Panel de control para generación y distribución de entornos panorámicos

Autor: Alberto García Ruiz
Tutor: David Sánchez Blancas
Profesor: Jordi Duch Gavaldà

Diseño y Programación de videojuegos
Nuevas tecnologías e investigación
06/2023

Créditos/Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Panel de control para generación y distribución de entornos panorámicos*
Nombre del autor:	*Alberto García Ruiz*
Nombre del colaborador/a docente:	*Jordi Duch Gavaldà*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega (mm/aaaa):	*06/2023*
Titulación o programa:	*Diseño y Programación de videojuegos*
Área del Trabajo Final:	*Nuevas tecnologías e investigación*
Idioma del trabajo:	*Español*
Palabras clave	*Panorámicas, Editor, Unity*

Resumen del Trabajo:

Mobile Media Content es una empresa tecnológica focalizada en la virtualización de recintos deportivos y culturales, así como la distribución de contenido digital generado en Unity, con la finalidad de ser utilizado en webs de venta de entradas a modo de previsualización de entornos.

Dado un proyecto de Unity con la virtualización de un recinto, se recoge un conjunto de piezas tecnológicas para automatizar la generación de entornos panorámicos desde cada una de sus localidades, la subida del contenido a un servicio en la nube y la escritura de datos para análisis posteriores. El proceso de generación y subida se realiza mediante el uso de la funcionalidad Extending the Editor ofrecida por el motor gráfico Unity. Gracias a ello, se diseña una herramienta adicional para crear y modificar recintos, así como herramientas específicas que parten de la principal generación.

A raíz de la estructura de datos construida para cada recinto, se obtiene como resultado una o varias imágenes (en función del modo de generación seleccionado) y un archivo *json* que contiene información sobre la vista y, en algunos casos, sobre la navegación entre ellas. Una vez subido el contenido al servidor, el cliente final puede integrar en su página web un visor 3D para previsualizar el recinto en cuestión desde la localidad seleccionada.

Abstract (in English, 250 words or less):

Mobile Media Content is a technological company focused on sport and cultural venues virtualization, as well as digital content distribution generated in Unity, for being used in ticketing webpages as a preview mode for environments.

Given a Unity project with a venue virtualization, a bunch of technological tools are gathered to automate the generation of panoramic environments from each one of its locations, upload the

generated content to cloud and write specific data to perform an analysis. The process of generation and upload is achieved by using the Extending the Editor tool powered by Unity engine. Thanks to it, an additional tool is designed and build to create and modify venues, as well as more specific tools whereupon the main generation.

Based in the data structured build for each venue, the final output consists of one or more images (depending on the selected generation mode) and a JSON file, which contains information about the 3D view and, in some cases, about the navigation between them. Once the content is uploaded to the server, the final client can integrate in his website a 3D viewer to preview the venue itself from the selected location.

Dedicatoria/Cita

Me gustaría dedicar este trabajo a mi pareja, Aranzazu, por su apoyo incondicional durante todos estos años. Tú me has ayudado a estar donde estoy y a seguir creciendo como persona, me has animado a tomar decisiones difíciles en los momentos más delicados y me has dado alas para creer en mi mismo.

Agradecimientos

Quiero agradecer a Mobile Media Content el apoyo durante el proceso de investigación y creación de la herramienta de Editor que forma una parte importante del desarrollo del producto de la empresa de cara al cliente. Poder desarrollar una herramienta del calibre de Venue Editor ha supuesto para mi un crecimiento, tanto a nivel personal como laboral, y me ha permitido formarme como desarrollador en los últimos dos años de mi vida.

En especial, me gustaría agradecer al responsable de mi equipo, David Sánchez Blancas, por su apoyo y su confianza en mi. Tener un jefe de equipo que sabe valorar el talento y ofrecer libertad en la toma de decisiones y autonomía técnica para aportar soluciones es imprescindible para seguir creciendo como trabajador y como persona.

Agradecer también a Ricardo, mi compañero de trabajo, por todo lo que me ha enseñado y lo que he podido aprender en estos últimos dos años gracias a él.

Abstract

Mobile Media Content is a technological company focused on sport and cultural venues virtualization, as well as digital content distribution generated in Unity, for being used in ticketing webpages as a preview mode for environments.

Given a Unity project with a venue virtualization, a bunch of technological tools are gathered to automate the generation of panoramic environments from each one of its locations, upload the generated content to cloud and write specific data to perform an analysis. The process of generation and upload is achieved by using the Extending the Editor (Unity Technologies, 2023) tool powered by Unity engine. Thanks to it, an additional tool is designed and build to create and modify venues, as well as more specific tools whereupon the main generation.

Based in the data structured build for each venue, the final output consists of one or more images (depending on the selected generation mode) and a JSON file, which contains information about the 3D view and, in some cases, about the navigation between them. Once the content is uploaded to the server, the final client can integrate in his website a 3D viewer to preview the venue itself from the selected location.

Resumen

Mobile Media Content es una empresa tecnológica focalizada en la virtualización de recintos deportivos y culturales, así como la distribución de contenido digital generado en Unity, con la finalidad de ser utilizado en webs de venta de entradas a modo de previsualización de entornos.

Dado un proyecto de Unity con la virtualización de un recinto, se recoge un conjunto de piezas tecnológicas para automatizar la generación de entornos panorámicos desde cada una de sus localidades, la subida del contenido a un servicio en la nube y la escritura de datos para análisis posteriores. El proceso de generación y subida se realiza mediante el uso de la funcionalidad Extending the Editor (Unity Technologies, 2023) ofrecida por el motor gráfico Unity. Gracias a ello, se diseña una herramienta adicional para crear y modificar recintos, así como herramientas específicas que parten de la principal generación.

A raíz de la estructura de datos construida para cada recinto, se obtiene como resultado una o varias imágenes (en función del modo de generación seleccionado) y un archivo JSON que contiene información sobre la vista y, en algunos casos, sobre la navegación entre ellas. Una vez subido el contenido al servidor, el cliente final puede integrar en su página web un visor 3D para previsualizar el recinto en cuestión desde la localidad seleccionada.

Palabras clave

Memoria, Trabajo de Final de Máster, Vistas panorámicas, Unity Editor, Venues, cloud, 3D, tecnología, innovación, contenido digital.

Índice

Figuras y tablas

Índice de figuras

Índice de tablas

1. Introducción

1.1. Introducción/Prefacio

Mobile Media Content (Mobile Media Content, s.f) es una empresa tecnológica focalizada en la virtualización de recintos deportivos y culturales, así como la distribución de contenido digital generado en el motor de videojuegos Unity (Unity Technologies, s.f), con la finalidad de ser utilizado principalmente en webs de venta de entradas a modo de previsualización de los entornos.

Actualmente, la empresa dispone de un software de edición usado para dar soporte al departamento de Producción, compuesto por artistas y arquitectos. Dicho software fecha de, aproximadamente, 2015; lo que requiere por parte de la empresa una actualización y refactor de todo el sistema para incluir mejoras y potenciar la escalabilidad y generalización de éste. Como empleado e ingeniero software para la compañía, es mi misión dar forma a esta nueva versión del editor, mejorando sus prestaciones y aportando soluciones actualizadas que hagan uso de las nuevas tecnologías incorporadas a Unity en los últimos años. Así pues, el trabajo se centra en la migración de la versión 3 a la versión 4 de la herramienta de Editor, cambando por completo el paradigma del aplicativo y la arquitectura de código. Para ello se partirá de un output deseado y se implementará de nuevo toda la herramienta siguiendo esta nueva arquitectura con tal de cumplir con las mismas funcionalidades existentes en la versión previa.

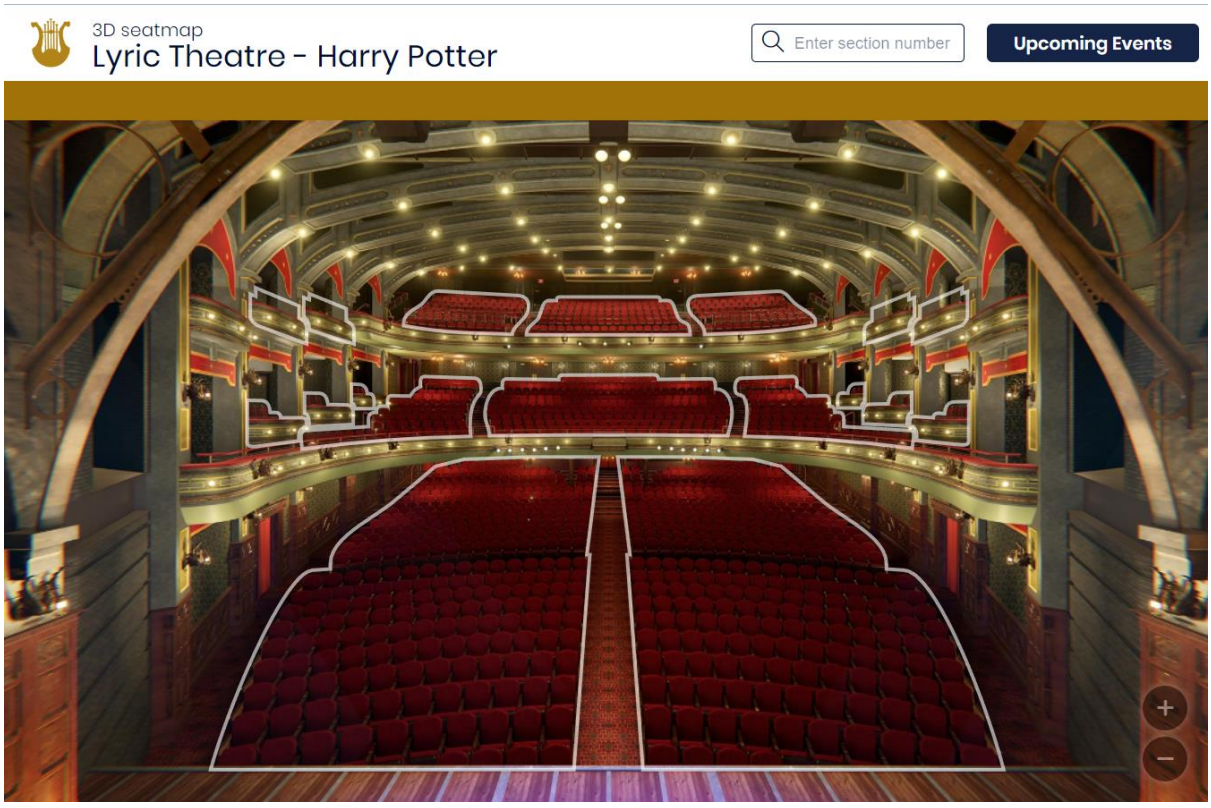


Figura 1: Mapa de asientos del Lyric Theatre

1.2. Descripción/Definición

El software en cuestión, al que llamaremos **Venue Editor** de ahora en adelante, consta de diferentes tipos de herramientas entre las cuales podemos encontrar algunas como:

- Generación automática de recintos.
- Generación de vistas panorámicas.
- Generación de mapas de bloques.
- Generación de mapas de asientos.
- Herramientas genéricas de edición para agilizar el trabajo.

Para poder entender y contextualizar el trabajo, es importante centrarnos en los dos primeros puntos que se detallan a continuación.

1.2.1. Generación automática de recintos

La base inamovible de Venue Editor consta de la generación automática de espacios virtuales a través de archivos con formato XML. Dichos archivos contienen información valiosa sobre las distintas entidades que forman parte del recinto; a saber: Grada, Sector, Fila y Silla.

Cada una de las entidades se mimetiza en una clase dentro de la aplicación y almacena toda la información leída; además de actualizarse con otros datos calculados internamente una vez hecha la carga del XML y la creación y asignación de objetos en la jerarquía de Unity. Según el tipo de entidad, en el archivo en cuestión podemos encontrar distintos tipos de datos relevantes:

- Grada: ID único de la entidad; extraído del sector evaluado.
- Sector: ID único de la entidad, valor de LOD, valor de rampa y grada asociada.
- Fila: ID único de la entidad; extraído de las sillas pertenecientes a la fila.
- Silla: ID único de la entidad, prefab a utilizar y posición y rotación en el espacio 3D.

Con todos estos datos leídos, se puede instanciar en la escena una versión preliminar del recinto deseado sobre el que trabajar. Una vez generada la estructura de datos principal del aplicativo, la gran mayoría de herramientas disponibles en Venue Editor hacen uso de ésta para su correcto funcionamiento y usabilidad.

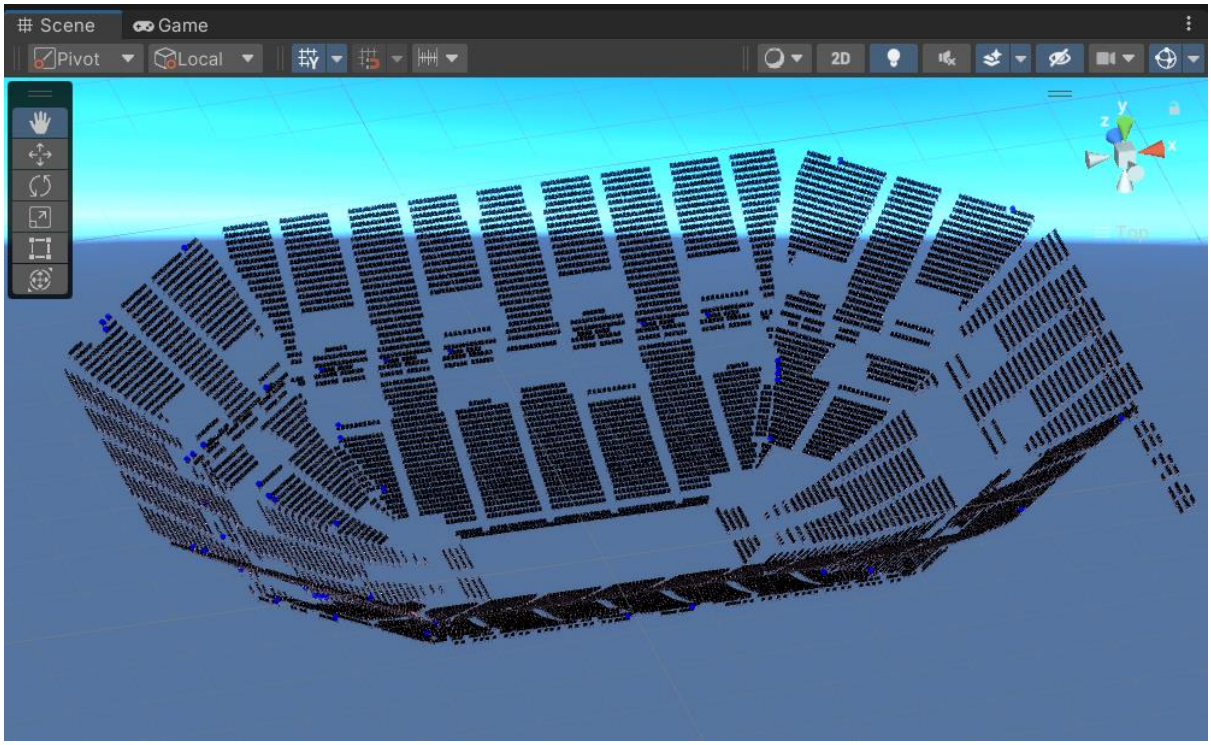


Figura 2: Estadio virtualizado de los Timberwolves

1.2.2. Generación de vistas panorámicas

Sobre la estructura de datos previamente generada se trabaja la generación de vistas panorámicas usadas en el visor 3D por el cliente final y que pueden usarse a modo de previsualización del recinto. Dicha herramienta es parte esencial del flujo de trabajo del equipo de producción y consta de una interfaz gráfica que permite realizar diferentes tipos de acciones.

El nivel más básico de uso pasa por la selección en jerarquía de cualquier tipo de entidad considerada válida para realizar la tirada de panorámicas. A continuación, en la interfaz gráfica aparecen diferentes casillas seleccionables que permiten al usuario escoger el tipo de tirada que se va a realizar. A modo de resumen, las opciones disponibles según el tipo de entidad son:

- Recinto:
    - Sectores: Una vista panorámica para cada uno de los sectores del recinto. La posición de lanzamiento se determina según la silla más cercana al centroide del sector evaluado.

    - Sectores + Sillas: Una vista panorámica para cada sector y cada una de sus sillas. La panorámica de sector se lanza de la misma manera que en el caso previo y, en el caso de las sillas, se lanza desde la posición de la silla evaluada.

- Grada:
    - Sectores: Una vista panorámica para cada uno de los sectores que contiene la grada; lanzada desde la silla más cercana al centroide.
    - Sillas: Una vista panorámica para cada una de las sillas que contiene la grada; independientemente del sector al que pertenezcan.
    - Sectores + Sillas: Siguiendo el funcionamiento de todo el recinto, se genera una vista panorámica para cada sector y sus sillas contenido en la grada.

- Sector:
    - Sector: La vista panorámica se genera a partir de la silla representativa asociada al sector seleccionado.
    - Sillas: Se genera una vista panorámica para cada una de las sillas del sector seleccionado.
    - Sector + Sillas: Se genera una vista panorámica a partir de la silla representativa asociada al sector y una vista adicional para cada una de las sillas que contiene.

- Fila:
    - Fila: La vista panorámica obtenida se genera a partir de la silla representativa asociada a la fila; en este caso, la silla más cercana al centro de la fila.
    - Sillas: Una vista panorámica para cada una de las sillas que contiene la fila seleccionada.
    - Fila + Sillas: Una vista panorámica desde la silla representativa de la fila y, adicionalmente, una vista panorámica para cada silla contenida en la entidad.

- Silla:
    - Silla: Se genera una vista panorámica desde la posición de la silla seleccionada en jerarquía.

Cada uno de estos procesos puede ser ejecutado mediante selección múltiple siempre y cuando el tipo de entidad seleccionada coincidan las unas con las otras. Es decir, se pueden generar, por ejemplo, vistas panorámicas de un conjunto de gradas y todos sus sectores y filas seleccionando en jerarquía varios objetos asociados a gradas. Lo mismo ocurre con el resto de las entidades salvo que se seleccionen tipos distintos al mismo tiempo; podría ser el caso de una fila y una silla. Si eso ocurre, la interfaz gráfica bloquea la generación de panorámicas.

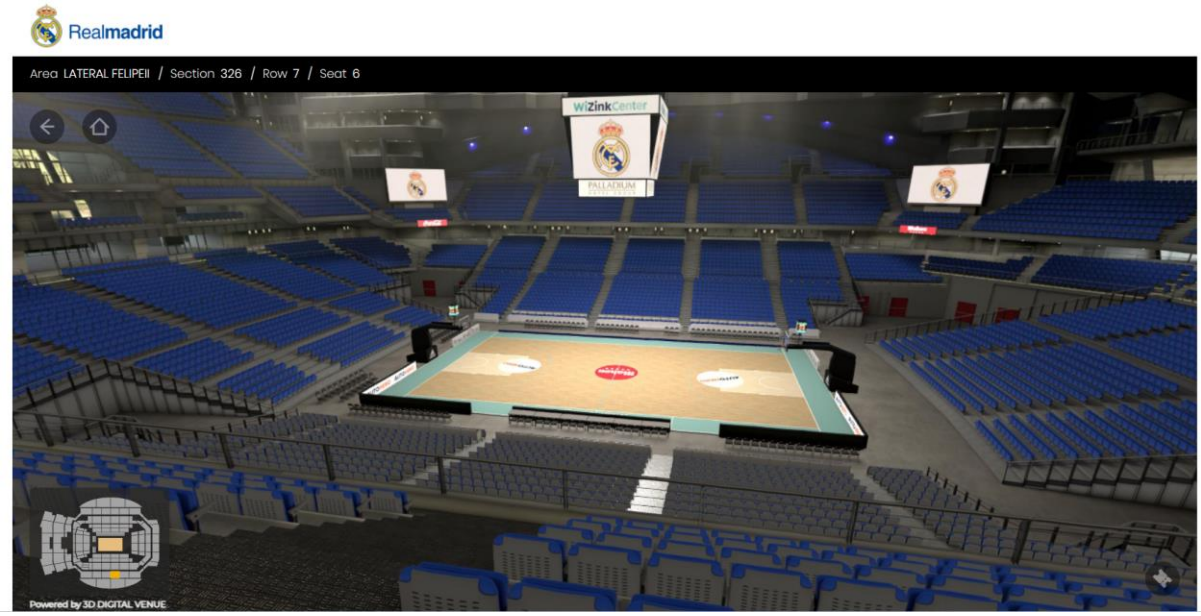


Figura 3: Vista panorámica del minisite del Real Madrid de baloncesto

Existen más tipos de funcionalidades dentro de la herramienta de generación de vistas panorámicas principal; como la subida de contenido al servidor de la empresa, la tirada de panorámicas desde objetos de referencia, el enlazado de vistas, la previsualización de contenidos generados y la configuración de sillas representativas para sectores y filas.

1.3. Objetivos generales

1.3.1. Objetivos principales

Objetivos de la aplicación/producto/servicio:

- Generación de estructura y modelo de datos de un recinto virtual.
- Gestión de escenas y states. Incluye la carga y descarga de escenas aditivas en función de la posición de la cámara, la gestión de las relaciones entre posiciones y escenas y la agrupación por tipologías para evitar cargas innecesarias.
- Gestión de colas de generación y distribución mediante el uso de procesos multithreading.
- Gestión de memoria, garbage collector y respuestas y errores de red.
- Maquetación de interfaz de usuario.
- Gestión de ficheros y serializado de clases para la escritura de datos relevantes a analizar como los tiempos de generación y subida.
- Metodología de test y QA de desarrollo e implantación en equipos de producción.

Objetivos para el usuario:

- Generación automática de vistas panorámicas de forma eficaz y eficiente.
- Agilidad en el proceso de generación de vistas.
- Control de errores.

Objetivos para el cliente:

- Previsualizado de vistas panorámicas integrado en web.
- Navegación entre vistas panorámicas en espacios de interior.

Objetivos personales del autor del TF:

- Desarrollo de un aplicativo siguiendo la metodología SOLID.
- Diseño de código robusto y eficiente.
- Diseño de interfaz intuitivo y fácil de utilizar.
- Conocimiento avanzado del framework de Editor de Unity.

1.3.2. Objetivos secundarios

- Mejoras de rendimiento.
- Mayor eficacia y rapidez con respecto a la versión 3 de Venue Editor.
- Portabilidad a futuras versiones de Unity.
- Metodología de test adicional mediante Unit Testing.
- Uso de metodología GitFlow.

1.4. Metodología y proceso de trabajo

El desarrollo completo del aplicativo y la herramienta de generación de vistas panorámicas se ha propuesto la adaptación del antiguo Venue Editor usado por Mobile Media Content. Dado que las funcionalidades existentes en el antiguo editor siguen siendo necesarias hoy en día, el proyecto es una readaptación del contenido antiguo.

Así pues, y con tal de no cometer los mismos errores, la base de la aplicación es totalmente distinta. La estructura de datos y su gestión cambian por completo el paradigma del editor. A diferencia de la versión anterior, el nuevo modelo de Venue Editor se basa en el uso de entidades y managers para la resolución de tareas de gestión y scripts de UI que contemplan el uso por parte del usuario final. De esta forma, se asegura el principio de responsabilidad única entre clases; siendo las entidades las encargadas de gestionar datos propios de cada una, los managers los encargados de la gestión más directa de funcionalidades de cada herramienta y los scripts de vista los encargados de organizar la disposición de información en la interfaz de usuario. Esto permite la realización de una nueva herramienta de generación de panorámicas que tiene que objetivo final cumplir con las funcionalidades y existentes en la versión previa de la herramienta de editor, pero remodelar por completo la arquitectura y la forma de programar el aplicativo.

Esta estrategia se considera más adecuada con respecto a la creación de un software completamente nuevo puesto que el motor gráfico utilizado por la empresa sigue siendo el mismo y, además, sigue existiendo la posibilidad de añadir nuevas funcionalidades bajo demanda si fuera necesario. Por otra parte, las antiguas herramientas carecen, en parte, de escalabilidad en algunos aspectos. Readaptar el editor permite conservar funcionalidades a la vez que replantear la estructura de código para un funcionamiento más estable, más robusto y dotado de un plus de rapidez con respecto a la versión antigua.

El producto final del aplicativo aún está por elaborar. Sin embargo, con los avances realizados en los últimos meses, ya es posible abarcar sin problemas la generación de vistas panorámicas. Para la correcta gestión de las tareas se ha usado Jira (Atlassian, s.f), acompañado de un método de programación ágil organizado por sprints semanales que reparten las tareas de forma escalonada. Dada la complejidad del asunto y el gran volumen de trabajo que supone, la jerarquía propuesta permite avanzar en el proyecto de forma paulatina, acotando objetivos a medio/corto plazo que sirven de base para completar objetivos más grandes.

Para cada subtarea, se hace uso de la metodología GitFlow (Atlassian, s.f), que permite tener mayor control sobre las versiones del código que se va generando. Cada set de commits integrados en un push requiere un pull request por parte del supervisor para comprobar que el código es correcto y eficiente. Además, esto permite mantener una versión estable del proyecto global sin introducir artefactos de la herramienta secundaria mientras no se asegure que funciona correctamente.

Para el cumplimiento de hitos más específicos, se recurre tanto a la documentación de Unity como de C#, que permiten valorar las opciones disponibles para la realización de cada una de las tareas y determinar cuál de ellas resulta más óptima en cada caso. Por otro lado, tareas como la metodología de subida de contenido al servidor han sido determinadas a base de realizar varios tipos de pruebas que permiten comprobar cuál de ellas ofrece mejores resultados en tiempo y gestión de recursos.

1.5. Planificación

Con tal de planificar correctamente el trabajo a realizar, se asume que la duración del proyecto es aproximadamente de cuatro meses. Esto implica, a priori, cuatro fechas clave dentro de la planificación total; una cada final de mes, empezando en marzo y terminando en junio.

En cada una de las fechas claves se termina uno de los grandes bloques de trabajo o hitos; a saber:

1. Plan de proyecto (PEC 1).
2. Estado del arte y primera versión del proyecto (PEC 2).
3. Implementación de versión usable (PEC 3).
4. Memoria y producto final (PEC 4).

Cada uno de bloques se divide en hitos parciales que incluyen un mayor desglose de las tareas a realizar:

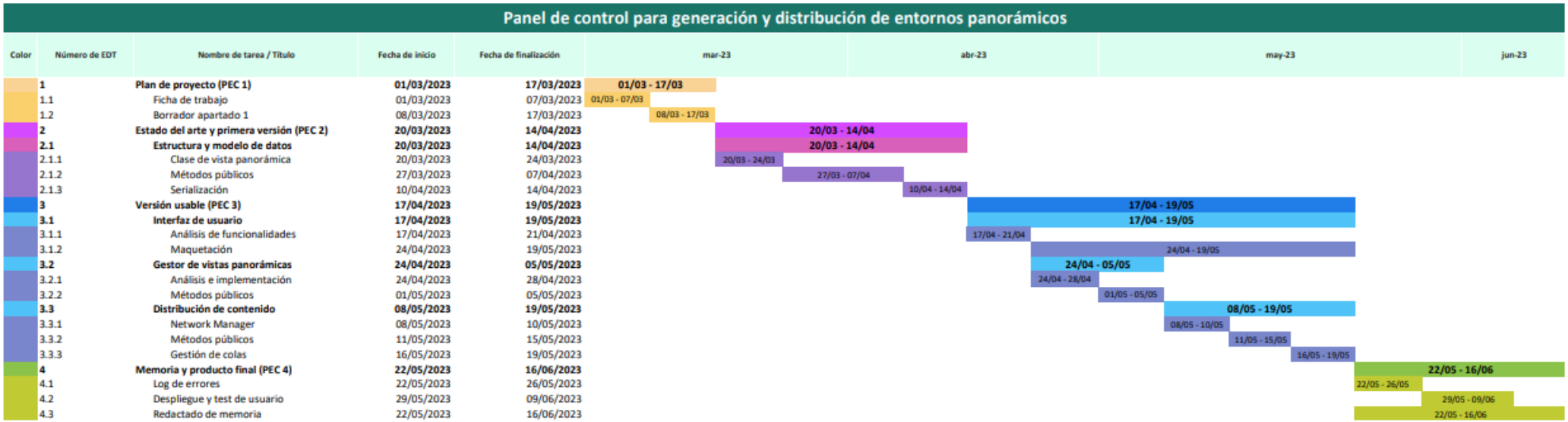
1. Plan de proyecto
   - 1.1. Cumplimentar ficha de trabajo.
   - 1.2. Borrador del apartado 1 de la memoria.

2. Estado del arte y primera versión del proyecto
   - 2.1. Estructura y modelo de datos de una vista panorámica:
     - 2.1.1. Creación de una clase asociada a la vista panorámica, análisis e implementación de propiedades de esta.
     - 2.1.2. Implementación de métodos públicos de generación de contenido.
     - 2.1.3. Implementación de métodos de serialización y almacenamiento de datos.

3. Implementación de versión usable
   - 3.1. Interfaz de usuario:
     - 3.1.1. Análisis de funcionalidades de la herramienta y creación de una clase asociada a la interfaz.
     - 3.1.2. Maquetación e integración de herramientas en el editor.
   - 3.2. Gestor de vistas panorámicas:
     - 3.2.1. Creación de una clase asociada a la gestión de panorámicas, análisis e implementación de propiedades asociadas a la interfaz.
     - 3.2.2. Implementación de métodos públicos para distintos tipos de generación.
   - 3.3. Distribución de contenido:
     - 3.3.1. Diseño e implementación de un gestor de red.
     - 3.3.2. Implementación de métodos públicos para generación y subida de contenido en el gestor de vistas.
     - 3.3.3. Gestión de colas de subida.

4. Memoria y producto final
    4.1. Log de errores e información para el usuario final.
    4.2. Despliegue y test de usuario.
    4.3. Redactado de la memoria.

A continuación, se muestra un diagrama de Gantt que ilustra todo el proceso:



Mesa 1: Diagrama de Gantt

1.6. Presupuesto

El coste del unitario de desarrollo del producto se estima en 4469€. Dicho precio incluye el coste horario del equipo humano, el coste asociado al equipo técnico y otros recursos necesarios para la realización de la tarea.

El desglose total del equipo humano consta de 300 horas de dedicación por parte del trabajador que realiza la tesis y 16 horas de supervisión por parte del responsable de tutorizar el trabajo dentro de la empresa. Partiendo de un sueldo medio de 9€ la hora, multiplicado por el total de horas, el coste total asciende a 2700€. Asumiendo un sueldo medio de 13,5€ la hora de supervisión, multiplicado por un total de horas, se añade un adicional de 216€ al estimado anterior.

En cuanto al equipo técnico y demás recursos, se incluye el ordenador para realizar el trabajo, cuyo coste es aproximadamente de 1000€; y la licencia de Unity Pro, cuyo coste es de 415€ el trimestre. Asumiendo que la duración es de 4 meses, el total de la licencia asciende a 553€.

Sumando todos los costes obtenemos el valor inicial de 4469€, que equivale al coste que asume la empresa para mantener a los trabajadores involucrados y el equipo necesario para la realización del mismo.

1.7. Estructura del resto del documento

El resto del documento sigue la siguiente estructura:

1. Análisis de mercado: Análisis de la competencia de Mobile Media Content y el valor añadido de la generación de vistas panorámicas con respecto a otras empresas.

    1.1. Público objetivo: Descripción de los clientes de Mobile Media Content, tanto internos como externos, a los que va dirigido el producto.

    1.2. Competencia: Análisis de otras empresas del sector; qué ofrecen en comparación con Mobile Media Content. Visión global de la situación actual del proyecto y análisis de mercado.

2. Propuesta: Resumen de la propuesta del TF en base al análisis de mercado realizado.

    2.1. Objetivos y especificaciones: Detalle de los objetivos a cumplir a lo largo del proyecto y sus particularidades.

3. Diseño: Explicación más en detalle de las especificaciones y prestaciones del producto.

    3.1. Arquitectura general: Explicación general de la arquitectura de código usada para la generación de recintos.

    3.2. Diagramas de navegación: Diagrama de la arquitectura general que ilustra de forma intuitiva el recorrido de la parte más Core del aplicativo.

    3.3. Diseño gráfico e interfaces: Propuesta de diseño de ventana de Editor para el uso del generador de vistas panorámicas.

    3.4. Lenguajes de programación y APIs: Breve explicación del framework utilizado para el desarrollo, así como el lenguaje de programación asociado.

4. Implementación: Explica en detalle el funcionamiento de la herramienta de generación de vistas panorámicas:

    4.1. Qué es una vista panorámica: Explicación de qué se considera una vista panorámica, tipos de vistas panorámicas y cómo se traduce a nivel de aplicación dentro de Unity.

    4.2. Gestor de vistas panorámicas: Detalle sobre la construcción de un manager encargado de gestionar el lanzamiento de vistas y su correlación con la interfaz de usuario.

    4.3. Distribución de contenido: Explicación general de la distribución de contenidos y su subida a servidor. Implementación de un gestor de red y metodologías de generación y subida.

    4.4. Requisitos de instalación: Recursos necesarios para instalar la herramienta y usarla.

    4.5. Instrucciones de instalación: Packaging y detección de versiones automáticas mediante el uso de git.

5. Demostración: Descripción de usabilidad del producto y su integración.

    5.1. Instrucciones de uso: Explicación detallada de cómo usar el aplicativo.

    5.2. Testing: Proceso detallado de testeo del aplicativo antes de su puesta en marcha e integración en el equipo de trabajo.

6. Conclusiones: Conclusiones personales sobre el proyecto realizado, el proceso realizado y los resultados obtenidos. Lecciones aprendidas, logro de objetivos y análisis de seguimiento de la planificación inicial.

    6.1. Líneas de futuro: Análisis de posibles mejoras o ampliaciones futuras de la herramienta.

2. Análisis de mercado

Con tal de entender mejor la situación general de mercado, los competidores y el público al que se dirige la herramienta, a continuación, se detalla información valiosa con respecto al uso e integración de la generación de vistas panorámicas de recintos.

2.1. Público objetivo y perfiles de usuario

Tal y como se ha mencionado con anterioridad, el producto final de la herramienta va destinado a todas aquellas empresas dedicadas al sector del ticketing 3D a nivel mundial. Esto quiere decir que el target de audiencia incluye clientela en, prácticamente cualquier lugar del mundo. En el caso de Mobile Media Content, la gran mayoría de clientes se concentran en Estados Unidos, aunque recientemente está entrando en juego de manera notable el mercado europeo.

Para hacer una correcta segmentación, debemos separar el público objetivo y los perfiles de usuario en dos tipos: clientela externa y clientela interna.

- Clientela externa: Formada por todas aquellas empresas que quieren integrar en su página web el contenido generado por parte de la empresa. Este tipo de cliente tiene a su disponibilidad una librería desarrollada en JavaScript para integrar y personalizar todo el contenido generado internamente a través de la herramienta. Son ellos quieren reportan un beneficio económico directo a Mobile Media Content. Todos ellos disponen de sus propios portales web, los cuales incluyen las vistas panorámicas generadas con la herramienta.

  Entre ellos, se pueden encontrar clientes más conocidos a nivel nacional como el Real Madrid (Real Madrid, s.f) y el Atlético de Madrid (Atlético de Madrid, s.f); y clientes a nivel internacional como el Manchester City (Manchester City, s.f) o los Chicago Cubs (Major League Baseball, s.f). Además de clubes deportivos, también existen clientes en el ámbito de los teatros y museos como el Lyric Theatre de Nueva York (Broadway Shows, s.f).

- Clientela interna: Formada por el equipo de producción de Mobile Media Content; aquellos que usan directamente la herramienta para generar las vistas panorámicas. Aquí también deberían incluirse los integradores de contenido de la propia empresa ya que, dentro del equipo de trabajo, existe un departamento dedicado a la integración de mapas 2D y 3D en webs de clientes externos.

2.2. Competencia

La industria del ticketing 3D se remonta prácticamente a hace más de 20 años. Aunque la necesidad de vender entradas lleve existiendo desde hace muchos años, la integración de herramientas

digitales para facilitar la toma de decisiones y la compra de asientos a aquellos usuarios que buscan el mejor sitio para visionar el evento es relativamente reciente. Existen muchas empresas dedicadas a ello, aunque la gran mayoría son residuales. Se puede decir que, a día de hoy, existen cuatro grandes empresas que controlan todo el mercado mundial: Mobile Media Content, IOMedia, Pacifa y Ballena.

- IOMedia (IOMedia, s.f): Empresa estadounidense fundada en 1997. Trabajan con motores de render para generar vistas panorámicas de estadio a nivel de sector. Actualmente ha sido absorbida por Ticketmaster (Ticketmaster, s.f) y trabaja directamente para la clientela de la empresa madre.

- Ballena (Ballena, s.f): Empresa estadounidense fundada en 1999. Trabajan con motores de render para la visualización virtual de recintos a nivel de sector. Actualmente ha sido absorbida por Paciolan (Paciolan, s.f), otra empresa estadounidense especializada en deportes universitarios.

- Pacifa (Pacifa Decision, s.f): Empresa francesa fundada en 2011. Trabajan con motores de render enfocado a la virtualización y visualización de recintos a nivel de sector. Actualmente ha sido absorbida por Platinum Group (Platinum Group, s.f), trabajando exclusivamente para ellos. Su área de mercado está fundamentalmente arraigada en Francia y Mobile Media Content ha conseguido arrebatar un gran porcentaje de los clientes que no estaban basados en la región francesa.

Todas las empresas de la competencia no utilizan motores de videojuegos para la generación de contenido y vistas panorámicas sino motores de render. Esto supone una serie de inconvenientes que ponen en ventaja el producto elaborado por Mobile Media Content. La principal desventaja es la imposibilidad de generar vistas panorámicas para cada una de las sillas de los estadios debido a la gran cantidad de tiempo que consume generar contenido en motores de render en vez de un motor de videojuegos como Unity. Poder generar contenido por silla beneficia en gran parte la venta de abonos, permitiendo a los clientes tener una visión más realista del recinto desde un asiento específico, y la generación de nuevos estadios.

Además, el hecho de no usar un motor de videojuegos como Unity para la generación de contenido hace más costosa tanto la edición como la previsualización de este; esto se debe a la imposibilidad de visualizar el contenido hasta que no ha sido totalmente renderizado. Dada la gran cantidad de tiempo que consume renderizar vistas panorámicas para todos los sectores del estadio (15 minutos aproximadamente para una sola panorámica) supone un impedimento para los clientes a la hora de pedir cambios rápidos en los contenidos generados. Además, dada la exigencia de la tarea, los costes de generación para cambios muy sutiles son demasiado elevados.

Mobile Media Content se beneficia de ello utilizando Unity como herramienta principal de generación de vistas panorámicas, no solo a nivel de sector sino a nivel de asiento. El contenido para los clientes se genera de forma más rápida y eficiente y con un menor coste dado el ahorro de tiempo que supone el uso del motor de videojuegos con respecto al motor de render. Aún así, para casos más específicos como generación de contenido para sectores interiores y vistas más generales de los estadios, sí se utiliza Blender como motor de render para obtener contenido de una mayor calidad.

Hoy en día, los competidores de Mobile Media Content sobreviven en la industria dada la tendencia general por parte de los clientes a pensar que el contenido generado mediante motores de render es de una calidad superior al contenido generado mediante el motor de videojuegos. A pesar de ello, tanto Pacifa como Ballena, ambas absorbidas por empresas más grandes, están al borde de la extinción debido a la falta de utilidad que las empresas madre han visto en ellas y la poca evolución tecnológica por la que han apostado en estos últimos años. Como prueba de esto, basta decir que dos de los fundadores de Ballena ahora forman parte del equipo de trabajo de Mobile Media Content.

Por otro lado, competidores como IOMedia, adquirido por Tickermaster, también están entrando en recesión debido a la envergadura de los proyectos con lo que trabaja la empresa madre. Como se ha mencionado con anterioridad, para recintos grandes es vital la velocidad de generación de contenidos, a pesar de ir en detrimento de la calidad de las imágenes. Al trabajar directamente en recintos con volumen de aforo elevado y con una gran demanda de cambios por parte del cliente final, trabajar con motores de render hace muy costosa la tarea.

En estos casos, poder realizar panorámicas por silla y sector en menos de 24 horas ofrece un claro valor añadido con respecto al resto de competidores. Además, Mobile Media Content no ha sido absorbida por ninguna otra empresa hoy en día. La potencial clientela de la empresa no se ve limitada a los clientes impuestos por una empresa más grande; esto permite atacar a clientes en todo el mundo sin depender de agentes externos.

Otra ventaja competitiva de la que dispone Mobile Media Content es la posible integración por parte de los clientes de los contenidos en su página web. El departamento de ingeniería ha desarrollado una librería en Javascript cuya API pública (Mobile Media Content, s.f) pueden usar los clientes tanto externos como internos para personalizar mucho más sus páginas y contenidos. Esto, lógicamente, ofrece mucha flexibilidad al usuario final, que también dispone de servicio de soporte técnico para solucionar cualquier incidencia que pueda surgir durante el proceso de integración.

Evidentemente, tal y como se ha comentado con anterioridad, la gran debilidad de Mobile Media Content es la menor calidad de los contenidos generados mediante el motor de videojuegos en comparación con aquellos generados usando motores de render. La gran amenaza para la empresa es la posible compra por parte de grandes empresas como Ticketmaster del resto de competidores; los cuales, en el caso de que esto llegara a suceder, absorberían el trabajo de toda la clientela que

pueda tener la empresa madre. Esto limitaría mucho las posibilidades de mercado que existen actualmente y la libertad a la hora de captar clientes.

3. Propuesta

3.1. Definición de objetivos/especificaciones del producto

Tal y como se ha detallado con anterioridad, el proyecto consta de dos objetivos muy distintos entre sí: la mejora y agilidad en la creación de contenidos por parte del equipo de producción y la posterior integración en web por parte de los clientes.

En este caso, es muy importante la diferenciación entre cliente interno y cliente externo. El generador de vistas panorámicas tiene uso exclusivo por parte de los clientes internos, formado por los departamentos más artísticos/técnicos de Mobile Media Content. La herramienta está destinada a ser usada única y exclusivamente por ellos. Aún así, el trabajo que estos departamentos realizan mediante su uso tiene repercusión directa en el producto final extraído de Unity. Junto con el uso de otras herramientas como el generador de mapas de asientos, los clientes externos pueden usar la API web desarrollada por el departamento de IT para integrar dichos contenidos en sus páginas web y plataformas de venta de entradas. Alternativamente, pueden solicitar al departamento de IT que sean ellos quienes integren los mapas generados.

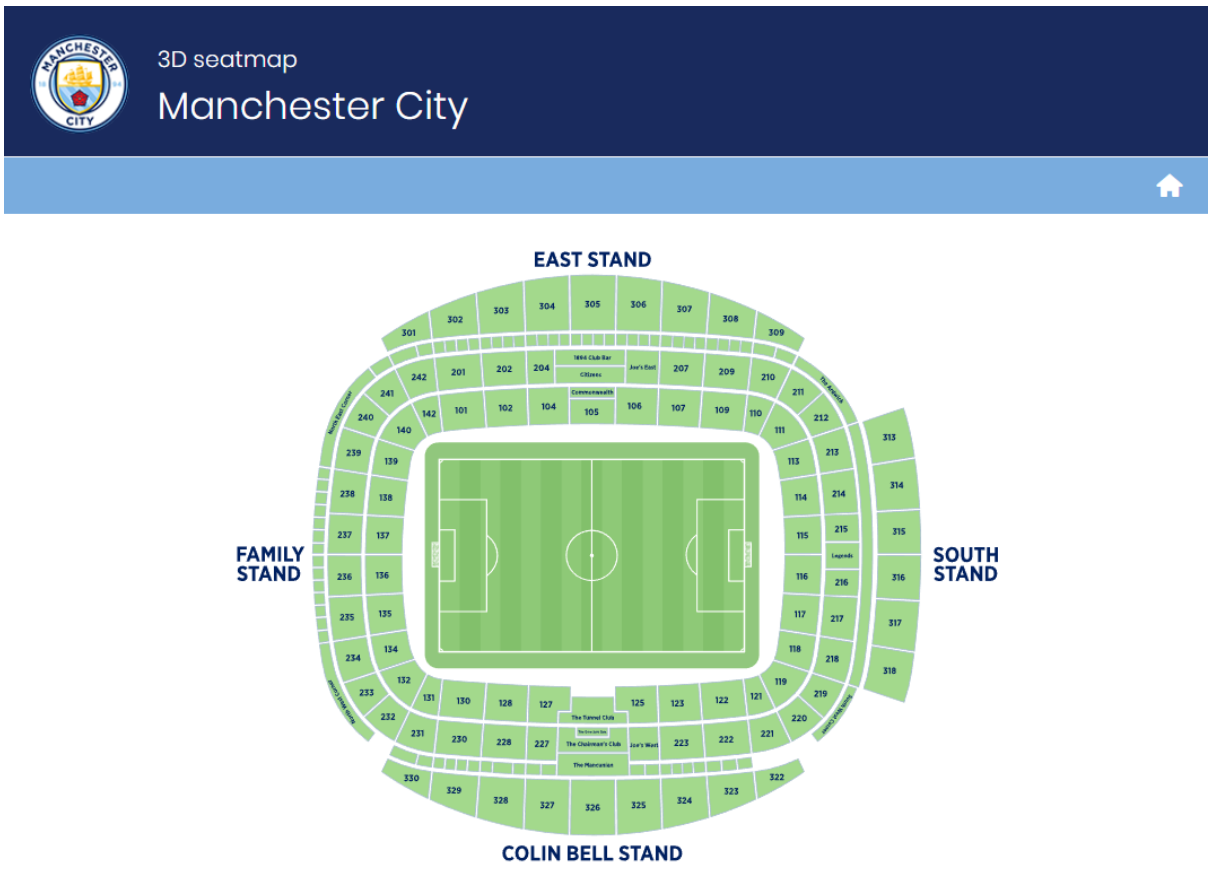


Figura 4: Mapa de sectores del minisite del Manchester City

De esta forma, los objetivos principales del proyecto quedan relegados al uso del cliente interno, que generará los contenidos finales que se usan a posteriori. A saber:

1. Generar vistas panorámicas: A partir de un recinto virtualizado en Unity mediante Venue Editor, se debe poder utilizar la herramienta para la generación de vistas 3D tanto cúbicas como esféricas. Según el tipo de vista panorámica que se desee generar, además de las imágenes también obtendremos un JSON con información valiosa con respecto a la vista. Dicho archivo permite saber el tipo de panorámica generada, desde qué posición y rotación; y desde qué entidad del aforo ha sido lanzada (ID).

2. Generar miniatura: A modo de previsualización del recinto previo a la visualización de la vista panorámica, el usuario final debe ser capaz de ver una miniatura que muestre una vista frontal desde la silla representativa el sector. Dicha miniatura se extrae de la propia generación de imágenes, reduciendo la resolución y guardándola en otra ubicación distinta.

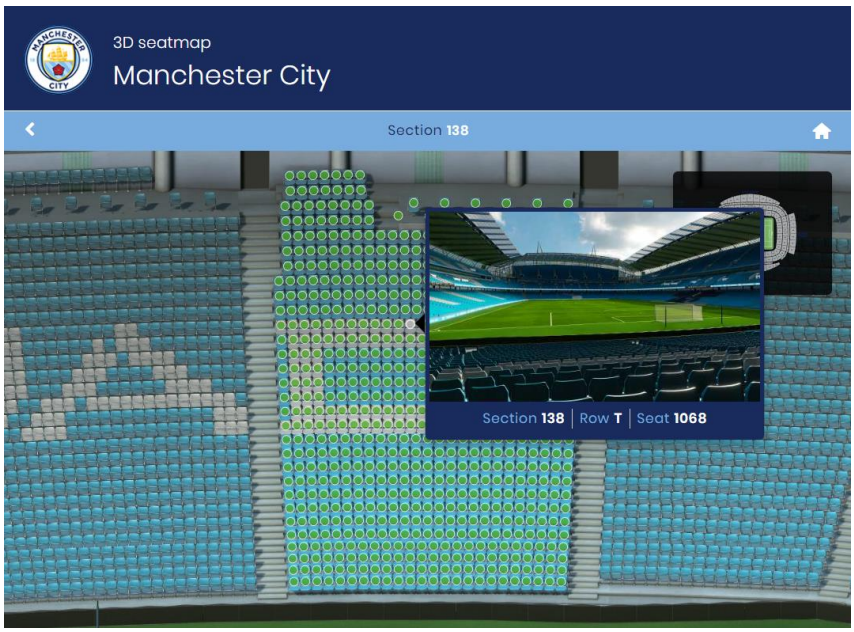


Figura 5: Miniatura del mapa de asientos del Manchester City

3. Selección del tipo de entidad: Los artistas y técnicos deben poder escoger sobre qué parte del aforo quieren lanzar la vista panorámica. Así pues, se debe ofrecer la posibilidad de tirar vistas panorámicas en cascada desde sillas representativas de sectores, desde cada una de las sillas que lo conforman de forma individual o ambas. De forma añadida, debe existir la posibilidad de generar vistas panorámicas en cascada desde varias entidades siempre y cuando compartan tipo. Es decir, permitir la selección múltiple desde la jerarquía de Unity y la generación de contenidos según lo que se haya seleccionado.

4. Enlazado de vistas: Para evitar redundancias, los archivos JSON de configuración deben contener información con respecto a panorámicas enlazadas entre sí. De esta forma, si se

selecciona la generación de vista panorámica por sector y sillas, la silla que equivale a la representativa del sector sólo contendrá información de la vista panorámica a la que hace referencia mediante el uso de un identificador único. Esta funcionalidad es muy útil para ahorrar tiempo en la generación y subida de contenidos al servidor, además de evitar duplicados a la hora de generar imágenes. En vez de generar una imagen para cada entidad, en el caso de que exista una vista enlazada y esta se haya generado, se usarán las imágenes de la vista referenciada.

El enlazado de vistas se puede hacer de forma automática en el caso de las sillas representativas de sectores y filas, o se puede especificar mediante un archivo de tipo CSV que se carga en el aplicativo y contiene información sobre qué vistas hacen referencia a otras.

5. Selección de silla representativa: Cada una de las entidades que contengan sillas dentro de un aforo dispone de un ID de silla que hace referencia a la silla más adecuada para la generación de la panorámica. Generalmente, dicha silla suele corresponder a la más cercana al centroide del sector o la fila, pero, en casos más específicos, el usuario debe poder escoger cuál es la silla deseada para la generación. Para ello, se debe serializar el ID almacenado en la estructura de datos correspondiente a dicha silla y se debe permitir cambiarlo, mediante selección en jerarquía, por cualquier otro.

6. Panorámicas de interior: En algunos casos, existen sectores de interior en los recintos. Dichos sectores conforman una casuística especial a la hora de generar vistas panorámicas. Este tipo de sectores contienen información adicional de navegación entre asientos que se utiliza a posteriori para moverse dentro del espacio. En la mayoría de los casos, se utiliza un archivo CSV que contiene información sobre las relaciones entre asientos para saber a cuáles se puede navegar desde el actual; alternativamente, se asume que se puede navegar a todas las localidades. Cuando esto ocurre, se tiene en cuenta una mesh de navegación generada por el departamento de artistas que se usa para detectar oclusiones entre nodos y bloquear la navegación. Dicha mesh debe subirse al servidor junto con un JSON que contenga información adicional que permita la navegación.

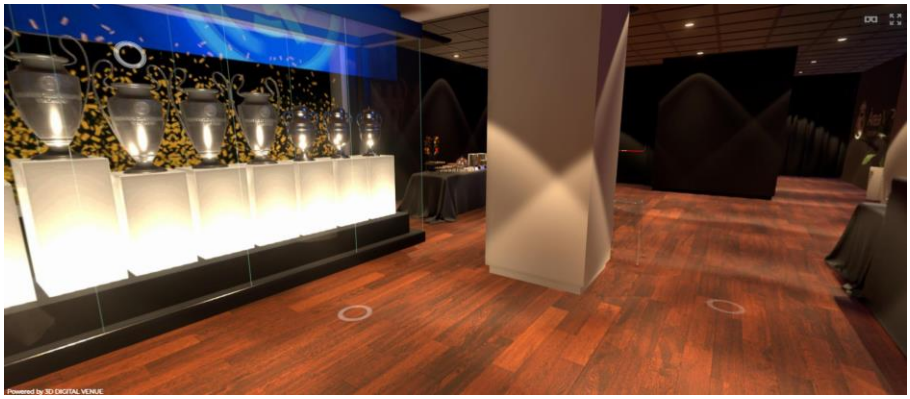


Figura 6: Panorámica de interior de la sala de trofeos del Real Madrid

7. Uso de objeto de referencia: En algunas situaciones, para clientes del mercado secundario (reventa de entradas), el equipo de producción utiliza una pseudo-reconstrucción del estadio basada en polígonos que representan cada uno de los sectores del estadio. Cuando esto ocurre, se debe calcular el sector más cercano al polígono en cuestión y utilizar la silla representativa de dicho sector para generar la panorámica.

8. Subida y distribución: Con tal de que se puedan solicitar los contenidos mediante peticiones http, la generación de panorámicas debe subirse al servidor deseado según la región seleccionada (US_WEST, AP_NORTHEAST, US_EAST, EU_WEST y EU_CENTRAL). Cada uno de los servidores está ubicado en una zona del mundo para hacer más rápida y accesible la petición.

9. Gestión de colas: La generación de panorámicas es muy costosa a nivel computacional para el ordenador que realiza la tarea. Para evitar colapsos de memoria se debe diseñar y probar un sistema de colas para no colapsar las máquinas. Dicho sistema de colas se encarga de generar cierto número de vistas panorámicas hasta un límite y subir los contenidos antes de empezar a generar la siguiente tirada.

10. Estimación de tiempos: Según el número de vistas, el tiempo que se tarda en generar todos los contenidos es muy elevado. El equipo de producción debe tener a su disposición una barra de progreso que estime el tiempo total de generación y subida de la selección de entidades. De esta manera, se pueden hacer a la idea de cuánto tiempo les va a suponer el proceso. Además, el progreso debe ser cancelable si el usuario lo desea; esto es esencial para poder parar la generación en caso de haber cometido un error.

11. Reporte de datos: Se debe generar un TXT en local que permita saber al usuario si se ha completado el proceso satisfactoriamente. Dicho archivo de texto contiene información sobre las entidades seleccionadas al inicio del proceso, la última panorámica lanzada y si se ha completado o no el proceso. Esto permite al usuario tener conocimiento sobre cuál ha sido la

última vista generada en caso de querer reactivar la generación o en caso de que ocurra un error en mitad del proceso.

12. Previsualización local: Al generar panorámicas se debe permitir la opción de guardar los contenidos en local y previsualizarlos. Existe un aplicativo desarrollado por Mobile Media Content llamado Pano Viewer. Se debe poder lanzar el aplicativo una vez se han generado las vistas para comprobar en local que los contenidos son correctos.

Figura 7: Previsualización de panorámica en Pano Viewer

4. Diseño

Con tal de definir de forma clara la arquitectura de la herramienta, es necesario echar un primero un vistazo a la arquitectura general de Venue Editor. Este aplicativo es bastante complejo y consta de diferentes sub-herramientas que ayudan al departamento de producción en la generación de contenidos. Aun así, existe una parte más Core dentro de la herramienta de Editor que vale la pena comentar. Todo este apartado es completamente nuevo con respecto a la versión de Venue Editor anterior. En la antigua herramienta todo estaba separado en 5 o 6 scripts que contenían métodos para realizar absolutamente todas las acciones esperadas, leyendo continuamente la jerarquía para encontrar las entidades que forman parte del recinto y mezclando conceptos en scripts difíciles de seguir y demasiado extensos. Así pues, la propuesta de arquitectura en Venue Editor 4 encapsula y define mucho mejor qué parte del aplicativo debe encargarse de qué y facilita muchísimo la lectura de datos. Por último, además de ofrecer mucha más flexibilidad, las acciones se realizan de forma más rápida, ya que muchas de ellas requerían hacer uso de “GameObject.Find()” en la versión anterior y ahora toda esa información queda almacenada una vez se realiza la carga de un recinto.

4.1. Arquitectura general de Venue Editor

Podemos dividir Venue Editor en dos partes: entidades y herramientas.

4.1.1. Entidades

Al tratarse de estadios, existen diferentes tipos de entidades que forman parte del código general de la aplicación y que se usan para generar recintos virtuales. Esta segmentación por entidades se ha introducido por completo en la versión 4 de la herramienta. Tal y como se ha comentado anteriormente, en la versión 3 toda la información se leía directamente de la jerarquía. Las clases que forman las entidades son:

- Grada: Representa un conjunto de sectores agrupados según la grada del recinto.

- Sector: Reúne un conjunto de filas formando un espacio determinado dentro de la grada.

- Fila: Conjunto de sillas contenidas en una línea dentro de sector. En este caso, se puede distinguir entre filas normales (determinadas por el cliente) y filas “físicas” (creadas a partir de un algoritmo en función de la distancia de los vecinos de cada silla).

- Silla: Unidad más básica del recinto. Representa un asiento en concreto contenido dentro de una fila o fila física.

A estas entidades se debe añadir el recinto en sí y la configuración correspondiente.

- Recinto: Contiene información y métodos genéricos para todo el espacio. Se encarga de almacenar información con respecto a las configuraciones, las escenas, el orden predominante de numeración de entidades y otro tipo de información relevante con respecto al recinto global.

- Configuración: Script enlazado a la escena en uso. Contiene estructuras de datos de las entidades que forman el recinto. Es el nexo de unión entre las diferentes partes del aplicativo y dispone de métodos para realizar acciones sobre entidades específicas o grupos de entidades. Pueden existir múltiples configuraciones por recinto según el uso que se le quiera dar. Cada configuración se corresponde con una escena y conjunto de subescenas que se denominan “States”. La estructura de datos general se serializa para cada una de las configuraciones y evita la realización de cálculos innecesarios previamente realizados tras la primera carga.

Todas estas entidades almacenan en estructuras de datos la información captada tras la carga de un XML y tras la realización de cálculos posteriores.

4.1.2. Herramientas

Existen varios tipos de herramientas disponibles en Venue Editor. Cada una de ellas forma parte de una pestaña dentro de la ventana de Editor y sólo es accesible según el tipo de usuario que vaya a usar la aplicación. A pesar de que las funcionalidades principales de las herramientas son prácticamente las mismas (a excepción de algún pequeño cambio), la forma de programarlas y trabajar con ellas es muy diferente. En la versión 3 del aplicativo, los scripts asociados a las herramientas no estaban lo suficientemente bien definidos, siendo así un compendio de métodos estáticos mezclando conceptos asociados a las entidades y conceptos asociados a los managers. En la versión 4, hay una clara diferenciación entre los scripts de interfaz, los managers (gestores y enlace entre interfaz y entidad) y las entidades en sí. A continuación, se detallan las herramientas que forman parte del editor:

- Seating: Herramientas para la virtualización de recintos que permiten, principalmente, realizar acciones sobre la estructura de datos básica del aplicativo, entre ellas cargar archivos XML de espacios a instanciar. Permite realizar acciones sobre los elementos del recinto a nivel más global como, por ejemplo, detectar potenciales errores en el XML y resolverlos in situ. Esta herramienta es capaz de guardar archivos XML modificados una vez arreglados los problemas detectados en el XML de origen, agrupar sectores en diferentes áreas para la generación de mapas, renombrar entidades, detectar bloques de sillas, etc.

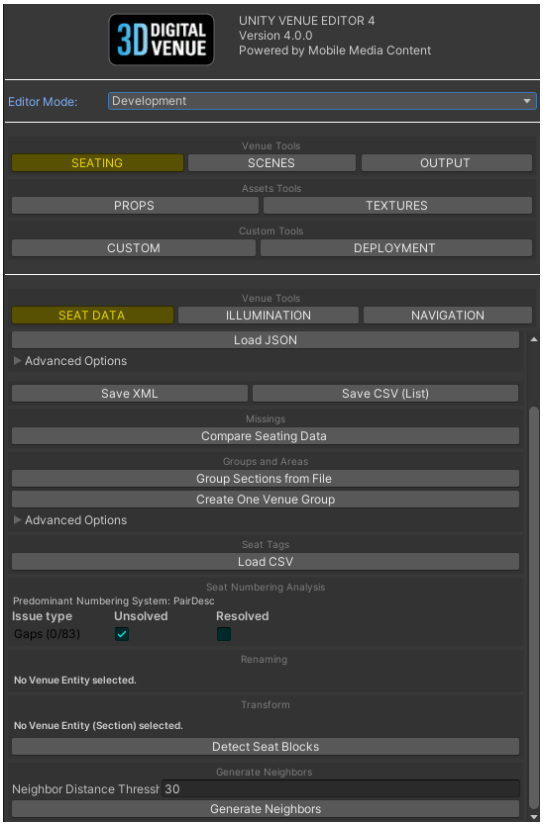


Figura 8: Vista de la herramienta de Seating

- Scenes: Permite generar diferentes configuraciones sobre un mismo proyecto, modificar el nombre de la configuración actual, configurar subescenas y crear States a partir de la configuración actual.

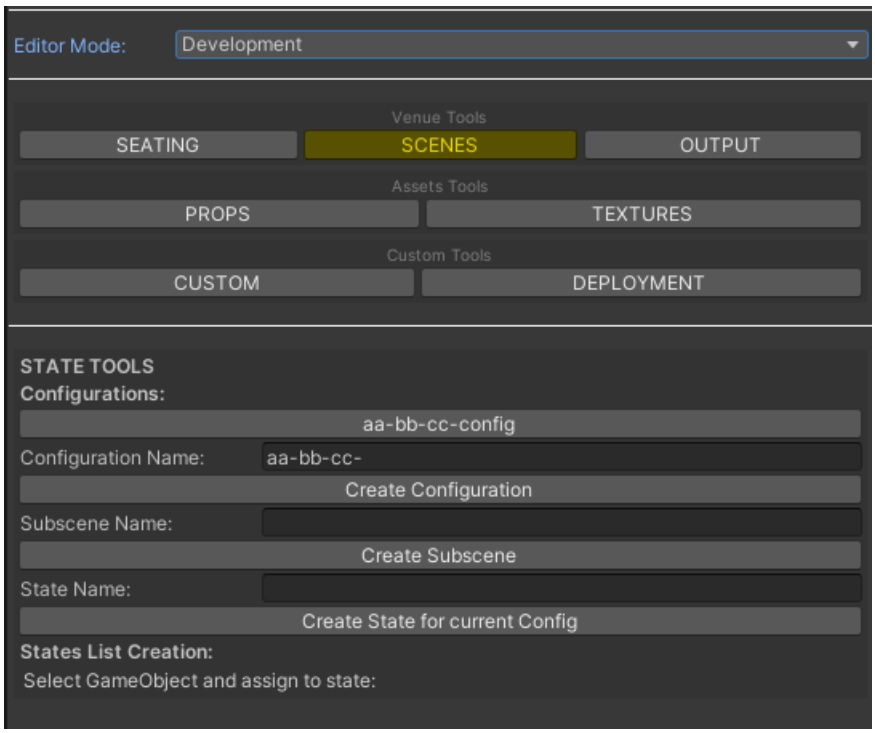


Figura 9: Vista de la herramienta de Scenes

- Output: Formado por sub-herramientas de generación de productos finales como mapas de bloques, mapas de asientos o vistas panorámicas. Sus apartados son:

    - Screenshot: Herramienta usada para generar capturas de pantalla a partir de la cámara principal de la escena. Permite seleccionar la resolución del screenshot, configurar la resolución de la “Game View”, generar la captura de pantalla en sí o generar un tilemap a partir de una imagen.

    - Blockmap: Herramienta usada para generar mapas de bloques. Permite configurar aspectos específicos del mapa que serán escritos en el JSON resultante, generar polígonos a partir de sectores utilizando sus sillas como puntos 3D y exportar un SVG resultante y subir el contenido generado al servidor o guardarlo en local.

    - Seatmaps: Herramienta usada para generar mapas de asientos. Permite generar una configuración a partir de sectores o áreas (grupos de sectores) y entrar en “Modo Edición”. El modo edición permite navegar entre los diferentes mapas de asientos generados y modificar algunas de sus características a la vez que se previsualiza parte del resultado en la vista de juego. Es probablemente la herramienta más compleja del Editor junto con el generador de vistas panorámicas.

    - Panos: Herramienta de generación y distribución de vistas panorámicas. Se trata de la herramienta en cuestión sobre la que se basa el proyecto. Vista en más detalle en otros apartados.

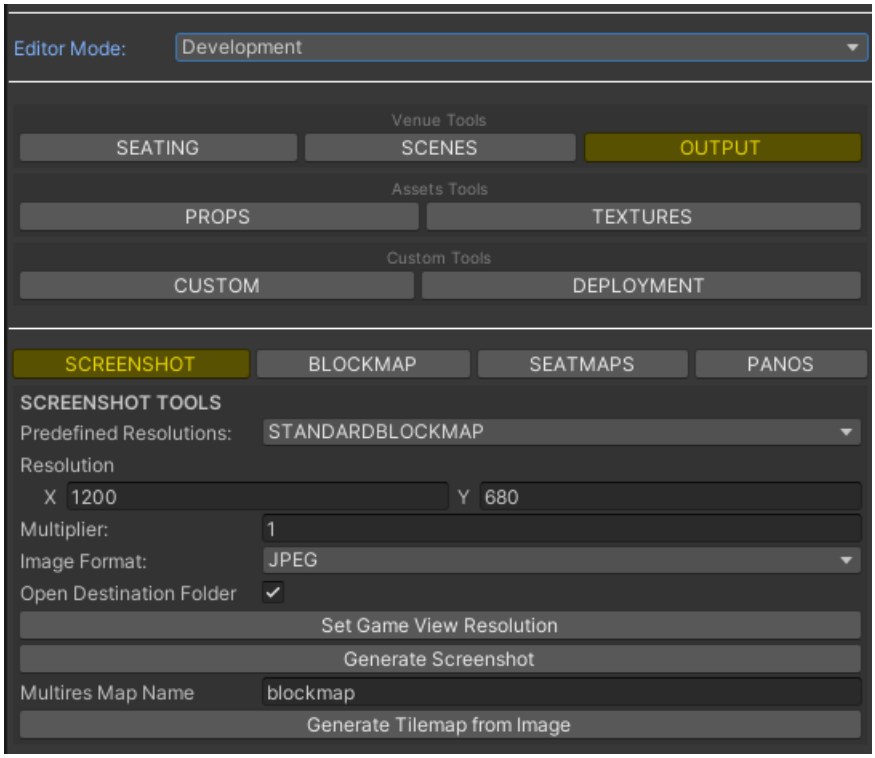


Figura 10: Vista de la herramienta de Output

- Props: Herramienta de trabajo sobre Objetos en la escena. Permite renombrar objetos al mismo tiempo que se modifica la información en la estructura de datos, instanciar objetos a partir de un XML, reemplazar objetos por prefabs, generar LODs y crear prefabs de forma específica a partir de objetos en jerarquía o de otros prefabs generados previamente.

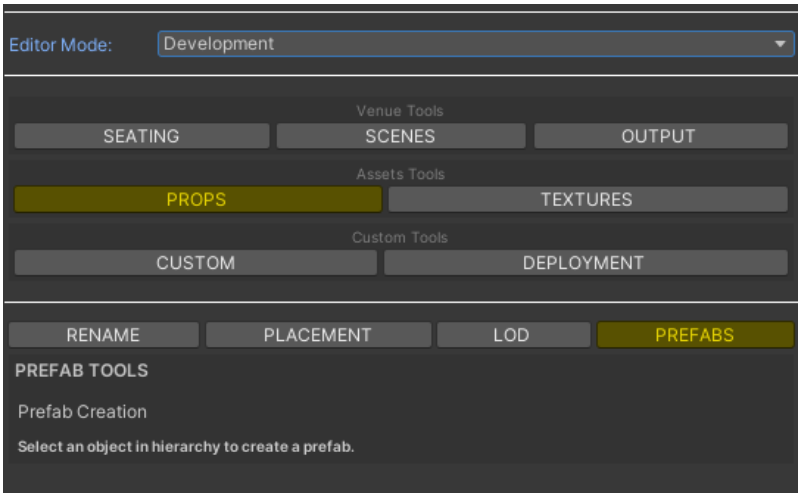


Figura 11: Vista de la herramienta de Props

- Textures: Herramienta usada para generar texturas a partir de otras. Dispone de campos para introducir imágenes y que éstas sean usadas en los distintos canales RGBA de la textura resultante. Además, permite reescalar texturas, ordenarlas en carpetas y otras opciones más específicas relacionadas con la estructura de carpetas del proyecto.

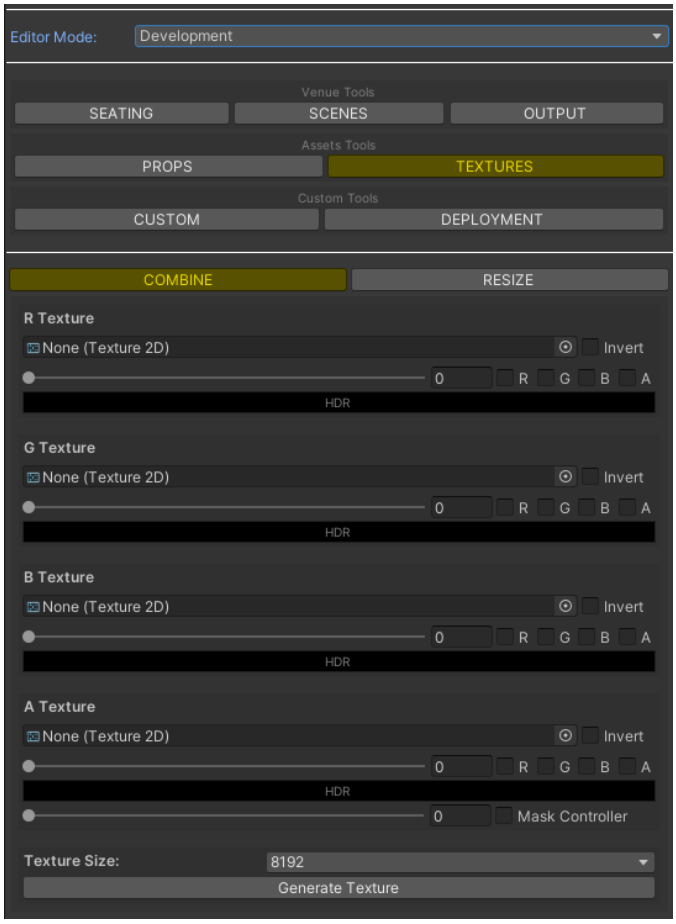


Figura 12: Vista de la herramienta de Textures

A parte de las herramientas principales de Venue Editor, existen dos campos más sólo disponibles en modo Development:

- Custom: Permite realizar scripts más concretos sobre peticiones especiales de los clientes. Este apartado no forma parte de las herramientas principales dada su especificidad.

- Deployment: Permite paquetizar Venue Editor y generar versiones. Esto facilita a los usuarios la actualización de la herramienta, siempre que se produzca un update significativo, de una forma trasparente mediante una ventana popup en el Editor.

Dentro de las herramientas derivadas del Core del aplicativo, podemos distinguir entre: scripts de UI, managers y entidades asociadas.

- UI: Scripts que construyen los elementos que forman parte de la parte más gráfica del aplicativo como menús desplegables, toggles, campos de texto, botones, etc.

- Managers: Scripts encargados de gestionar y ejecutar acciones en base al proceso realizado en la UI. Contienen métodos estáticos que realizan procesos más globales de la herramienta en sí.

- Entidades: Contienen información de la entidad asociada a la interfaz gráfica. Disponen de métodos para realizar acciones sobre la entidad en sí, no sobre acciones genéricas.

Un ejemplo claro de este es la herramienta de generación de vistas panorámicas. Se dispone de una interfaz de usuario que permite al usuario seleccionar el tipo de vista panorámica que se va a generar y los requerimientos de ésta. El manager ejecuta el método correspondiente basado en la información recopilada de la interfaz de usuario. La entidad almacena información sobre cada una de las vistas panorámicas generadas por el manager y es específica de cada una de las vistas.

En el caso de querer generar vistas panorámicas para cada una de las sillas de un sector y subirlas al servidor, se selecciona el sector en cuestión, se marca la casilla de “sillas” y se selecciona la región del servidor al que se quiere subir. Al apretar el botón de generar, se llama al método del manager encargado de crear una cola de panorámicas y subirlas por grupos al servidor. Para cada una de las panorámicas, la clase de la entidad (en este caso “Pano.cs”) ejecutará las acciones específicas asociadas a la panorámica que se está generando (por ejemplo, generar el JSON con la información de posición, rotación y tipo de panorámica).

4.1.3. Generador de vistas panorámicas

La herramienta principal en la que se basa el proyecto está incluida dentro del apartado de herramientas anterior. En concreto esta herramienta ha sufrido un cambio sustancial con respecto a su versión de origen en Venue Edior 3. En la versión anterior toda ella estaba desarrollada en Play, mientras que en la versión 4 se ha conseguido migrarla de forma satisfactoria al Editor. Se ha conseguido implementar una interfaz lo suficientemente parecida a la interfaz en Play como para poder usar la herramienta de la misma forma. Además, se han añadido barras de progreso cancelables en el Editor que permiten al usuario tener control sobre los tiempos estimados de generación y subida y poder parar los procesos en caso de que sea necesario. Al tratarse de un anexo a Venue Editor, existe un script de interfaz asociado, un manager y una entidad específicos para esta herramienta:

- UI: El script de interfaz de usuario se compone de un método estático principal **UI**, llamado desde el método **UI** del script de vista de Output. A su vez, este script es llamado desde el script principal de interfaces (EditorUI.cs) en el método **Body()** contenido en **OnGUI()**. La interfaz de usuario se genera mediante el uso de librería de Unity EditorGUILayout (Unity Technologies, 2023) y GUILayout, (Unity Technologies, 2023) usando los distintos elementos que proporciona la API.

- Manager(s): Para la correcta gestión de los elementos de la interfaz, existe un script que actúa como manager y enlace entre la clase “Pano.cs”, que contiene información específica de la vista panorámica, y la clase correspondiente a la interfaz. Dicho script da vida a las funcionalidades de los botones, campos de texto, desplegables y demás elementos existentes en la herramienta. Además, se cuenta con un network manager, diseñado para esta herramienta pero reutilizable en otros casos, que permite subir los contenidos generados a un endpoint determinado y que sean usados para su posterior integración en web.

- Entidad: Esta es la clase encargada de gestionar la vista panorámica en sí. Almacena información sobre posicionamiento de la cámara, el tipo de vista panorámica, los bytes asociados a las imágenes y la información a escribir en el JSON. Dispone también de métodos específicos de generación de cubos o esféricas, colocación de cámaras y renderizado, así como de enlazado de vistas y escritura de datos.

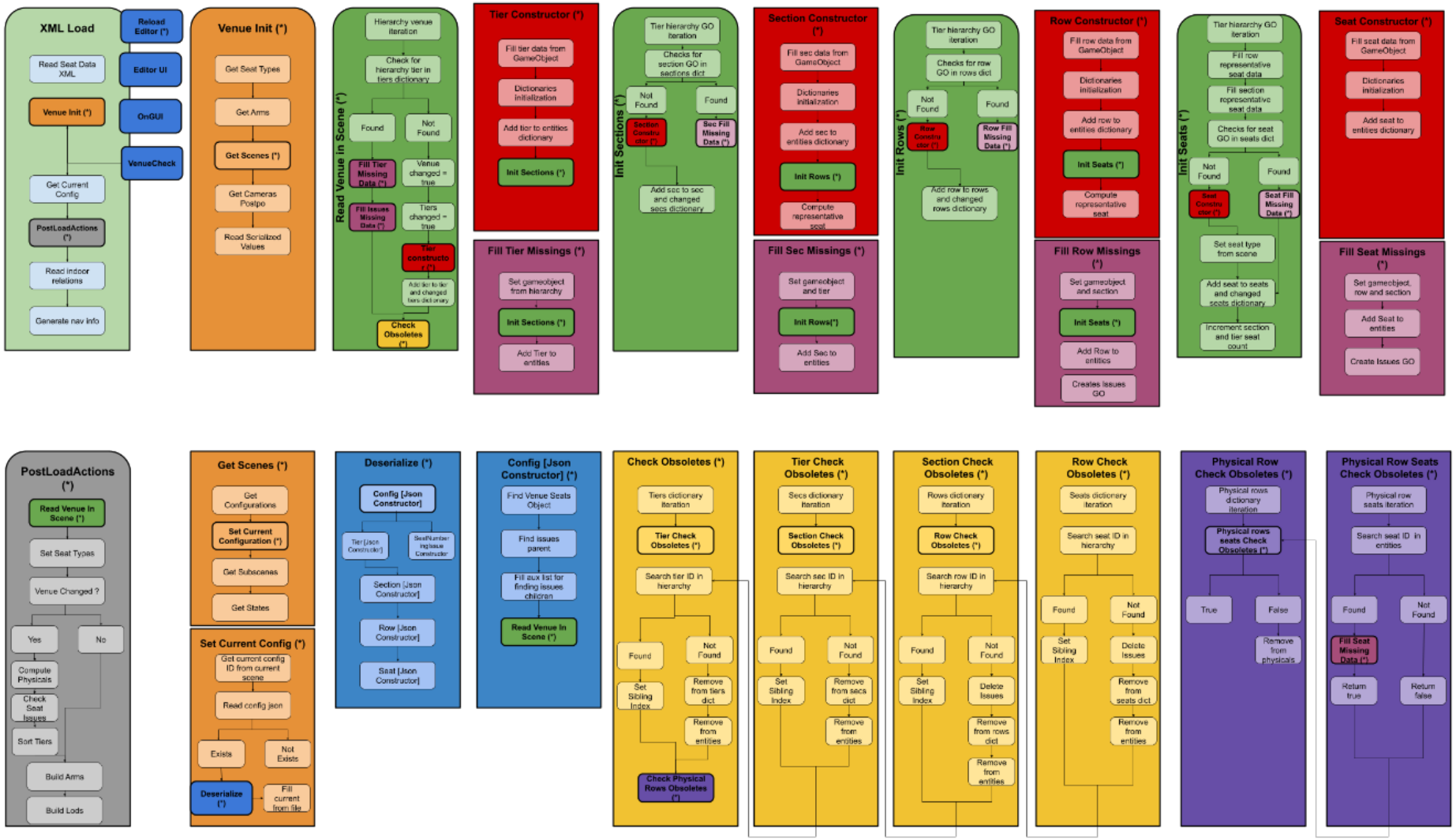
Para más información respecto al generador de vistas panorámicas, consultar el apartado 5 de la memoria.

4.2. Arquitectura de la información y diagramas de navegación

Para entender bien el conjunto global de la herramienta, es necesario hacer un pequeño repaso al diagrama de navegación de la parte más core de Venue Editor 4. Esto ayuda a entender con más facilidad cómo se inicializan y se usan las entidades y cómo crear una estructura de datos básica para trabajar en la edición de vistas panorámicas.

4.2.1. Diagrama de navegación de VE4

A continuación, se muestra un diagrama general del funcionamiento de Venue Editor 4 a partir de la carga de un XML y los posteriores refrescos que puedan suceder una vez inicializados los datos. Es importante entender que, para conservar la información generada tras la primera carga, es necesaria la serialización y lectura de datos con tal que el estado actual de la herramienta se conserve y se pueda trabajar con normalidad cerrando y abriendo Unity entre sesiones.



Mesa 2: Diagrama de navegación de VE

Existen diferentes secciones en el diagrama:

- Carga de XML: Es el primer paso en la inicialización de datos. Mediante un botón de interfaz se abre una ventana emergente que permite seleccionar un archivo XML que contiene los datos del recinto a virtualizar. Tras su lectura y análisis, se ejecuta el método Venue.Init() y se crea la jerarquía de objetos en Unity.

- Venue Init: Hay varios métodos interesantes a los que se llama desde esta función:

    - Get Seat Types: Inicializa un diccionario con los nombres de los prefabs que representan las sillas que se van a usar para virtualizar el recinto. Cada una dispone de un índice que será su identificador a la hora de instanciar.

    - Get Arms: Comprueba si existen prefabs para los brazos de las sillas en el proyecto y los asocia según Seat Type.

    - Get Scenes(*)

    - Get Cameras Postpo: Inicializa un diccionario con los diferentes tipos de cámara y sus componentes de postproducción asociados.

    - Read Serialized Values: Lee un fichero de configuración que contiene información sobre la numeración predominante del recinto. Estas varían en función de los IDs de las entidades, pudiendo ir de 1 a N, de N a 1, en orden ascendente o descendente y en saltos de número pares o impares. Esto es útil para detectar potenciales errores en la numeración de las entidades y su posterior corrección.

- Get Scenes: De los métodos comentados anteriormente, el más importante es Get Scenes. Este método se encarga de leer las escenas y subescenas que hay configuradas en el proyecto para crear un diccionario de configuraciones. Una configuración no es más que una variante del mismo recinto. Así pues, se pueden crear tantas configuraciones como se desee según las particularidades del recinto o el producto en el que se vaya a trabajar. A pesar de ello, sólo puede haber una configuración activa y esta se corresponde con la escena que se encuentre cargada en el momento en jerarquía.

  Además de tratas las posibles configuraciones del proyecto, se hace una lectura de subescenas y “estados”. Un estado es un conjunto de subescenas asociado a una configuración. Es decir, según el producto en el que se esté trabajando, al operador le puede interesar ocultar una parte del estadio (por ejemplo, el techo). Si esto ocurre, existe la posibilidad de configurar qué subescenas se cargan a la hora de generar los contenidos. De esta forma, podemos tener un “estado” en el que sólo se visualizan las gradas, el aforo y la pista, otro en el que también se incluya la cubierta y otro que sólo contenga espacios interiores. Se pueden cargar y descargar estados mediante la interfaz gráfica correspondiente, además de modificarlos y usarlos a conveniencia.

- Set Current Configuration: Tras determinar qué escena será la correspondiente a la configuración principal, se asigna la variable “current configuration” de la clase Venue a la configuración que coincide con el ID de la escena. A partir de este punto, el diagrama puede ramificarse en caso de que exista un JSON serializado con datos de la configuración actual previa inicializada. Esto sólo ocurre si se ha cargado un XML previamente y se verá más en detalle en los siguientes puntos.

- Get Current Configuration: Con la configuración principal activa inicializada, ya se puede llamar al método Venue.GetCurrentConfiguration(). Este método es la clave de todo el aplicativo. Dada una configuración actual, podemos extraer un gran número de datos útiles para el resto de herramientas simplemente accediendo a sus métodos y propiedades.

- Post Load Actions: Uno de los métodos más importantes de todo el proceso. Se encarga de leer la jerarquía en escena y comprueba si los objetos existen en la estructura de datos. En este punto también existe una posible bifurcación. Si la configuración actual ha sido deserializada se rellenan los datos de GameObjects e información que no se puede serializar mediante el uso de la jerarquía. En caso contrario, se crea una nueva estructura de datos y se añaden los nuevos elementos.

  Hecho esto, si se detecta que el recinto ha cambiado con respecto a lo que existía anteriormente (puede ser que hubiera ya un recinto cargado o sea una carga limpia) se realizan ciertas acciones que facilitan la construcción de nuevas herramientas y la posterior edición del recinto. En caso de que el recinto se conservé igual (otra persona abre el proyecto de Unity y no dispone de jerarquía) se salta todo el proceso de ordenación, detección de filas físicas, asignación de ranking y detección de errores; toda esta información se deserializa previamente y se ahorra mucha carga computacional.

Llegados a este punto, con el recinto ya creado y la estructura de datos inicializada, se puede realizar una segunda carga de XML (con o sin cambios con respecto al anterior) y el proceso sería ligeramente distinto; pasando por toda la sección de lectura de ficheros e información previamente almacenada. Si esto ocurre, tras la llamada a Set Current Configuration se lee el JSON que contiene toda la información relevante de trabajos realizados en el recinto con anterioridad. Así pues, mediante el uso del package de Newtonsoft, Json.NET, se accede a los “JsonConstructor” de cada una de las clases serializadas y se inicializan los datos (además de pasar por todo el proceso de rellenar información no serializable).

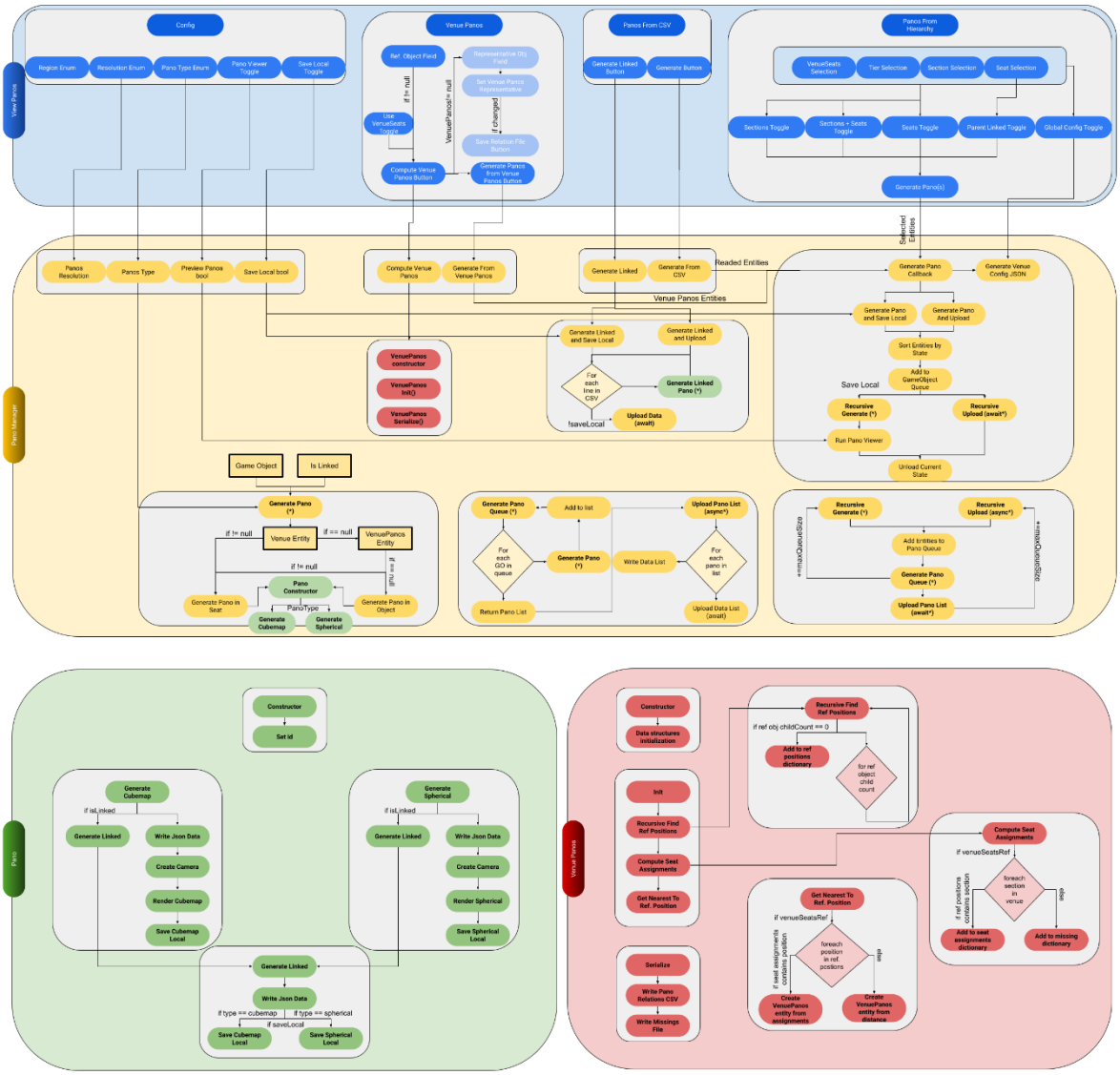
Este proceso de lectura de información permite detectar cambios entre la nueva carga y la lectura de la anterior. En caso de encontrar variantes entre un recinto u otro (si la configuración coincide), lo cambios encontrados generarán nuevas instancias de las clases correspondientes y se añadirán a la estructura como nuevas. Estos cambios pueden ser tanto de IDs, como de posiciones en el espacio en el caso de las sillas. Sin embargo, si se detecta que el XML corresponde a un recinto con un ID

distinto al anterior se borra la estructura de datos y se realiza de nuevo una carga limpia de todos los datos.

Todo este proceso está englobado en las secciones en amarillo que parten del método CheckObsoletes, incluido dentro de ReadVenueInScene (PostLoadActions).

4.2.2. Diagrama de navegación del generador de panorámicas

Entrando un poco más en detalle sobre la herramienta en cuestión, a continuación, se muestra un diagrama de navegación propio del generador de vistas panorámicas. En dicho diagrama se pueden observar cuatro bloques bien distinguidos que corresponden a cada una de las partes involucradas en la generación de contenidos: View Panos, Pano Manager, Pano y Venue Panos. Cada uno de ellos tiene una función específica a la hora de renderizar y distribuir las vistas generadas por el aplicativo; dicho diagrama nos muestra las partes que conforman cada uno de los bloques y cómo se relacionan entre sí.



Mesa 3: Diagrama de navegación del gestor de vistas panorámicas

Entrando un poco más en detalle en el diagrama, para cada uno de los bloques existen varios elementos que se encargan de cosas distintas y están conectados entre sí:

- View Panos: Bloque de interfaz de usuario que contiene todos los elementos de la interfaz.
    - Config: Menús desplegables y toggles que se asocian al manager para realizar determinadas acciones según la selección del usuario.
    - Venue Panos: Elementos de interfaz relacionados con la herramienta Venue Panos. Dispone de campos de objeto y botones para configurar la herramienta y generar panorámicas a partir de un GameObject de referencia.
    - Panos from CSV: Botones que despliegan ventanas emergentes para generar vistas panorámicas a partir de archivos CSV.
    - Panos from Hierarchy: Toggles y botón de generación de panorámicas a partir de selección de GameObjects en jerarquía.
- Pano Manager: Encargado de gestionar mediante métodos estáticos las acciones realizadas sobre en la interfaz.
    - Propiedades: Propiedades de la clase asociadas a los desplegables y toggles de la interfaz.
    - Venue Panos: Acciones asociadas a los botones existentes en la parte de Venue Panos de la interfaz. Principalmente se puede generar una estructura de datos adicional para generar vistas panorámicas a partir de ésta; ya sea mediante un archivo de configuración o mediante cálculos de distancia entre objetos.
    - Panos From CSV: Existen dos métodos correspondientes a las acciones realizables en interfaz:
        - Generate Linked: Genera panorámicas enlazadas entre sí a partir de un archivo CSV que contiene el ID de la panorámica y el ID de enlace.
        - Generate from CSV: Herramienta alternativa a la generación de vistas panorámicas mediante selección en jerarquía. Recibe un archivo CSV con los IDs de las entidades que se deben generar.

    - Panos From Hierarchy: Muestra el proceso global de generación de colas y subida al servidor recibiendo como input una lista de objetos sobre la que iterar. En todo este proceso se llevan a cabo muchas acciones que quedan explicadas en apartados posteriores.

- Pano: Clase asociada a la entidad más básica de todo el proceso: la vista panorámica en sí. Contiene métodos relacionados directamente con la generación en Unity de las vistas panorámicas según tipo. Se divide en tres grandes bloques:

    - Generate Cubemap: Posiciona la cámara, renderiza las 6 caras del cubo, escribe el JSON de la vista y guarda los datos en local si se especifica.

    - Generate Spherical: Posiciona la cámara, renderiza una vista esférica, escribe el JSON de la vista y guarda los datos en local si se especifica.

    - Generate Linked: Si cualquiera de los métodos anteriores recibe por parámetro que la panorámica a generar es de tipo “enlazada”, el método genera el JSON correspondiente según el tipo de renderizado (cúbica o esférica).

- VenuePanos: Clase asociada a la generación de vistas panorámicas mediante el uso de un objeto de referencia. A veces, los operadores quieren hacer generaciones rápidas de los sectores del estadio. En vez de usar la herramienta de selección por jerarquía, existe la posibilidad de usar una mesh simplificada del estadio y calcular internamente los sectores asociados más cercanos a cada uno de los polígonos que conforman el objeto de referencia. Las panorámicas asociadas se lanzan con el ID de las entidades específicas que se generan a partir de la mesh simplificada, pero el posicionamiento de la cámara se realiza mediante la entidad real dentro del recinto.

  Hecho esto, se genera un archivo CSV que contiene la información calculada sobre relaciones entre polígonos y entidades reales. Además, se incluye otro archivo con información relevante sobre objetos que no han sido identificados o asociados durante el proceso.

4.3. Diseño gráfico e interfaces

A día de hoy, y a expensas de que Unity lance al mercado su versión 2023, la configuración y personalización de interfaces gráficas para Editor es muy limitada. Así pues, no existe más que un compendio de menús desplegables, botones, campos de objeto, barras de progreso y alguna otra herramienta más para extender las prestaciones que ofrece el motor gráfico de base.

Así pues, con todo lo que puede ofrecer Unity EditorGUILayout se pueden diseñar interfaces gráficas lo suficientemente completas para que la herramienta funcione, pero con escasa personalización de estilos. A pesar de ello, aplicando cierta lógica de scripting y aprovechando la API extensa de Unity se ha generado una interfaz que permite llevar a cabo todas las acciones.

Para ello, se han recogido los datos necesarios de cada una de las tareas a realizar y se ha construido una herramienta de Editor preparada para evitar errores y acciones no deseadas por parte de los usuarios.

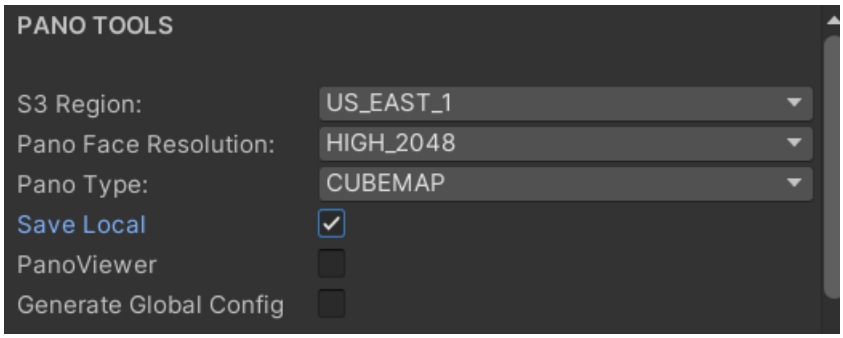


Figura 13: UI de configuración de panorámicas

Empezando por la parte de configuración, en la figura 13 se pueden observar 3 menús desplegables y 3 toggles que repercuten en las acciones del usuario.

- S3 Region: Desplegable que determina a qué servidor se subirán los datos generados. A escoger entre: US_WEST_1, AP_NORTHEAST_2, US_EAST_1, EU_WEST_1 y EU_CENTAL_1.

- Resolution: Determina la resolución final de las panorámicas generadas. A escoger entre: 512x512, 1024x1024, 2048x2048 y 4096x4096.

- Pano Type: Determina el tipo de panorámica que se va a generar. A escoger entre: cubemap y spherical.

- Save Local: Opción para guardar los contenidos en local en vez de subirlos al servidor.

- PanoViewer: Abre un aplicativo local para previsualizar los contenidos generados en local. Esta funcionalidad sólo está disponible en local y, por lo tanto, sólo se muestra si está activo el toggle Save Local.

- Generate Global Config: Este toggle genera un archivo JSON de configuración global para todas las panorámicas de un recinto que hace referencia al ID del venue y al ID del mapa generados.

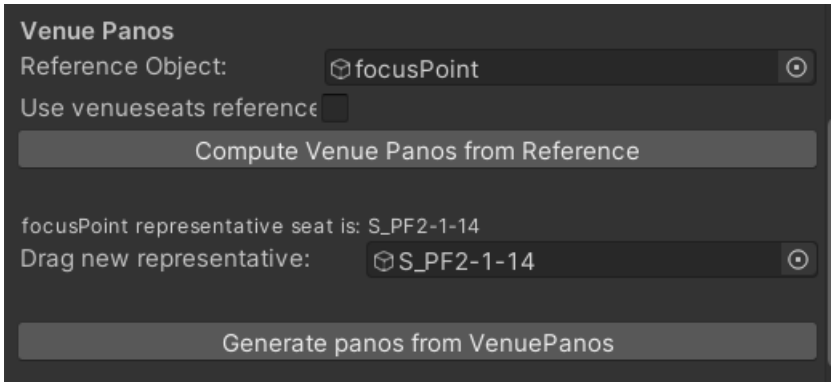


Figura 14: Venue Panos UI

A continuación, justo debajo de la configuración de panorámicas, encontramos la herramienta de Venue Panos. Esta se conforma de varios elementos que se ocultan y se muestran en función del set de datos que haya inicializado para evitar que el usuario cometa errores.

En primer lugar, tenemos un campo de objeto al que se puede arrastrar cualquier GameObject de la jerarquía y un toggle para usar como referencia los objetos que conforman el recinto. En sengundo lugar, con el objeto de referencia arrastrado al campo de objeto, se muestra justo debajo un botón para calcular las entidades que pertenecen a Venue Panos y un campo de objeto nuevo justo debajo marcado con un pequeño label. Este label indica cuál es la entidad del recinto que ha sido seleccionada como más cercana al objeto de referencia. Aún así, el campo de objeto que aparece justo debajo permite seleccionar cualquier otra entidad y reemplazar la que se ha calculado de forma automática.

Finalmente, se dispone de un botón que genera las vistas panorámicas de todas las entidades que han sido enlazadas a sectores reales del recinto una vez generada toda la información.

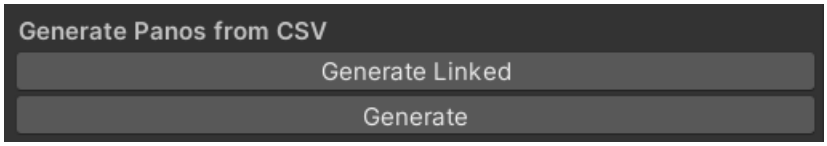


Figura 15: Panos from CSV UI

La interfaz para generar panorámicas a partir de archivos CSV es probablemente la más simple de todas. Dispone de dos botones para generar panorámicas enlazadas a partir de una lista y para generar panorámicas normales a partir de otra lista. Ambos botones abren una ventana desplegable que permite seleccionar el archivo CSV deseado y parsean la información correspondiente.

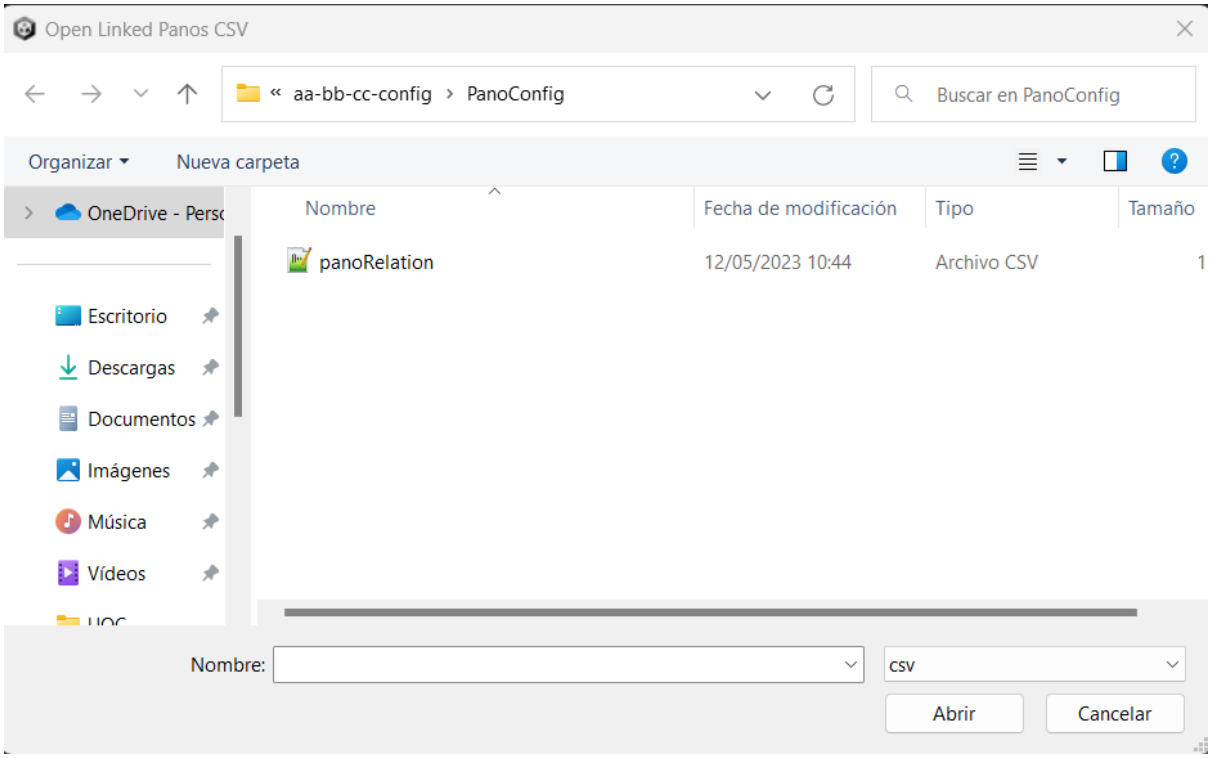


Figura 16: Ventana emergente de selección de archivo CSV

Por último, se ha diseñado la interfaz más compleja de toda la herramienta: panorámicas a partir de jerarquía. Por defecto, esta interfaz muestra el siguiente mensaje si el objeto seleccionado no es del tipo esperado por parte de la herramienta:

Figura 17: Selection not valid UI

En el momento en que se selecciona una entidad considerada como válida por parte de la aplicación, la interfaz cambia de aspecto y muestra distintas posibilidades. Cada tipo de entidad muestra información diferente en función de las acciones que se pueden realizar.

- Venue Seats: Existe la posibilidad de seleccionar todo el recinto y generar vistas panorámicas a nivel global.

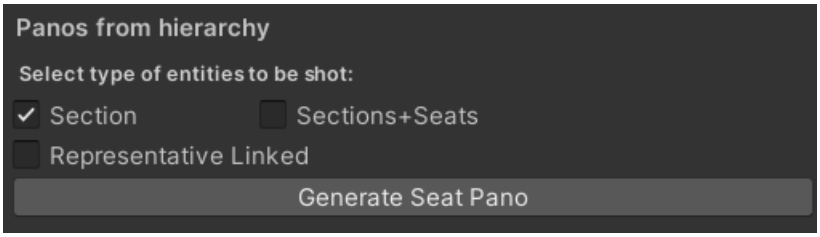


Figura 18: Panos from VenueSeats UI

Podemos escoger entre dos toggles que son excluyentes entre sí (no podemos seleccionar los dos a la vez). Si seleccionamos “Section” se generarán panorámicas de todas las secciones del recinto, mientras que si seleccionamos “Section+Seats” se generarán tanto de todas las secciones como de todas las sillas. Al seleccionar el toggle de sección (tanto en esta como en las selecciones posteriores) se muestra un toggle adicional para generar una panorámica enlazada a la silla representativa. Esto permite evitar la generación de imágenes adicionales y acortar el proceso mediante el uso de un JSON que enlaza la vista con la de la silla representativa ya generada.

- Tier: Seleccionando un tier en jerarquía, los toggles de tipos de entidades a generar cambian.

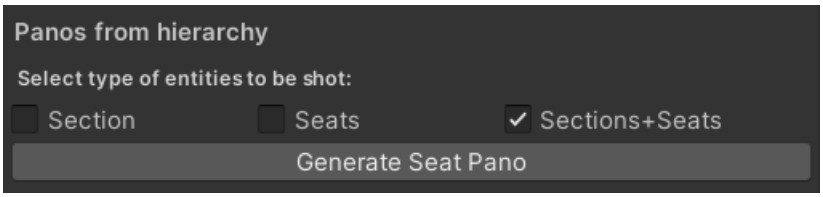


Figura 19: Panos from Tier UI

Esta vez se añade la posibilidad de generar vistas panorámicas sólo de las sillas que forman parte de cada uno de los sectores de la grada.

- Section/Row: La interfaz gráfica mediante selección de sector y fila es prácticamente igual que la de selección de grada. La única diferencia es el pequeño label que se muestra justo encima de los toggles que nos indica cuál es la silla representativa de dicha entidad; es decir, sobre qué silla se va a generar la vista.

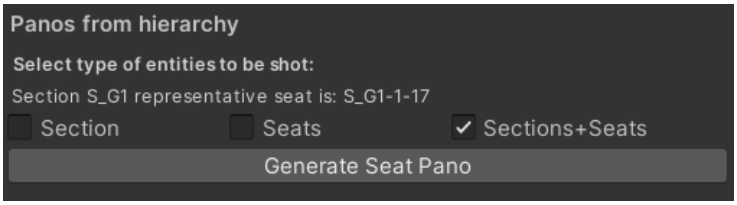


Figura 20: Panos from Section UI

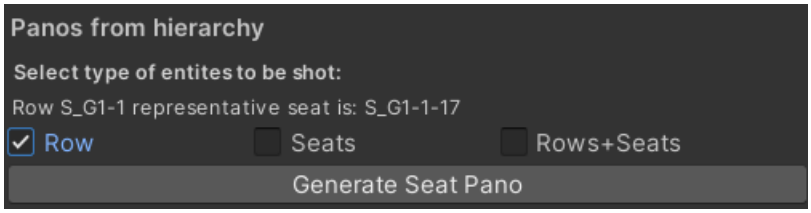


Figura 21: Panos from Row UI

- Seat: Al seleccionar una silla se activan varios elementos en la interfaz. Al ser la unidad más básica del recinto, los toggles desaparecen y sólo se muestra la opción de generar una

panorámica enlazada al parent. De esta forma, al integrar la panorámica en web ésta irá a buscar la relación de ID con su parent y mostrará dicho contenido usando el archivo JSON que indica el enlace, en vez de generar y subir todas las imágenes como si fuera una panorámica nueva.

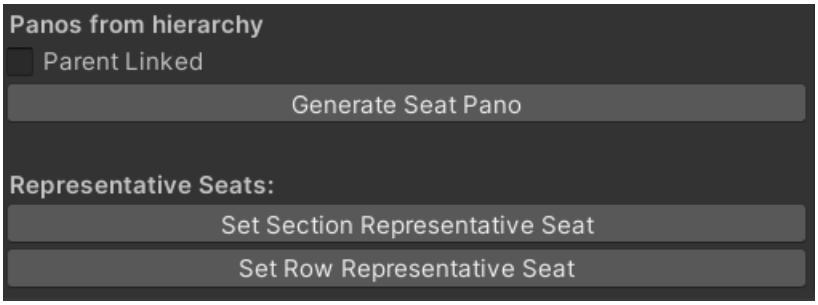


Figura 22: Panos from Seat UI

Además, se añade la posibilidad de cambiar tanto la silla representativa de la fila como del sector. Los dos botones que aparecen en la parte inferior actualizarán la información de los parents de dichas sillas.

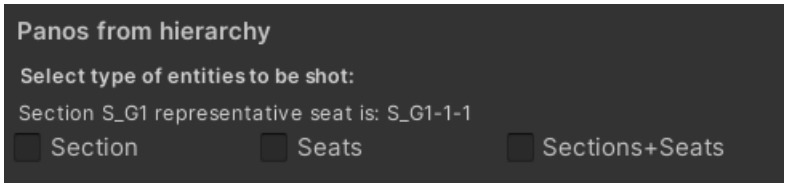


Figura 23: Cambio de silla representativa en S_G1

Todas las interfaces de generación mediante selección en jerarquía tienen en común el botón de Generate Seat Pano que, sencillamente, empezará a ejecutar el proceso y mostrará una barra de progreso cancelable que muestra el tiempo estimado que va a llevar la generación. Esto es muy útil de cara al operador para hacerse a la idea del tiempo que durará el proceso y poder cancelarlo a mitad de camino en caso de que se haya cometido un error.

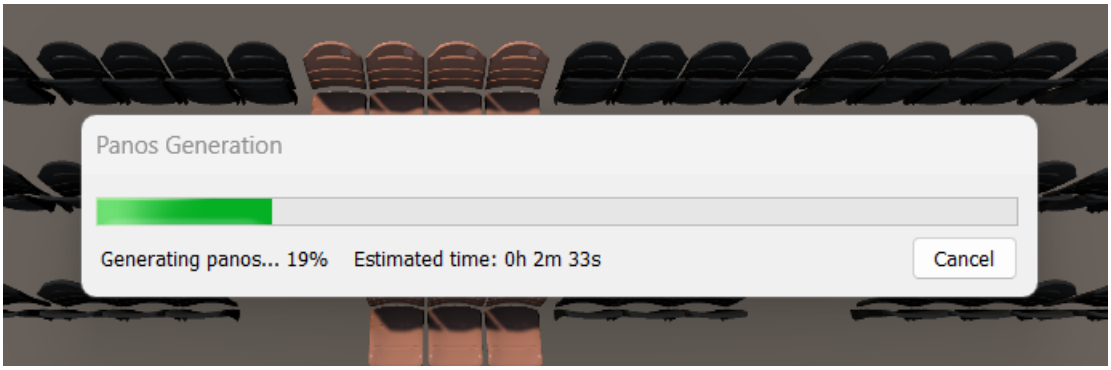


Figura 24: Generate Seat Pano progress bar

4.4. Lenguajes de programación y APIs utilizados

Tal y como se ha comentado en apartados anteriores, toda la herramienta se ha desarrollado en Unity, usando como base la API de scripting de UnityEngine y UnityEditor (Unity Technologies, 2023). Estas APIs nos permiten generar herramientas de Editor dentro del motor gráfico aprovechando en su totalidad la gestión y el manejo de GameObjects así como elementos de interfaz. El lenguaje principal de Unity como motor gráfico es C#, así que todo el código ha sido realizado usando dicho lenguaje de programación.

Además de las APIs específicas de Unity, se han aprovechado otros recursos para la gestión de diferentes tareas que forman parte de la API de Microsoft de .NET 7.0 (Microsoft Corporation, s.f.). Entre ellos están:

- System.IO (Microsoft Corporation, s.f.): Usado para la lectura y escritura de ficheros dentro de la aplicación, así como soporte básico para el manejo de archivos y directorios.

- Systrem.Linq (Microsoft Corporation, s.f.): Para manejar estructuras de datos y queries que usan Language-Integrated Query. Útil para trabajar con listas, diccionarios, hashsets, etc.

- System.Threading (Microsoft Corporation, s.f.): Permite realizar acciones multithread. Se usa para la gestión de métodos asíncronos que se encargan de la subida de contenidos al servidor y para bloquear determinadas acciones mientras el hilo asíncrono está activo.

Por último, para la serialización de datos se ha utilizado el paquete Json de Newtonsoft (Newtonsoft, 2023), compatible con C#, LINQ y XML. Este paquete facilita la escritura y lectura de datos que conforman las diferentes clases involucradas en los diferentes procesos.

5. Implementación

El apartado de implementación entra más en detalle sobre el trabajo de campo realizado a la hora de implementar el generador de vistas panorámicas. Actualmente, la empresa sigue utilizando Venue Editor 3 como software principal, lo que quiere decir que todas las herramientas se actualizan en paralelo hasta que se haya llevado a cabo la migración correspondiente a la nueva versión. Aún así, la forma de trabajar en la versión 4 es muy diferente a la versión 3. En la nueva versión se hace uso de la estructura de datos y las entidades pregeneradas al cargar el recinto para tener un acceso más directo a la información relacionada con las panorámicas. Además, el hecho de disponer de clases específicas para las entidades del recinto hace mucho más fácil el acceso y el almacenamiento de información con respecto a los datos relevantes para las vistas que no se calculan específicamente durante el proceso de generación. A pesar de esto, todo el proceso está sujeto al resultado que se debe obtener para que los datos finales coincidan y se entreguen en el formato esperado del visor web desarrollado por el departamento de ingeniería especializado en IT. Así pues, aunque la forma de hacerlo sea muy diferente, el output debe ser siempre el mismo.

5.1. Vistas panorámicas

Tras varios capítulos de explicación y propuesta del proyecto, es el momento de explicar más en detalle las partes correspondientes a la implementación de la herramienta. Para entender todo el proceso es importante detallar qué es en si una vista panorámica. Existen dos tipos de vistas panorámicas utilizadas dentro del entorno empresarial de Mobile Media Content: cubemap y spherical.

- Cubemap: Vista panorámica formada por las 6 caras de un cubo. Cada una de las caras son sencillamente renders de cámara tras su colocación en el punto de la escena adecuada. Una vez ubicada la cámara, se aplica la rotación correspondiente en los ejes x,y para enmarcar la cara del cubo deseada y se activa y aplica la textura adecuada. Con la textura y la cámara listas para renderizar, se almacena la información en un array de bytes que se usa posteriormente para subir el contenido por carpetas al servidor o guardarlo en local.

- Spherical: A diferencia del cubo, Unity dispone de herramientas para la generación de vistas panorámicas como tal. Para ello, se hace uso de los métodos RenderToCubemap (Unity Technologies, s.f) y ConvertToEquirect (Unity Technologies, s.f) a través de la cámara y la textura respectivamente. De esta forma, se renderiza la textura en cámara a un formato equirectangular mono, en el cual la versión monoscópica ocupa el 100% de la textura. Este tipo de generación es preferida a la generación cúbica dada la rápida generación de imágenes en comparación a la anterior. Aun así, dispone de mucha menos información en cuanto a resolución se refiere e introduce artefactos en los polos de la vista panorámica (allá donde se juntan los extremos de la imagen). Para solucionar este tipo de problemáticas,

existe la posibilidad de generar vistas panorámicas tileadas con información detallada a diferentes niveles que permiten ofrecer una mayor calidad a este tipo de generación.

La clase “Pano.cs” es la encargada de renderizar las texturas de una vista panorámica según la selección de la interfaz. La forma de hacerlo pasa por una llamada al constructor de la clase, para poder generar una nueva vista, y la posterior llamada a los métodos de renderizado en cubo o esférico. Estos métodos se encargan de generar una versión de las imágenes en alta resolución y otra con resolución más reducida. La idea es, una vez integrado en la web, descargar primero las imágenes de baja resolución (cuya velocidad de descarga es muy superior a las de alta) y esperar mientras se recogen los recursos de alta resolución del servidor. Para no sobresaturar la herramienta y ralentizarla más de la cuenta, en vez de renderizar de nuevo las imágenes en baja resolución, se aplica un escalado a las texturas generadas la primera vez.

Para el posicionado de cámara, se recibe por parámetro la posición y rotación del objeto sobre el cual se coloca y se asigna el mismo valor a la cámara con un pequeño offset de altura que crea la sensación de estar sentado sobre la silla desde la cual se ha generado la vista. Adicionalmente, se configura la profundidad de campo en 90, considerando que es un valor suficientemente bueno como para sacar contenidos de buena calidad visual. En el caso de las panorámicas esféricas, la imagen renderizada cubre el 100% de la vista de la escena. Sin embargo, para renderizar cubos se aplican rotaciones de 90º en los ejes “x” e “y” del Transform de la cámara para enfocarla en la dirección correcta.

Para terminar todo aquello correspondiente a la generación de imágenes, se renderiza una miniatura de la escena, correspondiente a la frontal de la cámara, que se utiliza a modo de previsualización en las webs de los clientes. Esta miniatura se genera a una resolución de 512x512, representando la cara frontal de un cubo y rotada de tal forma que mira al centro del recinto (configurado previamente por los trabajadores del departamento de producción).

La clase asociada a la vista panorámica es también la encargada de guardar los contenidos en local y generar los datos correspondientes del archivo JSON. Esta información se guarda por duplicado en dos directorios: un directorio dentro de la carpeta del proyecto de Unity pero ubicado fuera de la carpeta de assets y un directorio dentro de la carpeta del aplicativo usado para previsualizar contenidos simulando el visor web. La estructura de carpetas es la siguiente:

- Viewer3D/Panos: carpeta general que contiene todas las vistas panorámicas generadas. Aquí es donde se almacena el json genérico que engloba información sobre todo el conjunto de vistas.

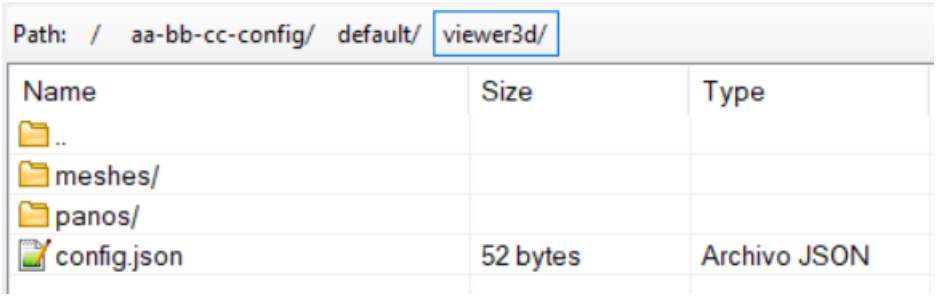


Figura 25: Carpeta con Venue Config JSON

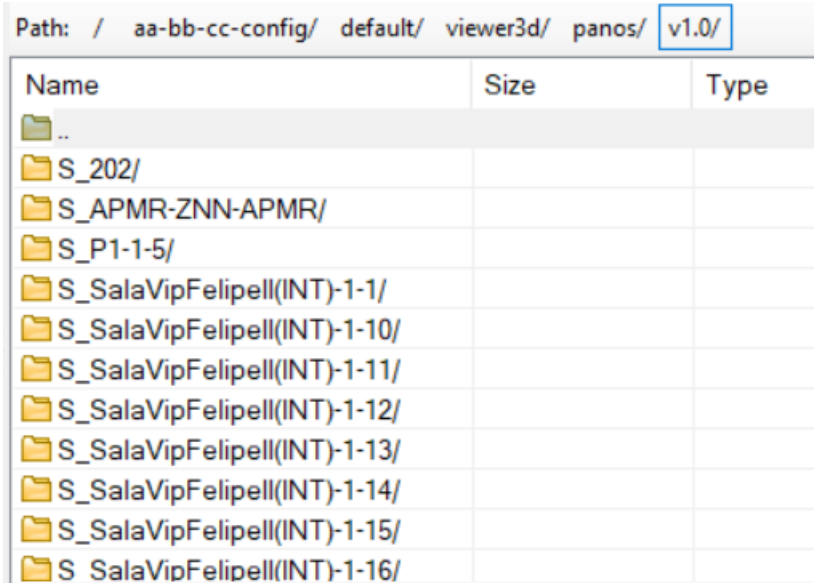


Figura 26: Carpeta de panorámicas organizadas por versión

- Meshes: carpeta que contiene el json que hace referencia a los vértices y triángulos de las meshes de navegación en caso de que sean necesarias para el conjunto de vistas.

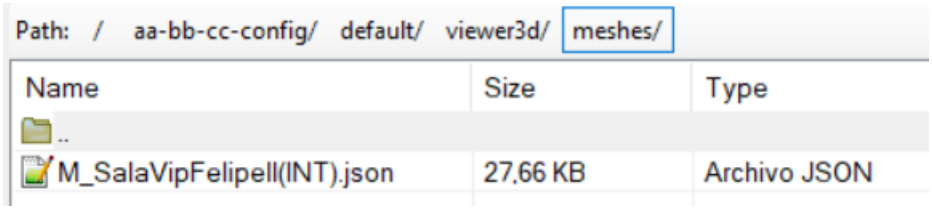


Figura 27: Carpeta con JSON de navmesh

- ID de vista: carpeta que contiene información respectiva a la panorámica con el identificador en cuestión. Aquí se almacena el json que contiene información específica de la vista panorámica como el ID, la posición, la rotación, etc.

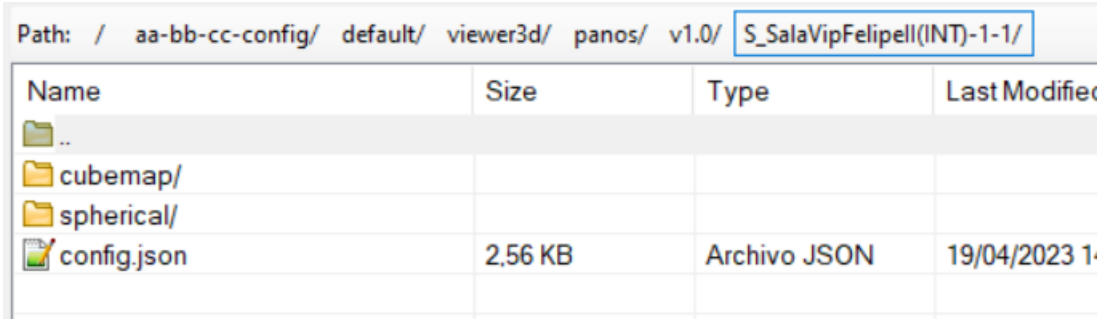


Figura 28: Carpeta con JSON de vista panorámica

- Cubemap/Spherical: Carpeta que contiene todas las imágenes renderizadas desde Unity.

    - Hres: Carpeta que contiene las imágenes en alta resolución.

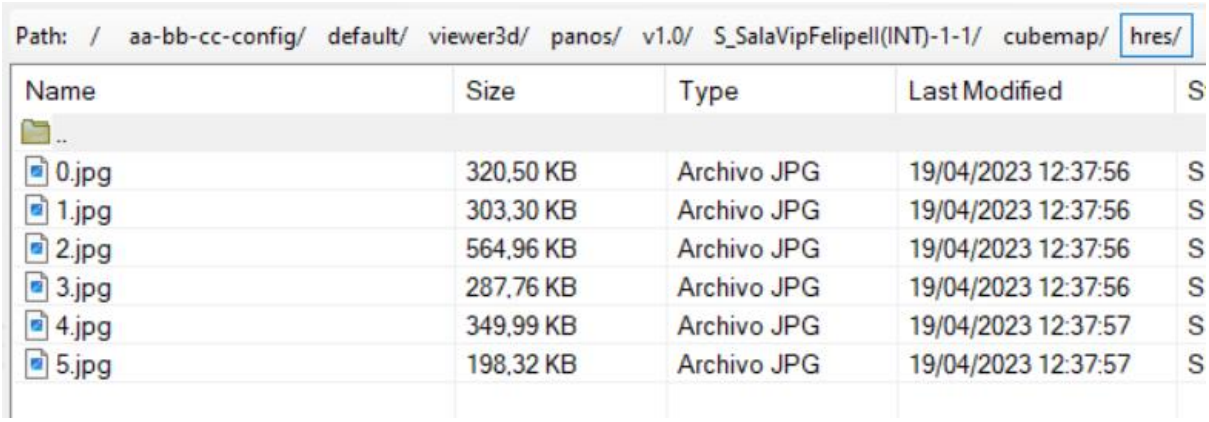


Figura 29: Carpeta con imágenes de vista panorámica cúbica en alta resolución

    - Lres: Carpeta que contiene las imágenes en baja resolución.

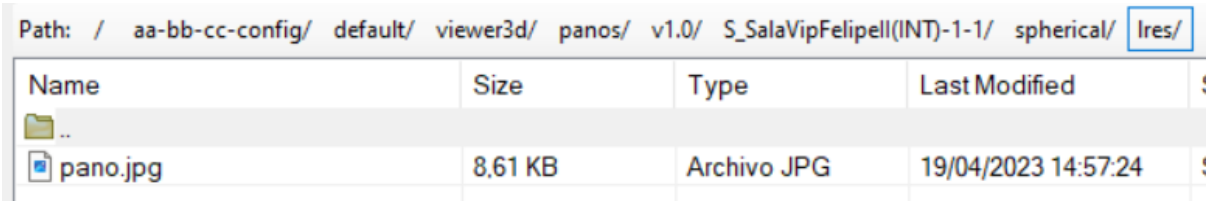


Figura 30: Carpeta con imagen de vista panorámica esférica en baja resolución

- Preview: Carpeta que contiene la miniatura generada desde Unity.

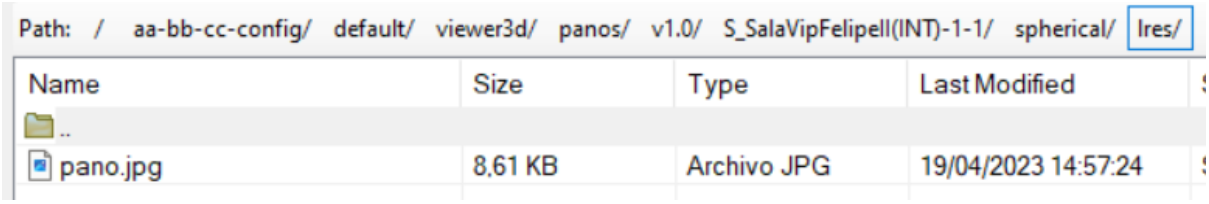


Figura 31: Carpeta con imagen miniatura

Por último, queda hablar de la implementación de la especificación de los JSON. Mobile Media Content, en su API web, dispone de una especificación detallada de cómo deben ser los datos recibidos de las vistas panorámicas para su integración en web posterior a la generación. Para ello, en Unity, se ha replicado una serie de interfaces que definen el conjunto de clases que, a posteriori, se serializan en estructuras de datos relevantes para el correcto funcionamiento del visor web. Así pues, hay varios elementos importantes en este proceso:

- Viewer3DVenueConfig: Esta clase es la encargada de serializar la información con respecto al JSON genérico de un conjunto de vistas panorámicas. Se compone de un string obligatorio determinando el ID de recinto e información adicional con respecto a las vistas. Esta información va desde los plugins que se marcan como activos para su uso en el visor web, hasta la versión de uso y referencias a otros jsons de estilos personalizables que escalan el apartado visual de los mapas en web.

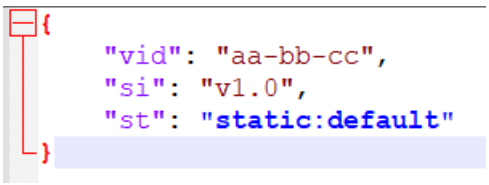


Figura 32: JSON de configuración de Venue

- Viewer3DPanoConfig: Esta clase es la encargada de serializar la información respectiva a la vista panorámica individual generada para cada entidad. Contiene información básica como el ID de la entidad, la posición y rotación de la cámara, el tipo de panorámica, etc. Además, extiende toda esta información haciendo uso de clases que conforman algunas de las propiedades de la clase Viewer3DPanoConfig. Entre ellas:

    - Límites de rotación: Array de floats que determinan bloqueos en el movimiento de cámara en los ejes “x” e “y”.

    - Tiles: Información sobre tileado de imágenes como el nivel de profundidad y los pixeles por nivel.

    - Título y descripción: Información adicional para mostrar en el visor con respecto al nombre “legible” y la descripción de la vista panorámica.

    - Viewer3DPanoPluginsConfig: Clase formada por los diferentes pluggins activables en el visor web. Principalmente, el plugin de navegación y el de relaciones. Cada plugin se conforma de sus propiedades específicas como la información de los nodos de navegación o un diccionario de relaciones entre vistas panorámicas.

- Node3D: Array de nodos 3D usados para navegación entre panorámicas. En el caso de que sea necesario, se genera una lista de nodos con información de ID, tipo de geometría (plano, esfera o mesh), tipo de nodo, interactuación, estados, posición, visibilidad, orden de renderizado y configuraciones de estilo más personalizadas. La clase Node3D es la clase madre de cada uno de los tipos de nodo que se pueden generar desde Unity. Las clases derivadas, extienden la información del parent con datos relevantes del tipo de geometría específico. Por ejemplo, la clase Node3DSphere añade información sobre la rotación y el radio de la esfera; la clase Node3DPlane añade información sobre la rotación y la normal del plano.

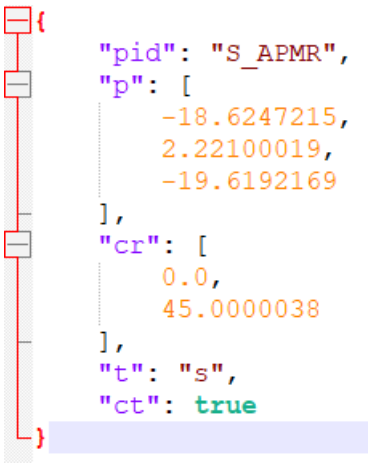


Figura 33: JSON de configuración de vista panorámica básica

La gran mayoría de parámetros configurables de la especificación están marcados como “nullable” en sus respectivas clases y su correspondiente interfaz. De esta manera, mediante la API de serialización de Newtonsoft, podemos ignorar todas aquellas propiedades que no se hayan inicializado y cuyo valor sea “null”.

Visto esto, hay que tener en cuenta dos casos especiales de vistas panorámicas generables: panorámicas enlazadas y panorámicas de navegación.

5.1.1. Panorámicas enlazadas

Tal y como se ha comentado con anterioridad, existe la posibilidad de generar panorámicas enlazadas. Este tipo de vistas, a diferencia de las demás, no necesitan de la generación de imágenes para su funcionamiento. Así pues, hacen uso de la especificación de panorámicas para generar un JSON en concreto que contiene la siguiente información:

- Viewer3DPanoConfig: Del mismo modo que el resto de vistas panorámicas, se crea una nueva instancia de la clase que se va a serializar para generar el archivo.

- ID: Propiedad de la clase Viewer3DPanoConfig. Determina el ID de la vista panorámica que se ha generado.

- Link: Propiedad de la clase Viewer3DPanoConfig. Determina el ID de la vista panorámica enlazada a la vista generada.

Este sencillo JSON simplifica mucho el trabajo en algunos casos. Supongamos que se ha generado una vista de sector “S_1” cuya silla representativa es la “S_1-1-10”. Las imágenes generadas tanto a nivel de sector como a nivel de silla representativa son exactamente las mismas. Así pues, y para reducir la carga de trabajo, podemos generar vistas enlazadas en ambas direcciones para que al generar la vista de sector se usen las imágenes previamente generadas desde su silla representativa y viceversa. Esto quiere decir, que solo nos tenemos que preocupar de generar la panorámica desde la entidad madre o la entidad hija y marcar en el archivo JSON a qué ID debe apuntar el visor web para descargar las imágenes pertinentes.

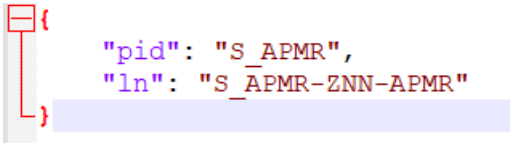


Figura 34: JSON de configuración de vista enlazada

5.1.2. Panorámicas de navegación

En apartados anteriores se han mencionado vistas panorámicas que permiten la navegación entre nodos 3D. Dichos nodos 3D están especificados en los archivos JSON de configuración tal y como se ha visto en el apartado 5.1. Este tipo de vistas están restringidas a espacios de interior, señalados con un “(INT)” justo al final del ID único de la entidad. Al inicializar el recinto, y si se dispone de meshes de navegación en el proyecto, se utiliza Physics.LineCast desde cada una de las sillas que conforman el espacio interior en dirección al resto de sillas del espacio. Si durante el proceso de raycasting existe una colisión con la mesh, el nodo de destino se ignora; si, por el contrario, no hay intersección con la mesh entre las dos posiciones, el nodo de destino se añade a una lista de nodos de navegación.

Una vez obtenida la lista de nodos de navegación para cada una de las sillas del espacio interior, a la hora de generar la vista panorámica se comprueba la existencia de dicha lista para serializar la información y almacenarla en el JSON. De ser así, se crea una nueva instancia del plugin de navegación para que el visor tenga toda la información correspondiente a los posibles destinos entre vistas panorámicas. Cuando esto ocurre, además de subir el JSON de configuración, se publica también un JSON adicional (en una carpeta aparte) con los vértices y triángulos de la mesh utilizada para la navegación. Esto permite al visor web realizar una proyección de un círculo sobre la posición del ratón y que esta proyección coincida con las paredes, el techo y el suelo del espacio. Además, en

el suelo del recinto interior, se utilizan estos mismos círculos para marcar las posiciones de los nodos visibles desde el que se encuentra el usuario.

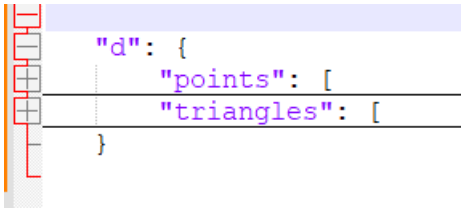


Figura 35: JSON de configuración de mesh

Figura 36: Plugin de navegación

Existe una segunda manera de generar vistas panorámicas de navegación a parte del uso de navmeshes: los archivos de navegación. De vez en cuando, puede ser que, en función del recinto, exista un archivo CSV que provee a la herramienta de la información de navegación entre nodos. De ser así, en vez de generar los nodos de navegación mediante raycasting, simplemente se parsea el archivo y se genera una lista de nodos a los que se puede viajar a través de dicha silla. A la hora de serializar la información, se comprueba la existencia tanto de nodos de navegación como nodos normales y, si existe cualquiera de las dos el resultado se plasma en el plugin de navegación del visor web.

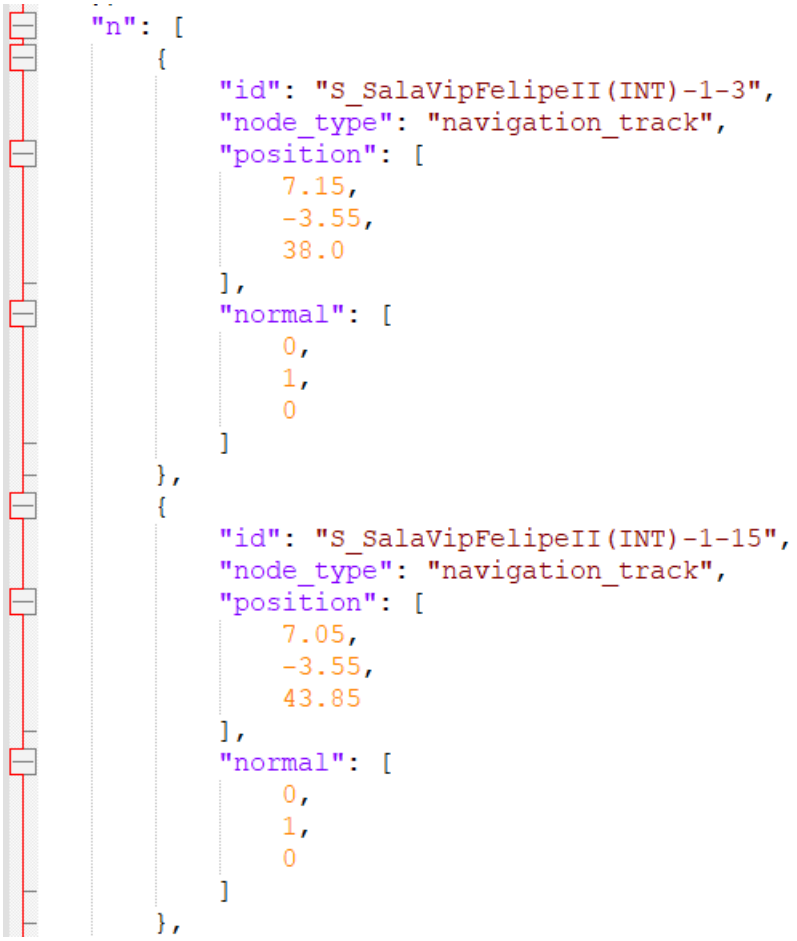


Figura 37: Lista de nodos de navegación

Por último, y para terminar con este apartado, hace falta destacar el uso de otro tipo de archivo relevante a la hora de gestionar las panorámicas de navegación. En la mayoría de los estadios se pueden encontrar localidades con salas VIP asociadas. Cuando esto ocurre, existe un fichero CSV que se lee al inicializar el recinto y que dispone de información relevante para el plugin Viewer3DPluginRelated del visor web. Básicamente, desde la vista panorámica del asiento se puede acceder a la sala VIP asociada que contiene la información generada mediante las meshes o el archivo de configuración previos. Puede existir más de una sala asociada a un solo asiento y todo ello queda reflejado como una lista de objetos que contienen el ID, el label, el label en negrita y el prefijo que debe tener a nivel de estilo en el visor la sala VIP asociada. En caso de no existir el archivo CSV, el plugin de relaciones se marca como “null” y se omite durante el proceso de serialización del JSON de configuración final.

Figura 38: Vista panorámica con información related

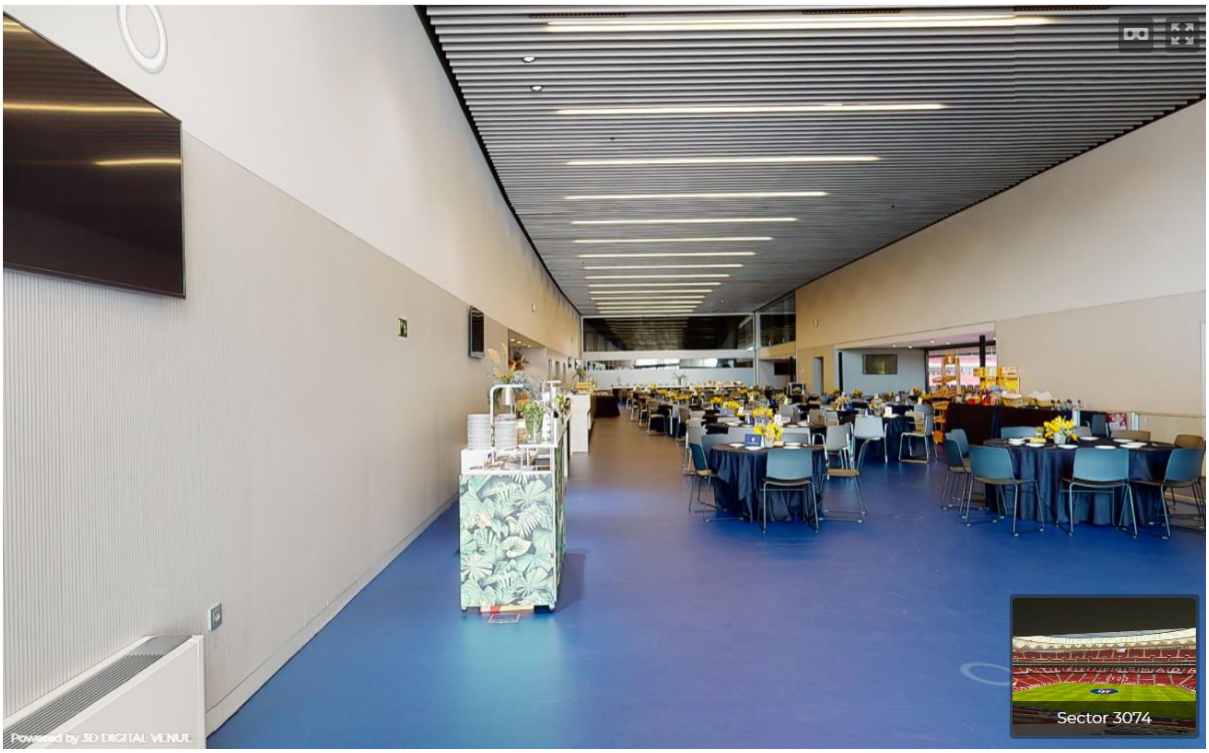


Figura 39: Sala VIP related de la figura 38

5.2. Gestor de vistas panorámicas

Cada uno de los tipos de vistas panorámicas mencionadas anteriormente se pueden generar de forma individual o grupal mediante el uso de la interfaz y el gestor de vistas panorámicas en sí. La clase “PanoManager.cs” es la encargada de gestionar la generación y distribución masiva de

contenido de forma fácil y transparente. Para ello, existen métodos público-estáticos que se llaman a través de botones existentes en la interfaz, tal y como se ha comentado en el apartado 4.2.2.

Así pues, el gestor de vistas panorámicas consta de varios métodos principales encargados de la generación de vistas de múltiples formas posibles y con distintos propósitos. Estos métodos se desglosan en:

- Generate from CSV: Este método abre una ventana emergente de selección de archivos para cargar un CSV con un listado de IDs que determinan las panorámicas a generar. Así pues, tras leer el fichero y mediante el uso de la configuración actual, se seleccionan las entidades correspondientes y se añaden a una lista de objetos que será utilizada en el método GeneratePanoCallback para generar de forma procedural todas las vistas en el archivo.

- Generate Linked: Método que permite cargar un CSV a partir de una ventana emergente que contiene información sobre panorámicas enlazadas. Tras leer el archivo, según si se ha seleccionado guardar en local o subir a servidor, se genera una pano de tipo enlazada y se almacena o se sube el JSON con la información de la vista correspondiente.

- Generate from VenuePanos: Método que permite generar panorámicas a través de la clase VenuePanos. Dicha clase guarda las relaciones entre polígonos y entidades seleccionadas para crear una lista de entidades de VenuePano y pasarla por parámetro a GeneratePanoCallback con el propósito de generar vistas a partir de dicha información.

- Generate from hierarchy: A pesar de no existir un método como tal para la generación desde jerarquía (se usa el propio GeneratePanoCallback), se puede considerar como una herramienta más. En este caso, la lista de objetos que se pasan por parámetro al método principal de generación se extrae de la selección de entidades en la escena activa.

El método principal del manager y en el que se basan el resto de subherramientas es GeneratePanoCallback. Todos y cada uno de los métodos que hacen referencia a la generación de imágenes renderizadas confluyen aquí. Este método consta de 3 partes:

- GenerateVenueConfig: En caso de seleccionar mediante la interfaz la generación del JSON correspondiente a Viewer3DVenueConfig (con información global del conjunto de panorámicas), se llama a este método para generar y serializar los datos generales de las vistas.

- GeneratePanoAndSaveLocal: Recibe una lista de objetos seleccionados, genera las vistas panorámicas correspondientes para cada uno de ellos y las guarda en local.

- GeneratePanoAndUpload: Recibe una lista de objetos seleccionados, genera las vistas panorámicas correspondientes para cada uno y las sube al servidor.

Los dos últimos métodos comentados son la clave para entender el funcionamiento de la herramienta de generación de panorámicas. Tanto el uno como el otro comparten todo aquello relevante en cuanto a generación y se bifurcan a la hora de decidir qué hacer con el contenido generado. Así pues, ambos, tras recibir una lista de objetos seleccionados previamente desde la interfaz (por ejemplo, usando archivos CSV) o la jerarquía (usando las entidades del recinto), la primera acción que realizan es una ordenación por “State” de las entidades recibidas.

Tal y como se ha comentado en apartados previos, un “State” no es más que un conjunto de subescenas asociadas a una entidad. Esto quiere decir que, por ejemplo, en el caso de las salas de interior, se dispone de una escena adicional con todos los props que conforman dicha sala. En el caso de asientos más estándar, la subescena podría estar formada por la pista, la cubierta y la grada del estadio. Existe una herramienta adicional en Venue Editor para gestionar los estados de las entidades y poder cargar o descargar las escenas correspondientes a dicho “State”. De esta forma, y mediante el uso del mánager asociado a esta herramienta, podemos cargar el “State” asociado a la entidad justo antes de generar la vista panorámica para liberar de carga el proyecto y sólo usar los “States” a la hora de generar el contenido final.

Volviendo al tema de la ordenación de entidades; la lista de objetos recibida puede estar formada por entidades con “States” muy diferentes entre sí. Para evitar sobrecargas a nivel computacional (cargar y descargar un estado y hacer lo mismo con el siguiente para, a posteriori, volver a cargar el estado inicial) se ordenan las entidades por “State” para generar vistas por grupos de todas aquellas que compartan un conjunto de subescenas. Esto hace mucho más rápido un proceso de generación que, de cualquier otra forma, sería muy pesado y añadiría una carga adicional extra a un proceso costoso de por sí.

Con todas las entidades ordenadas, tanto las que tienen “State” como las que no, se procede a generar una cola de GameObjects acumulativa que será la que gestione la cantidad de panorámicas que se generan y se almacenan al mismo tiempo. Tras la realización de distintos tipos de pruebas con recintos reales, se determina que 20 panorámicas al mismo tiempo es el máximo que puede soportar el Editor sin sufrir problemas de memoria. Así pues, en grupos de 20, se genera una segunda cola que contiene las entidades que se añaden a la cola de panorámicas. Sobre esta lista de entidades, se llama al método GeneratePanoQueue. Este método se encarga de iterar sobre los objetos en la cola de panorámicas y generar una a una hasta que se termina la cola. Todas las panorámicas generadas se añaden a una lista temporal de tipo “Pano” (instancias de la clase “Pano.cs”) que será la que se use para parar el proceso una vez terminado o para subir la lista de contenido al servidor de AWS.

Entrando algo más en detalle sobre la generación de cada una de las panorámicas, dentro de GeneratePanoQueue, y para cada una de las entidades, se realiza una llamada a GeneratePano. Este es el método encargado de renderizar el contenido de la siguiente forma:

- Se recibe por parámetro el GameObject sobre el cual se debe realizar la acción.

- Se comprueba si el GameObject forma parte de las entidades existentes en la configuración actual del recinto.

    - Si forma parte:
        - Mediante el uso de un switch-case se determina el tipo de entidad (Tier, Section, Row o Seat) y cuál es la entidad mínima sobre la cual se genera el contenido (silla representativa en el caso de Section o conjunto de sillas representativas de sectores en el caso de Tier).

        - Se llama al método GeneratePanoInSeat. Este método abre la silla como si hubiera alguien sentado en ella, carga el “State” correspondiente si no lo está, crea una nueva instancia de Pano y hace una llamada a GenerateCubemap o GenerateEquirectagular de ésta para generar el contenido en base a la selección de propiedades de la interfaz.

        - Se devuelve la instancia de la clase para que sea almacenada en la lista de panorámicas mencionada anteriormente.

    - Si no forma parte:
        - Se comprueba si la panorámica que se quiere generar es a partir de un objeto en jerarquía que no necesariamente es una entidad del recinto. En caso afirmativo, se llama al método GeneratePanoInObject; que genera una instancia de Pano y llama al método correspondiente según si es cúbica o esférica.

        - Se comprueba si existe una instancia de VenuePanos creada. En caso afirmativo, se recoge la silla representativa de la entidad almacenada en la instancia y se llama a GeneratePanoInSeat desde dicha silla.

Es a partir de este punto en que se bifurcan los caminos de la generación local y la generación con subida a servidor. En cuanto a la generación en local, al terminar de generar las primeras 20 panorámicas de la cola, se llama de forma recursiva al método RecursiveGenerate hasta que la lista de objetos inicial (la que incluye todos los seleccionados) ha sido completada. Dentro del proceso de

generación de cubos o esféricas, se guarda el contenido en local si así ha sido especificado en la interfaz. Para la distribución y subida de contenido, ver el siguiente apartado.

5.3. Distribución de contenido

Llegados a este punto, se dispone de una lista de instancias de la clase Pano que, salvo que se haya seleccionado la opción de guardar en local, debe ser subida a un servidor de AWS. Es aquí donde entra en acción el gestor de red “NetworkManager.cs”. Este script se ha desarrollado de forma genérica para permitir la subida de cualquier tipo de archivo a cualquiera de los servidores disponibles en la empresa. Así pues, las acciones a realizar son el login, la subida y la invalidación de caché. Existen distintos tipos de endpoint a los que realizar la petición https según el proceso a llevar a cabo. Cada uno de ellos viene definido por el departamento de IT de la empresa.

En el caso del login, se dispone de un archivo CSV interno ubicado en el proyecto de Unity en cuestión. A partir de la lectura de dichas credenciales, se rellena un WWWForm y se realiza una petición de tipo POST al endpoint asociado al inicio de sesión. Si la petición tiene éxito, se pueden realizar subidas de archivos sin problemas. Esto es un paso imprescindible para poder realizar el resto de las acciones.

Tras el inicio de sesión, se procede a la subida de datos. Existen dos maneras diferentes de realizar dicha subida: fragmentada o unificada. En el caso de la fragmentada, se define un máximo número de fragmentos que se pueden subir al mismo tiempo y, si los datos superan dicho número, se generan pequeños paquetes de datos que se suben de forma independiente y agilizan el proceso. Para la subida unificada, al no superar el límite establecido, se permite subir de golpe todos los archivos seleccionados sin fragmentar. En cualquiera de los dos casos, ya sea unificado o fragmentado, se rellena un WWWForm con los datos de la carpeta donde se sube el archivo, lo bytes que lo componen y el nombre del archivo en sí. Por último, se realiza una petición de tipo POST a endpoint asociado a la subida de datos.

Es importante destacar que la API diseñada por el departamento de IT permite generar formularios simples o múltiples. Esto significa que se puede generar un formulario de subida para cada archivo o, alternativamente se puede atacar a un conjunto de carpetas y archivos para gestionar en una sola pasada todos los archivos al mismo tiempo.

Para terminar con el gestor de red, falta detallar la parte de invalidación de caché. Existe un endpoint específico para realizar este tipo de petición y sirve para invalidar los contenidos subidos previamente a una ubicación en concreto. De esta forma, cuando se suban nuevos contenidos, se actualizarán los datos en la web y dejarán de mostrarse los que han quedado obsoletos. Se puede realizar invalidación de caché tanto para archivos específicos como para conjuntos de archivos localizados en una carpeta.

5.3.1. Metodologías de generación y subida

El proceso de generación de vistas panorámicas y subida a servidor es costoso y puede llevar varias horas según las dimensiones del proyecto. Es importante que el operador pueda disponer en todo momento de información relevante con respecto al tiempo total que va a suponer el proceso y que éste sea cancelable. Para ello, se han generado barras de progreso cancelables mediante el uso de CancellationTokenSource (Microsoft Corporation, s.f). Al inicializar la vista de interfaz del aplicativo se genera un nuevo Token asociado a las tareas asíncronas que se van a realizar. Después, mediante un sistema de eventos, se implementa la detección de teclas para generar un mensaje de cancelación a través del token que permite cancelar la tarea si se pulsa la tecla “Esc”. Este proceso bloquea la ejecución de los threads activos y cancela las acciones mostrando una ventana emergente que avisa al usuario de la cancelación de la tarea. Esto puede ocurrir también en el caso de que se produzca algún tipo de error bloqueante durante el proceso de subida de datos al servidor.

En cuanto a la estimación del tiempo, se muestra un mensaje en la parte inferior de la barra de progreso que da información al usuario de la cantidad de tiempo que va a tomar realizar la generación y la subida. Puesto que en la primera iteración no se dispone de información alguna, se fija un valor determinado para mostrar un primer número y, tras la subida de la primera vista, se actualiza el tiempo extrapolando los datos reales según la cantidad de vistas a subir en total.

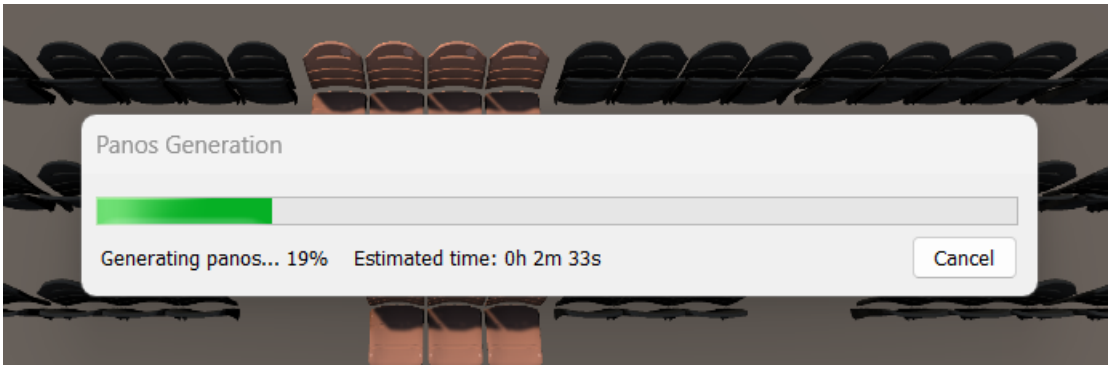


Figura 40: Tiempo estimado de generación y subida en barra de progreso

De cara a la optimización de tiempos y memoria del aplicativo, se han realizado distintos tipos de pruebas que hacen referencia a las metodologías de generación y subida de panorámicas. Los tipos de metodologías son:

- Procrecimiento 1:
    - Generar una cola de panorámicas de un tamaño predeterminado.
    - Subir toda la cola de golpe.
    - Esperar a la subida antes de la próxima generación.

- Procedimiento 2:
    - Generar una cola de panorámicas de un tamaño predeterminado.

  - Subir toda la cola de golpe.
  - Generar la siguiente cola mientras se sube la anterior, en paralelo.

- Procedimiento 3:
  - Generar una vista panorámica.
  - Subir la vista panorámica con todos sus ficheros en el mismo POST.
  - Generar la siguiente vista al terminar la subida de la anterior.

- Procedimiento 4:
  - Generar una vista panorámica.
  - Subir la vista panorámica con todos sus ficheros en el mismo POST.
  - Generar la siguiente vista mientras se sube la anterior, en paralelo.

En la siguiente tabla se pueden observar los resultados en cuanto a tiempos estimados de generación y subida con colas de un máximo de 20 panorámicas:

	Procedimiento 1	**Procedimiento 2**	**Procedimiento 3**	**Procedimiento 4**
30 panorámicas	3 mins y 42 seg	1 min y 37 seg	4 min y 24 seg	2 min y 50 seg
50 panorámicas	6 mins y 52 seg	3 mins y 30 seg	8 mins	4 mins
Memory issues	No	Yes	No	Yes

Mesa 4: Estimación de tiempos de generación y subida

Como se puede observar en la tabla anterior, generar panorámicas sin esperar a que termine la subida de la panorámica anterior causa serios problemas de memoria que se traducen en crashes de Unity a la hora de realizar el proceso; esto tiene fácil explicación. Por lo general la subida tarda algo más que la generación (en función de los elementos que hay en escena) y, para subir los contenidos, la memoria del engine no se puede limpiar hasta que no ha terminado el proceso. Esto provoca un overload en la generación de texturas para la creación de las imágenes que provoca el fallo del motor gráfico, que rápidamente se queda sin memoria por el poco margen de maniobra del que dispone el garbage collector para liberar la información. Así pues, estos datos descartan por completo las metodologías 2 y 4 y dejan como primera opción los procedimientos 1 y 3.

Así pues, sólo hace falta observar los tiempos de generación y subida obtenidos usando cada una de las metodologías. El procedimiento 1 es claramente más rápido que el procedimiento 3, puesto que el proceso de generación no se para con tal de realizar la subida de datos y esto nos permite después subir toda la información de golpe al servidor. Mediante el procedimiento 1, la subida se realiza de forma múltiple, agrupando todos los datos en un solo Form y realizando una sola petición. El procedimiento 2, sin embargo, requiere generar una panorámica y un formulario de datos para cada una de las vistas, lo que supone un incremento notable en los tiempos de generación y subida totales.

Con respecto a las mejoras de tiempo comparándolo con la versión 3, es difícil de analizar correctamente los datos, puesto que los entornos de desarrollo son muy diferentes a los entornos de producción reales. Esto quiere decir que, normalmente, en el departamento de producción trabajan con proyectos mucho más grandes que el proyecto base utilizado para desarrollar. Estos proyectos incluyen mucha más información y nivel de detalle con respecto a la cantidad de assets que se utilizan, la postproducción aplicada en las cámaras y la generación de contenidos a una mayor resolución que la trabajada en desarrollo. Al fin y al cabo, para optimizar los tiempos de desarrollo es necesario trabajar con aquellos datos que sea suficientes para corroborar el correcto funcionamiento de las herramientas, aunque esto implique, en la mayoría de los casos, no disponer de proyectos 100% reales.

5.4. Requisitos de instalación

Para el uso e instalación de la herramienta de generación de panorámicas, es necesario instalar por completo el software de Venue Editor 4, de la misma forma que se hizo en la versión 3. Para ello, se hace lo siguiente:

- Se genera un fichero binario mediante el uso del sistema de empaquetado que ofrece Unity (unitypackage) que contiene todos los scripts y assets necesarios para el uso de la herramienta de Editor.

- Se acuerda a nivel de empresa una versión concreta de Unity (la que se ha usado durante el desarrollo) para asegurar que la herramienta funciona en su totalidad y que el unitypackage sea compatible. Este paso es importante, puesto que se han encontrado fallos durante el ciclo de vida del Editor en la empresa a la hora de importar paquetes a versiones que difieren de la versión de Unity original.

- El package va acompañado de un fichero que indica la versión de Venue Editor 4 (para control de actualizaciones).

- Se sube el binario y el fichero de la versión a un directorio de S3 de AWS y el equipo de IT desarrolla una API de lectura y escritura.

- Se desarrolla un módulo de consulta de versión de esta API desde los distintos modos de operador que existen en Venue Editor 4 para comprobar si existe una actualización con respecto a la versión subida en AWS. Si esto ocurre, se descarga el binario con el unitypackage y se fuerza la descompresión y apertura del paquete.

- El operador siempre tiene disponible en la ventana de Editor la versión con la que está trabajando para que el departamento de ingeniería esté al tanto de si es o no la última creada.

Este es el procedimiento a seguir para generar un paquete de instalación para los distintos departamentos que usan la herramienta y sólo funciona si hay conexión a internet. Esto permite mantener en desarrollo el software e introducir cambios a medida que se avanza en la creación de la herramienta, se resuelven bugs o se añaden nuevas funcionalidades.

5.5. Instrucciones de instalación

Para instalar el software sólo es necesario importar el unitypackage mencionado en el apartado anterior desde el package manager de Unity una vez se ha subido a Amazon Web Services. Para ello, los distintos departamentos deben asegurarse de que la versión de Unity con la que están trabajando coincide con la versión de la herramienta de Editor y sólo es necesario realizar la instalación una vez. En iteraciones posteriores, una ventana emergente informando de nueva versión de la herramienta aparece nada más abrir cualquier proyecto de Unity que disponga de dicha herramienta. Esto facilita el trabajo a los distintos departamentos, que se pueden despreocupar por completo de revisar si existe o no una nueva versión. En caso de escoger no actualizar, esta ventana emergente se volverá a mostrar tras abrir y cerrar cualquier proyecto.

6. Demostración

6.1. Instrucciones de uso

Para el uso correcto de la herramienta es necesario seguir una serie de pasos para asegurar que el producto final es el esperado. Así pues, los primeros pasos a seguir antes de generar panorámicas son:

- Abrir la ventana de editor: Para ello nos dirigimos a la barra de herramientas de Unity y seleccionamos la pestaña “Venue Editor”.

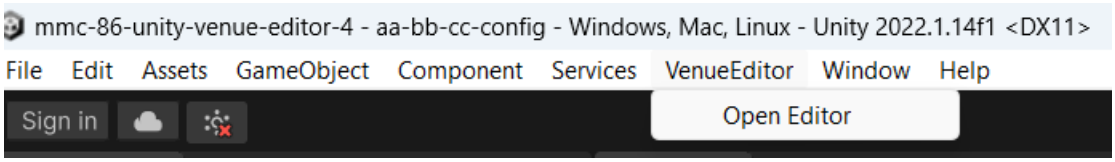


Figura 41: Abrir ventana de editor

- Descargar nueva versión si existe.

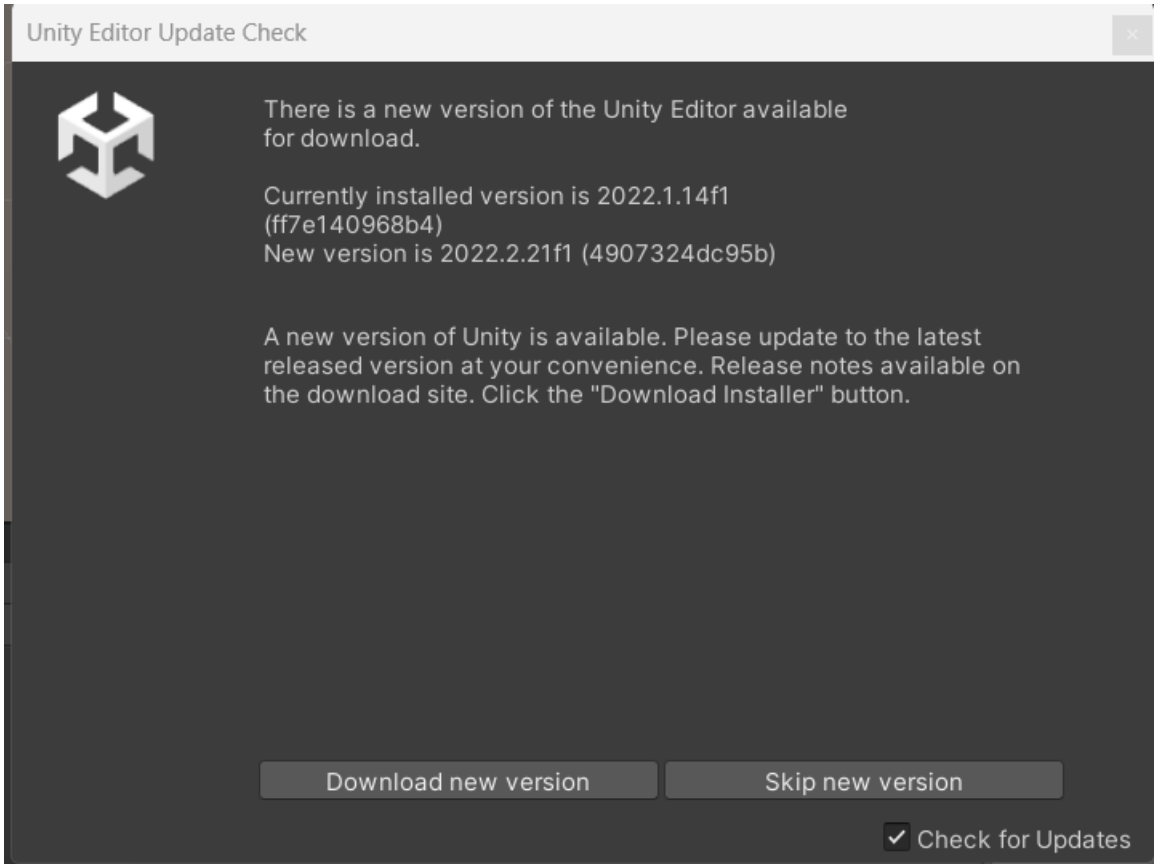


Figura 42: Pop up de nueva versión

- Cargar XML: Este paso es indispensable para realizar cualquier acción sobre un recinto. Para ello hay que acceder a la pestaña “Seating” dentro del editor y seleccionar el apartado “Seat Data”. Dentro de dicho apartado está disponible la opción de “Load XML” para cargar un recinto a partir de un set de datos específico.

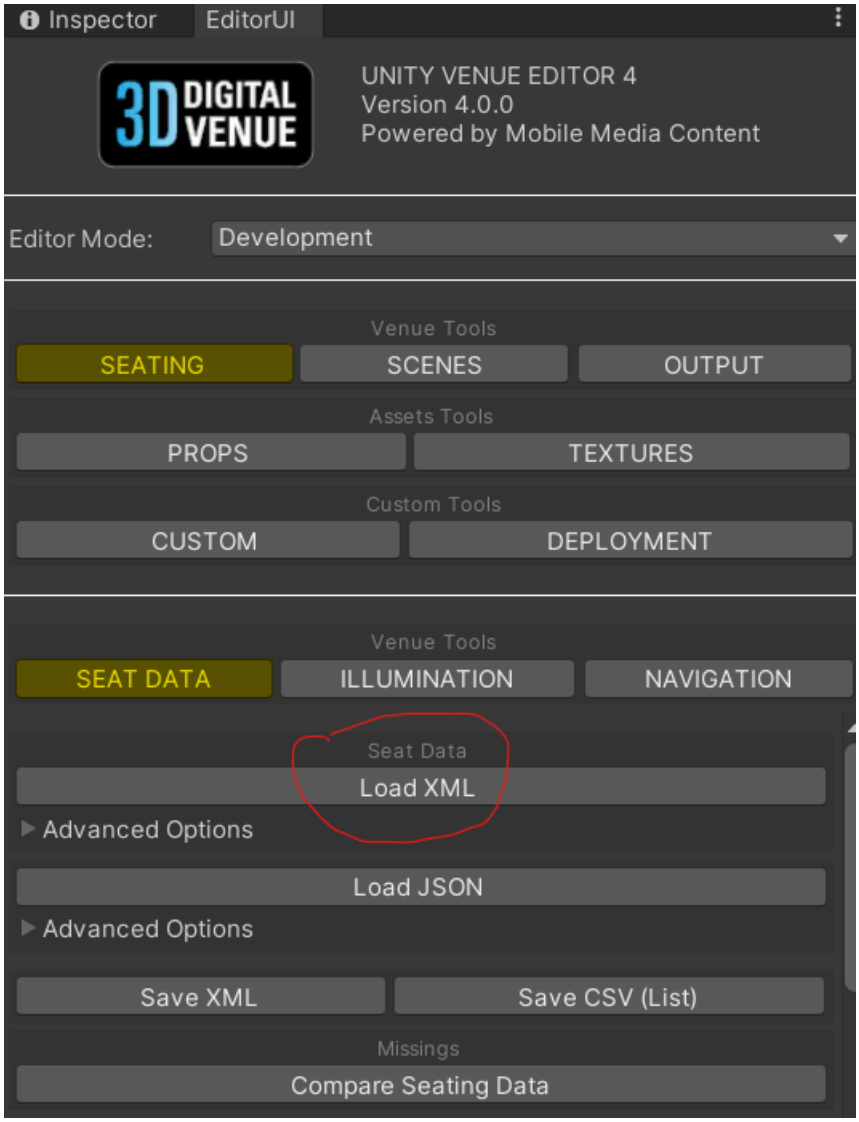


Figura 43: Load XML button

- Generar States: Tanto este paso como el siguiente son pasos opcionales. Por lo general pueden existir múltiples agrupaciones de escenas llamadas “States”. Para generar un nuevo state hay que cargar de forma aditiva las escenas que se quieran incorporar al State y crear uno nuevo desde la pestaña “Scenes” del editor. Dentro de la pestaña, se abre el desplegable “Create new state”, se asigna el nombre deseado y se pulsa el botón “Create State for Current Config”.

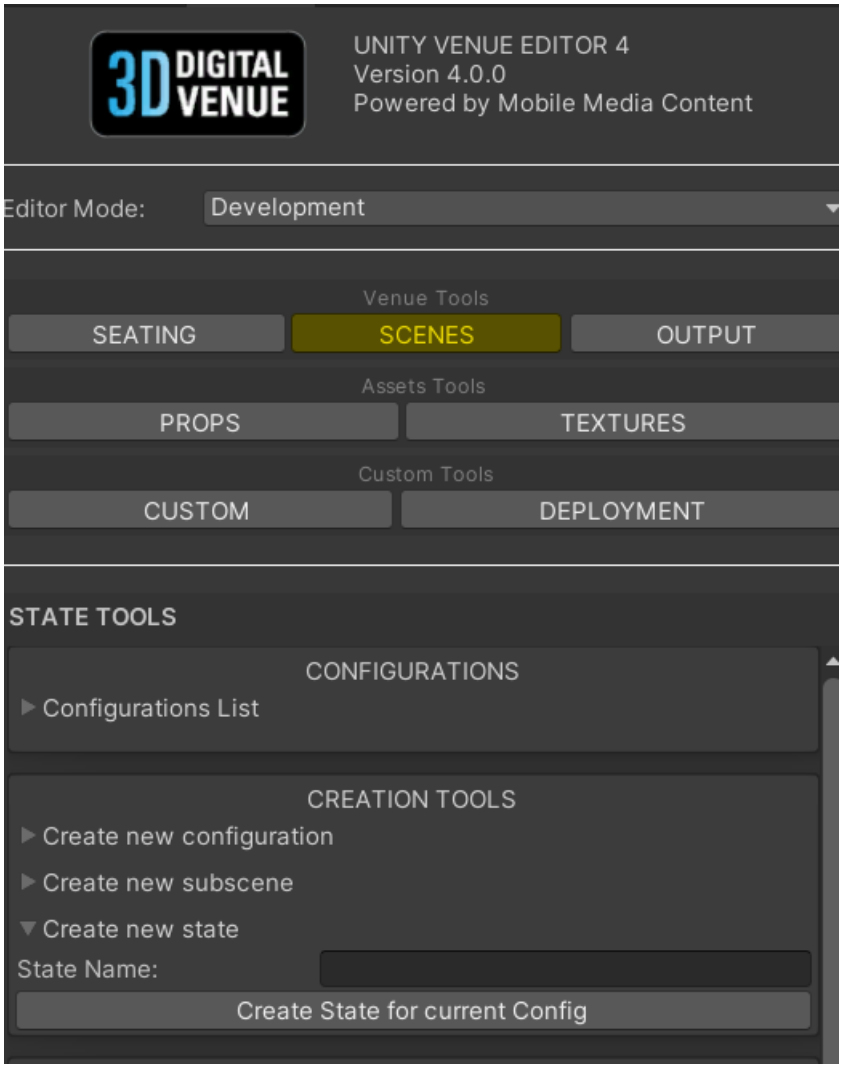


Figura 44: Create state window

- Asignación de States: Si se ha seguido el paso anterior y se han creado diferentes States para la configuración actual, estos se deben asignar a entidades específicas del recinto para tener en cuenta la carga aditiva de escenas a la hora de generar las vistas panorámicas. Para ello, seleccionamos en la jerarquía la entidad a la cual queremos asignar el State y desplegamos el menú “Assign State from list”. Aquí se encuentra una lista que incluye todos los States creados. Para asignarlo a la entidad sólo tenemos que pulsar el botón correspondiente al State deseado.

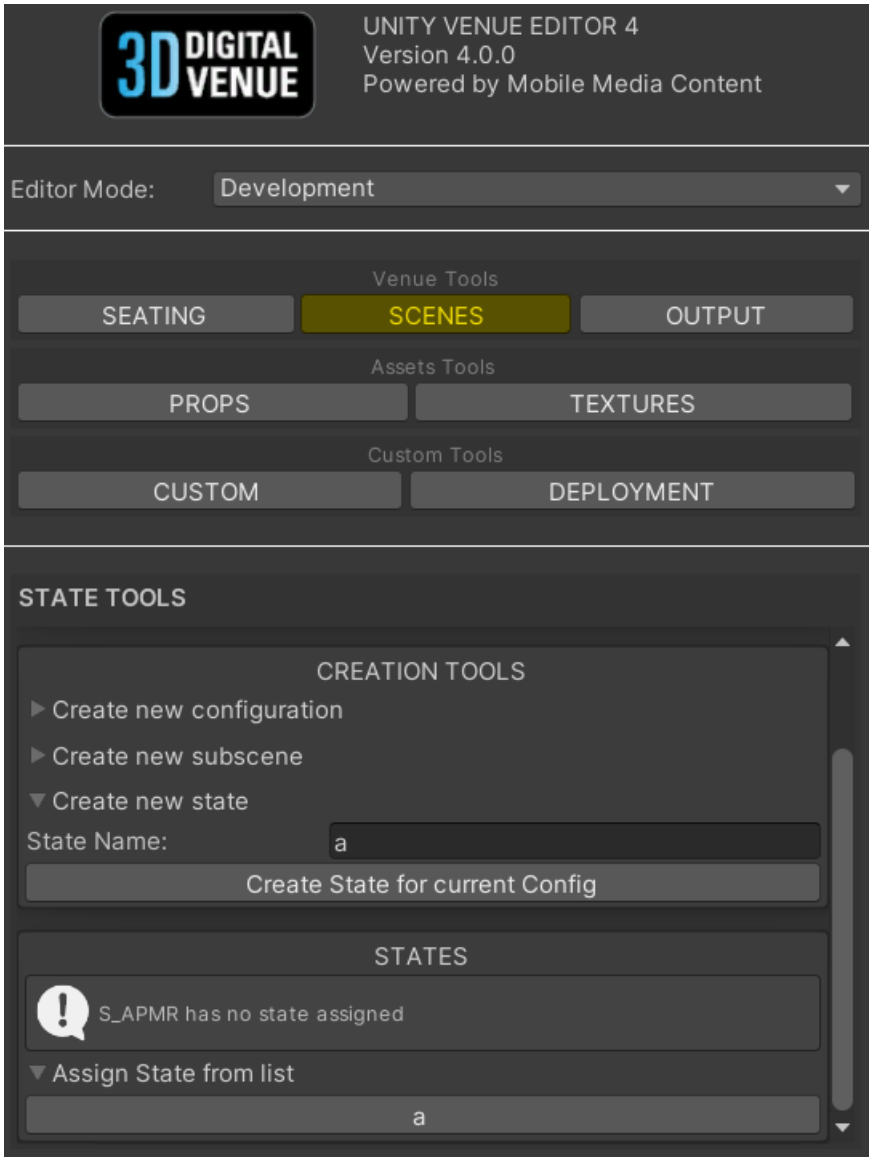


Figura 45: Assign state window

Con los pasos previos a la generación de panorámicas realizados, es el momento de entrar en detalle en la generación de vistas en sí. Hay distintos procedimientos según el proceso que se vaya a realizar. Aún así, lo primero de todo es escoger en el apartado de configuración los parámetros correspondientes a la vista panorámica. Para ello, navegamos a la pestaña “Output” y al menú “Panos” dentro del editor. Lo primero que se encuentra el usuario son varios menús desplegables y toggles con las diferentes opciones de configuración. Así pues:

- Seleccionamos la región a la que se va a subir el contenido (opcional en caso de generar contenido en local).
- Seleccionamos la resolución de las imágenes en alta.
- Seleccionamos el tipo de panorámicas que se van a generar.
- Seleccionamos el toggle “Save Local” si queremos guardar el contenido en local (esto es opcional y su selección invalida el desplegable de la región de S3 de AWS).
- Seleccionamos el toggle “Use Tiles” en caso de querer generar imágenes tileadas.

- Seleccionamos el toggle “Generate Global Config” si queremos generar el JSON que contiene las especificaciones genéricas del recinto sobre el cual se generan las panorámicas.
- Seleccionmos el aspecto ratio del thumbnail generado para las panorámicas (este paso es opcional y por defecto tiene valor 1:1).

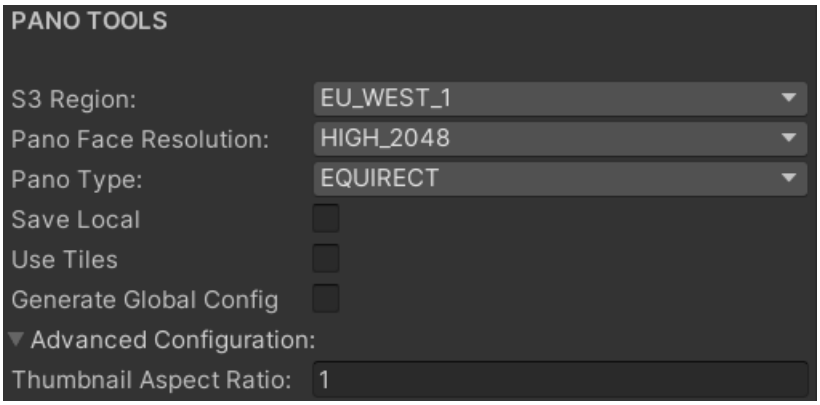


Figura 46: Panos config UI

Todas estas opciones tienen valores por defecto que deben ser modificados por parte del usuario según el tipo de panorámica que se vaya a generar y los datos que se quieren obtener a partir del proceso. Hecho esto, existen distintos procedimientos según el tipo de método de generación que se vaya a realizar:

- Venue Panos:
    - Arrastramos el objeto de referencia que se va a utilizar para la generación de Venue Panos.
    - Marcamos el toggle “Use venueseats reference” si queremos tener en cuenta los IDs de las entidades del recinto cargado para la generación de las carpetas de destino.
    - Pulsamos el botón “Compute Venue Panos from Referene” que aparece tras arrastrar el objeto al campo de la interfaz.
    - Pulsamos el botón “Generate panos from Venue Panos” en la interfaz para generar las vistas panorámicas.

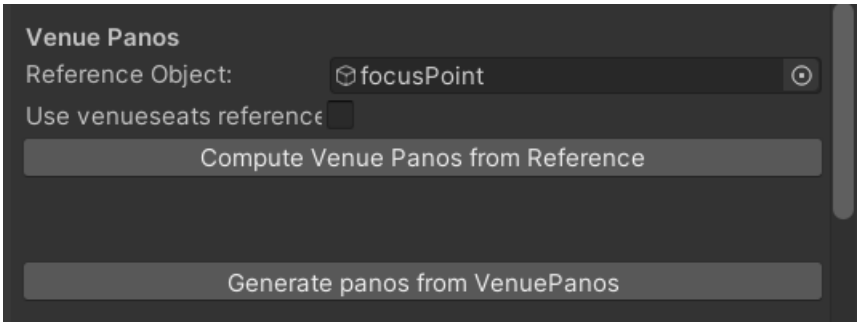


Figura 47: Venue Panos config UI

- Generate panos from CSV:
    - Pulsamos el botón “Generate Linked” para abrir un panel emergente que permita seleccionar un CSV con información de panorámicas enlazadas para empezar la generación.
    - Pulsamos el botón ”Generate” para abrir un panel emergente que permita seleccionar un CSV con un listado de IDs de entidades sobre las cuales se generarán las vistas.

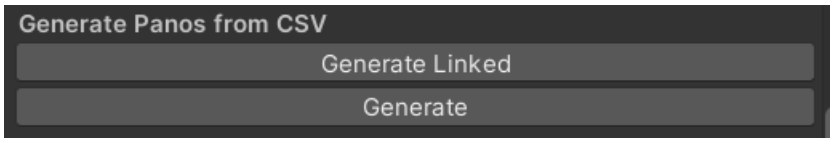


Figura 48: Panos from CSV config UI

- Panos from hierarchy:
    - Seleccionamos una o varias entidades válidas en la jerarquía: VenueSeats, Tier, Section, Row o Seat.
    - Marcamos el toggle del tipo de generación deseada: Section(s), Row(s), Seats o Parent + Seats. Esto determinará si las panorámicas se tiran solo sobre la entidad base o se tiran en cascada desde el padre hasta los hijos.
    - En caso de tener seleccionada una silla aparecen alternativas en la interfaz:
        - Seleccionamos el toggle “Parent Linked” para generar una vista enlaza a la entidad padre de la silla.
        - Opcionalmente podemos pulsar el botón “Set Section Representative Seat” o “Set Row Representative Seat” para cambiar la silla sobre la que se genera la panorámica en el caso de tener seleccionada una entidad padre.

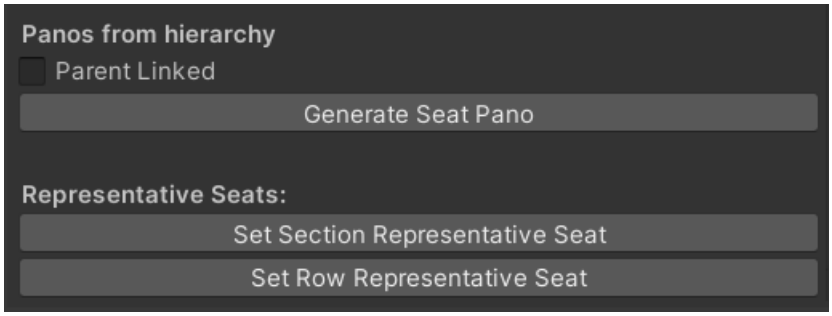


Figura 49: Seat hierarchy selection UI

    - Pulsamos el botón “Generate Seat Pano” para empezar la generación.

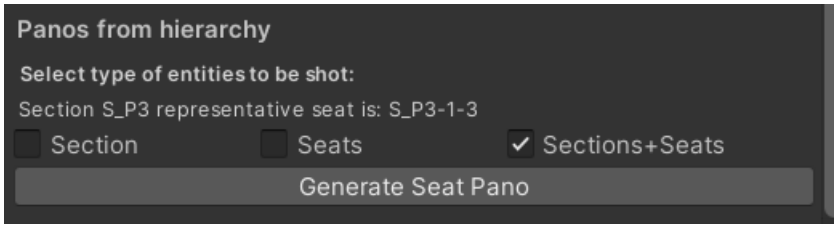


Figura 50: Panos from hierarchy config UI

6.2. Tests

Todos los proyectos de Unity en Mobile Media Content se trabajan a partir de un proyecto base o plantilla que se desarrolla conjuntamente entre el departamento de ingeniería y el departamento de producción. Es en este punto en el que se acuerda entre ambas partes una configuración del proyecto determinada teniendo en cuenta la forma de trabajar del departamento de producción y las restricciones que existen por parte del departamento de ingeniería en función del planteamiento del desarrollo. De esta forma, se define un mínimo de elementos que debe contener la jerarquía para que la herramienta de editor funcione correctamente y cómo se debe operar para la manipulación de estos elementos. Esto quiere decir que, en el departamento de producción deben ajustar sus prácticas de usabilidad para adaptarse, en cierta manera, a Venue Editor.

Por otro lado, y del mismo modo que la jerarquía, se acuerda entre ambas partes las configuraciones específicas en cuanto a los project settings (Unity Technologies, 2023) y los quality settings (Unity Technologies, 2023) de Unity. Por lo general, en aquellos proyectos que sirvan para trabajar en edición de recintos, esto no debe cambiar salvo que se acuerde entre ambas partes la realización de cambios por cualquier tipo de cuestión.

Esta plantilla lleva siempre incorporada la última versión incorporada de Venue Editor. De esta forma, cada vez que se abre a ventana de editor en cualquier proyecto trabajado a partir del template, este recibe un aviso de actualización siempre y cuando haya una nueva versión.

Una vez implantado en aquellos departamentos que lo requieran, los operadores deberán reportar bugs o propuestas de mejora mediante Jira. Esto escala al departamento de ingeniería, que será el encargado de planificar el momento adecuado para realizar dichas mejoras e incorporarlas en una nueva versión del Venue Editor 4. Estas mejoras no se lanzan de nuevo al departamento de producción sin más; en fase de prueba, se permite realizar versiones intermedias a las que sólo tienen acceso el departamento de ingeniería y perfiles más senior dentro del departamento de producción. Llegados a este punto, se repite el proceso de empaquetado explicado en el apartado 5.4 para obtener las actualizaciones de versión.

El proyecto de Venue Editor 4 está muy cerca de poder llegar a una primera fase de implantación sobre la cual puedan trabajar perfiles más senior del departamento de producción e involucrar a más gente en esta última fase del proceso. Sin embargo, al estar aún en desarrollo, no ha sido posible

más que probar las herramientas en un entorno controlado dentro del departamento de ingeniería y bajo la supervisión del jefe de departamento. A la hora de implantar Venue Editor 4 será necesario trabajar con proyectos de cero, usando la versión de Unity base sobre la que se ha trabajado (Unity 2022 hasta el momento, aunque es posible que antes del lanzamiento se realice una versión a Unity 2023). En caso de querer reaprovechar aquellos proyectos más antiguos, se realizará una migración específica del proyecto partiendo de los assets que sean necesarios para su funcionamiento. Una migración automática entre proyectos de la versión 3 y la 4 tiene altas probabilidades de fracasar. Para solucionarlo, se planteará un sistema de empaquetado de assets relacionados con el apartado más artístico del proyecto para facilitar dicha migración. Ambas versiones de Venue Editor no deberán convivir en entornos de producción reales.

Venue Editor 3 fue desarrollado en Unity 2017 y ha ido migrando a futuras versiones hasta Unity 2020, donde se empezaron a detectar problemas de retrocompatibilidad. Una potencial mejora de Venue Editor 4 y todas sus herramientas es la escalabilidad a nivel programático del proyecto a futuras versiones de Unity. Empezando por la versión 2022, se propone realizar actualizaciones de versión cada año y aprovechar la estructura de datos actual para seguir trabajando e incluso mejorar con el tiempo la arquitectura del programa. El nivel de detalle de las clases, así como su encapsulación y especificidad, permiten expandir de forma natural el proyecto sin interferir directamente en la parte más central de la herramienta. Esto permite a perfiles más junior desarrollar sobre una base sólida sin peligro de alterar el “core” del aplicativo y a perfiles más senior entrar a diseñar y retocar aquellas partes más complejas del código. Además, facilita enormemente la portabilidad a futuras versiones de Unity, teniendo en cuenta que se debe mantener siempre al día todo aquello relacionado con nuevas incorporaciones al motor gráfico.

Así pues, para probar que las funcionalidades implementadas funcionan correctamente, se ha hecho uso del visto web desarrollado por el departamento de ingeniería especializado en IT. Esta herramienta permite previsualizar los contenidos generados en el navegador sin necesidad de que sean subidos a un servidor en uso por parte de los clientes y que se generen problemas a la hora de sobrescribir contenidos actualmente en marcha en las webs de clientes. Para ello, hay carpetas habilitadas dentro de cada uno de los servidores donde se pueden subir los materiales para acceder posteriormente en web mediante una URL específica de un recinto de prueba. Tan solo necesitamos un ID de configuración inventado al que poder acceder y el ID de la entidad que se ha subido al servidor.

7. Conclusiones y líneas de futuro

7.1. Conclusiones

7.1.1. Lecciones aprendidas

Durante el proceso de construcción, no sólo la herramienta de generación de panorámicas sino de todo el editor, se ha aprendido a trabajar de forma ágil y eficiente con la API de Editor de Unity así como a lidiar con problemas derivados del workflow general trabajando en modo Play. Existe un cambio de paradigma importante entre el desarrollo de aplicaciones para funcionar como Standalone, o en otro tipo de plataformas, con respecto a la programación de aplicaciones pensadas para funcionar en editor. Aunque en aplicaciones de Editor se puedan usar prácticamente las mismas herramientas que en Play, la forma de usarlas no es exactamente la misma; empezando por el uso de Components asociados a GameObjects.

Esto es algo que se ha cambiado con respecto a la versión 3 de Venue Editor, que añadía y destruía componentes asociados a objetos en tiempo real y requería del método “GetComponent<>()” de Unity; conocido por ser extremadamente lento, de la misma forma que el “Find()”. La estructura de clases generada a nivel de entidad, configuración y recinto de Venue Editor 4 permite almacenar todos los datos necesarios en clases cuyo ciclo de vida trasciende al ciclo de vida de la aplicación. Para conservar todos los datos se ha optado por serializar una gran cantidad de datos correspondientes al estado actual del aplicativo en un archivo JSON. Lógicamente, esto tiene sus pros y sus contras; empezando por la dificultad añadida de debuggar los procesos de deserialización de los archivos, implícitamente anexados en la librería de Newtonsoft. Esto, sin embargo, permite que diferentes empleados del departamento de producción puedan trabajar en el mismo proyecto pero en ordenadores diferentes, conservando el estado actual.

Otro aspecto importante para comentar es el manejo de la asincronía en editor. A diferencia del modo Play, en que los métodos asíncronos se trabajan mediante el uso de corutinas, en Editor es necesario trabajar usando Threading. La creación de métodos async/await, Tasks a las que se puede pedir que el usuario espere a que sean completadas y devuelvan ciertos valores, es una forma muy común de control de asincronía cuando se trabaja con aplicaciones de escritorio desarrolladas directamente en .NET o C++. Sin embargo, también es una herramienta muy limitante cuando se utiliza un motor gráfico como Unity. Unity no es thread-safe, lo que quiere decir que pierde totalmente el control sobre aquellos hilos que nacen del thread principal. Esto impide el acceso a clases nativas del motor gráfico, impidiendo el acceso a las librerías preestablecidas con las que normalmente se trabaja en Play. Esto dificulta enormemente el proceso de desarrollo a la hora de intentar realizar según qué tipo de acciones e imposibilita el acceso a ciertos datos, lanzando directamente un mensaje de error en la consola del motor. Por supuesto, según lo que se quiera desarrollar, esto resulta un claro impedimento a la hora de completar las tareas y es un aspecto importante a tener en cuenta a la hora de diseñar la aplicación.

Durante el desarrollo de la aplicación, se ha implantado a nivel de empresa el uso de la metodología GitFlow para el control de ramas y tareas. El proyecto principal de Venue Editor, alojado en Bitbucket, dispone de una rama master principal y una rama de desarrollo. A partir de ahí, mediante el uso de Jira, se crean diferentes ramas asociadas a tareas que deben pasar por un proceso de validación mediante pull request para que sean aprobadas por un supervisor. Trabajar en una herramienta de este tamaño con más de una persona, requiere mucho control sobre los cambios realizados y el estado actual de la aplicación para no sobrescribir aquello que ha hecho otro compañero de trabajo.

Por último, cabe destacar el duro aprendizaje realizado a la hora de indagar en la documentación de Newtonsoft para llevar a cabo un proceso de serialización/deserialización de datos correcto. La documentación de la librería es muy extensa y dispone de muchas herramientas para generar archivos JSON desde Unity pero, al estar diseñada en C#, no dispone de las libertades existentes en Javascript a la hora de asignar tipos a las propiedades o diferenciar entre objetos “null” y “undefinded”. El proceso de replicar la especificación de JSONs existente en la API web de Mobile Media Content ha sido más complejo de lo esperado. Algunos de los problemas encontrados han sido:

- Deserialización de clases: La API de Newtonsoft, a diferencia de System.Text.Json (Microsoft Corporation, s.f) (no incluido en las librerías de Unity) no ofrece alternativas por si sola a la hora de deserializar clases que heredan de otras. Un ejemplo claro de esto son los Node3D utilizados en los JSON de navegación. Dichos Node3D contienen la información básica de un nodo tridimensional que puede ser de tipo Sphere, Plane, Mesh, etc. A la hora de serializar podemos crear un NodeSphere y guardarlo sin problema, pero al deserializar no es capaz de inferir los tipos derivados de Node3D en una de las clases que heredan de ella. La única forma de realizar este proceso ha sido mediante la inclusión de una librería adicional, anexada a Newtonsoft, llamada JsonSubtypes (Counasse, s.f). Dicha librería permite especificar en la clase madre, los distintos subtipos que derivan de ella. Además, dispone de diferentes maneras de inferir los tipos según la estructura de datos de la que disponemos; por ejemplo, mediante un valor determinado como puede ser “type” presente en el JSON o una propiedad sólo existente en una de las subclases.

  Este proceso se puede realizar de forma análoga mediante el uso de custom JsonConverters (Newtonsoft, s.f) en Newtonsoft. Sin embargo, este método requiere de tener una misma propiedad en todas las subclases cuyo valor cambia en función del tipo. Siguiendo el ejemplo de los nodos 3D, un nodo de tipo esfera debería tener una propiedad “type=sphere”, mientras que un nodo de tipo Plane debería tener la misma propiedad con el valor “type=plane”. En la mayoría de JSONs generados a día de hoy con Venue Editor 3, los nodos 3D no contienen dicha propiedad, puesto que es el visor web el que se encarga de inferir el tipo según los

valores recibidos. Esto hace poco factible la detección de subtipos mediante los valores de dicha propiedad, dado que implicaría generar de nuevo todos los JSON forzando la propiedad type en todos los tipos de nodo.

- C# vs Javascript: Javascript es un lenguaje no tipado que puede ser mejorado mediante el uso de Typescript. Typescript se basa en reglas más o menos estrictas que requieren del compilador un esfuerzo adicional para tipar e inferir tipos de variables que en Javascript serían de tipo “any“. Aún así, Typescript se beneficia de muchas cosas de Javascript (puesto que no deja de ser el lenguaje madre). Las más importantes a nivel comparativo con C# son:

    - Null vs Undefined: Una de las principales características de Typescript es la diferenciación entre variables de tipo “null” y de tipo “undefined”. Aunque de primeras puede parecer lo mismo, para este lenguaje no lo es. Esto dificulta mucho la escritura de JSONs en C# teniendo que contemplar los “null” como undefined para omitirlos en la escritura. Lógicamente, este comportamiento de omitir un “null” es el equivalente a omitir variables de tipo “undefined”, lo que provoca dudas a la hora de escribir datos entre cuyos posibles valores pueda existir ese “null” y disponga de un significado.

      Es importante estar en comunicación con el departamento de IT dentro de ingeniería para contemplar estos casos de la API web y encontrar una forma de solucionar el problema desde Unity para evitar que se generen JSONs con valores erróneos o inesperados por parte del visor web.

    - Tipos múltiples: Otro gran problema para C# con respecto a Typescript para la generación de archivos JSON es la imposibilidad de tener variables de múltiples tipos más allá del uso de variables de tipo “dynamic”. En Typescript es posible definir una variable como un valor integro y al mismo tiempo que contemple la posibilidad de recibir una estructura de datos compleja. Por el momento, la única forma existente de resolver estos temas es mediante el uso de clases anidadas que contengan métodos para inferir los tipos de forma manual o el uso de variables de tipo “dynamic” que dificultan enormemente el proceso de deserialización, obligando al programador a determinar el tipo durante la ejecución.

7.1.2. Logro de objetivos

A nivel de objetivos, se han cumplido la mayoría de ellos con alguna salvedad. No se han podido cumplir los objetivos que hacen referencia al test de usuario de la aplicación, así como el objetivo secundario de implantar metodologías de test unitario.

Aunque sí se han podido probar todas las funcionalidades y están han estado supervisadas por el jefe del departamento, el estado actual de Venue Editor 4 no permite realizar una implantación a nivel

empresarial y de cliente todavía. Las herramientas de la versión 4 están prácticamente completas pero el desarrollo constante de nuevos elementos en la versión 3 hace que el proyecto aún no esté en fase de ser entregado hasta que no exista una versión estable equivalente entre ambas versiones. Así pues, no se ha podido probar directamente el proceso de implantación en el departamento de producción a nivel más senior, pero sí se ha planificado y se ha tenido en cuenta cómo se va a realizar dicha implantación una vez sea necesario.

Lógicamente, la imposibilidad de generar contenido real por parte del departamento de producción hace inaccesibles los contenidos por parte de los clientes, que siguen usando los contenidos generados con la versión 3 de la herramienta de editor. Aún así, existen herramientas internas en el visor web para previsualizar los datos generados y corroborar que los resultados son los esperados. Mediante el uso de un visor de demo y la subida de archivos a S3 en carpetas específicas para testing permiten acceder desde el navegador a mapas y panorámicas generadas durante la fase de desarrollo. Esta herramienta permite al departamento de ingeniería evaluar los resultados y decidir en qué momento el plugin de editor está listo para ser implantado.

Otro de los objetivos secundarios que ha sido difícil de probar es la eficacia y rapidez de la versión 4 con respecto a la versión anterior. Dado que la fase de desarrollo aún contiene fallos que se van puliendo a medida que se prueban las herramientas, es difícil saber con certeza si el rendimiento del nuevo editor es mayor a la versión previa. La fase de implantación en equipos de producción constituye un parte importante de este proceso, puesto que es en entornos reales donde se observan este tipo de problemas. Por otro lado, el control de errores también es difícil de evaluar hasta que no se trabaja en un entorno real, aunque sí es cierto que muchos de ellos pueden detectarse en fases más incipientes del proyecto.

7.1.3. Seguimiento de planificación

Por lo general, el desarrollo del proyecto ha seguido la planificación definida al inicio del trabajo. Tanto los hitos de desarrollo como de redacción de memoria se han cumplido sin problemas dentro de las fechas estimadas, a excepción de la fase de test de usuario y despliegue. Tal y como se ha comentado anteriormente, el generador de vistas panorámicas forma parte de una herramienta mucho más grande y que dispone que muchos elementos que aún no están listos para su correcta implantación. Así pues, la fase de testing y log de errores se ha llevado a cabo internamente dentro del departamento de ingeniería en base a ciertas premisas que se sabe que la herramienta debe cumplir. Más allá de eso, errores más tangibles por parte de usuarios finales dentro del departamento de producción no se han podido evaluar para depurar la herramienta al 100%.

A lo largo del trabajo se han tenido que implementar algunas funcionalidades extra, así como corregir algunos errores de la parte más core dentro del aplicativo general, Venue Editor. Dada la fase prematura en la que se encontraba la herramienta de gestión de “States” previamente al inicio del trabajo, se ha tenido que desarrollar una pequeña parte de todo eso para integrarlo correctamente en

la generación de vistas panorámicas. Además, el cambio de paradigma entre la versión 3 y la versión 4 de Venue Editor ha provocado la constante iteración sobre fallos inesperados en la parte central del aplicativo, que involucra procesos generales de carga de estadios y serialización de datos. Así pues, esto ha supuesto un pequeño retraso general en el desarrollo de la herramienta, cuyo desarrollo se paró durante un par de semanas hasta que se aseguró que el procedimiento completo de carga de recintos y asignación de “States” funcionaba de la forma esperada, al menos para todo aquello que involucra la herramienta de panorámicas.

En cuanto a la metodología, se puede concluir que, efectivamente, partir de Venue Editor 3 para la generación de la herramienta en Venue Editor 4 ha sido acertada. Dado que el output final del aplicativo cumple al 100% con la herramienta desarrollada en la versión 3, las premisas han estado claras desde el primer minuto; teniendo únicamente que centrarse en el desarrollo desde 0 de la herramienta incluyendo el nuevo workflow y adaptándose a la nueva arquitectura de la versión 4 de Venue Editor. Actualmente, las versiones de Venue Editor 3 y Venue Editor 4 se trabajan paralelamente puesto que el departamento de producción necesita poder usar la herramienta para generar contenidos que puedan usar los clientes. Así pues, se están implementando paralelamente las nuevas funcionalidades en ambas versiones, salvando las distancias en cuanto a arquitectura se refiere.

Por último, cabe destacar que, en el durante el proceso de creación de la herramienta, se ha incluido un apartado de generación y subida de panorámicas con tiles. Esto no se ha incluido en el trabajo por falta de tiempo, pero es un apartado interesante que amplía una parte del desarrollo para generar imágenes sobre las cuales se puede hacer zoom y que gozan de una mayor resolución que las panorámicas convencionales. Este añadido es bastante reciente, tanto en la versión 3 como en la API web, por lo que más cambios de este estilo pueden llevarse a cabo en los próximos meses.

7.2. Líneas de futuro

En cuanto a futuros desarrollos, existen varias líneas de trabajo explorables en cuanto al generador de vistas panorámicas y a Venu Editor 4. Como se ha comentado con anterioridad, Venue Editor 4 aún se encuentra en fase de desarrollo y no se ha implantado en equipos de producción. Una primera modificación importante será la migración a Unity 2023 en cuanto se publique una versión Alpha de la API. En la conferencia del roadmap hacia Unity 2023 se explicaron nuevas funcionalidades que pueden ser de gran utilidad de cara a herramientas de Editor que pueden o no mejorar el rendimiento de la aplicación. Además, se prevé la inclusión de elementos predefinidos de interfaz que puedan ayudar a la maquetación de mejores herramientas y que dispongan de más alternativas para los programadores a la hora de diseñar nuevas funcionalidades.

Por otro lado, dada la reciento evolución de Mobile Media Content, su ampliación de plantilla y sus nuevos proyectos, existe una gran posibilidad de que se deban añadir nuevas funcionalidades a

Venue Editor 4 y que se sigan incorporando mejoras y cambios en la generación de vistas panorámicas. Un claro ejemplo son las vistas panorámicas de navegación, que hasta hace poco no seguían ningún tipo de especificación ni validación de schema dentro de Unity que corrobore que el output obtenido es válido como input al visor web.

En cuanto al proceso de serialización, se está explorando la posibilidad de migrar al uso de protocol buffers (Google LLC, 2023). Los protocol buffers permiten serializar datos independientemente del lenguaje o la plataforma en la que se esté desarrollando. Son una forma más rápida y eficiente de serializar y deserializar estructuras de datos. Entre sus ventajas se encuentran:

- Almacenamiento de datos compacto.
- Rapidez a la hora de parsear.
- Disponibilidad de uso independiente al lenguaje de programación.
- Funcionalidades optimizadas a través de generación automática de clases.

El uso de protocol buffers solventaría gran parte de los problemas comentados en el apartado 7.1.1 sobre serialización de datos usando JSON y sus incompatibilidades entre Javascript y C#.

Por último, se está explorando la posibilidad de migrar Venue Editor a otros motores gráficos como Unreal Engine, cuyas funcionalidades en cuanto iluminación y condiciones de trabajo son actualmente superiores a Unity. La migración a Unreal Engine supondría una mejora sustancial en la calidad del producto final y ahorraría parte del trabajo del departamento de producción. Sin embargo, dada la poca experiencia de la que dispone el departamento de ingeniería trabajado con dicho motor dificulta mucho la tarea. Este cambio implica un cambio de paradigma importante con respecto al actual y supondría una inversión de horas en investigación por parte de ciertos trabajadores del departamento para evaluar la viabilidad del porte que la empresa no puede asumir actualmente. Esto proceso probablemente requiera de incorporar a un programador experto en herramientas para Unreal Engine que sea capaz de replicar el desarrollo actual en Unity y explicar al resto del departamento cómo se trabaja utilizando este otro motor, lo que supone ciertas horas de formación y un proceso de adaptación importante por parte de todo el departamento.

Bibliografía

Atlassian. (s.f). *Gitflow Workflow*. Recuperado el 16 de 03 de 2023, de https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Atlassian. (s.f). *Jira Software*. Recuperado el 16 de 03 de 2023, de https://www.atlassian.com/software/jira

Atlético de Madrid. (s.f). *Atlético de Madrid: Compra de entradas*. Recuperado el 27 de 03 de 2023, de https://www.atleticodemadrid.com/listado-de-entradas

Ballena. (s.f). *Ballena: Seats 3D*. Recuperado el 03 de 03 de 2023, de https://www.seats3d.com/corp/

Broadway Shows. (s.f). *Lyric Theatre*. Recuperado el 27 de 03 de 2023, de https://broadway.harrypottertheplay.com/ticket-information/

Counasse, E. (s.f). *JsonSubTypes*. Recuperado el 31 de 05 de 2023, de https://www.nuget.org/packages/JsonSubTypes/

Google LLC. (2023). *Protocol Buffers Documentation*. Recuperado el 06 de 06 de 2023, de https://protobuf.dev/overview/

IOMedia. (s.f). *IOMedia*. Recuperado el 03 de 03 de 2023, de http://www.io-media.com

Major League Baseball. (s.f). *Chicago Cubs: Compra de Entradas*. Recuperado el 27 de 03 de 2023, de https://www.mlb.com/cubs/tickets/group-tickets/book

Manchester City. (s.f). *Manchester City: Compra de entradas*. Recuperado el 27 de 03 de 2023, de https://tickets.mancity.com/

Microsoft Corporation. (s.f). *System.Text.Json Namespace*. Recuperado el 31 de 05 de 2023, de https://learn.microsoft.com/en-us/dotnet/api/system.text.json?view=net-7.0

Microsoft Corporation. (s.f). *System.Threading CancellationTokenSource*. Recuperado el 30 de 05 de 2023, de https://learn.microsoft.com/en-us/dotnet/api/system.threading.cancellationtokensource?view=net-7.0

Microsoft Corporation. (s.f.). *.NET API Browser*. Recuperado el 12 de 05 de 2023, de https://learn.microsoft.com/en-us/dotnet/api/?view=net-7.0

Microsoft Corporation. (s.f.). *System.IO Namespace*. Recuperado el 12 de 05 de 2023, de https://learn.microsoft.com/en-us/dotnet/api/system.io?view=net-7.0

Microsoft Corporation. (s.f.). *System.Linq Namespace*. Recuperado el 12 de 05 de 2023, de https://learn.microsoft.com/en-us/dotnet/api/system.linq?view=net-7.0

Microsoft Corporation. (s.f.). *System.Threading Namespace*. Recuperado el 12 de 05 de 2023, de https://learn.microsoft.com/en-us/dotnet/api/system.threading?view=net-7.0

Mobile Media Content. (s.f). *DVM - Digital Venue Manager*. Recuperado el 03 de 03 de 2023, de https://docs.3ddvapis.com/js/dvm/

Mobile Media Content. (s.f). *Mobile Media Content*. Recuperado el 15 de 03 de 2023, de https://www.mobilemediacontent.com/

Newtonsoft. (2023). *Newtonsoft Json.NET*. Recuperado el 12 de 05 de 2023, de https://www.newtonsoft.com/json

Newtonsoft. (s.f). *Custom JsonConverter*. Recuperado el 31 de 05 de 2023, de https://www.newtonsoft.com/json/help/html/CustomJsonConverter.htm

Pacifa Decision. (s.f). *Pacifa*. Recuperado el 03 de 03 de 2023, de http://www.pacifa-decision.com/

Paciolan. (s.f). *Paciolan*. Recuperado el 03 de 03 de 2023, de https://www.paciolan.com/products/commerce

Platinum Group. (s.f). *Platinum Group: Global Ticketing Technology*. Recuperado el 03 de 03 de 2023, de https://www.pg-mc.com/

Real Madrid. (s.f). *Real Madrid Baloncesto*. Recuperado el 27 de 03 de 2023, de https://www.realmadrid.com/baloncesto

Ticketmaster. (s.f). *Ticketmaster*. Recuperado el 03 de 03 de 2023, de https://www.ticketmaster.es/

Unity Technologies. (28 de 04 de 2023). *Script Reference: EditorGUILayout*. Recuperado el 02 de 05 de 2023, de https://docs.unity3d.com/ScriptReference/EditorGUILayout.html

Unity Technologies. (28 de 04 de 2023). *Scripting Reference: GUILayout*. Recuperado el 02 de 05 de 2023, de https://docs.unity3d.com/ScriptReference/GUILayout.html

Unity Technologies. (31 de 03 de 2023). *Unity Documentation: Extending the Editor*. Recuperado el 15 de 03 de 2023, de https://docs.unity3d.com/Manual/ExtendingTheEditor.html

Unity Technologies. (25 de 05 de 2023). *Unity Documentation: Project Settings*. Recuperado el 31 de 05 de 2023, de https://docs.unity3d.com/2022.2/Documentation/Manual/comp-ManagerGroup.html

Unity Technologies. (25 de 05 de 2023). *Unity Documentation: Quality*. Recuperado el 31 de 05 de 2023, de https://docs.unity3d.com/2022.2/Documentation/Manual/class-QualitySettings.html

Unity Technologies. (05 de 05 de 2023). *Unity Scripting Reference*. Recuperado el 12 de 05 de 2023, de https://docs.unity3d.com/ScriptReference/

Unity Technologies. (s.f). *Unity*. Recuperado el 15 de 03 de 2023, de https://unity.com/es

Unity Technologies. (s.f). *Unity Script Reference, Camera.RenderToCubemap*. Recuperado el 22 de 05 de 2023, de https://docs.unity3d.com/ScriptReference/Camera.RenderToCubemap.html

Unity Technologies. (s.f). *Unity Script Reference, RenderTexture.ConvertToEquirect*. Recuperado el 22 de 05 de 2023, de https://docs.unity3d.com/ScriptReference/RenderTexture.ConvertToEquirect.html

Anexos

Anexo A: Currículum Vitae

CONTACT

Address
Avinguda de la Catalana, 178,
08930, Sant Adrià de Besós

Phone
+34 608 15 06 71

Web
albertogarciaruiz96@gmail.com
https://es.linkedin.com/in/alberto-garcia-ruiz
https://github.com/agarciaruiz

SKILLS

Software Development

Unity expertise

Teamwork and Collaboration

Critical thinking

Technical writing

TECHNICAL PROFILE

Unity
.NET/C#
C++

Blender
Python

Typescript
Angular
HTML

Git
Atlassian
AWS

Alberto García Ruiz

Software Developer

PROFILE

Complex problem-solver with analytical and driven mindset. Dedicated to achieving demanding development objectives according to tight schedules while producing impeccable code. Developer with experience as Project Manager and Graphics' Software Development . Excellent math and programming skills.

I consider myself a proactive person ready to keep learning and committed to everything I do and to those around me. I stand out for my capacity to solve problems, manage tasks, time and resources and so do I for my creativity and adaptability to different environments and working groups.

EXPERIENCE

2021-Present
Mobile Media Content I C. d'Àlaba, 61, 08005 Barcelona

SOFTWARE DEVELOPER

- Tools development to match company requirements.
- Unity programming. Use of Unity Editor to create new useful features for the Production team. Code architecture and R&D for standalone applications.
- Blender addons programming and maintenance for the Production team.
- Use of 3D maths and complex algorithms to solve relevant issues.
- Team leading and junior profiles follow up.

2019 - 2021
Lightsound Business S.L I C. dels Almogàvers, 118, 08018 Barcelona

PROJECT MANAGER

- Audiovisual project management
- Project design, plan elaboration using AutoCad and Revit, and technician working teams leading
- Schedule recurrent client installations, organize internal timings and budget elaboration.

2018 - 2019
MC Ingenieros I C. de Llull, 47, 08005 Barcelona

AUDIOVISUAL ENGINEER

- Pre-sales audiovisual engineering
- Project design, plan elaboration and home automation programming
- Requirements evaluation, audio and video connections diagrams performing, DSP and Crestron programming.

LANGUAGES

Spanish

English

Catalan

INTERESTS

EDUCATION

2021-Present
Universitat Oberta de Catalunya

VIDEO GAMES DESIGN AND PROGRAMMING

Advanced level of video games programming and design. Extensive knowledge of C# gameplay programming and low level graphics programming in C++. Specialization in graphics programming branch.

2014-2018
Universitat Pompeu Fabra

AUDIOVISUALS SYSTEMS ENGINEERING

Telecommunications engineering specialized in image and sound. Technologies related to sound and image processing, design of audio, video and multimedia systems. Math and programming related to audiovisual environments , computer graphics and digital signal processing.