ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UNA APLICACIÓN WEB PARA LA ENSEÑANZA DEL DISEÑO DE SOFTWARE OO CON ENFOQUE EN ABSTRACCIÓN

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

RODAS LEÓN CARLOS ANDRÉS
carlos.rodas@epn.edu.ec

DIRECTOR: FLORES NARANJO PAMELA CATHERINE, Ph.D.
pamela.flores@epn.edu.ec

Quito, mayo 2022

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Carlos Andrés Rodas León, bajo mi supervisión.

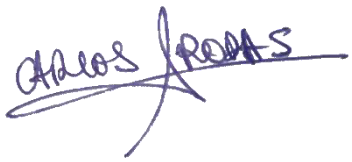
Flores Naranjo Pamela Catherine, Ph.D.

DIRECTOR

DECLARACIÓN

Yo, Carlos Andrés Rodas León, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual, correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.



Rodas León Carlos Andrés

AGRADECIMIENTO

Agradezco a mi familia pues sin su apoyo nada de esto sería posible.

Agradezco a mi directora de tesis, la Dra. Pamela Flores, quien vio en mí el potencial para ser su ayudante de investigación. Su apoyo, tenacidad y sabiduría han contribuido enormemente en mi vida, inspirándome a conseguir mis metas.

Agradezco a la Escuela Politécnica Nacional, a la facultad de Ingeniería de Sistemas, y a sus autoridades, por brindarme una educación completa, gratuita y de calidad.

Agradezco a todos los profesores, amigos y compañeros de carrera que conocí a lo largo de mi formación profesional. Cada experiencia vivida a través de mi carrera universitaria ha sido valiosa, gracias por ser parte de ella.

Rodas León Carlos Andrés

DEDICATORIA

A mi papá por su sabiduría.

A mi mamá por su amor.

A mi hermana Isabel por sus consejos.

A mi hermana Carolina por siempre estar.

Rodas León Carlos Andrés

ÍNDICE DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

El presente proyecto integrador tiene como objetivo el desarrollo e implementación de una aplicación web focalizada en la enseñanza de conceptos de diseño en programación orientada a objetos con un enfoque especial en la abstracción. El software utiliza experiencias interactivas para mejorar el interés de los estudiantes por la materia.

Para el desarrollo de la aplicación se utilizó el motor gráfico multiplataforma Unity, el entorno de desarrollo integrado Visual Studio, y el lenguaje de programación C# (C Sharp), siguiendo el marco de trabajo ágil Scrum.

Durante el desarrollo del proyecto se obtuvieron los requerimientos funcionales y no funcionales de la aplicación, se diseñó la arquitectura del aplicativo, se implementaron los requerimientos en un número determinado de Sprints, manteniendo reuniones semanales con el Product Owner, se hicieron pruebas de funcionalidad y usabilidad para comprobar que se cumpla con los estándares de calidad requeridos por los stakeholders y, finalmente, se desplegó la aplicación en un dominio web.

Gracias a las pruebas realizadas al finalizar el desarrollo de la aplicación se pudo comprobar que las necesidades de los stakeholders fueron cumplidas, que los estudiantes mostraron gran interés por el uso de nuevas tecnologías para la educación, y se recomienda el uso de estas herramientas para futuros estudios, no solo en materias relacionadas a la informática, sino en toda clase de asignaturas.

Palabras clave: Abstracción, Aplicación Web, Diseño de Software, Scrum, Unity.

ABSTRACT

The objective of this integrative project is the development and implementation of a web application focused on teaching design concepts in object-oriented programming with a special focus on abstraction. The software uses interactive experiences to enhance students' interest in the subject.

For the development of the web application, the Unity multiplatform graphics engine, the Visual Studio integrated development environment, and the C# (C Sharp) programming language were used, following the Scrum agile framework.

During the development of the project, the functional and non-functional requirements of the application were obtained, the architecture of the application was designed, the requirements were implemented in a certain number of Sprints, holding weekly meetings with the Product Owner, functionality and usability tests were carried out to verify that the quality standards required by the stakeholders were met and, finally, the application was deployed in a web domain.

Thanks to the tests carried out at the end of the development of the application, it was possible to verify that the needs of the stakeholders were met, that the students showed great interest in the use of new technologies for education, and the use of these tools is recommended for future studies, not only in matters related to computing, but in all kinds of subjects.

Keywords: Abstraction, Web Application, Software Design, Scrum, Unity.

CAPÍTULO 1 – INTRODUCCIÓN

1.1. Planteamiento del problema

El Ciclo de Vida de Desarrollo de Software se puede definir como un proceso iterativo en el cual se identifican cuatro etapas principales: Requisitos, Diseño, Desarrollo y Pruebas [1]. En particular, el diseño de software agrupa el conjunto de principios, conceptos y prácticas que llevan al desarrollo de un sistema o producto de alta calidad [2]; es el proceso que permite modelar el sistema o producto que se va a construir, velando por la mejora en su calidad, antes de generar código [2]. Justamente en la etapa de Diseño de Software, el concepto de abstracción juega un papel importante al ser uno de los fundamentos clave de la ingeniería de software [3]. Entiéndase abstracción como un cambio de fenómenos concretos y tangibles a conceptos en un nivel superior; que proporciona límites conceptuales definidos en relación con la perspectiva del espectador [3-5]. Ya en términos de diseño de software orientado a objetos (OO), abstracción puede ser entendida como el acto de crear clases para simplificar aspectos de la realidad usando distinciones inherentes al problema [6] colocar un nombre adecuado a una clase [7], separar lo esencial de lo auxiliar [8], el uso de clases abstractas [9] y otras manifestaciones.

La literatura evidencia que el aprendizaje de diseño de software orientado a objetos es a menudo difícil [10-12], y construir cursos de diseño de software para estudiantes universitarios es una tarea desafiante. En este sentido, los estudiantes enfrentan muchas dificultades debido a que los conceptos de objeto, clase y herencia son de índole abstracto [10]. Según el estudio “PII-17-11: Estudio longitudinal cualitativo sobre la abstracción y el diseño de software a través de ejercicios de diseño” [3], se ha demostrado que durante la enseñanza de diseño de software, los estudiantes muestran dificultades debido a la transferencia del enfoque estructurado al enfoque orientado a objetos, falta de entendimiento de los conceptos básicos del enfoque orientado a objetos, la copia estricta de la realidad al construir objetos [3]; dificultades que van de la mano con problemas relacionados a una falta de habilidad para abstraer.

En el mundo globalizado en el que vivimos el uso de herramientas digitales para la enseñanza no es la excepción sino la norma. En un estudio realizado por Stelios Xinogalos, un investigador griego, se muestra que una herramienta de enseñanza digital puede: *enseñar conceptos de programación sin importar la edad del estudiante, enseñar conceptos básicos de programación y principios de las técnicas de programación orientada a objetos, ayudar a los estudiantes a superar las dificultades enfrentadas en una enseñanza convencional, enseñar técnicas de resolución de problemas* [11], entre otras cosas.

Actualmente, las aplicaciones web funcionan como herramientas sofisticadas de cómputo que realizan sus acciones a través de un navegador web con conexión a internet [2], por lo que su uso es ampliamente difundido. Una aplicación web es software que funciona a través de un servidor no local, mediante archivos de hipertexto almacenados en la web, vinculados entre sí [2]. Según Robert S. Pressman, *las aplicaciones web están evolucionando hacia ambientes de cómputo sofisticados que no sólo proveen características aisladas, funciones de cómputo y contenido para el usuario final, sino que también están integradas con bases de datos corporativas y aplicaciones de negocios* [2].

Por otro lado, para el desarrollo de la aplicación, la plataforma Unity permite crear juegos, aplicaciones y experiencias inmersivas e interactivas, en 2D y 3D, a artistas, diseñadores y desarrolladores [13]. Unity funciona con el lenguaje de programación C# y permite con la misma base de código generar aplicaciones nativas para Windows, MacOS, Linux, Android, iOS, WebGL, consolas de videojuegos y gafas de realidad virtual [14]. Por su versatilidad y facilidad de uso será la herramienta utilizada para la creación del software propuesto.

El presente proyecto busca desarrollar una aplicación web, que funcione como un entorno interactivo para la enseñanza de conceptos de diseño de software, con un enfoque en la abstracción por su importancia para comprender conceptos de programación orientada a objetos [3, 5]. Con esta aplicación se pretende brindar una herramienta, para estudiantes y profesores de materias de pregrado relacionadas a la informática, que aumente su capacidad para abstraer y mejore el aprendizaje de diseño de software.

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar una aplicación web para la enseñanza y aprendizaje del diseño de software orientado a objetos con enfoque en abstracción, empleando la plataforma Unity y el marco de trabajo Scrum.

1.2.2. Objetivos Específicos

- Definir requerimientos funcionales y no funcionales de la aplicación.
- Diseñar la arquitectura de la aplicación.
- Implementar los requerimientos en un número determinado de Sprints.
- Implementar la aplicación web en un dominio específico en el internet.
- Ejecutar pruebas de funcionalidad y de usabilidad de la aplicación.

CAPÍTULO 2 – MARCO TEÓRICO

2.1. Diseño orientado a objetos

“El diseño de software es una actividad creativa donde se identifican los componentes del software y sus relaciones, con base en los requerimientos de un cliente” [15].

Un sistema orientado a objetos se constituye con objetos que interactúan y mantienen su propio estado local y ofrecen operaciones sobre dicho estado. Los procesos de diseño orientado a objetos implican el diseño de clases de objetos y las relaciones entre dichas clases; tales clases definen tanto los objetos en el sistema como sus interacciones [15].

- **Lenguaje de Modelado Unificado (UML)**

Es “un lenguaje estándar para escribir diseños de software, puede usarse para visualizar, especificar, construir y documentar los artefactos de un sistema de software intensivo” [16]. Como los arquitectos de edificios crean planos para que los use una compañía constructora, los arquitectos de software crean diagramas en UML para ayudar a los desarrolladores en la construcción de software [2]. Permiten comprender y especificar con mayor facilidad un sistema informático, y explicar su diseño a otros.

- **Diagrama de Clases UML**

El estándar UML proporciona entre sus diagramas, el conocido diagrama de clase, el cual aporta una visión estructurada de un sistema, mostrando sus clases, sus atributos, sus métodos y las relaciones que existen entre las clases [2].

Los elementos principales de estos diagramas son las clases, estas son representadas en forma de caja como se ve en la Figura 1, cada caja se divide en tres partes [2]:

- La parte superior contiene el nombre de la clase.
- La sección media contiene sus atributos. Un atributo es algo que un objeto de dicha clase conoce o puede proporcionar todo el tiempo. Podrían ser

valores que la clase puede calcular a partir de sus variables o valores que puede obtener de otros objetos de los cuales se compone.

- La tercera sección contiene los métodos de la clase. Estos representan las operaciones o comportamientos de la clase, es decir, lo que pueden hacer.

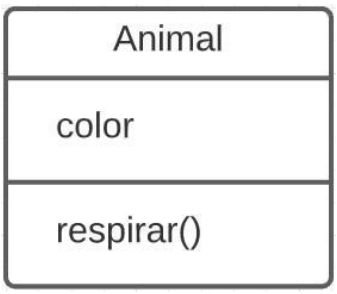


Figura 1 - Ejemplo de una clase en UML

Las relaciones entre las clases son representadas por distintos tipos de líneas como se ve en la Figura 2. Un ejemplo de cómo son utilizadas en el diagrama de clases UML se puede ver en la Figura 3.

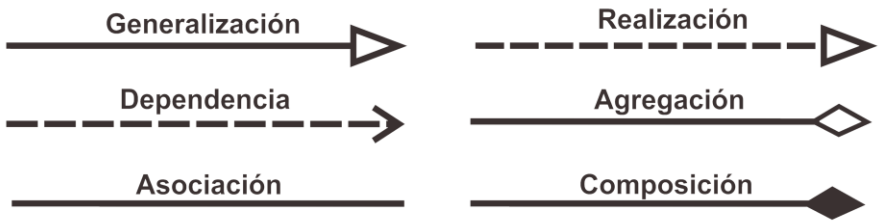


Figura 2 - Tipos de relaciones en el diagrama de clases UML

A continuación, una explicación de los tipos de relaciones que existen en el diagrama de clases UML [2]:

- ***Generalización**:* relación de una súper clase o clase padre con una subclase o clase hija. Significa que la subclase hereda atributos y métodos de la súper clase. El hijo puede agregar atributos y métodos propios.
- ***Realización:*** relación de abstracción especializada entre dos clases, una clase representa una especificación (por ejemplo, una interface) y la otra clase representa una implementación de dicha especificación.
- ***Dependencia**:* relación entre dos clases donde un cambio en la especificación de un elemento de una clase puede afectar a la otra clase que la utiliza, pero no a la inversa.

- ***Asociación:*** es una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro. Pueden ser bidireccionales cuando ambas clases usan elementos de la otra o direccionales si solo una clase usa elementos de otra.
- ***Agregación:*** indica que una clase es parte de otra clase. La destrucción del compuesto no significa la destrucción de los componentes. Habitualmente se da con mayor frecuencia que la composición.
- ***Composición:*** indica que una clase es parte de otra clase. La destrucción del compuesto significa la destrucción de los componentes.

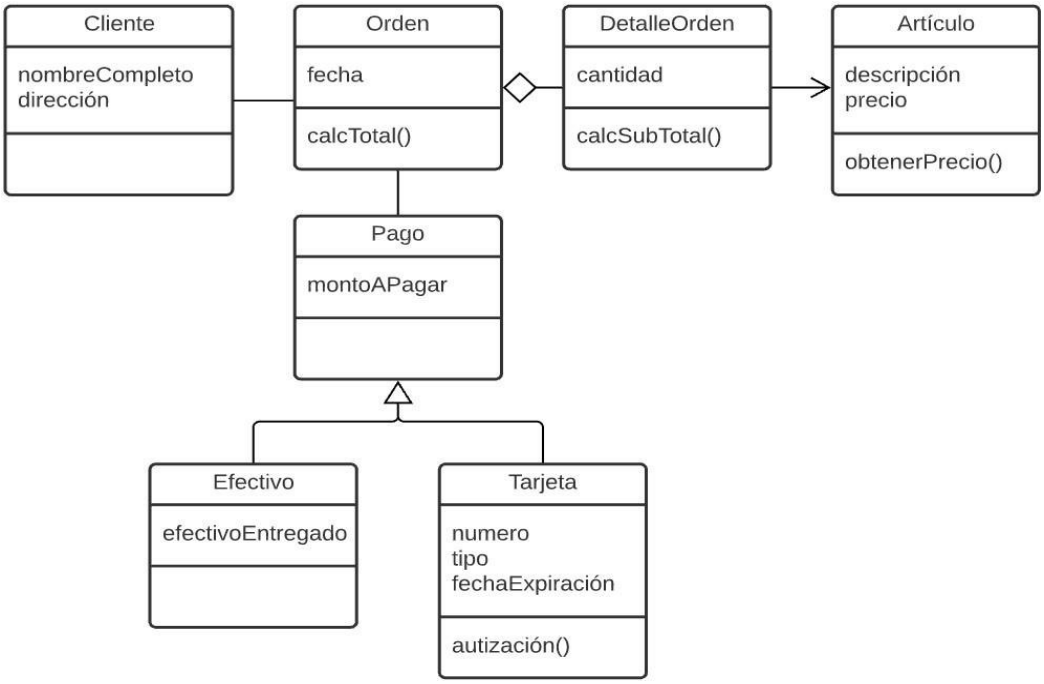


Figura 3 - Ejemplo de Diagrama de Clases: Sistema para realizar órdenes

2.2. Abstracción

La abstracción se utiliza mucho en disciplinas como el arte y la música, pero también es de suma importancia para las matemáticas, informática e ingeniería [17]. Puede ser definida de varias formas [17]:

- El acto de retirar o quitar algo.
- El acto o proceso de dejar fuera de consideración una o más propiedades de un objeto complejo para atender a otras.

- El proceso de formular conceptos generales mediante la substracción de propiedades comunes.
- Definir algo complejo de una manera simple.

En el campo del desarrollo de software ha sido estrechamente relacionada con la formación de algoritmos [18], la automatización de notaciones y modelos [19], la descomposición de tareas complejas [20] y la generalización mediante la definición de patrones [18]. Ya que juega un papel importante como herramienta para gestionar la complejidad del software [3], se puede aseverar que es una pieza fundamental en el diseño de software con enfoque orientado a objetos.

2.3. Aplicación Web

Las aplicaciones web son una categoría de software centrado en el internet, agrupa una amplia gama de funciones. Puede referirse desde a una simple página web que ayude al consumidor a calcular el pago del arrendamiento de un automóvil, a una aplicación comercial o educativa, hasta un sitio web integral que proporcione servicios completos de viaje para gente de negocios y vacacionistas [2]. En esta categoría se incluyen sitios web completos, funcionalidad especializada dentro de sitios web y aplicaciones de procesamiento de información que residen en internet o en redes internas de una organización [2].

2.4. Scrum

Scrum es un marco de trabajo ligero que ayuda a las personas, los equipos y las organizaciones a generar valor a través de soluciones adaptables para problemas complejos [21]. Se basa en un marco de desarrollo en el que equipos multifuncionales desarrollan productos o proyectos de manera iterativa e incremental. Estructura el desarrollo en ciclos de trabajo denominados Sprints. Estas iteraciones duran entre una a cuatro semanas y se llevan a cabo una tras otra sin pausa [22]. Scrum emplea un enfoque iterativo e incremental para optimizar la previsibilidad y controlar el riesgo. Scrum involucra a grupos de personas que colectivamente tienen las habilidades para realizar el trabajo o la capacidad para adquirirlas según sea necesario [21].

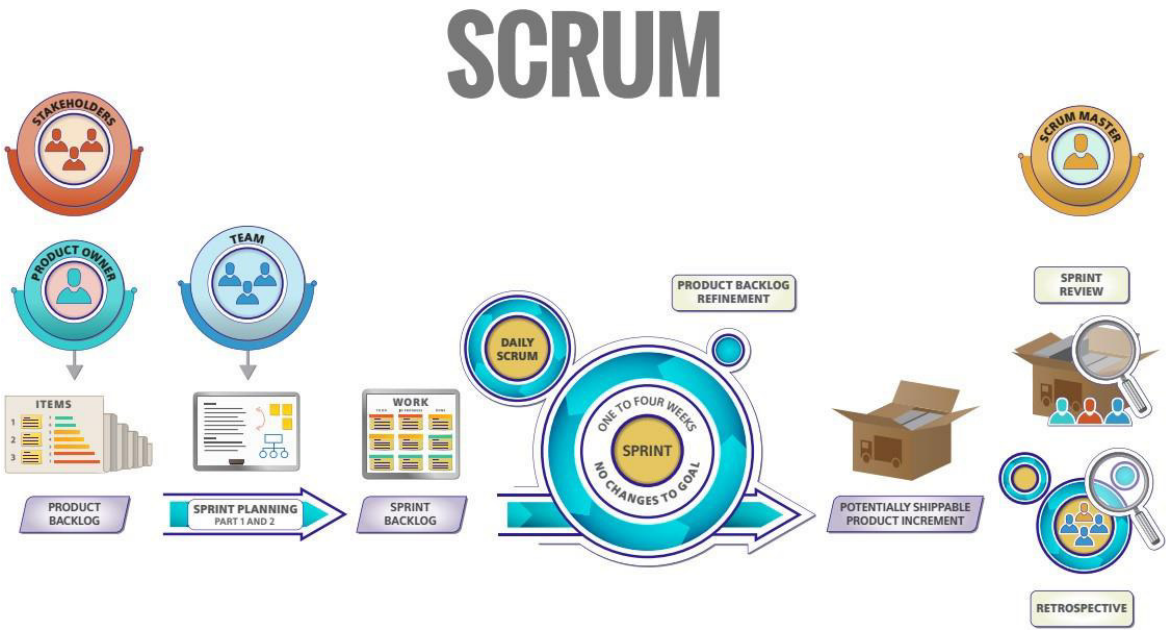


Figura 4 - Descripción general de SCRUM [22]

2.4.1 Roles de Scrum

En Scrum, hay tres roles: “Product Owner”, “Development Team” (o solo “Team”) y “Scrum Master”. Juntos, estos se conocen como “Scrum Team” [22].

- **Product Owner (Dueño del Producto)**

El Product Owner es responsable de maximizar el retorno de la inversión (ROI) identificando las características del producto, traduciéndolas a una lista de prioridades, decidiendo cuál debe estar en la parte superior de la lista para el próximo Sprint y continuamente priorizando y refinando la lista. En algunos casos, el Product Owner y el cliente son la misma persona; esto es común para aplicaciones internas [22].

- **Scrum Master**

Ayuda al equipo a aprender y aplicar Scrum para lograr valor comercial. Hace todo lo que está a su alcance para ayudar al equipo, al propietario del producto y a la organización a tener éxito. No es el gerente de los miembros del equipo, ni tampoco es un gerente de proyecto, líder de equipo o representante de equipo. En cambio, el Scrum Master sirve al equipo; ayuda a eliminar los impedimentos, protege al equipo de interferencias externas y ayuda al equipo a adoptar prácticas de

desarrollo modernas [22]. Él o ella educa, entrena y guía al Product Owner, al Development Team y al resto de la organización en el uso hábil de Scrum.

- **Development Team (Equipo de desarrollo)**

El equipo construye el producto que el Product Owner indica: la aplicación o el sitio web, por ejemplo. El equipo en Scrum es "multifuncional", incluye toda la experiencia necesaria para entregar el producto potencialmente entregable en cada Sprint, y es autogestionado, con un alto grado de autonomía y responsabilidad. El equipo decide cuántos elementos (del conjunto ofrecido por el Product Owner) construir en un Sprint y cuál es la mejor manera de lograr ese objetivo [22].

2.4.2. Eventos de Scrum

Los eventos se utilizan en Scrum para crear regularidad y minimizar la necesidad de reuniones no definidas anteriormente. De manera óptima, todos los eventos se llevan a cabo al mismo tiempo y en el mismo lugar para reducir su complejidad [21].

- **Sprint**

Los Sprints son la parte más fundamental de Scrum, donde las ideas se convierten en valor. Son eventos de duración fija, de entre una a cuatro semanas, realizados para crear coherencia en el desarrollo del proyecto. Un nuevo Sprint comienza inmediatamente después de la conclusión del Sprint anterior. Todos los otros eventos ocurren dentro de un Sprint [21]. Durante el Sprint:

  - No se realizan cambios que pongan en peligro el objetivo del Sprint;
  - La calidad no disminuye;
  - El Product Backlog se refina según sea necesario; y,
  - El alcance se puede aclarar y renegociar con el Product Owner a medida que se aprende más.

- **Sprint Planning (Planificación de Sprint)**

Sprint Planning inicia el Sprint al establecer el trabajo que se realizará. Este plan es creado por el trabajo colaborativo de todo el Equipo Scrum. El Product Owner se asegura de que los asistentes estén preparados para discutir los elementos más importantes del Product Backlog. El equipo Scrum también puede invitar a otras

personas a asistir a Sprint Planning para brindar asesoramiento [21]. Cada Sprint Planning requiere contestar las siguientes preguntas:

- ¿Por qué es valioso este Sprint?
- ¿Qué se puede hacer con este Sprint?
- ¿Cómo se realizará el trabajo elegido?

- **Daily Scrum (Scrum Diario)**

El Daily Scrum es un evento de 15 minutos para los desarrolladores del equipo Scrum. Normalmente se lleva a cabo a la misma hora y en el mismo lugar todos los días hábiles del Sprint. Si el Product Owner o Scrum Master están trabajando activamente en elementos del Sprint Backlog, también participan en el Daily Scrum. El propósito del Daily Scrum es inspeccionar el progreso hacia el Sprint Goal y adaptar el Sprint Backlog según sea necesario, ajustando el próximo trabajo planificado [21].

- **Sprint Review (Revisión de Sprint)**

El equipo Scrum y las partes interesadas revisan lo que se logró en el Sprint y lo que ha cambiado en su entorno. Con base en esta información, los asistentes colaboran sobre qué hacer a continuación. El propósito de Sprint Review es inspeccionar el resultado del Sprint y determinar futuras adaptaciones del Product Backlog [21].

- **Sprint Retrospective (Retrospectiva de Sprint)**

Se realiza después del Sprint Review, su propósito es planificar formas de aumentar la calidad y la eficacia del equipo durante el desarrollo del producto. El equipo Scrum identifica los cambios más útiles que puede realizar para mejorar su efectividad. Las mejoras más importantes se abordan lo antes posible [21].

2.4.3. Artefactos de Scrum

Los artefactos de Scrum representan trabajo o valor. Están diseñados para maximizar la transparencia de la información clave entre todos los integrantes del equipo. Su objetivo es que todos los que los trabajan en el proyecto tengan la misma información para su desarrollo [21].

- **Sprint Goal (Objetivo del Sprint)**

El Sprint Goal es el único objetivo del Sprint. Se crea durante el evento Sprint Planning y luego se agrega al Sprint Backlog. Mientras el equipo trabaja durante el Sprint tienen en mente este objetivo. Si el resultado de su trabajo resulta ser diferente de lo que esperaban colaboran con el Product Owner para negociar el alcance del Sprint Backlog dentro del Sprint sin afectar el Sprint Goal [21].

- **Product Backlog (Pendientes del Producto)**

Cuando un grupo planea hacer la transición a Scrum, antes de que pueda comenzar el primer Sprint, necesitan un Product Backlog. Es una lista priorizada de características centradas en las necesidades del cliente. El Product Backlog existe y evoluciona a lo largo de la vida útil del producto. En cualquier momento, el Product Backlog es la vista única y definitiva de "todo lo que el equipo debe hacer en algún momento, en orden de prioridad" [22]. Solo existe un Product Backlog para un producto.

- **Sprint Backlog (Lista de Pendientes del Sprint)**

El Sprint Backlog se compone del Sprint Goal (objetivo general), el conjunto de elementos del Product Backlog seleccionados para el Sprint (objetivos específicos), así como un plan para entregar el Incremento (cómo se llevarán a cabo). Es una imagen visible del trabajo que el equipo planea realizar durante el Sprint para lograr el Sprint Goal. En consecuencia, el Sprint Backlog se actualiza a lo largo del Sprint a medida que se aprende más [21]. Debe tener suficientes detalles para que puedan inspeccionar su progreso en el Daily Scrum.

- **Incremento (Increment)**

Un incremento representa el cumplimiento de una meta concreta del Product Backlog. Cada Incremento se suma a todos los incrementos anteriores y se verifica minuciosamente. Para proporcionar valor, el incremento debe ser utilizable. Se pueden crear múltiples incrementos dentro de un Sprint. La Definición de "terminado" (o "done" en inglés) es una descripción formal del estado del Incremento cuando cumple con las medidas de calidad requeridas para el producto.

En el momento en que un elemento del Product Backlog cumple con la definición de “terminado”, nace un Incremento [21].

2.5. Herramientas utilizadas

A continuación, una explicación corta de las herramientas utilizadas a lo largo del desarrollo de este proyecto.

Nombre	Descripción
Unity	Es un motor gráfico que permite crear juegos de video, aplicaciones y experiencias inmersivas e interactivas, en 2D y 3D [13]. Su modo de compilación WebGL ofrece desarrollo de aplicaciones web [14]. Fue utilizado para la creación de todas las interfaces de la aplicación web.
C#	Es un lenguaje de programación moderno, orientado a objetos y con seguridad de tipos. Permite a los desarrolladores crear aplicaciones seguras y sólidas que se ejecutan en “.NET”. Tiene sus raíces en la familia de lenguajes C y es familiar para los programadores de C, C++, Java y JavaScript [23]. Es el lenguaje de programación con el que se programa en Unity, se usó para crear todos los algoritmos de la aplicación web.
Microsoft Visual Studio 2019	Es un entorno de desarrollo integrado (IDE) de Microsoft. Se utiliza para desarrollar programas informáticos, así como sitios web, aplicaciones web, servicios web y aplicaciones móviles [24]. Es el IDE por defecto para C# en Unity, contiene varias integraciones con el motor gráfico lo que lo hacen ideal para el desarrollo de nuestra aplicación.

Git	Git es un sistema de control de versiones distribuido de código abierto y gratuito diseñado para manejar desde proyectos pequeños a muy grandes, con velocidad y eficiencia [25]. Fue el programa ideal para el control de versiones en nuestra aplicación.
GitHub	Es un proveedor de alojamiento de Internet para el control de versiones distribuidas y la funcionalidad de administración de código fuente de Git, además de sus propias características [26]. Nos provee de almacenamiento gratuito para nuestro repositorio Git y la aplicación web terminada.
GitHub Desktop	Es una aplicación que permite interactuar con GitHub usando una GUI en lugar de la línea de comandos o un navegador web [27]. Fue usado para facilitar el manejo de Git en Windows sin línea de comandos.
Paint.net	Es un software de edición de imágenes y fotografías para PC que ejecutan Windows. Cuenta con una interfaz de usuario intuitiva e innovadora con soporte para capas, deshacer ilimitado, efectos especiales y una amplia variedad de herramientas útiles y poderosas [28]. Fue utilizado para crear y modificar todo tipo de gráficos para la aplicación web.

Tabla 1 - Herramientas utilizadas

2.6. Ejercicios en forma de juegos de video

En la Sección de “Ejercicios de Diseño” de la aplicación desarrollada, se usaron videojuegos para explicar, de una manera más interactiva e interesante para los estudiantes, cómo funcionan los diagramas de clases UML.

Nombre	Descripción
Sokoban	Sokoban es un juego de rompecabezas clásico diseñado por Hiroyuki Imabayashi en 1982. Cada nivel representa un almacén, donde cajas parecen estar colocadas al azar. El jugador empuja las cajas por la habitación para que, al final, todas las cajas estén en los campos marcados [29].
Asteroids	Asteroids es un juego de arcade monocromo basado en vectores, diseñado por Lyle Rains y Ed Logg en 1979. El jugador controla una nave triangular que puede moverse, disparar y girar hacia la izquierda y hacia la derecha; su objetivo es destruir todos los asteroides que se desplazan aleatoriamente por la pantalla, evitando al mismo tiempo la colisión con ellos [30].
Space Invaders	Space Invaders es un juego de arcade de disparos de 1978 desarrollado por Tomohiro Nishikado. El objetivo es derrotar oleada tras oleada de alienígenas descendentes con un tanque que dispara láseres que se mueve horizontalmente para ganar tantos puntos como sea posible [31].

Tabla 2 - Juegos de video utilizados

CAPÍTULO 3 – ESTADO DEL ARTE

La enseñanza y el aprendizaje inicial de programación orientada a objetos se acompaña de muchos problemas, como el diseño de los programas, la complejidad de las características del lenguaje de programación y el frágil conocimiento de los principiantes [1]. La programación es una actividad compleja que involucra modelos mentales que los estudiantes luchan por aprender durante los cursos de introducción a la programación [32]. En particular, la programación orientada a objetos a menudo se considera problemática, dado que sus conceptos están estrechamente relacionados entre sí y no se pueden enseñar y aprender fácilmente de forma aislada [32]. Uno de los problemas más importantes con el que se enfrentan los estudiantes es que la programación se enseña tradicionalmente primero de manera estructurada para luego pasar a un enfoque orientada objetos y, además, la creación de software se basa en texto, lo que no resulta familiar ni atractivo para los estudiantes [33]. La mayoría de los alumnos de niveles iniciales en carreras relacionadas a la informática tienen más experiencia en el uso del mouse y entornos gráficos (interfaces gestuales) que en el uso de la línea de comandos (interfaces textuales) [33].

Debido a su complejidad, los cursos de programación tienen fama de tener promedios bajos, con tasas de falla que varían entre el 30% y el 50% en todo el mundo [34]. Además, los estudiantes que no logran comprender los conceptos fundamentales de programación en los primeros cursos introductorios a menudo no pueden recuperarse y ponerse al día, y es más probable que terminar abandonando los programas de enseñanza de informática [35, 36]. Según la literatura, las dificultades más comunes que suelen presentar estudiantes de programación están: *"el desinterés, la comprensión deficiente de los conceptos de la POO, el escaso entendimiento de los problemas planteados, la poca identificación de las estructuras a utilizar en el planteamiento de una solución eficiente y la escasa memorización o asimilación de la sintaxis del lenguaje de programación empleado"* [10]. Estos problemas pueden ser mucho más graves cuando los estudiantes se inician en la programación de la manera tradicional, que primero usa un enfoque estructurado para luego pasar al enfoque orientado a

objetos, y que se basan en lenguajes de programación convencionales con entornos de programación profesionales [12]. Si los estudiantes no logran superar sus dificultades y el resultado puede ser su frustración y abandono de clases [12].

En el diseño de software con enfoque orientado a objetos, la abstracción juega un papel importante como herramienta para manejar la complejidad, siendo referida como un concepto básico, junto con la herencia, encapsulación y polimorfismo [3, 6, 37, 38]. La abstracción y la capacidad de abstraer están entre los fundamentos clave de la informática y la ingeniería de software [3, 6, 37, 39], este concepto está estrechamente relacionado con la formación algorítmica, la automatización de notaciones y modelos, la descomposición de tareas complejas y la generalización a través de la definición de patrones [3, 18, 40].

El concepto el diseño orientado a objetos es bastante natural, ya que en la vida real pensamos en términos de objetos, que tienen determinadas propiedades y comportamientos [33]. Sin embargo, a la hora de redactar programas, aquellos estudiantes iniciados en programación estructurada tienden a adoptar una visión tradicional consistente en instrucciones que se ejecutan y estructuras de control que definen el flujo de control de los programas [41]. En este sentido, los programadores esperan que los programas de computadora tengan un principio y un final, y por lo tanto uno comprende un programa solo leyéndolo. Sin embargo, para diseñar un programa orientado a objetos se necesita comprender qué son los objetos, abstraer sus conceptos de manera correcta en el contexto del programa, y comprender cómo se transfieren los mensajes entre dichos objetos para realizar tareas [42].

Actualmente, la computadora es parte de la vida tanto de los profesores como de los estudiantes, y la mayoría de personas tiene experiencia en el uso de juegos de computadora, buscadores, redes sociales, aplicaciones web, procesadores de texto y mensajería instantánea [43]. Específicamente las aplicaciones web se han convertido en herramientas sofisticadas de cómputo que no sólo proporcionan funciones aisladas al usuario final, sino que también se han integrado con bases de datos corporativas y aplicaciones de negocios [2]. Powell sugiere que los sistemas y aplicaciones basados en web “involucran una mezcla entre las publicaciones impresas y el desarrollo de software, entre la mercadotecnia y la

computación, entre las comunicaciones internas y las relaciones exteriores, y entre el arte y la tecnología" [44]. A continuación se presentan varios de los aspectos beneficiosos de las aplicaciones web [2]:

- *Concurrencia*: Pueden acceder un gran número de usuarios a la vez.
- *Rendimiento*: Velocidad alta para entrar a la aplicación, para el procesamiento por parte del servidor y despliegue del lado del cliente.
- *Disponibilidad*: La propiedad de las aplicaciones web para estar listas y disponibles para llevar a cabo su tarea cuando las necesite. Aunque no es razonable esperar una disponibilidad de 100%, sí se puede esperar que la aplicación web esté disponible la mayoría del tiempo si es alojada en servidores confiables.
- *Evolución Continua*: A diferencia del software de aplicación convencional que evoluciona a lo largo de una serie de etapas planeadas y separadas cronológicamente, las aplicaciones web evolucionan en forma más rápida.
- *Estética*: Parte innegable del atractivo de una aplicación web es su apariencia y percepción.
- *Rendimiento*: Permiten un tiempo de respuesta rápido de la aplicación para el usuario.

En un estudio realizado por Stelios Xinogalos, un profesor e investigador griego, se muestra que una herramienta de enseñanza digital puede ser usada para: "*enseñar conceptos de programación sin importar la edad del estudiante, enseñar conceptos básicos de programación y principios de las técnicas de programación orientada a objetos, ayudar a los estudiantes a superar las dificultades enfrentadas en una enseñanza convencional, enseñar técnicas de resolución de problemas"* [11], entre otras cosas.

Adicionalmente, en la literatura se demuestra que el uso de herramientas digitales ayuda a los estudiantes de informática a mejorar su resultado de aprendizaje, mejora la capacidad de los estudiantes para abstraer, aumenta su nivel de motivación y acentúa sus habilidades para resolver problemas [5]. En general, los estudios de investigación discutidos anteriormente prueban la efectividad del uso de herramientas digitales para enseñar asignaturas de informática [11, 12].

En la literatura se han encontrado varios ejemplos de herramientas digitales para la mejora de la enseñanza de diferentes aspectos del ciclo de vida del software. Por ejemplo, juegos de video serios para enseñar principios básicos de programación orientada a objetos [33, 45-47], juegos de video serios para enseñar análisis y diseño orientados a objetos [48], uso de gamificación en materias de carreras universitarias para alentar a los estudiantes a obtener mejores calificaciones [49, 50], uso de entornos de aprendizaje interactivos para la enseñanza de técnicas de programación imperativas y orientadas a objetos [10-12], como también herramientas que permiten experimentación y diseño interactivo de diagramas de clases y objetos [32], entre muchas otras. Si bien encontramos varias herramientas enfocadas en la enseñanza de conceptos de programación orientada a objetos, también notamos la falta de dichas herramientas para la enseñanza específica del diseño de software orientado a objetos que, si bien es una parte esencial del ciclo de vida de desarrollo de software, ha sido ignorada en estos ámbitos.

Por otro lado, hoy en día tenemos herramientas para desarrollo de aplicaciones interactivas muy avanzadas, como es la plataforma Unity la cual permite crear juegos de video, aplicaciones y experiencias inmersivas e interactivas, en 2D y 3D, a artistas, diseñadores y desarrolladores [13]. Unity funciona con el lenguaje de programación C# y permite con la misma base de código generar aplicaciones nativas para Windows, MacOS, Linux, Android, iOS, WebGL, consolas de videojuegos y gafas de realidad virtual [14]. Por su versatilidad, facilidad de uso, y por la importancia de crear una aplicación web interactiva y atractiva, Unity será la herramienta utilizada para la creación de la aplicación web propuesta.

En un trabajo relacionado realizado por los profesores Stephen Frezza y Wayne Andersen [51] encontramos el uso de ejercicios interactivos para la enseñanza de modelado estructural UML para estudiantes universitarios. En este trabajo se desarrolló una aplicación con el programa FLASH en la cual se muestra un Diagrama de Clases con el que se describe el diseño del juego de mesa “Chitty-Chitty Bang-Bang”, con explicaciones tanto visuales como auditivas sobre cada clase del diagrama y sus relaciones. Según un análisis cualitativo realizado por los mismos profesores a sus estudiantes, se descubrió que los encuestados disfrutaron

el uso de tecnología para el aprendizaje de la materia y la capacidad de interactuar con los modelos de manera visual. Por otro lado, mediante un análisis cuantitativo se descubrió que existió una ligera mejora de notas en los estudiantes con calificaciones en promedio medias y bajas, pero no hubo diferencias para estudiantes de promedios altos.

En dos casos de estudio realizados por Paul Gestwicki [52, 53] se investigó sobre las aplicaciones de los videojuegos a la educación en informática e ingeniería de software. En ambos artículos se describe los resultados del uso de juegos de computadora para enseñar patrones de diseño para la programación orientada a objetos. Se muestra que los patrones de diseño a menudo se malinterpretan, pero a través de buenos ejemplos de su uso, y mediante experiencias interactivas, se puede motivar a los estudiantes a trabajar para comprenderlos.

Los estudios analizados anteriormente demuestran el interés de los alumnos por el uso de herramientas interactivas para el aprendizaje de conceptos de ingeniería de software, como también las dificultades que tienen para aprender patrones de diseño, abstracción y diseño estructural UML.

Nuestro proyecto difiere del trabajo de Frezza, Andersen y Gestwicki porque nos enfocamos en la enseñanza de la abstracción, como también la explicación paso a paso del desarrollo del diagrama de clases tomando como ejemplos distintos juegos de video, con explicaciones interactivas enfocadas en el aspecto visual más que en el auditivo.

CAPÍTULO 4 – DESARROLLO DE LA APLICACIÓN

4.1. FASE EXPLORATORIA

4.1.1. Alcance del proyecto

Este proyecto crea una aplicación web, enfocada en estudiantes de materias de pregrado universitario relacionadas con la ingeniería de software, que permita mejorar su aprendizaje de temas relacionados con el diseño de software, entrenar su habilidad de abstraer al desarrollar software, y ayudar con problemas relacionados a los conceptos propios del enfoque orientado a objetos. Esta herramienta es un complemento de los conceptos enseñados por los profesores de estas materias en su instrucción formal y de ninguna manera puede ser tomado como un reemplazo.

4.1.2. Stakeholders

La siguiente tabla define a las partes interesadas en esta aplicación (también denominados stakeholders):

Partes interesadas (Stakeholders)		
ID	**Stakeholder**	**Descripción**
S1	**Estudiante (Usuario)**	Será el usuario final de la aplicación a ser desarrollada. Necesita una aplicación web que le permita complementar su educación en conceptos relacionados al diseño de software y abstracción.
S2	**Profesor**	Sabe la materia que va a ser enseñada y guiará al Desarrollador durante el proyecto para que el software creado cumpla con las necesidades del Estudiante.
S3	**Desarrollador**	A cargo del diseño y desarrollo de la aplicación web. Desarrollará el programa que cumpla con las necesidades del Estudiante y del Profesor.

Tabla 3 - Partes interesadas (Stakeholders)

4.1.3. Planificación del proyecto

En el plan de trabajo de titulación se definió que el proyecto tendría una duración de 440 horas a lo largo de 6 meses de trabajo. Estas horas fueron divididas de la siguiente manera:

- 60 horas para la definición de requerimientos y diseño de la aplicación.
- 320 horas para el desarrollo de la aplicación web.
- 15 horas para subir la aplicación a un dominio en el internet.
- 45 horas para realizar pruebas de funcionalidad y usabilidad.

Actividad	Semana	Mes 1				Mes 2				Mes 3				Mes 4				Mes 5				Mes 6				
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	
Definición de requerimientos funcionales y no funcionales de la aplicación																										
Diseño de la arquitectura de la aplicación																										
Definición del backlog del producto																										
Planficación del sprint																										
Scrum diario																										
Revisión del sprint																										
Retrospectiva del sprint																										Total horas de trabajo
Implementación de la aplicación web en un dominio específico en el internet.																										
Ejecución de pruebas de funcionalidad y de usabilidad de la aplicación.																										
	Horas de trabajo	15	15	15	15	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	15	15	15	15	440

Figura 5 - Planificación del proyecto

Durante el proceso de desarrollo del software se realizaron reuniones semanales mediante la plataforma ZOOM con la doctora Pamela Catherine Flores Ph.D., directora del Proyecto Semilla PIS-19-11, ya que la aplicación desarrollada forma parte de dicho proyecto de investigación y Pamela es la dueña del producto. Estas reuniones sirvieron para obtener los requerimientos necesarios de la aplicación y de igual manera, verificar que todos los criterios de aceptación para cada requerimiento sean cumplidos a cabalidad.

Para cumplir con el tiempo establecido en el plan de trabajo de titulación las 320 horas de desarrollo fueron divididas en un total de 4 meses. Por lo que, se definió que se tendría 8 sprints, con una duración de dos semanas cada uno. Sin embargo, si bien se mantuvieron los tiempos de desarrollo, por obligaciones previas de la dueña del producto y del equipo de desarrollo, la duración de la implementación se vio alargada a 6 meses por pausas que se tuvo que tomar a lo largo del desarrollo del proyecto.

Siguiendo la metodología Scrum, explicada anteriormente, como primer paso en su implementación se definió los roles de los involucrados en el proyecto, esto se muestra en la siguiente tabla.

ROLES DE SCRUM	
Scrum Master	Pamela Flores
Product Owner	Pamela Flores
Scrum Team	Carlos Rodas

Tabla 4 - Roles de Scrum

4.2. FASE DE INICIALIZACIÓN

4.2.1. Herramientas

Para el manejo de versiones el equipo utilizó Git [25], junto con el programa GitHub Desktop [27], el cual permitió el manejo del repositorio en GitHub de una manera visual, sin tener que depender de la línea de comandos. El link al repositorio en GitHub se encuentra en el Anexo 1.

Por otro lado, para el desarrollo de la aplicación se usó Unity [13, 14], un motor gráfico de videojuegos multiplataforma, el cual permite también desarrollo de aplicaciones web.

Por último, para la programación se usó el entorno de desarrollo integrado Visual Studio [24], junto con el lenguaje de programación C# [23], ya que tienen integración directa con Unity.

4.2.2. Configuración del proyecto

- **Historias de Usuario**

Para el desarrollo de este proyecto se realizaron reuniones con la dueña del producto (Product Owner) donde se manifestó la necesidad de crear una aplicación web para la enseñanza de diseño de software orientado a objetos, como también conceptos básicos de abstracción. Se explicó que el software debe estar enfocado para estudiantes universitarios de pregrado, que cursan clases relacionadas a la informática. La aplicación deberá enseñar conceptos básicos de diseño de software

y abstracción, no un material muy complicado o avanzado. Adicionalmente, se solicitó que la herramienta sea bastante interactiva, llamativa, que capture la atención de los estudiantes y permita que aprendan la materia de manera fácil.

Para satisfacer estas necesidades se definió un conjunto de historias de usuario, las cuales se detallan en la siguiente tabla.

Código	Nombre	Historia de usuario
US01	Sección de teoría	Como profesor/a de diseño de software necesito que mis alumnos aprendan conceptos teóricos sobre diseño de software orientado a objetos, abstracción y el diagrama de clases UML.
US02	Sección de pruebas	Como profesor/a de diseño de software necesito que mis alumnos pongan a prueba sus conocimientos de diseño de software orientado a objetos, abstracción y el diagrama de clases UML mediante pruebas interactivas.
US03	Sección de ejercicios	Como profesor/a de diseño de software necesito que mis alumnos realicen ejercicios interactivos con los que puedan crear diagramas de clase UML por partes, diseñando videojuegos clásicos y con niveles donde se va aumentando la dificultad de los diseños progresivamente.
US04	Sección de juegos	Como profesor/a de diseño de software necesito que mis alumnos interactúen con los videojuegos que diseñan en la sección de ejercicios, mientras los diseñan, y después de terminar de diseñarlos.
US05	Sección de información	Como profesor/a de diseño de software necesito que mis alumnos tengan acceso a las referencias bibliográficas que se usaron en la sección teórica de la aplicación y que pueda revisar su progreso en la aplicación de forma visual.
US06	Menús de la aplicación	Como profesor/a de diseño de software necesito que los menús de la aplicación estén organizados

		de manera intuitiva, con elementos de sonido que mejoren la interactividad.
US07	Aplicación web	Como profesor/a de diseño de software necesito que mis alumnos tengan acceso inmediato y gratuito, a través del internet, a la aplicación web.

Tabla 5 - Historias de Usuario

- **Product Backlog**

Tras definir las historias de usuario se procedió a separarlas en tareas más pequeñas y, a partir de ellas, se obtuvo el Product Backlog (Lista de pendientes del producto). El Product Backlog está conformada por las tareas a realizarse, junto con una estimación de la complejidad para completar cada tarea y su prioridad para cumplir los objetivos del negocio. Para la estimación de prioridad se usaron los valores mostrados en la siguiente tabla.

VALORACIÓN DE LA PRIORIDAD	
Valoración	**Prioridad del negocio**
1	Baja
2	Media
3	Alta
4	Muy Alta

Tabla 6 - Valoración de prioridad

Para definir la complejidad de cada historia de usuario se utilizó “Planning Poker”, también conocido como “Scrum Poker” o “Pointing Poker”, es una técnica utilizada para calcular el esfuerzo de las tareas de gestión de proyectos de desarrollo de software. Esta herramienta indica que para medir la cantidad de puntos de la historia para las tareas relevantes el equipo de desarrollo necesita calcular el esfuerzo que tomaría completar cada tarea en una escala exponencial, conociendo, por ejemplo, que si una tarea es el doble de compleja tomaría más del doble del tiempo en ser completada [54, 55]. Primeramente, se escogió la tarea que tomaría el menor esfuerzo para ser usada como guía base, en este caso T18 (ver Tabla 8). La complejidad asignada a las demás tareas es una representación del tamaño de cada historia de usuario en comparación con la guía base. La valoración de la

complejidad de las historias de usuario, que fueron definidas por el equipo de desarrollo, está descrita en la tabla a continuación.

VALORACIÓN DE LA COMPLEJIDAD	
Valoración	**Complejidad**
1	Muy Baja
2	Baja
3	Media
5	Alta
8	Muy alta

Tabla 7 - Valoración de la complejidad

Para priorizar las tareas definidas se hizo énfasis en los comentarios de la "dueña del producto" o "Product Owner" y se analizó las tareas de mayor complejidad junto con las que tienen mayor prioridad para el negocio, así se fue definiendo el Product Backlog y el orden en el que se desarrollaría cada parte de la aplicación web. Dicho "Product Backlog" o "Lista de pendientes del producto" se puede observar en la siguiente tabla.

PRODUCT BACKLOG				
	Código	**Nombre**	**Complejidad**	**Prioridad**
US01	**T01**	Crear una sección teórica sobre Ciclo de vida del desarrollo del software	5	2
	T02	Crear una sección teórica sobre Abstracción	5	2
	T03	Crear una sección teórica sobre UML	5	3
	T04	Crear una sección teórica sobre el Diagrama de Clases UML	5	3
US02	**T05**	Crear una prueba para el Ciclo de vida del desarrollo del software	2	1

	T06	Crear una prueba para Abstracción	2	1
	T07	Crear una prueba para UML	2	2
	T08	Crear una prueba para el Diagrama de Clases UML	2	2
US03	**T09**	Desarrollo del sistema de guardado	3	2
	T10	Desarrollo del sistema de niveles	3	2
	T11	Desarrollar el ejercicio del diagrama de clases “Figuras”	8	3
	T12	Desarrollar el ejercicio del diagrama de clases “Sokoban”	8	4
	T13	Desarrollar el ejercicio del diagrama de clases “Asteroids”	8	4
	T14	Desarrollar el ejercicio del diagrama de clases “Space Invaders”	8	4
US04	**T15**	Desarrollar el videojuego “Sokoban”	5	2
	T16	Desarrollar el videojuego “Asteroids”	5	2
	T17	Desarrollar el videojuego “Space Invaders”	5	2
US05	**T18**	Crear una sección de información	1	3
	T19	Popular la sección de información con las referencias bibliográficas usadas en la sección teórica, las notas conseguidas en las pruebas y datos del progreso en la sección de ejercicios	2	3

US06	**T20**	Desarrollar un sistema de sonido para los menús de la aplicación	3	2
	T21	Crear los menús que conectan a todas las secciones de la aplicación	1	3
US07	**T22**	Subir la aplicación al internet para que sea de acceso público	2	4

Tabla 8 - Product Backlog

- **Planificación de Sprints**

Para el desarrollo del proyecto se planificaron un total de 7 sprints para cumplir con el Product Backlog, más un sprint 0 para la preparación del ambiente de desarrollo y repositorios Git. Todos los sprints tuvieron una duración de 2 semanas. Se proyectó que en caso de que se agreguen más tareas durante el transcurso de un sprint estas serán añadidas a uno de los próximos sprints para no interferir con el que está en curso, de acuerdo con la complejidad y prioridad del nuevo requerimiento. Esta planificación se puede observar en la tabla a continuación.

Sprint 0	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6	Sprint 7
Preparación del entorno de desarrollo	T09	T15	T16	T17	T01	T03	T18
	T10	T12	T13	T14	T05	T07	T19
	T11	-	-	-	T02	T04	T20
	-	-	-	-	T06	T08	T21
	-	-	-	-	-	-	T22

Tabla 9 - Planificación de Sprints

4.2.3. Iteración cero (Sprint 0)

- **Sprint 0 Planning**

En este Sprint se buscó preparar el entorno de desarrollo que se usará a lo largo de todo el proyecto, primero definiendo la arquitectura de este, definiendo los requisitos no funcionales que se necesitan para satisfacer las expectativas de los usuarios finales, y alistando las herramientas que se utilizarán para el desarrollo de la aplicación.

- **Ejecución del Sprint 0**
    - **Arquitectura del proyecto**

Se definió que la aplicación web tendrá una arquitectura de alto nivel Cliente-Servidor de dos niveles [15], ya que el sistema se implementará en un solo servidor lógico más un número indefinido de clientes que usarán dicho servidor. El proceso de la aplicación y el sistema de guardado se realizará localmente en el cliente pues la aplicación no necesita de servicios de cuenta, autenticación, ni una base de datos. El servidor será encargado solamente de permitir al cliente descargar la aplicación en su navegador web. Esta arquitectura está detallada en la Figura 6.

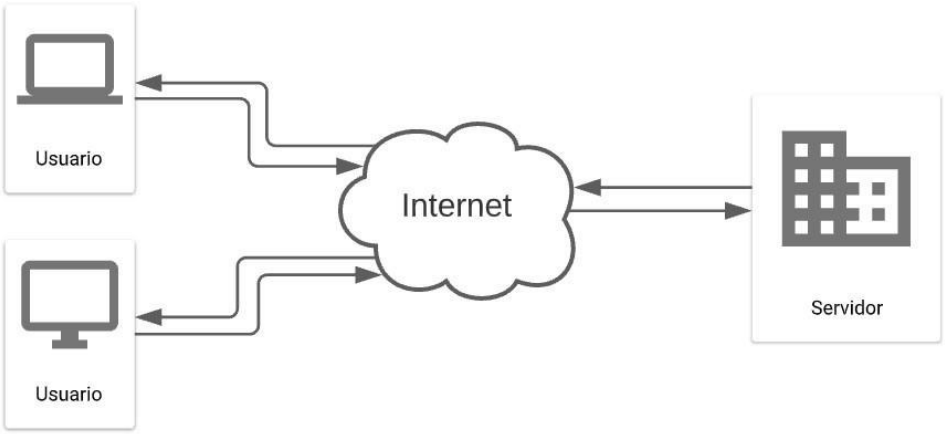


Figura 6 - Arquitectura Cliente-Servidor

En un nivel más bajo el patrón de arquitectura utilizado será Entidad-Componente-Sistema (ECS), es un patrón de arquitectura de software que se utiliza principalmente en el desarrollo de videojuegos para el almacenamiento de la lógica de objetos del mundo del juego [56]. El patrón ECS sigue el principio de composición sobre herencia, lo que significa que cada entidad se define por los componentes que están asociados con ella [56]. El motor de juegos Unity usa este patrón [57].

La arquitectura ECS separa la identidad (entidades), los datos (componentes) y el comportamiento (sistemas). La arquitectura se centra en los datos. Los sistemas leen flujos de datos de componentes y luego transforman los datos de un estado de entrada a un estado de salida, que luego indexan las entidades [57]. La Figura 7 muestra un ejemplo de cómo estas tres partes básicas funcionan juntas:

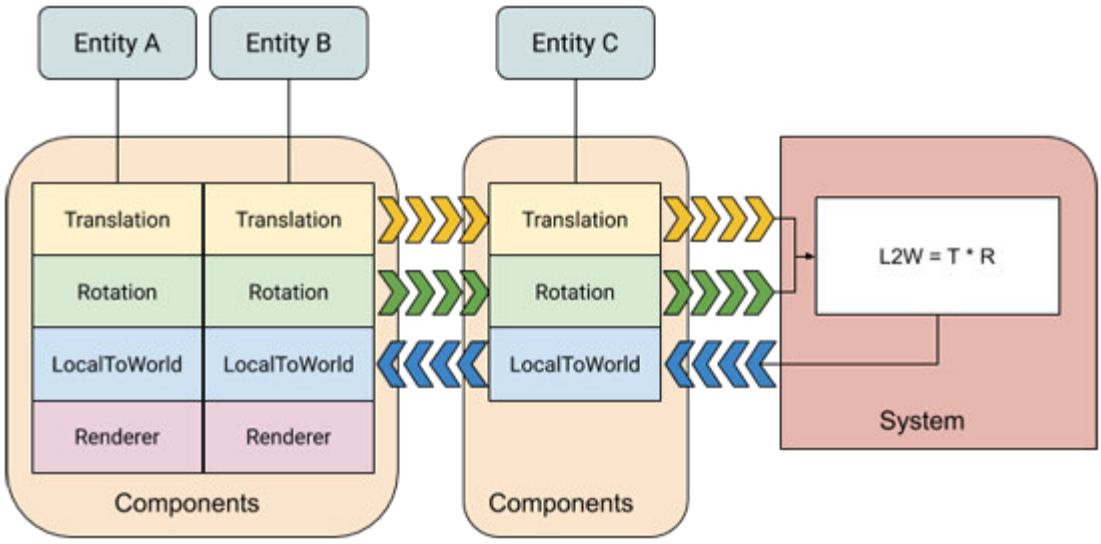


Figura 7 - Ejemplo ECS [57]

En la Figura 7 existe el sistema lee los componentes de “Traducción” y “Rotación”, los multiplica y luego actualiza los componentes “LocalToWorld” correspondientes (L2W = T*R). El hecho de que las entidades A y B tengan un componente “Renderer” y la entidad C no, no afecta al sistema, porque al sistema no le importan los componentes “Renderer”. Se puede configurar un sistema que requiera un componente “Renderer”, en cuyo caso, el sistema ignora los componentes de la entidad C; o, alternativamente, se podría configurar un sistema para excluir entidades con componentes “Renderer”, que ignoraría los componentes de las entidades A y B [57].

- **Definición de requisitos no funcionales**

Los requisitos no funcionales se definieron durante diferentes reuniones con el Product Owner. Estos requisitos fueron mayormente enfocados en la usabilidad de la aplicación para los usuarios finales, durante el desarrollo del proyecto se fueron afinando a las necesidades y gustos del cliente. A continuación, una lista resumida de estos requerimientos:

- Uso de colores llamativos en las diferentes pantallas de la aplicación
- Las explicaciones de las secciones teóricas necesitan ser claras y concisas
- Los ejercicios de diseño necesitan ser interactivos
- Las interfaces gráficas de la aplicación deben ser bien formadas y su uso agradable para el usuario

- **Repositorio Git y control de versiones**

Para el control de versiones de este proyecto se creó un repositorio público en GitHub [26] que permite rastrear y gestionar todos los cambios al código a lo largo del proyecto.

El flujo de trabajo utilizado para manejar las ramas de Git fue Gitflow [58]. Se usó dos ramas, la rama “Main” guarda los cambios finales en la aplicación, mientras que la rama “Develop” sirve para el desarrollo de nuevas características de la aplicación, una vez estas características están terminadas se unen a la rama “Main”, como se puede observar en la Figura 8.

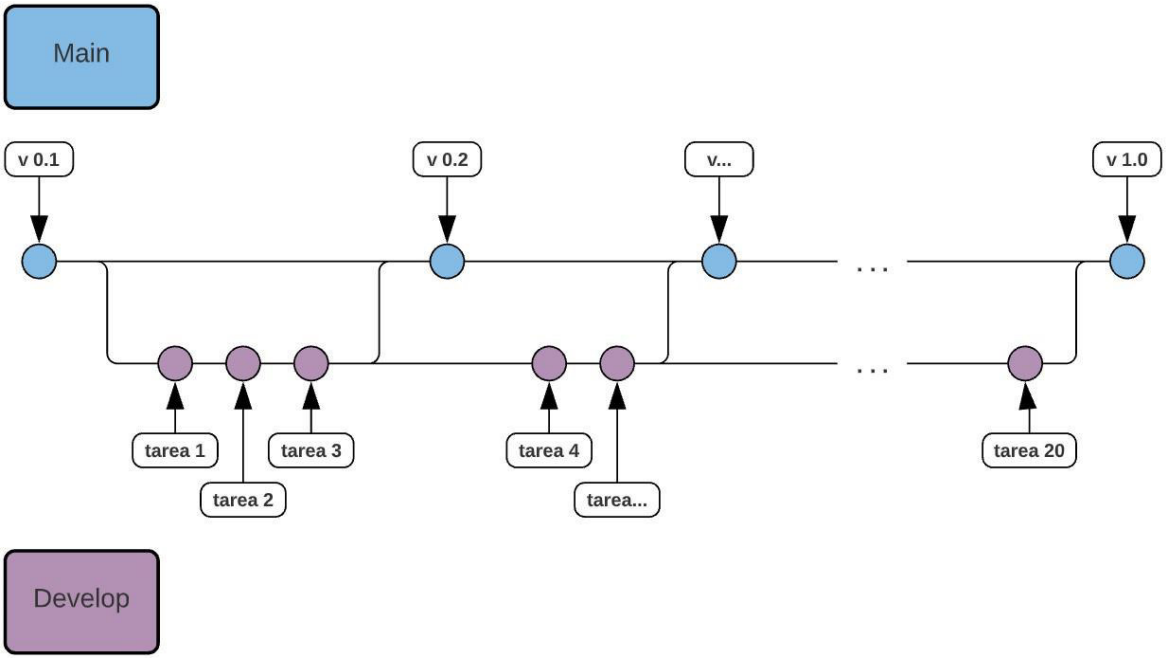


Figura 8 - Flujo de trabajo en Git

- **Sprint 0 Review**

Tras la finalización de este Sprint se logró obtener un ambiente de trabajo en el cual el equipo está preparado para empezar con el desarrollo del proyecto. Las herramientas de trabajo están listas para ser usadas, la arquitectura del proyecto está definida, y el equipo es consciente del flujo de trabajo que tiene que seguir para cumplir con sus objetivos.

- **Sprint 0 Retrospective**

- **¿Qué salió bien?**

Se logró definir la arquitectura de manera correcta, crear el repositorio Git para el sistema del control de versiones y las herramientas de trabajo están listas para ser utilizadas.

- **¿Qué se puede mejorar?**

Hay que familiarizarse con el motor gráfico Unity porque usa paradigmas nunca utilizados anteriormente por el equipo de trabajo.

4.3. FASE DE PRODUCCIÓN Y ESTABILIZACIÓN

A continuación, se explica en detalle lo que se hizo en cada Sprint a lo largo del desarrollo de la aplicación web. El Anexo 5 tiene una explicación visual de todas las pantallas de la aplicación después de su finalización.

4.3.1. Sprint 1

- **Sprint 1 Planning**

En este Sprint se planea realizar las historias de usuario que permiten que la aplicación tenga un sistema de guardado local en la máquina del usuario, como también el sistema para que los ejercicios de diseño tengan niveles, al pasar dichos niveles desbloquearán nuevos ejercicios y juegos de video mientras progresan a través de la aplicación. Finalmente, se planifica crear el primer y más básico ejercicio que funciona como una suerte de tutorial para enseñar al usuario como funcionarán los siguientes ejercicios de diseño de diagramas de clases UML.

- **Sprint 1 Backlog**

Sprint 1 Backlog	
Código	**Nombre**
T09	Desarrollo del sistema de guardado
T10	Desarrollo del sistema de desbloqueo de niveles
T11	Desarrollar el ejercicio del diagrama de clases “Figuras”

Tabla 10 - Sprint 1 Backlog

- **Ejecución del Sprint 1**

Primeramente, se necesita de un sistema que permita guardar la información del juego localmente, por suerte Unity viene con librerías que permiten lograr este objetivo con facilidad, se crearon varios “scripts” (la forma en que Unity llama a los algoritmos escritos en el lenguaje de programación C# que darán la funcionalidad a los componentes en la aplicación [13]) para completarlo.

“GameData” es una clase que indica los tipos de datos que se van a guardar. Los primeros cuatro “float” serán las calificaciones que obtengan en las pruebas de la sección teórica, mientras que los siguientes cuatro “bool” representan si terminó o no los ejercicios de la sección de ejercicios.

```
[System.Serializable]
7 references
public class GameData
{
    public float califCVS;
    public float califAbs;
    public float califUML;
    public float califDdC;

    public bool pasadoFig;
    public bool pasadoSoko;
    public bool pasadoAst;
    public bool pasadoFinal;
```

Figura 9 - “GameData” script

El script “SaveSystem” es el encargado de guardar y cargar los datos.

```
2 references
public static class SaveSystem
{
    1 reference
    public static void SaveGame(Game game)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "/game.save";
        FileStream stream = new FileStream(path, FileMode.Create);

        GameData data = new GameData(game);

        formatter.Serialize(stream, data);
        stream.Close();
    }

    1 reference
    public static GameData LoadGame()
    {
        string path = Application.persistentDataPath + "/game.save";
        if (File.Exists(path))
        {
            BinaryFormatter formatter = new BinaryFormatter();
            FileStream stream = new FileStream(path, FileMode.Open);

            GameData data = formatter.Deserialize(stream) as GameData;
            stream.Close();

            return data;
        }
        else
        {
            Debug.LogError("Save file not found in " + path);
            return null;
        }
    }
}
```

Figura 10 - “SaveSystem” script

La función “SaveGame” de “SaveSystem” guarda los datos localmente al ser llamada, mientras que “LoadGame” se encarga de leer los archivos guardados y usarlos mientras la aplicación está en funcionamiento.

Por último, para el sistema de guardado tenemos el script “Game”, este se encarga de usar los dos scripts anteriores mientras la aplicación web está en marcha. En toda pantalla de la aplicación donde se cargue o guarde información se llamará a esta clase.

```
Unity Script (11 asset references) | 30 references
public class Game : MonoBehaviour
{
    public float califCVS = 0f;
    public float califAbs = 0f;
    public float califUML = 0f;
    public float califDdC = 0f;

    public bool pasadoFig = false;
    public bool pasadoSoko = false;
    public bool pasadoAst = false;
    public bool pasadoFinal = false;

    Unity Message | 0 references
    public void Awake()
    {
        LoadGame();
    }

    4 references
    public void SaveGame()
    {
        SaveSystem.SaveGame(this);
    }

    1 reference
    public void LoadGame()
    {
        GameData data = SaveSystem.LoadGame();

        califCVS = data.califCVS;
        califAbs = data.califAbs;
        califUML = data.califUML;
        califDdC = data.califDdC;

        pasadoFig = data.pasadoFig;
        pasadoSoko = data.pasadoSoko;
        pasadoAst = data.pasadoAst;
        pasadoFinal = data.pasadoFinal;
    }
```

Figura 11 - "Game" script

El siguiente objetivo es crear el sistema de niveles de la sección de ejercicios, para cumplirlo se compuso en Unity la parte visual de dicha sección. Se crearon botones para cada una de los ejercicios como también para acceder a los juegos de video que se diseñarán en futuros ejercicios. Estos botones se muestra en la Figura 12.

Figura 12 - Botones de la sección de ejercicios

Después de crear estos botones se diseñó un sistema de paneles en forma de candados que existirán encima de los botones y bloquearán el acceso a estas secciones hasta cumplir con los ejercicios, como se ve en la Figura 13. Una vez se terminen los ejercicios estos paneles se desactivan.

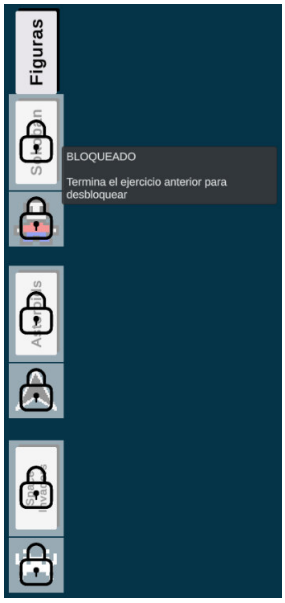


Figura 13 - Botones de la sección de ejercicios bloqueados

Para agregar o quitar los candados se creó el script “ShowLocks”, de la Figura 14, este revisa en el sistema de guardado, creado anteriormente, si pasó o no pasó el nivel anterior y, dependiendo del caso, activa o desactiva los candados.

```
Unity Script (5 asset references) | 0 references
public class ShowLocks : MonoBehaviour
{
    public GameObject lock1;
    public GameObject lock2;
    public GameObject lock3;
    public GameObject lock4;
    public GameObject lock5;
    public GameObject lock6;
    public int nextUnlock;

    Unity Message | 0 references
    private void Start()
    {
        if (FindObjectOfType<Game>().pasadoFig) { lock1.SetActive(false); } else { lock1.SetActive(true); }
        if (FindObjectOfType<Game>().pasadoSoko) { lock2.SetActive(false); } else { lock2.SetActive(true); }
        if (FindObjectOfType<Game>().pasadoSoko) { lock3.SetActive(false); } else { lock3.SetActive(true); }
        if (FindObjectOfType<Game>().pasadoAst) { lock4.SetActive(false); } else { lock4.SetActive(true); }
        if (FindObjectOfType<Game>().pasadoAst) { lock5.SetActive(false); } else { lock5.SetActive(true); }
        if (FindObjectOfType<Game>().pasadoFinal) { lock6.SetActive(false); } else { lock6.SetActive(true); }
    }
```

Figura 14 - "ShowLocks" script

También se agregó un sistema de Tooltips (un mensaje que aparece cuando se coloca el cursor sobre un ícono, imagen, u otro elemento de la interfaz gráfica de usuario [59]). Para conseguir esto se importó una librería externa gratuita de la

tienda de assets de Unity llamada “Simple Tooltip”, el link a esta herramienta está en el Anexo 3.

Por último, para terminar esta sección se desarrolló el ejercicio del diagrama de clases “Figuras”. Este es un ejercicio simple utilizado por la dueña del producto con sus estudiantes de diseño de software para enseñar conceptos básicos sobre el diagrama de clases UML. Este ejercicio consiste en diseñar un programa para dibujar figuras geométricas simples, moverlas y ponerlas una encima de otra. Explica de una manera sencilla lo que es una clase, una instancia, un método, un atributo, como también las relaciones de herencia y de asociación direccional.

El primer paso para desarrollar los ejercicios fue diseñar el diagrama de clases al que deben llegar los usuarios de la aplicación después de completar el ejercicio. Esto es porque son tutoriales guiados donde se enseñan conceptos específicos del diagrama de clases UML. Este diagrama de clases fue aprobado por la dueña del producto durante las reuniones semanas que se llevaron a cabo. El diagrama de ejercicio “Figuras” se puede observar en la Figura 15.

Después de crear el diagrama se procedió a separar a cada parte del mismo en distintos sprites (gráficos de computadora que se puede mover en la pantalla y manipular de manera interactiva [13]) que serán usados en distintas partes del ejercicio.

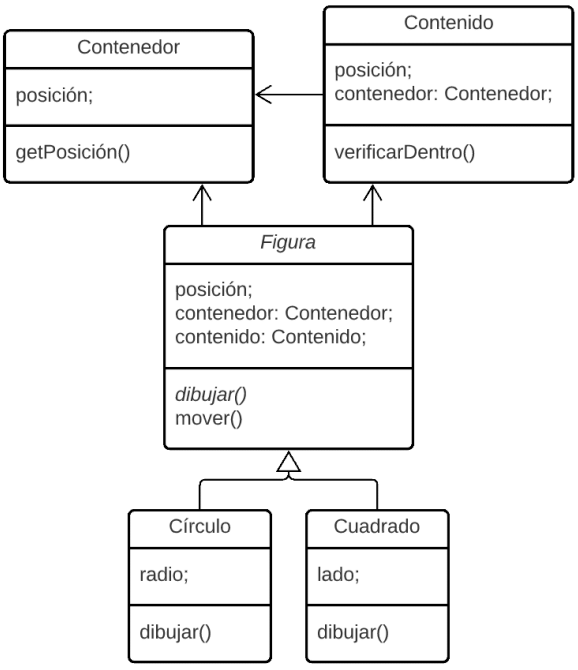


Figura 15 - Diagrama de clases de ejercicio "Figuras"

Después de algunos prototipos de como funcionarían los ejercicios se determinó que la mejor forma de mostrarlos sería con el uso de paneles. Estos paneles contendrían una sección con información importante para el usuario como también una sección interactiva donde el usuario realiza cierta acción. Al terminar la acción el usuario podría seguir hacia el siguiente panel.

Para poder cambiar interactivamente entre paneles se diseñó el script "PanelManager", como se ve en la Figura 16.

```
Unity Script (37 asset references) | 0 references
public class PanelManager : MonoBehaviour
{

    0 references
    public void PanelChanger(GameObject panelToActivate)
    {
        GameObject[] panels = GameObject.FindGameObjectsWithTag("Panel");
        foreach (var panel in panels)
        {
            panel.gameObject.SetActive(false);
        }
        FindObjectOfType<SoundManagerScript>().Play("Step");
        panelToActivate.gameObject.SetActive(true);
    }
```

Figura 16 - "PanelManager" script

"PanelManager" es llamado cuando se aplastan los botones para cambiar de panel, básicamente activa un panel y desactiva el resto.

En este ejercicio también se necesita arrastrar y soltar elementos, por eso también se creó el script "DragDrop" que permite hacer justamente eso. Lo mostramos en la Figura 17. Este script también hace que el sprite que está siendo movido se vuelva transparente para ver donde va a ser soltado.

```
Unity Script (247 asset references) | 0 references
public class DragDrop : MonoBehaviour, IPointerDownHandler, IBeginDragHandler, IEndDragHandler, IDragHandler
{
    [SerializeField] private Canvas canvas;
    [TextArea] public string infoL;

    private RectTransform rectTransform;
    private CanvasGroup canvasGroup;

    Unity Message | 0 references
    private void Awake()
    {
        rectTransform = GetComponent<RectTransform>();
        canvasGroup = GetComponent<CanvasGroup>();
    }

    0 references
    public void OnBeginDrag(PointerEventData eventData)
    {
        canvasGroup.alpha = 0.6f;
        canvasGroup.blocksRaycasts = false;
    }

    0 references
    public void OnDrag(PointerEventData eventData)
    {
        rectTransform.anchoredPosition += eventData.delta / canvas.scaleFactor;

        if (GetComponent<SimpleTooltip>() != null)
        {
            GetComponent<SimpleTooltip>().infoLeft = infoL;
        }
    }

    0 references
    public void OnEndDrag(PointerEventData eventData)
    {
        canvasGroup.alpha = 1.0f;
        canvasGroup.blocksRaycasts = true;

        FindObjectOfType<SoundManagerScript>().Play("Step");
    }
```

Figura 17 - "DragDrop" script

También se usó el sistema “Simple Tooltip” en los diagramas de clase que se ven a lo largo del ejercicio para que el usuario tenga información inmediata de lo que es cada parte del diagrama de clases, como se ve en la Figura 18.

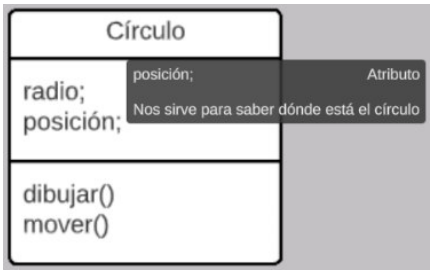


Figura 18 - Ejemplo de tooltip

El ejercicio Figuras terminó constando de catorce paneles interactivos que enseñan los primeros conceptos del diagrama de clases UML, como ejemplo se tomó una captura de pantalla del panel 8, mostrado en la Figura 19.

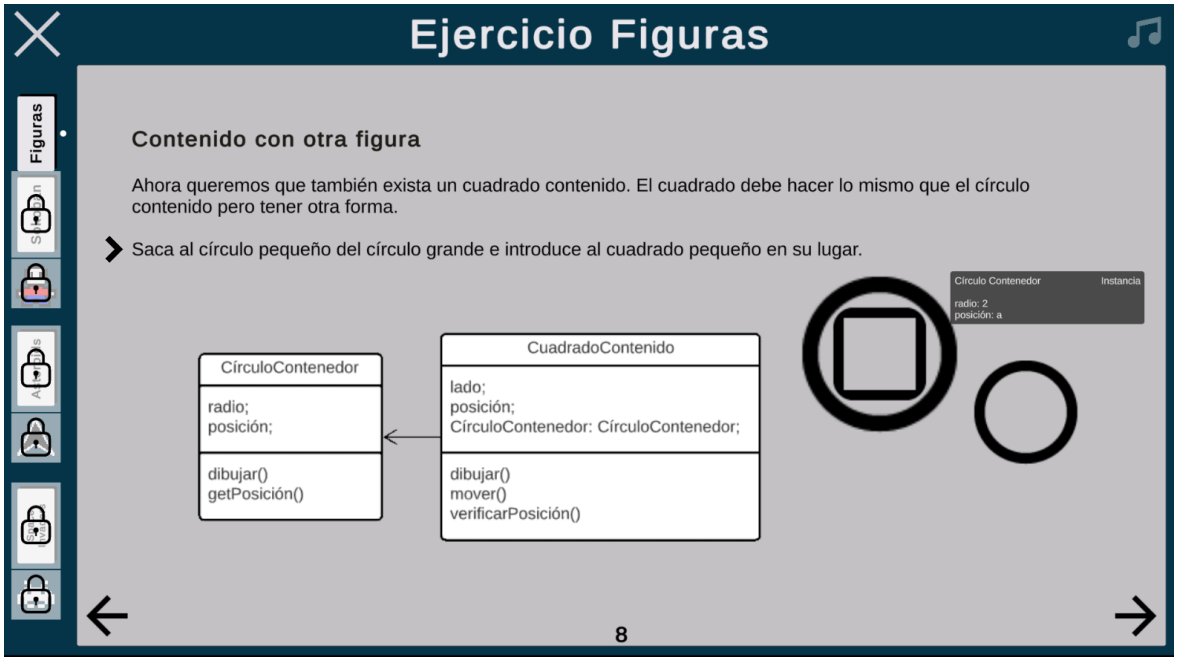


Figura 19 - Ejercicio Figuras

- **Sprint 1 Review**

Se creó el sistema de guardado, el sistema de niveles y el primer ejercicio de diseño satisfactoriamente. A lo largo del desarrollo del ejercicio la dueña del producto pidió varios cambios visuales para diferentes pantallas, esos cambios se hicieron correctamente para satisfacer sus necesidades. También se discutió con la dueña del producto formas de mejorar los siguientes ejercicios.

Historia de Usuario	Criterio de Aceptación	Cumplido
T09 - Desarrollo del sistema de guardado	Se desarrolló un sistema para guardar las notas de las pruebas	Sí
	Se desarrolló un sistema para guardar cuando el usuario termine un ejercicio	Sí
T10 - Desarrollo del sistema de desbloqueo de niveles	Se desarrolló un sistema para ir desbloqueando ejercicios en orden	Sí
	El sistema usa una interfaz gráfica agradable para el usuario	Sí
T11 - Desarrollar el ejercicio del diagrama de clases “Figuras”	Se desarrolló el ejercicio “Figuras” para enseñar conceptos básicos de diagrama de clases	Sí
	El ejercicio es interactivo	Sí

	El ejercicio es claro y conciso	Sí

Tabla 11 - Sprint 1 Review

- **Sprint 1 Retrospective**
    - **¿Qué salió bien?**

El equipo de desarrollo pudo familiarizarse rápidamente con el entorno de desarrollo en Unity.

- **¿Qué se puede mejorar?**

Los ejercicios requieren de más funcionalidades para mejorar su interactividad y así hacerlos más entretenidos para los usuarios finales.

4.3.2. Sprint 2

- **Sprint 2 Planning**

En este Sprint se desarrolló el videojuego Sokoban y se desarrolló el ejercicio mediante el cual el usuario final diseñará el videojuego Sokoban con el diagrama de clases UML. El ejercicio usa partes del juego de video durante el diseño para ilustrar el funcionamiento de los atributos y los métodos del diagrama de manera interactiva.

- **Sprint 2 Backlog**

Sprint 2 Backlog	
Código	**Nombre**
T15	Desarrollar el videojuego “Sokoban”
T12	Desarrollar el ejercicio del diagrama de clases “Sokoban”

Tabla 12 - Sprint 2 Backlog

- **Ejecución del Sprint 2**

El primer objetivo de este sprint era lograr desarrollar el videojuego Sokoban en Unity. Sokoban es un juego de rompecabezas clásico inventado en Japón, fue desarrollado por Hiroyuki Imabayashi en 1981 [29]. El nombre proviene del japonés y significa "encargado de almacén". El juego tiene varios niveles, cada nivel representa un almacén donde las cajas parecen estar colocadas al azar. El jugador

tiene que empujar las cajas por el laberinto de la habitación para que, al final, todas las cajas estén en los campos marcados [29]. Cada nivel tiene una estructura diferente, lo que requiere una estrategia diferente.

La simplicidad de las reglas, en combinación con los muchos niveles que van desde fácil hasta extremadamente difícil, hacen de Sokoban en un verdadero juego de video clásico.

Para empezar, se diseñaron los sprites que representarán a los elementos del juego, como se observa en la Figura 20.

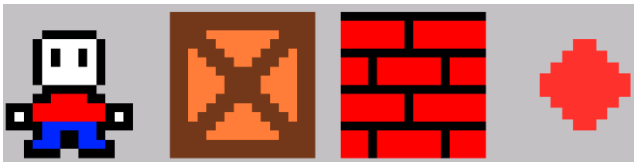


Figura 20 - Sprites de "Sokoban"

Luego se programó el movimiento del jugador, como se muestra en la Figura 21.

```
Unity Script (1 asset reference) | 3 references
public class SBPlayer : MonoBehaviour
{
    1 reference
    public bool Move(Vector2 direction)
    {
        // Debug.Log("Player is moving");
        if (Mathf.Abs(direction.x) < 0.5)
        {
            direction.x = 0;
        }
        else
        {
            direction.y = 0;
        }
        direction.Normalize();
        if (Blocked(transform.position, direction))
        {
            FindObjectOfType<SoundManagerScript>().Play("Blocked");
            return false;
        }
        else
        {
            transform.Translate(direction);
            FindObjectOfType<SoundManagerScript>().Play("Step");
            return true;
        }
    }
    1 reference
    bool Blocked(Vector3 position, Vector2 direction)
    {
        Vector2 newPos = new Vector2(position.x, position.y) + direction;
        GameObject[] walls = GameObject.FindGameObjectsWithTag("Wall");
        foreach (var wall in walls)
        {
            if (wall.transform.position.x == newPos.x && wall.transform.position.y == newPos.y)
            {
                return true;
            }
        }
        GameObject[] boxes = GameObject.FindGameObjectsWithTag("Box");
        foreach (var box in boxes)
        {
            if (box.transform.position.x == newPos.x && box.transform.position.y == newPos.y)
            {
                SBBox bx = box.GetComponent<SBBox>();
                if (bx && bx.Move(direction))
                {
                    return false;
                }
                else
                {
                    return true;
                }
            }
        }
        return false;
    }
}
```

Figura 21 - "SBPlayer" script

El jugador puede moverse diagonalmente hacia arriba, abajo, izquierda o derecha, si no tiene obstáculos en el camino, pero si se encuentra con una pared se queda quieto, si se encuentra con una caja esta tendrá que revisar si puede moverse o no, si la caja también puede moverse el jugador y la caja se mueven, pero si la caja no puede moverse el jugador y la caja se quedan quitos. El script de la caja puede observarse en la Figura 22.

La caja también tiene que revisar si está encima de un “punto de entrega”, si este es el caso debe cambiar de color para indicarlo, esto también está en la Figura 22.

```
Unity Script (1 asset reference) | 8 references
public class SBBox : MonoBehaviour
{
    public bool m_OnCross;
    3 references
    public bool Move(Vector2 direction)
    {
        if (BoxBlocked(transform.position, direction))
        {
            return false;
        }
        else
        {
            transform.Translate(direction);
            TestForOnCross();
            return true;
        }
    }

    1 reference
    private bool BoxBlocked(Vector3 position, Vector2 direction)
    {
        Vector2 newPos = new Vector2(position.x, position.y) + direction;
        GameObject[] walls = GameObject.FindGameObjectsWithTag("Wall");
        foreach (var wall in walls)
        {
            if (wall.transform.position.x == newPos.x && wall.transform.position.y == newPos.y)
            {
                return true;
            }
        }
        GameObject[] boxes = GameObject.FindGameObjectsWithTag("Box");
        foreach (var box in boxes)
        {
            if (box.transform.position.x == newPos.x && box.transform.position.y == newPos.y)
            {
                SBBox bx = box.GetComponent<SBBox>();
                if (bx && bx.Move(direction))
                {
                    return false;
                }
                else
                {
                    return true;
                }
            }
        }
        return false;
    }
    1 reference
    void TestForOnCross()
    {
        GameObject[] crosses = GameObject.FindGameObjectsWithTag("Cross");
        foreach (var cross in crosses)
        {
            if (transform.position.x == cross.transform.position.x && transform.position.y == cross.transform.position.y)
            {
                GetComponent<SpriteRenderer>().color = Color.gray;
                m_OnCross = true;
                return;
            }
        }
        GetComponent<SpriteRenderer>().color = Color.white;
        m_OnCross = false;
    }
}
```

Figura 22 - "SBBox" script

La pared y el punto de entrega no necesitaron sus propios scripts, ellos solo existen en cada nivel y el jugador o la caja interactúan con ellos dependiendo del caso.

Se procedió a crear 5 niveles para que el usuario pueda jugar el juego de video, estos funcionan en orden, uno después de otro, y van subiendo su dificultad. Cada nivel tiene un botón para reiniciarlo en caso de que el jugador haga mal sus movimientos y no puedo seguir avanzando, como también tiene un botón para

seguir al siguiente nivel una vez ponga todas las cajas en su lugar. Todo esto se puede observar en las Figuras 23 y 24.

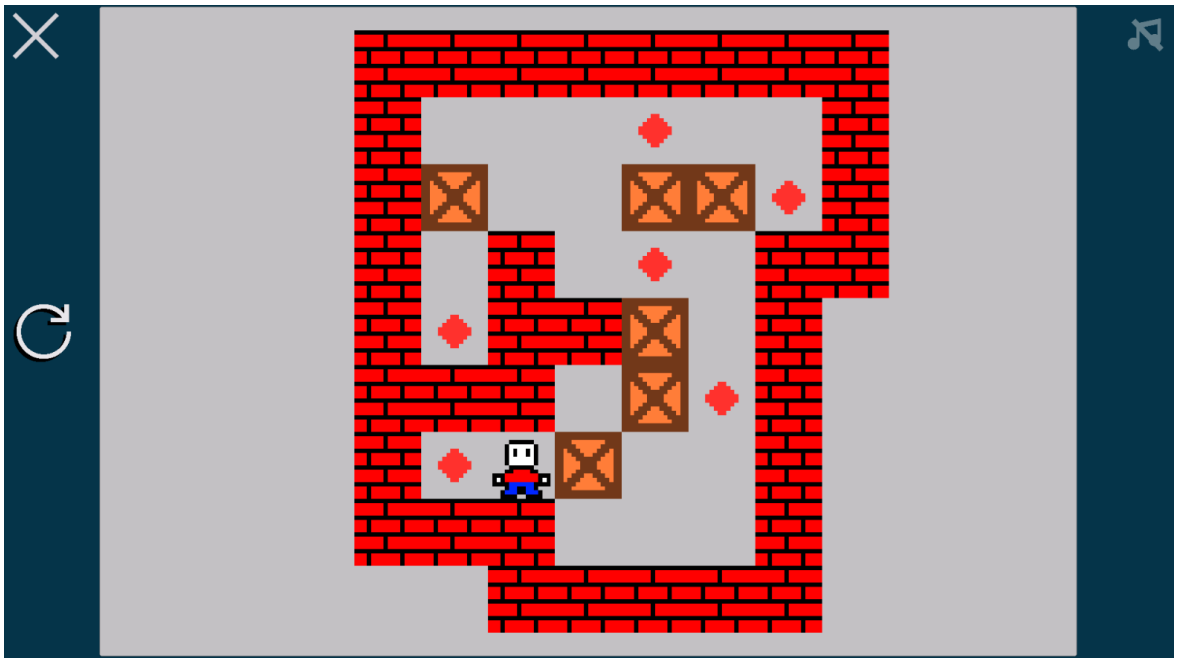


Figura 23 - Sokoban al iniciar un nivel

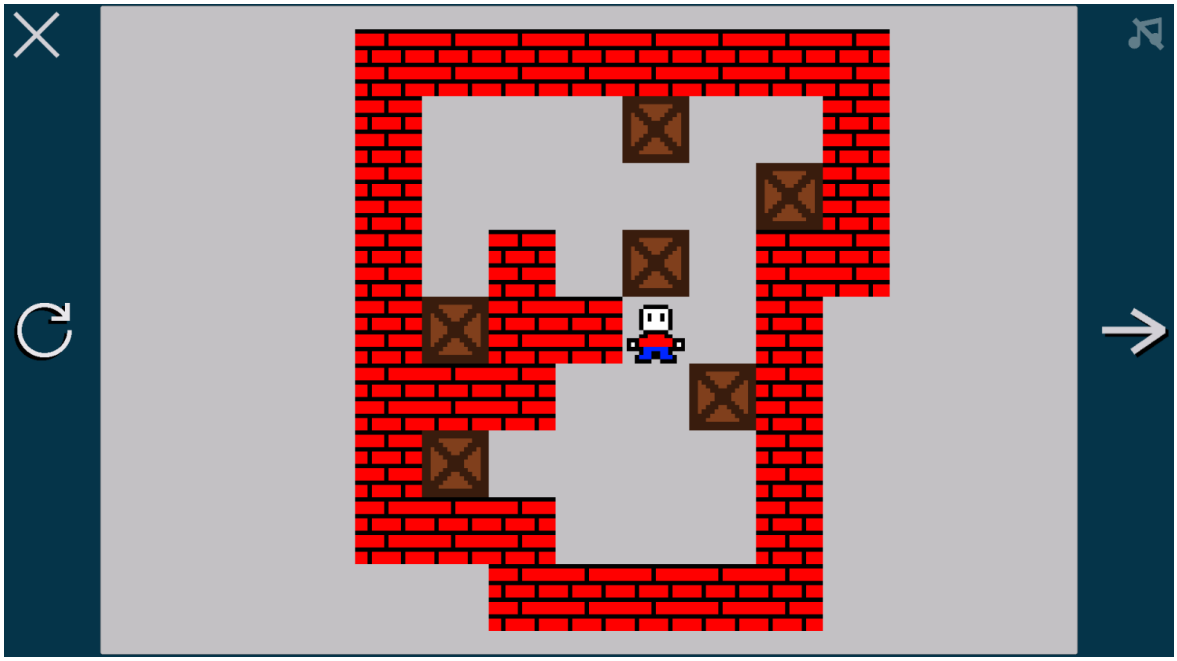


Figura 24 - Sokoban al terminar un nivel

Cuando se completan todos los niveles se coloca en pantalla un mensaje de felicitaciones para el jugador por haber completado la partida.

Después de completar el juego se procedió a diseñar el ejercicio para que los estudiantes aprendan conceptos de diseño de diagramas de clases UML. En este

ejercicio se enseña sobre Herencia, Agregación, Atributos, Métodos e Instanciación de clases.

Como en el ejercicio de las Figuras se empezó por diseñar el diagrama de clases final, que tuvo que ser aprobado primeramente por la dueña del producto, este diagrama puede observarse en la Figura 25.

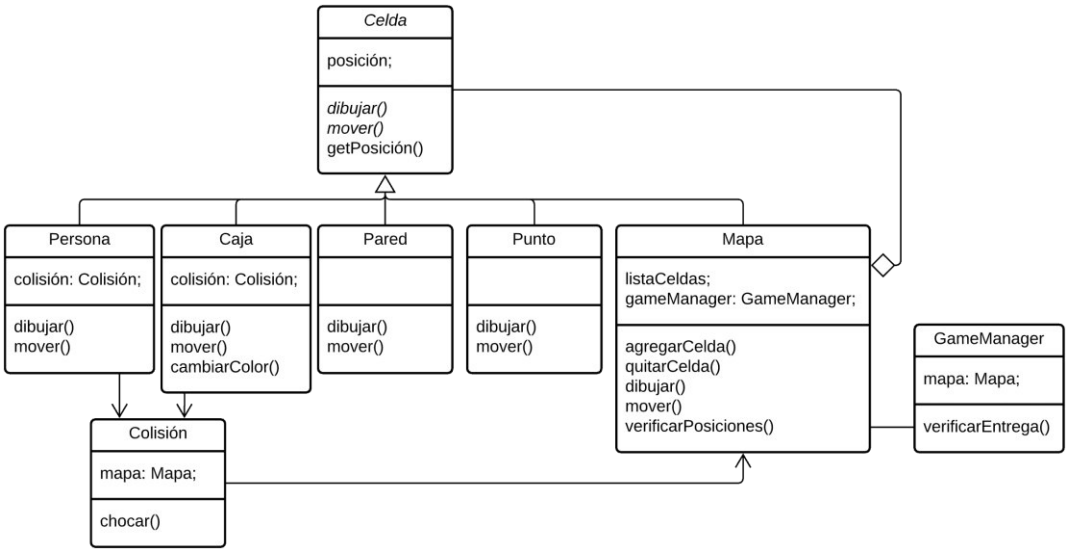


Figura 25 - Diagrama de clases de ejercicio "Sokoban"

En este ejercicio se usaron “splines” para hacer las explicaciones más interactivas, un “spline” es una forma de interpolar el movimiento de una figura entre dos puntos. Básicamente se creó un algoritmo que recibe dos posiciones y luego mueve un sprite entre esas dos posiciones, este algoritmo está en la Figura 26.

```
Unity Script (27 asset references) | 0 references
public class Spline : MonoBehaviour
{
    [SerializeField] private Transform pointA;
    [SerializeField] private Transform pointB;
    [SerializeField] private Transform pointAB;
    private float interpolateAmount;

    Unity Message | 0 references
    private void Update()
    {
        interpolateAmount = (interpolateAmount + Time.deltaTime / 1.2f) % 1f;
        pointAB.position = Vector3.Lerp(pointA.position, pointB.position, interpolateAmount);
    }
}
```

Figura 26 - "Spline" script

En la Figura 27 hay un ejemplo de cómo funciona “Spline”, simplemente se le da dos puntos y una componente, en este caso una flecha hacia abajo, y mientras el programa esté corriendo siempre tendrá una animación de la flecha moviéndose de arriba hacia abajo.

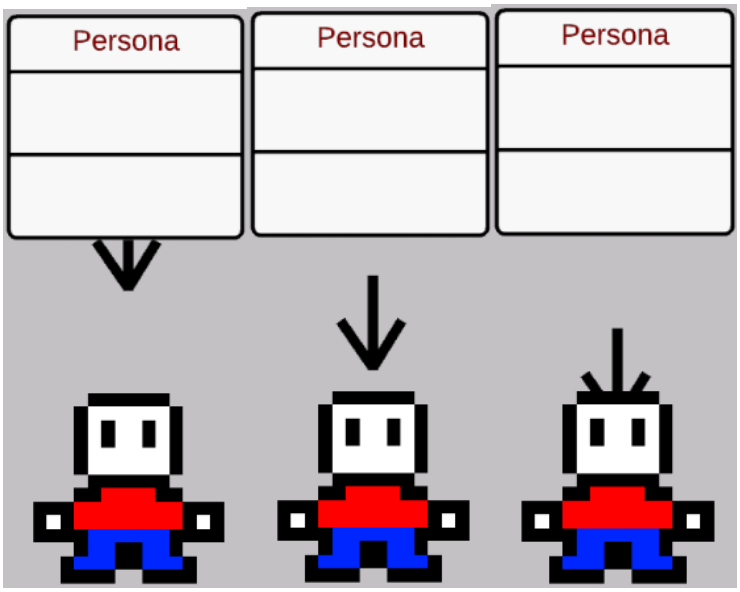


Figura 27 - Funcionamiento de "Spline"

En el ejercicio, mientras se va diseñando el diagrama de clases, se va viendo en el juego de video como se van agregando partes, por ejemplo, en el Panel 12 del ejercicio se agregó el método “cambiarColor()” en la clase Caja, y ahora el usuario puede jugar el juego y observar como la caja cambia de color, como se ve en las Figuras 28 y 29.

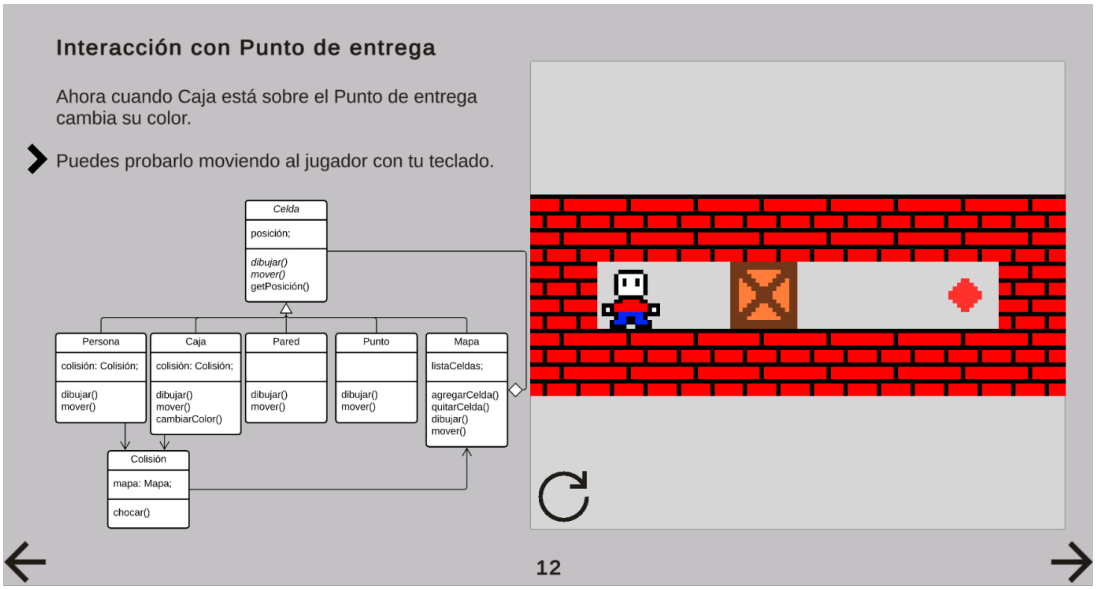


Figura 28 - Interación con Punto de Entrega 1

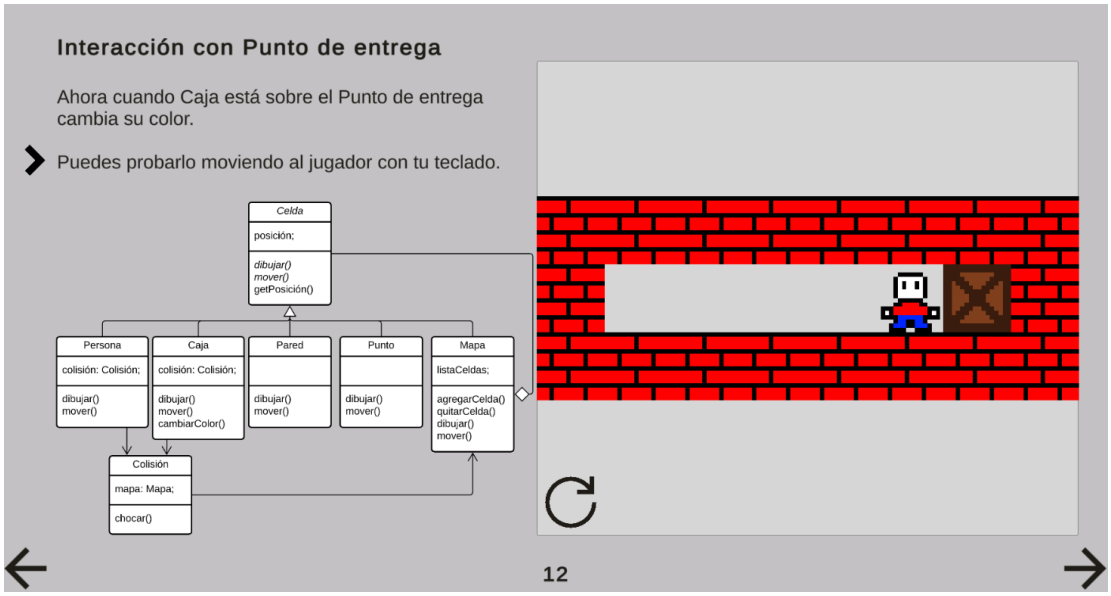


Figura 29 - Interación con Punto de Entrega 2

Una vez el jugador completa el ejercicio podrá jugar el juego que diseñamos anteriormente.

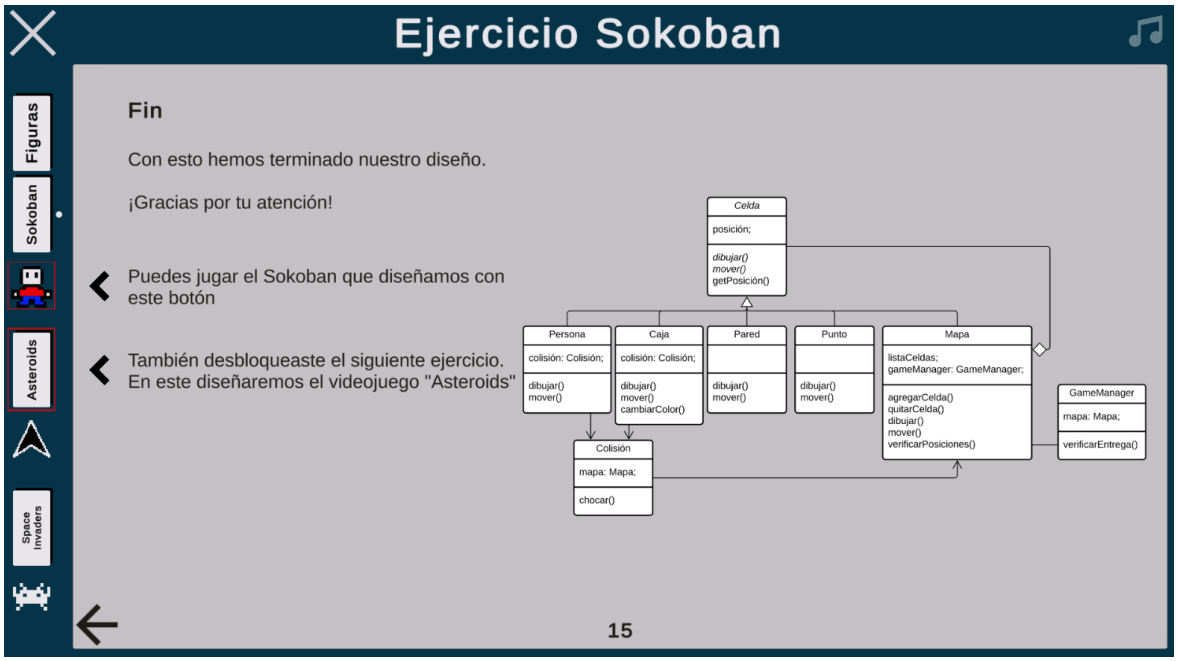


Figura 30 - Pantalla final ejercicio "Sokoban"

El ejercicio “Sokoban” terminó con 15 paneles nuevos y un videojuego clásico recreado satisfactoriamente en el motor gráfico Unity.

- **Sprint 2 Review**

Se pudo cumplir con el objetivo de crear el videojuego “Sokoban” como también su ejercicio de diseño.

Historia de Usuario	Criterio de Aceptación	Cumplido
T15 - Desarrollar el videojuego “Sokoban”	Se desarrolló el juego de video Sokoban en el motor gráfico Unity	Sí
	Se crearon cinco niveles del juego Sokoban para que puedan ser jugados por los usuarios	Sí
T12 - Desarrollar el ejercicio del diagrama de clases “Sokoban”	Se desarrolló el ejercicio de diseño de software “Sokoban” para enseñar conceptos de diagrama de clases	Sí
	El ejercicio es interactivo	Sí
	El ejercicio usa partes del juego para explicar como funciona el diagrama de clases	Sí
	Las explicaciones del ejercicio son claras	Sí

Tabla 13 - Sprint 2 Review

- **Sprint 2 Retrospective**
    - **¿Qué salió bien?**

Se mejoró la interactividad para los usuarios finales de la aplicación cuando realizan los ejercicios de diseño.

- **¿Qué se puede mejorar?**

La dueña del producto solicitó que el estilo de los ejercicios sea más conciso y coherente, se buscó que todas las pantallas sigan la misma estructura y tengan los mismos colores. Se harán esfuerzos para que esta estructura sea seguida en los siguientes ejercicios.

4.3.3. Sprint 3

- **Sprint 3 Planning**

En este Sprint se desarrolló el juego de video arcade clásico “Asteroids”, como también el ejercicio mediante el cual el usuario final diseñará el videojuego “Asteroids” con el diagrama de clases UML. El ejercicio usa partes del juego de video durante su ejecución para ilustrar el funcionamiento de varios elementos del diagrama de clases de una manera interactiva.

- **Sprint 3 Backlog**

Sprint 3 Backlog	
Código	**Nombre**
T16	Desarrollar el videojuego “Asteroids”
T13	Desarrollar el ejercicio del diagrama de clases “Asteroids”

Tabla 14 - Sprint 3 Backlog

- **Ejecución del Sprint 3**

El primer elemento de nuestro Sprint Backlog es desarrollar el videojuego “Asteroids” en Unity. Asteroids es un videojuego arcade de disparos con temática espacial diseñado por Lyle Rains y Ed Logg para la empresa Atari, fue lanzado al mercado en 1979 en Norte América [30]. A través de los años se convirtió en uno de los juegos de arcade clásicos más famosos de la historia.

En Asteroids el jugador controla una nave espacial triangular que puede en el sentido de las agujas del reloj, en el sentido contrario a las agujas del reloj y puede disparar hacia delante desde su "nariz". La nave también puede moverse hacia adelante, teniendo aceleración y desaceleración con inercia, la nave solo puede moverse en la dirección a la que apunta. Cada nivel comienza con algunos asteroides a la deriva y flotando en direcciones aleatorias a través de la pantalla. Todos los objetos del juego están envueltos alrededor de los bordes de la pantalla excepto las “balas” que dispara la nave, por ejemplo, un asteroide que se desvía del área superior de la pantalla reaparecería en la parte inferior de la pantalla y continuaría desplazándose en la misma dirección. A medida que el jugador dispara a los asteroides, estos se rompen en pedazos más pequeños que generalmente se mueven más rápido y son más difíciles de disparar [30]. Cada vez que se destruye un asteroide el jugador gana puntos y constantemente salen más asteroides, el objetivo del juego es sobrevivir el mayor tiempo posible, una vez la nave es destruida por un asteroide el juego termina.

Antes de crear la lógica del juego se diseñaron los sprites de Asteroids como se puede ver en la Figura 31.

Figura 31 - Sprites de "Asteroids"

La mayoría de la lógica del juego está en la nave espacial, como se ve en la Figura 32. Esta clase se encarga de decirle al juego como se mueve el jugador, cómo dispara y que si un asteroide lo llega a topar la nave se termina el juego.

```
Unity Script (4 asset references) | 1 reference
public class APlayer : MonoBehaviour
{
    [SerializeField]
    int scene;

    public float acceleration;
    public float maxSpeed;
    public float drag;
    public float angularSpeed;
    public float offsetBullet;
    public GameObject bulletPrefab;
    public bool canShoot = true;
    public float shootRate = 0.5f;

    private Rigidbody2D rb;
    private float vertical;
    private float horizontal;
    private bool shooting;

    Unity Message | 0 references
    private void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        rb.drag = drag;
    }

    Unity Message | 0 references
    private void Update()
    {
        vertical = AInputManager.Vertical;
        horizontal = AInputManager.Horizontal;
        shooting = AInputManager.Fire;

        Rotate();
        Shoot();
    }

    Unity Message | 0 references
    private void FixedUpdate()
    {
        var forwardMotor = Mathf.Clamp(vertical, 0f, 1f);
        rb.AddForce(transform.up * acceleration * forwardMotor);
        if (rb.velocity.magnitude > maxSpeed)
        {
            rb.velocity = rb.velocity.normalized * maxSpeed;
        }
    }

    1 reference
    private void Rotate()
    {
        if(horizontal == 0)
        {
            return;
        }
        transform.Rotate(0, 0, -angularSpeed * horizontal * Time.deltaTime);
    }

    1 reference
    private void Shoot()
    {
        if (shooting && canShoot)
        {
            FindObjectOfType<SoundManagerScript>().Play("Laser");
            StartCoroutine(FireRate());
        }
    }

    1 reference
    public void Lose()
    {
        // rb.velocity = Vector3.zero;
        // transform.position = Vector3.zero;
        SceneManager.LoadScene(scene);
    }

    1 reference
    private IEnumerator FireRate()
    {
        canShoot = false;
        var pos = transform.up * offsetBullet + transform.position;
        var bullet = Instantiate(bulletPrefab, pos, transform.rotation);
        Destroy(bullet, 2);
        yield return new WaitForSeconds(shootRate);
        canShoot = true;
    }
}
```

Figura 32 - "APlayer" script

Por otro lado, el script “AToroidalUniverse” es el encargado de que todos los elementos de juego (a excepción de las balas) que toquen los bordes de la pantalla se muevan hacia el lado contrario de la pantalla, como se ve en la Figura 33.

```
[System.Serializable]
1 reference
public struct Borders
{
    public float superiorBorder, inferiorBorder, leftBorder, rightBorder;
}

Unity Script (11 asset references) | 0 references
public class AToroidalUniverse : MonoBehaviour
{
    public Borders borders;

    Unity Message | 0 references
    private void Update()
    {
        var pos = transform.position;
        var x = transform.position.x;
        var y = transform.position.y;

        if (x > borders.rightBorder)
        {
            pos.x = borders.leftBorder;
            transform.position = pos;
        }
        if (x < borders.leftBorder)
        {
            pos.x = borders.rightBorder;
            transform.position = pos;
        }

        if (y > borders.superiorBorder)
        {
            pos.y = borders.inferiorBorder;
            transform.position = pos;
        }
        if (y < borders.inferiorBorder)
        {
            pos.y = borders.superiorBorder;
            transform.position = pos;
        }
    }
}
```

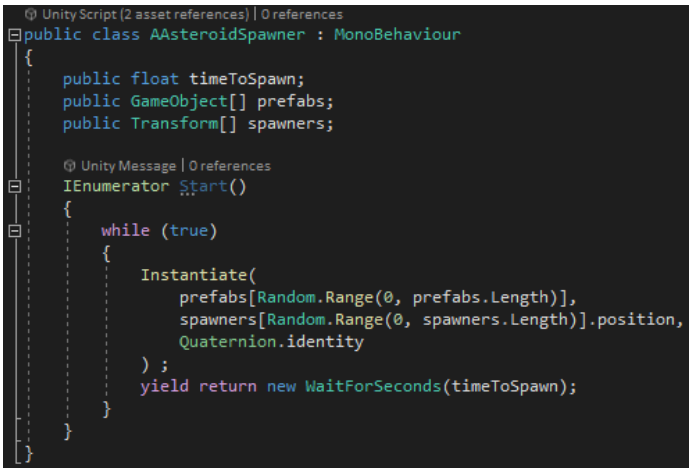
Figura 33 - "AToroidalUniverse" script

Para programar a los asteroides se usó el script “AAsteroid”, de la Figura 34. Este hace que los asteroides se muevan de manera aleatoria, que los asteroides se rompan en asteroides más pequeños, e indica que las balas destruyen a los asteroides y los asteroides destruyen a la nave espacial.

```
Unity Script (3 asset references) | 1 reference
public class AAsteroid : MonoBehaviour
{
    private Rigidbody2D rb;
    public float speed;
    public GameObject[] subAsteroids;
    public float numberOfAsteroids;

    private bool isDestroying = false;

    Unity Message | 0 references
    void OnEnable()
    {
        rb = GetComponent<Rigidbody2D>();
        rb.drag = 0;
        rb.angularDrag = 0;

        rb.velocity = new Vector3(
            Random.Range(-1f, 1f),
            Random.Range(-1f, 1f),
            0
        ).normalized * speed;

        rb.angularVelocity = Random.Range(-50f, 50f);
    }
    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D col)
    {
        if (isDestroying) return;

        if (col.CompareTag("Bullet"))
        {
            isDestroying = true;
            FindObjectOfType<SoundManagerScript>().Play("Explotion");
            Destroy(gameObject);
            Destroy(col.gameObject);
            for (var i = 0; i < numberOfAsteroids; i++)
            {
                Instantiate(
                    subAsteroids[Random.Range(0, subAsteroids.Length)],
                    transform.position,
                    Quaternion.identity
                    );
            }
        }

        if (col.CompareTag("Player"))
        {
            var asts = FindObjectsOfType<AAsteroid>();
            for (var i = 0; i < asts.Length; i++)
            {
                Destroy(asts[i].gameObject);
            }
            col.GetComponent<APlayer>().Lose();
        }
    }
```

Figura 34 - "AAsteroid" script

El script “AAsteroideSpawner” es el encargado de que constantemente se creen nuevos asteroides en lugares aleatorios a lo largo del juego. Puede ser visto en la Figura 35.



```
Unity Script (2 asset references) | 0 references
public class AAsteroidSpawner : MonoBehaviour
{
    public float timeToSpawn;
    public GameObject[] prefabs;
    public Transform[] spawners;

    Unity Message | 0 references
    IEnumerator Start()
    {
        while (true)
        {
            Instantiate(
                prefabs[Random.Range(0, prefabs.Length)],
                spawners[Random.Range(0, spawners.Length)].position,
                Quaternion.identity
            ) ;
            yield return new WaitForSeconds(timeToSpawn);
        }
    }
}
```

Figura 35 - "AAsteroidSpawner" script

El script “ABulletMovement”, de la Figura 36, indica la velocidad y dirección de las balas que son disparadas por la nave espacial.

```
Unity Script (2 asset references) | 0 references
public class ABulletMovement : MonoBehaviour
{
    public float speed;

    Unity Message | 0 references
    private void Update()
    {
        transform.position += transform.up * speed * Time.deltaTime;
    }
}
```

Figura 36 - "ABulletMovement" script

Y, la clase “AInputManager”, de la Figura 37, es complementaria de “APlayer”, nos sirve para indicar fácilmente los botones del teclado que serán usados para girar a la nave espacial y para disparar.

```
Unity Script | 6 references
public class AInputManager : MonoBehaviour
{
    2 references
    public static float Vertical
    {
        get { return Input.GetAxis("Vertical"); }
    }
    2 references
    public static float Horizontal
    {
        get { return Input.GetAxis("Horizontal"); }
    }
    2 references
    public static bool Fire
    {
        get { return Input.GetKey(KeyCode.LeftControl); }
    }
}
```

Figura 37 - "AInputManager" script

El videojuego “Asteroids” terminado puede ser visto en la Figura 38.

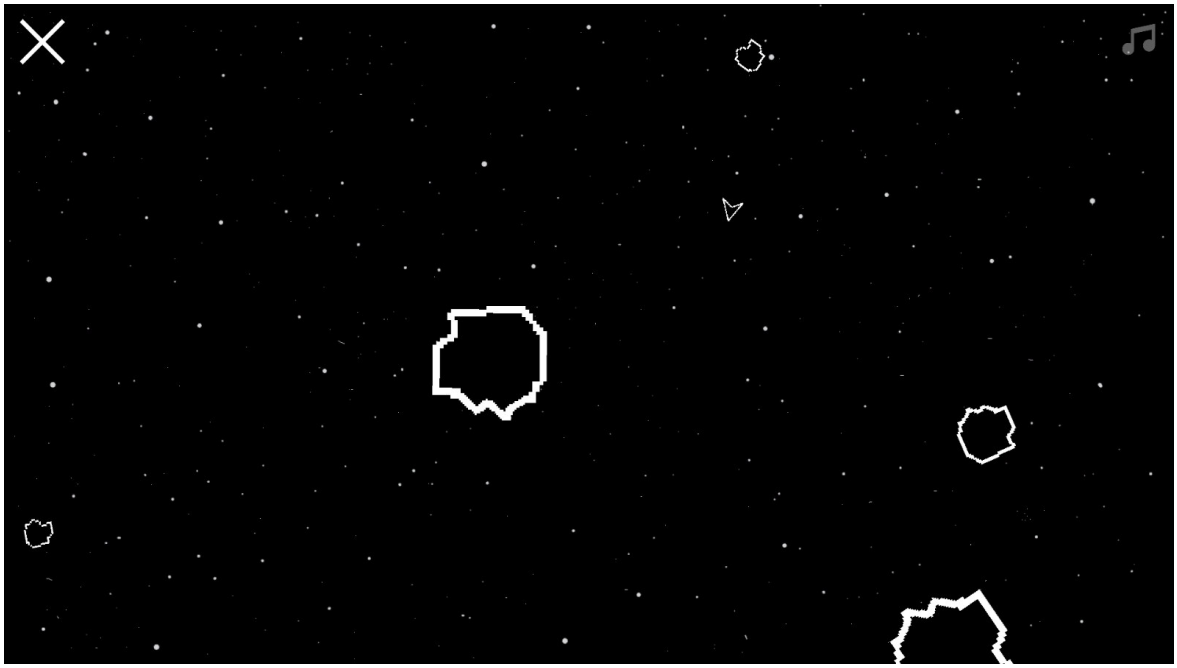


Figura 38 - Asteroids

Después de terminar el videojuego se procedió a crear el ejercicio de diseño. Como en los sprints anteriores, se diseñó el diagrama de clases y se consiguió la aprobación de la dueña del producto de este. Este diagrama está en la Figura 39.

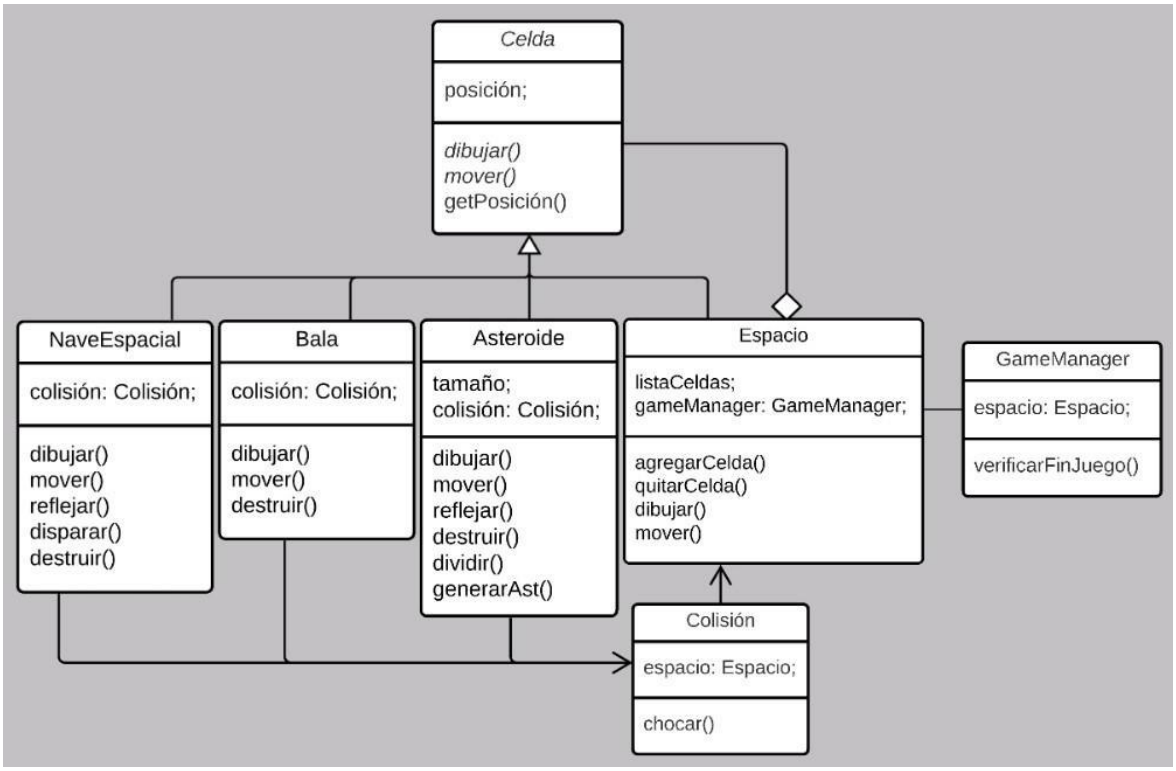


Figura 39 - Diagrama de clases de ejercicio "Asteroids"

Para hacer este ejercicio más interactivo, como fue solicitado por la dueña del producto, se crearon nuevos scripts que hacen a los ejercicios más dinámicos. Anteriormente hablamos del script “DragDrop” (Figura 16), este nos permite mover sprites de un lado al otro, ahora agregamos también el script “ItemSlot” (Figura 40), este recibirá el objeto cuando es soltado por “DragDrop” y revisará si está siendo puesto en el lugar correcto, si es así cambia a color verde y si el elemento ha sido soltado en un lugar incorrecto se pone de color rojo. Esta interacción puede ser observada en la Figura 41 y 42.

```
Unity Script (109 asset references) | 1 reference
public class ItemSlot2 : MonoBehaviour, IDropHandler
{
    [SerializeField] GameObject self;
    [SerializeField] GameObject activar;

    0 references
    public void OnDrop(PointerEventData eventData)
    {

        //Debug.Log("On Drop");
        if (eventData.pointerDrag != null)
        {
            eventData.pointerDrag.GetComponent<RectTransform>().anchoredPosition = GetComponent<RectTransform>().anchoredPosition;
            if (eventData.pointerDrag.gameObject.tag == self.tag)
            {
                self.GetComponent<Image>().color = Color.green;
                eventData.pointerDrag.gameObject.GetComponent<Image>().raycastTarget = false;
                self.GetComponent<ItemSlot2>().enabled = false;
                activar.SetActive(true);
                FindObjectOfType<SoundManagerScript>().Play("Success");
            } else
            {
                self.GetComponent<Image>().color = Color.red;
                FindObjectOfType<SoundManagerScript>().Play("Failure");
            }
        }
    }
}
```

Figura 40 - "ItemSlot" script

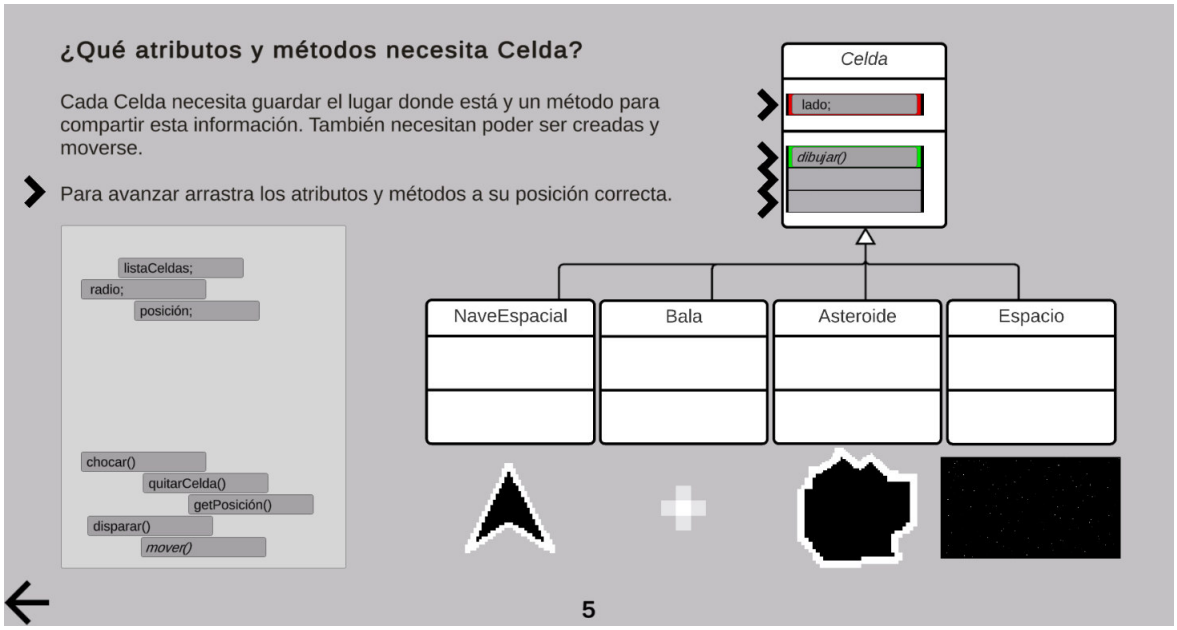


Figura 41 - "ItemSlot" cambia a color rojo

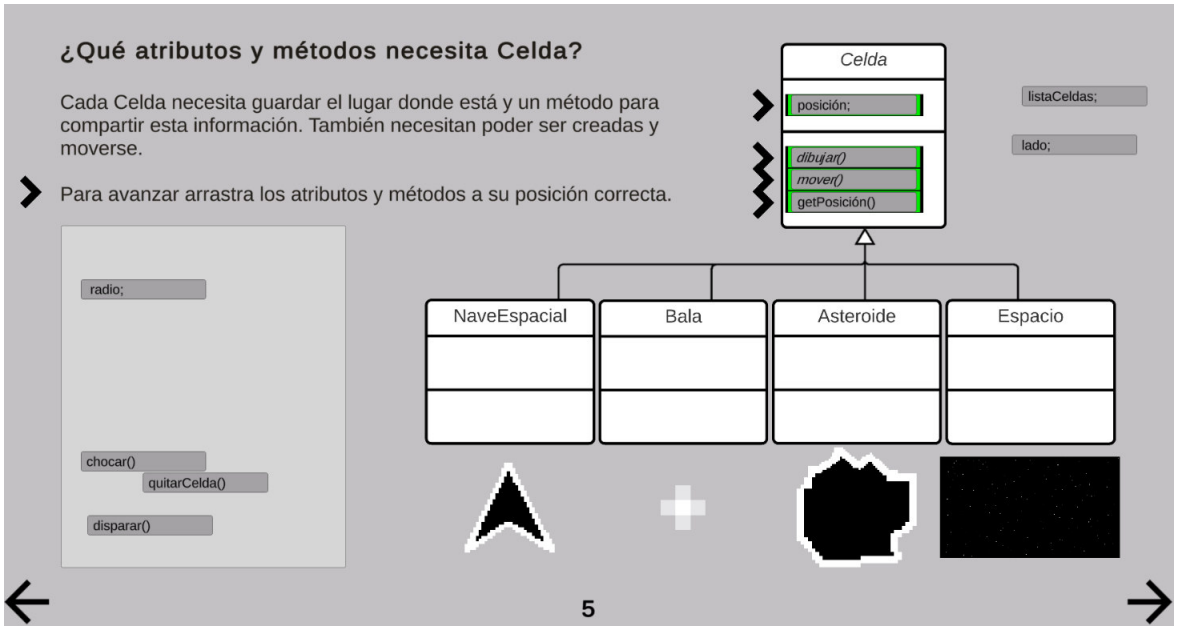


Figura 42 - "ItemSlot" cambia a color verde

Adicionalmente, se agregó el script “GreenChecker” que revisa si todos los “ItemSlot” están de color verde y solo en ese caso activa el botón para continuar con los siguientes ejercicios, se puede ver en la Figura 43.

```
Unity Script (16 asset references) | 0 references
public class GreenChecker : MonoBehaviour
{
    [SerializeField] GameObject caja1;
    [SerializeField] GameObject caja2;
    [SerializeField] GameObject caja3;
    [SerializeField] GameObject caja4;
    [SerializeField] GameObject activate;

    Unity Message | 0 references
    void Update()
    {
        if (caja1.GetComponent<Image>().color == Color.green &&
            caja2.GetComponent<Image>().color == Color.green &&
            caja3.GetComponent<Image>().color == Color.green &&
            caja4.GetComponent<Image>().color == Color.green)
        {
            activate.SetActive(true);
        }
        else
        {
            activate.SetActive(false);
        }
    }
}
```

Figura 43 - "GreenCheker" script

En el resto del ejercicio se reutilizaron elementos añadidos en ejercicios anteriores como los “tooltip”, los “splines” y tener elementos interactivos del juego que se está diseñando mientras está siendo diseñado, siempre asegurándonos que todos los ejercicios tengan coherencia visual entre ellos.

El ejercicio “Sokoban” terminó con 19 paneles nuevos y otro juego de video clásico recreado satisfactoriamente en el motor gráfico Unity.

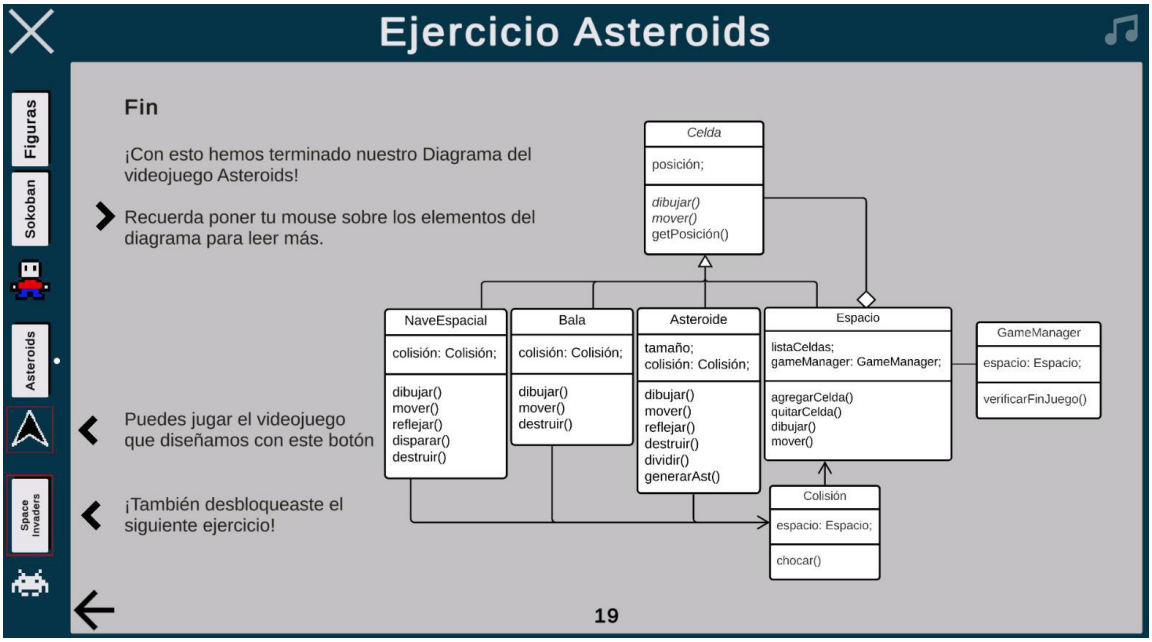


Figura 44 - Pantalla final ejercicio "Asteroids"

- **Sprint 3 Review**

Se cumplió con tarea de crear el videojuego “Asteroids”, como también se cumplió con la tarea de crear el ejercicio de diseño del juego de video “Asteroids”. En este sprint se siguió un estándar más rígido para el diseño del ejercicio, tomando en cuenta las recomendaciones de la dueña del producto.

Historia de Usuario	Criterio de Aceptación	Cumplido
T16 - Desarrollar el videojuego “Asteroids”	Se desarrolló el juego de video Asteroids en el motor gráfico Unity	Sí
	El juego es interactivo	Sí
	El juego tiene una interfaz agradable	Sí
T13 - Desarrollar el ejercicio del diagrama de clases “Asteroids”	Se desarrolló el ejercicio de diseño de software “Asteroids” para enseñar conceptos de diagrama de clases	Sí
	El ejercicio es interactivo	Sí
	El ejercicio usa partes del juego para explicar como funciona el diagrama de clases	Sí
	Las explicaciones del ejercicio son claras	Sí

	El ejercicio tiene un nivel de dificultad mayor que el anterior	Sí

Tabla 15 - Sprint 3 Review

- **Sprint 3 Retrospective**

    - **¿Qué salió bien?**

El diseño del ejercicio cumplió a cabalidad con los requerimientos de la dueña del producto y se pudo reutilizar sistemas creados anteriormente, lo que facilitó el trabajo.

    - **¿Qué se puede mejorar?**

Para el siguiente ejercicio se solicitó por parte de la dueña del producto mejorar aún más la interactividad de los usuarios finales con la aplicación, como también aumentar la dificultad de estos.

4.3.4. Sprint 4

- **Sprint 4 Planning**

En este Sprint se desarrolló el videojuego clásico “Space Invaders”, como también el ejercicio mediante el cual el usuario final diseñará el juego de video “Space Invaders” con un diagrama de clases UML. Al ser el ejercicio final, este debe tener más funcionalidad que los ejercicios anteriores para mostrar más partes de un diagrama de clases.

- **Sprint 4 Backlog**

Sprint 4 Backlog	
Código	**Nombre**
T17	Desarrollar el videojuego “Space Invaders”
T14	Desarrollar el ejercicio del diagrama de clases “Space Invaders”

Tabla 16 - Sprint 4 Backlog

- **Ejecución del Sprint 4**

Como en los sprints anteriores, primeramente, se procedió a desarrollar el juego de video. En este caso el juego fue “Space Invaders”, este es un juego arcade de disparos de 1978 desarrollado por Tomohiro Nishikado para la empresa Taito [31].

El objetivo del juego es derrotar oleada tras oleada de alienígenas que descienden desde la parte superior de la pantalla. El jugador controla un tanque, que dispara municiones desde la parte inferior de la pantalla hacia arriba, el tanque puede moverse únicamente en sentido horizontal [31]. Cuando el jugador destruye a los invasores va ganando puntos dependiendo del tipo de invasor que destruya. Los invasores también disparan láseres hacia abajo y si el jugador es impactado por un láser o toca a un invasor inmediatamente es destruido y pierde una vida. El jugador tiene un número determinado de vidas al empezar la partida, si pierde todas sus vidas pierde la partida, pero si el jugador logra destruir a todos los invasores gana la partida. También existe un invasor especial en forma de UFO que pasa por la parte superior de la pantalla cada cierto tiempo, este enemigo entrega puntos extra para el jugador si lo logra destruir. En este juego también existen unos escudos que sirven de protección para el jugador, estos se destruyen por partes por los láseres de los invasores o por los disparos del mismo jugador. “Space Invaders” cuenta también con una Interfaz gráfica que muestra la cantidad de vidas que tiene el jugador y su puntaje.

Para este juego se diseñaron los sprites de la Figura 45.

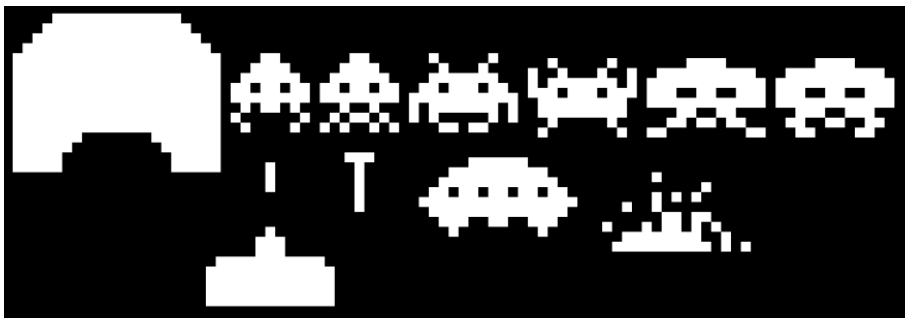


Figura 45 - Sprites de "Space Invaders"

Se definió el escudo que defiende al jugador, este tiene que romperse por partes así que se diseñó un elemento de juego con varias partes separadas y cada parte se le agregó el script “Bunker” (Figura 46), el cual hace que se destruya cuando colisiona con cualquier otro elemento de juego.

```
Unity Script (88 asset references) | 0 references
public class Bunker : MonoBehaviour
{
    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.layer == LayerMask.NameToLayer("Invader"))
        {
            this.gameObject.SetActive(false);
        }

        if (collision.gameObject.layer == LayerMask.NameToLayer("Missile"))
        {
            this.gameObject.SetActive(false);
        }

        if (collision.gameObject.layer == LayerMask.NameToLayer("Laser"))
        {
            this.gameObject.SetActive(false);
        }
        FindObjectOfType<SoundManagerScript>().Play("Explotion3");
    }
}
```

Figura 46 - "Bunker" script

Luego se creó la lógica para los invasores, la clase “Invader” (Figura 47) hace que cada invasor tenga una animación en la que cambia entre 2 sprites distintos cada segundo, por eso cada invasor tiene 2 sprites. También le indica que tiene que destruirse si choca con un proyectil lanzado por el jugador.

```
Unity Script (1 asset reference) | 2 references
public class Invader : MonoBehaviour
{
    public Sprite[] animationSprites;
    public float animationTime = 1.0f;
    public int score = 30;

    public System.Action killed;

    private SpriteRenderer _spriteRenderer;
    private int _animationFrame;

    Unity Message | 0 references
    private void Awake()
    {
        _spriteRenderer = GetComponent<SpriteRenderer>();
    }

    Unity Message | 0 references
    private void Start()
    {
        InvokeRepeating(nameof(AnimateSprite), this.animationTime, this.animationTime);
    }

    1 reference
    private void AnimateSprite()
    {
        _animationFrame++;

        if(_animationFrame >= this.animationSprites.Length)
        {
            _animationFrame = 0;
        }

        _spriteRenderer.sprite = this.animationSprites[_animationFrame];
    }

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.gameObject.layer == LayerMask.NameToLayer("Laser"))
        {
            this.killed.Invoke();
            this.gameObject.SetActive(false);
            FindObjectOfType<GameManager>().AddPuntos(score);
            FindObjectOfType<SoundManagerScript>().Play("Explotion");
        }
    }
}
```

Figura 47 - "Invader" script

También se creó el script “Invaders” (Figura 48), este nos sirve para crear todo el bloque de invasores durante el juego. Crea 11 columnas de invasores con 5 filas cada uno, pone invasores en su lugar dependiendo del tipo de invasor. También hace que todos los invasores se muevan en conjunto primero horizontalmente y luego verticalmente tras de tocar el filo de la pantalla. También guarda un listado de todos los invasores que existen y una vez todos están muertos termina el juego y le da la victoria al jugador, pero, si cualquiera de los invasores toca al jugador, termina el juego y le avisa al jugador que perdió.

```
Unity Script (1 asset reference) | 0 references
public class Invaders : MonoBehaviour
{
    public Invader[] prefabs;

    public int rows = 5;
    public int columns = 11;
    public AnimationCurve speed;
    public Proyectile missilePrefab;
    public float missileAttackRate = 1.0f;

    4 references
    public int amountKilled { get; private set; }
    1 reference
    public int amountAlive => this.totalInvaders - this.amountKilled;
    3 references
    public int totalInvaders => this.rows * this.columns;
    1 reference
    public float percentKilled => (float)this.amountKilled / (float)this.totalInvaders;

    private Vector3 _direction = Vector2.right;

    Unity Message | 0 references
    private void Awake()
    {

        for (int row = 0; row < this.rows; row++)
        {
            float width = 2.0f * (this.columns - 1);
            float height = 2.0f * (this.rows - 1);
            Vector2 centering = new Vector2(-width / 2, -height / 2);
            Vector3 rowPosition = new Vector3(centering.x, centering.y + (row * 2.0f), 0.0f);

            for(int col = 0; col < this.columns; col++)
            {
                Invader invader = Instantiate(this.prefabs[row], this.transform);
                invader.killed += InvaderKilled;

                Vector3 position = rowPosition;
                position.x += col * 2.0f;
                invader.transform.localPosition = position;
            }
        }
    }

    Unity Message | 0 references
    private void Start()
    {
        InvokeRepeating(nameof(MissileAttack), this.missileAttackRate, this.missileAttackRate);
    }
    Unity Message | 0 references
    private void Update()
    {
        this.transform.position += _direction * this.speed.Evaluate(this.percentKilled) * Time.deltaTime;

        Vector3 leftEdge = Camera.main.ViewportToWorldPoint(Vector3.zero);
        Vector3 rightEdge = Camera.main.ViewportToWorldPoint(Vector3.right);

        foreach (Transform invader in this.transform)
        {
            if (!invader.gameObject.activeInHierarchy)
            {
                continue;
            }

            if(_direction == Vector3.right && invader.position.x >= rightEdge.x - 13f)
            {
                AdvanceRow();
            } else if (_direction == Vector3.left && invader.position.x <= leftEdge.x + 13f)
            {
                AdvanceRow();
            }
        }
    }

    2 references
    private void AdvanceRow()
    {
        _direction.x *= -1.0f;

        Vector3 position = this.transform.position;
        position.y -= 1.0f;
        this.transform.position = position;
    }

    1 reference
    private void MissileAttack()
    {
        foreach (Transform invader in this.transform)
        {
            if (!invader.gameObject.activeInHierarchy)
            {
                continue;
            }

            if (Random.value < (1.0f / (float)this.amountAlive))
            {
                Instantiate(this.missilePrefab, invader.position, Quaternion.identity);
                FindObjectOfType<SoundManagerScript>().Play("Laser2");
                break;
            }
        }
    }

    1 reference
    private void InvaderKilled()
    {
        this.amountKilled++;

        if (this.amountKilled >= this.totalInvaders)
        {
            FindObjectOfType<GameManager>().WipeBoard();
        }
    }
}
```

Figura 48 - "Invaders" script

Luego se creó la lógica para el jugador en el script “Player” (Figura 49), en este script se definió que solo se puede mover horizontalmente, que puede disparar proyectiles, pero solo un proyectil a la vez, también que cuando es destruido se tienen que bajar el número de vidas, y si ya no hay más vidas se termina el juego avisando que el jugador perdió.

```
Unity Script (1 asset reference) | 0 references
public class Player : MonoBehaviour
{
    public Proyectile laserPrefab;

    public float speed = 5.0f;

    private bool _laserActive = false;

    Unity Message | 0 references
    private void Update()
    {
        // Player Movement
        if (Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow))
        {
            if (this.transform.position.x > -13.8f)
            {
                this.transform.position += Vector3.left * this.speed * Time.deltaTime;
            }

        } else if (Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow))
        {
            if (this.transform.position.x < 13.8f)
            {
                this.transform.position += Vector3.right * this.speed * Time.deltaTime;
            }

        }

        // Player Shooting
        if(Input.GetKeyDown(KeyCode.LeftControl) || Input.GetKeyDown(KeyCode.RightControl))
        {
            Shoot();
        }

    }

    1 reference
    private void Shoot()
    {
        if (!_laserActive)
        {
            Proyectile projectile = Instantiate(this.laserPrefab, this.transform.position, Quaternion.identity);
            projectile.destroyed += LaserDestroyed;
            _laserActive = true;
            FindObjectOfType<SoundManagerScript>().Play("Laser");
        }
    }
    1 reference
    private void LaserDestroyed()
    {
        _laserActive = false;
    }

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.layer == LayerMask.NameToLayer("Invader"))

        {
            FindObjectOfType<GameManager>().vidasTanque = 0;
            FindObjectOfType<GameManager>().UpdateVidas();
            FindObjectOfType<GameManager>().WipeBoard();
        }
        else if (collision.gameObject.layer == LayerMask.NameToLayer("Missile"))
        {
            FindObjectOfType<GameManager>().QuitarVida();
            this.gameObject.SetActive(false);
            if(FindObjectOfType<GameManager>().vidasTanque > 0)
            {
                Invoke("Respawn", 1.0f);
            }
            else if (FindObjectOfType<GameManager>().vidasTanque <= 0)
            {
                FindObjectOfType<GameManager>().WipeBoard();
            }
        }
        FindObjectOfType<SoundManagerScript>().Play("Explotion2");
    }

    0 references
    private void Respawn()
    {
        this.transform.position.Set( 0f, -13.0f, 0f);
        this.gameObject.SetActive(true);
    }
}
```

Figura 49 - "Player" script

Para la nave especial que aparece cada cierto tiempo en la parte superior de la pantalla se usó el script “MysteryShip” (Figura 50). Se selecciona dos puntos en la pantalla, la nave aparecerá cada cierto tiempo y se moverá entre esos dos puntos, primero de izquierda a derecha y luego de derecha a izquierda. También al ser destruida agrega más puntos que los invasores normales al puntaje del jugador.

```
Unity Script (1 asset reference) | 1 reference
public class MysteryShip : MonoBehaviour
{
    public float speed = 5.0f;
    public float cycleTime = 30.0f;
    public int score = 30;

    public System.Action<MysteryShip> killed;
    5 references
    public Vector3 leftDestination { get; private set; }
    4 references
    public Vector3 rightDestination { get; private set; }
    4 references
    public int direction { get; private set; } = -1;
    3 references
    public bool spawned { get; private set; }

    Unity Message | 0 references
    private void Start()
    {
        Vector3 leftEdge = Camera.main.ViewportToWorldPoint(Vector3.zero);
        Vector3 rightEdge = Camera.main.ViewportToWorldPoint(Vector3.right);

        Vector3 left = this.transform.position;
        left.x = leftEdge.x + 10.0f;
        this.leftDestination = left;

        Vector3 right = this.transform.position;
        right.x = rightEdge.x - 10.0f;
        this.rightDestination = right;

        this.transform.position = this.leftDestination;

        Despawn();
    }

    1 reference
    private void Spawn()
    {
        this.direction *= -1;
        if (this.direction == 1)
        {
            this.transform.position = this.leftDestination;
        }
        else
        {
            this.transform.position = this.rightDestination;
        }
        this.spawned = true;
    }
    4 references
    private void Despawn()
    {
        this.spawned = false;
        if (this.direction == 1)
        {
            this.transform.position = this.rightDestination;
        }
        else
        {
            this.transform.position = this.leftDestination;
        }
        Invoke(nameof(Spawn), this.cycleTime);
    }

    Unity Message | 0 references
    private void Update()
    {
        if (!this.spawned)
        {
            return;
        }
        if (this.direction == 1)
        {
            this.transform.position += Vector3.right * this.speed * Time.deltaTime;
            if (this.transform.position.x >= this.rightDestination.x)
            {
                Despawn();
            }
        }
        else
        {
            this.transform.position += Vector3.left * this.speed * Time.deltaTime;

            if (this.transform.position.x <= this.leftDestination.x)
            {
                Despawn();
            }
        }
    }

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.layer == LayerMask.NameToLayer("Laser"))
        {
            Despawn();
            this.killed?.Invoke(this);
            FindObjectOfType<GameManager>().AddPuntos(score);
            FindObjectOfType<SoundManagerScript>().Play("Explotion");
            FindObjectOfType<SoundManagerScript>().Play("Success");
        }
    }
```

Figura 50 - "MysteryShip" script

En el script “GameManager” (Figura 51) es donde ser guardan los puntos y las vidas, este verifica cuando se pierde o se gana el juego, también permite reiniciar el juego si es necesario.

```
Unity Script (1 asset reference) | 10 references
public class GameManager : MonoBehaviour
{
    public int puntos;
    public TextMeshProUGUI textoPuntos;

    public int vidasTanque = 3;
    public TextMeshProUGUI textoVidas;

    public GameObject win;
    public GameObject lose;

    2 references
    public void AddPuntos(int punt)
    {
        this.puntos += punt;
        UpdatePuntos();
    }

    1 reference
    private void UpdatePuntos()
    {
        this.textoPuntos.text = "PUNTOS: " + this.puntos;
    }

    1 reference
    public void QuitarVida()
    {
        this.vidasTanque -= 1;
        UpdateVidas();
    }

    2 references
    public void UpdateVidas()
    {
        this.textoVidas.text = "VIDAS: " + this.vidasTanque;
    }

    3 references
    public void WipeBoard()
    {
        GameObject[] panels = GameObject.FindGameObjectsWithTag("SI-");
        foreach (var panel in panels)
        {
            panel.gameObject.SetActive(false);
        }

        if (this.vidasTanque <= 0)
        {
            lose.SetActive(true);
            FindObjectOfType<SoundManagerScript>().Play("Failure2");
        }
        else
        {
            win.SetActive(true);
            FindObjectOfType<SoundManagerScript>().Play("Success2");
        }
    }

    0 references
    public void RestartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

Figura 51 - "GameManager" script

El proyectil también necesitaba ser programado, esto es porque el jugador solo puede disparar un proyectil a la vez, si el proyectil sale de pantalla sin chocar con nada nunca es destruido, por ende, el jugador no puede volver a disparar, por eso

tiene que autodestruirse si no choca con nada y sale de pantalla. Esto se puede ver en la Figura 52.

```
Unity Script (2 asset references) | 3 references
public class Proyectile : MonoBehaviour
{
    public Vector3 direction;
    public float speed;
    public System.Action destroyed;

    Unity Message | 0 references
    private void Update()
    {
        this.transform.position += this.direction * this.speed * Time.deltaTime;
    }

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {
        SelfDestroy();
    }

    Unity Message | 0 references
    private void Start()
    {

        Invoke("SelfDestroy", 1.5f);
    }

    1 reference
    private void SelfDestroy()
    {
        if (this.destroyed != null)
        {
            this.destroyed.Invoke();
        }
        Destroy(this.gameObject);
    }
}
```

Figura 52 - "Proyectile" script

Una vez tenemos toda la lógica del juego, armamos en Unity nuestro nivel, agregamos todos nuestros componentes y tenemos el juego funcionando, como se ve en la Figura 53.

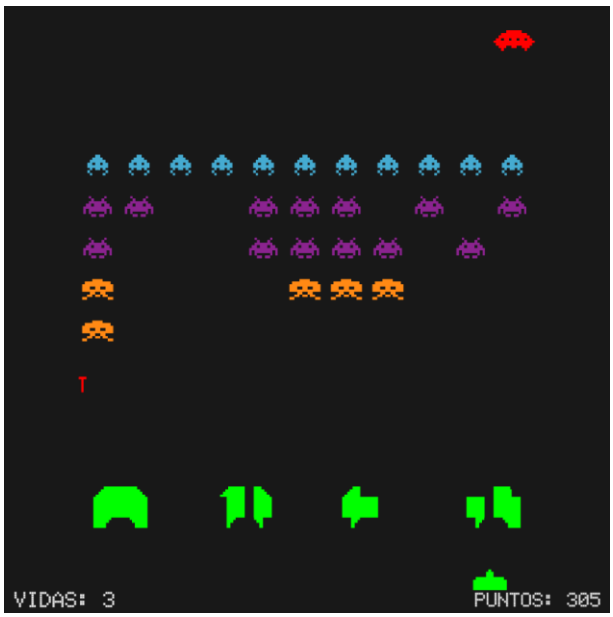


Figura 53 - Space Invaders

Ahora continuamos con el ejercicio, diseñando el diagrama de clases para su aprobación por la dueña del producto, la Figura 54 tiene el diagrama de clases aprobado.

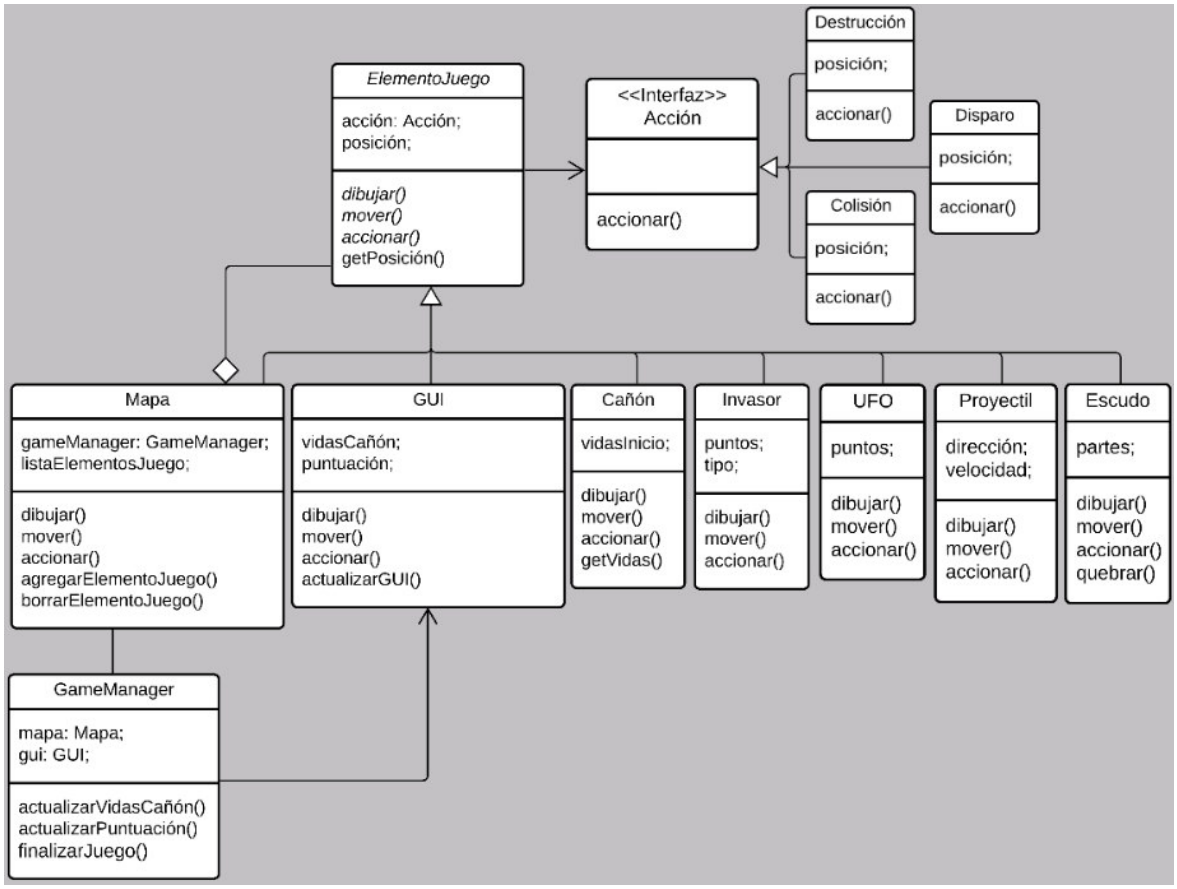


Figura 54 - Diagrama de clases del ejercicio "Space Invaders"

En este ejercicio se reforzaron conocimientos de elementos del diagrama de clases como herencia, uso de atributos, uso de métodos, agregación, asociación direccional y bidireccional, y también, se enseñó el uso de interfaces, como también elementos más complejos como la interfaz de usuario y un elemento como el “GameManager” que es esencial porque guarda lógica del negocio importante a pesar de no tener una instancia visible dentro del juego.

El ejercicio “Space Invaders” terminó con 14 nuevos paneles que reutilizaron elementos creados para ejercicios anteriores y otro juego de video arcade clásico recreado satisfactoriamente en el motor gráfico Unity.

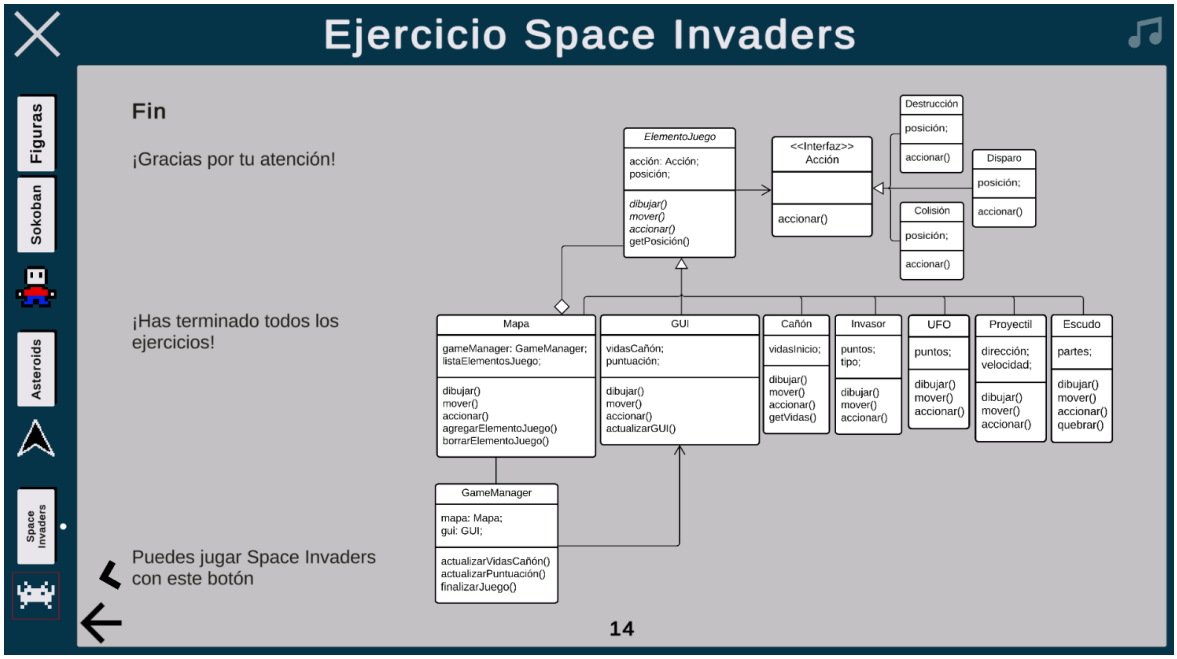


Figura 55 - Pantalla final ejercicio "Space Invaders"

- **Sprint 4 Review**

Se logró cumplir con tarea de crear el videojuego “Space Invaders”, como también se cumplió con la tarea de crear el ejercicio de diseño del juego de video “Space Invaders”. Al ser el ejercicio final se tuvo mayor rigidez con la calidad del diseño de las pantallas, manteniendo los estándares creados anteriormente. La dificultad del diseño también aumentó para incluir nuevas partes del diagrama de clases UML.

Historia de Usuario	Criterio de Aceptación	Cumplido
T17 - Desarrollar el videojuego “Space Invaders”	Se desarrolló el juego de video “Space Invaders” en el motor gráfico Unity	Sí
	El juego es interactivo	Sí
	El juego tiene una interfaz agradable	Sí
	El juego incorpora un sistema de vidas	Sí
	El juego guarda la puntuación del jugador	Sí
T14 - Desarrollar el ejercicio del diagrama de clases “Space Invaders”	Se desarrolló el ejercicio de diseño de software “Space Invaders” para enseñar conceptos de diagrama de clases	Sí
	El ejercicio es interactivo	Sí

	El ejercicio usa partes del juego para explicar como funciona el diagrama de clases	Sí
	Las explicaciones del ejercicio son claras	Sí
	El ejercicio tiene un nivel de dificultad mayor que el anterior	Sí

Tabla 17 - Sprint 4 Review

- **Sprint 4 Retrospective**
    - **¿Qué salió bien?**

La reutilización de código agilizó bastante el proceso de desarrollo de este sprint.

- **¿Qué se puede mejorar?**

Al terminar este ejercicio el equipo de desarrollo se percató que no se estaba guardando los archivos del programa de una forma correcta, se procedió a reorganizar el sistema de archivos para mejorar su legibilidad.

4.3.5. Sprint 5

- **Sprint 5 Planning**

El objetivo será crear dos secciones teóricas, una para enseñar conceptos del "Ciclo de vida del software desarrollo del software", que son de importancia para entender por qué es necesario el diseño de software, y conceptos de "Abstracción", que es importante para crear buenos diseños de software.

Estas dos secciones teóricas tienen que venir acompañadas de una lección donde los estudiantes pondrán a prueba los conocimientos adquiridos. Para esto será necesario crear un nuevo sistema para rendir pruebas, este nuevo sistema también hará uso de sistemas creados anteriormente, específicamente del sistema de guardado local, pues aquí se guardará las calificaciones que obtengan en las pruebas.

- **Sprint 5 Backlog**

Sprint 5 Backlog	
Código	**Nombre**

T01	Crear una sección teórica sobre Ciclo de vida del desarrollo del software
T05	Crear una prueba para el Ciclo de vida del desarrollo del software
T02	Crear una sección teórica sobre Abstracción
T06	Crear una prueba para Abstracción

Tabla 18 - Sprint 5 Backlog

- **Ejecución del Sprint 5**

Para crear la sección teórica del Ciclo de Vida del desarrollo del Software (CVDS) primero se consultó con la dueña del producto el contenido que debería tener esta sección. Tras llegar a un acuerdo se buscó la bibliografía correcta y se procedió a crear en Unity las pantallas donde se pondría la información.

Se usó el “PanelManager”, visto anteriormente, para poder cambiar fácilmente entre distintos paneles informativos. También se usó el “Spline” para añadir movimiento en diferentes secciones de los menús y volverlas más llamativas. Los colores iniciales fueron muy templados, tras una reunión con la dueña del producto se decidió usar colores más llamativos para los menús.

Figura 56 - Primera pantalla de sección teórica de Ciclo de Cida del Software

Figura 57 - Segunda pantalla de sección teórica de Ciclo de Cida del Software

Después de realizar la sección de teoría se procedió a crear el sistema para rendir lecciones dentro de la aplicación. Primero se creó la clase “QuestionsAndAnswers”, esta funcionará como una interface para las preguntas y respuestas que tendrá el sistema. Primero necesita una pregunta, luego necesita una lista con cuatro posibles respuestas, luego una explicación en caso de que el estudiante responda mal y necesite una explicación más clara de por qué se equivocó, y, por último, un número entero que indica cuál elemento de la lista de respuestas es el correcto. Esta clase está en la Figura 58.

```
[System.Serializable]
1 reference
public class QuestionsAndAnswers
{
    public string Question;
    public string[] Answers;
    public string Correction;
    public int CorrectAnswer;
}
```

Figura 58 - "QuestionsAndAnwers" script

```
Unity Script (9 asset references) | 1 reference
public class TestManager : MonoBehaviour
{
    public List<QuestionsAndAnswers> QnA;
    public GameObject[] options;
    public int currentQuestion;

    public GameObject MainPanel;
    public GameObject GameOverPanel;
    public GameObject WrongAnswerPanel;

    public Text QuestionTxt;
    public Text ScoreTxt;
    public Text WrongAnswerTxt;

    public int whichTest;

    private int totalQuestions = 0;
    private int score;

    Unity Message | 0 references
    void Start()
    {
        totalQuestions = QnA.Count;
        MainPanel.SetActive(false);
        WrongAnswerPanel.SetActive(false);
        MainPanel.SetActive(true);
        GenerateQuestion();
    }

    0 references
    public void Retry()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
    1 reference
    public void GameOver()
    {
        MainPanel.SetActive(false);
        GameOverPanel.SetActive(true);
        float puntuacionFinal = Mathf.Round(((((float)score / (float)totalQuestions) * 100) * 100f) / 100f);

        if (whichTest == 0)
        {
            FindObjectOfType<Game>().califCVS = puntuacionFinal;
        }
        else if(whichTest == 1)
        {
            FindObjectOfType<Game>().califAbs = puntuacionFinal;
        }
        else if (whichTest == 2)
        {
            FindObjectOfType<Game>().califUML = puntuacionFinal;
        }
        else if (whichTest == 3)
        {
            FindObjectOfType<Game>().califDdC = puntuacionFinal;
        }

        FindObjectOfType<Game>().SaveGame();

        if (puntuacionFinal >= 70)
        {
            ScoreTxt.text = puntuacionFinal.ToString() + " %\n\n" + "Pasaste!";
            FindObjectOfType<SoundManagerScript>().Play("Success2");
        } else
        {
            ScoreTxt.text = puntuacionFinal.ToString() + " %\n\n" + "No Pasaste :C";
            FindObjectOfType<SoundManagerScript>().Play("Failure2");
        }

    }

    1 reference
    public void Correct()
    {
        score += 1;
        QnA.RemoveAt(currentQuestion);
        GenerateQuestion();
        FindObjectOfType<SoundManagerScript>().Play("Success");
    }
    1 reference
    public void Wrong()
    {
        WrongAnswerTxt.text = QnA[currentQuestion].Correction;
        WrongAnswerPanel.SetActive(true);
        FindObjectOfType<SoundManagerScript>().Play("Failure");
    }

    0 references
    public void WrongCheck()
    {
        WrongAnswerPanel.SetActive(false);
        QnA.RemoveAt(currentQuestion);
        GenerateQuestion();
        FindObjectOfType<SoundManagerScript>().Play("Step");
    }

    1 reference
    void SetAnswers()
    {
        for (int i = 0; i < options.Length; i++)
        {
            options[i].GetComponent<AnswerScript>().isCorrect = false;
            options[i].transform.GetChild(0).GetComponent<Text>().text = QnA[currentQuestion].Answers[i];
            if (QnA[currentQuestion].CorrectAnswer == i)
            {
                options[i].GetComponent<AnswerScript>().isCorrect = true;
            }
        }
    }

    3 references
    void GenerateQuestion()
    {
        if(QnA.Count > 0)
        {
            currentQuestion = Random.Range(0, QnA.Count);
            QuestionTxt.text = QnA[currentQuestion].Question;
            SetAnswers();
        }
        else
        {
            GameOver();
        }
    }
}
```

Figura 59 - "TestManager" script

Después tenemos el “TestManager” (Figura 59), este script es la esencia del sistema para hacer las pruebas, recibe una lista de preguntas en el formato definido en “QuestionsAndAnswers” (Figura 58) y las presenta al usuario una por una. El usuario puede ir respondiendo cada pregunta de selección múltiple, si responde mal una pregunta le muestra un mensaje de error donde le dice por qué está mal, si la respuesta es correcta lo deja avanzar a la siguiente pregunta. También va contando todas las preguntas correctas e incorrectas y cuando termina la prueba le entrega su nota, si tiene una puntuación igual o mayor al 70% pasa y sino reprueba, esta nota es guardada con el script “SaveSystem”, introducido anteriormente.

También tenemos el script “AnswerScript” (Figura 60), donde se muestran las preguntas al usuario, luego accede a “TestManager” y verificar si la pregunta fue respondida de forma correcta o incorrecta.

```
Unity Script (36 asset references) | 2 references
public class AnswerScript : MonoBehaviour
{
    public bool isCorrect = false;
    public TestManager testManager;

    0 references
    public void Answer()
    {
        if (isCorrect)
        {
            testManager.Correct();
        }
        else
        {
            testManager.Wrong();
        }
    }
}
```

Figura 60 - "AnswerScript" script

Una vez nuestros scripts estuvieron terminados ahora se pueden hacer las pruebas con una interfaz visual dentro del editor de Unity, como se ve en la Figura 61.

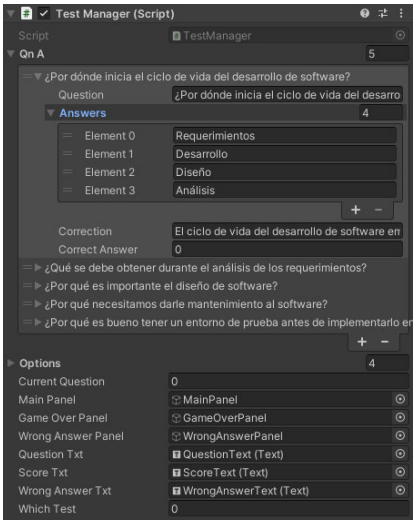


Figura 61 - Interfaz visual para hacer pruebas de “CVS” dentro de Unity

Una vez ingresas todas las pruebas a nuestro sistema para tomar lecciones se procedió a hacer la parte visual de las pruebas dentro del editor de Unity. En la Figura 62 se muestra cómo se ven estas pantallas cuando una prueba está en funcionamiento.

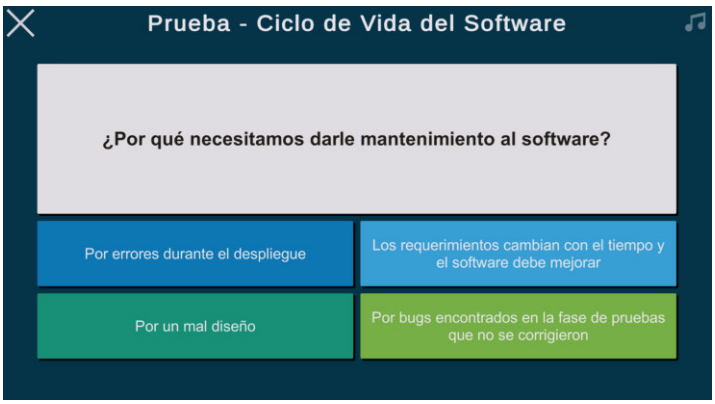


Figura 62 - Pantalla de pregunta para la prueba de "CVS"

En la Figura 63 se puede ver lo que pasa si se responde mal una pregunta, sale un mensaje de error con una corrección.

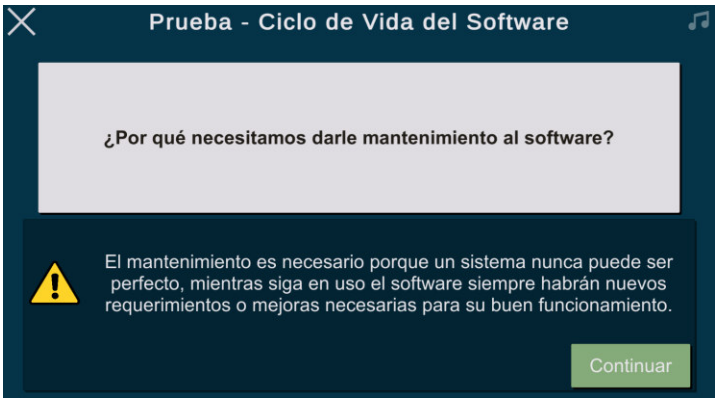


Figura 63 - Mensaje de error en prueba de "CVS"

La Figura 64 muestra la pantalla cuando el usuario termina la lección y tiene una nota suficiente para pasar.

Figura 64 - Pantalla final de la prueba de "CVS"

Ahora procedimos a desarrollar la sección teórica sobre Abstracción. Se siguieron los mismos pasos para la sección del Ciclo de Vida del Software, primero se habló con la dueña del producto, se eligieron las referencias bibliográficas y se procedió a la realización de las pantallas correspondientes en Unity. El resultado se puede ver en la Figura 65. Esta sección también reutilizó “PanelManager” y “Spline”.

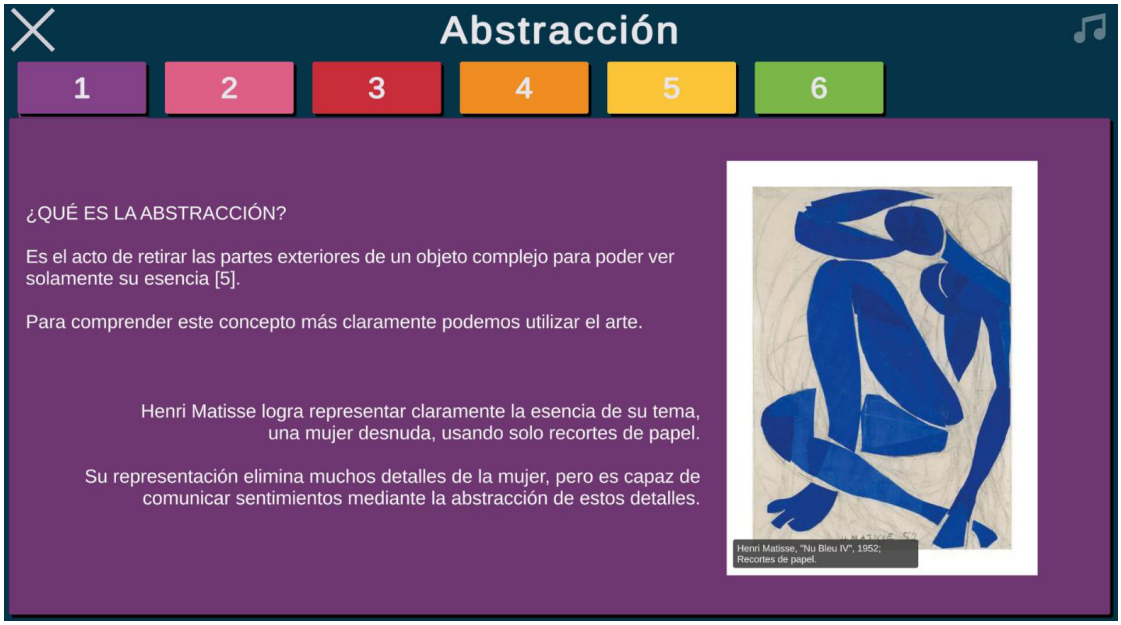


Figura 65 - Sección teórica de "Abstracción"

Posteriormente se desarrolló la prueba para la sección de Abstracción, como se ve en la Figura 66, reutilizando el sistema “TestManager”.

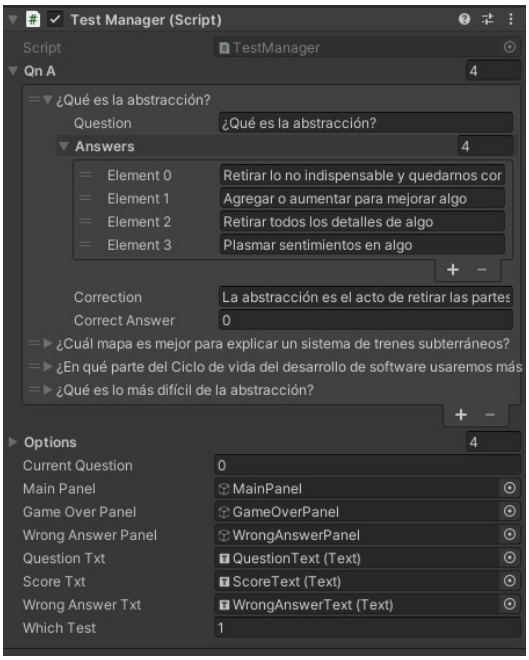


Figura 66 - Interfaz visual para hacer pruebas de “Abstracción”

En la figura 67 se muestra como quedaron las pruebas realizadas anteriormente, esta prueba también califica al estudiante, si saca más de 70% lo aprueba, sino lo reprueba, y usa “SaveSystem” para guardar la calificación localmente.

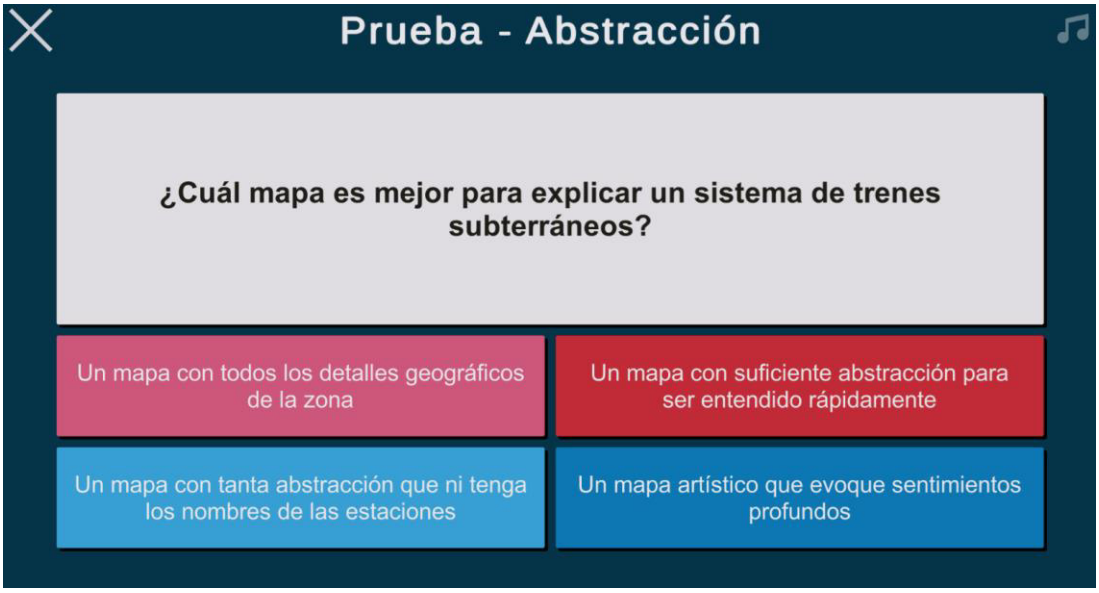


Figura 67 - Pantalla de pregunta para la prueba de "Abstracción"

- **Sprint 5 Review**

En este sprint se consiguió satisfactoriamente crear la sección teórica para Ciclo de vida del desarrollo del Software y para Abstracción, con sus respectivas pruebas, siempre adhiriéndose a las necesidades de la dueña del producto. Para el siguiente sprint se pidió realizar lo mismo, pero ahora enfocándonos en lo que es UML, primero conocimientos básicos y luego conocimientos avanzados.

Historia de Usuario	Criterio de Aceptación	Cumplido
T01 - Crear una sección teórica sobre Ciclo de vida del desarrollo del software	Se desarrolló una sección teórica para Ciclo de vida del desarrollo del software	Sí
	Las explicaciones en esta sección son claras	Sí
	La sección tiene referencias bibliográficas	Sí
T05 - Crear una prueba para el Ciclo de vida del desarrollo del software	Se desarrolló un sistema para tomar pruebas para el Ciclo de vida del desarrollo del software	Sí
	El sistema de pruebas da retroalimentación cuando se comete errores	Sí

	El sistema de pruebas da la nota final	Sí
	El sistema de pruebas guarda la nota de la prueba	Sí
T02 - Crear una sección teórica sobre Abstracción	Se desarrolló una sección teórica para Abstracción	Sí
	Las explicaciones en esta sección son claras	Sí
	La sección tiene referencias bibliográficas	Sí
T06 - Crear una prueba para Abstracción	Se usó el sistema de pruebas para crear una prueba sobre Abstracción	Sí
	El sistema de pruebas da retroalimentación cuando se comete errores	Sí
	El sistema de pruebas da la nota final	Sí
	El sistema de pruebas guarda la nota de la prueba	Sí

Tabla 19 - Sprint 5 Review

- **Sprint 5 Retrospective**
    - **¿Qué salió bien?**

Durante el desarrollo del sprint la continua interacción con la dueña del producto permitió avanzar y corregir errores a tiempo.

- **¿Qué se puede mejorar?**

Se decidió que los colores iniciales para las pantallas de teoría eran muy apagados, se corrigió usando colores más llamativos. Se debe seguir el uso de los mismos colores llamativos en las siguientes secciones teóricas. La sección de pruebas inicial tampoco tenía una forma de corregir al estudiante cuando se equivocaba, se añadió un mensaje para aclarar el por qué el estudiante se equivocó.

4.3.6. Sprint 6

- **Sprint 6 Planning**

En el Sprint 6 se busca crear dos secciones teóricas, una para enseñar conceptos del Lenguaje de Modelado Unificado (UML) básicos, y otra con conceptos más

específicos sobre el Diagrama de clases UML. Cada sección con sus respectivas lecciones para poner a prueba los conocimientos de los estudiantes.

- **Sprint 6 Backlog**

Sprint 6 Backlog	
Código	**Nombre**
T03	Crear una sección teórica sobre UML
T07	Crear una prueba para UML
T04	Crear una sección teórica sobre el Diagrama de Clases UML
T08	Crear una prueba para el Diagrama de Clases UML

Tabla 20 - Sprint 6 Backlog

- **Ejecución del Sprint 6**

Como en el Sprint 5, primeramente, se habló con la dueña del producto para conocer lo que se debería enseñar a los estudiantes en cada sección, luego se consiguió la bibliografía correcta y luego se procedió a crear las pantallas correspondientes. La Figura 68 muestra una pantalla de la sección teórica sobre UML.

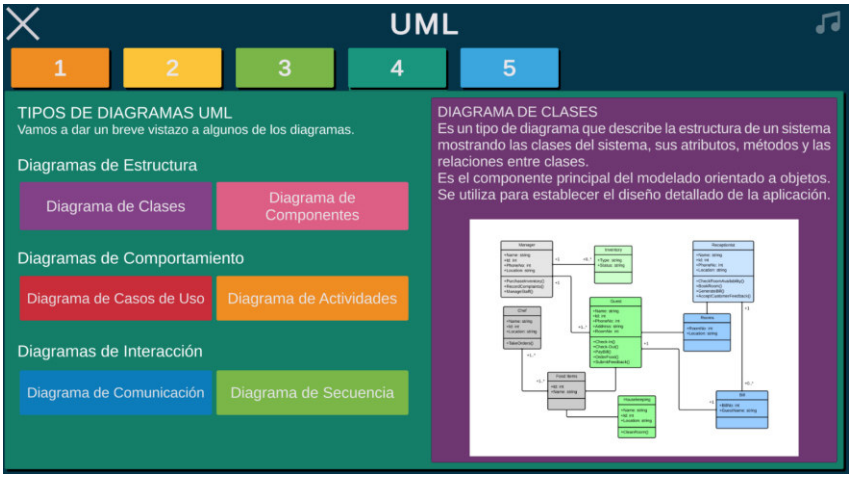


Figura 68 - Sección teórica de "UML"

Se reutilizó código creado anteriormente para crear estas pantallas, como es el “PanelManager” y “Spline”.

Luego de terminar la primera sección se procedió a crear su respectiva prueba, igualmente se reutilizó el código de “TestManager” para hacerlo. Esta nueva prueba puede ser vista en la Figura 69.

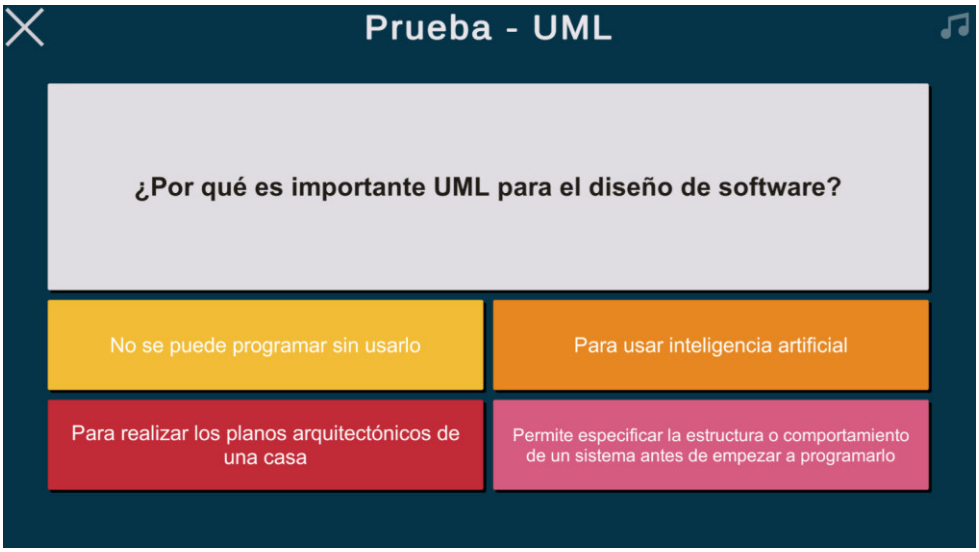


Figura 69 - Pantalla de pregunta para la prueba de "UML"

Hay que recordar que cada prueba usa el sistema de guardado o “SaveSystem” para guardar la calificación que reciben los estudiantes.

Posteriormente se realizó la parte teórica del “Diagrama de Clases UML”, como se muestra en la Figura 70. Esta sección también reusa todos los sistemas que la sección teórica de UML.

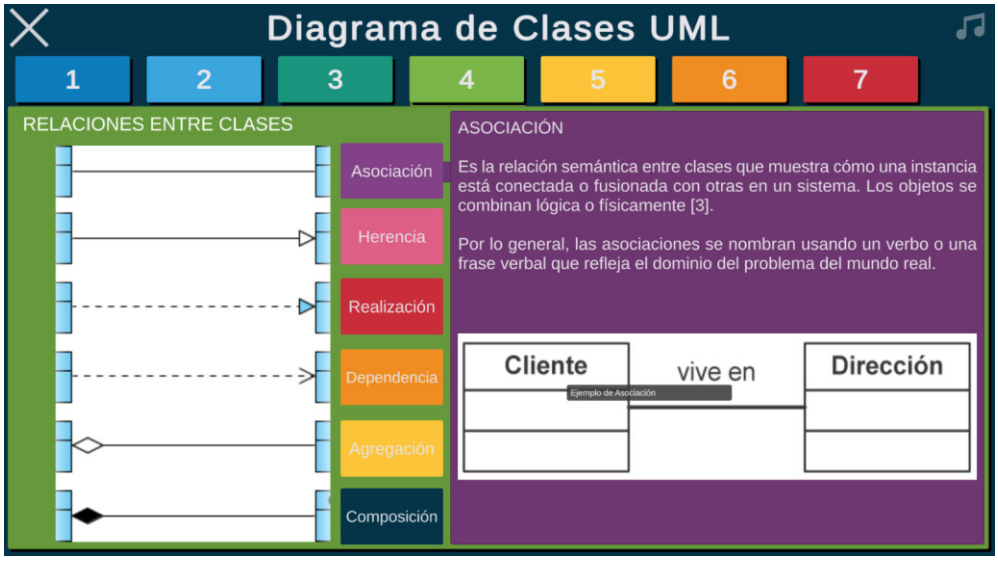


Figura 70 - Sección teórica de "Diagrama de Clases UML"

Y para terminar el sprint se creó la prueba de la sección teórica “Diagramas de Clases UML”, como se puede observar en la Figura 71, volviendo a usar los sistemas “TestManager” y “SaveSystem” explicados anteriormente.

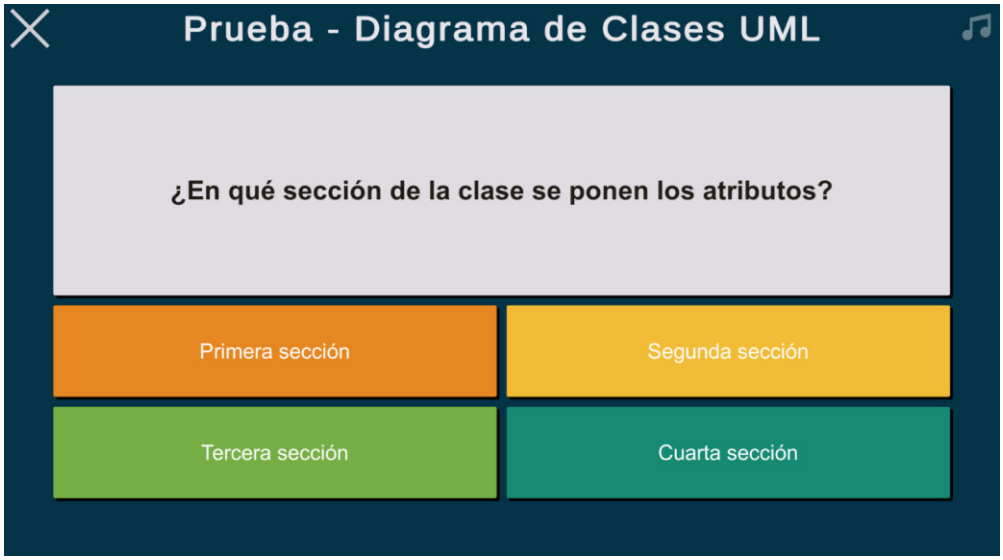


Figura 71 - Pantalla de pregunta para la prueba de "Diagrama de Clases UML"

- **Sprint 6 Review**

En este sprint se consiguió satisfactoriamente crear la sección teórica para UML y para el Diagrama de Clases UML, con sus respectivas pruebas, siempre buscando cumplir con a las necesidades de la dueña del producto. En este sprint no se crearon nuevos sistemas, solo se reusaron sistemas creados anteriormente lo que facilitó el trabajo.

Historia de Usuario	Criterio de Aceptación	Cumplido
T03 - Crear una sección teórica sobre UML	Se desarrolló una sección teórica para UML	Sí
	Las explicaciones en esta sección son claras	Sí
	La sección tiene referencias bibliográficas	Sí
T07 - Crear una prueba para UML	Se usó el sistema de pruebas para crear una prueba sobre UML	Sí
	El sistema de pruebas da retroalimentación cuando se comete errores	Sí
	El sistema de pruebas da la nota final	Sí

	El sistema de pruebas guarda la nota de la prueba	Sí
T04 - Crear una sección teórica sobre el Diagrama de Clases UML	Se desarrolló una sección teórica para Diagrama de Clases UML	Sí
	Las explicaciones en esta sección son claras	Sí
	La sección tiene referencias bibliográficas	Sí
T08 - Crear una prueba para el Diagrama de Clases UML	Se usó el sistema de pruebas para crear una prueba sobre Diagrama de Clases UML	Sí
	El sistema de pruebas da retroalimentación cuando se comete errores	Sí
	El sistema de pruebas da la nota final	Sí
	El sistema de pruebas guarda la nota de la prueba	Sí

Tabla 21 - Sprint 6 Review

- **Sprint 6 Retrospective**
    - **¿Qué salió bien?**

La reutilización de código y que ya se definió con anterioridad los colores para las pantallas facilitó enormemente la realización de este sprint.

- **¿Qué se puede mejorar?**

Existieron aspectos visuales que no estaban al gusto de la dueña del producto y tuvieron que ser mejorados. Se necesita de una mejor conexión entre todas las pantallas, por eso en el siguiente sprint se creará el menú principal y los submenús que conectan a las distintas partes de la aplicación que ya se encuentran en funcionamiento.

4.3.7. Sprint 7

- **Sprint 7 Planning**

En esta sección final se necesita crear el menú principal y los submenús con los que se conectan todas las otras partes de la aplicación. También es necesario tener

una sección de información donde los usuarios tienen acceso a la información bibliográfica de la parte teórica de la aplicación. Para agregar mayor interactividad se decidió crear un sistema de sonido para los menús de la aplicación. Por último, es necesario hospedar la aplicación en un servidor web en donde el público en general tenga acceso a ella.

- **Sprint 7 Backlog**

Sprint 7 Backlog	
Código	**Nombre**
T18	Crear una sección de información
T19	Popular la sección de información con las referencias bibliográficas usadas en la sección teórica, las notas conseguidas en las pruebas y datos del progreso en la sección de ejercicios
T20	Desarrollar un sistema de sonido para los menús de la aplicación
T21	Crear los menús que conectan a todas las secciones de la aplicación
T22	Subir la aplicación al internet para que sea de acceso público

Tabla 22 - Sprint 7 Backlog

- **Ejecución del Sprint 7**

Primeramente, se creó la pantalla de información, esta pantalla tendrá 3 secciones, la primera sección se llamará “Progreso” y guardará las notas sacadas en las pruebas, la segunda sección “Información del Proyecto” tiene la información de los creadores de la aplicación, y, finalmente, la tercera sección “Referencias” tendrá la información bibliográfica de los documentos usados para crear la sección teórica de esta aplicación y de las librerías gratuitas usadas para programar la aplicación.

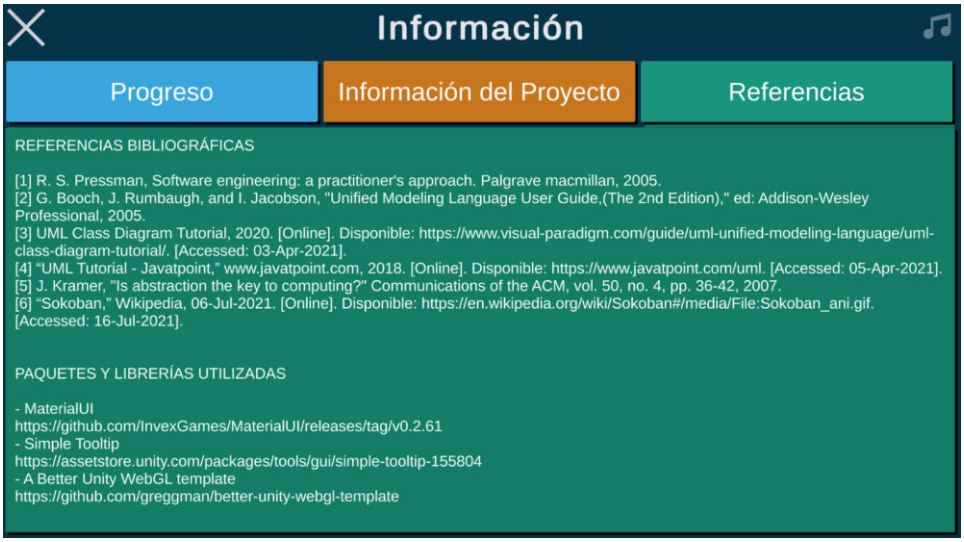


Figura 72 - Pantalla de "Información"

Posteriormente, se empezó a crear el sistema para tener sonido en los menús de la aplicación. El script “Sound” es una clase que guarda la información que debería contener cada sonido.

```
[System.Serializable]
5 references
public class Sound
{
    public string name;

    public AudioClip clip;

    [Range(0f, 1f)]
    public float volume = 0.5f;
    [Range(0.1f, 3f)]
    public float pitch = 1f;

    public bool loop;

    [HideInInspector]
    public AudioSource source;
}
```

Figura 73 - "Sound" script

El script “SoundManager” será el encargado de guardar los archivos de sonido y reproducirlos cuando son llamados por los distintos botones y pantallas de la aplicación. Tiene métodos para reproducir sonidos, para pausarlos y para detectar si un sonido está siendo reproducido, estos métodos serán usados de diferentes formas por diferentes partes de la aplicación. Este script se puede ver en la Figura 74.

```
Unity Script (1 asset reference) | 35 references
public class SoundManagerScript : MonoBehaviour
{
    public Sound[] sounds;

    public static SoundManagerScript instance;

    Unity Message | 0 references
    private void Awake()
    {
        if (instance == null)
            instance = this;
        else
        {
            Destroy(gameObject);
            return;
        }

        DontDestroyOnLoad(gameObject);

        foreach (Sound s in sounds)
        {
            s.source = gameObject.AddComponent<AudioSource>();
            s.source.clip = s.clip;
            s.source.volume = s.volume;
            s.source.pitch = s.pitch;
            s.source.loop = s.loop;
        }
    }

    Unity Message | 0 references
    private void Start()
    {
        Play("Theme");
    }

    33 references
    public void Play (string name)
    {
        Sound s = Array.Find(sounds, sound => sound.name == name);
        if (s == null)
        {
            Debug.LogWarning("Sound: " + name + " not found!");
            return;
        }
        s.source.Play();
    }
    1 reference
    public void Pause(string name)
    {
        Sound s = Array.Find(sounds, sound => sound.name == name);
        if (s == null)
        {
            Debug.LogWarning("Sound: " + name + " not found!");
            return;
        }
        s.source.Pause();
    }

    1 reference
    public bool CheckIfPlaying(string name)
    {
        Sound s = Array.Find(sounds, sound => sound.name == name);
        if (s == null)
        {
            Debug.LogWarning("Sound: " + name + " not found!");
        }
        return s.source.isPlaying;
    }
}
```

Figura 74 - "SoundManager" script

Procedimos a conseguir sonidos con licencias de uso libre para ponerlos en la aplicación, el uso de “SoundManager” con estos archivos de audio se ve en la Figura 75.

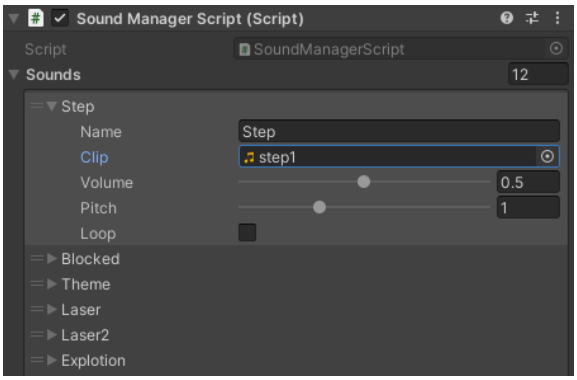


Figura 75 - Uso de "SoundManager"

La dueña del producto solicitó que adicionalmente exista una canción en el fondo al usar la aplicación, esta canción puede ser pausada y reproducida desde cualquier pantalla de la aplicación, para lograr esto se creó un nuevo script, como se ve en la Figura 76.

```
Unity Script (1 asset reference) | 0 references
public class SoundOff : MonoBehaviour
{

    public GameObject button1;
    public GameObject button2;

    private bool isItPlaying;

    Unity Message | 0 references
    private void Start()
    {
        Invoke("AutoPlay", 0.01f);
    }

    0 references
    private void AutoPlay()
    {
        isItPlaying = FindObjectOfType<SoundManagerScript>().CheckIfPlaying("Theme");
        if (isItPlaying)
        {
            button1.SetActive(true);
            button2.SetActive(false);
        }
        else if (!isItPlaying)
        {
            button1.SetActive(false);
            button2.SetActive(true);
        }
    }

    0 references
    public void PauseMusic()
    {
        FindObjectOfType<SoundManagerScript>().Pause("Theme");
        button1.SetActive(false);
        button2.SetActive(true);
    }

    0 references
    public void PlayMusic()
    {
        FindObjectOfType<SoundManagerScript>().Play("Theme");
        button2.SetActive(false);
        button1.SetActive(true);
    }
}
```

Figura 76 - "SoundOff" script

Adicionalmente se crearon dos botones para que “SoundOff” pueda funcionar en toda la aplicación, prendiendo y apagando la música de fondo, estos botones se ven en la Figura 77.

Figura 77 - Botones para prender o apagar música

Al tener todos los sistemas de la aplicación terminados procedimos a crear los menús y submenús que interconectan a todas las pantallas de la aplicación creadas anteriormente, estos reusaron código y diseños de pantallas y botones ya creados para otras pantallas, lo que facilitó mucho el trabajo.

Figura 78 - Menú principal

Tras tener la aplicación completa, con todas sus pantallas, pruebas, juegos de video y ejercicios de diseño, solo nos hacía falta subirla al internet. El equipo de desarrollo descubrió que GitHub ofrece un servicio de hosting gratuito para páginas web y está integrado dentro de su sistema para repositorios Git. “GitHub Pages” es un servicio de alojamiento de sitios estáticos que toma archivos HTML, CSS y JavaScript directamente de un repositorio en GitHub, opcionalmente ejecuta los archivos a través de un proceso de compilación y publica un sitio web [60].

El equipo de desarrolló procedió a seguir el proceso para finalizar la aplicación en Unity como se ve en la Figura 79.

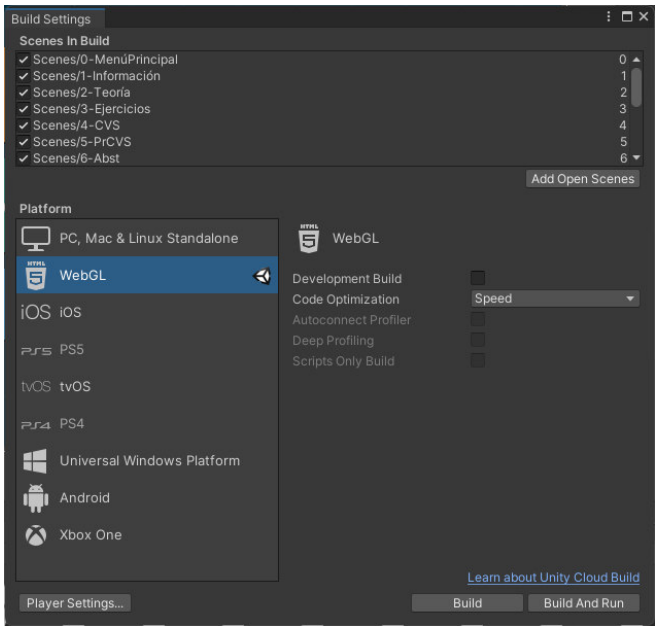


Figura 79 - Pantalla para finalizar la aplicación en Unity

Cuando la aplicación terminó de ser construida se procedió a subirla al repositorio de GitHub, activar GitHub Pages y ahora la aplicación está habilitada para ser usado por el público general. La aplicación final puede ser utilizada en el link del Anexo 2.

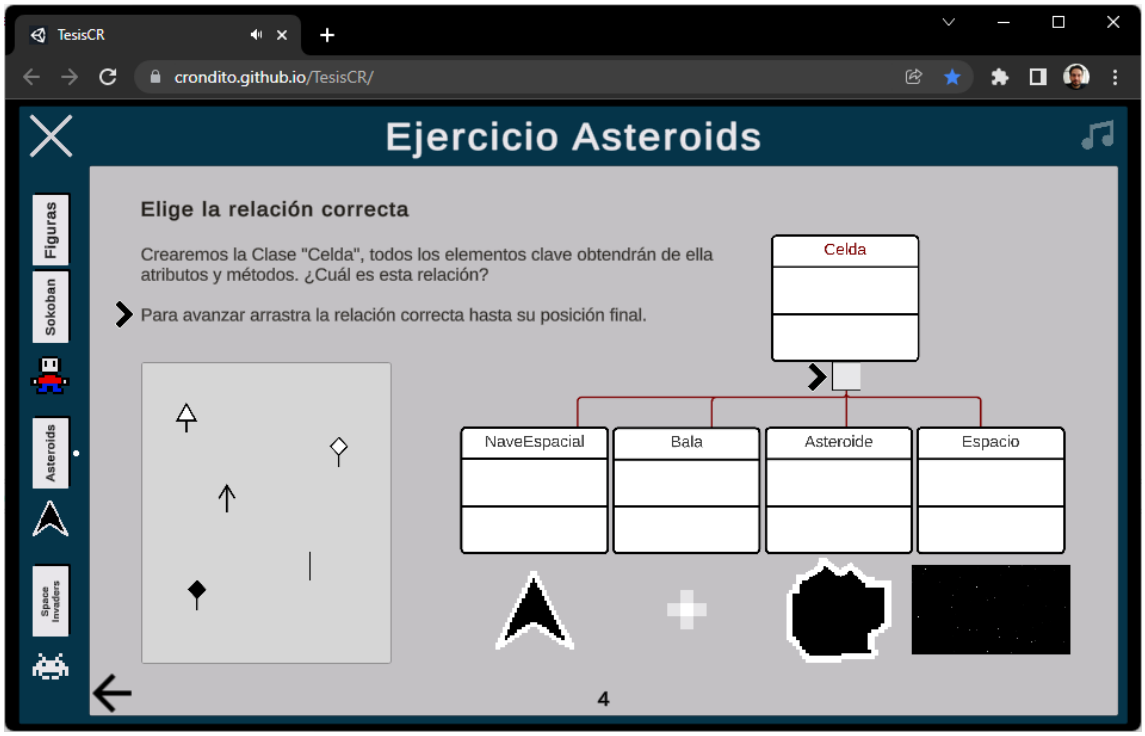


Figura 80 - Aplicación corriendo en un navegador web

- **Sprint 7 Review**

Se logró finalizar con todos los objetivos planteados, creando nuevos sistemas, nuevas pantallas y subiendo la aplicación a un hospedaje web gratuito.

Historia de Usuario	Criterio de Aceptación	Cumplido
T18 - Crear una sección de información	Se creó una sección de información	Sí
	La sección teórica tiene una subsección para mostrar el Progreso en las pruebas y en los ejercicios	Sí
	La sección teórica tiene un subsección para mostrar Información de los desarrolladores del proyecto	Sí
	La sección teórica tiene un subsección para mostrar las referencias bibliográficas utilizadas	Sí

T19 - Popular la sección de información con las referencias bibliográficas usadas en la sección teórica, las notas conseguidas en las pruebas y datos del progreso en la sección de ejercicios	Se puso la información de referencias bibliográficas en la sección de información	Sí
	Se puso la información sobre los desarrolladores de la aplicación en la sección de información	Sí
	La sección de información muestra de forma automática las notas obtenidas en las pruebas teóricas	Sí
	La sección de información muestra automáticamente los ejercicios que se han completado y los que no	Sí
T20 - Desarrollar un sistema de sonido para los menús de la aplicación	Se desarrolló un sistema para tener sonido en los menús y submenús de la aplicación	Sí
	Se implementó el sistema de sonido en todos los menús y submenús de la aplicación	Sí
T21 - Crear los menús que conectan a todas las secciones de la aplicación	Se creó el menú principal	Sí
	Se crearon los submenús que interconectan las diferentes partes de la aplicación	Sí
T22 - Subir la aplicación al internet para que sea de acceso público	Se usó el servicio “GitHub Pages” para usar su hosting gratuito de la aplicación en GitHub	Sí
	Se finalizó la aplicación en Unity	Sí
	Se subió la aplicación al internet y es de acceso público	Sí

Tabla 23 - Sprint 7 Review

- **Sprint 7 Retrospective**
    - **¿Qué salió bien?**

La reutilización de código a lo largo del sprint facilitó mucho el proceso de desarrollo. La herramienta GitHub Pages también facilitó bastante el tema de hospedaje web al usar nuestro repositorio ya creado.

- **¿Qué se puede mejorar?**

El equipo de desarrollo pudo tener una mejor organización de su tiempo para evitar retrasos a lo largo del sprint.

4.3.8. Pruebas de funcionalidad

Las pruebas de funcionalidad tienen el objetivo principal de revisar que el software funciona para lo que fue creado. Son esenciales debido a que permiten comprobar que los requisitos del Product Backlog han sido implementados correctamente. Pressman [2] explica que hay dos formas de realizar pruebas de funcionalidad en el software:

- Al conocer la función específica que se asignó a un producto para su realización, pueden llevarse a cabo pruebas que demuestren que cada función es completamente operativa [2]. Estas pruebas también son llamadas de *"caja negra".*
- Al conocer el funcionamiento interno de un producto, pueden realizarse pruebas para garantizar que las operaciones internas se realizan de acuerdo con las especificaciones del producto [2]. Estas pruebas también son llamadas de *"caja blanca".*

Por la naturaleza de esta aplicación se decidió optar por las pruebas de "caja negra" pues era necesario comprobar que cada uno de los requerimientos de la dueña del producto haya sido cumplidos, sin la necesidad de hacer pruebas internas a cada sección de la aplicación web. Según Pressman [2] una prueba de caja negra examina algunos aspectos fundamentales de un sistema con poca preocupación por la estructura lógica interna del software.

Resultados de las pruebas de funcionalidad

Las pruebas funcionales fueron realizadas con la dueña del producto al finalizar cada sprint a lo largo del desarrollo de la aplicación, sus resultados específicos se muestran en cada Sprint Review de la sección 4.3.1. hasta la sección 4.3.7.

A continuación, se presenta una tabla con el resumen de los resultados de cumplimiento de cada tarea de las historias de usuario.

	Código	Nombre	Cumplido
US01	**T01**	Crear una sección teórica sobre Ciclo de vida del desarrollo del software	Sí
	T02	Crear una sección teórica sobre Abstracción	Sí
	T03	Crear una sección teórica sobre UML	Sí
	T04	Crear una sección teórica sobre el Diagrama de Clases UML	Sí
US02	**T05**	Crear una prueba para el Ciclo de vida del desarrollo del software	Sí
	T06	Crear una prueba para Abstracción	Sí
	T07	Crear una prueba para UML	Sí
	T08	Crear una prueba para el Diagrama de Clases UML	Sí
US03	**T09**	Desarrollo del sistema de guardado	Sí
	T10	Desarrollo del sistema de niveles	Sí
	T11	Desarrollar el ejercicio del diagrama de clases “Figuras”	Sí
	T12	Desarrollar el ejercicio del diagrama de clases “Sokoban”	Sí
	T13	Desarrollar el ejercicio del diagrama de clases “Asteroids”	Sí
	T14	Desarrollar el ejercicio del diagrama de clases “Space Invaders”	Sí
US04	**T15**	Desarrollar el videojuego “Sokoban”	Sí
	T16	Desarrollar el videojuego “Asteroids”	Sí
	T17	Desarrollar el videojuego “Space Invaders”	Sí
US05	**T18**	Crear una sección de información	Sí
	T19	Popular la sección de información con las referencias bibliográficas usadas en la	Sí

		sección teórica, las notas conseguidas en las pruebas y datos del progreso en la sección de ejercicios	
US06	**T20**	Desarrollar un sistema de sonido para los menús de la aplicación	Sí
	T21	Crear los menús que conectan a todas las secciones de la aplicación	Sí
US07	**T22**	Subir la aplicación al internet para que sea de acceso público	Sí

Tabla 24 - Cumplimiento tareas de historias de usuario

4.3.9. Pruebas de usabilidad

Para evaluar la usabilidad de la aplicación web se utilizó el cuestionario de la Escala de Usabilidad del Sistema (SUS por sus siglas en inglés: System Usability Scale). Esta herramienta metodológica ha demostrado ser muy efectiva al ser utilizada para medir la usabilidad de un objeto, un dispositivo o una aplicación [61, 62].

Para realizar la prueba primeramente se presentó la aplicación web a los encuestados y posteriormente se les pidió llenar el cuestionario SUS. Este cuestionario está compuesto por diez preguntas, cada pregunta tiene un valor del 1 al 5, siendo 1 *"en total desacuerdo"* y 5 *"en total acuerdo"* [62]. Las preguntas pueden ser revisadas en la siguiente tabla.

Preguntas SUS	
Número	**Pregunta**
1	Creo que usaría esta aplicación frecuentemente
2	Encuentro esta aplicación innecesariamente compleja
3	Creo que la aplicación fue fácil de usar
4	Creo que necesitaría ayuda de una persona con conocimientos técnicos para usar esta aplicación
5	Las funciones de esta aplicación están bien integradas
6	Creo que la aplicación es muy inconsistente

7	Imagino que la mayoría de la gente aprendería a usar esta aplicación en forma rápida
8	Encuentro que la aplicación es muy difícil de usar
9	Me siento confiado al usar esta aplicación
10	Necesité aprender muchas cosas antes de ser capaz de usar esta aplicación

Tabla 25 - Preguntas SUS

Para obtener los resultados de la evaluación SUS hay que sumar los resultados promediados obtenidos de los cuestionarios realizados a nuestros usuarios, considerando lo siguiente [62]:

- Las preguntas impares (1, 3, 5, 7 y 9) tomarán el valor asignado por el usuario, se sumarán todas y luego se restará 5.
- Para las preguntas pares (2, 4, 6, 8, 10), el puntaje será 25 menos la suma del valor asignado por nuestros entrevistados.
- Se suman los dos valores anteriores y se multiplica por 2,5.

El resultado final es sobre 100, una calificación final superior a 68 es considerada por encima del promedio, mientras que cualquier calificación inferior a 68 es considerada inferior al promedio [61].

Adicionalmente, se hicieron dos preguntas extra para conocer las opiniones de los usuarios. La primera pregunta es un simple sí o no, mientras que la segunda solicita una opinión más directa sobre cómo se podría mejorar la aplicación en el futuro. Estas preguntas extra son:

- ¿Usaría la aplicación por su cuenta, en su tiempo libre, para desbloquear el resto de los juegos de video?
- Por favor comparta su opinión, ¿Cómo podría mejorar la aplicación?

Resultados de las pruebas de usabilidad

El Anexo 4 contiene un enlace a los resultados completos en una table de Excel. A continuación, presentamos un análisis de estos resultados.

- *Escala de Usabilidad del Sistema (SUS)*

Se realizó la prueba SUS a un total de veinte voluntarios, entre ellos constan estudiantes actuales (al momento de realizar este documento) y ex estudiantes de la Facultad de Ingeniería de Sistemas de la Escuela Politécnica Nacional, como también personas del público general. La siguiente tabla tiene los resultados obtenidos, las columnas representan las preguntas de la Tabla 24, mientras que las filas indican los encuestados.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
E1	5	5	5	5	5	5	5	5	5	5
E2	5	1	5	1	5	1	5	2	5	2
E3	5	1	5	1	5	1	5	1	5	3
E4	4	1	5	2	4	4	4	1	4	1
E5	5	2	5	4	5	1	5	1	5	2
E6	4	1	5	3	5	1	4	1	5	2
E7	4	1	5	2	4	1	5	1	5	1
E8	4	1	5	1	5	1	5	1	5	2
E9	4	1	5	1	4	1	4	1	5	1
E10	4	1	5	1	5	1	5	1	4	2
E11	3	1	5	1	5	1	5	5	5	1
E12	3	1	4	1	4	1	5	1	4	1
E13	3	1	4	1	5	1	5	1	5	1
E14	2	1	5	1	4	1	5	5	5	1
E15	4	1	5	1	5	1	5	1	5	1
E16	4	2	5	5	4	4	5	2	3	1
E17	5	1	5	1	5	1	5	1	4	1
E18	3	1	5	1	5	1	5	5	5	1
E19	4	2	5	1	5	1	5	2	5	1
E20	4	4	5	2	4	2	4	2	4	1
Promedio	3.95	1.5	4.9	1.8	4.65	1.55	4.8	2	4.65	1.55

Tabla 26 - Resultados encuesta SUS

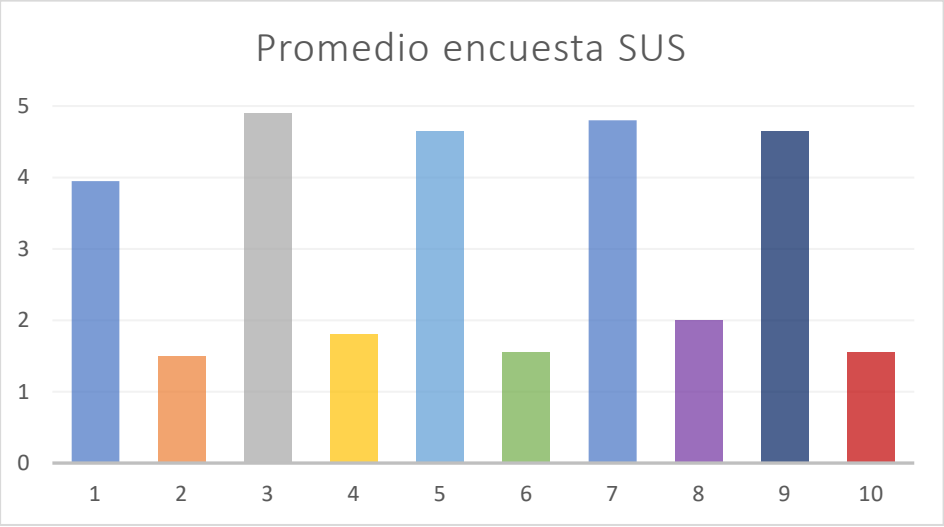


Figura 81 - Promedio encuesta SUS

El puntaje final obtenido con la metodología SUS [61] es de 86.4 puntos sobre 100. Este resultado está por encima del promedio lo que indica que la satisfacción de los usuarios es alta.

- *Preguntas adicionales*

Existieron dos preguntas adicionales como se explicó anteriormente.

La primera pregunta adicional fue: *¿Usaría la aplicación por su cuenta, en su tiempo libre, para desbloquear el resto de los juegos de video?*

El 100% de los encuestados respondieron que sí, otra vez demostrando que la satisfacción por parte de los usuarios es bastante amplia, y que la estrategia de la aplicación por usar juegos de video e interacciones entretenidas logró capturar la atención de los usuarios de forma positiva.

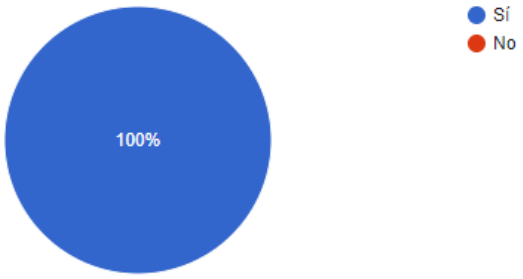


Figura 82 - Resultado primera pregunta extra

La segunda pregunta adicional fue: *Por favor comparta su opinión ¿Cómo podría mejorar la aplicación?*

Al ser una pregunta de estilo abierto se obtuvieron muchos tipos de réplicas. Las respuestas completas pueden ser vistas en el Anexo 4. La siguiente tabla muestra un resumen de las sugerencias de los participantes, junto con el número de participantes que sugirieron cada mejora, desde la sugerencia con más votos hasta las que tienen menos votos.

¿Cómo podría mejorar la aplicación?	
Respuesta	**Número**
Mejorar la interfaz gráfica	7
Agregar más ejercicios y juegos	4
Mejorar los ejercicios	4
Instrucciones más claras	3
Mejorar música y sonidos	2

Agregar información sobre patrones de diseño	2
Agregar videos explicativos	1
Links a páginas para obtener más información	1
Mejorar colores	1
Explicaciones teóricas más largas	1
Agregar pseudocódigo	1
Conectar mejor la parte teórica con la práctica	1

Tabla 27 - ¿Cómo podría mejorar la aplicación?

Como podemos observar, las mejoras más solicitadas tienen que ver con optimizar la interfaz gráfica de la aplicación, seguido con la petición de agregar más ejercicios y juegos de video, mejorar los ejercicios ya existentes, haciéndolos más interactivos, y finalmente, dar instrucciones más claras para completar dichos ejercicios.

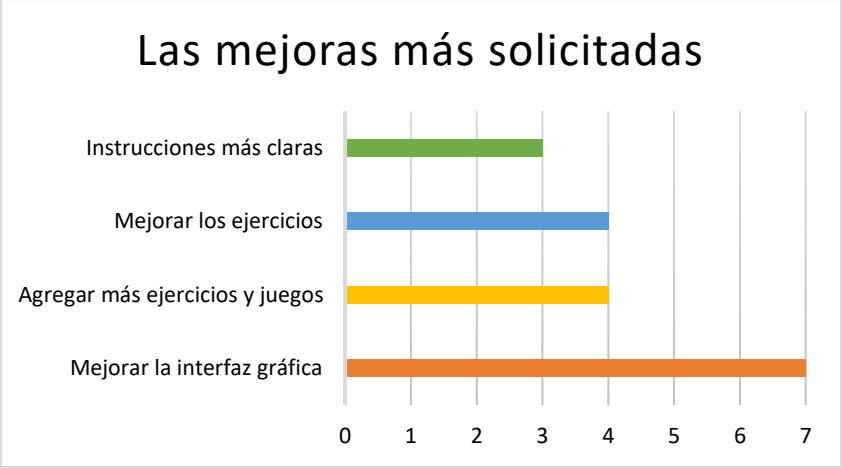


Figura 83 - Mejoras solicitadas

CAPÍTULO 5 – CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

- Con este proyecto integrador se cumplieron los objetivos propuestos pues logramos desarrollar efectivamente una aplicación web para el aprendizaje de diseño de software orientado a objetos con enfoque en abstracción, empleando efectivamente la plataforma de desarrollo Unity y el marco de trabajo Scrum.
- La aplicación web desarrollada usa tecnologías digitales modernas para la enseñanza de conceptos importantes sobre la ingeniería de software. Las pruebas de usabilidad realizadas demuestran el interés por parte del público general en el uso de estas tecnologías en la educación, pues logran enseñar asignaturas con bastante teoría, de una forma entretenida y atractiva para las nuevas generaciones de estudiantes.
- A lo largo del proyecto se definieron, por el equipo de desarrollo junto con la dueña del producto, requisitos funcionales y no funcionales. Estos requerimientos fueron revisados, cambiados cuando fue necesario, y aprobados por la dueña del producto, demostrando el poder de las metodologías ágiles para crear software de una manera no rígida, que se adapta a las necesidades del cliente, especialmente si los requisitos del proyecto no son completamente claros al empezarlo.
- Después de usar la arquitectura Entidad-Componente-Sistema (ECS) en este proyecto integrador, concluimos que es una arquitectura orientada a separar a los datos del comportamiento de una forma bastante sencilla, lo que facilita mucho el prototipado de diferentes funciones a lo largo del desarrollo. Esto es porque es una arquitectura pensada para la creación de videojuegos, que requieren de mucho prototipado durante su desarrollo.
- El uso de la metodología Scrum para la creación de esta aplicación web permitió que sus procesos de desarrollo e implementación se lleven a cabo de manera ágil, que la comunicación con la dueña del producto sea constante y efectiva, que se mantenga un enfoque firme en la mejora de

procesos, y que, sobre todas las cosas, se entregue un producto de calidad, que cumpla con las expectativas del cliente.

- Por último, se comprobó que los estudiantes universitarios actuales están abiertos a probar nuevas herramientas digitales de aprendizaje, que están conectados al internet, a las nuevas tecnologías, y esperan que la educación evolucione junto a ellos.

5.2. Recomendaciones

- Se recomienda en proyectos futuros seguir estudiando el uso de herramientas digitales interactivas para la educación, no solo en temas relacionados a la ingeniería de software, sino también en toda clase de asignatura o materia.
- Si bien se lograron los objetivos planificados para este proyecto integrador, se podría mejorar la aplicación con interfaces gráficas más claras, utilizando paletas de colores más llamativas, o enfocándose en hacer más y mejores ejercicios de diseño.
- Algunos estudiantes mostraron interés en tener acceso a una herramienta parecida a la creada en este proyecto, pero que se enfoque en patrones de diseño, un nuevo proyecto con este tema sería una expansión interesante para el concepto de esta aplicación web.
- Se recomienda que, si se usa el motor gráfico Unity para esta clase de proyectos en el futuro, ser muy organizados en la forma en la que se mantienen los archivos del programa, ya que es fácil perderse en toda la cantidad de archivos que se tienen que manejar.
- Es muy importante familiarizarse y utilizar un sistema de versionamiento como Git desde el principio del desarrollo de cualquier tipo de software, especialmente uno de tamaño grande como este proyecto. Usar buen versionamiento salvó varias veces al equipo de desarrollo de perder información valiosa por errores comunes.

REFERENCIAS BIBLIOGRÁFICAS

[1] N. R. Mead, J. H. Allen, S. Barnum, R. J. Ellison, and G. R. McGraw, *Software security engineering: a guide for project managers*. Addison-Wesley Professional, 2004.

[2] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave macmillan, 2005.

[3] P. Flores, J. Torres, and R. Fonseca-Delgado, "Design decisions under object-oriented approach: A thematic analysis from the abstraction point of view," in *Proceedings of the 8th Computer Science Education Research Conference*, 2019.

[4] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Connallen, and K. A. Houston, "Object-oriented analysis and design with applications," *ACM SIGSOFT software engineering notes,* vol. 33, no. 5, pp. 29-29, 2008.

[5] K. Nørmark, L. L. Thomsen, and B. Thomsen, "Object-oriented programming with gradual abstraction," *ACM SIGPLAN Notices,* vol. 48, no. 2, pp. 41-52, 2012.

[6] D. J. Armstrong, "The quarks of object-oriented development," *Communications of the ACM,* vol. 49, no. 2, pp. 123-128, 2006.

[7] A. F. Blackwell, L. Church, and T. R. Green, "The Abstract is an Enemy: Alternative Perspectives to Computational Thinking," in *PPIG*, 2008, p. 5.

[8] B. Meyer, "Software engineering in the academy," *Computer,* vol. 34, no. 5, pp. 28-35, 2001.

[9] I. Hadar, "When intuition and logic clash: The case of the object-oriented paradigm," *Science of Computer Programming,* vol. 78, no. 9, pp. 1407-1426, 2013.

[10] A. Olier, A. Gómez, and M. Caro, "Design and Implementation of a Teaching Tool for Introduction to Object Oriented Programming," *IEEE LATIN AMERICA TRANSACTIONS,* vol. 15, no. 1, p. 97, 2017.

[11] S. Xinogalos, "Designing and deploying programming courses: Strategies, tools, difficulties and pedagogy," *Education and Information Technologies,* vol. 21, no. 3, pp. 559-588, 2016.

[12] S. Xinogalos, "An interactive learning environment for teaching the imperative and object-oriented programming techniques in various learning contexts," in *World Summit on Knowledge Society*, 2010: Springer, pp. 512-520.

[13] U. Technologies. "Unity User Manual." Unity Technologies https://docs.unity3d.com/Manual/ (acceso Octubre 31, 2020).

[14] U. Technologies. "Multiplatform." https://unity.com/features/multiplatform (acceso Octubre 31, 2020).

[15] I. Sommerville and M. Alfonso Galipienso, "Ingeniería de Software (Novena edición ed.)," *MI Alfonso Galipienso, A. Batía Martínez, F. Mora Lizán, & JP Trigueros Jover, Trads.) Ciudad de México, México: Pearson Educación, SA,* 2011.

[16] G. Booch, J. Rumbaugh, and I. Jacobson, "Unified Modeling Language User Guide,(The 2nd Edition)," ed: Addison-Wesley Professional, 2005.

[17] J. Kramer, "Is abstraction the key to computing?," *Communications of the ACM,* vol. 50, no. 4, pp. 36-42, 2007.

[18] J. Wing, "Research notebook: Computational thinking—What and why," *The link magazine,* vol. 6, 2011.
[19] J. M. Wing, "Computational thinking and thinking about computing," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences,* vol. 366, no. 1881, pp. 3717-3725, 2008.
[20] J. M. Wing, "Computational thinking," *Communications of the ACM,* vol. 49, no. 3, pp. 33-35, 2006.
[21] K. Schwaber and J. Sutherland. "The Scrum Guide." SCRUM GUIDES https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf (acceso Noviembre 1, 2021).
[22] P. Deemer, G. Benefield, C. Larman, and B. Vodde. "THE SCRUM PRIMER; A Lightweight Guide to the Theory and Practice of Scrum." Good Agile. http://goodagile.com/scrumprimer/scrumprimer20.pdf (acceso Noviembre 1, 2021).
[23] B. Wagner *et al.* "A tour of the C# language." Microsoft. https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (acceso Noviembre 3, 2021).
[24] Microsoft. "Visual Studio 2019." Microsoft. https://visualstudio.microsoft.com/vs/ (acceso Noviembre 3, 2021).
[25] G. community. "git." Git community. https://git-scm.com/ (acceso Noviembre 14, 2021).
[26] GitHub. "GitHub Docs." GitHub. https://docs.github.com/en (acceso Noviembre 3, 2021).
[27] I. GitHub. "Getting started with GitHub Desktop." GitHub, Inc. https://docs.github.com/en/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop (acceso Noviembre 14, 2021).
[28] Paint.net. "About." Paint.net. https://www.getpaint.net/ (acceso Enero 9, 2022).
[29] M. Meger. "Welcome to the Sokoban Wiki." Sokoban Project. http://sokobano.de/wiki/index.php?title=Main_Page (acceso Enero 2, 2022).
[30] M. Allan. "Asteroids Game (History, Instructions, & FAQs)." Level Skip. https://levelskip.com/classic/Asteroids-Game (acceso Enero 2, 2022).
[31] J. Gallagher. "How Space Invaders Became a Gaming Phenomenon." Den of Geek. https://www.denofgeek.com/games/how-space-invaders-became-a-gaming-phenomenon/ (acceso Enero 2, 2022).
[32] A. L. Santos, "AGUIA/J: a tool for interactive experimentation of objects," in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, 2011, pp. 43-47.
[33] J. M. R. Corral, A. C. Balcells, A. M. Estévez, G. J. Moreno, and M. J. F. Ramos, "A game-based approach to the teaching of object-oriented programming languages," *Computers & Education,* vol. 73, pp. 83-92, 2014.
[34] J. Bennedsen and M. E. Caspersen, "Failure rates in introductory programming," *AcM SIGcSE Bulletin,* vol. 39, no. 2, pp. 32-36, 2007.
[35] C. A. Depradine, "Using gaming to improve advanced programming skills," *The Caribbean Teaching Scholar,* vol. 1, no. 2, 2011.

[36] K. Wood, D. Parsons, J. Gasson, and P. Haden, "It's never too early: pair programming in CS1," in *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136*, 2013, pp. 13-21.
[37] R. Or-Bach and I. Lavy, "Cognitive activities of abstraction in object orientation: an empirical study," *ACM SIGCSE Bulletin,* vol. 36, no. 2, pp. 82-86, 2004.
[38] K. Sanders and R. McCartney, "Threshold concepts in computing: past, present, and future," in *Proceedings of the 16th Koli Calling international conference on computing education research*, 2016, pp. 91-100.
[39] F. Détienne, "Difficulties in designing with an object-oriented language: An empirical study," in *Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, 1990, pp. 971-976.
[40] J. Wing, "Computational Thinking, communication of the ACM, n 49," 2006.
[41] M. Overmars, "Learning object-oriented design by creating games," *IEEE Potentials,* vol. 23, no. 5, pp. 11-13, 2004.
[42] M. B. Rosson and J. M. Carroll, "Climbing the smalltalk mountain," *ACM SIGCHI Bulletin,* vol. 21, no. 3, p. 76, 1990.
[43] L. Yan, "Teaching object-oriented programming with games," in *2009 Sixth International Conference on Information Technology: New Generations*, 2009: IEEE, pp. 969-974.
[44] T. A. Powell, D. L. Jones, and D. C. Cutts, *Web site engineering: beyond Web page design*. Prentice-Hall, Inc., 1998.
[45] S. Ramírez-Rosales, S. Vázquez-Reyes, J. L. Villa-Cisneros, and M. De León-Sigg, "A serious game to promote object oriented programming and software engineering basic concepts learning," in *2016 4th International Conference in Software Engineering Research and Innovation (CONISOFT)*, 2016: IEEE, pp. 97-103.
[46] Y. S. Wong, M. Y. M. Hayati, and W. H. Tan, "A propriety game-based learning game as learning tool to learn object-oriented programming paradigm," in *Joint International Conference on Serious Games*, 2016: Springer, pp. 42-54.
[47] E. Lotfi, B. Y. Othman, and B. Mohammed, "Towards a Mobile Serious Game for Learning Object Oriented Programming Paradigms," in *The Proceedings of the Third International Conference on Smart City Applications*, 2018: Springer, pp. 450-462.
[48] I. Inayat, Z. Inayat, and R. U. Amin, "Teaching and learning object-oriented analysis and design with 3D game," in *2016 International Conference on Frontiers of Information Technology (FIT)*, 2016: IEEE, pp. 46-51.
[49] K. Fernandez-Reyes, D. Clarke, and J. Hornbach, "The impact of opt-in gamification on students' grades in a software design course," in *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2018, pp. 90-97.
[50] M. I. Nawahdah, "Gamifying the teaching and learning process in an advanced computer programming course," in *International Conference on Collaboration Technologies*, 2018: Springer, pp. 89-95.
[51] S. Frezza and W. Andersen, "Interactive exercises to support effective learning of UML structural modeling," in *Proceedings. Frontiers in Education. 36th Annual Conference*, 2006: IEEE, pp. 7-12.

[52] P. V. Gestwicki, "Computer games as motivation for design patterns," *ACM SIGCSE Bulletin,* vol. 39, no. 1, pp. 233-237, 2007.
[53] P. Gestwicki and F.-S. Sun, "Teaching design patterns through computer game development," *Journal on Educational Resources in Computing (JERIC),* vol. 8, no. 1, pp. 1-22, 2008.
[54] M. Adams. "A brief overview of planning poker." Atlassian. https://www.atlassian.com/blog/platform/a-brief-overview-of-planning-poker (acceso Enero 2, 2022).
[55] M. Ruwe. "Pointing Poker." Pointing Poker. https://www.pointingpoker.com/ (acceso Noviembre 10, 2021).
[56] R. Nystrom, *Game programming patterns*. Genever Benning, 2014.
[57] U. Technologies. "ECS concepts." Unity Technologies. https://docs.unity3d.com/Packages/com.unity.entities@0.17/manual/ecs_core.html (acceso Enero 23, 2022).
[58] Atlassian. "Gitflow workflow: Atlassian Git Tutorial." Atlassian. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (acceso Enero 7, 2022).
[59] Techopedia. "Tooltip." Techopedia. https://www.techopedia.com/definition/5482/tooltip (acceso Enero 9, 2022).
[60] I. GitHub. "About GitHub Pages." GitHub, Inc. https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages (acceso Enero 24, 2022).
[61] J. Brooke. "System Usability Scale (SUS)." U.S. General Services Administration | Technology Transformation Services. https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html (acceso Enero 25, 2022).
[62] F. Devin. "Sistema de Escalas de Usabilidad: ¿qué es y para qué sirve?" UXpañol. https://uxpanol.com/teoria/sistema-de-escalas-de-usabilidad-que-es-y-para-que-sirve/ (acceso Enero 25, 2022).

ANEXOS

ANEXO 1: Repositorio del proyecto en GitHub

En el siguiente enlace se puede revisar el repositorio del proyecto en GitHub: https://github.com/crondito/TesisCR

ANEXO 2: Aplicación web

En el siguiente enlace se puede revisar la aplicación web terminada: https://crondito.github.io/TesisCR/

ANEXO 3: Unity Simple Tooltip

En el siguiente enlace se puede revisar la herramienta gratuita: https://assetstore.unity.com/packages/tools/gui/simple-tooltip-155804

ANEXO 4: Resultados prueba de usabilidad

En el siguiente enlace se puede ver los resultados de la prueba de usabilidad: https://docs.google.com/spreadsheets/d/1AewsbaY_l0AaVZyuPUrEHzxPDZ6wWRdKldAyts8ypq8/edit?usp=sharing

ANEXO 5: Muestra del producto final

En este anexo se muestra el producto final de este proyecto, el cual consta de 3 secciones, y un menú principal, las cuales serán descritas a continuación.

Primeramente, el usuario es recibido por el Menú Principal, desde este tiene acceso a las demás secciones.

Figura 84 - Menú Principal

Al hacer clic en el botón “Teoría” accede a la sección teórica de la aplicación.

Figura 85 - Menú Sección "Teoría"

Todas las pantallas tienen un botón “x” en la esquina superior izquierda para salir hacia la pantalla anterior. También constan de un botón para apagar o prender los sonidos de la aplicación en la esquina superior derecha.

La sección teórica consta de 4 subsecciones, la primera es la subsección del “Ciclo de vida del Software”.

Figura 86 - Sección teórica de "Ciclo de Vida del Software"

Todas las subsecciones teóricas también cuentan con una lección para que los estudiantes pongan a prueba sus conocimientos.

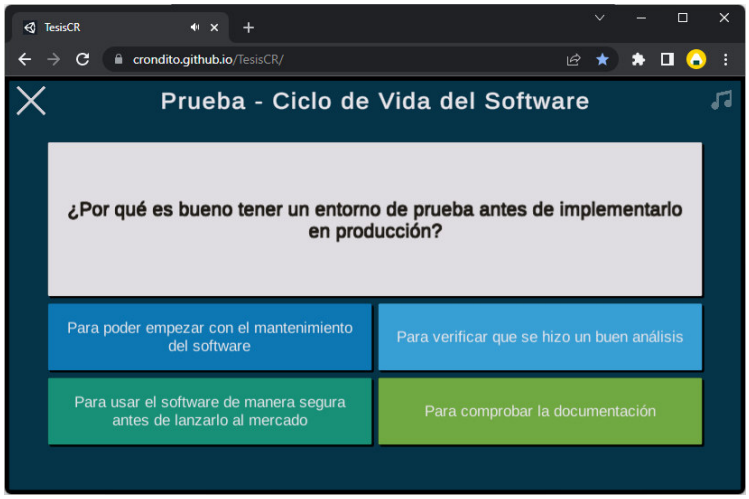


Figura 87 - Prueba de la sección teórica de "Ciclo de Vida del Software"

El tema de la segunda subsección teórica es “Abstracción”. También cuenta con una prueba para los estudiantes tras finalizarla.

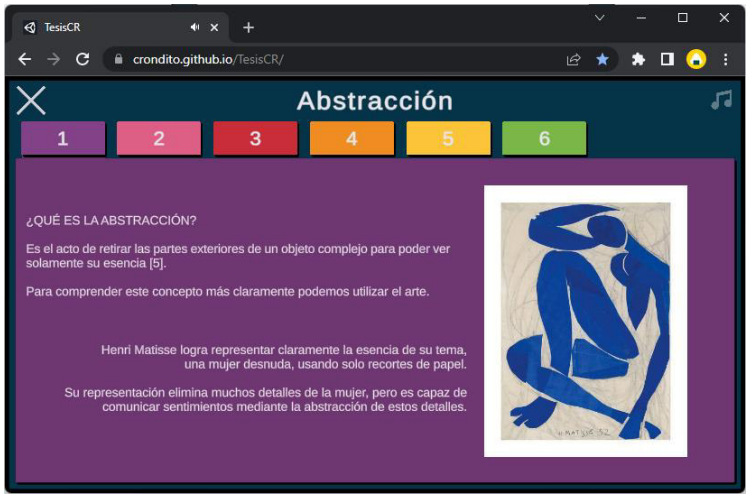


Figura 88 - Sección teórica de "Abstracción"

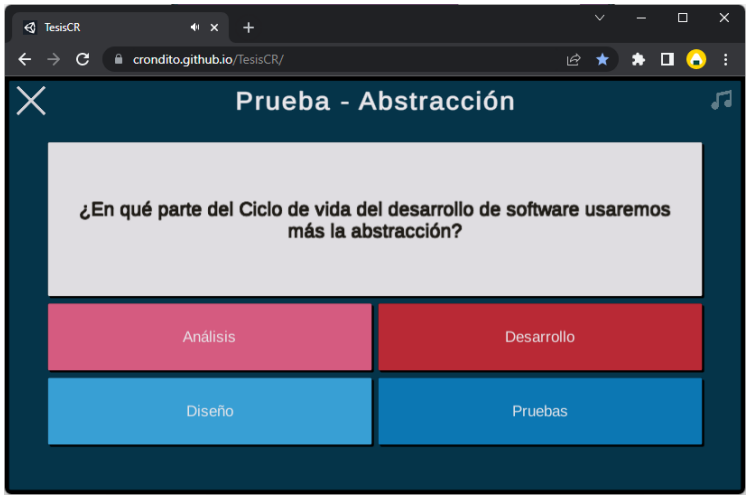


Figura 89 - Prueba de la sección teórica de "Abstracción"

El tema de la tercera subsección teórica es “UML”. Como las anteriores, también cuenta con una prueba para los estudiantes tras finalizar la teoría.

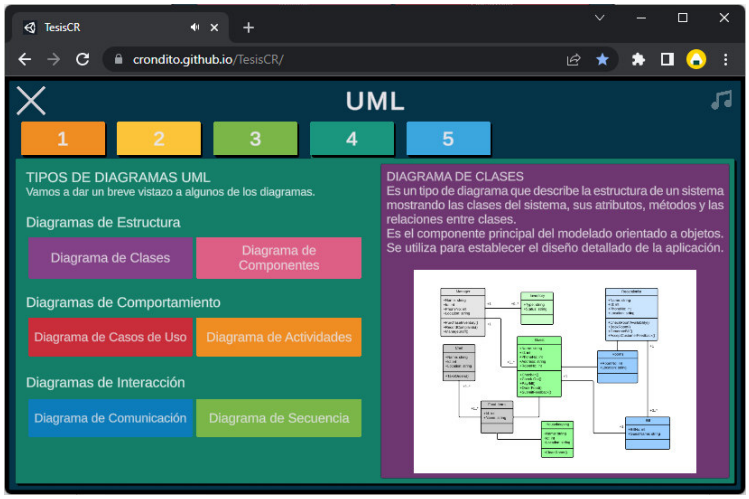


Figura 90 - Sección teórica de "UML"

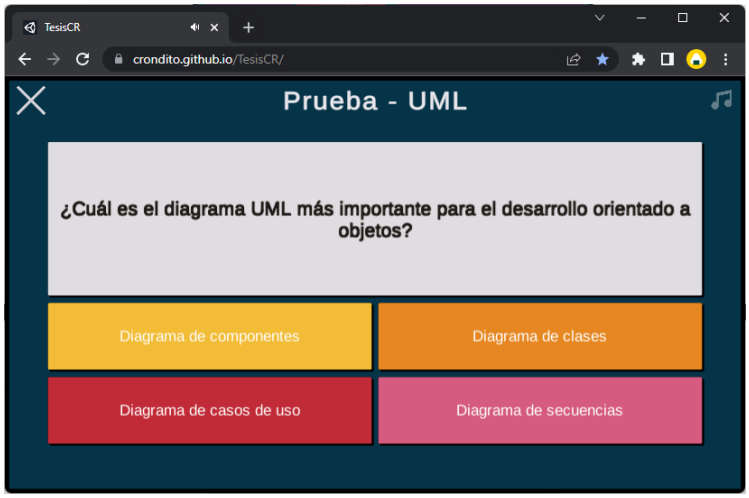


Figura 91 - Prueba de la sección teórica de "UML"

La cuarta subsección teórica trata sobre “Diagrama de Clases UML”. Cuenta con una prueba igual que las subsecciones teóricas anteriores.

Figura 92 - Sección teórica de "Diagrama de Clases UML"

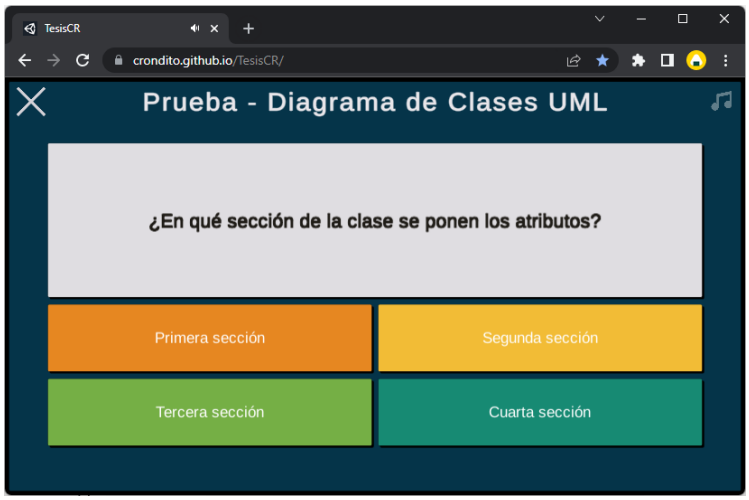


Figura 93 - Prueba de la sección teórica de "Diagrama de Clases UML"

A continuación, encontramos la sección de “Ejercicios de Diseño”. Esta sección cuenta con cuatro ejercicios de diseño y tres juegos de video que pueden desbloquearse al terminar los ejercicios.

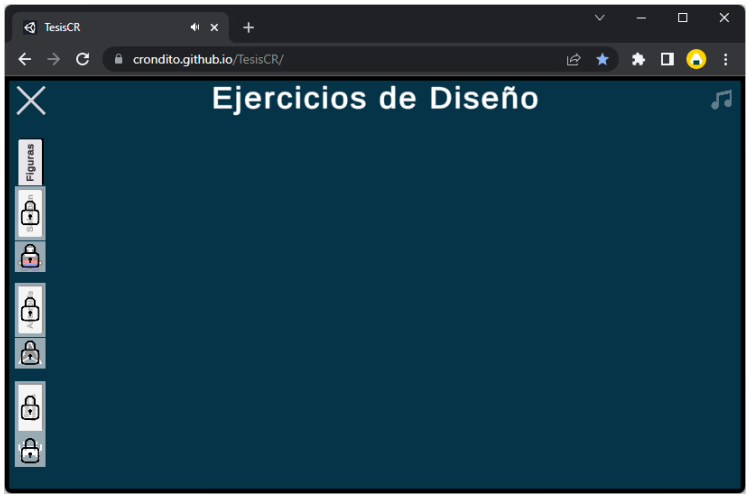


Figura 94 - Menú de Ejercicios de Diseño

El primer ejercicio se llama “Figuras”. El objetivo de este ejercicio es enseñar conceptos básicos de cómo diseñar software con el uso de diagramas de clases y figuras geométricas simples.

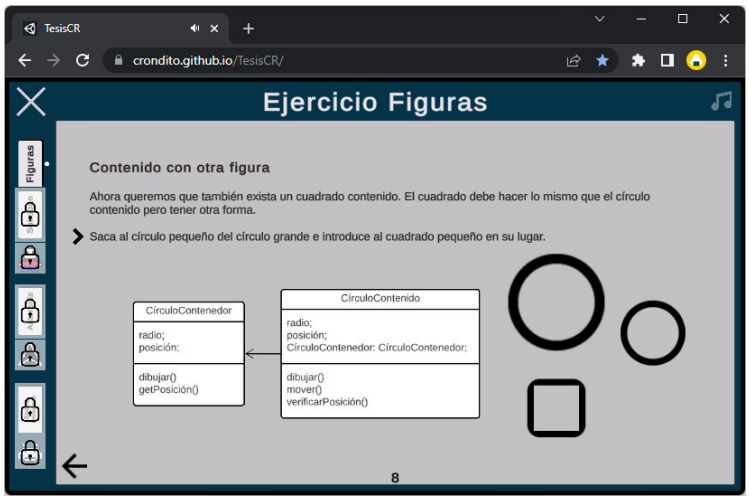


Figura 95 - Ejercicio "Figuras"

Al terminar el ejercicio de “Figuras” los estudiantes desbloquean el ejercicio “Sokoban”. En este ejercicio se diseña un juego de video usando diagramas de clases UML. El ejercicio es interactivo y entretenido para los estudiantes pues pueden jugar partes del juego de video mientras lo diseñan.

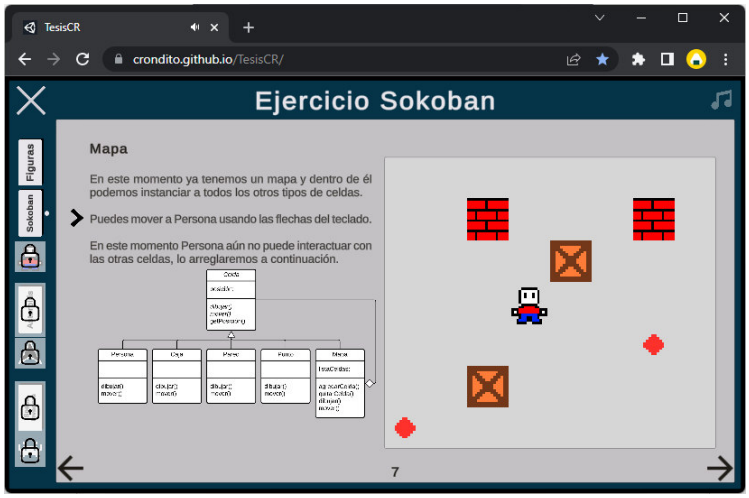


Figura 96 - Ejercicio "Sokoban"

Al terminar con el ejercicio desbloquean el siguiente ejercicio como también el juego de video completo, con 5 niveles.

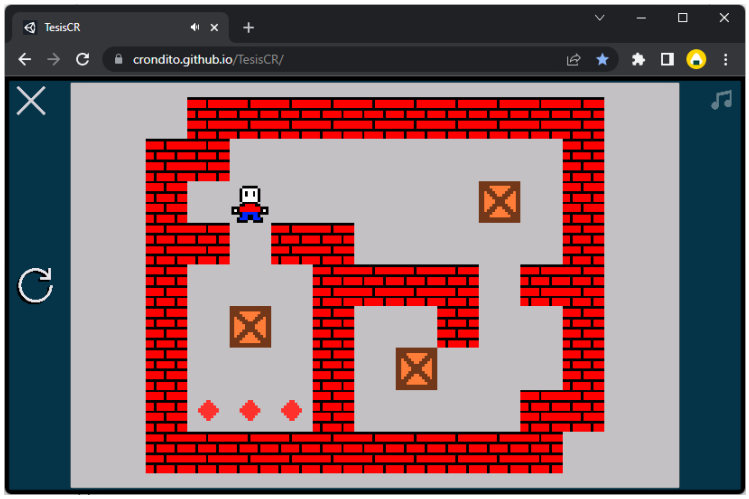


Figura 97 - Juego de video "Sokoban"

El siguiente ejercicio se llama “Asteroids”, en este los estudiantes también diseñan un juego de video clásico. Los conceptos del diagrama de clases UML se vuelven más difíciles pero se sigue teniendo bastanet interactividad.

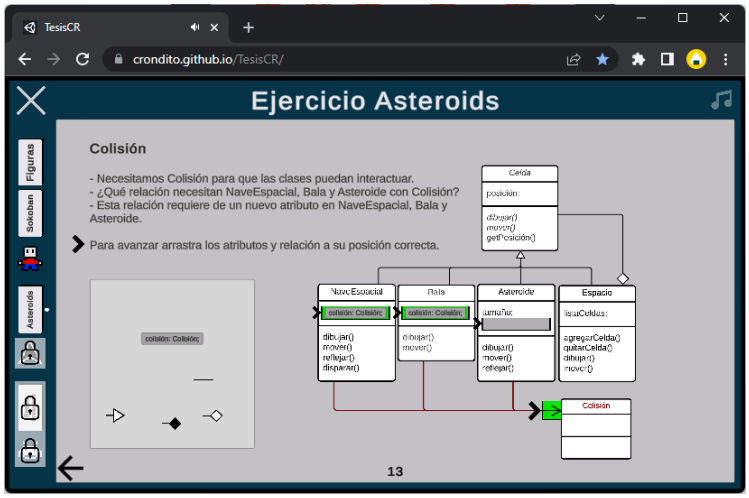


Figura 98 - Ejercicio "Asteroids"

Al terminar el ejercicio de diseño los estudiantes pueden jugar el videojuego, como también continuar al siguiente ejercicio.

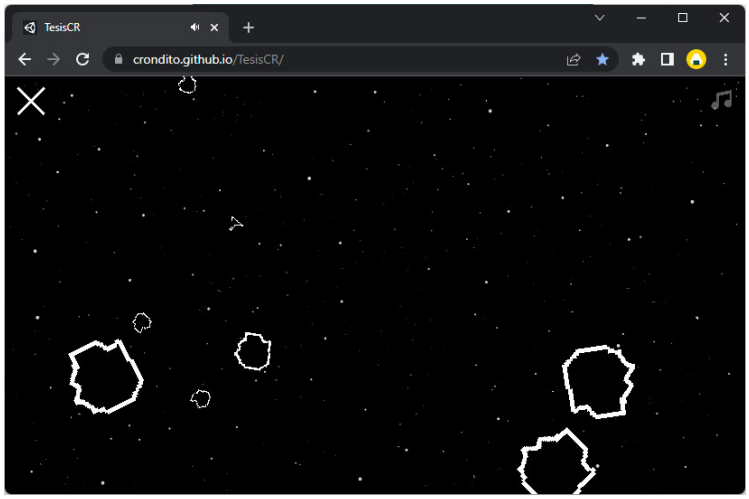


Figura 99 - Juego de video "Asteroids"

El ejercicio final tiene el nombre “Space Invaders”. Igual que los anteriores ejercicios se tiene que diseñar un juego de video clásico con el diagrama de clases, pero la dificultad aumenta ya que se le agrega conceptos como interfaces gráficas de usuario y puntuación. En este ejercicio también se les da menos ayuda a los estudiantes para que demuestren que entienden lo que se vio en los ejercicios anteriores.

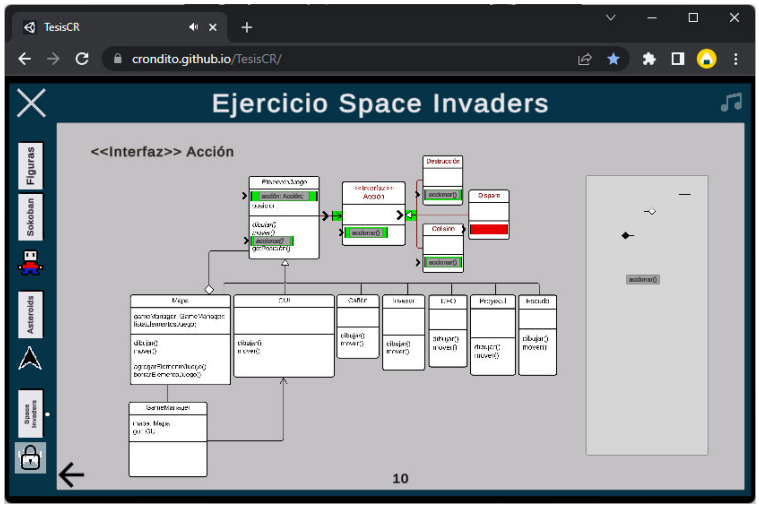


Figura 100 - Ejercicio "Space Invaders"

Al terminar el ejercicio final podremos jugar el videojuego final, “Space Invaders”.

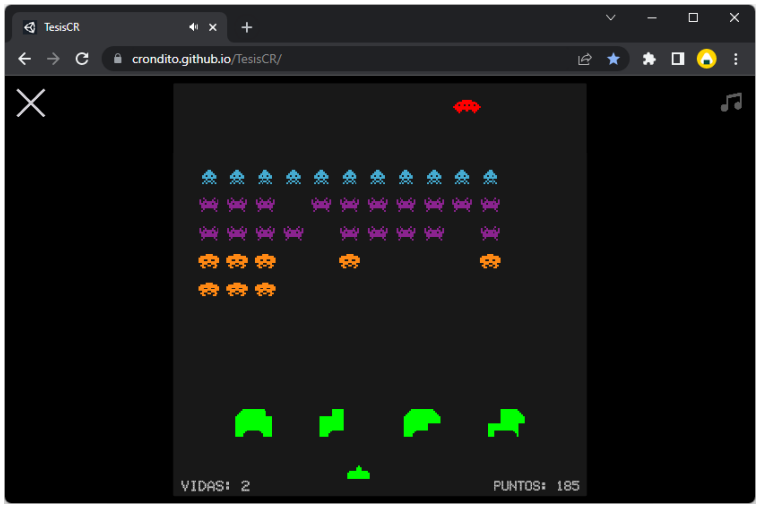


Figura 101 - Juego de video "Space Invaders"

Por último, tenemos la sección de información, el propósito de esta sección es mostrar el progreso del estudiante y las referencias bibliográficas. El primer panel de esta sección brinda información sobre el progreso, notas en las pruebas, y si ya terminó o no los ejercicios de diseño.

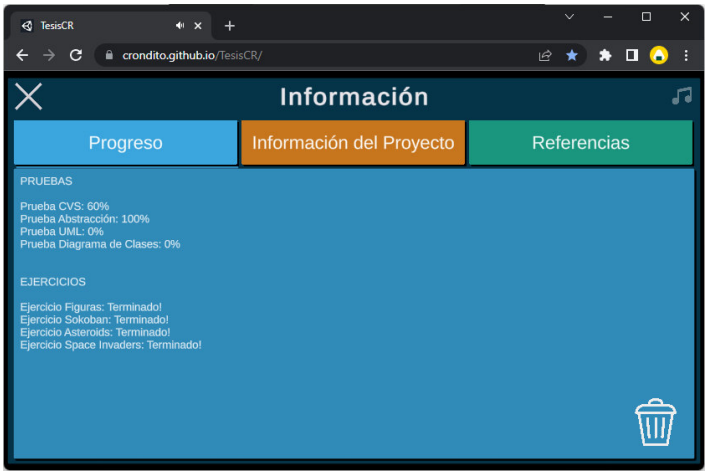


Figura 102 - Panel de Progreso

El siguiente panel brinda información sobre los desarrolladores de la aplicación.

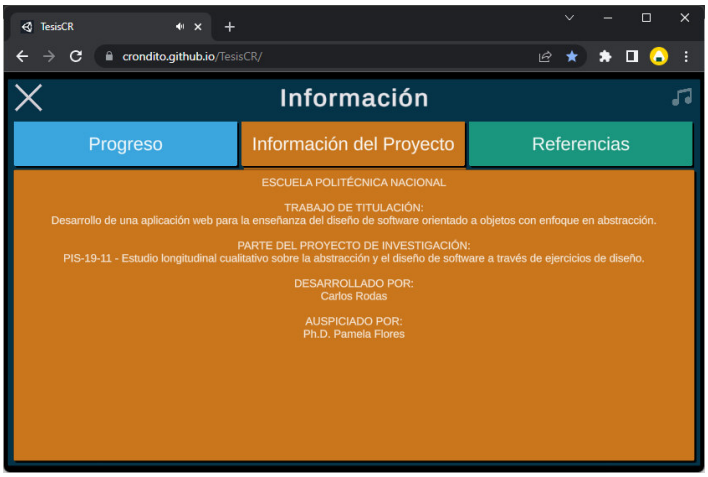


Figura 103 - Panel de Información del Proyecto

El último panel brinda información sobre las referencias bibliográficas usadas en este proyecto.

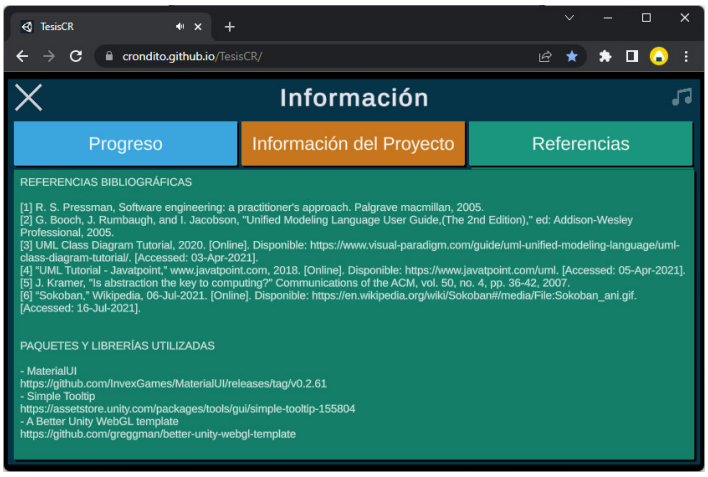


Figura 104 - Panel de Referencias