Escuela Universitaria de Ingeniería Técnica Industrial de Bilbao

Grado en Ingeniería Informática de Gestión y Sistemas de Información

Memoria del Trabajo Fin de Grado

Videojuegos y perfiles psicológicos

Autores: Joseba Gallaga Fernández y David Espino Gómez

Director: Aitziber Atutxa Salazar

Universidad del País Vasco 2017

Resumen

El mercado del videojuego ha conseguido un gran incremento y aceptación entre la ciudadanía hasta convertirse en uno de los mercados que más dinero mueve junto a otro tipo de entretenimiento como las películas o la música. Gracias a éxitos como Plasytation 2 o Nintendo Wii, el sector se ha expandido a un público muy diverso en edad y sexo. Esta situación ha provocado que las compañías hayan encontrado una nueva fuente de análisis en el comportamiento de las personas.

Se explicará por qué y cómo se ha implementado una aplicación, en forma de dos videojuegos, que será capaz de recopilar información sobre la personalidad del usuario, tratarla y crear un módulo de predicción en base a ella.

Lo que se busca en este Proyecto de Fin de Grado es encontrar relación entre como juega una persona, y ciertos aspectos de su personalidad.

Índice general

Contenido

Índice de figuras

Introducción:

Ahora mismo el sector de los videojuegos se ha convertido en uno de los sectores más lucrativos de la informática. Hace tiempo poder programar un juego requería una inversión considerable dado que no estaban disponibles herramientas libres para poder hacerlo. Ahora existe un gran abanico de motores gráficos como Source, Unreal Engine o Unity. Por otro lado, para diseñar e implementar un juego se requiere un amplio espectro de conocimientos, que van desde bases de datos e interfaces gráficas, hasta algoritmos de inteligencia artificial.

Por todo esto la idea de desarrollar un juego como trabajo de fin de grado es un reto a nivel conceptual, pero por otro lado, diseñar un juego parece algo frívolo cuyo único objetivo es buscar la diversión del usuario.

Basándose en la bibliografía consultada la forma en que una persona juega tiene mucho que ver con su personalidad, por ello se ha querido añadir una meta secundaria. En este caso utilizar el juego que se desarrolle cómo un método de estudio sobre el usuario. Se intentará relacionar cómo interactúan las personas en él con rasgos de su personalidad. Para este fin se utiliza la minería de datos. El propio juego se encargará de recopilar datos sobre el usuario, y mediante algoritmos de minería en un programa externo serán convertidos en información sobre su personalidad.

Es importante aclarar que para que el módulo de minería de datos aprenda los rasgos de personalidad y se pueda comprobar la calidad de los resultados sería necesario, por un lado muchos datos para el aprendizaje y por otro lado datos también para poder realizar la evaluación. Esto es imposible conseguirlo en un proyecto de fin de grado, dado que para poder recopilar todos esos datos, el juego que ha de desarrollarse debería estar acabado y se debería de disponer de cientos o miles de jugadores que lo probasen. Por eso es importante aclarar que el objetivo final es diseñar e implementar un juego que recopile información sobre la forma de jugar del usuario y diseñar e implementar el módulo de aprendizaje automático que permitirá a partir de los datos de un jugador obtener rasgos de su personalidad, pero que en ningún caso se podrá llegar a probar su fiabilidad dentro del alcance del presente proyecto. Así pues podría suceder que la información extraída de la interacción del usuario con el juego no aporte realmente información útil, es decir, información a partir de la cual se pueda hacer una predicción correcta. Para evitar que esto suceda se ha analizado bibliografía y se ha intentado en la medida de lo posible obtener información cuya utilidad haya sido probada en algún trabajo previo, pero el estado del arte es muy limitado y hay muy pocos trabajos accesibles.

Además de eso mucho de los datos que se extraigan pueden solaparse, es decir tener el mismo significado, esto puede causar que aunque a priori se extraiga una gran cantidad de datos finalmente solo unos pocos de ellos o ninguno sean útiles.

Motivación:

Motivación personal:

La motivación principal es poder demostrar que los conocimientos adquiridos durante los estudios universitarios nos permiten crear un proyecto basándonos en una idea propia y ser capaces de realizarlo desde un comienzo hasta su fin de forma correcta.

Este trabajo de fin de grado nos pone en la situación de un proyecto complejo al que podríamos enfrentarnos en nuestra vida laboral. Ser capaces de finalizarlo nos prepara y da confianza a enfrentarnos a cualquier posible nuevo reto que pueda aparecernos en el desempeño de nuestro oficio.

Motivación conceptual:

Este proyecto es novedoso e interesante dado que como se ha dicho previamente, obtener información sobre los rasgos de personalidad de usuarios es algo que muchas empresas buscan para así poder interactuar con el usuario de una forma más personalizada. Pero no hay muchos trabajos publicados al respecto lo cual lo convierte en un reto interesante.

Objetivos del proyecto

Objetivos

Implementar dos juegos funcionales de los que se extraigan datos sobre la forma de jugar del usuario a fin de analizarlo en un módulo de minería de datos para predecir información sobre la personalidad del mismo.

Propósito

Investigar si hay relación entre los movimientos y la forma de jugar de un usuario con datos de su personalidad.

Ámbito

Es un campo que se está empezando a investigar sobre todo por empresas a fin de saber mejor como es su cliente, en nuestro caso el motivo es la investigación.

Alcance

En el juego:

1. Estudiar la relación entre la forma de jugar y rasgos de la personalidad
2. Definir qué aspectos de la jugabilidad se quieren capturar.
3. Desarrollar dos juegos 2D que permitan esa captura.
4. Extraer datos del usuario según juega.
5. Enviar estos datos a un módulo de minería.

Módulo de minería:

1. Crear una encuesta con preguntas para poder definir a la persona.
2. Realizar el test y hacer probar el juego al mayor número de personas.
3. Con la encuesta y los datos del juego buscar correlaciones.
4. Conseguir que el modulo pueda ser utilizado con datos de diferentes juegos.

Herramientas

Unity

Figura 1: Herramientas - Unity

Motor de videojuego desarrollado por Unity Technologies para desarrollo de juegos multiplataforma.

Eclipse

Figura 2: Herramientas - eclipse

Plataforma de software que permite la programación de diferentes lenguajes como es Java.

Monodevelop

Figura 3: Herramientas - MonoDevelop

Entorno de desarrollo integrado en Unity, libre y gratuito diseñado para funcionar tanto con C# como Javascript. Cuenta con soporte tanto para GNU/Linux, Windows y Mac.

Visual Studio

Figura 4: Herramientas - VisualStudio

Entorno de desarrollo integrado para Windows. Soporta mayor variedad de lenguajes que MonoDevelop como son C++, Visual Basic o Java.

C# (C Sharp)

Figura 5: Herramientas - VisualC#

Lenguaje de programación orientado a objetos desarrollado por Microsoft. Uno de los lenguajes que pueden ser utilizados por los scripts de Unity

Javascript

Figura 6:Herramientas - JavaScript

Lenguaje de programación interpretado y orientado a objetos utilizable por los scripts de Unity.

PHP

Figura 7: Herramientas - php

Lengauje de programación del lado del servidor que permite la comunicación con las Bases de Datos y tratado de variables por formularios.

MySQL

Figura 8: Herramientas - MySql

Sistema de gestión de Bases de Datos relacional desarrollado por Oracle Corporation. Es open source y es de las mas utilizadas en el mundo.

CoreFTP

Figura 9: Herramientas - CoreFTP

Cliente FTP seguro para Windows desarrollado por CoreFTP.com. Utilizado en este proyecto para mandar los archivos PHP al servidor.

Visual Paradigm

Figura 10: Herramientas - VisualParadigm

Herramienta para el diseño de diagramas UML y generación de código a partir de ellos. Util para los diagramas de Uso, Clases o Secuencia entre otros.

Pencil Proyect

Figura 11: Herramientas - PencilProyect

Prototipador de GUI gratuito disponible en todas las plataformas, Windows, Linux e IOS. Fácil y rapido de usar siendo util para dibujar interfaces provisionales.

GanttProject

Figura 12: Herramientas - GanttProject

Plataforma para crear diagramas para la gestion temporal de los proyectos.

WBsTools

Figura 13: Herramientas - WBs Tools

Software gratuito web para crear diagramas de cascada organigramas y otros tipos de jerarquias.

Dropbox

Figura 14: Herramientas - Dropbox

Servicio de alojamiento de archivos multiplataforma en la nube, permite compartir entre varios usuarios y cuenta con control de versiones.

Git

Figura 15: Herramientas - git

Software de control de versiones diseñado por Linus Torvalds, pensado para ser eficiente y confiable cuando se controla una gran cantidad de archivos a la vez.

Paint.net

Figura 16: Herramientas - paint.net

Editor de imágenes para Windows desarrollado en el marco de trabajo .net.

Gimp

Figura 17: Herramientas - Gimp

programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Es un programa libre y gratuito.

Photoshop

Figura 18: Herramientas - photoshop

Editor de gráficos rasterizados desarrollado por Adobe. Usado principalmente para el retoque de fotografías y gráficos.

ShareX

Figura 19: Herramientas - shareX

Programa para realizar capturas de pantalla a fin de documentar el proyecto con imágenes.

Audacity

Figura 20: Herramientas - Audacity

Aplicación informática multiplataforma libre, que se puede usar para grabación y edición de audio, distribuido bajo la licencia GPL.

Microsoft Word

Figura 21: Herramientas - Word

Procesador de textos desarrollado por Microsoft. Es el procesaor mas utilizado en el mundo.

Power Point

Figura 22: Herramientas - PowerPoint

Programa parte de Microsoft Office empleado para realizar presentaciones con pases de diapositivas.

Notepad++

Figura 23: Herramientas - Notepad++

Editor de texto y de código fuente libre con soporte para varios lenguajes de programación. Se parece al Bloc de notas en cuanto al hecho de que puede editar texto sin formato y de forma simple. No obstante, incluye opciones más avanzadas que pueden ser útiles para usuarios avanzados como desarrolladores y programadores.

Skype

Figura 24: Herramientas - Skype

Sofware adquirido por Microsoft que permite la realización de llamadas y videoconferencias.

ObsStudio

Figura 25: Herramientas - ObsStudio

Aplicación libre y de código abierto para la grabación y transmisión de video por internet mantenida por *OBS Project.*

Weka

Figura 26: Herramientas - WEKA

Plataforma para el aprendizaje automático y minería de datos desarrollado por la Universidad de Waikato, es un software libre y la base de este proyecto.

.

Funcionamiento del programa

Para llegar al objetivo final se definen una serie de pasos que deberá seguir el proyecto.

Primero como es necesario recopilar información es necesaria una base de datos donde se guarden los usuarios con su información, estos datos pueden ser sensibles, será necesario investigar los derechos de uso y como tratar la privacidad de los datos.

Se quiere desarrollar un sistema predictivo. Para construirlo primero hacen falta definir qué información es la que tiene que ser captada mientras juega el usuario. Seguidamente hay que conseguir instancias de prueba, es decir usuarios de los que se conozca tanto como son y como juegan. Para conseguirlo se necesita un modo de extraer la verdadera personalidad del usuario (la clase), ya sea mediante encuestas o entrevistas, esto necesitará investigación.

Con el juego también se extraerán datos, en este caso del desempeño del usuario mientras lo utiliza, estos datos pueden extraerse mediante observación directa o analizando las pulsaciones del teclado.

Con los datos del juego y la clase real se formará un fichero .arff para poder analizarlo con Weka. Si Weka consigue extraer una correlación entre los datos extraídos y la clase, entonces se formará un sistema predictivo, con este se podrá hacer un acercamiento a la personalidad de un jugador con solo utilizar la parte del juego y sin necesidad del test.

Hay que señalar que existirán dos juegos, uno por cada desarrollador, el de Joseba será referido como juego “Shooter”, y el de David como juego “Arcade”.

Ciclo de vida

Al ser un proyecto extenso, con partes claramente diferenciables; implementación del juego, recopilación de los datos sobre la jugabilidad y predicción de los rasgos de personalidad; se cree que el ciclo de vida más conveniente sería **Iterativo por Prototipos** ya que combina el modelo tradicional de cascada con la filosofía interactiva del **Prototipado**.

Los tres prototipos serían los siguientes:

Prototipo 1: Implementación de un juego teniendo en mente que se recogerán datos de él. En esta fase se conseguirá un juego sencillo con niveles básicos del que todavía no se extraen datos.

Prototipo 2: Implementar la recogida y envió de datos desde el juego. El almacenamiento se realizará en una Base de datos. Para ello será necesario también un menú.

Prototipo 3: Tratar los datos recopilados en el prototipo 2, crear un modelo de predicción y mostrar resultados en el juego.

Al final de estas 3 iteraciones tendremos un software completo y funcional. El prototipo 3 se considera un objetivo secundario, con lo que el programa debe funcionar perfectamente a pesar de no haber sido implementado.

Arquitectura

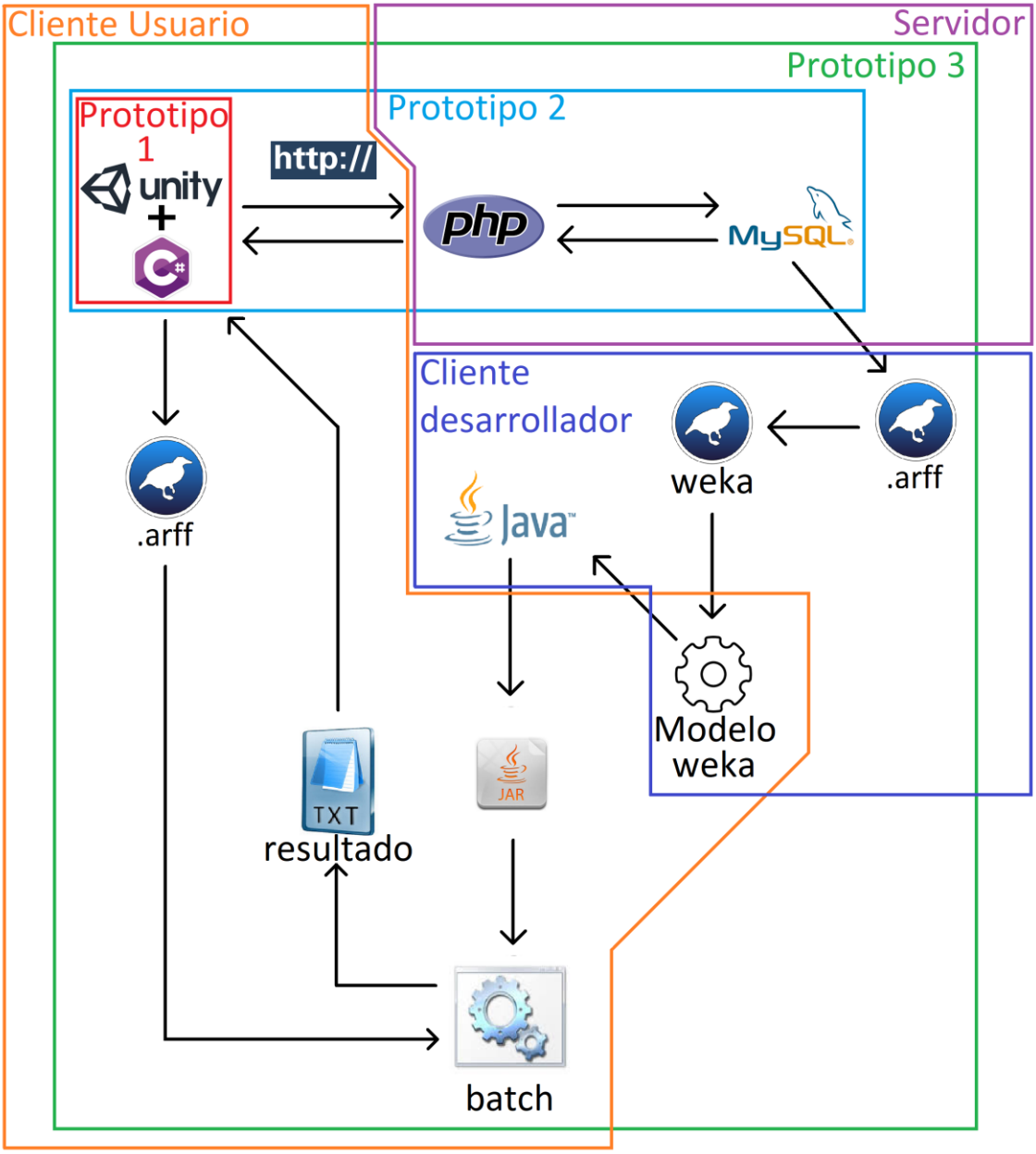


Figura 27: Arquitectura

El prototipo 1 (rojo) es el juego. En el prototipo 2 (azul claro) el usuario realiza tanto la encuesta como el juego en modo test (cliente usuario en naranja) y se envía la información al servidor (morado). El servidor se encarga de guardar los datos.

El desarrollador (azul oscuro)con la información de la base de datos genera los archivos .arff. Mediante Weka se crea un modelo de predicción almacenado en un archivo . jar. Una vez se tiene todo se crea una nueva versión del programa.

Cuando ya existe un modelo predictor el usuario puede descargar una versión del juego completa (prototipo 3 en verde). En esta versión el jugador puede hacer una partida predictora directamente sin necesidad de realizar encuesta.

Estructura de Descomposición del Trabajo (EDT)

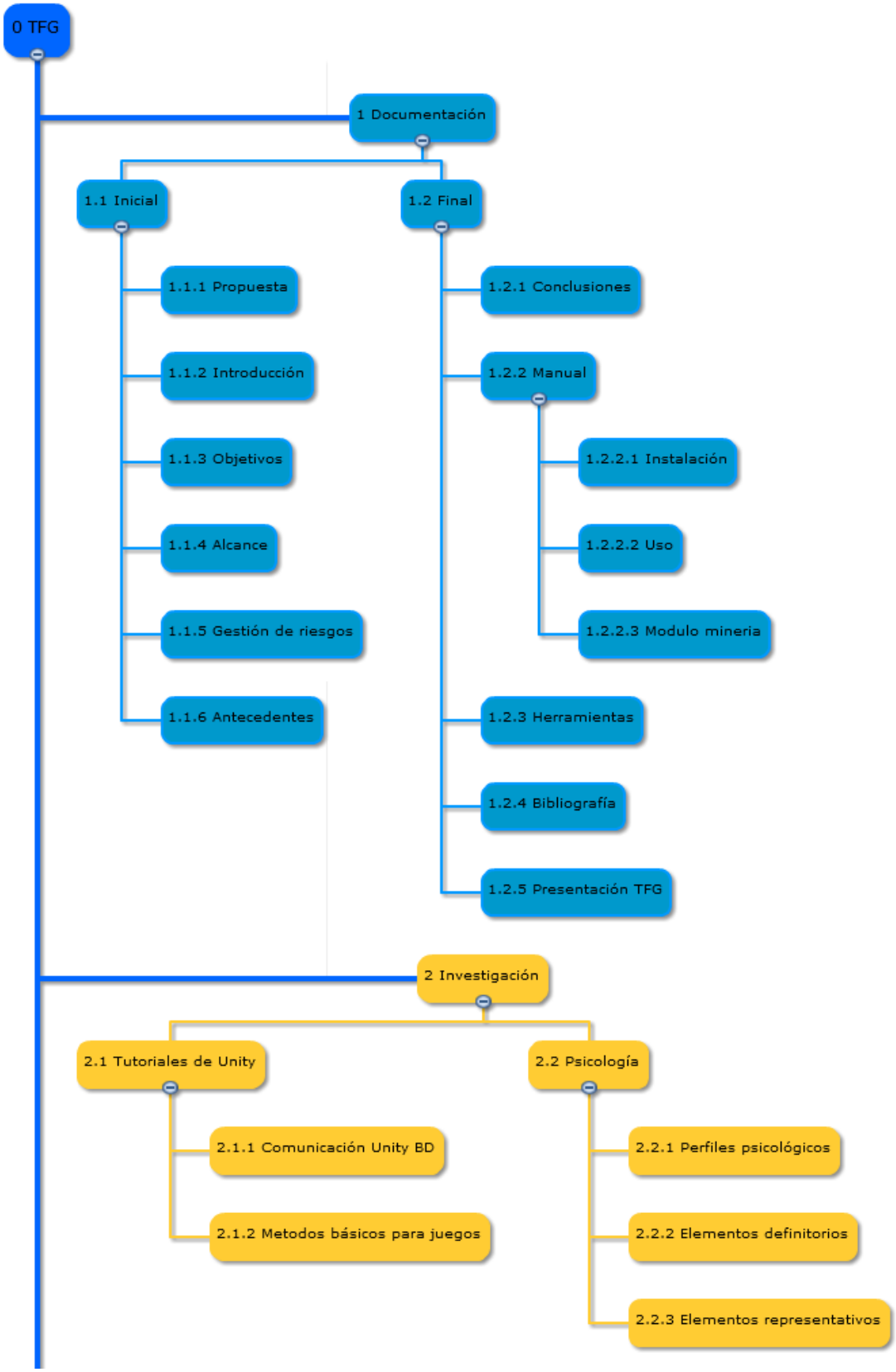


Figura 28: EDT 1

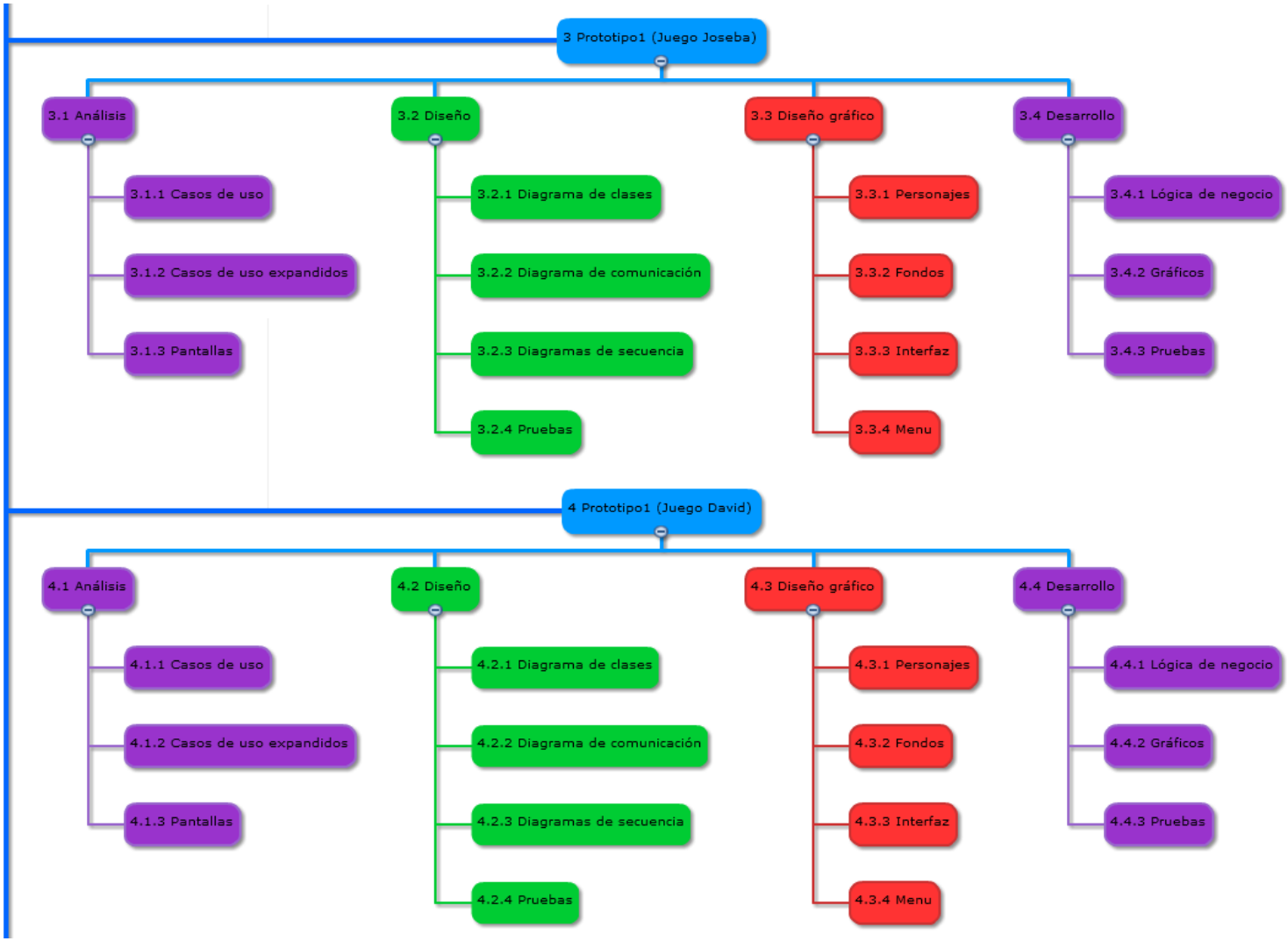


Figura 29: EDT 2

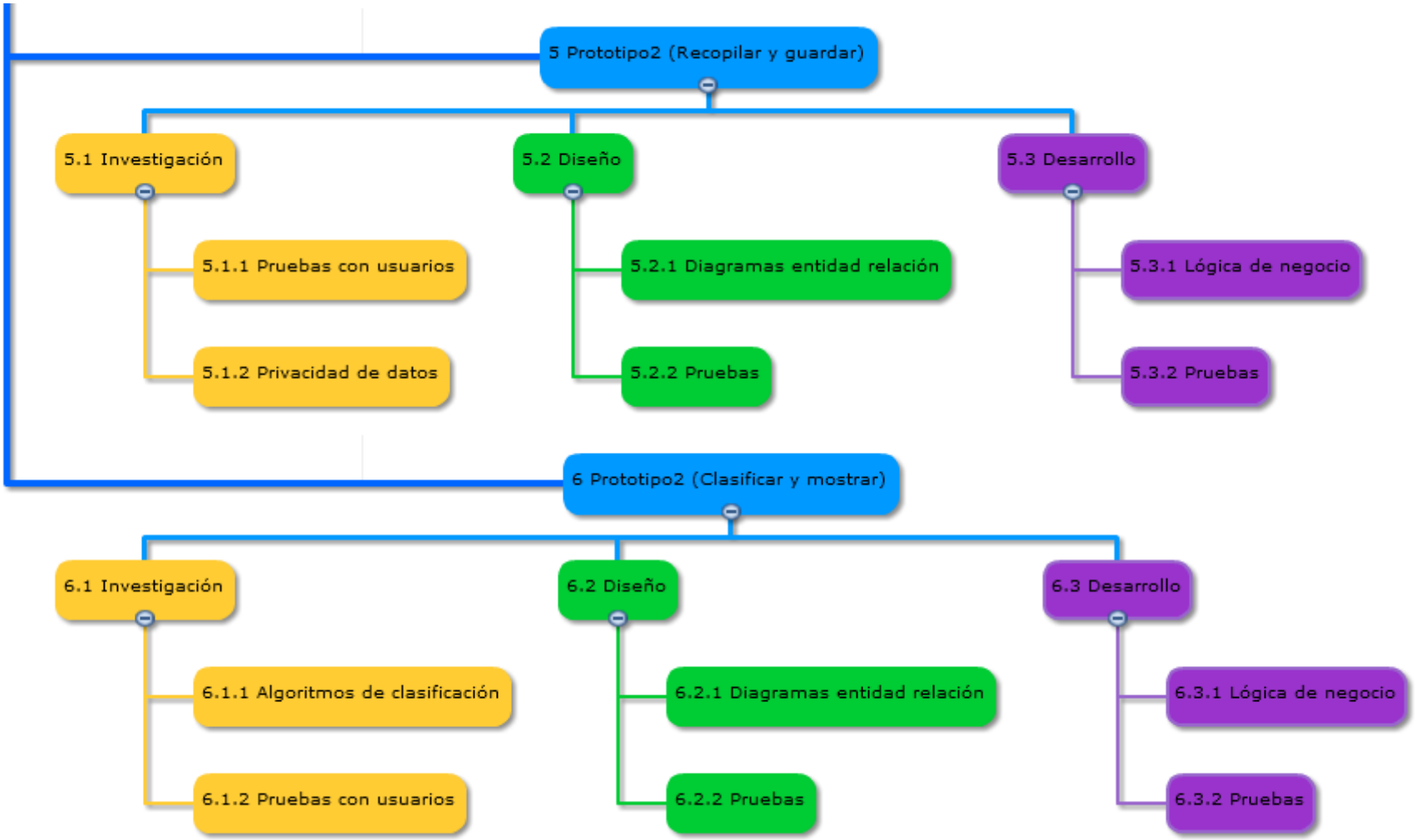


Figura 30: EDT 2

Desglose de Tareas del EDT

Documentación

Documentación inicial

Paquete de trabajo: 1.1.1 Propuesta
Responsables: Joseba, David

Descripción: Documento desarrollando nuestra idea de proyecto, explicando pros, contras y herramientas necesarias
Entrada: Ninguna
Entregables: Documento de texto
Recursos: Internet, procesador de texto
Precedencias: 2.2. Psicología

Paquete de trabajo: 1.1.2 Introducción
Responsables: Joseba, David

Descripción: Breve descripción de lo que se tratará en este proyecto
Entrada: Ninguna
Entregables: Documento de texto
Recursos: Internet, procesador de texto
Precedencias: Ninguna

Paquete de trabajo: 1.1.3 Objetivos
Responsables: Joseba, David

Descripción: Objetivos, propósito y ámbito del proyecto
Entrada: Ninguna
Entregables: Pequeño texto informando de cada uno de los puntos
Recursos: Internet, procesador de texto
Precedencias: 2.2 Psicología

Paquete de trabajo: 1.1.4 Alcance
Responsables: Joseba, David

Descripción: Que implicará el proyecto, hasta donde tenemos que llegar y todo lo que abarcará
Entrada: Ninguna
Entregables: Resumen por puntos, EDT y desglose de tareas
Recursos: Internet, procesador de texto, editor EDT
Precedencias: 1.1.3 Objetivos

Paquete de trabajo: 1.1.5 Gestión de riesgos
Responsables: Joseba, David

Descripción: En todo proyecto existen inconvenientes y posibles parones por diferentes circunstancias, hay que tener en cuenta estos peligros y considerar su riesgo
Entrada: Ninguna
Entregables: Resumen de cada uno de los posibles riesgos y sus consecuencias
Recursos: Procesador de texto
Precedencias: 1.1.4 Alcance

Paquete de trabajo: 1.1.6 Antecedentes
Responsables: Joseba, David

Descripción: Antes de realizar algo hay que investigar si existen referencias en las que basarnos o si aportaremos algo nuevo
Entrada: Ninguna
Entregables: Documentos de texto
Recursos: Internet, procesador de texto
Precedencias: 2.2. Psicología

Documentación final

Paquete de trabajo: 1.2.1 Conclusiones
Responsables: Joseba, David

Descripción: Después de terminar el proyecto, que hemos conseguido, que hemos aportado, que no ha sido posible. Reflexión final sobre todo lo realizado
Entrada: La información aportada por la aplicación sobre los usuarios
Entregables: Documento de texto
Recursos: Internet, procesador de texto, resultados de minería
Precedencias: 1.1 Documentación inicial, 3(4) Prototipo 1, 5 Prototipo 2, 6 Prototipo 3

Paquete de trabajo: 1.2.2 Manual
Responsables: Joseba, David

Descripción: Explicación sobre el uso de todo el software implementado para que cualquier usuario pueda ejecutar y emplearlo
Entrada: Los dos juegos terminados junto a los módulos de minería
Entregables: Manual PDF sobre el uso de las diferentes partes implementadas
Recursos: Procesador de texto
Precedencias: 3(4) Prototipo 1, 5 Prototipo 2, 6 Prototipo 3

Paquete de trabajo: 1.2.3 Herramientas
Responsables: Joseba, David

Descripción: Recopilación de todo el software y hardware necesario para la finalización de este proyecto
Entrada: Ninguna
Entregables: Documento de texto
Recursos: Procesador de texto
Precedencias: 3(4) Prototipo 1, 5 Prototipo 2, 6 Prototipo 3

Paquete de trabajo: 1.2.4 Bibliografía
Responsables: Joseba, David

Descripción: Para realizar cualquier trabajo hace falta documentarse, las referencias y páginas consultadas deben tenerse en cuenta y nombrarlas
Entrada: Ninguna
Entregables: Documento
Recursos: Procesador de texto, internet
Precedencias: 3(4) Prototipo 1, 5 Prototipo 2, 6 Prototipo 3

Paquete de trabajo: 1.2.5 Preparación presentación
Responsables: Joseba, David

Descripción: Una vez todo terminado hay que mostrar lo logrado, para ello hará falta preparar videos, y pases de diapositivas teniendo en cuenta un guion.
Entrada: Los dos juegos terminados junto a los módulos de minería
Entregables: Diapositivas, videos y guion
Recursos: Procesador de texto, herramienta de captura de imagen, procesador de diapositivas
Precedencias: 3(4) Prototipo 1, 5 Prototipo 2, 6 Prototipo 3

Investigación

Tutoriales Unity

Paquete de trabajo: 2.1.1 Comunicación Unity BD
Responsables: Joseba, David

Descripción: Necesitamos recopilar datos, y para ello debemos enviarlos desde Unity a un servidor, hay que investigar la mejor forma
Entrada: Ninguna
Entregables: Documento de texto
Recursos: Procesador de texto, internet, Unity, Base de datos
Precedencias: Ninguna

Paquete de trabajo: 2.1.2 Métodos básicos para juegos
Responsables: Joseba, David

Descripción: Nos enfrentamos a una plataforma nueva y queremos diseñar cosas que nunca hemos hecho, para ello necesitaremos investigar y realizar diferentes tutoriales que nos permitan adquirir el conocimiento necesario
Entrada: Ninguna
Entregables: Ninguno
Recursos: Unity, internet
Precedencias: Ninguna

Psicología

Paquete de trabajo: 2.2.1 Perfiles psicológicos
Responsables: Joseba, David

Descripción: Que diferentes tipos de personas existen, ¿Podríamos hacer una clasificación?
Entrada: Ninguna
Entregables: Documento de texto
Recursos: Procesador de texto, internet
Precedencias: Ninguna

Paquete de trabajo: 2.2.2 Elementos definitorios
Responsables: Joseba, David

Descripción: De los tipos de personas que existen, que es lo que les hace diferentes. Investigar qué cualidades hace a cada persona diferente y la pone en un grupo
Entrada: Ninguna
Entregables: Documento
Recursos: Procesador de texto, internet
Precedencias: 2.2.1 Perfiles psicológicos

Paquete de trabajo: 2.2.3 Elementos representativos
Responsables: Joseba, David

Descripción: Sabiendo lo que define a una persona, ¿cómo podemos recopilarlo desde un juego? Resumen de los datos que podemos extraer desde nuestros juegos
Entrada: Ninguna
Entregables: Documento
Recursos: Procesador de texto
Precedencias: 2.2.2 Elementos definitorios, 3.1(4.1) Análisis

Prototipo 1 Juego

Análisis

Paquete de trabajo: 3(4).1.1 Casos de uso
Responsables: Joseba / David

Descripción: Descripción de los pasos y actividades que conlleva el proyecto
Entrada: Ninguna
Entregables: Diagrama de Casos de uso
Recursos: Procesador de texto, Visual Paradigm
Precedencias: Ninguna

Paquete de trabajo: 3(4).1.2 Casos de uso expandidos
Responsables: Joseba / David

Descripción: Ampliar los casos de uso especificando paso por paso cada uno de ellos
Entrada: Casos de uso
Entregables: Documento de texto
Recursos: Procesador de texto, Visual Paradigm
Precedencias: 3(4).1.1 Casos de uso

Paquete de trabajo: 3(4).1.3 Pantallas
Responsables: Joseba / David

Descripción: Bocetos de las diferentes pantallas de las que constará el juego
Entrada: Casos de uso expandidos
Entregables: Croquis y bocetos escaneados
Recursos: scaner y procesadores de imágenes
Precedencias: 3(4).1.2 Casos de uso expandidos

Diseño

Paquete de trabajo: 3(4).2.1 Diagrama de clases
Responsables: Joseba / David

Descripción: Diagrama que describe la estructura del sistema
Entrada: Casos de uso y casos de uso expandidos
Entregables: Diagrama de clases
Recursos: Procesador de texto, Visual Paradigm
Precedencias: 3(4).1.1 Casos de uso, 3(4).1.2 Casos de uso expandidos

Paquete de trabajo: 3(4).2.2 Diagrama de comunicación
Responsables: Joseba / David

Descripción: Diagrama que describe la interacción entre las clases
Entrada: Diagrama de clases
Entregables: Diagrama de secuencia
Recursos: Procesador de texto, Visual Paradigm
Precedencias: 3(4).2.1 Diagrama de clases

Paquete de trabajo: 3(4).2.3 Diagramas de secuencia
Responsables: Joseba / David

Descripción: Diagrama para modelar interacción entre objetos
Entrada: Ninguna
Entregables: Diagrama de Casos de uso y casos de uso expandidos
Recursos: Procesador de texto, Visual Paradigm
Precedencias: 3(4).2.2 Diagrama de comunicación, 3(4).2.1 Diagrama de clases

Paquete de trabajo: 3(4).2.4 Pruebas
Responsables: Joseba / David

Descripción: Diseñar las pruebas que deberá superar el software para comprobar que funciona correctamente
Entrada: Análisis
Entregables: Documento
Recursos: Procesador de texto
Precedencias: 3.1(4.1) Análisis

Diseño gráfico

Paquete de trabajo: 3(4).3.1 Personajes
Responsables: Joseba / David

Descripción: Diseño y dibujo de los personajes de los juegos, véase protagonistas, naves, enemigos, etc…
Entrada: Ninguna
Entregables: Dibujos
Recursos: Procesador de imágenes
Precedencias: Ninguna

Paquete de trabajo: 3(4).3.2 Fondos
Responsables: Joseba / David

Descripción: Diseño y dibujo de los fondos del juego, suelos, cielo, etc…
Entrada: Ninguna
Entregables: Dibujos
Recursos: Procesador de imágenes
Precedencias: Ninguna

Paquete de trabajo: 3(4).3.3 Interfaz
Responsables: Joseba / David

Descripción: Durante el juego hay que mostrar puntos, vidas, etc, diferente información necesaria para el desarrollo de la partida
Entrada: Ninguna
Entregables: Dibujos
Recursos: Procesador de imágenes
Precedencias: Ninguna

Paquete de trabajo: 3(4).3.4 Menú
Responsables: Joseba / David

Descripción: Diseño y dibujo de los menús del juego
Entrada: Ninguna
Entregables: Dibujos
Recursos: Procesador de imágenes
Precedencias: Ninguna

Desarrollo

Paquete de trabajo: 3(4).4.1 Lógica de negocio
Responsables: Joseba / David

Descripción: Implementación de todo lo diseñado
Entrada: todo el diseño
Entregables: Software
Recursos: Internet, Unity, Git
Precedencias: 3(4).2 Diseño, 3(4).3 Diseño de gráfico

Paquete de trabajo: 3(4).4.2 Gráficos
Responsables: Joseba / David

Descripción: Todos los gráficos pensados en la fase de diseño hay que realizarlos para conseguir un resultado vistoso
Entrada: Diseño de todos los gráficos
Entregables: Archivos de imágenes
Recursos: Procesador de imágenes
Precedencias: 3(4).3 Diseño gráfico

Paquete de trabajo: 3(4).4.3 Pruebas
Responsables: Joseba / David

Descripción: Implementar las pruebas diseñadas
Entrada: Juego implementado
Entregables: Software
Recursos: Internet, Unity, Git
Precedencias: 3(4).4.1 Juego, 3(4).2 Gráficos

Prototipo 2 (Recopilar y guardar)

Investigación

Paquete de trabajo: 5.1.1 Pruebas con usuarios
Responsables: Joseba, David

Descripción: Cuando tengamos el prototipo 2 terminado, necesitamos usuarios que lo utilicen a fin de recopilar información de ellos, también debemos observar su interacción con el programa para saber si todo funciona correctamente
Entrada: Juegos con prototipo 2 funcionando
Entregables: Base de datos con información recopilada
Recursos: Juegos, personas
Precedencias: 3(4) Prototipo 1, 5.3 Desarrollo

Paquete de trabajo: 5.1.2 Privacidad de datos
Responsables: Joseba, David

Descripción: Vamos a recopilar muchos datos de los usuarios, por ello es esencial antes de hacerlo ver cómo podemos hacerlo, que podemos y que no podemos guardar y que documento necesitamos para pedirles permiso
Entrada: ninguna
Entregables: Documento de texto
Recursos: Internet, procesador de texto
Precedencias: ninguna

Diseño

Paquete de trabajo: 5.2.1 Diagramas entidad relación
Responsables: Joseba, David

Descripción: Modelo para representar las entidades relevantes en el sistema de información
Entrada: ninguna
Entregables: Diagrama entidad relación
Recursos: Editor de diagramas
Precedencias: 3(4)Prototipo 1

Paquete de trabajo: 5.2.2 Pruebas
Responsables: Joseba, David

Descripción: Pruebas para comprobar que la recopilación de datos funciona correctamente
Entrada: Ninguna
Entregables: Documento de texto
Recursos: Editor de texto
Precedencias: 5.2.1 Diagramas entidad relación

Desarrollo

Paquete de trabajo: 5.3.1 Lógica de negocio
Responsables: Joseba, David

Descripción: Programación de todo lo diseñado
Entrada: Diseño del prototipo 2
Entregables: Software
Recursos: Unity, internet
Precedencias: 5.2 Diseño

Paquete de trabajo: 5.3.2 Pruebas
Responsables: Joseba, David

Descripción: Implementar las pruebas diseñadas
Entrada: Diseño de pruebas
Entregables: Pruebas a realizar o programadas
Recursos: Unity, internet
Precedencias: 5.2 Diseño

Prototipo 3 (Clasificar y mostrar)

Investigación

Paquete de trabajo: 6.1.1 Algoritmos de clasificación
Responsables: Joseba, David

Descripción: Una vez tenemos todos los datos recopilados sobre la interacción de los usuarios con los juegos deberemos ver como clasificar estos datos, que método nos aporta más información, ¿Cuál vamos a usar?
Entrada: Archivo .arff con los datos de los usuarios
Entregables: Documento con las conclusiones
Recursos: Weka, archivo .arff
Precedencias: 5.1.1 Pruebas con usuarios

Paquete de trabajo: 6.1.2 Pruebas con usuarios
Responsables: Joseba, David

Descripción: Una vez clasificado los usuarios deberemos probar si les mostramos correctamente o predecimos su clase directamente después de jugar
Entrada: Software completo
Entregables: Documento con conclusiones
Recursos: Personas, programa
Precedencias: 6.3 Desarrollo

Diseño

Paquete de trabajo: 6.2.1 Diagramas entidad relación
Responsables: Joseba, David

Descripción: Modelo para representar las entidades relevantes en el sistema de información
Entrada: ninguna
Entregables: Diagrama entidad relación
Recursos: Editor de diagramas
Precedencias: 5 prototipo 2 (Recopilar y guardar)

Paquete de trabajo: 6.2.2 Pruebas
Responsables: Joseba, David

Descripción: Pruebas para comprobar que la recopilación de datos funciona correctamente
Entrada: Ninguna
Entregables: Documento con las pruebas a realizar y programar
Recursos: Editor de texto
Precedencias: 6.2.1 Diagramas entidad relación

Desarrollo

Paquete de trabajo: 6.3.1 Lógica de negocio
Responsables: Joseba, David

Descripción: Programación de todo lo diseñado
Entrada: Diseño del prototipo 3
Entregables: Todo programado
Recursos: Weka
Precedencias: 6.2 Diseño

Paquete de trabajo: 6.3.2 Pruebas
Responsables: Joseba, David

Descripción: Implementación de las pruebas diseñadas
Entrada: Diseño de las pruebas
Entregables: Pruebas implementadas
Recursos: Weka, aplicación
Precedencias: 6.2.2 Pruebas, 6.3.1 Lógica de negocio

Análisis de riesgos

En todo proyecto ocurren situaciones inesperadas que cambian la planificación del mismo. Para amortiguar estos imprevistos se analizan los distintos riesgos a los se encuentra expuesto el proyecto y se establecen planes de contingencia para cuando surja alguno.

El riesgo se mide asumiendo la vulnerabilidad frente a cada tipo de peligro, para realizar este estudio, se acota la probabilidad de ocurrencia y el impacto de la siguiente manera:

Medidas de probabilidad de ocurrencia.

- Baja [0-33 %].
- Media [34-66 %].
- Alta [67-100 %].

Medidas de impacto sobre la planificación del proyecto.

- Bajo.
- Medio.
- Alto.

Riesgos específicos

Descripción	No encontrar teoría sobre psicología acorde a lo que queremos extraer del juego
Prevención	Dedicar suficiente tiempo antes de empezar el proyecto para evitar el problema en un futuro
Plan de contingencia	Basarnos en otros proyectos para buscar cosas que si sabemos que podemos obtener
Probabilidad	Media
Impacto	Alto

Descripción	No reunir usuarios de prueba suficiente con lo que no tendremos suficientes datos
Prevención	Tener tiempo suficiente para reunir más gente, pero esto queda fuera del alcance del proyecto y quedaría como trabajo futuro
Plan de contingencia	Utilizar desconocidos en la universidad
Probabilidad	Alta
Impacto	Alto

Descripción	No conseguir información concluyente de los datos conseguidos
Prevención	Enfocar los juegos a extraer datos útiles basados en la bibliografía existente
Plan de contingencia	Encontrar conclusiones de porque esto ha pasado
Probabilidad	Alta
Impacto	Alto

Descripción	Ambos juegos aportan exactamente la misma información sobre puntos de la personalidad del usuario
Prevención	Intentar conseguir datos sobre diferentes cosas en ambos juegos generando niveles que sean sustancialmente diferentes
Plan de contingencia	Crear nuevos niveles variados
Probabilidad	Media
Impacto	Medio

Descripción	Usuario pide que borremos sus datos
Prevención	Informar y pedir aceptar un documento explicando para que se usarán todos los datos obtenidos
Plan de contingencia	Borrar los datos dado su usuario
Probabilidad	Baja
Impacto	Bajo

Descripción	Un usuario crea más de una cuenta
Prevención	Informar del modo de uso y la utilidad de tener una sola cuenta
Plan de contingencia	Eliminar datos duplicados
Probabilidad	Baja

Impacto	Bajo

Descripción	Usuario no se toma en serio los test y sus datos no son validos
Prevención	Concienciar al usuario de la finalidad del proyecto
Plan de contingencia	Eliminar datos “outlayer” en weka
Probabilidad	Media
Impacto	Bajo

Descripción	El usuario no comprende como jugar correctamente
Prevención	Obligar al usuario a realizar un tutorial del juego hasta que lo complete correctamente
Plan de contingencia	Modificar el tutorial con las opiniones de los usuarios
Probabilidad	Media
Impacto	Medio

Descripción	Falta de comunicación entre los miembros del equipo
Prevención	Reuniones semanales
Plan de contingencia	Reunión extra para sincronizar
Probabilidad	Baja
Impacto	Medio

Descripción	Corrupción de la base de datos
Prevención	Realizar copias de seguridad
Plan de contingencia	Cargar copias de seguridad
Probabilidad	Media
Impacto	Alto

Riesgos Comunes

Descripción	Perdida de hardware
Prevención	Correcto cuidado del material
Plan de contingencia	Ordenador de repuesto
Probabilidad	Media
Impacto	Medio

Descripción	Perdida de software
Prevención	Copias de seguridad
Plan de contingencia	Cargar copia de seguridad
Probabilidad	Media
Impacto	Alto

Descripción	Perdida de días de trabajo por enfermedad o asuntos personales
Prevención	Margen de error en el diseño del Gantt
Plan de contingencia	Dedicar horas extras para recuperar horas perdidas y utilizar fines de semana
Probabilidad	Media
Impacto	Medio

Descripción	Perdida de días por contrato laboral
Prevención	Margen de error en el diseño del Gantt
Plan de contingencia	Retrasar la totalidad del gantt correspondiendo a la duración del contrato laboral
Probabilidad	Media
Impacto	Medio

Descripción	Falta de lugar físico para las reuniones de grupo
Prevención	Buscar sitios alternativos
Plan de contingencia	Reuniones por Skype
Probabilidad	Baja
Impacto	Bajo

Planificación temporal

Gantt

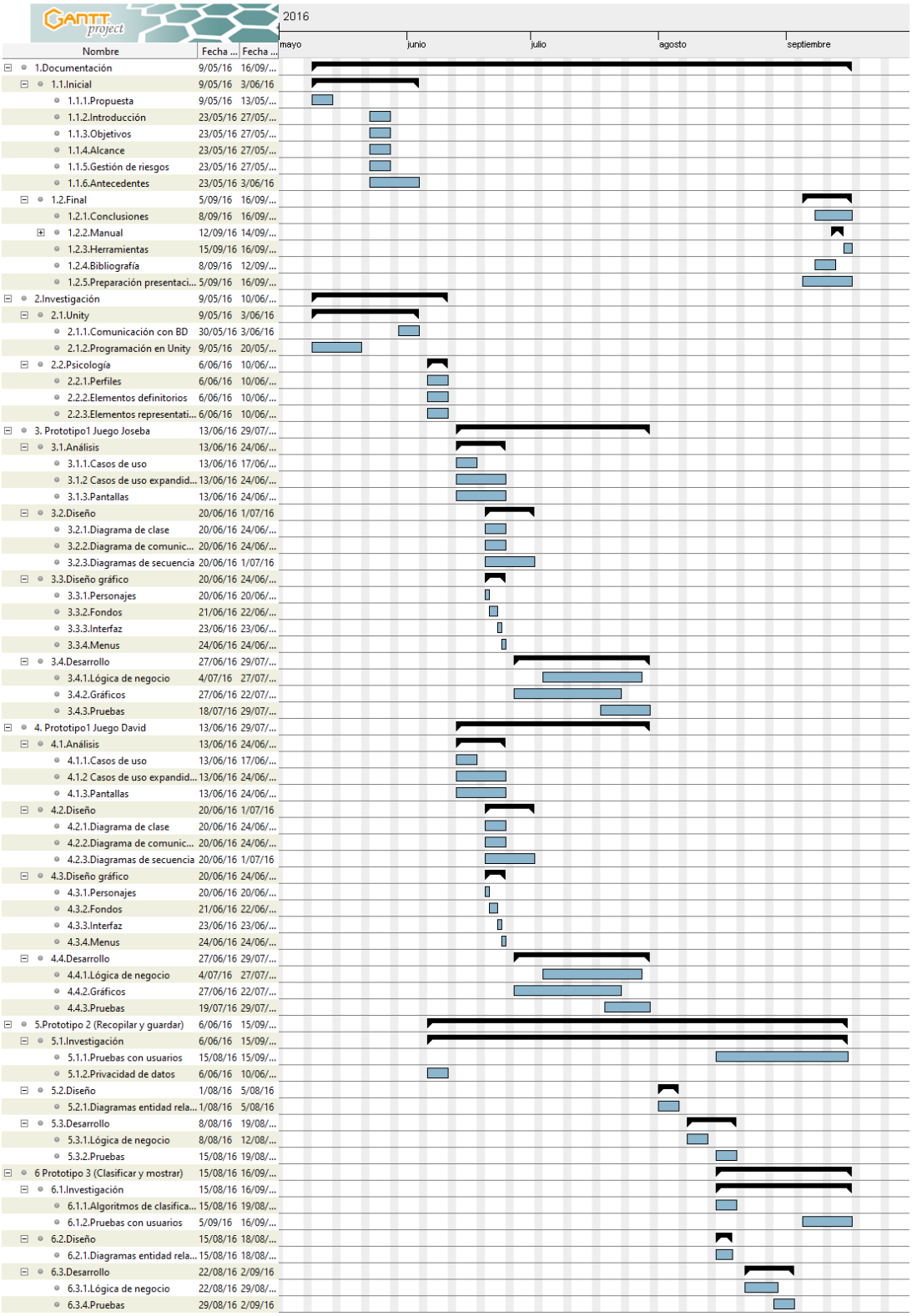
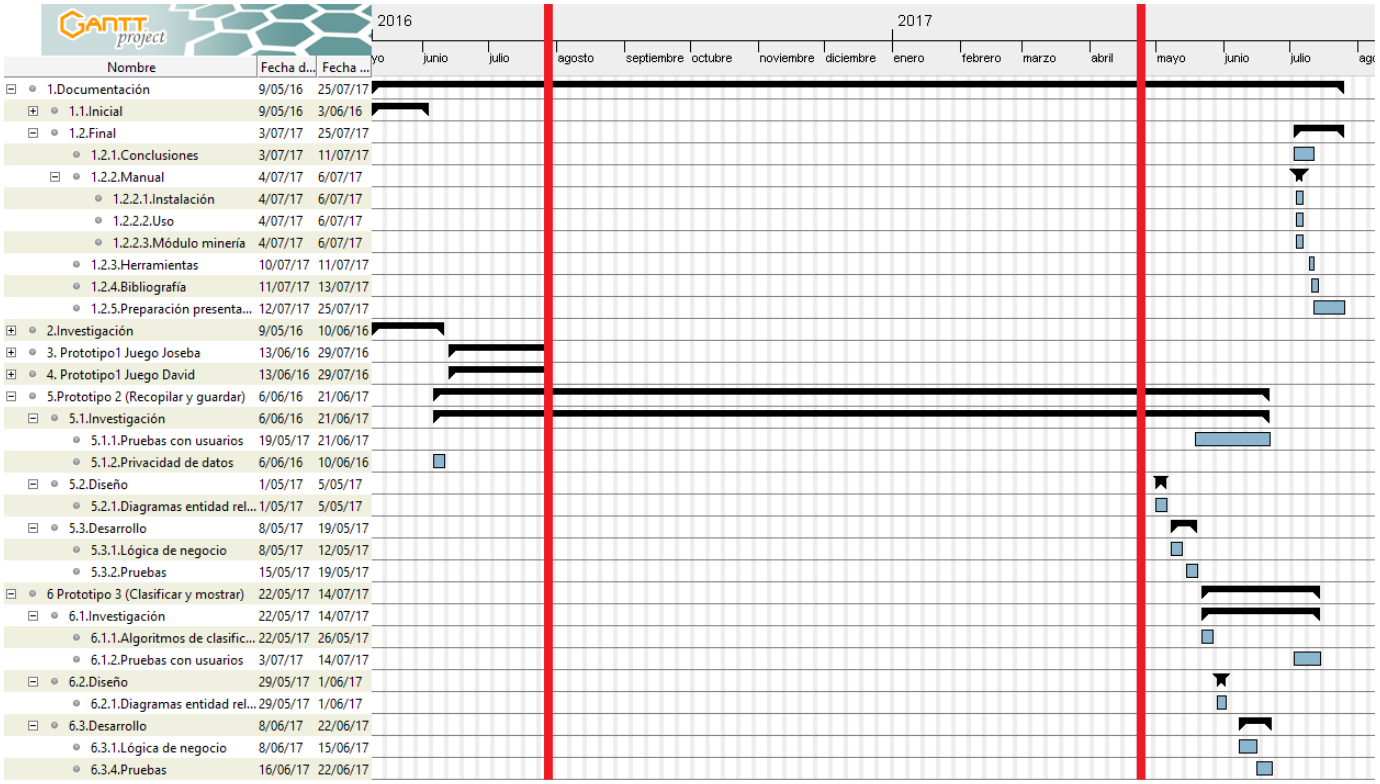


Figura 31: Planificación temporal - Gantt inicial

Gantt final



Nombre	Fecha d...	Fecha ...
1.Documentación	9/05/16	25/07/17
1.1.Inicial	9/05/16	3/06/16
1.2.Final	3/07/17	25/07/17
1.2.1.Conclusiones	3/07/17	11/07/17
1.2.2.Manual	4/07/17	6/07/17
1.2.2.1.Instalación	4/07/17	6/07/17
1.2.2.2.Uso	4/07/17	6/07/17
1.2.2.3.Módulo minería	4/07/17	6/07/17
1.2.3.Herramientas	10/07/17	11/07/17
1.2.4.Bibliografía	11/07/17	13/07/17
1.2.5.Preparación presenta...	12/07/17	25/07/17
2.Investigación	9/05/16	10/06/16
3. Prototipo1 Juego Joseba	13/06/16	29/07/16
4. Prototipo1 Juego David	13/06/16	29/07/16
5.Prototipo 2 (Recopilar y guardar)	6/06/16	21/06/17
5.1.Investigación	6/06/16	21/06/17
5.1.1.Pruebas con usuarios	19/05/17	21/06/17
5.1.2.Privacidad de datos	6/06/16	10/06/16
5.2.Diseño	1/05/17	5/05/17
5.2.1.Diagramas entidad rel...	1/05/17	5/05/17
5.3.Desarrollo	8/05/17	19/05/17
5.3.1.Lógica de negocio	8/05/17	12/05/17
5.3.2.Pruebas	15/05/17	19/05/17
6 Prototipo 3 (Clasificar y mostrar)	22/05/17	14/07/17
6.1.Investigación	22/05/17	14/07/17
6.1.1.Algoritmos de clasific...	22/05/17	26/05/17
6.1.2.Pruebas con usuarios	3/07/17	14/07/17
6.2.Diseño	29/05/17	1/06/17
6.2.1.Diagramas entidad rel...	29/05/17	1/06/17
6.3.Desarrollo	8/06/17	22/06/17
6.3.1.Lógica de negocio	8/06/17	15/06/17
6.3.4.Pruebas	16/06/17	22/06/17

Figura 32: Planificación temporal - Gantt Final

A finales de julio por una mezcla de problemas personales y motivos laborales el proyecto sufrió una pausa en septiembre del 2016 (la pausa y el retorno viene marcada en rojo), el mes de mayo se ha retomado el proyecto donde se dejó sufriendo el Gantt como se previó en los riesgos un desplazamiento, este ha sido de 8 meses.

Evaluación Económica

Una vez claro el coste temporal del proyecto representado mediante el Gantt se puede calcular cual sería el coste económico del mismo. (4 meses 1 semana)

Coste hardware

Para la realización del proyecto cada uno de los programadores dispone de un ordenador, en este caso se ha utilizado el ordenador personal estimado en unos 1000€.

El tiempo de amortización son unos 5 años, podemos saber que en 4 meses y 1 semana, que es la duración del proyecto, podríamos imputar un gasto de 70,83€ por programador.

Coste hardware: 141,66€.

Coste software

El software utilizado es en su mayoría gratuito. Solo hay unos pocos programas de pago como son Microsoft Word, Photoshop y Visual Paradigm.

Microsoft Word se paga mensualmente, ahora mismo su precio son 7€ mes. Dado que solo uno de los programadores necesita disponer de él calculamos un coste de 28€.

En el caso de Photoshop, hay un paquete especial para estudiantes, su coste es de 19,66€ mes. El diseño de los gráficos se realiza en unos dos meses. Su coste serían 39,32€.

Visual Paradigm dispone de licencias flexibles, la más económica capaz de cubrir todas las necesidades cuesta 6€ mes. Ambos programadores necesitan el programa, en total son 48€.

También se ha utilizado un servidor contratado por uno de los dos desarrolladores, su precio es 5€ mes, total 20€

Coste Software :135,32

Coste personal

Como se ha observado en el apartado de planificación temporal, el proyecto se realizará en un periodo de tiempo de 4 meses y una semana, eliminando los fines de semana. En este tiempo están incluidas las reuniones con el cliente.

El horario se supone de 8 horas diarias, siendo en 17 semanas 85 días laborales, las horas totales son 680 horas por programador.

Considerando la categoría profesional de Ingeniero recién salido de la carrera, cuyo salario es de 15 € la hora, se obtendría el siguiente coste:

680 x 15 = 10.200 €

Teniendo en cuenta que son dos desarrolladores:

Coste Personal: 20.400€

Coste total

Coste Total: 20.601€

Siendo juegos sencillos podría considerarse vender cada uno por unos 3€. Para conseguir amortizar la inversión haría falta vender 6893 juegos.

Aunque la cantidad de juegos a vender es elevada el objetivo de este proyecto no es la venta del juego en sí, es la investigación. Si se demostrará la relación de la personalidad con la forma de jugar y se encontrará un beneficio de esa información las ganancias serían mayores.

Antecedentes

Análisis de personalidad mediante videojuegos

Según el diccionario Oxford la psicología es la ciencia que trata la conducta y procesos mentales de las personas (en.oxforddictionaries.com). Siempre ha sido un objetivo discretizar los procesos mentales de una persona y definirlos por hechos como su nacimiento, lugar de crianza, raza o aspectos claramente diferenciables. El problema surge cuando dos personas con un entorno similar desarrollan personalidades totalmente opuestas. Intentar analizar una persona es un trabajo complicado que requiere tener en cuenta infinidad de aspectos y matices junto a un experto en la materia.

Tener conocimiento sobre los rasgos de personalidad de un sujeto permite optimizar el entorno, definir cuál es su mejor puesto de trabajo, que cosas necesitará para satisfacer sus necesidades emocionales y en último caso permite saber lo que quiere a fin de poder dárselo.

El problema es, como se ha dicho anteriormente, que para definir a una persona la información necesaria es enorme. ¿Dónde se pueden conseguir todos estos datos?

A raíz del uso global de internet y el nacimiento del **Big data** esta traba está desapareciendo y por supuesto las empresas se han interesado en obtener beneficios de toda esta información. Aunque pensemos que apenas se aporta nada o se intente reducir al mínimo los datos introducidos, en el momento que se accede a la web estos están siendo recogidos. En el artículo "Data Collection Techniques" (whonix.org) se explica como con las *cookies* o *javascript* se puede seguir el recorrido de un usuario por la web y extraer información de ellos, impulsando para evitarlo, un sistema operativo aislado y anónimo. Los datos obtenidos en internet en un principio han sido utilizados sobre todo para personalizar la publicidad, por ejemplo, el código postal sirve para hacer campañas de geomarketing (www.pymesyautonomos.com).

De estos datos simples se ha pasado a analizar el comportamiento del usuario, es decir, no acciones aisladas sino un recorrido. En el artículo "Behavioral Data Collection" (https://ecsd-fl.schoolloop.com/best/data) de la escuela del distrito de Escambia, Florida, se sugiere recopilar diferentes datos como frecuencias de acceso, duración de las visitas e intervalos de una visita a otra. Estos nuevos datos dan acceso a una nueva gama de información.

A medida que la minería de datos ha evolucionado se ha querido dar un paso más allá, ya no solo se intenta vender algo, ¿es posible saber cómo es una persona a partir de su interacción con internet?, si se sabe cómo es, será mucho más fácil ofrecerle lo que quiere. Hay aplicaciones que muestran características notables sobre la personalidad de las personas, las más importantes son las redes sociales. En ellas la gente introduce información de su vida personal muy significativa. En la investigación "Mining Facebook Data for Predictive Personality Modeling" (www.michalkosinski.com) se clasifica un grupo de 250 personas según el contenido de su página de Facebook, en un estudio inicial se obtuvieron buenos resultados indicando que con una mayor inversión se podría acotar mucho los rasgos de una persona.

Por supuesto saber cómo es alguien teniendo datos de toda una vida parece factible, pero ¿es posible acercarnos a la misma información con datos aparentemente sencillos? Por ejemplo para entrevistas de trabajo se utilizan métodos muy simples, uno es dibujar un animal en una hoja, dependiendo del tamaño, la posición, y rasgos aparentemente poco significantes se encuadra a la persona para saber qué tipo de trabajador es.

Ahora mismo uno de los sectores con mayores beneficios y que más dinero mueve es el de los videojuegos. ¿Puede implantarse todo lo dicho anteriormente en ellos? La realidad es que ya se hace. Juegos como Candy Crush (youtube.com/ExtraCredits) utilizan la minería para analizar cuando se aburre una persona, así consiguen que la gente no deje de jugar. Una de las plataformas principales de distribución digital de videojuegos es Steam (Steampowered.com), existen páginas con gráficas y estadísticas utilizadas para mejorar la experiencia de juego. Todo esto consigue vender más, pero no queremos vender, queremos saber cómo es la persona, ¿es posible?

Existen juegos cuyo objetivo es ese, saber cómo es la persona. En su mayoría son juegos educacionales. El inconveniente es que estos juegos no son atrayentes para un gran público, tienen un solo objetivo, ¿quiere decir esto que para las grandes empresas la psicología no es importante? Es falso, hay compañías que tienen psicólogos empleados para este fin. Se encargan de analizar los sectores de usuarios a fin de mejorar ventas. Esto nos confirma que la psicología en los videojuegos es un campo de interés real y en evolución.

La tesis doctoral titula "Quantifying Individual Player Differences" emplea cuatro juegos para analizar a los usuarios. Los cuatro juegos utilizados están especialmente seleccionados o implementados para obtener diferentes aspectos de los usuarios. Cada juego se centra en una característica como puede ser la capacidad de socialización o la perseverancia. Al final de la tesis se llega a la conclusión de que mediante juegos si se pueden extraer diversos aspectos del usuario. Se utilizará este documento como base para el proyecto.

Quantifying Individual Player Differences

En el año 2013 Giel van Lankveld presenta la tesis doctoral "Quantifying Individual Player Differences" (van Lankveld, G., 2013), en ella profundiza en la capacidad que tienen los videojuegos para discretizar la personalidad. Se utilizará esta tesis como base de aprendizaje para este proyecto y se recorrerán los puntos más importantes y de aplicación más práctica.

Aunque se espere que los resultados reflejen la personalidad, realmente enseñan el desempeño en el juego. Si realmente se quiere conocer o cuantificar el jugador es necesario analizar el comportamiento durante el mismo. El comportamiento o interacción del jugador se centrará en tres procesos, la cognición (pensamiento del jugador), la percepción (observación de los elementos del juego) y la capacidad (habilidad o manejo del medio). De las tres solo una describe la personalidad, la cognición.

Los datos que pueden ser recogidos son las pulsaciones de teclado o interacciones con los periféricos de entrada, por si solas no tienen significado, pero dentro de un contexto aportan información. Además debe conseguirse separar la habilidad adquirida del jugador de su personalidad, es decir no fiarse en el desempeño sino de porque obtiene esos resultados.

En la informática y en la inteligencia artificial se trata de crear un modelo de la personalidad del usuario, sabiendo cómo es un jugador se puede mejorar su interacción con el juego haciéndolo más divertido. Para este fin harían falta tanto un experto en informática como uno en psicología. Un equipo mixto aportaría mucho a un proyecto como este pero también es difícil de reunir y por ello se presentan varias limitaciones, es posible cometer errores en el apartado de psicología y su definición.

El objetivo final es obtener un modelo psicológico del usuario, en la investigación de videojuegos se buscan modelos con otros dos propósitos totalmente diferentes: saber porque un jugador reacciona como lo hace y mejorar la experiencia de juego. Los juegos fallan en poder crear un modelo general. Es decir lo que aplica en un juego no suele servir para otro.

Para llegar al objetivo se intenta extender métodos existentes en el campo de la psicología e introducirlos al juego. Es decir enlazar métodos psicológicos a desempeños del juego.

La metodología utilizada es primero realizar cuestionarios. Se hacen cuatro cuestionarios:

1. General: Edad, sexo, estudios...
2. Incongruencia: Emociones resultantes entre la dificultad y la habilidad
3. Emocional: Aburrimiento, frustración y placer
4. Personalidad: NEO-PI-R, test de personalidad

También existen video observaciones y entrevistas, dado que en el equipo no hay expertos en psicología no es posible centrarse en este tipo de métodos.

La conclusión es la necesidad de un test escrito, se debe buscar un cuestionario que requiera un tiempo asequible para ser completado y aporte un perfil aproximado de la persona. En la tesis se utiliza el método NEO-PI-R comentado anteriormente, método que consta de más de 240 elementos que dividen la personalidad en cinco facetas principales.

Una vez se tienen los resultados de los cuestionarios se contrastan con el juego, en el caso de la tesis se utilizan tres juegos diferentes dando un total de cuatro escenarios de los que recoger datos. Las variables recogidas son seleccionadas de forma aleatoria, prácticamente se recoge todo dato pensado como pueden ser tiempos, pulsaciones, elecciones, etc.

Contrastando las variables recogidas con los datos del test se consiguen resultados positivos demostrando que los videojuegos pueden acercarnos a la personalidad del usuario. De las cinco facetas en las que divide la personalidad consigue resultado de más del 70% en cuatro de ellas. De más de las 150 variables utilizadas solo poco más de la mitad han aportado información útil, para evitar esto al diseñar el proyecto se buscara forzar escenarios que puedan aportar información útil.

Las conclusiones del documento y acciones que aplicaremos a nuestro proyecto son las siguientes:

1. Buscar un test de personalidad utilizado fuera del campo de la informática
2. Buscar que posibles variables pueden ser extraídas desde el programa
3. Aplicar algoritmos de minería para buscar correlación

Para poder aplicar los anteriores objetivos es necesario profundizar en la psicología y posteriormente analizar el propio videojuego presentado para el proyecto.

Estudio sobre psicología

Durante el proyecto se quiere estudiar si es posible unir el comportamiento o la interacción de un usuario con su verdadera personalidad. Para ello es necesario investigar que es un perfil psicológico y como obtenerlo.

Perfil psicológico. Definición: Conjunto de patrones producidos por una persona que determinan la aptitud, carácter y actitud (en.oxforddictionaries.com).

¿Por qué dos personas actúan diferente ante una misma situación? ¿Porque un mismo problema no les afecta igual? Esto se debe a que la forma en que funciona la mente de cada sujeto es única. No existen dos personas iguales, luego es difícil tener dos reacciones iguales. Todos nacen con una personalidad única que se va desarrollando a lo largo de la vida aportando un carácter único. Esta personalidad se verá influenciada por situaciones intrínsecas a cada persona, como pueden ser edad, sexo o lugar de nacimiento; y por valores externos, como son las diferentes experiencias que tenemos que afrontar.

La personalidad es única e inherente a cada persona, luego no es posible determinarla del todo, pero sí clasificarla en ciertos parámetros. Existen muchos intentos para lograr esto, por ejemplo Freud (Sigmund Freud, 1921) teorizó sobre un inconsciente activo en toda persona que de forma involuntaria puede llevarnos a chistes, lapsus o actos fallidos . Freud intento clasificar las personas diferenciando los aspectos del inconsciente que intentan reprimir cada una de ellas.

Una de las formas más aceptadas a día de hoy para clasificar personalidades es mediante ciertos impulsos o aptitudes primarias como son la percepción, cognición, atención, emoción, inteligencia o motivación. Diferentes teorías seleccionan diferentes aptitudes y clasifican a las personas según su afinidad a cada una de ellas. Es decir generan diferentes grupos en los que siguen entrando un gran arco de personas nunca siendo muy específicos.

Para obtener la personalidad de un sujeto y saber valorar sus aptitudes hay diferentes métodos: observación, entrevistas, test, etc. Para la mayoría de ellos es necesario un profesional en la materia que pueda analizar perfectamente cada acción o respuesta. En este proyecto, siendo la primera vez que se trabaja en este campo, la opción más viable será el test escrito. El test escrito es una autoevaluación sencilla que no requiere participación externa ni supervisión, esta falta de supervisión crea un problema, el usuario debe ser honesto con sus respuestas si queremos que los resultados no sean falsos.

La teoría más apropiada encontrada ha sido el indicador Myers-Briggs. Es un test de personalidad creado por Katharine Cook Briggs y su hija Isabel Briggs Myers durante la segunda guerra mundial (Cook Briggs, K. and Briggs Myers, I., 1944). Los criterios que utilizaron se basan en las teorías de Carl Gustav Jung (Jung, C.G. 1921/1971). El test propuesto trata de cuatro pares de características, cada sujeto tomará preferencia en una u otra pudiendo pertenecer a uno de los dieciséis grupos.

Indicador Myers-Briggs

El objetivo del indicador Myers-Briggs es hacer la teoría de psicología de tipos descrita por Jung, entendible y útil para las vidas de las personas. Para ello hay que identificar las cuatro dicotomías especificadas por la teoría de Jung y describir los dieciséis tipos de personalidad resultantes.

Las cuatro dicotomías

1. **Actitud**: Cuál es su mundo favorito, prefieren centrarse en su yo interior o en lo que les rodea. Si se utiliza la energía para relacionarse con las personas, las cosas y las situaciones existe una preferencia a la extroversión. Si las energías se centran en ideas, información y creencias habrá una preferencia en la introversión.
    a. *Extroversión (E)*: Personas de acción, activas, necesitan amplitud, buscan personas y objetos.
    b. *Introversión (I)*: Personas reflexivas, buscan conceptos e ideas.

2. **Percepción**: Como la persona recopila información, se conforman con la información básica o necesitan interpretar y añadirle significado. Si se enfrenta a los hechos, a lo que ve, tiene preferencia en lo sensorial. Si lidia con ideas y busca nuevas posibilidades se inclina a lo intuitivo.
    a. *Sensorial (S)*: Confían en información disponible, tangible y concreta
    b. *Intuitivo (N)*: Obtienen la información de forma abstracta y teórica asociada a otras informaciones.

3. **Decisión**: Que se hace con la información disponible, se mira la información o las personas y sus circunstancias. Una persona que se esfuerza en tomar decisiones lógicas y objetivas será más racional, en cambio alguien que valora las creencias personales y principios será más emocional.
    a. *Racional (T)*: Toma de decisión lógica, objetiva y ajustada a las reglas
    b. *Emocional (F)*: Toma de decisión por empatía, subjetiva, tiene en cuenta las partes involucradas

4. **Estructura**: Al enfrentarse al mundo se prefiere tener las cosas resueltas o estar abierto a cambios o nueva información. Alguien con una vida planificada, estable y organizada será de tipo calificador. Alguien flexible que sigue la corriente de los acontecimientos será más perceptivo.
    a. *Calificador (J)*: Su función principal entre las dos anteriores es la decisión, quieren tener las cosas resueltas
    b. *Perceptivo (P)*: Le gusta mostrar al mundo sus ideas y percepción sin tener porque llevarlo a un fin.

Los dieciséis tipos

Cuando se decide una preferencia en cada categoría se llega a un tipo de personalidad, puede ser descrita como un código de cuatro letras teniendo cada uno de ellas unas características básicas.

1. **ISTJ**: Sistemáticos, organizados, lógicos, detallados, analíticos, responsables, pragmáticos, críticos, conservativos, estables, concretos y eficientes.

2. **ISFJ**: Cálidos, simpáticos, detallados, dependientes, organizados, pensativos, concienzudos, sistemáticos, conservativos, realistas, atentos, prácticos, estables, colaboradores.

3. **INFJ**: Orientados a la visión y el significado, intensos, perspicaces, creativos, sensitivos, armoniosos, serios, amantes del lenguaje, símbolos, perseverantes, inspiradores.

4. **INTJ**: Orientados a la visión, innovadores, perspicaces, conceptuales, lógicos, persiguen el entendimiento, críticos, decisivos, independientes, determinados, competentes, buscan la mejora.

5. **ISTP**: Lógicos, analíticos, prácticos, adaptables, curiosos, calmados, observadores, solucionadores de problemas, exactos, realistas, variados, aventureros, independientes.

6. **ISFP**: Amables, cariñosos, compasivos, adaptables, modestos, estéticos, observadores, leales, ayudantes, realistas, detallistas, espontáneos, disfrutan la acción.

7. **INFP**: Cariñosos, compasivos, buscan el significado y la armonía, creativos, idealistas, ayudantes, inquisitivos, disfrutan las ideas el lenguaje y la escritura, independientes y adaptables.

8. **INTP**: Lógicos, conceptuales, analíticos, objetivos, críticos, ingeniosos, complejos, curiosos intelectualmente, aman las ideas, buscan el entendimiento, cuestionadores, adaptables e independientes.

9. **ESTP**: Buscan emociones, activos, pragmáticos, directos, fáciles de tratar, observadores, concretos, realistas, adaptables, eficientes, analíticos, solucionadores de problemas, espontáneos, les gustan las aventuras.

10. **ESFP**: Enérgicos, sociables, prácticos, amistosos, cariñosos, expresivos, abiertos, entusiastas, buscan emociones, espontáneos, adaptables, observadores, generosos y amantes de la diversión.

11. **ENFP**: Entusiastas, imaginativos, enérgicos, creativos, amables, orientados al futuro, individualistas, optimistas, orientados a las posibilidades, abiertos, espontáneos, divertidos.

12. **ENTP**: Enérgicos, inventivos, entusiastas, abstractos, lógicos, teóricos, analíticos, complejos, ingeniosos, orientados al cambio, independientes y adaptables.

13. **ESTJ**: Organizadores activos, asertivos, interesados en los hechos, decisivos, prácticos, buscan resultados, analíticos, sistemáticos, concretos, críticos, responsables, toman el mando y tienen sentido común.

14. **ESFJ**: Activamente sociables, cálidos, armoniosos, preocupados, entusiastas, empáticos, orientados a la gente, prácticos, responsables, concretos, ordenados, concienzudos, cooperadores, apreciativos y leales.

15. **ENFJ**: Activamente sociables, entusiastas, armonizadores, cálidos, idealistas, empáticos, orientados a las posibilidades, perspicaces, cooperadores, imaginativos, concienzudos y apreciativos.

16. **ENTJ**: Organizadores, planificadores, centrados en la visión, decisivos, comienzan cosas, conceptuales, estratégicos, sistemáticos, asertivos, críticos, lógicos, organizados. Buscan la mejora y los logros.

Test Myers-Briggs

A fin de ayudar a los usuarios a saber su clase real para luego poder contrastarla con los datos dentro del juego se utilizara un test. Hay que señalar como verdadero o falso cada una de las oraciones. Después sumar los resultados positivos y consultar sobre que apartados se tienen preferencias obteniendo así uno de los dieciséis grupos.

1. Te enorgullece tu objetividad, a pesar del hecho de que algunos te acusan de ser frío e indiferente.
2. Necesitas recargar energías solo, después de reuniones, llamadas telefónicas o socialización; cuanto más intenso es el encuentro más agotado te sientes posteriormente.
3. Crees que el amor no puede ser definido; te molesta los que tratan de hacerlo.
4. No te importa tomar decisiones difíciles y no comprendes por que algunas personas se alteran por cosas que no son relevantes para el asunto que se está tratando.
5. Tiendes a hablar primero y pensar después; con frecuencia te regañas con cosas como "¿aprenderé alguna vez a mantener mi boca cerrada?"
6. Recuerdas los números y cifras más fácilmente que las caras y los nombres.
7. Conoces mucha gente y a muchos de ellos los consideras como amigos íntimos; te gusta incluir tanta gente como sea posible en tus actividades.
8. Consideras como una buena decisión la que toma en cuenta los sentimientos de otros.
9. Ensayas las cosas antes de decirlas; a menudo contestas con "lo tendré que pensar o le contesto más tarde"
10. No te importa leer o tener una conversación mientras se desarrolla otra actividad (como una conversación, la TV, Radio, etc.); en realidad puedes permanecer indiferente a esta distracción.
11. No dudas en retirar lo dicho si percibes que ha ofendido a alguien; como consecuencia eres acusado de no tener convicciones.
12. No te gusta que te obliguen a tomar decisiones; prefieres mantener tus opciones abiertas.
13. Eres accesible y trabas conversación fácilmente con amigos, compañeros del trabajo y extraños teniendo quizás un papel dominante en la conversación.
14. Prefieres la armonía a la claridad; el conflicto te abruma y tratas de evitarlo (cambiemos de tema, o démonos las manos y seamos todos amigos).
15. Te sientes frustrado cuando las personas te dan instrucciones poco claras o cuando alguien te dice "este es el plan general, nos ocuparemos de los detalles después".
16. En una conversación cambias a menudo de tema; el nuevo tema puede ser algo que te viene a la mente o que atrae su atención en ese momento.
17. Tiendes a que las cosas no sean definitivas, aunque no siempre.
18. Consideras más importante tener razón que caer bien; no crees necesario que la gente deba caerte bien para poder trabajar con ella y realizar un buen trabajo.
19. No te gustan las sorpresas y esto lo haces saber a los demás.

20. Tiendes a dar más crédito a cosas que son lógicas y científicas; por ej. hasta que no recibas más información que justifique los beneficios de este test, te mantendrás escéptico acerca de su utilidad.
21. Haces listas y las utilizas; si haces algo que no está en su lista puede que lo agregues a la misma sólo para poder tacharlo.
22. Conviertes todo trabajo en una diversión; si un trabajo no puede ser algo entretenido probablemente no sea digno de hacerse.
23. Encuentras que escuchar es más difícil que hablar; te gusta ser la estrella de la conversación, y te aburre cuando no puedes participar activamente en ella.
24. Te deleita el orden; tienes tu manera especial para guardar las cosas en su escritorio, en sus archivos o para colgar cosas en las paredes.
25. Prefieres generar ideas en un grupo que por tu cuenta; te sientes agotado si pasas mucho tiempo reflexionando sin tener la oportunidad de intercambiar tus ideas con otros.
26. Suelen acusarte de estar enojado cuando no lo estás; es sólo tu manera de expresar tu opinión.
27. Disfrutas prestando servicios necesarios a la gente aunque algunos se aprovechen de ti.
28. Te distraes fácilmente; te pierdes de casa al garaje.
29. Te gusta completar un trabajo hasta acabarlo y sacártelo de encima aun cuando sabes que deberás rehacerlo de nuevo más tarde para hacerlo bien.
30. Adoras explorar lo desconocido, aun cuando sea algo tan simple como el camino del trabajo a casa
31. Te extralimitas tratando de satisfacer las necesidades de otros; harás casi cualquier cosa para acomodar a otros incluso a expensas de tu propio confort.
32. No planificas una tarea hasta ver qué es lo que se requiere; la gente te acusa de ser desorganizado aunque sabes mejor qué es lo que hay que hacer.
33. Te pones en el lugar de los demás; eres quien en una reunión probablemente pregunta cómo afectará esto a la gente involucrada.
34. Tienes un lugar para cada cosa y no te sientes satisfecho hasta que cada cosa esté en su sitio.
35. A menudo te preguntas si alguien se preocupa por lo que quieres aunque tengas dificultad en decírselo a alguien.
36. Siempre tienes que esperar a los otros, quienes nunca parecen ser puntuales.
37. Crees que hablar de detalles aburridos es una pérdida de tiempo.
38. Dependes de tus descargas de adrenalina en el último minuto para cumplir con tus fechas límite; habitualmente cumples con la fecha límite aunque vuelvas loco a todo el mundo para lograrlo.
39. Piensas en varias cosas al mismo tiempo; a menudo tus amigos te señalan que estás "como ausente".
40. Recuerdas con más facilidad el rostro y las circunstancias en que conociste a alguien que su nombre.
41. Encuentras el futuro y sus posibilidades interesantes, más que atemorizantes; generalmente te atrae más a donde vas que donde estás.
42. "Sabes" que si cada uno hiciera lo que se supone debe hacer (y en el momento que se supone debe hacerlo) el mundo sería un lugar mejor.
43. Eres de la idea que hay que ver para creer; si alguno te dice que llegó el correo, no lo crees hasta que no está sobre tu escritorio.

44. Crees que el tiempo es relativo; no importa la hora a menos que la reunión, cena o evento haya comenzado sin ti.
45. Usas las palabras en forma literal; a menudo te ves en la necesidad de preguntar si lo que dicen es en serio o es un chiste.
46. Te despiertas por la mañana y sabes bastante bien cómo será tu día; tienes una agenda y la sigues; puede alterarse si las cosas no marchan como estaba planeado.
47. Te interesa saber cómo funcionan las cosas solo por placer.
48. Te gusta demostrar tu punto de vista por motivos de claridad; es habitual en ti discutir ambos puntos de vista en un debate simplemente para ampliar tu horizonte intelectual.
49. Te gusta expresar tus pensamientos o ideas sin interrupciones; dejas que otros hagan lo mismo, esperando que sea recíproco.
50. Eres capaz de mantenerte frío calmado y objetivo en situaciones donde todo el mundo está alterado.
51. Encuentras más fácil ver los árboles que el bosque; eres feliz de concentrarte en tu trabajo y no te preocupas acerca de cómo encaja éste en un esquema más amplio.
52. Tienes inclinación a las chanzas y los juegos de palabras.
53. Encuentras más satisfactorios aquellos trabajos que producen resultados tangibles; preferirías limpiar tu escritorio que pensar en lo que le depara el futuro de su carrera.
54. Desearías imponer tus ideas con más fuerza. Te molesta que otros digan antes cosas que estabas por decir.
55. Preferirías resolver una disputa basándose en lo que es justo y verdadero más que en lo que hace feliz a la gente.
56. Prefieres respuestas específicas a preguntas específicas; cuando preguntas la hora, prefieres que te digan 3:42, y no que falta un poco para las 4 o que es hora de salir.
57. Prefieres fantasear sobre cómo gastar tu próximo sueldo que sentarte a balancear tu cuenta bancaria.
58. Eres una persona de ideas firmes; si estas en desacuerdo con las personas prefieres decirlo que callar y que crean que está de acuerdo.
59. Te gusta concentrarse en lo que estás haciendo en este momento y generalmente no te preocupa lo que sigue; es más, prefieres hacer algo que pensar en ello.
60. No crees que la prolijidad sea importante, aunque preferirías tener las cosas en orden; lo importante es la creatividad, la espontaneidad y la capacidad de respuesta.
61. Necesitas aprobación de los colegas superiores y subordinados acerca de quién eres, de lo que haces, cómo luce y casi sobre todo lo demás; puedes pensar que haces un buen trabajo pero hasta que no escuches a alguien decírtelo, no lo creerás realmente.
62. Encuentras más atractivo buscar las relaciones y conexiones subyacentes a las cosas, más que aceptarlas tal como aparecen; siempre estás preguntando qué es lo que eso significa.

63. Es posible que creas que los que hablan mucho, sean charlatanes; cuando escuchas a otros conversando puede venirte a la mente que están perdiendo el tiempo.
64. Tiendes a dar respuestas generales a las preguntas; no comprendes por qué tanta gente no puede seguir tus instrucciones y te irrita cuando la gente te presiona en busca de especificaciones.
65. Eres percibido como una persona que escucha, pero sientes que otros toman ventaja de ello y se aprovechan.
66. Piensas que la fantasía es una mala palabra; dudas de la gente que parece dedicar demasiado tiempo a jugar con su imaginación.
67. A veces te han calificado de tímido; estés o no de acuerdo, puedes parecer a otros como alguien reservado y pensativo.
68. Consideras que las llamadas telefónicas son interrupciones bienvenidas; no dudas en usar el teléfono cuando tiene algo que decir o necesita ver a alguien.
69. Te gusta compartir ocasiones especiales sólo con alguna otra persona o quizás con algunos amigos íntimos.
70. Prefieres resultados con hechos y números que con ideas y teorías; prefieres escuchar las cosas en forma secuencial y no al azar.
71. Te gusta ir a reuniones y tiendes a manifestar tu opinión; en realidad te sientes frustrado si no te dan la oportunidad de expresar tu punto de vista.
72. Disfrutas de la paz y la tranquilidad de tener tiempo para ti mismo; tu tiempo privado se encuentra fácilmente invadido y tiendes a adaptarte desarrollando un alto poder de concentración.

Resultados:

- (E) 5 - 7 - 10 - 13 - 23 - 25 - 61 - 68 - 71
- (I) 2 - 9 - 49 - 54 - 63 - 65 - 67 - 69 - 72
- (S) 15 - 43 - 45 - 51 - 53 - 56 - 59 - 66 - 70
- (N) 37 - 39 - 41 - 44 - 47 - 52 - 57 - 62 - 64
- (T) 1 - 4 - 6 - 18 - 20 - 48 - 50 - 55 - 58
- (F) 3 - 8 - 11 - 14 - 27 - 31 - 33 - 35 - 40
- (J) 19 - 21 - 24 - 26 - 29 - 34 - 36 - 42 - 46
- (P) 12 - 16 - 17 - 22 - 28 - 30 - 32 - 38 – 60

Interpretación:

Para cada par perteneciente a cada una de las cuatro dicotomías se suman las preguntas respondidas como verdadero. Se comparan los pares y el resultado será la mayor de las dos (por ejemplo si obtiene más E, que I, el resultado será E, en caso de empate X). Con las cuatro dicotomías calculadas se tendrá la clase compuesta de cuatro letras (por ejemplo ESFP).

Privacidad de datos

En el proyecto es necesario recoger datos personales de los usuarios, como pueden ser nombre, edad o sexo. Cuando recogemos datos de esta índole debemos saber cómo tratarlos. Para conocer el reglamento a cumplir se acude a la Ley Orgánica 15/1999 (www.boe.es).

El artículo 2 de la LOPD dice que los ficheros mantenidos por personas físicas para actividades exclusivamente personales quedan exentos de esta ley. Este proyecto será utilizado con fines educativos y no distribuido, aun así se aplicarán varios de sus puntos por considerarlos positivos y necesarios.

Se consideran datos personales cualquier información concerniente a personas identificadas o identificables, en este caso se guardan datos como la edad o sexo, siendo claramente datos que hay que proteger.

El objetivo de esta ley es garantizar la protección y buen tratamiento de los datos de carácter personal. Hay dos personas responsables, la persona que utiliza estos datos y la que se encarga de guardarlos, en este caso ambas responsabilidades recaen en la misma persona.

Los datos recogidos deben ser utilizados solo para la finalidad para la que han sido recolectados siempre habiendo informado con antelación. El usuario debe saber que existe un fichero donde se recogen, para que, y quien va a poder acceder a esa información. Además al utilizar minería de datos se pueden obtener datos, sean veraces o no, que el usuario no ha aportado directamente, eso también debe serle advertido. Por supuesto ningún usuario cae en la obligación de tener que aportar ningún tipo de dato. Hasta que no existe un consentimiento no se debe recoger ninguna información.

Se debe tener especial cuidado también en aportar las medidas de seguridad suficientes para evitar la alteración, perdida y/o acceso no autorizado a la base de datos. En caso de no poder cumplir con ello se deben tener medidas de contingencia como es el cifrado de esos datos.

Con todo lo anterior en cuenta antes de crear un registro, y por tanto poder recopilar información de un usuario, se le pedirá que acepte unos términos.

Términos de usuario

En cumplimiento con lo establecido en la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal, le informamos que sus datos personales serán tratados y quedarán incorporados en nuestra base de datos.

Los datos serán tratados de forma confidencial. El servidor donde se almacenan y tratan los datos personales disponen de las medidas de seguridad de índole técnica y organizativas necesarias para evitar la alteración, pérdida y/o acceso no autorizado a datos personales.

Los datos que se le solicitan serán utilizados sin fines lucrativos, el objetivo de la recopilación de ellos es solo la investigación.

Aparte de los datos que usted aporte, el módulo de minería generará más de forma indirecta mediante algoritmos propios.

Usted, puede ejercer sus derechos de acceso, rectificación, cancelación, oposición y revocación a toda la información aportada en cualquier momento.

Si no está de acuerdo con todo lo anterior pulse "Cancelar", si desea continuar y colaborar pulse "Aceptar".

Casos de uso

Dado que este proyecto consiste en la implementación de dos juegos distintos. Los casos de uso que aquí plasmamos estarían duplicados para el juego de David Espino (“Arcade”) y para el juego de Joseba Gallaga (“Shooter”).

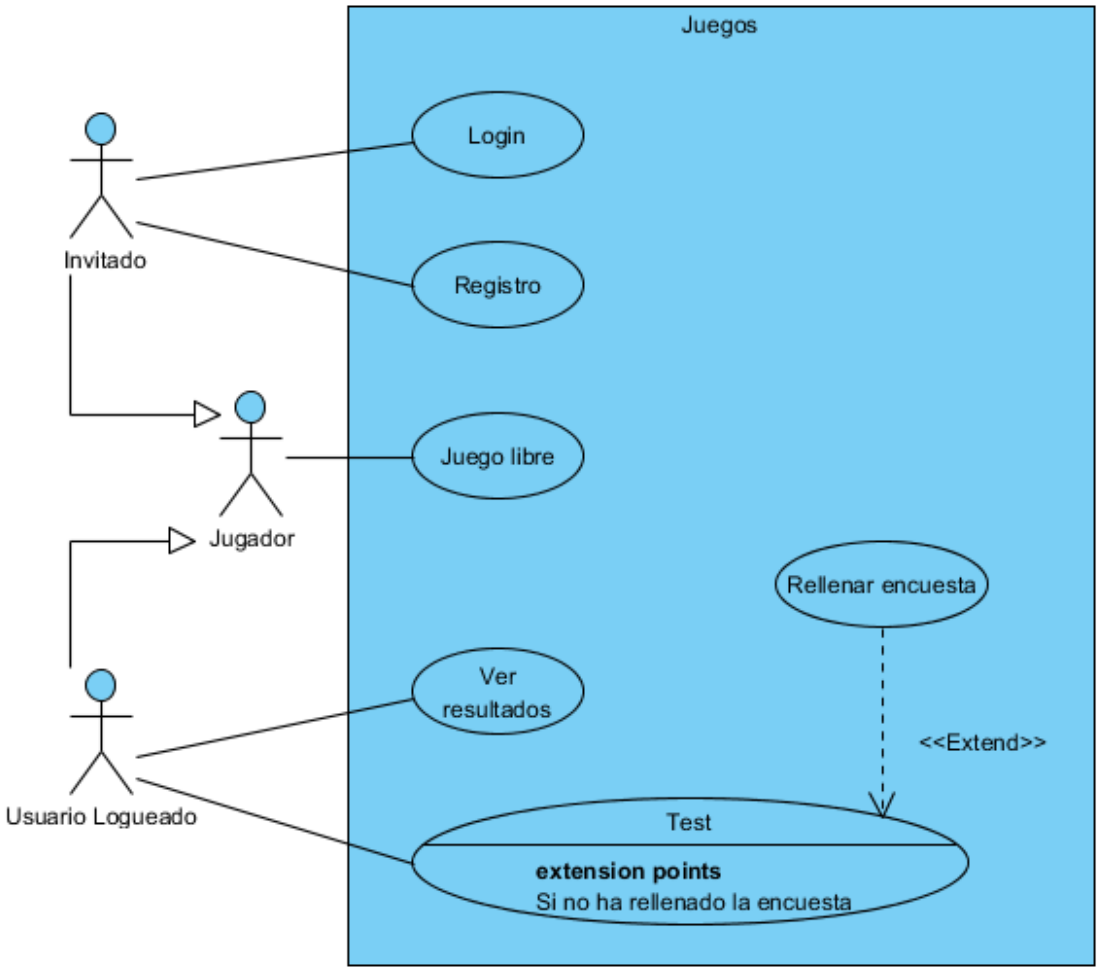
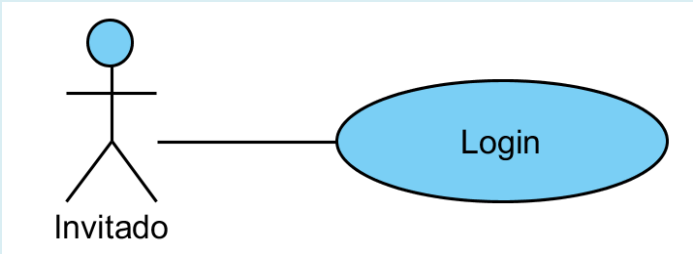


Figura 33: Captura de Requisitos - Casos de Uso

Casos de uso expandidos

Login

Caso de uso:  Figura 34: Casos de Uso - Login	
Nombre:	Login
Descripción:	Permite al usuario invitado loguearse a fin de poder utilizar más funcionalidades del sistema
Actores:	Invitado
Precondiciones	El usuario ha seleccionado juego pero no está logueado
Requisitos no funcionales	Ninguna
Flujo de eventos	1. En el menú invitado el usuario pulsa login. ***Ilustración 1*** 2. Se le muestra la ventana de login. ***Ilustración 2*** [Si rellena y pulsa aceptar] 2.1. Se comprueban los datos en el sistema [Si los datos son correctos] 2.1.1. Es logueado y se le lleva al menú principal logueado. ***Ilustración 3*** [Si los datos son incorrectos] 2.1.2. Se le muestra el error para que los corrija. [Si pulsa cancelar] 2.2. Se le devuelve al menú principal invitado. ***Ilustración 1***
Poscondiciones	Ninguna

Pantallas:

Ilustración 1

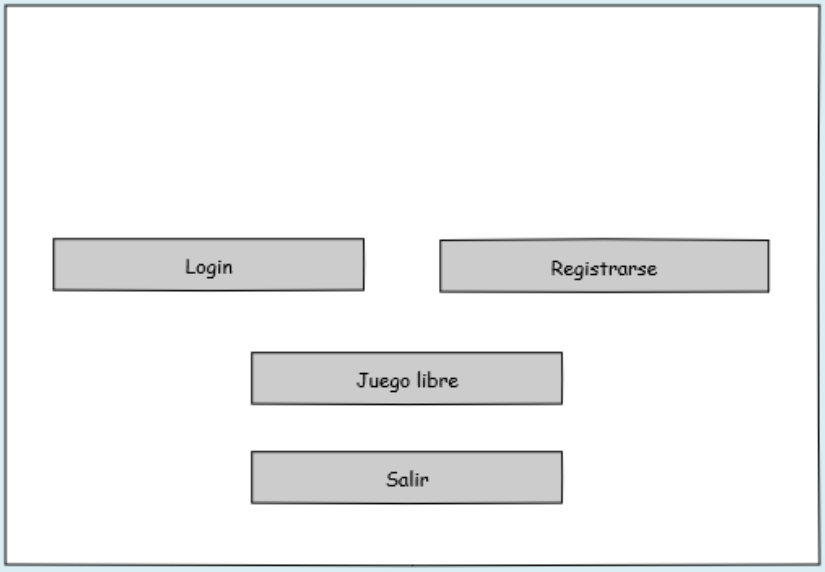


Figura 35: Prototipo - Menú deslogueado

Ilustración 2

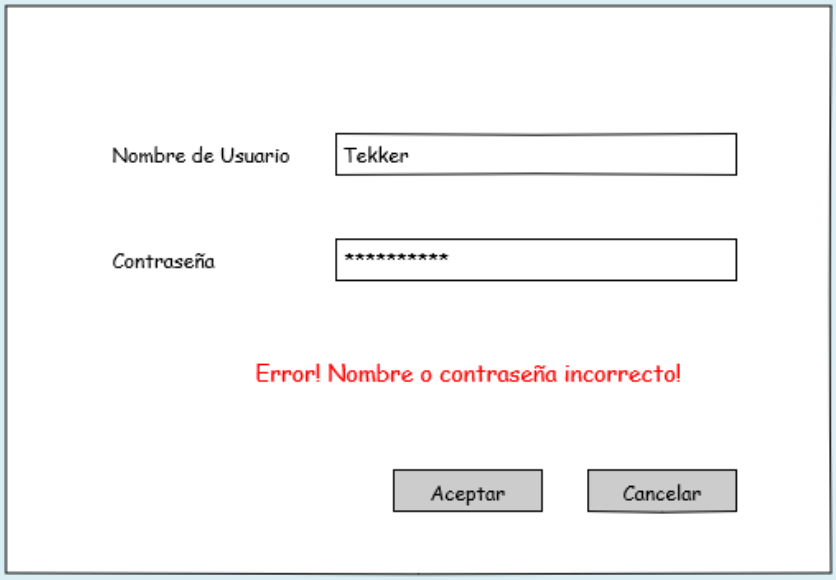


Figura 36: Prototipo - Pantalla Login

Ilustración 3

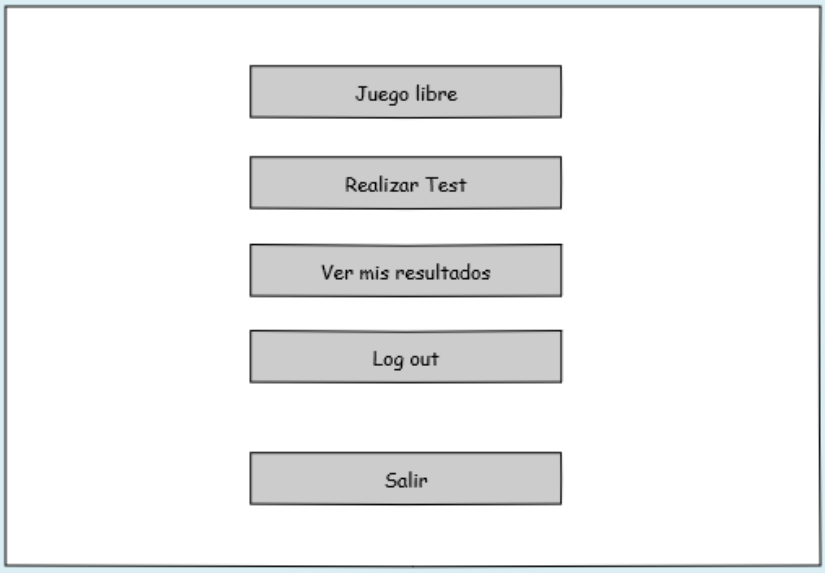
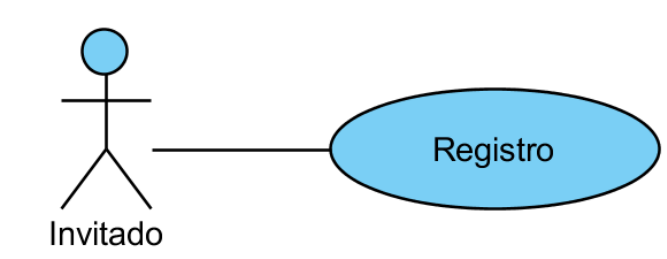


Figura 37: Prototipo - Menú Logueado

Registro

Caso de uso:	 Figura 38: Casos de Uso - Registro
Nombre:	Registro
Descripción:	Permite a un usuario crear un registro para poder logearse en el sistema
Actores:	Invitado
Precondiciones	El usuario ha seleccionado juego pero no está logueado
Requisitos no funcionales	Ninguna
Flujo de eventos	1. Usuario pulsa registro en el menú principal. ***Ilustración 1*** 2. Se le muestra un texto que debe aceptar. ***Ilustración 5*** [Si pulsa aceptar] 2.1. Se le muestra la pantalla de registro. ***Ilustración 4*** [Si pulsa registrarse] 2.1.1. Se comprueban los datos [Si los datos son correctos] 2.1.1.1. Accede al menú principal logueado. ***Ilustración 3*** [Si los datos son incorrectos] 2.1.1.2. Se le muestra un mensaje de error para que corrija los datos. [Si pulsa cancelar] 2.1.2. Vuelve al menú principal invitado. ***Ilustración 1*** [Si pulsa cancelar] 2.2. Vuelve al menú principal invitado. ***Ilustración 1***
Poscondiciones	Se ha creado un registro con los datos del usuario

Pantallas:

Ilustración 4

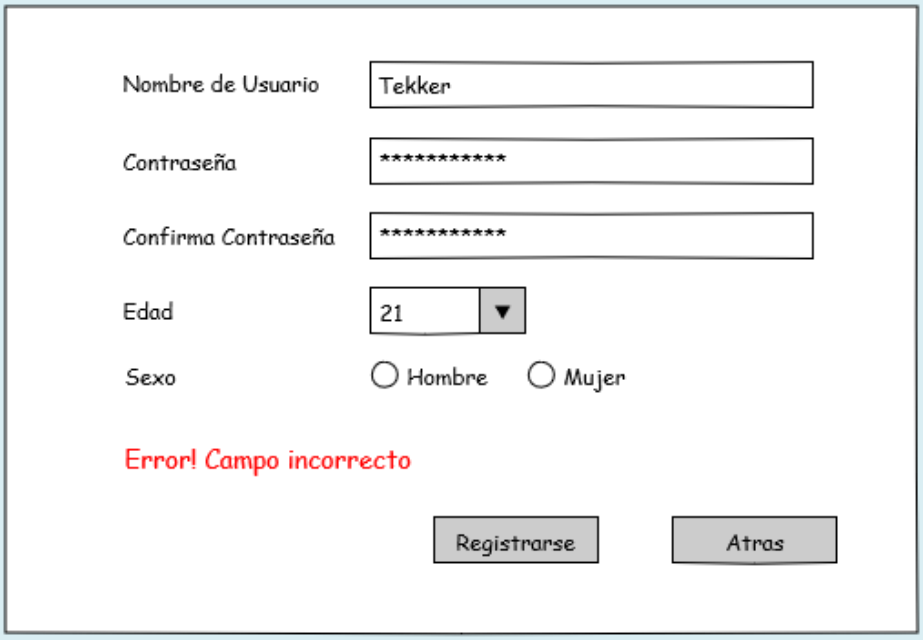


Figura 39: Prototipo - Pantalla Registro

Ilustración 5

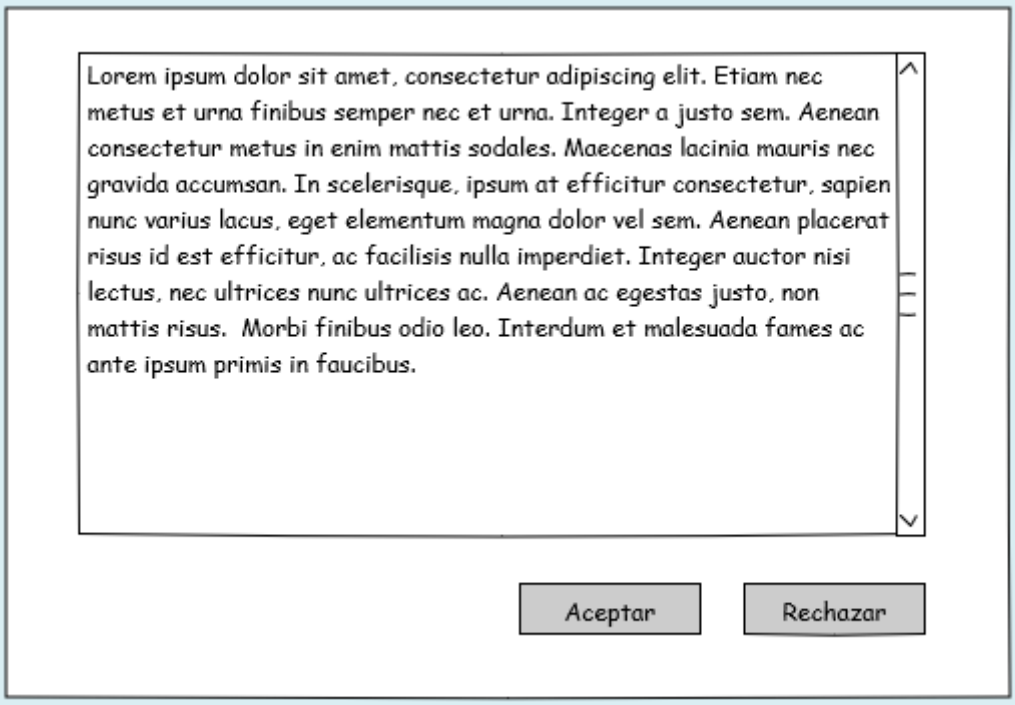


Figura 40: Prototipo - Disclaimer

Juego Libre

Caso de uso:

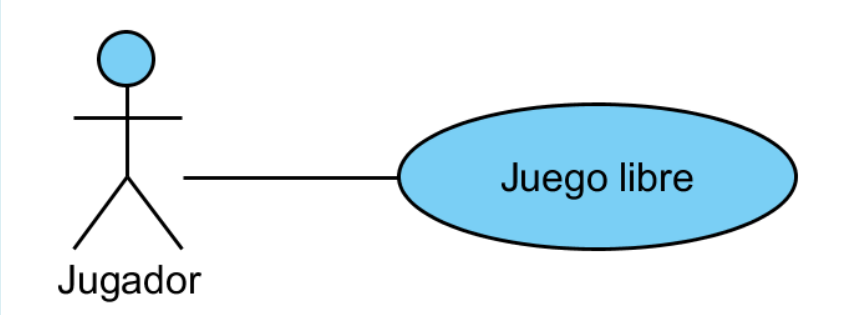


Figura 41: Casos de Uso - Juego Libre

Nombre:	Juego libre
Descripción:	Permite a un usuario jugar una partida en la que no se guarda información
Actores:	Jugador
Precondiciones	El usuario ha seleccionado juego
Requisitos no funcionales	Ninguna
Flujo de eventos	1. El jugador desde el menú invitado o logueado pulsa Juego libre. ***Ilustración 1 o Ilustración 3*** 2. El jugador juega una partida. ***Ilustración 6*** [Si el jugador pulsa *esc*] 2.1. Se abre el menú pausa. ***Ilustración 7*** [El jugador pulsa continuar] 2.1.1. El juego sigue donde se había dejado [El jugador pulsa menú principal] 2.1.2. Se vuelve al menú principal invitado o logueado. ***Ilustración 1 o Ilustración 3*** 3. El jugador muere o termina el juego. 4. Se muestra la pantalla *Game Over* y el usuario pulsa cualquier botón. ***Ilustración 8*** 5. Se vuelve al menú principal invitado o logueado. ***Ilustración 1 o Ilustración 3***
Poscondiciones	Ninguna

Pantallas:

Ilustración 6

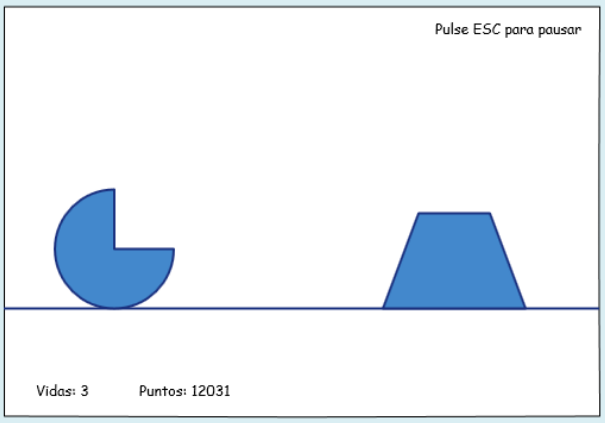


Figura 42: Prototipo - Juego

Ilustración 7

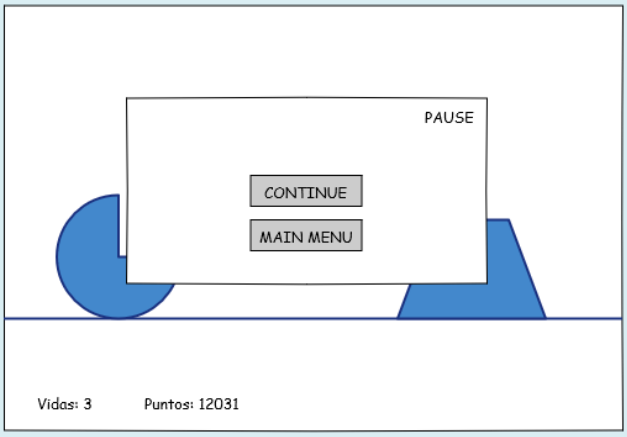


Figura 43: Prototipo - Juego Pausado

Ilustración 8

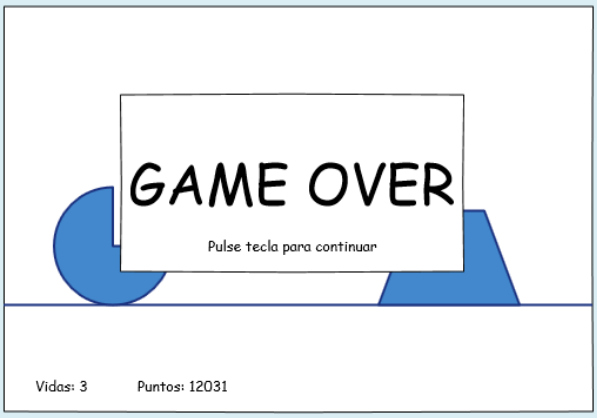
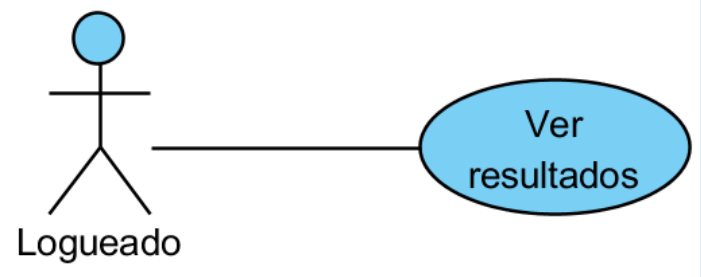


Figura 44: Prototipo - Juego Game Over

Ver resultados

Caso de uso:  Figura 45: Casos de Uso - Ver Resultados	
Nombre:	Ver resultados
Descripción:	Permite a un usuario logueado mirar sus resultados psicológicos y sus datos
Actores:	Usuario logueado
Precondiciones	Ninguna
Requisitos no funcionales	Ninguna
Flujo de eventos	1. El usuario selecciona ver mis resultados en el menú principal. ***Ilustración 3*** 2. Se le muestra al usuario todos los datos recopilados sobre él. ***Ilustración 9*** [Usuario pulsa aceptar] 2.1. Vuelve al menú principal logueado. ***Ilustración 3*** [Usuario pulsa eliminar datos] 2.2. Se le muestra una pantalla con información. ***Ilustración 10*** [Usuario pulsa confirmar] 2.2.1. Se borran sus datos del sistema 2.2.2. Se le devuelve al menú principal no logueado. ***Ilustración 1*** [Usuario pulsa cancelar] 2.2.3. Se le devuelve al menú principal logueado. ***Ilustración 3***
Poscondiciones	Ninguna, si ha pulsado eliminar datos se borrará todo registro sobre él del sistema

Pantallas:

Ilustración 9

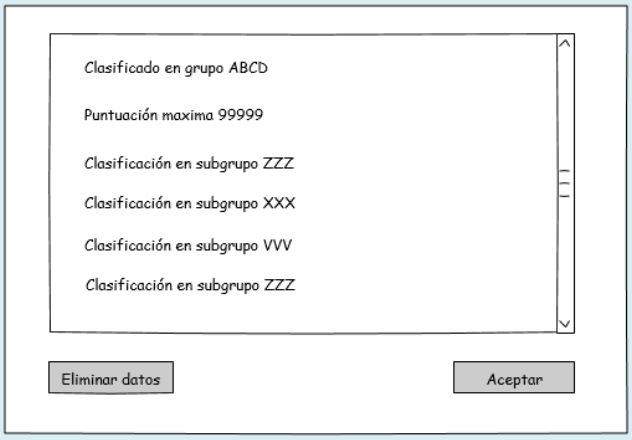


Figura 46: Prototipo - Resultados

Ilustración 10

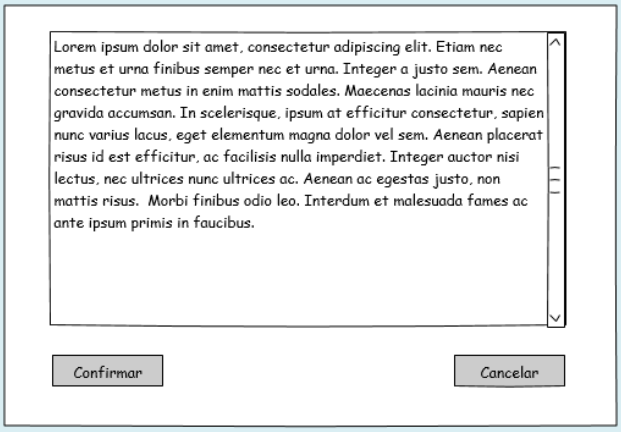
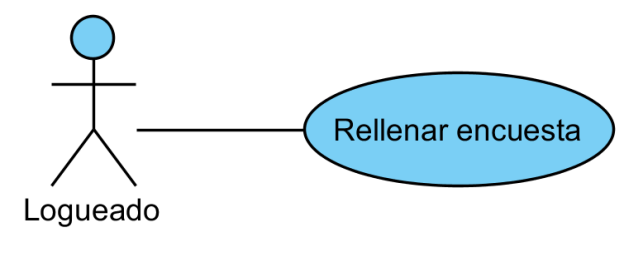


Figura 47: Prototipo - Explicación eliminar

Rellenar encuesta

Caso de uso:	 Figura 48: Casos de Uso - Rellenar Encuesta
Nombre:	Rellenar encuesta
Descripción:	Encuesta pregunta por pregunta para recoger datos del jugador
Actores:	Usuario logueado
Precondiciones	No ha finalizado la encuesta en ningún juego del modulo
Requisitos no funcionales	Ninguna
Flujo de eventos	1. El usuario pulsa realizar encuesta en el menú test. ***Ilustración 11*** 2. Se le muestran las instrucciones del test. ***Ilustración 12*** [Si pulsa responder preguntas] 2.1. Se le muestra la pregunta que tiene que contestar. ***Ilustración 13*** [Si pulsa siguiente pregunta] 2.1.1. El sistema comprueba si ha elegido opción. [Si no es correcta] 2.1.1.1. Se le muestra un mensaje de error para que la corrija. [Si es correcta] 2.1.1.2. Se comprueba si quedan más preguntas. [Si quedan más preguntas] 2.1.1.2.1. Se le muestra la siguiente pregunta [Si es la última pregunta] 2.1.1.2.2. Se le indica que ha terminado 2.1.1.2.3. Se bloquea la opción de realizar la encuesta 2.1.1.2.4. Se desbloquea la opción de realizar tutorial 2.1.1.2.5. Se le lleva a menú test. ***Ilustración 14*** [Si pulsa atrás] 2.2. Se le devuelve al menú test. ***Ilustración 11***
Poscondiciones	Los datos de las preguntas contestadas son guardados en el sistema

Pantallas:

Ilustración 11

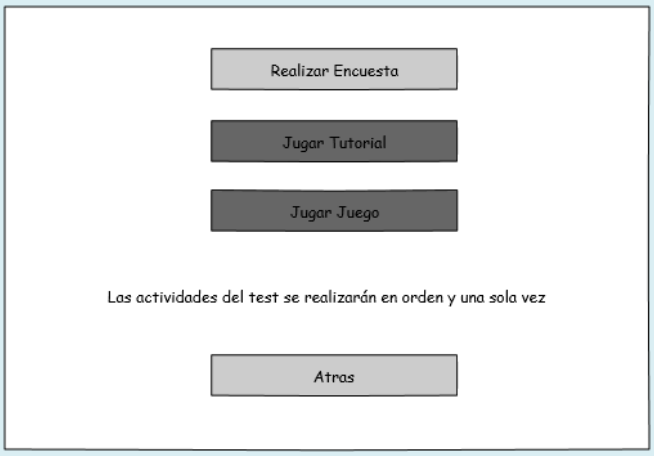


Figura 49: Prototipo - Menu Test 1

Ilustración 12

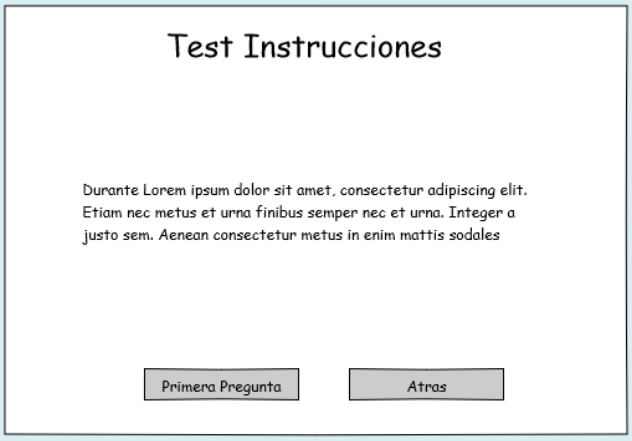


Figura 50: Prototipo - Test Instrucciones

Ilustración 13

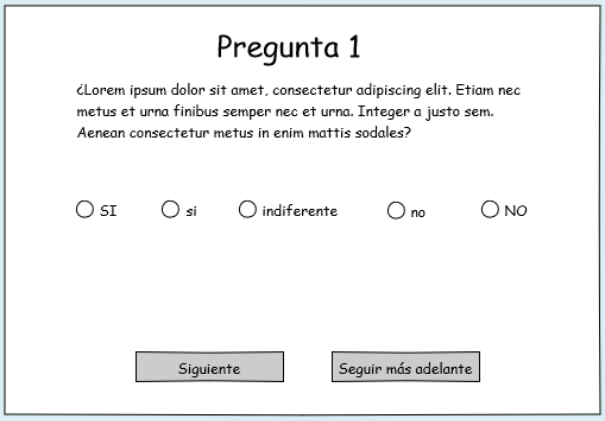


Figura 51: Prototipo - Test Preguntas

Ilustración 14

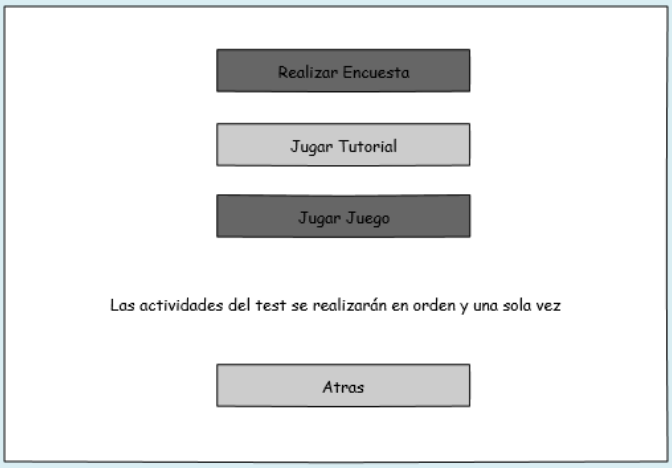


Figura 52: Prototipo Menú Test 2

Test

Caso de uso:

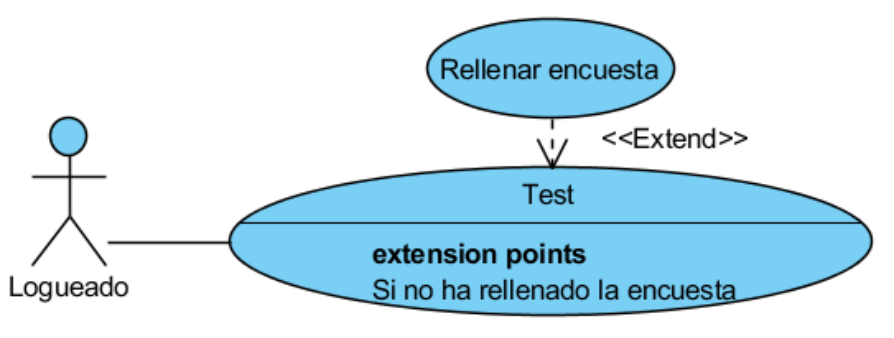


Figura 53: Casos de Uso - Test

Nombre:	Test
Descripción:	Permite al usuario realizar todos los pasos necesarios para realizar un estudio sobre su personalidad
Actores:	Usuario logueado
Precondiciones	Esta logueado y ha hecho la encuesta
Requisitos no funcionales	Ninguna
Flujo de eventos	1. El usuario selecciona Realizar Test en el menú principal. ***Ilustración 3*** 2. Comprobamos si ha realizado la encuesta. [Si no ha hecho la encuesta] 2.1. Menú con encuesta desbloqueada. ***Ilustración 11*** [Si pulsa realizar encuesta] 2.1.1 Caso de uso "Rellenar encuesta" [Si pulsa volver] 2.1.2 Vuelve al menú principal logueado. ***Ilustración 3*** 3. Comprobamos si ha realizado el tutorial. [Si no ha hecho el tutorial] 3.1. Mostramos el menú con la opción de tutorial. ***Ilustración 14*** [Si pulsa realizar tutorial] 3.1.1. El jugador juega una partida hasta superarla. ***Ilustración 6*** [Si pulsa volver] 3.1.2. Vuelve al menú principal logueado. ***Ilustración 3*** 4. Comprobamos si ha realizado el test de juego. [Si no ha jugado el test] 4.1. Mostramos el menú con la opción jugar. ***Ilustración 15*** [Si pulsa jugar] 2.2.1. El jugador juega una partida hasta superarla. ***Ilustración 6***

	2.2.2. El sistema recopila la información [Si pulsa volver] 2.2.3. Vuelve al menú principal logueado. ***Ilustración 3*** 3. Mostramos el menú bloqueado. ***Ilustración 16*** 4. Jugado pulsa volver, se le devuelve al menú principal logueado. ***Ilustración 3***
Poscondiciones	Ninguna

Pantallas:

Ilustración 15

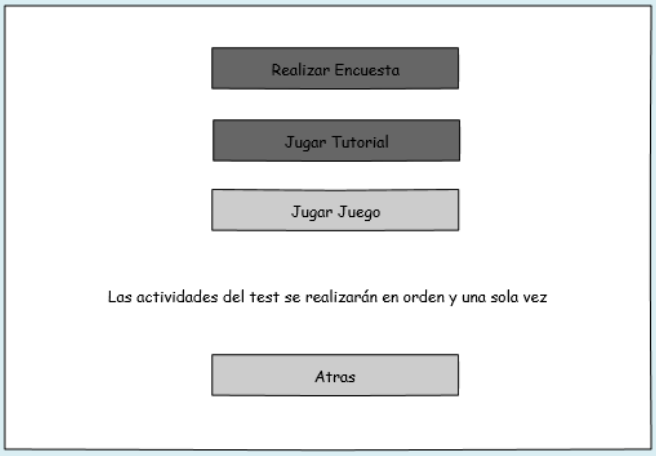


Figura 54: Prototipo - Menú Test 3

Ilustración 16

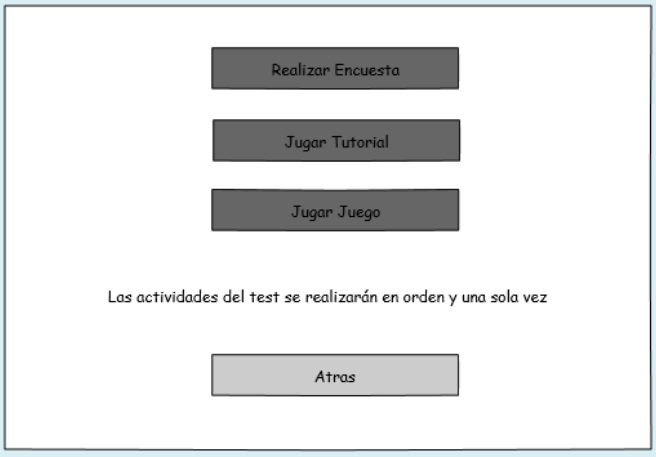


Figura 55: Prototipo - Menú Test 4

Prototipo 1

Juego Shooter

Descripción del juego

El jugador deberá manejar una nave que será capaz de cambiar de color para poder disparar de diferentes formas. Dependiendo del tipo de color, el proyectil lanzado se comportará de manera distinta. Los enemigos al ser eliminados tendrán un probabilidad de soltar una mejora de proyectil que podrá ser de uno de los cuatro colores disponibles de la nave. Si el jugador toca esa mejora haciendo coincidir los colores tanto de la nave como de la mejora, el disparo de dicho color será mejorado incrementando su tamaño y daño. Si al tocar la mejora los colores no coinciden, el nivel del proyectil que esté usando en ese momento perderá un punto.

La nave podrá moverse libremente por todo el escenario por lo que será cuestión del usuario en qué lugar ponerse y utilizar el tipo de disparo que mejor le convenga en ese momento. Los enemigos también podrán lanzar sus propios proyectiles. Si la nave choca con cualquiera de éstos o con el propio enemigo, todas sus mejoras de disparo se verán reducidas en un punto.

Transcurridos un número de enemigos aparecerá un jefe final como último reto. Una vez vencido a este gran enemigo la partida se dará por concluida.

Eliminar enemigos y coger correctamente las mejoras de disparo otorgan puntos.

Objetos en el juego

Jugador

Para moverse por el escenario deberá usar las teclas WASD. Podrá moverse libremente pero sin poder salirse de los bordes de la interfaz.

La nave dispone de cuatro colores distintos que podrá cambiar cuando el jugador así lo desee mediante las teclas de dirección del teclado.

Figura 56: Gráficos - Naves

Para disparar se pulsa la tecla de espacio. Dependiendo del color de la nave, el proyectil tomará una dirección diferente. Dependiendo del nivel, el aspecto también cambiará además de la cantidad de daño que produce al enemigo.

Figura 57: Gráficos - Disparos

Mejoras

Al derrotar un enemigo, éste puede dejar una mejora de uno de los cuatro colores de la nave. Se desplazarán hacia la parte izquierda de la pantalla, llegando a desaparecer el juego si el jugador no los recoge.

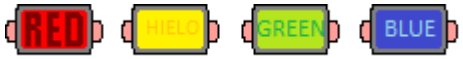


Figura 58: Gráficos - Power Ups

Enemigos

Existen tres tipos de enemigos, además de un jefe final. Cada uno de estos enemigos tienen un comportamiento concreto con un tipo de disparo también determinado. Pueden tener la propiedad de ser inmunes o de que no disparen.

Figura 59: Gráficos - Enemigos

Interfaz

En ella aparecerá el nivel de cada bala y los puntos que ha conseguido hasta el momento. En los bordes de la interfaz aparecen los colores que corresponden a la posición de las teclas de cambiar color para que le resulte más fácil saber cuál es cada color.

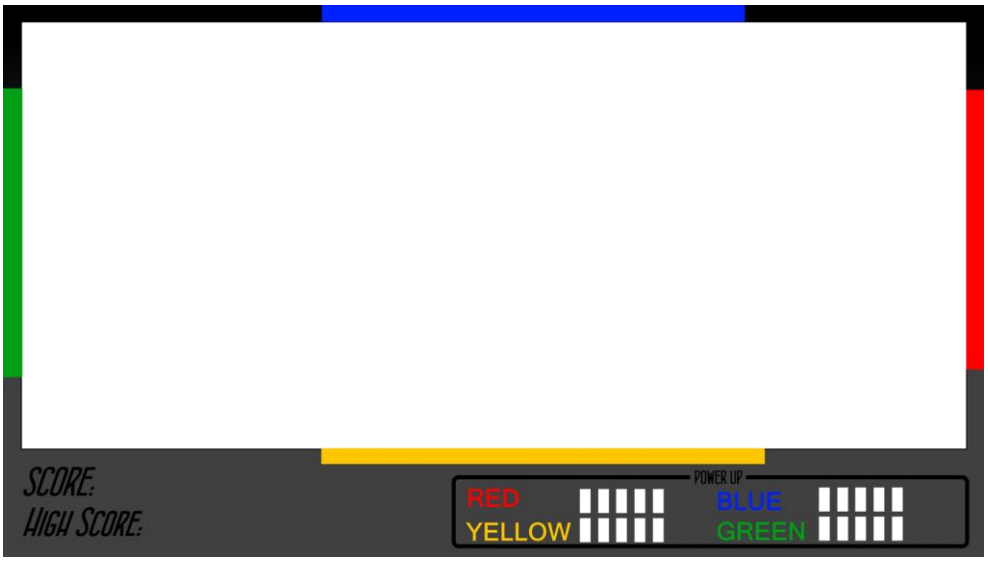


Figura 60: Gráficos – GUI

Diagrama de clases

Diagrama con las conexiones y dependencias relevantes entre cada clase estudiadas para el desarrollo del proyecto.

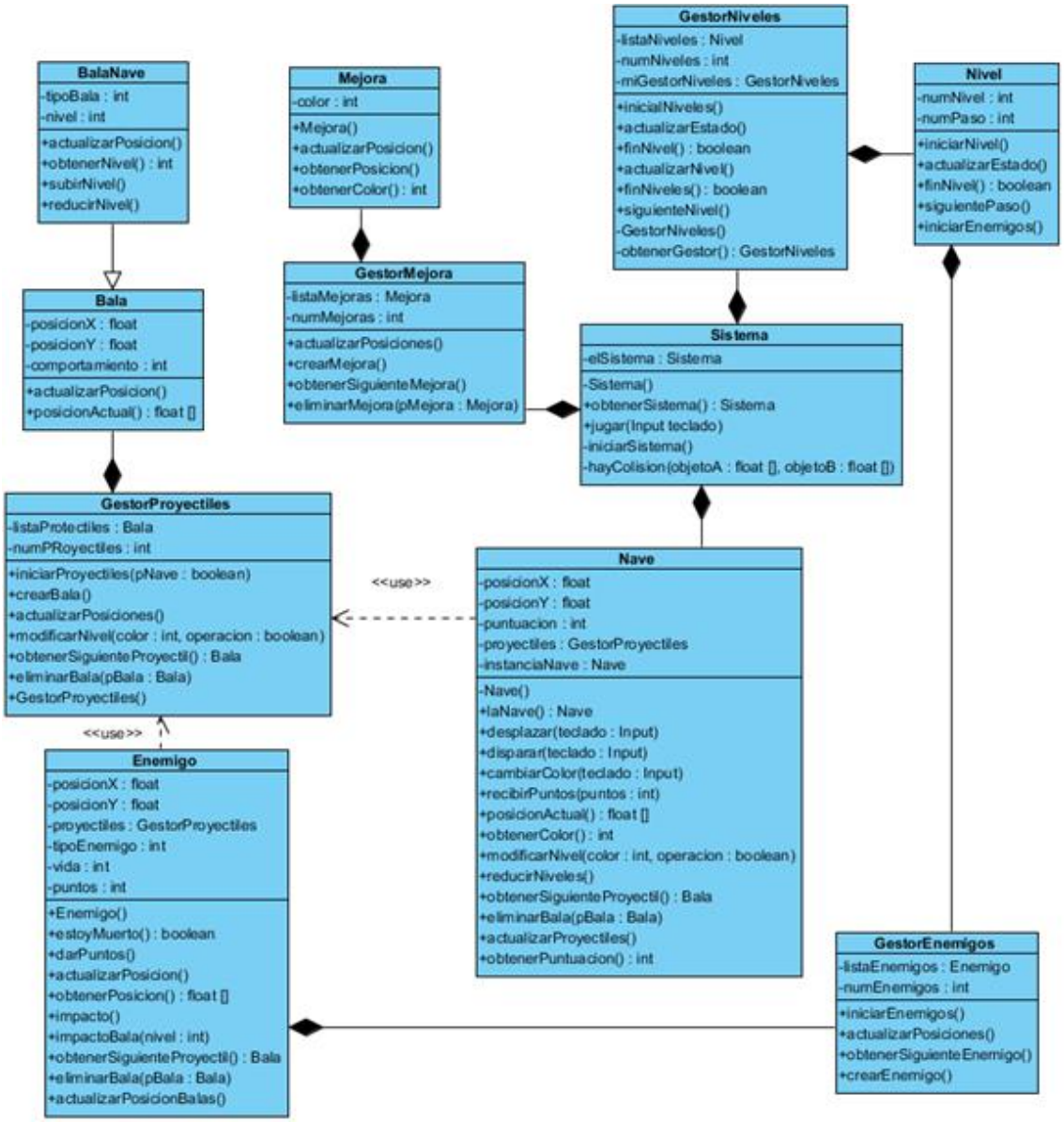


Figura 61: Shooter Prototipo 1 - Diagrama de Clases

Sistema

Es la clase encargada de controlar la situación del juego. Ya reciba o no entradas por parte del usuario, el sistema debe actualizar la situación de todos sus elementos para que el juego continúe.

Su función es la de pedir a los elementos que actualicen su posición y comprobar que si hay alguna colisión entre ellos actúen en concordancia.

Nivel

Tiene el orden de ejecución de los eventos que ocurren en la pantalla, como la creación de enemigos o saber cuándo ha terminado un nivel, así como la posición inicial de la nave.

Nave

Es el objeto que controla el usuario, puede moverse, cambiar de color y disparar. Si consigue alguna mejora podrá subir o bajar de nivel.

Al eliminar enemigos consigue una puntuación, pero si choca contra enemigos o proyectiles pierde niveles.

Bala

Se encarga de desplazase por la pantalla según el patrón que se a elegido para el objeto. Le tienen que decir en qué situación se encuentra para actuar.

Bala Nave

Similar a la Bala normal pero tiene la diferencia de tener niveles además de nuevas formas de comportamiento.

Enemigo

Su objetivo es la de desplazarse para entorpecer al jugador, cada poco disparará un proyectil. Si es destruido dará puntos al jugador y dejará una mejora de nivel.

Mejora

Tienen cuatro posibles colores y se desplazan en un patrón sencillo para ser obtenidos por el jugador.

Gestor Proyectiles

Se encarga de controlar los proyectiles lanzados por el objeto que lo contiene. Actualiza su posición y se encarga de eliminarlo si golpea su objetivo o sale de la pantalla.

Gestor Enemigos

Controla los enemigos que aparecen en el nivel, actualiza su posición y comprueba si es necesario eliminarlos por golpear al jugador o quedarse sin vida.

Gestor Niveles

Controla el progreso del nivel y los eventos que ocurren en él, como la creación de enemigos. Si acaba el nivel cargará el siguiente o avisara de que ya no quedan más.

GestorMejora

Controla la posición de las mejoras y si el jugador ha conseguido cogerlos.

Diagrama de secuencia

El diagrama se ha creado teniendo en cuenta que el juego sigue una ejecución independiente del usuario. Es decir, que aunque el jugador no toque ningún botón la aplicación deberá seguir realizando la secuencia abajo descrita para comprobar que funciona como se espera.

Existe un primer bloque que comprueba la posición de todos los elementos de la partida. Actualiza cual sería la siguiente posición de cada elemento. Si hay elemento fuera de la ventana de juego, se eliminarán. Si se crean nuevos objetos, se les asigna su posición correspondiente.

El segundo bloque se encarga de comprobar cómo interactúan los objetos entre ellos, enemigos que chocan con el usuario, mejoras obtenidas por la nave, si los proyectiles han alcanzado algún objetivo...

IniciarSistema

Se inicializan y cargan las variables que se utilizarán durante la partida.

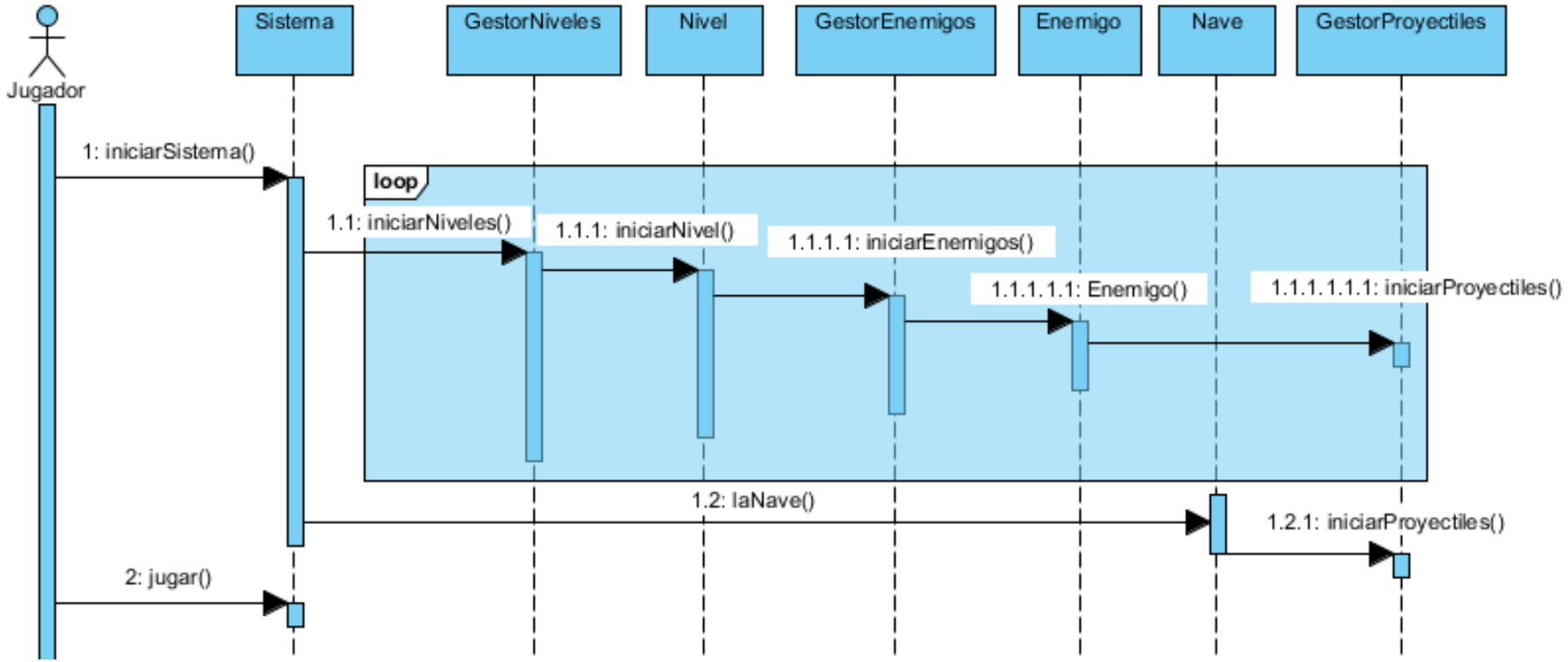


Figura 62: Shooter Prototipo 1 - Diagrama de Secuencia – Iniciar Sistema

Los siguientes diagramas de secuencia ocurren dentro de un bucle jugar hasta que la partida termina. En cada ciclo se hacen las comprobaciones para conocer el nuevo estado de todos sus elementos, cuales hay que crear, modificar o eliminar.

ActualizarPosiciones

Acciones que el usuario quiere realizar para actualizar el estado de la nave. El sistema por su parte toma el resto de elementos y gestiona sus siguientes acciones. Si un enemigo en este momento se encuentra en el estado de muerto por haber perdido todos sus puntos de vida, generará una mejora en su posición y dará puntos a la nave. Si no cambiará de posición y creará una nueva bala.

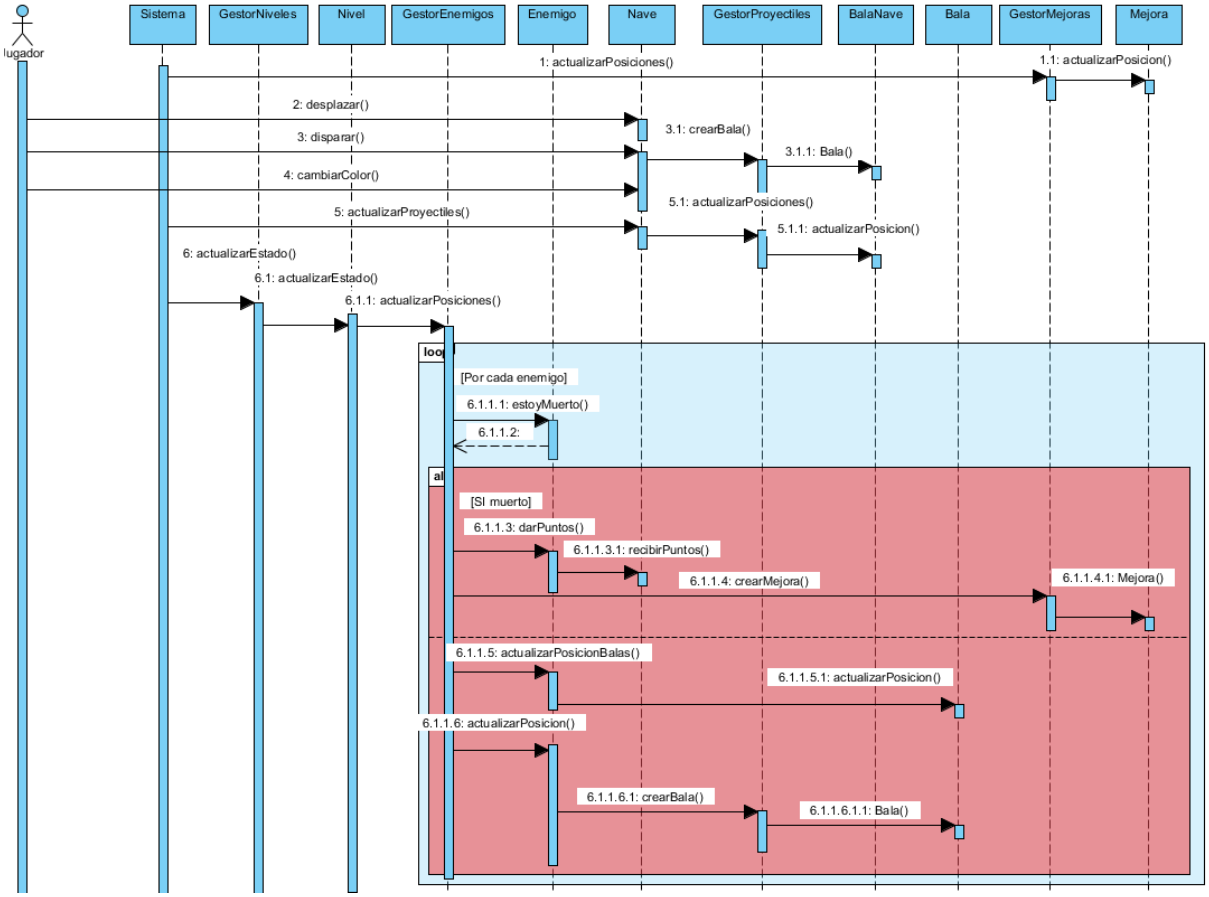


Figura 63: Shooter Prototipo 1 - Diagrama de Secuencia – Actualizar Posiciones

RecogerMejora

Cuando la nave toca una mejora hay que comprobar si los colores de ambos coinciden para incrementar o no correctamente el tipo de disparo del jugador.

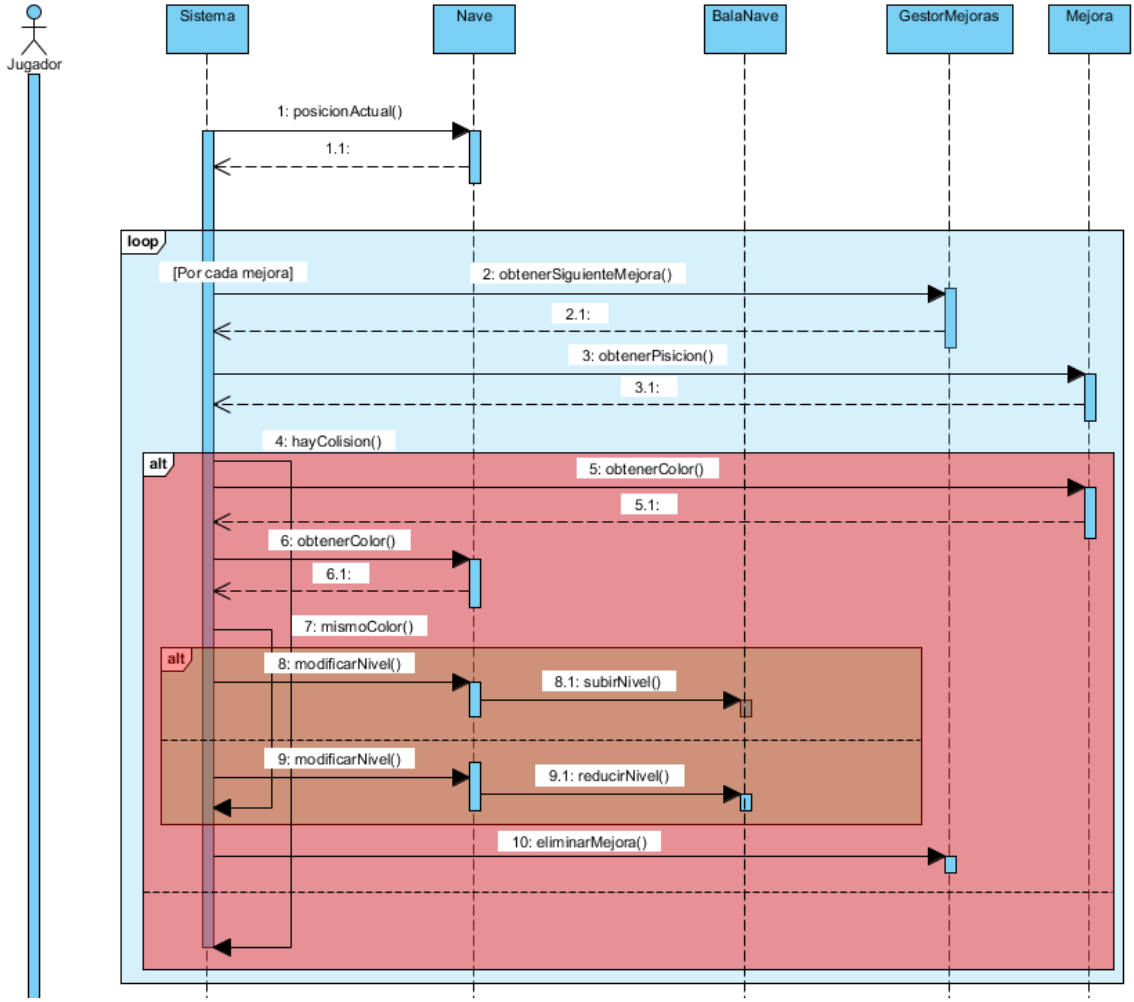


Figura 64: Shooter Prototipo 1- Diagrama de Secuencia - RecogerMejora

ComprobarChoque1

Hay que comprobar que las naves enemigas no estén chocando con el jugador, de ser así habrá que penalizar a la nave e indicar al enemigo que ha sido golpeado para reducir sus puntos de vida a 0 y así poder eliminarlo en la fase de comprobación de la posición de los elementos del juego.

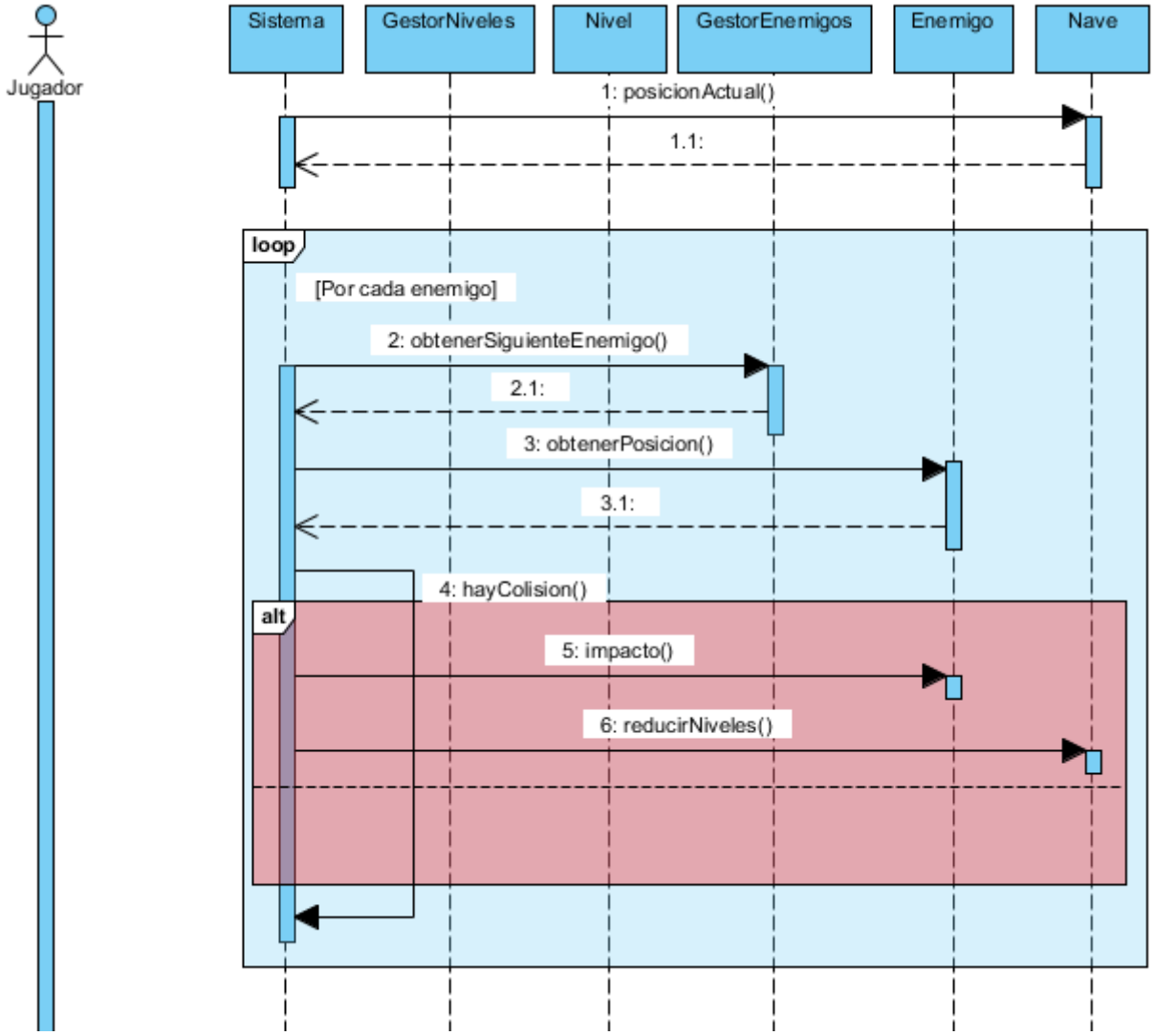


Figura 65: Shooter Prototipo 1 - Diagrama de Secuencia - Comprobar Choque 1

ComprobarChoque2

Para comprobar si el jugador ha conseguido golpear a algún enemigo, hay que realizar una comprobación de la posición de todos los enemigos con todos los proyectiles. Si hay colisión la bala desaparece y el enemigo recibe el aviso de que ha sido golpeado.

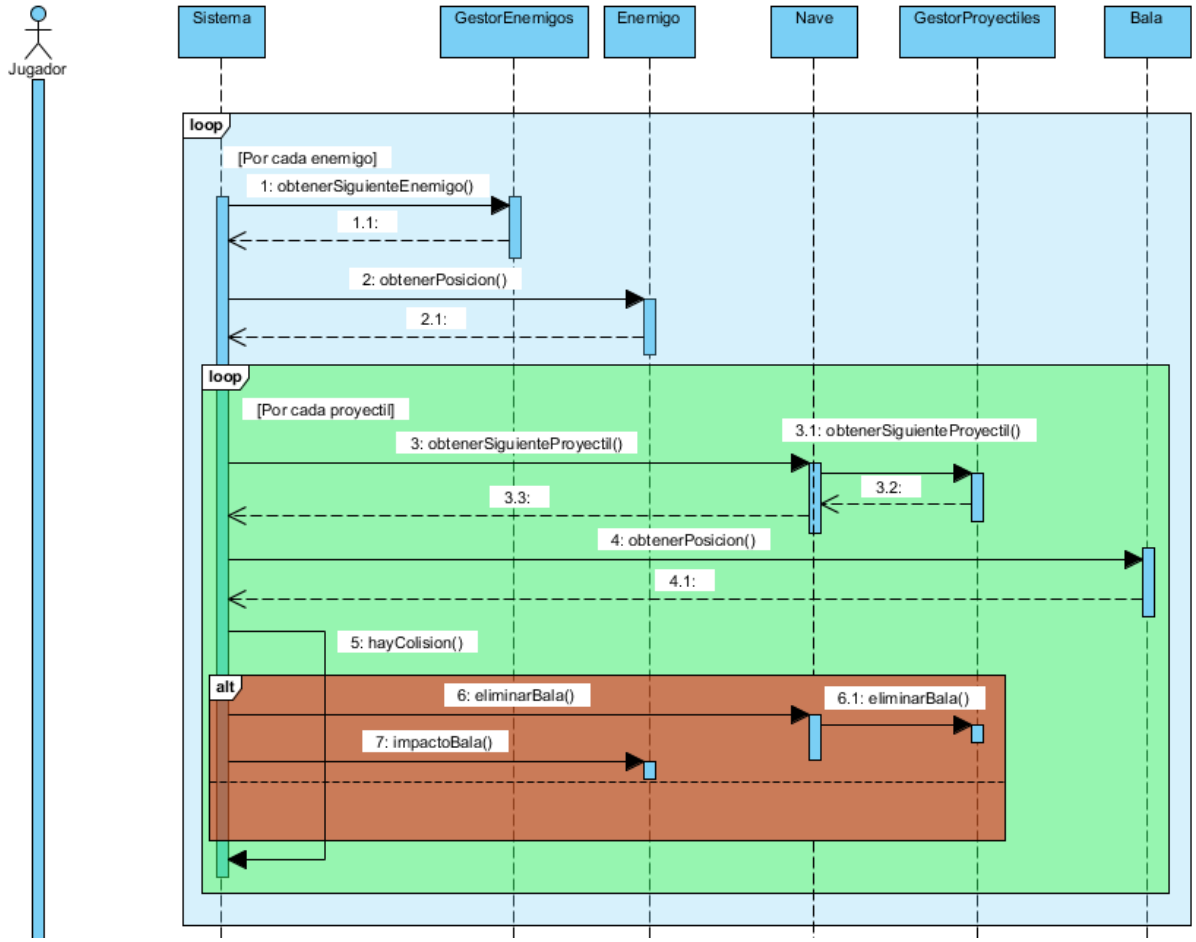


Figura 66: Shooter Prototipo 1 - Diagrama de Secuencia - Comprobar Choque 2

ComprobarChoque3

De la misma el jugador puede ser golpeado por proyectiles disparados por los enemigos. Se mira por cada enemigo, que proyectiles ha disparado y si alguno ha golpeado a la nave.

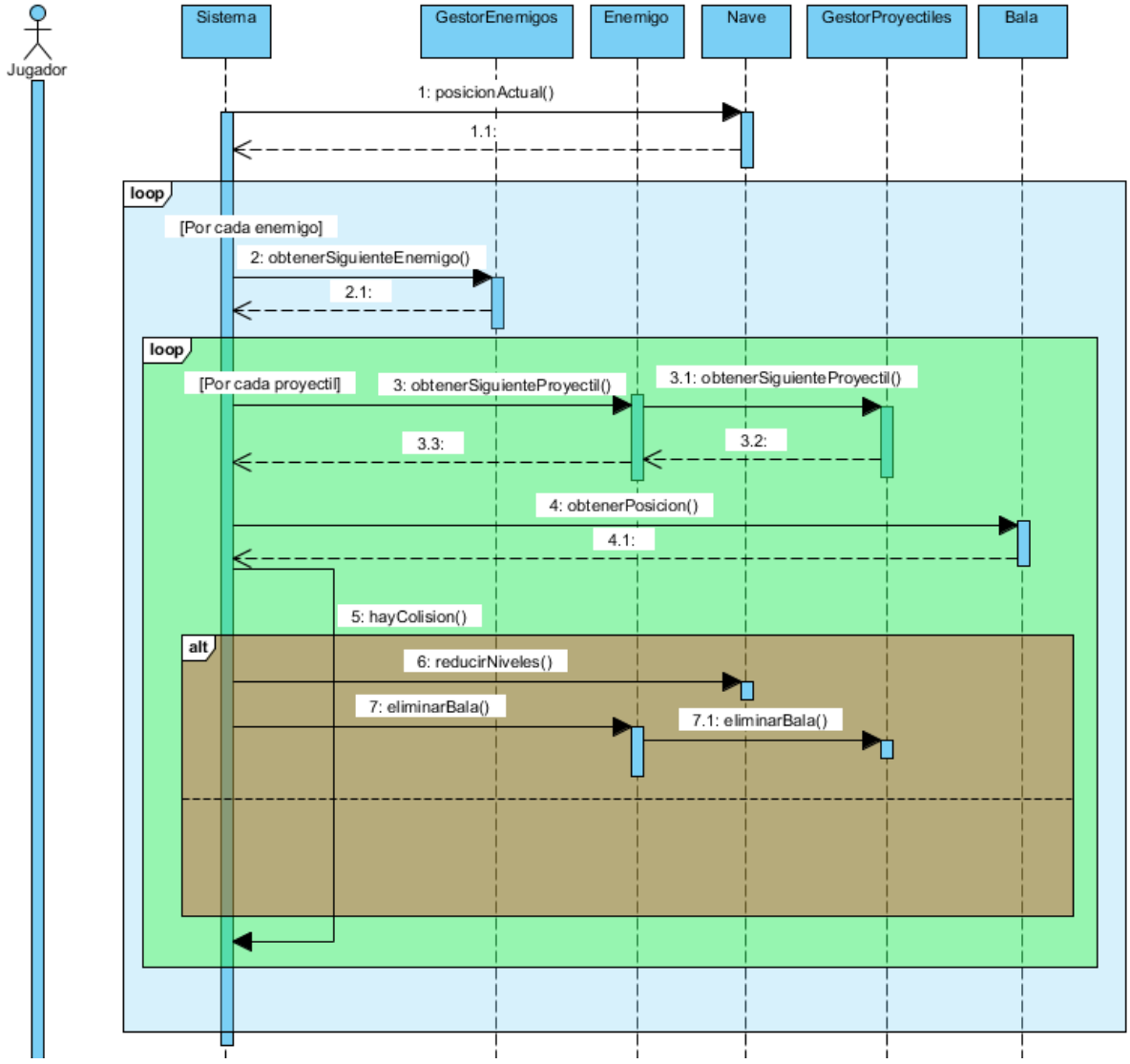


Figura 67: Shooter Prototipo 1 - Diagrama de Secuencia - Comprobar Choque 3

ComprobarNivel

Antes de comenzar de nuevo el proceso de comprobar el estado del juego se mira si el nivel ya ha terminado. Si es así se procede a preparar el siguiente nivel. Si ya no quedan más niveles el bucle del juego se romperá.

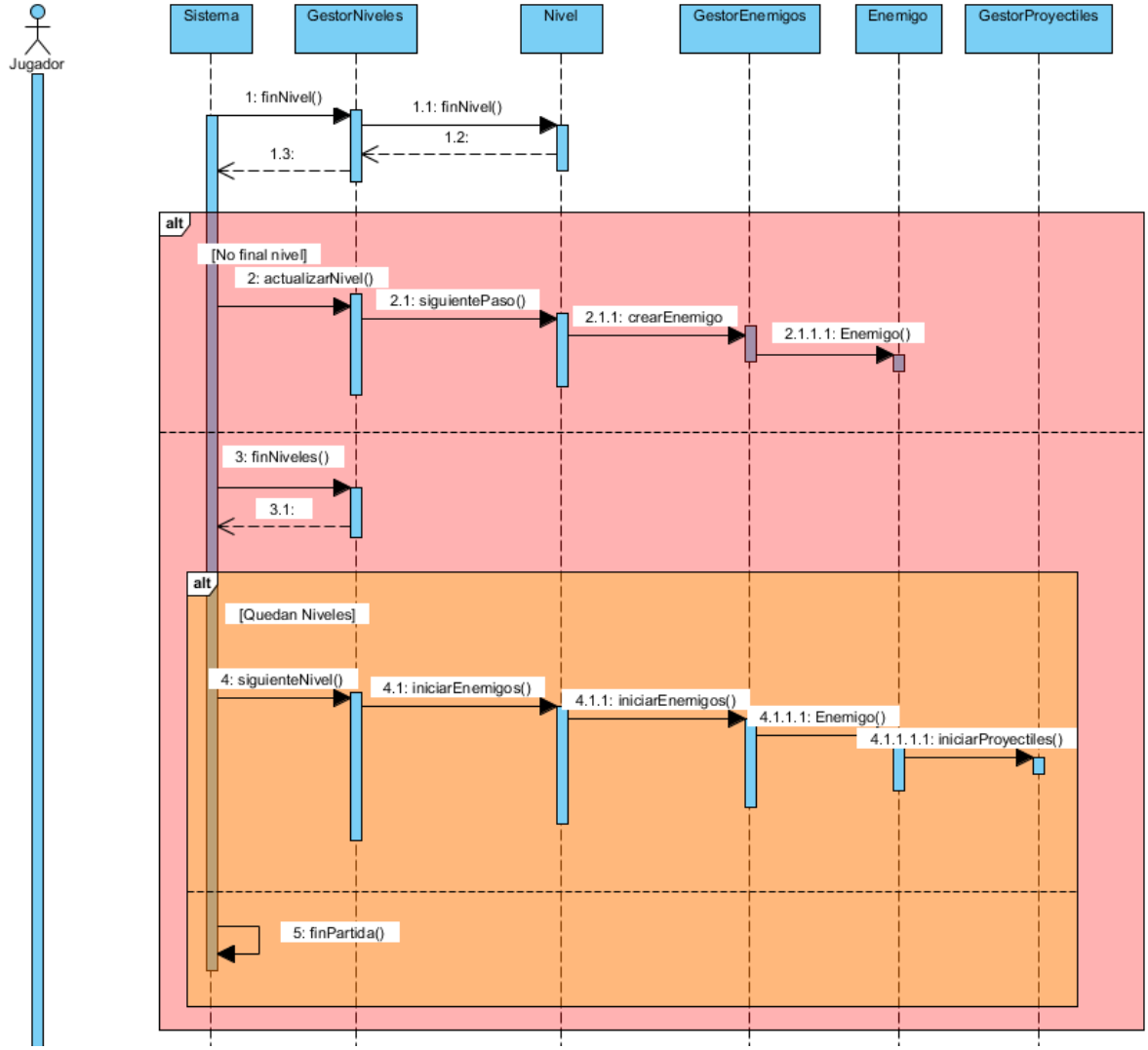


Figura 68: Shooter Prototipo 1 - Diagrama de Secuencia - Comprobar Nivel

Mostrar Resultados

Se muestran los resultados por pantalla

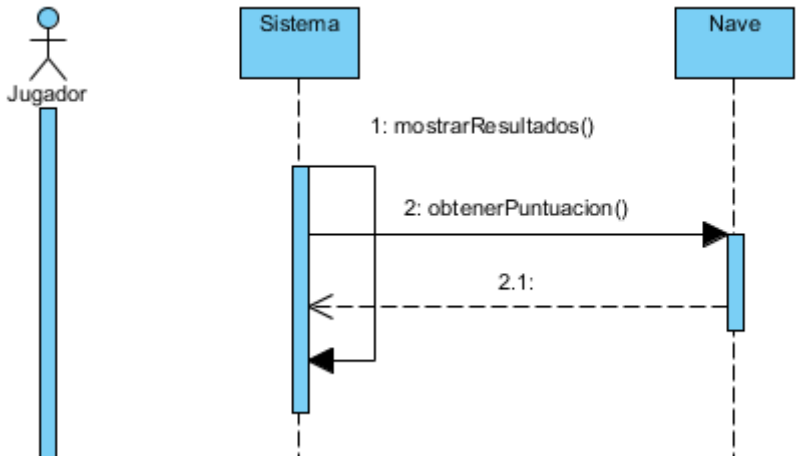


Figura 69: Shooter Prototipo 1 - Diagrama Secuencia - Mostrar Resultados

Juego Arcade

Descripción del juego

El juego será un scroll lateral 2d infinito sobre un vehículo. Cuando comience la partida se situará a un personaje montado en una carreta en medio de la pantalla, éste será controlado por el jugador. El personaje deberá transitar por un recorrido y mantenerse con vida en el mismo el mayor tiempo posible, teniendo en cuenta que habrá enemigos y obstáculos que tratarán de impedírselo. Para ello podrá acelerar, desacelerar, subir la altura de su carreta, agacharse y disparar. La carreta ira sobre vías en dirección hacia delante encontrándose con diferentes objetos, habrá dos tipos: objetos de ayuda y enemigos. Entre los objetos de ayuda existirán monedas, es decir puntos, y sombreros, que serán equivalentes a vidas adicionales. Como enemigos tendrá pinchos, rocas rodantes, murciélagos y humanos que montarán otra carreta. Para interactuar con objetos de ayuda simplemente hay que tocarlos. Los enemigos podrán dañar al contacto y podrán ser destruidos con las balas de la pistola dependiendo el tipo. Por ejemplo los pinchos del suelo no podrán ser destruidos mientras que los murciélagos sí.

Cuando comienza el juego tendremos tres vidas o puntos de salud. Si perdemos los tres puntos el juego terminará volviendo al menú principal.

Durante el juego existe una pequeña interfaz que mostrará datos sobre lo que está ocurriendo en la partida. Los datos serán munición restante, vidas, puntos obtenidos y monedas recogidas.

El objetivo del juego es aguantar el máximo tiempo posible sin morir acumulando puntos y monedas, para ello es necesario un mapa infinito que vaya añadiendo secciones según se avance.

Objetos en el juego

Jugador

Irá montado en una carreta. La velocidad de la carreta dependerá del tramo de escenario donde se encuentra. Es decir la velocidad será controlada por el escenario. Comenzará con un número fijo de vidas, cada vez que sea golpeado las irá perdiendo, pudiendo recuperarlas con objetos durante el camino. También podrá recolectar monedas. Si llega a 0 vidas el juego terminará.

Figura 70: Gráficos - Protagonista

Acciones:

Elevarse: La carreta se levanta dejando solo el sistema de ruedas en el suelo, esto permite esquivar obstáculos como pinchos y rocas rodantes pequeñas.

Agacharse: El personaje agacha la cabeza dentro de la carreta, mientras esté agachado no podrá disparar, esto le permite esquivar balas o murciélagos

Disparar: El personaje dispara su revolver enviando una bala hacia delante, el revolver tendrá seis balas y será necesario recargarlo cuando se agota la munición. Si un enemigo como murciélagos o enemigos humanos son golpeados serán destruidos. Las balas también servirán para cambiar secciones de vías y tomar desvíos.

Acelerar: La carreta añadirá algo de velocidad a su velocidad base, además la cámara se desplazara hacia el lado contrario para condicionar al jugador.

Decelerar: La carreta restará velocidad a su velocidad base, además la cámara se desplazará al lado contrario.

Interfaz

Un cuadro que mostrará datos de lo que está ocurriendo en la partida, se incluirán el número de vidas restantes, la munición para saber si necesitamos recargar, la distancia recorrida y las monedas recolectadas. Dependiendo el diseño gráfico final podrían añadirse datos como enemigos derrotados, secciones de mapa superadas o puntuación total.

Figura 71: Gráficos - GUI

Enemigos

Pinchos: Obstáculos inmóviles que golpearán al jugador si pasan por encima, la forma de esquivarlos será agacharse o elevarse, dependiendo si el pincho está en el suelo o en el techo. No serán destruidos de ningún modo.

Figura 72: Gráficos - Spyke

Roca rodante: Pueden ser pequeñas o grandes. Si son pequeñas podrán ser esquivadas elevando la carreta y en caso de tocarla harán un daño siendo automáticamente destruidas. La velocidad y la dirección variarán dependiendo de su instanciación. Si son grandes rodaran por detrás del jugador y en caso de tocarlo lo matarán automáticamente sin depender de las vidas restantes.

Figura 73: Gráficos - Rock

Murciélagos: Pequeños murciélagos que harán un vuelo predefinido hacia el jugador, si golpean al jugador sin estar agachado le harán un daño, también pueden ser destruidos si son disparados.

Figura 74: Gráficos - Bat

Enemigo humano: Enemigos que montan en carreta como nosotros, en principio siempre irán algo separados a nosotros, para eliminarlos habrá que dispararlos varias veces, también nos dispararán y el contacto con ellos nos causará daño.

Figura 75: Gráficos - Human Enemy

Objetos recolectables

Monedas: Al final de la partida se mostrará cuantas se han recogido y darán un bono a la puntuación.

Figura 76: Gráficos - Coin

Sombreros: Son vidas adicionales, se sumarán a las que tenía el jugador sin poder superar el máximo original.

Figura 77: Gráficos - Hat

Escenarios

Un gestor de escenarios se encargará de ir generando el terreno, cada terreno tendrá sus enemigos diseñados teniendo que invocarlos y colocarlos antes de aparecer por pantalla. El escenario también se encargará de indicar a qué velocidad debe ir la carreta para poder adaptarse a los diferentes retos.

Al ser el prototipo 1 todavía no se busca extraer datos, los escenarios serán diseñados para entender Unity y crear un juego sencillo siendo el recorrido y la duración del juego cortos.

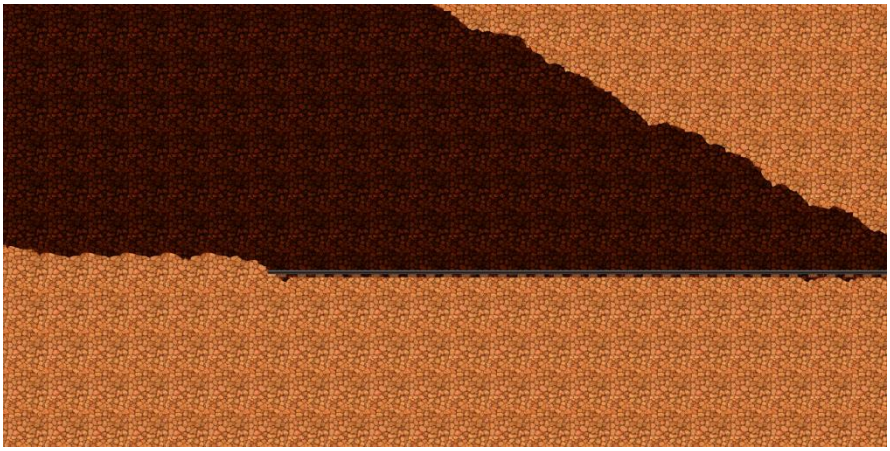


Figura 78: Gráficos - Escenario

Diagrama de clases

Diagrama con las conexiones y dependencias relevantes entre cada clase estudiadas para el desarrollo del juego. En este diagrama se ignoran todas las funcionalidades de las que se encarga el motor Unity, como son la interfaz gráfica, el gestor de sonidos y el gestor de físicas. Esto permite que el esquema sea entendible, práctico y aplicable a otros motores de videojuegos.

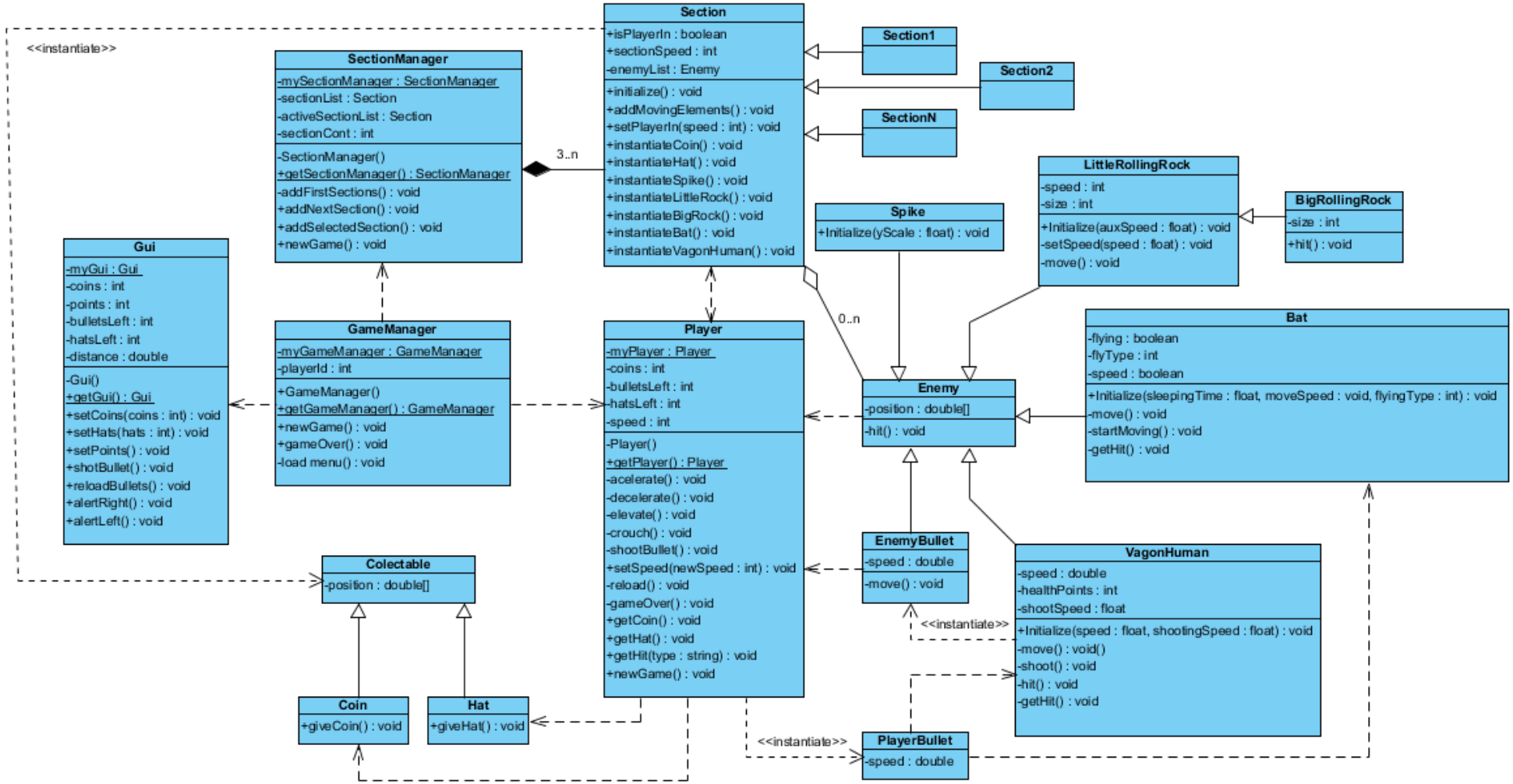


Figura 79: Arcade Prototipo 1 - Diagrama de Clases

GameManager:

Se encarga del principio y final de la partida dando orden al personaje principal y a las secciones para situarse en posición de inicio.

Gui:

Se trata de una capa de información. Encargada de mostrar al usuario datos sobre la partida en curso por pantalla. Enseña vidas, munición, puntos o mensajes.

SectionManager:

Durante la partida no hay un solo escenario, un nivel es compuesto por una gran variedad de distintos escenarios con una longitud predeterminada. SectionManager se encarga de iniciar los escenarios dependiendo del modo de juego, establece el escenario inicial y dos más, cuando se llega a la última sección se encarga de añadir la siguiente y eliminar la primera de forma que en memoria nunca existen más de tres y el jugador no llegue al vacio.

Section:

Cada una de las diferentes partes que componen un nivel, hay una gran variedad de secciones, todas heredan de sección. Cada una tiene sus gráficos de suelo y techo. También en cada sección los enemigos o recolectables generados son diferentes. Por último cada sección tiene su velocidad que es a la que avanzara la carreta del personaje principal.

Player:

Personaje principal, es el elemento que utiliza el jugador. Puede realizar movimientos y disparar.

PlayerBullet:

Balas que el personaje principal puede disparar para intentar dañar a los enemigos antes de que ellos lo dañen a él.

Colectable:

Elementos que el personaje puede recoger durante la partida para obtener beneficios.

Coin:

Hereda de colectable, son monedas que otorgan puntos extra al jugador

Hat:

Hereda de colectable, son sombreros que aportan vidas con las que poder seguir jugando.

Enemy:

Cada uno de los elementos que pueden dañar al personaje para impedir que llegue a su objetivo. Todos los enemigos son hijos de esta clase. El método hit de enemigo elimina una vida del jugador cuando colisionan.

Spike:

Pinchos que dañan al personaje si no los esquiva. Respecto a la clase Enemigo cambia sus gráficos y su inicialización.

LittleRollingRock:

Piedra rodante, arrebata una vida al jugador si entra en contacto.

BigRollingRock

Igual que la piedra pequeña pero de mayor tamaño y en caso de entrar en contacto con el jugador le arrebata todas las vidas.

Bat:

Murcielago que tiene un tiempo de sueño y luego empezará a volar. Puede ser golpeado y destruido por el jugador.

VagonHuman:

Carreta enemiga que en contacto con el personaje le arrebata todas las vidas. También puede disparar balas.

EnemyBullet:

Balas disparadas por la carreta enemiga que dañan al jugador en una vida.

Diagramas de secuencia

El desarrollo de un juego consiste en eventos, no hay un orden establecido en la ejecución de sus métodos sino que se ejecutaran unos u otros dependiendo de las acciones del jugador y como interaccione con el entorno de juego. Por tanto los diagramas de secuencia se dividirán en cada uno de esos posibles eventos haciendo que sean comprensibles y entendibles.

SecuenciaNewGame

Este diagrama de secuencia empezará cuando el jugador comience una partida. Es necesario cargar los datos principales, generar un escenario y colocar el jugador en él.

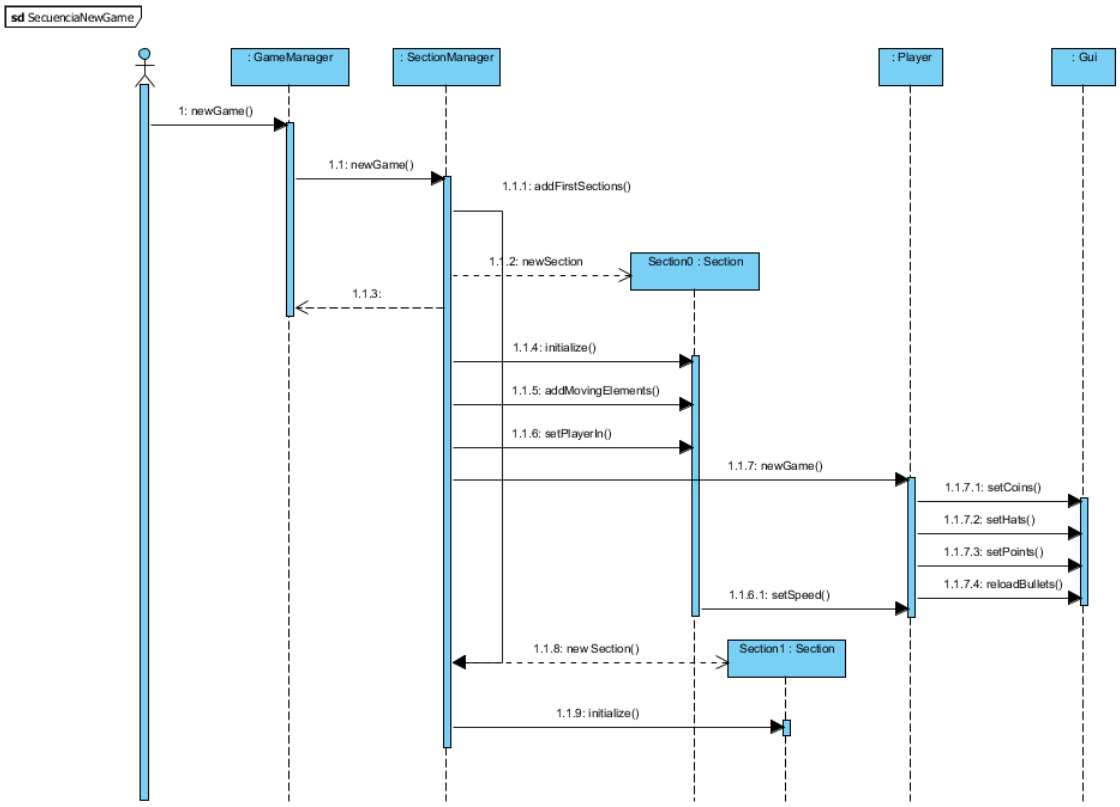


Figura 80: Arcade Prototipo 1 - Diagrama de Secuencia - New Game

SecuenciaPlayerEntersSection

El personaje recorre diferentes secciones montado encima de una carreta, cuando se termina una sección y se entra a otra hace falta ir preparando las siguientes y eliminando las ya no visibles.

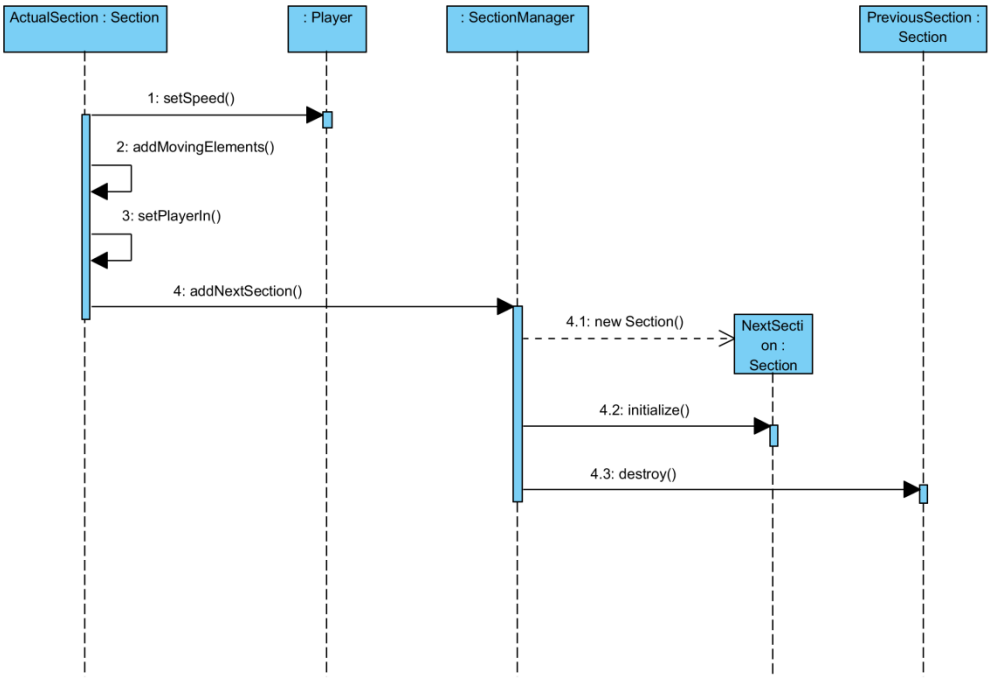


Figura 81: Arcade Prototipo 1 - Diagrama de Secuencia - PlayerEntersSection

SecuenciaPlayerInputs

El jugador puede pulsar uno de los cinco botones disponibles con los que mover el personaje, cada botón generará una secuencia diferente.

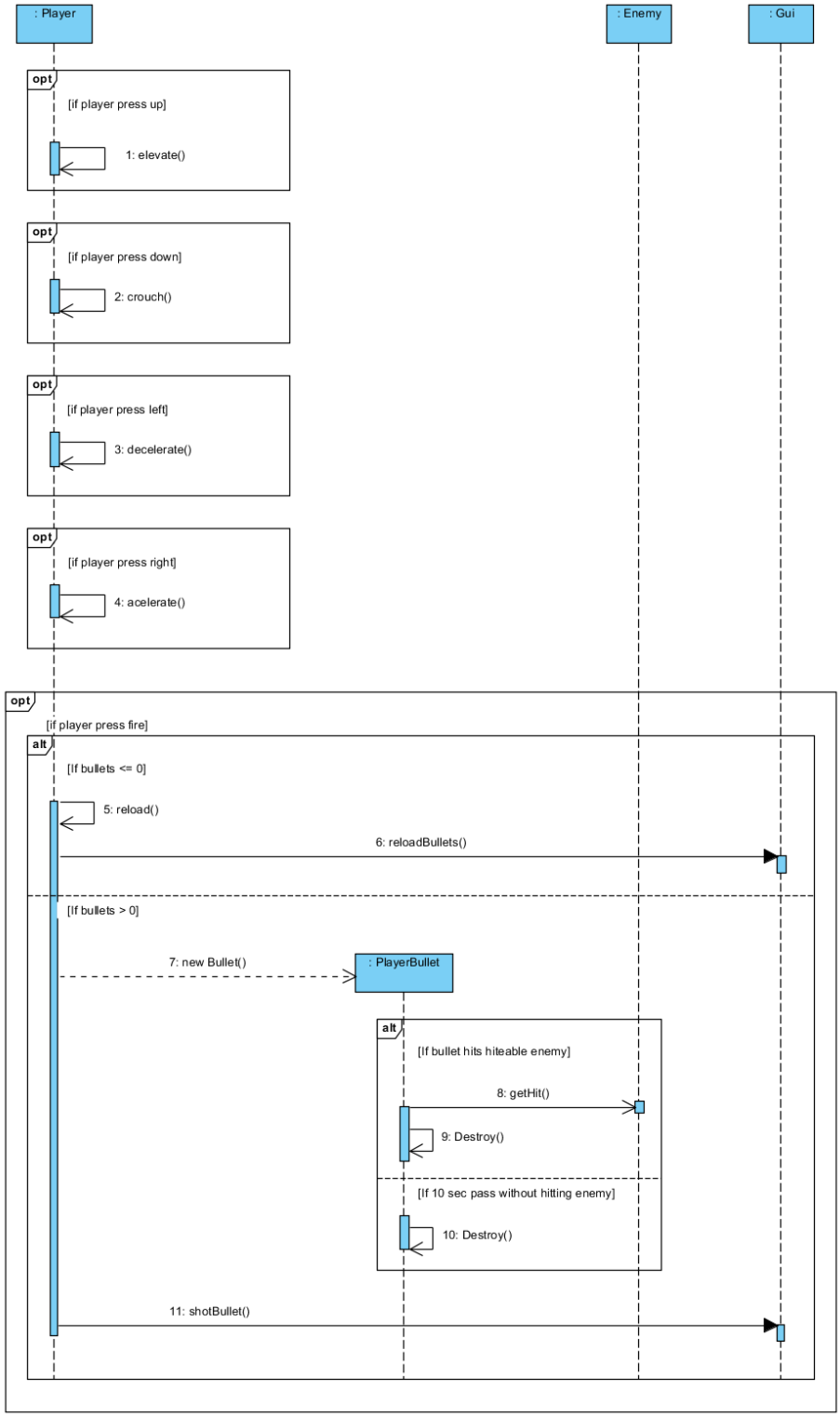


Figura 82: Arcade Prototipo 1 - Diagrama de Secuencia -PlayerInputs

PlayerGetHits

Si uno de los enemigos o sus balas golpea al jugador este debe recibir daño pudiendo resultar en el fin de la partida.

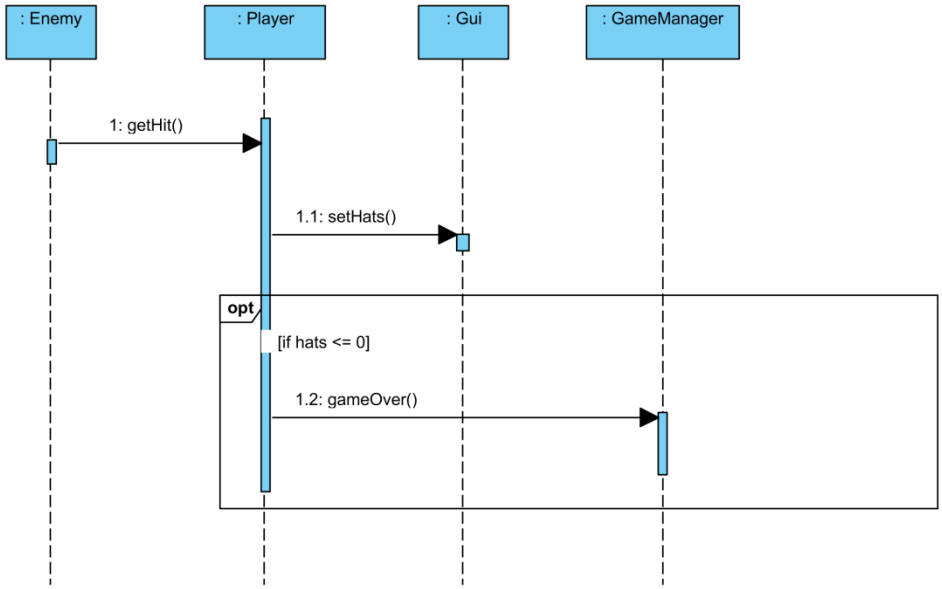


Figura 83: Arcade Prototipo 1 - Diagrama de Secuencia - PlayerGetHits

PlayerTakesColectable

De forma parecida a cuando el jugador toca un enemigo recibe daño, cuando toca una moneda o sombrero debe recibir puntos o vidas respectivamente.

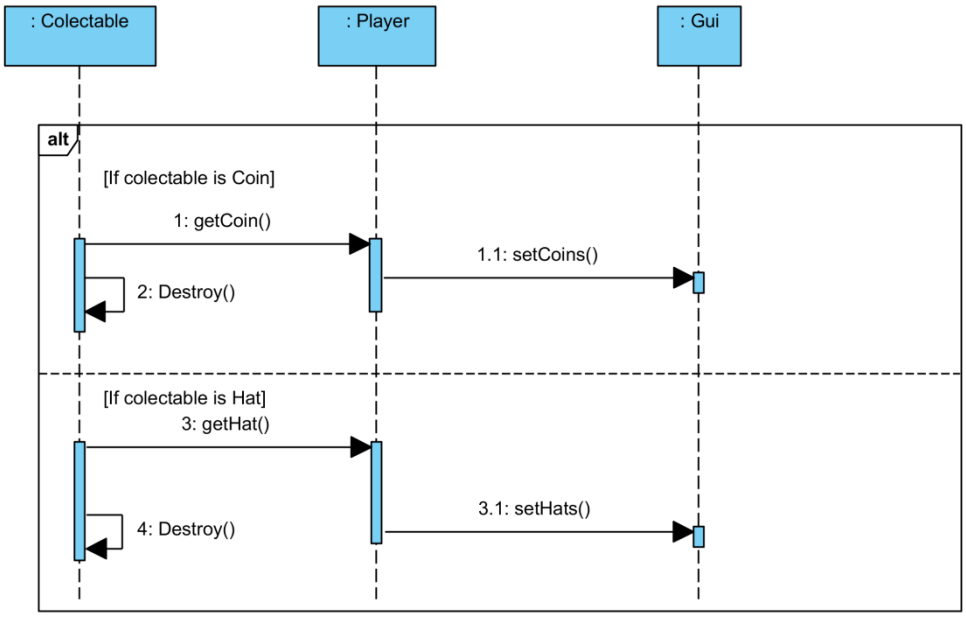


Figura 84: Arcade Prototipo 1 - Diagrama de Secuencia -

Pruebas Prototipo1

Las pruebas se han realizado de forma manual, al ser dos programadores cada uno ha podido probar su propio juego y el del compañero reportando los posibles fallos existentes.

En un videojuego se destaca la comprobación de que los valores que representan la posición de los objetos. Hay que evitar que no salgan del entorno del juego, procurando que si lo hacen el código haga que vuelvan a una posición correcta. Existen también muchas variables que van cambiando según las decisiones y acciones del jugador, como el número de vidas o puntuación. Gracias a Unity esos valores pueden verse y comprobar manualmente desde el inspector de elementos.

En el caso del Juego Arcade la mayor parte de los fallos consistían en bugs con las físicas del juego al no utilizar correctamente las clases y métodos aportados por Unity. También se ha tenido especial atención en comprobar que valores numéricos como las vidas del usuario se resten y se sumen correctamente siendo que si llegan a 0 el juego termine.

En el juego de naves el mayor problema era crear una zona correcta donde generar los objetos, puesto que una incorrecta posición harían que automáticamente fueran destruidos o generados directamente. Los niveles de disparo o la puntuación resultaron un elemento sencillo a vigilar.

Conclusiones Prototipo 1

Al finalizar el prototipo uno se obtienen dos juegos sencillos y funcionales sin menú. La única opción es jugar un escenario simple. Durante este prototipo el mayor reto es aprender a utilizar Unity y conseguir todos los recursos necesarios ajenos a nuestro campo de la programación, por ejemplo los gráficos y los sonidos.

Es complicado ceñirse a lo planteado, en un juego por sencillo que sea siempre surgen cientos de ideas de cómo mejorarlo tanto gráficamente como a nivel de jugabilidad, estas mejoras pueden crear conflictos en el código y problemas generando retrasos. Es importante no abrir varias ramas de mejora simultaneas o modificar código de una forma caótica sin tener presente el nivel de impacto en otras zonas del programa.

Dentro de Unity también existen varias formas de hacer la misma cosa, no siempre una de ellas es la óptima, depende de la situación o lo que se quiera lograr. Hay partes de la programación en Unity que podrían ser mejoradas, algunas han sido localizadas en una versión temprana y se han cambiado. Otras aunque no sean optimas son funcionales y el impacto de una modificación es demasiado grande.

En esta fase mediante observación y prueba surgen ideas de que datos podrían ser recopilados durante el prototipo 2, estos serán discutidos en su sección propia.

Prototipo 2

El prototipo 2 se divide en tres partes. El menú desde el que acceder a todas las funcionalidades del programa, el diseño de los niveles dentro de ambos juegos para que recopilen la información y la base de datos encargada de guardar los datos.

Menú

Esta sección será sólo para la parte concerniente al menú, más específicamente la comunicación entre ventanas.

Desde este menú será posible acceder a los juegos tanto en modo normal, tutorial o test. Además dentro del menú se podrá realizar la encuesta para poder obtener la clase real de las personas a partir de las que investigar.

Diagrama de clases

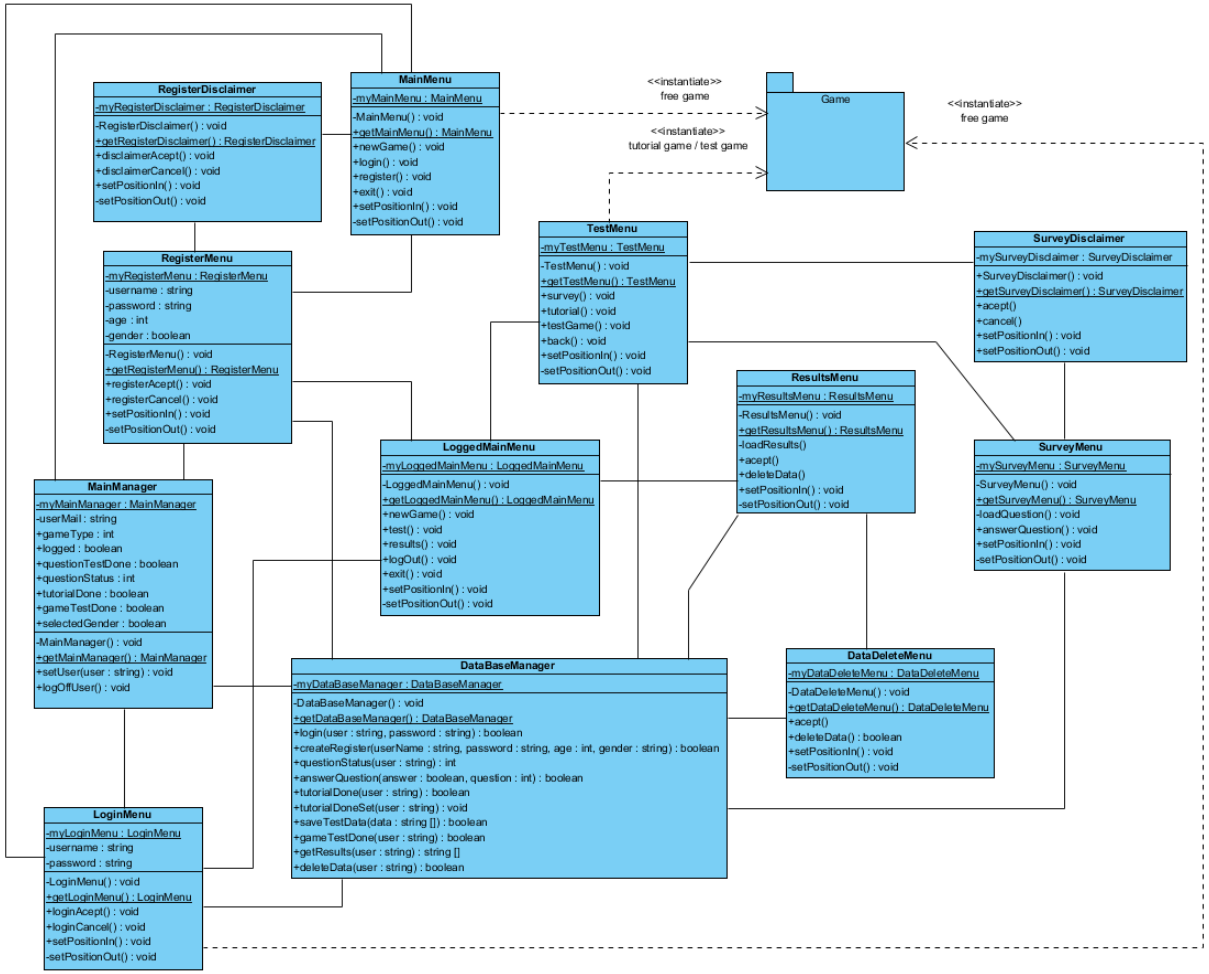


Figura 85: Menú Prototipo 2 - Diagrama de Clases

Diagramas de secuencia

Juego libre

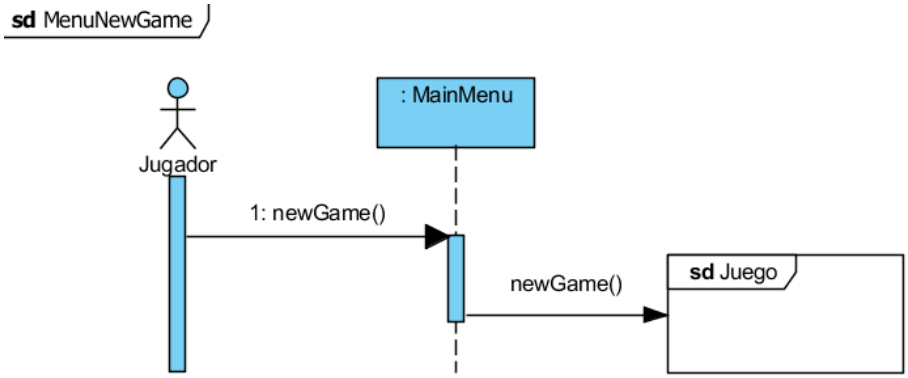


Figura 86: Menú Prototipo 2 - Diagrama de Secuencia - Juego Libre

Login

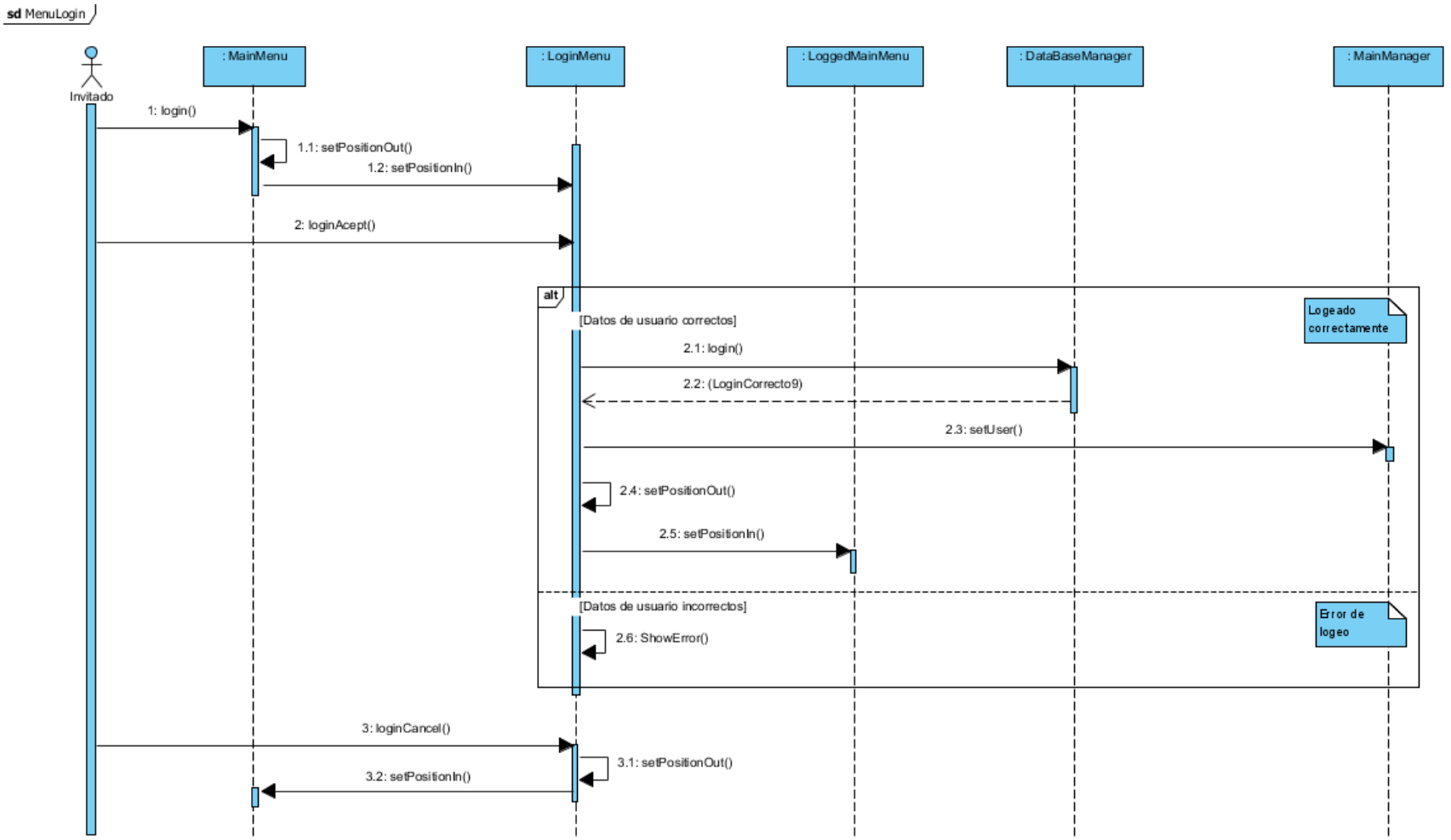


Figura 87: Menú Prototipo 2 - Diagrama de Secuencia - Login

Registro

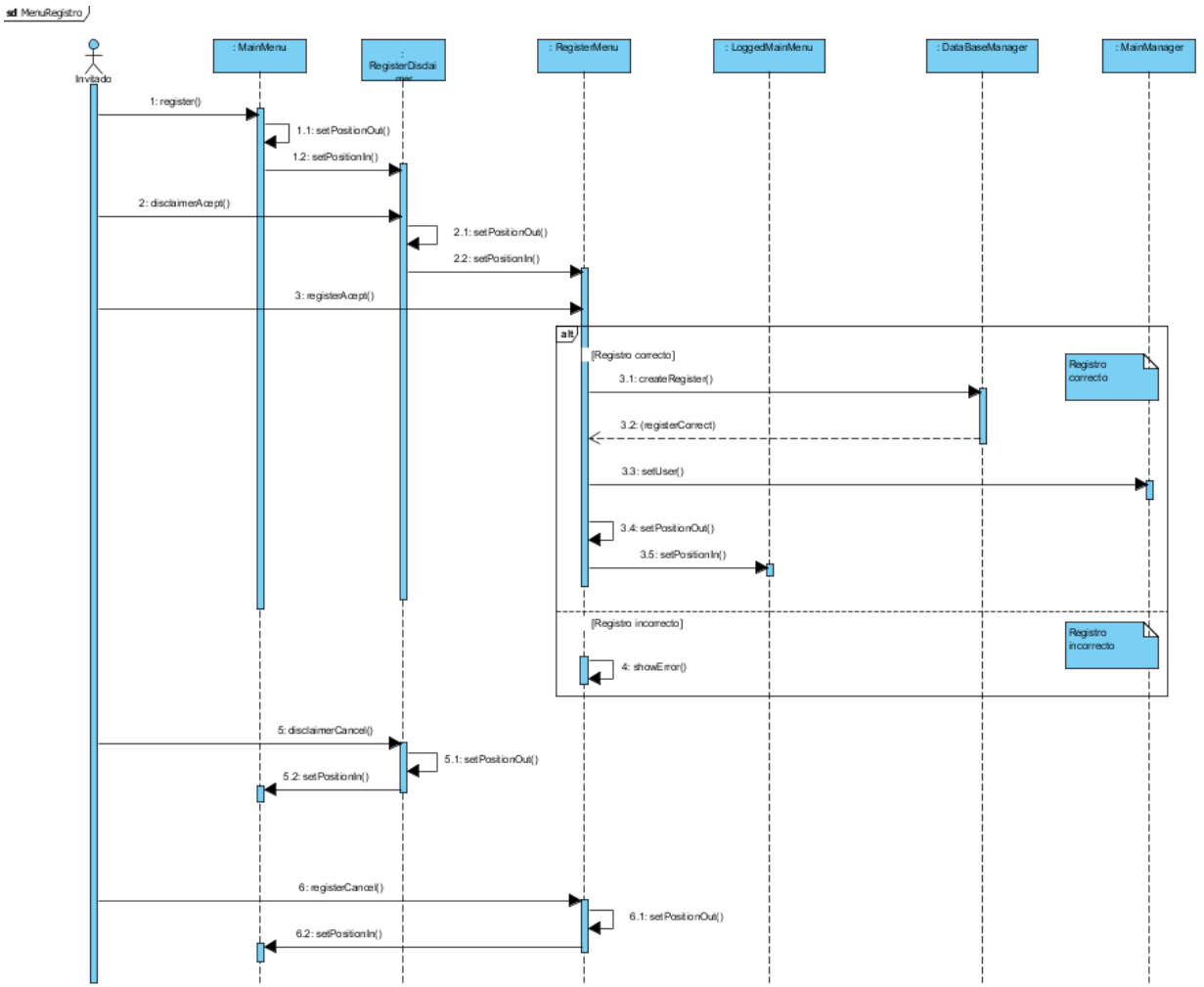


Figura 88: Menú Prototipo 2 - Diagrama de Secuencia - Registro

Rellenar encuesta

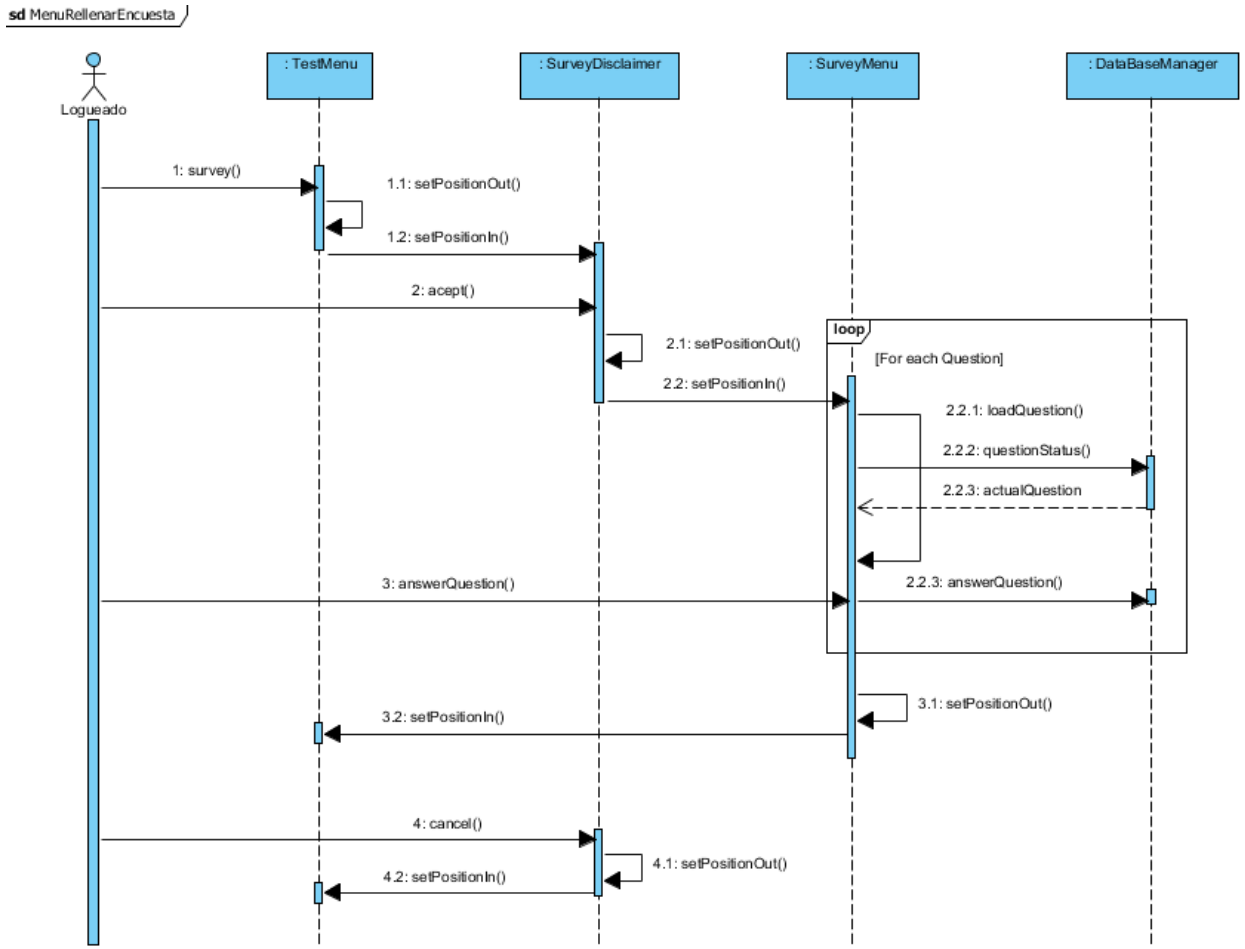


Figura 89: Menú Prototipo 2 –Diagrama de Secuencia – Rellenar Encuesta

Test

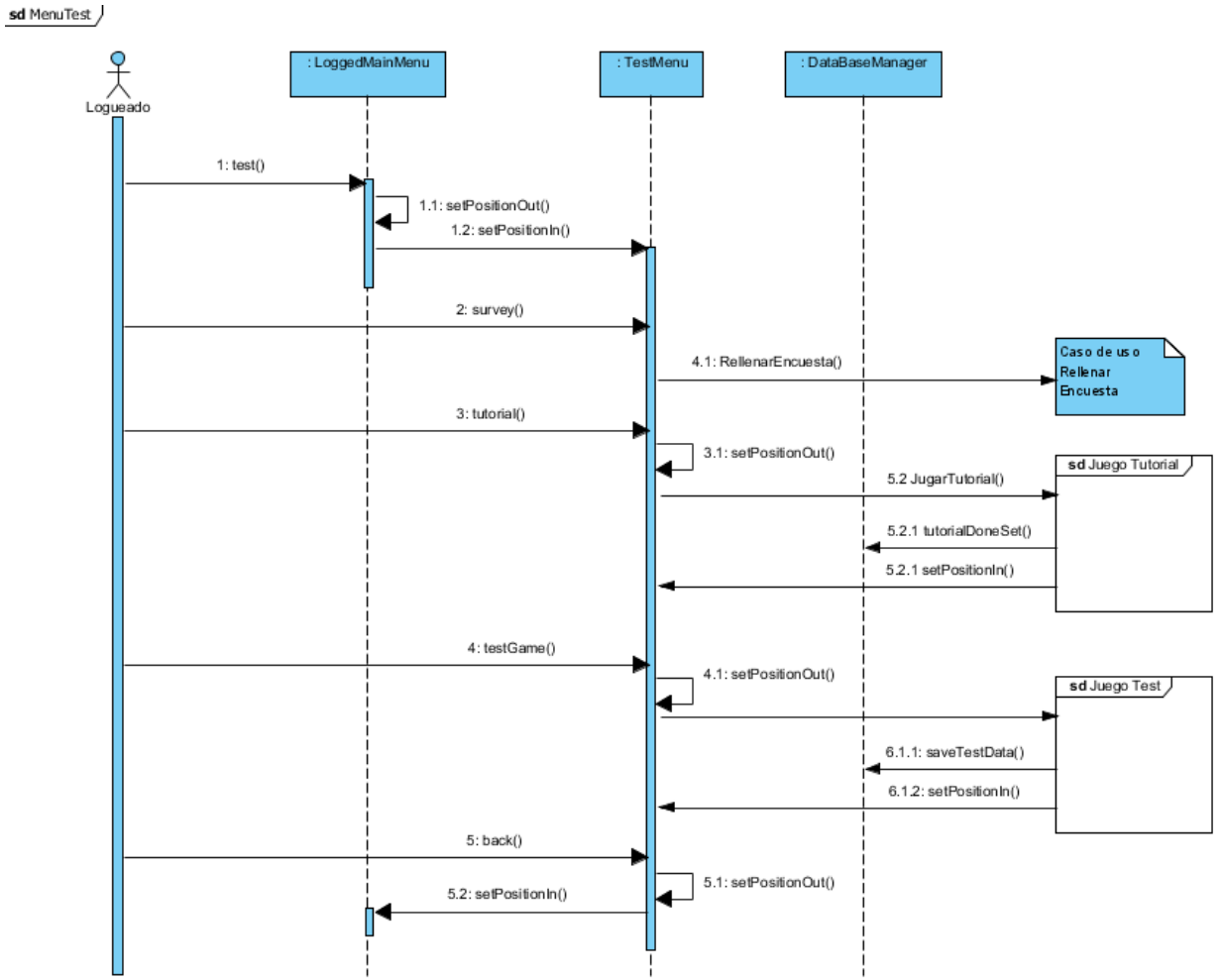


Figura 90: Menú Prototipo 2 - Diagrama de Secuencia - Test

Ver resultados

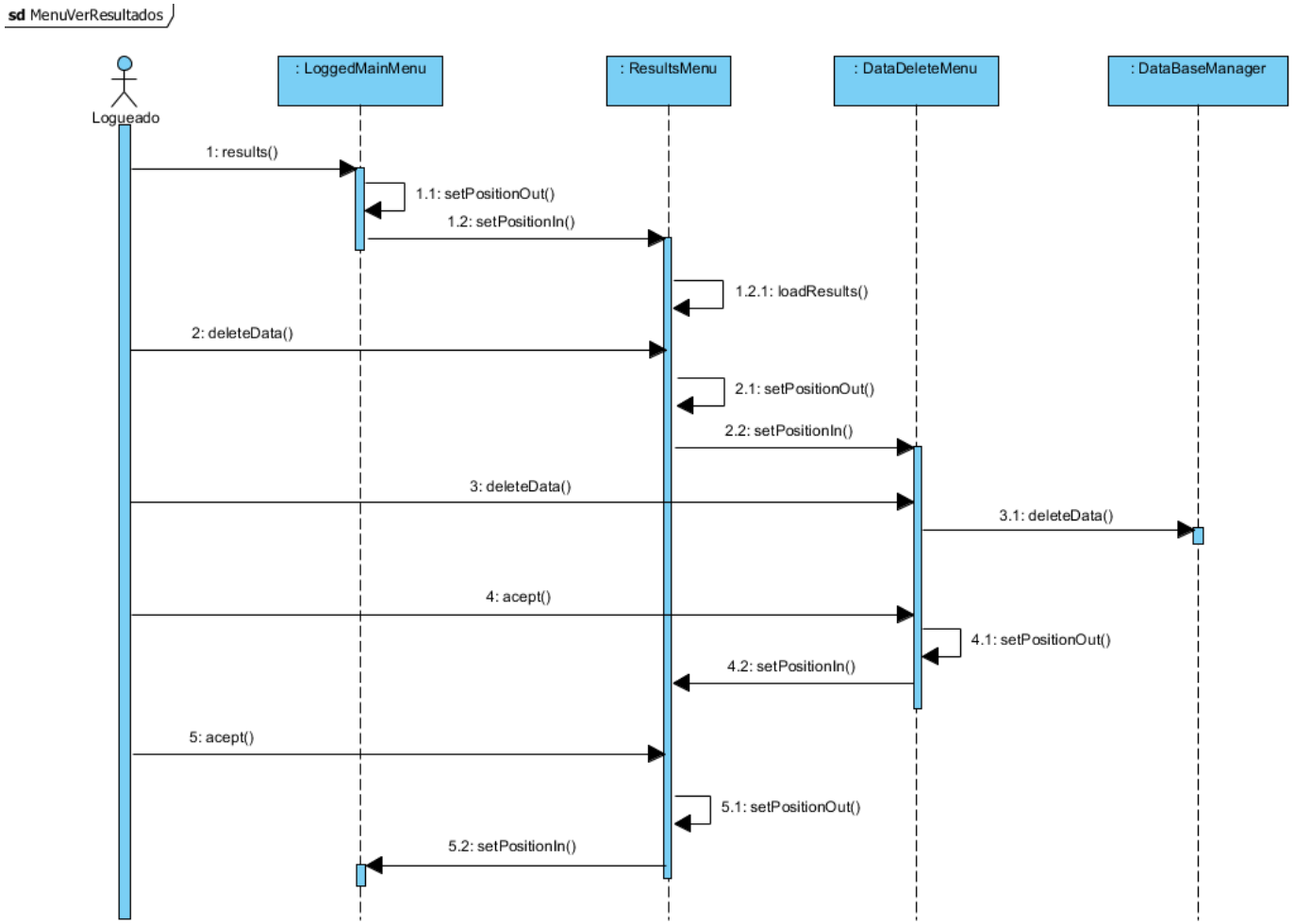


Figura 91: Menú Prototipo 2 - Diagrama de Secuencia - Ver Resultados

Juego Shooter

Parámetros a tener en cuenta en la recogida de datos

Se necesita preparar la toma de información para poder generar un modelo predictor. Es necesario modificar el proyecto del juego para que tenga en cuenta ciertos parámetros adicionales como la forma en la que se mueve le jugador, o que acciones realiza en situaciones concretas.

Para esto se ha decidido crear un total de 27 variables que toman distintos aspectos del juego. A continuación se presentan la lista de situaciones que se tendrán en cuenta para la captura de información. Para varias de ellas se plantea una hipótesis que podrán resultar ser ciertas o no una vez se realice el estudio.

Posición en pantalla (Nave)

Durante la partida, la nave cambiará su posición para colocarse en una zona más ventajosa en la pantalla, ya sea para atacar al enemigo como para esquivar posibles peligros. Se tomará en cuanta el tiempo que pasa en cada pantalla y se calculará y guardará el porcentaje de tiempo que pasa en cada una de las zonas.

Está hipótesis busca comprobar si el jugador prefieres posicionarse cerca de donde aparecen los enemigos mostrando una actitud extrovertida al buscar ser más activo o si es introvertido retirándose a una posición más alejada para preparar una estrategia ante el enemigo.

La zona de juego se dividirá en ocho zonas teniendo en cuenta el tamaño de la pantalla. Se calculará de cuánto será el ancho y alto de cada zona y mediante ese tamaño podremos saber en qué cuadrante se encuentra.

0	1	2
3	4	5
6	7	8

Figura 92: Gráficos - Posición en Pantalla

Variables a guardar:

- Porcentaje en zona 0
- Porcentaje en zona 1
- Porcentaje en zona 2
- Porcentaje en zona 3
- Porcentaje en zona 4
- Porcentaje en zona 5
- Porcentaje en zona 6
- Porcentaje en zona 7
- Porcentaje en zona 8

Elección de colores

El jugador puede elegir en cualquier momento el color de la nave para poder usar el tipo de proyectil asociado. En situaciones concretas un tipo de color puede ser ventajoso. Aun así no es obligatorio hacer ese cambio pues utilizando los otros colores también se puede progresar. O puede ignorar el enemigo completamente sin tener que considerar cambiar de color. Se guardará el porcentaje de tiempo que pueda estar en cada color.

Se busca conseguir información respecto a su percepción para hacer frente a los enemigos eligiendo el color óptimo para enfrentarse al enemigo sin ponerse en riesgo.

Variables a guardar:

- Porcentaje en color azul
- Porcentaje en color rojo
- Porcentaje en color amarillo
- Porcentaje en color verde

Disparos realizados

En cualquier momento de la partida el jugador puede disparar proyectiles, puede que lo haga con intención de prever por donde aparecerá un nuevo enemigo para recibir antes el golpe o simplemente por el placer de realizar una acción. Se calculará la cantidad de veces que la nave dispara durante la partida.

Se intenta tomar en cuenta la actitud del jugador viendo si muestra interés en enfrentarse al enemigo lo antes posible o prefiere esperar a ver como actúa. También servirá para analizar su percepción sabiendo si pone en práctica lo aprendido durante la partida o simplemente actúa de forma intuitiva.

Variable a guardar:

- Veces que ha disparado

Cuantos aciertos

Se calculará cuando el proyectil ha conseguido dar a un enemigo vulnerable. Existen ciertos enemigos que al ser golpeados no reciben daño por lo que no se tomará en cuenta.

En este caso la habilidad y puntería jugador toman un papel importante. Se espera que mediante esta variable se pueda conseguir ver una correlación que permita describir una característica concreta del jugador, aunque a priori no se vea ninguna.

Variable a guardar:

- Veces que se acierta a un enemigo vulnerable

Disparos sin enemigos

Una acción tan simple a veces llama a ser utilizando incluso cuando no es de necesidad si al hacerla no supone ninguna desventaja. El jugador puede disparar incluso si no hay enemigos a los que vencer. Se tomará en cuenta si sigue disparando incluso si no hay peligro en pantalla. Esto puede que aporte información en relación a su actitud, pudiendo ser las personas extrovertidas más activas en una situación controlada.

Variable a guardar:

- Veces que dispara sin haber enemigos en pantalla

Disparos contra enemigos invencibles

Ciertos enemigos tienen la habilidad de no sufrir daño cuando son golpeados con proyectiles. Cuando ocurre esto suena un sonido particular para que el jugador intente asociar lo que ocurre. Puede que a pesar de saber que el enemigo sea invencible desee seguir disparándole como entretenimiento, o simplemente que no se dé cuenta de esta característica hasta pasado un tiempo.

En este caso se intenta prever si el jugador ha comprendido que este tipo de enemigos no pueden ser vencidos. Tal vez puede que cierto grupo de personas disparen a propósito a estos adversarios simplemente por el hecho de tener algo a los que disparar.

Variable a guardar:

- Veces que se dispara a un enemigo invencible

Mejoras creadas

Cuando un enemigo es vencido hay una probabilidad de que deje tras de sí una mejora de disparo para el jugador. Si las recoge puede mejorar sus disparos si lo hace correctamente. Puede darse el caso que lo omita o no pueda cogerlo. Por ello se tomará en cuenta cuantas mejoras han aparecido durante la partida.

Esta variable trabajará con las otras relacionadas con las mejoras. Respecto a estos objetos se intenta buscar la actitud del jugador y su habilidad a la hora de obtener estas mejoras. Una mezcla de actitud y decisión indicarán si el usuario quiere arriesgarse en obtener esas mejoras o procurar estar en una situación segura.

Variable a guardar:

- Veces que aparece una mejora.

Total de mejoras recogidas

Si el jugador toca con su nave una de estos objetos , tendrá la oportunidad de mejorar el armamento de su nave para eliminar al enemigo de forma más eficaz.

Variable a guardar:

- Cantidad de mejoras que recoge.

Mejoras recogidas correctamente

Para que la mejora obtenida incremente el daño de los proyectiles debe ser tomado haciendo coincidir el color de la nave con el color del objeto. Se contará de las mejoras obtenidas cuales han coincidido correctamente al tocarlos.

Variable a guardar:

- Cantidad de mejoras cogidas correctamente.

Choques con enemigo

Aunque no parece una acción que se suela considerar no hay que olvidar que si el jugador así lo desea puede chocarse contra un enemigo. También puede ser de forma involuntaria durante una maniobra. Cuando un enemigo colisiona con una la nave, éste es destruido. Por ello se tomará en cuenta cuantas veces choca con un enemigo.

Mediante esta variable se intenta saber si aporta información en la actitud y decisión.

Variable a guardar:

- Cantidad de veces que choca con un enemigo

Choques contra proyectiles enemigos

El obstáculo principal del juego son los distintos proyectiles que aparecen. Cada uno se comporta de una forma diferente, por lo que evitarlos también requiere una estrategia distinta. Se contará cuantos de cada proyectil recibe el jugador.

De la misma forma que con los enemigos se comprobará la actitud del jugador por si tomara la decisión de evitar los proyectiles o no.

Variable a guardar:

- Cantidad de proyectiles horizontales toca
- Cantidad de proyectiles guiados toca
- Cantidad de proyectiles dirigidos toca

Enemigos creados

Durante la partida aparecerán un número de enemigos a los que se enfrentarán al jugador. Se tomará en cuenta cuántos de estos aparecen.

Esta variable únicamente servirá para buscar alguna correlación entre el resultado del perfil del jugador y como ha jugado.

Variable a guardar:

- Cuantos enemigos aparecen

Enemigos derrotados

Si el jugador es capaz de disparar continuamente a un enemigo hasta que quede sin energía, quedará destruido. Se tomará en cuenta cuantos es capaz de eliminar.

No es una acción requerida ni obligatoria. El jugador si así lo desea puede evitarlos completamente. Es por ello que tener en cuenta este comportamiento servirá para obtener actitud, así como su percepción y decisión.

Variable a guardar:

- Cuantos enemigos son derrotados

Diagramas Prototipo 2

En esta fase el juego debe guardar el comportamiento del jugador, por suerte muchas de las actividades que suceden por el juego ya están siendo tratadas por el sistema, por lo que añadir esta funcionalidad resulta más sencilla de lo que en un principio podría suponer. Tanto la Nave como el Gestor de Enemigos pueden llevar la cuenta de la mayoría de acontecimientos, el resto de eventos como el de los enemigos o los proyectiles crearían nueva comunicación para los datos relacionado con ellos.

Se añade una nueva clase que se encargará de ir recogiendo los datos que se utilizarán durante la fase de entrenamiento del modelo predictor.

A los métodos principales se les añade un método para que vayan enviando la información al nuevo método para cuando acabe la partida.

Diagrama de clases

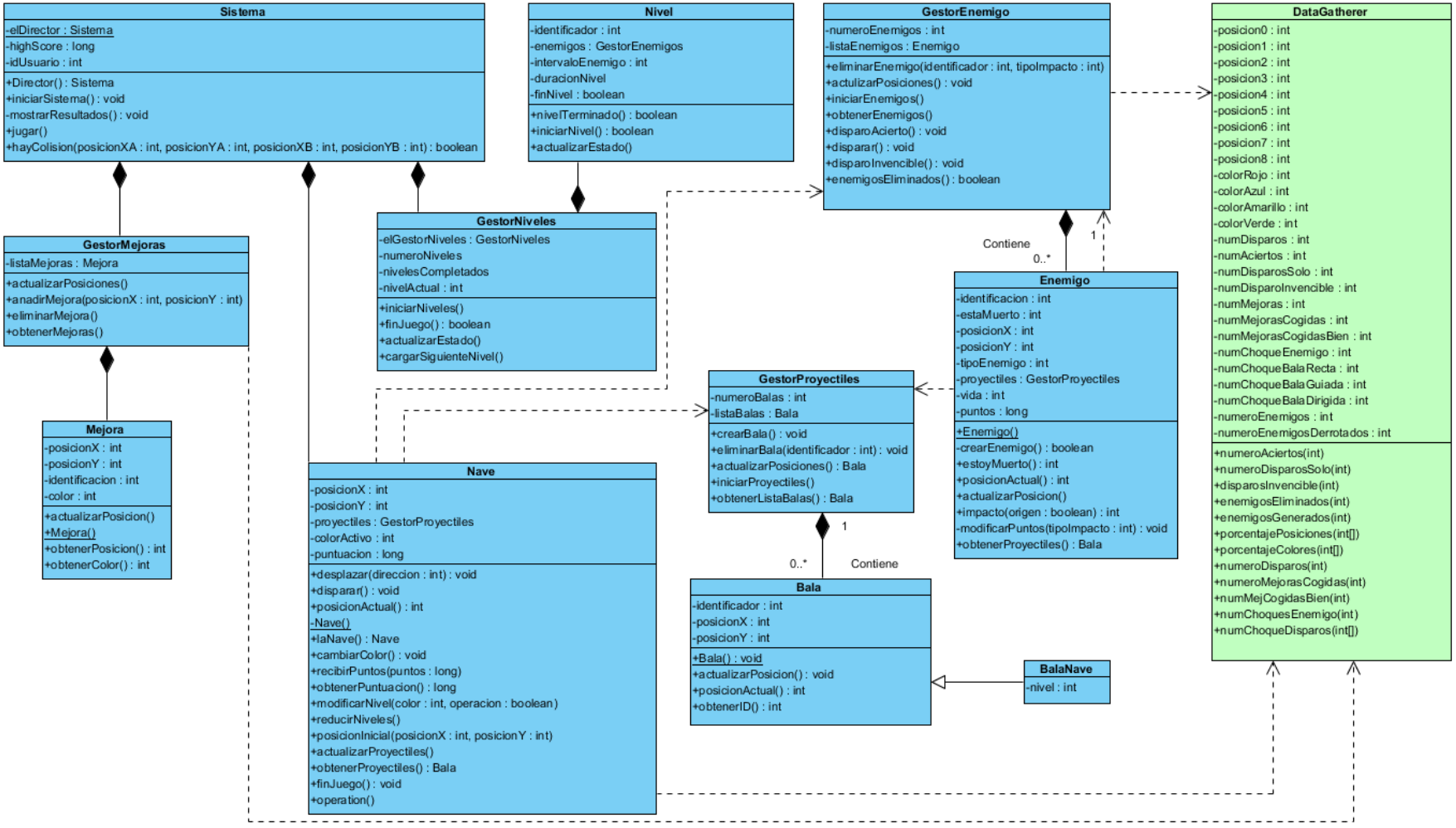


Figura 93: Shooter Prototipo 2 - Diagrama de Clases

Diagrama de secuencia

Los diagramas de secuencia no cambian respecto al prototipo 1. Solo se mostrará el proceso de guardar los datos en la base de datos que ocurre al finalizar la partida cuando se llega a la última sección.

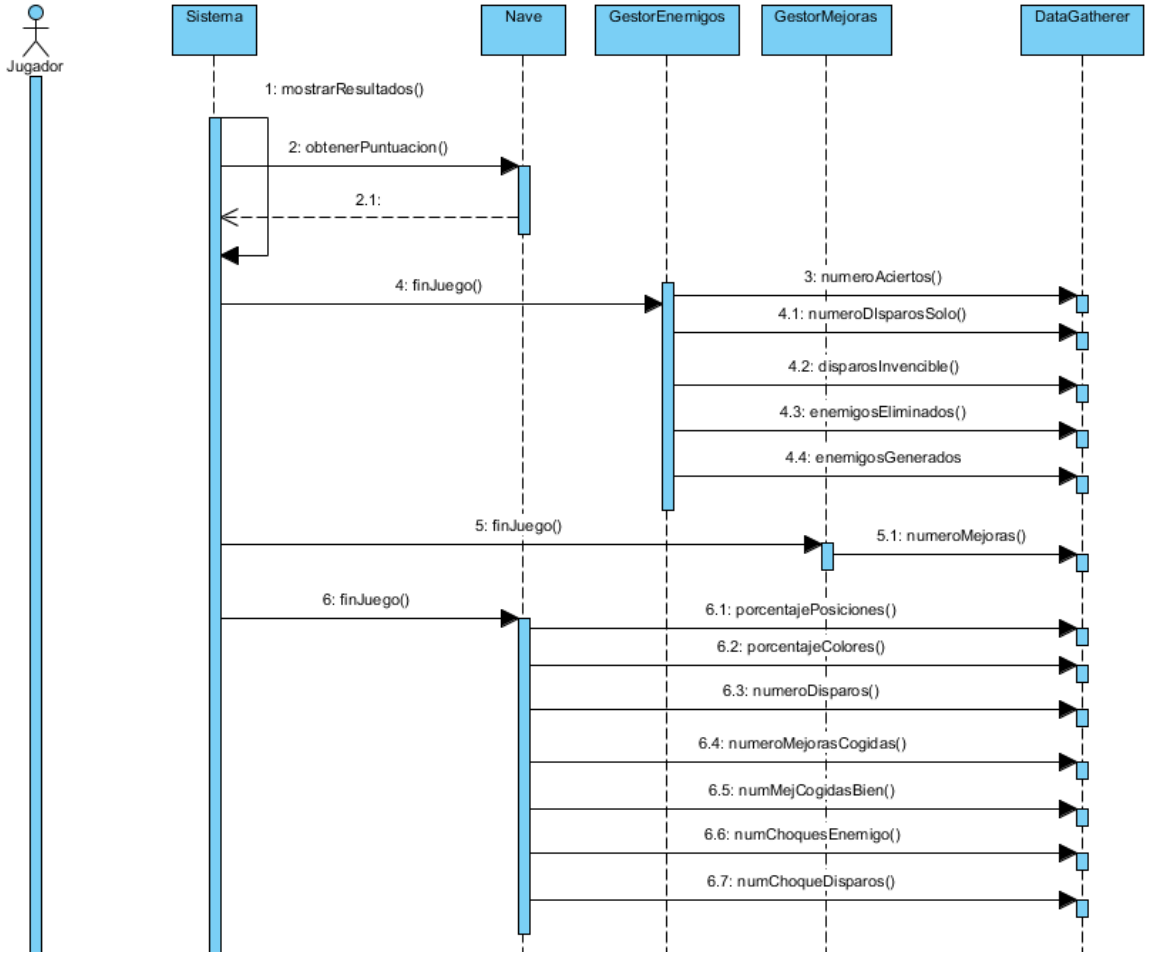


Figura 94: Shooter Prototipo 2 - Diagrama de Secuencia

Juego Arcade

Parámetros a tener en cuenta en la recogida de datos

Una vez realizado el estudio de psicología, aunque recogeremos varios datos del juego, algunos de ellos serán buscados esperando una información específica. Es decir acciones que nos aporten información en uno de los cuatro grupos del indicador Myers-Brigss descrito en su sección correspondiente. Estas acciones son ideas que se han ido recogiendo durante el desarrollo del proyecto y que tendrán que ser implementadas como escenarios dentro del juego. Con ellas en mente lanzamos diferentes hipótesis, todas basadas en las descripciones de los rasgos del test Myers-Brigss. Algunos de los campos que se guardarán no tienen relación directa con ningún campo, pero quizá el módulo de minería de datos pudiera extraer alguna conexión.

Selección de personaje:

Esta hipótesis se centra en la posibilidad de que una persona de un sexo específico seleccione el sexo contrario. Esta persona está tomando una decisión alejada de la realidad, esto puede estar relacionado con la percepción de la persona, alguien intuitivo busca nuevas posibilidades.

Al comenzar la partida aparecerán dos recuadros, cada uno con el retrato de una mujer y un hombre. La elección será guardada en una variable booleana dentro de Player y será contrastada con el sexo introducido en el test.

Enviar información errónea al usuario:

Si el usuario recibe información, como quizás aviso sobre un peligro, y es mentira, ¿se fiara? Quizá la primera vez sea engañado, o siga confiando en las indicaciones más que en su propia visión. Esta prueba puede que aporte información tanto en el apartado de percepción como de decisión ya que tiene en cuenta si el jugador se fía más de lo que le decimos que de su propia opinión.

Se realizarán tres pruebas seguidas, dos falsas y una última verdadera, si en la última es dañado significará que ya no se fiaba de las alertas o consejos, esta información se guarda dentro de cada sección.

Tendencia a adelantarse o atrasarse:

El jugador puede tanto acelerar como decelerar, esto hace que la visión del mapa cambie permitiéndole ir más despacio o rápido y ver los obstáculos desde más lejos. Un jugador que tienda a decelerar puede ser una persona calificadora, necesita tener todo planificado y espera para conseguir más información, es más insegura y cuidadosa. En cambio un jugador que decide ir más rápido será una persona perceptiva a quien le gustan los riesgos y prefiere la diversión sobre la seguridad. Esto puede aportarnos información sobre la estructura.

Se utilizarán diferentes variables, esta tendencia tiene relación con la pulsación de dos teclas, la de ir hacia adelante y la de hacia atrás. De cada una de las direcciones se pueden guardar dos datos, número de pulsaciones, y tiempo total de pulsación. En conclusión tendremos un método que vaya calculando y sumando cada una de ellas. En total necesitaremos cuatro variables, dos integer y dos floats que se guardarán dentro de la clase Player.

Durante el juego cuantos botones pulsa:

Pulsa botones de forma aleatoria, prefiere solo pulsar cuando es necesario, o tiende a estar en movimiento en todo momento aunque no sea necesario. Esta prueba tiene la complicación de que quizá un jugador más inexperto tienda a tocar más botones. Puede aportar información sobre la actitud. Una persona extrovertida tiende más a la acción con lo que puede que realice más acciones que una persona introvertida, que al ser más reflexiva quizás dedique más tiempo a pensar que a pulsar.

En cierto modo se solapará con el punto tres, pero seremos más genéricos, guardaremos el número total de pulsaciones y de tiempo de pulsaciones que implique a todas las teclas. Para ello utilizaremos, por cada tecla, una variable integer, para la cantidad de pulsaciones, y otra doublé para el tiempo de pulsación dentro de la clase Player.

Como actúa frente a enemigos:

Existen enemigos que no son necesarios de matar. Jugadores activos preferirán ganar puntos y arriesgarse a dañarlos, jugadores más prudentes se conformaran con esquivar.

Se les añadirá a los enemigos un nuevo script, antes de ser destruido enviará un aviso para poder contabilizarlo, así al final de la partida sabremos el número total de enemigos y el número de enemigos destruidos, serán dos variables que podrán convertirse en un porcentaje.

Ante un mismo peligro reacciona siempre igual:

Hay jugadores que prefieren experimentar diferentes sensaciones, por ello intentan superar un mismo obstáculo de todas las formas posibles. Otros van a lo seguro. Esto puede aportar información sobre la estructura. Además algunas formas de evitar estos enemigos no serán explicadas, es el propio jugador quien por lógica llegará a esa conclusión, esto también nos aporta información sobre la percepción, por ejemplo si decide esquivarlos, el resultado se guardará dentro de la sección.

Reacción en caso de quedarse a ciegas:

Podemos forzar que durante una sección del mapa el jugador no pueda ver qué sucede. Algunos jugadores decidirán golpear botones aleatoriamente o disparar en caso de que un enemigo venga, otros se quedarán quietos hasta que esta situación anormal pase. Nos aportara información sobre la percepción.

Durante el tiempo que dure la sensación de ceguera, la cual conseguiremos con una tormenta de nieve como se muestra en la imagen, guardaremos que teclas ha pulsado y en qué cantidad. Se guardará en la clase Player.

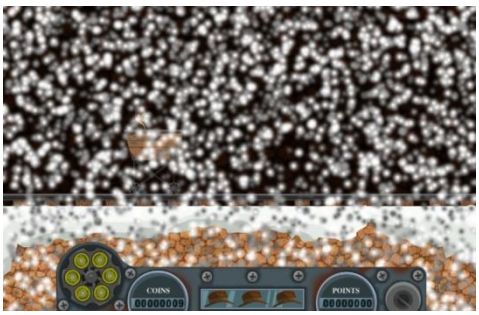


Figura 95: Gráficos - Nieve

Colocar recompensas en sitios aparentemente inaccesibles:

Que pasara si una recompensa se encuentra en una pared aparentemente sólida, ¿intentará recogerla? ¿Y si está de forma que el jugador sepa que va a recibir un daño? ¿Preferirá la recompensa o la seguridad? Nos dará información tanto de la percepción como de la decisión.

Esto se logrará situando sombreros (vidas) y monedas atravesando el techo, si el jugador al que previamente se le ha avisado sobre el daño que puede causar el techo se arriesga y las recoge se guardará el resultado positivo en una variable de tipo numérico para saber cuántas de las recompensas que requieren riesgo a recolectado. Se guardará en la clase Player.

Vidas o puntos

Si le damos la oportunidad a un jugador de obtener vidas con lo que poder jugar más tiempo, o acumular una puntuación mayor que elegirá.

Se situará una vida en forma de sombrero, y una moneda a la par. Repetiremos esto y sabremos en qué porcentaje ha decidido las vidas sobre las monedas. Se guardará en la clase Player.

Camino fácil o difícil

En un momento del camino se mostrarán dos posibles desvíos, uno será sencillo y con pobres recompensas, y otro complicado con mayores recompensas. Aportará información sobre la estructura.

Realmente los dos caminos pueden ser exactamente igual, será la propia elección, no el camino, lo que marcará la inclinación del jugador hacia el riesgo. Se guardara en la sección.

Conclusiones

En todos los casos la información que se podrá aportar no es segura, puede no informar en nada, o ser útil en más de un grupo. Estos diseños de niveles son intentos de forzar la obtención de información. Aparte de los anteriores datos se intentarán recoger muchos más, como pueden ser tiempos de pulsación, munición utilizada, tiempo de recarga a recarga, etc., esperando que alguno de ellos nos consiga aportar también información significante.

Diagramas Prototipo 2

Se deben guardar los datos anteriormente citados, la base del juego ya está programada por lo que añadir esta funcionalidad es sencillo. Se creará un nuevo objeto al que se le notificará cada una de las acciones para que lleve cuenta de los datos. Todos los datos a recoger se accionan en algún momento del juego mediante métodos propios de Unity. Cuando la partida termina el nuevo objeto pide los datos a cada elemento implicado para posteriormente enviarlo a la base de datos.

Se recopila información desde tres fuentes diferentes: las secciones, el jugador y los enemigos.

Diagrama de clases

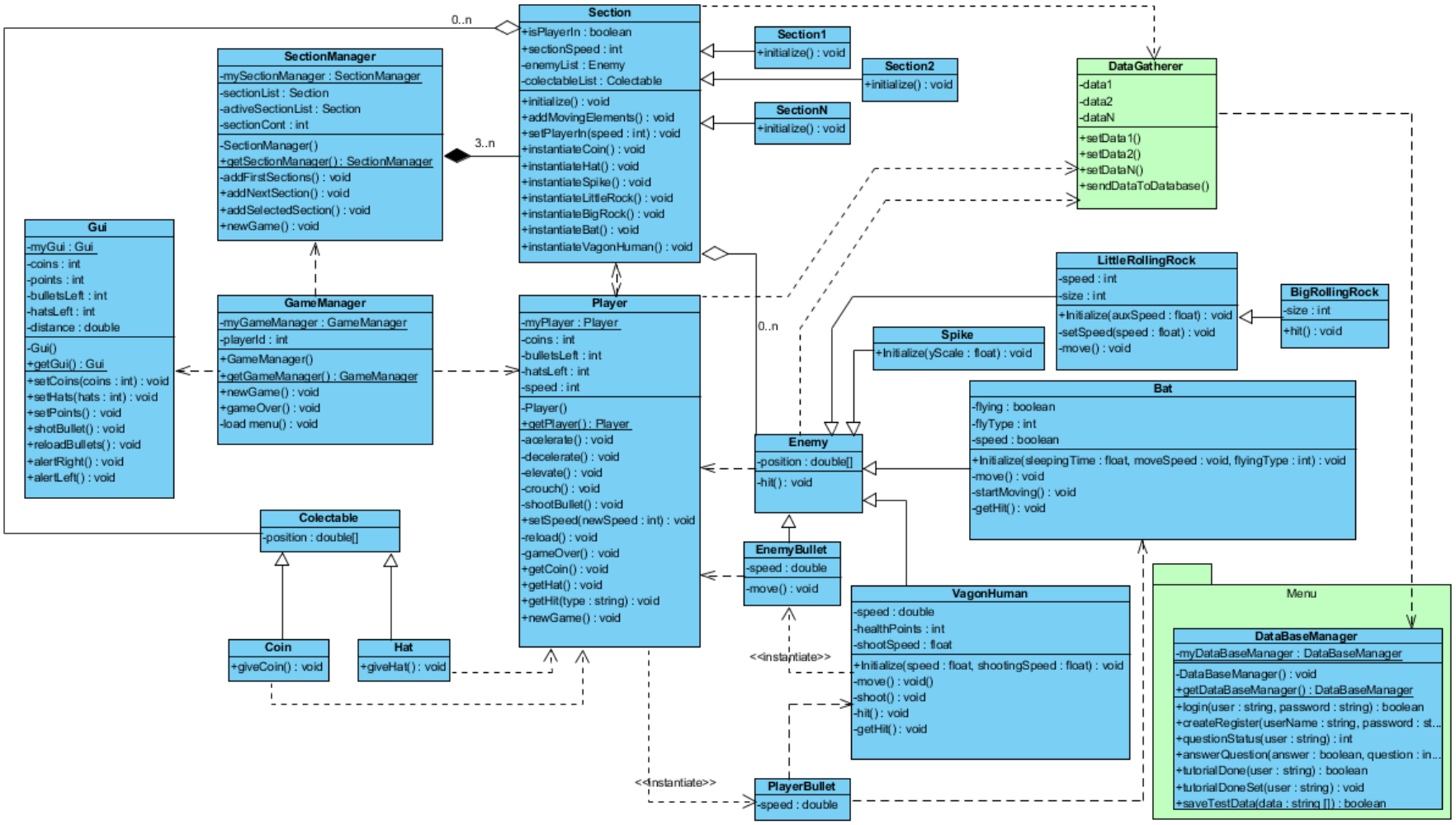


Figura 96: Arcade Prototipo 2 - Diagrama de Clases

Diagrama de secuencia

Los diagramas de secuencia no cambian respecto al prototipo 1. Solo se mostrará el proceso de guardar los datos en la base de datos que ocurre al finalizar la partida cuando se llega a la última sección.

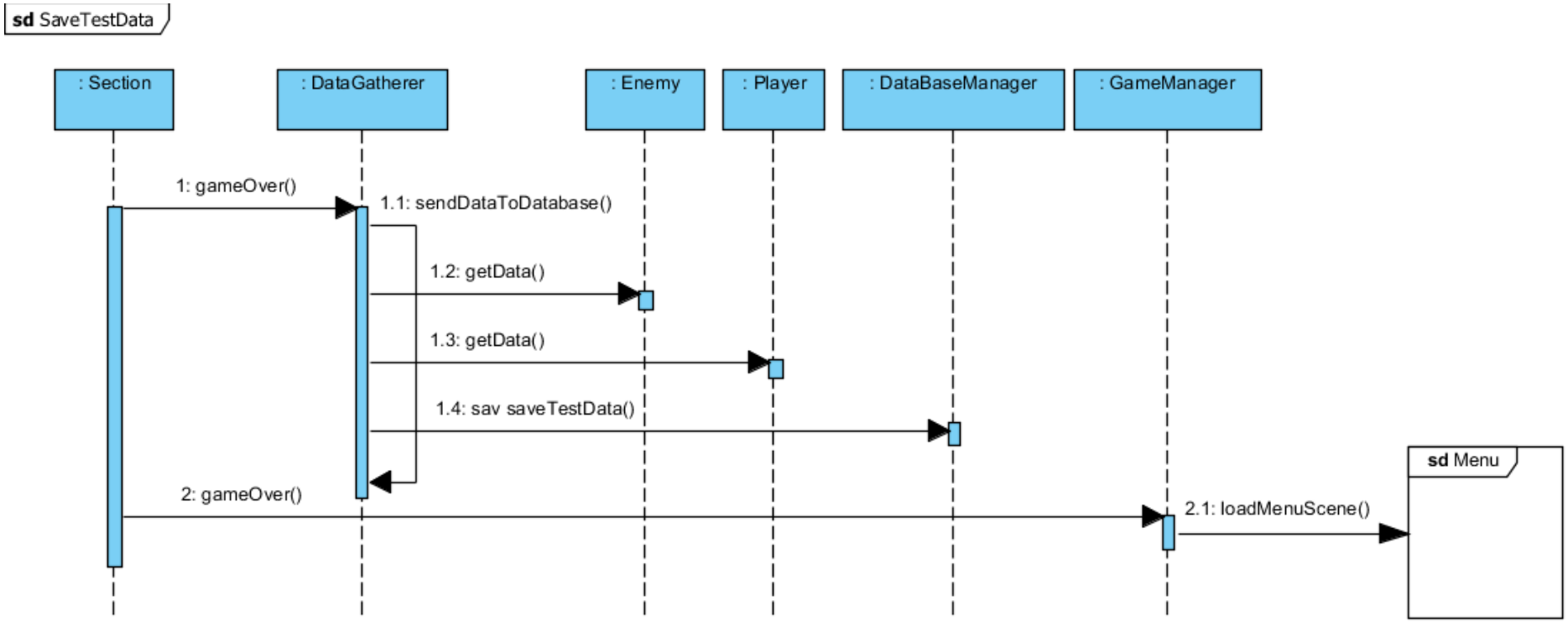


Figura 97: Arcade Prototipo 2 - Diagrama de Secuencia

Base de datos

Diseño de la base de datos

Con intención de poder almacenar toda la información que se desea recopilar se decide que el juego se conectará a una base de datos. En ella se guardarán los datos del jugador tanto de sus respuestas del test escrito así como sus resultados durante los juegos.

Del usuario se necesitará un usuario y contraseña, además de una edad y género por si se desea realizar un estudio también a partir de esas características.

Del test escrito se guardarán las respuestas a cada pregunta. Para permitir al usuario poder continuar más adelante puesto que el test puede resultar demasiado largo para hacerlo de una vez, se almacenará cual es la pregunta en la que ha decidido parar. Si se retoma el test más adelante, se sabrá donde lo dejo. Una vez terminado el test, se guardará el resultado.

Ambos juegos disponen de un número de variables que almacenar. Además existe un tutorial al que se debe comprobar que ya ha completado para poder realizar la partida normal. Se guardarán cuando se han realizado ambas cosas.

Las preguntas al cuestionario también se almacenarán en la misma base de datos para que resulten fácilmente accesibles para la aplicación. La ID de la pregunta coincidirá con el número de la pregunta que está respondiendo en ese momento el usuario.

Mediante esta descripción, se crea el siguiente diseño de base de datos:

- user: (mail, password, age, gender, id_quiz, id_game_shooter, id_game_arcade)
- quiz_test: (id_quiz_test, current_question, results_test, ans_1, ans_2, ans_3, …, ans_4)
- game_arcade: (id_game_arcade, tutorial_done, game_done, v1, v2, v3, …, v28)
- game_shooter: (id_game_shooter, tutorial_done, game_done, v1, v2, v3, …, v27)
- questions: (id_question, question)

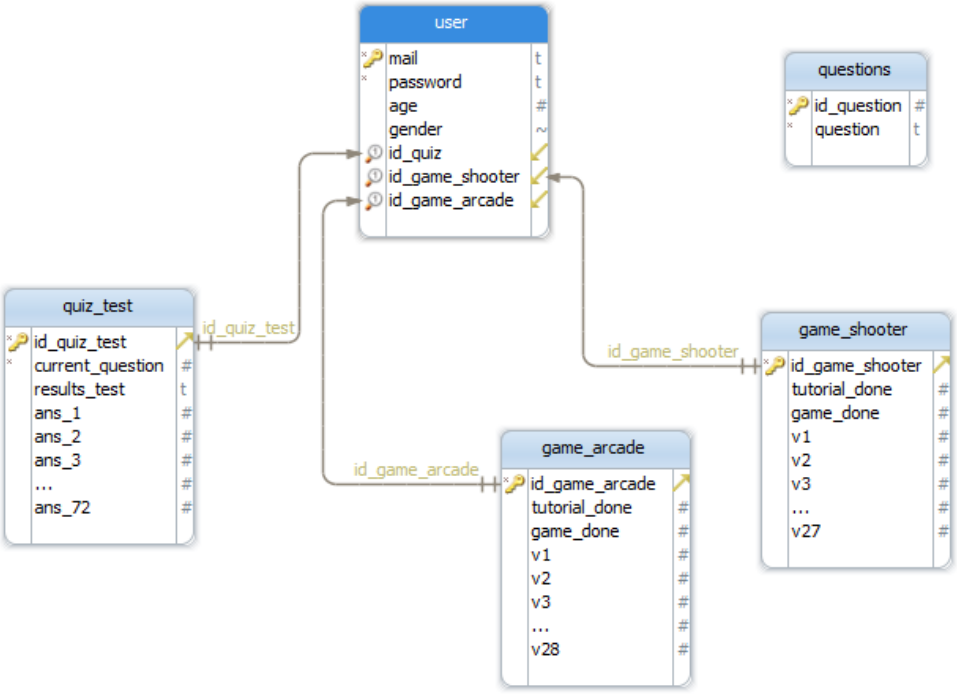


Figura 98: Prototipo 2 – Modelo Entidad Relación

La programación de toda la parte referente a la comunicación entre la base de datos y el proyecto se encuentra en el anexo correspondiente.

Pruebas prototipo 2

Las pruebas se han dividido en partes, las del menú , la recogida de datos del juego y el envío de los datos al servidor. Nuevamente se han hecho manualmente.

La primera prueba es comprobar que cada botón del menú lleva al usuario a la ventana correspondiente. Uno de los errores ocurridos es el solapamiento de menús. En la representación gráfica de Unity quedaron unos menús sobre otros, así cuando se pinchaba un botón también se accedía al que estaba “oculto” detrás. También se ha tenido que comprobar y corregir errores respecto al estado del usuario. Las acciones de Login y Registro comprueban que los datos introducidos sean del formato correcto, posteriormente en la base de datos en el caso de Login se comprueba la contraseña. Es importante que después de la acción el usuario quede Logueado, aunque se muestre el menú correspondiente si la variable que nos indica que el usuario esta Logueado es incorrecta la navegación entre ventanas causa errores.

Para poder comprobar el correcto funcionamiento se ha desactivado el cifrado, de esta forma es posible ver en la base de datos que los resultados son los esperados. Estotambién sirve para la recogida de datos desde el juego. Una vez pueden verse losdatos comprobar el correcto funcionamiento del programa es fácil.

La encuesta consta de 72 preguntas, rellenar todas las preguntas y que en la Base deDatos a cada pregunta se le correspondiera su respuesta necesito varias pruebas. Se probó a pausar la encuesta sin responder ninguna pregunta, solo respondiendo una, todas menos una y todas.

Por ultimo para que un usuario realice el test en orden las opciones se bloquean y desbloquean a la marcha dejando solo un botón activo a la vez. Es una parte delicada ya que más de una variable está implicada junto a la base de datos. Se hicieron varios usuarios de prueba y se realizó toda la secuencia varias veces para buscar posibles errores.

Para poder enviar los resultados al servidor hay que tener en cuenta que al ser información protegida hay que asegurarse de que los datos enviados o recibidos por el servidor son validados. Este requisito permite cerciorarse que ninguna persona ajena al juego podrá introducir o recuperar cualquier tipo de información de la base de datos. Se comprueban distintas posibles combinaciones.

Antes de empezar a recopilar datos de usuarios estas cuentas se eliminaron para noaportar datos falsos al estudio.

Conclusiones prototipo 2

Una vez terminados ambos juegos el menú genérico ha sido integrado en ellos teniendo solo que cambiar el aspecto gráfico, los sprites. En este momento del proyecto ya se puede recopilar información con los usuarios, esta es guardada en la base de datos de forma automática.

Un usuario debe realizar el test una sola vez, en el momento que lo rellena en uno de los dos juegos el otro no se lo pedirá. Para recopilar datos no es necesario que un usuario juegue a ambos juegos. Cada juego recopila su información de forma independiente. Más adelante se podría decidir si utilizar la información de los juegos de forma individual o conjunta en caso de que algunos usuarios hayan finalizado ambos.

Como se propuso a la hora de decidir el ciclo de vida del proyecto este prototipo ya es totalmente funcional.

Como el menú y la base de datos se han diseñado de una forma genérica e independiente del juego, otra persona podría seguir añadiendo juegos, estos podrían aportar más datos permitiendo generar un decisor mejor.

Prototipo 3

El tercer ciclo del proyecto es considerado una ampliación de los prototipos 1 y 2. Esta fase implicaba un reto muy grande que es la recopilación de instancias de test, es decir personas que utilicen el software y aporten datos de los que extraer información. El número de instancias conseguidas es muy bajo haciendo que los resultados que obtengamos no sean fiables.

A pesar del problema del reducido número de instancias se ha seguido adelante con el proyecto al fin de tener la aplicación completa programada, tanto la parte de Unity como su conexión con el modulo predictor. En caso de que en un futuro se consiguieran suficientes instancias solo habría que estudiar los modelos predictores óptimos.

Funcionamiento

El tercer prototipo es el que más partes separadas contiene. Requiere combinar varios lenguajes de programación y estructuras.

Cuando el jugador termine una partida lo primero es crear un archivo ".arff". Este archivo contiene la información recopilada del jugador. Con el archivo necesitamos que Weka lo analice. Esto lo conseguimos con un archivo ".jar" que llama el modelo predictor que hay que crear previamente, para ejecutarlo utilizaremos un ".bat" que si podemos llamar desde Unity. Por ultimo Unity leerá los resultados desde un archivo ".txt" creado por él ".jar".

En resumen el proceso será el siguiente (teniendo en cuenta que primero hay que crear los modelos predictores):

1. Unity crea archivo ".arff"
2. Unity llama archivo ".bat"
3. ".bat" ejecuta ".jar"
4. ".jar" crea ".txt" con resultados
5. Unity lee ".txt"

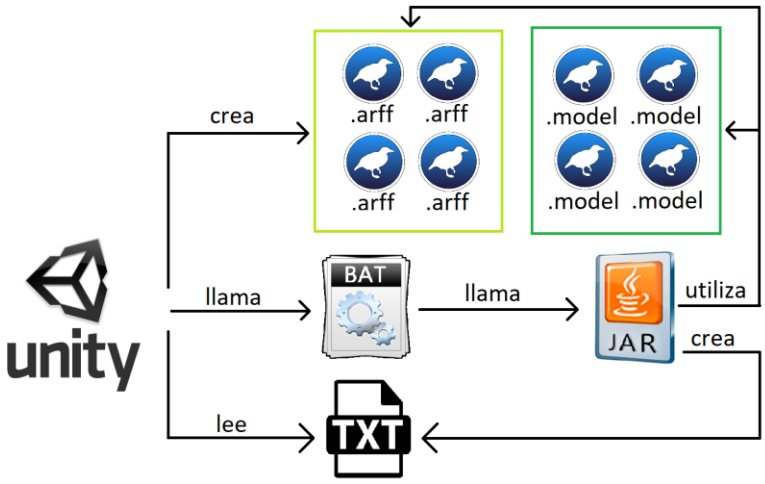


Figura 99: Prototipo 3 - Estructura

Modelo predictor

Una vez que se ha conseguido reunir un número de entradas en la base de datos se procede a comenzar a tratar esta información para poder preparar el modelo predictor. Para ello se tiene que elegir un algoritmo que satisfaga las necesidades teniendo en cuenta la cantidad de datos conseguidos.

Primero se procede a analizar la forma en que esta almacenada la información. Cada entrada dispone de un atributo que indica como se ha clasificado la personalidad del usuario. La cadena de caracteres está compuesta por cuatro indicadores que componen la clase. El resto de atributos de la entrada son valores que recogen el comportamiento analizado.

Teniendo en cuenta las cuatro posibles características extraídas mediante el test Myers-Briggs, se ha decidido analizar cada una de ellas por separado.

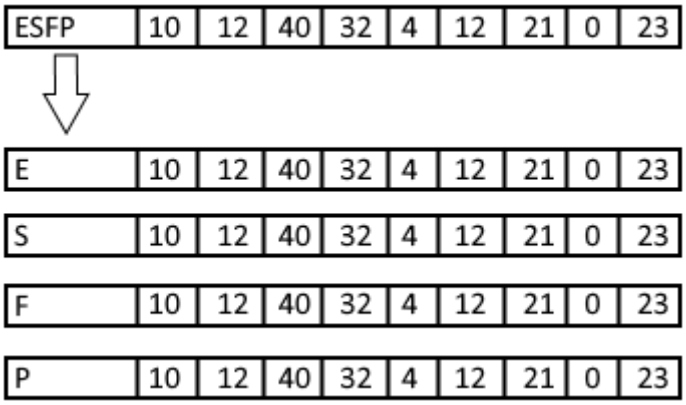


Figura 100: Gráfico Atributos

1. El primero modelo comprobará si el usuario es (E) extrovertido o (I) introvertido.
2. El segundo modelo comprobará si el usuario es (S) sensorial o (N) Intuitivo.
3. El tercer modelo comprobará si el usuario es (T) racional o (F) emocional.
4. EL cuarto modelo comprobará si el usuario es (J) calificador o (P) perceptivo.

La combinación posible de todas las variantes de clase que se pueden generar en el test es de 16, haciendo que un modelo basado en multiclase resulte en un proceso complejo. Optar por dividir en cuatro partes nos aporta varias ventajas.

La primera ventaja es que al dividir en cuatro, aunque los predictores necesarios sean más, son mucho más sencillos al ser de valores binarios.

La segunda ventaja es que un juego no tiene por qué aportar información en los cuatro rasgos, teniéndolos delimitados puede obtenerse un juego que sea una gran predictor en alguno de ellos siendo el resultado especifico superior al global.

También permite analizar los resultados de una forma más clara, pudiendo comprobar mejor lo que está ocurriendo en las tasas de acierto.

Algoritmos predictores

La elección de un algoritmo optimo requiere un numero de instancias de test suficientes, como no se sabe la cifra real necesaria para demostrar la fiabilidad del estudio y las instancias conseguidas han sido pocas se ha continuado con el material disponible.

Los algoritmos planteados han sido NaiveBayes, RandomForest, Knn y J48.

Para cada conjunto de datos se comprueba con cuál de ellos se obtienen mejores resultados.

Al separar los datos en cuatro grupos se decide llamar a cada módulo por el nombre de la característica al que hará referencia; modulo actitud, modulo percepción, modulo decisión y modulo estructura.

En todos los casos se ha dejado la configuración de los módulos por defecto, se realiza el estudio mediantes cross-validation con 5 folds. A continuación se presentan los resultados obtenidos al entrenar con cada algoritmo.

Modulo Actitud

En este conjunto el algoritmo con mejor resultado es Naive-Bayes al no conseguir clasificar correctamente ninguna instancia. Al tener una clase binaria se puede hacer que clasifique lo contrario para que su tasa de acierto se vuelva del 100%.

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances           0               0      %
Incorrectly Classified Instances         5             100      %
Kappa statistic                         -0.9231
Mean absolute error                      1
Root mean squared error                  1
Relative absolute error                176.4706 %
Root relative squared error            174.6668 %
Total Number of Instances                5

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,000    1,000    0,000      0,000    0,000      -1,000   0,000     0,478     E
               0,000    1,000    0,000      0,000    0,000      -1,000   0,000     0,400     I
Weighted Avg.  0,000    1,000    0,000      0,000    0,000      -1,000   0,000     0,447

=== Confusion Matrix ===

 a b   <-- classified as
 0 3 | a = E
 2 0 | b = I
```

Modulo percepción

En este módulo se ha optado por usar J48.

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances           1              20      %
Incorrectly Classified Instances         4              80      %
Kappa statistic                         -0.6667
Mean absolute error                      0.7
Root mean squared error                  0.7906
Relative absolute error                123.5294 %
Root relative squared error            138.0862 %
Total Number of Instances                5

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,000    0,667    0,000      0,000    0,000      -0,667   0,333     0,500     S
               0,333    1,000    0,333      0,333    0,333      -0,667   0,333     0,733     N
Weighted Avg.  0,200    0,867    0,200      0,200    0,200      -0,667   0,333     0,640

=== Confusion Matrix ===

 a b   <-- classified as
 0 2 | a = S
 2 1 | b = N
```

Modulo decisión

En este módulo se ha utilizado el algoritmo de Random Forest.

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances           1               20      %
Incorrectly Classified Instances         4               80      %
Kappa statistic                         -0.6667
Mean absolute error                      0.618
Root mean squared error                  0.6299
Relative absolute error                109.0588 %
Root relative squared error            110.029  %
Total Number of Instances                5

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,333    1,000    0,333      0,333    0,333      -0,667   0,000     0,478     T
                 0,000    0,667    0,000      0,000    0,000      -0,667   0,000     0,325     F
Weighted Avg.    0,200    0,867    0,200      0,200    0,200      -0,667   0,000     0,417

=== Confusion Matrix ===

 a b   <-- classified as
 1 2 | a = T
 2 0 | b = F
```

Modulo estructura

Para el último modelo se ha optado por el algoritmo Naive Bayes.

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances           4               80      %
Incorrectly Classified Instances         1               20      %
Kappa statistic                          0
Mean absolute error                      0.2
Root mean squared error                  0.4472
Relative absolute error                 46.1538 %
Root relative squared error             93.7043 %
Total Number of Instances                5

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,000    0,000    0,000      0,000    0,000      0,000    0,500     0,200     J
                 1,000    1,000    0,800      1,000    0,889      0,000    0,500     0,800     P
Weighted Avg.    0,800    0,800    0,640      0,800    0,711      0,000    0,500     0,680

=== Confusion Matrix ===

 a b   <-- classified as
 0 1 | a = J
 0 4 | b = P
```

Implementación del módulo predictor

Generar archivo de instancias .arff

Dentro de Unity se crea un método que escriba a un archivo .arff los datos del usuario de forma que el programa WEKA pueda leerlo. Como se tienen cuatro módulos predictores serán necesarios cuatro archivos .arff

```
string wekaDefault = "% 1.Title: Resultados Shooter\n%\n% 2.Sources:\n% (a)Creator: Joseba Gallaga &David Espino\n% (c)Date: July, 2017\n%\n@RELATION actitud\n@ATTRIBUTE v1 NUMERIC\n@ATTRIBUTE v2   NUMERIC\n@ATTRIBUTE…
…NUMERIC\n@ATTRIBUTE vN  NUMERIC \n";

string wekaStructure = wekaDefault + "@ATTRIBUTE class {E,I}\n@DATA\n"+data;
string fileName = "actitud.arff";

StreamWriter sr = File.CreateText(fileName);
sr.WriteLine(wekaStructure);
sr.Close();
```

Llamar a un archivo .BAT

Una vez se dispone de los .arff es necesario que una aplicación .Jar los analice. Para iniciar este procedimiento, ya que Unity no puede llamar a un .Jar directamente, se utiliza un archivo .BAT de la siguiente manera:

```
var myProcess = new Process();
myProcess.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
myProcess.StartInfo.CreateNoWindow = true;
myProcess.StartInfo.UseShellExecute = false;
myProcess.StartInfo.FileName = "prueba.bat";
myProcess.Start();
myProcess.WaitForExit();
```

El archivo .BAT contendrá la siguiente sentencia:

```
java -jar shooterModel.jar actitud.arff percepcion.arff decision.arff estructura.arff
```

Clasificación del archivo .arff mediante JAVA(WEKA)

Se diseña un programa sencillo en java utilizando eclipse que recoja los archivos .arff generados y mediante los módulos clasificadores de weka ya preparados generar resultados de clasificación.

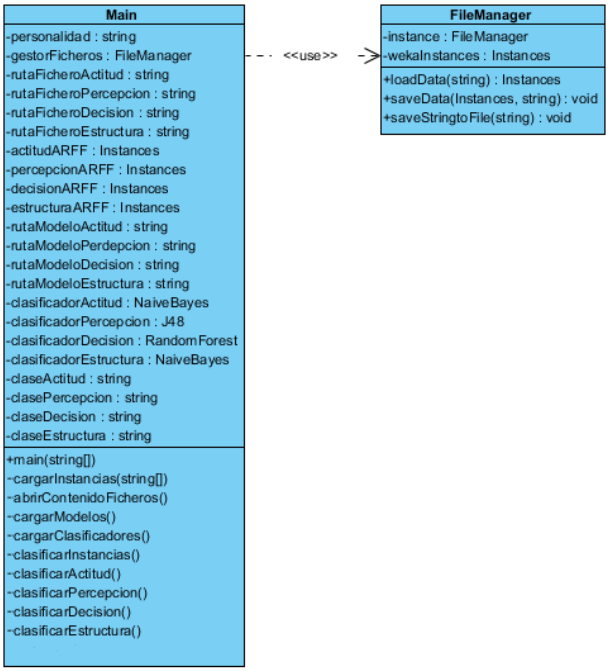


Figura 101: Prototipo 2 - Java - Diagrama de Clases

Mostrar resultados al usuario

Una vez generado el archivo de resultados Unity lo detecta, lo recoge y muestra por pantalla al usuario.

```
StreamReader theReader = new StreamReader("resultado.txt");
evaluationString = theReader.ReadLine();
```

Conexión de Unity con el modelo predictor

Juego Shooter

A la clase que recoge toda la información de la partida se le añade un nuevo método que activándolo el modo evaluación se encarga de ejecutar los modelos predictores para mostrar al usuario su perfil.

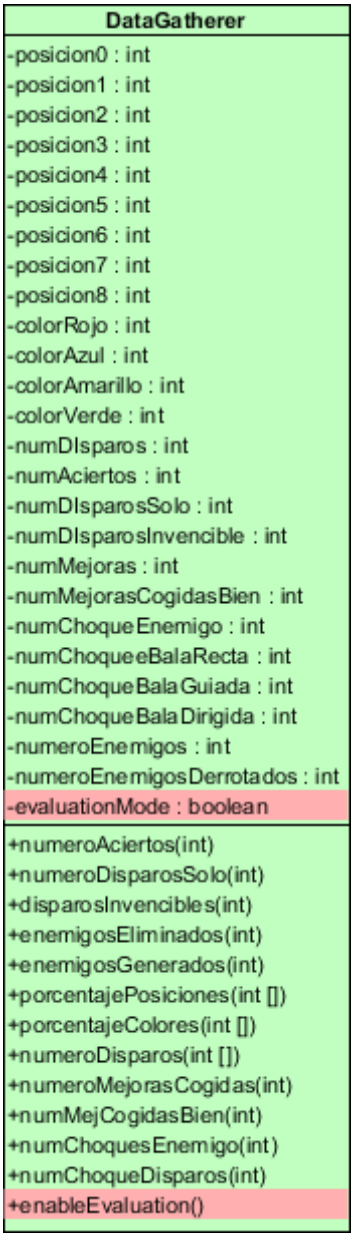


Figura 102: Shooter Prototipo 3 - Diagrama de Clases

Diagrama de secuencia

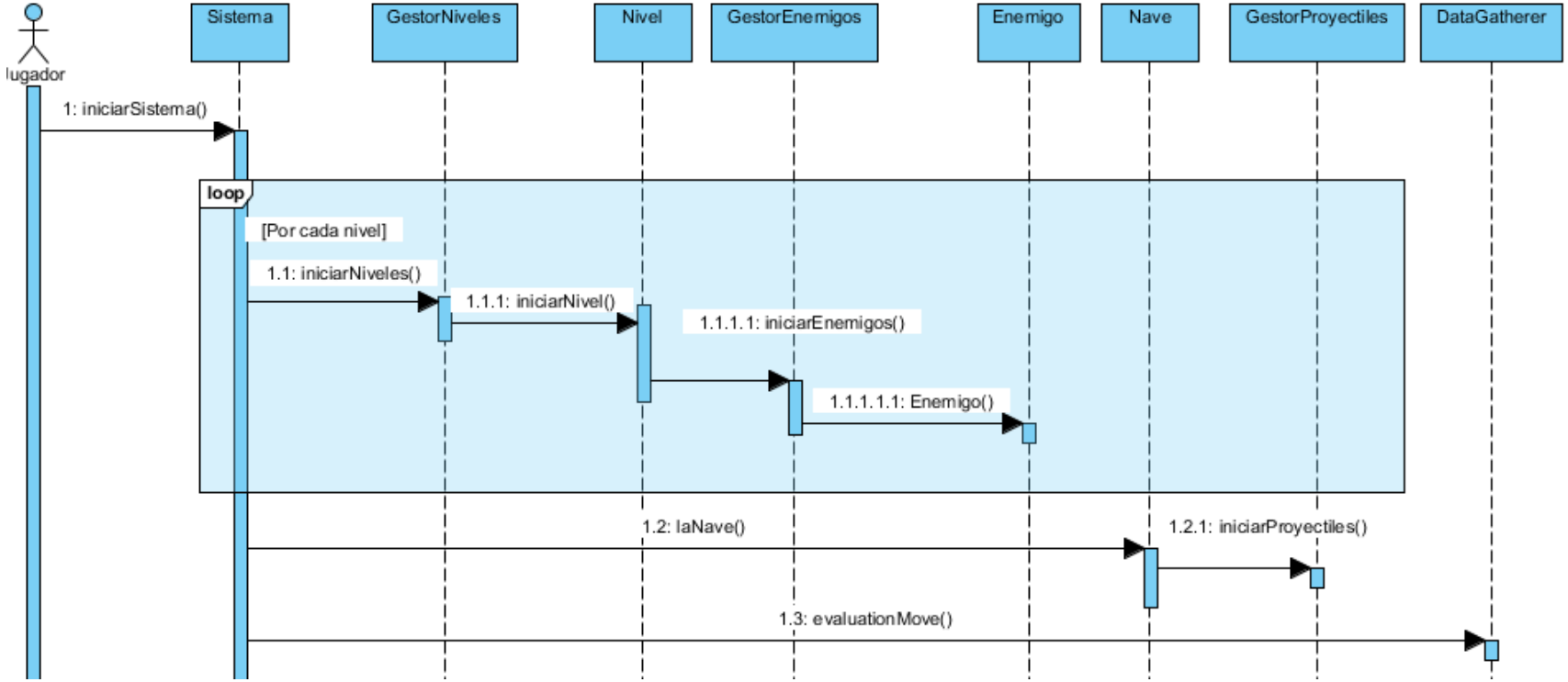


Figura 103: Shooter Prototipo 3 - Diagrama de Secuencia

Juego Arcade

Para que el juego cargue el módulo de minería debemos ejecutar el predictor. Para ello se hace una pequeña modificación en Unity a la clase DataGatherer quedando el diagrama de clases igual que en el prototipo 2 a falta de una variable y dos métodos.

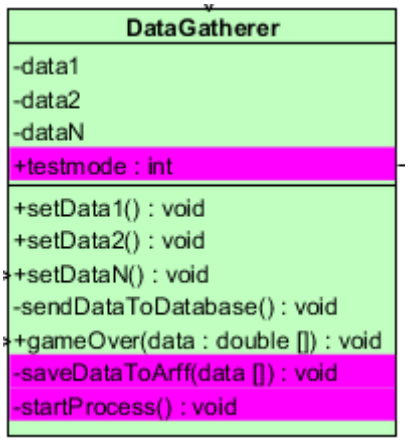


Figura 104: Arcade Prototipo 3 - Diagrama de Clases

Con esta modificación DataGatherer dependiendo en qué modo este(testMode) enviara los datos a la base de datos para investigación o utilizara el módulo de predicción para mostrar el resultado. SaveDataToArff crea el archivo que necesita el .jar para realizar la predicción. StartProcess activa el archivo que ejecuta el jar desde Unity.

El resultado se puede ver desde el menú en la pestaña resultados.

Diagrama de secuencia

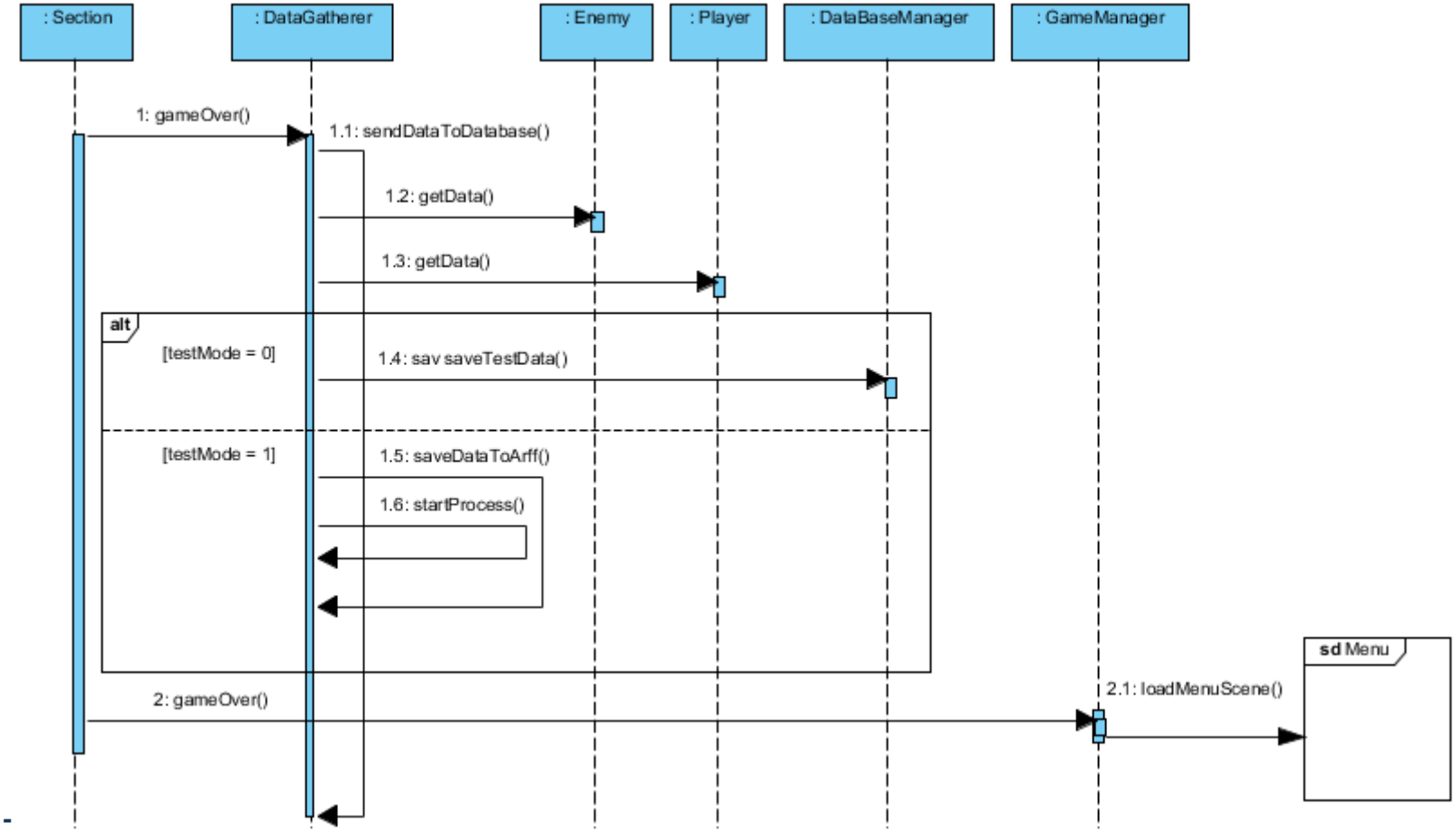


Figura 105: Arcade Prototipo 3 - Diagrama de Secuencia

Pruebas Prototipo 3

En esta última parte hay que comprobar los resultados que dan los modelos predictores. Desgraciadamente dado que el número de instancias para entrenarlos no ha sido muy alto se tendrá que dar por sentado que los resultados no serán los esperados en la mayoría de los casos.

Aun así hay que comprobar que la llamada al método predictor se hace correctamente. Por cada partida se comprueba que los archivos .arff se crean correctamente. También se mira que los modelos son tomados correctamente por el archivo .jar al ejecutar el comando y que tras leer los archivos de instancias genera el archivo TXT con los resultados.

Finalmente se asegura que lee correctamente el archivo y lo muestra por pantalla para el usuario.

Conclusiones Prototipo 3

En este momento el proyecto queda completo y es totalmente funcional en todos sus aspectos. Se han implementado correctamente todas las ideas planteadas para intentar conseguir predecir la personalidad del jugador. A pesar de que los modelos puedan no ser los más adecuados para ésta labor, se ha podido asegurar que aun así todo el sistema realiza todas las operaciones necesarias para que el usuario pueda ver cuál sería su perfil.

En un futuro si se consiguen un número mayor de instancias, únicamente haría falta actualizar los modelos predictores, realizando su correspondiente estudio, y el programa JAR sin tener que modificar nada más del resto del proyecto.

Resultados y conclusiones finales

Lo que en principio parecía un proyecto interesante con ideas que parecían sencillas de realizar, ha terminado siendo un trabajo de una envergadura enorme que ha requerido un gran proceso de aprendizaje e implementación en nuevas herramientas. Ha servido también para expandir el conocimientos en aspectos vistos durante la docencia, pudiendo mezclar distintas asignaturas y crear así una relación entre todas ellas y hacerlo funcionar.

El objetivo principal era intentar ver si existe alguna correlación entre la forma de jugar de una persona y su aspecto psicológico. Se ha hecho jugar a un número de amigos y conocidos a la versión final con la esperanza de conseguir alguna opinión favorable en éste aspecto. En general todos indican que la descripción que les aporta el juego es bastante acertado a lo que ellos creen que es su personalidad.

Al recibir estas reacciones parece ser que aunque el proyecto no sea muy preciso parece acertar en ciertos campos. Desgraciadamente sin saber realmente como es esa persona no se puede confirmar si las predicciones son aproximadas o totalmente aleatorias.

Haber obtenido un mayor número de instancias hubiera sido ideal para poder afirmar con más certeza que podría existir una relación. De ser así, la expansión de este proyecto podría abrir nuevos campos de estudio para centrarse en aspectos más concretos de la personalidad del individuo.

El diseño de cada prototipo permite que cualquier otro compañero interesado pueda retomar este trabajo y expandirlo sin demasiada dificultad. Crear nuevos juegos o conseguir más muestras podría ofrecer nuevos resultados de mayor intereses y poder así confirmar con mayor seguridad que la personalidad de las personas puede ser evaluada.

Bibliografía

Estudio Psicológico

Antonio Martínez Ron (2014). Queremos saber por qué los jugadores de videojuegos de acción son buenos en tantas tareas. Recuperado de http://www.vozpopuli.com/altavoz/next/Neurociencia-Videojuegos-Daphne_Bavelier-Tecnologias-Ciencia-Atenciones-Vision_0_755924454.html

Brigite Micaela Alves Pereira Henriques (2014). Evaluación Psicológica de los jugadores de videojuegos. Recuperado de http://dehesa.unex.es:8080/xmlui/bitstream/handle/10662/1773/TDUEX_2014_HenriquesBM.pdf?sequence=1

Carlos Roberto (2016), ¿Por qué las grandes superficies piden el código postal y los pequeños comercios no?. Recuperado de https://www.pymesyautonomos.com/marketing-y-comercial/por-que-las-grandes-superficies-piden-el-codigo-postal-y-los-pequenos-comercios-no

Cook Briggs, K. and Briggs Myers, I. (1944) The Briggs-Myers Type Indicator Handbook Part I Privately Published

David N. Chin, William R. Wright. Social Media Sources for Personality Profiling. Recuperado de http://ceur-ws.org/Vol-1181/empire2014_paper_09.pdf

Extra Credits (2014), Candy Crush's Success - Why People Can't Get Enough Candy Crush. Recuperado de https://www.youtube.com/watch?v=Sz4WXVNq7v8

Game desing and analytics from achievment data. Recuperado de http://www.neogaf.com/forum/showthread.php?t=1057105

Jamie Madigan (2015), They Psychology of video games. Recuperado de http://www.psychologyofgames.com/2015/04/podcast-2-big-data-and-becoming-a-video-game-psychologist/

John Bohannon (2013), Facebook Preferences Predict Personality Traits. Recuperado de http://www.sciencemag.org/news/2013/03/facebook-preferences-predict-personality-traits

Jung, C.G. (1921/1971) Psychological Types Trans R.F.C. Hull CW6 Princeton

Sigmund Freud (1921). Psicología de las masas y análisis del yo

Twitter. Así es como se ve a la gente en Twitter. Recuperado de http://your-personality-test.com/es/

Valve (2008). Half Life 2, data collected. Recuperado de http://www.steampowered.com/status/ep2/ep2_stats.php

van Lankveld, G. (2013). Quantifying individual player differences Tilburg: TiCC Ph.D.Series 25

Whonix. Data Collection Techniques. Recuperado de https://www.whonix.org/wiki/Data_Collection_Techniques

Herramientas y programación

Anthony Stonehouse (2014). User interface design in video games. Recuperado de https://www.gamasutra.com/blogs/AnthonyStonehouse/20140227/211823/User_interface_design_in_video_games.php

Core-A Gaming (2016) Analysis: What Makes a Move Overpowered?. Recuperado de https://www.youtube.com/watch?v=uQnfm911Xoc

Furrtek (2011) How Metal Slug 3 uses sprites. Recuperado de https://www.youtube.com/watch?v=-2PiaH8CO64

Ilja Astahovs, (2012) Use of design patterns for mobile game development

Incompetech. Recuperado de http://incompetech.com/

Glen Steven, (2013) Unity3D Best Practices

Kevin Lindeman (2016), Server Side Highscores. Recuperado de http://wiki.unity3d.com/index.php?title=Server_Side_Highscores

Mehdi A., Friedhelm B., Antony D., Nuno L., Hannes M., Georg R., Damien S., Jakub V. (2017) Documentación PHP. Recuperado de http://php.net/manual

Unity (2017) Documentación. Recuperado de https://docs.unity3d.com

Pixabay. Imágenes gratuitas de alta calidad. Recuperado de https://pixabay.com/

Pluralsight (2014). Designing a HUD That Works for Your Game. Recuperado de https://www.pluralsight.com/blog/film-games/designing-a-hud-that-works-for-your-game

Soundbible. Recuperado de http://soundbible.com/

Anexos

Diseño de un Juego

Modelado de usuario (User model)

Se trata de crear un software específico que permita el análisis de la interacción entre un usuario y el programa. Para ello hay que conseguir que el jugador obtenga la información necesaria para saber la situación en la que se encuentra en cualquier momento y únicamente cuando sea requerida.

Hay que mostrarle el efecto que produce cualquier acción que realiza para que comprenda el funcionamiento y reglas de la aplicación. Si se producen cambios en el entorno del juego estos deberán estar claramente mostrados para evitar que se produzca alguna confusión.

Por ejemplo en el juego de naves, para cambiar a un color distinto se pulsa uno de los cuatro botones correspondientes. La disposición de los botones, que permiten cambiar el color del jugador, están situados en la misma formación que el que se muestra en el interfaz de la pantalla. De esta forma sin tener que apartar la vista del monitor, el jugador puede cambiar fácilmente entre los colores disponibles al haber una relación entre lo que ve y lo que puede hacer.

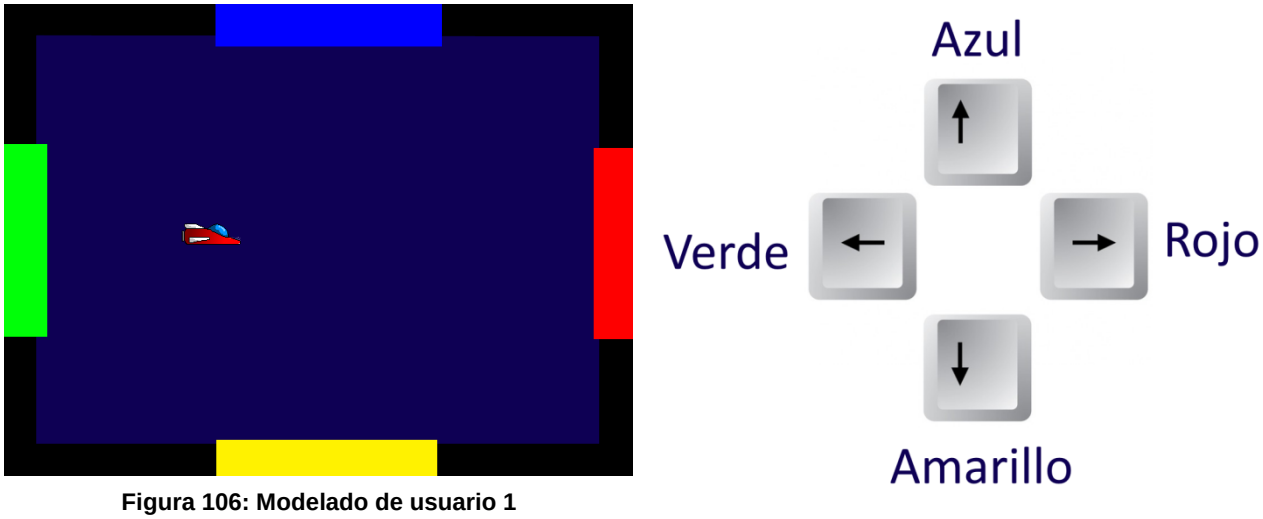


Figura 106: Modelado de usuario 1

Para que el jugador comprenda que sus acciones son correctas y que se busca que las realice con más asiduidad, se le recompensa con una puntuación que se incrementa según realiza las distintas acciones disponibles según la aplicación: al eliminar un enemigo utilizando proyectiles su puntuación mejora, pero al chocarse contra él no se incrementa y además ve a su personaje penalizado perdiendo potenciadores. En vez de puntuación se puede recompensar de distintas formas, con efectos visuales y sonoros, con objetos útiles para el personaje, etc.

Disposición (Affordance)

Un término que además de ser utilizado en distintos campos de la psicología, es un aspecto importante en el diseño de interacción en un juego. Define el comportamiento de un producto con el usuario. El objetivo es buscar que de forma natural un jugador sea capaz de saber cómo jugar sin la necesidad de que un elemento externo tenga que explicárselo.

El caso más utilizado como ejemplo para el buen diseño de un juego es el primer nivel de Super Mario Bros, un juego de Nintendo publicado en 1983. La idea básica es presentar al usuario una situación inicial controlada donde el error del jugador no tenga demasiado efecto en la partida. Cuando el jugador supera estos retos iniciales, se le presenta una nueva situación más complicada donde aplicar los conocimientos aprendidos anteriormente generan una recompensa.

Este aprendizaje es un aspecto especialmente importante en el proyecto. Para poder recoger información útil se debe evitar en las que las acciones del usuario son aleatorias por no saber qué hacer. A la hora de preparar los niveles a los que se enfrentara el jugador hay que dejar claro cuál es la situación en la que se encuentra y así recoger cómo se comporta para solucionarlo. La solución adoptada en el proyecto para que el aprendizaje sea rápido y sencillo es la implementación de un tutorial antes de comenzar a realizar el juego en modo test.

Modelado de enemigos

Durante la creación de un juego hay que decidir si el usuario se enfrentará a una serie de enemigos.

Éstos presentan distintas acciones y comportamiento que se definen mediante una inteligencia artificial programada. Dependiendo del estilo puede que no aparezca ninguno siendo un juego más individual donde los retos a afrontar son más pausados. No hay que cometer la equivocación de pensar que porque en pantalla no aparezca una criatura o máquina que amenace al jugador no exista un enemigo. En un videojuego el adversario puede aparecer en forma de estructura, esta puede requerir desplazarse por ella de una forma concreta para evitar trampas o para abrir el camino que permita progresar.

Durante el desarrollo de estos restos hay que decidir el comportamiento tanto de los elementos que participan como el entorno donde ocurre. Utilizando todos estos elementos en una disposición diferente produce resultados diferentes. Para una evaluación correcta, tenemos que procurar que los enemigos siempre actúen de una forma determinada sin importar las acciones del jugador haciendo que sus patrones de comportamiento sean previsibles.

Es importante aclarar que aunque la IA de estos enemigos sea concreta, las condiciones para enfrentarse a ellos no tienen por qué ser iguales en cada caso. La experiencia al jugar al juego tiene que ser positiva y ello requiere que el jugador reciba motivación para enfrentarse a nuevos retos que le hagan participar en buscar una solución al problema. En esta situación la opción más adecuada es no cambiar las reglas del juego pero si las condiciones en las que se presenta el desafío. Por ejemplo

un enemigo se puede comportar igual que las anteriores veces pero puede que para poder eliminarlo haya que realizar cierto patrón de acciones de forma repetida o inversa.

Es una situación ideal para poder obtener información que permita saber el comportamiento del jugador dependiendo de la situación y así poder perfilar los datos que se obtienen.

Modelado psicológico

De forma resumida se puede describir un videojuego mediante su “género”, es decir su estilo de juego. Al existir una forma sencilla de categorizar distintos videojuegos, un individuo puede verse decantado por jugar juegos de una temática similar, es fácil buscar juegos concretos mediante filtros. La repetida interacción con estilos de juegos similares permite al jugador superar de una forma más fácil futuros retos en un juego diferente pero que comparte mecánicas.

Estos reflejos y factores relacionados con la habilidad de la persona pueden acarrear un problema a la hora de recoger información sobre su comportamiento. Cuanto más se perfila su forma de jugar más similitud mostrará con una segunda persona que también se especialice en dicho género.

Esta situación produce una condición, donde sin importar la persona, la acción a realizar es la misma no pudiendo conseguir información dispar. No es posible evitar en un proyecto que una persona no tenga conocimientos previos en el género en el que se desarrolla la aplicación, es por ello que se evitará que sea posible jugar varias veces las partes que aportan información.

El jugador podrá jugar al test una única vez. Presentar un reto sin dar explicaciones tampoco es lo óptimo para extraer información, por tanto dispondrá de una primera partida controlada donde se explica el funcionamiento de la aplicación.

Desarrollo de un juego

Para crear el juego tenemos que tener en cuenta como se debe comportar el algoritmo. En general todos los videojuegos comerciales realizan los mismos pasos para poder mostrarse en pantalla.

De forma resumida se puede describir este comportamiento en tres pasos que son tres secuencias de código que se ejecutan igualmente en todos los juegos. Esta serie de eventos se denomina Ciclo de Juego o Game Loop.

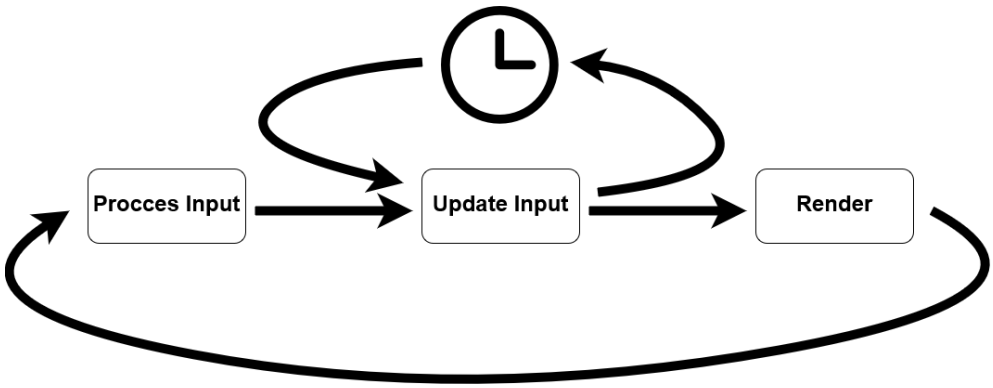


Figura 107: Ciclo de Juego

Estas secuencias se ejecutan de forma continua durante toda la partida hasta terminar. La tarea del programador es añadir a cada secuencia el código y elementos que crearán el juego.

Primera secuencia

La primera secuencia es la de carga que se ejecuta al comenzar la aplicación, en este momento el programa tiene que recoger y preparar todos los elementos multimedia que se utilizarán en el nivel del juego para que estén disponibles desde el principio. Es común en juegos y aplicaciones de gran envergadura mostrar una pantalla de carga antes de ver cualquier otra cosa.

Figura 108: Pantalla de Carga

Esta fase de carga puede extenderse por mucho tiempo, por lo que el programador puede decidir si quiere cargar por completo el juego o, con objetivo de agilizar el acceso a la aplicación, distribuirlo. En éste segundo, únicamente aquellos elementos que serán usados durante un nivel concreto serán cargados y al pasar de nivel volveremos a una pantalla de carga para el siguiente.

Nuestros juegos al ser sencillos con una única carga del juego completo será suficiente. En esta primera ejecución es donde también se realiza la inicialización e instanciación de todas las variables y objetos que aparecerán en el juego.

Segunda secuencia

La segunda fase se encarga de la recogida de las acciones del jugador y del cálculo de todo lo que está ocurriendo en el juego. En un primer vistazo se puede pensar que en este apartado únicamente se calcula la posición de los objetos en pantalla así como la puntuación, las vidas, en qué nivel se encuentra y otros aspectos que normalmente se muestran por pantalla.

En realidad en segundo plano ocurre el proceso más importante y que ocupa más empeño y poder de procesamiento en un videojuego, el cálculo de colisiones.

Colisión es el término que se utiliza para describir como los elementos del juego interactúan entre ellos, son una representación o varias no visibles que están en el contorno de cada elemento. Su forma geométrica puede ser variada por lo que para generalizarlos se usa el término de *hitbox*.

Inicialmente esta palabra da a entender que se refiere a que dos objetos tienen que tocarse físicamente para que algo ocurra. No es ese el caso.

Existen distintos tipo de *hitbox* que pueden cumplir distintas funcionalidades, cada una de ellas pueden tener su propio termino pero los más utilizados son el propio *Hitbox* y *HurtBox*. Además, estas regiones no tienen por qué cubrir por completo al objeto que lo representa, pueden incluso ser mucho mayores que él.

Hitbox es el entorno en el que un elemento produce un efecto sobre otro. El ejemplo más sencillo es el de un proyectil cuyo objetivo es la de eliminar un enemigo, la región de ese proyectil debe definirse para indicar que parte es la que realmente debería hacer daño, como la punta de una flecha.

El enemigo a su vez dispone de un *Hurtbox*, donde un elemento recibe el efecto de otro. Para recibir ese daño el *Hitbox* del primer elemento debe entrar en el entorno del *Hurtbox* del segundo. Es decir, existen distintos dominios de colisiones que dependiendo de que otros tipos sean y accedan a ella puede ocurrir una situación u otra.

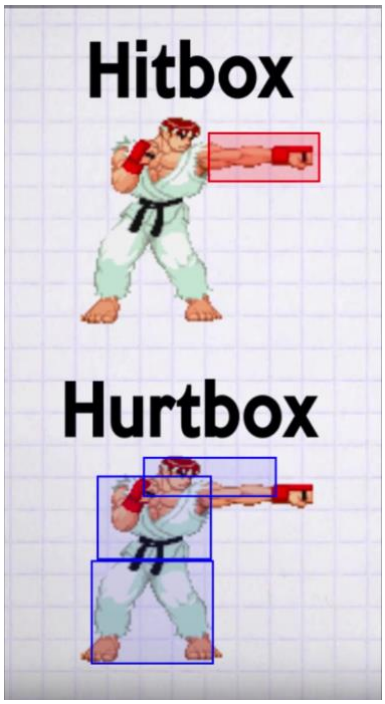


Figura 109: Hitbox - Hurtbox

Como ejemplo también existe la colisión que permite “ver” a un elemento. Si un jugador entra dentro de ese campo de un enemigo. Éste le podrá ver y saber su posición. En este caso, al región que representa la vista del enemigo será mucho mayor, y puede que tenga forma cónica para que simule el campo de visión. Otro tipo de colisiones pueden usarse para impedir el paso, como el suelo o paredes.

Para que todo interactúe y ocurra como el jugador espera, el juego debe analizar cada uno de estos *hitbox* y sus variantes de forma individual sobre el resto de elementos durante un preciso momento en la ejecución de la aplicación. Cuantos más elementos, mas calculo. Si existen demasiados a la vez y el equipo en el que se ejecuta la aplicación no dispone de suficiente capacidad de procesamiento puede que ocurran ralentizaciones produciendo parones o una ejecución más lenta.

Tercera secuencia

Ésta es más sencilla de explicar, ya que su cometido es la de mostrar las imágenes y reproducir los sonidos del momento concreto en el que se encuentra el juego. Aun así no le exime de complicación. La imagen que genera en juego puede asimilarse a la técnica que se utiliza en dibujos animados para representar la escena, donde los distintos elementos se muestran en capas de profundidad, más conocidos con el término de *layers.*

Cada capa se coloca encima o detrás de otro por lo que el orden de estos es importante para mostrar la imagen correcta que se mostrará en pantalla. En cada capa se elige que elementos debe contener. Por ejemplo, los fondos del niveles se encontrarán en las capas más alejadas, que además si esto se mueven y concretamente a distintas velocidades se crea la técnica denominada *Scroll Parallax*.

Scroll parallax crea una sensación de profundidad en el nivel dando un aspecto más realista a los acontecimientos que ocurren durante un juego 2D.

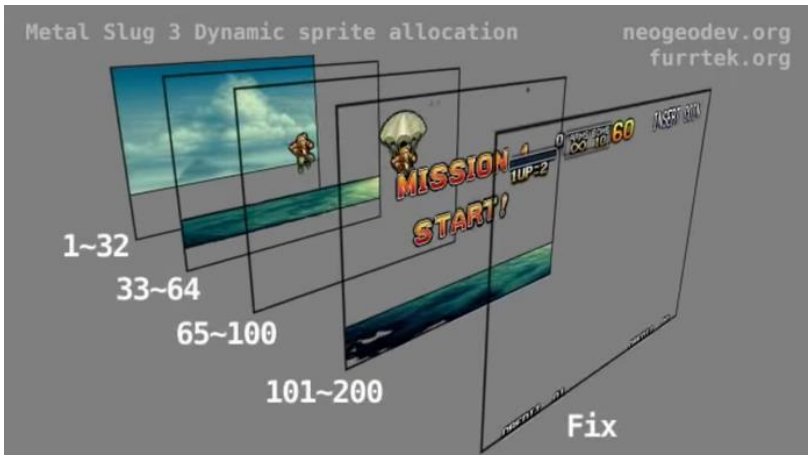


Figura 110: Scroll Parallax

La puntuación e información que se muestra en todo momento se encuentran en las capas más superiores para que estén siempre accesibles y a la vista del jugador.

En las capas intermedias se encuentran aquellos elementos que están en constante movimiento como el propio jugador, los enemigos y los elementos peligrosos u obstáculos.

Gráficos

Cualquier programa con una interfaz gráfica requiere un diseño de pantallas, en el caso de un juego esta necesidad es aún mayor. Es necesario crear personajes, enemigos, vehículos y gran cantidad de pequeños detalles que den vida a lo que se nos muestra mediante el monitor, además necesitamos colocar todos estos elementos en un lienzo o escenario. Esto sin un método sería tedioso y largo, por ello se ha estudiado una forma de dibujar de una forma rápida y eficaz.

Personajes

En el caso de todos los personajes, enemigos o elementos se ha partido cuando ha sido posible de una fotografía y mediante *Photoshop* se han transformado en dibujos. Para conseguir un efecto *cartoon* cada color o parte de un personaje ha pasado de tener degradados a tonalidades de colores, es decir, ha pasado de como se ve el mundo real, a tres tonos.

Se buscan referencias, para mostrarlo se utiliza un ejemplo con el personaje principal sutilmente inspirado en Indiana Jones. La primera referencia permite obtener la forma, ya que el personaje va sentado en una carreta es una buena foto lateral, será necesario girarla para que el personaje se encuentre erguido. La segunda foto al ser una tomada con luz natural permite obtener los colores base que deseamos aplicar. Con estas dos referencias se abre *Photoshop*, y se copia cada una de ellas en una nueva capa.

Figura 111: Diseño Gráficos 1

Hay que adaptar la primera referencia a la posición que se quiere que termine el dibujo. Una vez todo listo hay que fijarse en cada parte de las que consta. En este caso sombrero (cuero y cinta), cazadora, camiseta y cabeza (cara, pelo, ojos). Por cada uno de los elementos se conseguirán tres colores diferentes. Se borra toda parte del fondo restante, esto hace fácil más tarde seleccionar el contorno, también se oculta la capa de referencia para los colores.

Figura 112: Diseño Gráficos 2

Con el lazo magnético se selecciona la primera zona, por ejemplo el rostro, incluido cuello, orejas, etc. Una vez se tiene la selección es rellenada con el tono más oscuro para la piel. Para poder tomar referencia más fácilmente, volvemos a mostrar la segunda imagen(la tomada con luz natural) y con la herramienta cuentagotas se selecciona una parte oscura de la piel.

Se duplica la capa, así se consigue una capa oscura y media(se le cambia los nombres para no confundirse). Para cambiar el color se accede a ajustes, brillo. Y se aumenta en 50. Se coloca la capa media encima de la oscura. Se pone en primer plano la imagen de referencia y fijándose en las zonas de color oscuras, simplemente con la herramienta goma, aunque no se vea el resultado, se pasa por encima de estas zonas oscuras, al ocultar la referencia se ve el resultado.

Se duplica la capa media, se sube el brillo de la superior en 50 y se consigue el tercer tono, color claro. Se repite el proceso de colocar la capa de referencia y borrar en las zonas más claras obteniendo el resultado final del rostro.

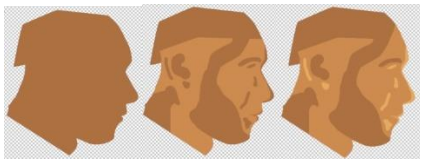


Figura 113: Diseño Gráficos 3

Por cada sección de la imagen, es decir pelo, chaqueta, gorro y camisa, se repite el proceso anteriormente descrito obteniendo los siguientes resultados.

Figura 114: Diseño Gráficos 4

Una vez se ha dibujado todo, el resultado final aunque sencillo, es práctico y atrayente

Figura 115: Diseño Gráficos 5

Escenarios

Cada sección de escenario hay que dibujarla, si se dibujase cada trozo del escenario poco a poco las horas necesarias serian excesivas, para ello se ha buscado un método que permita generar escenarios de una forma rápida y eficaz.

Se genera una sección estándar de la que partir. Es necesaria una plantilla que simule el suelo, como el escenario va a ir rotando requiere que la parte izquierda encaje perfectamente con la derecha para cuando se añada una nueva sección no se note la transición.

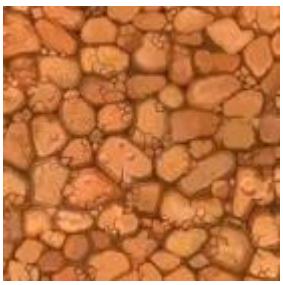


Figura 116: Diseño Gráficos 6

En el juego existe tanto suelo como techo, así que se hace que la trama cubra absolutamente toda la pantalla repitiéndola de forma que el tamaño del dibujo se ajuste a nuestras necesidades.

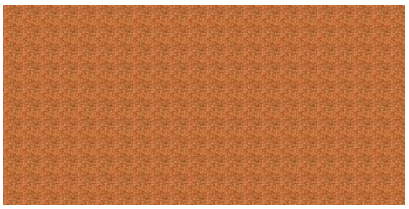


Figura 117: Diseño Gráficos 7

Dejando el terreno de lado, son necesarias las vías por donde corre la vagoneta, por ello usando el método de los personajes se dibujarán tanto las traviesas de madera como el rail.

Figura 118: Diseño Gráficos 8

Con todos los elementos necesarios, trama, maderos y rail se procede a dibujar escenarios. Como los escenarios empiezan igual que el anterior ha terminado, con la regla de *Photoshop* se crean guías. Cada vez que se genera un escenario se comprueba al nivel de que guía termina el anterior.

Se hace una selección y se borra toda la zona que se quiere que quede vacía. Para borrar es útil el lazo magnético ya que seguirá la trama de las piedras y conseguirá un resultado más realista.

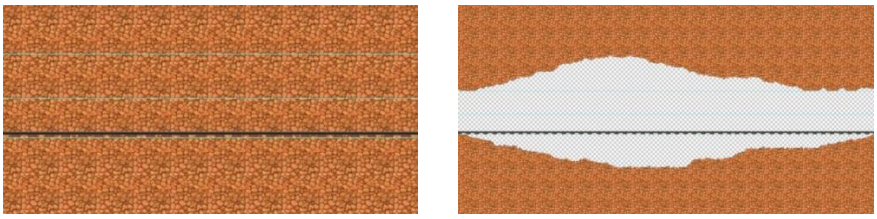


Figura 119: Diseño Gráficos 9

Se re la vía adaptándola al escenario, para ello debemos crear una selección con la forma del trazado. Utilizando la herramienta pluma, una vez se tiene el recorrido seleccionado en una nueva capa se pulsa el botón derecho del ratón y contornear trazado.

Ahora es necesario duplicar tres veces la capa para dejar la vía nuevamente como el resto de elementos a tres colores.

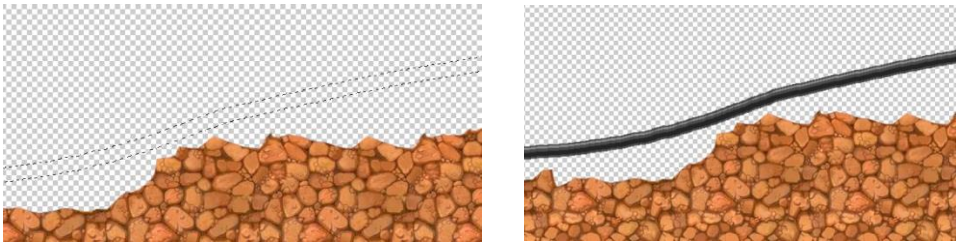


Figura 120: Diseño Gráficos 10

Mediante la herramienta transformación libre cada uno de los maderos se colocan debajo de la vía de forma que encaje lo mejor posible. No hace falta poner todos, ni que se encuentren a la misma distancia, solo de forma que el resultado sea satisfactorio.

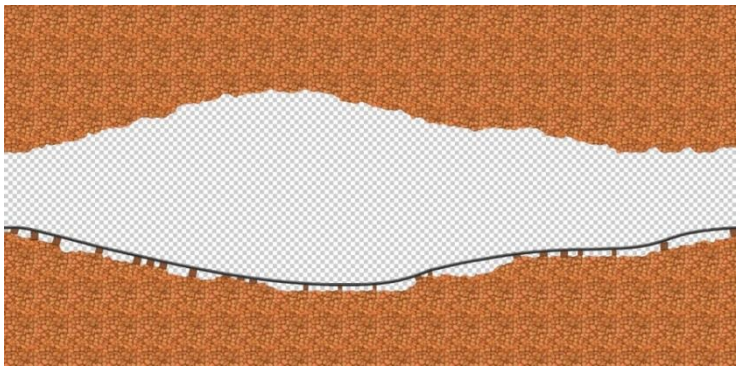


Figura 121: Diseño Gráficos 11

Puntos destacados de Unity

Elección de Unity

En el momento de plantear el proyecto la plataforma de Unity se encuentra en apogeo. La elección de este motor requiere contrastar varios pros y contras por lo que hay que ---comprobar si es adecuado para los juegos que se pretenden desarrollar. Para ello se consultan páginas y opiniones tanto online como de conocidos.

Ventajas

- Multiplataforma, desarrollo para Iphone, Android y Windows Phone aparte de PC e IOS. En el proyecto en primera instancia se desarrollara para PC, pero la idea de poder importarlo para Tablet o Móvil como experimento personal es atrayente.
- Contiene módulos con métodos concretos para 2D. Aunque como se constatará en desventajas Unity, está desarrollado para juegos en 3D, existen muchos métodos y físicas para 2D.
- Gran parte de Unity está disponible de forma gratuita para cualquier usuario, en este proyecto no ha sido necesario pagar ninguna librería o *asset*. Las herramientas del Unity básico permiten desarrollar un juego en su totalidad.
- *Asset store*, una gran biblioteca de métodos, tanto gratuitos como de pago creados por los propios usuarios. Estos nos permiten mejorar el juego integrando código de otras personas, por ejemplo existen módulos de explosiones, generadores de terreno o gestores de sonido.
- Relativamente fácil de usar, aunque las primeras horas son difíciles, rápidamente se aprende de forma intuitiva como seguir progresando, el aprendizaje es exponencial.
- Los scripts pueden ser escritos tanto en *Javascript* como en *C#*, es versátil. A la hora de programar permite utilizar el lenguaje mejor controlado por el desarrollador.
- Arrastrar y soltar para referenciar. No es necesario buscar en ejecución, crear referencias a objetos o asignar variables. Desde la interfaz gráfica se puede configurar el proyecto antes de empezar a programar permitiendo partir de una base estable.

Inconvenientes

- No existen plantillas, el juego debe hacerse desde 0. En otras plataformas existen pequeños tutoriales que crean una base de la que partir, introduciendo el tipo de juego que se quiere desarrollar se crea un escenario básico el cual ir alterando. En Unity hay que introducir desde el primer elemento hasta el último. La única excepción son los *asset* prefabricados que pueden suplir esta carencia.
- Funciones de pago para obtener mejoras de rendimiento y acceso a más plataformas. Con pagos no excesivamente caros se obtiene acceso a una mayor gama de opciones, en este caso no necesarias
- Principalmente centrado para 3D, aunque con modulo para 2D. Los métodos 2D realmente son adaptaciones del 3D, muchos de ellos son métodos 3D

simplemente vistos desde una perspectiva lateral para dar la sensación 2D. Esto hace que algo supuestamente más sencillo al tener una dimensión menos, sea lo mismo que programar en 3D.

- En desarrollo se consume muchos recursos, los proyectos ocupan mucho espacio haciendo que generar copias de seguridad sea tedioso. También en desarrollo el requerimiento de hardware es mucho mayor que el del resultado final.
- Versiones, Unity es actualizado cada poco tiempo haciendo que de una versión a otra se deban cambiar partes del proyecto y actualizar métodos. De una actualización a otra se crean errores y se debe repasar la totalidad del proyecto.

Una vez comparadas las ventajas e inconvenientes las ventajas resultan muy útiles, y los inconvenientes apenas tienen efecto. Para comprobar la viabilidad de la herramienta se hacen pruebas y tutoriales a fin de aprender el funcionamiento básico. El uso de Unity requiere aprender sistemas de desarrollo muy diferentes a lo acostumbrado, haciendo que los diagramas de clase pensados, aunque útiles, necesiten ser adaptados.

Implementación en Unity

En esta sección se explicarán partes de Unity aprendidas durante el proyecto que han sido útiles. Son muchas cosas más de las nombradas las que se han aprendido pero se han incluido unas pocas por su relevante interés.

Orientación a componentes (Inspector)

Uno de las primeras características aprendidas de Unity es que su programación orientada a objetos recibe el nombre particular de “Programación Orientado a Componentes”. Su característica radica en que a pesar de que la implementación del diseño es igual al aprendido en otros lenguajes, en este se puede trocear esa implementación en archivos diferentes y añadirlos como partes de un objeto. De esta forma no solo el algoritmo puede ser reutilizado por otros objetos, también en otros elementos de uso común dentro de Unity.

Un ejemplo sencillo es un posible objeto enemigo. Tiene como implementación, una primera parte dedicada a su movimiento por la pantalla y control de colisión contra el usuario y su segunda parte que es un pequeño código que le permite decidir qué tipo de proyectil quiere utilizar a la hora de disparar. En Unity es posible separar estos dos algoritmos en dos archivos distintos y después añadírselos a una instancia del enemigo.

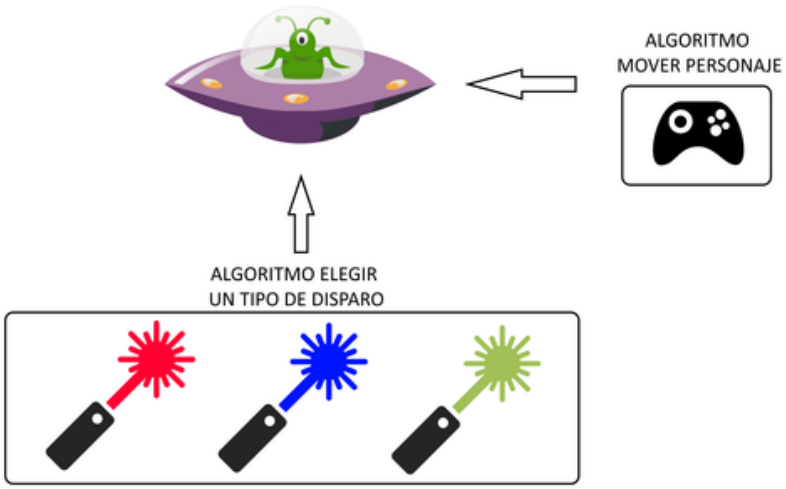


Figura 122: Orientación a componentes

De la misma manera es posible hacer que el proyectil elegido se agregue un componente de movimiento que puede ser tomado al azar siendo uno de ellos incluso el propio algoritmo de mover el personaje.

La compatibilidad de estos algoritmos de movimiento radica en que Unity incorpora a todos los objetos de unos componentes estándar dedicados a definir su posición en pantalla, apariencia y caja de colisión... entre otros. De esta forma que el acceso a esos datos son iguales para todos los elementos.

Componente posición (Transform)

Para que un objeto pueda acceder a estos valores debe referirse a si mismo (gameObjetct) e indicar a que componente (Transform) quiere acceder. Por ejemplo si se quiere saber la posición de un elemento:

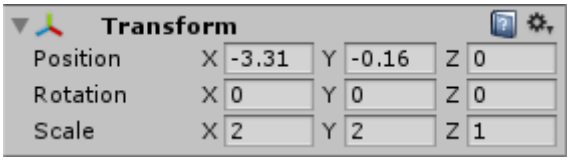


Figura 123: Unity - Transform

```
Vector3 posicion = gameObject.GetComponent<Transform>;
```

El dato que recogeremos será de tipo Vector3, ya que obtendremos los valores de X, Y y Z. Después si hace falta podemos acceder individualmente a cada campo.

Por suerte para el programador, Unity ya ofrece variables reservadas que permiten acceder a estos datos de forma mucho mas rápida ya que el acceso a ellos es muy común.

```
Vector3 posicion = transform.position;
```

Esta facilidad para acceder a campos cotidianos de los objetos que se utilizan en Unity permite un mayor libertad para separar distintas tareas de programación que pueden adjudicar a distintos miembros del equipo sin necesidad de depender de otros.

Todos estos componentes aparecen listado en cada objeto bajo el termino de Inspector. Unity permite acceder a todos los datos de forma visual, pudiendo modificar sus valores y componentes incluso durante la ejecución del juego.

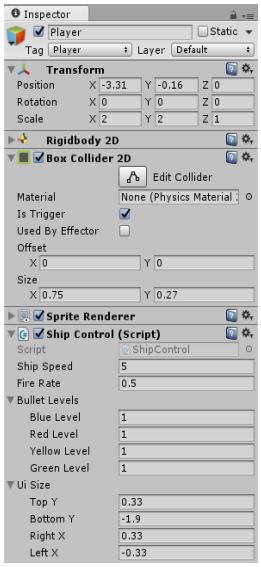


Figura 124: Unity - Inspector

El Script, que es el algoritmo, contendrá las variables que se usarán durante la ejecución. Si se hacen públicas estas variables o se les indica la propiedad de ser serializables, Unity permite acceder a ellas desde el propio programa, pudiendo así hacer las modificaciones necesarias. Durante las pruebas en las que el juego esta ejecución permite cambiar los valores de estos campos para poder hacer comprobaciones o correcciones de forma directa reflejándose en el juego. Estas correcciones no son permanentes, sirven como guía para llegar al resultado deseado.

```
[SerializeField]
private float shipSpeed;

[SerializeField]
private float fireRate;
```

Jerarquía de objetos

Un elemento instanciado debe tener una variable que le haga referencia para no perderlo en memoria. En Unity existe un panel de jerarquía. Ésta herramienta se encarga de hacer referencia a todos los objetos que están instanciados en el programa, ya no es necesario usar una variable para acceder a ella, la aplicación sabrá que existe hasta que finalice el programa o se elimine el objeto.

Esta jerarquía permite eliminar *arrays* donde almacenar sus referencias en objetos que son capaces de funcionar individualmente. Por ejemplo un proyectil que su función es únicamente desplazarse y si llega AL borde del nivel, eliminarse. Aun así no significa que si fuera necesario acceder a ese objeto por algún razón no sea posible hacerle referencia. Unity permite realizar esto encontrando un elemento mediante la búsqueda de su nombre. Si existieran más de uno con el mismo nombre, se recibiría una lista de esos elementos. Todos los elementos en uso aparecerán en esta lista se tenga su referencia en una variable o no.

Figura 125: Unity - Inspector

La jerarquía también permite la creación de hijos dentro de un elemento. Estos hijos son objetos que heredan de sus padres una serie de propiedades que les permiten tener la ventaja de poder acceder a los valores de cada uno recíprocamente de una manera muy sencilla. No solo eso, un hijo ahora tendrá como referencia a su padre. Una de sus funciones es por ejemplo si un objeto padre debe instanciar un hijo, éste segundo tendrá como posición inicial en pantalla la posición actual de su padre.

Figura 126: Unity- Posición en Pantalla

En este ejemplo una nave está en la posición (200,100) y se quiere instanciar un proyectil justo donde se encuentra. A la hora de indicar cual será la posición inicial del proyectil, es posible decirle que la posición inicial será la del padre. El objeto instanciado sabe que tiene que aparecer en pantalla en la posición (200,100), pero al ser hijo, tomará esta referencia como el punto (0,0) pues ese es el origen, su padre, desde el que saldrá.

Event Functions

Cada objeto en Unity por el hecho de heredar de *Monobehaviour* dispone de unas funciones básicas. Estas funciones o eventos se disparan en un orden establecido, los más usados en el proyecto son:

Awake: Este método es llamado cuando se activa un objeto, justo antes de *start* o cuando un *prefab* es instanciado.

Start: Es llamado antes del primer *frame* cuando un objeto ha sido creado. En el proyecto se ha podido utilizar para inicializar objetos.

Update: Es llamado una vez por *frame*. Sirve tanto para recalcular datos como pueden ser las distancias, como para generar movimientos de *sprites*, como es el desplazamiento de un enemigo por el mapa. *Update* es utilizado por todos los objetos que necesiten mostrar movimiento o una comprobación en todo momento de la ejecución.

Físicas

Unity tiene una gran cantidad de físicas pre programadas. Hay juegos que no necesitan físicas como es el caso del juego de naves, en cambio el juego de la carreta ha utilizado una gran cantidad. Para utilizarlas simplemente hay que añadir al objeto el componente necesario ya existente en Unity.

En el caso de la carreta, al tener ruedas hemos utilizado una física de ruedas que consigue que las ruedas giren (WheelJoint2D), pero a su vez para que al girar las ruedas la carreta se mueva, ha sido necesario añadirle rozamiento a la superficie de ellas. También existen físicas como son la gravedad, en el caso de la carreta para evitar posibles vuelcos y que superar un nivel sea imposible, el centro de gravedad ha tenido que ser colocado no en el centro del *sprite*, sino por debajo de las ruedas.

Todos estos métodos ahorran horas de trabajo y cientos de líneas de código que harían la implementación de todo el proyecto casi imposible.

Detección de colisiones

La detección de colisiones es una de las tareas más arduas a la hora de programar un videojuego. Es necesario no dejar la comprobación de ningún objeto si se quiere que todo funcione como es debido. Esto supone parte del código analizando uno a uno cada elemento instanciado contra otros posibles objetos con los que podrían estar colisionando.

Unity ofrece hacer esta comprobación de forma independiente al código. Añadiendo un componente tipo *collider* permite hacer esto de forma automática. Con sencillas llamadas a métodos relacionados con las colisiones en el algoritmo ya se sabrá como tratar estas situaciones. Existen distintos tipos, pero los más comunes son aquellos que permiten comprobar si algún elemento ha entrado en el campo de colisión del objeto, si permanece dentro y si ya ha dejado ese campo, permitiendo así concretar en qué momento se quiere que ocurra cada acción.

Es una característica muy importante que agiliza la implementación del proyecto, todo el diseño relacionado con la comprobación de colisiones entre elementos puede obviarse casi por completo, dejando únicamente las acciones a realizar dependiendo de cada situación.

Prefabs

Una vez creado un objeto con toda la configuración deseada y con los componentes y variables predefinidos, es posible si así lo deseamos poder guardar esa instancia como un único archivo. Éstos se denominan prefabs. Todos los archivos que necesite ese objeto se almacenarán con él.

Si existe un objeto que vaya a ser utilizado varias veces durante el proyecto es útil disponer de este archivo para poder instanciar de manera muy sencilla tantos elementos como se requieran simplemente llamandolo. Al estar almacenado físicamente puede ser exportado para ser utilizado en otros proyectos totalmente diferentes.

Unity da la oportunidad de poder adquirir este tipo de archivos en su propia tienda. El precio puede variar y puede conseguirse también de forma gratuita. Algunos de los elementos usados en nuestro proyecto están tomados de esta tienda.

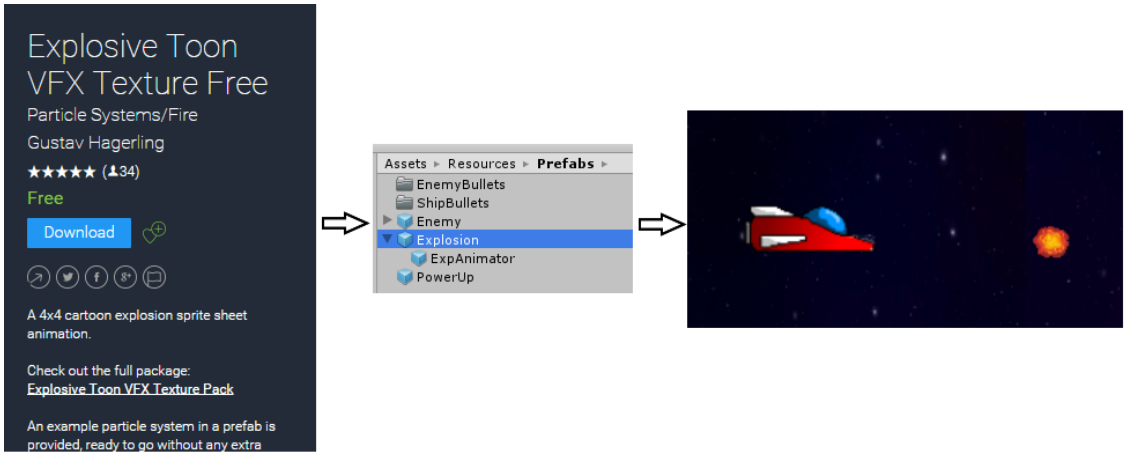


Figura 127: Unity Prefabs

Escenas

Por defecto Unity coloca una primera escena, un lienzo que representa un nivel. Permiten que a la hora de ejecutar el juego resulte más fácil poder cambiar de un estado del juego a otro completamente diferente, por ejemplo al cambiar de nivel, donde cargaríamos otro lienzo.

En cada una de estas escenas se incluyen los elementos que se usarán durante la ejecución, su función principal es el de poder seleccionar que elementos se deben cargar en memoria para poder ejecutar esa parte del juego. Al ser independientes entre ellas se agiliza el tiempo que el usuario debe esperar antes de ponerse a jugar.

Una vez, mas Unity se encarga de una parte del diseño. La herramienta por sí misma es capaz de almacenar y cargar todos los elementos necesarios que se usaran durante la ejecución del programa. El papel del programador en este caso es indicar a cada objeto cuál de estos elementos debe usar sin tener que preocuparnos de cargarlos manualmente.

Cada escena se guarda por separado y al ser independientes pueden incluirse en otros proyectos.

Para mantener información entre escena es necesario crear un objeto y asignarle la propiedad *dontDestroyOnLoad()*, esto permite que al pasar a una nueva escena este objeto siga existiendo y este referenciado.

Multimedia

En Unity podemos importar elementos como archivos de música o imágenes. Cada uno de estos elementos se puede tratar correctamente con métodos dedicados a ellos.

Los archivos de sonido se enlazan con componentes multimedia que ofrecen controlar su volumen, cuando reproducirlos, cambiar de canción y otra opciones. Los objetos instanciados del juego pueden incorporar uno de estos componentes para poder realizar tantos sonidos como sean necesarios a fin de enriquecer la experiencia de juego.

El componente de apariencia tiene un apartado al aspecto visual del objeto, es decir, su apariencia en pantalla. Para ello hay que importar imágenes que harán la función de *Sprites*. También pueden ser fondos de escenario o texturas para objetos tridimensionales. Todas estas pueden modificarse y cambiar durante la ejecución del juego, pudiendo crear efectos de animación para dar la impresión de que los personajes se mueven.

Recursos

La interfaz de Unity nos permite añadir cualquier recurso del ordenador, ya sean imágenes o sonidos. Pero si en ejecución se quiere modificar por ejemplo el *sprite* de un objeto desde script, Unity obliga a guardar los recursos en una carpeta especifica. Esta carpeta tiene como nombre *Resources* y tiene que existir en la raíz de *Assets*. Dentro de esta carpeta pueden existir tantas subcarpetas como se quieran, una vez aquí cualquier elemento puede ser accedido mediante el método siguiente:

Resources.Load<Type>(“Ruta”);

Type es el tipo de recurso y ruta su dirección incluida la extensión del archivo relativa a la carpeta *Resources*

Corrutinas

Una corrutina es un función que se ejecuta independiente de Unity, durante su uso el código interno se interpreta como C++ en vez de C# o JavaScript por lo que el editor no es capaz de recuperar la información que se calcula. Es decir, si dentro se ha calculado un dato que interesa reutilizar no se podrá hacer mediante un *return*. Ese valor se deberá almacenar en una variable de la clase.

Su característica principal es que este método queda en ejecución mientras el resto de la aplicación sigue su curso. De esta forma se pueden seguir realizando actividades que pueden requerir de un tiempo de cálculo y no impedir que el resto del programa quede a la espera de que termine.

Sobre todo han sido utilizadas para realizar consultas a la base de datos, tanto para guardar como recuperar información. Como se desconoce el tiempo que puede tardar en realizar estas acciones, por ejemplo debido al estado de la conexión a un servidor, es necesario dejar el método en ejecución hasta que termine.

Para saber cuándo ha terminado, en el interior de una corrutina está la sentencia *yield return* que espera a que la variable indicada reciba el valor antes de continuar el proceso. Para que esto funcione correctamente la función de la corrutina debe indicar que el tipo de retorno sea *IENumerator*.

Por ejemplo, si se quiere recuperar un numero de una base de datos.

```
public IENumerator cogerDato(){
  //Sentencias SQL
  resultado = (sentencia SQL);
  yield return resultado;
  print(resultado);
}
```

En este caso, hasta que resultado no reciba respuesta de la sentencia SQL, la función no ejecutara el código que aparece después de *yield return*.

Toda la información necesaria para programar con Unity se encuentra en, docs.unity3d.com. Una extensa documentación de todos y cada uno de los métodos existentes en Unity y de cómo manejarlos.

Diseñando un personaje móvil

Con un solo objeto es posible conseguir un personaje dentro de Unity que cumpla todas las funcionalidades, pero en un juego se quiere que el personaje sea vistoso, teniendo movimientos y diferentes físicas. Para ello lo que es un solo objeto a priori se divide en varios. Por ejemplo tenemos el personaje en su carreta, es una sola imagen, pero en ella podemos detectar distintos componentes: Sombrero, hombre, brazo con pistola, carreta, ruedas y chasis.

Figura 128: Unity - Diseñando un Personaje

Si queremos que estos diferentes objetos se desplacen por separado necesitamos tener cada sprite A la hora de programar se ha abordado cada personaje con su diagrama propio a fin de facilitar la comprensión de los esquemas. Cada componente tendrá funcionalidades propias, todas serán llamadas desde la clase principal jugador.

Como ahora el personaje está dividido en varios objetos también nos permite utilizar métodos de Unity para físicas y que cada uno disponga de sus propios sonidos. Como puede ser la unión del chasis con las ruedas mediante WheelJoint2D, a continuación se explica cada una de las utilizadas

WheelJoint2D

Este método une un objeto con otro haciendo que el segundo actúe como la rueda de un motor. Aporta el código necesario para poder utilizar ruedas pudiendo configurar la potencia y dirección del motor por un lado y un módulo para generar la suspensión deseada en la rueda.

SliderJoint2D

Una unión recta para poder fijar una distancia. En la carreta se utiliza para unir la carreta con el chasis. La distancia mínima y máxima de la unión puede alterarse en la ejecución para poder generar desplazamientos. Esta unión dispone programada un motor, para cambiar de posición debe activarse el motor en una dirección u otra.

SpriteRender

Permite que un objeto tenga representación en pantalla mediante una imagen. Esta imagen puede cambiarse en ejecución. En el caso de la carreta se ha utilizado para seleccionar el personaje femenino o masculino cambiando el sprite.

AudioSource

Componente para poder reproducir sonidos o música.

Diagrama de clases

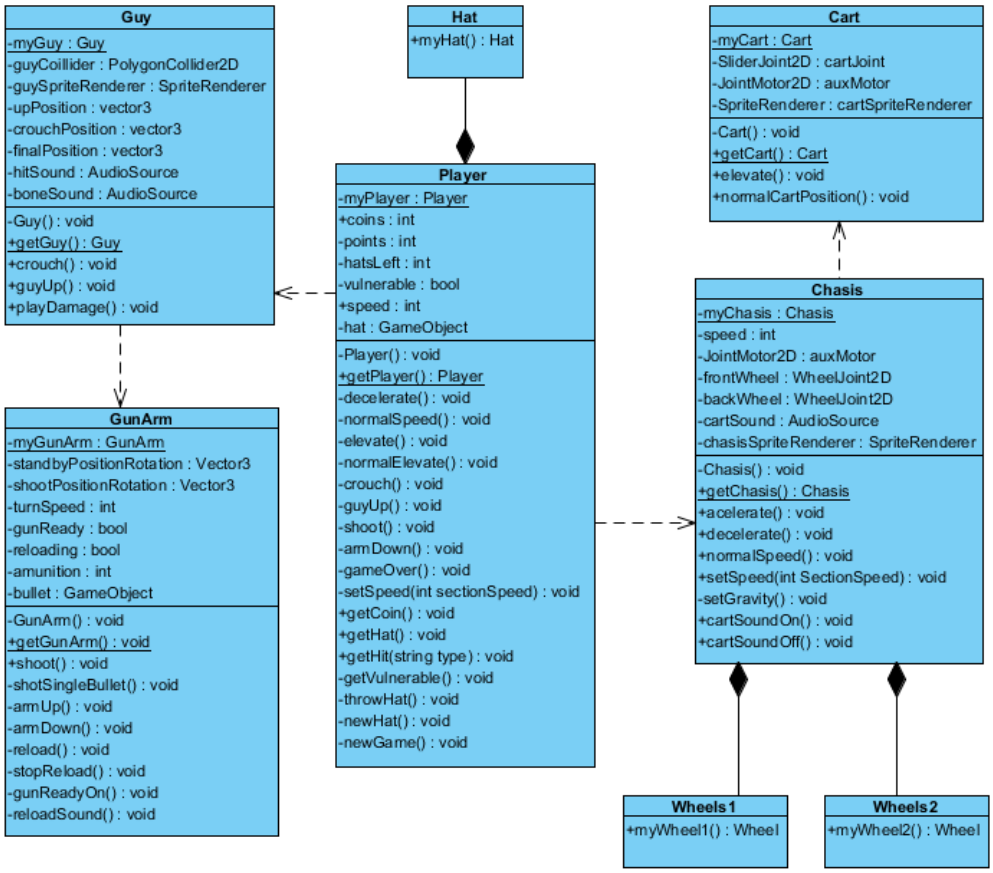


Figura 129: Unity - Diagrama de Clases Personaje

Diagrama de secuencia

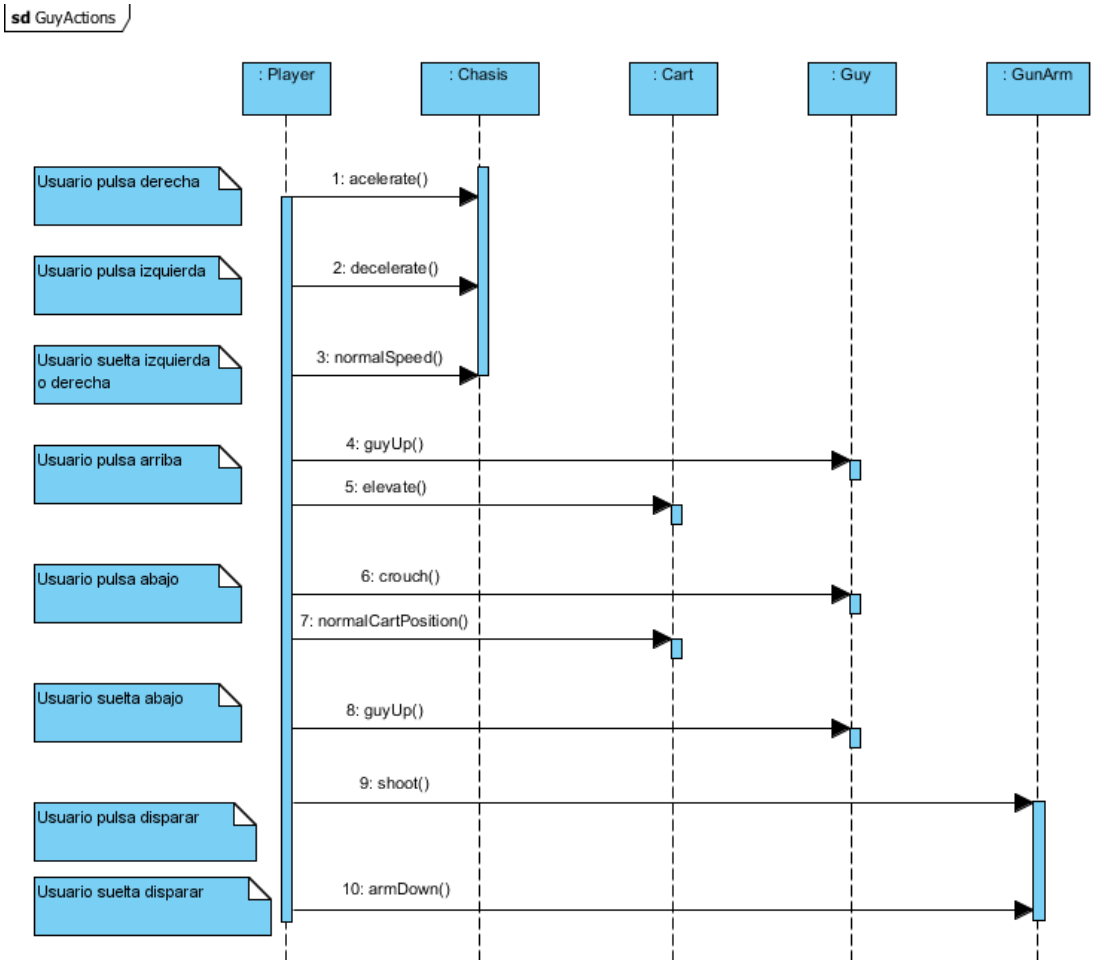


Figura 130: Unity - Diagrama de Secuencia Personaje

Animación de sprites

En un juego los personajes por normal general no son estáticos, se tienen que desplazar por el entorno y moverse para ser visualmente atractivos. Una de las formas de conseguir esto es cambiar el sprite de un objeto. Es decir el objeto no cambia, solo su representación en pantalla.

En Unity para poder animar un objeto existe el componente *Animator*. Este componente se encarga en cada momento de decidir qué tipo de sprite tiene que mostrar el componente *Sprite Renderer*. Esta decisión se toma mediante un pequeño diagrama y variables. Se explicara mediante el ejemplo de los murciélagos del juego *Arcade*.

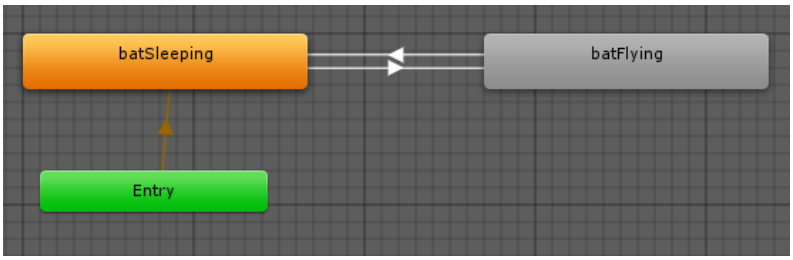


Figura 131: Unity - Sprites Secuencia

Cuando el objeto es creado siempre empieza en el estado *Entry*, desde este pasara automáticamente al estado conectado mediante flecha, en este caso a *batSleeping*. Como se puede observar de *batSleeping* a *batFlying* existe una flecha tanto de ida como de vuelta. Esta transición se realiza con el cambio de una variable. Cuando esta variable cambia dentro del juego mediante un script, la propiedad Animator se encarga automáticamente de cambiar de un estado a otro.

Además de todo lo anterior en cada estado puede existir una transición entre más de un sprite. En el ejemplo tenemos tres sprites para mostrar el murciélago dormido, y tres para mostrar el murciélago volando. En cada estado existe un bucle que se encarga de cambiar los sprites cuando pasa un tiempo definido por el programador.

Figura 132: Unity - Sprites Bat

Conexión y envío de información a base de datos

Debido al interés de guardar la información para su posterior consulta y tratado, la elección de usar una base de datos acarrea la necesidad de crear un sistema que sea seguro tanto para almacenar como para recuperar dichas entradas.

Realizar una conexión directa entre una aplicación creada en C# y una base de datos puede resultar muy peligrosa e insegura debido a que a través de ingeniería inversa se podrían obtener las credenciales para poder acceder a la base de datos.

Debido a esto, la propia herramienta Unity no permite esta conexión y hay que utilizar otros medios para poder conseguirlo. A continuación explicamos uno de estos métodos y que pasos conlleva establecer el vínculo entre aplicación y una BBDD. Para facilitar como lo hemos implementado usaremos la tecnología que hemos aplicado para ello ya que permite distintos lenguajes y bases de datos dependiendo de las necesidades del desarrollador.

Figura 133: Conexión Base de Datos

La comunicación e intercambio de información se realiza a partir de una conexión entre Unity y un servicio web que a su vez hará la conexión a la base de datos.

En nuestro caso hemos decidido utilizar MySQL como sistema de base de datos y PHP como el lenguaje web para comunicarnos con ella desde nuestro juego. Por una parte hemos elegido estas dos herramientas pues han sido las que hemos utilizado durante nuestra docencia, la otra razón es que la documentación oficial de Unity ofrece ejemplos sencillos para implementar este sistema usando estas mismas aplicaciones.

Implementación de Base de Datos

La base de datos no requiere de una configuración concreta o especifica más allá de lo habitual a la hora de crear tablas. Con diseñar e implementar como serán las relaciones y crear cada tabla es suficiente en este caso.

Ejemplo:

```
CREATE TABLE `datos` (
  `id` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `name` VARCHAR(15) NOT NULL DEFAULT 'Desconocido',
  `edad` INT(3) UNSIGNED NOT NULL DEFAULT '0 )
```

ID	Nombre	Edad

Siendo cada celda:

- ID: Numérico auto incremento y clave primaria
- Nombre: Texto
- Edad: Numérico

Implementación del formulario web en PHP

Los archivos web incluyen el código necesario para establecer la conexión con la base de datos. Los parámetros necesarios son:

- Dirección IP donde se encuentra la base de datos
- Base de datos a la que hay que conectarse y sus credenciales
- Sentencia SQL para insertar o recoger información de la BBDD

El propio archivo PHP es el que realiza la consulta, por lo que nuestra aplicación solo se tiene que encargar de enviarle la información de cada parámetro que contendrá los datos de cada variable de la tabla. Es importante documentar correctamente como se llamará cada elemento del formulario donde se guardará la información recibida pues serán utilizados cuando se implemente en la programación del juego.

Para recoger la información usaremos el array asociativo *$_GET* y guardaremos cada dato en una variable. Después crearemos la sentencia SQL que introducirá los datos en la base de datos y la ejecutamos.

Ejemplo:

```php
<?php

$db = mysql_connect('host', 'user_database', 'password')
or die('Could not connect: ' . mysql_error());

mysql_select_db('BBDD_proyecto') or die('Could not select database');

$nombre = mysql_real_escape_string($_GET['nombre'], $db);
$edad = mysql_real_escape_string($_GET['edad'], $db);

$query = "insert into datos values (NULL, '$nombre', '$edad');";
$result = mysql_query($query) or die('Query failed: ' . mysql_error());

?php>
```

mysql_real_escape_string : Permite hacer una conexión con la BBDD para comprobar que el dato tiene los caracteres correctos para ser aceptado. Para comprobarlo, además del dato de la variable se envía la conexión de la base de datos para su verificación.

Crearemos un archivo PHP por cada tipo de consulta que se hará con la base de datos. Estos archivos se almacenarán en el servidor web.

Implementación del script en Unity

El archivo C# se creará con una serie de variables y métodos para poder realizar la llamada al formulario web. Con el ejemplo a continuación explicamos que es necesario.

Usaremos una variable de tipo String donde guardaremos la URL al archivo php donde se encuentra el formulario que hemos programado anteriormente.

```
string ejemploURL = “http://ejemplo.com/unity/formulario.php?”;
```

Es importante añadir la interrogación al final de la dirección porque luego se concatenará con las variables que se usarán para completar la URL que se enviará al formulario para que la comprenda correctamente.

```
void main(){
        StartCoroutine(EnviarDatos("Pepito", 13));
}
```

```
IEnumerator EnviarDatos(string nombre, int edad)
  {
  string post_url =
  ejemploURL + "nombre=" + WWW.EscapeURL(nombre) + "&edad=" + edad;
  WWW hs_post = new WWW(post_url); // Se accede a la dirección URL
  yield return hs_post; // Esperar a que la descarga se complete
  if (hs_post.error != null)
    {
      print("Error al enviar los datos: " + hs_post.error);
    }
  }
```

Una vez concatenamos toda la sentencia URL, quedaría de la forma de una dirección de formulario una vez enviado:

```
http://ejemplo.com/unity/formulario.php?nombre=variable_nombre&edad=variable_edad
```

Esta acción se realiza en lo que se denomina una corrutina. Durante la ejecución natural de un videojuego, las funcionen se esperan que cumplan sus tareas dentro del mismo cuadro de ejecución para que todos los parámetros estén actualizados en cada ciclo. Gracias a la corrutina, la función ahora puede esperar a otro próximo ciclo para poder seguir trabajando y permitirle realizar una secuencia de eventos en el tiempo. El envío del formulario podrá tomarse su tiempo para poder realizar sus tareas.

Cuando se utiliza una función con la característica de ser una corrutina hay que realizar la llamada de la función indicando que debe ser ejecutado de esa forma. En el código del programa la función estará contenida en la siguiente sentencia:

```
StartCoroutine(funcion());
```

La función llamada necesitará además cumplir ciertos requisitos para funcionar correctamente de esta forma. *IEnumerator* es el tipo de dato que devolverá, un enumerado que contendrá el número de elementos conseguidos durante la ejecución de la función. Para recuperar esa información, tenemos que indicar delante de la sentencia *return* el atributo *yield*. Esta es la opción que permite a la función seguir trabajando hasta que ha completado sus tareas antes de poder continuar. En este caso, recuperamos cualquier mensaje que nos haya devuelto el servicio web tras la ejecución de toda la función.

Importante aclarar que la sentencia *yield return* no es la misma que el *return* estándar de una función que devuelve un valor. Ésta permite a la aplicación continué funcionando mientras aún se espera a que la aplicación externa complete sus funciones para que cuando acabe pueda continuar.

Para poder enviar los datos como si fuera una dirección web, Unity utiliza el módulo *WWW*. Es una *API* que le permite trabajar con contenidos de direcciones web. Por defecto realiza la tarea de recuperar información desde una web, pero también se puede utilizar para enviarlo mediante *WWWForm*. Los datos que puede recoger de una web pueden ser de una gran variedad. En nuestro proyecto se usará unicamente para enviar y recibir información en forma de texto.

La función *WWW.EscapeURL(string)* permite adaptar el texto para que sea comprensible como dirección web y adaptar cualquier carácter para que no produzca algún problema.

Implementación de seguridad de información

La principal razón de implementar este tipo de comunicación es la seguridad. La implementación utilizada en los ejemplos arriba mencionados no implementan seguridad alguna y alguien con información sobre la base de datos podría inyectar información para realizar actividades no deseadas como modificar la información introducida o recogida.

La solución es utilizar el método de cifrado md5 ayudándonos de una palabra clave que ayudará a dificultar cualquier posible alteración y recuperación no deseada de los datos. MD5 es un algoritmo que dado una sentencia de texto dará siempre una misma respuesta que no tendrá relación alguna con la información recibida.

La palabra clave tiene como intención ser un trozo de texto adicional a la información que queremos enviar o recibir. Como es una palabra que no está almacenada en la base de datos, cualquier ataque que se reciba no podrá acceder a esa información incluso sabiendo cómo está construida la base de datos. Esta palabra se necesitará tanto en el formulario PHP en el servidor web como en el script C# donde tenemos las funciones relacionadas con el envío de los datos.

Ejemplos de envió de datos al servidor:

Archivo PHP en el servidor Web

En el archivo PHP con el formulario habrá que añadir una nueva variable con la palabra clave. Además ahora crearemos otra variable adicional para el formulario que recogerá una nuevo dato al que llamaremos "hash":

```
$palabraClave="pAlbr4Cl4Be";
$hash = $_GET['hash'];
```

El contenido de la variable hash es la combinación de las variables que se introducirán más la palabra clave creada en el código del juego antes de haber sido enviado. Su función es la de ser una frase que permite comprobar que la información que recibe el formulario no ha sido modificada. Para recrearlo se hará el mismo proceso y se guardará en la variable *real_hash*. Como es una comprobación, antes de ejecutar la sentencia SQL añadiremos una condición que nos verifica que tanto el *hash* recibido desde el juego como el que se acaba de crear en *real_hash* son idénticos y todo esta correcto:

```
$real_hash = md5($nombre . $edad . $palabraClave);
if($real_hash == $hash)
    {
    $query = "insert into scores values (NULL, '$nombre', '$edad');";
$result = mysql_query($query) or die('Query failed: ' .
mysql_error());
        }
```

En la variable result se almacenará el resultado de la acción realizada y será lo que la función en el proyecto del juego recibirá y devolverá cuando finalice su tarea.

Archivo C# en el proyecto de Unity

Para el caso de C# el único cambio que tenemos que hacer es calcular el hash que enviaremos e incluirlo en la variable con la URL que se envía al formulario. Para realizar un cifrado Md5 en Unity, hay que usar el sistema de criptografía incluido, por lo que realizando una llamada al método adecuado con la frase a cifrar ya tenemos el *hash* deseado. No hay que olvidar que tenemos que usar la misma palabra clave para que la frase cifrada sea idéntica en ambos archivos.

```
string palabraClave = "pAlbr4Cl4Be";
string hash = Md5Sum("Pepito" + 13 + palabraClave);
```

Ahora hay que modificar la sentencia a enviar al formulario de forma que el original:

```
string post_url =
   ejemploURL + "nombre=" + WWW.EscapeURL(nombre) + "&edad=" + edad;
```

Se reemplaza por este otro en el que se incluye el *hash* a los datos a enviar:

```
string post_url =
   ejemploURL + "nombre=" + WWW.EscapeURL(nombre) + "&edad=" + edad + "&hash=" +
hash;
```

Ahora ya está implementado el módulo para poder guardar cualquier información del juego en nuestra base de datos. Los dos archivos quedarían, una vez terminados, de la siguiente forma.

Archivo PHP completo

```
<?php
$db = mysql_connect('host', 'user_database', 'password')
or die('Could not connect: ' . mysql_error());

mysql_select_db('BBDD_proyecto') or die('Could not select database');

$nombre = mysql_real_escape_string($_GET['nombre'], $db);
$edad = mysql_real_escape_string($_GET['edad'], $db);
$hash = $_GET['hash'];

$palabraClave="pAlbr4Cl4Be";

$real_hash = md5($nombre . $edad . $palabraClave);

if($real_hash == $hash)
{
        $query = "insert into scores values (NULL, '$nombre', '$edad');";
        $result = mysql_query($query) or die('Query failed: ' . mysql_error());
}
?php>
```

Archivo C# completo

```
public class Ejemplo {

string ejemploURL = “http://ejemplo.com/unity/formulario.php?”;
string palabraClave = "pAlbr4Cl4Be";
string hash = Md5Sum("Pepito" + 13 + palabraClave);

void main(){

        StartCoroutine(EnviarDatos("Pepito", 13));
}

IEnumerator EnviarDatos(string nombre, int edad)
  {

  string post_url =
ejemploURL + "nombre=" + WWW.EscapeURL(nombre) + "&edad=" + edad +
"&hash=" + hash;

  WWW hs_post = new WWW(post_url); // Se accede a la dirección URL

  yield return hs_post; // Esperar a que la descarga se complete

  if (hs_post.error != null)
    {
      print("Error al enviar los datos: " + hs_post.error);
    }
  }
}
```

Ejemplos de recogida de datos desde el servidor:

Si en cambio lo que queremos es recuperar información de la base de datos, el proceso se realiza como si el archivo PHP fuera a mostrar datos por pantalla al usuario, mediante la acción *echo*. La API de Unity se encarga de recoger cada línea como entrada de información para tratarla en el juego como un texto.

Sentencias PHP para recoger información

```
 $query = "SELECT * FROM `datos` ORDER by `edad` DESC LIMIT 5";

$result = mysql_query($query) or die('Query failed: ' . mysql_error());

$num_results = mysql_num_rows($result);

for($i = 0; $i < $num_results; $i++)
   {
   $row = mysql_fetch_array($result);
   echo $row['nombre'] . "\t" . $row['edad'] . "\n";
 }
```

Como la solicitud es solo de recoger datos, en el código del juego no nos hace falta crear variables que haya que enviar en forma de formulario. Así pues con solo hacer la llamada a la URL del archivo PHP encargado de recuperar esa información ya es suficiente.

Método C# para recoger información

```
IEnumerator obtenerDatos()
{
        string ejemploURL = “http://ejemplo.com/unity/recoger.php”;

        WWW hs_get = new WWW(ejemploURL);

        yield return hs_get; //Esperar a descargar los datos.

        if (hs_get.error != null)
        {
                print("Hubo un problema al recoger los datos: " + hs_get.error);
        }
        else
        {
                string resultado = hs_get.text;
        }
}
```

Hay que indicar enérgicamente que esta parte de implementar seguridad al envío de información a la base de datos solo hace hincapié en que la información enviada al

servidor y los datos recibidos son iguales y no han sido modificados durante el trayecto. NO implica que la información enviada haya sido cifrada. Si alguna aplicación externa recoge esa información podrá ver la información enviada menos la palabra clave que está escondida dentro del cifrado del hash.

Para ello, además de estos pasos, se incluirá un sistema de cifrado de los datos delicados antes de ser enviados a la base de datos para que esta información pueda viajar de forma completamente segura.

BBDD + PHP + UNITY

Para realizar la comunicación entre la aplicación y la base de datos se utilizarán una serie de archivos PHP que tendrán funciones concretas. Dependiendo de lo que se necesite, cada archivo PHP actuará como intermediario para obtener o almacenar información. La información que genera el archivo PHP se muestra por pantalla como texto plano, Unity puede recogerlos y tratarlos.

Cada uno de estos procesos contiene una comprobación para que los datos recibidos sean correctos y enviados realmente por la aplicación y no un tercero que pretenda acceder al sistema. Valores como el correo y la contraseña se envían cifrados para que no puedan ser obtenidos durante la comunicación.

A continuación indicamos las páginas a las que se pueden acceder durante la ejecución de la aplicación.

register.php

Esta página se encarga de introducir un nuevo usuario al sistema. Es llamado desde Unity a la hora de querer crear un nuevo usuario para la aplicación. Dependiendo de lo que pueda ocurrir durante su funcionamiento se ofrecen algunos valores para comprobar cuál ha sido el problema.

Para insertar un nuevo usuario en necesario que la página reciba los siguientes valores:

- Correo electrónico del usuario
- Contraseña
- Edad
- Sexo
- Valor Hash generado en el juego

Al recibir estos valores, el registro realizará los siguientes pasos para introducir el nuevo usuario:

1. Genera de nuevo el valor del Hash a partir de los valores recibidos.
2. Compara los dos valores de Hash.
    a. Si coincide se pasa al punto 3º.
    b. Si no coincide se muestra por pantalla un número (0) y se detiene el proceso.
3. Se comprueba que el correo no existe en la base de datos.
    a. Si no existe se pasa al punto 4º.
    b. Si existe se muestra por pantalla un número (1) y se detiene el proceso.
4. Se procede a guardar el nuevo usuario.
5. Si no se guarda se mostrará el error que el sistema ha generado y se detiene el proceso.
6. Si se guarda correctamente se muestra un número (2) y se realiza el punto 5º.
7. Se crean datos iniciales en la base de datos para la encuesta, los tutoriales y las partidas para el estudio y se asignan al nuevo usuario.

Leyenda de los números que se mostrarán:

0. El Hash enviado por el formulario no coincide con el creado por la aplicación.
1. El correo del usuario ya existe en la base de datos.
2. El registro ha sido correcto y se han introducido los datos en la BBDD.

login.php

Para poder acceder a todas las secciones necesarias para participar en el estudio, Unity debe recoger la información ya almacenada del jugador. Puede darse el caso de que los datos introducidos sean incorrectos o inventados, en cuyo caso habrá que avisar de que tal usuario no existe en la base de datos.

Para poder hacer login es necesario que la página reciba los siguientes valores:

- Correo Electrónico
- Contraseña
- Valor Hash generado en el juego

Al recibir estos valores, el login realizará los siguientes pasos para comprobar si existe el usuario:

0. Genera de nuevo el valor del Hash a partir de los valores recibidos.
1. Compara los dos valores de Hash.
    a. Si coincide se muestra por pantalla el texto “OK”
    b. Si no coincide se muestra por pantalla el texto “MAL” y se detiene el proceso.

userdata.php

Cuando Unity recibe el OK de la llamada a login, procede a recoger información del usuario que indica en que situación del estudio está. Se obtienen datos sobre que pregunta de la encuesta se encuentra, el resultado de dicha encuesta si la ha terminado, si ha completado los tutoriales de los juegos y si ya ha jugado las partidas de recogida de datos.

Como existen dos aplicaciones en el que la encuesta condice pero el juego no, el archivo necesitará saber cuál es el que está solicitando la información. Sabiendo cual es, mostrará únicamente la situación del juego indicado.

En este caso, Unity se encarga de tomar estos valores para tratarlos y comprobar que corresponden correctamente a lo que el juego espera. Como estos datos se solicitan tras hacer correctamente el login, se asume que también existen, al menos, datos de una situación inicial.

Para poder recoger información del usuario es necesario que la página reciba los siguientes valores:

- Correo Electrónico
- Que juego está jugando
- Valor Hash generado en el juego

Al recibir estos valores, se realizarán los siguientes pasos para obtener la información:

0. Genera de nuevo el valor del Hash a partir de los valores recibidos.
1. Compara los dos valores de Hash.
    a. Si coincide se pasa al punto 3º.
    b. Si no coincide se muestra por pantalla el texto “bad hash” y se detiene el proceso.
2. Se obtienen los datos sobre la encuesta y la situación con el tutorial y el juego.
    a. Se muestran los siguientes valores separados por guión para que Unity los trate.

edad – género – id_encuesta – numero_pregunta – resultado_test – tutorial_completado – partida_completada

quiz.php

Esta página se encarga de gestionar toda la parte del cuestionario y tiene tres funciones. La primera permite obtener la siguiente pregunta que el usuario tiene que responder. La segunda permite almacenar la respuesta a la pregunta actual a la que está respondiendo. La última muestra todas las respuestas realizadas por pantalla para que Unity pueda trabajar con ellas.

Para obtener la siguiente pregunta es necesario que la página reciba los siguientes valores:

- Indicar modo recoger pregunta (get)
- Numero siguiente pregunta
- ID del cuestionario del usuario
- Valor Hash generado en el juego

Al recibir estos valores, se realizarán los siguientes pasos para obtener la pregunta:

0. Genera de nuevo el valor del Hash a partir de los valores recibidos.
1. Compara los dos valores de Hash.
    a. Si es correcto mostrará la pregunta por pantalla.
    b. Si es incorrecto indicará “mal hash”.

Para almacenar la respuesta de una pregunta es necesario que la pagina reciba los siguiente valores:

- Indicar modo responder pregunta (save)
- ID del cuestionario del usuario
- Numero de pregunta a la que se responde
- Respuesta
- Valor Hash generado en el juego

Al recibir estos valores, se realizan los siguientes pasos para almacenar la respuesta:

0. Genera de nuevo el valor del Hash a partir de los valores recibidos.
1. Compara los dos valores de Hash.
    a. Si es correcto guardará la pregunta en el servidor.
    b. Si no es correcto se mostrará el mensaje “ERROR”.

Para recoger todas las respuestas por pantalla es necesario que la página reciba los siguientes valores:

- Indicar modo recoger respuestas (final)
- ID del cuestionario del usuario
- Valor Hash generado en el juego

Al recibir estos valores, se realizan los siguientes pasos para recoger la información:

0. Genera de nuevo el valor del Hash a partir de los valores recibidos.
1. Compara los dos valores de Hash.
    a. Si es correcto mostrará las respuestas almacenadas por pantalla.
    b. Si es incorrecto no mostrará nada.

Las respuestas al ser un valor binario se pueden mostrar como una única sentencia y dejar a Unity que se encargue de gestionar esos datos.

evaluation.php

Su función es sencilla, únicamente almacena o recupera el resultado del test escrito.

Para almacenar el resultado de la evaluación es necesario que la página reciba los siguientes valores:

- Indicar modo guardar evaluación (post)
- ID del cuestionario del usuario
- Valor Hash generado en el juego

Al recibir estos valores, se realizan los siguientes pasos para guardar el resultado:

0. Genera de nuevo el valor del Hash a partir de los valores recibidos.
1. Compara los dos valores de Hash.
    a. Si es correcto guardará la información en la base de datos.
    b. Si es incorrecto mostrará por pantalla “bad hash”.

Para obtener el resultado de la evaluación en necesario que la página reciba los siguientes valores:

- Indicar modo recoger evaluación (get)
- ID del cuestionario del usuario
- Valor Hash generado en el juego

Al recibir estos valores, se realizan los siguientes pasos para guardar el resultado:

0. Genera de nuevo el valor del Hash a partir de los valores recibidos.
1. Compara los dos valores de Hash.
    a. Si es correcto mostrará la información por pantalla.
    b. Si es incorrecto mostrará por pantalla “bad hash”.

savearcade.php & saveshooter.php

Son archivos similares que se centran en almacenar los resultados de la partida de la aplicación a la que hayan jugado. Se han hecho dos archivos por claridad a la hora de implementar las llamadas a la base de datos. Permite almacenar si se ha completado el tutorial además de poder guardar los resultados de la partida de evaluación.

Para almacenar que se ha completado el tutorial es necesario que la página reciba los siguientes valores:

- Indicar modo guardar tutorial completado (tutorial)
- Correo electrónico del usuario

Al recibir estos valores, se realizan los siguientes pasos para guardar que se ha hecho el tutorial:

0. Genera de nuevo el valor del Hash a partir de los valores recibidos.
1. Compara los dos valores de Hash.
    a. Si es correcto mostrará se guardará la información en la base de datos.
    b. Si es incorrecto mostrará por pantalla “bad hash tutorial”.

Para almacenar los resultados de la partida de evaluación en necesario que la página reciba los siguientes valores:

- Indicar modo guardar tutorial completado (game)
- Correo electrónico del usuario
- Todas las variables necesarias de la evaluación del juego.

Al recibir estos valores, se realizan los siguientes pasos para guardar la evaluación:

0. Genera de nuevo el valor del Hash a partir de los valores recibidos.
1. Compara los dos valores de Hash.
    a. Si es correcto mostrará se guardará la información en la base de datos.
    b. Si es incorrecto mostrará por pantalla “bad hash Game”.

Manuales

Manual instalación

Ambos juegos son instalados de la misma forma. El software se entrega en una carpeta comprimida de formato .zip.

Dentro del archivo comprimido se encuentra una carpeta donde están todos los datos necesarios para la ejecución del juego. Debe descomprimirse la carpeta en una ubicación a elección del usuario. Para ejecutar el programa iniciaremos el archivo .exe respectivo de cada juego.

Dentro de la carpeta comprimida existen los siguientes archivos:

Carpeta Data:

Contiene todos los recursos que necesita el juego para ejecutarse: escenas, gráficos, sonidos… Esta carpeta es generada por Unity junto al ejecutable.

Juego.exe

Ejecutable generado por Unity para lanzar el juego.

Modelos .model

Modelos para la predicción de la clase del usuario en caso de que juegue directamente al test

Archivo .bat

Necesario para comunicar Unity con Java cuando se quiere que weka analice los resultados del test

Juego .jar

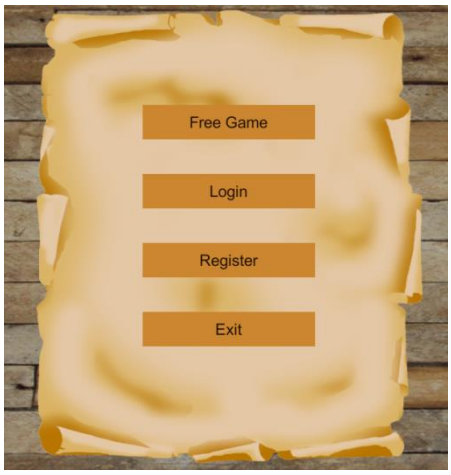
Archivo que utilizando el modelo y el archivo .arff generado con los resultados del text predice la clase del usuario.

Aparte de los archivos citados cuando se ejecuta el programa se generan archivos extras:

- Archivos .arff con los datos recopilados del usuario para ser analizados por el modelo
- Archivo txt con los resultados del modelo.

Navegación por el menú:

Menú Inicial



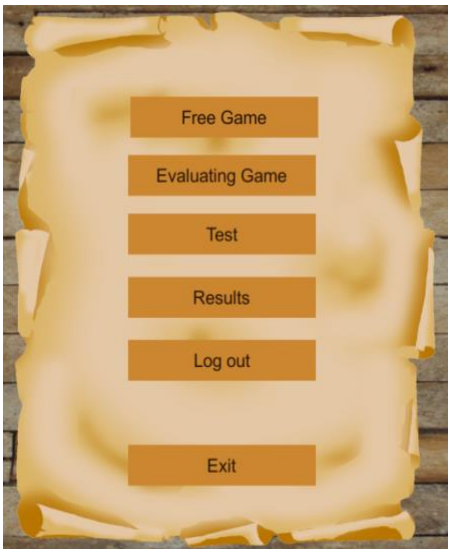
Free Game: Partida sin recopilación de datos, permite jugar una partida normal.

Login: Pestaña para iniciar sesión en el programa.

Register: Pestaña para iniciar el proceso de crear un nuevo usuario.

Exit: Salir del programa.

Menú Registrado



Free Game: Partida sin recopilación de datos, permite jugar una partida normal.

Evaluating Game: Partida con recopilación de datos y predicción.

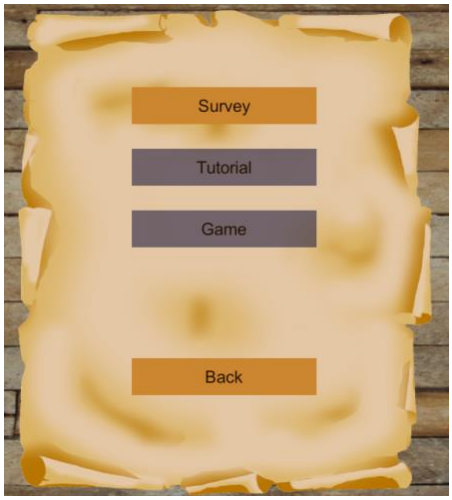
Test: Accede al menú para realizar la encuesta y el test para poder colaborar con la investigación.

Results: Muestra los resultados de la partida de evaluación.

Log Out: Vuelve al menú inicial.

Exit: Salir del programa.

Menú Test



Survey: Accede a la encuesta de personalidad.

Tutorial: Aprende a utilizar el juego.

Game: Juega una partida aportando datos para mejorar los módulos predictores.

Back: Vuelve al menú registrado.

Juego Shooter

Controles

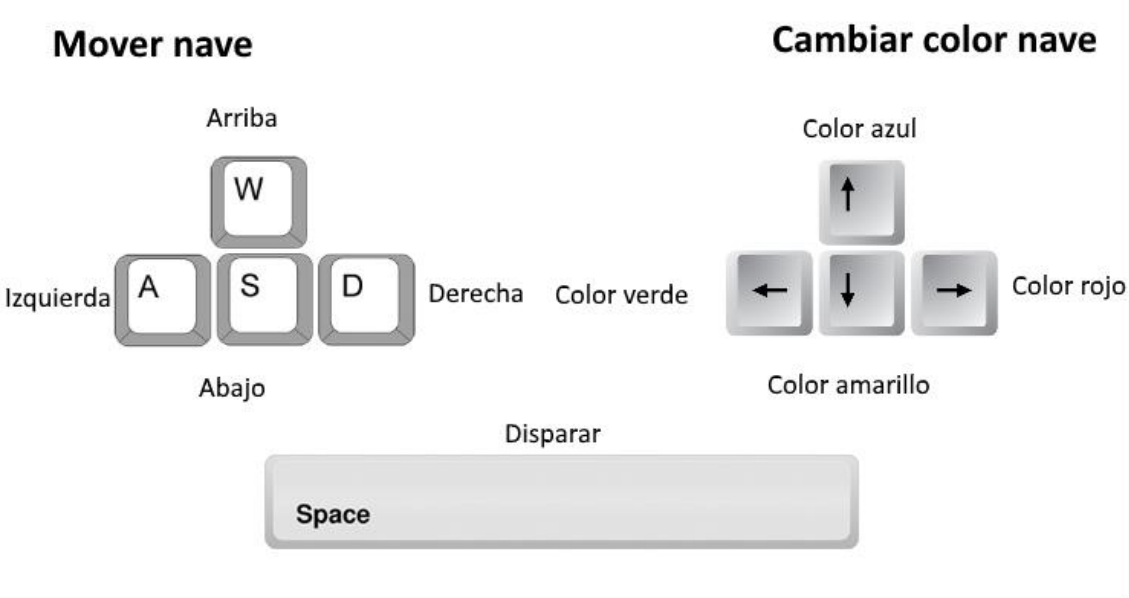


Figura 134: Manual - Arcade Controles

Juego Arcade

Controles

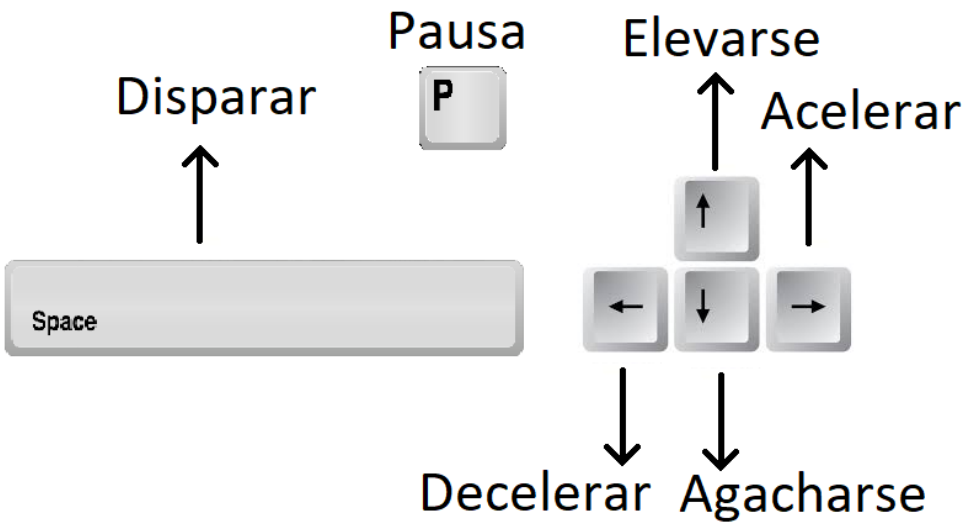


Figura 135: Manual - Shooter Controles