UNIVERSITAT DE BARCELONA

Trabajo final de grado

GRADO DE INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática
Universidad de Barcelona

Ampliación del juego serio Geopieces para el aprendizaje de la geometría en 3D

Autor: Leandro Iván Zardaín Rodríguez

Director:	Dra. Anna Puig
Realizado al:	Departamento de Matemáticas e Informática

Barcelona, 3 de febrero de 2017

Abstract

The main goal of this project is to implement a 3D video game where primary school students would learn and improve their knowledge about geometric figures and their locations in the 3D space.

Specifically, it goes in depth in the Geopieces game modes, creating new diverse educational activities from a new type of game that will be created. Its main objective is to join 3D figures, which were obtained in the previous activities, in order to build a new 3D solid figure.

Another important point is to connect Geopieces games with the server that had been developed initially for the Fracsland game. Therefore, teachers would be able to create new type of missions, edit their learning competences and their helps. In addition, they would have the possibility to customize and to follow the progress of each student.

Resumen

El objetivo principal de este Trabajo Final de Grado es crear un juego 3D donde los estudiantes de primaria aprendan y refuercen sus conocimientos sobre las figuras geométricas y su relación con el espacio 3D.

En particular, se trata de profundizar y ampliar el juego Geopieces, desarrollado en dos trabajos finales de grado anteriores, añadiendo nuevas actividades pedagógicas a partir de un nuevo nivel que se agregará, en el cual las piezas 3D obtenidas en las actividades anteriores se podrán acoplar para crear nuevas figuras 3D más complejas.

Otro de los puntos importantes desarrollado en este Trabajo Final de Grado es la conexión del juego final de Geopieces con el servidor que se desarrolló en un Trabajo Final de Grado anterior para el juego de Fracsland, de forma que el profesorado pueda crear nuevas misiones, editar las competencias y las ayudas de los juegos del Geopieces así como podrá personalizar y seguir el progreso que van adquiriendo cada alumno a medida que van jugando.

Agradecimientos

Quiero agradecer especialmente a mi tutora Anna Puig por toda la ayuda y dedicación recibida durante todo el curso.

Me gustaría agradecer a mis compañeros Sergi Cebrián y Cristian Muriel por los proyectos que realizaron y toda la ayuda que me han dado para poder continuarlos.

También destacar la ayuda recibida de Inmaculada Rodriguez por sus consejos de usabilidad sobre el juego.

Y por último, agradecer el soporte dado por mi familia.

Tabla de contenidos

1. Introducción

El fracaso escolar es un gran problema de actualidad que repercute directamente sobre la sociedad. Se habla de fracaso escolar cuando un niño no reúne el nivel esperado para su edad y nivel pedagógico [4].

El abandono escolar y el fracaso escolar están altamente relacionados, se puede observar en el informe de la Obra Social La Caixa [5] que cuanto mayor es el número de alumnos que no se presentan a las asignaturas, más probabilidad tienen de acabar dejando los estudios.

Según el INE [6], España es uno de los países de la Unión Europea con mayor tasa de abandono escolar. Una de las asignaturas donde hay mayor de fracaso y abandono escolar corresponde a Matemáticas.

Los expertos dicen que el problema fundamental se centra en el sistema educativo, en el cual sugieren incorporar otros mecanismos para evaluar al alumnado en vez de solo utilizar las calificaciones que obtengan a través de exámenes.

El sistema educativo actual es antiguo: las habilidades adquiridas por los estudiantes son las equivocadas, en vez de estudiar para adquirir conocimiento, están estudiando para aprobar.

Otras de los puntos claves es que los estudiantes están bastante desmotivados.

Debido a este sistema educativo el alumnado ve las Matemáticas como una de las asignaturas menos atractivas, por lo que habría que buscar otras formas para hacer llegar el conocimiento a los estudiantes.

Mediante este proyecto se pretenderá reforzar los conocimientos matemáticos, en particular sobre geometría, principalmente para los alumnos del Ciclo Superior de Primaria y Ciclo Inicial de Secundaria.

Se estudiarán diversas competencias sobre geometría, como la identificación, composición y rotación de figuras geométricas en el espacio 3D, con las que podrá trabajar el alumnado y se implementarán en un entorno 3D con el objetivo de que el estudiante se divierta a medida que va aprendiendo.

También se permitirá al profesor poder ajustar las competencias que quiera que sus estudiantes trabajen para esa clase, dándole la oportunidad de personalizarlas incluso para cada estudiante. De esta forma se podrá adaptar el trabajo que deberán realizar los estudiantes de una manera individualizada por si se necesitase que unos reforzasen más otras competencias que otros.

Por último, el profesorado podrá además observar la evolución de cada estudiante a medida que estos vayan jugando.

1.1. Ámbito del proyecto

Este proyecto surge como una necesidad en las aulas para reforzar el aprendizaje donde se ha de trabajar con más profundidad las actividades experimentales-manipulativas mediante el desarrollo de un juego serio [7] [8] [9], el cual además de tener como objetivo el entretenimiento y la diversión, tenga también un objetivo más didáctico y educativo debido a que los alumnos que están más motivados aprenden mejor. Utilizar juegos es una buena forma de motivarlos.

Lo que se busca es crear un juego que sirva como ayuda adicional para respaldar los conocimientos que se hayan adquirido previamente en clase. Dentro del plan de estudios del grado de Ingeniería Informática, hay un gran número de asignaturas de las cuales nos pueden servir para implementar el juego y el servidor.

Para iniciarnos en la gestión de eventos, la programación de objetos y la arquitectura MVC (Modelo-Vista-Controlador) tenemos la asignatura de Programación 2 y Diseño de Software.

Para mejorar las interfaces con la que tanto el juego como el servidor se comunicarán y darán feedback al usuario, está la asignatura de Factores Humanos y Computación.

Para aprender sobre el motor de juego Unity (el cual se usará para el juego) destacamos la asignatura de Ingeniería de Sotfware ya que en esta asignatura se ha tenido que implementar también un juego complejo en Unity.

Para implementar un servidor básico basándose en el Framework Django, nos basaremos en los conocimientos de la asignatura de Software Distribuido.

Para entender la física con la cual los elementos operan en un entorno 3D, viendo como son afectados cuando una luz aparece en la escena y también para entender cómo cambiar entre diferentes tipos de sistemas de coordenadas, nos basaremos en los conocimientos impartidos en la asignatura de Gráficos y Visualización de Datos.

Se usarán los siguientes lenguajes de programación:

- HTML y Javascript: utilizados en las asignaturas de Factores Humanos y Computación y Software Distribuido.
- Python: impartido en las asignaturas de Algorítmica y Algorítmica Avanzada.
- C#: basándonos en C++ aprendido en Gráficos y Visualización de Datos.

También sería importante remarcar dos asignaturas del plan de estudios del grado de Matemáticas que han sido útiles a la hora de implementar el algoritmo principal en el que se basará el juego: Grafos y Topología 2.

1.2. Motivación

Este trabajo surge a partir de la necesidad que tienen los estudiantes para reforzar sus competencias en geometría en el Ciclo Superior de Primaria y en el Ciclo Inicial de Secundaria, ya sea tanto para trabajar dentro de clase como fuera de ella, donde las competencias que principalmente se tratan consisten en la identificación, construcción y composición de figuras geométricas 3D y sus proyecciones en el plano.

Además, otro de los objetivos que se busca es crear un servidor en el cual se desarrollará una serie de herramientas de *authoring* de modo que se le dará libertad al profesor para diseñar el juego, eligiendo que competencias deben sus alumnos trabajar y pudiendo personalizarlo individualmente para cada estudiante.

Por consiguiente, con este trabajo se pretende construir un juego divertido, estimulante que consista en un entorno 3D en el cual el alumno podrá trabajar las diversas competencias que el profesor, de una forma sencilla, flexible y guiada, pueda diseñar desde un servidor. Además, el juego debe ser compatible tanto para los ordenadores de los centros escolares como para tablet.

1.3. Objetivos generales

Uno de los objetivos principales de este Trabajo Final de Grado es ampliar el juego Geopieces, juego desarrollado en dos proyectos Finales de Grado del curso pasado [1] [2], de forma que se le añada el nivel 3D, en el cual se tratarán diversas competencias sobre figuras geométricas en 3D y además estas competencias serán lo suficientemente adaptables para poder ser gestionadas por el servidor.

El segundo objetivo principal es el de ampliar el servidor FracServer, desarrollado en un Trabajo Final de Grado previo [3], para que el profesorado pueda gestionar también el juego de Geopieces.

1.4. Objetivos específicos

Los objetivos generales se pueden desglosar en los siguientes objetivos más específicos, siguiendo el orden en el que se prevé que se implementarán (ver figura 1):

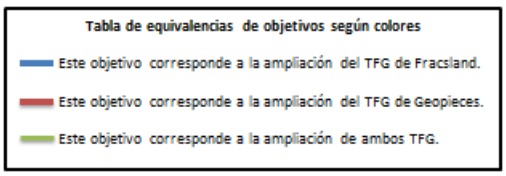

Figura 1: Tabla de equivalencias de objetivos de este proyecto con las ampliaciones mencionadas en objetivos de otros proyectos.

1. **Diseño y desarrollo de la estructura básica del juego de Geopieces en 3D:**

   (a) **Diseño del escenario.**

   (b) **Diseño de la dinámica del juego (poder desplazar, rotar objetos predefinidos como cubos).**

   (c) **Diseño de la interfaz que comunicará el juego con el usuario. Diseño adaptable de la interfaz según la plataforma (PC, tablet).**

   (d) **Diseño de los objetos básicos del juego junto con todas sus propiedades.**

2. **Diseño integro de un juego de construcción de objetos 3D: Juego consistente en construir figuras 3D complejas a partir de los poliedros elementales conseguidos al juego 2D-3D.**

   (a) **Desarrollo del mecanismo que permita comprobar cuando dos caras estén bien encaradas, esto es que ambas caras sean coplanarias y tengan la misma rotación.**

   (b) **Desarrollo del isomorfismo entre el objeto complejo final y el que va construyendo el usuario.**

   (c) **Desarrollo de la rotación y comprobación de si se trata una traslación entre el objeto complejo final y el que va creando el usuario.**

3. **Diseño y desarrollo del servidor de forma que permita la creación de nuevas campañas:**

   (a) **Diseño y desarrollo de campañas que permitan escoger las competencias a reforzar para el alumno.**

   (b) **Diseño y desarrollo de campañas creadas por defecto.**

   (c) **Diseño y desarrollo de sistemas para añadir nuevas campañas personalizadas para el profesor.**

4. **Desarrollo de un sistema de conexión con servidor externo para:**

   (a) **Gestión de usuarios. Dar un software que permita gestionar los usuarios creados junto con sus evaluaciones y guardar sus avances.**

   (b) **Gestión de campañas. Dar un software que permita gestionar las campañas creadas y las nuevas campañas.**

1.5. Planificación temporal

Se utilizará el diagrama de Gantt para representar la planificación temporal de este proyecto (ver figura 2).

El tiempo estimado es de unas 24 semanas. Como el TFG se acordó por Junio, ya se habló de empezar a trabajar en el proyecto a partir de a mediados de Agosto de 2016 hasta el 26 de Enero de 2017. Se han empleado por semana unas 16h aproximadamente.

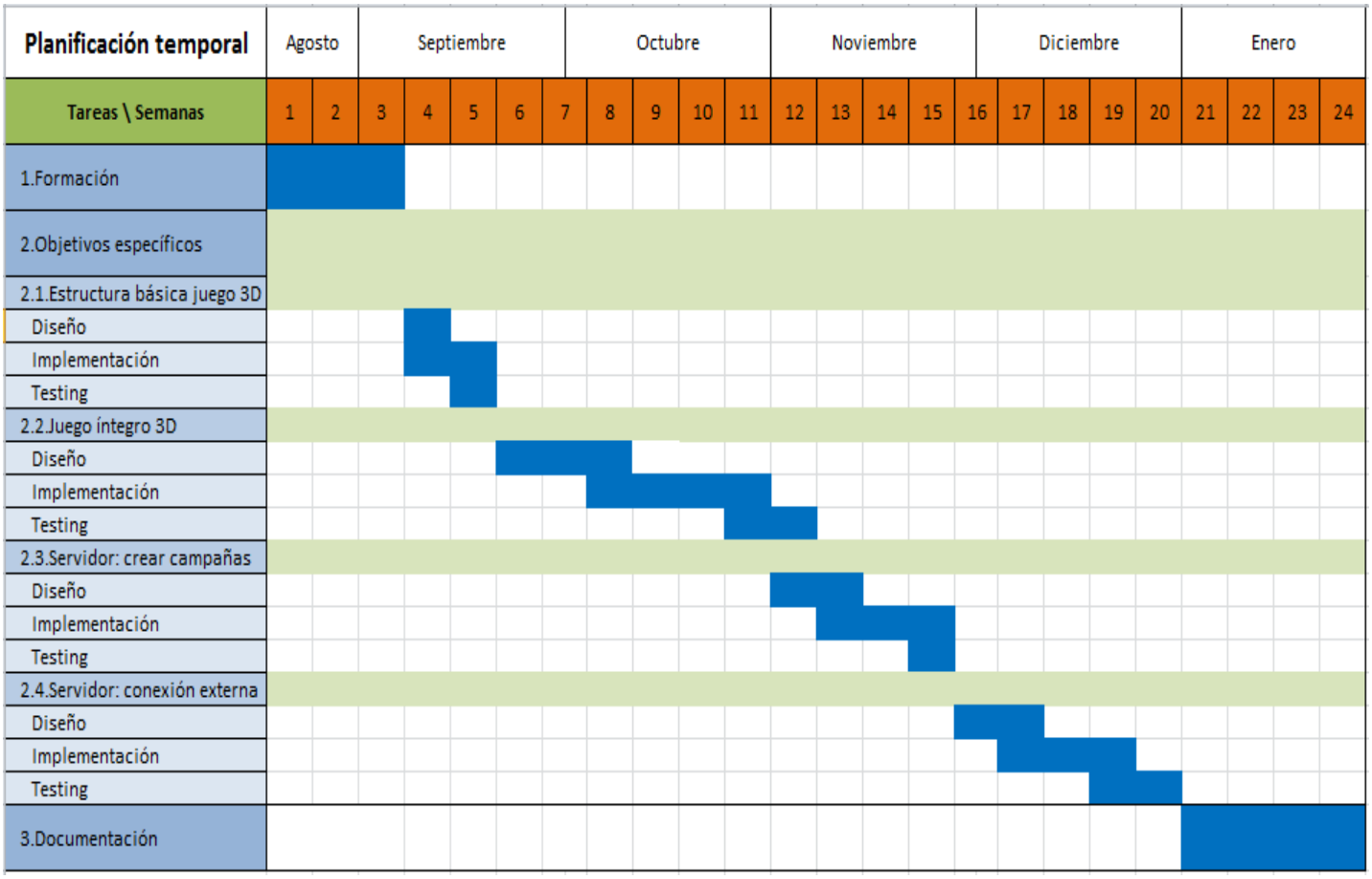

Figura 2: Planificación temporal inicial del proyecto.

Como se puede observar en el diagrama, se ha añadido un tiempo de formación que correspondería al aprendizaje sobre el motor de juego Unity.

Luego, para cada uno de los objetivos, hemos destacado 3 partes: diseño, implementación y testing. Finalmente se ha añadido un tiempo para la documentación.

1.6. Organización de la memoria

La memoria está organizada de la forma siguiente:

- Primer capítulo: Introducción, en la cual se hace una breve presentación del proyecto, mencionando cuales son los objetivos que se quieren alcanzar y cómo se han planificado.
- Segundo capítulo: Trabajos relacionados. Se estudia la teoría docente que hay detrás del aprendizaje de la geometría en 3D. Se verá algunos trabajos relacionados con este proyecto y se analizará el tema de aprendizaje del jugado, creando varios tipos de niveles y también sobre las herramientas de *authoring* que podrá usar el profesorado.
- Tercer capítulo: Análisis y diseño del juego 3D. Se hará un análisis del juego, que consiste en ver cuáles son los objetivos de aprendizaje, las competencias que se tratarán y trabajarán y además, cómo se ha diseñado el juego.
- Cuarto capítulo: Desarrollo del juego 3D. Se estudiará primero la teoría, desde un punto de vista bastante formal, sobre cómo poder resolver uno de los algoritmos más importantes de este proyecto.
- Quinto capítulo: Resultados y simulaciones. Aquí se explicarán los resultados que se han obtenido y se mostrarán algunas simulaciones del proyecto.
- Sexto capítulo: Valoración económica. En este apartado se reflexionará sobre el coste del propio trabajo que se ha realizado.
- Séptimo capítulo: Conclusión. Se recordarán los objetivos del proyecto y se verá cuáles de ellos se han podido cumplir y finalizar. También se hablará de las posibles líneas de continuación que podría tener este proyecto.

Por último, se han añadido los apéndices, en los cuales se explicará la instalación de programas y los manuales técnicos tanto para el estudiante como para el profesorado.

2. Trabajos relacionados

2.1. Teoría docente

En este apartado se procederá a estudiar diversas teorías docentes sobre el aprendizaje de los estudiantes en la geometría [10].

2.2. Teoría del desarrollo de conceptos espaciales de Piaget

Piaget, después de realizar varios experimentos, propuso una teoría sobre el desarrollo de los conceptos espaciales que aprender los niños de pequeños [11]. Piaget en su teoría define el conocimiento como el "conocimiento de objetos resultante del contacto director entre ellos" y representación como que "comporta la evocación de objetos en ausencia de ellos".

Las capacidades de percepción de un niño se desarrollan hasta los dos años, que es cuando se empieza a profundizar la capacidad de reconstrucción de imágenes espaciales y que es perfeccionada hacia los siete años.

También define dos tipos de pruebas. Una de ellas es el test de percepción que se basa en la capacidad de distinguir objetos que se han presentado visualmente. Mientras que la otra prueba se le llama test de representación que consiste en la identificación de formas con el tacto y la forma de poder reproducir mediante otros objetos.

En cada una de las etapas del desarrollo del niño, Piaget distingue un grupo de propiedades geométricas diferentes que va adquiriendo:

1. Propiedades topológicas: Son las propiedades globales de los objetos sin tener en cuenta la forma o el tamaño que tengan.

2. Propiedades proyectivas: Son las propiedades que tienen que ver con la capacidad de determinar un objeto 3D a partir de ser visto desde diversos ángulos.

3. Propiedades euclidianas: Son las propiedades que tienen que ver con los tamaños, distancias y direcciones que por tanto se trata sobre la capacidad de determinar longitudes, áreas y ángulos, entro otros.

2.2.1. Modelo de Van Hiele

Se trata de una teoría didáctica de los años 50 que desarrolló el matrimonio Dina van Hiele-Geldof y Pierre van Hiele tras ver que sus alumnos cometían siempre los mismos errores. De ello pudieron definir una serie de niveles del pensamiento con el cual diseñaron un modelo por el que describían la evolución del aprendizaje de la geometría por medio de estos niveles [12].

Los niveles del pensamiento son los siguientes:

- Nivel 1: De reconocimiento. Este es el nivel más básico para llegar al razonamiento y correspondería a los alumnos de los primeros años de primaria que solamente pueden reconocer las figuras geométricas por su forma y aspecto físico, teniendo una percepción más global con lo que no podrían distinguir ni partes ni elementos de las figuras.

- Nivel 2: De análisis. Los alumnos empiezan a poder analizar las figuras y por ello, a comenzar a deducir propiedades matemáticas sobre las figuras aunque sea solo por experimentación. En este nivel es cuando surge el pensamiento matemático. Corresponderían a los estudiantes del Ciclo Superior de primaria o primeros años de la secundaria.

- Nivel 3: De clasificación. Los estudiantes ahora son capaces de relacionar propiedades entre sí, comprendiendo que una propiedad se puede deducir de otras y por tanto, de determinar una figura geométrica a partir de una definición matemática. Es típico de los alumnos que estén en los últimos años de secundaria obligatoria.

- Nivel 4: De deducción formal. Aquí ya pueden realizar razonamientos lógicos formales y demostraciones, entendiendo que son necesarias para verificar las propiedades. Correspondería a los alumnos del Bachillerato científico o de principios de universidad.

- Nivel 5: Rigor. Los alumnos pueden trabajar sin problemas en diferentes sistemas axiomáticos. Incluso podrán llegar a aceptar una demostración que vaya en contra de la intuición y del sentido común si el argumento de la demostración es válido. Este nivel lo alcanzan prácticamente solo los alumnos universitarios.

Para alcanzar un nivel superior, en el cual ya se da por sentado todo lo apareció en los niveles anteriores, el matrimonio desarrolló una serie de fases para ello:

- Fase de información: En esta primera fase, el profesor explica sobre lo que se trabajará y los alumnos deberán adquirir algunos conocimientos básicos.

- Fase de orientación dirigida: Al alumno se le dará una serie de actividades dirigidas más bien al aprendizaje, descubrimiento de las propiedades fundamentales. Se han de presentar las actividades gradual y progresivamente.

- Fase de explicitación: Se revisa lo que el estudiante haya estado trabajando en las fases anteriores de forma que se refuerzan las ideas y conclusiones que se hayan sacado.

- Fase de orientación libre: Ahora han de aplicar el conocimiento que han adquirido últimamente a nuevas situaciones por tal de perfeccionar y poner en práctica lo que el alumno haya aprendido.

- Fase de integración: El profesor ha de resumir los conocimientos y formas de razonamiento que aprendió los alumnos por tal de darle una visión más general del asunto con lo que finalmente, el alumno ya estaría listo para prepararse para la siguiente fase.

Este proyecto está destinado a los alumnos que se encuentren en el nivel 2 del pensamiento. Las fases en las que se centrará este proyecto son las fases 1 y 2: Se explicará al alumno en qué consiste la tarea a realizar y se le propondrá actividades para que aprenda propiedades fundamentales de la geometría.

2.3. Análisis sobre las aplicaciones y juegos relacionados

En este capítulo se estudiarán otras aplicaciones o juegos que están relacionados con este proyecto, es decir, que tengan que ver con el aprendizaje sobre la geometría en 3D:

De los juegos explorados, se pueden destacar:

- **WisWeb** es un applet que contienen una lista variada de applets cuyas competencias se centran principalmente en el aprendizaje de la geometría [13].
- **Building with blocks** es un applet que contiene un conjunto de juegos basados sobretodo en la identificación y rotación de objetos en 3D [14].
- La web de mundo primaria [15] contiene una gran cantidad de juegos sobre matemáticas y en especial sobre geometría, destinado a estudiantes de primaria.
- **The land of Venn** es un juego disponible para la App Store que consiste en un juego del estilo Tower Defense sobre geometría [16].

También hay una aplicación que aunque no sea un juego, se utiliza mucho en los institutos y universidades y se llama **GeoGebra** [17]. Es un software libre, interactivo y que básicamente se trata de un procesador geométrico y algebraico.

Todos ellos son juegos que motivan el aprendizaje de geometría pero son difícilmente configurables por el profesor y no se pueden personalizar a los niveles de aprendizaje de distintos perfiles de estudiantes.

2.4. Análisis sobre las herramientas de *authoring* en los juegos

Utilizar herramientas de *authoring* para que un profesor pueda de alguna forma adaptar los componentes de un juego y monitorizar las actividades de sus estudiantes es un tema bastante nuevo por lo que no hay muchos juegos que lo implementen y sean educativos. Existen algunas herramientas que permiten realizar juegos de forma fácil, como **StoryTec** [19], **e-Adventure** [20], **ITyStudio** [21], pero están muy

dirigidas a la creación propia de juegos o a juegos de tipo pregunta-respuesta, pero no se centran en la elección de distintas competencias ni objetivos de aprendizaje.

Esta es una de las razones de la propuesta de este proyecto, para crear un juego serio utilizando herramientas de *authoring*.

De los juegos que se ha visto que utilizaban herramientas de *authoring*, destacamos uno muy completo llamado **what2learn** el cual contiene una gran cantidad de juegos [22]. Además, se registran automáticamente los avances de los usuarios por lo que es una herramienta muy útil para el profesorado.

3. Diseño del juego 3D

3.1. Idea principal

La idea inicial de este juego corresponde con la que se introduce en los proyectos anteriores en los que se basa el actual proyecto: aprender y reforzar los conocimientos de los estudiantes sobre geometría y monitorizar su progreso para los profesores.

El juego está pensado para que se pueda jugar ya sea desde el ordenador como con la tablet.

El objetivo del juego Geopieces, en general, era construir un poblado en el cual se podrían ir construyendo diferentes estructuras creadas a partir de figuras geométricas regulares (ver figura 3).

Figura 3: Menú de las campañas del juego Geopieces.

Para crear una estructura en concreto se necesita realizar su 'campaña' para poder llegar a generar el objeto en cuestión. Una campaña consiste en el proceso de pasar por todos los modos de juegos empezando por el más bajo, por tal de acabar creando un objeto sólido final.

Primero, el usuario ha de empezar en el nivel 2D (el más bajo) recolectando figuras geométricas 2D, en el cual le salen misiones variadas según el nivel y las competencias que se traten (ver figura 4).

Una vez se han tenido las figuras suficientes, se procede al modo 2D-3D que, en este caso, se podría decir que hay una única competencia: Crear el desdoblamiento adecuado para poder generar un objeto simple 3D (ver figura 5).

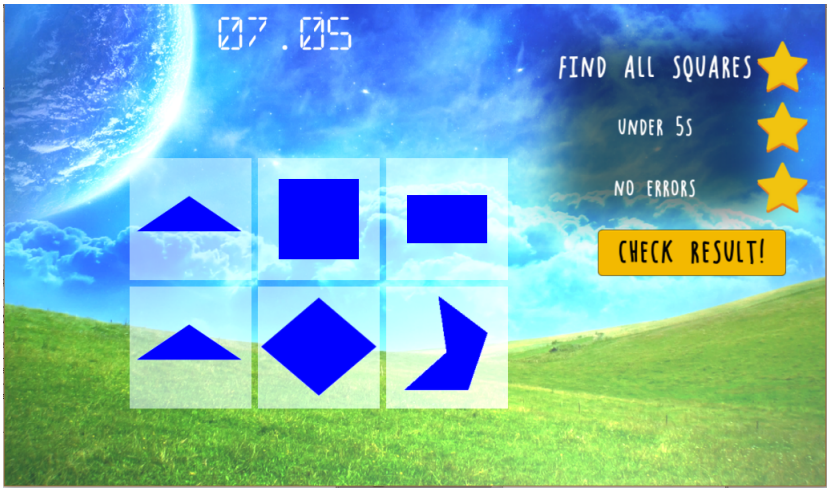

Figura 4: Actividad del modo 2D: seleccionar todas las figuras que correspondan a un cuadrado.

Figura 5: Actividad de desdoblamiento: a partir de figuras 2D se crean figuras 3D simples.

Se realizan las actividades anteriores cuantas veces sean necesarias hasta tener todos los objetos simples 3D generados. Es entonces cuando se pasa al último modo que consiste en la construcción en modo 3D, cuyo objetivo dotar al estudiante de un entorno 3D en el que podrá ir uniendo los objetos simples conseguidos en los pasos previos para acabar creando el objeto final del poblado.

Este proyecto se centra especialmente en el modo 3D.

Existen dos tipos de poblados a construir. En el primero, los objetos 3D a construir están pre-establecidos y el jugador tiene la misión de reconstruirlos. En el segundo, el jugador es libre de construir edificios, a modo de entrenamiento, mediante el cual el alumno podría ir reforzando sus conocimientos.

El primer tipo de poblado puede ser diseñado por el profesor desde un servidor. Esto es, que el profesorado puede decidir qué competencias y qué estructuras querer

proponer al jugador y personalizarlo, si hiciese falta, para cada alumno.

En este caso, también lo que se pretende es que el profesor pueda monitorizar y seguir el progreso que los estudiantes hayan adquirido y ver las faltas que hayan cometido.

3.2. Game Design Canvas Framework

Crear un juego que combine entretenimiento con el hecho de que sea educativo y además suponga un reto para el estudiante es una tarea bastante ardua. Por ello se utilizará un Framework llamado Game Design Canvas (GDC) que permitirá diseña el juego, teniendo en cuenta distintos aspectos del juego y siguiendo una serie de conceptos que proporciona [23].

Para poder desarrollar el Game Design Canvas del juego 3D se ha partido de los GDC que estaban creados para el modo 2D y 2D-3D, ya que tienen bastantes puntos en común y están basados en la misma idea general (ver figura 6).

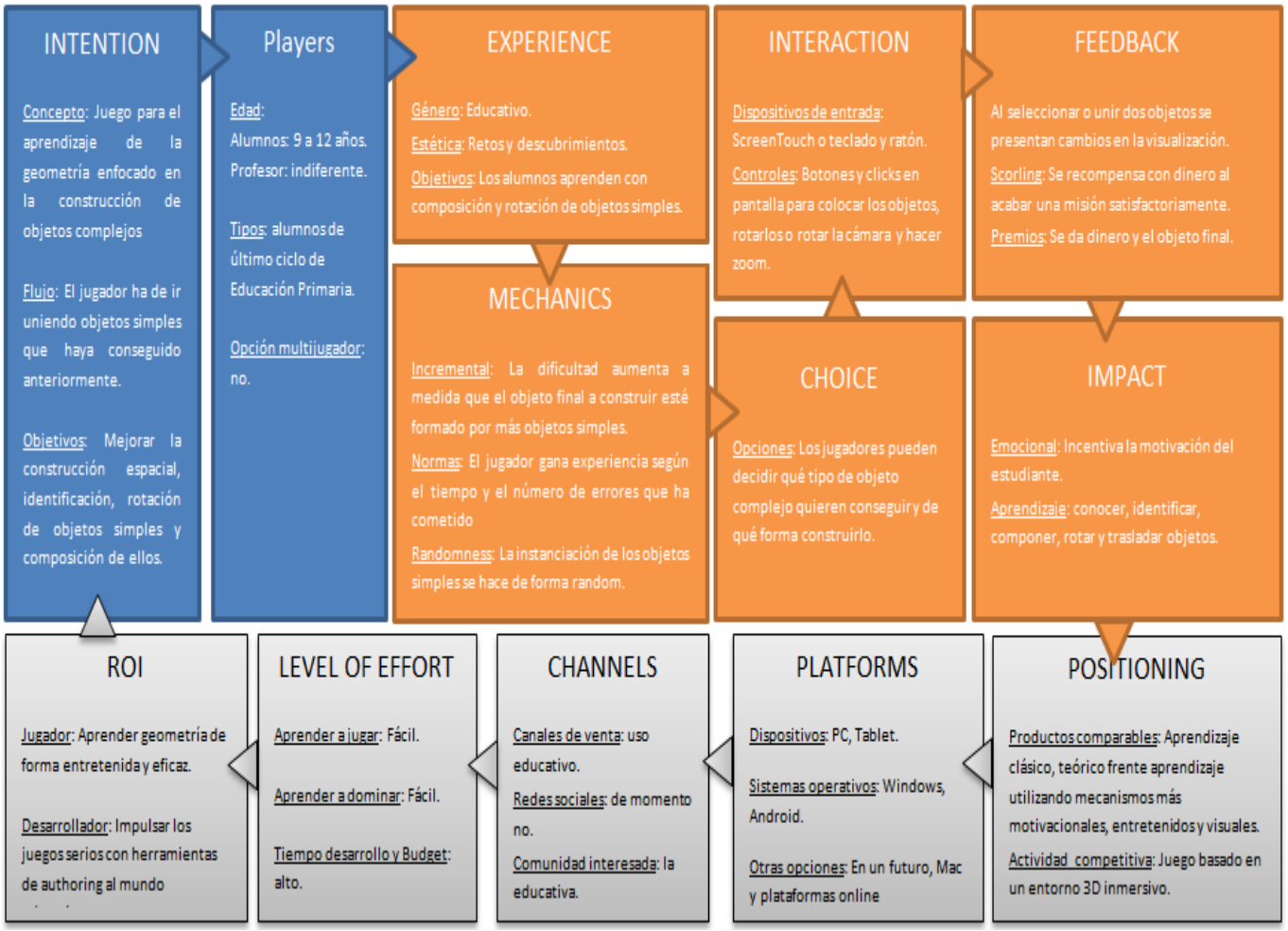

Figura 6: Game Design Canvas. En azul se muestran los aspectos de Setup, en naranja los que afectan al Game Design y en gris a los aspectos de Considerations.

Se puede observar en la figura 6 que el GCD está dividido en tres partes:

- Setup: Define la intención (objetivos) del juego y los jugadores a los que se está dirigido este juego.
- Game Design: Describe la mecánica del juego, la interacción de éste con el usuario y el feedback que proporciona para crear la experiencia del juego.
- Considerations: Tiene en cuenta requerimientos adicionales que podrían impactar con el diseño, desarrollo y marketing del juego, como puede ser el tipo de dispositivo en el que se jugará o el tipo de interacción que se desea

3.2.1. Setup del juego 3D

Como se ha mencionado anteriormente, el SetUp consiste en identificar cuáles son los objetivos de aprendizaje del juego y a que usuarios va destinado.

3.2.1.1 Objetivos de aprendizaje

El juego 3D tiene como objetivo practicar y reforzar el aprendizaje de las siguientes competencias:

- Identificación: Esta es la competencia más básica. Es como una extensión de la competencia de identificación en 2D en la cual en el entorno 3D aparecen varias posibles figuras cuyas caras tienen un color distinto y el objetivo es identificar las figuras dadas que correspondan con la figura objetivo.
- Movimientos en el espacio: Este competencia se centra en los movimientos de traslación y rotación. Si, se diseñan actividades dirigidas a la obtención de estas competencias:
  - Rotaciones: En esta actividad, el usuario habrá de rotar las figuras que hagan falta para que coincidan, rotacionalmente, con las figuras del objetivo.
  - Traslaciones: Esta actividad se trabaja después de la actividad de rotación. Como ya se tienen las figuras básicas rotadas adecuadamente, el objetivo es trasladar estas figuras, utilizándolas todas, para que ocupen las sombras que genera la figura objetivo.
- Representación 3D : Sobre la representación en 3D se destacan dos competencias:
  - Proyecciones: Se muestran las sombras según una vista axonométrica de la figura objetivo en el entorno 3D y el objetivo es completar los sitios donde habían sombras con el mínimo número de figuras posible básicas. La diferencia con la actividad de traslaciones, es que allí las figuras ya están dadas y rotadas adecuadamente y el objetivo es trasladarlas todas

para ocupar las sombras del objetivo mientras que en esta se plantea el reto de usar en la construcción el mínimo de figuras posibles.

- Vistas diédricas: Se divide la pantalla en 4 viewports o subáreas: 3 de ellas corresponden a las vistas diédricas de la figura objetivo y la cuarta es donde trabajará el usuario en una vista axonométrica con el objetivo de crear una figura que correspondiera a las 3 vistas del objetivo si la cámara se rotase.

- Composición: Esta competencia es la más completa y compleja de algunas de las competencias las mencionadas anteriormente. Consiste en combinar todas las competencias anteriores para poder construir un objeto complejo objetivo (por ejemplo, un edificio del poblado). Dada una serie de figuras básicas, el jugador ha de decidir cuáles usar y trasladarlas o rotarlas de forma que se vayan uniendo y formen la figura objetivo. Es la más completa porque abarca bastantes competencias mencionadas anteriormente, como:

  - la identificación para saber qué tipo de figuras básicas es la que contiene la figura objetivo.
  - Movimientos en el espacio: Se utilizan traslaciones y rotaciones para unir las figuras.

El juego, además tendrá un modo de ayudas que el profesor podrá configurar y decidir si activarlas o no.

Una de ellas consiste en activar las proyecciones de los figuras que va instanciando el usuario, las cuales podrían ayudar bastante para situar bien los objetos en el espacio 3D y de esta forma poder trasladar mejor los objetos.

La otra ayuda consiste en utilizar las vistas diédricas no sólo en la visualización del objetivo, sino también sobre el escenario donde trabaja el usuario. Es decir, la ventana se dividirá en 4 partes donde una es una vista axonométrica de la figura objetivo y las otras 3 corresponderán a las vistas diédricas del escenario de montaje del usuario.

3.2.1.2 Usuarios del juego

Los usuarios a los que esta aplicación está destinada son por un lado los niños del Ciclo Superior de Primaria y Ciclo Inicial de Secundaria para aprender y reforzar sobre competencias de la geometría. Y por otro están los profesores que serán los encargados de gestionar las competencias que los alumnos trabajarán y recopilarán los datos de los alumnos para monitorizar su avance en el juego y los fallos que cometan.

3.2.2. Diseño del juego

Una vez analizadas las competencias que tendrá el juego, en este apartado se explicará el diseño de la actividad correspondiente a la competencia más compleja,

que es la competencia de composición. Es la actividad que se ha completado en este proyecto. Las demás actividades propuestas relacionadas a las competencias descritas en el apartado anterior se podrían desarrollar fácilmente a partir de la actividad de composición diseñada e implementada.

Pero antes de ello, primero habrá que decidir dónde integrar en entorno 3D de trabajo en la dinámica del juego. Es en este entorno 3D donde se añadirá la actividad relacionada con la competencia de composición.

Estudiando el proyecto Geopieces generado a partir de los proyectos anteriores, se propone conectar este nuevo entorno en el menú de campaña (ver figura 7).

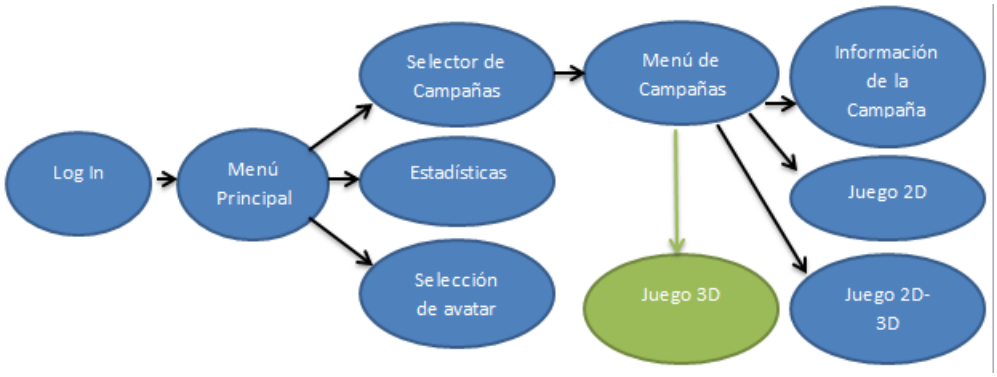

Figura 7: Esquema de las relaciones entre escenas del juego.

Es decir, sería en el momento en el que el usuario está intentando completar una campaña, en la cual se añadirá una opción adicional que llevará al usuario a la escena de este juego en 3D (ver figura 8).

Figura 8: Actividades correspondientes a la campaña *Fountain*.

A continuación se detalla el Caso de Uso Textual donde constarán las interacciones del usuario con el sistema relativas a la construcción del objeto complejo en este entorno 3D y las utilidades que dispone el usuario durante el proceso de construcción (ver figura 9 10).

Caso de Uso Principal de actividad en el juego: Composición de figuras simples para formar una figura objetivo.

Usuario: jugador.

Precondición: El usuario se encuentra en la escena de Menú de Campañas. Ha seleccionado una campaña y ha completado las actividades correspondientes al modo 2D y modo 2D-3D (lo que sería el modo de desdoblamiento de caras) para obtener las figuras simples que servirán para construir la figura objetivo.

Flujo principal:

1. El usuario selecciona la actividad del modo 3D que corresponde a la composición de figuras simples para formar la figura objetivo.
2. El sistema carga la escena del juego 3D y se la muestra al usuario.

Loop

3. El usuario clica en un boton para seleccionar la insercion de una figura simple.
4. El servidor instancia la figura simple que correspondía al botón presionado por el usuario.

5. El usuario traslada un objeto simple de forma que una de sus caras coincide con una de las caras de otro objeto simple de la escena.
6. El sistema da el resultado de comprobación CORRECTA sobre si esas dos figuras se pueden unir para formar parte de la figura objetivo.
7. El usuario selecciona el botón de volver a la campaña.
8. El servidor termina el modo de juego 3D y lleva al usuario a la escena del menú de campañas.

Flujo alternativo:

3a. El usuario clica en un botón para activar la inserción de una figura compleja o simple.

3b. El usuario clica en la opción finalizar para acabar la actividad.

El sistema cierra el escenario 3D y vuelve al menú de campañas del juego.

5a. El usuario traslada un objeto simple de forma que una de sus caras coincide con una de las caras de un objeto complejo.

5b. El usuario traslada un objeto complejo de forma que una de sus caras coincide con una de las caras de un objeto complejo.

6a. El sistema de resultado de comprobacion INCORRECTA, indicando al usuario de forma visual donde está el error.

Figura 9: Caso de uso textual sobre la actividad principal del juego 3D.

Eventos que puede realizar el usuario en cualquier momento:

*. El usuario mantiene pulsado el botón derecho del ratón mientras lo va moviendo.

ii El sistema rota la camara segun el desplazamiento del raton que se haya hecho.

*. El usuario gira la ruedecita del ratón y/o interacciona con un slider.

ii. El sistema hace zoom en el sentido en que se ha girado la rueda del ratón o sino sobre el valor que se haya obtenido al interaccionar el usuario con el slider.

3*. El usuario clica sobre la herramienta de guardar objetos.

ii. El sistema cambia al modo de guardar objetos.

iii. El usuario clica sobre un objeto.

iv. El sistema saca una foto del objeto y crea un nuevo boton de instanciacion de figuras en el cual la figura a instanciar corresponde con la clicada por el usuario previamente.

3*. El usuario clica sobre la herramienta de borrar objetos

ii. El sistema cambia al modo de borrar objetos.

iii. El usuario clica sobre una figura o un botón de instanciación de figuras añadido por el usuario.

iv. El sistema borra esa figura o botón en cuestión.

*. El usuario clica sobre la herramienta de mover objetos.

ii. El sistema cambia al modo de mover objetos.

3*. El usuario hace doble clic sobre un objeto y mientras lo mantiene, arrastra el mouse hacia otra posición.

ii. El sistema rota el objeto según el desplazamiento que realizó el ratón.

Figura 10: Segunda parte del caso de uso textual sobre la actividad principal del juego 3D.

3.2.3. Requerimientos tecnológicos

En esta sección se detallan los requerimientos tanto software como hardware que hacen falta para poder ejecutar el juego.

Sobre la plataforma, el juego se ha creado teniendo en cuenta que el usuario puede utilizar teclado y ratón o que esté usando una tablet.

Se podrá jugar al juego en los ordenadores cuyo sistema operativo sea Windows o Mac aunque también se está pensado para que funcione en las tablets de Android.

Para la resolución de pantalla se ha optado usar la 1024x600. Este requerimiento es especialmente importante para disponer el entorno 3D de trabajo de forma que sea confortable al jugador de forma que no limite sus movimientos.

4. Desarrollo del juego 3D

4.1. Justificación de la tecnología utilizada

Ahora se verán qué motores gráficos se barajaban y se explicarán los motivos por los cuales se ha creído que este juego era mejor implementarlo en Unity.

- Unity3D: Es un motor de juego, el cual te permitirá crear juegos para diversas plataformas. Se programa en C# y Javascript. La versión personal es gratuita.
- CryEngine: Es un motor de juego desarrollado por la empresa Crytek, también multiplataforma y se programa en C++, Lua y C#. La versión CryEngine V adopta el modelo "paga lo que quieras" en la cual es el propio desarrollador el que decide la cantidad de dinero que quiere pagar.
- Unreal Engine4: Es un motor de juegos gratuito para desarrolladores, es multiplataforma y se programa en C++.
- Qt y GL: Qt no es tanto un motor de juegos, sino que se trata de un Framework multiplataforma utilizado para crear aplicaciones que usen interfaz gráfica. Se programa en C++.

Cada uno de ellos tiene sus ventajas propias como inconvenientes pero al final se decidió optar por Unity.

La razón de más peso y de hecho, bastante obvia, por la que se ha utilizado Unity es porque este proyecto es una ampliación de dos proyectos en los cuales se implementó el juego de Geopieces con los modos de juegos del 2D y 2D-3D en Unity.

En ese proyecto se decidió utilizar este motor de juego ya que los desarrolladores tenían experiencia previa con este programa, es fácil de exportar el juego a distintas plataformas y también, un punto importante fue que Unity te permite decidir qué modo de juego se quiere crear, si 2D o 3D, por lo que les fue muy útil.

4.2. Arquitectura del sistema

La arquitectura del juego que se ha utilizado en este proyecto es la del Modelo-Vista-Controlador. La razón de haber elegido este patrón de arquitectura es que permite gestionar mejor el código que se implementará y la comunicación entre las diversas clases, clasificándolas en tres componentes según el concepto que representen.

Por consiguiente, el código resultante es mucho más reutilizable y facilita su mantenimiento.

En la carpeta 'Assets/scripts/Multi 3D/' se encuentran las tres carpetas que corresponden a los componentes de la arquitectura Modelo-Vista-Controlador y dentro de ellas se encuentra los scripts del proyecto.

A continuación se define el concepto que representa cada componente:

- Modelo: Parte donde se tienen todas las clases que definen propiedades adicionales a los objetos que intervienen en la escena.
- Vista: Corresponde a todas las clases que directamente interactúan con los elementos del HUD.
- Controlador: Contiene las clases que gestionan el comportamiento y buen funcionamiento del juego. Además, manejan la comunicación entre el modelo y la vista.

En los siguientes apartados de este mismo capítulo se explicará qué tipo de clases se han implementado en cada uno de los componentes.

En primer lugar, se introducirá la teoría de forma abstracta sobre cómo resolver el algoritmo más importante del juego. Luego se explicará la implementación de los objetos definidos teóricamente en Unity (lo que vendrían a ser los componentes del modelo y del controlador) y por último se estudiará la interfaz gráfica implementada (correspondiente al componente de la vista).

4.3. Teoría del algoritmo principal desarrollado

4.3.1. Formalización y explicación de los algoritmos propuestos para detectar/guiar la correcta construcción del sólido 3D final a partir de sus partes 3D iniciales.

Los retos asociados al juego 3D desarrollado en este proyecto tienen una parte común que consiste en la construcción de un objeto sólido 3D (ver figura 11), el cual se le llamará **figura objetivo** (o *target*), creado a partir de figuras 3D más simples (**primitivas básicas**), formadas por caras planas, como cubos, pirámides, tetraedros, etc.

Figura 11: Ejemplo de figura objetivo que contiene dos primitivas básicas.

A cada paso, el jugador construye un nuevo objeto intermedio juntando una nueva primitiva básica al objeto anterior, hasta llegar a una construcción final que deberá coincidir con la figura objetivo.

El algoritmo propuesto permite guiar al jugador en cada paso de la construcción de forma que si el jugador coloca una primitiva básica en un sitio incorrecto del objeto en construcción, se le puede ofrecer un feedback inmediato, guiándolo, si es

necesario, en el siguiente paso. Además, el algoritmo permite comprobar si la pieza final obtenida se corresponde correctamente a la pieza objetivo.

Para poder comprobar la equivalencia de dos objetos 3D formados por primitivas básicas, se ha definido un objeto sólido como un grafo formado por primitivas básicas, reduciendo el problema de equivalencia entre sólidos a comprobar si dos grafos son isomorfos.

El problema del isomorfismo entre grafos es un problema NP completo [26], para el que no se conoce ningún algoritmo eficiente en tiempo polinómico que nos permita encontrar el isomorfismo entre grafos sin acabar recurriendo a calcular las n! posibles combinaciones (fuerza bruta). Aún así, los grafos que se definen a continuación para representar y definir los objetos cumplen ciertas propiedades que permiten solucionar el problema del isomorfismo entre grafos de forma **exponencial** (al final de cada algoritmo se calculará la complejidad).

A continuación se describen las hipótesis iniciales sobre los objetos y el proceso de construcción, después se introduce la teoría básica de grafos sobre la que se basan el resto de definiciones y, finalmente, se incluye el algoritmo propuesto utilizando la notación de las secciones anteriores.

4.3.2. Hipótesis iniciales sobre los objetos y el proceso de construcción

Se definen **objetos simples** o **figuras primitivas** como sólidos 3D delimitados por **caras planas**. Estos objetos 3D se consideran simples ya que que no están formados por otros sólidos y no contienen en su interior ninguna cara plana 2D (como un cuadrado, triángulo). Este sería el caso, por ejemplo, del cubo, la pirámide, etc.

No se considera una figura primitiva al objeto 3D formado por la unión de dos tetraedros que comparten una cara (que podría formar un cubo), ya que el sólido resultante contendría dos caras planas en su interior, es decir, las caras por las que se han unido los dos tetraedros (ver figura 12).

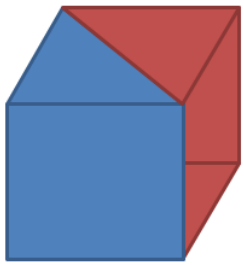

Figura 12: Ejemplo de un cubo formado por la unión de dos tetraedros.

Se define como **objeto complejo** o **sólido 3D** a todo objeto que se puede descomponer como unión de figuras primitivas simples.

Se consideran las **figuras planas 2D** o **caras** como grafos. A continuación se enumera el conjunto de hipótesis iniciales que se asumen en el problema, con el fin de simplificar y hacer más eficiente el proceso de validación de la equivalencia entre dos objetos complejos 3D.

Hipótesis 1. *Se considera que todas las caras planas, que delimitan los objetos 3D, corresponden a* ***polígonos regulares 2D*** *y además, tienen la* ***misma escala****.*

Al ser polígonos regulares, todas las aristas que limitan la cara tienen el mismo tamaño. Adicionalmente, se considera que todas las caras que formen el sólido 3D tengan la misma escala, es decir, que el tamaño de los lados sea constante para todas las caras del objeto simple. Se asume también que durante el proceso de construcción este tamaño es constante, sin poderse variar. Asumiremos este tamaño constate como K.

Hipótesis 2. *Se considera que todas las caras 2D que definen las figuras primitivas 3D, están formados con el mínimo número de caras planas.*

Esto es, una cara plana es indivisible. Por ejemplo, en el caso de un cubo, éste estará formado únicamente por caras cuadradas y sus caras no podrán estar formadas por conjuntos de triángulos (ver figura 13).

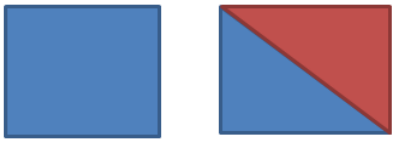

Figura 13: Ejemplo de un cuadrado simple (imagen a la izquierda) y de un cuadrado compuesto por dos triángulos (imagen a la derecha).

Hipótesis 3. *En el proceso de construcción, el usuario sólo puede unir dos objetos juntando caras de cada uno de ellos (ver figura 14).*

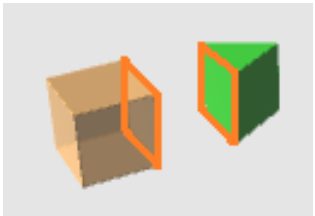

Figura 14: Se ha remarcado con un cuadrado naranja las dos caras por las cuales el usuario quiere unir esos dos objetos simples.

No se admiten unir dos objetos por aristas o vértices dado que podría dar lugar a objetos *non-manifold*.

Hipótesis 4. *En el proceso de construcción, a cada paso, el usuario sólo realiza la unión de dos objetos, siendo libre de unir dos objetos simples, un objeto simple y uno complejo, así como dos objetos complejos (ver figura 15).*

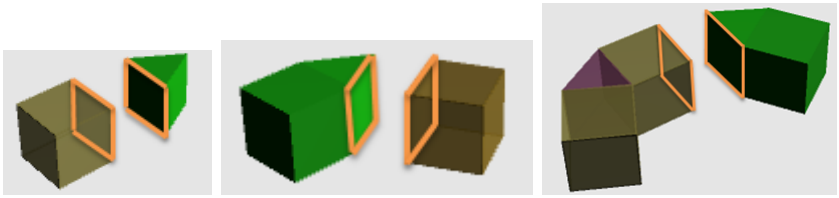

Figura 15: Se ha remarcado con un cuadrado naranja las dos caras por las cuales el usuario quiere unir: dos objetos simples (imagen de la izquierda), objeto simple con objeto complejo (imagen central) y dos objetos complejos (imagen de la derecha).

4.3.3. Teoría básica de grafos.

Definición 1. *Un* ***grafo*** $G = (V, E)$ *está formado por un conjunto no vacío* V *de vértices y un conjunto* E *de aristas o pares de elementos de* V*,* (V_1, V_2)*. Una arista de* G *es, pues, un subconjunto* $\{v_1, v_2\}$*, con* $v_1, v_2 \in V$*.*

Definición 2. *Se define* $V = V(G)$ *como el* ***conjunto de vértices*** V *del grafo* G *y* $E = E(G)$ *como* ***el conjunto de aristas*** E *del grafo* G*, para indicar a qué grafo pertenece el conjunto de vértices y el de aristas, respectivamente.*

Definición 3. *Si dos o más aristas contienen exactamente los mismos pares de vértices, se consideran* ***aristas múltiples****. En cambio, si una arista une un vértice consigo mismo, esto es, dado un* $v_1 \in V$*,* $e = \{v_1, v_1\}$ *y* $e \in E(G)$ *,se considera que dicha arista es un* ***bucle****.*

Definición 4. *Un grafo es* ***simple*** *si no tiene aristas múltiples, ni bucles.*

Definición 5. *Si* $G = (V, E)$ *es un grafo dirigido o no, entonces* $G_1 = (V_1, E_1)$ *es un* ***subgrafo*** *de* G *si* $V_1 \subset V, V_1 \neq \emptyset$ *y* E_1 *es subconjunto de* E*, donde cada arista de* E_1 *es incidente con los vértices de* V_1*.*

Se denota que G_1 *es subgrafo de* G *como* $G_1 \subset G$*.*

Definición 6. *Un* ***isomorfismo entre dos grafos*** G_1, G_2 *consiste en encontrar una función biyectiva*

$$\phi : V(G_1) \longrightarrow V(G_2)$$

tal que a cada vértice de G_1 *se le asigna un vértice de* G_2 *y además preserva la relación de adyacencia de las aristas, esto es, se cumple que:*

$$\forall\ v_1, v_1' \in V(G_1), \qquad e_1(v_1, v_1') \in E(G_1) \iff e_1'(\phi(v_1), \phi(v_1')) \in E(G_2)$$

Se denota que G_1*,* G_2 *son isomorfos según* ϕ *como* $G_1 \sim_\phi G_2$*.*

Definición 7. *El* ***grado*** *de un grafo simple* G*, denotado como* $deg(G)$*, se define como* $deg(G) := \#V(G)$*.*

4.3.4. Representación de caras regulares planas como grafos

A continuación se aplican las definiciones generales de grafos a la representación de caras, objetos simples y objetos complejos, restringiendo y definiendo algunos conceptos adicionales, tales como el tipo de adyacencias, y de isomorfismo.

De ahora en adelante se considerará que los vértices son coordenadas de un espacio vectorial (x, y, z) de R^3, dado que los objetos geométricos están representados en el espacio R^3. También, se utilizará como distancia, la **distancia euclidiana** cuando se calcule la distancia entre dos puntos.

Definición 8. *Se considera que un grafo simple G= (V,E) representa una* ***cara plana regular*** *ssi:*

• $V(G) = \{\{v_1, \ldots, v_n\} \subset R^3 \mid \forall i, j \in \{1, \ldots, n\}$ *si* $i \neq j$ *entonces* $v_i \neq v_j$ *donde* $n \in N$, $n \geq 3\}$ *(una figura plana ha de tener al menos 3 aristas).*

• $E(G) = \{\{v_i, v_j\} \cup \{v_k, v_i\} | \forall i \in \{1, \ldots, n\}, \exists j, k \in \{1, \ldots, n\}$ *con* $j, k \neq i, j \neq k$ *tal que* $dist(v_i, v_j) = dist(v_i, v_k) = K\}$. *Es decir, todas las aristas tienen el mismo tamaño, pues es una cara regular.*

• *Los* v_i *están contenidos en un mismo plano, esto es, considerando los vectores* $u_i = v_i - v_1$ *con* $i \in \{2, \ldots, n\}$, *entonces el* $Rango(u_2, \ldots, u_n) = 2$ *(como* $n \geq 3$,*se podrán crear al menos 2 vectores* u_i*).*

• *G es un grafo plano, es decir, se puede dibujar en el plano sin que se crucen sus aristas.*

Por ejemplo, el grafo simple formado por los vértices $v_1 = (-1, -1)$, $v_2 = (-1, 1)$, $v_3 = (1, 1)$ y $v_4 = (1, -1)$ y las aristas $e_1 = \{v_1, v_2\}$, $e_2 = \{v_2, v_3\}$, $e_3 = \{v_3, v_4\}$, $e_4 = \{v_4, v_1\}$ es una cara plana regular centrada en el origen y que además corresponde a la figura de un cuadrado y tienen grado 4. Ver figura 16.

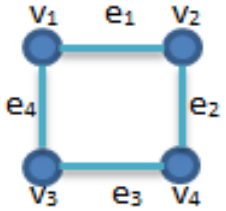

Figura 16: Ejemplo de una cara plana regular (en este caso, de un cuadrado) centrada en el origen.

Definición 9. *Se define F como el conjunto de todas las caras planas regulares.*

Definición 10. *Sean* G_1, G_2 *dos caras planas regulares, se dice que* G_1 *y* G_2 *son* ***compatibles*** *si* $deg(G_1) = deg(G_2)$.

Definición 11. *Si dos caras planas regulares,* G_1, G_2, *comparten una misma arista, es decir,*

$$\exists e \in E(G_1) \mid e \in E(G_2) \text{ donde } e = e(v_1, v_2), \text{ con } v_1, v_2 \in V(G_1) \text{ y } v_1, v_2 \in V(G_2)$$

entonces se dice que son ***adyacentes por arista****.*

Definición 12. *Dadas dos caras planas regulares, G_1 y G_2, y el isomorfismo entre ellas ϕ. Se dice que G_1 y G_2, son* ***equivalentes*** *según ϕ, $G_1 \sim \phi G_2$, ssi:*

- *Son compatibles.*
- $\forall v_i \in V(G_1)\ \exists! v_j \in V(G_2)\ tq\ v_j = \phi(v_i)$
- $\forall v_i, v_j \in V(G_1), \qquad e(v_i, v_j) \in E(G_1) \iff e'(\phi(v_i), \phi(v_j)) \in E(G_2)$

4.3.5. Representación de objetos simples como grafos de caras planas

A continuación se definen los **objetos simples** o **figuras primitivas 3D**. Estos objetos son los que inicialmente tendrá el jugador para empezar la construcción. Se tratan de sólidos 3D básicos como cubos, tetraedros, etc. Un objeto simple está delimitado por caras planas y estará representado por un grafo cuyos vértices representarán las caras planas y en el que las aristas del grafo unirán los vértices con el centro del objeto 3D.

Definición 13. *Se define* ***objeto simple****, O_{simple} a la tupla $< G_{simple}, H, f >$ donde:*

- *G_{simple} es un grafo simple tal que:*

  • *$V(G_{simple}) = \{c\} \cup \{v_1, \ldots, v_n\} \subset R^3$, donde c se dice que es el vértice centro y $v_1, \ldots, v_n$ son las coordenadas globales de las caras planas regulares.*

  • *$E(G_{simple}) = \{\{c, v_i\} | i \in \{1, ..., n\}\}$, es decir, todo vértice que representa una cara plana está conectado al centro del objeto simple.*

- *Sea $H = \{H_1, \ldots, H_n\} \subset F$ un subconjunto de caras planas regulares tal que $\forall i, j \in \{1, \ldots, n\}$, si $i \neq j$, entonces $H_i \neq H_j$ y además se cumple que toda cara regular H_i de H comparte cada una de sus aristas con alguna cara del resto del conjunto H. Esto es:*

$$\forall H_i \in H, \forall e \in E(H_i),\ \exists! H_j \in H \text{ con } i \neq j \text{ tal que } e \in E(H_j)$$

- *f es una aplicación biyectiva, $f : V(G_{simple}) \backslash \{c\} \longrightarrow H$. A cada vértice de G_{simple} que no sea el centro se le asigna una cara plana regular, H_i.*

En la figura 17 se puede ver un ejemplo de como un cubo se puede expresar como un objeto simple.

Definición 14. *Dado dos objetos simples (G_1, H_1, f_1), (G_2, H_2, f_2) se dice que son* ***adyacentes de arista*** *si existe una cara de H_1 que sea adyacente por arista a alguna cara de H_2, es decir, si $\exists \bar{H} \in H_1$ y $\exists \bar{H}' \in H_2$ tal que $E(\bar{H}) \cap E(\bar{H}') \neq \emptyset$.*

Definición 15. *Sean (G_1, H_1, f_1) y (G_2, H_2, f_2) dos objetos simples, se dice que son* ***adyacentes de cara*** *si $\exists \bar{H} \in H_1$ y $\exists \bar{H}' \in H_2$ tal que $V(\bar{H}) = V(\bar{H}')$.*

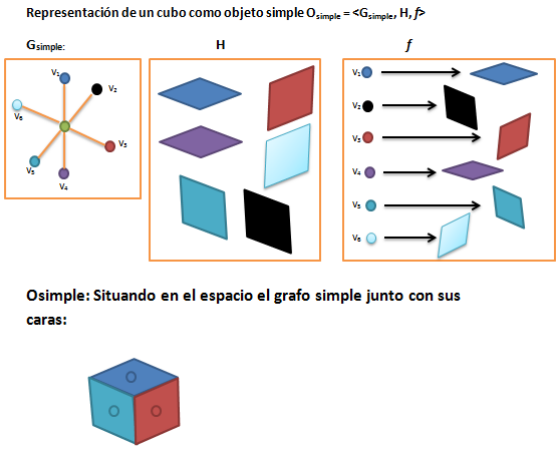

Figura 17: Ejemplo de como un cubo se puede expresar como un objeto simple.

Definición 16. *Dado dos objetos simples* $O_{simple_1} = (G_1, H_1, f_1)$, $O_{simple_2} = (G_2, H_2, f_2)$, *se dice que son* ***compatibles*** *si* $deg(G_1) = deg(G_2)$.

Definición 17. *Dados dos objetos simples* $O_{simple_1} = (G_1, H_1, f_1)$, $O_{simple_2} = (G_2, H_2, f_2)$, *se dice que son* ***equivalentes*** *según el isomorfismo* ϕ, $O_{simple_1} \sim_\phi O_{simple_2}$, *ssi:*

- *Son compatibles.*
- $\forall v_i \in V(G_1)\ \exists! v_j \in V(G_2)\ tq\ v_j = \phi(v_i)$.
- $\forall v_i, v_j \in V(G_1), \quad e(v_i, v_j) \in E(G_1) \iff e'(\phi(v_i), \phi(v_j)) \in E(G_2)$.
- $\exists R, \forall i, \exists\ !\text{J},$ *tal que* $R(H_i, c_1)$ *es equivalente a* H_j, *donde* $H_i \in H_1$ *y* $H_j \in H_2$.

donde $R(H, c)$ *es la rotación aplicada a todas las caras regulares planas de* H *en relación al centro* c *del objeto simple.*

4.3.6. Representación de objetos complejos como grafos de objetos simples

Definición 18. *Sean* $O_{simple_1}, \ldots, O_{simple_n}$ *objetos simples, se define un* ***objeto complejo*** $O_{complejo}$ *como un grafo simple* G *tal que:*

- $V(G) := \{O_{simple_i} \mid i \in \{1, \ldots, n\}\}$
- $E(G) := \{\{v, v'\} |$ *si* $v, v' \in V(G)$ *son adyacentes de cara*$\}$
- *Se cumple que todo elemento de* G *es adyacente de cara al menos con otro elemento de* G.

Definición 19. *Se le dirá* S *al conjunto formado por todos los objetos simples y* S_{obj} *al conjunto formado por todos los objetos simples que forman el objeto complejo objetivo.*

Definición 20. *Dados* φ *un isomorfismo,* $\varphi : S \longrightarrow S$ *y dos objetos simples,* O_{simple_1} *y* (O_{simple_2}), *que pertenecen a los objetos complejos* $O_{complejo_1}$ *y* $O_{complejo_2}$,

respectivamente. Se dice que una cara regular plana $H_1 \in O_{simple_1}$ *es* ***equivalente en adyacencias de arista*** *a una cara plana* $H_2 \in O_{simple_2}$ *según* φ *ssi:*

- $H_1 \sim_\varphi H_2$ *según* φ.
- $\forall e_1 \in E(H_1)$, *con* H_1' *adyacente por la arista* $e_1 \in O_{simple_1}$, $\exists!\ e_2 \in E(H_2)$, *con* H_2' *adyacente por la arista* $e_2 \in O_{simple_2}$ *tal que* $H_1' \sim_\varphi H_2'$.

Definición 21. *Dados dos objetos complejos,* $O_{complejo_1} = G_1$ *y* $O_{complejo_2} = G_2$, *se dice que* $O_{complejo_1}$ *es* ***un objeto contenido*** *en* $O_{complejo_2}$, *ssi:*

- G_1 *es subgrafo de* G_2, $G_1 \subset G_2$
- *Al menos existe un isomorfismo* φ *entre el* G_1 *y el subgrafo correspondiente a* G_2 *en que cumpla que todas sus aristas son equivalentes en adyacencia a las aristas de* G_1

Definición 22. *Dados dos objetos complejos,* $O_{complejo_1}$ *y* $O_{complejo_2}$ *se dice que son* ***compatibles*** *si* $deg(O_{complejo_1}) = deg(O_{complejo_2})$.

Definición 23. *Dados dos objetos complejos,* $O_{complejo_1} = G_1$ *y* $O_{complejo_2} = G_2$, *son equivalentes según* φ, $O_{complejo_1} = G_1 \sim \varphi O_{complejo_2} = G_2$ *ssi:*

- *Son compatibles.*
- $\forall O_i \in V(G_1) \exists! O_j \in V(G_2)\ tq\ O_j = \varphi(O_i)$.
- $\forall O_i, O_j \in V(G_1), \qquad e(O_i, O_j) \in E(G_1) \iff e'(\varphi(O_i), \varphi(O_j)) \in E(G_2)$.

y el isomorfismo φ *entre el* G_1 *y* G_2 *cumple que todas las aristas de* G_2 *son equivalentes en adyacencia a las aristas de* G_1.

Según estas definiciones, cabe destacar que dados dos objetos complejos, donde uno está contenido en el otro, puede tener múltiples isomorfismos que cumplan esa inclusión. Así pues, a cada paso del proceso de construcción de objeto complejo, al comparar con el objeto que se quiere construir, u objeto objetivo o *target*, G_{obj}, se abren tantas posibilidades de construcción como isomorfismos que cumplan esta inclusión. Esto produce que el algoritmo propuesto se base en *backtracking*, para comprobar y descartar los posibles caminos de construcción en el proceso.

4.3.7. Algoritmo principal para calcular el isomorfismo entre dos objetos complejos.

Para comenzar, se ha remarcar que el algoritmo de comprobación de equivalencia entre objetos se ejecuta a cada paso del proceso de construcción. No se ejecuta una vez el usuario ha terminado de crear el objeto complejo, sino que a medida que el usuario va construyendo su objeto, el algoritmo le va guiando, permitiéndole o no unir los objetos básicos o complejos que va añadiendo al proceso.

Inicialmente el usuario dispone de objetos simples para ensamblar durante la construcción del objeto objetivo. No obstante, durante el proceso, el usuario puede construir subobjetos complejos para poder utilizar más tarde en el ensamblaje del

objeto final. Así pues, el algoritmo contempla tres posibilidades de ensamblaje: entre dos objetos simples, entre un objeto simple y un objeto complejo y entre dos objetos complejos.

A cada paso de construcción, se crean los posibles isomorfismos de entre el objeto en construcción como objeto contenido del objeto objetivo (podría haber más de una solución posible, no tiene porqué ser única). Estos posibles isomorfismos se representan como un diccionario, que permitirá relacionar los objetos básicos que va uniendo el usuario con los del objetivo.

Tal y como se ha mencionado, el algoritmo tratará 3 casos dependiendo del tipo de los objetos (si son simples o compuestos) que el usuario ensamble en su construcción. Se denotará $O_{objetivo} = G(V_o, E_o)$ al objeto *target* que el usuario desea construir. El ensamblaje es siempre entre dos objetos y se realiza por adyacencia de caras (ver hipótesis 3):

• **Si ambos objetos son simples**: En este caso se buscarán los subgrafos de $O_{objetivo}$ que son equivalentes al objeto complejo formado por los dos objetos simples, conectados por una arista que define la adyacencia de la cara correspondiente al ensamblaje.

Se definen $O_{simple_1} = (G_1, H_1, f_1)$ y $O_{simple_2} = (G_2, H_2, f_2)$ a los objetos simples, ensamblados por la cara h, con $h \in H_1$ y $h \in H_2$. Y $O_{new} = G$ es el objeto complejo que se crea a partir (G_1, H_1, f_1) y (G_2, H_2, f_2) unidos por la cara h.

En el algoritmo se utiliza la función *add* para indicar la creación de este grafo nuevo resultado de unir los dos objetos por h mediante una nueva arista.

Data: $O_{objetivo}$, $O_{simple_1} = (G_1, H_1, f_1)$, $O_{simple_2} = (G_2, H_2, f_2)$, h
Result: $\tau(G)$ o no se pueden unir
begin
  $tau(G) = \emptyset$;
  for *each* $v_o \in V_o$ **do**
    if $v_o \sim \phi\ O_{simple_1}$ **then**
      for *all* $v'_o in V_o$ *adyacente por cara a* v_o **do**
        if $v'_o \sim \phi\ O_{simple_2}$ **then**
          O_{new} := **add**(v_o , v'_o, h)
          $\tau(G) := \tau(G) \cup O_{new}(v_o, v'_o)$;
        end
      end
    end
  end
  if $\tau(G) == \emptyset$ **then**
    return *No se pueden unir*;
  end
  return $\tau(G)$;
end

Algorithm 1: Algoritmo para comprobar la unión de dos objetos simples por cara.

Complejidad: Analizando el algoritmo anterior, se puede ver que la complejidad es de $O(\#V_o * \#E(G))$ donde V_o son los objetos simples del objetivo y G es el objeto simple de máximo $\#E(G')$ para cada objeto simple G' de V_o.

• **Si un objeto es simple y el otro es complejo**: En este caso se supone que el objeto complejo es equivalente al menos a un subgrafo de $O_{objetivo}$, es decir, el objeto complejo con el que se opera es resultado del proceso de construcción supervisado y, por lo tato, se garantiza que es parte del objeto *target* del juego.

Se definen $O_1 = (G_1, H_1, f_1)$ como el objeto complejo a los que se le quiere unir el objeto simple $O_{simple_2} = (G_2, H_2, f_2)$ mediante la cara h, con $h \in H_1$ y $h \in H_2$. Y $O_{complejo} = G$ es el objeto complejo que se crea a partir (G_1, H_1, f_1) y (G_2, H_2, f_2) unidos por la cara h. El objeto O_1 es isomorfo según φ al menos a un subgrafo de $O_{objetivo}$. $\tau(G)$ contiene los posibles isomorfismos de O_1 en relación a $O_{objetivo}$.

```
Data: O_objetivo, O_1 = (G_1, H_1, f_1), O_simple_2 = (G_2, H_2, f_2), h, τ(G)
Result: τ(G) o no se pueden unir
begin
    O_new := add ((G_1, H_1, f_1) , (G_2, H_2, f_2), h) ;
    for each O ∈ τ(G) do
        for each v ∈ O_objetivo que sea adyacente por la cara h2 a φ(O) do
            if h ∈ O_new es equivalente en adyacencias de aristas a algun
             h2 en O_objetivo then
                τ(G) := τ(G) ∪ O_new;
            end
        end
    end
    if τ(G) == ∅ then
        return No se pueden unir;
    end
    return τ(G);
end
```

Algorithm 2: Algoritmo para comprobar si la unión de un objeto complejo con un objeto simple está incluida en el objeto objetivo.

Complejidad: Analizando el algoritmo anterior, se puede ver que la complejidad es de $O(\#\tau(G) * \#E(\varphi(O)) * (\#E(\varphi(O)) + \#E(O)))$ donde $\tau(G)$ contiene los posibles isomorfismos. El primer $E(\varphi(O))$ consiste en mirar las posibles aristas de $\varphi(O)$ y luego $\#E(\varphi(O)) + \#E(O)$ es la complejidad de comprobar si un objeto es equivalente en adyacencias de aristas ya que se tiene que recorrer las aristas del objeto simple y de su correspondiente en el objetivo.

• **Si ambos objetos son complejos**: En este caso, ambos objetos complejos son isomorfos a algún subgrafo del objeto *target*. Sean O_1, O_2 los dos objetos complejos y sean $(G_1, H_1, f_1) \in O_1$ y $(G_2, H_2, f_2) \in O_2$, los dos objetos simples de O_1 y O_2 que se quieren unir mediante la cara h. Se supone que los conjuntos de posibles isomorfismos de O_1 y O_2 están definidos en los conjuntos $\tau(O_1)$ y $\tau(O_2)$

respectivamente.

Data: $O_{objetivo}$, O_1, (G_1, H_1, f_1), O_2, (G_2, H_2, f_2), h, $\tau(O_1)$, $\tau(O_2)$
Result: $\tau(G)$ o no se pueden unir
begin
  $\tau(G) := \emptyset$;
  for *each* $O \in \tau(O_1)$ **do**
    for *each* $O' \in \tau(O_2)$ **do**
      O_{new} := **add** (O , O', h);
      if $h \in O_{new}$ *es equivalente en adyacencias por aristas a* $h \in \varphi(O)$ *en* $O_{objetivo}$ **then**
        $\tau(G) := \tau(G) \cup O_{new}$;
      end
    end
  end
  if $\tau(G) == \emptyset$ **then**
    return *No se pueden unir*;
  end
  return $\tau(G)$;
end

Algorithm 3: Algoritmo para comprobar si la unión de dos objetos complejos está incluida en el objeto objetivo.

Complejidad: Analizando el algoritmo anterior, se puede ver que la complejidad es de $O(\#\tau(O_1) * \#\tau(O_2) * (\#E(\varphi(O)) + \#E(O_{new})))$ donde $\tau(O_1)$ y $\tau(O_2)$ contiene los posibles isomorfismos del primer y segundo objeto complejo. $\#E(\varphi(O)) + \#E(O_{new}))$ es la complejidad de comprobar si un objeto es equivalente en adyacencias de aristas ya que se tiene que recorrer las aristas del objeto simple y de su correspondiente en el objetivo.

Mediante estos algoritmo, se tendrían cubierto todos los casos, y una vez que el usuario haya unido todos los objetos básicos necesarios, ya se tendrían definidos todos los posibles isomorfismos.

El algoritmo supone que a cada paso el usuario va uniendo un objeto a otro. El coste de unir un par de objetos simples se mantiene constante. En principio es una buena característica ya que independientemente del tamaño del objeto objetivo (es decir, el número de objetos simples que contiene el objeto a construir), el coste de cada unión seguirá siendo el mismo.

La razón por la cual el coste es constante es porque cuando se unen dos objetos solo se comprueba localmente si la posible unión de estos dos objetos corresponden con alguna arista del objetivo. Localmente significa que se comprueba si al unir un objeto simple (G_1, H_1, f_1) al objeto simple (G_2, H_2, f_2) se respetan las adyacencias de arista entre (G_1, H_1, f_1) y los objetos simples que comparten una cara con (G_2, H_2, f_2).

El principal problema que podría haber es que se detectaran falsos positivos, es decir, al no hacer una búsqueda exhaustiva para comprobar las relaciones topológi-

cas globales, es decir cómo están situados respecto no a solo los objetos simples locales que comparten con una cara de (G_2, H_2, f_2) ,sino respecto a todos los demás del objeto complejo que contiene a (G_2, H_2, f_2).

Atacar este problema mediante la búsqueda en el grafo produciría algoritmos muy complejos en eficiencia, problema inicial que se ha intentado evitar mediante la aproximación propuesta.

La solución que se propone es realizar una comprobación final. Al terminar, para asegurar que dos objetos complejos son iguales (más bien isomorfos), se rotará el objeto complejo creado por el usuario de forma que esté orientado de la misma forma que el objetivo.

Como del algoritmo anterior, se dispone de todos los posibles isomorfismos creados, se calcula la distancia de cada objeto simple del objeto en construcción con su correspondiente objeto simple del objetivo. Si la distancia es la misma para todo los objetos simples, entonces se puede garantizar que efectivamente los dos objetos complejos son equivalentes.

Ahora se explicará este proceso con más detenimiento:

- Primero, identificar cada objeto simple del objeto complejo creado por el usuario con algún objeto simple de la figura objetivo (ver figura 18).

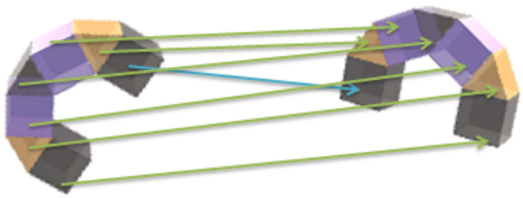

Figura 18: Isomorfismo entre el objeto complejo del usuario y el objetivo.

- Segundo, se calculan los dos vectores más característicos del objeto complejo creado por el usuario. Para ello se busca que objetos simples tienen distancia mayor según el eje X, Y y Z. De estos 3 vectores que se pueden formar, se escogen dos con mayor distancia.

  A partir del isomorfismo visto en la figura 18 se pueden crear los vectores característicos del objetivo (ver figura 19).

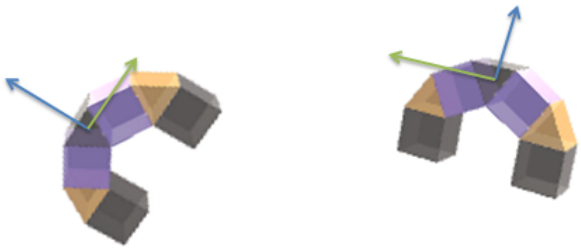

Figura 19: Cálculo de los dos vectores más característicos.

- Tercero y último paso, se rota el objeto complejo creado por el usuario según estos vectores característicos de forma que coincidan con los del objetivo.

  Finalmente, se calcula la distancia entre cada objeto simple con su correspondiente en el objetivo y si la distancia es la misma para todo objeto simple, entonces ambos objetos complejos son iguales. (ver figura 20).

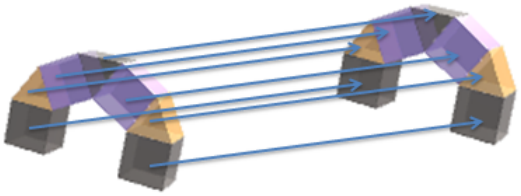

Figura 20: Rotación del objeto complejo creado por el usuario y se calcula la distancia entre los objetos simples de este objeto complejo con el objetivo.

4.4. Paso de la teoría a la implementación

Una vez se ha visto la manera en la que se definen los objetos complejos y sus propiedades, ahora llega el momento de ver cómo poder implementar estos conceptos teóricos en Unity, motor gráfico sobre el que se ha decidido implementar este juego.

En primer lugar, se introduce la representación escogida de una figura plana regular. Por definición, una figura plana regular es un grafo que contiene una serie de vértices y aristas que cumplen ciertas propiedades para que representan correctamente una figura geométrica regular en 2D. Se ha definido teóricamente de esta forma ya que luego se pueden definir de forma simple las adyacencias, ya sea de cara o de arista, o ver si dos de ellas son equivalentes, etc.

En la implementación, sin embargo se ha decidido utilizar objetos sólidos que representan figuras geométricas regulares debido a que se permite al usuario ver las caras como sólidos, en vez de en forma de grafos. Además, con esta representación también de puede incluir una forma de poder representar igualmente las propiedades de las figuras planas regulares sobre estos sólidos.

Para poder representar sus propiedades, se ha decidido añadirle un script a cada cara (de ahora en adelante le diremos **cara** a los objetos sólidos que representan figuras planas regulares). En dicho script se guarda una lista de caras que son adyacentes por arista con la cara en cuestión. También se guarda la cara con la cual es adyacente por cara. Así, de esta forma, se podrán unir objetos para formar objetos complejos.

Los scripts añadidos a las caras son: *AttributesSquare*, *AttributesTriangle*, *AttributesPentagon*. Es un script por cada tipo de cara diseñada. De momento, se han diseñado 3 objetos sólidos en Unity que representan las figuras planas regulares: el cuadrado, el triángulo y el pentágono.

Una vez se tienen las caras implementadas, hay que pasar a representar un objeto simple (G, H, f). En este caso, su implementación ha sido prácticamente idéntica a

la que se ha visto en el apartado teórico.

Se ha creado un *GameObject* vacío (correspondería al vértice central del objeto simple) en el cual se le han incluido las caras como si fueran hijos de este *GameObject*. Se ha adoptado esta medida por tres razones:

1. En Unity, si tenemos varios objetos y se quiere que al trasladar uno, los demás también se muevan, entonces la mejor forma de hacerlo es poner los otros objetos como hijos del objeto en cuestión al que se le quiere aplicar la traslación.

2. Esta representación sirve como implementación de las aristas de G, ya que, si se recuerda la definición, las aristas están formadas por par de vértices el vértice central, cualquier otro vértice.

3. También sirve para representar la función f del objeto simple. Esta función relaciona vértices que son distintos del centro con una cara. Como todo elemento de Unity tiene como propiedad un vector que corresponde a su posición en el entorno 3D, las caras que representan figuras planas regulares pasan a representar también los vértices del grafo del objeto simple distintos del vértice central.

El único problema que queda por solucionar consiste en detectar cuándo dos objetos simples han colisionado para poder determinar si se pueden unir o no. Para ello se utiliza un componente de Unity llamado *collider* que básicamente es un objeto invisible que si entrase en contacto con otro collider (u objeto), lanza la llamada a un método que permite tratar la colisión.

Asignando un collider que recubriese todo el objeto simple, se podría tratar el caso de la colisión entre los colliders de esos objetos simples. No obstante, el problema que surge es que no se sabría cuál de las caras han sido las que en verdad fueron colisionadas. Por esta razón se ha decidido, en vez de añadir un collider a todo el objeto simple, añadir un collider a nivel de cada una de las propias caras.

Se ha decidido crear un algoritmo genérico (independiente del tipo de la cara) para detectar si las dos caras son adyacentes por cara. Es decir, se ha creado un nuevo script llamado *AttributesFigure*, abstracción de los otros tres scripts de las caras de forma que éste llama al método que se acabará implementando en la clase de *FacesCollisionController* (es el componente controlador para ejecutar el algoritmo).

Por lo tanto, mediante esta propuesta se implementa un objeto simple, y utilizando el concepto de *prefab* de Unity, se puede guardar el objeto simple junto con sus configuraciones (scripts y otras propiedades que tengan). De esta forma se pueden crear copias de este objeto tal que cada copia hereda la configuración del objeto y sus caras serían copias independientes de las caras del objeto simple.

En este punto, ya se tiene definido todo lo necesario para crear los objetos complejos:

- Se pueden crear instancias de los objetos simples.

- Se pueden tratar las colisiones de los objetos simples para determinar si se pueden unir o no.

Por último, se necesitan implementar una serie de controladores para poder implementar la física con la cual se moverán los objetos y la lógica para determinar si se pueden unir los objetos simples o compuestos con otros objetos.

Para el tema de crear nuevas instancias se ha creado la clase de *SceneController* que es el controlador principal que se comunica con los otros componentes. Para la implementación del algoritmo principal se ha creado la clase de *GraphController*. Para generar la física con la cual se podrán trasladar y rotar los objetos, se ha creado la clase de *MovementController*. Y la que ya se ha mencionado anteriormente, *FacesCollisionController*, para determinar si dos caras son adyacentes de caras.

4.5. Análisis sobre la interfaz gráfica implementada

En esta sección se procederá a explicar los elementos más característicos del HUD. Se supone que el usuario ya se está situado en la escena del menú de campañas y se ha seleccionado una campaña, por ejemplo la del **SimpleHouse** y le ha aparecido el entorno del juego 3D son el objetivo que se debe conseguir construir.

Para este proyecto se destacaron dos versiones del interfaz del usuario.

Versión preliminar del interfaz de usuario

Se muestra en la figura 21 un diseño preliminar del interfaz de usuario que sería como la primera versión que se hizo.

Figura 21: Diseño preliminar del interfaz de usuario en el entorno 3D.

Para empezar se tiene el inventario en la columna de la izquierda, que contiene objetos básicos que, al ser clicados, se instancian de forma aleatoria en el escenario de trabajo.

Si se quisiera cambiar el modo de trabajo (modo de rotación/traslación, modo para guardar o eliminar un objeto creado) se ha de seleccionar una de las herramientas.

Para reinicializar el nivel o comprobar si el objeto construido es el objetivo, se disponen de los botones de Reset y Ready.

En el PC, si el usuario quiere girar la cámara, puede hacerlo manteniendo pulsado el botón derecho del ratón y luego arrastrándolo pero en la tablet, como no se tiene ratón, se ha decidido añadir un joystick, el cual solo es visible si el sistema operativo es Android.

En el HUD del objetivo, se puede observar la figura objetivo que el usuario ha de intentar recrear. Se le añadió un slider para que los jugadores con tablet pudieran hacer zoom. Los que jueguen con un ordenador, pueden además hacer zoom con la rueda del ratón.

Luego están las estrellas, las cuales sirven como incentivo para motivar al estudiante de que realice la actividad pronto y sin cometer errores. Esta idea se hereda del juego de partida donde en cada reto el jugador tiene la posibilidad de ganar estrellas si realiza bien y en un tiempo concreto la prueba.

También se ha añadido el botón para volver hacia atrás, como ya se tenía en las otras escenas del juego preliminar.

Y por último, en la parte central del HUD se tiene el escenario 3D donde el usuario jugará instanciando objetos simples y uniéndolos para crear la figura objetivo.El zoom slider es un slider que permite hacer zoom en el escenario 3D de trabajo, donde se está construyendo la figura objetivo.

Con este diseño se consigue tener todas las herramientas visibles y fáciles de acceder durante el proceso de construcción.

Versión final del interfaz de usuario

El objetivo de esta nueva versión es mejorar la apariencia, el *look and feel* y, en general, la usabilidad del juego.

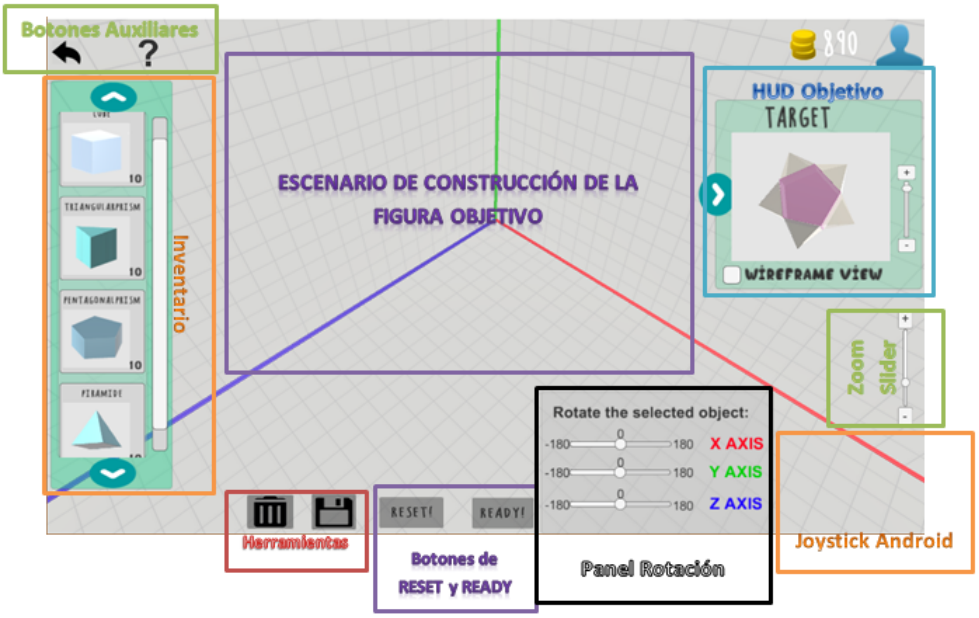

Figura 22: Diseño final del interfaz de usuario en el entorno 3D.

En la figura 22 se puede observar como ha quedado la interfaz finalmente. Ahora se pasará a explicar las diferencias entre las dos versiones.

Para empezar, donde antes estaba el botón para volver al menú de campañas (parte superior, izquierda de la figura 22) ahora se ha añadido también un botón con la forma ? que abrirá una ventana emergente para explicarle al usuario cual es el objetivo de esta actividad (ver figura 23) .

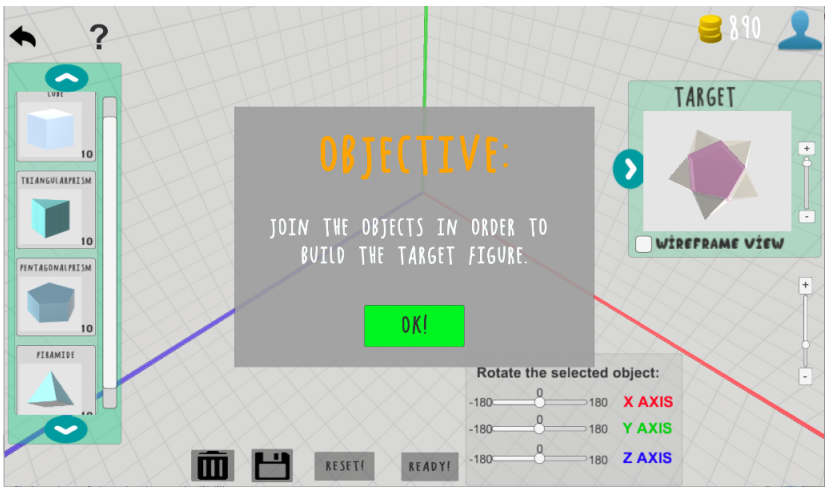

Figura 23: Aparece una ventana emergente para indicarle el objetivo de la actividad al usuario.

En la parte del HUD Objetivo se ha añadido una opción en la cual si el usuario la activa, se pasará a ver la figura objetivo en modo *wireframe* (ver figura 24)

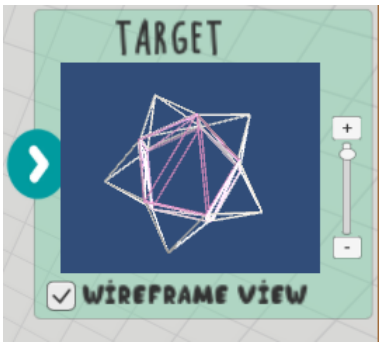

Figura 24: Aparece una ventana emergente para indicarle el objetivo de la actividad al usuario.

El siguiente cambio es bastante importante: ahora cuando el usuario clica un objeto, este se convierte en el objeto seleccionado por el usuario (se pintará de verde, ver figura 25) el cual activará los botones de Ready, Reset, los de herramientas y el panel de rotación (de hecho este panel al principio se oculta pero se activó en las imágenes anteriores para poder mostrarlo).

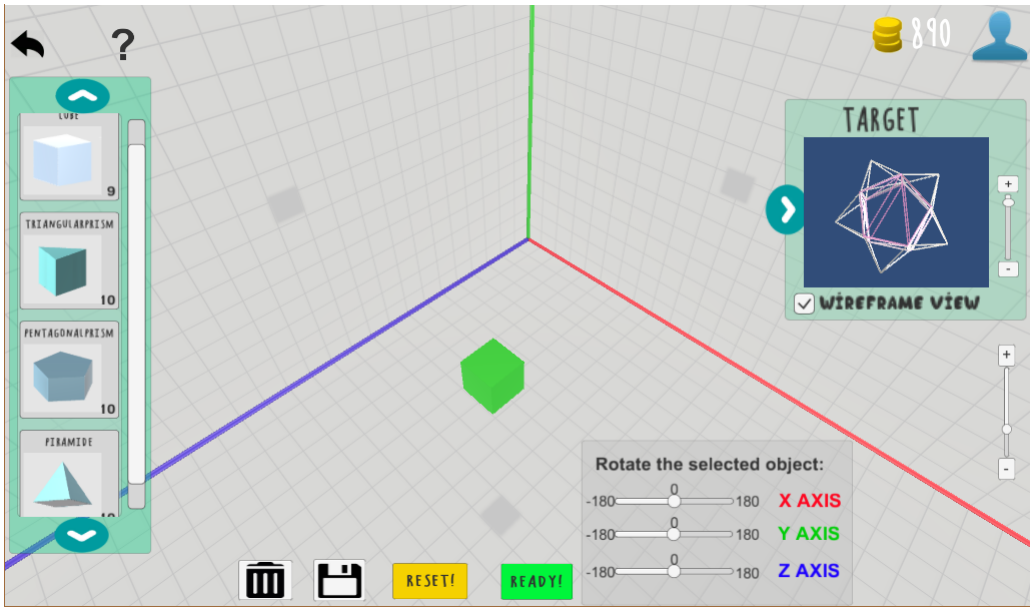

Figura 25: Aparece una ventana emergente para indicarle el objetivo de la actividad al usuario.

Ahora cuando se ha seleccionado un objeto, con el panel de rotación se puede girar el objeto en cuestión según los 3 ejes cartesianos. Si se observa el escenario, se han añadido 3 líneas (ver figura 25), cada una correspondiente con uno de los ejes mediante su color (el color rojo representa el eje X, el verde el eje Y y el azul el eje Z).

Se ha quitado los modos de juego (las herramientas) ya que para guardar o eliminar un objeto se necesitaba primero seleccionar el modo de trabajo y luego clicar sobre el objeto para eliminarlo. Ahora que se tiene lo del objeto seleccionado, ya todo esto no hace falta, solo basta en clicar en los botones de eliminar o guardar.

También se ha mejorado cada uno de los botones anteriores. El de eliminar te muestra una ventana emergente para confirmar si quieres eliminar un objeto (ver figura 26) y cuando guardas un objeto, el scrollbar del inventario baja hacia abajo de todo para mostrarle al usuario el objeto que acaba de guardar (ver figura 27).

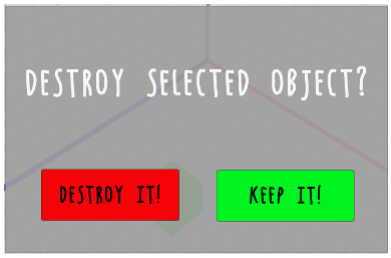

Figura 26: Si se clica en el botón de eliminar (el que tiene pinta de papelera) aparecerá una ventana emergente.

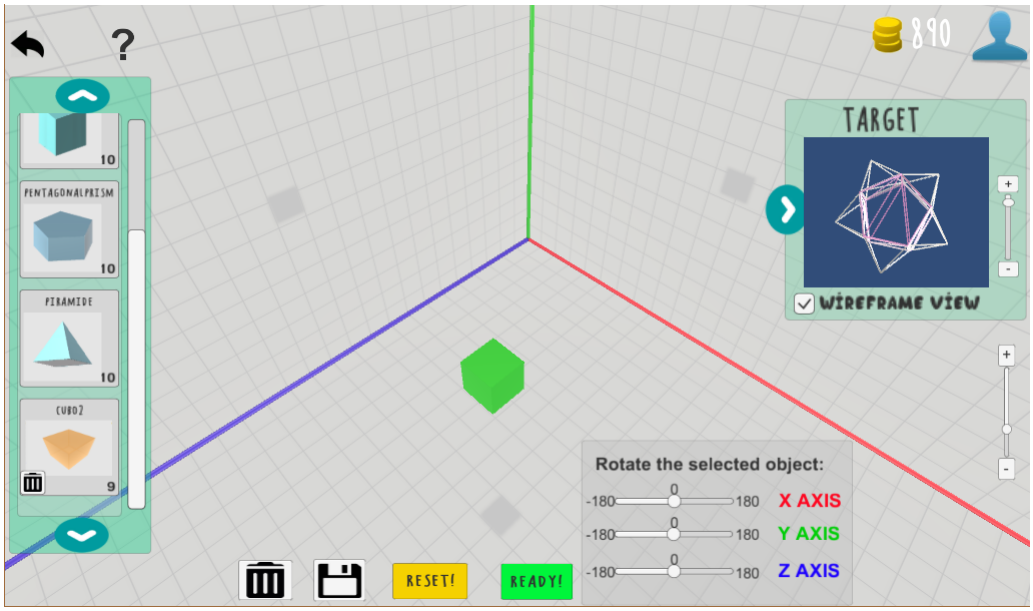

Figura 27: Si se clica en el botón de guardar (el que está a la izquierda del botón Reset) aparecerá en el inventario un nuevo botón.

Con el botón Reset y Ready pasa lo mismo que con el de eliminar: se muestran ventanas emergentes. Para el Reset, consiste en confirmar si se quieren borrar todos los objetos (ver figura 28). Mientras que para el Ready, mostrará al usuario un mensaje de éxito o fracaso dependiendo de si ha conseguido crear correctamente la figura objetivo. También dará la posibilidad de volver al menú anterior (ver figura 29).

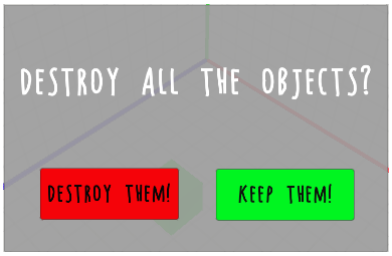

Figura 28: Ventana emergente requerida para pedirle confirmación al usuario si quiere eliminar todos los objetos.

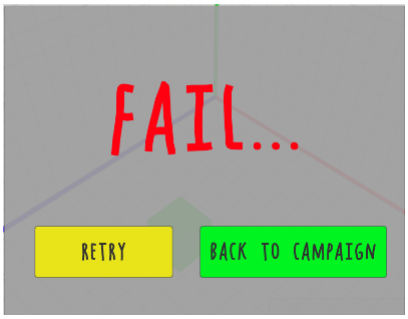

Figura 29: En este caso, el mensaje que se le muestra al usuario en la ventana emergente es de fracaso.

Se añadió en los botones nuevos que se agregan en el inventario un botón propio de eliminar (ver figura 30).

Figura 30: Botón nuevo que se ha añadido al inventario.

Por último, también se implementó que cuando dos piezas que el usuario ha intentado unir no fuesen correctas, se mostrase el objeto seleccionado de color:

- Rojo si las piezas no están bien rotadas.
- Rojo, un poco más oscuro, si las piezas no eran las correctas.
- Rojo bastante oscuro si las caras que se intentaron unir no son del mismo tipo.

Y que se mostrase una imagen de que el programa estaba cargando cuando se ejecutaba el algoritmo, pues este podía tardar varios segundos.

5. Diseño y desarrollo de la ampliación del servidor

Uno de los objetivos claves de este proyecto consiste en ampliar el servidor desarrollado en el proyecto final de grado de Fracsland, de forma que el profesor pueda configurar las competencias de los diversos modos de juego del Geopieces.

Por tanto, para el servidor los usuarios son los profesores, no como antes cuando se hablaba sobre el juego 3D donde los usuarios podían ser los alumnos.

Para empezar, se explicará la razón de porqué se utiliza Django para el servidor y a continuación, se hablará de cómo se llegó a diseñar e implementar la ampliación del servidor.

5.1. Justificación de la tecnología utilizada

Para la ampliación del servidor, como éste se había desarrollado en Django [18], está claro que se debía seguir utilizando este Framework para poder implementar la ampliación para que también configurase el juego de Geopieces.

Las razones por la cual el servidor se desarrolló con Django son:

- Permite desarrollar aplicaciones web de alto nivel.
- Una vez adquirido unos conocimientos básicos sobre su funcionamiento, es bastante sencillo de utilizar por parte del programador.
- Provee medidas para mejorar la de seguridad en la aplicación Web, de forma que ayuda al programador a evitar errores típicos que pudiese dejar vulnerable la aplicación.
- Es bastante escalable: a medida que el servidor va creciendo, no pierde calidad en sus servicios.

Para el tema de la programación del cliente se usaron los programas:

• HTML5: Es bastante popular actualmente.

• Bootstrap Admin: Permite crear interfaces típicas de forma fácil y usable.

• JQuery i JQueryUI : Permite implementar funciones complejas pero que son usuales de forma sencilla.

Sobre la ampliación del cliente se programaron además en Javascript las funciones un tanto complejas para el tema de guardar/cargar los datos de las competencias.

El sistema gestor de base de datos utilizado es PostgreSQL ya que es robusto, soporta las transacciones atómicas, y favorece la consistencia. También, porque se enseña en la asignatura de Base de Datos de este grado.

5.2. Diseño inicial de la configuración del juego Geopieces en el servidor

Para empezar, se ha de tener un primer esbozo del la GUI con la que interactuará el profesor para configurar el juego, una primera idea inicial sobre lo que se querrá implementar. Para ello, se ha desarrollado un primer prototipo el cual recoge todas las competencias y ayudas que podría configurar el profesor en el servidor (ver Figura 31.

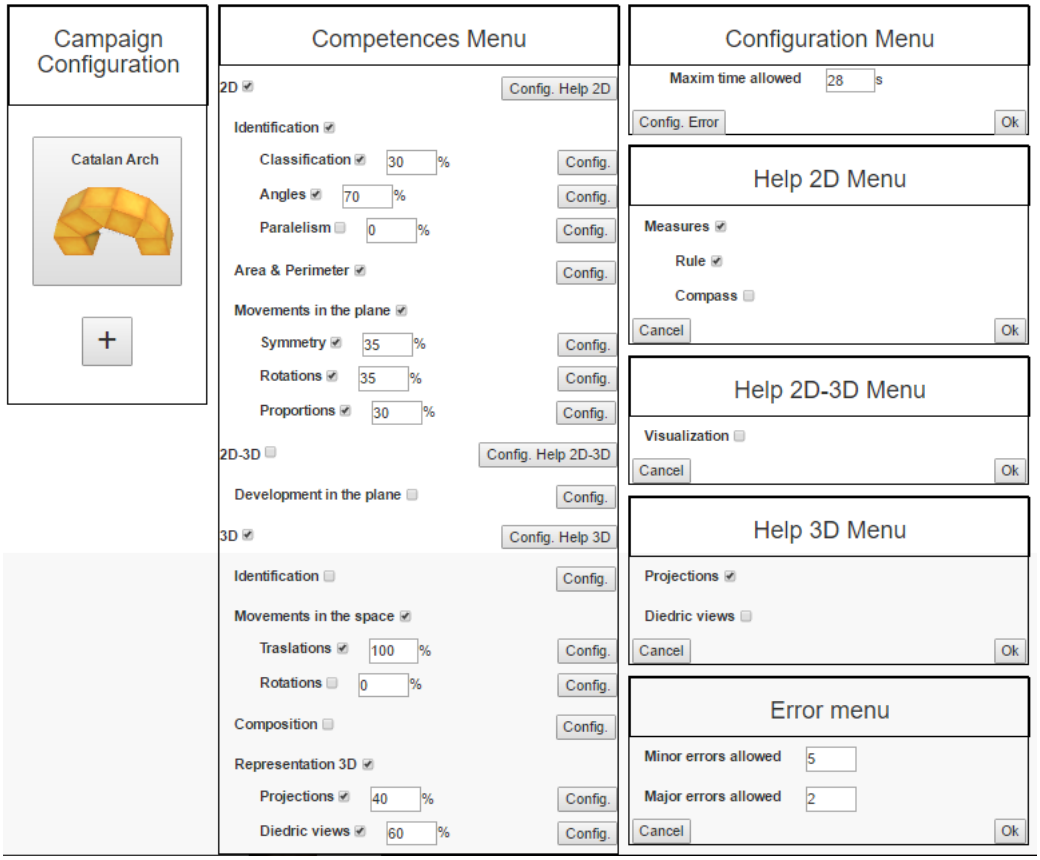

Figura 31: Primera aproximación de la GUI del servidor.

En la figura 31, se pueden apreciar las competencias y un menú de ayuda para cada uno de los modos de juego (2D, 2D-3D y 3D). También aparece un menú para gestionar los errores que se le permitiría cometer al usuario.

Antes de introducir cómo se ha realizado la implementación de este menú de configuración, primero se analizará dónde se guardarán estos datos una vez se envíen del cliente al servidor. Será necesario para ello ampliar primero la base de datos del servidor.

5.3. Diseño e implementación de la ampliación de la base de datos

Para poder ampliar la base de datos para añadir los datos referentes al juego de Geopieces, se han estudiado en primer lugar las entidades que habían sido creadas para gestionar el juego de Fracsland.

Sobre el diseño de la ampliación, se comenzó con la generalización de las entidades en las cuales ambos juegos (Geopieces y Fracsland) compartían atributos y finalmente se diseñó el resto de entidades necesarias para poder guardar los datos de Geopieces, teniendo siempre en mente la consistencia de la base de datos.

En la figura 5.3.0.1 se pude observar el estado final del diseño de la base de datos.

En la figure 5.3.0.1 se ha marcado en azul las entidades nuevas o modificadas que se han utilizado para Geopieces. Para Fracsland se ha puesto el color verde claro. El resto son entidades comunes para ambos juegos.

5.3.0.1 Análisis de las entidades creadas y modificadas

Las entidades ya existentes que fueron actualizadas son:

- QuestConfiguration: Se modificó para abstraer algunos atributos (la entidad padre es GameConfiguration) ya que tanto esta entidad como la de CampaignConfiguration tenían atributos en común.
- Quest:Attribute: Se modificó para abstraer algunos atributos (la entidad padre es BasicAttribute) ya que tanto esta entidad como la de CampaignAttribute tenían atributos en común.
- Game: Antes, cada juego se relacionaba con instancias de la entidad QuestConfiguration, pero ahora como se quiere hacer que cada juego guarde los datos tanto si son de Fracsland como de Geopieces, se ha hecho que ahora Game esté relacionado con la entidad GameConfiguration que era la entidad padre.

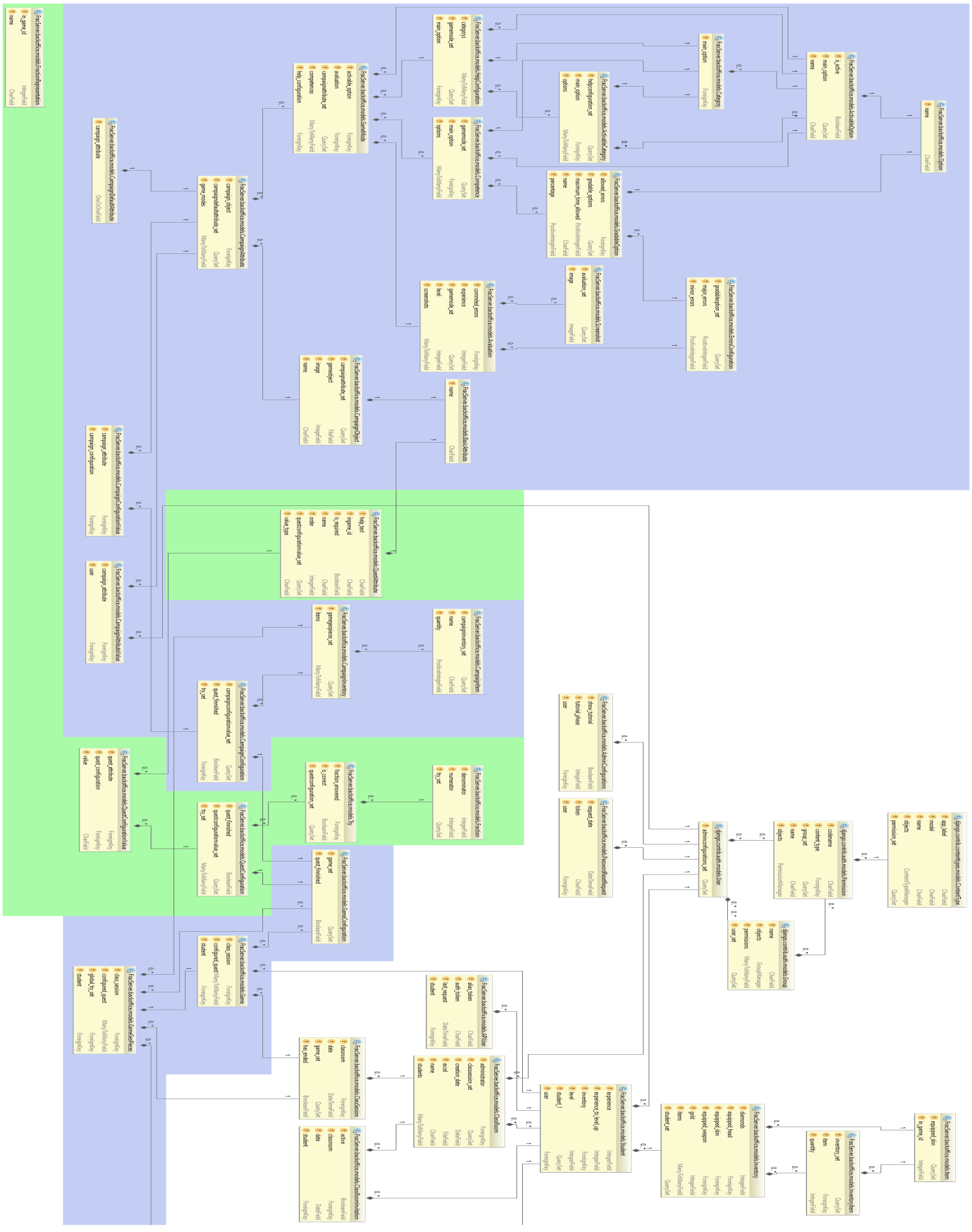

Figura 32: Diseño final del servidor.

Se han implementado nuevas entidades:

- GameGeopieces: Hereda de la entidad Game y contiene una instancia de CampaignInventory que correspondería al inventario compartido por todas las campañas.
- CampaignInventory y CampaignItem: CampaignInventory vendría a ser el inventario de figuras que se hayan obtenido en la campaña. Las figuras se representan como la entidad CampaignItem que guarda la propiedad del nombre de la figura y la cantidad que hay de esta figura.
- GameConfiguration: Contiene un booleano para indicar si la misión se ha finalizado. Es la entidad padre de GameConfiguration y QuestConfiguration.
- CampaignConfiguration: Corresponde a la entidad que representaría a una campaña de Geopieces. Hereda de la entidad GameConfiguration y tiene una instancia de CampaignInventory el cual representaría el inventario de figuras de esta campaña en particular.
- CampaignConfigurationValue: Es una entidad que relacionará una instancia de CampaignConfiguration con una instancia de CampaignAttribute.
- CampaignAttributeValue: Es una entidad que relacionará una instancia de User con una instancia de CampaignAttribute. Así, cada usuario podría guardar las nuevas campañas que crease.
- CampaignAttribute: Corresponde a los datos esenciales de una Campaña. Tiene una instancia de CampaignObject y una serie de instancias de GameMode.
- CampaignDefaultAttribute: Son las instancias de CampaignAttribute que son comunes para todos los usuarios.
- BasicAttribute: Entidad que sólo tiene de atributo un nombre. Es la entidad padre de QuestAttribute y CampaignObject.
- CampaignObject: Hereda de BasicAttribute y contiene una url de una imagen y una url del objeto (prefab) que corresponden a la figura objetivo que ha de recrear el usuario en el juego 3D.
- GameMode: Esta entidad representa los datos de un modo de juego (2D, 2D-3D, 3D). Contiene una instancia de Avaluation (la evaluación del alumno), ActivableOption (nombre del modo de juego), HelpConfiguration (menú de ayudas) y una serie de instancias de Competence (las competencias de este modo de juego).
- Option: Vendría a ser la clase padre de ActivableOption y GradableOption haciendo que ambas clases compartan el mismo atributo: el nombre de la opción.
- ActivableOption: Opción activable, hereda de Option y tiene además un booleano para indicar si es activable o no.

- GradableOption: Opción graduable, hereda de Option y tiene un entero para ajustar el porcentaje de esta opción y otro para indicar el tiempo máximo permitido para realizar esa opción. También tiene una instancia de ErrorsConfiguration para saber qué número de errores leves y graves son permitidos.
- ErrorsConfiguration: Contiene dos enteros para indicar el número de errores leves graves permitidos.
- Category: Contiene una opción activable. Hace de entidad padre de ActivableCategory, HelpConfiguration y Competence.
- ActivableCategory: Hereda de Category, por lo cual tiene un ActivableOption (nombre y booleano de esta categoría) y una serie de instancias de ActivableOption que serían las opciones activables de esta categoría.
- Competence: Vendría a ser como 'GradableCategory' es decir, es una categoría y sus opciones son graduables.
- HelpConfiguration: Representa el menú de ayuda, es una categoría con opciones activables.
- Avaluation: Corresponde a la evaluación del alumno en este modo de juego. Guarda el nivel, experiencia, los errores leves y graves cometidos y las fotos de cuando cometió un error grave.
- Screenshot: Foto que se saca del juego cuando el alumno comete un error grave para más adelante indicarle al profesor sobre ello.

5.4. Inicialización de las competencias

Las competencias, tal y como están definidas, son genéricas, por lo que de alguna manera habrá que ver de crear las competencias específicas del juego Geopieces.

Para ello, se estudiaron varias formas de hacerlo. Una primera posibilidad era hacer un método que crease directamente las competencias de forma específica de cada modo de juego, lo cual se observó que no reutilizaba mucho el código y se hacía difícil el mantenimiento, por lo que se descartó esta opción. Se decidió que las competencias se cargasen a partir de los datos de un objeto JSON.

De esta forma es mucho más sencillo de cambiar, editar o borrar las competencias, por ejemplo.

5.5. Ampliación del panel de administración del profesor

En esta sección se hablará de las páginas web del servidor que se han modificado para ampliarlo y poder configurar el juego Geopieces.

Básicamente son tres páginas:

- La primera sirve para elegir el tipo de juego a configurar (puede ser Fracsland o Geopieces, ver figura 33).

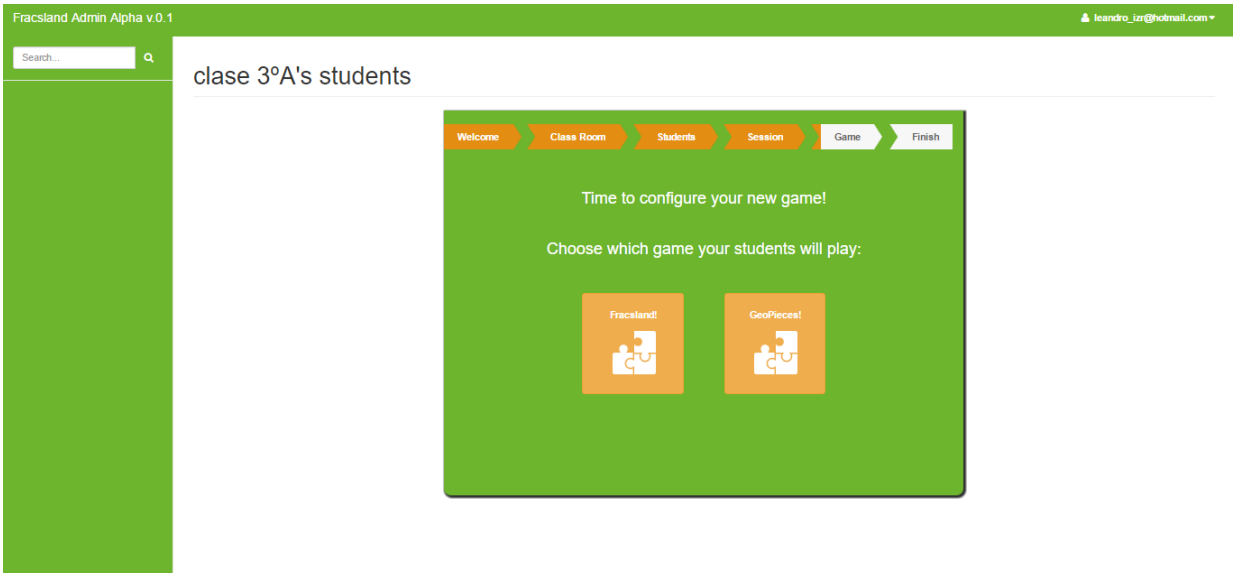

Figura 33: Selección de juego a configurar.

- La siguiente página es la de descargas, que se añadió que se pudiese descargar el juego Geopieces (ver figura 34).

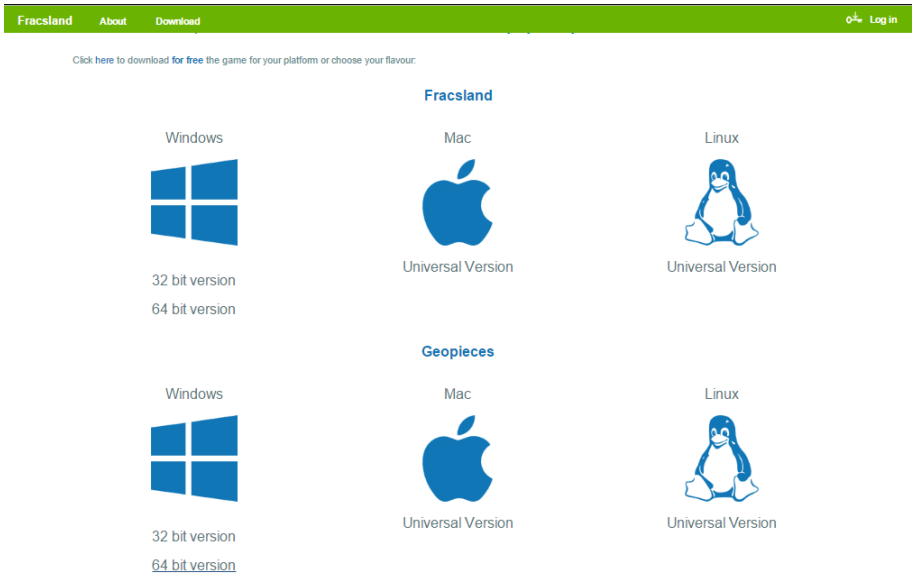

Figura 34: Página de descarga de los juegos.

- Y por último, la página de configuración del juego en concreto (ver figura 35).

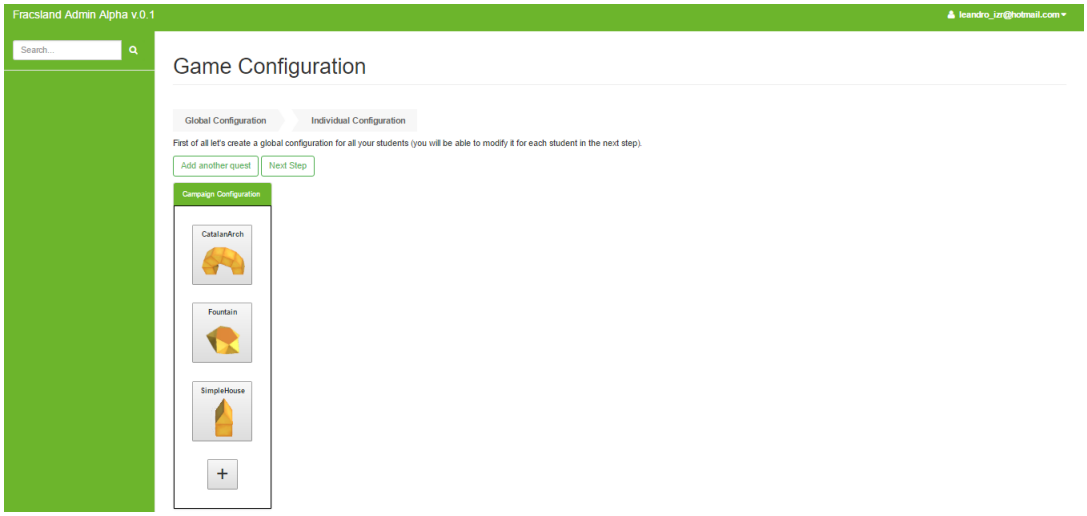

Figura 35: Página de configuración del juego.

Si clicásemos sobre algún objeto nos saldría una ventana emergente en la cual se podría configurar las competencias y las ayudas de cada modo de juego de dicho objeto (ver figura 36). Los modos de juego están situados en pestañas para que, de esta forma, no se sobrecargue visualmente al profesor con demasiados datos de golpe. Así, él podrá seleccionar qué modos configurar y cuáles no.

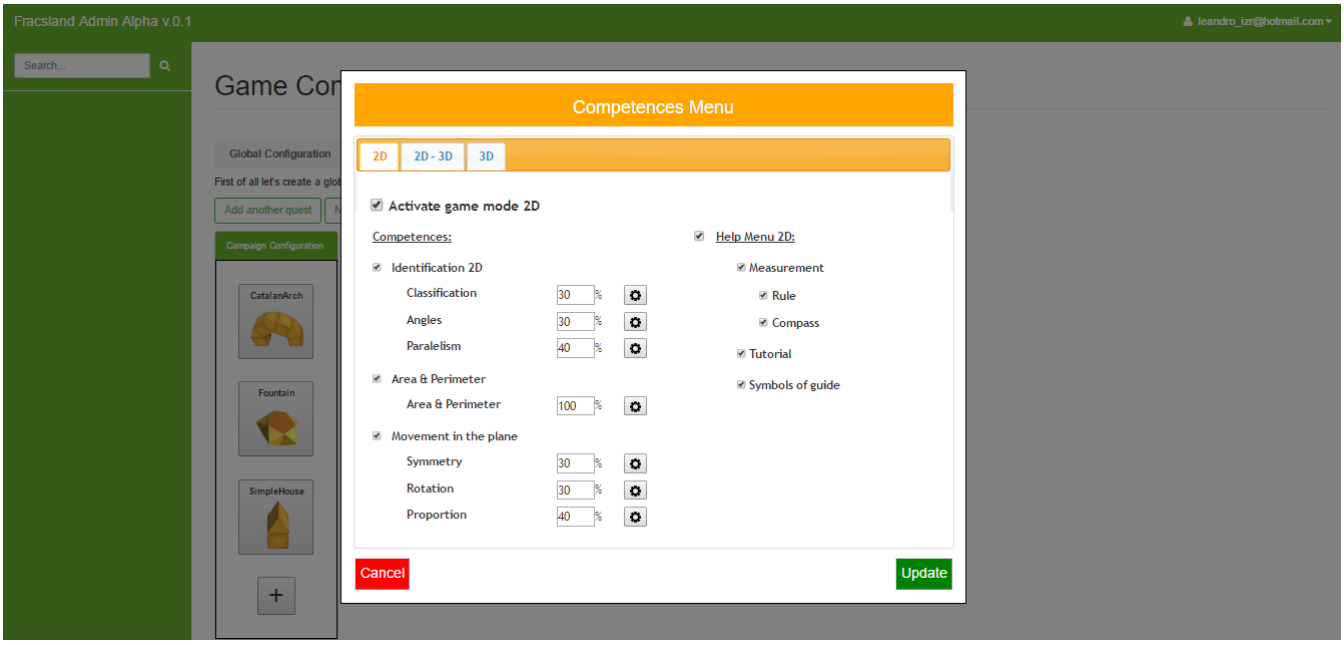

Figura 36: Página de configuración de las competencias.

Si se selecciona el botón del engranaje, se abrirá otra ventana emergente para

poder configurar la opción que contiene dicho engranaje (ver figura 37). En esta ventana, se podrán configurar los números de errores graves y leves que se le permite al alumno cometer. Otro atributo a configurar es el tiempo máximo permitido para realizar esta actividad.

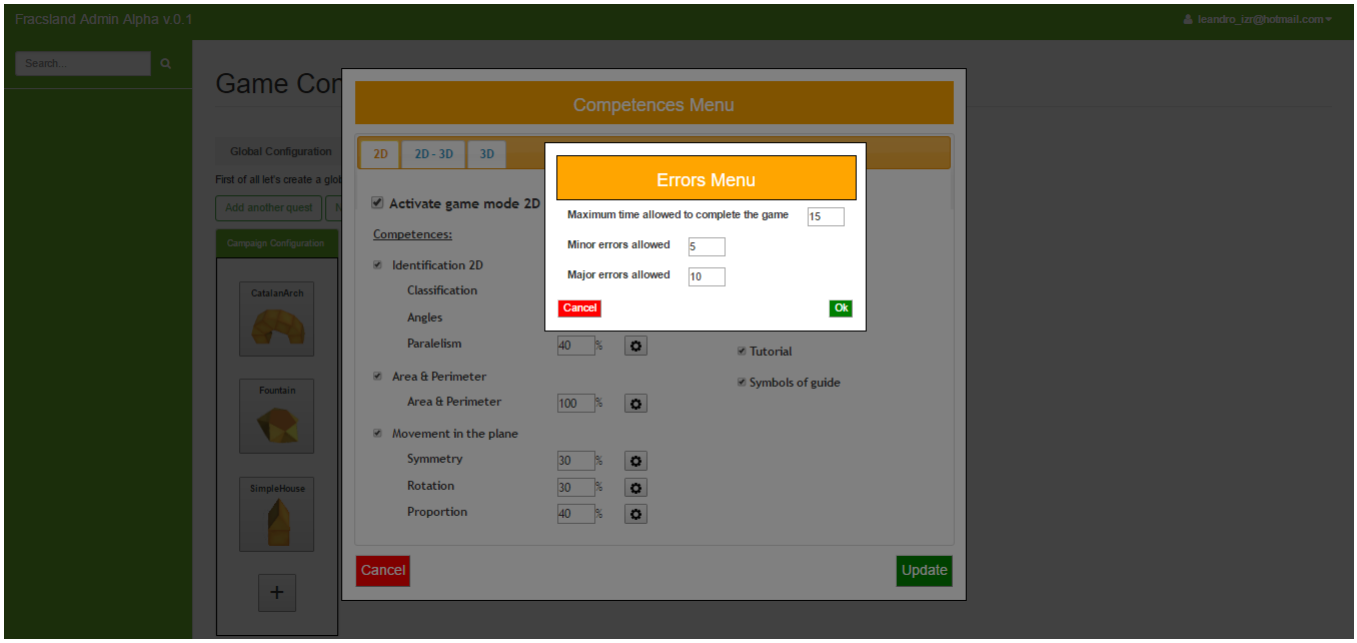

Figura 37: Página de configuración de los errores de una competencia en concreto.

6. Resultados y Simulaciones

A continuación se verá como ha quedado la planificación final del proyecto y seguidamente se presentan algunas simulaciones que se han hecho sobre el juego y el servidor.

6.1. Planificación final del proyecto

En la siguiente imagen 38 se puede observar cómo ha quedado la planificación final del proyecto. Para ello, se ha tomado la figura correspondiente a la planificación inicial (se mantiene el color azul para el tiempo que se había estipulado inicialmente) y se utiliza el color rojo para el tiempo que se ha ampliado y el morado para las zonas donde coincide el tiempo inicial con el final. Se recuerda que cada unidad de tiempo usada en el gráfico corresponde a una semana.

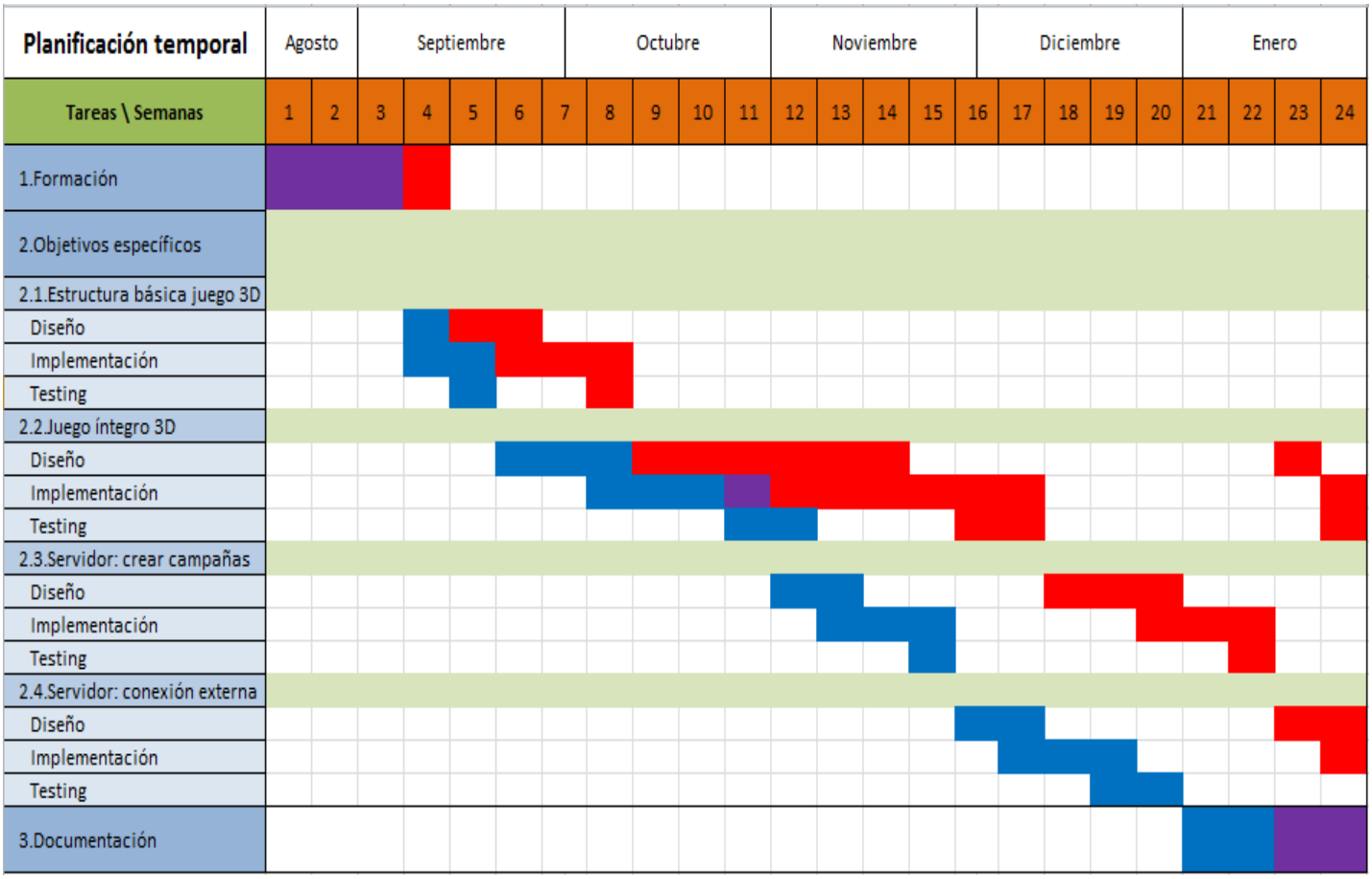

Figura 38: Planificación final del proyecto.

Se puede ver en el diagrama que se ha seguido prácticamente el mismo orden a la hora de realizar los objetivos respecto al orden inicial. Aunque, viendo a simple vista, se aprecia que debido al retraso de terminar un objetivo tal y como se había

planificado, hizo que los posteriores se retrasasen un poco más, haciendo como un efecto en cadena.

El problema que ocasionó este efecto fue que se dejó muy poco tiempo para la conexión del servidor con el juego por lo que no se pudo al final terminar completamente este objetivo.

Se puede ver que hubo solapamiento entre diseño e implementación, esto es debido que a medida que se iba diseñando también se iba implementando, creando pequeños prototipos tanto para el juego como el servidor, siguiendo un proceso cíclico e iterativo en el desarrollo del software.

En Enero, se volvió a continuar con el diseño e implementación del juego 3D aunque no era tanto para programar los algoritmos más complejos, sino más bien, mejorar la interfaz del juego en términos de usabilidad.

6.2. Resultado y simulación del estado final del juego 3D

Se ha desarrollado toda la parte del juego 3D y parcialmente la parte del servidor. La conexión con el servidor ha quedado incompleta .

La escena inicial del Geopieces (la pantalla de login) todavía no se ha modificado en este proyecto por lo cual, sigue estando la configuración inicial que le habían puesto en el proyecto anterior de Geopieces. Es decir, para que un usuario se pueda loguear, se necesita que el nombre de usuario introducido tenga al menos 3 caracteres y 8 para la contraseña. Una vez introducidos ambos datos, hay que pulsar el botón de “Sign Up” y a continuación, el de “Log In” (ver figura 39).

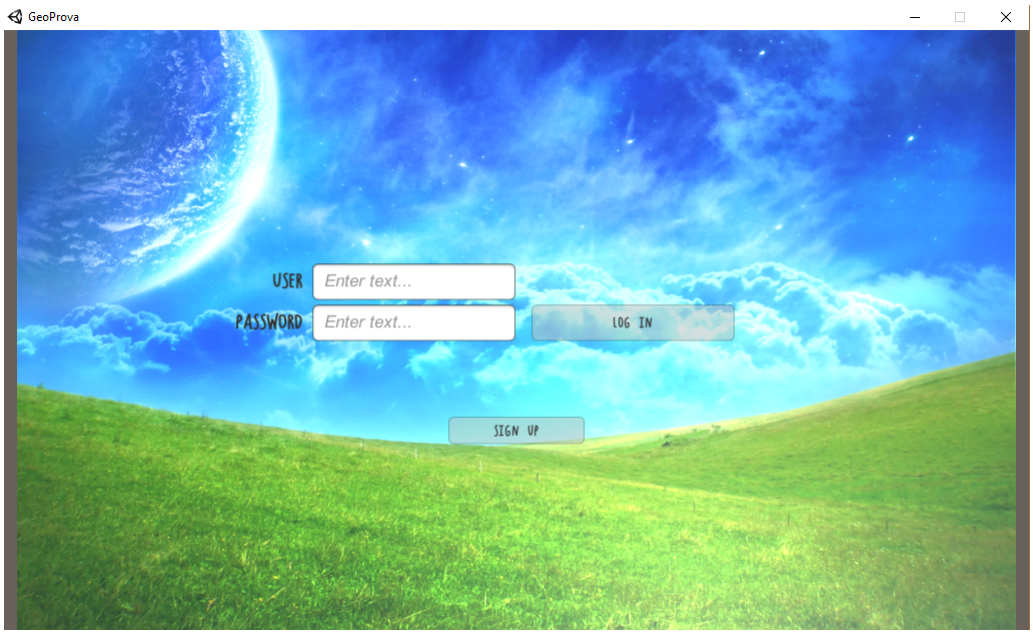

Figura 39: Escena de inicio del juego.

Una vez se ha pulsado el botón para loguearse, el sistema cargará la escena que contiene el menú del juego (ver figura 40).

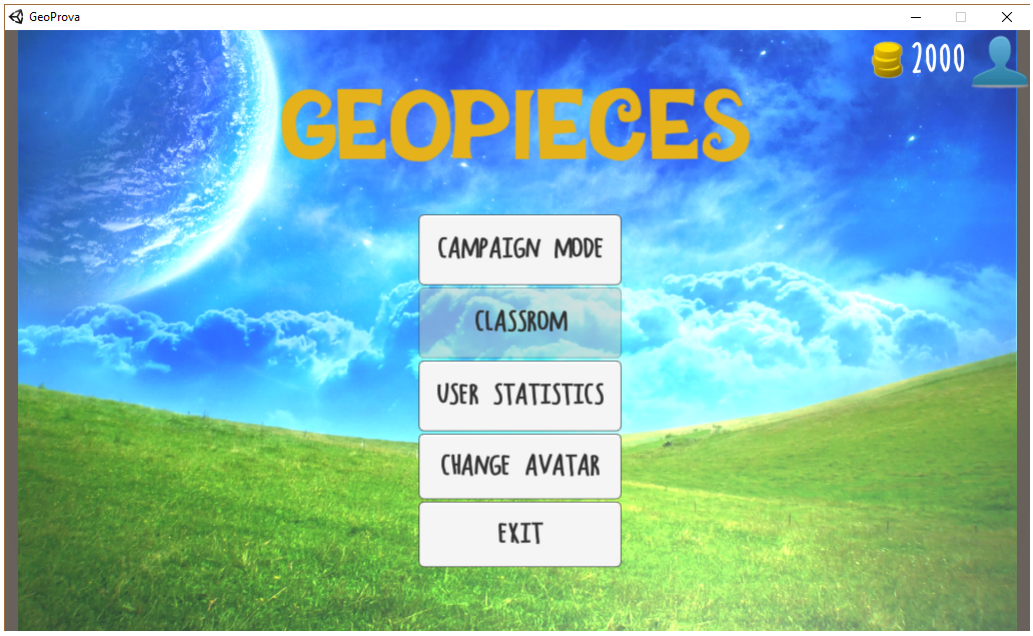

Figura 40: Escena del menú principal del juego.

Centrándonos en el apartado del juego 3D, si se selecciona el botón de las Campañas, se cargará la escena donde están las campañas a realizar (ver figura 41). Clicando en una de las campañas aparecen las actividades correspondientes al modo 2D, 2D-3D y 3D (ver figura 42).

Figura 41: Escena de las diversas campañas.

Figura 42: Escena de las actividades de una campaña.

Suponiendo ahora que ya se han completado todas las actividades del modo 2D y 2D-3D, se habilita un botón el cual te permitirá ir a la escena del juego 3D para conseguir la figura final de la campaña (ver figura 43).

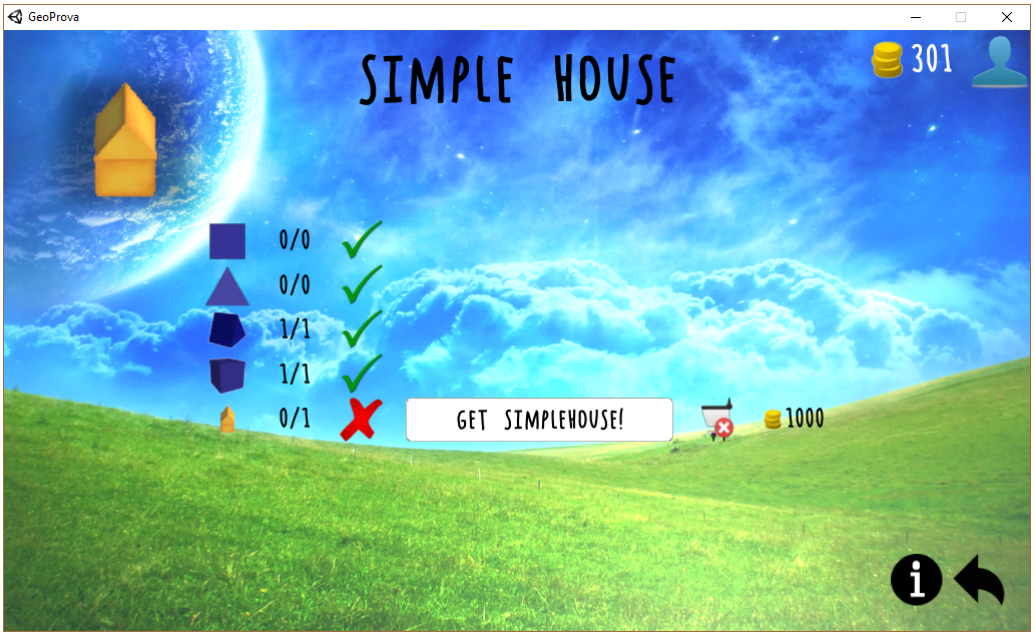

Figura 43: Escena del menú principal del juego.

Una vez se está en la escena del juego 3D, utilizando el inventario que se encuentra en la parte izquierda del juego, se podrán ir instanciando figuras simples (o complejas si después de crearlas se ha decidido guardarlas) y unirlas de forma que se consigue crear la figura objetivo (ver figura 44).

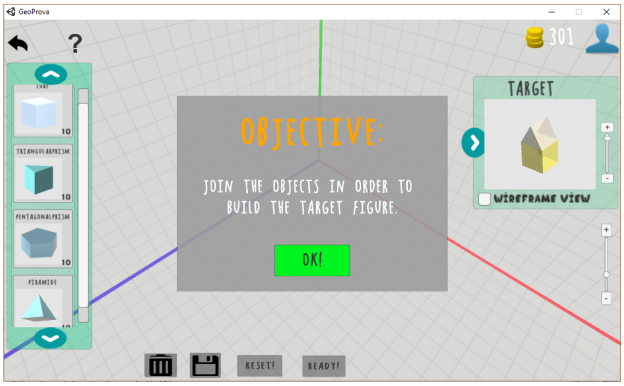

Figura 44: Escena inicial del juego 3D.

Se mostrará un ejemplo de cómo conseguir la casa simple.

- Primero se instancia un cubo y un prisma triangular.

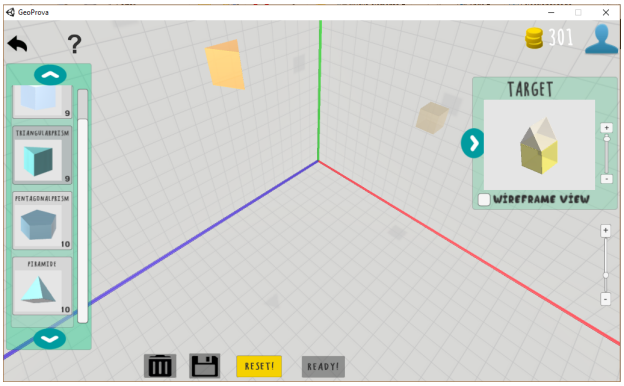

Figura 45: Se instancia un cubo y un prisma triangular.

- A continuación se desplaza el prisma triangular para acercarlo al cubo y se rota para adecuarlo.

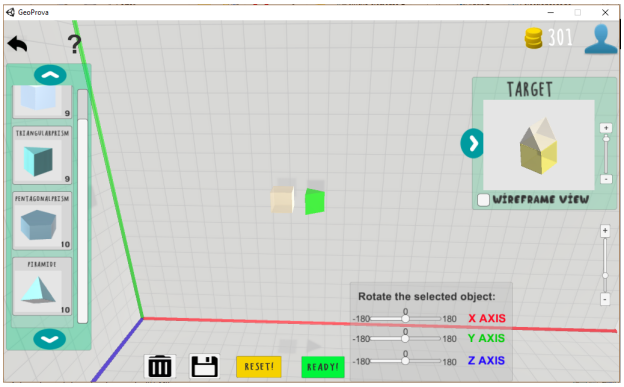

Figura 46: Se traslada y rota el prisma triangular.

- Se unen los dos objetos.

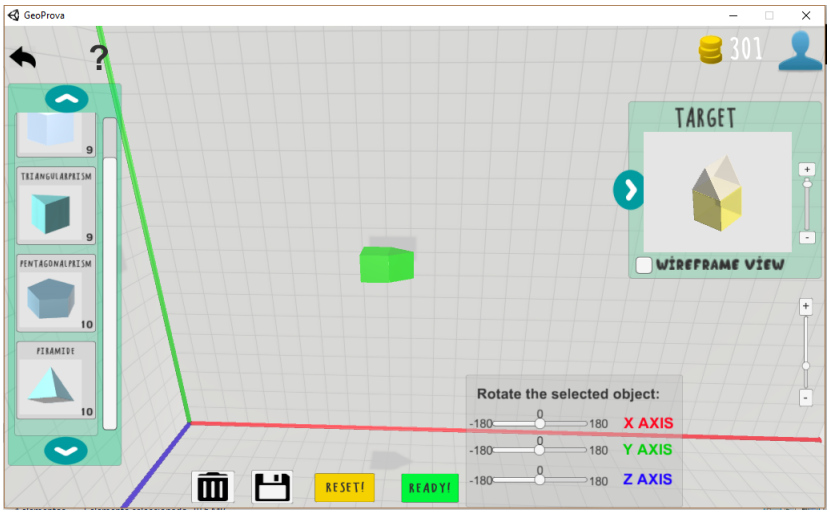

Figura 47: Se unen los dos objetos.

- Finalmente, se clica en el botón "Ready" el cual rotará el figura final creada para adecuarla a la rotación de la figura objetivo y comprobará si esas dos figuras son válidas.

  En este caso, saldrá que sí lo son y mostrará un mensaje de "Success" y permitirá volver a la escena anterior.

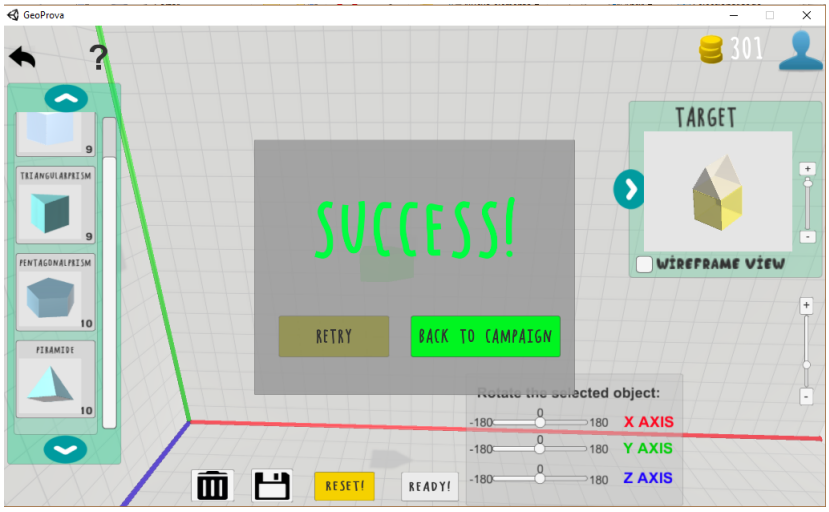

Figura 48: En esta escena se instancia un cubo y un prisma triangular.

- Si ahora se vuelve al menú anterior, se verá que sale un mensaje diciendo que la campaña está completada.

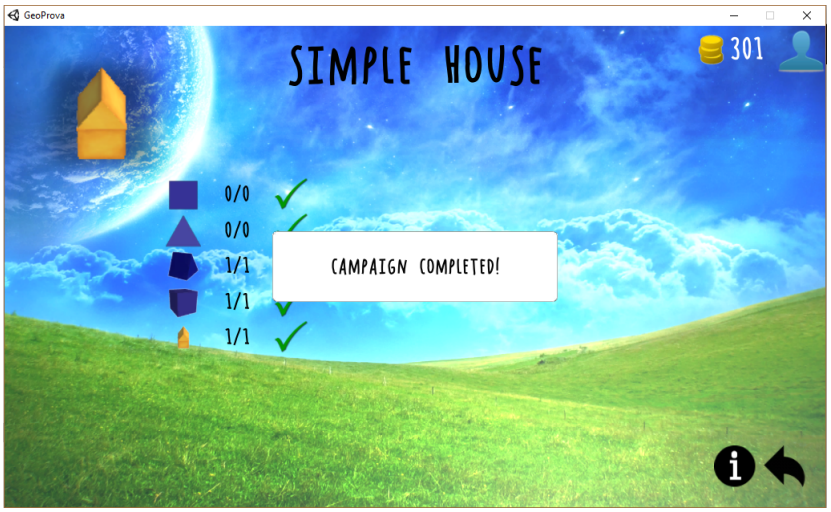

Figura 49: Escena de la campaña con sus actividades completadas.

- Y en el poblado se verá que se han actualizado todas las figuras correspondientes a la casa simple.

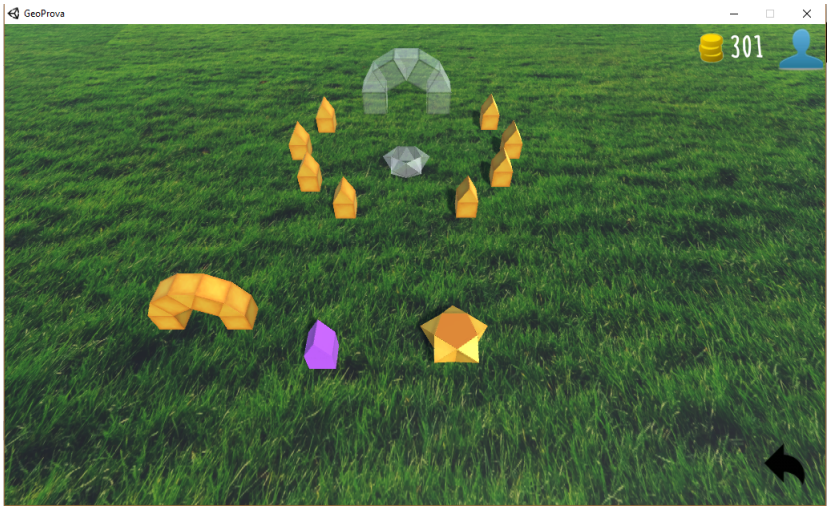

Figura 50: Escena del poblado.

6.3. Resultado y simulación del estado final del servidor

Del servidor, se ha podido diseñar e implementar completamente la ampliación de la base de datos, añadiendo toda la información necesaria para poder guardar los datos referentes al juego de Geopieces (los modos de juego: 2D, 2D-3D y 3D, las competencias y las ayudas que se quiera activar).

Ahora se hará como una simulación del servidor donde se irán explicando las variaciones que se hicieron y como han quedado finalmente. No se entrará en profundidad ya que en los apartados anteriores ya se ha explicado qué aspectos en concreto se han cambiado.

Como se ha dicho antes, el primer cambio corresponde a la página inicial (ver figura 51), ya que en la parte de debajo se encuentra el apartado de descargas, en el cual se ha añadido también el juego de Geopieces (ver figura 34).

Figura 51: Página de inicio del servidor.

A continuación, si el usuario decide loguearse o registrarse, se procede a una página introductoria (ver figura 52) y de aquí se lleva a otra en la cual se pide el nombre con el que crear una nueva clase (ver figura 53).

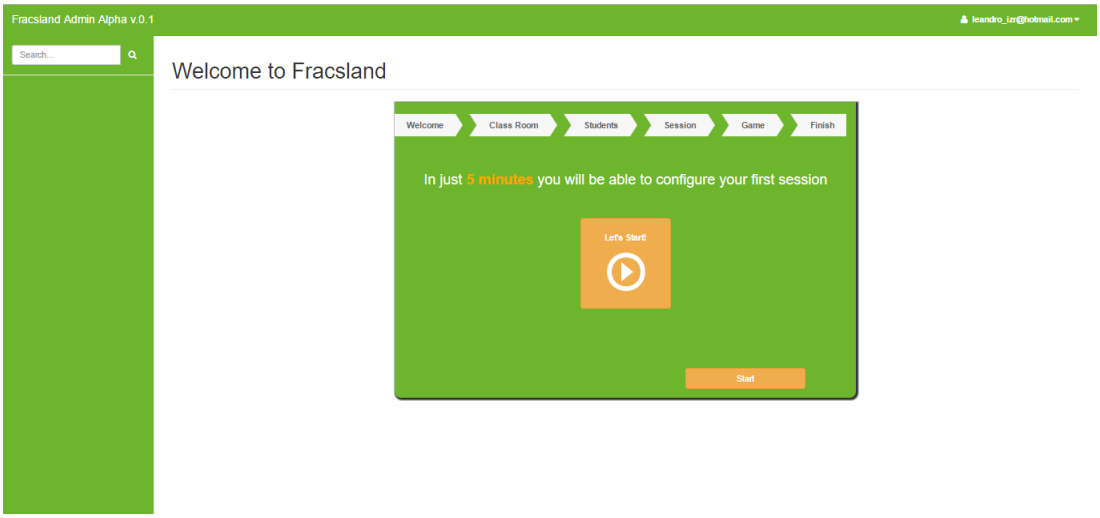

Figura 52: Página introductoria una vez el usuario se ha registrado.

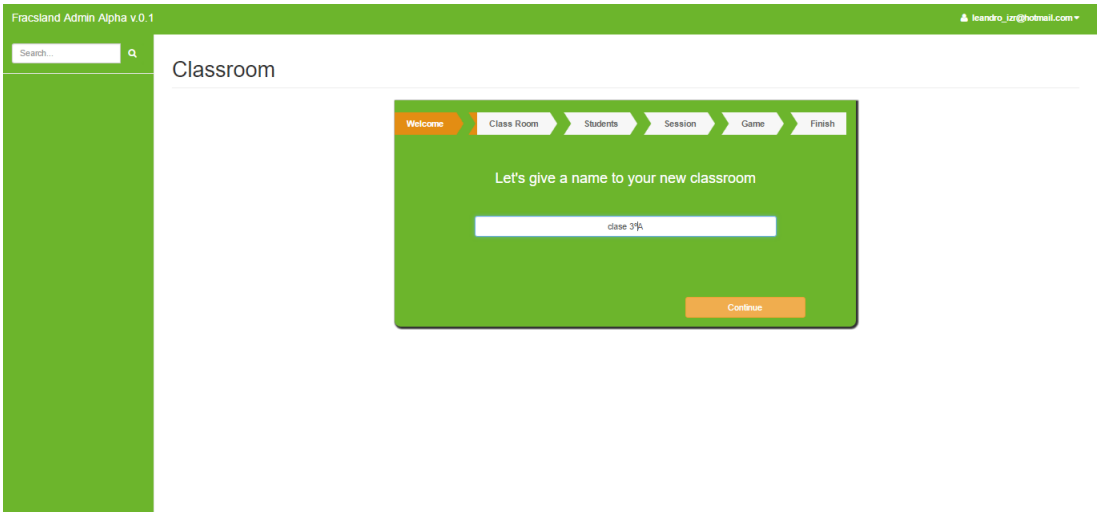

Figura 53: Página en la cual se solicita el nombre de la clase.

Lo primero será ponerle un nombre a la clase. Después aparecerá una página en la que se podrán entrar los datos de los alumnos vía Excel o manualmente (ver figura 54).

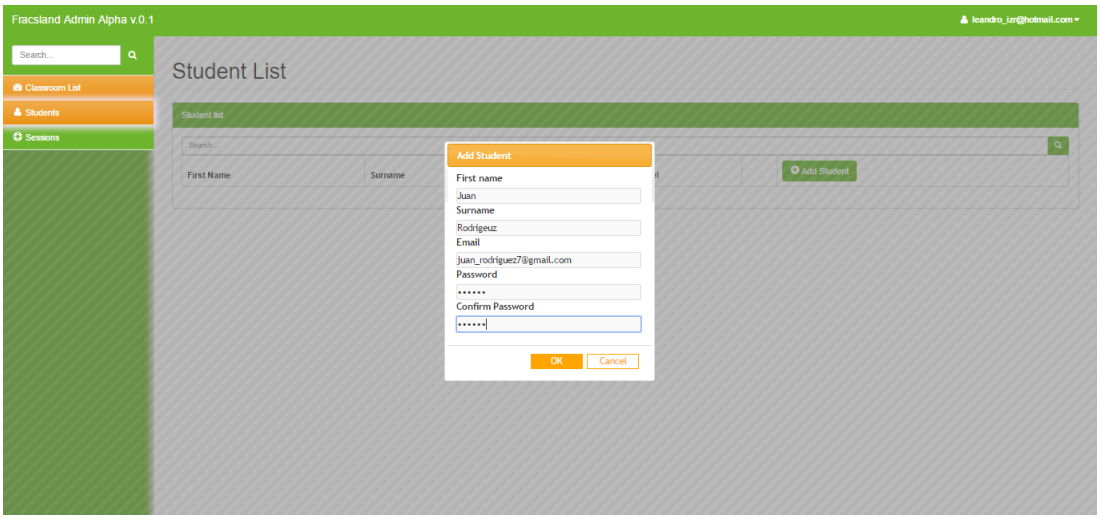

Figura 54: Escena de la campaña con sus actividades completadas.

Una vez se ha añadido, como mínimo, un alumno, se podrá elegir el tipo de juego que quieran jugar (ver figura 33). Obviamente para esta simulación, se elegirá el juego de Geopieces.

En este punto, se podrá configurar una partida de Geopieces de forma global, para toda la clase (ver figuras 35 36 37). Si se pulsa el botón de “next step” se podrá configurar la partida individualmente, personalizada a cada alumno.

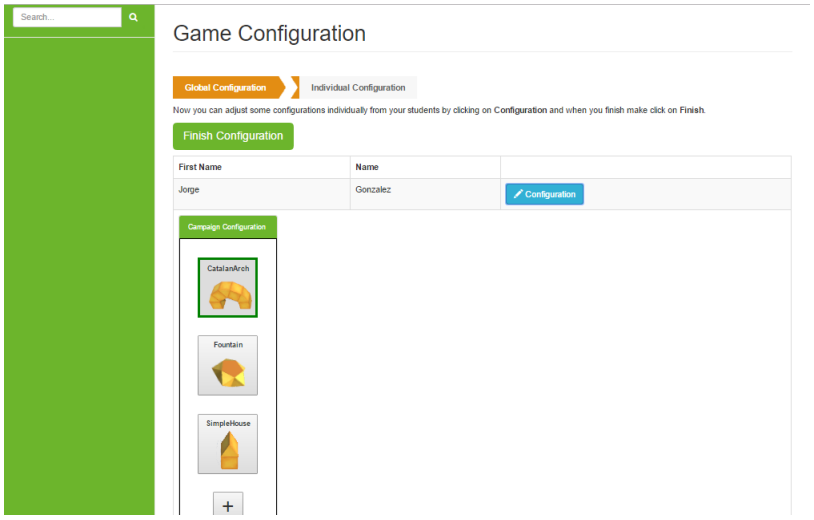

Figura 55: Página de configuración de la campaña de forma individualizada.

Por ultimo el profesor puede descargar el juego (lleva al usuario a la página inicial) (ver figura 34).

7. Conclusión

En este proyecto, se tenían dos objetivos principales:

- Ampliar el juego Geopieces, añadiéndole un juego serio sobre figuras 3D para el aprendizaje de alumnos de primaria sobre la geometría en el espacio.
- Ampliar el servidor para que el profesorado pueda configurar también el juego Geopieces.

Del primer punto, se ha podido completar prácticamente en su totalidad. Se ha conseguido crear un juego 3D en el cual se realiza la actividad basada en la competencia de composición, la más completa y de hecho en esta misma también se trabajan otras competencias como la identificación, la traslación o la rotación, por ejemplo.

Del segundo punto, se ha podido completar parcialmente. La base de datos ha sido ampliada, añadiéndole toda la información necesaria para poder configurar una partida del juego Geopieces.

También se ha implementado el mecanismo para que el profesor pueda configurar y crear varias partidas, aunque esta funcionalidad de momento se encuentra en fase beta.

El objetivo de conectar el servidor con el juego se ha conseguido parcialmente. Se ha podido avanzar bastantes funcionalidades. Ya se han creado los métodos que pueden convertir los datos de una partida en un archivo JSON por lo cual, con un único mensaje, ya se podrían pasar todos los datos y también se ha modificado el juego 3D de forma que el juego funciona dependiendo únicamente de la figura objetivo que se le pase.

Por estos motivos, aunque no se haya podido terminar la conexión del servidor con el juego, se ha podido avanzar bastante en su materia.

El trabajo realizado en este proyecto se considera muy satisfactorio, debido a que se ha completado en gran parte los objetivos propuestos inicialmente.

7.1. Trabajo futuro

En este apartado se presentan algunos posibles trabajos futuros que podrían implementarse en proyectos sucesivos:

Del juego 3D:

- Crear el resto de competencias del juego 3D de Geopieces a partir de la competencia más completa de Composición.

Del servidor:

- Se podría hacer que el profesor pudiese crear nuevos figuras objetivo. Para ello se podría coger el juego actual 3D y sería simplemente desactivar la opción de comprobación con la figura objetivo a la hora de calcular el algoritmo para determinar si dos objetos se pueden unir. En este caso se devolvería siempre 'True' y de esta forma se permitiría unir los objetos que se quisiesen mientras que las caras a unir estuvieran bien rotadas y fuesen del mismo tipo.
- Terminar la conexión del juego Geopieces con el servidor.
- Como la base de datos se ha ampliado, haciendo que las competencias del juego 3D sean tan genéricas e incluso se define que competencias se podrán configurar leyéndolas de un archivo JSON, se podría utilizar esta base de datos implementada para cualquier otro tipo de juego serio en el cual funcionase a partir de competencia y se quisiese utilizar herramientas de *authoring*.

Apéndice: Manual técnico

A.1 Instalación: Requerimientos mínimos y pasos a seguir

A.1.1 Requerimientos mínimos

Para poder instalar el servidor se requiere un mínimo de 197 Mb de memoria i 201 Mb de disco en un Pentium 2 o superior.

Los programas que se usarán para el servidor son estos dos:

- Eclipse: Se necesitará la última versión de java [24] (mínimo usar un Pentium 2. En este link se pueden ver con detalle cuáles son los requerimientos mínimos) y un intérprete de Python con varios paquetes que se mencionarán más adelante.
- Postgres: como la base de datos que se trata es pequeña no se requiere en verdad ningún requisito.

En cambio para instalar el juego Geopieces, como se ha utilizado un motor de juego (Unity) se requerirán unos requisitos superiores (los datos que aparecerán a continuación fueron sacados de la página de Unity [25]):

Para el desarrollo:

- **OS**: Windows 7 SP1+, 8, 10,; Mac OS X 10.8+. Windows XP y Vista no son compatibles. Las versiones del servidor de Windows y OS X no se probaron.
- **GPU**: Tarjeta gráfica con DX9 (modelo de shader 3.0) o DX11 con capacidades de funciones de nivel 9.3.

Para ejecutar juegos de Unity:

- Escritorio:
  - Sistema operativo: Windows XP SP2+, Mac OS X 10.8+, Ubuntu 12.04+, SteamOS+.
  - Tarjeta gráfica: DX9 (modelo de shader 3.0) o DX11 con capacidades de funciones de nivel 9.3.
  - CPU: compatible con el conjunto de instrucciones SSE2.
- El reproductor para iOS requerirá iOS 7.0 o superior (ya no 6.0).
- Android: OS 2.3.1 o posterior; CPU ARMv7 (Cortex) con tecnología NEON o CPU Atom; OpenGL ES 2.0 o posterior.

- WebGL: Cualquier versión de escritorio reciente de Firefox, Chrome, Edge o Safari
- Windows Phone: 8.1 o posterior.
- Windows Store Apps: 8.1 o posterior.

A.1.2 Instalación del entorno del juego Geopieces

Para poder programar el juego de Geopieces, se necesitará instalar el motor de juegos Unity3D y luego un programa para la edición del código: Monodevelop o Visual Studio.

Para instalar estos programar (al menos, para Windows que es donde se ha hecho el desarrollo de este proyecto):

1. Ir la página y seleccionar la versión de Unity 5.3.5: `https://unity3d.com/es/get-unity/download/archive`
2. Instalar clicando en el botón "Descargas (Win)":
   a. Editor Unity 64 bits: Se instala el editor de Unity y el editor de código.
   b. Activos estándar: ¡Importante! Instalar los módulos siguientes, de lo contrario el proyecto no funcionará: Seleccionar los módulos referentes a Window build, Linux build, Android build para poder compilar el código.

A.1.3 Instalación del entorno del servidor Fracsland

A continuación se detallarán los programas necesarios para poder instalar el servidor.

- Eclipse: `https://www.eclipse.org/downloads/download.php?file=/oomph/epp/neon/R1/eclipse-inst-win64.exe`
- Java 8: `http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2html` (obligatorio versión 8 para que funcione pydev).
- Pydev: Seguir el tutorial: `http://www.pydev.org/manual_101_install.html`

  En uno de los últimos pasos del tutorial cuando se va al apartado de instalar nuevo software, el link que hay que usar es `https://dl.bintray.com/fabioz/pydev/5.3.1`
- Postgres: `http://www.enterprisedb.com/products-services-training/pgdownload`

Ahora se necesitarán varios paquetes que hay que instalar de forma adicional a Python. Se recomienda encarecidamente usar la versión 2.7 ya que con esta versión se programó el servidor.

Si todavía no se tiene una versión de Python instalada, ir al siguiente enlace: `https://www.python.org/downloads/`

Para instalarle los paquetes se usará el sistema de gestión de paquetes: pip. Así pues se habrá de descargar pip `https://bootstrap.pypa.io/get-pip.py` (en Windows)

NOTA IMPORTANTE: pip instalará los paquetes sobre la versión de Python que se encuentre en el PYTHONPATH (variable de entorno del sistema). Por lo que, antes de instalar paquetes, hay que asegurarse de que la versión de Python que se ha instalado ponerla en el PYTHONPATH.

Ahora desde una consola de comandos, ejecutar las siguientes instrucciones:

- pip install Django==1.7.10
- pip install django-grappelli==2.7.3
- pip install openpyxl==2.2.6
- pip install psycopg2==2.6.1

Una vez se han instalado todos los componentes, se necesita terminar de configurar el servidor:

1. Importar el proyecto al Workspace (clicar sobre "file ¿import ¿General ¿Existing Projects into de Workspace"). Se busca donde esté situado Fracsland y se le da al botón "finish".
2. Lo más probable es que aparezca un mensaje diciendo que no se encuentra ningún intérprete de Python, por lo que hay que seleccionar "Manual Config".
3. En la lista de intérpretes de Python, crear uno nuevo, dándole un nombre y haciendo clic en "Browse..." navegar hasta encontrar el ejecutable de Python que se había bajado y el cual tiene los paquetes que se han instalado con el pip. A continuación, darle todo al botón de Ok y Apply.

Una vez llagados a este punto, hay que modificar el archivo *settings.py* para cambiar algunos parámetros.

1. Definir una SECRET_KEY (o pedirle una al coordinador del proyecto).
2. En el apartado de DATABASES, rellenar los datos necesarios.
3. Rellenar también los datos de EMAIL_HOST, y demás para que el servidor pueda enviar mails.

En el archivo `backoffice>migrations>0002_auto_20160605_1222.py` hay que definir un super usuario. Mirar la línea:

$User.objects.create_superuser('<nombreusuario>','<email>','<password>')$

También se debe acceder al pgAdmin donde habrá que crear una base de datos que tenga el mismo nombre y el mismo puerto que el que se definió en *settings.py*.

Por último, ejecutar la línea siguiente en una consola situada en la ruta del proyecto:

$$python \quad manage.py \quad migrate$$

para aplicar todas las migraciones que se hubiesen creado y llenar la base de datos con tablas definidas.

Una vez se han importado todas las migraciones, ya se puede ejecutar el servidor, haciendo clic en

$$RunAs > 1 : Pydev : Django.$$

A.2. Manual del desarrollador

A.2.1 Configuración del juego 3D

Configuración de un nuevo objeto 3D

Como se ha mencionado anteriormente, se utilizan como objetos simples el cubo, el prisma pentagonal, el prisma triangular y la pirámide. El algoritmo utilizado para detectar las caras cuando colisionan y el algoritmo principal del juego ambos se han implementado teniendo en cuenta que en un futuro se podría ampliar los objetos y por tanto se ha tratado a los objetos simples de forma genérica.

En esta sección se explicarán los pasos a realizar para crear un nuevo objeto simple.

1. Se crea un nuevo *GameObject* en el entorno de Unity3D que será el contenedor del resto de caras. Ponerle el tag "Object3D" y copiar todos los componentes que tenía por ejemplo el cubo (el cual se encuentra en la carpeta "Assets/-prefabs/Basic Objects/objects of faces" al gameobject que se acaba de crear.

2. Si el objeto contiene caras del tipo: cuadrado, triángulo o pentágono, copiar la cara definida en alguno de los otros objetos simples y pegarlo a dentro del *GameObject* tantas veces como el objeto simple que se quiera crear requiera de esa cara.

3. Si el objeto contiene caras que no pertenecen al punto anterior, con el programa Blender crear e importar la nueva cara.

    a Importante: Ajustar el tamaño de la nueva cara de forma que sus aristas puedan coincidir con la de las caras ya creadas.

b Añadirle un *Collider*, imitar la forma en que se pone un *Collider* a alguna de las caras ya creadas.

c Añadirle el resto de propiedades que tenga la cara, copiándolas de una cara que ya está creada a excepción del script y del tag.

d Para el script, habrá que crear uno nuevo de forma idéntica, por ejemplo de *AttributesSquare* y que herede de *AttributesFigure*. El nombre del script debe ser del estilo *"AttributesX"* donde X será el tag de esta cara.

e Si la figura no es regular, en esa clase habrá que definir cuál es el grado de rotación de la cara de forma que la figura sea invariante por rotación. Si no llegase a ser invariante por ningún ángulo, hay que ponerle 0.

f Un ejemplo del caso anterior sería: Se quiere crear un hexágono. Por lo que se crea el script *AttributesHexagon* y a la cara se le pone el tag "Hexagon". Un hexágono, es regular y por tanto usando la fórmula que se puede encontrar por ejemplo en *AttributesSquare*:

```
int n = 6;
float degreeRotation = 180.0f * (n - 2) / n;
```

4. Ajustar las posiciones y rotaciones de las caras de manera que formen el objeto simple deseado.

5. En el archivo Enum.cs añadir al enum RewardType el nombre del nuevo objeto simple.

6. En SceneController.cs realizar los siguientes cambios:

   (a) Crear una variable global private List¡GameObject¿newSimpleObjectTarget;

   (b) Modificar los siguientes métodos para incluir la nueva lista de objetos:

   I `initRegisterTargetObjects()`
   II `addObjectTargetList(Transform child)`
   III `checkReadyGame()`
   IV `GetListPossibleObjects(AttributesObject3D obj)`

Para crear una nueva campaña o editar alguna ya existente en la escena del menú de campañas, hay que modificar el archivo JSON

`Assets/Resources/Campaigns/campaigns.json`

De este JSON se destaca la parte del inventario que contiene todas las piezas que se necesitan para finalizar esta campaña. Por ejemplo se tendría el siguiente ítem.

{

`name:0,` //Objeto cuyo nombre corresponde al de RewardType en la

//posición "name".

`quantity:0,` // Cantidad que se tiene.

`campaignGoal:33,` // Cuántas misiones se quiere que el usuario haga.

`initialGoal:33,`

`type:0,` // "type" es el modo de juego.

`cost:[0,0,0,0,0,0,0]` // Objetos que se requieren tener previamente para

// formar el ítem.

}

Para más información sobre cómo se configura una campaña, mirar el TFG de Cebrián, S. y de Raussel D ([1],[2]).

A.2.2 Configuración del servidor

Para poder definir que competencias y que opciones tendrán, no se utilizará el panel de administración que había desarrollado Muriel C. en la primera versión del servidor [3].

No se utilizará porque se dejará el "Quest Configurator" para modificar Fracsland y para Geopieces se ha decidido hacerlo vía archivos JSON.

En la carpeta `FracServer/FracServer/init_data/` se puede encontrar una carpeta y un archivo. El archivo se llama `inventory.json` y contiene los datos del inventario del alumno. Mientras que dentro de la carpeta mencionada, se debe tener la definición de las campañas de cada objeto simple que se quiera poner por defecto.

Dentro de la carpeta `FracServer/media` se pueden encontrar dos carpetas: `prefabs` (contendrá los objetos de tipo prefab que se utilizan en el juego 3D) y `images` (las imágenes a las figuras objetivos de la carpeta prefab).

Modificación de la IP donde el juego (Fracsland) realiza las peticiones

Hay que tener en cuenta que cuando se cambie la IP del servidor, hay que cambiar en el archivo `Constants.cs` del juego de Fracsland de forma que se actualiza la variable `API_PRODUCTION_IP` con la nueva IP del servidor.

Apéndice B: Manual de usuario del juego

B.1. Manual del profesor

Para el profesor, ha de conectarse a la *url* del servidor a partir de la IP de donde se esté ejecutando el servidor.

En la sección de 6.3 del capítulo 6 se encuentra explicado un ejemplo de como configurar un juego.

B.2. Manual del estudiante

El juego se encuentra en la carpeta `FracServer/FracServer/static/GeopiecesGames/`. De ahí se podrá extraer el juego en la plataforma que se quiera.

Una vez descomprimido el juego, ejecutar el archivo Geopieces y se recomienda utilizar la resolución 1024x600 y desactivar la opción de 'windowed'.

En la sección de 6.2 del capítulo 6 se encuentra explicado un ejemplo de como jugar al juego.

Referencias

[1] Cebrián, S.: Geopieces: Joc seriòs per l'aprenentatge de la geometria. Part 2D-3D, 2016.

[2] Rausell, D.: Geopieces: Joc seriòs per l'aprenentatge de la geometria. Part 2D, 2016.

[3] Muriel, C.: Fracsland: Joc seriòs per aprendre fraccions, 2016.

[4] García Rodríguez, E.; González de Pablo, M.; Martínez Sánchez, A.; Nieto González, E.: El fracaso escolar, www.uam.es/personal_pdi/stmaria/resteban/Archivo/TrabajosDeClase/FracasoEscolar.pdf, 2017.

[5] Fernández Enguita, M.; Mena Martínez, L.; Riviere Gómez, J.: Fracaso y abandono escolar en España, obrasociallacaixa.org/documents/10280/240906/vol29_completo_es.pdf/d871a110-3e34-4c7f-aca7-6a9ae329caa2, 2017.

[6] Instituto Nacional de Estadística: 3.2 Abandono temprano de la educación-formación, www.ine.es/ss/Satellite?L=es_ES&c=INESeccion_C&cid=1259925480602&p=1254735110672&pagename=ProductosYServicios%2FPYSLayout, 2017.

[7] Margulis, L.; Bastos, A.; Szklanoy, C.; Zentner, J.: Juego serio, www.juegoserio.com/juego_serio.htm, 2017.

[8] Fernández Manjón, B.: Serious Games, es.slideshare.net/BaltasarFernandezManjon/serious-games-juegos-educativos-con-ejemplos-y-experimentos, 2017.

[9] Fernández Manjón, B.: Serious Games, Juegos educativos, es.slideshare.net/BaltasarFernandezManjon/serious-games-juegos-educativos, 2017.

[10] Paz, J. A.: Teorías pedagógicas para el aprendizaje de la geometría, sites.google.com/site/aprendamossobrelosangulos/assignments, 2017.

[11] Godino, J. D.; Ruíz, F.: Geometría y su didáctica para maestros, www.ugr.es/~jgodino/edumat-maestros/manual/4_Geometria.pdf, 2017.

[12] MARITO426: Teoría de las situaciones didácticas de Guy Brousseau, es.slideshare.net/MARITO426/teora-de-las-situaciones-didcticas-de-guy-brousseau 2017.

[13] WisWeb: Applets sobre juegos de geometría, www.fi.uu.nl/wisweb/applets/mainframe_en.html, 2017.

[14] Building with blocks: Applets sobre juegos de geometría, www.fisme.science.uu.nl/toepassingen/00724/, 2017.

[15] mundo primaria: Juegos y actividades: Figuras Geométricas para 1º primaria, www.mundoprimaria.com/juegos-matematicas/juegos-actividades-figuras-geometricas-primaria/, 2017.

[16] The land of Venn, www.thelandofvenn.com/, 2017.

[17] Geogebra, www.geogebra.org/, 2017.

[18] Django Project, www.djangoproject.com, 2017.

[19] StoryTec, www.storytec.de/index.php, 2017.

[20] e-Adventure, e-adventure.e-ucm.es, 2017.

[21] ITyStudio, www.itystudio.com/en/, 2017.

[22] What2Learn, www.what2learn.com/, 2017.

[23] Game Design Canvas, richardcarey.net/game-design-canvas/, 2017.

[24] Java 8: Requisitos mínimos, www.java.com/es/download/help/sysreq.xml, 2017.

[25] Unity: Requisitos mínimos, unity3d.com/es/unity/system-requirements, 2017.

[26] Garey M.R.; Johnson D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness, *W. H. Freeman Publisher*, 1979.