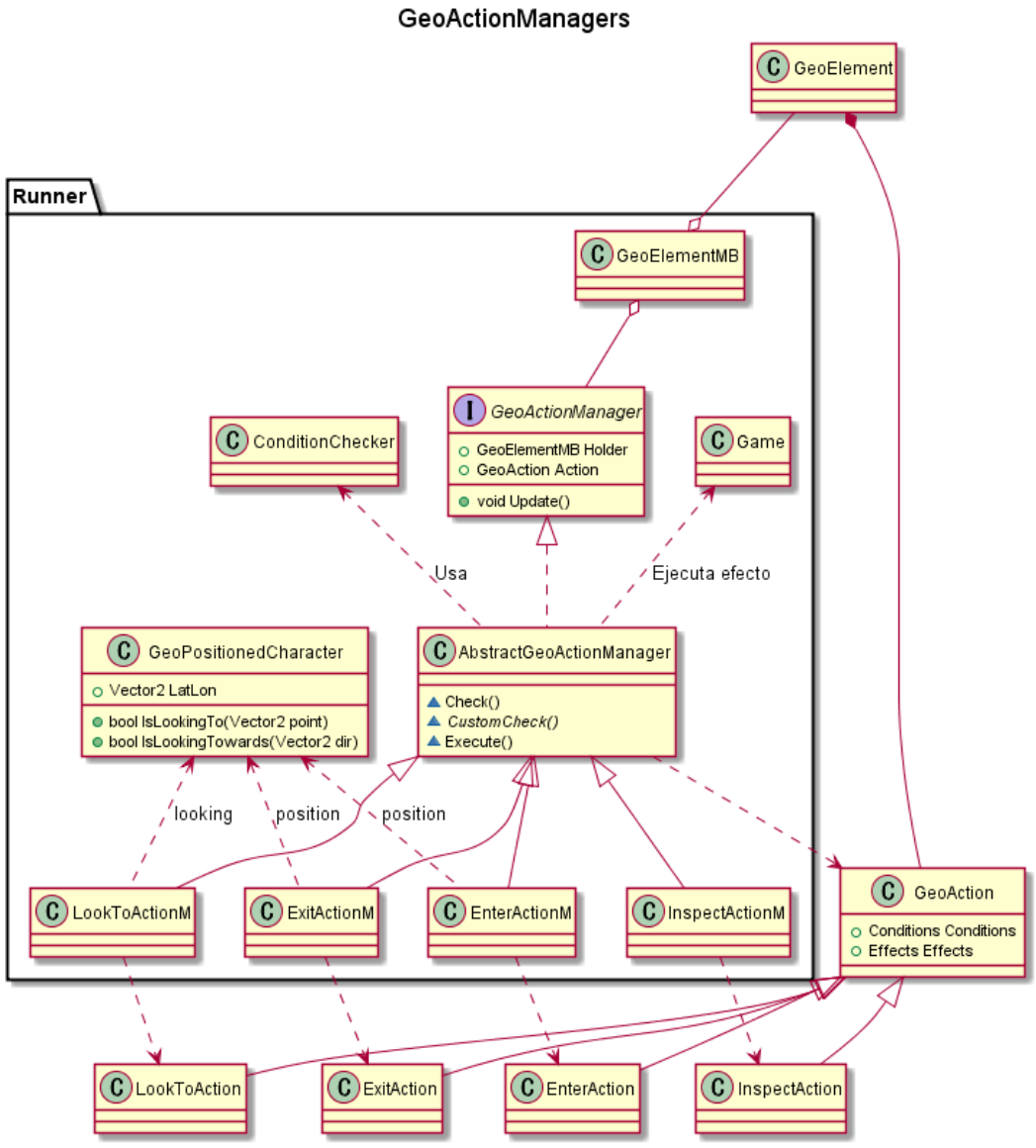

Fig. 6.27 Diagrama de clases de acciones geoposicionadas en ejecución.

Nuevos efectos de geoposicionamiento

Relacionados con el geoposicionamiento se han creado además tres nuevos efectos aprovechando las nuevas funcionalidades de extensibilidad de CustomEffect. Estos efectos permiten, por un lado, el lanzamiento de escenas limitadas a una zona concreta y, por otro lado, gestionar la navegación.

Para el lanzamiento de escenas limitado a una zona concreta se ha extendido el efecto TriggerSceneEffect del cual se ha extendido su funcionalidad para que, una vez cargada a nueva escena, se monitorice al jugador para comprobar que siga en la zona deseada. En este nuevo efecto TriggerZonedSceneEffect, si el jugador saliera de la zona, la escena retornaría a la escena anterior al lanzamiento del efecto. Gracias a este efecto es posible simular no sólo la entrada a lugares por geoposicionamiento, sino la generar una experiencia de estancia del jugador en dicho lugar. Para ello, la clase CustomEffectRunner encargada de ejecutar el efecto, que es TriggerZonedSceneEffectRunner instancia un GameObject con la componente ZoneControl que se encargará de retornar a la escena cuando se salga de la zona. Para editar el efecto, un editor de efectos personalizado TriggerZonedSceneEffectEditor que extiende TriggerSceneEffectEditor (Fig. 6.28), permite seleccionar la zona utilizando para ello los GeoElement existentes y almacenando en el efecto una referencia al elemento. Por último, también se han creado dos clases para la serialización y deserialización del efecto llamadas TriggerZonedSceneEffectWriter y Parser respectivamente.

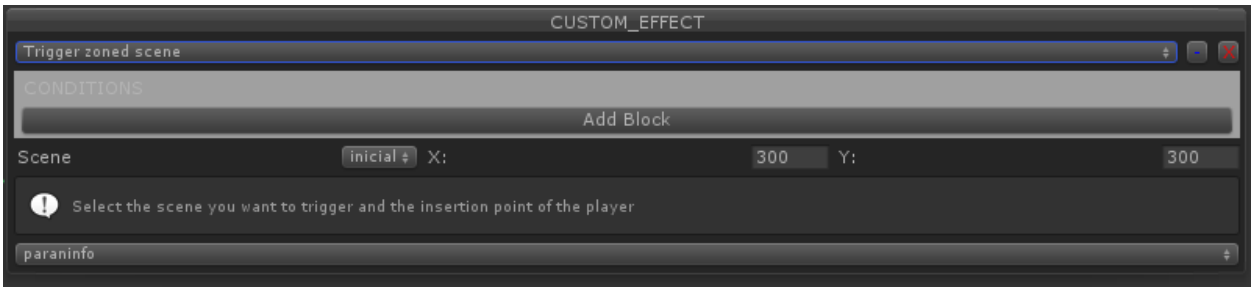

Fig. 6.28 Editor de TriggerZonedSceneEffect que contiene un selector de GeoElements.

Por otro lado, para la realización de navegación a través del mapa se ha creado un navegador que se encarga de procesar la secuencia de pasos. Cada paso consiste en la navegación hacia un elemento del mapa, que puede ser, GeoElement o un elemento de uAdventure. Éstos pueden contener además una propiedad *locks* que impedirá que el navegador avance hasta que se

solicite a través de un script o efecto, teniendo como fin esperar hasta que el usuario complete la tarea en dicho destino. Para navegar existen dos tipos de seguimientos: por orden o por cercanía. Si se configura el navegador por orden, consumirá los pasos por orden, mientras que, si se configura por cercanía irá consumiendo los pasos en base a la cercanía al jugador. Además, en el caso de los caminos deberán recorrerse todos sus puntos para completar la navegación. Aprovechando esto, el usuario puede simplificar la creación de una ruta compleja (similar a la que entregaría un planificador de rutas convencional) unificándola en un único elemento.

Para su funcionamiento, se ha añadido una clase NavigationController que recibe una lista de NavigationStep que contienen en su interior a referencia o *id* y el booleano para bloquear. Una vez se insertan los pasos se pasa al navegador el tipo de navegación. El navegador, entonces, seguirá una serie de pasos sencillos: seleccionar un paso no completado, comprobar si se ha alcanzado, actualizar la flecha de navegación y, en el caso de que se haya alcanzado, completar el paso. Esto se traduce en cuatro métodos: SelectStep, IsReached, UpdateArrow y CompleteStep. Dentro de SelectStep se realiza la búsqueda por referencia del elemento en el mapa en ejecución hacia el cual se navegará. Si se ha configurado por orden, se selecciona el primer paso, mientras que, si se ha configurado por cercanía, se realiza una comparación de la distancia de cada elemento hasta el jugador y se retorna el paso más cercano. Dentro de IsReached se analiza el estado de completitud del paso actual y se retorna si se ha alcanzado la posición requerida para completarse. Si son GeoElements se utiliza el método InsideInfluence de la geometría, excepto para los caminos, que se busca la cercanía al punto que se esté completando en ese momento. Si son elementos externos de uAventure se accede a sus propiedades *position* e *influence* de los atributos del descriptor y se utilizan para calcular si se ha alcanzado. Dentro de UpdateArrow se actualiza la dirección de la flecha (Fig. 6.29).

Fig. 6.29 Flecha de navegación.

Para puntos y elementos externos se apunta a su único punto, para los caminos se selecciona el punto siguiente sin completar y para los polígonos se apuntará a su centro. Finalmente, en CompleteStep se marcará el paso como completado (siempre que no esté *locks* activo), permitiéndose además que, si es un camino, sólo se marque como completado si se han superado todos los puntos del camino. De esta forma, el navegador funciona de forma transparente a los elementos a los que navega, siguiendo un ciclo de vida claro y sencillo.

Para utilizar el navegador se han incluido NavigateEffect y NavigationControlEffect.

El primero, NavigateEffect, permite configurar el navegador para que realice una serie de pasos. Los pasos se seleccionan del modelo de uAdventure utilizando las referencias, incluyendo además el atributo *locks*. Además, el efecto incluye el tipo de acercamiento que se utilizará. Para su integración con uAdventure se han añadido un *plugin* para el editor de efectos NavigateEffectEditor, un *plugin* para su serialización NavigateDOMWriter, un *plugin* para su deserialización NavigateDOMParser y un *plugin* de ejecución de efecto NavigateEffectRunner. El primero, NavigateEffectEditor (Fig. 6.30) utiliza una lista de tipo ReorderableList para poder organizar, añadir y quitar elementos. Para añadirlos se utiliza un menú contextual que muestra por categorías todos los elementos que existen en el modelo a los que se puede navegar. Para seleccionar el tipo de navegación se utiliza un desplegable de tipo Popup. Por otro lado, el NavigateEffectRunner busca un NavigationController en la escena al que asignará los atributos del efecto y, posteriormente, iniciará utilizando el método Navigate, eliminando con ello cualquier navegación anterior. Si no hubiera un navegador, el ejecutor entonces crearía uno nuevo cargando el *prefab* utilizando Resources.

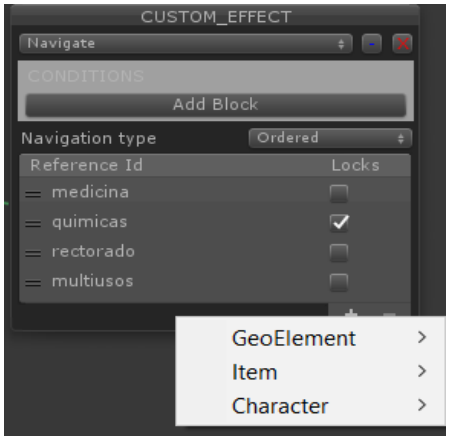

Fig. 6.30 Editor de efecto de navegación NavigateEffectEditor.

Para el segundo efecto, NavigationControl, se utiliza un atributo que permite diferenciar el tipo de control a realizar. El control permite cuatro acciones: completar el paso y avanzar al siguiente, retroceder un paso, cambiar el paso utilizando una referencia o cambiar el paso utilizando un índice. Para realizarlo, se han añadido también cuatro *plugins*. El primero es un *plugin* de editor de efectos NavigationControlEffectEditor (Fig. 6.31), en el que se puede seleccionar el tipo de control y, en función del tipo de control seleccionado, el parámetro asociado al tipo de control, que puede ser, un desplegable que permite seleccionar la referencia o un campo para números enteros que permite indicar el índice. El segundo *plugin* es el *plugin* de ejecución NavigationControlEffectRunner que sencillamente aplica el control sobre el navegador en ejecución. Además, se han incluido otros dos *plugin* para serialización y deserialización NavigationControlDOMWriter y Parser trivialmente sencillos.

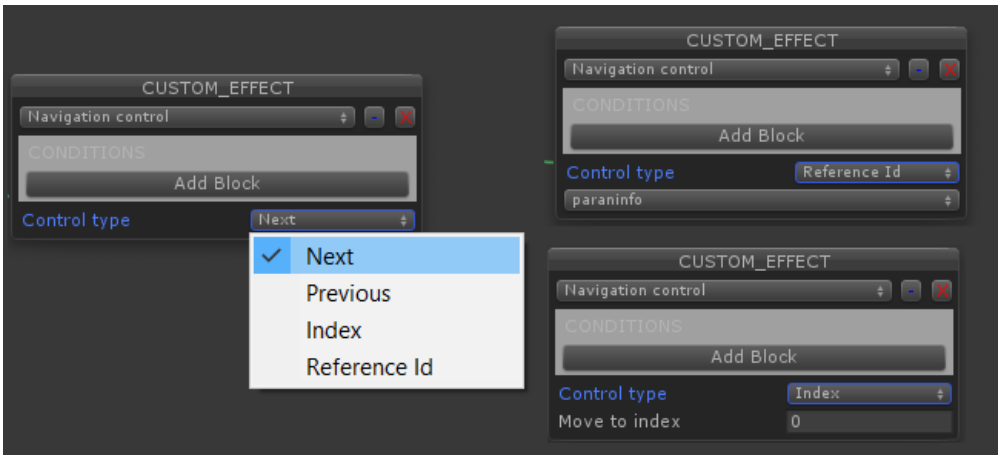

Fig. 6.31 Editor para NavigationControlEditor mostrando los diferentes tipos de control.

Con todas las extensiones implementadas se cumplen todos los objetivos propuestos para el geoposicionamiento de los apartados 5.1.1, 5.1.2 y 5.1.3 de las aportaciones. Gracias a todas las capacidades de extensibilidad de uAdventure, las nuevas extensiones de geoposicionamiento se han introducido de forma limpia en el modelo sin tener que modificar el comportamiento nativo del núcleo y con ello, demuestra su efectividad.

6.3.2. Extensiones para códigos QR

Tomando como punto de partida los módulos creados de QRReader y QRCoder se han desarrollado diversas extensiones para editor y ejecución que permiten al desarrollador integrar códigos QR como mecánica de juego, permitiendo utilizar mecánicas de posicionamiento en interiores al garantizar que el jugador se encuentra físicamente delante del QR que escanea.

De cara al modelo, los códigos QR se representan con una única clase que se almacena en el Chapter, que puede contener información, tener una serie de condiciones para poder ser escaneado y generar una serie de efectos tras escanearlo. En su información hay dos campos, uno para documentación y un campo de contenido que se mostrará directamente al jugador cuando éste escanee el QR. Sin embargo, para prevenir que el usuario conozca el contenido, el QR se generará a partir del *id* del QR y no por su contenido como se hace habitualmente. De esta forma uAdventure hace de capa de seguridad frente a las acciones indebidas.

Para crear los códigos QR se ha creado un *plugin* para el editor principal. Este editor hereda del tipo ReorderableListEditorWindowExtension y permite editar los QR del modelo. En su ventana principal se muestra el código QR que se está rellenando, los campos de texto para el contenido y la documentación, dos botones para abrir los editores de las condiciones y los efectos y dos botones que permiten guardar la imagen o imprimirla directamente. Para que el QR no aparezca sobre el fondo del editor directamente se utiliza el estilo *preBackground* de Unity que crea el efecto de estar utilizándose un visualizador. Para el botón de guardar la imagen, en primer lugar, se abre el selector de ficheros integrado con Unity utilizando el método SaveFilePanel de EditorUtility y, en caso de que retorne un nombre de fichero válido se utiliza el método WriteAllBytes de System.IO.File para escribir los *bytes* de la imagen directamente en el fichero. Para extraer los *bytes* de la imagen, Unity provee dos métodos que los permiten extraer de la textura en formato PNG o JPG. En este caso se ha utilizado EncodePNG para obtener y salvar la imagen en formato PNG. Por otro lado, para imprimir directamente desde Unity, éste no provee de ninguna utilidad integrada con el editor. Por ello se ha utilizado PrintDialog y PrintDocument de System.Drawing.Printing[42] perteneciente a las librerías nativas de .NET. Para ello se asigna el documento al diálogo de impresión y se lanza el diálogo. Este, de resultar satisfactorio solicitará que se dibujen las páginas a través del delegado PrintPage del documento.

[42] https://msdn.microsoft.com/es-es/library/system.drawing.printing.printdocument(v=vs.110).aspx

Entonces, en el método asociado al delegado se recibe la página a rellenar, en la cual se pueden dibujar imágenes en formato BMP. Dado que el formato que entrega Unity al extraer toda la información de la textura es incompatible con la clase Bitmap de System.Drawing se ha utilizado una conversión píxel a píxel, transformando para cada píxel su color de Unity.Color a System.Drawing.Color e insertándolo en el Bitmap. En el proceso, además, dado que las coordenadas del documento crecen de forma inversa en el eje Y comparado con el editor se han invertido los píxeles. Con todo ello, es posible imprimir los códigos QR desde el editor, mejorando la facilidad de uso para usuarios inexpertos. La Fig. 6.32 muestra el editor final.

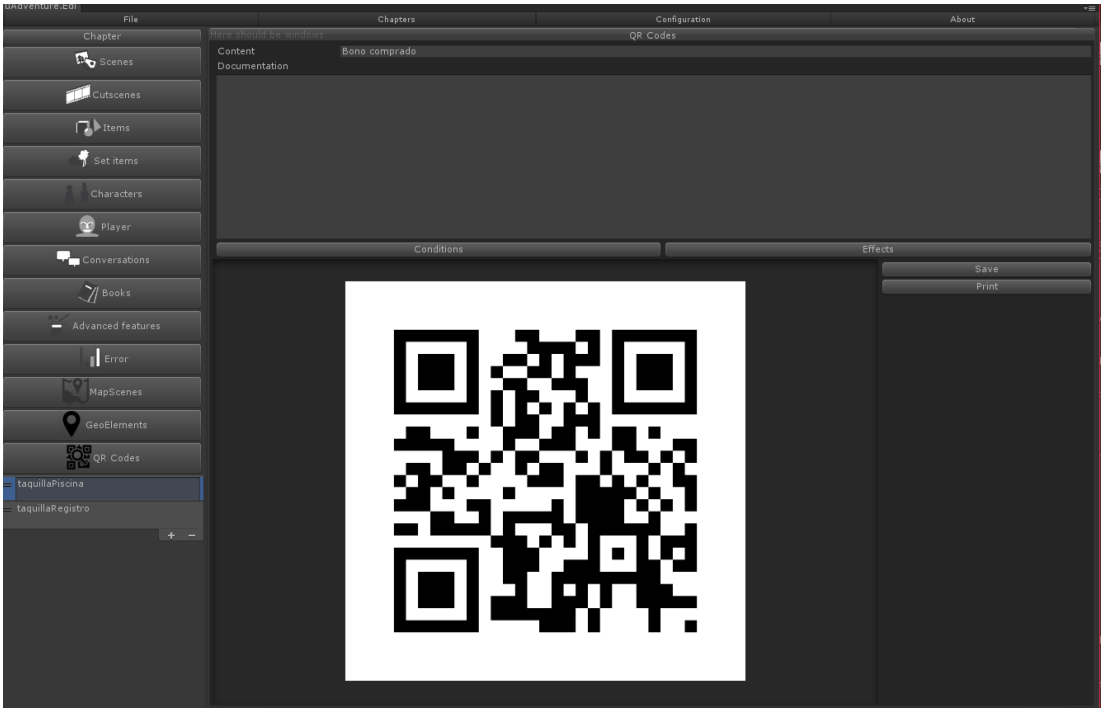

Fig. 6.32 Plugin para el editor de uAdventure de códigos QR.

Para ejecutar los QR, a la hora de analizar el problema existían dos posibles soluciones con sus pros y contras. La primera, integrar un botón en las escenas para abrir la cámara, pese a ser la forma más sencilla de cara al usuario, requeriría modificar el núcleo de uAdventure en lugar de utilizar las extensiones que se han creado, y, por otro lado, sería una solución menos restrictiva para el usuario y más forzosa de cara al desarrollador. En contra a esto, la segunda solución, que es mucho más flexible, es la creación de un efecto personalizado que sirva para abrir un escáner de QRs. El principal inconveniente de este acercamiento es que, para poder

usarse es necesario integrar elementos como objetos o áreas para representar el botón que lanzará el efecto.

Debido a la mejor integración con el sistema, en contra de la facilidad de uso, se ha tomado la segunda opción, con la intención de que sea más fácil de integrar en el futuro si se llegan a implementar extensiones de botones o iconos.

Para llevarlo a cabo se ha creado un nuevo efecto OpenQRPrompt que lanza el escáner de códigos QR en ejecución. Dicho escáner implementa la clase CustomEffectRunner y se conecta con el QRReader mediante el UnityEvent que notifica los escaneos de QR. Sin embargo, dado que el escáner es un MonoBehaviour que incluye en su interior diversos elementos gráficos, éste no es el runner encargado de ejecutar el efecto, sino que es QRPromptEffectRunner la clase encargada de dicha tarea. En el interior de dicho efecto se obtiene un *prefab* utilizando Resources.Load que incluye todos los componentes gráficos y de UI del QRPrompt. Tras ello, se instancia y se obtiene la componente QRPrompt tras lo cual, QRPromptEffectRunner se convierte en una clase de tipo mediadora (patrón Mediator [41]) que recibe los eventos del EffectHolder y los transmite al QRPrompt. Para la detección de QRs el efecto introduce una lista de QR y una variable que define el tipo de lista, pudiendo ser lista blanca o lista negra. Si es lista blanca, sólo se podrán detectar los QR de la lista, mientras que de ser lista negra ocurriría lo contrario. Este filtrado de referencias se realiza en el método OnQRCode en función del tipo de lista y, de ser una referencia válida, se busca el QR en el modelo, se comprueban sus condiciones y si son positivas, se ejecutan sus efectos. Sin embargo, dado que QRPromptEffectRunner es un efecto en sí mismo se crea un nuevo EffectHolder y se le transmiten los eventos de *execute* provenientes del efecto padre. De esta forma, cuando termina el efecto del interior del código QR termina también el efecto OpenQRPrompt y continúa la ejecución de efectos anterior. Además, en el momento de escanear el QR se ocultan todos los gráficos del escáner para dejar visible la escena que esté abierta por debajo. En la figura QR puede observarse el escáner de códigos QR en ejecución. Para hacerlo más dinámico se ha añadido una barra roja que se mueve por el espacio escaneable haciendo visible la interacción del escáner.

Fig. 6.33 Escáner de códigos QR en ejecución.

Para editar el efecto se ha añadido un *plugin* de editor de efectos QRPromptEffectEditor que contiene en su interior un desplegable para seleccionar el tipo de lista y un botón que permite añadir a la lista cualquier QR que no esté contenido en ella previamente. Para representar la lista se ha utilizado una ReorderableList, aunque el orden de las referencias no influye en el resultado final. Por otro lado, se han añadido también dos *plugins* para la serialización y deserialización que son trivialmente simples.

Como conclusión, la integración de códigos QR ha sido sencilla debido a la cantidad de extensiones para Unity de dicha temática y gracias a las capacidades de extensibilidad desarrolladas para uAdventure ha sido realmente simple integrar las soluciones de Unity en el motor de uAdventure. En resumen, se han implementado un total de 5 *plugins* (editor general, editor de efecto, serializador, deserializador y runner) para integrar las mecánicas de códigos QR.

6.4. Learning Analytics de Geoposicionamiento

Con la creación de uAdventure se cambió el formato de evaluación y seguimiento de las actividades educativas para utilizar análisis basados en *datamining* aplicados al aprendizaje conocido como Learning Analytics. Gracias a las analíticas el proceso de aprendizaje es más completo y más dependiente de acciones que de resultados. Para el seguimiento de acciones de

los jugadores utilizamos el estándar xAPI. *Experience API* (xAPI) es el estándar utilizado para LAs enfocado en el seguimiento de experiencias de los jugadores. En nuestro rastreo de seguimiento hemos utilizado un perfil de acciones orientadas a juegos serios [26] definido sobre xAPI.

uAdventure integra el servicio de LAs del proyecto RAGE del H2020 desarrollado por el grupo e-UCM. Para ello, uAdventure utiliza el *tracker* de RAGE para Unity que permite traducir las interacciones del juego en experiencias de xAPI que se transmiten a RAGE, como, por ejemplo: el acceso a escenas, la interacción mediante acciones, las respuestas a preguntas, el completado de tareas, etc. Utilizando dicho *tracker,* se han integrado (extendiendo en algunos casos) los nuevos elementos desarrollados para geoposicionamiento, ampliando el perfil de experiencias sobre juegos serios de xAPI: el movimiento del jugador, la realización de acciones geoposicionadas y el descubrimiento de objetos en el mapa.

Para trazar el movimiento del jugador ha sido necesario incorporar un nuevo verbo *moved* que representa la acción de mover como experiencia de juego, pues dicho caso no era contemplado en el perfil de xAPI para juegos serios. El movimiento se define además con un tipo que puede ser de lo más variado, desde incluir movimiento del personaje por la escena hasta representar el movimiento de las coordenadas GPS. Este último tipo se ha bautizado con el nombre de *geolocation* y debe de añadir a la traza la extensión[43] de coordenadas. Esta extensión debe llevar el nombre de *location* y en su interior contener un formato de tipo *geopoint*[44]. En adición a este verbo, también se ha creado una extensión *location*, que permite almacenar en ella unas coordenadas.

Para integrar la traza de tipo *moved* en el *tracker* se ha añadido una nueva clase que permite incorporar dicho verbo y tipo, y, además se han añadido casos especiales para incorporar la extensión de localización, que permiten que sea correctamente almacenada en CSV y enviada posteriormente RAGE en forma de objeto JSON.

Una vez resueltos los problemas de formato, se incorporaron las trazas y extensiones a los *plugins* de uAdventure. Para ello, en el GPSController se ha añadido la traza de tipo *moved* para que sea enviada cada cinco segundos con la posición del jugador, siempre que ésta esté

[43] https://github.com/adlnet/xAPI-Spec/blob/master/xAPI-Data.md#41-extensions

[44] https://www.elastic.co/guide/en/elasticsearch/reference/current/geo-point.html

disponible. Una vez es recibida en RAGE, se incorpora al índice que almacena los datos el tipo *geo_point* y se aprovecha la funcionalidad de *tile maps* incorporada en Kibana, que es la herramienta de visualizaciones de RAGE. El resultado permite la creación de varios tipos de mapas, desde mapas con puntos o frecuencias, hasta mapas de calor. En el caso de este proyecto se han decidido integrar mapas de calor por su fácil visualización y rápida asociación de la información. El resultado de uno de los nuevos mapas de calor puede verse en la Fig. 6.34.

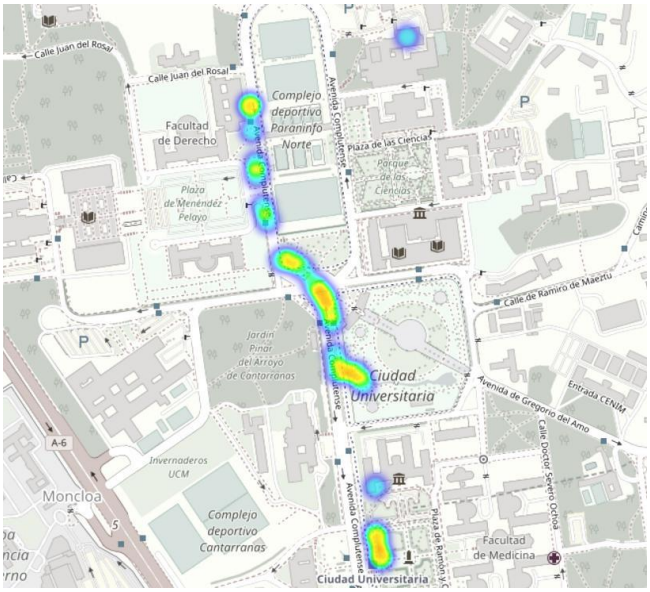

Fig. 6.34 Mapa de calor generado por Kibana basado en las trazas de tipo moved.

Para trazar la creación de trazas en base a eventos de tipo acción geoposicionada, generados por las GMLGeometries, se han adaptado otros tipos de acciones de xAPI a las que se ha incorporado la extensión de *location*. Tanto para Enter, Exit y LookTo se han añadido trazas del tipo *accesed* cuyo tipo de acceso es el de *zona* dado que se trata de una región de coordenadas. A esta traza se añaden dos extensiones, el tipo de acción a través de la extensión *geocommand*, que puede tomar los valores enter, exit y lookto, y, además, la extensión *location* para conocer el punto exacto donde se produce la acción. Con este tipo de trazas podrán generarse mapas de calor de las acciones de tipo *geocommand* cuando ocurran.

Por último, para trazar el control de acceso a elementos geoposicionados se añade una extensión de tipo *geo_element_<nombre>* que podrá ser utilizado para conocer, en la próxima traza, que previamente se accedió al elemento.

Con estos tres tipos de trazas el experimento podrá generar datos interesantes que permitirán evaluar el correcto funcionamiento de los sistemas para su futura mejora.

Capítulo 7. Experimento

7.1. Creación del juego

En base a las herramientas desarrolladas se ha creado el experimento expuesto en el capítulo 4 de aportaciones. Para ello, se ha realizado un diseño en diagrama actividades (Fig. 7.1) en el cual se modela la especificación del diseño de forma secuencial. En la secuencia se muestra claramente que, pese a que los elementos hacia los que se navega son distintos, la experiencia final de juego es la misma dado que derivan en la misma actividad en el destino. El juego se divide en cuatro frases en las que intervienen un personaje nuevo en cada una.

La primera fase es la fase introductoria en la que aparece Ayudante que es el cisne de la complutense. A diferencia de los demás personajes este nos acompañará en toda la aventura permitiéndonos conocer la tarea actual. Nada más comenzar Ayudante nos dará una breve introducción de la actividad que se va a realizar y preguntará al jugador si conoce el edificio de estudiantes. Si la respuesta es sí se solicitará que lo mire desde su ubicación, mientras que, si se responde que no, antes de mirarlo se explica dónde está. Si el jugador mira al lado contrario, hacia medicina, se activará la broma que sólo aparecerá una vez. Para ambos edificios se han utilizado GeoElements utilizando zonas, a los cuales se ha añadido una acción LookTo con la propiedad OnRange desactivada para sus respectivos eventos. Una vez se ha mirado al edificio correcto, Ayudante dará la opción de elegir el polideportivo al que ir o que se elija uno por cercanía. Dependiendo de la opción se lanzarán distintos NavigationEffect, siendo el último destacable por utilizar el tipo de navegación por cercanía. Para navegar, ambos polideportivos se han modelado como GeoElements.

La segunda fase comienza al alcanzar cualquiera de los polideportivos, con la acción Enter que se lanza cuando el jugador entra en su influencia. Al entrar se abre una nueva MapScene única para cada polideportivo y a continuación se lanza un diálogo desde Ayudante que indica al jugador que camine hasta el pabellón y lanzará una navegación hacia el pabellón adecuado al polideportivo que también es un GeoElement. Cada pabellón en su acción Enter abrirá utilizando TriggerZonedScene una escena de uAdventure en la que aparecerá el personaje Entrenador que hará una introducción y mandará al jugador a buscar cuatro pelotas por el polideportivo. Cada pelota es un elemento de uAdventure con el tipo de posicionamiento global

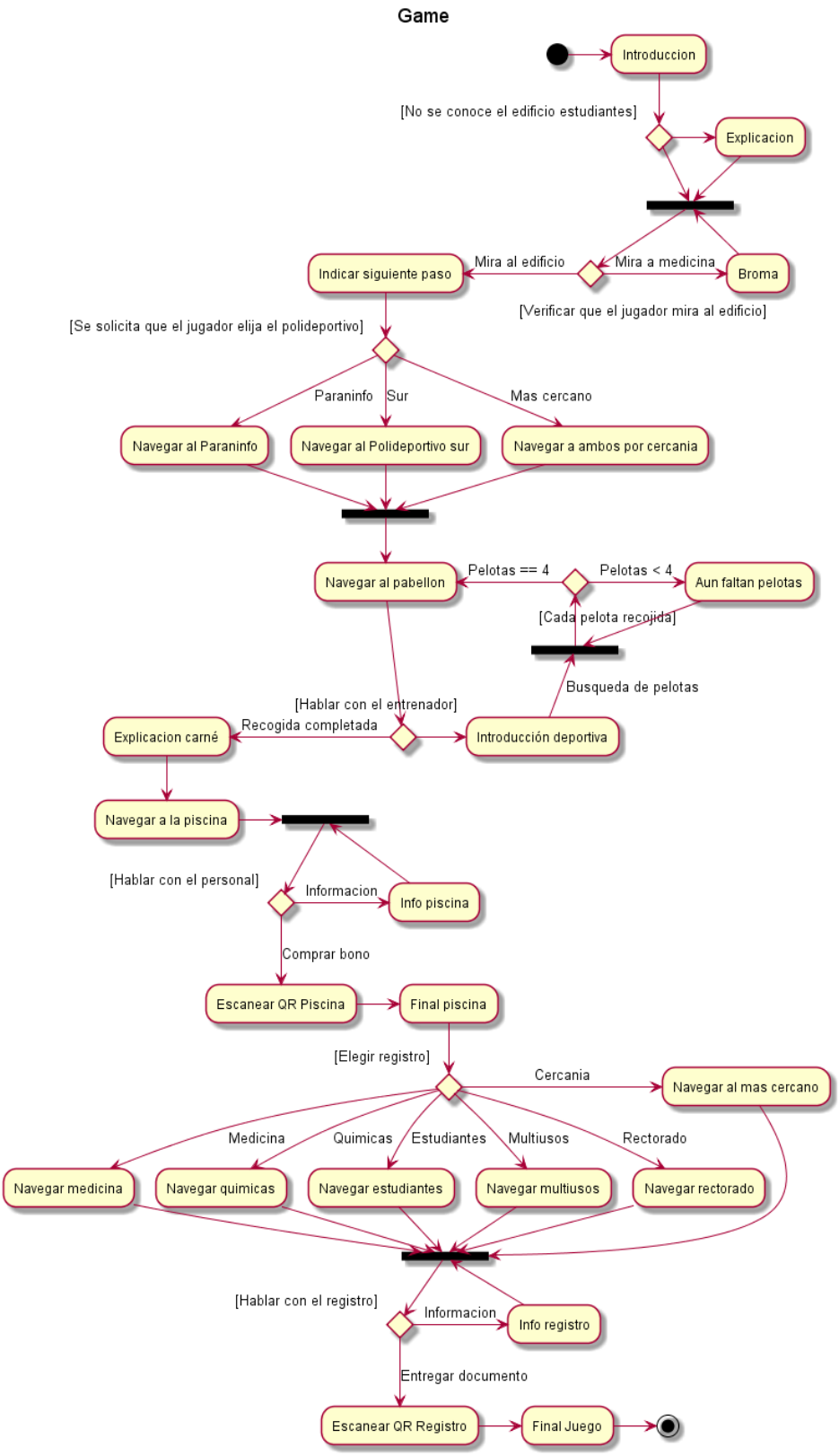

Fig. 7.1 Diagrama de actividades de la secuencia de juego

utilizando ShowOnlyOnRange para que el jugador tenga que buscarlas por el polideportivo y así conocer los distintos campos deportivos. La Fig. 7.2 muestra las diferentes pelotas del escenario del paraninfo, así como la ubicación del pabellón. Cada vez que se recoja una pelota se muestra un diálogo que anuncia la cuenta de pelotas restantes, salvo que se hayan recogido todas, en cuyo caso se navegará hasta el pabellón. Alcanzado el pabellón el entrenador entonces mostrará un diálogo explicando el carné anual y, con ello, introducirá a la piscina. Tras ello, se inicia una navegación hacia la piscina utilizando diversos pasos para ello. En función del polideportivo se irá hacia la piscina más cercana.

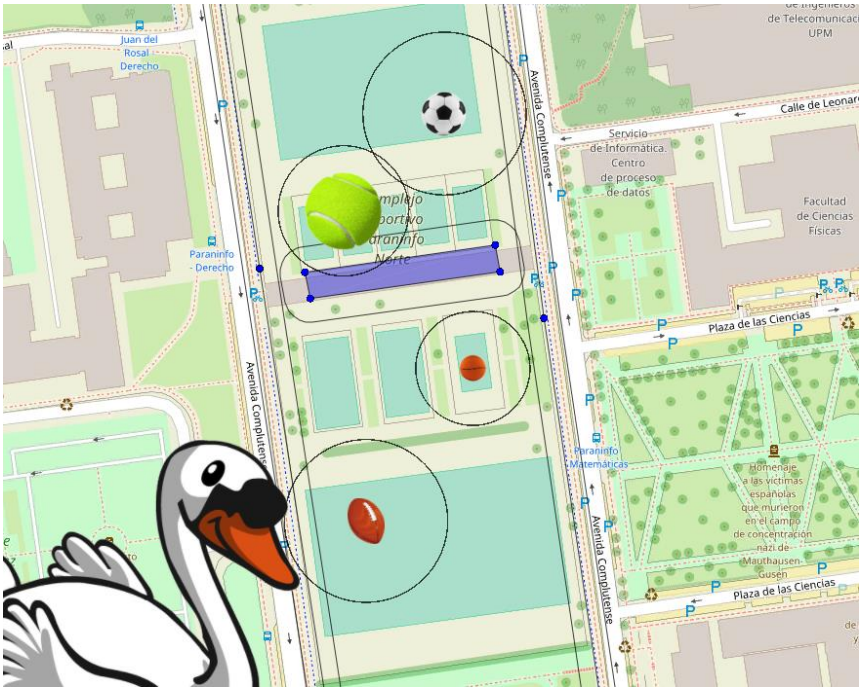

Fig. 7.2 Escena de juego del polideportivo del Paraninfo.

La tercera fase comienza con la navegación hacia la piscina utilizando la MapScene del polideportivo anterior. Una vez se finaliza la navegación, para garantizar que se encuentra la entrada de la piscina se utiliza un GeoElement de tipo punto que, en su acción Enter lanza la escena de la piscina. Dado que se andará por interiores, en lugar de utilizar un lanzamiento con zona se utiliza el lanzamiento clásico. Una vez en la escena, aparece un nuevo personaje Personal que te indicará el funcionamiento de la piscina en su conversación y te permitirá entregar un documento utilizando un QR como comprobación. Para ello, al seleccionar la acción adecuada se utiliza un efecto OpenQRPrompt al que se pasa un QR preparado para la piscina. Al

detectar el QR, en su interior contiene un efecto que mostrará un diálogo de finalización de la piscina y cargará la siguiente MapScene para la búsqueda de registro.

La cuarta y última fase comienza con un diálogo en la MapScene de registros en la que Ayudante te permitirá seleccionar el registro al que desees ir. Se puede elegir entre los cinco registros o, si se prefiere, se puede elegir navegar al más cercano. Para ello, en la escena existen cinco GeoElements representando los cinco registros principales del Campus de Moncloa. Cuando se alcance uno de ellos se lanza una escena genérica para todos ellos en los que aparece un nuevo personaje Secretaria. Ésta ofrece al jugador dos opciones, elegir información o entregar un documento en el registro. En el caso de escoger la segunda opción se abre el escáner de QR con el efecto OpenQRPrompt que permite escanear un QR junto a la taquilla del registro. Al hacerlo, se lanza un diálogo y finalmente se carga una escena en la que se indica al jugador que se ha llegado al final del juego.

Durante todas las fases, Ayudante permitirá que el jugador conozca en una frase el objetivo actual, exceptuando el caso de la búsqueda de pelotas, que mostrará además la información de las pelotas que restan de la recogida.

El juego además incluye una serie de hitos que podrán ser temporizados. Estos hitos son: la búsqueda del edificio de estudiantes, la llegada al pabellón deportivo, la completitud de la tarea de la búsqueda de pelotas, la llegada a la piscina y la llegada al registro. Para ello, se añadieron cinco completables que se complementarán con las analíticas de RAGE.

A modo de resumen, el juego se compone de un total de 7 escenas de uAdventure, una inicial, dos para pabellones, dos para piscinas, una para registro genérica y una final, y cuatro escenas de mapa, entre las que se encuentran la inicial, las de los dos polideportivos y la que permite navegar a cualquier registro. También incluye 4 personajes y 4 objetos que aparecen en las diferentes escenas y 15 conversaciones diferentes para todos los casos. Por otro lado, para las escenas de mapas existen un total de 16 elementos geoposicionados para todos los edificios, puntos y caminos utilizados en las escenas de mapa. El juego se desarrolló en aproximadamente 3~4 días que se extendió a una semana para *testing*.

7.2. Metodología del experimento

El experimento consiste en poner en práctica el juego y el conocimiento adquirido por los jugadores.

Para la puesta en práctica se da a los estudiantes acceso a la aplicación para Android que deberán instalar antes de comenzar a jugar. Además, se da una serie de instrucciones de cómo configurar los terminales, en los cuales no deben estar encendidas las redes WiFi, pero sí tener acceso a datos para la descarga de mapas y envío de trazas. Una vez abierta la aplicación del juego esta no se debe cerrar para mantener toda la sesión unificada. El juego se realizará con entre uno y dos grupos que corresponden a la cantidad de polideportivos para los que se ha preparado el juego. Previamente se instalarán los códigos QR en las piscinas y registros para que se puedan utilizar llegado el momento. Al finalizar, los jugadores deberán regresar al metro de Ciudad Universitaria para reunirse y poder realizar el cierre del evento.

Para la obtención de resultados se realizará una encuesta antes y después del juego que se contrastarán. Para ello se utilizará el software LimeSurvey generando una lista de códigos anónimos que se entregarán a cada jugador. De esa forma se podrá contrastar la efectividad del evento en el aprendizaje de la tarea. Por otro lado, el envío de trazas para la generación de Learning Analytics estará activado durante toda la sesión, permitiendo revisar al final de la sesión los patrones de comportamiento de los jugadores.

7.3. Realización del experimento

Para la realización del experimento se utilizó un pequeño conjunto de jugadores para la primera prueba en la que participaron cuatro personas. A todas ellas se les hizo la encuesta antes y después en la que se preguntaban los conocimientos acerca de las pistas, el precio y ventajas del carné anual deportivo, los precios y tipos de piscinas y los registros, así como su funcionamiento. Para realizar las encuestas se utilizó el software de LimeSurvey utilizando un método de anonimización de los participantes, pero manteniendo la correlación entre las dos encuestas. En el Apéndice A - de la memoria pueden encontrarse los resultados de las encuestas que se analizarán a continuación.

Para la sesión de juego, se organizó un pequeño evento el día 2 de febrero por la tarde que comenzó en el metro de Ciudad Universitaria. Se le proporcionó a los participantes acceso a la aplicación para Android que debía ser instalada antes de comenzar y se explicó la forma de juego. Al ser un número tan reducido de jugadores voluntarios sólo se formó un equipo para la realización del juego. El equipo comenzó el juego y decidió tomar como zona de juego el polideportivo del sur, por estar más cerca del metro.

El primer completable, la búsqueda del edificio de estudiantes sólo tomó unos segundos en realizarse ya que muchos de los jugadores lo pudieron realizar casi instantáneamente.

El segundo completable, llegar hasta el pabellón del polideportivo, tomó algo más de tiempo, durante el cual los jugadores mencionaron que el juego debería de haber explotado dicho recorrido para alguna tarea, ya que resultó especialmente aburrido al tener que llevar el móvil siempre como guía. En total, este paso se completó en 13 minutos, debido al tráfico.

El tercer completable, la búsqueda de pelotas, resultó ser la parte más divertida del evento, durante el cual los jugadores salieron del pabellón y buscaron las pelotas en el polideportivo. La primera en aparecer fue la pelota de fútbol, a los dos minutos de comenzar. Seguido a ella, se descendió a buscar la pelota de tenis que se encontró en aproximadamente otros dos minutos, junto a las canchas de tenis. La siguiente pelota, fue la de baloncesto, que se encontró cuatro minutos después y finalmente se encontró la pelota más lejana, la del campo de rugby en unos cuatro minutos más, dado que hubo que acceder hasta casi el interior del campo para encontrarla. En total, el tercer completable tomó 18 minutos.

El cuarto completable, la llegada a la piscina fue especialmente sencillo dado que en el polideportivo del sur la piscina se encuentra colindante. Los jugadores salieron juntos del campo y llegaron a la piscina, dónde escanearon el QR. Todos los jugadores consultaron la información de la piscina, en lugar de ir directamente a escanear el QR. Este completable tomó a los jugadores 6 minutos.

El quinto y último completable, llegar al registro, tomó algo más de tiempo que la llegada a la piscina. Los jugadores escogieron acceder al más cercano y el juego les indicó cómo ir. Sin embargo, la flecha de navegación no ayudó especialmente a los jugadores, dado que les indicó caminar hacia una calle donde la ruta era significativamente más larga. Al llegar al rectorado sólo algunos de los jugadores consultaron la información, pero todos escanearon correctamente el código QR. Este paso tomó unos 14 minutos.

En total, la sesión de juego duró 51 minutos, a lo que, sumando el tiempo de preparación y el tiempo de vuelta hasta el metro sumó aproximadamente 1 hora y 20 minutos. Habiéndose completado ambas encuestas al finalizar.

En general todos los participantes manifestaron que la experiencia había sido buena, divertida e instructiva y que habían disfrutado especialmente con la recogida de pelotas que había recordado al estilo de juego de Pokémon GO.

7.4. Análisis de los resultados de encuestas y analíticas

Tras el evento se analizaron los resultados obtenidos de los dos puntos de colecta de información: encuestas y LAs. Si bien ambos resultados son interesantes por separado, no se pudieron conectar las encuestas con la experiencia de juego de cada jugador, impidiendo asociar los fallos en las encuestas a conductas específicas. Además, dado que todos los jugadores realizaron el mismo patrón de juego no era posible detectar anomalías en comportamientos.

En el lado de las encuestas, la tabla 1 muestra las tasas de éxito en las respuestas antes y después del evento. Como se puede observar, en la primera encuesta existe una tasa de aciertos del 29%, siendo bastante baja. Al finalizar el evento, los jugadores demostraron haber mejorado los conocimientos llegando a una cifra del 86% de aciertos. En las preguntas que respondieron físicamente al finalizar el evento se anotó una puntuación media de 7.3 en la experiencia. Además, se destacó positivamente el cambio automático de escenas por GPS, siendo un hecho que había mejorado la experiencia inmersiva del juego. Por otro lado, todos ellos estaban de acuerdo en que las partes de tránsito eran muy aburridas y que faltaban elementos para poder competir con los demás jugadores.

En cuanto a los datos de LA analizados en RAGE se destacan las siguientes gráficas. La primera gráfica de la Fig. 7.3 muestra los tiempos en los que se realizaron los completables, coincidiendo con los tiempos tomados durante la ejecución del juego. Los completables son, desde izquierda a derecha: edificio de estudiantes, llegada al pabellón, búsqueda de pelotas, llegada a la piscina y llegada al registro.

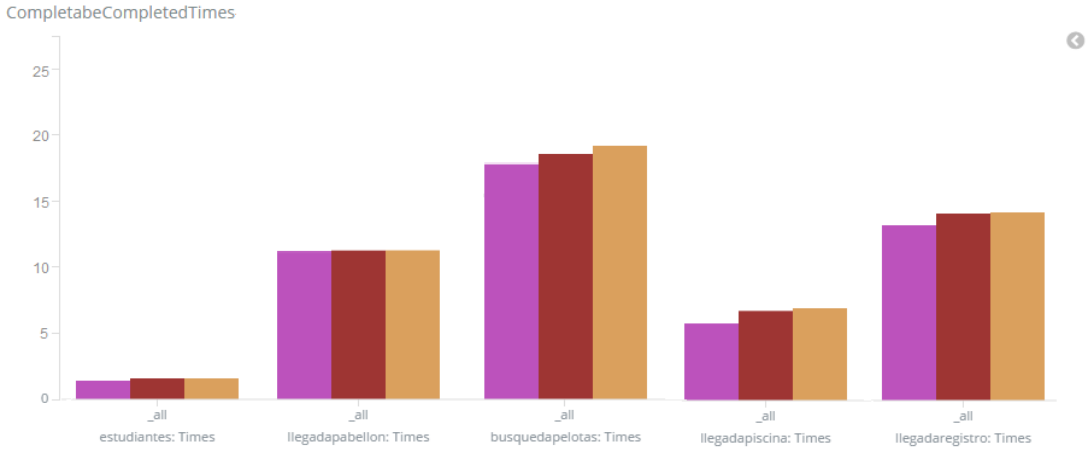

Fig. 7.3 Tiempo de realización de los diferentes completables.

La segunda gráfica de la Fig. 7.4, se observa cuantas veces se ha fallado en mirar correctamente al edificio de estudiantes como resultado de la acción *lookAt* del primer completable. En ella se muestra que sólo uno de los cuatro jugadores (25%) miró hacia medicina fallando la primera prueba y provocando el evento de broma.

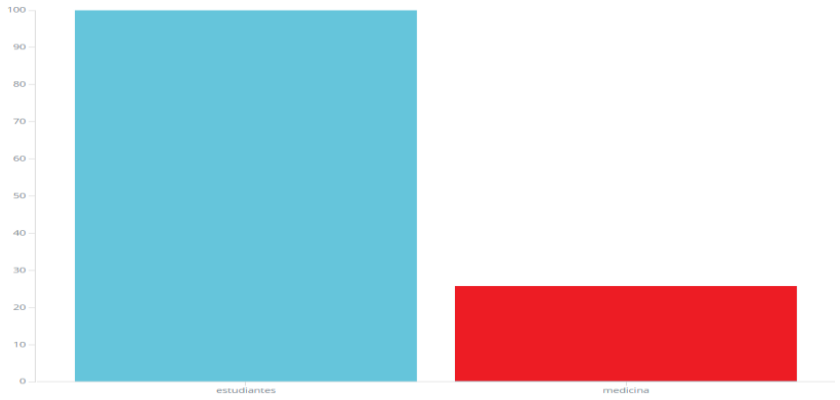

Fig. 7.4 Gráfica comparativa de la acción geoposicionada "*lookAt*".

La tercera gráfica de la Fig. 7.5 muestra las diferentes acciones geoposicionadas de "*enter*" que se realizaron a lo largo del evento, destacando que solo se accedió al polideportivo del sur de los dos polideportivos posibles y al rectorado de todos los registros posibles.

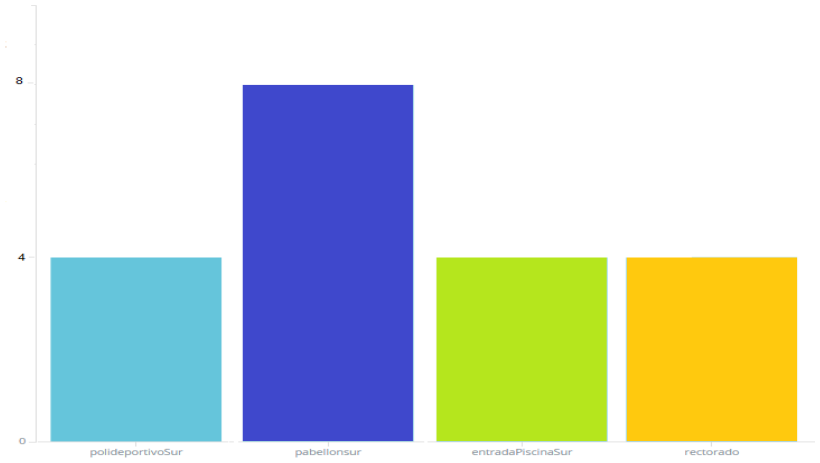

Fig. 7.5 Gráfica comparativa de la acción geoposicionada "*enter*".

La cuarta y última gráfica de barras de la Fig. 7.6 muestra la comparativa en la interacción con los distintos personajes que inicia el usuario. Como se puede observar el elemento más interactuado fue el cisne para consultar el evento actual, el segundo el entrenador para la búsqueda de pelotas. Dado que se interactuó con el ayudante de la piscina 8 veces (para

un total de 4 jugadores) se puede asumir que el 100% de los jugadores preguntó la información en la escena de la piscina, mientras que, dado que la secretaria del registro sólo fue interactuada 7 veces, se puede deducir que sólo un 75% preguntaron la información en el registro.

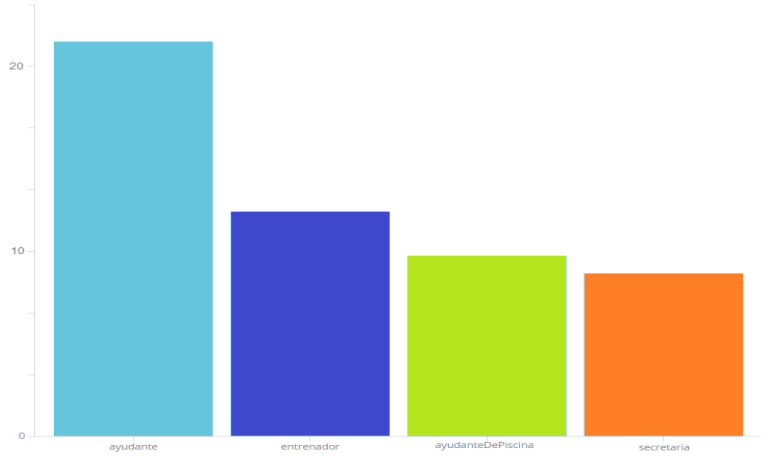

Fig. 7.6 Interacción con los personajes del juego.

Con respecto a los mapas de calor pueden observarse de manera conjunta dado que todos los jugadores realizaron la misma ruta en la Fig. 7.7. En él puede verse el avance desde el metro hasta el polideportivo del sur, se realizó un tránsito en la búsqueda de pelotas en su interior, se caminó siguiendo la ruta propuesta hasta la piscina y finalmente se llegó al paraninfo caminando junto a la autopista.

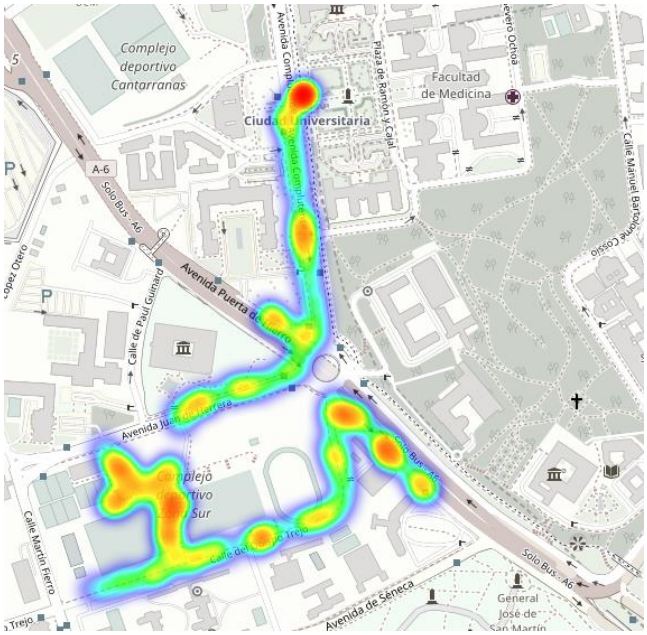

Fig. 7.7 Mapa de calor de la realización del experimento

Por otro lado, el mapa de calor de la Fig. 7.8 muestra, los puntos en los cuales los se realizaron acciones geoposicionadas como la entrada (*enter*) y el mirado (*lookAt*) a elementos del mapa.

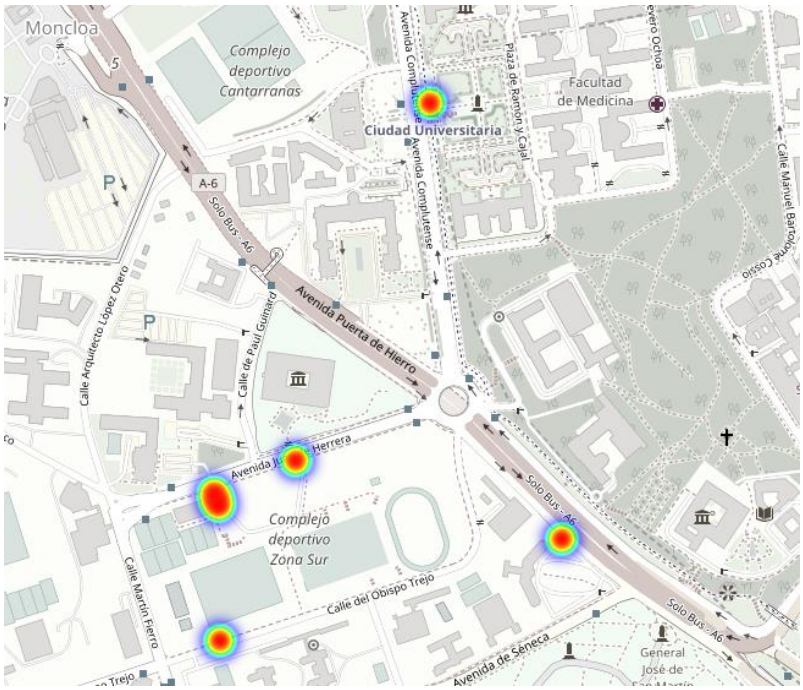

Fig. 7.8 Mapa de calor de la realización de acciones geoposicionadas.

Capítulo 8. Conclusiones y trabajo futuro

Este proyecto tenía como objetivo fundamental introducir en uAdventure características y mecánicas que permitan realizar juegos geoposicionados. Con todo el desarrollo asociado y la reestructuración necesaria, uAdventure ha dado un paso más hacia una mejor integración con Unity y una mejor explotación de las posibilidades de los dispositivos móviles. Además, se ha convertido en una herramienta más depurada y apta tanto para profesores como para desarrolladores. Además del propio desarrollo en la maduración y ampliación de la plataforma uAdventure se ha creado un juego de ejemplo y se han realizado incluso unas pequeñas pruebas con usuarios. No obstante, hay que ser realista y aunque uAdventure es ahora un programa funcional todavía tiene que recorrer un largo camino hasta que pueda considerarse un proyecto maduro y completamente estable.

Al comienzo del proyecto, uAdventure se encontraba en una fase prematura en la cual sus dos partes, editor y ejecutor se encontraban funcionando juntos, pero con claros problemas heredados de haber sido desarrollados de manera independiente y sin haberse realizado pruebas finales con un número significativo de usuarios. En la fase inicial del proyecto el editor tenía claros fallos de usabilidad y la cohesión de los elementos internos era demasiado alta hasta el punto de que para poder realizar cualquier mejora era necesario ser experto del sistema. Habiendo estudiado todos los fallos y teniendo en mente el objetivo del proyecto se tomó la decisión de realizar una reestructuración del código mejorando la arquitectura software para obtener como resultado una herramienta no sólo más robusta y adaptada a Unity, sino que fuera capaz de ser extendida para poder sacar provecho de la flexibilidad que aporta el nuevo motor.

A continuación, se presentan los principales aspectos sobre los diferentes puntos de extensibilidad que ha sido necesario introducir. Posteriormente se expondrán las soluciones aplicadas sobre esta arquitectura extensible para cubrir los objetivos de permitir usar geoposicionamiento y códigos QR en los juegos. Las mejoras en extensibilidad son:

- **Modelo**: A través de la clase Chapter ahora es posible almacenar nuevos elementos del modelo de manera transparente e independiente.
- **Serializadores y deserializadores**: Estos procesos se pasan a realizar a través de las utilidades para serializado y deserializado que permiten conectar con los distintos *writers* y *parsers* de forma desacoplada.

- **Editor principal**: Ahora funciona usando de extensiones con autodescubrimiento que trata a todas las ventanas de uAdventure por igual. Esto implicó modificar todos los editores para adaptarse al nuevo sistema y permitió además corregir errores de diseño que incrementaban el acoplamiento de los elementos del editor. Todas las ventanas de uAdventure son independientes al editor y actúan como extensiones del mismo.
- **Elementos principales**: Para conectar los nuevos elementos del modelo con la ejecución la nueva arquitectura de *targets* para el capítulo se permite que no sólo las escenas sean elementos principales, sino también cualquier elemento que implemente una interfaz adecuada.
- **Efectos**: Los nuevos efectos personalizados, permiten la integración de mecánicas en cualquier elemento principal, configurándolos utilizando editores y ejecutores individuales.

Todas estas partes extensibles se centran en el uso de atributos de C# y el **autodescubrimiento de los *plugins*** creando un sistema más extensible y transparente que permita que incluso programadores no expertos puedan implementar sus propias mecánicas o minijuegos sobre uAdventure.

Partiendo de estos elementos que buscan mejorar la robustez y la extensibilidad, el primer caso de extensibilidad real de esta nueva arquitectura de uAdventure ha sido el propio desarrollo de la inclusión de las características de geoposicionamiento y códigos QR.

Para cumplir los objetivos de geoposicionamiento, primero, se han desarrollado elementos de Unity capaces de proporcionar un adecuado soporte para mapas. Estos son los mapas en ejecución y en edición.

- **Representación de mapas en ejecución:** Para el lado de la **ejecución** se ha utilizado **MapzenGO** parda simplificar el desarrollo. Pese a que pueda ser la parte más técnica del proyecto, tomar como referencia un proyecto funcional como MapzenGO permitió una toma de contacto inicial con los servicios de geoposicionamiento como los TMS y las diferentes conversiones en unidades a métricas terrestres. Tras su modificación expuesta en el punto 5.1.1, el nuevo **sistema de *plugins*** no sólo permitirían aprovechar todo lo que

MapzenGO daba, sino también **crear la propia jerarquía de *plugins*** para elementos **de uAdventure**.

- **Representación de mapas en editor:** El elemento **GUIMap** expuesto en el punto 5.1.2 permite representar tanto mapas basados en OSM con soporte a **geometrías basadas en el estándar GML**, así como para **recursos gráficos** con diversos tipos de posicionamiento. Estos tipos de posicionamiento se alinean con los requisitos de funcionalidades del proyecto, siendo **posicionamiento global, radial y en pantalla**. Para mejorar la calidad de uso **PlaceSearcher** integra un sistema de geolocalización inversa (*reverse geocoding*) de posiciones basado en direcciones.

Tomando los mapas como base se han desarrollado los objetivos propuestos sobre geoposicionamiento, que consisten en la integración de elementos geoposicionados, escenas de mapas y un sistema de navegación

- **Elementos geoposicionados:** Los elementos geoposicionados **GeoElements** son la parte del modelo de uAdventure que representa zonas y ubicaciones. Se componen de una geometría sobre la cual el jugador puede interactuar. Para su manipulación se ha añadido una extensión que permite modificar la geometría a través de un mapa y permite el manejo de acciones geoposicionadas de forma sencilla. Para su visualización en tiempo de ejecución, se ha creado un *plugin* para MapzenGO que traduce las geometrías en mallas de Unity que son capaces de detectar como entradas o interacciones del juego las acciones geoposicionadas.
- **Escena de mapas:** Las escenas de mapas **MapScene** pueden incluir referencias a elementos geoposicionados, **GeoElements**, como elementos de uAdventure. Para su manipulación, el *plugin* de extensión de editor permite manipular los elementos que componen el mapa. Dado que el posicionamiento en GUIMap se debe traducir en un posicionamiento en MapzenGO los **descriptores** son los encargados de mantener las configuraciones independientes al tipo de representación del elemento, permitiendo crear manejadores especializados a cada situación de forma extensible. Para su ejecución, las MapScenes son elementos principales *target* del capítulo y pueden seleccionarse de diversas formas. En ejecución utilizan un mapa de MapzenGO que, además de incorporar

el *plugin* para geometrías incorpora también un *plugin* que aplica sobre los elementos de uAdventure los posicionadores utilizando *wrappers* conservando la interacción original.

- **Sistema de navegación:** a través de efectos es posible controlar la navegación, permitiendo iniciarla con diversos tipos de navegación y control.

Para la parte de **códigos QR**, se integraron librerías para la lectura y creación de QRs que se conectan con uAdventure mediante *plugins* para el editor y efectos.

- **Editor de QR**: Como extensión del editor se permite la creación de un QR, la incorporación de contenidos y la asignación de efectos como resultado de su identificación. Para que sea más sencillo de utilizar, el propio editor incluye botones para guardar la imagen o incluso imprimirla directamente para evitar errores o confusiones.
- **Escáner de QR en ejecución**: En la parte de ejecución, el **efecto de apertura de escáner de QR** permite integrar dicha mecánica con cualquier juego.

En la parte de **analíticas de aprendizaje** ha habido que hacer diversas tareas para poder tener en cuenta estos nuevos aspectos de geoposicionamiento. Por un lado, el perfil de aplicación de xAPI para juegos serios que se ha desarrollado en el grupo e-UCM en colaboración con ADL no lo contemplaba, de modo se ha extendido dicho perfil integrando un nuevo verbo *moved* con el objeto de poder comunicar la movilidad espacial del jugador. Además, se crearon nuevas extensiones para geoposicionamiento que se integran en algunas de las trazas como el acceso a zonas geoposicionadas (entrar, salir, mirar, etc.) o el descubrimiento de elementos. Con ello, se pudieron generar nuevos tipos de trazas y, a partir de ellas, crear nuevos análisis y visualizaciones de la información de seguimiento como, por ejemplo, las basadas en mapas de calor para visualizar el movimiento del usuario.

Como ya hemos mencionado no sólo se ha mejorado el sistema y se ha incorporado el soporte al geoposicionamiento, sino que también se ha creado un juego como caso de estudio. El caso de prueba del punto 4.1 que se expuesto en el capítulo 6 y con el que se han obtenido las siguientes conclusiones. El juego se ha desarrollado en menos de una semana (incluso con la resolución de problemas en el software de soporte), demostrando la eficacia de las herramientas y se ha realizado una evaluación formativa que contó con una participación en su ejecución de cuatro investigadores del grupo e-UCM. Los probadores han mostrado una buena aceptación del

juego que recordó a Pokémon GO en algunos de sus aspectos y que arrojó resultados de encuestas que demostraban una mejora de los conocimientos de los jugadores sobre las instalaciones deportivas y registros de la UCM.

En cuanto a mi opinión más personal sobre el proyecto he tenido una experiencia enriquecedora y diferente respecto a otros desarrollos que he realizado. En general, he podido experimentar trabajar con arquitecturas y código previos de gran tamaño sobre el que es necesario realizar tanto un análisis a bajo nivel como una reestructuración a alto nivel para lograr mejoras. Además, en este proyecto he podido trabajar con muchas librerías y estándares diferentes, siendo un proyecto muy instructivo de cara a la integración de nuevos elementos en ecosistemas existentes. Gracias a mis aportaciones espero que futuros desarrolladores no tengan que realizar análisis tan profundos de la arquitectura de uAdventure para poder integrar sus mecánicas en el sistema, pudiendo beneficiarse como he hecho yo posteriormente de las nuevas mejoras de extensibilidad.

Como conclusión final, el sistema de uAdventure ha dado un gran salto hacia convertirse en su objetivo, ser el sucesor de eAdventure en Unity y aprovechar todas las capacidades que este cambio le ofrece. Para autores, uAdventure supone una simplificación enorme en la creación de juegos que ahora pueden disfrutar de nuevas mecánicas como es el geoposicionamiento en juegos. Para desarrolladores, que pueden ser expertos en Unity, uAdventure supone una interesante herramienta extensible de donde partir para desarrollar juegos o incluso nuevas extensiones. Si bien uAdventure tiene mucho camino por recorrer, se está adaptando adecuadamente a las necesidades de los nuevos tiempos, llegando a móviles y a motores de videojuegos de última generación y evolucionando para un nuevo tipo de evaluación orientado al seguimiento mediante analíticas del aprendizaje.

8.1. Trabajo futuro

En este proyecto se han cubierto suficientemente todos los elementos planteados en su inicio. Sin embargo, como en todo trabajo de desarrollo además de mejorar la calidad como producto de sistema creado existen diversas oportunidades de expansión y mejora. La siguiente lista expone algunas ideas sobre cómo se podría continuar este trabajo.

Sobre uAdventure:

- Mejorar el sistema de persistencia del estado, para poder cerrar y abrir el juego notificando a todos los elementos en ejecución dicho evento de modo que se puedan restaurar posteriormente.
- Mejorar la integración de los editores en el inspector de Unity, utilizando una ventana de escena propia y una ventana para la jerarquía.
- Unificar el sistema de serialización con el sistema de *assets* de Unity.

Sobre la extensibilidad:

- Añadir formas de extender elementos de uAdventure ya existentes, como las escenas.
- Añadir formas de extender de forma global, permitiendo la incorporación automática de *managers* como el GPSManager, el TimerManager, etc. Siendo estos elementos no principales de segundo plano.
- Añadir formas de extender las configuraciones y atributos globales del proyecto.
- Analizar la posibilidad de cachear la extensibilidad en elementos principales y efectos en ejecución para mejorar el rendimiento y garantizar el soporte en iOS.
- Incorporar un sistema para habilitar y deshabilitar los *plugins*, en lugar de realizarse de manera global.
- Permitir relacionar los *plugins* relacionados a través de paquetes de *plugin*.

Sobre las escenas de mapas:

- En edición, mejorar el sistema de cacheado en ficheros dado que es un requisito de OSM para el uso de su API que por ahora no se está haciendo.
- Mejorar los zooms de los mapas. Permitir hacer zoom hacia el ratón y no hacia el centro del elemento GUIMap. Asimismo, realizar éste de forma progresiva.
- Permitir habilitar y deshabilitar la visualización de elementos en el editor de escenas de mapas para mejorar su uso.
- Permitir configurar diferentes tipos de cámara y configuraciones de cámara en las escenas geoposicionadas, a elegir entre los tipos Aérea2D, Ortográfica3D y Perspectiva3D.

- Incorporar movilidad a la cámara en la escena en ejecución y control del zoom.
- Permitir configurar MapzenGO para habilitar y deshabilitar los diferentes tipos de plugins y su visualización personalizada.
- Permitir configurar el estilo del jugador en ejecución sobre el mapa, pudiendo utilizarse el mismo personaje que se utiliza en los juegos en tercera persona.
- Mejorar la navegación para utilizar algún servicio de rutas online, además del servicio de caminos, que mejoraría el tiempo de creación del juego.
- Utilizar el acelerómetro y giroscopio para detectar la dirección hacia la que mira el jugador, para poder detectar si mira al cielo o al suelo, por ejemplo.
- Mejorar la diversión durante los trayectos largos en el mapa añadiendo, por ejemplo, mecánicas de generación aleatoria de objetos que otorguen puntuación.
- Añadir vistas de realidad aumentada que se incorporen como nuevo tipo de escena y permitan interactuar con los elementos cercanos.

Sobre las analíticas:

- Modificar la ampliación del perfil de juegos serios para incluir geolocalización. Solicitar a ADL la incorporación del evento moved en dicho perfil.
- Mejorar e integrar la orientación del jugador.
- Detectar información de acelerómetro y transmitirla también, así como detectar comportamientos en base al acelerómetro.
- Mejorar las gráficas y la dependencia del posicionamiento con los resultados.

Bien aplicado, este proyecto podría llegar a generar juegos similares a PokémonGO con los beneficios educativos que tiene la calidad visual y la interacción rica.

Esta lista podría parecer muy amplia y difícil de cubrir en un futuro cercano, no obstante, algunos de estos elementos ya se han empezado a cubrir en otros trabajos relacionados. Por ejemplo, actualmente ya hay otro grupo trabajando en un sistema de minijuegos para uAdventure y otro programador está analizando la posibilidad de integrar aspectos básicos de realidad aumentada en el editor.

Capítulo 9. Conclusions and future work

This project had as fundamental objective to introduce inside uAdventure characteristics and mechanics that will allow the system to create geopositioned games. With all the development associated and the necessary restructuration, uAdventure has made one more step to have a better integration with Unity and a better exploration of the possibilities of mobile devices. Furthermore, it has become a more debugged tool, more prepared to teachers and developers. In addition of the development in the mature of the system and the extensions of the uAdventure platform, an example game has been developed and even a very small set of tests with real users. Even so, being realistic and even if uAdventure is now a functional program it has a very long way until can be considered a mature and completely stable project.

At the beginning of the project, uAdventure was in a premature stage in which its two parts, editor and executor were working together but with clear problems inherited of have being developed in a separated way and without have it tested properly with a significant subset of users. In the initial stage of this project the editor had clear usability errors and a very cohesive internal design, till the point that any developer that could have wanted to modify the code needed to be an expert on the system. Having studied this errors and having in mind the objective of this project, we took the decision of make a restructuration of the code, improving the software architecture to obtain as a result a tool not only more robust and adapted to Unity, but capable to be extended to take advantage of the flexibility of the new engine.

Bellow there are presented the main points of extensibility that have been implemented in the system. After that, solutions applied to this architecture to cover the geopositioning and QR code games will be exposed. These extensibility points are:

- **Model**: By using the Chapter now it is possible to store new kind of elements in a non-restrictive and extensible way.
- **Serialization and deserialization**: These processes are now being managed by utilities that act as a middleware connecting with all the possible writers and parsers in a non-coupled way.
- **Main Editor**: Now it Works using extensions with auto discovering that uses the same interface will all the possible uAdventure windows. This implied modifying all the editors to adapt to the new system and also correct the design errors that were increasing

coupling in the editor window. All the uAdventure windows are now independent to the editor and act as extensions to it.

- **Main Elements**: In order to connect the new elements in the model with the runtime part, the new architecture based on targets for the chapter allow not only to scenes to be main elements, but also any element that implement the right interface.
- **Efectos**: The new custom effects allow the integration of new mechanics within the execution of any main element, configuring it by using custom editors and running it using individual executors.

All these extensible parts are focused in the use of C# attributes and the **autodiscovering of the plugins** creating a system more extensible and transparent that allows even non-expert programmers to integrate their own mechanics or minigames inside uAdventure.

Starting up from this extensibility elements, the first case of real extensibility based on this architecture are the own development of geopositioning features and QR codes.

To fulfill the geopositioning objectives, first there have been developed elements for Unity able to provide support for maps. These are, maps for runtime and for edition time.

- **Runtime maps:** On the runtime side MapzenGO has been used to simplify the development. Even if this is the most technical part of the project, by using a working project allowed the project to take a soft initiation into TMS and GPS features. After the modifications to MapzenGO exposed in the point 5.1.1, the new plugin system for MapzenGO will not only allow to use all the native features, but also to integrate our own plugin hierarchy for uAdventure elements.
- **Editor maps:** As there are no implementations for maps in the editor of Unity, the element GUIMap exposed in the point 5.1.2 allows the system to represent maps based on OSM tile service with full support to include GML geometries and graphical resources with three kinds of positioning. Those three are aligned to the functional requirements for this project and are global, radial and on-screen positioning. Also, to improve the user experience, PlaceSearcher integrates a reverse geocoding system allowing the users to search for places and obtain locations.

Using these two kind of maps as the start point there have been developed all the objectives for geopositioning that consist in the integration of geopositioned elements, map scenes and a navigation system.

- **Geopositioned elements:** The geopositioned elements GeoElements are the part of uAdventure model that represent zones and locations. These are composed by a geometry where the player can interact with. To manipulate them, an editor extension has been added to modify the geometry inside a map and to manage the possible geo actions that happen when the player interacts with it in a simple way. To visualize them in real-time, there has been developed a plugin for MapzenGO that translates geometries into Unity meshes able to detect all the possible geo actions.
- **Map scenes:** The map scenes MapScene can include references to both GeoElements and uAdventure native elements. To their management, the editor extension plugin for map scenes allows to add, remove and position elements into the map. To relate the position in the editor map with the runtime map, there are descriptors are in charge of maintain the configurations independently to the kind of representation, allowing to create unlimited kinds of positioning systems without having to modify the existing ones, nor the editor or desec systems. To execute them, the map scenes are main targets for the chapter being capable to be loaded from different ways. Finally, in order to position the uAdventure elements and apply the descriptors, in addition to the geometry plugin, there have been developed another plugin that wrap uAdventre elements to manage their position maintaining the interaction systems.
- **Navigation systems:** can be used by launching custom effects that can be configured to navigate by using order or closeness of the provided locations. Runtime, the navigation is presented with a green arrow indicating the direction.

To cover the QR codes part, libraries for reading and creation of QR codes were integrated with unity and connected with uAdventure by using plugins for the editor a new custom effects.

- **QR code editor**: As an extension editor, it allows the creation of QR codes and the assignment of contents and effects as the result of its identification. To be used in a more user-friendly way, the editor includes buttons to save or even to directly print the codes.

- **QR code runtime scanner**: For runtime, the QR prompt effect is a custom effect that will allow any effect to launch a QR scanner, making it possible to integrate the QR features in any kind of game.

For the learning analytics part, it has required many tasks to be able to include these new geopositioning aspects. On one hand, the application profile of xAPI for serious game developed by e-UCM in cooperation with ADL didn't contemplate movement as an independent event. This way, the profile has been extended to integrate a new verb moved with the purpose of being able to communicate player (or any actor) movement in space. Furthermore, there have been created new extensions for geopositioning that are integrated into existing traces like the access to geopositioned zones (enter, exit, look, etc.) or the discovery of elements in the map. With that, it was possible to generate new kind of traces and, from them, to create new kinds of analysis and visualizations of the tracking information, such as, the ones based on heat maps to visualize the user movement.

As it has been mentioned, this project hasn't only improved the system and developed support for geopositioning an QR but it also has developed a game as a study case. The game defined in point 4.1 and exposed in chapter 6 and has provided the following conclusions. The game has been developed in less than a week (even counting testing and bugfixing in real environments), demonstrating the effectiveness of the tools. The real setup of the game has been put into practice with a formative evaluation that counted with the participation in its execution of four e-UCM investigators. The testers have shown a good acceptation of the game that reminded to Pokémon GO in some of its aspect and that have been complemented with polls to test the learning quality, demonstrating a notable improvement in the knowledge of the participants of the sport facilities and UCM registry facilities.

In my personal opinion about this project I've had a very enriching experience and different compared to other developments I've made. In general, I've been able to experiment working inside big architecture, within a complex code, where I have to analyze the code from the low level to be able to restructure the code at high level, in order to achieve an overall improvement. Furthermore, I've been able to work with many different libraries and standards, being a very instructive project for the integration of new element into existing systems. Thanks to my work in this project I expect future developers to not have to deal with this low level

approach to code to be able to integrate their own mechanics into the system, being able to benefit as I did after I made the extensibility features.

As fot the final conclusion, the uAdventure system has made a big step towards becoming its objective, becoming the successor of eAdventure inside Unity and taking advantage of all the capabilities the change provides. For authors, uAdventure means an enormous simplification in the creation of games that now can use new mechanics as the geopositioning. For developers, that could be experts in Unity, uAdventure means an interesting and extensible tool where they can start to develop games or even new extensions. Even if uAdventure has a long run to perform, it is getting prepared well to the new needs, reaching mobile technologies and top tier videogame engines and evolving for a new kind of evaluation oriented to use learning analytics.

9.1. Future work

This project has covered enough all the objectives proposed in the beginning. However, as in every development work, in addition to improve the overall quality of the system as a product, there are different opportunities to expand it and improve it. The list bellow exposes some of the ideas the future work could start from.

For uAdventure:

- Improve the game state persistence system, to be able to open and close the game notifying all the elements runtime to be able to later restore them.
- Improve the integration of the editors into the Unity inspector, using a separated window to edit scenes and a brand new hierarchy for uAdventure references.
- Unify the serialization system with Unity asset system.

For the extensibillity:

- Add new ways to extend already existing uAdventure systems such as scenes.
- Add new ways to extend globally, allowing to include managers automatically such as the GPSManager or the time Manager, being those background elements.
- Add new ways to extend the configurations and global attributes of uAdventure.
- Analyze the chance of caching extensibility on runtime of main elements and custom effects to improve performance and guarantee it will work in iOS.

- Include a system to enable and disable the plugins instead of using them globally.
- Allow to package related plugins.

For map scenes:

- In edition time, improve the caching system for files since it is a requirement of OSM for its API usage that is not being filled.
- Improve the zoom in maps, allowing to make zoom towards the mouse instead of to the center of the GUIMap. Also, making it progressively.
- Allow to enable and disable element visualization in the editor to make it easier to use when there are too many elements present in the scene or overlapping elements.
- Allow configure different kinds of cameras and configurations for the camera in the map scenes, to be able to choose between Aereal2D, Ortographic3D and Persperctive3D.
- Include mobility to the camera and zoom control runtime.
- Allow to configure MapzenGO to enable and disable the different kinds of plugins and its custom visualization.
- Allow to configure the style of the character in the maps, giving the option to use the character used in third person uAdventure games.
- Improve the navigation to use a pathfinding online service.
- Use the accelerometer and the gyroscope to detect the direction the player is looking, for example, to know if the player is looking upwards or downwards.
- Improve the enjoyment during long navigation periods, adding, for example, mechanics to generate random objects that give score to the player.
- Add views of RA in a new kind of RA scene using uAdventure elements, enabling the interacting with them in this close ranged experience.

For the analytics:

- Modify the enlargement of the serious game profile including geolocation. Request to ADL the addition of the moved verb to that profile.
- Improve and integrate the orientation of the player.

- Detect and send information from the accelerometers, and if possible, detect behaviors of the player based on that information, such as standing, walking, etc.
- Improve the visualization and the geolocation dependency with the results.

Well applied, this project could end up being able to generate games very similar to Pokémon GO with the benefits for education that generate a better polished graphics and interaction.

This list of features could look very wide and difficult to cover in a close future. However, some of these elements have already been started to develop in related works. For instance, in this moment there is other group working on a minigame system for uAdventure and another programmer is analyzing the possibility to integrate basic aspects of augmented reality into the editor.

Referencias

[1] T. Susi, M. Johannesson, and P. Backlund, “Serious Games – An Overview,” *Elearning*, vol. 73, no. 10, p. 28, 2007.

[2] M. Ulicsak, “Games in Education: Serious Games,” *A Futur. Lit. Rev.*, p. 139, 2010.

[3] P. Moreno-Ger, D. Burgos, J. L. Sierra, and B. F. Manjón, “A Game-Based Adaptive Unit of Learning with IMS Learning Design and <e-Adventure>,” in *Creating New Learning Experiences on a Global Scale*, vol. 4753 LNCS, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 247–261.

[4] R. Van Eck, “Building Artificially Intelligent Learning Games,” in *Games and Simulations in Online Learning*, IGI Global, 2007, pp. 271–307.

[5] J. Torrente, A. del Blanco, E. J. Marchiori, P. Moreno-Ger, and B. Fernandez-Manjon, “<e-Adventure>: Introducing educational games in the learning process,” in *IEEE EDUCON 2010 Conference*, 2010, pp. 1121–1126.

[6] I. J. Pérez-Colado, “uAdventure: Desarrollo del intérprete y de un emulador de videojuegos de eAdventure sobre Unity3D,” Complutense University of Madrid, 2016.

[7] V. McKalin, “Augmented Reality vs . Virtual Reality : What are the differences and similarities ?,” *Tech Times*, vol. 5, no. 6, pp. 1–6, 2015.

[8] C. Magerkurth, A. D. Cheok, R. L. Mandryk, and T. Nilsen, “Pervasive games,” *Comput. Entertain.*, vol. 3, no. 3, p. 4, Jul. 2005.

[9] S. Thomas, “Pervasive learning games: Explorations of hybrid educational gamescapes,” *Simul. Gaming*, vol. 37, no. 1, pp. 41–55, Mar. 2006.

[10] B. Coulter and E. Klopfer, “Discovering Familiar Places: Learning through Mobile Place-Based Games,” *Games, Learn. Soc. Learn. Lead. Digit. Age.*, pp. 327–354, 2016.

[11] BEN GILBERT, “Pokémon Go has been downloaded over 500 million times,” *Business Insider*, 2016. [Online]. Available: http://www.businessinsider.sg/pokemon-go-500-million-downloads-2016-9/#IDjt8zdmC4K31Uzl.97.

[12] T. Althoff, R. W. White, and E. Horvitz, “Influence of Pokémon Go on Physical Activity: Study and Implications,” *J. Med. Internet Res.*, vol. 18, no. 12, p. e315, Dec. 2016.

[13] G. Kipper, “What Is Augmented Reality?,” in *Augmented Reality*, Elsevier, 2013, pp. 1–27.

[14] D. R. Berryman, “Augmented Reality: A Review,” *Med. Ref. Serv. Q.*, vol. 31, no. 2, pp. 212–218, Apr. 2012.

[15] D. Casey, “u-Learning = e-Learning + m-Learning,” in *World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education (ELEARN)*, 2010.

[16] P. M. Ger, “eAdventure: Serious games, assessment and interoperability,” in *2014 International Symposium on Computers in Education (SIIE)*, 2014, pp. 231–233.

[17] J. R. Cole and H. Foster, *Using Moodle*. 2008.

[18] Á. del Blanco, J. Torrente, I. Martínez-Ortiz, and B. Fernández-Manjón, “Análisis del Uso del Estándar SCORM para la Integración de Juegos Educativos,” *Ieee-Rita*, vol. 6, no. xxxx, pp. 118–127, 2011.

[19] D. Cotroneo, S. Orlando, and S. Russo, “Characterizing Aging Phenomena of the Java Virtual Machine,” in *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, 2007, pp. 127–136.

[20] G. McGraw and E. W. Felten, *Java Security: Hostile Applets, Holes&Amp;Antidotes*. New York, NY, USA: John Wiley & Sons, Inc., 1996.

[21] Á. Serrano-Laguna, D.-C. Rotaru, A. Calvo-Morata, J. Torrente, and B. Fernández-Manjón, “Creating Interactive Content in Android Devices: The Mokap Hackaton,” in *End-User Development SE - 31*, vol. 9083, 2015, pp. 287–290.

[22] J. Torrente *et al.*, “Introducing Mokap,” in *Proceedings of the 5th International Conference on Digital Health 2015 - DH ’15*, 2015, vol. 1, no. 1, pp. 17–24.

[23] P. Marszał, “Adding Unity3D an Authoring Layer for Non-Programmers,” Complutense University of Madrid, 2016.

[24] A. T. Korucu and A. Alkan, “Differences between m-learning (mobile learning) and e-learning, basic terminology and usage of m-learning in education,” *Procedia - Soc. Behav. Sci.*, vol. 15, pp. 1925–1930, 2011.

[25] I. J. Perez-Colado, V. M. Perez-Colado, I. Martínez-Ortiz, and B. Fernández-Manjón, “uAdventure: The eAdventure reboot,” in *IEEE Education Engineering EDUCON 2017 Conference (en prensa)*, 2017.

[26] M. Freire, Á. Serrano-Laguna, and B. Iglesias, *Game learning analytics: Learning analytics for serious games*. 2016.

[27] RAGE Project, “Realising an Applied Gaming Eco-system (RAGE),” *rageproject.eu*,

2016. [Online]. Available: http://rageproject.eu/.

[28] A. del Blanco, A. Serrano, M. Freire, I. Martinez-Ortiz, and B. Fernandez-Manjon, "E-Learning Standards and Learning Analytics," *2013 Ieee Glob. Eng. Educ. Conf.*, pp. 1255–1261, 2013.

[29] T. Lior, "Unity Attributes," *tallior.com*, 2014. [Online]. Available: http://www.tallior.com/unity-attributes/.

[30] J. Ruiz, Y. Li, and E. Lank, "User-defined motion gestures for mobile interaction," in *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*, 2011, p. 197.

[31] M. Bilandzic and M. Foth, "A review of locative media, mobile and embodied spatial interaction," *Int. J. Hum. Comput. Stud.*, vol. 70, no. 1, pp. 66–71, Jan. 2012.

[32] N. Savio and J. Braiterman, "Design Sketch : The Context of Mobile Interaction," *Mob. HCI*, pp. 284–286, 2007.

[33] R. F. Falconi, "Usability and game design : improving the MITAR Game Editor ; Improving the MITAR Game Editor ; Usability and game design : improving the Massachusetts Institute of Technology Augmented Reality Game Editor," *MIT*, 2010.

[34] DiscoveryAgents, "Calgary Parks and the Agents of Nature App," 2016.

[35] K.-M. Ricker, "Digitale Schnitzeljagd mit Actionbound - ein Quiz zum Thema Planeten für Smartphones erstellen," *Biol. 5-10*, vol. 10, pp. 18–21, 2015.

[36] T. Á. Martín-Nieto, "LAS STARTUPS ESPAÑOLAS QUE ESTARÁN EN EL MOBILE DE SHANGHAI," *cincodias.com*, 2016. [Online]. Available: http://cincodias.com/cincodias/2016/06/27/emprendedores/1467048294_049780.html.

[37] R. Crespo-Cepeda, M. El-Yamri, and J. M. Carrera-García, "CtOS Enabler," Complutense University of Madrid, 2015.

[38] Karl, "Post Fata Resurgo, una gymkhana histórica en El Escorial," *gremiodelassombras.com*, 2016. [Online]. Available: http://gremiodelassombras.com/noticias/post-fata-resurgo-una-gymkhana-historica-en-el-escorial/.

[39] R. G. Dromey, "A model for software product quality," *IEEE Trans. Softw. Eng.*, vol. 21, no. 2, pp. 146–162, 1995.

[40] International Organization For Standardization Iso, "ISO/IEC 9126-1," *Software Process:*

Improvement and Practice, vol. 2, no. 1. pp. 1–25, 2001.

[41] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software (Adobe Reader)*. Pearson Education, 1994.

[42] osgeo, "Tile Map Service Specification," 2012. [Online]. Available: http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification.

[43] osgeo, "EPSG:3857," 2010. [Online]. Available: http://wiki.openstreetmap.org/wiki/EPSG:3857.

[44] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub, "The GeoJSON Format," Aug. 2016.

[45] C. Portele, S. J. D. Cox, P. Daisey, R. Lake, and A. Whiteside, "OpenGIS® Geography Markup Language (GML) Encoding Standard," *OGC Implement. Specif.*, vol. 07–036, p. viii + 426, 2007.

[46] E. Berberich, M. Kerber, and M. Sagraloff, "An efficient algorithm for the stratification and triangulation of an algebraic surface," *Comput. Geom.*, vol. 43, no. 3, pp. 257–278, Apr. 2010.

[47] D. R. Finley, "Point-In-Polygon Algorithm — Determining Whether A Point Is Inside A Complex Polygon," *alienryderflex.com*, 2007. [Online]. Available: http://alienryderflex.com/polygon/.

[48] R. Mautz, "Indoor Positioning Technologies," *Inst. Geod. Photogramm. Dep. Civil, Environ. Geomat. Eng. ETH Zurich*, no. February 2012, p. 127, 2012.

[49] J. Brassil, "Improving Indoor Positioning Accuracy with Dense, Cooperating Beacons," *Procedia Comput. Sci.*, vol. 40, no. C, pp. 1–8, 2014.

[50] C. Yang and H. Shao, "WiFi-based indoor positioning," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 150–157, Mar. 2015.

[51] I. J. Pérez-Colado and V. M. Pérez-Colado, "Un conjunto de herramientas para Unity orientado al desarrollo de videojuegos de acción-aventura y estilo retro con gráficos isométricos 3D," Complutense University of Madrid, 2014.

Apéndice A -Respuestas de las encuestas

Respuestas antes de comenzar con el experimento

Respuesta a la encuesta	
Contraseña	x89
¿Cuáles son los principales polideportivos del Campus de Moncloa?	Polideportivo del Norte y Polideportivo del Sur
¿Qué tipos de campos hay en los polideportivos?	Fútbol, bádminton, basket y golf
¿Cuánto cuesta el carné anual deportivo?	350€ al año
¿Qué permite hacer el carné anual?	Permite acceder tanto al gimnasio como a la piscina todo el año
¿Qué hay que hacer para reservar una pista?	Poner una instancia de reserva de pista y pagar la pista
¿Qué piscinas hay en el Campus de Moncloa?	Una una cubierta y otra descubierta y ambas están en el Polideportivo de la Almudena
¿Qué formas hay de entrar en la piscina?	Comprando un pase individual o con el carné anual
¿Cuánto cuestan las piscinas?	3€ (2€ para los miembros de UCM) y 25€ (15€ para los miembros de la UCM)
¿Cuáles son los registros principales del Campus de Moncloa?	Todas las facultades tienen su propio registro
¿Es obligatorio realizar todas las convocatorias a través del registro?	No, es recomendable pero también puede usarse cualquier otro registro de la ciudad, así como correo certificado
¿En el registro tienen información sobre las convocatorias?	Pueden aconsejarte sobre la documentación, pero no conocen todas las convocatorias

Respuesta a la encuesta	
Contraseña	fdc
¿Cuáles son los principales polideportivos del Campus de Moncloa?	Polideportivo Complutense y Polideportivo Rectorado
¿Qué tipos de campos hay en los polideportivos?	Rugby, tenis, basket, jockey, golf y pista de hielo
¿Cuánto cuesta el carné anual deportivo?	350€ al año
¿Qué permite hacer el carné anual?	Permite acceder tanto al gimnasio como a la piscina todo el año
¿Qué hay que hacer para reservar una pista?	Reservar la pista y pagarla
¿Qué piscinas hay en el Campus de Moncloa?	
¿Qué formas hay de entrar en la piscina?	Comprando un pase individual o un bono de 10 baños
¿Cuánto cuestan las piscinas?	10€ (5€ para los miembros de UCM) y 80€ (40€ para los miembros de la UCM)
¿Cuáles son los registros principales del Campus de Moncloa?	Todas las facultades tienen su propio registro
¿Es obligatorio realizar todas las convocatorias a través del registro?	Sí, sin excepción dado que los registros de la UCM son los únicos que permiten la gestión interna
¿En el registro tienen información sobre las convocatorias?	En los registros no hay personas, sólo máquinas, por lo que no puede atenderte nadie

Respuesta a la encuesta	
Contraseña	wUE
¿Cuáles son los principales polideportivos del Campus de Moncloa?	Polideportivo Paraninfo y Polideportivo del Sur
¿Qué tipos de campos hay en los polideportivos?	Fútbol, tenis, basket, rugby, pádel y volley playa
¿Cuánto cuesta el carné anual deportivo?	
¿Qué permite hacer el carné anual?	Permite acceder tanto al gimnasio como a la piscina todo el año
¿Qué hay que hacer para reservar una pista?	Reservar la pista y pagarla
¿Qué piscinas hay en el Campus de Moncloa?	
¿Qué formas hay de entrar en la piscina?	Comprando un pase individual o con el carné anual
¿Cuánto cuestan las piscinas?	5€ (4€ para los miembros de UCM) y 40€ (25€ para los miembros de la UCM)
¿Cuáles son los registros principales del Campus de Moncloa?	Hay registros en las facultades de informática, químicas, odontología y en el vicerrectorado.
¿Es obligatorio realizar todas las convocatorias a través del registro?	
n el registro tienen información sobre las convocatorias?	

Respuesta a la encuesta	
Contraseña	Hkq
¿Cuáles son los principales polideportivos del Campus de Moncloa?	Polideportivo del Norte y Polideportivo del Sur
¿Qué tipos de campos hay en los polideportivos?	Rugby, tenis, basket, jockey, golf y pista de hielo
¿Cuánto cuesta el carné anual deportivo?	300€ al año
¿Qué permite hacer el carné anual?	Permite acceder al gimnasio todo el año
¿Qué hay que hacer para reservar una pista?	Reservar la pista y pagarla
¿Qué piscinas hay en el Campus de Moncloa?	Hay tres y las tres son cubiertas y están en el Polideportivo del Sur
¿Qué formas hay de entrar en la piscina?	Comprando un pase individual o con el carné anual
¿Cuánto cuestan las piscinas?	5€ (4€ para los miembros de UCM) y 40€ (25€ para los miembros de la UCM)
¿Cuáles son los registros principales del Campus de Moncloa?	Hay registros en las facultades de informática, químicas, odontología y en el vicerrectorado.
¿Es obligatorio realizar todas las convocatorias a través del registro?	Sí, sin excepción dado que los registros de la UCM son los únicos que permiten la gestión interna
¿En el registro tienen información sobre las convocatorias?	En los registros no hay personas, sólo máquinas, por lo que no puede atenderte nadie

Respuestas tras la realización del experimento

Respuesta a la encuesta	
Contraseña	x89
¿Cuáles son los principales polideportivos del Campus de Moncloa?	Polideportivo del Norte y Polideportivo del Sur
¿Qué tipos de campos hay en los polideportivos?	Fútbol, tenis, basket, rugby, pádel y volley playa
¿Cuánto cuesta el carné anual deportivo?	250€ al año
¿Qué permite hacer el carné anual?	Permite acceder tanto al gimnasio como a la piscina todo el año
¿Qué hay que hacer para reservar una pista?	Poner una instancia de reserva de pista y pagar la pista
¿Qué piscinas hay en el Campus de Moncloa?	Hay una en el Polideportivo de la Almudena y es cubierta y otra en el Polideportivo del Sur descubierta
¿Qué formas hay de entrar en la piscina?	Comprando un pase individual o con el carné anual
¿Cuánto cuestan las piscinas?	5€ (4€ para los miembros de UCM) y 40€ (25€ para los miembros de la UCM)
¿Cuáles son los registros principales del Campus de Moncloa?	Hay registros en las facultades de químicas, medicina, multiusos, en el rectorado y en el edificio de estudiantes.
¿Es obligatorio realizar todas las convocatorias a través del registro?	No, pueden hacerse por correos también siempre y cuando se mande antes de finalizar la convocatoria
¿En el registro tienen información sobre las convocatorias?	Pueden aconsejarte sobre la documentación, pero no conocen todas las convocatorias

Respuesta a la encuesta	
Contraseña	Hkq
¿Cuáles son los principales polideportivos del Campus de Moncloa?	Polideportivo Paraninfo y Polideportivo del Sur
¿Qué tipos de campos hay en los polideportivos?	Fútbol, tenis, basket, rugby, pádel y volley playa
¿Cuánto cuesta el carné anual deportivo?	250€ al año
¿Qué permite hacer el carné anual?	Permite acceder tanto al gimnasio como a la piscina todo el año
¿Qué hay que hacer para reservar una pista?	Poner una instancia de reserva de pista y pagar la pista
¿Qué piscinas hay en el Campus de Moncloa?	Hay una en el Polideportivo de la Almudena y es cubierta y otra en el Polideportivo del Sur descubierta
¿Qué formas hay de entrar en la piscina?	Comprando un pase individual o un bono de 10 baños
¿Cuánto cuestan las piscinas?	5€ (4€ para los miembros de UCM) y 40€ (25€ para los miembros de la UCM)
¿Cuáles son los registros principales del Campus de Moncloa?	Hay registros en las facultades de químicas, medicina, multiusos, en el rectorado y en el edificio de estudiantes.
¿Es obligatorio realizar todas las convocatorias a través del registro?	No, es recomendable pero también puede usarse cualquier otro registro de la ciudad, así como correo certificado
¿En el registro tienen información sobre las convocatorias?	Pueden aconsejarte sobre la documentación, pero no conocen todas las convocatorias

Respuesta a la encuesta	
Contraseña	wUE
¿Cuáles son los principales polideportivos del Campus de Moncloa?	Polideportivo Paraninfo y Polideportivo del Sur
¿Qué tipos de campos hay en los polideportivos?	Fútbol, tenis, basket, rugby, pádel y volley playa
¿Cuánto cuesta el carné anual deportivo?	250€ al año
¿Qué permite hacer el carné anual?	Permite acceder tanto al gimnasio como a la piscina todo el año
¿Qué hay que hacer para reservar una pista?	Poner una instancia de reserva de pista y pagar la pista
¿Qué piscinas hay en el Campus de Moncloa?	Hay una en el Polideportivo de la Almudena y es cubierta y otra en el Polideportivo del Sur descubierta
¿Qué formas hay de entrar en la piscina?	Comprando un pase individual o con el carné anual
¿Cuánto cuestan las piscinas?	5€ (4€ para los miembros de UCM) y 40€ (25€ para los miembros de la UCM)
¿Cuáles son los registros principales del Campus de Moncloa?	Hay registros en las facultades de químicas, medicina, multiusos, en el rectorado y en el edificio de estudiantes.
¿Es obligatorio realizar todas las convocatorias a través del registro?	No, pueden hacerse por correos también siempre y cuando se mande antes de finalizar la convocatoria
¿En el registro tienen información sobre las convocatorias?	Son personal especializado en todas las convocatorias ofrecidas en la UCM

Respuesta a la encuesta	
Contraseña	fdc
¿Cuáles son los principales polideportivos del Campus de Moncloa?	Polideportivo Paraninfo y Polideportivo del Sur
¿Qué tipos de campos hay en los polideportivos?	Fútbol, tenis, basket, rugby, pádel y volley playa
¿Cuánto cuesta el carné anual deportivo?	250€ al año
¿Qué permite hacer el carné anual?	Permite acceder tanto al gimnasio como a la piscina todo el año
¿Qué hay que hacer para reservar una pista?	Reservar la pista y pagarla
¿Qué piscinas hay en el Campus de Moncloa?	Hay una en el Polideportivo de la Almudena y es cubierta y otra en el Polideportivo del Sur descubierta
¿Qué formas hay de entrar en la piscina?	Comprando un pase individual o con el carné anual
¿Cuánto cuestan las piscinas?	5€ (4€ para los miembros de UCM) y 40€ (25€ para los miembros de la UCM)
¿Cuáles son los registros principales del Campus de Moncloa?	Hay registros en las facultades de químicas, medicina, multiusos, en el rectorado y en el edificio de estudiantes.
¿Es obligatorio realizar todas las convocatorias a través del registro?	Sí, sin excepción dado que los registros de la UCM son los únicos que permiten la gestión interna
¿En el registro tienen información sobre las convocatorias?	Pueden aconsejarte sobre la documentación, pero no conocen todas las convocatorias

Apéndice B – Paper EDUCON 2017

uAdventure: The eAdventure reboot

Combining the experience of commercial gaming tools and tailored educational tools

Ivan J. Perez-Colado, Victor M. Perez-Colado, Ivan Martínez-Ortiz, Manuel Freire-Moran, Baltasar Fernández-Manjón
Dept. of Software Engineering and Artificial Intelligence, Facultad de Informática, Universidad Complutense de Madrid Madrid, Spain
{ivanjper, victormp}@ucm.es, {imartinez, manuel.freire, balta}@fdi.ucm.es

***Abstract*—Educational games (aka serious games, SG) are powerful educational contents. However, they are costly to develop, and once developed, SGs become dependent on software and hardware combinations that may become obsolete, such as Adobe Flash or Java Applets. Addressing these problems would allow a much greater use of SGs in education. The eAdventure authoring tool, developed by the e-UCM research group, addressed high development costs, and resulted in the creation of multiple SGs in collaboration with different institutions. However, eAdventure's Java Applets have become increasingly difficult to run due to platform obsolescence. To maintain the benefits of the eAdventure platform and user base, we have created new platform called uAdventure: an SG editor built on top of the game engine Unity that allows for the creation of educational adventure games without requiring programming. Since Unity is supported on a majority of platforms (including mobile). By developing SGs with uAdventure, the games become future-proof, as they can be updated and retargeted for new platforms as required. In this sense, uAdventure improves the lifecycle of SGs by reducing both authoring and maintenance costs.**

Keywords—serious games; Unity 3D; learning analytics; geolocalization

I. INTRODUCTION

Educational simulations, gamification approaches and educational games in general (hereinafter referred to as serious games, SGs) have been attracting lot of attention by education and training communities due to their intrinsic motivating and engaging characteristics [1]. SGs have been applied in different educational settings and application domains, most notably in business management, health, and military. SGs are used to facilitate training of abilities tactics that are specifically suited to be acquired in role-playing educational settings, such as conflict management or business negotiation skills. Moreover, SGs can be used to acquire or refresh a required ability that is infrequent or rare in real life (medicine) or poses security or cost difficulties (military). In addition to education and training purposes, SGs are also used in other domains such as public health or environmental protection to improve awareness or to affect behaviors.

As a result of the increasing use and acceptance of SGs, a niche market is being created around their development and maintenance. This market is expected to grow over 16% annually between 2015 and 2020 [2]. For instance, CodinGame, an online SG to learn programming, has around half a million registered users, two thousand of which are online at any given instant[3].

The process of developing and procuring SGs can be classified using the following categories: i) *Commercial Off-the-Shelf (COTS)*, ii) *Pre-existing Serious Game*s and iii) *Tailored Serious Games*. This classification is based on the alignment of the learning goals of the educators against the actual learning goals (if any) that are part of the SG's actual design. Generally, the greater the degree of personalization, the better the game will suit educators' needs. As we get closer to a fully-personalized game, the development process of the game will become more complex and costly (per user).

In some cases, an already available commercial game would suit the educator's needs. These COTS games, when applied in an adequate educational context, can have a relevant impact. For example, Zoo Tycoon[1] is a game focused around building and running successful zoo scenarios where players learn concepts related to economics or business management; and has been used as a SG to help teach these concepts. The economical advantage of this approach is clear: the game does not need to be developed, only licensed. However, the effective application of this approach also requires an additional effort to design the pedagogical scenarios and a degree of teacher support while the game is deployed as an SG.

Pre-existing Serious Games are commercial SGs that are usually supported by training and consultancy companies that may offer a customization layer. This customization layer, is usually offered as an *a la carte* service, where the organization/ instructor can choose a set of learning goals and the company provides a set of SGs or a single SG that cover the required competencies or learning goals. This approach can be considered as a *Games as a Service*, where the learning organization/instructor does not have to either develop or maintain the game, for a fee. Other advantages include streamlined integration with the learning process, or assessment

[1] https://en.wikipedia.org/wiki/Zoo_Tycoon

tools for monitoring and behavior analysis. An example of this genre is Classcraft[2], where the classroom is transformed into a role-playing game that improves students' collaboration and teamwork.

Finally *tailored serious games*, despite having the greatest potential effectiveness, also tend to require large, multidisciplinary development teams and, therefore, have high production costs. However, for simpler SGs, it is feasible to provide educators with an easy to use platform, with limited features, that can allow them to create their own SGs; or at least to modify or localize it to their needs.

The eAdventure (eA) platform [4] provides an authoring tool that allows for SG development without requiring advanced computer skills (i.e. programming). In addition, the eA platform allows for the reuse and repurposing of preexisting eA games. This allows educators to customize the games for a specific educational setting. For example, changing the students' evaluation included in the game or simply changing graphic assets of the game elements to customize their aspect to the specific learning context.

However, the eA platform was launched in 2007, and although it has received several updates, its underlying technology has run into technical limitations. In particular, eA's targeted the Java Plugin, then present in most browsers. However, as described in JEP 289[3], the Java Plugin is to be deprecated in newer versions of Java; and, as of this writing, most browsers already ship with this component either absent or disabled. To address these concerns, and to streamline the development of new educational features, we have designed uAdventure (uA), an evolution of eA.

The rest of the paper is structured as follows; section II provides insight into eA analyzing the limitations that required an evolution from both technical and educational perspectives. Section III and IV describe uA and the new features and educational opportunities that it offers. Finally, section V provides some conclusions and future lines of work.

II. FROM EADVENTURE TO UADVENTURE

A. *eAdventure*

Developing games, and especially SGs, requires addressing and balancing both educational and technological aspects. One of the first aspects to consider is the platform or technology used to develop the game. For example, using a professional game engine such as *Unity* or *Unreal Engine* requires a game development team with good programming background; and this will prevent most teachers from participating in the SG's core development team.

Another aspect the adequate balance between engagement and education [5]. On the one hand, if the game is too focused on the educational facets it may not be engaging enough for the student. On the other hand, if the game is too narrowly focused on the fun facet, the student may not learn enough, yielding only limited educational value.

Within games there are several genres, and not all of them are equal from an educational perspective. Dickey [6] and Amory [7] have identified point-and-click adventure games as one of the most suitable genres for SG, due to the presence of elements such as a slow pace, reflection, study of the environment, and problem-solving [8]. There are specific commercial editors for creating these kind of games, such as *Adventure Game Studio*[4], *Adventure Maker*[5] or *3D Adventure Studio*[6] [9]. However, these platforms do not usually include support for SG educational elements (e.g. evaluation).

eA is a platform for simplifying the development of 2D point-and-click adventure serious games [4]. In eA, educators can actively participate in the development of the SG side-by-side with the game developers or even develop the SG themselves without requiring programming skills. This way, educators are at the core of the development process taking care of the educational aspects that fits for their own needs.

eA has four characteristics that make it especially suited for the creation of educational SGs: i) game development can be achieved with small budgets, ii) maximizes the Return on Investment (ROI), iii) customizes the learning experience and iv) facilitates both distribution and evaluation. One of the most expensive aspects of game development is the creation of graphic assets. With eA, it is possible to use pictures taken with any digital camera to create the graphic assets. eA games are editable, meaning that existing games can always be opened with the editor to either customize them for specific learning scenarios, or to update them for new requirements.

In addition, eA games can be packaged as an e-learning standard content package that includes the game as a Java applet making possible to distribute it using a Learning Management System (LMS). Furthermore, if the LMS it is compatible with the ADL SCORM [10] e-learning specification the eA games can send back assessment results to the LMS.

B. *Why does eAdventure need a refurbishment?*

eA design started in 2007, and the first version was delivered in early 2008 as an open source project. The project has been maintained since then by different contributors, mainly by researchers that have been actively using the platform. These contributors have often added features requested by educators participating in the eA community. Nevertheless, there were some architectural and technological decisions that were correct at that time, but are now hindering the maintenance and evolution of the platform.

eA was built with Java technology to take advantage of the *"build once run anywhere"* motto. Java technology powered both the eA editor (used in game development) and the games themselves, which could be deployed in different platforms without changes. This way it is possible to alleviate, or completely avoid, compatibility issues between the computer

[2] https://www.classcraft.com

3 February 2016's JEP 289: http://openjdk.java.net/jeps/289

[4] http://www.adventuregamestudio.co.uk/

[5] http://www.adventuremaker.com/

[6] http://3das.noeska.com/

used by the educator (or the development team) to create the SG and the usual heterogeneous environment that can be found at school labs (including security restrictions). In addition, this multiplatform nature and the ability of eA games to be customized fostered the collaboration between educators in a community of practice by sharing SGs.

Although Java's *"build once run anywhere"* motto may still hold for the desktop platform, desktop computers, once the main platform in school, have since disappeared from multiple institutions, displaced by tablets. Desktop Java is generally unavailable on tablets and smartphones: although Java as a language is still relevant (due to the Android platform), development in Java is not longer providing effective multiplatform support. Moreover, mobile devices offer new ways to interact with the device (touch, accelerometer, geolocation, etc.) that open new opportunities to create ubiquitous educational experiences that should be used in SGs.

eA SG deployment relies on Java applets (i.e. Java programs that run inside the browser) to facilitate SG distribution and integration into existing e-learning platforms. When eA was created, e-learning content interoperability was mainly focused on distribution of web content. The use of Java applets in eA facilitates the reuse of e-learning standards allowing SGs to be packaged as an IMS-CP content [11]. This allows students to access and play the game just like any other web content. The eA platform could also take advantage of ADL's SCORM e-learning standard to communicate back with the LMS, mainly to store SG progress and assessment information inside the hosting Learning Management System (LMS).

However, due to the continuous security problems [12] in Java Applets in particular and in browsers' plugins in general, most of the browsers have already deprecated the Java applets or are in the process of deprecating them [13].

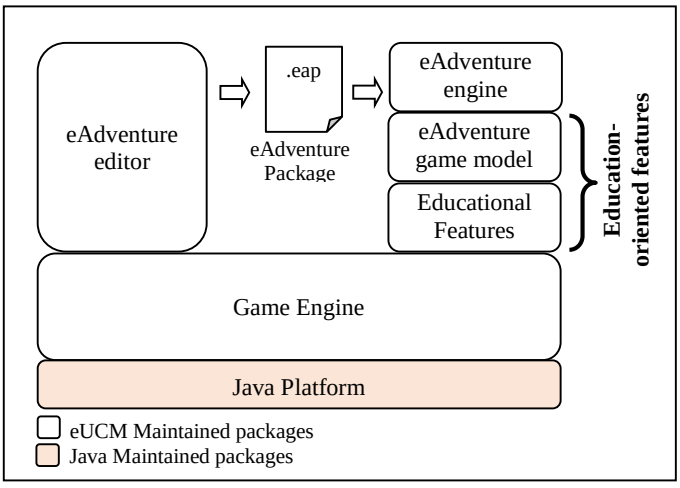

Fig. 1 eAdventure's main components

The eA platform, besides being an open source point-and-click editor, complements the usual game platform features with educational features (e.g. assessment, e-learning standards support) that make eA still unique and relevant (see Fig. 1). Although only two of eA's components have been specifically designed for educational purposes, considerable effort is required to maintain the platform as a whole. This imbalance it is dangerous for the maintenance and survival of an open source research project with limited resources (developers). In particular, keeping up with technological evolution makes it harder to add and update the educational features that teachers expect and demand.

Nonetheless, we consider that eA's main ideas are still valid and relevant to the educational community. Therefore, we decided build a new SG platform, which we describe in the following section.

III. uAdventure

uA development aims to achieve the following goals: i) to address the technical issues of the eA platform, ii) to face the project management/survival issues, iii) to provide a solid base platform to build new educational features and iv) maintain compatibility with eA (at least importing previous eA games).

uA is built on top of Unity mainly because it has become the most popular game development platform [14]. One of the reasons of this popularity is its multi-platform support[7] including desktop (Windows, Mac OS and Linux), consoles (PlayStation, Xbox, PSVita or Nintendo 3DS), web and mobile (Android and iOS). Unity has attracted attention both from big companies like Blizzard with "Hearthstone"[8], Nintendo with "PokemonGO"[9] or Ubisoft with "Grow Up!"[10], and is also one of the top choices for so-called "indie" developers[11]. The "indie" label is generally used to refer to low-budget games, and in this sense encompasses most SG developments. Compared to the other choices, Unity stands out for its simplicity and extensibility, rich documentation, and a very active community of developers.

In addition, the choice of Unity was an answer to project management issues. uA can take advantage of a preexisting strong platform and community. Thus uA development time can be focused on the development of those characteristics that made eA unique. Moreover, due to the use of a well-known platform, finding contributors or hiring developers to create new features should be greatly simplified.

Furthermore, the expansion of mobile technologies has created new opportunities to apply them in education. As we move into mobile technologies, new forms of "anytime and anywhere" learning are possible. Learning can happen at any moment with a range of devices: at home with the PC, at school with a tablet, or just outside during the free time with the smartphone.

uA have been created in a context where previously developed eA games have serious deployment problems. This new uA framework, built on top of Unity, allows eA projects to be imported, allowing them to be tested and modified on the new execution core. This extends, supports, and simplifies their

[7] https://unity3d.com/unity/multiplatform
[8] http://eu.battle.net/hearthstone/en/
[9] http://www.pokemongo.com/
[10] https://www.ubisoft.com/en-GB/game/grow-up/
[11] https://unity3d.com/public-relations

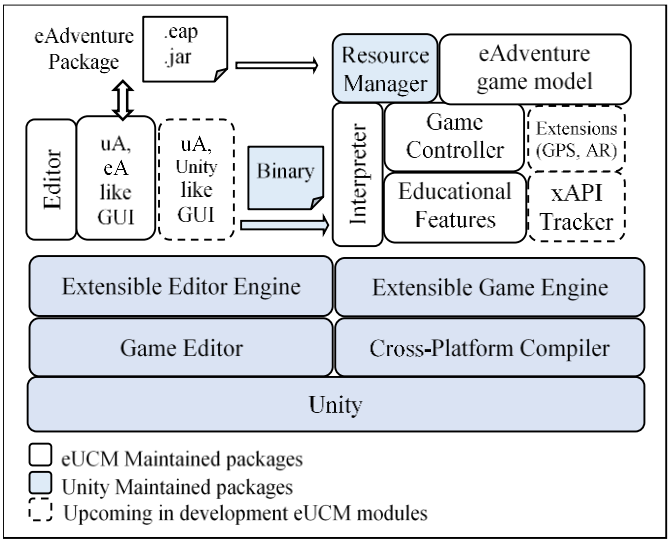

Fig. 2: uAdventure modules architecture, including Unity provided modules.

lifecycles. After opening a game in uA, and using the *Unity* compilation tools, it can be exported as a standalone game for a range of different platforms and operating systems.

A. *uAdventure Architecture*

eA is composed of several layers (see Fig. 1), such as editor, game player, representation core and project builder. In tightly-coupled, complex software solutions developers need to cope with all the complexity; and in this case, developers wishing to extend eA needed to be familiar with all of these layers. In contrast, uA is composed only of two main layers: *Interpreter* and *Editor*. Most of the previous ad-hoc eA layers have been replaced by Unity's standard layers. Thus, uA will be easier to maintain and evolve.

The uA module architecture is presented in Fig. 2. It shows how the graphical editor (GUI), the interpreter and the emulator are built on top of Unity. New features are highlighted in Fig. 2 with dashed lines: the xAPI tracker, extensions such as location support, and a future Unity-like GUI.

Compared to the previous architecture, it may seem more complex, since there are more modules. However, this is misleading: even though there are more e-UCM maintained modules, they are organized into only two layers: *Editor* and *Interpreter*. The rest of the e-UCM maintained modules are smaller and easy to maintain compared to the eAdventure's packages (Editor and Game Engine) shown in Fig. 1, whose cores have been replaced with the Extensible Editor Engine and the Extensible Game Engine provided by Unity.

Therefore, the two main layers that compose uA are: *Interpreter* and *Editor*. The *Editor* is focused on creating a game specification that, in conjunction with the required game resources, will make it playable. It uses the eA data model plus some new additions, like new map scenes and new assessment specification [15]. The *Interpreter* relays on Unity's representation core and translates all eA model components into Unity game elements.

B. *The uAdventure editor*

The *Editor* allows users to modify the game specification, manage and create new game resources. Some key aspects of

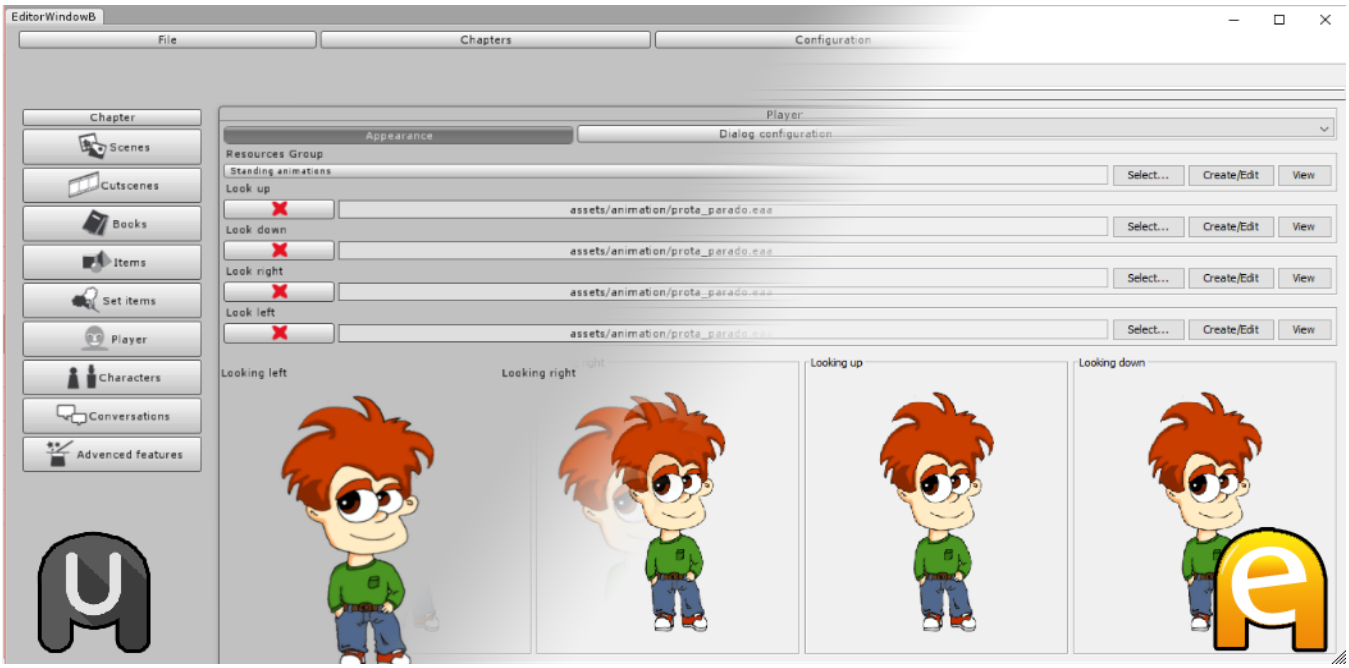

Fig. 3: Fade transition from players appearance view on uAdventure (left) to previous players appearance view on eAdventure (right). The editor is quite similar in both systems.

eA have been recreated in uA, such as the main graphical user interface, including menus, windows and general appearance. This way, it will be easier for the current eA users to migrate to uA. However, uA will also provide a brand-new Unity-Unified GUI. This new GUI will be based on the idea of tighter integration with the Unity editor, and is inspired by the most used Unity windows, such as the Scene preview and object (model) hierarchy, Unity animator, or element inspector. This enables greater modularity and allows game developers familiarized with Unity to take advantage of uA's added value for educational videogames. A comparison between the old eA GUI and the new uA GUI is presented in Fig. 3. In this case, an eA user that is just starting out with uA can quickly become productive: the interaction with the GUI is very similar to what they already know, making for a smoother learning curve.

As uA evolves inside Unity, it will be able to benefit from other Unity features, such as 3D assets (downloadable from the Unity Asset Store), the physics engine, navigation meshes for movement, or native animation state machines. Games will result richer and developers should need less time to produce them. In addition, Unity's native features are already well documented, can be reused between multiple games and distributed as packages, and will be improved and maintained as Unity evolves.

One of the improvements of uA is multimedia support with the usage of the *FFmpeg* project for export-time video conversion. Multimedia resources including video and audio are complex to deal with when deploying a game in different platforms. uA addresses those problems, not only when playing the game (using Unity's multimedia players), but also during game creation. It uses FFmpeg for converting videos on-the-go, increasing, for example, the number of supported video formats, from AVI (DIVX 4, XVID, DIVX3low, S-Mpeg 4 V2, DIVX5) and MPG (MPEG1), to almost every video format available. This will greatly simplify the inclusion of new assets in games.

Therefore, using Unity as a support layer has not only allowed us to include new features; but also has improved the development and maintenance of uA. On one hand, development requires a deep knowledge of the Unity platform and experience in extending. On the other hand, some of the most time-consuming maintenance aspects are no longer required (e.g. coping with changes in mobile platforms, multimedia playback) as they are provided by Unity. This allows e-UCM's resources to focus on improving and increasing the educational features and the added value offered by uA.

C. The uAdventure interpreter: an emulator for eAdventure

The *Interpreter* loads game description file and makes it playable. Taking advantage of Unity's features, the *Interpreter* extends the Extensible Game Engine, developing behaviors for every element that is visible or plays a role in the game.

Furthermore, the *Interpreter* also manages the *GameState*, including the state of the *flags*, *variables*, *global-states*, and other game elements such as the *inventory* or the *current scene*. Finally, it also orchestrates, relying on Unity's *MonoBehaviour* lifecycle, the execution of the game.

The *Interpreter* is not only responsible for representation, but also for handling user interaction. Video games developed with uA can be exported to a variety of platforms, which raises a serious issue: different platforms imply different interaction methods. uA supports different interaction methods, including traditional mouse and keyboard as well as touch screen environments. The framework provides developers with some tools for making the game easier to play depending on user's runtime environment. For example, when a game is *exported* for PC/Mouse-Keyboard system, the game will be presented like older eA games, with different cursors, floating names, and lists of answers at the bottom of the screen. On other hand, when a game is exported for us in a Tablet/Touch system, interactive elements glow periodically to capture the player's attention, because tablet players can no longer "mouse-over" scenes to find interactive elements.

uA also includes a multi-platform eA game *emulator* that can take a ".jar" executable eA game (an already-exported, ready-to-run eA game) and run it on every platform available within uA. This emulator runs the previous games, but supports new uA features (provided by the Interpreter). This includes better animations, more effects, improved and adapted GUI, or the ability to play eA games on mobile devices. The emulator allows users to browse their filesystem, and easily import those games they want to play on their preferred platform. Previously launched games will be saved in the emulator with its configuration that will enable learners to use old games on pre-configured devices.

D. Migrating SGs from eAdventure to uAdventure

eA's SG life cycle has been greatly improved with the implementation of the uA *Interpreter* as most of the games developed with eA in the past are now supported by uA. This prevents the considerable effort that went into designing, implementing and testing these games from being lost. As the game is now difficult to deploy or does not run in new environments it would have otherwise required a complete redevelopment to achieve the same degree of support. That is, the use of uA avoids the high cost in time and money that would otherwise be required to reimplementation these older games in new platforms.

There is an actual case study of adapting a game to a new game engine, where an experienced game developer of the e-UCM research group reproduced the First Aid [16] game inside Unity. The production took almost 50 hours of development, and even with that effort, only 85% of the game was supported (some game situations were ignored). Based on this data, we can estimate that 60 to 80 hours of development are necessary to fully rebuild a simple eA game into a new game engine, assuming that the developer responsible has access to the initial detailed design document, and without factoring extensive user testing, which may reveal additional problems. This is a significant cost for the maintenance of a previously produced SG, where in many cases there is no budget at all for on-going maintenance. Indeed, the 6 educational games developed by the educational organization CATEDU[17] using eA are in this situation. Thanks to uA, this cost can be reduced to the few minutes it takes to import the project into uA and generate new

versions. Indeed, we have taken the First Aid [16] game and done exactly that; and are currently piloting it again in different schools in Madrid to test the learning analytics features described in the next section.

IV. NEW EDUCATIONAL FEATURES IN UADVENTURE

In terms of educational features, there are few differences between uA and eA. This section will focus on description of the two main educational features that are being implemented in uA: i) learning analytics as an assessment tool and ii) geolocated games.

A. *Learning Analytics for SGs*

The evaluation of a formative activity that uses a SG poses significant challenges. Usually the teacher has little control over, or information of, the learner's activity through the process of playing a serious game [1]. This also happens usually when using other eLearning technologies in the classroom, such as MOOC or LMS. Some research results show that the more teachers know about the learners' behavior using a learning tool, the better teachers can control that learning tool [18].

eA's assessment feature was based on conditions that tracked player actions inside the game such as "The player has failed test X" or "Player refuses to accept help from Non-Player Character Y" [11]. These conditions, although useful for evaluation, were one of the most complex features to master for developers of eA games. This requires mastering conditions and, to some extent, to have computational thinking abilities to understand how to link the game states that have a correlation with the educational learning goal; or with the skill that is put into practice while playing the game. Moreover, the eA editor allows authors to create multiple assessment profiles (sets of conditions) [19]. The evaluation outcome was a textual report, which had to be interpreted in terms of actual learning outcomes by an instructor.

In addition, evaluation based on pre-set conditions implies that, the more the authors want to evaluate, the more time they must spend creating conditions. Still, there are some aspects of the learner's behavior that are lost because conditions can only track scenarios that are devised before the learning activity takes place. Finally, this approach works for small groups of learners, but as the groups become larger, the difficulty of supporting corrective interventions during the learning activity increases: more learners result in more the individual documents that must be read and analyzed.

A complementary approach to condition based assessment, is based on the application of big data and analytics techniques where learners' gameplay data is harvested, processed and presented in a dashboard to the educator. This approach can be used to provide metrics that will help the instructor to assess learners or even discovering game design errors [20], [19].

In order to apply big data and analytics techniques, all gameplay interactions must first be collected. Instead of creating a custom format to represent theses interactions, we use an existing e-learning specification called xAPI (eXperience API) [21]. In xAPI, traces of interactions (called statements) are composed of: i) an *actor*, that defines who performed the action, ii) a *verb*, that defines the action between the actor and the object, iii) an *object*, that defines the thing that was acted on, iv) and optional properties such as *result*, representing a measured outcome related to the statement, or a *timestamp* of when did that happen [22]. Under this umbrella it is indeed possible to represent gameplay actions; but in order to facilitate the development of analytic tools that automate insights of the learners' gameplay the e-UCM team, in collaboration with the ADL initiative (developers of x-API), authored a specific xAPI application profile for use with serious games [15].

The reasons for using xAPI are twofold. On the one hand, this allows uA SG to interact with existing e-learning tools that are compliant with the xAPI specification and also with, for example, a Learning Record Store (a storage service for xAPI statements). On the other hand, and even more important, gameplay sessions will be compatible with future xAPI tools, for example, it will be possible to apply new learning analytics tools to gameplay sessions recorded in the past.

For example, with the current xAPI support (although incipient) it is possible to integrate the SGs created with uAdventure with the Learning Analytics application created as part of the H2020 RAGE[12] and BEACONING[13] projects.

B. *Embracing Ubiquity: geolocated games*

One of the main advantages of the uA and its supporting platform is the ability to create SGs for mobile devices. Moreover, these devices usually include a plethora of sensors that are unique to the mobile platform.

Getting into mobile allows games to be played anywhere and anytime. This opens new opportunities to use SGs in different educational scenarios, and using alternative interaction mechanisms. For example, MIT's MitAR[14] is a serious game platform that uses augmented reality and geolocation [23]; although it was never successfully ported to today's major mobile platforms. More recently, 2016 was the year of the engaging AR mobile game "PokemonGo", which not only attracted large numbers of users, but also indirectly had an impact in the players' wellness [24].

uA is starting to take advantage of the GPS, compass and camera sensors that are already available to Unity games. This will allow the creation of geolocated games that were beyond the capabilities of eA, such as gymkhanas, interactive tours, or cultural visitor guides. This new feature has been implemented adding a new type of *scene* to the uA data model. Until now, a game scene was a metaphor to represent a place and environment where players or their avatars interact with the game world. This new type of scene actually is a virtual representation of the real environment where the game takes place. This new *map scene* (and its companion editor) provides access to a map to define POIs (points of interest), regions, or even routes to link all the existing mechanics to the geolocation gaming paradigm.

[12] http://www.rageproject.eu/

[13] http://beaconing.eu/

[14] http://web.mit.edu/mitstep/projects/mitar-games.html

Finally, in order to be effective indoors, QR code scan support was added. QR codes are easily generated, and can be placed on interesting locations to be scanned by those playing on a mobile device; however, their use requires access to the device's in-built camera from within the game. In addition to enabling indoor use, support for QR codes also allows games to be changed or adapted to new locations by moving the (physically placed) QR codes around. For instance, one of most well-known games in the eA community is the "Fire Protocol" game[15], originally created as a demonstrator of eA's capabilities, and also as a drill for the actual fire protocol procedure in the Computer Science building at Complutense University of Madrid. Although the already uses pictures of actual locations inside the building, forcing the players to physically move around the actual locations depicted in the game provides a much greater degree of immersion.

V. Conclussions and Future work

The eAdventure (eA) platform has reached its technological limit due to Java platform's limitations supporting mobile and web-based deployments. However, the main goals and advantages of the eA platform are still relevant for the SG community. eA's SG life cycle has been greatly improved with the implementation of uAdventure (uA) as most of the games developed with eA in the past are now supported by uA.

The uA project addresses these technological limitations, and provides new educational capabilities that were not envisioned when eA was created. In order to address current and future technological challenges, uA is built on top of the Unity platform, which provides solid technical underpinnings, together with a strong user base and support in the game development community.

The uA tool has been created around two main elements: the *Editor* and the game *Interpreter*. The editor provides easy-to-use game authoring tool for non-programmers that is very similar to eA user interface, but includes some new features. Some of them are: touchscreen interaction support, Learning Analytics support, and geo-positioned game support with or without maps. Touchscreen support includes features that make playing point-and-click adventure games easier on touchscreens. Learning analytics support is provided by an integrated xAPI tracker that generates traces of what the learner does during gameplay. Finally, geo-positioning support uses mobile GPS and wireless capabilities, and new map scenes. Use of Unity has also allowed us to make multiple game representation improvements, giving games a better look and feel. Furthermore, by taking advantage of Unity's geolocation capabilities, uA will allow the creation of games such as gymkhanas or historical city tours.

The interpreter has been built using the very same model used in eA, in order to maintain the compatibility with games created with eA and facilitate the migration to the new platform. In the case-study presented in Section III-D, it can reduce up to 85% percent of the maintenance and migration costs associated with making a pre-existing SG capable of running on new platforms.

Integration of uAdventure with the real-time Learning Analytics platform used in the H2020 RAGE and BEACONING projects is ongoing. This LA platform can populate learning dashboards from incoming flows of xAPI traces, and can display alerts and warnings when learners enter risky areas or evidence anomalous behavior.

Although uA still requires significant work before public release, we believe that uA with the capabilities described in this paper will be a worthy heir of eA's legacy. Hopefully, the present work and the work done in the H2020 projects RAGE and BEACONING (where the e-UCM team participates) will facilitate a more widespread adoption of SGs in tomorrow's classrooms.

Acknowledgment

We would like to thank Piotr Marszal for his coding contribution on the first beta of uAdventure. This research has been partially financed by the Regional Government of Madrid [eMadrid S2013/ICE-2715], by the Ministry of Education [TIN2013-46149-C2-1-R] and by the European Commission [RAGE H2020-ICT-2014-1-644187, BEACONING H2020-ICT-2015-687676].

References

[1] M. Nichols, "A theory for eLearning," *Educational Technology and Society*, vol. 6, no. 2. pp. 1–10, 2003.

[2] "Serious Game Market worth $5,448.82 Million by 2020." [Online]. Available: http://www.marketsandmarkets.com/PressReleases/serious-game.asp.

[3] "CodinGame: General Leaderboard." [Online]. Available: https://www.codingame.com/leaderboards/global?column=codingpoints&value=all.

[4] J. Torrente, Á. Del Blanco, E. J. Marchiori, P. Moreno-Ger, and B. Fernández-Manjón, "e-Adventure: Introducing Educational Games in the Learning Process," in *IEEE Education Engineering (EDUCON) 2010 Conference*, 2010, pp. 1121–1126.

[5] M. Prensky, "Digital game-based learning," *Comput. Entertain.*, vol. 1, no. 1, p. 21, Oct. 2003.

[6] M. D. Dickey, "Game Design Narrative for Learning: Appropriating Adventure Game Design Narrative Devices and Techniques for the Design of Interactive Learning Environments," *Educ. Technol. Res. Dev.*, vol. 54, no. 3, pp. 245–263, Jun. 2006.

[7] A. Amory, "Building an Educational Adventure Game: Theory, Design and Lessons," *J. Interact. Learn. Res.*, vol. 12, no. 2, pp. 249–263, 2001.

[8] R. Van Eck, "Building Artificially Intelligent Learning Games," in *Games and Simulations in Online Learning*, IGI Global, 2007, pp. 271–307.

[9] P. Moreno-Ger, J. L. Sierra, I. Martínez-Ortiz, and B. Fernández-

[15] http://e-adventure.e-ucm.es/course/view.php?id=29

Manjón, “A documental approach to adventure game development,” *Sci. Comput. Program.*, vol. 67, no. 1, pp. 3–31, Jun. 2007.

[10] J. Torrente, P. Moreno-Ger, I. Martínez-Ortiz, and B. Fernandez-Manjon, “Integration and deployment of educational games in e-learning environments: The learning object model meets educational gaming,” *Educ. Technol. Soc.*, vol. 12, no. 4, pp. 359–371, 2009.

[11] P. M. Ger, “eAdventure: Serious games, assessment and interoperability,” in *2014 International Symposium on Computers in Education (SIIE)*, 2014, pp. 231–233.

[12] G. McGraw and E. W. Felten, *Java Security: Hostile Applets, Holes&Amp;Antidotes*. New York, NY, USA: John Wiley & Sons, Inc., 1996.

[13] “NPAPI deprecation: developer guide.” [Online]. Available: https://www.chromium.org/developers/npapi-deprecation.

[14] F. Messaoudi, G. Simon, and A. Ksentini, “Dissecting games engines: The case of Unity3D,” in *2015 International Workshop on Network and Systems Support for Games (NetGames)*, 2015, vol. 2016–Janua, pp. 1–6.

[15] Á. Serrano-Laguna, I. Martínez-Ortiz, J. Haag, D. Regan, A. Johnson, and B. Fernández-Manjón, “Applying standards to systematize learning analytics in serious games,” *Comput. Stand. Interfaces*, vol. 50, pp. 116–123, Feb. 2017.

[16] E. J. Marchiori, G. Ferrer, B. Fernandez-Manjon, J. Povar-Marco, J. F. Suberviola, and A. Gimenez-Valverde, “Video-game instruction in basic life support maneuvers,” *Emergencias*, vol. 24, no. 6, pp. 433–437, 2012.

[17] CATEDU, “E-ADVENTURES. Download page.,” 2011. [Online]. Available: http://www.catedu.es/webcateduantigua/index.php/descargas/e-adventures.

[18] A. Ravenscroft, “Designing E-learning Interactions in the 21st Century: revisiting and rethinking the role of theory,” *Eur. J. Educ.*, vol. 36, no. 2, pp. 133–156, Jun. 2001.

[19] W. L. Wong, C. Shen, L. Nocera, E. Carriazo, F. Tang, S. Bugga, H. Narayanan, H. Wang, and U. Ritterfeld, “Serious video game effectiveness,” in *Proceedings of the international conference on Advances in computer entertainment technology - ACE ’07*, 2007, no. January, p. 49.

[20] Á. del Blanco, J. Torrente, P. Moreno-Ger, and B. Fernández-Manjón, “Enhancing Adaptive Learning and Assessment in Virtual Learning Environments with Educational Games,” in *Intelligent Learning Systems and Advancements in Computer-Aided Instruction*, IGI Global, 2013, pp. 144–163.

[21] M. Manso Vazquez, M. Caeiro Rodriguez, and M. Llamas Nistal, “Development of a xAPI application profile for self-regulated learning requirements for capturing SRL related data,” in *2015 IEEE Global Engineering Education Conference (EDUCON)*, 2015, pp. 358–365.

[22] “xAPI-About @ github.com.” [Online]. Available: https://github.com/adlnet/xAPI-Spec/blob/master/xAPI-About.md#partone.

[23] B. Coulter, E. Klopfer, J. Sheldon, and J. Perry, “Discovering Familiar Places,” in *Games, Learning, and Society*, C. Steinkuehler, K. Squire, and S. Barab, Eds. Cambridge: Cambridge University Press, 2016, pp. 327–354.

[24] T. Althoff, R. W. White, and E. Horvitz, “Influence of Pokémon Go on Physical Activity: Study and Implications,” *J. Med. Internet Res.*, vol. 18, no. 12, p. e315, Dec. 2016.