

Roberto Daniel Torres Cruz

PROYECTO FINAL DESARROLLO DE SOFTWARE: KILLERGAME



Tutor:
Juan Miguel Ramon Tur

CIFP FRANCESC DE BORJA MOLL
Desarrollo Multiplataforma
2022/23

Roberto Daniel Torres Cruz

PROYECTO FINAL DESARROLLO DE SOFTWARE: KILLERGAME

Tutor:
Juan Miguel Ramon Tur

CIFP FRANCESC DE BORJA MOLL
Desarrollo Multiplataforma
2022/23

ÍNDICE GENERAL

ÍNDICE GENERAL	IV
ÍNDICE DE FIGURAS	VII
ÍNDICE DE CUADROS	VIII
1 - INTRODUCCIÓN	8
1.1 - DESCRIPCIÓN DEL PROYECTO GLOBAL	8
1.2 - PROPUESTA	8
1.3 - ESTRUCTURA DEL DOCUMENTO	9
2 - DESCRIPCIÓN DE LA ARQUITECTURA GLOBAL DEL PROYECTO	10
2.1 METODOLOGÍA	10
2.2 FASES DEL DESARROLLO DEL SOFTWARE	11
2.3 METODOLOGÍA SCRUM	12
2.4 - PLANIFICACIÓN DEL PROYECTO	13
2.5 - ANÁLISIS	14
2.5.1 - PARTES INTERESADAS	14
2.5.2 - REQUISITOS DE LA APLICACIÓN	14
2.5.2.1 - REQUISITOS FUNCIONALES	14
2.5.2.2 - REQUISITOS NO FUNCIONALES	15
2.5.2.3 - REQUISITOS DE SISTEMA	15
2.5.3 - REGLAS DEL JUEGO	15
2.5.4 - PRUEBAS DE ACEPTACIÓN DE REQUISITOS	16
2.5.4.1 - OBJETIVOS DE LAS PRUEBAS DE ACEPTACIÓN DE REQUISITOS	16
2.5.4.2 - DESCRIPCIÓN DE LAS PRUEBAS	16
2.5.5 - DISEÑO	17
2.5.5.1 - MODELO CONCEPTUAL DEL PROYECTO	17
2.5.5.2 - DIAGRAMA DE CLASES	18
2.5.5.2.1 - DIAGRAMA DE CLASES GENERAL APLICACIÓN PRINCIPAL	18
2.5.5.2.2 - DIAGRAMA DE CLASES GENERAL APLICACIÓN MÓVIL	19
2.5.5.2.3- DIAGRAMA DE CLASES GENERAL APLICACIÓN SONIDO	19
3 - PLANIFICACIÓN Y DESCRIPCIÓN DE SUBPROYECTOS	20
3.1 SUBPROYECTOS	20
3.1.1 Coordinación	20
3.1.2 Main Controller / Integrador	20
3.1.3 Visual	20
3.1.4 Comunicaciones	21
3.1.5 Aplicación móvil	21
3.1.6 Sonido	21
3.1.7 Cluster	21
3.2 Asignación de tareas / roles por subtareas y por colaborador	22
4 - DESCRIPCIÓN DETALLADA DEL SUBPROYECTO INDIVIDUAL	24

4.2 - ROL ASIGNADO	24
4.2.1 - DESARROLLO	24
4.2.2 - RESPONSABILIDADES ADICIONALES	24
5 - PLANIFICACIÓN DE LAS TAREAS DEL SUBPROYECTO INDIVIDUAL	26
5.1 - INTRODUCCIÓN	26
5.2 - DIAGRAMA DE GANTT	26
6 - DETALLE DE LAS DEDICACIONES Y TAREAS REALIZADAS	27
6.1 - DISEÑO DE LAS CLASES	27
6.2 - ASIGNACIÓN DE TAREAS	27
6.3 - DESARROLLO DE LAS CLASES	27
6.4 - SPRINTS REVIEWS SEMANALES	28
7 - DESCRIPCIÓN DEL ENTORNO TECNOLÓGICO Y HERRAMIENTAS USADAS	29
7.1 - VISUAL STUDIO CODE v1.67.1	29
7.2 - ECLIPSE IDE v2022-23 (4.23.0)	29
7.3 - ANDROID STUDIO 2021.3.1 PATCH 1	29
7.4 - INTELLIJ IDEA 2022.3.1	30
7.5 - ATLASSIAN JIRA CLOUD	30
7.6 - GITHUB	30
7.7 GIT	30
7.7.1 - ESTRUCTURA DE REPOSITORIOS	31
7.7.2 - ESTRUCTURA DE RAMAS	31
7.7.3 - ESTRUCTURA DE PERMISOS	32
8 - DESCRIPCIÓN DE LA ARQUITECTURA DEL SUBPROYECTO	33
9 - DESCRIPCIÓN DE LOS MÓDULOS O ÍTEMES DESARROLLADOS.	34
9.1 - PHYSICS ENGYNE	34
9.2 - DYNAMIC VISUAL OBJECT	35
9.3 - ANIMATION	37
9.4 - POSITION	39
10 - MANUALES DE USUARIO E INSTALACIÓN	40
10.1 - APLICACIÓN PRINCIPAL	40
10.2 - APLICACIÓN MÓVIL	40
10.2.1 - INSTALACIÓN	40
11 - CONCLUSIONES SOBRE EL PROYECTO GLOBAL Y SUBPROYECTO	45
11.1 - INTRODUCCIÓN	45
11.1.1 - CONCLUSIONES DEL PROYECTO GLOBAL	45
11.1.1.1 - REQUISITOS LOGRADOS	45
11.1.1.2 - EXPECTATIVAS Y MEJORAS	46
11.1.1.1.1 - MEJORAS EN EL PROYECTO GLOBAL	46
11.1.1.3 - LECCIONES APRENDIDAS	47
11.1.2 - CONCLUSIONES DEL SUBPROYECTO PROYECTO INDIVIDUAL	48
11.1.2.1 - EXPECTATIVAS Y MEJORAS	48
11.1.2.1.1 - MEJORAS EN LA PLANIFICACIÓN Y GESTIÓN	48

ÍNDICE DE FIGURAS

1.1 - Diagrama básico del proyecto	10
2.1 - Ilustración desarrollo metodología scrum	14
2.2 - Diagrama de Gantt. Fases 1, 2 y 3	14
2.3 - Diagrama de Gantt. Fases 4 y 5	14
2.4 - Gráfico proceso de reunión-aproximación	18
2.5 - Modelo conceptual del proyecto	19
2.6 - Diagrama de clases general aplicación principal	19
2.7 - Diagrama de clases general aplicación móvil	20
2.8 - Diagrama de clases general aplicación sonido	20
5.1 - Diagrama de Gantt subproyecto coordinador, Fases 1, 2, 3 y 4	27
5.2 - Diagrama de Gantt subproyecto coordinador, Fases 4 y 5	27
7.1 - Diagrama de ramas visual	34
8.1 - Arquitectura responsabilidades coordinador	35
9.1 - Workflow proyecto Jira	38
9.2 - Dashboard personalizado Jira	39
10.1 - Código script.sh	40
10.2 - Activación opciones desarrolladores	41
10.3 - Activación depuración por USB	41
10.4 - Importar proyecto android 1	42
10.5 - Importar proyecto android 2	42
10.6 - Instalación aplicación en dispositivo	43
10.7 - Inicio aplicación móvil	43
10.8 - Pantalla inicial del mando de juego	44

ÍNDICE DE CUADROS

3.1 - Tabla de asignación de subproyectos / responsabilidades	24
11.1 - Matriz requisitos logrados	45

1 - INTRODUCCIÓN

1.1 - DESCRIPCIÓN DEL PROYECTO GLOBAL

El proyecto final para el ciclo formativo de desarrollo de aplicaciones multiplataforma se basa en desarrollar un juego que a gran escala partiendo de la base del video juego “asteroids”.

El desarrollo en lugar de centrarse en un aplicativo monousuario y mono equipo, se centra en replicar la idea del videojuego “asteroides” a gran escala, donde una aplicación se conecta con diferentes ordenadores, manejando así la pantalla de cada uno de estos equipos. Estas pantallas forman una matriz tipo “videowall” de 3 x 4 pantallas, formando así en su conjunto una pantalla gigante de juego, donde cada pantalla representa una porción del mapa y del juego en sí.

Mediante comunicaciones todos los equipos o pantallas, quedan interconectados entre sí permitiendo a los jugadores poder desplazarse por la totalidad de las pantallas e interactuar con objetos representados tanto en la misma pantalla como en cualquier otra pantalla que conforme el video wall.

Por otro lado, a fin de interactuar con la parte visual del videojuego. Se desarrolla paralelamente una aplicación móvil que sirve como mando a distancia del juego, en esta aplicación se representa por una parte, un menú del juego, donde poder realizar las configuraciones pertinentes, permitir la conexión con la matriz de pantallas y por último, una pantalla con los controles del juego para poder manejar la nave y realizar las diferentes acciones configuradas.

1.2 - PROPUESTA

A fin de llevar a cabo el proyecto, se propone lo siguiente:

El desarrollo de una aplicación que instalada en cada equipo que conforma la matriz de pantallas se encarga de las comunicaciones entre pantallas y móviles, la gestión de la lógica del juego (físicas, reglas del juego, etc...) y de visualizar la porción de juego que le corresponde según su ubicación en la matriz de pantallas.

El desarrollo de un aplicativo móvil que servirá de controlador del juego que permite la interacción del usuario con las pantallas y los controles de juego.

Por último, el desarrollo de una aplicación paralela a la aplicación principal encargada de gestionar los sonidos del juego.

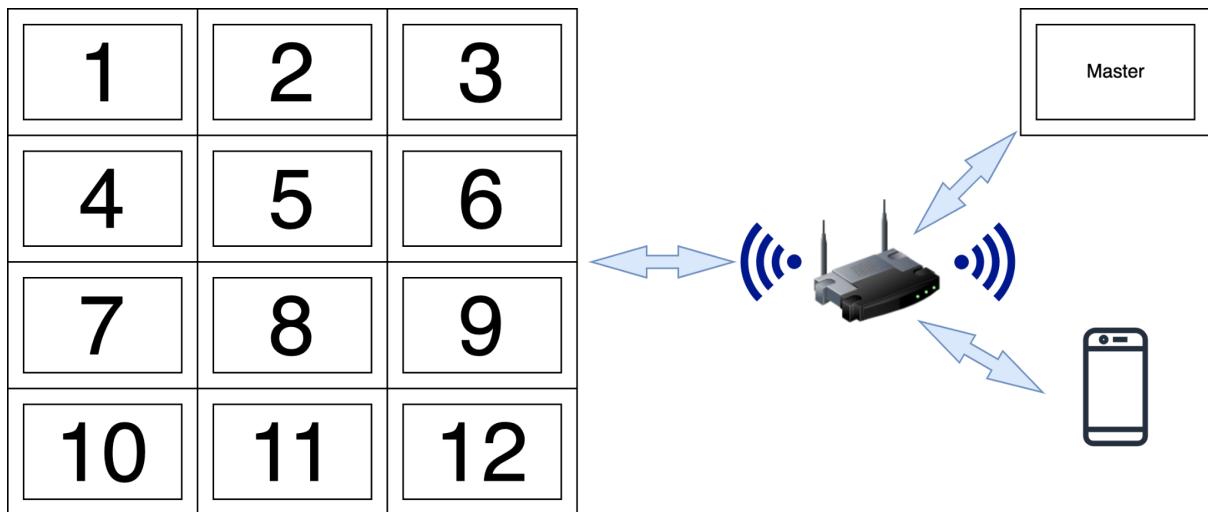


Figura 1.1 : Diagrama básico del proyecto

1.3 - ESTRUCTURA DEL DOCUMENTO

La estructura del resto de documento es la siguiente:

- Capítulo 2: se describe la metodología usada y la arquitectura global del proyecto.
- Capítulo 3: se realiza una planificación y descripción de todos los subproyectos involucrados.
- Capítulo 4: se realiza una descripción detallada del subproyecto individual.
- Capítulo 5: se describe la planificación de las tareas del subproyecto individual.
- Capítulo 6: se detallan las dedicaciones y tareas realizadas.
- Capítulo 7: se describe el entorno tecnológico usado y las herramientas usadas para el desarrollo.
- Capítulo 8: se describe la arquitectura de todos los subproyectos realizados.
- Capítulo 9: se describen todos los módulos desarrollados.
- Capítulo 10: se indican los manuales de usuario y guías de instalación.
- Capítulo 11: conclusiones finales sobre el proyecto global y subproyectos.

2 - DESCRIPCIÓN DE LA ARQUITECTURA GLOBAL DEL PROYECTO

2.1 METODOLOGÍA

La metodología de desarrollo de software se define como un marco de trabajo en el cual se planifica, se estructura y se controla todo el proceso de desarrollo de software.

El uso de una metodología de desarrollo de software resulta imprescindible para poder tener una correcta organización, poder controlar todas las fases del desarrollo y al tener una guía procedural podemos gestionar mejor los recursos disponibles, así como por consecuencia tener una mayor eficiencia del tiempo invertido, convirtiéndose en un ahorro de tiempo y costes.

Además, al seguir todo un proceso se obtiene un resultado final de mejor calidad.

Existen diferentes modalidades y cada una de ellas se organiza y funciona de manera distinta, todo ello para disminuir la tasa de fallos y conseguir un resultado final adecuado. No obstante, cada una de ellas está más indicada según el tipo de desarrollo de software que se lleve a cabo. Por eso es importante elegir la metodología adecuada según el desarrollo de cara a tener un trabajo más fluido y un resultado final acorde.

Las metodologías se dividen en dos grandes grupos:

-Metodologías tradicionales: son aquellas que tienen una estructura bien marcada, se planifica la totalidad del trabajo y una vez está completamente detallado se comienza el desarrollo.

-Metodología ágil: son aquellas que trabajan de manera más flexible y con menos documentación que las metodologías tradicionales. La metodología ágil busca proporcionar en poco tiempo pequeñas piezas de software en funcionamiento para mejorar la satisfacción del cliente. Por otro lado, al ser más flexible acepta cambios constantes que puedan surgir durante las diferentes etapas por tal de adaptarse a los cambios de manera efectiva.

2.2 FASES DEL DESARROLLO DEL SOFTWARE

1 - ESTUDIO DE VIABILIDAD

En esta primera fase del desarrollo de software se analiza el conjunto de necesidades del software, los criterios que se tienen en cuenta son tácticos, relacionados con aspectos económicos, técnicos, legales y operativos.

En esta primera fase, los resultados del estudio de viabilidad serán la base para tomar la decisión de seguir adelante o abandonar el proyecto.

2 - ANÁLISIS

En esta fase se determina cuáles serán las necesidades y los objetivos que tiene que cumplir el proyecto. Reunir todos estos requisitos que se deben cumplir para el desarrollo del software para llevar a cabo todo el proceso y cumplir con los objetivos finales.

De esta última parte se obtiene una memoria de especificación de requisitos, que contiene una especificación completa de todo lo que debe de hacer el programa.

En esta etapa es muy importante consensuar todo lo que se requiere que haga el software y será aquello que se seguirá en todas las etapas. Ya sea para un cliente final o para un desarrollo propio o interno, ya que no una vez iniciado el proceso no se podrán requerir nuevos resultados.

3 - DISEÑO

En esta fase se define la organización de la estructura y de todos los elementos necesarios para el desarrollo de software en base a las exigencias definidas en la fase anterior.

Se diseña la arquitectura del programa, así como un diseño detallado del mismo, con todos los componentes concretos e interfaces necesarias y cómo se relacionan cada uno de ellos entre sí para que funcionen de manera correcta.

Seguidamente, en esta fase se realizan los algoritmos necesarios para el cumplimiento de requisitos y también se realizan los análisis necesarios para saber qué herramientas usar en la etapa de codificación.

4 - IMPLEMENTACIÓN

Todo el diseño y arquitectura provenientes de la fase anterior se tiene que codificar en el lenguaje de programación elegido.

Los componentes del software se desarrollan por separado, se buscan los errores y se realizan pruebas unitarias. Después se ensamblan para componer el programa final, este se encontraría en su versión “alfa”. La primera versión del programa final para poder acceder a la siguiente etapa.

5 - VERIFICACIÓN Y ACEPTACIÓN

Partiendo de la primera versión alfa del software, en esta etapa se debe probar y ejecutar el código final para verificar que todo funciona correctamente y comparar los resultados finales con los requisitos iniciales para comprobar que cumple con todos ellos.

Una vez se ha comprobado que todo funciona correctamente y cumple con todos los requisitos el software estará listo para su entrega al cliente final y su lanzamiento.

6 - MANTENIMIENTO

Para finalizar, en esta última etapa se realiza el mantenimiento y mejora continua del software.

En caso de que se implemente alguna mejora es probable que se tenga que volver a la fase diseño para comprobar que se adapta a los cambios solicitados.

Es esencial mantener el programa constantemente actualizado para que siga siendo relevante.

2.3 METODOLOGÍA SCRUM

Durante la realización de este proyecto se hace uso de la metodología ágil scrum, que consiste en dividir el trabajo en períodos cortos llamados "sprints", generalmente de 2 a 4 semanas. El equipo selecciona las tareas que se compromete a completar durante cada sprint.

Durante el sprint, el equipo tiene reuniones diarias rápidas para mantenerse al tanto del progreso y resolver cualquier problema. Al final del sprint, se muestra el trabajo realizado al cliente y se recibe su retroalimentación.

Después de cada sprint, el equipo se reúne para analizar cómo fue el proceso y buscar formas de mejorarlo. Scrum se enfoca en la colaboración y la adaptación continua.

El Product Owner es responsable de definir y priorizar el trabajo, y el Scrum Master ayuda al equipo a seguir el proceso y a superar los obstáculos.

En resumen, Scrum es un enfoque ágil que divide el trabajo en sprints, tiene reuniones diarias y busca la mejora continua del equipo. Se centra en la colaboración, la adaptación y la entrega de valor al cliente.

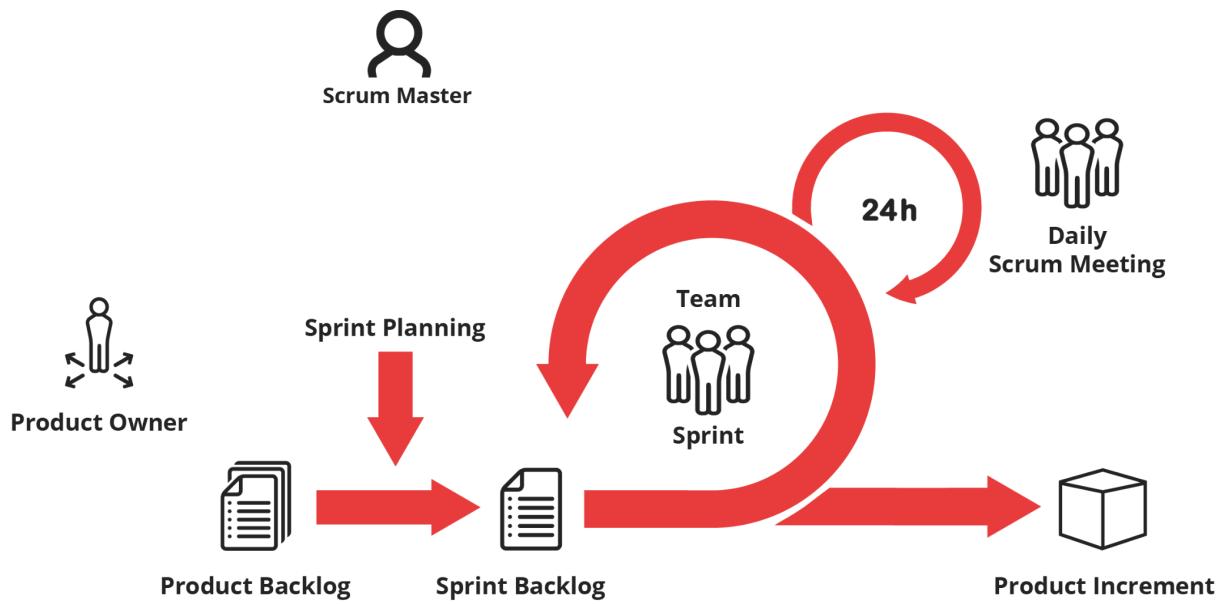


Figura 2.1: Ilustración desarrollo metodología scrum

2.4 - PLANIFICACIÓN DEL PROYECTO

Con el fin de llevar un control exhaustivo de las fases del proyecto, hitos y fechas límite, se realiza un diagrama de gantt que servirá de guía inicial sobre las fechas en las que se llevará a cabo cada fase del proyecto .

Figura 2.2: Diagrama de Gantt. Fases 1, 2 y 3.

Figura 2.3: Diagrama de Gantt. Fases 4 y 5.

2.5 - ANÁLISIS

En este capítulo, se identifican las partes interesadas de la aplicación, que se corresponden a los actores que utilizarán la aplicación, además se especifica cuál es su función dentro de esta. Se definen los requisitos de la aplicación, que hacen referencia a las funcionalidades que debe tener la misma. Seguidamente se llevan a cabo las pruebas de aceptación de requisitos donde se comprueba que el sistema cumple los requisitos definidos inicialmente.

2.5.1 - PARTES INTERESADAS

Los actores que harán uso de la aplicación son :

- Cliente / Jugador : interactúa con la aplicación para ejecutar el juego.

2.5.2 - REQUISITOS DE LA APLICACIÓN

Para determinar que deben poder hacer los usuarios de la aplicación y las características de esta, hay que definir los requisitos funcionales, no funcionales y de sistema.

2.5.2.1 - REQUISITOS FUNCIONALES

Las funcionalidades que deben tener las partes interesadas son :

- CLIENTE

- RF_Cliente_01: Conectarse con la matriz de pantallas
- RF_Cliente_02: Editar las propiedades del juego
- RF_Cliente_03: Elegir diferentes escenarios de juego
- RF_Cliente_04: Elegir tipo de nave
- RF_Cliente_05: Visualizar estadísticas del juego
- RF_Cliente_06: Jugar al video juego
- RF_Cliente_07: Debe tener modo de juego - Duelo por equipos
- RF_Cliente_08: Debe tener máximo 2 equipos
- RF_Cliente_09: Máximo 8 jugadores en total
- RF_Cliente_10: Asignación aleatoria de equipos
- RF_Cliente_11: Posibilidad de disparar
- RF_Cliente_12: Límite tiempo de partida
- RF_Cliente_13: Objetos estáticos interactuables

2.5.2.2 - REQUISITOS NO FUNCIONALES

Los requisitos no funcionales se corresponden a :

- RnF_01 : El sistema debe ser confiable y seguro
- RnF_02 : La aplicación debe admitir varios usuarios
- RnF_03 : La aplicación debe ofrecer transparencia del proceso a los diferentes tipos de usuarios
- RnF_04 : La aplicación tiene que emitir sonidos para interactuar con el jugador.
- RnF_05 : El desarrollo de la aplicación debe llevarse a cabo mediante el uso de Java.

2.5.2.3 - REQUISITOS DE SISTEMA

Los requisitos de sistema de la aplicación son los siguientes :

- RS_01 : La aplicación debe ser adaptable a cualquier dispositivo / plataforma

2.5.3 - REGLAS DEL JUEGO

1. Modo de juego: Duelo por equipos.
2. 2 equipos.
3. Máximo 8 jugadores (4 en cada equipo).
4. Asignación aleatoria de equipo (si los jugadores son pares se realiza asignación aleatoria de equipo, si los jugadores son impares se asigna al equipo que menos jugadores tiene).
5. No hay fuego amigo.
6. Se establece un límite de tiempo de partida, gana el equipo con más bajas y en caso de las mismas bajas se declara empate.
7. 1 arma por nave (con posibilidad de ampliación a varias armas).
8. Movimiento de las naves tipo “tanque” (rotación 360° sobre el mismo eje y avance, no hay botón de frenado).
9. Se declara un máximo de vida por nave por el valor de “100”.
10. Posibilidad de configurar el nivel de daño del arma.
11. Las naves pueden chocar pero no restan vida.
12. Existen objetos estáticos contra los que las naves pueden chocar (asteroides, planetas, etc...).
13. 3 escenarios diferentes de juego.

2.5.4 - PRUEBAS DE ACEPTACIÓN DE REQUISITOS

Las pruebas de aceptación pertenecen a una de las últimas etapas previas a la implementación del software, es muy importante realizar las pruebas correspondientes en cada etapa de desarrollo ya que un error no detectado al inicio del desarrollo del proyecto puede necesitar cincuenta veces más esfuerzos para ser solucionado que si es detectado a tiempo.

2.5.4.1 - OBJETIVOS DE LAS PRUEBAS DE ACEPTACIÓN DE REQUISITOS

El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario / cliente de dicho sistema que determine su aceptación, desde el punto de vista de la funcionalidad y rendimiento.

2.5.4.2 - DESCRIPCIÓN DE LAS PRUEBAS

Las pruebas de aceptación son definidas por el usuario / cliente y preparadas por el equipo de desarrollo.

Estas pruebas van dirigidas a comprobar que el sistema cumple los requisitos de funcionamiento, recogidos en el catálogo de requisitos. Para así conseguir la aceptación final del sistema por parte del usuario / cliente

Para ello, en primer lugar se realiza una reunión inicial con el cliente para tener una primera aportación sobre los requisitos iniciales del software. Una vez obtenidos los requisitos iniciales, estos son contrastados por el equipo de desarrollo y mediante reuniones posteriores con el cliente se hacen aproximaciones para verificar los requisitos obtenidos.

Durante la fase de pruebas de aceptación se realizan tres reuniones de aproximación en la que se han definido en cada reunión los requisitos a cumplir.

Finalmente, se realiza una reunión final con el cliente, con ya una aproximación final a los objetivos que debe realizar la aplicación en la que el cliente acepta los requisitos funcionales, no funcionales y de sistema la aplicación a desarrollar.

Este proceso de reunión-aproximación queda plasmado en el siguiente gráfico:

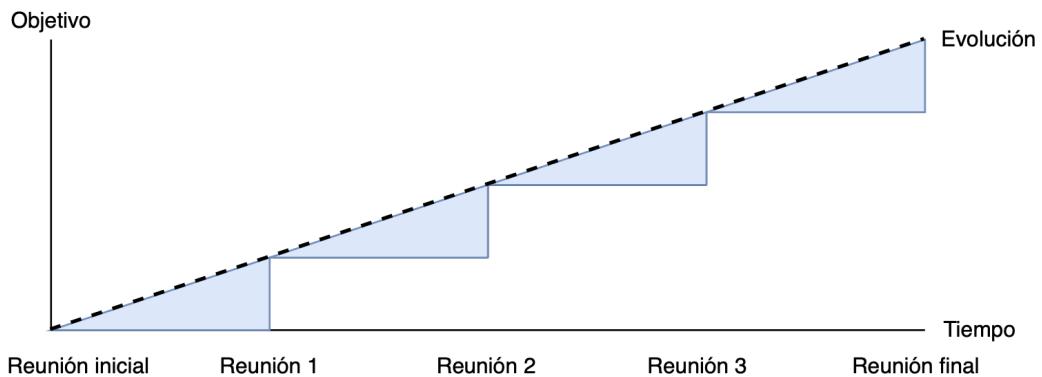


Figura 2.4 : Gráfico proceso de reunión-aproximación

2.5.5 - DISEÑO

En este apartado se procede a diseñar, de forma conceptual, la aplicación que permite jugar al juego killer game. En el capítulo apartado se han especificado los requisitos de la aplicación y tras este análisis se procede a diseñar la aplicación.

2.5.5.1 - MODELO CONCEPTUAL DEL PROYECTO

El diseño de la aplicación sigue la estructura marcada por los requisitos funcionales y no funcionales definidos en el capítulo anterior, se opta por hacer uso de 4 elementos principales que darán soporte completo a la aplicación, un conjunto de punto de acceso wifi + switch ethernet para asegurar las comunicaciones entre equipos y móviles, un equipo master que lanza un script que al ser ejecutado en el equipo master ejecuta en todos los equipos que conforman el videowall la aplicación principal del juego y un programa java que controla el sonido del juego, una aplicación que se ejecuta en cada equipo del videowall que se encarga de la lógica del juego, de la visualización del juego y de las comunicaciones y por último, una aplicación móvil que permite interactuar con los equipos del videowall, realizar configuraciones y servir de mando de juego.

Se entiende que con estos 4 elementos se da soporte completo y son suficientes para realizar una aplicación funcional que permite cumplir con todos los requisitos funcionales y no funcionales

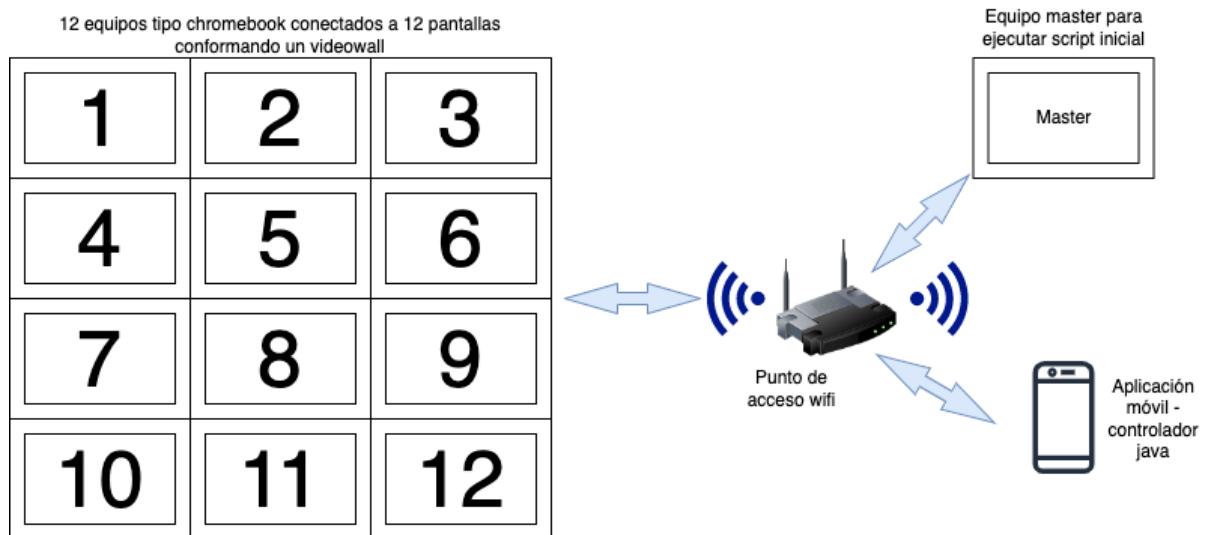


Figura 2.5 : Modelo conceptual del proyecto

2.5.5.2 - DIAGRAMA DE CLASES

En este punto se muestran los diagramas de clases de los subproyectos derivados del proyecto principal de manera esquemática.

2.5.5.2.1 - DIAGRAMA DE CLASES GENERAL APLICACIÓN PRINCIPAL

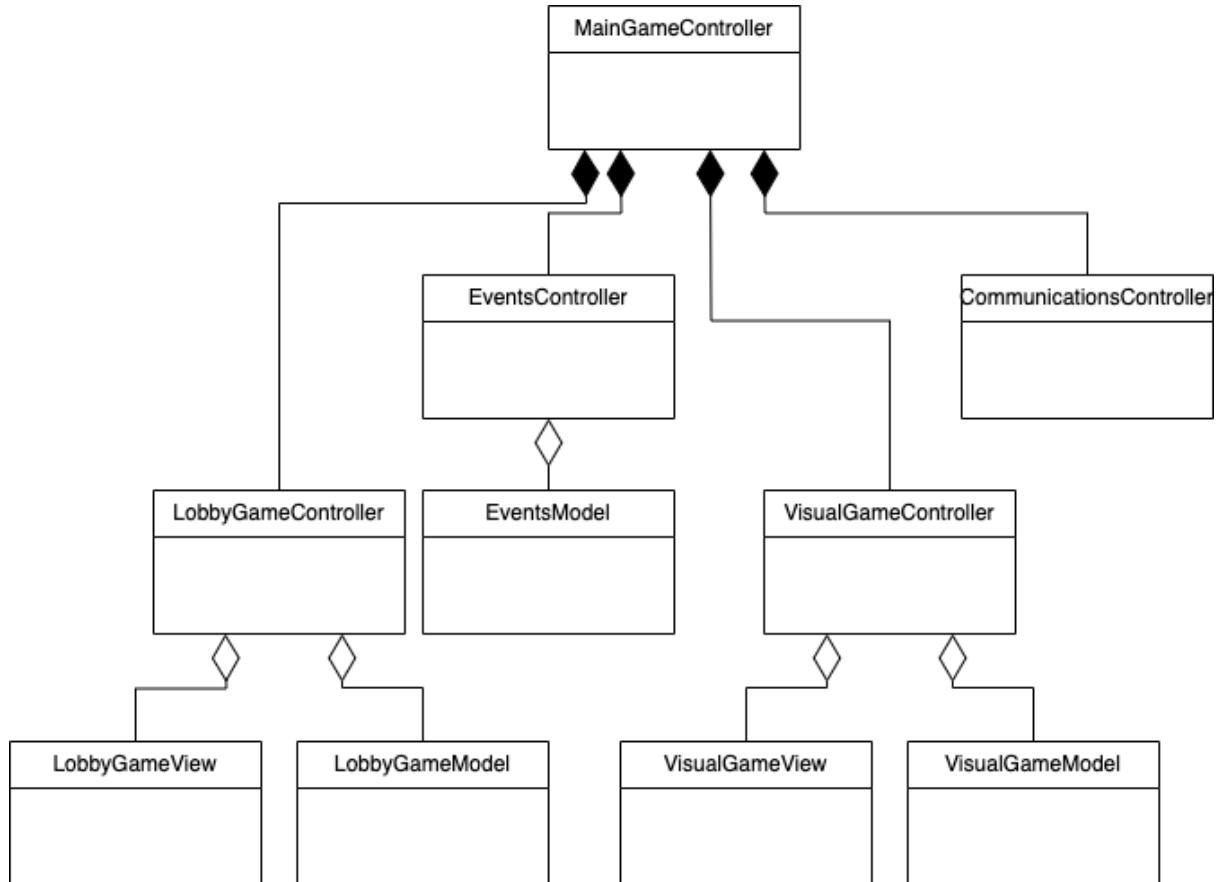


Figura 2.6 : Diagrama de clases general aplicación principal

2.5.5.2.2 - DIAGRAMA DE CLASES GENERAL APLICACIÓN MÓVIL

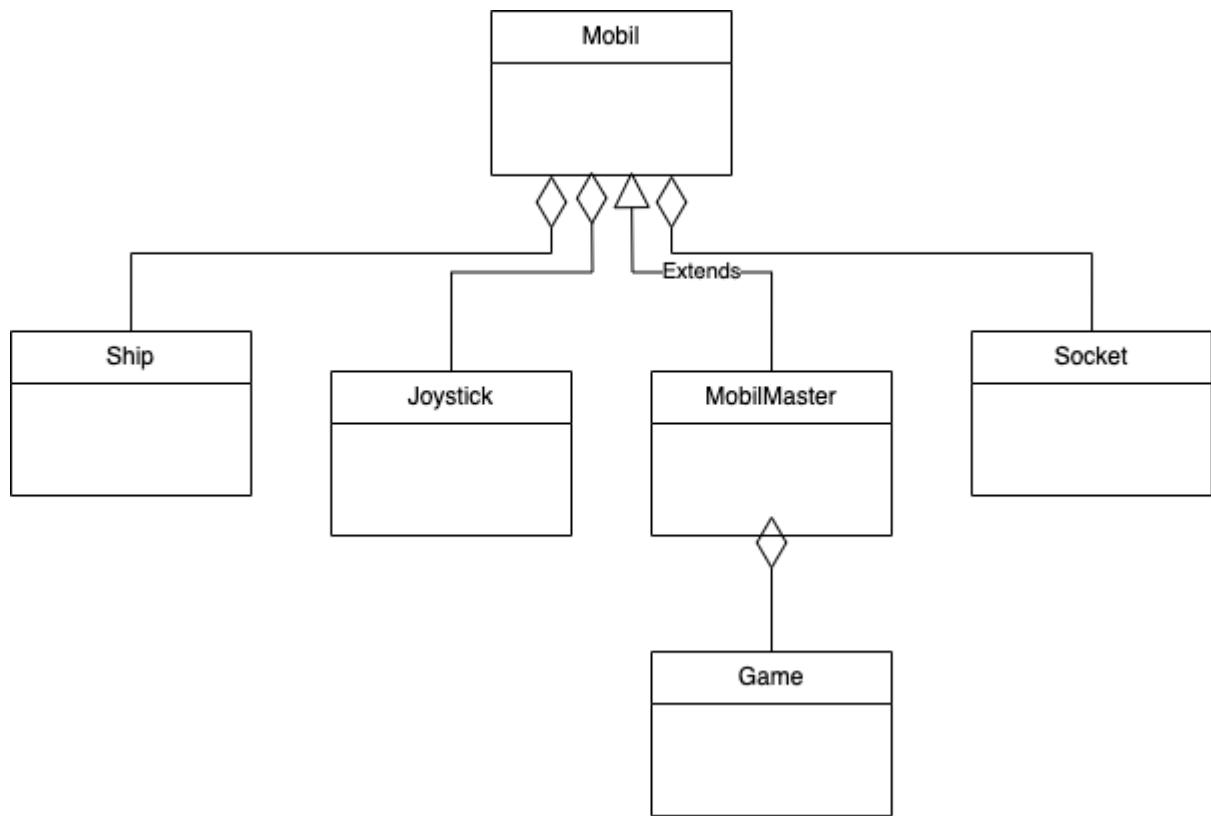


Figura 2.7 : Diagrama de clases general aplicación móvil

2.5.5.2.3- DIAGRAMA DE CLASES GENERAL APLICACIÓN SONIDO

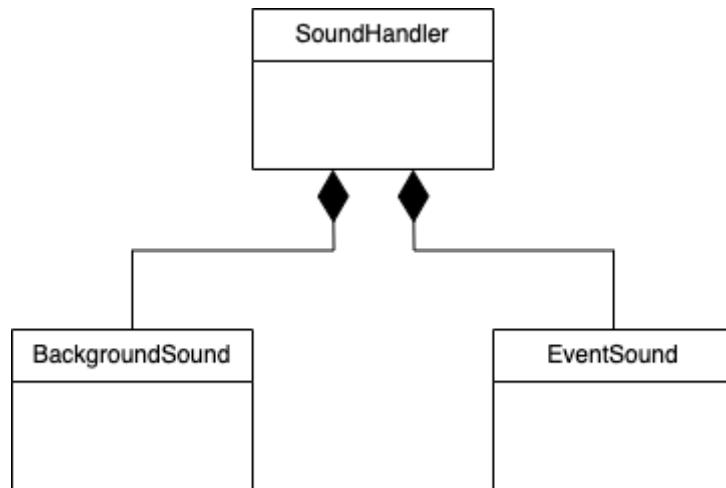


Figura 2.8 : Diagrama de clases general aplicación sonido

3 - PLANIFICACIÓN Y DESCRIPCIÓN DE SUBPROYECTOS

En este capítulo se describe la planificación y descripción de todos los subproyectos relacionados con el desarrollo global del proyecto. También por otra parte, se da forma a la estructura jerárquica a seguir a fin de definir correctamente los roles involucrados en el desarrollo así como su función en cada punto de este.

3.1 SUBPROYECTOS

A fin de cumplir con todos los objetivos y requerimientos funcionales del proyecto se han establecido 7 subproyectos.

3.1.1 Coordinación

En este proyecto se realiza la coordinación general del proyecto global, asegurando una correcta comunicación entre todos los integrantes del equipo, resolución de dudas, seguimiento de fechas límites, calidad del proyecto y la consecución del objetivo final. Este subproyecto vendría a estar englobado en la figura de “Scrum Master” dentro del marco de la metodología ágil scrum.

Adicionalmente, cada subproyecto cuenta con un coordinador que se encarga de coordinar su subproyecto asignado y tener una correcta interlocución vertical con el “Scrum Master”.

3.1.2 Main Controller / Integrador

Este subproyecto consiste en elaborar un controlador principal que sirve de nexo de comunicación entre las distintas partes del proyecto principal, la configuración del juego, el controlador de eventos y la lógica de juego. A su vez, también se encarga de la correcta integración de las distintas partes del proyecto.

3.1.3 Visual

Este subproyecto consiste en representar de manera visual todos los elementos del juego, actúa de interfaz gráfica y gestiona la capacidad de mostrar en cada monitor el campo de juego, estadísticas, etc...

Además se encarga de la incorporación y gestión de las físicas del juego para asegurar una experiencia de juego satisfactoria.

3.1.4 Comunicaciones

Este subproyecto consiste en implementar todos los métodos necesarios para asegurar las comunicaciones tanto entre los diferentes equipos que conforman el cluster de juego principal, así como las comunicaciones entrantes y salientes de los dispositivos móviles que actuarán de mando de juego.

3.1.5 Aplicación móvil

Este subproyecto desarrolla una aplicación móvil completa para servir de interfaz entre el jugador y el cluster principal. Esta aplicación cuenta con la posibilidad de personalizar aspectos del juego así como de proporcionar al jugador un mando de juego completo.

3.1.6 Sonido

Este subproyecto gestiona mediante una aplicación externa todo lo referente a los sonidos del juego, actúa para reproducir tanto sonido ambiental del juego así como de sonidos específicos para cada evento del juego, como pueden ser disparos o explosiones.

3.1.7 Cluster

Este subproyecto se encarga del montaje físico del videowall y cluster de equipos, así como toda la arquitectura de hardware necesaria para el correcto funcionamiento del entorno de juego.

3.2 Asignación de tareas / roles por subtareas y por colaborador

En este punto se especifica mediante una tabla los diferentes subproyectos que componen el proyecto global así como los responsables asignados a cada subproyecto y tarea dentro de este subproyecto.

Coordinación / Scrum Master	Cosme Torandell
Móvil	
Responsabilidad	Nombre
Coordinador	Erika Lisbeth
Parte partida	Àngel Barceló
Parte configuración	Erika Lisbeth
Visual	
Responsabilidad	Nombre
Coordinador	Carlos Blanco
Parte Visual Chromebooks / Testing - Rendimiento	Marc Barceló
Parte Visual Chromebooks / Fondo y coordinación background	Carlos Rubio
Parte Visual Chromebooks	Robert
Parte Visual Chromebooks / Físicas	Melissa
Parte Físicas	Carlos Blanco
Parte Visual Chromebooks / Representación objetos estáticos	Marcos Nazco
Main Controller / Integrador	
Responsabilidad	Nombre
Coordinador	Marcos Lopez
Controller config game	Zhiyun
Events controller	Juan Sanchez

Controller config game	Antoni X Bascuñana
Sonido	
Responsabilidad	Nombre
Coordinador	Marcos Lopez
Sonido	Sergio Torres
Cluster	
Responsabilidad	Nombre
Coordinador	Josep Faios Suau
Comunicaciones	
Responsabilidad	Nombre
Coordinador	Miquel Andreu
Comunicaciones Chromebook	Miquel Andreu
Comunicaciones móvil	Karina

Cuadro 3.1 : Tabla de asignación de subproyectos / responsabilidad

4 - DESCRIPCIÓN DETALLADA DEL SUBPROYECTO INDIVIDUAL

En este punto se describe detalladamente el subproyecto individual asignado sobre el proyecto global, así como los puntos claves relacionados con el proyecto individual.

4.2 - ROL ASIGNADO

Inicialmente, asumí el rol de desarrollo en el proyecto y me encargué de crear varias clases. Sin embargo, más adelante, se realizaron cambios en la asignación de tareas y esas clases fueron reasignadas a otros compañeros, ya que a mí se me asignaron otras clases.

4.2.1 - DESARROLLO

La tarea de un desarrollador es la creación, implementación y mantenimiento de software y aplicaciones informáticas. Su función principal es convertir las ideas y requisitos del cliente o del equipo de desarrollo en código de programación funcional. Además de depurar y mejorar el código.

Las responsabilidades del rol de desarrollador incluyen los siguientes puntos:

1. Análisis y diseño: Comprender los requisitos del proyecto y diseñar soluciones técnicas efectivas.
2. Codificación: Escribir código limpio y eficiente utilizando el lenguaje de programación adecuado y siguiendo las mejores prácticas de desarrollo.
3. Pruebas y depuración: Realizar pruebas exhaustivas para identificar y corregir errores (bugs) en el código y garantizar que el software funcione correctamente.
4. Mantenimiento y mejoras: Realizar actualizaciones, correcciones de errores y mejoras en el software existente para mantenerlo en buen funcionamiento y mejorar su rendimiento.
5. Colaboración en equipo: Trabajar en estrecha colaboración con otros miembros del equipo, como diseñadores, analistas y probadores, para garantizar una implementación exitosa del proyecto.
6. Documentación: Crear documentación clara y completa del código desarrollado, incluyendo comentarios en el código, para facilitar su comprensión y mantenimiento.
7. Investigación y aprendizaje: Mantenerse actualizado con las últimas tecnologías y tendencias en el campo del desarrollo de software, investigar nuevas soluciones y aprender nuevas habilidades para mejorar continuamente como desarrollador.

4.2.2 - RESPONSABILIDADES ADICIONALES

En mi caso tuve la responsabilidad exclusiva de hacer que funcionase la clase de Animation totalmente solo, con lo cual era importante conseguir que se pudiesen llevar a cabo todas las animaciones ya que gracias a ese aspecto visual, daría una mejor imagen de cara al proyecto y jugabilidad.

5 - PLANIFICACIÓN DE LAS TAREAS DEL SUBPROYECTO INDIVIDUAL

5.1 - INTRODUCCIÓN

En este apartado se especifica la planificación de las tareas del subproyecto individual plasmadas en un diagrama de gantt.

Al plasmar una planificación en un diagrama de gantt se obtiene una visión general de los tiempos, tareas e hitos a lograr.

De este modo se pueden detectar rápidamente desviaciones en el tiempo de ejecución del proyecto y aplicar medidas correctivas para asegurar la entrega final.

5.2 - DIAGRAMA DE GANTT

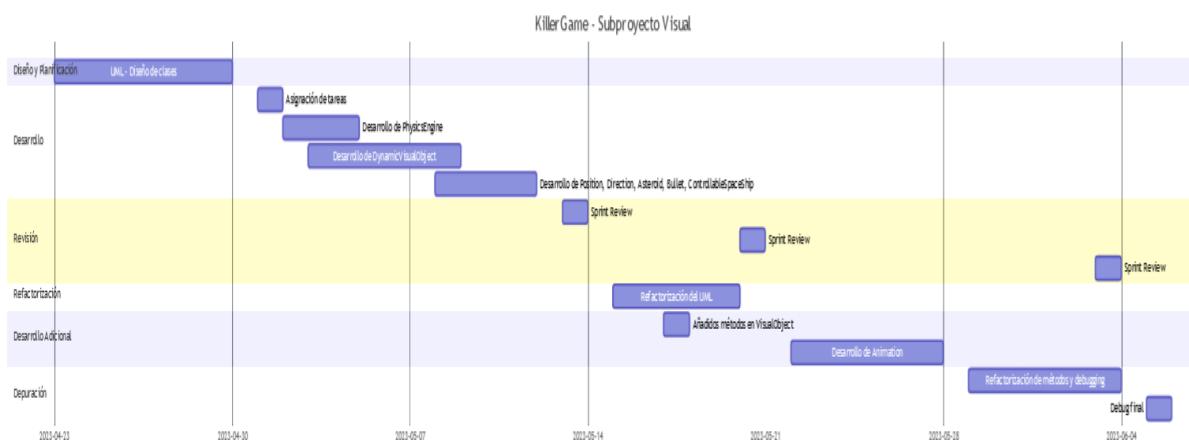


Figura 5.1 : Diagrama de Gantt subproyecto Visual, Fases

6 - DETALLE DE LAS DEDICACIONES Y TAREAS REALIZADAS

En este punto se detallan y posteriormente describen todas las tareas realizadas en el.

6.1 - DISEÑO DE LAS CLASES

En el diseño de las clases debemos definir las estructuras y comportamientos de los objetos del sistema. Para ello, identificamos las entidades clave y creamos las clases correspondientes. Establecemos las relaciones entre ellas, definimos los atributos y métodos necesarios, organizamos la estructura del sistema y validamos el diseño. Este proceso es fundamental para el desarrollo de software, ya que sienta las bases para la implementación y nos asegura la reutilización del código, la mantenibilidad y la prevención de errores.

6.2 - ASIGNACIÓN DE TAREAS

En este punto, el objetivo principal es la de elaborar una correcta asignación de las clases correspondientes para cada uno del subgrupo de Visual.

Se ha considerado en primer lugar una distribución justa de las clases para asegurar una carga de trabajo equilibrada en todos los integrantes del grupo de Visual. Por último, con el objetivo de asegurar una ejecución exitosa del proyecto y un entorno de trabajo favorable, se han asignado a los colaboradores de acuerdo a sus habilidades específicas, así como teniendo en cuenta sus preferencias y afinidades.

6.3 - DESARROLLO DE LAS CLASES

Además de haber diseñado las clases conforme al UML, hemos logrado crear una estructura coherente y organizada para nuestro proyecto. Este enfoque nos ha permitido visualizar claramente las relaciones entre las clases y comprender cómo interactúan entre sí. Al seguir los principios de diseño del UML, como la encapsulación, la herencia y la composición, hemos establecido una base sólida para el desarrollo de software.

Asimismo, al diseñar las clases de manera ordenada, hemos facilitado la comprensión y mantenibilidad del código. Cada clase tiene una responsabilidad claramente definida, lo que nos ayuda a mantener un código limpio y modular. Además, al utilizar el UML, hemos documentado adecuadamente las clases, sus atributos y métodos, lo que será de gran utilidad para futuras referencias y modificaciones del proyecto.

6.4 - SPRINTS REVIEWS SEMANALES

A fin de dar seguimiento a todas las tareas definidas, cada semana se establece un dia (en el caso del proyecto, todos los Sábados) en el que se hace una revisión del trabajo realizado durante la semana.

El objetivo principal es verificar personalmente el progreso de la tarea asignada, identificar cualquier obstáculo que esté impidiendo su avance, proponer soluciones y alternativas para resolver los problemas que surjan, y finalmente establecer nuevas tareas que sean necesarias.

7 - DESCRIPCIÓN DEL ENTORNO TECNOLÓGICO Y HERRAMIENTAS USADAS

En este apartado se describen todas las herramientas y tecnologías utilizadas en la fase de implementación.

7.1 - VISUAL STUDIO CODE v1.67.1

Visual Studio Code es un editor de código fuente desarrollado por Microsoft, compatible con Windows, Linux, macOS.

Incluye soporte para depuración, resaltado de sintaxis, finalización inteligente de código (texto predictivo) y refactorización de código.

La gran ventaja de este editor de código es que es personalizable por los usuarios y compatible con gran cantidad de extensiones, esto permite que sea extremadamente versátil. Gracias a las extensiones es compatible con un gran número de lenguajes.

7.2 - ECLIPSE IDE v2022-23 (4.23.0)

Eclipse es un entorno de desarrollo de software multi-plataforma y el más usado para el desarrollo en Java.

Es de código abierto y gratuito.

Eclipse permite extender sus funciones mediante el desarrollo de plugins además de proporcionar herramientas para la gestión de espacios de trabajo, escribir, desplegar, ejecutar y depurar aplicaciones.

7.3 - ANDROID STUDIO 2021.3.1 PATCH 1

Android Studio es un entorno de desarrollo integrado (IDE) utilizado para crear aplicaciones móviles en la plataforma Android. Es la herramienta principal para los desarrolladores de Android, ya que proporciona una interfaz completa y eficiente para escribir, depurar y probar aplicaciones.

Android Studio está basado en el popular IDE IntelliJ IDEA de JetBrains y está respaldado por Google. Proporciona una amplia gama de características y herramientas que facilitan el desarrollo de aplicaciones para dispositivos Android.

7.4 - INTELLIJ IDEA 2022.3.1

IntelliJ IDEA es un entorno de desarrollo integrado (IDE) creado por JetBrains. Es una potente herramienta diseñada específicamente para el desarrollo de software en diferentes lenguajes de programación, como Java, Kotlin, Groovy, Scala y más.

IntelliJ IDEA ofrece una amplia gama de características y funcionalidades que aumentan la productividad de los desarrolladores.

7.5 - ATlassian Jira Cloud

Atlassian Jira es una plataforma de gestión de proyectos y seguimiento de problemas diseñada para equipos de desarrollo de software. Proporciona a los equipos una forma flexible de planificar, rastrear y colaborar en el desarrollo de software, así como en la gestión de proyectos en general.

Jira se centra en la metodología ágil y utiliza tableros Kanban y Scrum para facilitar la organización y el seguimiento de tareas. Los equipos pueden crear proyectos, definir tareas y asignarlas a miembros del equipo, establecer fechas límite y prioridades, y hacer un seguimiento del progreso en tiempo real.

7.6 - GITHUB

GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git.

Se utiliza principalmente para la creación de código fuente de programas informáticos. La plataforma está creada para que los desarrolladores suban el código de sus aplicaciones y herramientas, y que como usuario puedas descargarla la aplicación y también entrar en el perfil para leer sobre dicha aplicación o colaborar en su desarrollo.

Mediante un sistema de gestión de versiones los desarrolladores pueden administrar su proyecto, ordenando el código de cada una de las nuevas versiones que sacan de las aplicaciones para evitar confusiones. Así, al tener copias de cada una de las versiones de su aplicación, no se perderán los estados anteriores cuando se vaya a actualizar.

7.7 GIT

A fin de obtener una estructura lógica de colaboración mediante GitHub se ha creado una organización llamada “killergameorg” en la que dentro de ella, se han creado 3 repositorios independientes.

7.7.1 - ESTRUCTURA DE REPOSITORIOS

Cada uno de estos repositorios atiende a un subproyecto del proyecto global y contiene el código fuente de las aplicaciones desarrolladas.

La estructura es la siguiente:

- KillerGame: En el siguiente [enlace](#) se puede acceder a este repositorio de GitHub donde está alojado el subproyecto que se encarga de la lógica del juego, de la visualización y de las comunicaciones del proyecto.
- KillerGameMobile: En el siguiente [enlace](#) se puede acceder a este repositorio de GitHub donde está alojado el subproyecto que se encarga del aplicativo móvil android que sirve de mando a distancia para interactuar con el juego.
- KillerGameSound: En el siguiente [enlace](#) se puede acceder a este repositorio de GitHub donde está alojado el subproyecto que se encarga del aplicativo que controla los sonidos del juego.

7.7.2 - ESTRUCTURA DE RAMAS

Adicionalmente, en cada uno de los repositorios se ha creado un esquema de ramas para permitir un desarrollo colaborativo ordenado y seguro.

La estructura de ramas es la siguiente:

- Main: Esta es la rama principal de la aplicación, esta rama contiene el código funcional y final de la aplicación.
- Rama subproyecto: Estas ramas son las dedicadas al desarrollo de cada subproyecto antes de ser finalmente subidas y juntadas con la rama principal.
En este caso, existe una rama para cada subproyecto, por lo tanto la estructura de ramas de subproyecto es la siguiente:
 - Visual
 - Communications
 - Lobby
 - Events
 - Main-controller
- Dev: Dentro de cada subrama de cada subproyecto se crea esta rama a fin de que cada colaborador pueda realizar su desarrollo sin interferir con la subrama de cada subproyecto. Una vez el colaborador ha finalizado una parte del código libre de errores y funcional se subirá el código a la rama principal de su subproyecto.

Mediante el uso de conventional commits y versionado semántico se han registrado cada una de las partes y actualizaciones de las aplicaciones hasta llegar al desarrollo completo de la aplicación. En la siguiente ilustración se refleja la estructura de ramas del subproyecto principal con la rama visual.

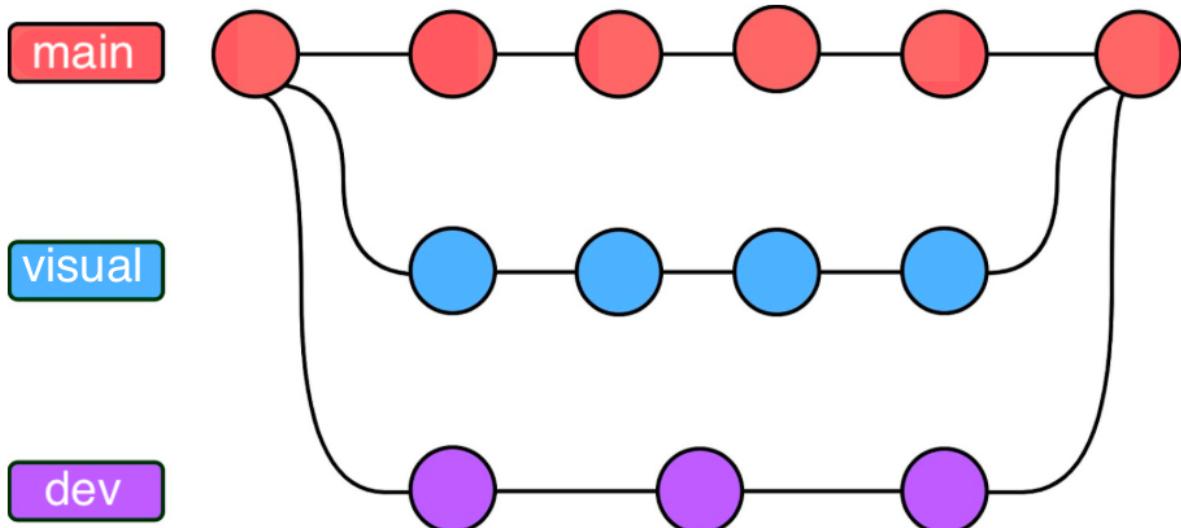


Figura 7.1 : Diagrama de ramas visual

7.7.3 - ESTRUCTURA DE PERMISOS

Con el fin de obtener una seguridad sobre los commits realizados en los repositorios se ha definido una estructura de permisos adecuada.

Para ello, se han creado 3 grupos de colaboradores clave con diferentes roles, cada uno de ellos tiene los permisos asignados para poder realizar un tipo de acciones predefinidas.

- **Directive**: Es el rol jerárquico más elevado, tiene los permisos totales de control sobre el repositorio y es el único que puede autorizar subidas de código a la rama principal. Este rol se le otorga al coordinador del equipo o scrum master.
- **Controllers**: Este rol tiene poder dentro de su rama de desarrollo de subproyecto, es la persona que autoriza a los colaboradores con rol “Devs” a subir código a la rama principal de su subproyecto. Este rol se le otorga a los coordinadores de cada subproyecto.
- **Devs**: Este rol tiene el poder usar la rama “dev” y poder subir el código en ella. No tiene permisos para subir código a ninguna otra rama y por tanto, necesita de aprobación de su coordinador asignado. Este rol se le otorga a los desarrolladores de cada subproyecto.

8 - DESCRIPCIÓN DE LA ARQUITECTURA DEL SUBPROYECTO

En esta sección se detalla la estructura y organización del subproyecto asignado a un nivel más detallado. Se incluyen los diagramas que representan visualmente la arquitectura del subproyecto, así como los módulos en los cuales se colabora.

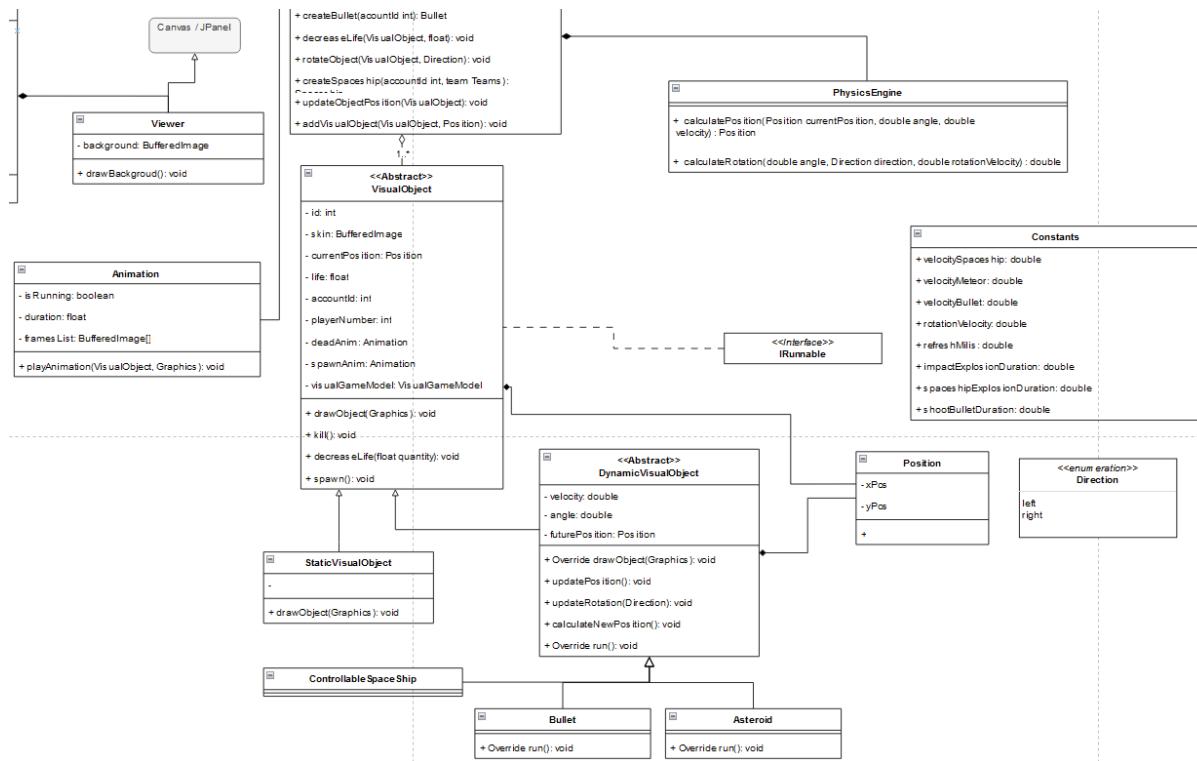


Figura 8.1 Arquitectura responsabilidades dev visual

9 - DESCRIPCIÓN DE LOS MÓDULOS O ÍTEMS DESARROLLADOS.

En este módulo, que forma parte del subproyecto de desarrollo en el área visual, se documenta el proceso de implementación de las clases que he diseñado específicamente para esta tarea.

9.1 - PHYSICS ENGYNE

La clase "PhysicsEngine" es parte del paquete "visual" y contiene métodos para realizar cálculos relacionados con la física en un entorno visual. Realizada a partir de la implementación que nos facilitó Melisa y conjunto a Marc hicimos esta clase.

```
1  package visual;
2
3  public class PhysicsEngine {
4
5      // * Methods
6
7      public Position calculatePosition(Position position, double angle, double velocity) {
8          return new Position(position.getxPos() + (velocity * Math.sin(angle)),
9                  position.getyPos() - (velocity * Math.cos(angle)));
10     }
11
12     public double setDirection(Direction direction, double angle, double rotationVelocity) {
13         double resAngle = 0;
14         if (direction == Direction.LEFT) {
15             resAngle = angle - rotationVelocity;
16         } else if (direction == Direction.RIGHT) {
17             resAngle = angle + rotationVelocity;
18         }
19         return resAngle;
20     }
21
22 }
```

1. Método "calculatePosition": Este método calcula y devuelve una nueva posición basada en una posición inicial, un ángulo y una velocidad. Toma como argumentos un objeto "Position" que representa la posición inicial, un valor de ángulo y un valor de velocidad. Utiliza fórmulas trigonométricas para calcular la nueva posición en función del ángulo y la velocidad. La nueva posición se crea y se retorna como un objeto "Position".
2. Método "setDirection": Este método establece la dirección de rotación y devuelve el nuevo ángulo resultante. Recibe como argumentos una dirección (representada por el objeto "Direction"), un ángulo inicial y una velocidad de rotación. Dependiendo de la dirección especificada (izquierda o derecha), el método ajusta el ángulo inicial sumándole o restándole la velocidad de rotación. El nuevo ángulo resultante se devuelve como un valor double.

9.2 - DYNAMIC VISUAL OBJECT

```
package visual;

import java.awt.geom.AffineTransform;
import java.awt.Graphics2D;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import maincontroller.gameinfo.Team;

public abstract class DynamicVisualObject extends VisualObject {

    private double velocity;
    private Position futurePosition;

    // * Constructor

    public DynamicVisualObject(int id, BufferedImage skin, Position position, float life, int accountId,
        VisualGameModel visualGameModel, int playerNumber, Animation deadAnim, Animation spawnAnim, Team team,
        double velocity, double angle) {
        super(id, skin, position, life, accountId, visualGameModel, playerNumber, deadAnim, spawnAnim, angle, team);
        this.velocity = velocity;
        this.futurePosition = null;
    }

    // * Getters & Setters

    public Position getFuturePosition() {
        return futurePosition;
    }

    public void setFuturePosition(Position futPos) {
        this.futurePosition = futPos;
    }

    public double getVelocity() {
        return this.velocity;
    }

    public void setVelocity(double velocity) {
        this.velocity = velocity;
    }
}
```

```

// * Methods

@Override
public void updatePosition() {
    setPosition(futurePosition);
    this.futurePosition = null;
}

@Override
public void updateRotation(Direction direction) {
    double angle = getVisualGameModel().getPhysicsEngine().setDirection(direction, getAngle(),
        VisualConstants.rotationVelocity);
    setAngle(angle);
}

@Override
public void calculateNewPosition() {
    this.futurePosition = getVisualGameModel().getPhysicsEngine().calculatePosition(getPosition(), getAngle(),
        getVelocity());
}

@Override
public void drawObject(Graphics g) {
    AffineTransform at = AffineTransform.getTranslateInstance(this.getPosition().getxPos(), this.getPosition().getyPos());
    at.rotate(Math.toRadians(this.getAngle()));

    Graphics2D g2d = (Graphics2D) g;

    g2d.drawImage(getSkin(), at, null);
}

@Override
public void run() {
    while (getIsAlive()) {
        if (futurePosition != null) {
            getVisualGameModel().notifyToVGC(new NotificationMsg(NotificationType.positionUpdate, this));
        }
    }
}

```

La clase "DynamicVisualObject" es una clase abstracta que extiende la clase "VisualObject" y pertenece al paquete "visual". Esta clase se utiliza para representar objetos visuales dinámicos en un entorno gráfico. A continuación se describe la implementación y los componentes clave de la clase:

1. Atributos: La clase tiene dos atributos privados: "velocity" (velocidad) y "futurePosition" (posición futura). Estos atributos representan la velocidad del objeto y la posición a la que se moverá en el futuro.
2. Constructor: Se proporciona un constructor que inicializa los atributos de la clase y recibe varios parámetros, como la identificación, la apariencia gráfica, la posición inicial, la vida, el número de cuenta, el modelo de juego visual, el número de jugador, las animaciones de muerte y aparición, el equipo, la velocidad y el ángulo inicial del objeto dinámico visual.
3. Métodos Getter y Setter: Se proporcionan métodos para obtener y establecer los valores de los atributos "futurePosition" y "velocity".
4. Métodos Override: La clase sobrescribe varios métodos de la clase padre "VisualObject". Estos métodos incluyen "updatePosition" (actualizar posición), "updateRotation" (actualizar rotación), "calculateNewPosition" (calcular nueva posición) y "drawObject" (dibujar objeto). Estos métodos implementan la lógica para

actualizar la posición y la rotación del objeto, calcular la nueva posición basada en la velocidad y el ángulo, y dibujar el objeto en el entorno gráfico.

5. Método "run": La clase implementa el método "run" de la interfaz "Runnable". Este método se ejecuta en un hilo separado y contiene un bucle que verifica si hay una posición futura establecida. Si hay una posición futura, se notifica al modelo de juego visual mediante una notificación.

9.3 - ANIMATION

```
package visual;

import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.util.concurrent.atomic.AtomicBoolean;

public class Animation {

    public Boolean isRunning;
    public double durationSleep;
    public BufferedImage[] framesList;

    // * Constructor
    public Animation(Boolean isRunning, double durationSleep, BufferedImage[] framesList) {
        this.isRunning = isRunning;
        this.durationSleep = durationSleep;
        this.framesList = framesList;
    }

    public Boolean isIsRunning() {
        return this.isRunning;
    }

    public Boolean getIsRunning() {
        return this.isRunning;
    }

    public void setIsRunning(Boolean isRunning) {
        this.isRunning = isRunning;
    }

    public double getDurationSleep() {
        return this.durationSleep;
    }

    public void setDurationSleep(double durationSleep) {
        this.durationSleep = durationSleep;
    }

    public BufferedImage[] getFramesList() {
        return this.framesList;
    }
}
```

```

public void setFramesList(BufferedImage[] framesList) {
    this.framesList = framesList;
}

public void playAnimation(VisualObject visualObject, Graphics g) {
    Thread animationThread = new Thread(() -> {

        final AtomicBoolean running = new AtomicBoolean(isRunning); // Variable final para capturar el valor de
        while (running.get()) { // Utilizar la variable capturada
            for (int i = 0; i < framesList.length; i++) {
                BufferedImage frameToDraw = framesList[i];

                // Dibujar la imagen en el objeto visual utilizando el objeto Graphics
                g.drawImage(frameToDraw, (int) visualObject.getPosition().getxPos(),
                           (int) visualObject.getPosition().getyPos(), null);

                // Actualizar la representación visual del objeto
                // g.repaint();

                try {
                    Thread.sleep((long) durationSleep); // Pausa durante la duración específica
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            running.set(false);
        }
    });
    animationThread.start();
}

```

La clase "Animation" pertenece al paquete "visual" y se utiliza para representar una animación en un entorno gráfico. A continuación se describen los componentes clave de la clase:

1. Atributos: La clase tiene tres atributos públicos: "isRunning" (indica si la animación está en ejecución), "durationSleep" (duración de pausa entre los fotogramas de la animación) y "framesList" (una lista de imágenes que conforman la animación).
2. Constructor: Se proporciona un constructor que recibe los valores para los atributos "isRunning", "durationSleep" y "framesList" y los asigna a los respectivos atributos de la clase.
3. Métodos Getter y Setter: Se proporcionan métodos para obtener y establecer los valores de los atributos "isRunning", "durationSleep" y "framesList".
4. Método "playAnimation": Este método inicia la reproducción de la animación. Crea un nuevo hilo de ejecución utilizando la interfaz funcional y lambda de Java 8. Dentro del hilo, se itera a través de los fotogramas de la animación y se dibuja cada uno utilizando un objeto "Graphics". La duración de pausa entre los fotogramas se controla mediante la pausa del hilo utilizando el método "Thread.sleep()". El bucle se repite hasta que se establezca "isRunning" como false.

9.4 - POSITION

```
package visual;

public class Position {
    private double xPos;
    private double yPos;

    // * Constructor

    public Position(double x, double y) {
        this.xPos = x;
        this.yPos = y;
    }

    // * Methods

    public double getxPos() {
        return xPos;
    }

    public void setxPos(double xPos) {
        this.xPos = xPos;
    }

    public double getyPos() {
        return yPos;
    }

    public void setyPos(double yPos) {
        this.yPos = yPos;
    }
}
```

La clase "Position" pertenece al paquete "visual" y representa una posición en un espacio bidimensional. A continuación se describen los componentes clave de la clase:

- Atributos: La clase tiene dos atributos privados: "xPos" y "yPos", que representan las coordenadas x e y de la posición respectivamente.
- Constructor: Se proporciona un constructor que recibe dos parámetros, "x" y "y", que se utilizan para inicializar los atributos "xPos" y "yPos" de la clase.
- Métodos Getter y Setter: Se proporcionan métodos getter y setter para obtener y establecer los valores de los atributos "xPos" y "yPos".

10 - MANUALES DE USUARIO E INSTALACIÓN

En este módulo se describe como el usuario final puede ejecutar la aplicación final, tanto la aplicación principal que se ejecuta en el cluster de pantallas como la aplicación móvil.

10.1 - APLICACIÓN PRINCIPAL

Para ejecutar la aplicación principal se tienen que seguir los siguientes pasos:

1. Tal como se indica en el diagrama básico del proyecto, el cluster se compone de 12 equipos conectados a 12 pantallas y un equipo master. En este equipo se carga el archivo P2P.jar del juego en la siguiente carpeta “/home/dam/mountedFolder/P2P.jar”
2. Seguidamente, se ha elaborado un script (script.sh), que una vez ejecutado entra en comunicación con los 12 equipos del cluster y ejecuta simultáneamente la aplicación, simplificando así el procedimiento de ejecución.

```
#!/usr/bin/env sh

# Montar carpeta compartida
sudo mount -t cifs -o username=anonymous,password= //192.168.1.50/public /home/dam/mountedFolder/

# Seleccionar monitor
export DISPLAY=:0

# Configurar resolucion de monitor
echo "--DELMODE"
xrandr --delmode VGA1 1024x768
echo "--NEWMODE"
xrandr --newmode "1024x768" 85.25 1368 1440 1576 1784 768 771 781 798 -sync
echo "--ADDMODE"
xrandr --addmode VGA1 1024x768
echo "--OUTPUT"
xrandr --output VGA1 --mode 1024x768

# Ejecutar JAR
cd /home/dam/java/
java -jar /home/dam/mountedFolder/P2P.jar
```

Figura 10.1 : Código script.sh

10.2 - APLICACIÓN MÓVIL

Para ejecutar la aplicación móvil se tienen que seguir los siguientes pasos:

10.2.1 - INSTALACIÓN

En primer lugar, hay que preparar el dispositivo móvil android para permitir que se pueda conectar con Android Studio e instalar la aplicación directamente desde ahí. Para ello, tenemos que habilitar el modo depuración USB de las opciones de desarrollador del móvil android. Los pasos a seguir son los siguientes:

1. En el menú de ajustes e información del terminal, buscar la opción “Número de compilación” y hacer click en esa opción 7 veces hasta que aparezca el mensaje “¡Ahora están activadas las opciones para desarrolladores!”

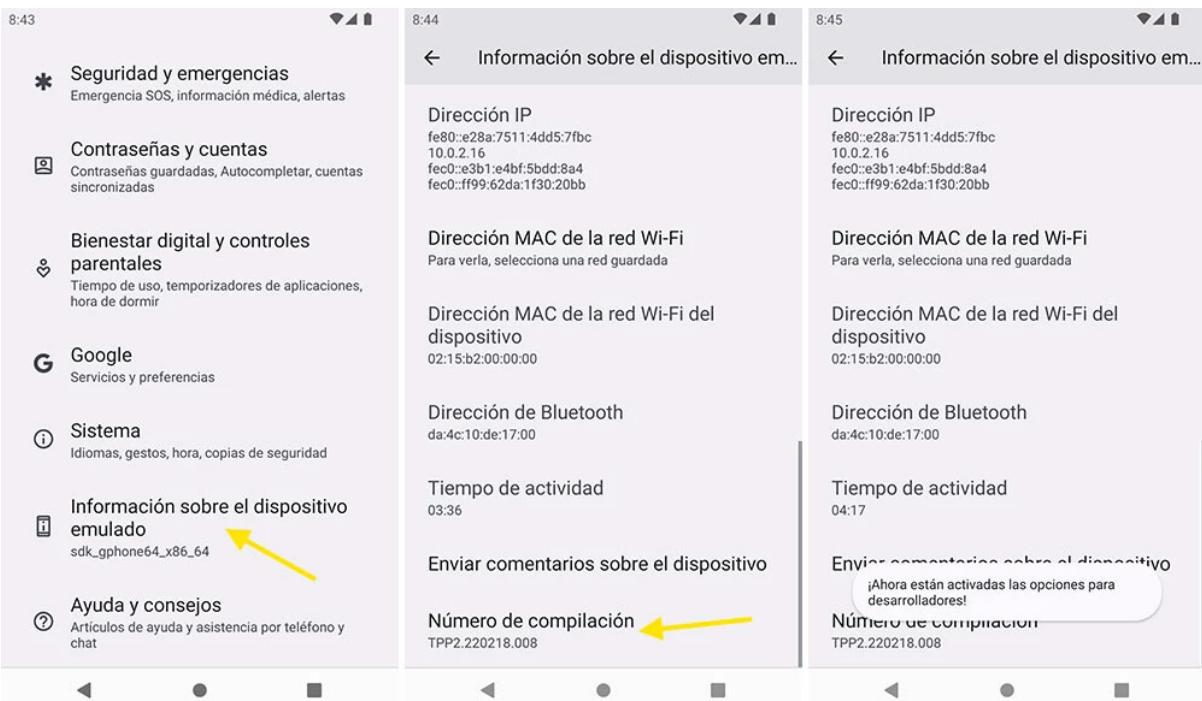


Figura 10.2 - Activación opciones desarrolladores

2. Una vez activado el menú de desarrollador, activar la opción de “Depuración por USB”.

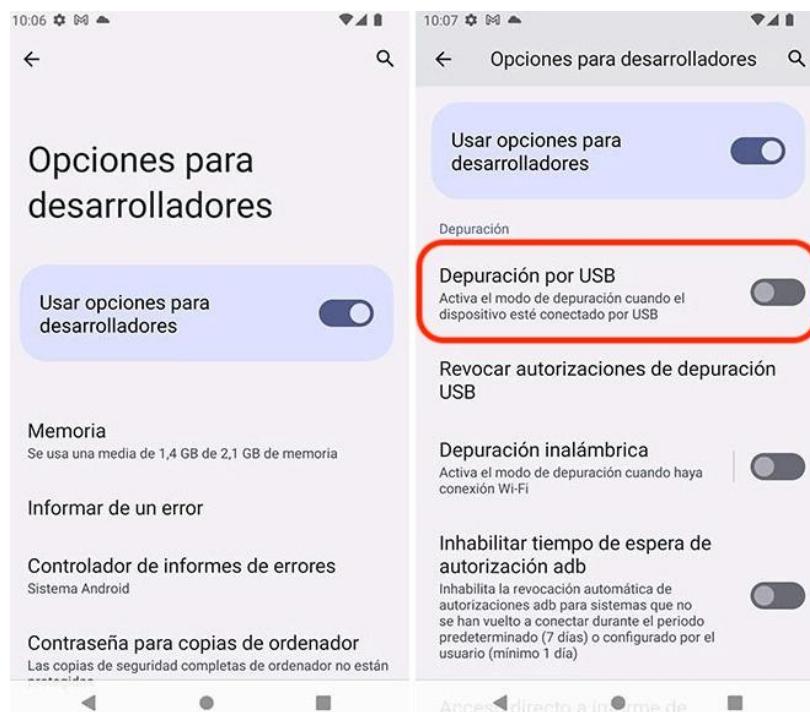


Figura 10.3 - Activación depuración por USB

Estos dos pasos anteriores tienen que realizarse la primera vez que conectamos un dispositivo móvil android a Android Studio, una vez realizados estos dos pasos ya no es necesario volver a realizarlos.

3. Seguidamente, descargamos la aplicación móvil desde el repositorio de GitHub desde el siguiente [enlace](#).
4. Importamos el proyecto en Android Studio, para ello nos dirigimos a la opción “File” -> “New” -> “Import Project”. Seguidamente seleccionamos el proyecto descargado.

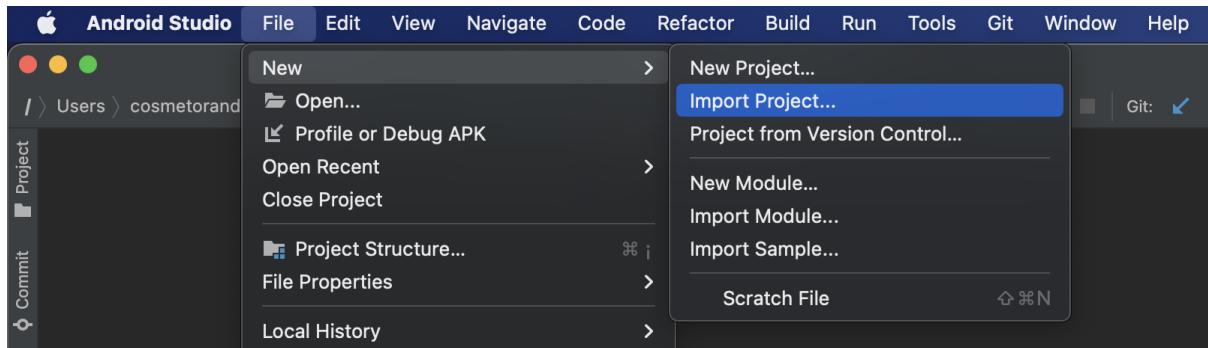


Figura 10.4 - Importar proyecto android 1

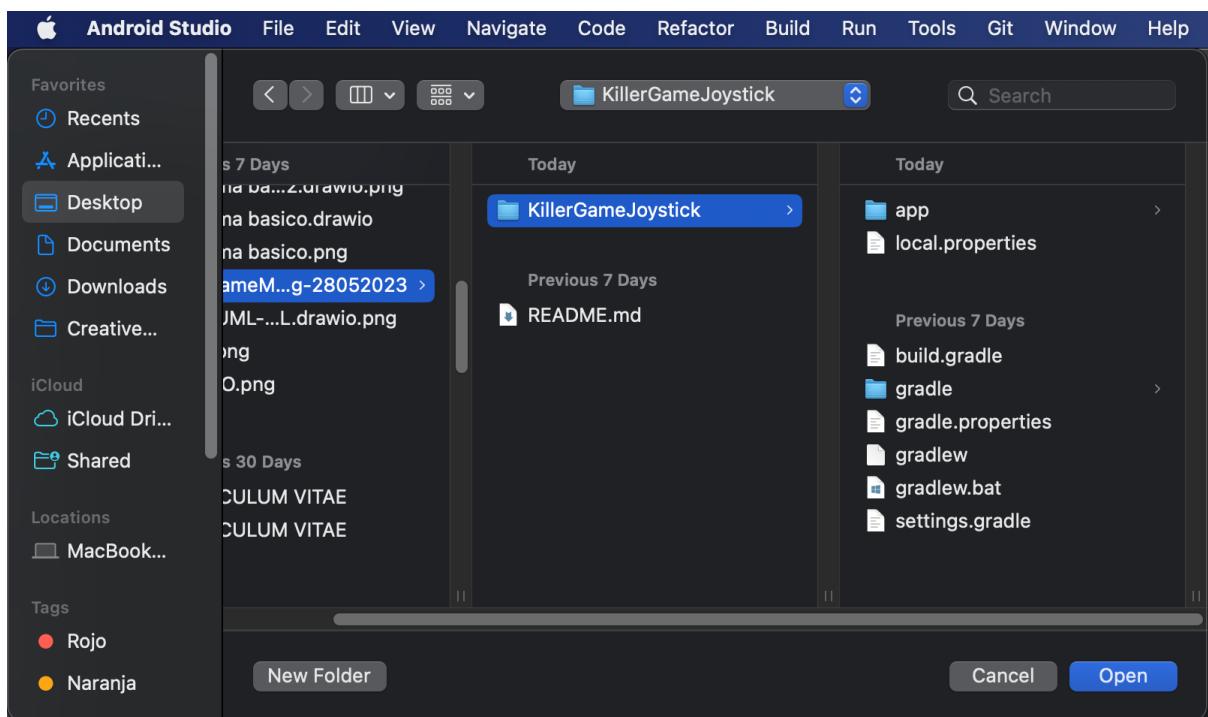


Figura 10.5 - Importar proyecto android 2

5. Una vez Importado el proyecto, conectamos el dispositivo android al ordenador, nos aseguramos de que el equipo reconoce el dispositivo correctamente y hacemos click sobre el botón “Play”.

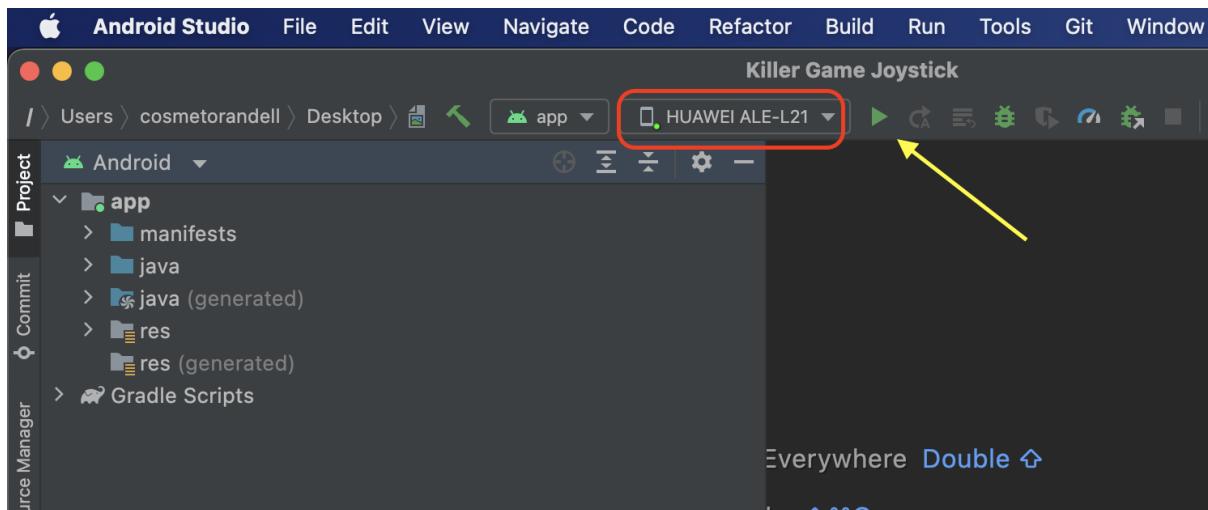


Figura 10.6 - Instalación aplicación en dispositivo

6. Iniciamos la aplicación presionando el icono de la aplicación.

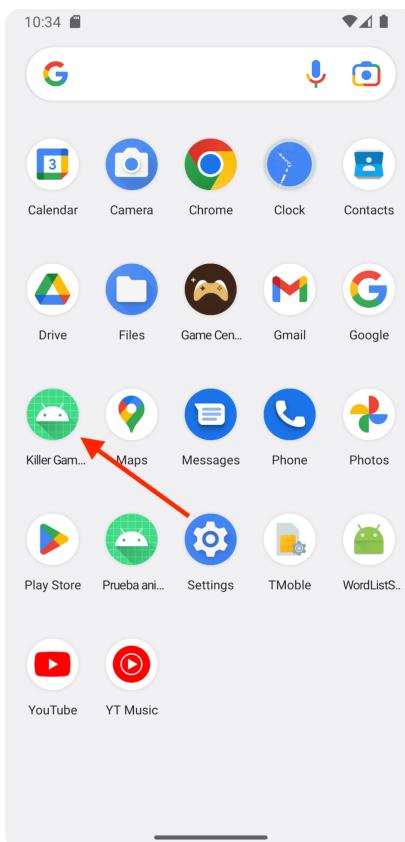


Figura 10.7 - Inicio de la aplicación móvil

7. Una vez ejecutada la aplicación, presionamos el botón “Press to connect”

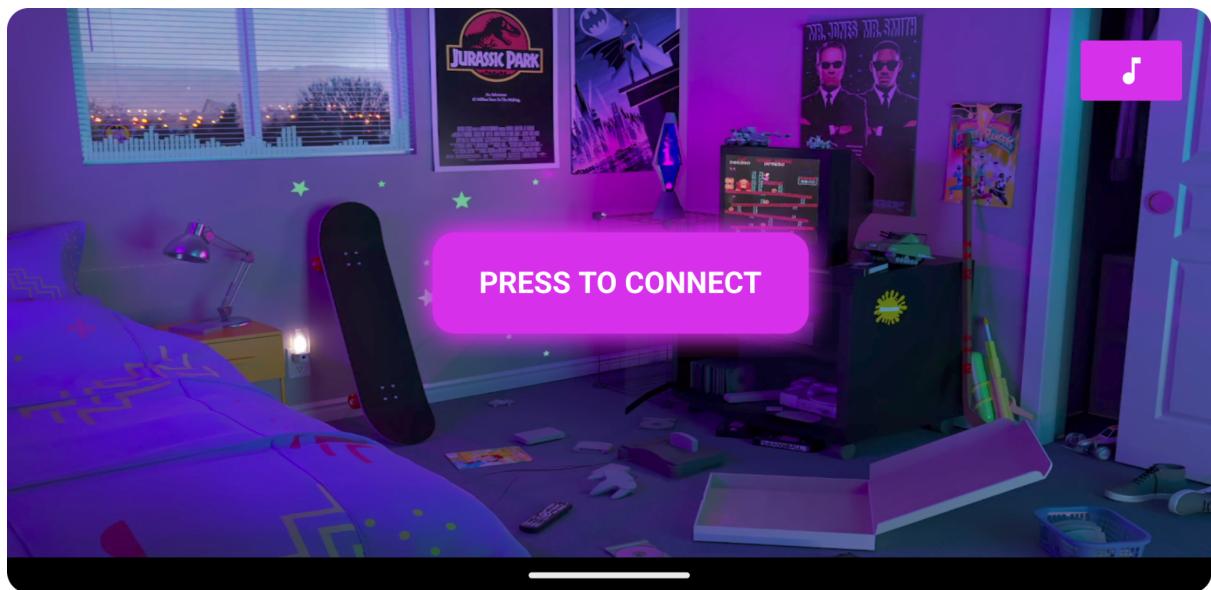


Figura 10.8 - Pantalla inicial del mando de juego

11 - CONCLUSIONES SOBRE EL PROYECTO GLOBAL Y SUBPROYECTO

11.1 - INTRODUCCIÓN

El desarrollo de esta aplicación ha sido una experiencia cargada de matices que van a ser desgranados a continuación, tanto por la parte del proyecto global como por todos los subproyectos contenidos en el mismo. Centrándose en los objetivos conseguidos, mejoras pendientes, dificultades del proyecto y lecciones aprendidas.

El desarrollo de esta aplicación ha sido una experiencia cargada de matices que van a ser desgranados a continuación, tratando, en primer lugar, los objetivos conseguidos, los cuales serán contrastados con las expectativas iniciales y las posibles mejoras pertinentes.

Por último, se dará una valoración general de las virtudes del proyecto, exponiendo la opinión más sincera del equipo acerca de porqué es una metodología que debería aplicarse en futuros cursos.

11.1.1 - CONCLUSIONES DEL PROYECTO GLOBAL

En este punto se aborda desde una perspectiva global las conclusiones de la totalidad del proyecto, analizando los objetivos logrados contrastados con las expectativas iniciales. También se hace hincapié en la dificultad general del proyecto no solo a nivel técnico sino también a nivel organizativo.

11.1.1.1 - REQUISITOS LOGRADOS

ID	Requisito	Estado
RF01	Conectarse con la matriz de pantallas	X
RF02	Editar las propiedades del juego	X
RF03	Elegir diferentes escenarios de juego	
RF04	Elegir tipo de nave	
RF05	Visualizar estadísticas del juego	
RF06	Jugar al video juego	X
RF07	Debe tener modo de juego - Duelo por equipos	
RF08	Debe tener máximo 2 equipos	
RF09	Máximo 8 jugadores en total	

RF10	Asignación aleatoria de equipos	
RF11	Posibilidad de disparar	
RF12	Límite tiempo de partida	
RF13	Objetos estáticos interactuables	
RNF01	El sistema debe ser confiable y seguro	X
RNF02	La aplicación debe admitir varios usuarios	X
RNF03	La aplicación debe ofrecer transparencia del proceso a los diferentes tipos de usuarios	X
RNF04	La aplicación tiene que emitir sonidos para interactuar con el jugador	X
RNF05	El desarrollo de la aplicación debe llevarse a cabo mediante el uso de Java	X
RS01	La aplicación debe ser adaptable a cualquier dispositivo / plataforma	X

Cuadro 11.1: Matriz requisitos logrados

11.1.1.2 - EXPECTATIVAS Y MEJORAS

En términos generales, el proyecto no ha cumplido con las expectativas del cliente, dado que durante la fase de análisis se especificaron 13 requisitos funcionales, 5 no funcionales y 1 requisito de sistema resultando en un total de 19 requisitos, de los cuales el 47,3 % fueron ejecutados.

Los requisitos que no se han alcanzado han sido debido a problemas en la definición inicial que han impedido cumplir con la deadline.

La consecuencia inicial de no haber llegado mínimo al 75% de los requisitos es que si bien se puede visualizar en el cluster principal de pantallas como todas se interconectan entre sí e incluso la aplicación móvil se conecta a la pantalla y genera una nave jugador. No se puede considerar que el juego sea jugable.

Actualmente está en una fase “demostración” de que es posible lograr el objetivo pero queda por implementar en la aplicación principal toda la lógica del juego y ofrecer una visual que vaya acorde con el juego.

11.1.1.1.1 - MEJORAS EN EL PROYECTO GLOBAL

Como es palpable, al haber conseguido una tasa de éxito en los requisitos por debajo del 50%, el objetivo principal en cuanto a mejoras aplicables a este inicialmente es cumplir en una fase inicial el 80% de los requisitos inicialmente establecidos.

Una vez conseguida esta tasa del 80% de requisitos lograda, la siguiente fase se establece en afinar el 20% restante.

Al desarrollar este proyecto mediante metodología ágil, a medida que avanza el proyecto y se logran los objetivos marcados pueden aparecer nuevas mejoras, tanto a nivel de opciones de juego, agregar componentes, nuevos elementos visuales, nuevos modos de juego o nuevas armas.

Para esto, en la fase de diseño de la aplicación ya se ha tenido en cuenta que debe de ser una aplicación con una estructura fácilmente escalable que permita realizar los cambios que se consideren oportunos sin afectar a la estructura principal del proyecto.

11.1.1.3 - LECCIONES APRENDIDAS

Desde el punto de vista didáctico, el proyecto ha sido una estupenda herramienta para, en primera instancia, asimilar y aplicar en un proyecto real la metodología tratada en clase acerca de cómo realizar un proyecto de desarrollo de software y en última instancia, comprendiendo la manera en la que realizar un despliegue completo de una aplicación mediante comunicaciones en diferentes dispositivos.

Por un lado, dividiendo el proceso en fases y aprendiendo a planificar cada una de ellas, experimentando con material tangible y no solo conceptualizando, se logra consolidar de forma más eficiente todos los conocimientos obtenidos durante el curso. Además, la creación y edición de documentación técnica y detallada es un gran plus a la hora de trabajar en un contexto profesional.

Adicionalmente, gracias a que se trata de un proyecto global en el que los diferentes miembros del equipo colaboran en elaborar una aplicación de manera colaborativa, mediante metodología ágil. Se logra tener una visión más amplia y común acerca de cómo se desarrolla habitualmente el trabajo en una empresa de software actual, esto hace que la experiencia sea mucho más enriquecedora.

Finalmente, partiendo desde el punto anterior, el hecho de trabajar en un proyecto global colaborativo ha puesto de manifiesto la dificultad que supone, tanto en infraestructura, colaboración, herramientas involucradas, y sobre todo de la coordinación de los diferentes miembros del equipo. A nivel organizativo es mucho más sencillo elaborar proyectos individuales que colectivos, pero por otro lado habitualmente no refleja la realidad de trabajo actual. Por tanto, de esta experiencia se tienen que sacar como conclusión final la importancia de seguir con las tareas marcadas en tiempo y forma, ya que en caso contrario no se consigue el objetivo global marcado.

11.1.2 - CONCLUSIONES DEL SUBPROYECTO PROYECTO INDIVIDUAL

En este punto, se presentan las conclusiones del subproyecto de desarrollo, centrándose especialmente en la planificación y creación de las diversas clases. Se destaca la importancia de una adecuada planificación para el desarrollo de las clases, lo cual fue fundamental para el éxito del proyecto. Además, se resalta el valor de la elaboración cuidadosa de las clases, garantizando su funcionalidad y coherencia en el sistema. Estas conclusiones refuerzan la importancia de una planificación sólida y una implementación adecuada en el proceso de desarrollo de software.

11.1.2.1 - EXPECTATIVAS Y MEJORAS

Uno de los principales desafíos del proyecto ha sido la falta de una planificación sólida y el rendimiento general del equipo. En un principio, había muchas ideas y expectativas elevadas que no se materializaron debido a la falta de enfoque. Nos encontramos con dificultades para tomar decisiones concretas y se perdió mucho tiempo en el desarrollo del UML sin llegar a conclusiones definitivas.

El mayor obstáculo fue la gestión del tiempo, ya que nos vimos obligados a trabajar a contrarreloj para cumplir con los plazos establecidos. Esto resultó en la necesidad de realizar ajustes y modificaciones en el UML y en las clases ya creadas para adaptarlas a las cambiantes necesidades del proyecto. Esta falta de una base sólida y la constante refactorización generaron dificultades adicionales y afectaron el rendimiento general del equipo.

En resumen, el mayor problema radicó en la falta de una planificación adecuada, la gestión del tiempo y la necesidad de realizar cambios constantes en el UML y las clases. Estos desafíos nos enseñaron la importancia de establecer una base sólida desde el principio y de gestionar eficientemente los recursos y plazos para evitar contratiempos en el desarrollo del proyecto.

