ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE VIDEOJUEGO EDUCATIVO PARA LA ASIGNATURA DE PROGRAMACIÓN AVANZADA

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN INGENIERÍA EN ELECTRÓNICA Y REDES DE INFORMACIÓN

CRISTHIAN DAVID IMBA BRAVO

EDISON AMBROCIO SAICO PÉREZ

DIRECTOR: RAÚL DAVID MEJÍA NAVARRETE, M.Sc.

Quito, enero 2021

AVAL

Certifico que el presente trabajo fue desarrollado por Cristhian David Imba Bravo y Edison Ambrocio Saico Pérez, bajo mi supervisión.

_____

RAÚL DAVID MEJÍA NAVARRETE, M. Sc
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Nosotros, Cristhian David Imba Bravo y Edison Ambrocio Saico Pérez, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejamos constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

CRISTHIAN DAVID IMBA BRAVO	EDISON AMBROCIO SAICO PÉREZ

DEDICATORIA

A mis padres Martha Bravo y Pedro Imba, a mis hermanos Bryan Imba y Steven Imba, a mi abuelita Marina Mora y a mi sobrino Gabriel Imba, por cuidarme, apoyarme, siempre preocuparse por mí y por brindarme todo su amor. Todos y cada uno de mis logros pasados, presentes y futuros están dedicados a ustedes.

Cristhian Imba

DEDICATORIA

A Dios porque me ha guiado con su sabiduría y compañía para poder sobrellevar una vida responsable lejos de mi familia.

Este trabajo se lo dedico enteramente a mis padres y mis hermanas por brindarme su apoyo completo para lograr esta meta. Por sus grandes consejos de enseñanza, de superación y de valores sembrados en mí, que me han forjado a ser una persona de bien.

También se lo dedico a mi persona, porque reconozco el interés y esfuerzo que he dedicado y nunca me dado por vencido hasta lograr mis objetivos.

Edison Saico

AGRADECIMIENTO

Agradezco a mis padres por apoyarme y confiar en todo momento, sin ustedes jamás hubiera llegado hasta este punto.

Agradezco a mi madre por tener tanta paciencia y amor para mí. Por ser mi mayor ejemplo en la vida.

Agradezco a mis hermanos y a mi sobrino, por ser un apoyo, por darme su cariño y por siempre estar junto a mis en los mejores y peores momentos.

Agradezco a mi compañero de este trabajo de titulación Edison Saico, quien puso todo su empeño, dedicación, paciencia y tiempo para alcanzar la finalización de este trabajo.

A todas las personas que en algún momento fueron parte de mi vida y aportaron toda esa buena vibra durante estos años. Sobre todo, a mis amigos del cole (Alfonso, JOCHA, Jorge, Henry, Daniel, Juan) que estuvieron ahí desde el prepo hasta el día de hoy.

A todos mis amigos que compartieron tiempo de estudio, esfuerzo y amistad en las aulas de la poli.

Agradezco a mi director y quien también fue mi profesor durante los últimos periodos académicos de mi carrera, David Mejía, M.Sc., quien dedico su tiempo, paciencia y esfuerzo para que este trabajo de titulación se concrete.

Cristhian Imba

AGRADECIMIENTO

Mi agradecimiento infinito a Dios por sembrar en mi tanta paciencia y amor, que con su bendición me ha ayudado a fortalecer lazos familiares, apreciar a nuestros semejantes y sobre todo amar la vida misma.

A mis padres Luis y Mariana por su gran apoyo incondicional, por sus consejos en cada etapa de mi vida, por su paciencia y confianza depositada en mi para lograr un sueño anhelado por todos.

A mis hermanas Yoli, Ligi, Moñi, Elsi, Luci, mi hermano Segundo y mi sobrina Maya quienes me han apoyado desde siempre, más que agradecido me siento en deuda con ellos por su tiempo dedicado en apoyarme y ayudarme a superar barreras presentadas en mi vida.

A mi director de este trabajo de titulación, David Mejía, M.Sc., por su compromiso y confianza prestada para desarrollar este trabajo que considero será de gran utilidad para muchos estudiantes.

A mi compañero de trabajo de titulación Cristhian Imba por su dedicación y compromiso.

A todos mis amigos y amigas de la carrera (Katy Liz, Zaidi Liz, Vicky Liz, Panchín, Cris, Marcus y Lucho) por su gran amistad y apoyo en momentos que más se los necesitaba, por el tiempo compartido en los trabajos, proyectos y en los laboratorios durante la carrera y por todo lo vivido en la poli.

Edison Saico

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

El presente trabajo de titulación presenta el desarrollo de un prototipo de videojuego educativo enfocado en la asignatura de Programación Avanzada, utilizando un enfoque de juego serio, el cual está compuesto por tres elementos principales: componente educativo, componente lúdico y base de datos. Además, el prototipo posee un componente adicional que permite realizar la gestión de usuarios y contenido educativo.

Este prototipo fue implementado utilizando el motor de desarrollo Unity junto con *Photon Unity Networking* (PUN). Mientras que, para la gestión del contenido educativo se emplea el gestor de base de datos MySQL Server.

Este documento se encuentra organizado de la siguiente manera:

En el primer capítulo se presentan los objetivos, el alcance e información teórica relacionada a los juegos serios y la metodología ágil de desarrollo SCRUM.

En el segundo capítulo, se presentan los resultados del análisis de requerimientos obtenidos durante la fase *Pre-game*; así también se incluye el diseño de la base de datos, historias de usuarios, *mockups* y la arquitectura del prototipo realizados en la fase *Game*. Por último, se muestra el proceso de implementación de la base de datos y los componentes administrativo, educativo y lúdico.

En el tercer capítulo, se muestran los resultados de las pruebas de conectividad y funcionamiento del prototipo, así como los resultados de las encuestas realizadas a estudiantes a fin de validar el correcto funcionamiento del prototipo y su aceptación.

En el cuarto capítulo se encuentran las conclusiones y recomendaciones obtenidas tras haber culminado con el desarrollo del prototipo.

Finalmente, en los Anexos se incluye: el contenido almacenado en la base de datos, los *scripts* empleados para la implementación de la base de datos, las historias de usuario, los *mockups,* el código fuente del prototipo desarrollado en Unity y los archivos que conforman la aplicación web.

PALABRAS CLAVE: Unity, *Photon Unity Networking*, MySQL, SCRUM, videojuego educativo.

ABSTRACT

The present project consists of the development of an educational video game prototype focused on the Advanced Programming subject using a serious game approach, which is composed of three main elements: educational component, playful component and database. In addition, the prototype has an additional component that allows the management of users and educational content.

This prototype was implemented using the Unity development engine in conjunction with Photon Unity Networking (PUN). While, for the management of educational content, the MySQL Server database manager is used.

This document is organized as follows:

The first chapter presents the objectives, scope and theoretical information related to serious games and the agile SCRUM development methodology.

In the second chapter, the results of the analysis of the requirements obtained during the Pre-game phase are presented; also includes the design of the database, user stories, mockups and the architecture of the prototype made in the Game phase. Lastly, the database implementation process and the administrative, educational and recreational components are shown.

In the third chapter, the results of the connectivity and operation tests of the prototype are shown, as well as the results of the surveys carried out with students in order to validate the correct operation of the prototype and its acceptance.

The fourth chapter contains the conclusions and recommendations obtained after having completed the development of the prototype.

Finally, the Annexes include: the content stored in the database, the scripts used for the implementation of the database, user stories, mockups, the source code of the prototype developed in Unity and the files that make up the web application.

KEYWORDS: Unity, Photon Unity Networking, MySQL, SCRUM, educational video game.

1. INTRODUCIÓN

En la actualidad dentro de las asignaturas que se ofrecen en la Facultad de Ingeniería Eléctrica y Electrónica (FIEE) de la Escuela Politécnica Nacional se observa que la utilización de herramientas educativas digitales es escasa. Programación Avanzada es una de las asignaturas de carácter obligatorio impartidas en las carreras de telecomunicaciones y tecnologías de la información. Dicha asignatura tiene una extensión considerable y manejo de conceptos complejos. Además, las horas destinadas para completar el plan micro curricular de la asignatura resultan, en ciertos casos, insuficientes para profundizar y reforzar conceptos importantes. Por estas razones, tanto docentes como los estudiantes buscan herramientas y técnicas que permitan optimizar el aprendizaje y el tiempo dedicado al estudio de esta asignatura.

En la FIEE se brindan recursos educativos digitales para los estudiantes y docentes, tales como simuladores, bibliotecas digitales, aulas virtuales; a estas opciones se las puede considerar como recursos educativos “tradicionales”. Pero durante los últimos años, las herramientas pedagógicas han ido evolucionando con la aparición de nuevas tecnologías [1], lo que ha dado origen a herramientas educativas de vanguardia que aplican conceptos como realidad virtual, realidad aumentada, “gamificación” (videojuegos), entre otros. Por otro lado, varios estudios han demostrado que la utilización de recursos pedagógicos en el aula por parte de los docentes favorece en los procesos educativos. Los videojuegos desarrollados para entornos educativos son un claro ejemplo de aplicación de herramientas educativas de vanguardia y también son apuestas recurrentes cuando se analizan las tendencias actuales y futuras de la educación [2].

Por lo mencionado anteriormente, en el presente Trabajo de Titulación se plantea la creación de una herramienta educativa digital, en este caso un videojuego, que utilice el enfoque de los “juegos serios”, es decir, un videojuego diseñado primordialmente con una finalidad educativa por encima del entretenimiento, que permita a los estudiantes reforzar conocimientos obtenidos durante las clases y familiarizarse con la asignatura.

Además, en el ámbito educativo, los videojuegos serios como recursos didácticos han demostrado tener un impacto positivo, esto debido a la capacidad que poseen para atraer a los estudiantes y hacerlos participar activamente de su aprendizaje [3]. Los videojuegos, en general, son ambientes diseñados específicamente para proporcionar experiencias interactivas a los jugadores, y para completarlos se requiere el mismo tipo de aprendizaje, estudio, comprensión y práctica requeridos para cualquier actividad educativa. Estas características muestran el potencial de los videojuegos para ser utilizados como

complemento a los recursos didácticos tradicionales. También, emplear videojuegos durante el proceso educativo puede despertar el interés de los estudiantes en este campo bastante redituable del desarrollo.

1.1. OBJETIVOS

El objetivo general de este Trabajo de Titulación es desarrollar un prototipo de videojuego educativo para la asignatura de Programación Avanzada.

Los objetivos específicos de este Trabajo de Titulación son:

- Analizar la literatura referente a juegos serios orientados a la educación y el sílabo de la asignatura de Programación Avanzada.
- Diseñar los componentes que el videojuego tendrá para cumplir con los requerimientos de la aplicación.
- Implementar los componentes del videojuego en base al diseño realizado.
- Analizar los datos obtenidos de las pruebas realizadas al videojuego.

1.2. ALCANCE

El videojuego será desarrollado en el motor de videojuegos Unity y funcionará como una aplicación web que utilice un modelo cliente-servidor, como se muestra en la Figura 1.1. El cliente utilizará un navegador web para acceder al videojuego. Mientras que, en el servidor se encontrarán alojados la base de datos, la aplicación web (videojuego) y el servidor web que son los componentes principales del sistema.

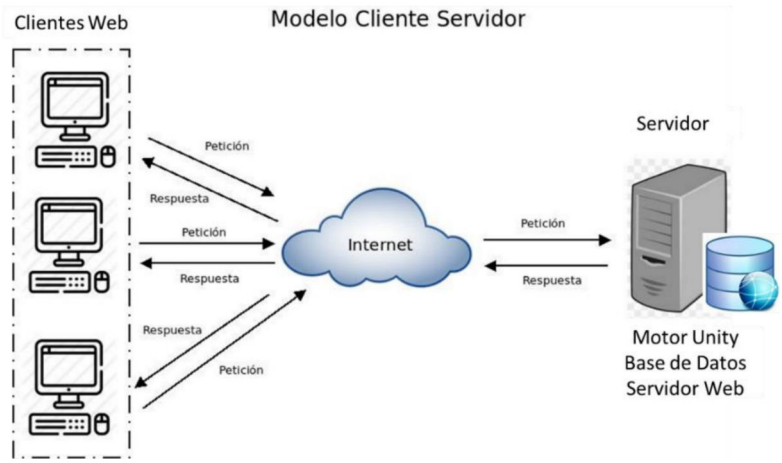

Figura 1.1 Modelo Cliente-Servidor [4]

Para el desarrollo del videojuego, en primer lugar, debe analizarse el sílabo de la asignatura de Programación Avanzada, que consta de siete capítulos que contienen aproximadamente treinta temas a tratarse durante el semestre. El resultado de este análisis

permitirá estructurar la base de datos que almacenará en sus registros preguntas, conceptos y ejemplos. Por cada capítulo de la asignatura se tendrán veinte preguntas, veinte conceptos y quince ejemplos (segmentos de código) que se obtendrán del análisis del silabo y servirán para el desarrollo del videojuego.

También, se analizarán tres ejemplos de juegos serios dentro del ámbito educativo que permitirán desarrollar los requerimientos para el videojuego planteado en este documento. Estos son *Codecombat*[1], *Kahoot!*[2] y Perobo[3] ,de los cuales se pueden apreciar escenas en la Figura 1.2, Figura 1.3 y Figura 1.4 respectivamente.

Figura 1.2 Escena del videojuego *Codecombat* [5]

Figura 1.3 Trivia generada en la plataforma *Kahoot!* [6]

[1] *Codecombat*: Videojuego que tiene como objetivo que los estudiantes aprendan ciencias de la computación y refuercen estos conceptos a través de la programación [5].
[2] *Kahoot!*: Juego serio cuya plataforma permite crear juegos basados en preguntas [6].
[3] Perobo: Juego que ayuda a introducir los conceptos básicos del lenguaje de programación C [7].

Figura 1.4. Escena de juego serio Perobo [7]

Para el diseño del videojuego se tomará como referencia la metodología ágil de desarrollo de software SCRUM, la cual será adaptada a las necesidades de diseño del videojuego. Esta metodología permitirá estructurar, planificar y controlar el proceso de diseño y desarrollo del videojuego.

El videojuego constará de dos componentes, el primero será el componente educativo que estará compuesto por cuatro minijuegos de estilo "*Quiz*". Estos serán: un minijuego de trivia, un minijuego de batalla naval, un minijuego del ahorcado y un minijuego de sopa de letras. Todos los minijuegos tendrán una interfaz donde el jugador podrá seleccionar un capítulo específico de la asignatura, lo que generará diferentes y variadas preguntas con cada partida, volviendo más dinámica la experiencia con estos minijuegos. Adicionalmente, los minijuegos poseerán una interfaz que permita alimentar la base de datos con nuevo contenido para mantener actualizada la información respecto al sílabo de la asignatura.

Dependiendo del minijuego, se utilizarán las preguntas, conceptos o ejemplos almacenados en la base de datos. El minijuego de trivia en cada partida mostrará al jugador siete preguntas, cada una con cuatro opciones de respuesta. En el minijuego del ahorcado se mostrarán cinco palabras o frases por cada partida, con cuatro intentos.

En el minijuego de sopa de letras se debe encontrar siete palabras por cada partida. Las palabras o frases que se utilizarán tanto en el minijuego del ahorcado como en el minijuego de sopa de letras estarán asociadas a los conceptos almacenados previamente en la base de datos.

Adicionalmente, de estos conceptos también se podrá extraer información adicional (*hint*) que será de ayuda para el jugador durante estos minijuegos.

Por otro lado, el minijuego de batalla naval presentará variantes con respecto al original; al momento de recibir un ataque, el jugador podrá defenderse y salvar sus naves respondiendo correctamente desafíos que consisten en analizar segmentos de código. Todos los minijuegos detallados anteriormente tendrán retroalimentación en caso de errar en una respuesta.

El segundo componente del videojuego será el componente lúdico. Este componente tendrá un sistema multijugador que soporte hasta cuatro jugadores. El juego será de tipo *Tower Defense*, donde cada jugador tendrá como objetivo defender un castillo de las oleadas de enemigos que se volverán más resistentes, aumentando sus puntos de vida con cada oleada superada.

Para defenderse, cada jugador podrá colocar torres de ataque a lo largo del camino hacia el castillo. Estas torres tendrán un costo, por lo que cada jugador iniciará con una cantidad de monedas que serán proporcionales a los puntos obtenidos en el componente educativo (diez por ciento de total de puntos). Se tendrán quince oleadas por cada mapa y se tendrán tres mapas diferentes para ser jugados.

Además, los usuarios del videojuego contarán con un *ranking*, es decir, acumularán puntaje que se obtendrá en cada minijuego del componente educativo con cada acierto. Este puntaje permitirá, mediante sistema de logros, desbloquear diferente contenido para el componente lúdico. El contenido que puede ser desbloqueado constará de cinco tipos de enemigos y cinco tipos de torres de defensa adicionales.

Finalmente, se realizarán pruebas de integración del software. Mientras que, para validar la funcionalidad del videojuego, se procederá a realizar pruebas del software con tres grupos de cuatro estudiantes que se encuentren cursando la materia de Programación Avanzada.

1.3. MARCO TEÓRICO

En este capítulo se analiza el concepto de juegos serios, la forma en la que son adaptados al campo educativo y los beneficios de su aplicación. Por otra parte, se describe aspectos relevantes sobre la metodología ágil para el desarrollo de software SCRUM y sus componentes. Realizando una adaptación de la metodología SCRUM y combinándola con el enfoque de los juegos serios, se estructura el marco de trabajo que permitirá desarrollar el videojuego educativo.

Además, se analiza el sílabo de la asignatura Programación Avanzada que permitirá extraer la información necesaria para alimentar al videojuego con conceptos, preguntas y ejemplos acerca de cada tema tratado. Finalmente, se muestra una breve descripción de las herramientas utilizadas para el desarrollo del videojuego.

1.3.1. JUEGOS SERIOS

El término juegos serios pareciera ser contradictorio, puesto que el vocablo "juego" representa diversión, alegría, fantasía y relax, se conciben como una acción que aleja de las cosas “serias” de la vida. Mientras que, el término “serios” alude a responsabilidad, sensatez, realidad y acciones con consecuencias a considerar [8].

Los juegos serios mantienen una dualidad, son juegos y no son juegos a la vez. Son juegos en la medida en que tienen reglas, simulan comportamientos, aceptan comentarios del jugador y brindan retroalimentación dentro del contexto de las reglas y comportamientos. Pero al decir que no son considerados como juegos, se hace referencia a que los tópicos tratados son considerados “serios” en la sociedad o dentro de la formación del jugador.

En la actualidad, se considera juegos serios a los videojuegos y simuladores en los que la educación es objetivo principal en lugar del entretenimiento. El enfoque de juegos serios para el área de desarrollo y creación de videojuegos ha surgido como una forma inteligente de combinar los beneficios de los videojuegos, su poder de penetración en la población con las necesidades que se presentan dentro del ámbito político, educativo, empresarial, comercial entre otros [9].

Los juegos serios al tener un propósito educativo explícito y cuidadosamente pensado, como se menciona anteriormente, no están destinados a ser jugados principalmente por diversión. Esto no significa que los juegos serios no sean entretenidos, agradables o divertidos. Sino que hay otro propósito o motivo adicional que supera al entretenimiento como prioridad del videojuego. También, los juegos serios ofrecen un nuevo mecanismo para la enseñanza y el entrenamiento al combinar videojuegos con educación. Los juegos serios pueden extender el valor aprender con libros al permitirle al jugador no solo aprender, sino también demostrar y aplicar lo que ha aprendido.

Normalmente, los diseñadores de juegos no son educadores y los educadores no son diseñadores de juegos. Sin embargo, al combinar las habilidades de los diseñadores de juegos con las de los educadores, los juegos serios pueden ser una fuerza en la enseñanza de estudiantes de todas las edades [10]. Los juegos serios se pueden usar para la educación en todos los niveles, desde preescolar y primaria, pasando por la secundaria, hasta colegios y universidades, e incluso en el mercado laboral. Un juego no tiene que

soportar todos esos niveles, pero algunos podrían hacerlo. Aun así, los juegos serios nunca reemplazarán a los maestros, pero tales juegos pueden estimular mucho interés y entusiasmo [11].

Por otra parte, los estudiantes de hoy tienen una visión completamente diferente de la tecnología. Han crecido rodeados por las nuevas tecnologías, por lo que exigen una constante actualización. La diversión es un factor crítico para este segmento de usuarios. Un juego aburrido, sea educativo o no, será evitado por los estudiantes. Sin embargo, la diversión es una valiosa herramienta para atraer a los usuarios [10].

1.3.1.1. Diferencias entre juegos de entretenimiento y juegos serios

Los juegos serios siguen siendo juegos y, por lo tanto, comparten muchas de las mismas consideraciones de diseño y problemas de desarrollo que otros videojuegos. Las técnicas que son usadas por los diseñadores en los videojuegos de entretenimiento son muy similares para el desarrollo de un juego serio. Los desarrolladores de juegos pueden tener una ventaja competitiva porque muchos de los procesos y la tecnología son similares. Sin embargo, existen diferencias significativas, causadas por el cambio en la meta del entretenimiento a la educación.

Por ejemplo, una de las principales diferencias hace referencia a la simulación de efectos y procesos del mundo real. En un videojuego de entretenimiento, el efecto o proceso solo necesita estar "lo suficientemente cerca" para aproximarse al resultado deseado y, a su vez, preserve la diversión.

Sin embargo, en un juego serio, tener precisión en la simulación de eventos del mundo real tiene una importancia abrumadora. En el ejército o en la medicina, esto resulta indispensable.

Los juegos serios de simulación de entrenamiento para respuesta a una emergencia o cualquier otro caso donde las vidas se ven afectadas son un claro ejemplo de situaciones donde la precisión en la realidad triunfa sobre la diversión [10]. Un ejemplo de este tipo de videojuegos se muestra en la Figura 1.5.

Entre las características distintivas de este tipo de videojuegos con relación a los videojuegos de entretenimiento se encuentran [9]:

- Están destinados para la educación, el entrenamiento en habilidades determinadas, la comprensión de procesos complejos, sean sociales, políticos, económicos o religiosos; también para publicitar productos y servicios.

- Están vinculados en forma evidente con algún aspecto de la realidad. Esto favorece la identificación del jugador con el área de la realidad que se está representando en el ambiente virtual.
- Constituyen un ambiente tridimensional virtual el cual permite realizar prácticas de manera "segura" a los aprendices en algunas áreas. Por ejemplo, en los casos de entrenamiento en el campo militar, los pilotos de combate pueden manipular los paneles de control y armamento de las aeronaves sin mayores riesgos.
- Hay intereses manifiestos en sus contenidos (políticos, económicos, psicológicos, etc.)

Figura 1.5 Escena del videojuego de entrenamiento de bomberos *Hazmat: Hotzone* [12]

También debe mencionarse como diferencia con los juegos de entretenimiento, que en los juegos serios debe ser considerado si el juego realmente le está enseñando al jugador y qué le está enseñando. Si el jugador aprende a vencer el juego, pero no puede aplicar lo que ha aprendido en el mundo real, entonces el juego serio ha fallado en su misión educativa.

A diferencia de los juegos de entretenimiento, los juegos serios también pueden integrarse fácilmente en los planes de lecciones existentes y el material del curso porque también son de ayuda para la enseñanza, pero esto no quiere decir que sean sustitutos de maestros y formadores. Por este motivo, los diseñadores y desarrolladores deben asegurarse de que los juegos serios sean adaptables a una variedad de situaciones dentro de las aulas y del entrenamiento, además de ser fáciles de usar [10].

1.3.1.2. Tipos de juegos serios

Al igual que se pueden encontrar varias definiciones de juegos serios, también hay varios intentos diferentes de clasificar juegos serios en géneros o tipologías similares. Los criterios utilizados para clasificar los juegos varían en gran medida, pero para realizar esta clasificación han sido considerados solo los juegos existentes que han sido etiquetados como "serios" por sus desarrolladores, en bases de datos específicos o en sitios web de revisión. En general, fueron revisados un total de 612 juegos y generaron un sistema de clasificación a partir de las descripciones que estaban disponibles para estos juegos [13]. Mediante la revisión de expertos y un análisis iterativo llegaron a un sistema de clasificación que consta de cuatro dimensiones:

- Contenido educativo primario
- Principio de aprendizaje primario
- Grupo de edad objetivo
- Plataforma

En general, esta clasificación ofrece una descripción bastante detallada de los tipos de juegos serios existentes que sus desarrolladores han etiquetado y comercializado como tales.

1.3.2. VIDEOJUEGOS EN LA EDUCACIÓN

Los videojuegos ofrecen un enfoque de aprendizaje flexible, no lineal y dirigido por el alumno. Estas características están llevando a los videojuegos a convertirse en parte del proceso educativo.

Sin embargo, no importa que tan relevantes se vuelvan los juegos serios, estos no reemplazarán a los maestros, profesores y otros facilitadores educativos. En cambio, estos juegos serán parte de las herramientas educativas que tengan a disposición [14].

Los juegos en general, y los videojuegos en particular, se encuentran actualmente en un proceso de escrutinio que intenta demostrar su efectividad como herramientas de capacitación y enseñanza.

Gradualmente, la aceptación de los juegos como otra herramienta educativa está creciendo, así como en el pasado ocurrió con el Internet o incluso las computadoras.

Sin embargo, los videojuegos aún son observados con recelo por los actores dentro del ámbito educativo. Existen muchas dudas sobre el tema, pudiendo mencionarse entre las principales la utilidad de lo que se está enseñando a los usuarios y qué tan bien enseñan los videojuegos en comparación con los métodos más tradicionales [10].

Aunque los videojuegos han avanzado en el aula, la llegada de los videojuegos a las aulas dependerá significativamente de los maestros y los administradores escolares. La clave para que los videojuegos sean aceptados por los maestros son las pruebas concretas, respaldadas por una investigación sólida, de que los juegos realmente hacen un mejor trabajo que los métodos actuales. Con la posible adopción de videojuegos en el aula, los maestros pueden adaptarse a un nuevo estilo de aprendizaje y preparar mejor a los estudiantes para el mundo moderno. Los juegos se muestran como herramientas de enseñanza eficaces, y los desarrolladores están trabajando con los maestros para integrar esos juegos en el aula.

Además, los estilos de aprendizaje desarrollados a partir de los videojuegos son muy diferentes de los que se adquieren en las aulas tradicionales. Un estudio del Instituto de ciencias del comportamiento en Alexandria, Virginia, encontró que las tasas de retención de aprendizaje aumentan de 75 a 80 por ciento cuando se atiende al estilo de aprendizaje de los jugadores, en comparación con la tasa de retención de aprendizaje del 5 por ciento de la enseñanza basada en conferencias [10].

1.3.2.1. Ejemplos de videojuegos aplicados a la educación

La empresa de desarrollo de videojuegos Monte Cristo[4] creó títulos de simulación de gestión, incluidos *Wall Street Trader*[5], *Start-up*[6] y *Airport Tycoon*[7] como entretenimiento, aprovechando el desarrollo financiado por el editor para crear una serie de juegos de simulación.

A pesar de estar diseñados como entretenimiento, los estudiantes universitarios, especialmente aquellos en la escuela de negocios, han encontrado que los productos de Monte Cristo son útiles para la capacitación. Juegos como *Zoo Tycoon*[8] y *Dance Dance Revolution*[9] se están utilizando en algunas escuelas. *Dance Dance Revolution*, por ejemplo, se ha utilizado en clases de educación física en escuelas de los Estados Unidos, el Reino Unido y Europa como se muestra en la Figura 1.6.

A medida que se conoce el potencial de los juegos serios, es probable que más juegos minoristas incluyan características y material de soporte para que sean usados más fácilmente en el aula [15].

---

[4] Monte Cristo: Extinta empresa francesa desarrolladora y distribuidora de videojuegos.
[5] *Wall Street Trader*: Videojuego que brinda una idea acerca del comercio en la bolsa de valores.
[6] *Start-up*: Videojuego en el que los jugadores deben intentar construir una empresa exitosa.
[7] *Airport Tycoon*: Videojuego que permite al jugador diseñar su propia compañía aérea.
[8] *Zoo Tycoon*: Serie de videojuegos de simulación empresarial que permiten al jugador construir y administrar zoológicos.
[9] *Dance Dance Revolution*: Serie de videojuegos simuladores de baile producida por Konami.

1.3.2.2. Problemas que enfrenta el desarrollo de juegos serios

Los principales problemas de desarrollo de los juegos serios para la educación son los presupuestos pequeños y la amplia gama de hardware y software posibles que se admitirán. La necesidad de trabajar en estrecha colaboración con los educadores, por su experiencia y porque son una parte necesaria de la experiencia educacional, también podría considerarse una limitante para el desarrollo.

Figura 1.6 Escuela de Washington usando *Dance Dance Revolution* en gimnasia [16]

Colaboración con educadores

Como la mayoría de las personas fuera de la industria minorista de videojuegos, muchos maestros tendrán poca o ninguna información o experiencia con el desarrollo de juegos. Los desarrolladores de juegos deberán ayudarlos a comprender el proceso. Es posible que tengan que explicar qué es posible y qué no, especialmente con los presupuestos limitados disponibles [10].

Presupuestos pequeños

Los editores minoristas de videojuegos tienden a no estar interesados en los juegos educativos. Los editores no están interesados en vender a un mercado pequeño, quieren vender millones de copias. Las condiciones actuales limitan severamente la cantidad de financiamiento disponible para juegos orientados a la educación general.

Incluso las organizaciones muy dispuestas a desarrollar este tipo de videojuegos a menudo tienen solo una fracción del presupuesto típico de videojuegos disponible. Los desarrolladores deberán ser extremadamente conscientes de los costos y centrarse en

aprovechar los recursos. Sin embargo, la industria de los videojuegos siempre ha estado centrada en las ganancias. Los juegos serios para el mercado militar y otras agencias gubernamentales vienen con un sistema de financiación incorporado. El gobierno no tiene problemas para proporcionar los fondos para mejorar la capacitación de sus tropas o para estimular la investigación y el desarrollo de nuevas tecnologías.

Del mismo modo, las empresas están dispuestas a gastar dinero para obtener un mejor retorno de sus inversiones en capacitación. Sin embargo, las escuelas, especialmente las escuelas públicas, rara vez tienen los ingresos o los fondos adicionales necesarios para invertir en el desarrollo de un nuevo software [10].

Hardware y software

La gama de sistemas informáticos en la educación es bastante amplia. Hay computadoras de escritorio y portátiles e incluso dispositivos de mano que abarcan varias generaciones de software y hardware. Además, los equipos no siempre tendrán versiones de la misma plataforma, sino que pueden ser una combinación de computadoras viejas y nuevas de Apple, así como PC basadas en Windows o Linux. Esto pone énfasis en apoyar el desarrollo multiplataforma para llegar a un segmento de mercado más amplio [10].

1.3.3. METODOLOGÍA ÁGIL SCRUM

SCRUM fue desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Es una metodología ágil utilizada para controlar y gerenciar el desarrollo de un producto software, es iterativa e incremental, cada ciclo o iteración termina con una pieza de software ejecutable que incorpora nueva funcionalidad.

Las iteraciones en general tienen una duración entre dos y cuatro semanas. SCRUM se utiliza como marco para otras prácticas de ingeniería de software como RUP[10] o XP[11] [17].

Está orientada a la comunicación y a los resultados, disciplina y una rápida producción de versiones visibles del software, de manera que se puedan hacer las evaluaciones cualitativas que exigen una visión e interacción del producto. El trabajo está estructurado en ciclos conocidos como *sprints*.

Durante cada *sprint* los equipos toman los requerimientos de una lista ordenada por prioridades conocidas como historias de usuario. Al terminar cada *sprint*, se tiene una versión potencialmente final del producto [18].

---

[10] RUP (*Rational Unified Process*): Metodología que utiliza el enfoque orientado a objetos para el diseño combinando UML (*Unified Modeling Language*).

[11] XP (*Extreme Programming*): Metodología ágil basada en la comunicación, la reutilización del código y la realimentación.

1.3.3.1. Sprint

El corazón de SCRUM es el *sprint*, es un intervalo de tiempo de un mes o menos, durante el cual se crea un incremento de producto, utilizable y potencialmente desplegable. Cada nuevo *sprint* comienza inmediatamente después de la finalización del *sprint* previo. Durante un *sprint* no se realizan cambios que puedan afectar al objetivo que se persigue durante este intervalo de tiempo. A medida que los *sprints* se van generando, el alcance puede ser clarificado y renegociado entre el dueño del producto y el equipo de desarrollo [19].

1.3.3.2. Conceptos adicionales de SCRUM

- El *sprint planning meeting* o reunión de planificación de *sprint,* es una reunión en la cual se crea el plan de trabajo que se va a realizar durante un *sprint*. Para *sprints* de corta duración (15 días o menos), el tiempo que se destina a estas reuniones suele ser menor a 4 horas.
- Un *sprint goal* es una meta establecida para el *sprint* que puede ser alcanzada mediante la implementación de la Lista de Producto (*Product Backlog*). Proporciona una guía para los desarrolladores para acercarse al producto entregable del *sprint*.
- Un *sprint review* es una reunión que se lleva a cabo al final del sprint, en el cual se presentan los avances del producto con el objetivo de facilitar la retroalimentación. Durante la revisión de un *sprint*, los desarrolladores y los usuarios interesados del software colaboran para determinar las siguientes cosas que podrían hacerse para optimizar el valor.
- El *product backlog* o lista de producto es una lista ordenada que contiene todo lo que podría ser necesario para completar el producto. La lista de producto evoluciona de acuerdo con el avance del producto software. La lista de producto cambia constantemente para identificar los requerimientos del producto para ser adecuado, competitivo y útil. La lista de producto enumera todas las características, funcionalidades, requisitos, mejoras y correcciones que constituyen cambios a ser hechos sobre el producto para entregas futuras.
- El *sprint backlog* o lista de pendientes es un conjunto de elementos seleccionados de la lista de producto para el *sprint* que incluye un plan con un nivel de detalle suficiente para completar los requisitos de la iteración.

1.3.3.3. Características

Como principales características que posee la metodología ágil SCRUM podemos mencionar que [20]:

- Posee una gran flexibilidad.

- Emplea el modelo de construcción incremental basado en iteraciones y revisiones.
- Su prioridad es conseguir la satisfacción del cliente mediante continua interacción entre éste y el equipo de desarrolladores.
- Los requisitos para el desarrollo del software pueden ser cambiantes.
- Se enfoca en conseguir pequeños incrementos de software que sean completamente funcionales.
- Está orientado a las personas, más que a los procesos.

Entre las características de SCRUM mencionadas se destaca que esta metodología no indica prácticas específicas a seguir durante el desarrollo, lo que brinda flexibilidad y permite ajustar el proceso a la realidad y forma de trabajo de cada proyecto, así como a los diferentes requerimientos de los clientes [21].

1.3.3.4. Metodología SCRUM aplicada a los videojuegos

Las metodologías utilizadas para el desarrollo de videojuegos siguen principios ágiles por ser iterativas e incrementales, tener interacción frecuente con el cliente y ser flexibles ante los requerimientos cambiantes. Además, las decisiones se toman en base a la experiencia, sin existir un proceso definido ni técnicas específicas a seguir.

En promedio, cada proyecto lo realizan de tres a cuatro personas que cubren los roles de productor, programador, artista gráfico, diseñador de juego y tal vez hasta artista sonoro. Aunque, tareas como diseño gráfico o sonorización del videojuego son realizadas habitualmente por empresas externas especializadas. La tendencia a utilizar metodologías ágiles para videojuegos tomó fuerza en los últimos años por existir varios casos de empresas en la industria que lograron adaptar estas metodologías a sus proyectos de manera exitosa. A pesar de esto, ninguna de estas adaptaciones está especificada formalmente.

Por otra parte, la metodología SCRUM puede ser adaptada para el desarrollo de un videojuego con la estructura en tres fases denominadas *pre-game, game y post-game* [22] que se muestra en la Figura 1.7. Durante el desarrollo de videojuegos es común que se separen estas fases para organizar y controlar el tiempo que va a ocupar cada una. Las fases deben ser revisadas por el equipo de desarrolladores para comprobar que todos los requisitos se cumplen, y en tal caso comenzar una nueva fase o iteración [19].

Pre-game

Durante el *pre-game* se definen las características y funcionalidades en base a los requerimientos impuestos por los clientes o realizando diferentes estudios para obtener los requerimientos para el desarrollo. También se analiza el sistema a construir, se define la

arquitectura y se realiza un diseño de alto nivel de la solución. Esta etapa tiene como objetivo principal planificar las tareas que se realizarán durante el desarrollo del software. Para ello es necesario definir el cronograma del proyecto junto con sus principales hitos, definir las tareas para la fase llamada *game* de acuerdo con las necesidades técnicas del proyecto y especificar el videojuego.

Game

La fase de *game* consta de iteraciones, que duran de dos a seis semanas, donde se desarrollan las características del producto. Al comienzo de cada una se realiza su planificación, donde se describen, priorizan y estiman las características que se van a desarrollar y al concluir se evalúa su resultado.

El objetivo de esta fase es implementar el videojuego. Para ello se trabaja en forma iterativa e incremental para lograr una versión ejecutable del videojuego al finalizar cada iteración.

Post-game

El *post-game* es el cierre del proyecto, donde se prepara la liberación del producto, se verifican las versiones a entregar y se realiza la documentación final. Esta fase tiene como objetivos evaluar y ajustar distintos aspectos del videojuego como por ejemplo jugabilidad, diversión, aprendizaje, además de eliminar la mayor cantidad de errores detectados [21].

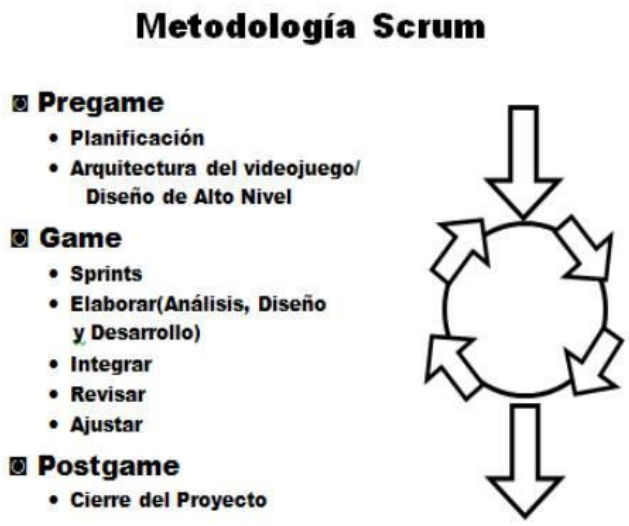

Figura 1.7 Fases de SCRUM para videojuegos [23]

1.3.4. SÍLABO DE LA ASIGNATURA PROGRAMACIÓN AVANZADA

El sílabo de la asignatura Programación Avanzada dictada en la carrera de Ingeniería en Tecnologías de la Información de la Facultad de Ingeniería Eléctrica y Electrónica de la Escuela Politécnica Nacional consta de siete capítulos que se encuentran detallados en el ANEXO A.

El primer capítulo muestra la introducción a la programación utilizando MATLAB[12]. Brinda una introducción breve acerca de la creación de diagramas de flujo, manejo de matrices y sentencias básicas de control.

Además, en este capítulo se aplican estos temas como base para la creación y utilización de funciones, así como el uso del entorno de programación visual llamado GUIDE[13] y Simulink[14] que permite simular sistemas dinámicos, además de diferentes herramientas para la industria, estadísticas, financieras, científicas y para ingeniería.

En el segundo capítulo se introduce al estudiante al lenguaje de programación Java, la arquitectura y los tipos de datos que maneja. También, se mencionan las estructuras de control que permiten modificar el flujo de ejecución de las instrucciones de un código. Finalmente, se revisan conceptos acerca de vectores.

En el tercer capítulo se estudia los paradigmas de la programación orientada a objetos o también llamados como los cuatro pilares de la programación orientada a objetos.

En primer lugar, se menciona a la abstracción y el encapsulamiento (ocultación de la información).

Dentro del paradigma de abstracción se menciona conceptos como clases, objetos, atributos y propiedades. En cuanto al encapsulamiento, se hace referencia a modificadores de acceso (`public`, `private`, `protected`), modificadores de métodos y atributos (`final`, `static`).

Durante este capítulo también se analizan los paradigmas de la programación orientada a objetos llamados herencia y polimorfismo. Se revisa su definición y sus características y también se introduce los conceptos de superclases y arreglos de datos.

En el cuarto capítulo se trata los principios del análisis y diseño de software orientado a objetos utilizando UML[15], en donde se hace énfasis en los diagramas de clase, de secuencia y de casos de uso.

En el quinto capítulo se introduce al estudiante a los conceptos que involucran el manejo de excepciones. Se analiza la jerarquía de excepciones, así como, la forma de utilizarlas dentro de un programa.

---

[12] MATLAB: Software matemático destinado para el cómputo numérico y la simulación.
[13] GUIDE: Entorno de programación visual de MATLAB para generar interfaces gráficas.
[14] Simulink: Entorno de programación con diagramas de bloques basado en diseño de modelos.
[15] UML (*Unified Modeling Languaje*): Estándar para la representación de procesos o esquemas de software.

El sexto capítulo se analiza el manejo de archivos y flujos que permiten al usuario tener datos persistentes. También se realiza el estudio del acceso secuencial a archivos de texto y la serialización de objetos.

El séptimo capítulo realiza un análisis de algoritmos y estructuras de datos en Java utilizando conceptos como pilas y colas de datos, algoritmos de búsqueda y listas enlazadas.

Analizando la temática de cada capítulo y basándose en la bibliografía recomendada en el sílabo, se puede estructurar las preguntas, conceptos y ejemplos que alimentarán la base de datos para el videojuego.

Por cada capítulo de la asignatura se tienen veinte preguntas, veinte conceptos y quince segmentos de código (ejemplos que incluyen un enunciado junto con una imagen del segmento de código) que se encuentran detallados en el ANEXO B.

1.3.5. HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO

1.3.5.1. Unity

Unity es un motor de videojuegos desarrollado por Unity Technologies. Está programado en C, C++ y C#, es compatible en sistemas tanto Microsoft Windows, Mac OS X como Linux, y permite crear juegos para Plataformas como Xbox 360[16], PlayStation 3[17], Wii[18], Wii U[19], iPad, iPhone, Android, Windows Phone, entre otros [24].

Unity se estructura mediante el manejo y la creación de escenas para el desarrollo de aplicaciones, una escena puede ser una parte del juego o la animación, ya sea un nivel del juego o un área determinada.

Se empieza con un espacio en blanco en el cual se puede dar forma a todo lo que se desee crear usando las herramientas de Unity. El motor se encarga de englobar y darle sentido a los archivos que creados para generar escenarios, personajes y componentes del videojuego en general.

Estos archivos conocidos en Unity 3D como "*assets*" se importarán en subdirectorios bajo el directorio principal [25]. Un *asset* es una representación de cualquier ítem que puede ser utilizado en el juego o proyecto. Este puede provenir de un archivo creado afuera de Unity, tal como un modelo 3D, un archivo de audio, una imagen, o cualquiera de los otros tipos

[16] Xbox 360: Consola de videojuegos desarrollada por Microsoft en colaboración con IBM y ATI.
[17] PlayStation 3: Consola de videojuegos desarrollada por Sony.
[18] Wii: Consola de videojuegos desarrollada por Nintendo.
[19] Wii: Consola de videojuegos sucesora del Wii que permite gráficos de alta definición.

de archivos que Unity soporta. También hay otros tipos de *assets* que pueden ser creados dentro de Unity, como *Animator Controller*[20], *Audio Mixer*[21] o *Render Texture*[22] [24].

El éxito que ha alcanzado Unity dentro del ámbito de los videojuegos se debe a que se enfoca en las necesidades de los desarrolladores independientes que no pueden crear ni su propio motor del juego ni las herramientas necesarias o adquirir licencias para utilizar plenamente las opciones que aparecen disponibles. El enfoque de la compañía es hacer el desarrollo de contenidos interactivos en 2D y 3D lo más accesible posible a tantas personas en todo el mundo como sea posible [25]. Una de las principales ventajas de Unity, además de contar con una basta documentación, es la forma de programación sencilla para desarrollar juegos. En Unity, puede usarse *scripts*[23] para desarrollar casi todas las partes de un videojuego o contenido interactivo [24].

1.3.5.2. Visual Studio

El entorno de desarrollo integrado[24] (IDE) de Visual Studio es una plataforma de lanzamiento creativa que puede usar para editar, depurar y crear código, y luego publicar una aplicación. Es compatible con múltiples lenguajes de programación, tales como C++, C#, Visual Basic .NET, F#, Java, entre otros, al igual que entornos de desarrollo web.

Además del editor y depurador estándar que proporcionan la mayoría de los IDE, Visual Studio incluye compiladores, herramientas de finalización de código, diseñadores gráficos y muchas más funciones para facilitar el proceso de desarrollo de software. Unity permite la creación y edición de *scripts* a través de este IDE [26].

1.3.5.3. MySQL

MySQL es el sistema de gestión de bases de datos relacional más extendido en la actualidad al estar basada en código abierto. Desarrollado originalmente por MySQL AB, fue adquirido por *Sun MicroSystems* en 2008 y este a su vez fue comprado por *Oracle Corporation* en 2010, el cual ya era dueño de un motor propio InnoDB para MySQL. MySQL es un sistema de gestión de bases de datos que cuenta con una doble licencia. Por una parte, es de código abierto, pero por otra, cuenta con una versión comercial gestionada por la compañía Oracle.

---

[20] *Animator Controller*: *Asset* que permite mantener un conjunto de animaciones para un personaje u objeto dentro de videojuego.
[21] *Audio Mixer*: Permite mezclar varias fuentes de audio, aplicarles efectos y realizar masterizaciones.
[22] *Render Texture*: Tipos especiales de texturas que se crean y actualizan en tiempo de ejecución.
[23] *Script*: Conjunto de instrucciones guardadas en un archivo de texto.
[24] Entorno de desarrollo integrado (IDE): Programa que ofrece servicios y funciones necesarias para aspectos de desarrollo de software.

MySQL utiliza SQL que es un lenguaje generalizado dentro de la industria. Al ser un lenguaje estandarizado, ofrece plena compatibilidad. Por lo que si se ha trabajado en otro motor de bases de datos no se presentarán problemas al migrar a MySQL. Por otro lado, las diferencias de sintaxis entre motores de bases de datos siguen siendo importantes y vale la pena tenerlas en cuenta.

En la Tabla 1.1 se muestran las sentencias más usadas en las bases de datos.

Tabla 1.1 Algunas sentencias de MySQL

Sentencia	Descripción
`Select`	Usada para consultar datos de una tabla
`Distinct`	Elimina los duplicados en las consultas de datos
`Where`	Clausula utilizada para incluir condiciones en las consultas de datos
`And`, `Or`	Conectores lógicos para incluir dos o más condiciones a una consulta
`Insert`	Usada para insertar datos a una tabla
`Update`	Usada actualizar o modificar datos ya existentes
`Delete`	Usada borrar datos de una tabla
`Order by`	Usada para ordenar los resultados de una consulta

MySQL presenta algunas ventajas que lo hacen muy interesante para los desarrolladores. La más evidente es que trabaja con bases de datos relacionales, es decir, utiliza tablas múltiples que se interconectan entre sí para almacenar la información y organizarla correctamente. Al ser basada en código abierto es fácilmente accesible y la inmensa mayoría de programadores que trabajan en desarrollo web han usado MySQL en alguno de sus proyectos porque al estar ampliamente extendido cuenta además con una comunidad que ofrece soporte a otros usuarios.

MySQL basa su funcionamiento en un modelo cliente y servidor. Es decir, clientes y servidores se comunican entre sí de manera diferenciada para un mejor rendimiento. Cada cliente puede hacer consultas a través del sistema de registro para obtener datos, modificarlos, guardar estos cambios o establecer nuevas tablas de registros [27].

1.3.5.4. Photon Unity Networking (PUN)

PUN es un *asset* gratuito que se encuentra en la *Asset Store* de Unity. Este permite añadir multijugador online a proyectos de forma sencilla y entre distintas plataformas, a través de

Photon Cloud[25], que permite conectar hasta veinte clientes sin ningún costo, en un servidor gratuito, y hasta 200.000 usuarios creando un servidor propio a cambio de una cuota mensual. A su vez, dichos servidores se organizan en la creación de distintas salas internas.

La principal ventaja de utilizar PUN respecto a la herramienta multijugador-nativa de Unity, reside en la arquitectura cliente-servidor, que minimiza el consumo de recursos y garantiza una baja latencia y alta fiabilidad.

Por otra parte, para utilizar PUN en un proyecto se configura *Photon Cloud* para obtener un identificador que debe ser ingresado en el servidor. Dicho identificador es utilizado en el *script Photon Server Settings*, donde se incluyen los datos necesarios para que el proyecto se conecte correctamente a la nube.

La clase más importante al utilizar PUN es `PhotonNetwork`, que funciona igual que la clase `Network`[26] de Unity y permite gestionar todo el proceso de conexión a través de ella, como la creación o búsqueda de una sala, y también la creación de `GameObjects`[27] cuando un usuario se une a dicha sala o realiza alguna acción en particular.

También asigna de forma automática un ID a cada usuario al conectarse a la sala, lo que permite reconocerlos entre ellos. La actualización de las características de un `GameObject` según sus acciones se realiza mediante la clase `PhotonView`[28], que intercambia dicha información con el servidor.

PUN incorpora *scripts* configurables con esta clase, de manera que al asignarlo a un `GameObject` se declare de manera rápida y sencilla los componentes que se desea enviar, e incluso permite alterar el intervalo de actualización de dicho componente [28].

Además, PUN permite un emparejamiento flexible que lleva a los jugadores a salas donde los objetos se pueden sincronizar a través de la red. La comunicación rápida y confiable se realiza a través de servidores dedicados de *Photon*, por lo que los clientes no necesitan conectarse uno a uno [29].

Realizando una comparación de la transferencia de los mensajes entre los nodos de una red en Unity y en PUN se tiene que la arquitectura propia de Unity admite una arquitectura de tipo cliente/servidor en la que todos los mensajes deben pasar por el cliente *host* y no

[25] *Photon Cloud*: Es una solución de software como servicio totalmente administrado.
[26] Clase `Network`: Permite configurar la interfaz de red y todos los parámetros de red.
[27] `Game Object`: Objetos con propiedades específicas dentro de un proyecto de Unity.
[28] *Photon View*: Clase propia de *Photon* que brinda una puerta de entrada para crear juegos multijugador.

pueden enviarse directamente entre nodos. Por ejemplo, la Figura 1.8 indica que los mensajes se transmiten desde el cliente B al cliente C usando la siguiente ruta:

Cliente B ▸ Servidor de retransmisión ▸ Host A ▸ Servidor de retransmisión ▸ Cliente C.

Por otra parte, PUN tiene una arquitectura de cliente/servidor similar, pero también admite el envío de mensajes de igual a igual. La Figura 1.8 muestra que los mensajes se transmiten desde el Cliente B al Cliente C utilizando la siguiente ruta:

Cliente B ▸ Servidor de retransmisión ▸ Cliente C.

Eso es un total de dos saltos en comparación con los cuatro en Unity para la misma transferencia de mensaje entre dos nodos. Además de esto, PUN podría potencialmente pasar por alto el servidor de retransmisión por completo y el cliente B puede comunicarse directamente con el cliente C, reduciendo así los saltos a uno. Además de esto, en el caso de la arquitectura de Unity, si el *host* se desconecta de la red, el juego se detiene. Por lo mencionado, se considera que PUN es más rápido y eficaz que la arquitectura propia de Unity para el manejo de juegos multijugador [30].

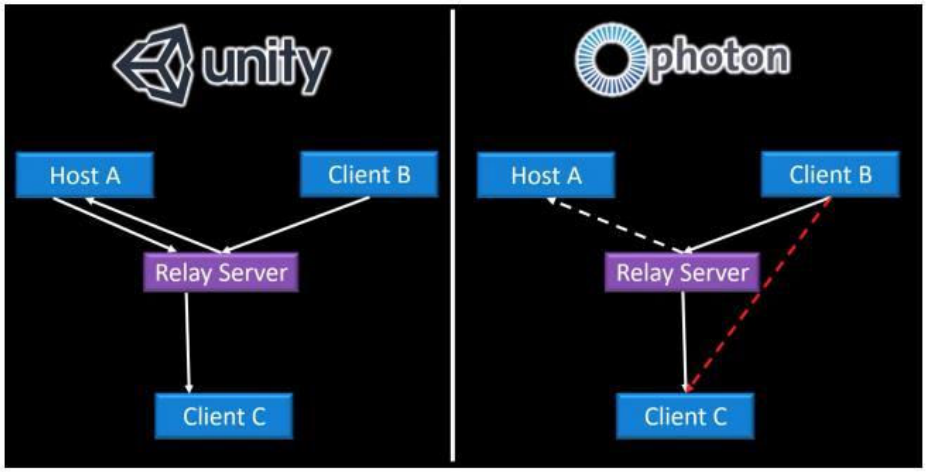

Figura 1.8. Transferencia de mensajes entre nodos de una red en Unity y PUN [30]

1.3.5.5. UnityWebRequest

Para la comunicación entre el cliente y el servidor se utilizará `UnityWebRequest` que proporciona un sistema modular para componer peticiones HTTP y manejar respuestas HTTP. El objetivo principal del sistema de `UnityWebRequest` es permitir al videojuego creado en Unity interactuar con el servidor.

`UnityWebRequest` divide una transacción HTTP en tres operaciones distintas:

- Proporcionar datos al servidor

- Recibir datos del servidor
- Control de flujo HTTP (re-direccionamiento, manejo de errores, etc.)

Para proporcionar una mejor interfaz para usuarios avanzados, estas operaciones están gobernada por sus propios objetos:

- El objeto `UploadHandler` maneja la transmisión de datos al servidor
- El objeto `DownloadHandler` maneja el recibimiento, *buffering* y post-procesamiento de datos recibidos del servidor
- El objeto `UnityWebRequest`, maneja otros dos objetos, y también maneja el control del flujo HTTP. En este objeto se definen encabezados personalizados y URL, y dónde la información de error y re-dirección está almacenada.

El flujo del código genérico para cualquier transacción HTTP se muestra en la Figura 1.9 [24].

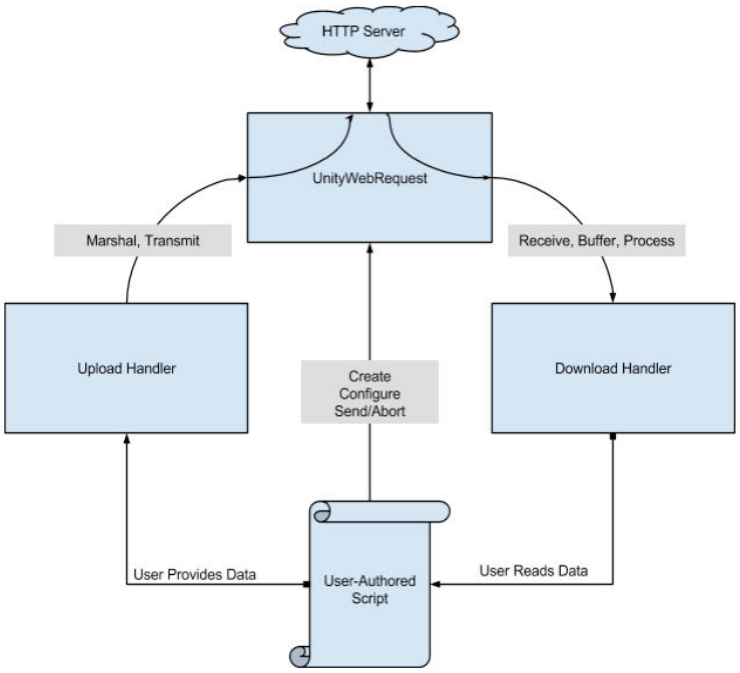

Figura 1.9. Flujo de código para transacción HTTP usando `UnityWebRequest` [24]

1.3.5.6. Unity WebGL

WebGL es una opción de construcción que permite a Unity publicar contenido como programas JavaScript que utilizan tecnologías HTML5[29] y el API[30] de renderización WebGL

[29] HTML5 (*Hyper Text Markup Language*): Es un lenguaje usado para estructurar y presentar el contenido para la web.
[30] API (*Application Programming Interface*): Conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

para correr el contenido de Unity en el explorador web. Para ejecutar un código en WebGL necesita estar en lenguaje JavaScript. Unity utiliza herramientas de compilación (*emscripten*[31]) para hacer una compilación cruzada del código de ejecución de Unity (escrito en C y C++) a asm.js[32] JavaScript. Para convertir el código del juego .NET[33] (sus *scripts* C# y UnityScript) a JavaScript, Unity utiliza una tecnología llamada IL2CPP que toma *bytecode*[34] .NET y lo convierte en archivos fuente correspondientes a C++, el cual luego se compila utilizando *emscripten* para que sus *scripts* sean convertidos a JavaScript.

Finalizado este proceso el videojuego puede ser ejecutado en un explorador web, Unity WebGL soporta la mayoría de los navegadores de escritorio. Sin embargo, el nivel de soporte y el rendimiento esperado varía entre los diferentes navegadores. Por ejemplo, Internet Explorer no soporta audio y es muy lento para soportar la mayoría de contenido de Unity WebGL [24].

1.4. COMPARACIÓN CON TRABAJOS AFINES

El trabajo de titulación llamado “Desarrollo de prototipo de videojuego distribuido basado en realidad virtual para dispositivos Android” [31] trata acerca del desarrollo de un prototipo de videojuego para Android. La implementación del proyecto se realizó combinando conceptos como realidad virtual y desarrollo de videojuegos. La finalidad de este prototipo es conseguir un videojuego estable con posibilidad de interacción con el entorno creado y estableciendo estándares de conexión adecuados para multijugadores, sin descuidar la correcta adaptación de los visores de realidad virtual con las funciones y el entorno del videojuego. Además, se analiza la plataforma de *Google Cardboard*, su compatibilidad y las posibilidades de desarrollo que se tienen con la utilización de esta tecnología.

Por otra parte, el trabajo de titulación propuesto plantea el desarrollo de un prototipo de videojuego enfocado en la educación. Utilizando el motor de desarrollo de videojuegos Unity en conjunto con una metodología ágil de desarrollo de software se pretende desarrollar un prototipo de videojuego que permita reforzar conocimientos de la materia de Programación Avanzada. A diferencia del trabajo de titulación citado, el videojuego planteado utilizará el enfoque de juegos serios, que implica priorizar a la formación antes que el entretenimiento.

---

[31] *Emscripten*: Programa informático considerado un compilador que puede procesar *bytecode* que se crea normalmente al compilar código C o C++.
[32] asm.js: Es un subconjunto de JavaScript que permite a los motores de JavaScript compilar código asm.js a un código nativo de buena calidad.
[33] .NET: Plataforma para el desarrollo de software lanzada por Microsoft.
[34] *Bytecode*: Código intermedio entre el código fuente y el código máquina.

2. METODOLOGÍA

En el presente Trabajo de Titulación se propone desarrollar un prototipo de videojuego utilizando la metodología ágil SCRUM, misma que será adaptada para cumplir los requerimientos de diseño, así como las particularidades de este trabajo. Como parte de esta metodología se emplea una lista de producto llamada *product backlog* y una lista de pendientes conocida como *sprint backlog* con el fin de visualizar el flujo de trabajo en todo momento.

En este capítulo se describirán las tareas realizadas como parte de dos etapas de acuerdo con lo indicado en la sección 1.3.3.4, las cuales son: *Pre-Game* (análisis) y *Game* (diseño e implementación). En la fase de *Pre-Game* se establecerán los requerimientos del juego, para lo cual se realizará un análisis de tres videojuegos: "*Kahoot!*", "Perobo" y "*Codecom*bat".

Mientras que en la fase de *Game,* se llevará a cabo el diseño del prototipo, lo cual incluye la generación de diagramas de casos de uso, diagramas entidad-relación, diagramas de clases y *mockups*; además, en esta fase se realizará la implementación del prototipo de videojuego empleando las herramientas Unity, Visual Studio, MySQL y WebGL, así como los *backlogs* de SCRUM.

Finalmente, en la fase *Post-Game* se probará el funcionamiento de todos los componentes del videojuego y se realizarán las correcciones respectivas en caso de ser necesario; además, se realizarán encuestas de validación a estudiantes que estén cursando la materia de Programación Avanzada de la Facultad de Ingeniería Eléctrica y Electrónica de la Escuela Politécnica Nacional. Los detalles de este proceso se presentarán dentro del Capítulo 3.

2.1. ANÁLISIS DE REQUERIMIENTOS - FASE PREGAME

2.1.1. CODECOMBAT

Codecombat es un juego de rol que permite introducir a los usuarios a conceptos básicos de programación y de los lenguajes de codificación como Javascript o Python; entre su funcionalidad se puede mencionar que ayuda con el aprendizaje de la sintaxis, los bucles y secuencias, puesto que los fragmentos de código requeridos deben ser escritos por los usuarios y a su vez estos pueden reaccionar a las instrucciones en tiempo real [5].

El videojuego consiste en manejar un personaje al que se le pueden personalizar ciertos elementos como las armaduras, armas o habilidades, los cuales son desbloqueados mediante un sistema de recompensas.

A medida que se avanza en el juego, el personaje consigue mejores objetos y habilidades que lo hacen más fuerte.

Para realizar los movimientos del personaje, el jugador debe ingresar instrucciones codificadas usando Python o JavaScript de igual manera para realizar acciones de ataque; un ejemplo se muestra en la Figura 2.1. Esta forma de codificación con instrucciones intuitivas resulta entretenida y permite al jugador aprender la sintaxis utilizada con cada nivel del videojuego.

Este juego es considerado un juego serio ya que combina los aspectos de un videojuego en 2D con características de gamificación[35], despertando interés en trabajar con estudiantes que empiezan a codificar o a trabajar con computadoras.

Figura 2.1 Escena de ingreso de instrucciones en *Codecombat* [32]

La dificultad del videojuego, adicional al desconocimiento del manejo de instrucciones para los jugadores principiantes, está en las iteraciones o condicionales, o los métodos especiales como saltar ejecutar una acción, que deben seleccionarse de manera cuidadosa, ya que, sin los objetos adecuados, resulta muy difícil la resolución de los niveles.

En cuanto a la interfaz gráfica, este videojuego tiene un ambiente 2D con animaciones sencillas, pero a su vez atractivas para el usuario. Maneja interfaces para el registro he ingreso de usuarios con componentes visuales coloridos y menús de sencillo manejo como se muestra en la Figura 2.2.

[35] Gamificación: Técnica de aprendizaje que traslada la mecánica de los juegos al ámbito educativo-profesional.

De igual forma, dentro del videojuego los gráficos llamativos e instrucciones detalladas para introducir al jugador al mundo virtual mantienen el interés del jugador en el videojuego. Como ejemplo, se puede mencionar la escena de manejo de inventario y personalización del personaje que se muestra en la Figura 2.3.

El entorno en el que se desenvuelve el jugador es intuitivo, ya que, a medida que el jugador avanza a través de más niveles, se vuelve más fácil reconocer las instrucciones que deben escribirse y detectar cualquier atajo para reducir líneas de código.

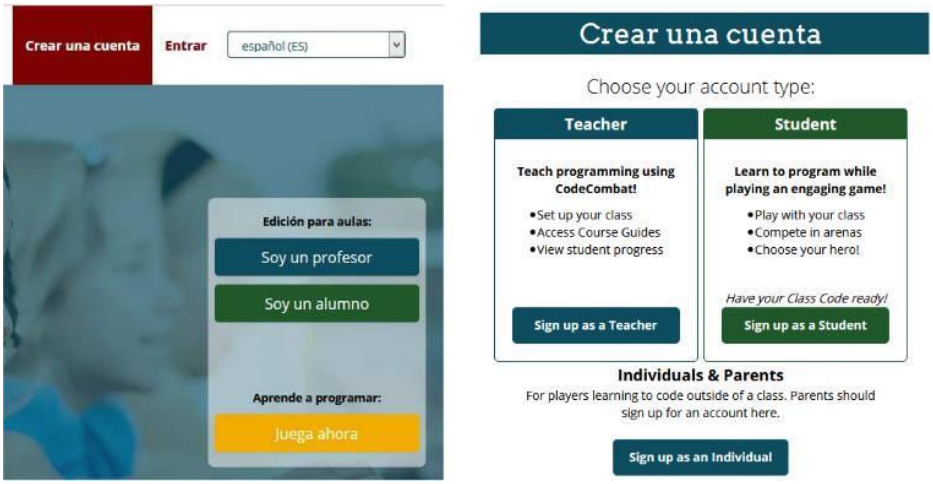

Figura 2.2 Registro de usuarios de *Codecombat* [33]

También, ofrece ayuda para superar un nivel. El videojuego ofrece realimentación oportuna al jugador en caso de errar con las instrucciones que permiten alcanzar los objetivos de cada nivel. Finalmente, se debe resaltar que independientemente de si el jugador posee o no una suscripción al juego, este permite guardar el progreso del mismo.

Figura 2.3 Escena de inventario de *Codecombat* [33]

2.1.2. KAHOOT!

Kahoot! es una plataforma de juego que a través de cuestionarios, discusiones y encuestas prefabricadas o improvisadas engancha tanto a docentes como a estudiantes. Es una herramienta digital gratuita y el usuario puede acceder desde cualquier dispositivo que tenga un navegador web. La mecánica del videojuego consiste en que los estudiantes respondan preguntas formuladas por el profesor dentro de un determinado tiempo. Las preguntas son proyectadas en el aula mientras que los alumnos las responden desde un dispositivo conectado a internet [34].

Esta es una aplicación dinámica que proporciona informes de evaluación de manera inmediata. Estos informes ayudan a los profesores para tener una visión realista del nivel de conocimiento de sus estudiantes. Además, permite cambiar el rol y hacer que los alumnos desarrollen las preguntas e interactúen entre ellos usando cuatro modos distintos de juego llamados: cuestionario, encuesta, discusión, *jumble*[36], en los cuales se varía en nivel de competitividad, así como el número y orden de las respuestas.

Para crear los cuestionarios los usuarios crean las preguntas e ingresan hasta cuatro posibles opciones de respuesta. Las preguntas están limitadas a 80 caracteres, mientras que las respuestas están limitadas a 60 caracteres. Otra característica que el usuario puede ajustar es la cantidad de tiempo que se tiene para responder cada pregunta y cuántos puntos vale cada pregunta.

El tiempo para responder una pregunta está preestablecido en 30 segundos y el valor de cada pregunta se establece en 1000 puntos. Además, para hacerlo más atractivo para los estudiantes, el docente puede incluir una imagen o un video que será presentado en el cuestionario a los estudiantes, como se muestra en la Figura 2.4.

En cuanto a las interfaces gráficas, este videojuego se puede dividir en dos partes, la primera parte consta de formularios para el registro de usuario y el establecimiento de la escena de juego en la que se mostrarán las preguntas y las posibles respuestas al estudiante; estas interfaces son coloridas y atractivas para el usuario ya que permiten usar imágenes o videos relacionados con el tema, lo que hace llamativo al juego durante las partidas.

Por otra parte, se tienen los formularios de creación de cuestionarios, que son sencillos, pero intuitivos para el usuario. En esta interfaz se despliegan cuadros de texto para ingresar

[36] *Jumble* (revoltijo): Modalidad de juego de *Kahoot!* que permite ordenar o secuenciar cuatro respuestas en el orden correcto. Las posibles respuestas se presentan desordenadas, y puede establecerse un temporizador para ordenar cada pregunta de 2 minutos.

los campos que deben ser llenados, así como los detalles que pueden ser agregados (imágenes, videos, etc.) como se observa en la Figura 2.5. Es menos colorida y detallada que la interfaz que es mostrada a los estudiantes, pero que cumple con el objetivo administrativo del juego.

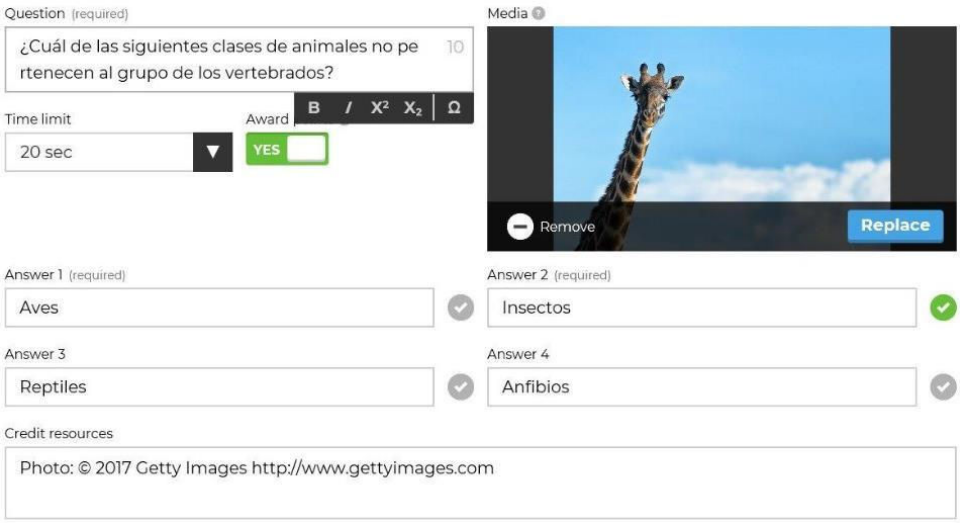

Figura 2.4 Menú de ingreso de preguntas en *Kahoot!* [34]

Este juego maneja un sistema de *ranking* que fomenta la competitividad de los estudiantes en el aula, envolviéndolos en un contexto lúdico, pero a su vez educativo. La retroalimentación en este programa no se realiza de forma automática y es crítica en el proceso de aprendizaje, por lo que es necesario que el profesor analice las falencias de los estudiantes en las diferentes preguntas y solvente las dudas basándose en el *ranking* obtenido después de la realización de un cuestionario.

Figura 2.5 Registro de usuarios en *Kahoot!* [6]

2.1.3. PEROBO

Este es un videojuego creado para fomentar el aprendizaje de programación utilizando el lenguaje de programación C, ha sido creado utilizando el software Stencyl[37] y cuenta con cuatro niveles. El primer nivel introduce al jugador en temáticas relacionadas con punteros, mientras que el segundo se nivel ilustra cómo usar y declarar dichos punteros. En el tercer nivel se manejan conceptos de tablas y su funcionamiento en conjunto con los punteros. En el nivel final del juego se maneja conceptos sobre asignación y liberación de memoria usando un puntero [7].

El juego crea una historia inspirada en el funcionamiento de la memoria de la computadora al usar punteros. El propósito es permitir a los alumnos usar su pensamiento abstracto para imaginar cómo funcionan los punteros en C. La idea es usar objetos para construir conocimiento. Durante los diferentes niveles del juego se presenta tutoriales que muestran paso a paso a los jugadores instrucciones y objetivos de aprendizaje. Además, el juego presenta un desafío para el estudiante ya que el número de intentos se limita a tres.

Este juego se enfoca en priorizar al personaje principal que es un estudiante que debe atravesar una serie de desafíos al interactuar con el entorno del juego y construir su propio conocimiento del funcionamiento de los punteros. El estudiante siente que tiene una misión que cumplir y esto ayuda a que sienta que tiene el control de la situación.

El juego maneja mecánicas como "arrastrar y soltar", "apuntar y hacer clic" o completar frases durante el desarrollo de los niveles. Además, el entorno del juego tiene efectos de sonido y eventos visuales que atrapan al estudiante en el mundo virtual con el propósito de hacer que el juego sea divertido a través de movimientos realistas.

El videojuego presenta una interfaz de retroalimentación evaluando algunos conceptos relacionados con lo aprendido durante el nivel. Al manejar conceptos abstractos como el uso de punteros este videojuego intenta que el estudiante no solo aprenda el significado de una palabra, sino que puede asociarla con una imagen, acción o experiencia.

Se debe destacar como aspecto interesante que el jugador puede interactuar con un número extenso de objetos mientras explora el entorno. Aquellos objetos que serán útiles para progresar en el juego son almacenados en el inventario del jugador. Perobo también promueve la sensación de logro al recompensar la curiosidad del alumno dándole comentarios positivos y brindando algunos segmentos de ayuda con contenido educativo. Los conceptos que se manejan en el videojuego son divididos en pequeños trozos y se

[37] Stencyl: Plataforma gratuita de creación de videojuegos.

presenta a los alumnos como una serie de desafíos secuenciados en una complejidad creciente. Después de que los alumnos se familiarizan con cada concepto, se les pide que completen un cuestionario para evaluar experiencia de aprendizaje, como se muestra en la Figura 2.6. Los estudiantes pueden repetir la prueba tantas veces como sea necesario, pero con diferentes preguntas cada vez.

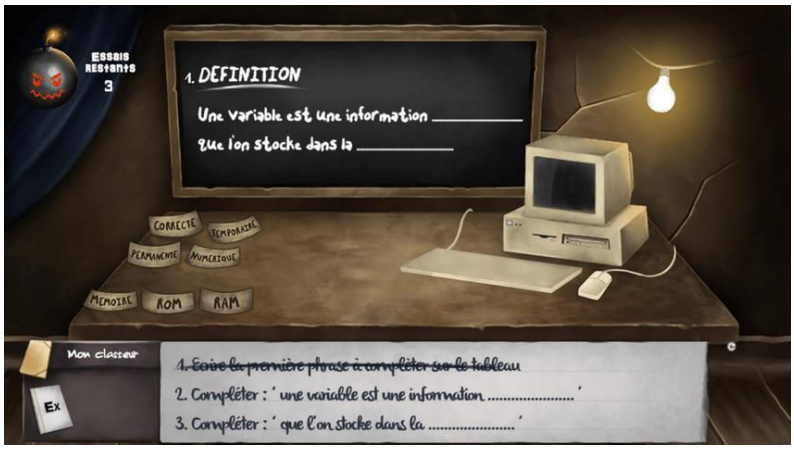

Figura 2.6 Ejemplo de retroalimentación en Perobo [7]

2.1.4. ANÁLISIS DE JUEGOS

A continuación, se detalla los aspectos importantes que serán tomados como base para el desarrollo del videojuego que se plantea en este trabajo de titulación.

Se puede observar que una constante en cada uno de ellos es la presencia de retroalimentación o información que ayude al jugador a comprender o reforzar la temática educativa. También puede observarse que los tres videojuegos cuentan con interfaces bastantes coloridas y atrayentes para el usuario.

Del videojuego llamado *CodeCombat* se toma como referencia el manejo de objetos desbloqueables, así como el manejo de personajes que tiene; ideas que se emplearán en el desarrollo del juego de *Tower Defense.* El videojuego *Kahoot!* presenta diseños de menús llamativos e intuitivos que serán tomados como base para las pantallas de menú del videojuego. Además, al poseer una temática de trivia brindará una aproximación ideal para el desarrollo del minijuego de trivia, así como para el minijuego de batalla naval. El videojuego Perobo muestra diversos desafíos en los que se usan conceptos de informática para el aprendizaje, estas ideas del manejo de conceptos educativos serán aplicadas en todos los minijuegos educativos; además de la mecánica de juego de “arrastrar y soltar” la cual será utilizada para la colocación de torres en el juego de *Tower Defense*. Finalmente, para la parte administrativa del videojuego se tomará como base las interfaces que ofrece el videojuego *Kahoot!.*

2.2. DISEÑO DEL VIDEOJUEGO - FASE GAME

2.2.1. LISTA DE REQUERIMIENTOS

Tomando como base el análisis realizado a los tres videojuegos anteriores se han planteado los siguientes requerimientos:

1. Existirán tres perfiles de usuario: Administrador, Profesor y Estudiante. Cada uno tiene distintos privilegios de acceso al sistema.
2. El videojuego debe permitir la administración de los usuarios con perfil Estudiante y Profesor.
3. El videojuego debe permitir al usuario con perfil Profesor agregar contenido educativo para alimentar a los minijuegos.
4. El perfil de usuario Administrador permite realizar la gestión de usuarios de tipo profesor es decir añadir, eliminar o editar estos registros.
5. El perfil de usuario Profesor permite añadir, eliminar y editar estudiantes, así como la gestión de preguntas, conceptos o segmentos de código que se utilizan en el videojuego.
6. El perfil de usuario Estudiante no tiene permitido realizar acciones de administración del juego, únicamente podrá interactuar con el videojuego.
7. Se tendrán cuatro minijuegos educativos, un minijuego de trivia, un minijuego del ahorcado, un minijuego de sopa de letras y un minijuego de batalla naval.
8. Los minijuegos educativos de trivia, ahorcado y sopa de letras tendrán un ambiente 2D, mientras que para el juego de batalla naval y *Tower Defense* se tendrá un ambiente 3D.
9. En los minijuegos educativos (juego de trivia, sopa de letras, ahorcado y batalla naval), el jugador puede seleccionar el capítulo de la asignatura que desea practicar con cada minijuego.
10. Los minijuegos educativos deben manejar temporizadores para las preguntas o desafíos, con intervalos de tiempo prudentes dependiendo de cada minijuego. Estos temporizadores no podrán ser configurados por los usuarios.
11. El jugador recibirá retroalimentación acerca de la temática o preguntas tratadas en cada uno de los minijuegos educativos.
12. En cada partida del minijuego del ahorcado se mostrarán cinco palabras o frases para las cuales el jugador tendrá cuatro intentos y treinta segundos para resolver cada desafío.
13. En el minijuego de trivia se mostrarán al jugador siete preguntas cada una con cuatro opciones de respuesta y un tiempo de treinta segundos para responder.

14. El minijuego de sopa de letras cuenta con siete palabras escondidas por cada partida que deben ser descubiertas por el jugador en menos de cinco minutos.
15. El minijuego de batalla naval debe contar con cinco barcos que el jugador puede ubicar en el tablero.
16. En caso de que el barco del jugador sea atacado en la batalla naval, se mostrará un segmento de código mediante el cual el jugador deberá seleccionar una solución de código entre al menos cuatro opciones de respuesta en un tiempo de dos minutos.
17. El videojuego tendrá un *ranking* donde se desplegará el puntaje y el nombre de los jugadores mejor puntuados en los minijuegos educativos.
18. Los minijuegos educativos solo permitirán el modo de un jugador, mientras que el juego de *Tower Defense* permite multijugador.
19. El juego *Tower Defense* permitirá la creación de salas de juego que acojan hasta cuatro jugadores por partida.
20. Se permitirá desbloquear cinco torres y cinco enemigos que podrán ser utilizados en el juego de *Tower Defense.*
21. El juego de *Tower Defense* contará con tres mapas diferentes.
22. Cada partida en el juego de *Tower Defense* contará con quince oleadas de enemigos que deben ser derrotados para conseguir la victoria.
23. Dentro del juego de *Tower Defense* se utilizarán “monedas” para realizar la compra de torres, siendo cada moneda equivalente a diez puntos ganados en los minijuegos educativos.
24. El prototipo será utilizado dentro de una arquitectura cliente-servidor.
25. El prototipo debe ser probado en una red con al menos doce estudiantes conectados simultáneamente.

2.2.2. HISTORIAS DE USUARIO

El análisis realizado a los videojuegos en la Sección 2.1 permitió establecer los requerimientos iniciales para el videojuego y esto sirvió como punto de partida para establecer las historias de usuario. En la Tabla 2.1 se muestra el formato que tendrán las historias de usuarios las cuales poseerán los siguientes campos:

1. **Identificación:** Indica el número de secuencia que permitirá identificar a la historia de usuario, generalmente inicia con las siglas HU (Historia Usuario) seguido de un número de dos o tres cifras según sea requerido, por ejemplo: HU007.
2. **Prioridad**: Indica el nivel de importancia que tendrá la historia de usuario entre la pila de tareas que se tienen pendientes.

3. **Nombre de historia**: Es un nombre descriptivo que permite identificar el requerimiento del usuario.
4. **Descripción**: Es una breve narración que detalla los objetivos, es decir, el para qué del requerimiento de usuario.

Tabla 2.1. Formato de historia de usuario

HISTORIA DE USUARIO	
ID:	Prioridad:
Nombre:	
Descripción:	

En la Tabla 2.2 se muestran los cinco niveles de prioridad que pueden presentarse para las historias de usuario, ordenadas de forma ascendente, siendo el nivel cinco el de mayor prioridad. Estos niveles de prioridad generalmente son definidos por el cliente (*Product Owner*). Sin embargo, para el desarrollo del prototipo de videojuego de este documento, la prioridad de las tareas fue definida de acuerdo a la complejidad y tiempo estimado de desarrollo de cada historia de usuario por los desarrolladores.

Tabla 2.2. Niveles de prioridades de historias de usuario

Nivel muy bajo (Trivial)	**Nivel bajo (Tolerable)**	**Nivel medio (Moderada)**	**Nivel alto (Importante)**	**Nivel muy alto (Urgente)**
1	2	3	4	5

En la Tabla 2.3 se describe un requerimiento cuyo nivel de prioridad es moderado. Esta historia de usuario indica que debe ser desarrollada una interfaz que permita a los jugadores realizar la selección de un capítulo de la asignatura previo al inicio de una partida de cualquier minijuego educativo.

Tabla 2.3. Ejemplo de historia de usuario

HISTORIA DE USUARIO	
ID: HU008	**Prioridad:** 3
Nombre: Crear interfaces de selección de contenido para los minijuegos educativos.	
Descripción: El jugador puede seleccionar el capítulo de la asignatura que desea practicar en cada minijuego previo al inicio de una partida.	

Por otra parte, en el ANEXO C se incluyen todas las historias de usuario que se han generado para el desarrollo del videojuego.

2.2.3. PRODUCT BACKLOG

El *product backlog* detalla los requerimientos que va a tener el videojuego, estos requerimientos han sido mencionados previamente en las historias de usuario. Además, se incluye la prioridad y a través del *product backlog* se puede realizar el seguimiento de estos requerimientos. En la Tabla 2.4 se presenta el *product backlog*, en donde se debe destacar el identificador de la historia de usuario, así como, el *sprint* en el que se implementará cada uno de estos requerimientos.

Tabla 2.4. *Product backlog*

ID HU	Nombre de la historia de usuario	Sprint
HU001	Crear la base de datos	1
HU002	Crear interfaz para administración de usuarios	1
HU003	Administrar usuarios de tipo Estudiante	1
HU004	Administrar usuarios de tipo Profesor	1
HU005	Administrar contenido educativo	2
HU006	Crear interfaz para el registro de usuarios al videojuego	2
HU007	Crear interfaz inicial para selección de juegos	2
HU008	Crear interfaces de selección de contenido para los minijuegos educativos	2
HU009	Crear interfaces visuales para cada minijuego educativo	2
HU010	Generar palabras o frases para el minijuego del ahorcado	2
HU011	Codificar acciones para que el jugador interactúe con el minijuego del ahorcado	2
HU012	Codificar parámetros para minijuego del ahorcado	2
HU013	Generar preguntas para el minijuego de trivia	3
HU014	Codificar parámetros para minijuego de trivia	3
HU015	Generar palabras para el minijuego de sopa de letras	3
HU016	Codificar acciones para que el jugador interactúe con el minijuego de sopa de letras	3
HU017	Codificar parámetros para minijuego de sopa de letras	3
HU018	Crear elementos visuales para el minijuego de batalla naval	4
HU019	Codificar el jugador automático (computadora) para el minijuego de batalla naval	4

ID HU	Nombre de la historia de usuario	Sprint
HU020	Codificar acciones para que el jugador (estudiante) interactúe dentro del minijuego de batalla naval	4
HU021	Codificar parámetros para minijuego de batalla naval	5
HU022	Codificar el puntaje que se obtiene en cada minijuego	5
HU023	Codificar la retroalimentación para cada partida en cada minijuego	5
HU024	Crear interfaz para el *ranking* del videojuego	5
HU025	Crear interfaz que permita al jugador crear, unirse y mostrar las salas del juego *Tower Defense*	6
HU026	Crear interfaz para dar inicio al juego de *Tower Defense*	6
HU027	Codificar parámetros para que el jugador pueda crear y unirse a las salas del juego *Tower Defense*	6
HU028	Codificar parámetros previos al inicio de juego de *Tower Defense*	6
HU029	Crear mapas para el juego de *Tower Defense*	7
HU030	Monedas para compra en el juego de *Tower Defense*	7
HU031	Contenido desbloqueable para el juego de *Tower Defense*	7
HU032	Colocar torres en el juego de *Tower Defense*	7
HU033	Generar oleadas de enemigos en *Tower Defense*	8
HU034	Definir la victoria en el juego de T*ower Defense*	8
HU035	Perfeccionar los detalles visuales del videojuego	8

2.2.4. ARQUITECTURA DEL PROTOTIPO

El prototipo está compuesto por el servidor que aloja la lógica del videojuego y a su base de datos, los clientes y el servicio en la nube de *Photon Unity Networking* (PUN), como se lo observa en la Figura 2.7. Además, se emplea una arquitectura cliente-servidor. Para la comunicación entre el cliente y el servidor se utiliza `UnityWebRequest` que proporciona un sistema modular para construir peticiones y manejar respuestas HTTP.

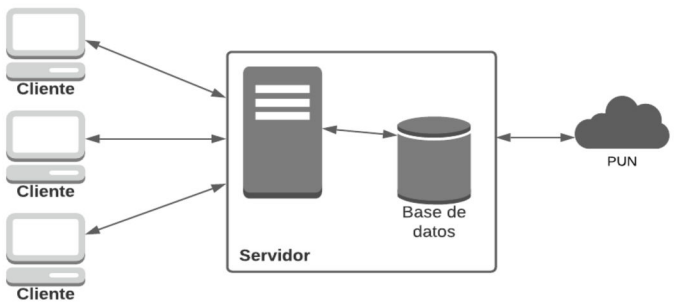

Figura 2.7. Arquitectura del prototipo

2.2.5. CASOS DE USO

En el listado de requerimientos del videojuego se han definido tres perfiles de usuario: Administrador, Profesor y Estudiante.

Tanto el usuario con perfil de Administrador como el usuario con perfil de Profesor están encargados de la parte administrativa del videojuego, mientras que el usuario con perfil de Estudiante tiene acceso al videojuego como tal.

El usuario con perfil de Administrador se utiliza como usuario autorizado para la administración de usuarios del tipo Profesor.

El usuario con perfil de Profesor es el encargado de administrar a los usuarios de tipo Estudiante y también del manejo del contenido educativo. Finalmente, el usuario con perfil de Estudiante tiene acceso a los minijuegos educativos, al juego de *Tower Defense* y al *ranking* del videojuego.

Como se puede observar en la Figura 2.8, el perfil de usuario Administrador puede realizar el CRUD[38] de los usuarios con perfil de Profesor.

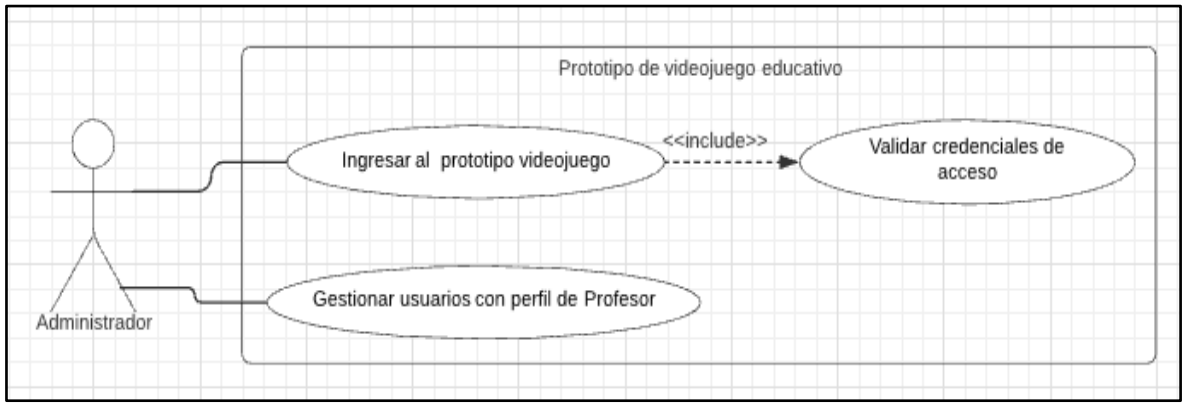

Figura 2.8. Casos de uso del perfil de usuario Administrador

En la Figura 2.9 se muestra el diagrama de casos de uso del perfil de usuario Profesor, este perfil puede realizar la administración de los usuarios con perfil de Estudiante, así como la administración de los capítulos y el contenido educativo de la asignatura. Este perfil de usuario no puede ingresar usuarios con perfil de Profesor, ya que solo los usuarios con perfil de Administrador son los que están autorizados para realizar esta tarea.

Por otra parte, se tienen el perfil de Estudiante, que es el perfil de usuario principal del videojuego cuyo diagrama de casos de uso se muestra en la Figura 2.10.

[38] CRUD: Es el acrónimo de "Crear, Leer, Actualizar y Borrar" que proviene del original en inglés "*Create, Read, Update and Delete".*

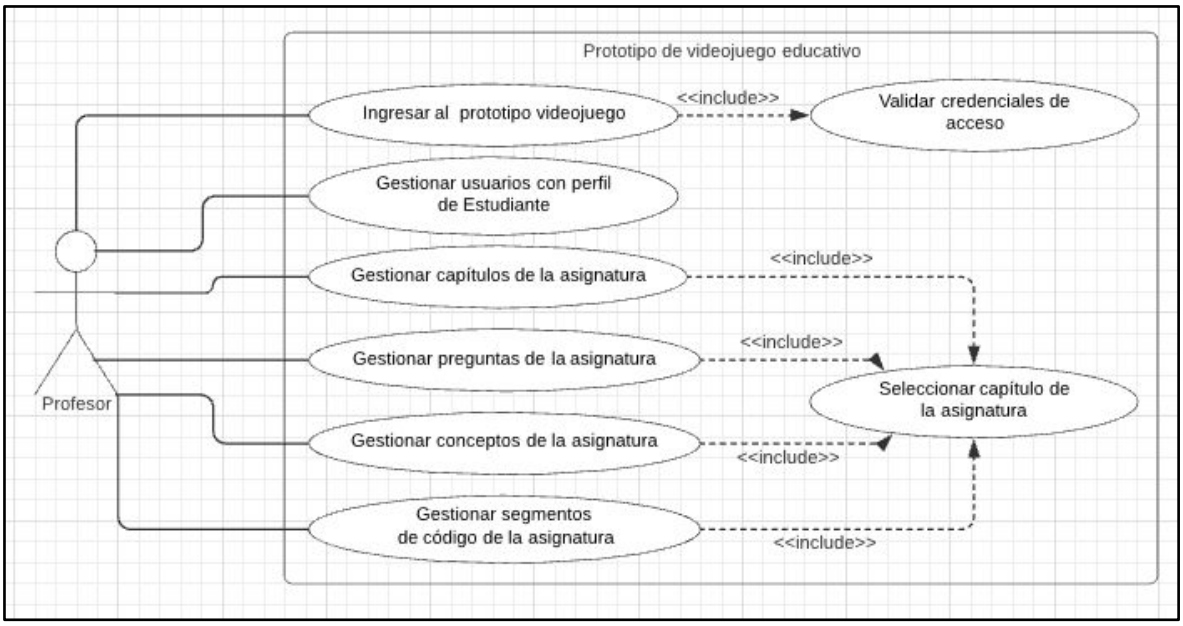

Figura 2.9. Diagrama de casos de uso del perfil de usuario Profesor

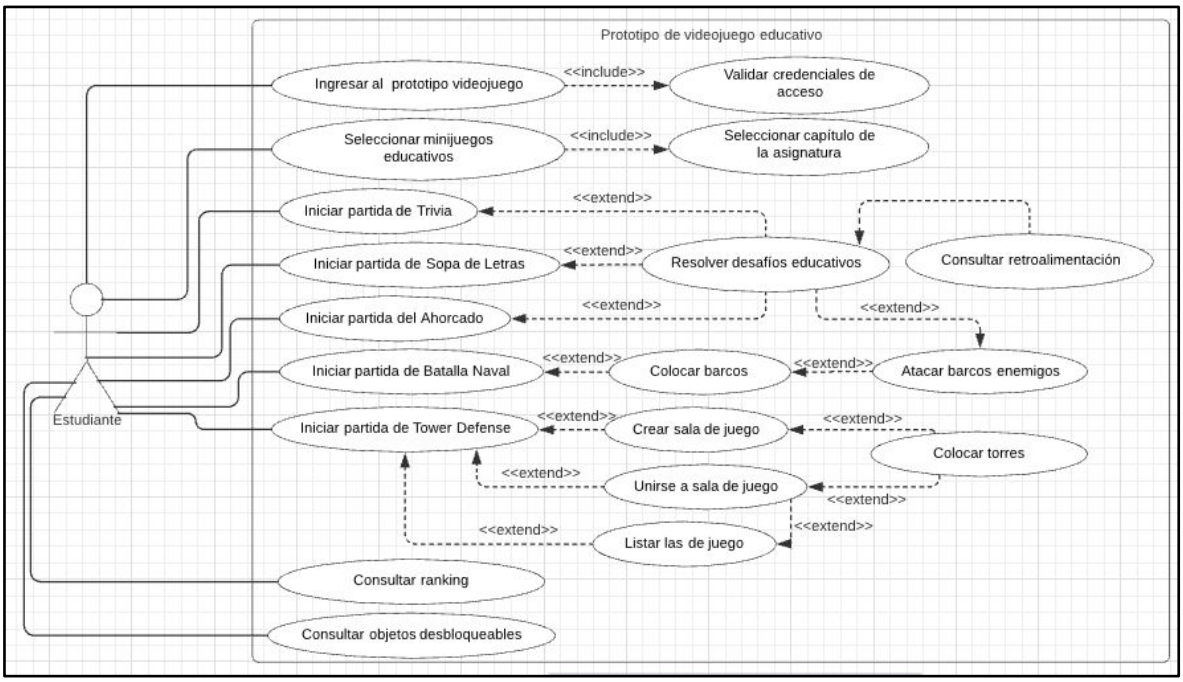

Figura 2.10. Diagrama de casos de uso del perfil de usuario Estudiante

Ingresar al videojuego: Permite al usuario ingresar en el sistema y validar que sea un usuario autorizado para la utilización del videojuego.

Seleccionar minijuegos educativos: Permite al usuario elegir el minijuego educativo en el cual se desea iniciar una partida pudiendo ser estos: Ahorcado, Trivia, Batalla naval, Sopa de letras.

Seleccionar capítulos de la asignatura: Permite al usuario, en cualquier minijuego educativo, elegir el capítulo que desea practicar antes de iniciar una partida.

Iniciar partida en los minijuegos educativos: Permite al usuario, en el caso del minijuego del ahorcado, resolver los cinco desafíos para encontrar la frase o palabra que se muestren en pantalla permitiendo solo cuatro letras erróneas. En el caso de la trivia se muestran siete preguntas cada una con cuatro opciones de respuesta en donde el jugador debe seleccionar la correcta. En el minijuego de batalla naval, el usuario juega contra la computadora. En primer lugar, el usuario debe colocar los barcos en su tablero y posteriormente, en caso de que la computadora ataque a un barco del jugador, se muestra al jugador un segmento de código relacionado con un capítulo de la asignatura. Si el jugador consigue responder correctamente al desafío, el barco se puede salvar del ataque, caso contrario se resta una vida del barco. Finalmente, en el juego de sopa de letras se debe encontrar siete palabras durante cada partida.

Consultar retroalimentación en cada minijuego educativo: Permite al usuario, al final de cada interacción con cada minijuego en cada desafío, conocer más acerca de la asignatura.

Interactuar con el juego de *Tower Defense*: Permite al usuario ingresar al juego lúdico y visualizar las salas de juego creadas por otros jugadores, crear salas e ingresar al mapa del videojuego. Una vez que el juego inicie, cada jugador podrá colocar torres por los senderos correspondientes de los mapas.

Crear salas en el juego de *Tower Defense:* Permite al jugador crear una sala de juego en donde puede configurar el nombre de la sala y el número de participantes que tendrá la misma (máximo cuatro), además de seleccionar uno de los tres mapas disponibles.

Ingresar a una sala creada en el juego de *Tower Defense*: Permite al usuario unirse a sala de juego creada por otro jugador.

Listar salas de juego en *Tower Defense:* Permite al jugador mostrar las salas de juego activas y que esperan por jugadores para iniciar una partida.

Consultar el *ranking* de los jugadores: Permite al usuario realizar consultas acerca de su puntaje y el de otros jugadores, es decir, la interacción que ha tenido cada jugador con los minijuegos educativos.

Consultar objetos desbloqueables: Permite al usuario, de acuerdo con la puntación obtenida en los minijuegos educativos, desbloquear tanto torres como enemigos para el juego de *Tower Defense*.

2.2.6. DIAGRAMA ENTIDAD RELACIÓN

El diagrama entidad relación ilustra como los objetos se relacionan entre sí en la base de datos. De acuerdo con los requerimientos que se han establecido anteriormente, se ha generado el diagrama entidad relación que se muestra en la Figura 2.11.

La tabla `enemy` almacena datos del objeto desbloqueable denominado “enemigo” que es utilizado en el juego de *Tower Defense.* De igual manera, la tabla `tower` permite almacenar datos del objeto desbloqueable denominado “torre” para el juego de *Tower Defense.* Estas tablas tienen como identificador único el campo denominado `idEnemy` e `idTower` respectivamente.

La tabla `unlockable` es una tabla intermedia que permite relacionar a los estudiantes con los objetos desbloqueables.

La tabla `codec` permite almacenar información acerca de los segmentos de código que son utilizados en el minijuego de batalla naval, entre estos campos se destaca al campo denominado `codecPath` que almacena la ubicación o ruta (*path*) de las imágenes dentro del servidor.

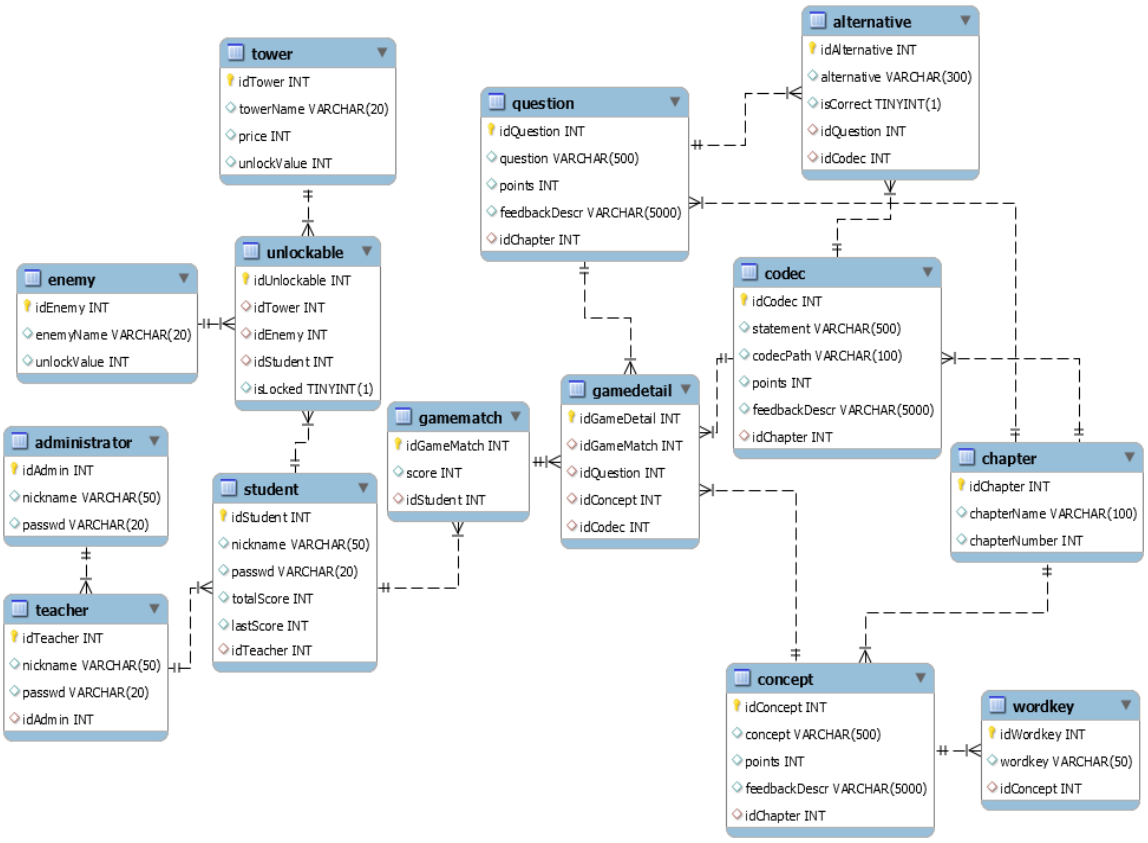

Figura 2.11. Diagrama Entidad - Relación

La tabla concept permite almacenar los conceptos que son utilizados en los minijuegos de ahorcado y sopa de letras.

La tabla question permite almacenar las preguntas que se utilizan en el minijuego de trivia.

Las tablas concept, question y codec poseen el campo feedbackDescr en el que se almacenan datos informativos acerca de la teoría relacionada con los desafíos que se presentan al estudiante en cada minijuego educativo y también poseen el campo points en el que se almacena el puntaje que se le otorga al estudiante por responder correctamente a cada desafío.

La tabla chapter permite almacenar los datos acerca de los capítulos de la asignatura de Programación Avanzada. Su identificador único se denomina idChapter.

La tabla alternative almacena las opciones de respuesta para las preguntas de la tabla question y también para los segmentos de código de la tabla codec. El campo isCorrect permitirá definir si la alternativa de respuesta es la correcta. Su identificador único se denomina idAlternative.

La tabla wordkey almacena las palabras o frases que están relacionadas a un concepto de la tabla concept en el campo wordkey. Su identificador único se denomina idWordkey.

Las tablas administrator, teacher y student almacenan información de cada usuario del sistema.

La tabla gamematch permite almacenar el puntaje obtenido por un estudiante en cada partida realizada en los minijuegos educativos.

La tabla gamedetail sirve como una tabla auxiliar que rompe la relación de varios a varios que se tiene entre las tablas concept, codec y question con la tabla gamematch.

2.2.7. DIAGRAMA DE CLASES

En la Figura 2.12 se observa el diagrama de las clases que heredan de la clase Object.

Todas las clases escritas en el lenguaje de programación C# poseen una herencia implícita de la clase Object, por esta razón en el diagrama mostrado anteriormente se presenta la herencia indicada. Además, estas clases se utilizan para modelar los principales objetos que son extraídos de la base de datos. Por ejemplo, la clase Question que es empleada para gestionar las preguntas existentes en la base de datos.

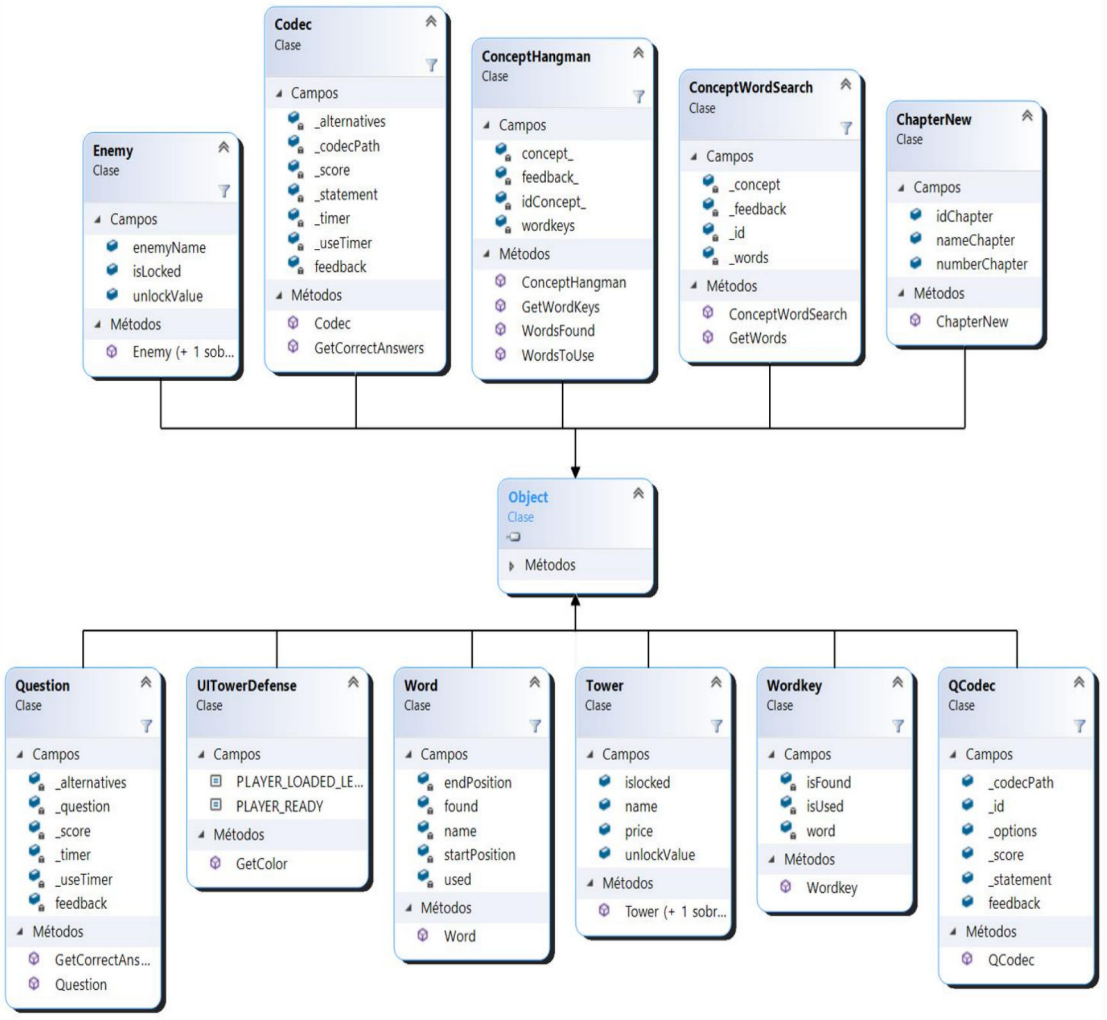

Figura 2.12. Diagrama de clases heredadas de la clase `Object`

Estas instancias serán utilizadas posteriormente dentro de las diferentes interfaces del videojuego. A continuación, se listan todas las clases que heredan de la clase `Object`:

- **`Enemy`:** Esta clase permite gestionar los parámetros básicos de los enemigos, tales como el nombre del objeto y en caso de estar desbloqueado puede emplearse en cada oleada del juego de *Tower Defense*.
- **`Codec`:** Esta clase permite crear instancias para manipular los segmentos de código que son empleados en el minijuego de batalla naval.
- **`ConceptHangman`:** Esta clase permite crear instancias para gestionar los conceptos que son extraídos de la base de datos. Además, posee métodos como `WordsToUse` que son utilizados por el minijuego educativo del ahorcado.
- **`ConceptWordSearch`:** Esta clase permite crear instancias para gestionar los conceptos que son extraídos de la base de datos. Además, posee métodos como `GetWords` son utilizados por el minijuego educativo de sopa de letras.

- **`ChapterNew`:** Esta clase permite crear instancias para gestionar la información de los capítulos de la asignatura Progamación Avanzada. Esta información es extraída de la base de datos.
- **`Question`:** Esta clase permite crear instancias para manipular las preguntas que son extraídas de la base de datos y son utilizadas por el minijuego educativo de trivia.
- **`UITowerDefense`:** Esta clase permite definir parámetros de un usuario que se haya unido a una sala de *Tower Defense*. Uno de estos parámetros es el color con el que se muestra el nombre del jugador en la lista de usuarios de la sala.
- **`Word`:** Esta clase permite definir parámetros para las palabras que son utilizadas en el minijuego de sopa de letras. Uno de estos parámetros indica si la palabra ha sido encontrada en el tablero.
- **`Tower`:** Esta clase permite gestionar los parámetros básicos de las torres, tales como el nombre del objeto y si esta desbloqueado para su uso en cada partida del juego de *Tower Defense*.
- **`Wordkey`:** Esta clase permite definir parámetros para las palabras que son utilizadas en el minijuego del ahorcado. Uno de estos parametros indica si es que la palabra ha sido encontrada en el tablero.
- **`QCodec`:** Esta clase permite crear instancias para gestionar los segmentos de código que son extraídos de la base de datos, que son posteriomente utilizados en el minijuego de batalla naval.

En la Figura 2.13 se observa el diagrama de las clases que heredan de las clases pertenecientes a `PhotonUnityNetworking`.

Dado que el videojuego de *Tower Defense* es de tipo multijugador, se debe manejar un marco de trabajo que permita realizar esta tarea en Unity, en este caso la herramienta encargada de esta tarea es PUN.

Esta se asegura que las acciones realizadas por el usuario, sucedan en el dispositivo local, y a su vez, sean enviados comandos al servidor para ejecutar dichas acciones. Entre las principales clases que heredan de las clases pertenecientes a `PhotonUnityNetworking` tenemos:

- **`LobbyMainPanel`:** En esta clase se definen los parámetros y las acciones relacionadas con la creación y manejo de las salas a las que cada usuario puede unirse para empezar una partida del juego *Tower Defense*.

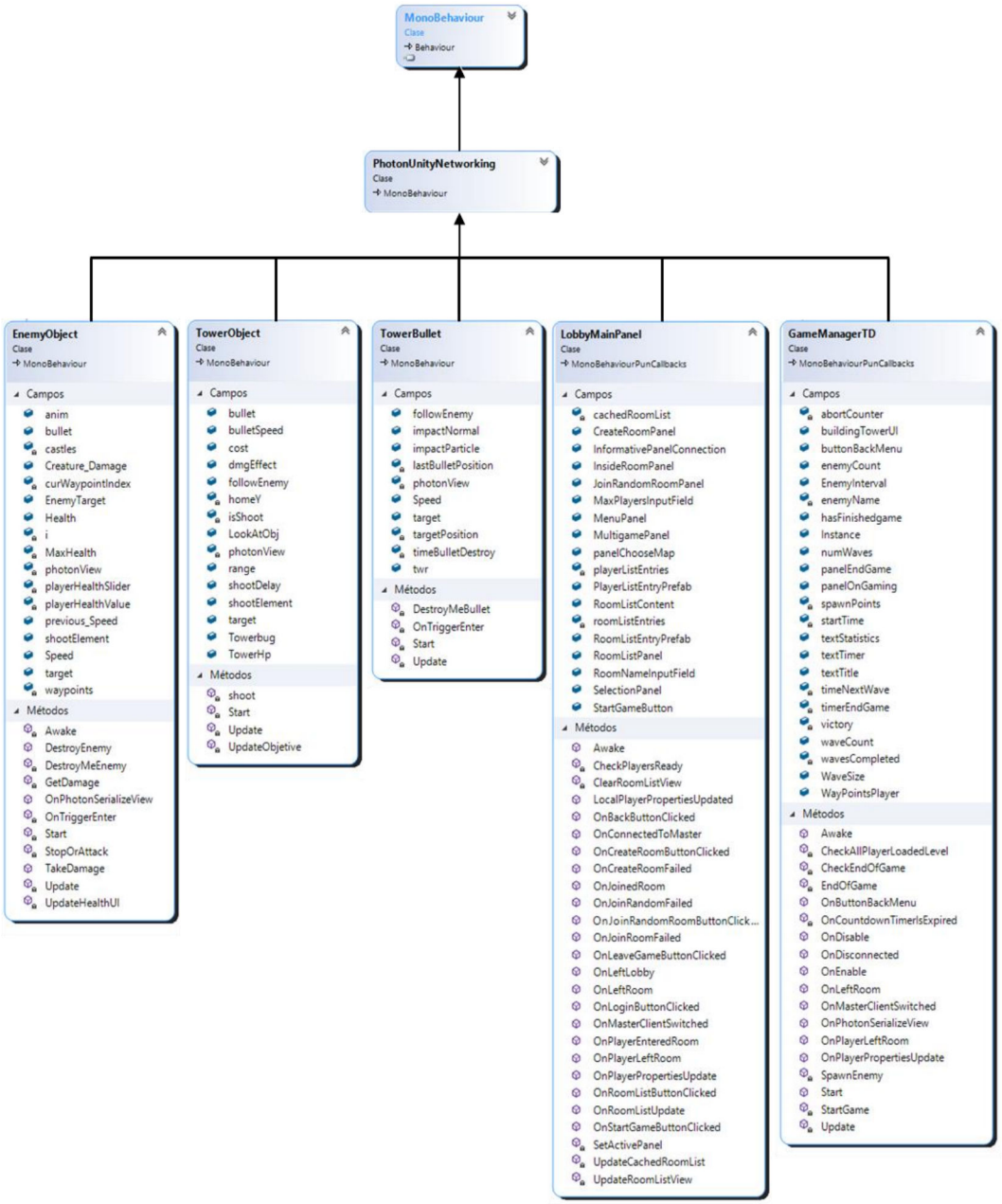

Figura 2.13. Diagrama de clases heredadas de las clases `PhotonUnityNetworking`

- **`EnemyObject`:** Esta clase permite gestionar a los enemigos que se generarán en cada oleada durante una partida del juego *Tower Defense*. Se definen los parámetros y las acciones que realizará un objeto de tipo “Enemigo” tales como, el daño que ejerce a las estructuras, la velocidad con la que desplazan, los puntos de vida que poseen, entre otros.
- **`TowerObject`:** Esta clase permite gestionar a las torres que el usuario puede colocar en el mapa durante una partida del juego *Tower Defense*. Se definen los parámetros y las acciones que realizará un objeto de tipo “Torre” tales como, el

daño que ejerce a los enemigos, la velocidad del disparo, el rango de alcance de la torre, su costo de colocación, entre otros.

- **`TowerBullet`:** Esta clase permite gestionar a las balas que se generan con cada ataque de una torre colocada en el mapa durante una partida del juego *Tower Defense*. Se definen los parámetros y las acciones que realizará un objeto de tipo "Bala" tales como, el tiempo de destrucción de la bala, el movimiento que realiza entre la torre y su objetivo, entre otros.
- **`GameManagerTD`:** En esta clase se definen los parámetros y las acciones relacionadas con las mecánicas del juego *Tower Defense*.

2.2.8. MOCKUPS

A continuación, se presentarán los principales *mockups* del videojuego.

En la Figura 2.14 se muestra el boceto de la pantalla que permitirá la autenticación de usuarios del videojuego.

Figura 2.14. *Mockup* del registro de usuarios (*Login*)

Una vez validado el usuario, en caso de tratarse de un estudiante, se mostrará una pantalla en donde pueda elegir entre los cuatro minijuegos educativos, así como el juego de *Tower Defense*, el boceto de esta interfaz se muestra en la Figura 2.15.

Figura 2.15. *Mockup* de selección de juegos

Por otro lado, si al realizar la autenticación se trata de un usuario con perfil de Profesor o Administrador, la pantalla que se muestre contará con opciones que le permitirán al usuario realizar la administración de Estudiantes, Profesores y contenido educativo como se observa en el boceto de la Figura 2.16.

Figura 2.16. *Mockup* de administración de contenido educativo y usuarios

Cuando un usuario ingrese al juego de *Tower Defense* se mostrará la interfaz que contiene el mapa de en donde interactúan los jugadores, los mapas tendrán una base similar a la que se muestra en el boceto de la Figura 2.17.

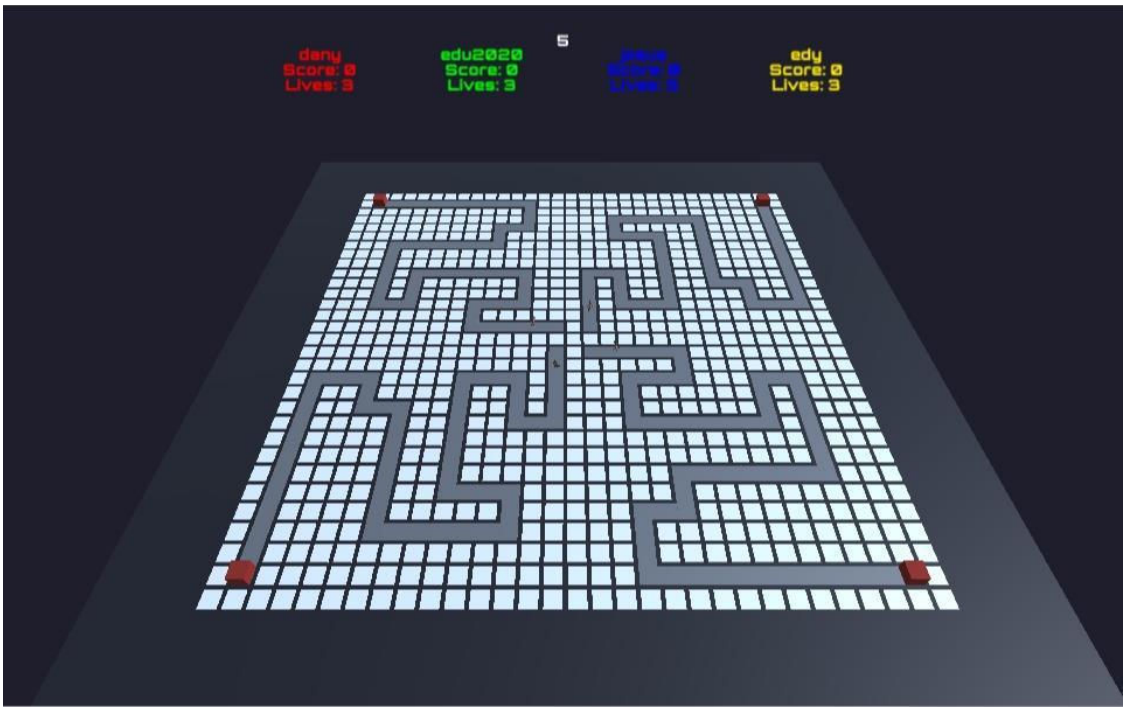

Figura 2.17. *Mockup* de un mapa del juego *Tower Defense*

En la Figura 2.18, Figura 2.19, Figura 2.20 y Figura 2.21 se muestran los *mockups* de cada uno de los minijuegos educativos.

Figura 2.18. *Mockup* del minijuego de trivia

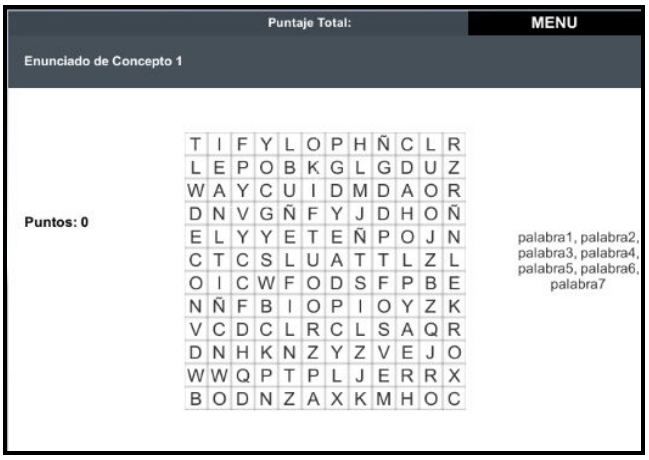

Figura 2.19. *Mockup* del minijuego de sopa de letras

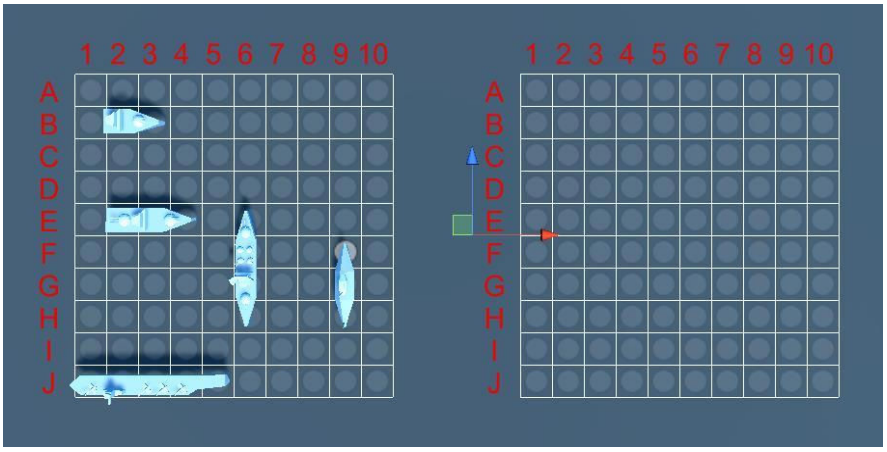

Figura 2.20. *Mockup* del minijuego de batalla naval

Figura 2.21. *Mockup* del minijuego del ahorcado

En el ANEXO E se pueden encontrar algunas *mockups* adicionales.

2.3. IMPLEMENTACIÓN DEL VIDEOJUEGO - FASE *GAME*

Se codificará la base de datos, así como cada uno de los minijuegos educativos y el juego de *Tower Defense*, se codificará también el componente administrativo del videojuego.

Se realizarán las adecuaciones necesarias para que el videojuego pueda ser construido utilizando WebGL[39]. Se codificará la lógica que permita la interacción entre los clientes y el servidor. Se generarán las interfaces y componentes gráficos de cada uno de los minijuegos, así como del juego de *Tower Defense*. Adicionalmente, se alojará al videojuego en un servidor web que permitirá el acceso a los usuarios desde un navegador web.

Para la implementación se han definido siete componentes para el videojuego: base de datos, interfaces administrativas, minijuego del ahorcado, minijuego de sopa de letras, minijuego de trivia, minijuego de batalla naval y juego de *Tower Defense*. A continuación, se presentará un resumen de la implementación de cada componente.

2.3.1. BASE DE DATOS

Para la codificación de la base de datos se emplea el sistema de gestión de base de datos MySQL Workbench 8.0. A continuación se presenta una parte del *script* utilizado para la generación de la base, la creación de las tablas, la inserción de datos y la creación de procedimientos almacenados. En el Código 2.1, en la línea 3, se crea la base de datos llamada `game`, la cual alojará toda la información del videojuego.

[39] WebGL: La opción de construcción WebGL le permite a Unity publicar contenido que utiliza tecnologías HTML5 y el API de renderización WebGL para correr el contenido de Unity en el explorador web.

La sentencia `default character set utf8mb4` permite indicar que se empleará la codificación UTF-8 que define un tamaño de cuatro bytes por carácter multibyte (utf8mb4). Mientras que, la sentencia `default collate utf8mb4_spanish_ci` es una manera de decirle a la base de datos cómo debe comparar el texto y ordenarlo, en este caso se utiliza el idioma español para este proceso. En la línea de código 4, se especifica al motor que use la base de datos previamente creada.

```
3 •  create database game default character set utf8mb4 default collate utf8mb4_spanish_ci;
4 •  use game;
```

Código 2.1. Sentencias SQL para crear la base de datos

En el Código 2.2, se muestran las sentencias para crear una tabla que guardará información acerca de los segmentos de código que se mostrarán como desafío educativo en el minijuego de batalla naval.

```
95 •  create table Codec(
96      idCodec int NOT NULL AUTO_INCREMENT,
97      statement varchar(500),
98      codecPath varchar(100),
99      points int default 0,
100     feedbackDescr varchar(5000),
101     idChapter int,
102     PRIMARY KEY (idCodec),
103     FOREIGN KEY (idChapter) REFERENCES Chapter(idChapter)
104     );
```

Código 2.2. Sentencias SQL para crear la tabla `codec`

En la línea 95 se especifica el nombre de la tabla, que en este caso es `codec`, en la que se definen las columnas que la componen dentro de las líneas 96 a la 101. La línea 96 establece que el identificador único de la tabla es `idCodec`, que es de tipo entero, y mediante las palabras clave `NOT NULL` y `AUTO_INCREMENT`, se especifica que este campo no puede ser nulo y que el identificador incrementará automáticamente cada vez que sean ingresados nuevos registros. En las líneas 97, 98 y 100 se definen las columnas denominadas `statement`, `codecPath` y `feedbackDescr` respectivamente. La columna `statement` permite almacenar el enunciado relacionado con el segmento de código que será mostrado al jugador durante una partida. La columna `codecPath` permite almacenar la ruta (*path*) en donde se encuentra almacenada la imagen que contiene el segmento de código. La columna `feedbackDescr` permite almacenar cadena de caracteres que contiene la retroalimentación que se mostrará a los jugadores para cada segmento de código.

En las líneas 99 y 101 se definen las columnas denominadas `points` e `idChapter`, respectivamente. La columna `points` permite almacenar el puntaje que le será asignado al jugador por responder correctamente. Mientras que, la columna `idChapter` es la clave foránea que permite conocer el capítulo al cual pertenece cada segmento de código y que relaciona la tabla `codec` con la tabla `chapter`.

Se debe mencionar que las demás tablas que componen la base de datos fueron implementadas de manera similar al proceso detallado anteriormente. Todos los *scripts* de la base de datos se encuentran en el ANEXO F.

Adicionalmente, se han creado varios procedimientos almacenados que pueden ayudar a mejorar el rendimiento ya que se necesita enviar menos información entre el servidor y el cliente. En el Código 2.3, se muestran las sentencias SQL que se utilizan para realizar la autenticación de los usuarios y permitir el ingreso al videojuego. En la línea 17, se crea el procedimiento almacenado utilizando el comando `create procedure`, el cual se denomina `sp_Login` que recibe dos parámetros de entrada llamados `nickN` y `pass` que son el nombre de usuario y su contraseña, respectivamente.

```
17  create procedure sp_Login (nickN varchar(50),pass varchar(20))
18  begin
19      declare counter int;
20      declare result varchar(100);
21
22      set counter = (select idAdmin from administrator where nickname=nickN and passwd=pass);
23      if counter>0 then
24          set result = concat(counter,';Administrador');
25      else
26          set counter = (select idTeacher from teacher where nickname=nickN and passwd=pass);
27          if counter>0 then
28              set result = concat(counter,';Profesor');
29          else
30              set counter = (select idStudent from student where nickname=nickN and passwd=pass);
31              if counter>0 then
32                  set result = concat(counter,';Estudiante');
33              else
34                  set counter = 0;
35                  set result = concat(counter,';Usuario o Contraseña incorrecto');
36              end if;
37          end if;
38      end if;
39
40  select result;
41  end |
```

Código 2.3. Procedimiento almacenado para autenticación de usuarios

En la línea 18 y 41 se utiliza la sintaxis `BEGIN...END` que permite escribir sentencias compuestas que serán parte de los procedimientos almacenados.

En la línea 19 y 20 se declaran dos variables llamadas `counter` y `result`. La variable `counter` almacena el número de resultados generados al realizar una consulta a las tablas `administrator`, `teacher` y `student`. Mientras que, la variable `result` almacena una cadena de caracteres concatenando la variable `counter` junto con el tipo de usuario que se ha autenticado.

Entre las líneas 22 y 38 se utilizan bucles anidados para determinar si el usuario cuyas credenciales existe dentro de alguna de las tablas de usuarios de la base de datos. En este procedimiento almacenado, se realiza una primera consulta a la tabla `administrator` para determinar si el usuario maneja un perfil de Administrador. Si la consulta resulta afirmativa el contador incrementa y sale del bucle, caso contrario se realiza una consulta a la tabla `teacher` para determinar si el usuario maneja un perfil de Profesor. Si esta consulta resulta afirmativa el contador incrementa y sale del bucle, caso contrario se realiza una consulta a la tabla `student` para determinar si el usuario maneja un perfil de Estudiante. Si la consulta resulta afirmativa el contador incrementa y sale del bucle, caso contrario se determina que el usuario no está registrado en el sistema. En la línea 40, se utiliza el comando `select` para devolver el valor almacenado en la variable `result` y determinar el perfil de usuario que trata de ingresar al videojuego.

Así como el procedimiento almacenado mostrado anteriormente, se tienen varios procedimientos almacenados que ayudan con diferentes tareas en las cuales el videojuego requiere extraer, insertar o consultar datos para cumplir con la funcionalidad de cada componente del videojuego.

Como ejemplo se presenta el procedimiento almacenado que permite seleccionar la información acerca de los segmentos de código que serán utilizados en el minijuego de batalla naval. En el Código 2.4 se muestran las sentencias SQL que se utilizan para extraer los identificadores de los registros de la tabla `codec`.

En la línea de código 6, se crea el procedimiento almacenado utilizando el comando `create procedure`, el cual se denomina `sp_GetRandomCodecs`, y recibe un parámetro de entrada de tipo de dato `int` llamado `idChapterX` que es el identificador del capítulo del cual se quiere seleccionar los segmentos de código. En la línea 8, 9 y 10 se declaran tres variables utilizando el comando `declare`. Estas son `numCodecs`, `minNumCodec` y `result`. Posteriormente, en la línea 11 se asigna el valor de 30 a la variable `minNumCodec` utilizando el comando `set`, el cual es definido para limitar la extracción de registros de la base de datos. La variable `result` será utilizada posteriormente para almacenar la lista de identificadores que serán seleccionados.

```
1  use game;
2  -- ////////////////////////////////////////// BATTLESHIP TRIVIA CODEC PROCEDURE ////////////////////////////
3  delimiter ;
4  drop procedure if exists sp_GetRandomCodecs;
5  delimiter |
6  create procedure sp_GetRandomCodecs (idChapterX int)
7  begin
8      declare result varchar(50);
9       declare numCodecs int;
10     declare minNumCodecs int;
11     set minNumCodecs = 30;
12     SELECT COUNT(idCodec) into numCodecs FROM codec where idChapter=idChapterX;
13     -- Crea tabla temporal
14     drop table if exists temp1;
15     if numCodecs <= minNumCodecs then
16         create temporary table temp1 select idCodec as id from codec where idChapter=idChapterX order by rand();
17         set result = (select group_concat(id separator ',')   from temp1);
18     else
19         create temporary table temp1 select idCodec as id from codec where idChapter=idChapterX order by rand()
20         limit minNumCodecs;
21         set result = (select group_concat(id separator ',')   from temp1);
22     end if;
23 select result;
24 end |
```

Código 2.4. Procedimiento almacenado para extracción de datos de la tabla `codec`

En la línea 12, utilizando el comando `select count` se asigna el valor a la variable `numCodecs`, la cual almacena el número total de registros que tiene la tabla `codec` cuyo identificador de capítulo sea igual que el parámetro ingresado a este procedimiento almacenado.

Entre las líneas de código 15 y 22, se utilizan bucles anidados para determinar si existe un número mayor a 30 registros en la tabla `codec`.

En este caso de que la tabla tenga un número menor a 30 registros, se seleccionan todos los registros que posea dicha tabla, caso contrario se seleccionan al azar un máximo de 30 identificadores de la tabla `codec`.

En las líneas de código 16 y 19, se utiliza el comando `select` para escoger los identificadores y almacenarlos en una tabla temporal para posteriormente concatenarlos y almacenarlos en la variable `result` que retorna a la aplicación para satisfacer la llamada a este procedimiento almacenado.

De manera similar al proceso detallado anteriormente, han sido implementados varios procedimientos almacenados que ayudan a la consulta, inserción y extracción de datos que se necesiten dentro del videojuego. Los *scripts* en los que están detallados estos procedimientos almacenados se encuentran en el ANEXO F.

2.3.2. INTERFACES ADMINISTRATIVAS

2.3.2.1. Login

Esta interfaz administrativa permite el inicio de sesión de un usuario en el videojuego. A esta interfaz está asociado un *script* que rige el comportamiento de los componentes gráficos del *Login*, este *script* se denomina `AdminGeneral.cs`.

En el fragmento de código de la clase `AdminGeneral.cs` que se observa en el Código 2.5, se utiliza una corutina[40] que permite comunicarse con el servidor de base de datos y realizar la verificación de los usuarios utilizando el procedimiento almacenado para autenticación de usuarios mostrado en la Sección 2.3.1 de este documento.

```
// Corutina para iniciar sesion
IEnumerator co_Login(string nickname, string password) {
    WWWForm form = new WWWForm();
    form.AddField("loginUser", nickname);
    form.AddField("loginPass", password);

    //localhost debe ser sustituido por la IP o direccion del servidor que almacena los scripts PHP
    using (UnityWebRequest www = UnityWebRequest.Post("http://localhost/game/Login.php", form)) {
        yield return www.SendWebRequest();

        if (www.isNetworkError || www.isHttpError) {
            informe(www.error);
        } else {
            BuildDataLogin(www.downloadHandler.text);
        }
    }
}
```

Código 2.5. Corutina `co_Login` de la clase `AdminGeneral.cs`

Para realizar la comunicación con el servidor de base de datos se utiliza un *script* escrito en el lenguaje de PHP denominado `Login.php`, mismo que se muestra en el Código 2.6.

En la línea 3, se puede observar la inclusión del *script* llamado `Connection.php`, el cual posee los parámetros de conexión que se utilizan para establecer la comunicación con la base de datos, como se observa en el Código 2.7.

En las líneas de código 4 y 7 del Código 2.7, se definen las variables de conexión con las credenciales del servidor de base de datos. En la línea de código 8, se utiliza la función llamada `mysqli()` para abrir una nueva conexión al servidor de MySQL.

Finalmente, en la línea de código 9, la función de conexión a la base de datos retorna la variable `conn` que contiene la respuesta a la petición de conexión.

[40] Corutina: Función que tiene la habilidad de pausar su ejecución y devolver el control a Unity para luego continuar donde lo dejó en el siguiente cuadro (*frame*).

```php
<?php
// se agrega el archivo de conexion a la Base de datos
include("Connection.php");
// variables
$loginUser = $_POST["loginUser"];
$loginPass = $_POST["loginPass"];
// Se establece la conexion
mysqli_report(MYSQLI_REPORT_STRICT);
try{
    $conn = connectdb();
}
catch(Exception $e){
    echo "fallo";
    exit();
}
// call sp_Login('edy','edy123');
$sql = "call sp_Login('".$loginUser."','".$loginPass."');";

$result = $conn->query($sql);

if ($result->num_rows > 0) {
  // output data of each row
    while($row = $result->fetch_assoc()) {
        echo $row["result"];
    }
}
else {
    echo "No existe el usuario";
}
$conn->close();
?>
```

Código 2.6. *Script* `Login.php`

```php
<?php
    function connectdb(){

        $servername = "192.168.1.2";
        $username = "Administrador";
        $password = "admin20";
        $dbname = "game";
        $conn = new mysqli($servername, $username, $password, $dbname);
        return $conn;
    }
?>
```

Código 2.7. *Script* `Connection.php`

Al igual que este *script* de PHP, se han usado varios *scripts* similares para realizar la comunicación entre el videojuego y la base de datos. Todos estos *scripts* se encuentran detallados en el ANEXO G. Finalmente con la lógica de programación establecida, la interfaz gráfica de *login* es la que se muestra en la Figura 2.22.

Figura 2.22. Interfaz gráfica de *Login* del videojuego

2.3.2.2. Menús administrativos del videojuego

Existen dos diferentes menús que aparecerán de acuerdo con el tipo de usuario que ingrese al sistema. El primer menú se despliega en caso de que el usuario que ingrese al videojuego tenga un perfil de Administrador o Profesor. Mientras que, el segundo menú se muestra en caso de que el usuario que ha ingresado tenga un perfil de Estudiante. Cuando el usuario que ha ingresado al sistema tenga un perfil de Administrador, se muestra un menú que solo tiene activo un botón para realizar la administración de profesores, ya que un usuario con este perfil solo tiene la tarea de registrar, editar o eliminar a un usuario con perfil de Profesor. El menú que se despliega para un perfil de Administrador se muestra en la Figura 2.23.

Figura 2.23. Menú para usuario con perfil de Administrador

Por otro lado, si el usuario que ha ingresado al sistema tiene un perfil de Profesor, también se muestra un menú. Pero en este caso, la interfaz cuenta con cinco botones activos, ya que un usuario con perfil de Profesor puede realizar la administración de usuarios con perfil de Estudiante, preguntas, conceptos, segmentos de código y capítulos de la asignatura. El menú que se despliega para un usuario con perfil de Profesor se muestra en la Figura 2.24.

Figura 2.24. Menú para usuario con perfil de Profesor

Finalmente, si el usuario que ha ingresado al sistema tiene un perfil de Estudiante, se muestra una interfaz que contiene los botones que permiten la selección de cada minijuego, así como el juego de *Tower Defense*. El menú que se despliega para un usuario con perfil de Estudiante se muestra en la Figura 2.25.

Figura 2.25. Menú para usuario con perfil de Estudiante

2.3.2.3. Interfaces para administración de usuarios y contenido educativo.

Existen seis interfaces que permiten realizar la administración de usuarios y contenido educativo en el videojuego, y son:

1. Interfaz para administración de preguntas
2. Interfaz para administración de conceptos
3. Interfaz para administración de segmentos de código
4. Interfaz para administración de capítulos
5. Interfaz para administración de estudiantes
6. Interfaz para administración de profesores

A continuación, se mostrará un resumen del proceso seguido para la implementación de la interfaz para administrar preguntas.

La interfaz que permite realizar la administración de preguntas se muestra en la Figura 2.26. Esta interfaz posee una columna en la que se muestran los enunciados de las preguntas asociadas a un capítulo en específico. Para obtener el listado de preguntas que se despliegan en la tabla de la interfaz, se utiliza una corutina denominada `co_GetQuestions`, que se observa en la línea 48 del Código 2.8. Esta corutina se

encuentra en la clase llamada `AdminQuestion.cs` y recibe como parámetro de entrada el identificador de capítulo que ha sido seleccionado previamente.

Figura 2.26. Interfaz para administración de preguntas

```
48      IEnumerator co_GetQuestions(int idChapter) // Obtener lista de preguntas
49      {
50          ClearListQuestions(); // primero se borra la lista de preguntas de escena
51
52          questions = new List<QuestionPrefab>();
53          WWWForm form = new WWWForm();
54          form.AddField("idChapter", idChapter);
55
56          using (UnityWebRequest www = UnityWebRequest.Post("http://localhost/game/GetQuestions.php", form))
57          {
58              yield return www.SendWebRequest();
59
60              if (www.isNetworkError || www.isHttpError)
61              {
62                  informe(www.error);
63              }
64              else
65              {
66                  informe(www.downloadHandler.text);
67                  Debug.Log("HANDLER" + www.downloadHandler.text);
68                  BuildQuestion(www.downloadHandler.text);
69                  StartCoroutine(co_GetAlternatives());
70              }
71          }
72      }
```

Código 2.8. Corutina `co_GetQuestions`

En la línea 56, esta corutina llama al *script* `GetQuestions.php`, el cual realiza una petición a la base de datos utilizando el procedimiento almacenado denominado `sp_LoadQuestions`. Este procedimiento almacenado realiza una consulta a la tabla que almacena los registros de las preguntas, seleccionando todos los registros cuyo identificador de capítulo concuerde con el identificador del capítulo que ha seleccionado el jugador. La respuesta que retorna este procedimiento almacenado se procesa en la

corutina `co_GetQuestions` utilizando el método llamado `BuildQuestion`. En este método se separa la cadena de caracteres obtenida desde la base de datos y se asignan los valores de cada registro a una instancia de un objeto de tipo `Question`, que será el que se muestre posteriormente en la tabla de la interfaz.

Por otra parte, cuando el usuario selecciona el botón “Nuevo”, se despliega un panel con varios cuadros de texto y dos botones, como se observa en la Figura 2.27.

Figura 2.27. Panel para ingreso de nuevas preguntas

La función de este panel es permitir el ingreso de una nueva pregunta a la base de datos. Para crear un nuevo registro de tipo pregunta en la base de datos, dentro de la clase `AdminQuestion.cs` se utiliza la corutina denominada `co_CreateQuestion` que se muestra en el Código 2.9.

```
1 referencia
IEnumerator co_CreateQuestion(string question, string answer, string option1, string option2,
    string option3, string feedback, int idChapter)
{
    WWWForm form = new WWWForm();
    form.AddField("question", question);
    form.AddField("answer", answer);
    form.AddField("option1", option1);
    form.AddField("option2", option2);
    form.AddField("option3", option3);
    form.AddField("feedback", feedback);
    form.AddField("idChapter", idChapter);
    using (UnityWebRequest www = UnityWebRequest.Post("http://localhost/game/CreateQuestion.php", form))
    {
        yield return www.SendWebRequest();

        if (www.isNetworkError || www.isHttpError)
        {
            Debug.Log(www.error);
            informe(www.error);
        }
        else
        {
            string resp = www.downloadHandler.text;
            if (resp == "#REG#")
            { // si se agrega me muestra la lista actualizada
                StartCoroutine(co_GetQuestions(idChapter));
            }
```

Código 2.9. Fragmento de código de la corutina `co_CreateQuestion`

Esta corutina recibe como parámetros de entrada el enunciado de la pregunta, su respuesta, tres alternativas erróneas de respuesta, información acerca de la pregunta (*feedback*) y el identificador del capítulo al que pertenece la pregunta. En la línea de código 115, esta corutina hace un llamado al *script* `CreateQuestion.php`, el cual realiza una petición a la base de datos utilizando el procedimiento almacenado denominado `sp_RegQuestion`, que se encarga de almacenar el nuevo registro dentro de la tabla `question`.

Por otra parte, cuando un usuario selecciona el botón que permite la edición, se despliega un panel en donde se muestran todos los campos relacionados con la pregunta que se desea editar, como se puede observar en la Figura 2.28.

Figura 2.28. Edición de un registro de tipo pregunta

Para realizar la edición de un registro de tipo pregunta de la base de datos, se utiliza la corutina denominada `co_UpdateQuestion` que se encuentra dentro de la clase `QuestionPrefab.cs` y se muestra en el Código 2.10.

```
66      IEnumerator co_UpdateQuestion()
67      {
68          WWWForm form = new WWWForm();
69          form.AddField("idQuestion", idQuestion);
70          form.AddField("question", question);
71          form.AddField("answer", answer);
72          form.AddField("option1", option1);
73          form.AddField("option2", option2);
74          form.AddField("option3", option3);
75          form.AddField("feedback", feedback);
76          using (UnityWebRequest www = UnityWebRequest.Post("http://localhost/game/UpdateQuestion.php", form))
77          {
78              yield return www.SendWebRequest();
79              if (www.isNetworkError || www.isHttpError)
80              {
81                  Debug.Log(www.error);
82                  informe(www.error);
83              }
84              else
85              {
86                  string resp = www.downloadHandler.text;
87                  if (resp == "#REG#")
88                  { // si se agrega me muestra la lista actualizada
89                      informe("modificado? " + resp);
90                      questionListText.text = question;
91                  }
```

Código 2.10. Fragmento de código de la corutina `co_UpdateQuestion`

En la línea 76, esta corutina hace un llamado al *script* de PHP `UpdateQuestion.php`, el cual realiza una petición a la base de datos utilizando el procedimiento almacenado denominado `sp_UpdateQuestion`. Este procedimiento almacenado realiza la actualización del registro de la pregunta y de sus alternativas de respuesta utilizando como parámetro de búsqueda al identificador de dicha pregunta.

Cuando un usuario elige el botón que permite borrar un registro, se despliega una ventana de alerta que permite confirmar o desistir la eliminación del registro, como se muestra en la Figura 2.29.

Figura 2.29. Ventana de confirmación de eliminación de registro de tipo pregunta

Para eliminar un registro de tipo pregunta de la base de datos, en la clase `AdminQuestion.cs` se crea la corutina denominada `co_EraseQuestion` que se muestra en el Código 2.11.

```
1 referencia
IEnumerator co_EraseQuestion(int idQuestion)
{
    WWWForm form = new WWWForm();
    form.AddField("idQuestion", idQuestion);


    using (UnityWebRequest www = UnityWebRequest.Post("http://localhost/game/DeleteQuestion.php", form))
    {
        yield return www.SendWebRequest();

        if (www.isNetworkError || www.isHttpError)
        {
            Debug.Log(www.error);
            informe(www.error);

        }
        else
        {
            StartCoroutine(co_GetQuestions(idChapter));
            informe("Borrado? > " + www.downloadHandler.text);

        }
    }
}
```

Código 2.11. Corutina `co_EraseQuestion`

Esta corutina recibe como parámetros de entrada el identificador de la pregunta que se desea eliminar de la base de datos. En la línea 146, esta corutina hace un llamado al *script* `DeleteQuestion.php`, el cual realiza una petición a la base de datos utilizando el

procedimiento almacenado denominado `sp_DeleteQuestion`. Este procedimiento almacenado es el encargado de eliminar el registro de la base de datos.

De manera similar al proceso descrito anteriormente se realiza la implementación de las demás interfaces que permiten realizar la administración del contenido educativo. Los procedimientos almacenados y *scripts* codificados en lenguaje PHP que han sido mencionados pueden ser encontrados en el ANEXO F y ANEXO G, respectivamente.

2.3.3. MINIJUEGO DE TRIVIA

Al iniciar una partida en el minijuego de trivia, se muestra una interfaz similar a la que se observa en la Figura 2.30.

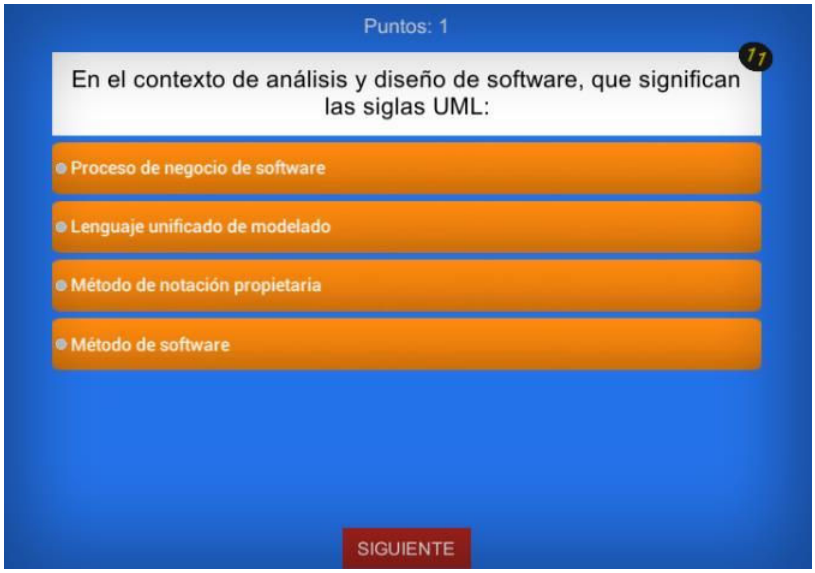

Figura 2.30. Ejemplo de pregunta en el minijuego de trivia

En este minijuego deben mostrarse siete preguntas por cada partida, las cuales son seleccionadas de la base de datos utilizando el procedimiento almacenado llamado `sp_GetRandomQuestions`. Este procedimiento almacenado selecciona de manera aleatoria siete identificadores de pregunta de acuerdo al capítulo que el estudiante haya seleccionado previo al ingreso a la partida. Este procedimiento almacenado es invocado por el *script* `GetRandomQuestions.php`, y este a su vez, es llamado dentro de la clase `GameManager.cs` por la corutina `co_GetRandomQuestions`, como se observa en el Código 2.12.

Por otra parte, una vez que se obtienen los siete identificadores de las preguntas que van a ser utilizadas, se debe extraer la información de cada una de ellas de la base de datos, para lo cual se utiliza otra corutina denominada `co_GetNextQuestion`, que se muestra en el Código 2.13.

```
1 referencia
IEnumerator co_GetRandomQuestions(int idChapter) {
    WWWForm form = new WWWForm();
    form.AddField("idChapter", idChapter);
    using (UnityWebRequest www = UnityWebRequest.Post("http://localhost/game/GetRandomQuestions.php", form)) {
        yield return www.SendWebRequest();
        if (www.isNetworkError || www.isHttpError) {
            Debug.Log(www.error);
        } else {
            Debug.Log("Ids Received: "+www.downloadHandler.text);
            _idsquestion = www.downloadHandler.text.Split(',');
        }
    }
}
```

Código 2.12. Corutina `co_GetRandomQuestions`

```
1 referencia
IEnumerator co_GetNextQuestion(int idQuestion) {
    WWWForm form = new WWWForm();
    form.AddField("idQuestion", idQuestion);
    using (UnityWebRequest www = UnityWebRequest.Post("http://localhost/game/GetAQuestion.php", form)) {
        yield return www.SendWebRequest();
        if (www.isNetworkError || www.isHttpError) {
            Debug.Log(www.error);
        } else {
            string[] qa = www.downloadHandler.text.Split('Ç');
            BuildQuestionAlternative(qa);
        }
    }
}
```

Código 2.13. Corutina `co_GetAQuestion`

Esta corutina llama al *script* `GetAQuestion.php` que se encarga de extraer los datos de cada pregunta, así como sus alternativas de respuesta utilizando el procedimiento almacenado denominado `sp_GetAQuestion`. Ambas corutinas son ejecutadas al iniciar una partida de trivia.

En la línea 123 de la corutina `co_GetAQuestion`, se llama al método `BuildQuestionAlternative`, el cual recibe como parámetros de entrada a los datos obtenidos desde la base. Este método permite procesar la cadena de texto y asociar los atributos de la pregunta a una instancia de un objeto de la clase `Question`. Mientras que, las opciones de respuesta son asociadas a una estructura de datos llamada `Alternative`. Estos objetos serán mostrados en cada desafío educativo en la interfaz presentada anteriormente. Los procedimientos almacenados y *scripts* codificados en lenguaje PHP que han sido mencionados pueden ser encontrados en el ANEXO F y ANEXO G, respectivamente.

2.3.4. MINIJUEGO DE BATALLA NAVAL

Al iniciar una partida en el minijuego de batalla naval, se muestra una interfaz similar a la que se observa en la Figura 2.31, en donde el jugador inicia la partida realizando la colocación de sus barcos en el tablero.

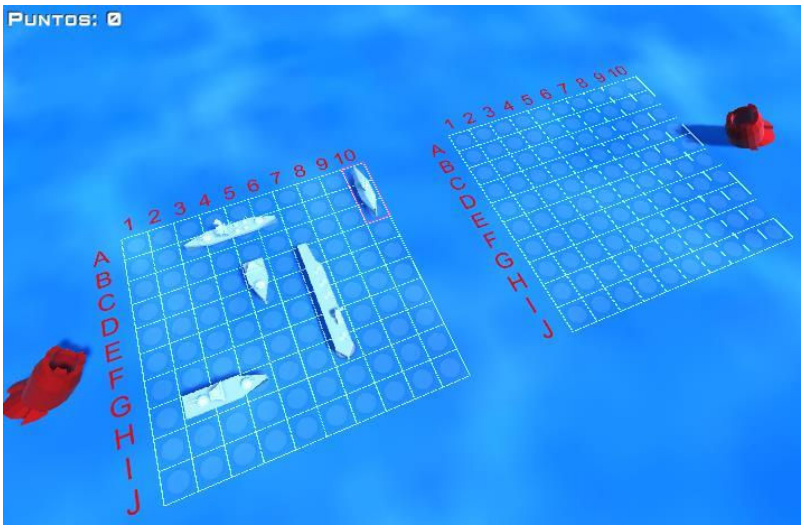

Figura 2.31. Inicio de partida en minijuego de batalla naval - Colocación de barcos

Durante la fase de colocación de barcos, se utilizan algunos objetos prefabricados para la creación de esta escena. En la Figura 2.32 se muestran a los objetos prefabricados más relevantes. Estos son objetos han sido obtenidos de la tienda de Unity llamada *Asset Store*[41] y son creados usando los objetos base de Unity (Cubos, rectángulos, cilindros, esferas, entre otros).

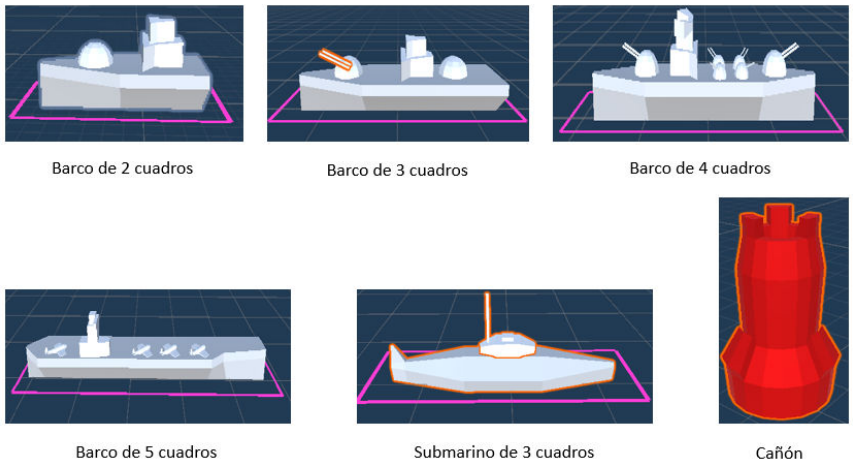

Figura 2.32. Objetos prefabricados usados durante la colocación de barcos

La lógica de programación que está asociada a la colocación de barcos en el tablero es codificada en el método `FixedUpdate`, el cual se encarga del manejo de varios de los eventos principales de este minijuego. Dentro de este método se debe realizar la distinción entre el turno de colocación de barcos del jugador y de la computadora. En el Código 2.14

[41] *Asset Store*: Es una biblioteca de objetos, texturas, modelos y animaciones los cuales pueden ser comerciales o gratuitos y, son creados por Unity Technologies y miembros de la comunidad [30].

y Código 2.15 se muestra la lógica de programación utilizada para realizar la colocación de barcos del jugador.

```
154     void FixedUpdate()
155     {
156         if (shouldExecute)
157         {
158             ////////////////////////////////
159             // ubicacion de botes
160             ////////////////////////////////
161             //caso para el jugador
162             if (player == 1)
163             {
164                 if (overObject != null)
165                 {
166                     //solo selecciona un cuadro del mapa del jugador
167                     if (overObject.tag == "tilePlayer")
168                     {
169                         if (click == 0)
170                         {
171                             boats_player1[selectedBoat].transform.position = overObject.transform.position;
172                         }
173                         else if (click == 1)
174                         {
175                             //se obtiene la direccion de rotacion para el barco
176                             Vector3 dir = overObject.transform.position - boats_player1[selectedBoat].transform.position;
177                             // restringir que un barco no se coloque en diagonal
178                             if (Mathf.Abs(dir.z / dir.x) > 1e5 || Mathf.Abs(dir.z / dir.x) < 0.1)
179                             {
180                                 if (dir!=Vector3.zero) {
181                                     boats_player1[selectedBoat].transform.forward = dir;
182                                 }
183                             }
184                         }
```

Código 2.14. Lógica de programación para colocación de barcos (Parte 1)

```
185         else if (click == 2)
186         {
187             //solo en caso que sea posible ubicar un bote en el mapa, se crea un nuevo bote
188             if (canPlace)
189             {
190                 boats_player1[selectedBoat].transform.GetChild(1).GetComponent<SimpleFloating>().enabled = true;
191                 boats_player1[selectedBoat].transform.GetChild(0).GetComponent<Boat>().disableCanvas();
192                 //incrementa el numero del bote seleccionado
193                 selectedBoat += 1;
194                 //cuando ya no hay mas botes para colocar
195                 if (selectedBoat >= boats.Length)
196                 {
197                     player = 2;
198                     selectedBoat = 0;
199                     //deshabilita el render de los botes
200                     disableBoats(boats_player1);
201                     disableBoats(boats);
202                     //Inicio de colocacion de los botes para la computadora
203                     boats_player2[selectedBoat] = GameObject.Instantiate(boats[selectedBoat], transPos.position, transPos.rotation) as GameObject;
204                     boats_player2[selectedBoat].transform.SetParent(c_p2);
205                 }
206                 else
207                 {
208                     boats_player1[selectedBoat] = GameObject.Instantiate(boats[selectedBoat], transPos.position, transPos.rotation) as GameObject;
209                     boats_player1[selectedBoat].transform.SetParent(c_p1);
210                 }
211                 click = 0;
212                 canPlace = false;
213             }
```

Código 2.15. Lógica de programación para colocación de barcos (Parte 2)

El proceso de colocación de barcos del jugador se realiza en base al número de clics que el jugador realiza sobre el tablero tomando en cuenta el cuadro del tablero que haya seleccionado con el primer clic. En las líneas 164 y 167 del Código 2.14, se realiza la verificación que permite determinar si algún cuadro del tablero ha sido ocupado y también si dicho cuadro pertenece al tablero del jugador. Si estas dos condiciones se cumplen, el

primer barco que va a ser colocado se le asigna las coordenadas de posición correspondientes al cuadro del tablero que el usuario se encuentre seleccionando con el puntero del *mouse*. Este proceso se observa entre las líneas de código 169 y 172.

Cuando el jugador realice el primer clic sobre el cuadro del tablero en el que se desea colocar el barco, se establece la posición del cuadro seleccionado al bote y se procede a obtener la dirección en la que se desea ubicar el bote, restringiendo que pueda colocarse en diagonal dentro del mapa. Este proceso se realiza entre las líneas de código 173 y 184.

Cuando el usuario realice el segundo clic sobre el cuadro del tablero del jugador, el bote se ubica en la posición y dirección que han sido especificadas y se repite el mismo procedimiento para colocar el siguiente bote sobre el tablero. En caso de que todos los botes del jugador ya hayan sido colocados, se realiza el cambio de jugador hacia el contrincante (computadora) para que realice la colocación de sus barcos.

A diferencia de la colocación de barcos del usuario, la colocación de barcos del contrincante (computadora) se define de manera aleatoria, tanto la posición como la rotación de cada uno de los barcos toman valores aleatorios dentro del tablero del contrincante.

Una vez que los barcos hayan sido colocados en los tableros, se procede con la fase de ataque. Durante esta fase se muestra al usuario una interfaz similar a la que se observa en la Figura 2.33.

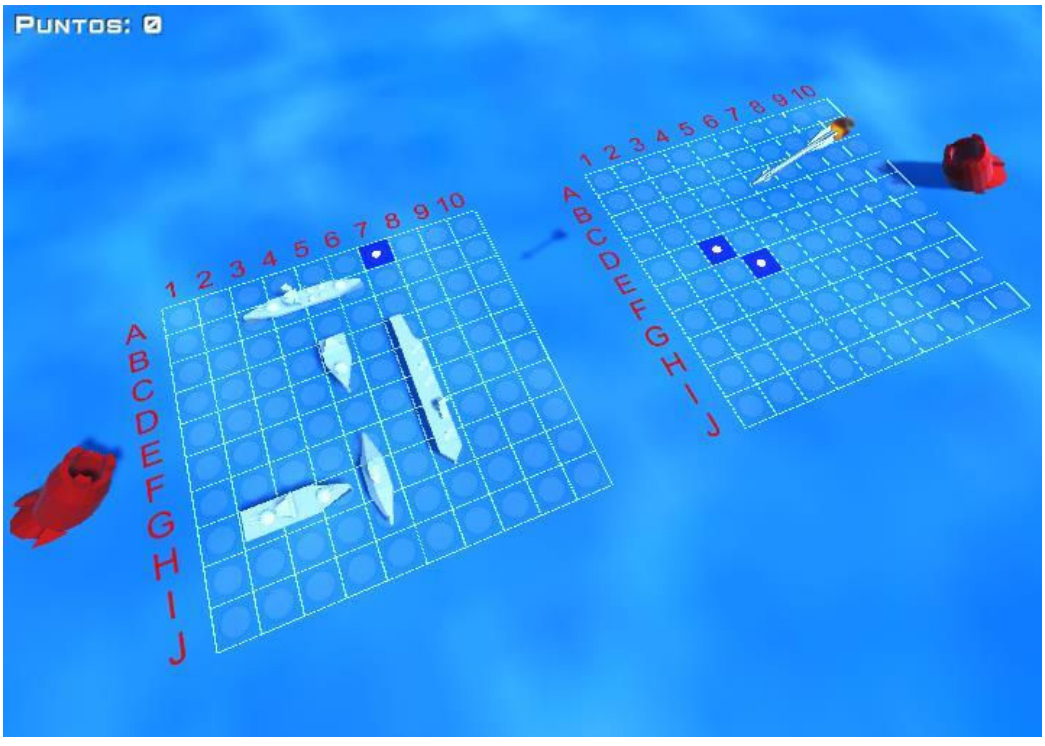

Figura 2.33. Minijuego de batalla naval – Fase de ataque

De igual forma que en la fase de colocación de barcos, la lógica de programación asociada con el modo de ataques por turnos ha sido codificada dentro del método `FixedUpdate` de la clase `GameManagerBattleship`. Las líneas de código asociadas con el turno de ataque del jugador se muestran en el Código 2.16.

```
311             ////////////////////////
312             //Modo de juego por turnos
313             ////////////////////////
314             else if (player == -1)
315             {
316                 //Solo muestra los botes del jugador
317                 enableBoats(boats_player1);
318                 disableBoats(boats_player2);
319                 if (clickObject != null)
320                 {
321                     if (click > 0 && clickObject.tag == "tileCOM" && !shoot)
322                     {
323                         //muestra la marca de ataque en el mapa
324                         shooting_square.SetActive(true);
325                         shooting_square.transform.position = clickObject.transform.position;
326                         shootScript.p = 1;
327                         player = -1;
328                     }
329                     else
330                     {
331                         shooting_square.SetActive(false);
332                     }
333                 }
334                 if (checkGameOver(boats_player1, boats_player2))
335                 {
336                     shouldExecute = false;
337                     if (m_quizDB != null) {
338                         StartCoroutine(SetGameMatch());
339                     }
340                     panelGameOver.SetActive(true);
341                     panelGameOver.transform.GetChild(0).GetComponent<Text>().text = "Tu puntuación es: " + score.ToString();
342                     panelChallenge.SetActive(false);
343                 }
344             }
```

Código 2.16. Lógica de programación para turno de ataque del jugador

En primer lugar, los barcos que han sido colocados en el tablero del contrincante (computadora) no deben ser visibles para el jugador, mientras que los barcos del jugador siempre deben ser visibles en la escena del minijuego. Estas acciones se realizan en las líneas de código 317 y 318, respectivamente.

Luego, se verifica que el jugador haya seleccionado un objetivo dentro del tablero del contrincante y se determinan las coordenadas del cuadro del tablero que ha sido seleccionado por el jugador para realizar el ataque, ya que estas coordenadas corresponden a la posición final que se le asignará al misil para el ataque. Este proceso se realiza entre las líneas de código 319 y 328.

Además, entre las líneas de código 334 y 343, se verifica si una partida ha llegado a su fin. Para este proceso se toma como referencia las listas que contienen a los botes de cada jugador; y se analiza si en alguna de las dos listas todos los botes han sido destruidos, utilizando el método llamado `checkGameOver`.

Durante la fase de ataque, cuando algún misil del contrincante colisiona con un bote del jugador se muestra una interfaz similar a la que se muestra en la Figura 2.34. En caso de

que el jugador responda correctamente al desafío que se le presenta, su barco no recibirá daño. Pero si responde de manera incorrecta, se le restará un punto de vida a su barco.

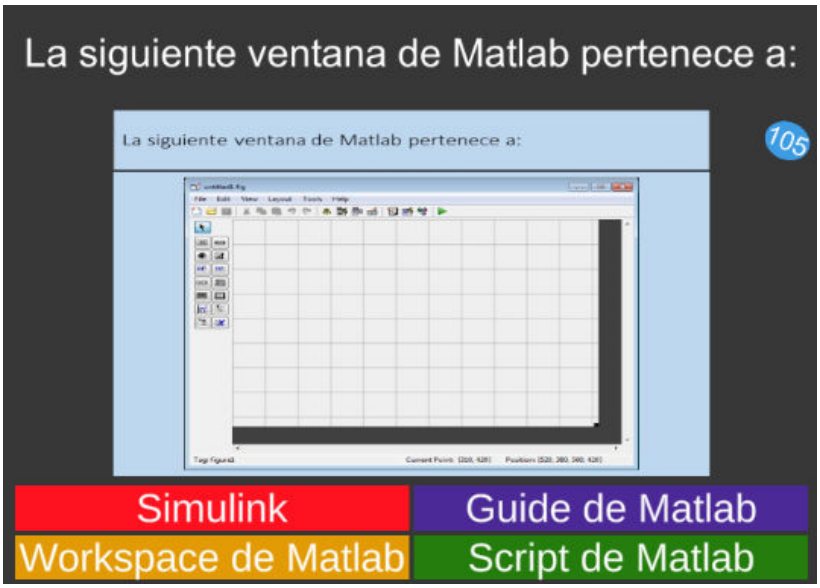

Figura 2.34. Ejemplo de segmento de código en el minijuego de batalla naval

Para obtener los segmentos de código de la base de datos para mostrarlos en la interfaz presentada anteriormente, se sigue un proceso similar al que fue detallado en el minijuego de trivia presentado en la Sección 2.3.3 de este documento.

2.3.5. MINIJUEGO DEL AHORCADO

Al iniciar una partida en el minijuego del ahorcado, se muestra una interfaz similar a la que se observa en la Figura 2.35.

Figura 2.35. Ejemplo de desafío en el minijuego del ahorcado

En este minijuego se deben mostrar cinco palabras relacionadas a un concepto durante cada partida, por lo que se ha creado un procedimiento almacenado llamado `sp_GetAConcept`. Este procedimiento almacenado recibe como parámetros de entrada el identificador de capítulo y el número de palabras que se necesita, siendo para este caso cinco palabras. Como resultado se obtiene un concepto al azar de la base de datos que posea cinco opciones de respuesta.

Para mostrar el desafío educativo en la interfaz se utiliza la corutina llamada `StartGame` que se observa en el Código 2.17.

```
2 referencias
62  IEnumerator StartGame()
63  {
64      MainDialogue.SetActive(true);
65      FinalDialogue.SetActive(false);
66      currentHangmanSprite = 0;
67      HangmanImage.sprite = HangmanSprites[currentHangmanSprite];
68      yield return new WaitUntil(() => conceptAlready == true);
69      PickRandomWordkey();
70      if (currWordkey.Word.Contains(' ')) {
71          UpdateAnswerText(' ');
72      } else if (currWordkey.Word.Contains('.')) {
73          UpdateAnswerText('.');
74      }
75  }
```

Código 2.17. Corutina `StartGame`

En la línea 66 y 67, se establece la imagen inicial de la secuencia de “ahorcado” que será mostrada cuando el jugador falle en la elección de las letras utilizadas para descubrir la palabra oculta. La secuencia de imágenes se muestra en la Figura 2.36.

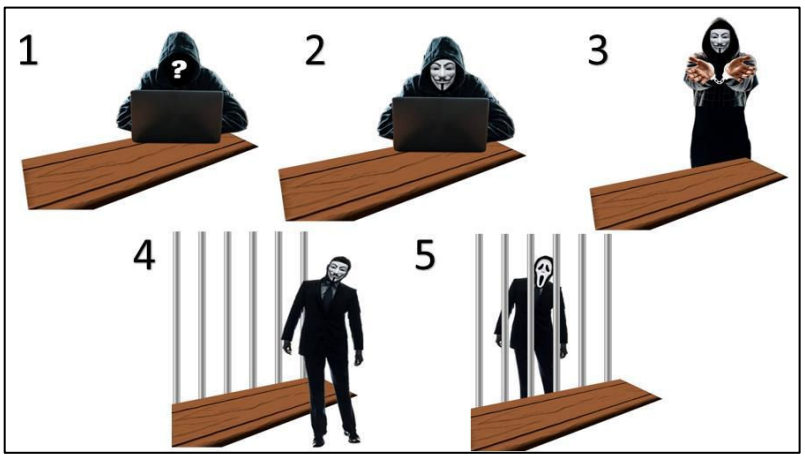

Figura 2.36. Secuencia de imágenes del ahorcado

En la línea 69, se realiza una llamada al método `PickRandomWordkey`, el cual permite seleccionar una de las cinco respuestas de manera aleatoria para que sea descubierta por el jugador. Entre las líneas de código 70 y 74, se realiza la verificación y ajuste para las

palabras contengan espacios vacíos o puntos para que sean desplegadas de manera correcta en la interfaz. Este proceso se realiza utilizando el método `UpdateAnswerText`.

Para que el jugador ingrese las letras que le permitan descubrir la palabra del desafío, se genera un teclado virtual como se muestra en la interfaz. El método llamado `OnGuessSubmited` se encarga de determinar la acción a realizarse de acuerdo a la letra que el usuario haya seleccionado. Este método se muestra en el Código 2.18.

```
282     public void OnGuessSubmitted(Button button) {
283         char letter = button.GetComponentInChildren<Text>().text.ToCharArray()[0];
284         // se verifica si la respuesta contiene el caracter seleccionado,
285         // caso contrario se verifica el numero de intentos faltantes o se dibuja siguiente condicion de ahorcado
286         if ( answer.Contains(letter) ) {
287             // si contiene el caracter seleccionado,
288             // se llamada al metodo para actualizar la palabra de respuesta.
289             UpdateAnswerText(letter);
290             // muestra el dialogo de Ganar si cumple la condicion
291             if ( CheckWinCondition() ) {
292                 StopAllCoroutines();
293                 currWordkey.IsFound = true;
294                 StartCoroutine(ShowImageIsCorrect(true));
295                 Debug.Log("Palabra encontrada !");
296                 textScore.text = "Puntos: "+(currConcept.WordsFound()*10).ToString();
297                 //StopCoroutine(StartCountdown());
298                 NextWord();
299             } else {
300                 currWordkey.IsFound = false;
301             }
302         }
303         else
304         {
305             // muestra el dialogo de Perdida si cumple la condicion
306             if (CheckLoseCondition()) {
307                 StopAllCoroutines();
308                 currWordkey.IsFound = false;
309                 StartCoroutine(ShowImageIsCorrect(false));
310                 Debug.Log("Palabra no encontrada");
311                 NextWord();
312             }
313             else { DrawNextHangmanPart(); }
314         }
315     }
```

Código 2.18. Método `OnGuessSubmited` de la clase `MainScript`

En caso de que la letra que se haya seleccionado pertenezca a la palabra del desafío, se debe mostrar la letra en la interfaz, para lo cual se utiliza el método llamado `UpdateAnswerText`, como se observa entre las líneas de código 286 y 289.

Además, en caso de que el jugador descubra la palabra completa se debe actualizar la puntuación utilizando el método `NextWord`, el cual también permite mostrar la siguiente palabra. Este proceso se realiza entre las líneas de código 296 y 298.

Por otra parte, en caso de que la letra ingresada no sea la correcta, se debe avanzar en la secuencia de imágenes del ahorcado y esto se consigue utilizando el método llamado `DrawNextHangmanPart`, como se muestra en la línea de código 313.

2.3.6. MINIJUEGO DE SOPA DE LETRAS

Al iniciar una partida en el minijuego de sopa de letras, se muestra una interfaz similar a la que se observa en la Figura 2.37.

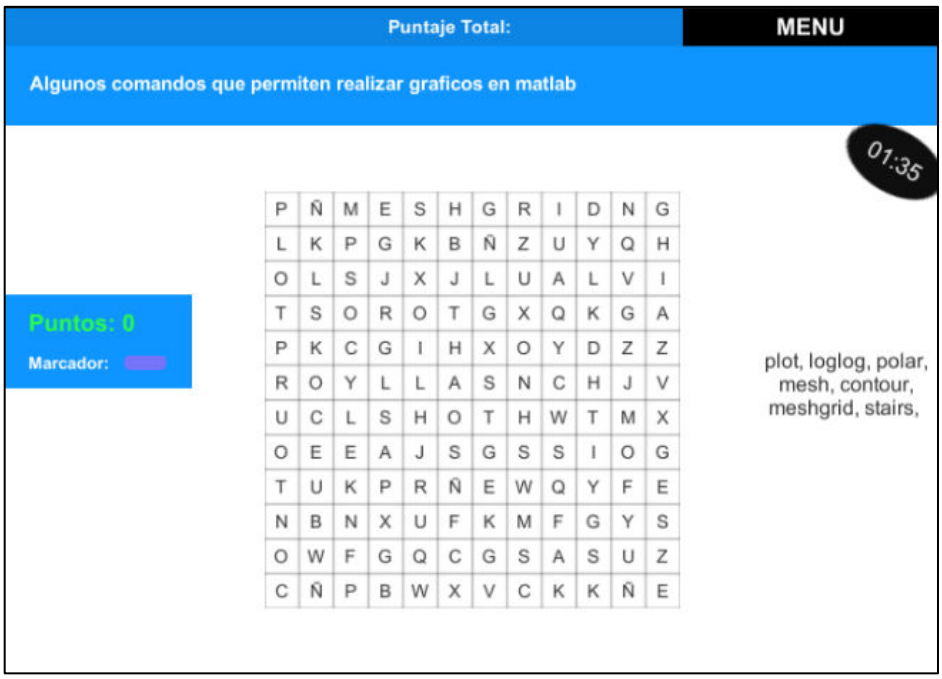

Figura 2.37. Ejemplo de desafío en el minijuego de sopa de letras

En este minijuego deben mostrarse siete palabras asociadas a un concepto durante cada partida, por lo que se ha creado un procedimiento almacenado llamado `sp_GetAConcept`. Este procedimiento almacenado recibe como parámetros de entrada el identificador de capítulo y el número de palabras que se necesita, siendo para este caso siete palabras. Como resultado se obtiene un concepto al azar de la base de datos que posea siete opciones de respuesta.

La corutina llamada `initGame` es la encargada de llevar la mayor parte de los procesos y eventos que se desencadenan durante el videojuego. Para iniciar la partida, en primer lugar, debe dibujarse la cuadrícula en la que se deben ocultar las palabras. Para realizar esta acción se debe fijar un eje de referencia con coordenadas (X, Y) por lo que se definen las variables `actY` y `actX` en las líneas 213 y 215 del Código 2.19. Después en la línea de código 218, se crea un componente gráfico de texto que va a ser almacenado dentro del objeto `charMatrix` como se muestra en la línea de código 219.

Entre las líneas de código 220 y 230 del Código 2.19, se configuran las propiedades necesarias para este componente de texto que va a contener a la letra que se va a mostrar en cada cuadro del tablero. También, se crea un objeto de tipo imagen, como se observa en la línea de código 233, la cual va a permitir dibujar cada uno de los cuadros del tablero. Estos objetos se almacenan en un contenedor llamado `gridMatrix`, como se observa en la línea de código 234. Entre las líneas de código 235 y 238, se configuran las propiedades para cada uno de estos objetos.

```
212             // Se usa la cuadricula para dibujar el rompecabezas en la pantalla
213             float actY = startY;
214             for (int y = 0; y < matrixSizeY; y++) {
215                 float actX = startX;
216                 for (int x = 0; x < matrixSizeX; x++) {
217                     // Creando Caracter individual
218                     Text text = Instantiate(charText, Vector3.zero, Quaternion.identity) as Text;
219                     text.transform.SetParent(charMatrix.transform);
220                     text.GetComponent<RectTransform>().anchoredPosition = new Vector2(actX + x * gridScale, actY - y * gridScale);
221                     text.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);
222                     text.GetComponent<RectTransform>().sizeDelta = new Vector2(gridScale, gridScale);
223                     text.GetComponent<Text>().text = "A";
224                     text.GetComponent<Text>().text = matrix[y, x];
225                     text.GetComponent<Text>().fontSize = 60;
226                     text.GetComponent<Text>().alignment = TextAnchor.MiddleCenter;
227                     text.name = "puzzleChar";
228                     text.tag = "puzzleChar";
229                     text.GetComponent<PuzzleChar>().setMatrixPosition(y, x);
230                     text.GetComponent<PuzzleChar>().setScreenPosition(actX + x * gridScale, actY - y * gridScale);
231
232                     // Creando Imagen Grid cada caracter
233                     Image image = Instantiate(gridImage, Vector3.zero, Quaternion.identity) as Image;
234                     image.transform.SetParent(gridMatrix.transform);
235                     image.GetComponent<RectTransform>().anchoredPosition = new Vector2(actX + x * gridScale, actY - y * gridScale);
236                     image.GetComponent<RectTransform>().localScale = new Vector3(1, 1, 1);
237                     image.GetComponent<RectTransform>().sizeDelta = new Vector2(gridScale, gridScale);
238                     image.tag = "puzzleGrid";
239                 }
240             }
```

Código 2.19. Segmento de código de la corutina `initGame` para generar cuadrícula

Después de generar el tablero, se debe ubicar a cada una de las siete palabras dentro del mismo. Para esto, se utiliza las líneas de código que se muestran en el Código 2.20.

```
155             // iterar en la lista de palabras
156             for (int indexWord = entries.Count - 1; indexWord >= 0; indexWord--) {
157                 for (tries = 1; tries <= maxtries; tries++) {
158                     if (entries[indexWord].Name != "") {
159                         // genera una direction para la palabra
160                         int[] direction = generateDirection();
161                         // genera una position para ubicar la palabra
162                         int[] position = generatePosition(entries[indexWord].Name, direction);
163                         // verifica si la posicion y direccion son correctos (si se ajustan)
164                         if (checkWord(entries[indexWord].Name, position, direction)) {
165                             // ubicar la palabra en la matriz
166                             placeWord(entries[indexWord].Name, position, direction);
167                             // Configura las coordenadas "StartPosition" de la palabra
168                             entries[indexWord].StartPosition = new Vector2(position[0], position[1]);
169                             // Configura las coordenadas "EndPosition" de la palabra
170                             entries[indexWord].EndPosition = new Vector2(position[0] + direction[0] * (entries[indexWord].Length - 1),
171                                 position[1] + direction[1] * (entries[indexWord].Length - 1));
172                             // inicialmente se configura la palabra como no encontrada.
173                             entries[indexWord].Found = false;
174                             // inicialmente se configura la palabra como no usada.
175                             entries[indexWord].Used = true;
176                             // se incrementa el numero de palabras en la matriz (# palabras por encontrar)
177                             wordsToFind++;
178                             break;
179                         }
180                     }
181                 }
182             }
```

Código 2.20. Segmento de código de la corutina `initGame` para colocar las palabras

En primer lugar, se utiliza el método `generateDirection` para definir la dirección en la que se ubicará la palabra dentro del tablero, como se observa en la línea de código 160. Posteriormente, se define la posición en la que se colocará la palabra en el tablero utilizando el método `generatePosition`, como se observa en la línea de código 162. Luego, se utiliza el método `checkWord` para verificar si tanto la posición como la dirección antes definidas son válidas para realizar la colocación de la palabra, como se observa en

la línea de código 164. En caso de que la palabra pueda ser colocada, se utiliza el método `placeWord` para realizar este proceso, como se observa en la línea de código 166. Entre las líneas de código 168 y 175, se definen parámetros importantes como la posición inicial y final donde está ubicada cada palabra en el tablero.

Para seleccionar las palabras en el tablero se utilizan los eventos asociados al *mouse* como son `GetMouseButtonDown` y `GetMouseButtonUp`. Estos eventos permiten definir el inicio y el final de una palabra seleccionada respectivamente, para posteriormente verificar si se trata de alguna de las palabras que deben ser encontradas. Para resaltar a estas palabras en el tablero se utiliza el método `drawLine` que se muestra en el Código 2.21.

En este método se utilizan las posiciones inicial y final definidas al presionar o soltar el clic izquierdo del *mouse* para trazar una línea sobre el tablero, la cual solo puede dibujarse en ángulos que sean múltiplos de 45 grados.

```
599         public void drawLine()
600         {
601             if (isDown) {
602                 if (!isDrawing1) {
603                     actualLine = Instantiate(imgLine, new Vector3(startPosition.x, startPosition.y, 0), Quaternion.identity) as GameObject;
604                     actualLine.transform.SetParent(GameObject.Find("lines").transform);
605                     actualLine.transform.localPosition = new Vector3(startPosition.x, startPosition.y, 0);
606                     actualLine.GetComponent<RectTransform>().localScale = new Vector3(1, 1.5f, 1);
607                     actualLine.GetComponent<Image>().color = currColorDraw; // Change color lineDraw
608                     actualLine.tag = "lineDrawing";
609                      isDrawing1 = true;
610                 }
611                 Vector2 length = (endPosition - startPosition);
612                 float angle = Mathf.Atan2(length.y, length.x) * Mathf.Rad2Deg;
613                 angle = (angle + 360) % 360;
614                 bool drawit = false;
615                 if (angle > 355 && angle < 5) {
616                     angle = 0;
617                 }
618                 if (angle > 40 && angle < 50) {
619                     angle = 45;
620                 }
621                 if (angle > 85 && angle < 95) {
622                     angle = 90;
623                 }
624                 if (angle > 130 && angle < 140) {
625                     angle = 135;
626                 }
627                 if (angle > 175 && angle < 185) {
628                     angle = 180;
629                 }
630                 if (angle > 220 && angle < 230) {
631                     angle = 225;
632                 }
633                 if (angle > 365 && angle < 275) {
634                     angle = 270;
635                 }
636                 if (angle > 310 && angle < 320) {
637                     angle = 315;
638                 }
639                 // permite tomar valores de angulo de 0, 45, 90, 135, 180, 225, 270, 315
640                 if (angle % 45 == 0) {
641                     drawit = true;
642                 }
```

Código 2.21. Segmento de código del método `drawLine`

2.3.7. JUEGO DE TOWER DEFENSE

El videojuego de *Tower Defense* presenta inicialmente una interfaz similar a la que se observa en la Figura 2.38.

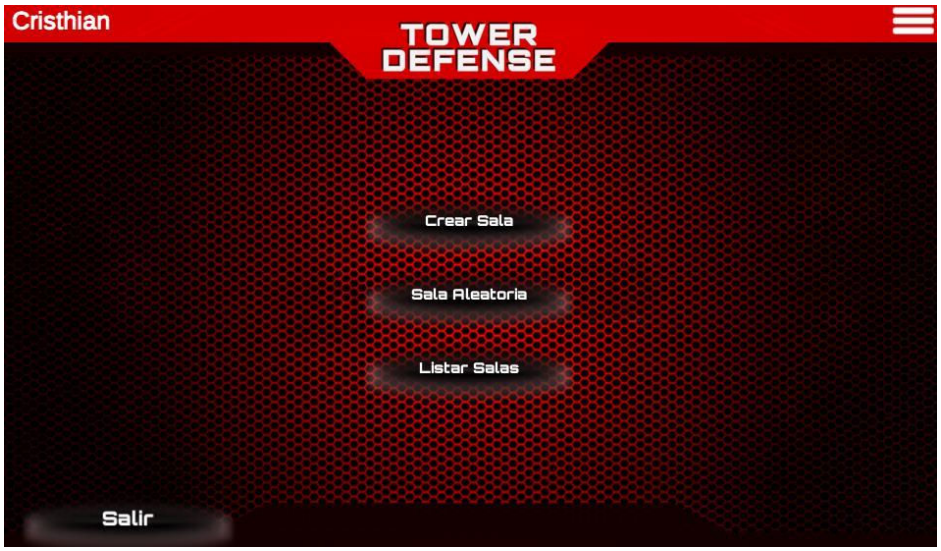

Figura 2.38. Menú inicial del juego de *Tower Defense*

El botón “Crear Sala” permite ingresar a la interfaz en la que los jugadores pueden crear una sala especificando varios parámetros. El botón “Sala Aleatoria” permite al jugador ingresar a una sala de juego al azar. El botón “Listar Salas” permite ingresar a la interfaz en la que se encuentran listadas todas las salas de juego creadas por los jugadores. A cada uno de estos botones se le asocia un evento que permite desplegar las respectivas interfaces. Cuando el jugador ha seleccionado el botón que permite crear una sala, se muestra una interfaz similar a la que se observa en la Figura 2.39.

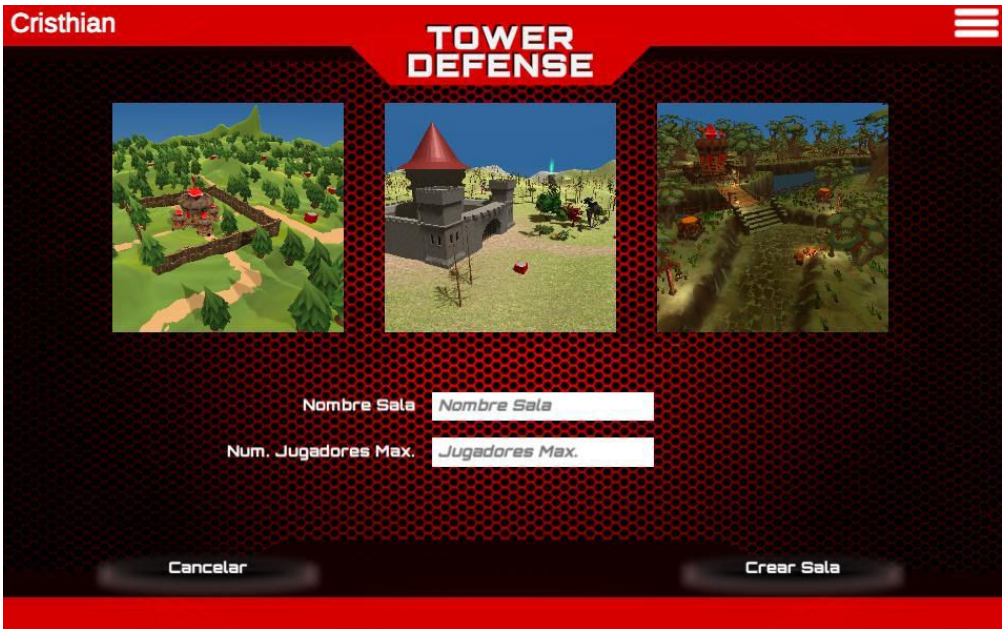

Figura 2.39. Interfaz para creación de salas de juego

La lógica de programación utilizada para la creación de una sala de juego se muestra en el Código 2.22.

```
225         public void OnCreateRoomButtonClicked()
226         {
227             string roomName = RoomNameInputField.text;
228             roomName = (roomName.Equals(string.Empty)) ? "Room " + Random.Range(1000, 10000) : roomName;
229             byte maxPlayers;
230             byte.TryParse(MaxPlayersInputField.text, out maxPlayers);
231             maxPlayers = (byte) Mathf.Clamp(maxPlayers, 1, 4);
232             RoomOptions options = new RoomOptions {MaxPlayers = maxPlayers};
233             PhotonNetwork.CreateRoom(roomName, options, null);
234         }
```

Código 2.22. Método `OnCreateRoomButtonClicked`

En la línea de código 227, se asigna el nombre de la sala que el usuario ha ingresado a la variable `roomName`. En la línea de código 228, se verifica que esta variable no esté vacía, y en caso de que si lo este, se procede a crear un nombre aleatorio.

Entre las líneas de código 229 y 232, se asigna el número máximo de jugadores verificando que como máximo se encuentren cuatro jugadores por cada sala. Finalmente, en la línea de código 233 se utiliza el método `CreateRoom` de la clase `PhotonNetwork` para crear la sala con los parámetros que han sido especificados.

Por otra parte, si el usuario ha seleccionado el botón que permite listar las salas que han sido creadas, se muestra una interfaz similar a la que se observa en la Figura 2.40.

Figura 2.40. Interfaz para listar salas de juego creadas

Para que la lista de salas se muestre en la interfaz presentada anteriormente se utiliza el método llamado `OnRoomListButtonCLicked` de la clase `LobbyMainPanel.cs` que se observa en el Código 2.23.

```
282         public void OnRoomListButtonClicked()
283         {
284             if (!PhotonNetwork.InLobby)
285             {
286                 PhotonNetwork.JoinLobby();
287             }
288             SetActivePanel(RoomListPanel.name);
289         }
```

Código 2.23. Método `OnRoomListButtonCLicked`

En la línea de código 286, se utiliza el método `JoinLobby` de la clase `PhotonNetwork`, el cual permite al jugador ingresar a la interfaz en la que se listan las salas creadas. Además, se utiliza algunos métodos de la clase `RoomInfo` que ofrece *Photon Unity Networking*, los cuales permiten mantener actualizados los datos referentes al número de jugadores conectados en cada sala creada.

Por otro lado, para conseguir que un usuario ingrese a una sala aleatoria se utiliza el método llamado `OnJoinRandomRoomButtonClicked` de la clase `LobbyMainPanel.cs` que se observa en el Código 2.24.

```
0 referencias
236         public void OnJoinRandomRoomButtonClicked()
237         {
238             SetActivePanel(JoinRandomRoomPanel.name);
239             PhotonNetwork.JoinRandomRoom();
240         }
```

Código 2.24. Método `OnJoinRandomRoomButtonClicked`

En la línea de código 239, se utiliza el método `JoinRandomRoom` de la clase `PhotonNetwork`, el cual permite unirse a una sala aleatoria. En caso de que no exista alguna, crea y se une a una sala de nombre aleatorio y número máximo de cuatro jugadores.

Cuando un usuario ha creado o se ha unido a una sala, se le muestra una interfaz similar a la que se observa en la Figura 2.41.

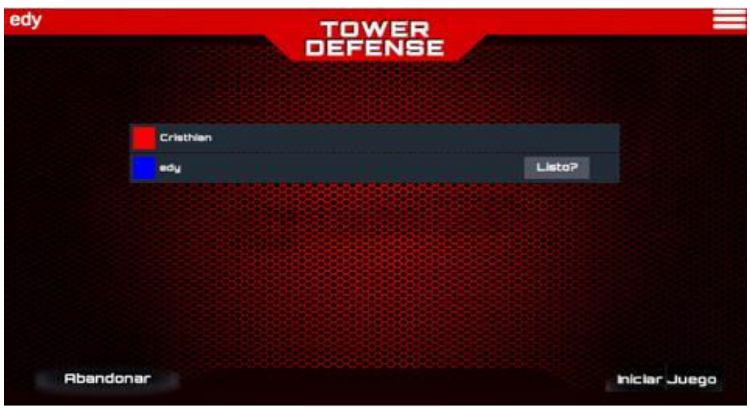

Figura 2.41. Interfaz que muestra la sala creada con sus jugadores

La lógica de programación que permite listar a cada uno de los jugadores que ingresan a una sala se encuentra detallada en el método llamado `OnPlayerEnteredRoom` que se puede observar en el Código 2.25.

```
160     public override void OnPlayerEnteredRoom(Player newPlayer)
161     {
162         GameObject entry = Instantiate(PlayerListEntryPrefab);
163         entry.transform.SetParent(InsideRoomPanel.transform);
164         entry.transform.localScale = Vector3.one;
165         entry.GetComponent<PlayerListEntry>().Initialize(newPlayer.ActorNumber, newPlayer.NickName);
166         playerListEntries.Add(newPlayer.ActorNumber, entry);
167         StartGameButton.gameObject.SetActive(CheckPlayersReady());
168     }
```

Código 2.25. Método `OnPlayerEnteredRoom`

Este método recibe como parámetro de entrada un objeto de la clase `Player`, el cual permite realizar la gestión de los jugadores dentro de una sala de juego. Los jugadores se pueden identificar mediante el número de actor asignado automáticamente al ingresar a una sala o un identificador personalizado que está asociada a la base de datos de MySQL, como se muestra en las líneas de código 165 y 166.

Cuando el jugador que ha creado la sala da inicio a la partida, dependiendo el mapa que haya seleccionado previamente, se les muestra a los jugadores uno de los tres mapas que se pueden observar en la Figura 2.42, la Figura 2.43 y la Figura 2.44. La lógica de programación asociada al botón que permite iniciar la partida se muestra en el Código 2.26.

```
291     public void OnStartGameButtonClicked()
292     {
293         PhotonNetwork.CurrentRoom.IsOpen = false;
294         PhotonNetwork.CurrentRoom.IsVisible = false;
295         string mapName = GameController.mapChosed;
296         PhotonNetwork.LoadLevel(mapName);
297     }
```

Código 2.26. Método `OnStartGameButtonClicked`

En la línea de código 293, se utiliza la propiedad llamada `IsOpen` de la clase `Room` de *Photon Unity Networking,* la cual permite mantener abierta la sala creada mientras se unen los jugadores y también permite cerrar dicha sala para dar inicio a la partida.

Mientras que en la línea de código 294, se utiliza la propiedad `IsVisible` de la clase `Room` de *Photon Unity Networking,* la cual permite mostrar u ocultar a los jugadores la sala que ha sido creada. En la línea de código 295, se obtiene el nombre del mapa que fue seleccionado previamente y que se encuentra almacenado en la clase llamada `GameController.cs`.

Finalmente, en la línea de código 296 se utiliza el método `LoadLevel` de la clase `PhotonNetwork`. Este método recibe como parámetro el nombre del mapa que va a

desplegarse y es llamado únicamente por el cliente que ha creado la sala. Permite cargar el mapa que se ha seleccionado previamente y mostrarlo a todos los clientes conectados en la misma sala.

Figura 2.42. Primer mapa del juego de *Tower Defense*

Figura 2.43. Segundo mapa del juego de *Tower Defense*

Figura 2.44. Tercer mapa del juego de *Tower Defense*

Los mapas mostrados anteriormente están compuestos por diversos objetos prefabricados que han sido obtenidos del *Asset Store* de Unity. Estos objetos son utilizados para complementar el entorno y también para crear las estructuras como castillos o torres. Algunos de los objetos prefabricados más representativos utilizados como estructuras y para decorar el entorno de los mapas se muestran en la Figura 2.45.

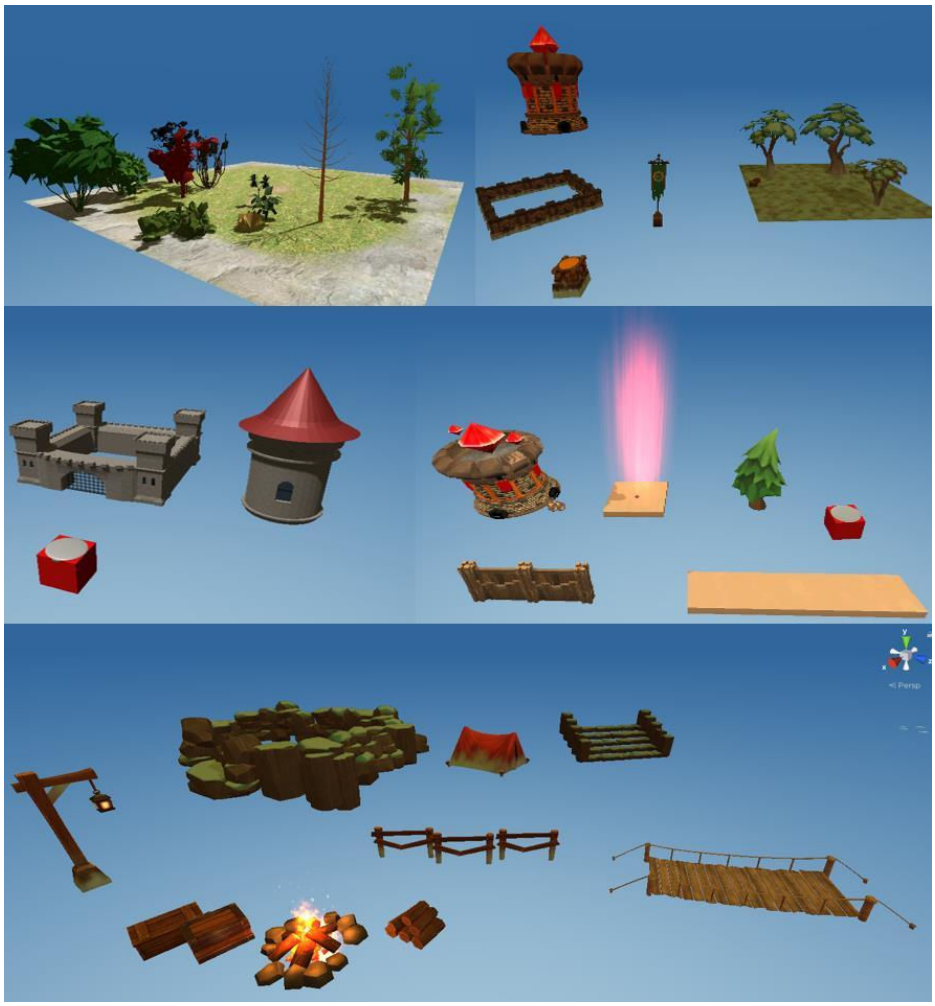

Figura 2.45. Objetos prefabricados representativos de los mapas de *Tower Defense*

Además, se necesitan varios enemigos y diferentes tipos de torres que aparecerán durante la partida y que deben ser desbloqueados utilizando los puntos obtenidos en los diferentes minijuegos educativos. El jugador tiene a disposición varios tipos de torres cuyos rangos de ataque varían al igual que su poder de ataque, estos modelos de torres se pueden observar en la Figura 2.46. De la misma forma, en la Figura 2.47 se puede observar varios tipos de enemigos que pueden aparecer aleatoriamente durante las oleadas de cada partida. Tanto las torres, como los enemigos son objetos prefabricados obtenidos en el *Asset Store* de Unity que pueden ser desbloqueados por los jugadores obteniendo cierta cantidad de puntos en los minijuegos educativos.

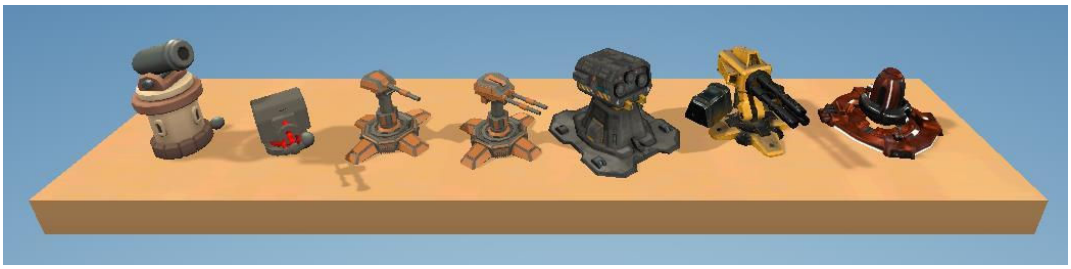

Figura 2.46. Modelos de torres utilizadas en el juego de *Tower Defense*

Figura 2.47. Modelos de enemigos utilizados en el juego de *Tower Defense*

En el videojuego de *Tower Defense,* uno de los eventos más importantes que deben ocurrir durante una partida es la generación de enemigos. La lógica de programación asociada a este evento se desarrolla dentro de la corutina llamada `SpawnEnemy` que se encuentra en la clase `GameManagerTD`, como se observa en el Código 2.27.

```
252         private IEnumerator SpawnEnemy()
253         {
254             while (true) {
255                 if (hasFinishedgame) {
256                     break;
257                 }
258                 enemyCount++;
259                 // Control por Jugadores
260                 // contador i=0 corresponde al objeto padre
261                 for (int i = 0; i < PhotonNetwork.CurrentRoom.PlayerCount; i++) {
262                     // Se instancia en la Escena para que no se destrura en un cambio de MasterClient
263                     object[] instantiationData = { WayPointsPlayer[i].gameObject.name };
264                     GameObject enemy = PhotonNetwork.InstantiateSceneObject("Enemies/" + enemyName, spawnPoints[i + 1].position,
265                         Quaternion.identity, 0, instantiationData) as GameObject;
266                     enemy.name = "Enemy (" + enemyCount + ")";
267                 }
268                 if (enemyCount == WaveSize) {       // Paso a siguiente Oleada
269                     enemyCount = 0;
270                     waveCount++;
271                     enemyName = buildingTowerUI.GetEnemyRandom();
272                     yield return new WaitForSeconds(timeNextWave);
273                 } else {
274                     yield return new WaitForSeconds(EnemyInterval);
275                 }
276                 if (waveCount == numWaves) {        // permite terminar el juego
277                     waveCount = 0;
278                     wavesCompleted = true;
279                     // Aqui se debe verificar el fin del juego (PERO PUEDE UN ENEMIGO LLEGAR AL CASTILLO ANTES)
280                     break;
281                 }
282             }
283         }
```

Código 2.27. Corutina `SpawnEnemy`

En la línea de código 263, se crea una instancia del objeto de tipo enemigo utilizando el método llamado `InstantiateSceneObject` de la clase `PhotonNetwork`. Este método

permite instanciar un objeto en red a partir de su nombre, por lo que los objetos prefabricados deben estar ubicados dentro del directorio llamado "*Resources*" para ser ubicados fácilmente. Es por esta razón que los objetos de tipo enemigo se encuentran en un directorio llamado "*Enemies*" el cual está ubicado dentro del directorio "*Resources*".

Además, todos los objetos que participen en una escena multijugador, es decir que sean objetos en red, deben ser sincronizados para que sean vistos por todos los jugadores durante la partida. Esto se consigue utilizando el método que ofrece *Photon Unity Networking* llamado `OnPhotonSerializeView`, el cual es llamado varias veces por segundo durante la partida, con la finalidad de sincronizar las variables mediante lectura y escritura entre todos los jugadores de la misma sala. Este método puede observarse en el Código 2.28 y se encuentra dentro de la clase `GameManagerTD`.

```
324         // Se sincroniza parametros de oleadas para posible siguiente MasterClient
            11 referencias
325         public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info) {
326             if (stream.IsWriting) {
327                 stream.SendNext(enemyCount);
328                 stream.SendNext(waveCount);
329                 stream.SendNext(hasFinishedgame);
330                 stream.SendNext(timerEndGame);
331
332             } else {
333                 enemyCount = (int)stream.ReceiveNext();
334                 waveCount = (int)stream.ReceiveNext();
335                 hasFinishedgame = (bool)stream.ReceiveNext();
336                 timerEndGame = (float)stream.ReceiveNext();
337             }
338         }
```

Código 2.28. Método `OnPhotonSerializeView`

El primer parámetro de entrada de este método es un contenedor que permite enviar los datos al servidor de *Photon* para que este envíe la información a todos los jugadores dentro del objeto llamado `stream`. Los objetos que son sincronizados para ser mostrados a todos los jugadores son: el número de enemigos, el número de oleadas, una variable que indica si la partida ha finalizado y un temporizador que se ejecuta cuando ha terminado una partida, como se observa entre las líneas de código 327 y 330.

Otro evento que debe generarse durante una partida de *Tower Defense* y cuya lógica de programación se muestra en el Código 2.29, es la colocación de torres en el mapa. Este evento debe producirse en base a parámetros locales de cada jugador. Es decir, las torres que cada jugador haya desbloqueado, la cantidad de monedas que posee el jugador, entre otros. Es por esta razón que para la programación de este evento se utiliza un método de la clase `PhotonNetwork` llamado `LocalPlayer`, el cual permite identificar al propio jugador para atribuirle o negar ciertas operaciones.

Para realizar la colocación de torres se implementa un proceso de “arrastrar y soltar” utilizando el puntero del *mouse.* Para esto, se deben manejar tres métodos que se encuentran dentro de la clase llamada `ItemDragAndDropHandler`.

En la línea de código 66 se crea el método llamado `OnDrag`, el cual permite obtener la posición del cursor del *mouse* y dibujar el rango de alcance que tiene la torre que se está arrastrando con el cursor. Luego, en la línea de código 73, se crea el método llamado `OnEndDrag`, el cual fija la posición donde se terminó de arrastrar el puntero del *mouse* para definir el lugar donde se dibujará la torre. Finalmente, en la línea de código 79, se crea el método llamado `OnDrop`, el cual verifica si el jugador dispone de las monedas suficientes para realizar la compra de la torre cuando se ha soltado el clic del *mouse*. En el caso de que el jugador disponga de las monedas necesarias la torre será colocada en el mapa.

```
65      // Al inicio de mover toma la posicion del puntero
        0 referencias
66      public void OnDrag(PointerEventData eventData) {
67          transform.position = Input.mousePosition;
68          // dibujar el rango de la torre
69          EventsTowerDefense.eventDrawTowerRange(prefabTurret);
70      }
71
72      // Permite volver a colocar el objeto 2D en el slot
        0 referencias
73      public void OnEndDrag(PointerEventData eventData) {
74          transform.localPosition = Vector3.zero;
75
76      }
77
78      // Al soltar se invoca la instancia del prefab Torre
        0 referencias
79      public void OnDrop(PointerEventData eventData) {
80          // Verificar SCORE antes de colocar
81          if (newScore >= priceTower) {
82              EventsTowerDefense.eventUpdateScore(priceTower);
83              EventsTowerDefense.eventDrawTower(prefabTurret.name);
84              Debug.Log("Torre comprada!");
85          } else {
86              Debug.Log("INSUFICIENTES MODENDAS");
87          }
88      }
```

Código 2.29. Proceso *Drag and Drop* para colocar torres

2.3.8. INTERFACES ADICIONALES

2.3.8.1. Interfaz de fin de juego para los minijuegos educativos

Todos los minijuegos educativos poseen una interfaz para mostrar que una partida ha concluido. A continuación, se muestra un ejemplo de la implementación de la interfaz de fin de partida para el minijuego de batalla naval. En este minijuego cuando una partida ha finalizado, la interfaz que se muestra al usuario es similar a la que se observa en la Figura 2.48.

Para definir el puntaje final obtenido con cada acierto en los desafíos educativos del minijuego de batalla naval se utiliza el método llamado `Acierto`, que se encuentra en la clase `GameManagerBattleShip.cs` y que se observa en el Código 2.30.

Figura 2.48. Interfaz de fin de partida del minijuego de batalla naval

```
464     1 referencia
        public void Acierto()
465     {
466         acierto = "RESPUESTA";
467         int value = currQCodec.Score_;
468         score += value;
469         StartCoroutine(showMessageScore("+" + value + "puntos"));
```

Código 2.30. Segmento de código del método `Acierto`

En la línea de código 467, se obtiene el puntaje del segmento de código y luego este valor es almacenado en una variable llamada `score` como se observa en la línea de código 469.

Por otra parte, al finalizar una partida también se ejecuta la corutina `SetGameMatch` que es la encargada de almacenar en la base de datos el puntaje obtenido, así como los identificadores de las preguntas que se han usado durante la partida. Esta corutina se muestra en el Código 2.31.

```
588     IEnumerator SetGameMatch()
589     {
590         WWWForm form = new WWWForm();
591         form.AddField("idStudent", GameController.idStudent);
592         form.AddField("score", score);
593         foreach (var item in m_quizDB.lstFeedback)
594         {
595             form.AddField("list[]", item.id_);
596         }
597         using (UnityWebRequest www = UnityWebRequest.Post("http://localhost/game/SetGameMatchBattleship.php", form))
598         {
599             yield return www.SendWebRequest();
600             if (www.isNetworkError || www.isHttpError)
601             {
602                 Debug.Log(www.error);
603             }
604             else
605             {
606                 Debug.Log("Puntos acumulados de bdd: " + www.downloadHandler.text);
607             }
608         }
609     }
```

Código 2.31. Corutina `SetGameMatch`

Entre las líneas de código 591 y 596, se observa los parámetros que son almacenados en la base de datos. Entre estos se encuentran el identificador del estudiante, el puntaje que ha obtenido y los identificadores de los segmentos de código que han sido mostrados durante la partida de batalla naval. Esta información es enviada hacia la base de datos utilizando el *script* llamado `SetGameMatchBattleship.php` el cual se encarga de realizar la comunicación e inserción de los registros en la base de datos.

2.3.8.2. Interfaz de Retroalimentación

Todos los minijuegos educativos poseen una interfaz que permite mostrar al usuario información adicional acerca de los desafíos educativos (preguntas, conceptos y segmentos de código) que han sido mostrados durante cada partida. A continuación, se muestra un ejemplo de la implementación de la interfaz de retroalimentación para el minijuego de trivia. En dicho minijuego educativo, cuando el usuario selecciona el botón que permite consultar la retroalimentación, se muestra una interfaz similar a la Figura 2.49.

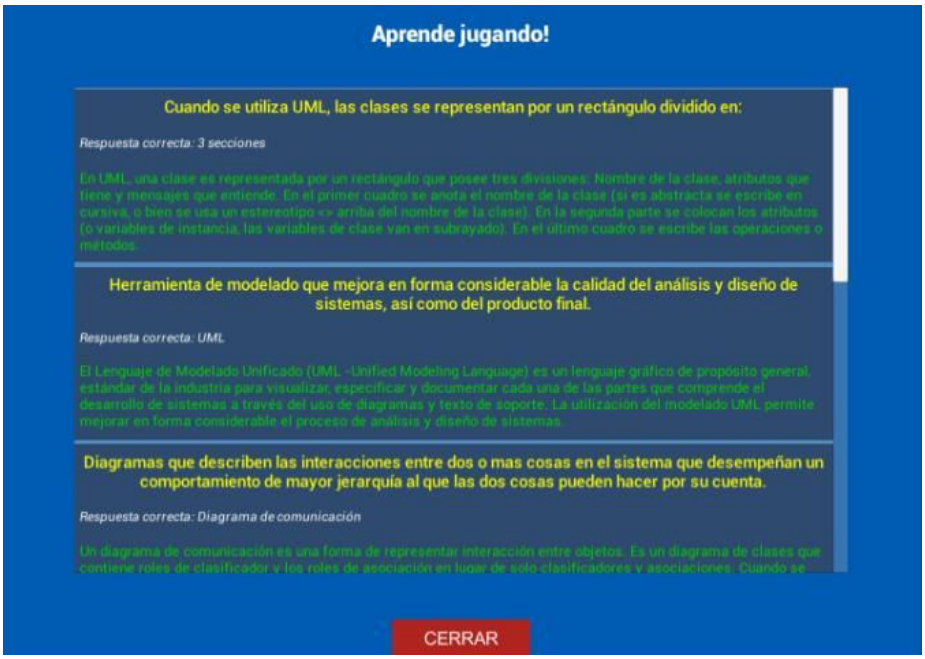

Figura 2.49. Interfaz de *feedback* del minijuego de trivia

La lógica de programación asociada al botón que muestra la retroalimentación se encuentra codificada en el método `ButtonOpenFeedback`, de la clase `GameManager`, que se muestra en el Código 2.32. Este método recorre la lista de preguntas, así como la lista de alternativas para mostrarlas en esta interfaz junto con la respuesta correcta. Luego, se crea una instancia del objeto prefabricado `QuestionPrefab` en el cual se muestra el enunciado de la pregunta, la respuesta y la retroalimentación de cada pregunta que ha sido mostrada durante la partida.

```
368     public void ButtonOpenFeedback() {
369         int i = 1;
370         foreach (var q in _questions) {
371             foreach (var a in q.Alternatives_) {
372                 if (a.IsCorrect == true) {
373                     Debug.Log("pregunta " + i + ": " + q.Question_ + "\n Respuesta: " + a.Alternative_);
374                     GameObject go_Question = Instantiate(questionPrefab,quizContainer) as GameObject;
375                     go_Question.transform.GetChild(0).GetComponent<Text>().text = q.Question_;
376                     go_Question.transform.GetChild(1).GetComponent<Text>().text = "Respuesta correcta: "+a.Alternative_;
377                     go_Question.transform.GetChild(2).GetComponent<Text>().text = q.Feedback;
378                 }
379             }
380             i++;
381         }
382         panelFeedbackQuiz.SetActive(true);
383     }
```

Código 2.32. Método `ButtonOpenFeedback`

De manera similar al proceso detallado anteriormente, han sido implementadas todas las interfaces que permiten mostrar la retroalimentación en cada uno de los minijuegos educativos.

2.3.8.3. Ranking

Cuando un jugador ingresa al videojuego puede observar en la esquina superior derecha un menú desplegable, el cual posee tres opciones como se observa en la Figura 2.50.

Figura 2.50. Menú desplegable del videojuego

La primera opción de este menú permite visualizar el *ranking* de jugadores en base al puntaje obtenido en los minijuegos educativos. La lógica de programación asociada al botón del *ranking* se muestra en el Código 2.33. En la línea de código 156, se observa que para obtener los datos que se mostrarán en el *ranking* se utiliza la corutina llamada `co_GetRanking` que se observa en el Código 2.34.

```
0 referencias
153     public void ButtonRankingClicked()
154     {
155         // Se cargan los datos de ranking
156         StartCoroutine(co_GetRanking());
157         ButtonOpenPanelMenu();
158         SetActiveMenuSide(panelRanking.name);
159     }
```

Código 2.33. Método `ButtonRankingClicked` de la clase `AdminGeneral.cs`

En la línea 285 del Código 2.34, se realiza una llamada al *script* codificado en lenguaje PHP, el cual realiza una petición a la base de datos para extraer la información acerca de los estudiantes con sus nombres y puntajes respectivos. En la línea de código 298, se crea una instancia del objeto que va a mostrar los datos en la interfaz de *ranking* llamado `prefabRanking`. Mientras que entre las líneas de código 299 y 314, se asignan los valores obtenidos de la base de datos a sus respectivos componentes gráficos. Cuando el usuario selecciona el botón para presentar el *ranking,* se muestra una interfaz similar a la que se observa en la Figura 2.51.

```
282     IEnumerator co_GetRanking() {
283         WWWForm form = new WWWForm();
284         form.AddField("idStudent", idUser);
285         using (UnityWebRequest www = UnityWebRequest.Post("http://localhost/game/GetClassmates.php", form)) {
286             yield return www.SendWebRequest();
287             if (www.isNetworkError || www.isHttpError) {
288                 informe(www.error);
289             } else {
290                 string dataIn = www.downloadHandler.text;
291                 string[] dataStudents = dataIn.Split(';');
292                 for (int i = 0; i < containerRanking.childCount; i++) {
293                     Destroy(containerRanking.GetChild(i).gameObject);
294                 }
295                 for (int i = 0; i < dataStudents.Length; i++) {
296                     string item = dataStudents[i];
297                     string[] columns = item.Split(',');
298                     GameObject go = Instantiate(prefabRanking, containerRanking) as GameObject; // prefab
299                     GameObject goName = go.transform.GetChild(1).gameObject as GameObject; // Nombre
300                     goName.GetComponent<Text>().text = columns[0];
301                     goName.transform.SetParent(go.transform);
302                     GameObject goScore = go.transform.GetChild(2).gameObject as GameObject; // Puntaje
303                     goScore.GetComponent<Text>().text = columns[1];
304                     goScore.transform.SetParent(go.transform);
305                     GameObject goNumber_bg = go.transform.GetChild(3).gameObject as GameObject; // Numero bk
306                     goNumber_bg.transform.SetParent(go.transform);
307                     GameObject goText = goNumber_bg.transform.GetChild(0).gameObject as GameObject;
308                     goText.GetComponent<Text>().text = (i+1).ToString();
309                     goText.transform.SetParent(goNumber_bg.transform);
310                     if (i<TopRanking) {
311                         GameObject goTop = go.transform.GetChild(4).gameObject as GameObject; // Nombre
312                         goTop.transform.SetParent(go.transform);
313                         goTop.SetActive(true);
314                     }
315                 }
316             }
317         }
318     }
```

Código 2.34. Corutina `co_GetRanking` de la clase `AdminGeneral.cs`

Figura 2.51. Interfaz de *ranking* del videojuego

2.3.8.4. Inventario

Cuando el usuario selecciona la segunda opción del menú desplegable, se muestra el inventario que posee el jugador, es decir, las torres y los enemigos que tiene desbloqueados junto con el número de puntos y créditos que posee. Esta interfaz se observa en la Figura 2.52.

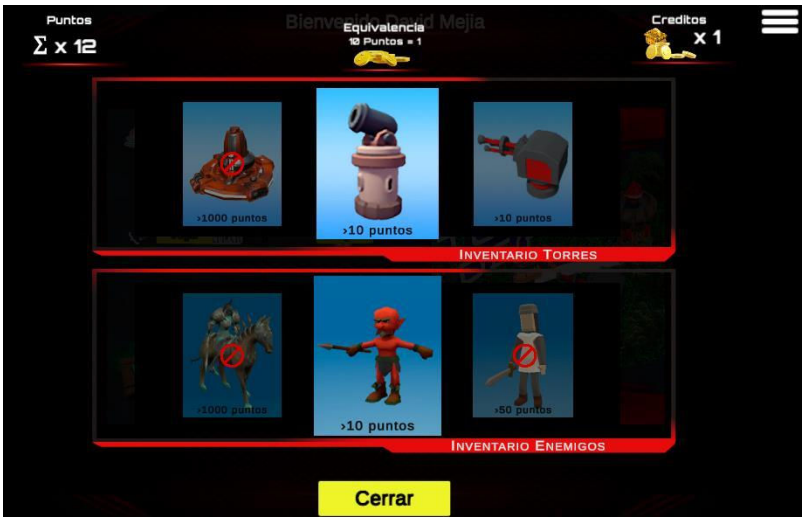

Figura 2.52. Interfaz para mostrar el inventario de un jugador

2.3.9. DESPLIEGUE

Para construir al videojuego utilizando WebGL, se debe seleccionar las escenas del videojuego en la ventana de *Build Settings* y se debe oprimir el botón *Build* en la interfaz que se muestra en la Figura 2.53.

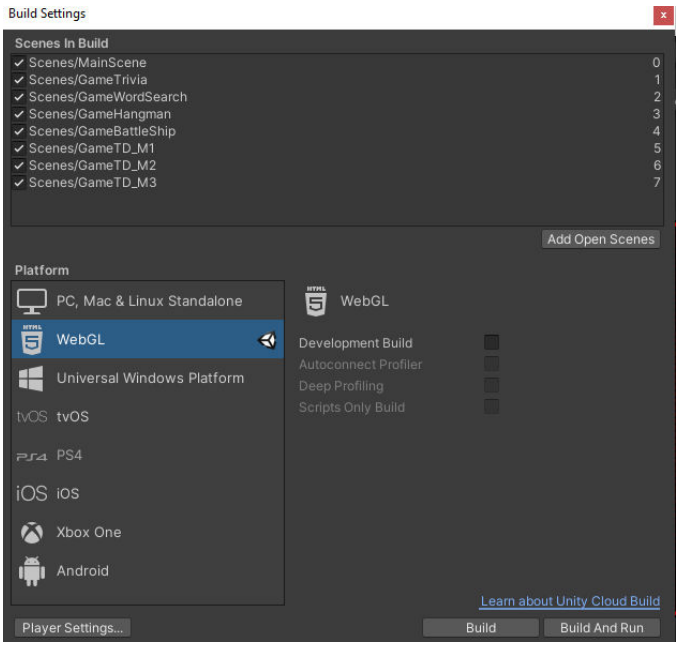

Figura 2.53. Construcción del videojuego utilizando WebGL

Esta opción de construcción permite generar la aplicación web. Cuando el proyecto ha sido construido, Unity creará los siguientes archivos dentro de un directorio que ha sido especificado por el usuario:

- ***index.html***: Este archivo permite que los navegadores web puedan cargar el contenido del videojuego.
- ***Build***: Esta carpeta contiene los archivos resultantes de realizar la construcción del proyecto.
- ***TemplateData***: Esta carpeta contiene componentes como barras de carga y otros *assets* plantilla.

Para el despliegue del videojuego se utilizó la plataforma Microsoft Azure, que provee servicios de computación en la nube como el alojamiento de sitios web o creación de máquinas virtuales. Para acceder a sus servicios es necesario tener una cuenta en la plataforma. Se ha utilizado una máquina virtual cuyo tamaño corresponde a una máquina virtual B1ms estándar de Azure.

Esta máquina virtual ha sido seleccionada para realizar el despliegue de este prototipo de videojuego debido a que este tipo de máquinas virtuales son ideales para cargas de trabajo que no necesitan el rendimiento completo de la CPU de forma continua, como servidores web, pruebas de conceptos, bases de datos pequeñas y entornos de construcción de desarrollo.

El sistema operativo de la máquina virtual es Windows 10 Pro. Cuando se crea la máquina virtual se debe especificar, entre otras cosas, el nombre DNS[42] cuya URL[43] será utilizada para acceder al videojuego mediante un navegador web. Otras características importantes se muestran en la Figura 2.54.

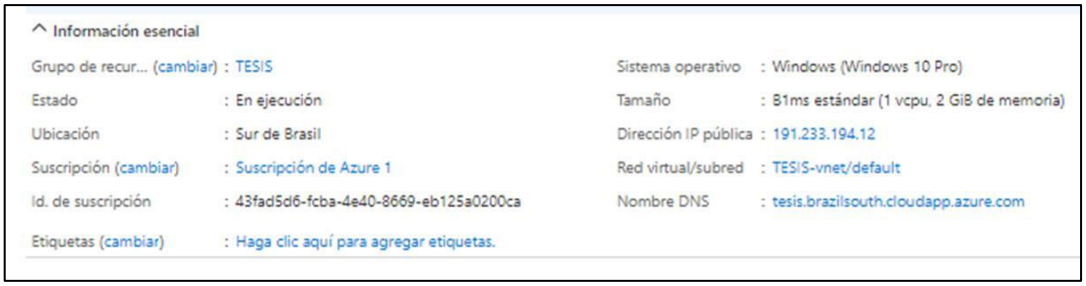

Figura 2.54. Información general de la máquina virtual de Azure

[42] DNS (*Domain Name System*): Permite apuntar los dominios al servidor correspondiente, sirve para traducir las direcciones IP en los nombres de dominio correspondientes.
[43] URL (*Uniform Resource Locator*): Es la dirección específica que se asigna a cada uno de los recursos disponibles en la red con la finalidad de que estos puedan ser localizados.

Dentro de la máquina virtual ha sido instalado MySQL Server, que permite gestionar y levantar la base de datos que necesita el videojuego. Además, ha sido instalado el entorno de desarrollo PHP llamado XAMPP, el cual ofrece una distribución gratuita del servidor web Apache. Precisamente este servicio se utiliza para alojar al videojuego.

Después que ha sido instalado XAMPP, dentro del directorio del mismo nombre se encuentra la carpeta llamada *htdocs*. En esta carpeta deben ser colocados los archivos que han sido generados después de la construcción del videojuego utilizando WebGL y también los *scripts* codificados en el lenguaje de programación PHP.

3. RESULTADOS Y DISCUSIÓN

Se realizaron varias pruebas para verificar el correcto funcionamiento del sistema, tanto por cada módulo como en la totalidad del videojuego.

Una vez que el videojuego fue desplegado en Internet, se realizaron pruebas utilizando los navegadores Google Chrome y Mozilla Firefox. Entre las tareas realizadas están:

- Pruebas de conectividad entre el servidor web y la base de datos.
- Pruebas de conectividad entre un cliente y el servidor web.
- Pruebas de autenticación de usuarios.
- Pruebas de creación de nuevos registros, ya sean de usuarios, capítulos de la asignatura o contenido educativo.
- Pruebas de eliminación de registros.
- Pruebas de modificación de registros de usuarios, capítulos de la asignatura o contenido educativo.
- Pruebas de asignación de puntaje a jugadores al finalizar una partida en cada minijuego.
- Pruebas para verificación del funcionamiento del *ranking.*
- Pruebas de desbloqueo de torres y enemigos para *Tower Defense.*

Por otra parte, se realizaron 12 encuestas dirigidas a estudiantes de la Facultad de Ingeniería Eléctrica y Electrónica que hayan cursado la materia de Programación Avanzada para complementar la validación del correcto funcionamiento del videojuego.

3.1. PRUEBAS DE FUNCIONAMIENTO

A continuación, se presentan algunos ejemplos de las pruebas realizadas.

3.1.1. PRUEBAS DE CONECTIVIDAD

Para realizar la prueba local de conectividad con la base de datos se utiliza un *script* codificado en el lenguaje de programación PHP, este *script* se muestra en la Figura 3.1. En la línea de código 2, se coloca la dirección URL del servidor en donde se encuentra alojada la base de datos. Mientras que entre las líneas de código 3 y 5, se colocan las credenciales para acceder a la base de datos. Entre las líneas de código 6 y 17 se establece la conexión con la base de datos. En caso de que la conexión sea exitosa, en el navegador se debe mostrar el mensaje que se indica en la línea de código 19. Como resultado de la ejecución de este *script,* en el navegador se observa el mensaje que indica que la conexión con la base de datos ha sido exitosa como se observa en la Figura 3.2.

TestConnection.php

C: > xampp > htdocs > game > TestConnection.php

```php
<?php
    $servername = "tesis.brazilsouth.cloudapp.azure.com";
    $username = "Administrador";
    $password = "admin20";
    $dbname = "game";
    $conn = new mysqli($servername, $username, $password, $dbname);


    // Se establece la conexion
    mysqli_report(MYSQLI_REPORT_STRICT);
    try{
        $conn = connectdb();
    }
    catch(Exception $e){
        echo "fallo";
        exit();
    }

    echo "Conexión Exitosa!";
    $conn->close();
?>
```

Figura 3.1. *Script* para prueba de conexión con la base de datos

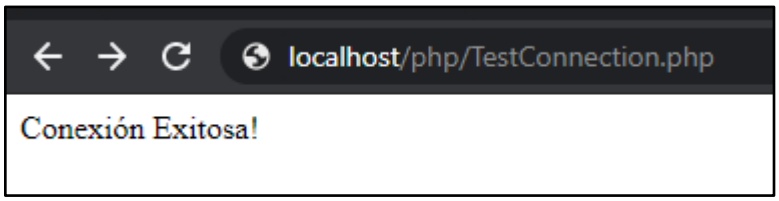

Figura 3.2. Resultado de la prueba de conectividad con la base de datos

Por otra parte, para realizar la prueba de conexión entre un cliente y el servidor web, se debe habilitar el puerto 80 en la máquina virtual, como se muestra en la Figura 3.3.

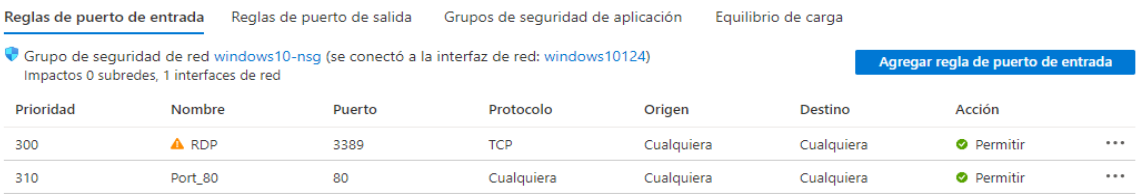

Figura 3.3. Reglas de puertos de entrada de la máquina virtual de Azure

Con el servidor alojado en la nube de Azure permitiendo el acceso al puerto 80, se realiza una consulta al *script* llamado `Login.php` desde el navegador web de un cliente como se muestra en la Figura 3.4. La respuesta obtenida al realizar esta petición muestra como resultado que las credenciales ingresadas pertenecen a un usuario de tipo Estudiante cuyo identificador es 8, como se observa en la Figura 3.5.

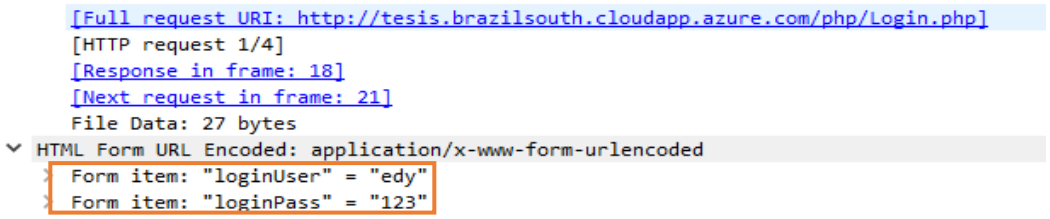

Figura 3.4. Mensaje de petición a `Login.php`

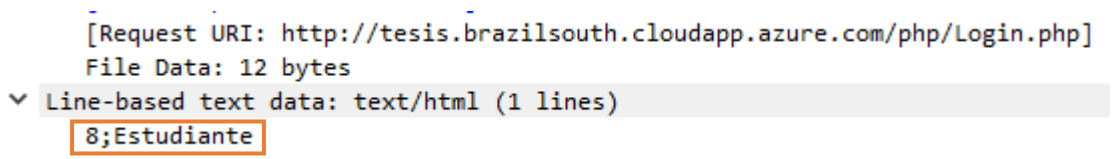

Figura 3.5. Mensaje de respuesta de `Login.php`

Al consultar la base de datos se comprueba que las credenciales ingresadas corresponden al usuario que se encuentra en la tabla `student` de la base de datos cuyo identificador es 8, como se muestra en la Figura 3.6.

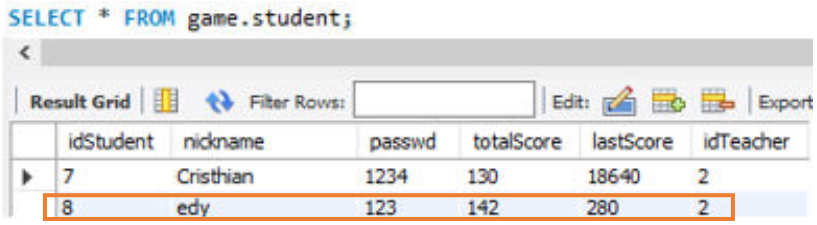

Figura 3.6. Resultados de la consulta a la tabla `student`

3.1.2. AUTENTICACIÓN DE USUARIOS

En la interfaz principal del videojuego, se muestra el `Login` que permite al usuario iniciar sesión en el sistema. Para iniciar sesión, el usuario debe ingresar su nombre de usuario y la contraseña. Dependiendo del perfil del usuario que haya iniciado sesión, el videojuego redirige al usuario a una u otra interfaz. Para un usuario con perfil de Administrador, la interfaz que se visualiza al iniciar sesión es la que se presenta en la Figura 3.7. En esta interfaz se observa un único botón que permite registrar a los profesores en el sistema. Si el usuario que ha iniciado sesión tiene perfil de Profesor, la interfaz que se muestra es la indicada en la Figura 3.8. A diferencia del perfil de Administrador, este perfil tiene permitido gestionar estudiantes y contenido educativo por lo que esta interfaz muestra cinco botones habilitados. Finalmente, en el caso de que el usuario tenga perfil de Estudiante, la interfaz que se visualiza al iniciar sesión es la que se presenta en la Figura 3.9. Dado que este perfil tiene acceso a los diferentes juegos, se le muestra el menú de selección de juegos.

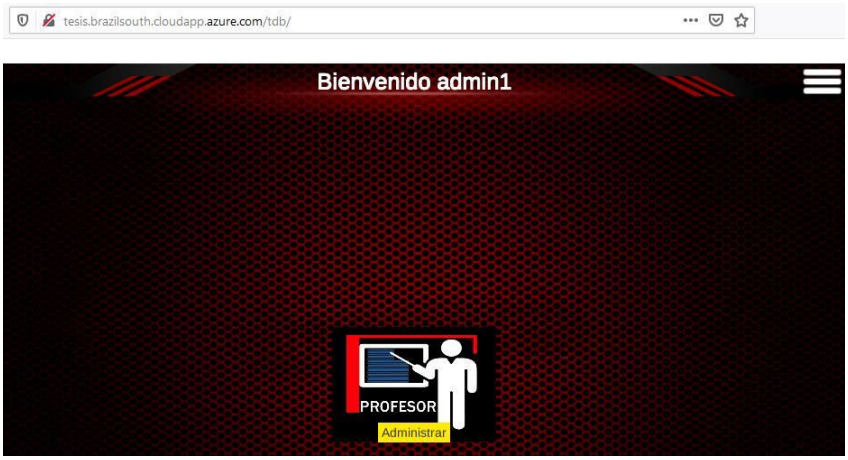

Figura 3.7. Inicio de sesión exitoso para un usuario con perfil de Administrador

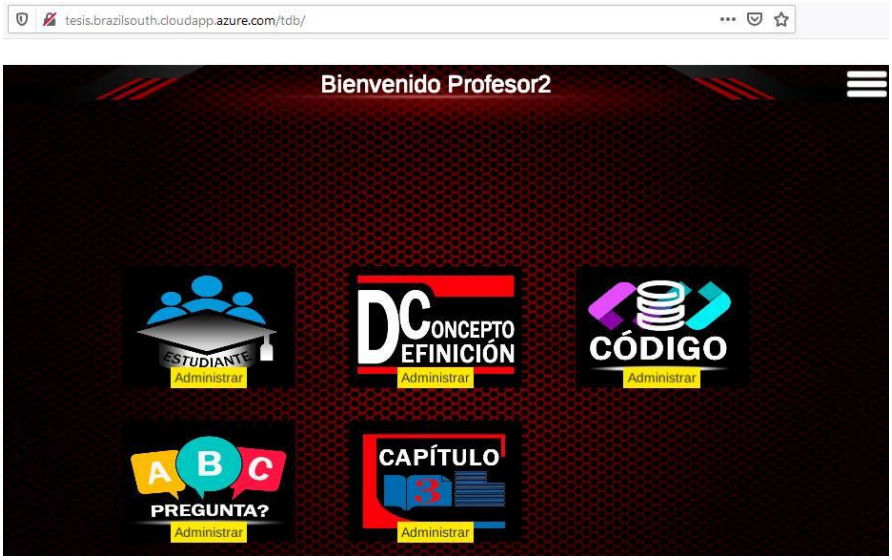

Figura 3.8. Inicio de sesión exitoso para un usuario con perfil de Profesor

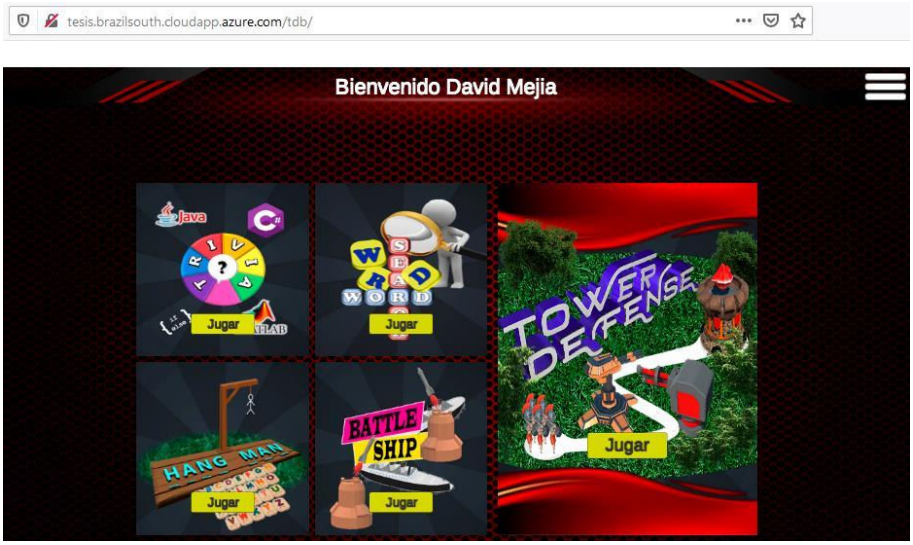

Figura 3.9. Inicio de sesión exitoso para un usuario con perfil de Estudiante

3.1.3. CREACIÓN DE REGISTROS

Un usuario con perfil de Profesor puede agregar nuevos registros de estudiantes, capítulos, preguntas, conceptos y segmentos de código. En la Figura 3.10, se muestra la interfaz que permite agregar nuevos estudiantes. Todos los campos deben llenarse de forma obligatoria. La información especificada se almacena como un nuevo registro en la tabla correspondiente de la base de datos, esto se visualiza en la Figura 3.11.

Figura 3.10. Interfaz para agregar nuevos estudiantes

```
11 •   SELECT * FROM game.student where idTeacher=2;
```

Result Grid | Filter Rows: | Edit: | Export

idStudent	nickname	passwd	totalScore	lastScore	idTeacher
7	Cristhian	1234	130	18640	2
8	edy	123	142	280	2
91	Jorge Velasco	jorvel05	12	12	2
NULL	NULL	NULL	NULL	NULL	NULL

Figura 3.11. Tabla `student`

3.1.4. MODIFICACIÓN DE REGISTROS

Los usuarios con perfil de Profesor pueden realizar la modificación de registros de estudiantes, capítulos, preguntas, conceptos y segmentos de código. La Figura 3.12 muestra la interfaz correspondiente a la edición de registros de pregunta. Ha sido modificada la pregunta cuyo enunciado es: “Esta es una nueva pregunta”.

En la Figura 3.13, se muestra la tabla `question` con el registro de la pregunta antes de ser modificada. Luego de realizar un cambio al enunciado y a la retroalimentación de la pregunta, al consultar sobre este registro a la tabla correspondiente de la base de datos se puede observar que los cambios han sido realizados, como se puede constatar en la Figura 3.14. De esta manera el nuevo enunciado es “Esta es una nueva pregunta ACTUALIZADA”.

Figura 3.12. Interfaz para edición de registro de preguntas

```
5 •  SELECT * FROM game.question where idChapter=1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

idQuestion	question	points	feedbackDescr	idChapter
19	Cuando se ingresa un símbolo de punto y coma ...	10	Si se añade un punto y coma (;) al final de la ins...	1
20	Para que sirve el comando pwd en Matlab	10	El Comando pwd puede ser ejecutado en la ven...	1
121	Esta es una nueva pregunta	1	Esta es una realimentacion nueva	1
NULL	NULL	NULL	NULL	NULL

Figura 3.13. Tabla `question` antes de modificar el registro

```
5 •  SELECT * FROM game.question where idChapter=1 and idQuestion=121;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

idQuestion	question	points	feedbackDescr	idChapter
121	Esta es una nueva pregunta ACTUALIZADO	1	Esta es una realimentacion nueva ACTUALIZADO	1
NULL	NULL	NULL	NULL	NULL

Figura 3.14. Tabla `question` después de modificar el registro

3.1.5. ELIMINACIÓN DE REGISTROS

Los usuarios con perfil de Profesor pueden realizar la eliminación de registros de estudiantes, capítulos, preguntas, conceptos y segmentos de código. La Figura 3.15 muestra la interfaz que se despliega al eliminar registros. Se ha seleccionado el concepto cuyo enunciado es “Este es un nuevo concepto ACTUALIZADO” para ser eliminado. En la

Figura 3.16 se muestra la tabla `concept` con el registro del concepto antes de ser eliminado. Una vez que el usuario ha confirmado la eliminación del registro, se realiza una consulta a la base de datos con el identificador del concepto y el resultado se puede visualizar en la Figura 3.17. Al mostrarse una respuesta vacía se puede afirmar que el concepto ha sido eliminado de la tabla.

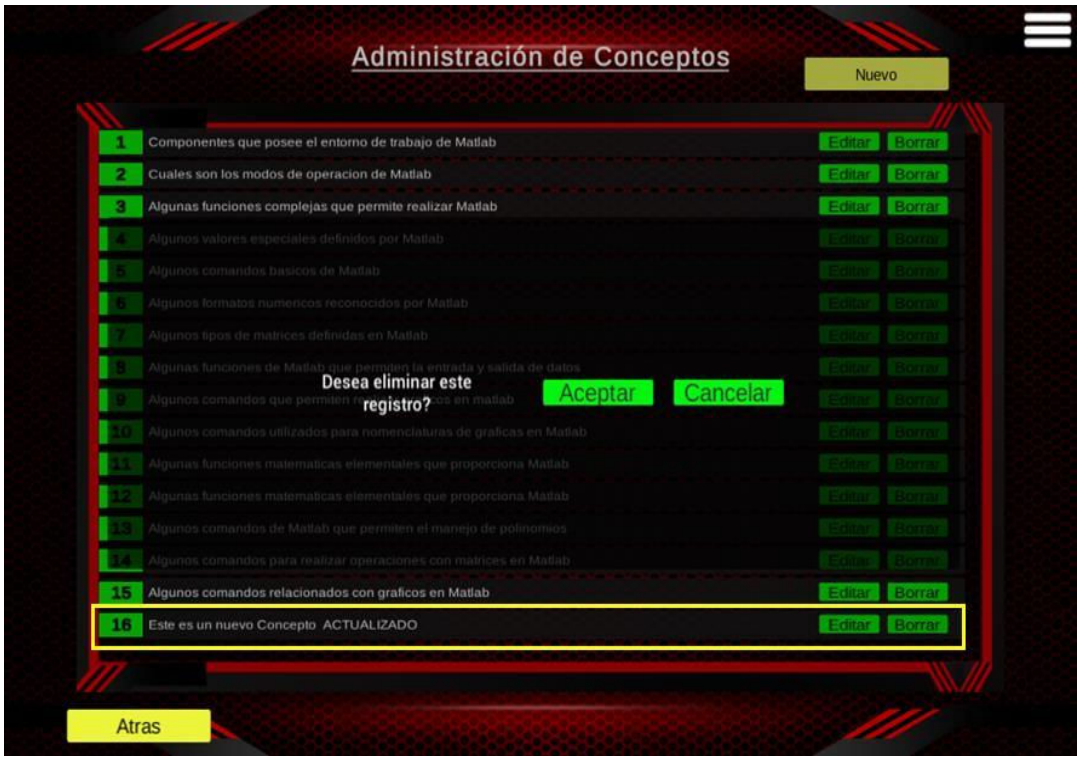

Figura 3.15. Interfaz para eliminación de registro de concepto

```
5 •  SELECT * FROM game.concept where idChapter=1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

idConcept	concept	points	feedbackDescr	idChapter
14	Algunos comandos para realizar operaciones co...	7	El comando find(A) devuelve los índices donde las entra...	1
15	Algunos comandos relacionados con graficos en ...	7	El comando area permite colorear el área bajo una gráfi...	1
91	Este es un nuevo Concepto ACTUALIZADO	5	Esta es una realimentacion de prueba ACTUALIZADA	1
NULL	NULL	NULL	NULL	NULL

Figura 3.16. Tabla `concept` antes de eliminar el registro

```
5 •  SELECT * FROM game.concept where idChapter=1 and idConcept=91;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Ce

idConcept	concept	points	feedbackDescr
NULL	NULL	NULL	NULL

Figura 3.17. Tabla `concept` después de eliminar el registro

3.1.6. ASIGNACIÓN DE PUNTAJE EN LOS MINIJUEGOS EDUCATIVOS

Para realizar esta prueba se ha creado un nuevo estudiante cuyo puntaje sea cero. El registro de este nuevo estudiante cuyo nombre será “Estudiante Prueba” se muestra en la Figura 3.18.

```
5 •  SELECT * FROM game.student;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

idStudent	nickname	passwd	totalScore	lastScore	credit	idTeacher
1	Estudiante1	estudiante1	13	13	0	1
11	Estudiante Prueba	123456	0	0	0	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 3.18. Creación de estudiante de prueba en la tabla `student`

Con este perfil de estudiante se procede a ingresar en el videojuego e iniciar una partida del minijuego de trivia. Una vez finalizada la partida se ha obtenido siete puntos como se observa en la Figura 3.19.

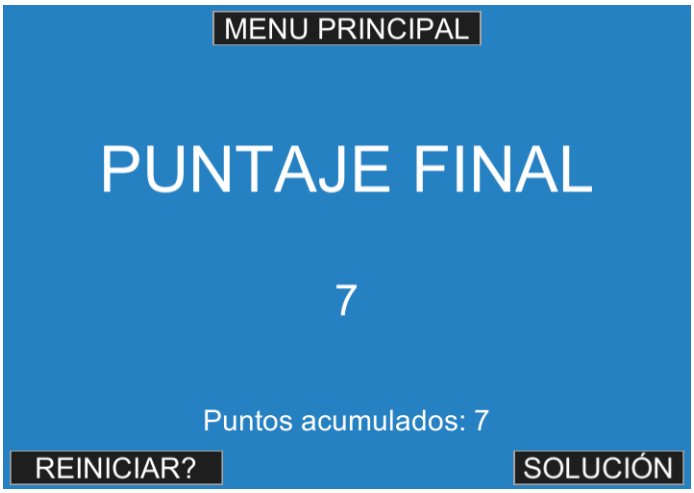

Figura 3.19. Interfaz de fin de juego del minijuego de trivia

Al realizar una nueva consulta a la tabla `student`, se puede observar en la Figura 3.20 que los puntos obtenidos durante la partida han sido asignados al estudiante correspondiente.

```
5 •  SELECT * FROM game.student;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

idStudent	nickname	passwd	totalScore	lastScore	credit	idTeacher
1	Estudiante1	estudiante1	13	13	0	1
11	Estudiante Prueba	123456	7	7	0	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 3.20. Tabla `student` después de terminar una partida

3.1.7. VERIFICACIÓN DEL *RANKING*

Desde cualquier perfil de estudiante se puede ingresar a la interfaz que muestra el *ranking* de los estudiantes de acuerdo al puntaje obtenido en cada minijuego educativo. Al ingresar en esta interfaz se muestra una lista con los estudiantes de prueba ordenados de manera descendente de acuerdo al puntaje, como se observa en la Figura 3.21. Estos registros corresponden a los mismos estudiantes que se encuentran en la base de datos dentro de la tabla `student` que se muestra en la Figura 3.22.

Figura 3.21. Interfaz de *ranking*

```
12 •  SELECT idStudent,nickname,totalScore FROM game.student where idTeacher=2 order by totalScore desc;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

idStudent	nickname	totalScore
92	Luis Criollo	232
8	edy	192
7	Cristhian	130
93	e2	82
94	Estudiante Prueba	7
NULL	NULL	NULL

Figura 3.22. Registros de estudiantes de la tabla `student`

3.1.8. DESBLOQUEO DE TORRES Y ENEMIGOS

Tras jugar la primera partida de un minijuego con el usuario llamado “Estudiante Prueba” se ha ingresado al inventario que se muestra en la Figura 3.23, constatando que todas las torres y enemigos están bloqueados debido a que los puntos acumulados son insuficientes. Al realizar una consulta a la tabla `student`, se puede observar en la Figura 3.24 que el usuario posee solo siete puntos. Después de jugar varias partidas en los minijuegos, nuevamente se ha ingresado al inventario que se muestra en la Figura 3.25 constatando que en esta ocasión varias torres y enemigos se encuentran desbloqueados. Al realizar

una consulta a la tabla `student`, se puede observar en la Figura 3.23 que el usuario ha acumulado cien puntos, con los cuales ha podido desbloquear las torres y enemigos cuyo costo de desbloqueo sea menor a cien puntos.

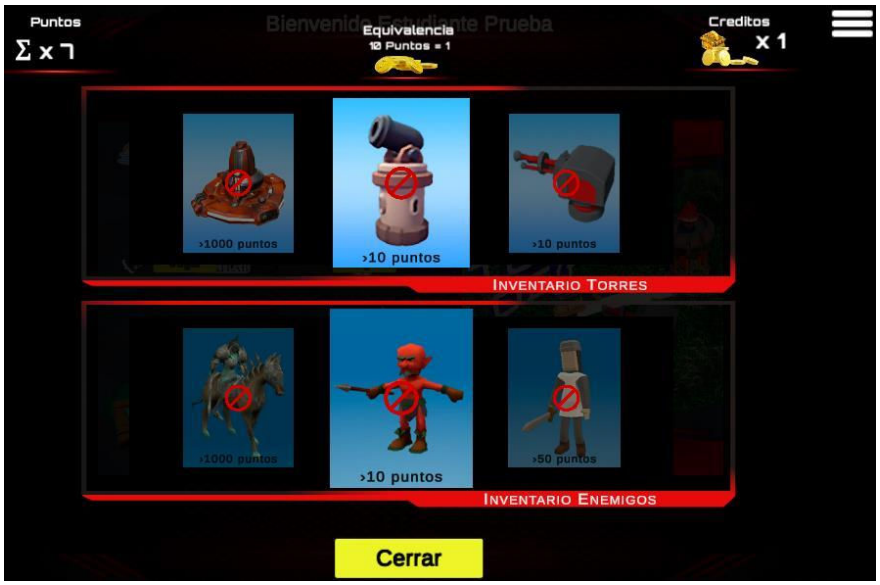

Figura 3.23. Inventario antes de acumular puntos

```
5 •  SELECT * FROM game.student;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

idStudent	nickname	passwd	totalScore	lastScore	credit	idTeacher
1	Estudiante1	estudiante1	13	13	0	1
11	Estudiante Prueba	123456	7	7	0	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 3.24. Tabla `student` antes de acumular puntaje

Figura 3.25. Inventario después de acumular puntos

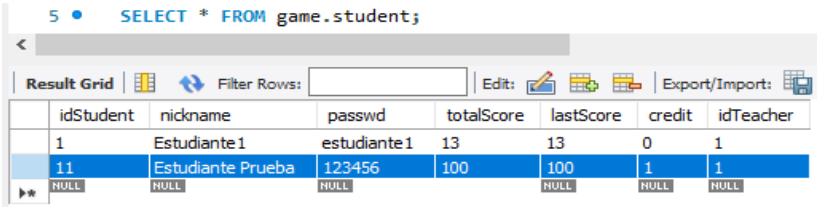

Figura 3.26. Tabla `student` después de acumular puntaje

Las pruebas realizadas anteriormente han permitido verificar el correcto funcionamiento de algunos componentes del videojuego.

3.2. ENCUESTAS DE VALIDACIÓN

Se solicitó a doce estudiantes de la Facultad de Ingeniería Eléctrica y Electrónica que hayan cursado la asignatura de Programación Avanzada utilizar el videojuego desarrollado para la realización de las pruebas respectivas. Después de utilizar el videojuego, se pidió a los estudiantes llenar una encuesta de validación. Los formatos de las encuestas aplicadas y sus resultados se encuentran en el ANEXO H.

La primera pregunta permite confirmar si los usuarios han podido ingresar al videojuego usando sus credenciales en la interfaz de *Login*. En la Figura 3.27 se indica que todos los encuestados han podido acceder al videojuego.

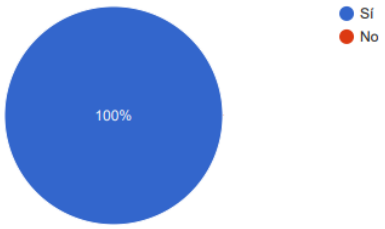

Figura 3.27. Respuesta a la primera pregunta de la encuesta de validación

La segunda pregunta permite confirmar si los usuarios han podido ingresar a cada uno de los minijuegos educativos. En la Figura 3.28 se indica que todos los encuestados han podido acceder a cada uno de los minijuegos educativos.

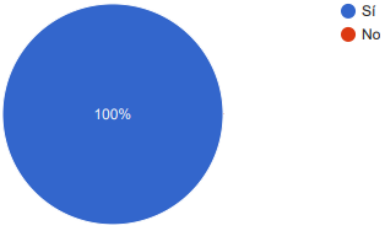

Figura 3.28. Respuesta a la segunda pregunta de la encuesta de validación

La tercera pregunta permite confirmar si los usuarios han podido elegir el capítulo de la asignatura antes de iniciar cualquiera de los minijuegos educativos. En la Figura 3.29 se indica que la totalidad de los encuestados han podido seleccionar el capítulo de la asignatura antes de iniciar una partida en cualquiera de los minijuegos educativos.

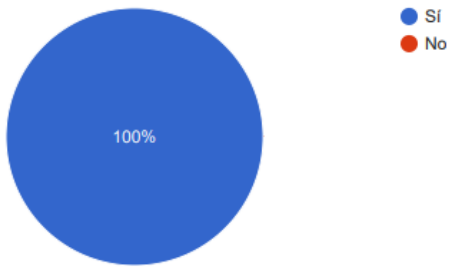

Figura 3.29. Respuesta a la tercera pregunta de la encuesta de validación

La cuarta pregunta buscaba comprobar si los desafíos educativos iban acordes con el capítulo que fue seleccionado anteriormente. En la Figura 3.30 se indica que el 91.7% de los encuestados consideran que los desafíos educativos están relacionados con el capítulo de la asignatura que seleccionaron previamente.

Mientras que, el 8.3% de los encuestados consideran que los desafíos educativos no tienen relación con el capítulo de la asignatura que seleccionaron previamente.

Dado que el usuario que respondió de manera negativa pudo consultar la retroalimentación, se considera que el contenido educativo fue desplegado de manera correcta, pero el usuario no se encuentra conforme con la temática del contenido educativo presentado durante sus partidas.

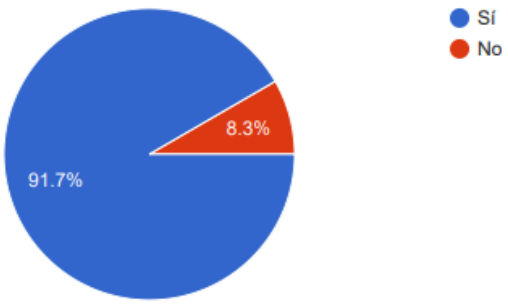

Figura 3.30. Respuesta a la cuarta pregunta de la encuesta de validación

La quinta pregunta busca comprobar si los usuarios han conseguido visualizar la retroalimentación en cada minijuego. En la Figura 3.31 se indica que la totalidad de los encuestados han conseguido visualizar la retroalimentación en cada minijuego educativo.

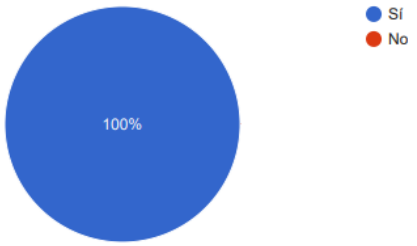

Figura 3.31. Respuesta a la quinta pregunta de la encuesta de validación

La sexta pregunta busca comprobar si los usuarios han conseguido visualizar el puntaje obtenido al finalizar una partida en cada minijuego educativo. En la Figura 3.32 se indica que la totalidad de los encuestados han conseguido visualizar el puntaje asignado al final de la partida en cada minijuego educativo.

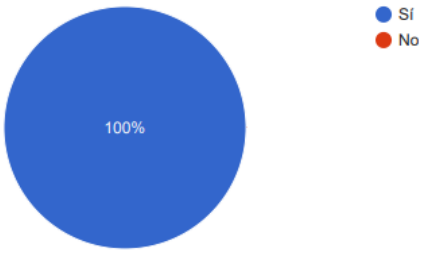

Figura 3.32. Respuesta a la sexta pregunta de la encuesta de validación

La séptima pregunta busca comprobar si tanto el *ranking* como el inventario del videojuego han sido mostrados correctamente a los usuarios. En la Figura 3.33 se indica que la totalidad de los encuestados han conseguido visualizar el *ranking* y su inventario.

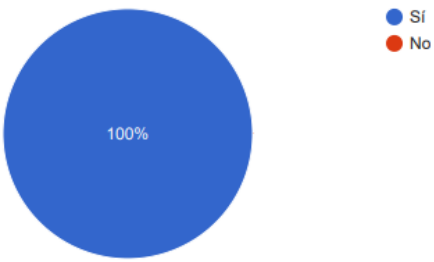

Figura 3.33. Respuesta a la séptima pregunta de la encuesta de validación

La octava pregunta busca comprobar si un usuario ha conseguido crear o unirse a una sala del juego de *Tower Defense.* En la Figura 3.34 se indica que el 91.7% de los encuestados han podido crear o unirse a una sala. Mientras que, el 8.3% de los encuestados no han conseguido crear o unirse a una sala del juego de *Tower Defense*.

La respuesta negativa de uno de los usuarios puede ser resultado de un fallo en la conexión del servidor con *Photon Unity Networking* debido a que se utilizó la versión gratuita de este *framework* lo que ocasiona que sea intermitente cuando se presenta la conexión de varios usuarios simultáneos, ya que los servidores gratuitos suelen sufrir saturaciones temporales.

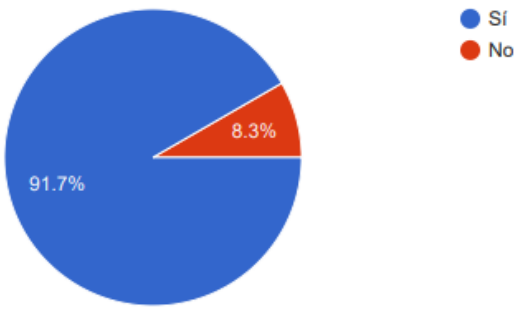

Figura 3.34. Respuesta a la octava pregunta de la encuesta de validación

La novena pregunta busca comprobar si un usuario ha conseguido desplazarse por alguno de los mapas del juego de *Tower Defense.* En la Figura 3.35 se indica que el 91.7% de los encuestados han podido desplazarse por alguno de los mapas. Mientras que, el 8.3% de los encuestados han tenido problemas para desplazarse por los mapas del juego de *Tower Defense*. La respuesta negativa de uno de los usuarios se debe a que, de acuerdo al resultado de la pregunta número ocho, el jugador no pudo conectarse a una partida de *Tower Defense.*

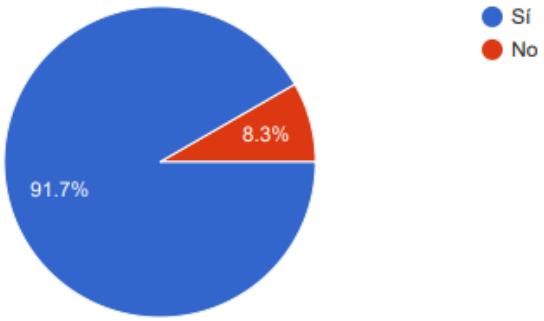

Figura 3.35. Respuesta a la novena pregunta de la encuesta de validación

La décima pregunta busca comprobar si un usuario ha conseguido colocar torres en alguno de los mapas del juego de *Tower Defense.* En la Figura 3.36 se indica que el 91.7% de los encuestados han podido colocar torres. Mientras que, el 8.3% de los encuestados ha tenido problemas para realizar la colocación de torres en el juego de *Tower Defense*. La respuesta

negativa de uno de los usuarios se debe a que, de acuerdo al resultado de la pregunta número ocho, el jugador no pudo conectarse a una partida de *Tower Defense.*

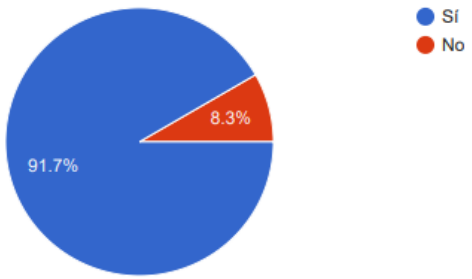

Figura 3.36. Respuesta a la décima pregunta de la encuesta de validación

La décimo primera pregunta permite al usuario calificar entre valores del 1 (Totalmente en desacuerdo) al 5 (Totalmente de acuerdo) la relación que existe entre las preguntas, conceptos y segmentos de código mostrados en los desafíos educativos; con respecto a la asignatura de Programación Avanzada. En la Figura 3.37 se indica que el 66.7% de los encuestados están totalmente de acuerdo con que el contenido educativo guarda estrecha relación con la asignatura de Programación Avanzada. El 25% de los encuestados están acuerdo con que el contenido educativo guarda estrecha relación con la asignatura de Programación Avanzada. Y finalmente un 8.7% de los encuestados están medianamente de acuerdo con que el contenido educativo guarda estrecha relación con la asignatura de Programación Avanzada.

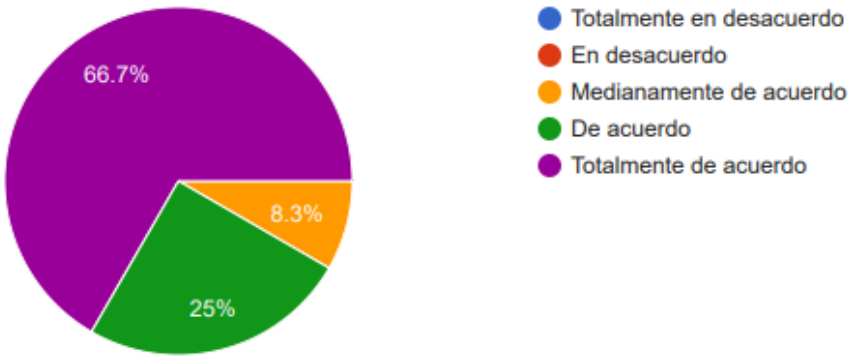

Figura 3.37. Respuesta a la décimo primera pregunta de la encuesta de validación

La décimo segunda pregunta permite conocer si el videojuego ha permitido que los usuarios aprendan o recuerden temas de la asignatura de Programación Avanzada, para esto se les pidió a los usuarios calificar entre valores del 1 (Totalmente en desacuerdo) al 5 (Totalmente de acuerdo) a esta pregunta. En la Figura 3.38 se indica que el 58.3% de los encuestados están totalmente de acuerdo con que el videojuego les ha permitido aprender

o recordar temas de la asignatura de Programación Avanzada. Mientras que, el 41.7% están de acuerdo con que el videojuego les ha permitido aprender o recordar temas de la asignatura de Programación Avanzada.

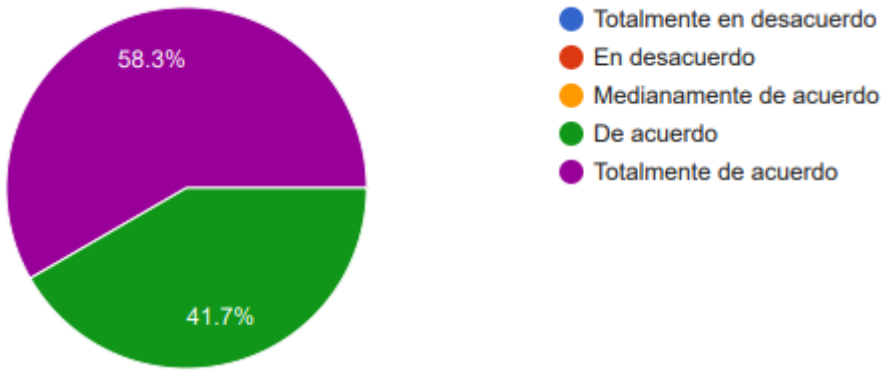

Figura 3.38. Respuesta a la décimo segunda pregunta de la encuesta de validación

La décimo tercera pregunta permite conocer si el videojuego ha influido en el nivel de conocimiento sobre la asignatura de Programación Avanzada de los usuarios, para esto se les pidió a los encuestados calificar entre valores del 1 (Menor) al 3 (Mayor) a esta pregunta. En la Figura 3.39 se indica que el 91.7% de los encuestados consideran que su conocimiento acerca de la asignatura de Programación Avanzada ha aumentado después de usar el videojuego. Mientras que, el 8.3% de los encuestados consideran que su conocimiento acerca de la asignatura de Programación Avanzada se ha mantenido igual después de usar el videojuego.

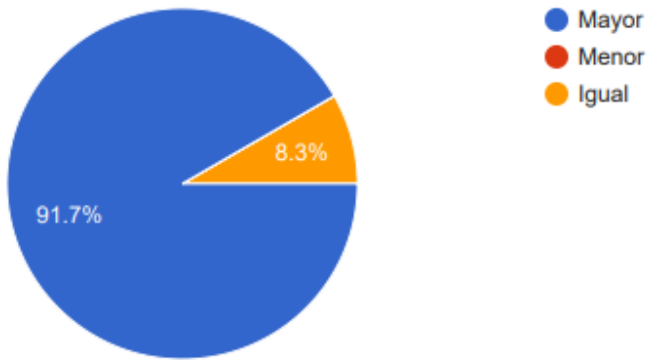

Figura 3.39. Respuesta a la décimo tercera pregunta de la encuesta de validación

La décimo cuarta pregunta permite conocer la aceptación del videojuego como una herramienta adicional para el proceso de aprendizaje de la asignatura de Programación Avanzada, para esto se les pidió a los encuestados calificar entre valores del 1 (Totalmente en desacuerdo) al 5 (Totalmente de acuerdo) a esta pregunta. En la Figura 3.40 se indica

que el 50% de los encuestados están totalmente de acuerdo en que el videojuego puede ser una considerada herramienta adicional para el aprendizaje de la asignatura de Programación Avanzada. El 41.7% de los están de acuerdo en que el videojuego puede ser una considerada herramienta adicional para el aprendizaje de la asignatura de Programación Avanzada. Mientras que, el 8.3% de los encuestados están medianamente de acuerdo en que el videojuego puede ser una considerada herramienta adicional para el aprendizaje de la asignatura de Programación Avanzada.

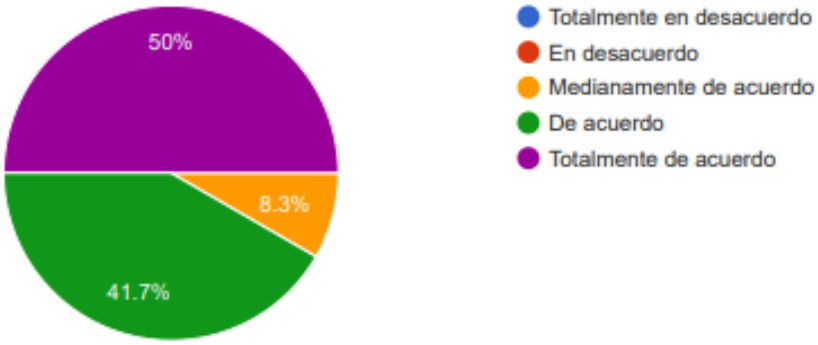

Figura 3.40. Respuesta a la décimo cuarta pregunta de la encuesta de validación

Las preguntas 15 y 16 fueron planteadas con el objetivo de reconocer posibles errores no controlados durante el proceso de desarrollo. El 58.3% de los encuestados no encontró errores en la ejecución del prototipo. Mientras que, el 41.7% de los encuestados ha encontrado errores durante la prueba de validación que realizaron, como se observa en la Figura 3.41.

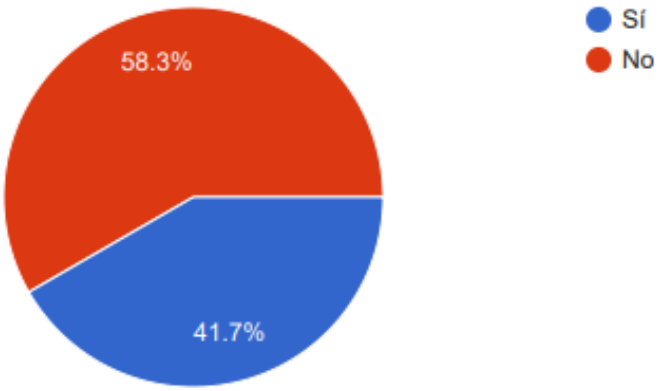

Figura 3.41. Respuesta a la décimo quinta pregunta de la encuesta de validación

Los errores encontrados por los usuarios se detallan en la Figura 3.42.

El error que se presenta de manera frecuente es el desplazamiento por los mapas del juego de *Tower Defense.* También se menciona que, existe un error gráfico en el tablero de

batalla naval. Además, se menciona que el puntaje al abandonar de manera prematura una partida del minijuego de batalla naval ocasiona la perdida de los puntos acumulados. La afirmación final no es considerada un error, ya que el minijuego de batalla naval está codificado con la finalidad de que no se almacene el puntaje en caso de que un usuario abandone una partida.

Por otra parte, para solventar los demás errores presentados, se utiliza el *sprint review* en el que se muestran las correcciones que han sido realizadas.

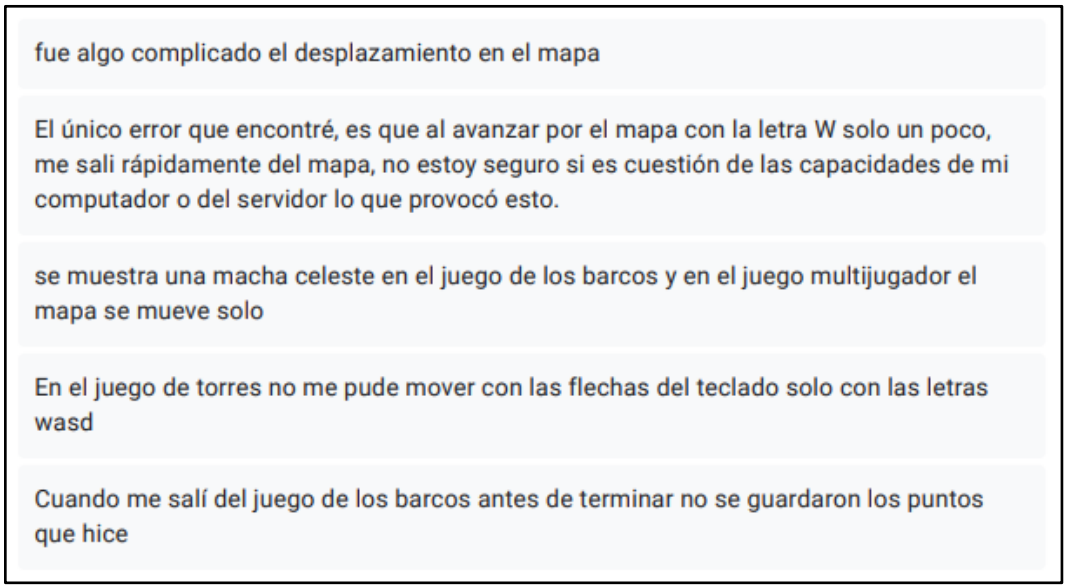

Figura 3.42. Respuestas a la décimo sexta pregunta de la encuesta de validación

3.3. CORRECCIONES – *SPRINT REVIEW*

Después de cada *sprint* se realizaron reuniones con el Director del Trabajo de Titulación, con el objetivo de presentar y revisar los avances para determinar las mejoras a realizarse. Adicionalmente, una vez que han sido procesadas las encuestas de validación se han encontrado errores que también han sido corregidos. Los detalles se describen a continuación:

- Las preguntas ingresadas en la base de datos no pueden tener opciones de respuesta tales como: “Todas las anteriores” o “Ninguna de las anteriores”. Para solventar esta corrección, dichas opciones de respuesta han sido reemplazadas por nuevas opciones de respuesta en la base de datos.
- En varios enunciados de los desafíos educativos se debe mejorar la redacción. Para solventar esta corrección se ha revisado cada enunciado de las preguntas, conceptos y segmentos de códigos, siendo modificados aquellos enunciados que resultaban complejos, ambiguos o cuya redacción no era la correcta.

- El texto mostrado en los botones del videojuego no puede estar en algunas interfaces en idioma inglés y en otras en idioma español. Por esta razón, se revisó el texto de todos los botones presentados en las interfaces y aquellos que usaban idioma inglés fueron cambiados para que muestren sus etiquetas de texto en español.
- Fue sugerido que el color de las interfaces del videojuego mantenga estética con el resto de las interfaces del videojuego. Por lo que se homogeneizó el color de las interfaces del videojuego manteniendo un tono de color rojo y negro.
- El efecto de oleaje que se muestra en el minijuego de batalla naval debe ser estilizado para que el usuario no se distraiga durante la partida. Para corregir este error visual, fue modificado el *script* llamado `Waves.cs`, que es el encargado de controlar el efecto de oleaje con el fin de reducir el tamaño de las olas, así como su frecuencia de aparición.
- Durante las partidas de batalla naval fue detectado un error cuando un misil enviado por la computadora impactaba en un barco, pero su correspondiente desafío educativo no era mostrado al jugador. Para corregir este error se realizó un cambio en el *script* llamado `Missile.cs` que es el encargado de controlar al misil, con el fin de aumentar la precisión del punto de impacto, haciendo que el misil solo detecte una verdadera colisión con los barcos o el agua y no se presente una dualidad entre ambos.
- Varios usuarios del videojuego sugirieron que se debe mejorar la forma de desplazamiento por los diferentes mapas del juego de *Tower Defense.* Motivo por el cual, se sustituyó el desplazamiento que se realizaba con el movimiento del *mouse* por el desplazamiento por teclado usando las teclas A, S, D, W.
- Los usuarios del videojuego sugirieron que el efecto de sonido del temporizador (pitido) que posee cada minijuego educativo, solo debe sonar cuando falten pocos segundos para que el tiempo se agote y no durante toda la cuenta regresiva. Por esta razón, se han modificado los *scripts* en donde se encuentran codificados los temporizadores para que el efecto de sonido solo se ejecute cuando el temporizador sea menor a cinco segundos.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- El objetivo de este Trabajo de Titulación fue desarrollar un prototipo de videojuego educativo para la asignatura de Programación Avanzada. El mismo que fue desarrollado utilizando el motor de videojuegos Unity y codificado utilizando el lenguaje de programación C#. Al concluir este Trabajo de Titulación se tiene un prototipo que consta de tres componentes principales: componente educativo del videojuego, componente lúdico del videojuego y la base de datos para almacenar la información relacionada a usuarios, preguntas, conceptos, segmentos de código, entre otros. Además, dispone de un componente administrativo para gestionar los usuarios y contenido educativo del videojuego. Para alojar a este prototipo se ha utilizado un servidor web proporcionado por Microsoft Azure.
- El componente educativo del videojuego consta de cuatro minijuegos en los que se presentarán a los estudiantes los desafíos educativos que incluyen preguntas, conceptos y segmentos de código relacionados con la asignatura de Programación Avanzada. Estos minijuegos son: Minijuego de Trivia, Minijuego de Sopa de Letras, Minijuego del Ahorcado y Minijuego de Batalla Naval.
- El componente lúdico del videojuego es un juego multijugador de *Tower Defense,* en el que los estudiantes pueden crear o unirse a salas del juego con sus compañeros con el fin de defender un castillo de las oleadas de enemigos colocando torres en los senderos de cualquiera de los tres mapas que posee este juego. Las tareas de colocación de torres y desbloqueo de objetos del juego dependen de los puntos que solo pueden ser obtenidos en el componente educativo. Esto nos permite combinar el componente lúdico del videojuego con el componente educativo.
- Para cada minijuego se desarrolló una clase principal, la cual es la responsable de iniciar y terminar el juego, además controla todos los eventos que han sido programados en el mismo. Esta clase tiene que estar asociada a la interfaz del juego, de tal manera que se presente al momento de iniciar el juego.
- El prototipo de videojuego posee una interfaz que permite al profesor agregar nuevos estudiantes, así como nuevo contenido educativo. Dentro del contenido educativo que se puede administrar se tienen las preguntas con sus respectivas alternativas de respuesta que serán mostradas en el minijuego de trivia. También se puede administrar conceptos que se mostrarán en los desafíos de los minijuegos

del ahorcado y la sopa de letras. Además, se pueden administrar segmentos de código permitiendo agregar imágenes que servirán como desafíos para el minijuego de batalla Naval.

- Para este prototipo de videojuego se utiliza la clase `UnityWebRequest` que permite gestionar las solicitudes HTTP de tipo GET y POST, lo cual permite al cliente web interactuar con el *backend* de prototipo, para que el usuario pueda utilizarlo a través de la web.
- El juego de *Tower Defense* contiene elementos como: árboles, arbustos, torres, enemigos entre otros, los cuales han sido obtenidos de la tienda de objetos de Unity llamada *Asset Store*. Sin embargo, estos objetos tenían un tamaño considerable y dado que este juego va a ser utilizado a través de la web se redujo la calidad de los mismos, proceso que permitió optimizar el consumo de recursos cuando el videojuego sea desplegado en el navegador web de los clientes.
- Unity dispone de funciones llamadas corutinas, las cuales fueron empleadas para procesar las peticiones entre el cliente y el servidor dado que permiten pausar la ejecución del código, realizar otra tarea y reanudar la ejecución del código una vez que las tareas definidas en la corutina sean completadas, estas funciones resultaron útiles para realizar tareas como la extracción de la información de la base de datos.
- Para obtener las imágenes que se van a mostrar a los jugadores en el minijuego de batalla naval desde el servidor web, se utiliza el método llamado `GetTexture` que ofrece la clase `UnityWebRequest`. Este método permite descargar una imagen con formato JPG o PNG a través de una petición HTTP de tipo GET y convertirla en una textura que puede mostrarse en Unity lo que permite simplificar el proceso de extracción y creación de una textura para obtener la imagen.
- Una de las tareas que se debe considerar al momento de desarrollar un videojuego multijugador, es que todos los jugadores puedan observar en el mismo instante los mismos objetos, esta tarea se puede simplificar utilizando *Photon Unity Networking* y empleando el método `OnPhotonSerializeView` junto con la clase `PhotonView` que permiten sincronizar los objetos que se muestran en el ambiente del juego.
- Para cargar imágenes en la interfaz de administración de segmentos de código del prototipo de videojuego se utiliza una plantilla WebGL, que permite a la aplicación web interactuar con la ventana de carga de archivos del navegador del cliente, ya que los eventos de Unity no se pueden sincronizar con los eventos del navegador debido a restricciones de seguridad que poseen algunos navegadores web.

- Como parte del desarrollo del prototipo de videojuego se empleó la herramienta llamada *Photon Unity Networking*, la cual permite simplificar el proceso relacionado con la creación de salas, el emparejamiento de jugadores y el despliegue de los componentes de un ambiente multijugador.

4.2. RECOMENDACIONES

- Como alternativa para mejorar el ingreso de información por parte de los docentes, se pueden añadir filtros que permitan buscar información dentro de las diferentes tablas que han sido desarrolladas, con la finalidad de evitar que un tema se duplique.
- Sería importante que personas relacionadas con el área de la educación desarrollen el contenido educativo para el videojuego con el fin de que este contenido sea adecuado o atractivo para los estudiantes y consigan el objetivo de que la educación a través de los juegos serios pueda desplegar en la Escuela Politécnica Nacional.
- Sería recomendable que este prototipo pueda extenderse a otras asignaturas relacionadas a la programación, puesto que, de acuerdo a los resultados obtenidos en las encuestas de validación, ha existido una gran aceptación por muchos de los estudiantes después de probar el videojuego, dado que tuvieron que resolver desafíos entretenidos y eso lo vuelve llamativo para los jugadores.
- El prototipo de videojuego implementado en este Trabajo de Titulación está diseñado para mostrar preguntas, conceptos o segmentos de código relacionados con la asignatura de Programación Avanzada en los desafíos educativos. Para futuros desarrollos se recomienda implementar un módulo que permita el ingreso de videos, tutoriales, enlaces web y contenido audiovisual en general, que complemente a la teoría con la que cuenta el videojuego actualmente.
- Dado que este prototipo de videojuego está limitado a una plataforma web. En un futuro, se podría incluir implementaciones adicionales para conseguir que el prototipo sea soportado en múltiples plataformas.
- El prototipo de videojuego realiza todas las peticiones y respuestas en texto plano, las cuales son legibles cuando son capturadas con analizadores de tráfico, por lo que es recomendable a futuro incluir protocolos de seguridad para la comunicación.
- En un futuro, debería incluirse menú de ayuda e instrucciones para cada uno de los minijuegos dado que actualmente estos puntos no fueron considerados en el plan de trabajo propuesto para este prototipo, por lo que no han sido implementados.

- Dado que la industria de los videojuegos está creciendo a un ritmo acelerado y se posiciona de manera muy fuerte en el mercado, resulta importante enseñar a los estudiantes cuyas carreras son afines a la programación acerca del desarrollo de videojuegos.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] J. Adell y L. Castañeda, «Tecnologías emergentes, ¿pedagogías emergentes?,» diciembre 2012. [En línea]. Available: https://digitum.um.es/digitum/bitstream/10201/29916/1/Adell_Castaneda_emergentes2012.pdf. [Último acceso: 03 octubre 2019].

[2] Á. Serrano Laguna, «Mejorando la evaluación de juegos serios mediante el uso de analíticas de aprendizaje,» 26 octubre 2018. [En línea]. Available: https://eprints.ucm.es/49772/1/T40486.pdf. [Último acceso: 29 octubre 2019].

[3] Mobile World Capital Barcelona, «Los efectos positivos de los videojuegos sobre la educacion,» 28 abril 2015. [En línea]. Available: https://mobileworldcapital.com/es/2015/04/28/los-efectos-positivos-de-los-videojuegos-sobre-la-educacion/. [Último acceso: 31 octubre 2019].

[4] A. Schiaffarino, «Modelo cliente servidor,» 12 marzo 2019. [En línea]. Available: https://blog.infranetworking.com/modelo-cliente-servidor/. [Último acceso: 24 Noviembre 2019].

[5] CodeCombat Inc, «CodeCombat,» febrero 2013. [En línea]. Available: https://codecombat.com. [Último acceso: 2 enero 2020].

[6] Kahoot!, agosto 2016. [En línea]. Available: https://kahoot.com/. [Último acceso: 2 enero 2020].

[7] A. Yassine, D. Chenouni, M. Berrada y A. Tahiri, «A Serious Game for Learning C Programming Language Concepts Using Solo Taxonomy,» 01 03 2017. [En línea]. Available: https://online-journals.org/index.php/i-jet/article/view/6476/4321. [Último acceso: 25 noviembre 2019].

[8] R. Samaniego Ocampo, S. Cruz Naranjo, M. Arboleda Berrezueta, J. Encalada Cuenca y B. Jimenez Villamar, «Diseño, desarrollo y validación de un juego serio en educación superior. Un caso de estudio,» 10 agosto 2017. [En línea]. Available: investigacion.utmachala.edu.ec/proceedings/index.php/utmach/article/download/231/200/. [Último acceso: 14 diciembre 2019].

[9] B. Marcano, «Juegos serios y entrenamiento en la sociedad digital,» noviembre 2008. [En línea]. Available: https://www.redalyc.org/pdf/2010/201017343006.pdf. [Último acceso: 16 diciembre 2019].

[10] D. Michael y S. Chen, Serious Games: Games that eduacate, train and inform, Boston: Thomson course technology PTR, 2006.

[11] E. Guerra Cruz, «¿Podrá la tecnología reemplazar a los maestros?,» 4 agosto 2016. [En línea]. Available: http://www.educacionfutura.org/podra-la-tecnologia-reemplazar-a-los-maestros/. [Último acceso: 18 diciembre 2019].

[12] «Hazmat: Hotzone,» Entertainment Technology Center and Carnegie Mellon University, 2005. [En línea]. Available: https://www.etc.cmu.edu/projects/hazmat_2005/screenshot.php?item=36. [Último acceso: 4 enero 2020].

[13] U. Ritterfeld, M. Cody y P. Vorderer, Serious games mechanisms and effects, New York: Routledge, 2009.

[14] E. Morales Corral, «El uso de los videojuegos como recurso de aprendizaje en educación primaria y teoría de la comunicacion,» 2009. [En línea]. Available: https://dialnet.unirioja.es/servlet/articulo?codigo=3719704. [Último acceso: 21 diciembre 2019].

[15] J. Breuer y G. Bente, «Why so serious? On the relation of serious games and learning,» 27 abril 2012. [En línea]. Available: https://hal.archives-ouvertes.fr/hal-00692052/document. [Último acceso: 20 diciembre 2019].

[16] V. Hallett, «Dance Dance Revolution hits D.C. schools,» 6 febrero 2013. [En línea]. Available: https://www.washingtonpost.com/lifestyle/wellness/dance-dance-revolution-hits-dc-schools/2013/02/05/31d440a4-6c83-11e2-bd36-c0fe61a205f6_story.html. [Último acceso: 5 enero 2020].

[17] G. A. Armerdáriz Barreno y M. G. Saltos Guaraca, «Adaptación de las metodologias ágiles scrum y extreme game development en una metodologia para desarrollo de videojuegos en android. Caso práctico: Desarrollo de un videojuego,» Junio 2013. [En línea]. Available:

http://dspace.espoch.edu.ec/bitstream/123456789/2711/1/18T00530.pdf. [Último acceso: 21 Diciembre 2019].

[18] G. A. Morales Urrutia, C. E. Nava López, L. F. Fernández Martínez y M. A. Rey Corral, «Procesos de desarrollo para videojuegos,» enero 2010. [En línea]. Available: https://dialnet.unirioja.es/servlet/articulo?codigo=3238114. [Último acceso: 21 diciembre 2019].

[19] K. Schwaber y J. Sutherland, «La Guía Definitiva de Scrum: Las Reglas del Juego,» julio 2013. [En línea]. Available: https://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ES.pdf. [Último acceso: 5 enero 2020].

[20] J. Palacio, Flexibilidad con SCRUM, Safe creative, 2007.

[21] N. Acerenza, A. Coppes, G. Mesa, A. Viera, E. Fernández, T. Laurenzo y D. Vallespir, «Una metodología para desarrollo devideojuegos: versión extendida,» 2009. [En línea]. Available: https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/3420/1/TR0913.pdf. [Último acceso: 22 diciembre 2019].

[22] N. Acerenza, A. Coppes, G. Mesa, A. Viera, E. Fernández, T. Laurenzo y D. Vallespir, «Una Metodologıa para Desarrollo de Videojuegos,» 28 agosto 2009. [En línea]. Available: https://www.fing.edu.uy/inco/grupos/gris/wiki/uploads/Proceedings/ASSE_2009_16.pdf. [Último acceso: 24 diciembre 2019].

[23] J. Palacio, «Lo que entendemos por Scrum,» 11 marzo 2016. [En línea]. Available: https://scrummanager.net/blog/author/jpalacio/. [Último acceso: 24 diciembre 2019].

[24] Unity Technologies, «Unity Documentation,» diciembre 2016. [En línea]. Available: https://docs.unity3d.com/es/530/Manual/UnityManual.html. [Último acceso: 26 diciembre 2019].

[25] F. Ruiz Pérez, «Desarrollo de un videojuego para dispositivos móviles,» junio 2016. [En línea]. Available: https://rua.ua.es/dspace/bitstream/10045/56709/1/Desarrollo_de_un_videojuego_para_dispositivos_moviles_Ruiz_Perez_Francisco.pdf. [Último acceso: 26 diciembre 2019].

[26] Microsoft, «Visual Studio docs,» 18 marzo 2019. [En línea]. Available: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019. [Último acceso: 29 diciembre 2019].

[27] Á. Robledano, «Qué es MySQL: Características y ventajas,» 24 septiembre 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-mysql/. [Último acceso: 29 diciembre 2019].

[28] R. M. Ballester, «Entorno Virtual Multiusuario en Unity,» 1 julio 2016. [En línea]. Available: https://riunet.upv.es/handle/10251/92573. [Último acceso: 30 diciembre 2019].

[29] Exit Games, «Photon,» Exit Games, [En línea]. Available: https://doc.photonengine.com/en-us/realtime/current/getting-started/realtime-intro. [Último acceso: 20 julio 2020].

[30] G. Raunaq Singh, «Introduction to Multiplayer Games With Unity and Photon,» 19 junio 2019. [En línea]. Available: https://www.raywenderlich.com/1142814-introduction-to-multiplayer-games-with-unity-and-photon#toc-anchor-002. [Último acceso: 22 julio 2020].

[31] E. M. Merizalde Herrería y J. D. Ortiz Báez, «Desarrollo de un prototipo de videojuego distribuido basado en realidad virtual para dispositivos Android,» enero 2018. [En línea]. Available: https://bibdigital.epn.edu.ec/handle/15000/19230. [Último acceso: 12 febrero 2020].

[32] Young Engineers Of Today, «Code Combat,» febrero 2016. [En línea]. Available: https://youngengineersoftoday.com/wp-content/uploads/2016/02/0b164b6dc6ccabfcadb920d42eebb11c.pdf. [Último acceso: 06 febrero 2020].

[33] J. H. Cernuda, «Introducción a la programación con CodeCombat (Python),» marzo 2017. [En línea]. Available: http://jhcernuda.com/blog/wp-content/uploads/2017/03/codecombat.pdf. [Último acceso: 07 febrero 2020].

[34] R. Herrero Abellán, «Manual de kahoot para docentes,» febrero 2018. [En línea]. Available: https://erasmusmedina.files.wordpress.com/2018/02/manual-de-kahoot-para-docentes.pdf. [Último acceso: 14 febrero 2020].

[35] Microsoft, «Microsoft Azure,» 2 Marzo 2020. [En línea]. Available: https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-b-series-burstable. [Último acceso: 15 diciembre 2020].

[36] M. F. Jiménez Salvador, «La filosofía serious games aplicada a la identificación de riesgos en project management. Diseño y validación de un modelo aplicado,» enero 2018. [En línea]. Available: https://riunet.upv.es/bitstream/handle/10251/97653/PSC6310743_TFM_1515752542905363635489393539 1808.pdf. [Último acceso: 16 diciembre 2019].

[37] J. J. Velasco Rivera, «Apple II, el ordenador que hizo avanzar a una industria,» 25 Agosto 2011. [En línea]. Available: https://hipertextual.com/2011/08/apple-ii-computador-avanzo-la-industria. [Último acceso: 17 diciembre 2019].

[38] R. C. Salas, J. Lau y J. Martínez Catillo, «Los modelos tecnoeducativos revolucionando el aprendizaje del siglo XXI,» Diciembre 2014. [En línea]. Available: https://www.uv.mx/personal/iesquivel/files/2015/03/los_modelos_tecno_educativos_ _revolucionando_el_aprendizaje_del_siglo_xxi-4.pdf. [Último acceso: 18 Diciembre 2019].

[39] R. Barzanallana, «Apuntes de las asignaturas sobre informática, y algo más,» 13 agosto 2013. [En línea]. Available: https://www.um.es/docencia/barzana/II/Ii04.html. [Último acceso: 18 diciembre 2019].

[40] C. López Reventós, «El videojuego como herramienta educativa. Posibilidades y problemáticas acerca de los serious games,» Abril 2016. [En línea]. Available: http://www.udgvirtual.udg.mx/apertura/index.php/apertura/article/view/825/539. [Último acceso: 19 Diciembre 2019].

[41] J. V. Quiñónez Sical, «El uso de Photoshop como herramienta publicitaria,» marzo 2010. [En línea]. Available: http://biblioteca.usac.edu.gt/tesis/16/16_0746.pdf. [Último acceso: 29 diciembre 2019].

[42] J. Wilson, «Apple's classroom of tomorrow,» Hulton Archive, 1 junio 1994. [En línea]. Available: https://allthatsinteresting.com/90s-pictures#3. [Último acceso: 3 enero 2020].

[43] J. Gutierrez, «¿Qué es un framework web?,» 2014. [En línea]. Available: http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf. [Último acceso: 4 Octubre 2020].

ANEXOS

ANEXO A. Sílabo de la asignatura de Programación Avanzada

ANEXO B. Contenido educativo (Preguntas, conceptos y segmentos de código)

ANEXO C. Historias de Usuario

ANEXO D. Diagrama de clases que heredan de clase `MonoBehaviour`

ANEXO E. *Mockups*

ANEXO F. *Scripts* de la base de datos

ANEXO G. *Scripts* codificados en lenguaje PHP

ANEXO H. Encuestas de validación

ANEXO I. Proyecto de Unity (comprimido)

ANEXO J. Aplicación Web (comprimido)

*Debido a la extensión, todos los Anexos se han incluido en el disco compacto adjunto a este documento.

ORDEN DE EMPASTADO