

Carlos Rubio Morell

PROYECTO FINAL DESARROLLO DE SOFTWARE: KILLERGAME



**Francesc
de Borja Moll**

Centre Integrat de
Formació Professional

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	9
1.1 - DESCRIPCIÓN DEL PROYECTO GLOBAL	9
1.2 - PROPUESTA	9
1.3 - ESTRUCTURA DEL DOCUMENTO	10
2 - DESCRIPCIÓN DE LA ARQUITECTURA GLOBAL DEL PROYECTO	11
2.1 METODOLOGÍA	11
2.2 FASES DEL DESARROLLO DEL SOFTWARE	12
2.3 METODOLOGÍA SCRUM	13
2.4 - PLANIFICACIÓN DEL PROYECTO	14
2.5 - ANÁLISIS	15
2.5.1 - PARTES INTERESADAS	15
2.5.2 - REQUISITOS DE LA APLICACIÓN	15
2.5.2.1 - REQUISITOS FUNCIONALES	15
2.5.2.2 - REQUISITOS NO FUNCIONALES	16
2.5.2.3 - REQUISITOS DE SISTEMA	16
2.5.3 - REGLAS DEL JUEGO	16
2.5.4 - PRUEBAS DE ACEPTACIÓN DE REQUISITOS	17
2.5.4.1 - OBJETIVOS DE LAS PRUEBAS DE ACEPTACIÓN DE REQUISITOS	17
2.5.4.2 - DESCRIPCIÓN DE LAS PRUEBAS	17
2.5.5 - DISEÑO	18
2.5.5.1 - MODELO CONCEPTUAL DEL PROYECTO	18
2.5.5.2 - DIAGRAMA DE CLASES	19
2.5.5.2.1 - DIAGRAMA DE CLASES GENERAL APLICACIÓN PRINCIPAL	19
2.5.5.2.2 - DIAGRAMA DE CLASES GENERAL APLICACIÓN MÓVIL	20
2.5.5.2.3- DIAGRAMA DE CLASES GENERAL APLICACIÓN SONIDO	20
3 - PLANIFICACIÓN Y DESCRIPCIÓN DE SUBPROYECTOS	21
3.1 SUBPROYECTOS	21
3.1.1 Coordinación	21
3.1.2 Main Controller / Integrador	21
3.1.3 Visual	21
3.1.4 Comunicaciones	22
3.1.5 Aplicación móvil	22
3.1.6 Sonido	22
3.1.7 Cluster	22
3.2 Asignación de tareas / roles por subtareas y por colaborador	23
4 - DESCRIPCIÓN DETALLADA DEL SUBPROYECTO INDIVIDUAL	25

4.2 - ROL ASIGNADO	25
4.2.1 - PREPARACIÓN DE LOS CHROMEBOOKS	
4.2.2 - CREACIÓN DEL FONDO DEL TERRENO DE JUEGO	
4.2.3 - DISTRIBUCIÓN DEL FONDO EN LAS PANTALLAS	
4.2.4 CARGA Y GESTIÓN DE RECURSOS VISUALES	
4.2.5 APOYO A COMPAÑEROS DEL EQUIPO	25
	26
5 - PLANIFICACIÓN DE LAS TAREAS DEL SUBPROYECTO INDIVIDUAL	27
5.1 - INTRODUCCIÓN	27
5.2 - DIAGRAMA DE GANTT	27
6 - DETALLE DE LAS DEDICACIONES Y TAREAS REALIZADAS	28
6.1 - PREPARACIÓN DE LOS CHORMEBOOKS	28
6.2 - CREACIÓN DEL FONDO DEL TERRENO DE JUEGO	28
6.3 - DISTRIBUCIÓN DEL FONDO EN LAS PANTALLAS	29
6.4 - CARGA Y GESTIÓN DE RECURSOS VISUALES	29
	30
7 - DESCRIPCIÓN DEL ENTORNO TECNOLÓGICO Y HERRAMIENTAS USADAS	31
7.1 - VISUAL STUDIO CODE v1.67.1	31
7.2 - ECLIPSE IDE v2022-23 (4.23.0)	31
7.3 - ANDROID STUDIO 2021.3.1 PATCH 1	31
7.4 - INTELLIJ IDEA 2022.3.1	32
7.5 - ATlassian JIRA CLOUD	32
7.6 - GITHUB	32
7.7 GIT	32
7.7.1 - ESTRUCTURA DE REPOSITORIOS	33
7.7.2 - ESTRUCTURA DE RAMAS	33
7.7.3 - ESTRUCTURA DE PERMISOS	34
8 - DESCRIPCIÓN DE LA ARQUITECTURA DEL SUBPROYECTO	35
9 - DESCRIPCIÓN DE LOS MÓDULOS O ÍTEMES DESARROLLADOS.	36
9.1 - BackgroundaManager	
9.2 - AssetsManager	
9.3 - DynamicVisualObject	
9.4 - SpaceShip	
9.5 - Bullet	
9.6 - Asteroid	36
	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 - MEJORAS EN EL PROYECTO GLOBAL	47
11.1.1.3 - LECCIONES APRENDIDAS	47
11.1.2 - CONCLUSIONES DEL SUBPROYECTO PROYECTO INDIVIDUAL	48
11.1.2.1 - EXPECTATIVAS Y MEJORAS	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	27
7.1 - Diagrama de ramas visual	34
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.

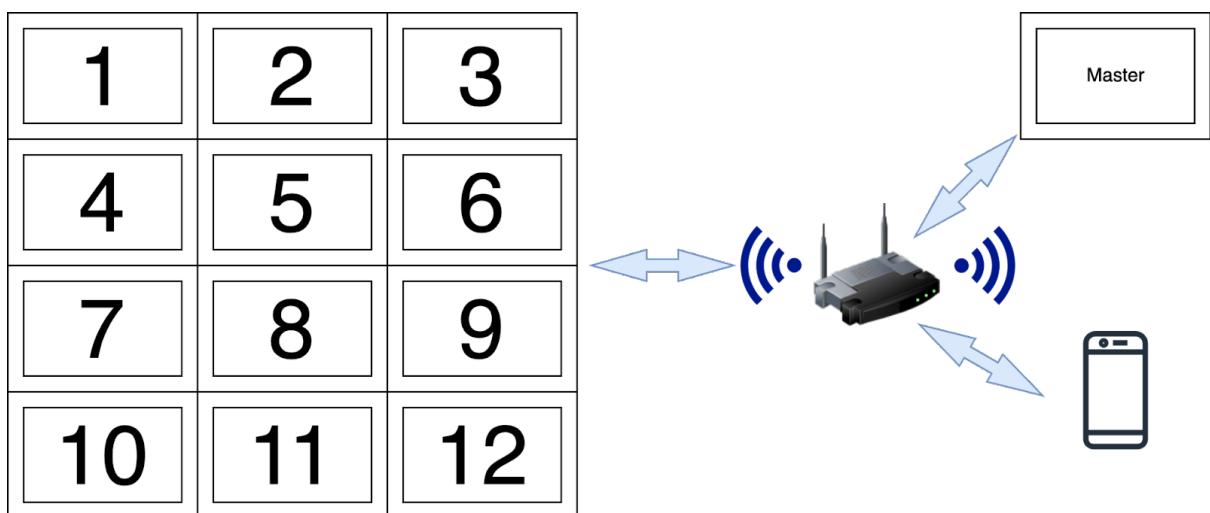


Figura 1.1 : Diagrama básico del proyecto

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. Despues 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.

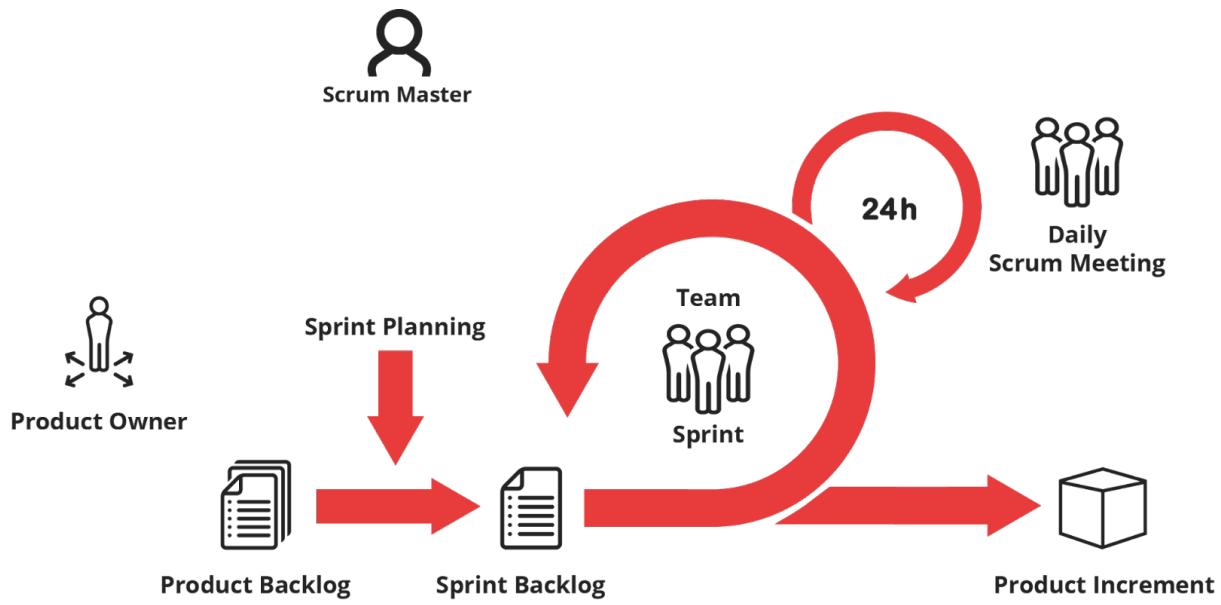


Figura 2.1: Ilustración desarrollo metodología scrum

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 .

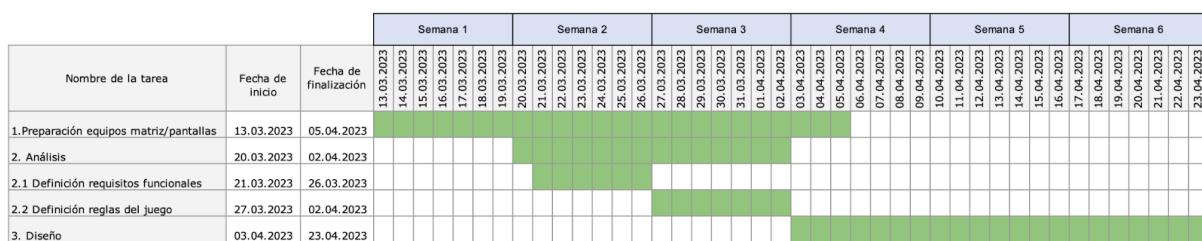


Figura 2.2: Diagrama de Gantt. Fases 1, 2 y 3.

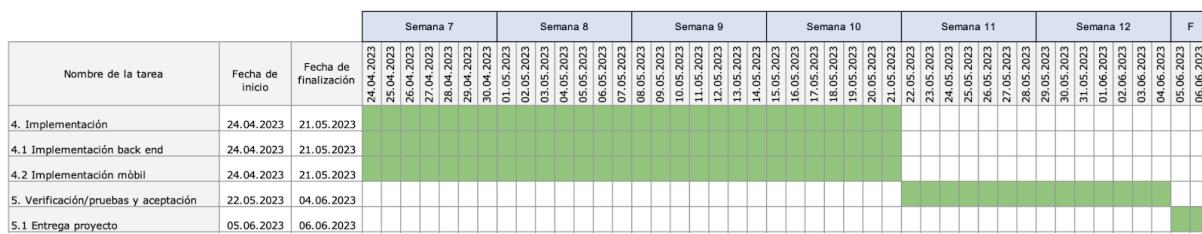


Figura 2.3: Diagrama de Gantt. Fases 4 y 5.

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 interactivos

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:

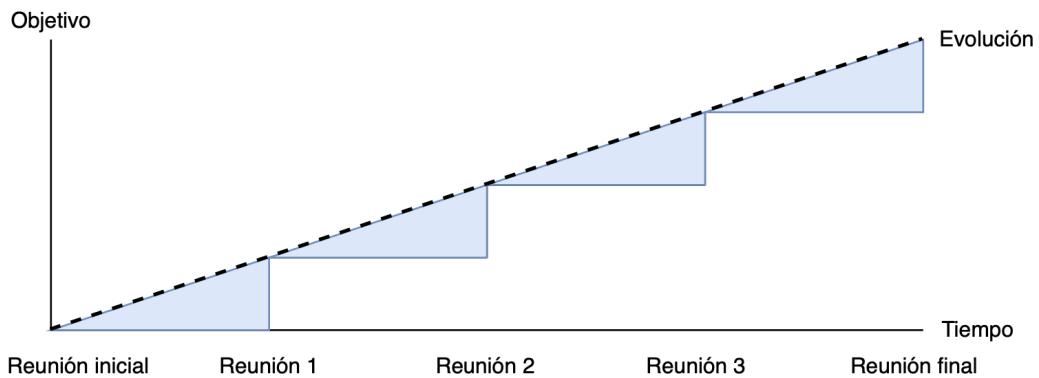


Figura 2.4 : Gráfico proceso de reunión-aproximación

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

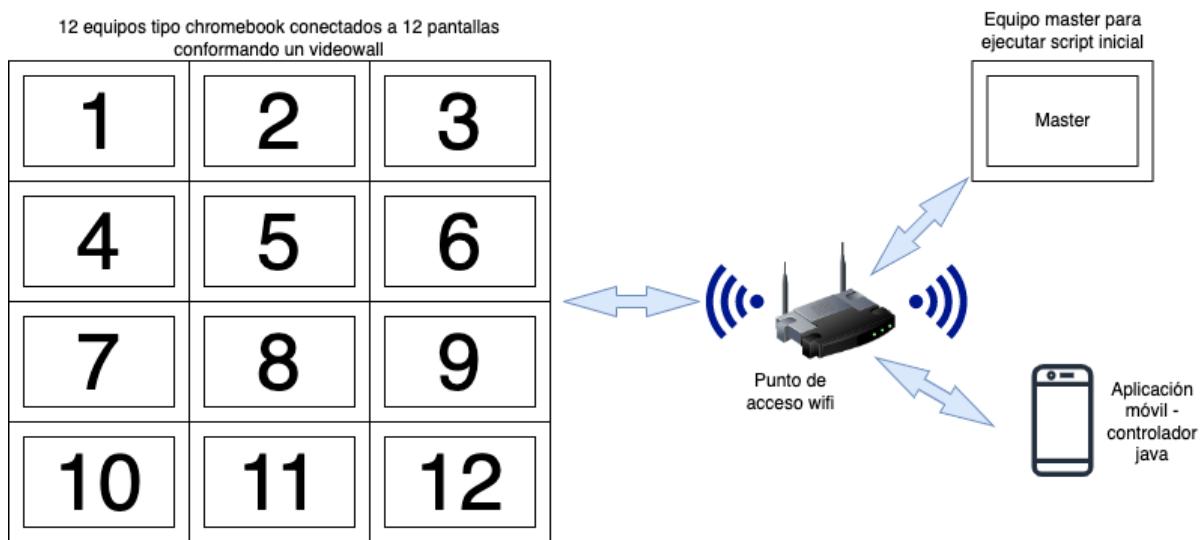


Figura 2.5 : Modelo conceptual del proyecto

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

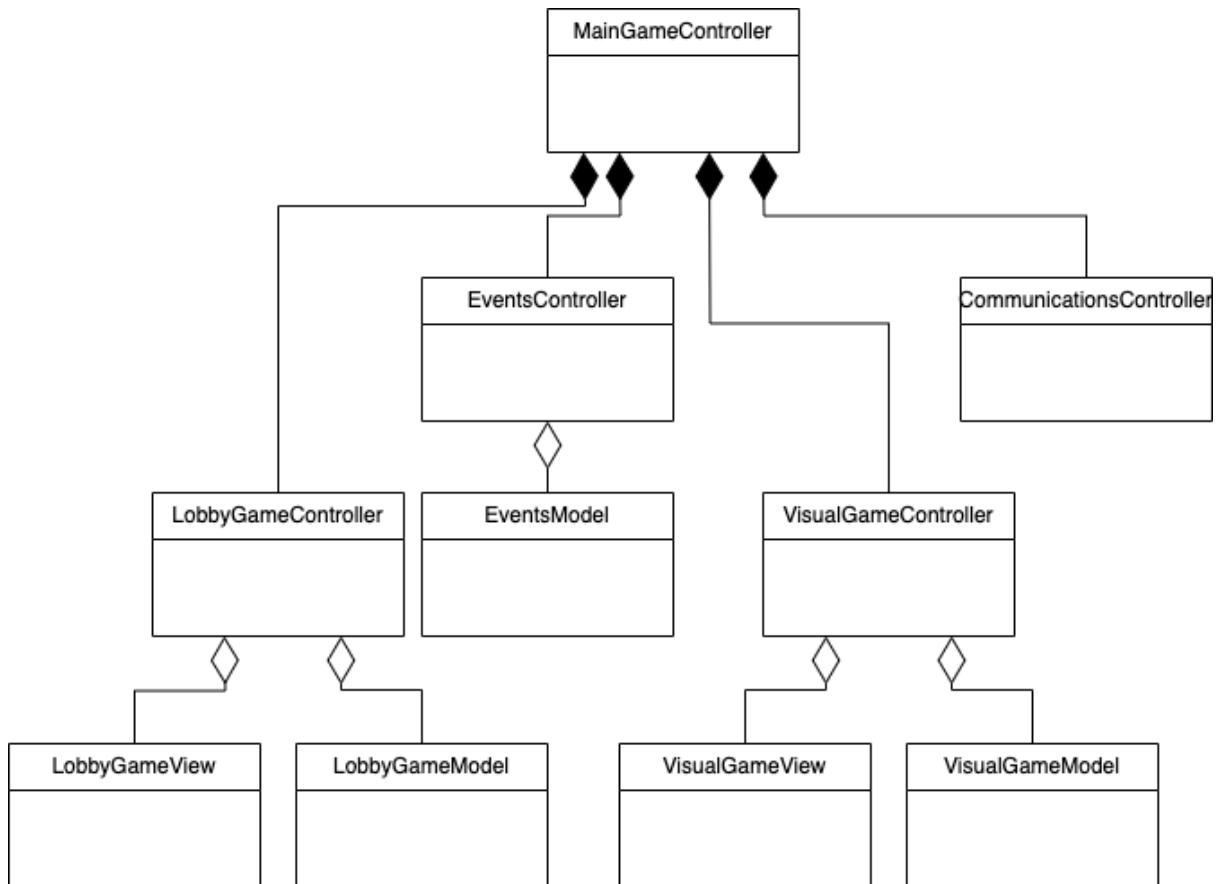


Figura 2.6 : Diagrama de clases general aplicación principal

2.5.5.2.2 - DIAGRAMA DE CLASES GENERAL APLICACIÓN MÓVIL

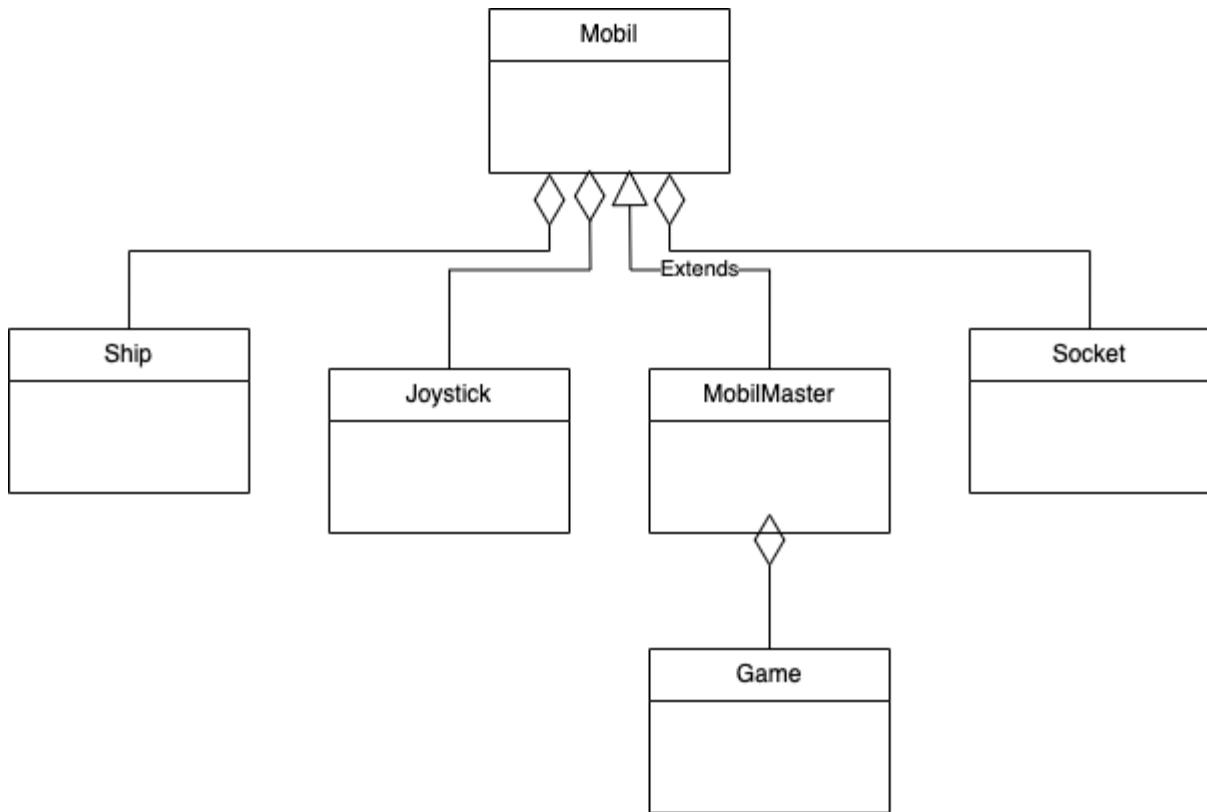


Figura 2.7 : Diagrama de clases general aplicación móvil

2.5.5.2.3- DIAGRAMA DE CLASES GENERAL APLICACIÓN SONIDO

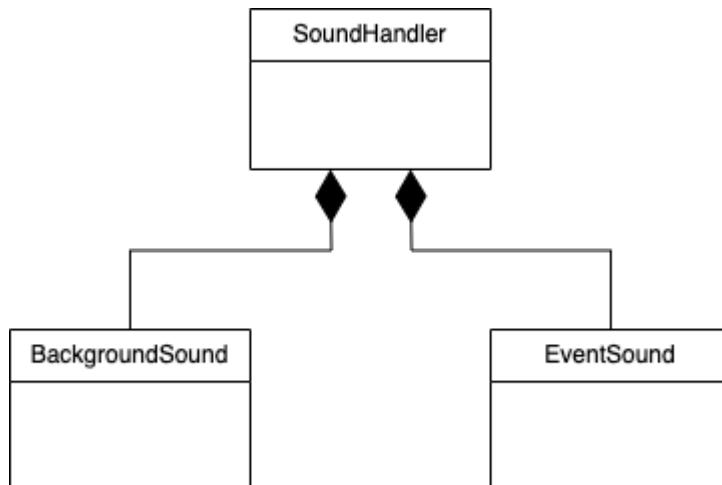


Figura 2.8 : Diagrama de clases general aplicación sonido

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

En este caso, se ha asignado como proyecto individual realizar tareas relacionadas con la parte visual del proyecto, concretamente la elección y elaboración de los fondos del terreno de juego, la distribución y visualización en las pantallas de dichos fondos, la carga y gestión de todas las imágenes usadas en el juego y el apoyo general al resto de compañeros del equipo de efectos visuales en las tareas de implementación de la gestión y visualización de los efectos visuales del proyecto.

4.2.1 PREPARACIÓN DE LOS CHROMEBOOKS

Durante los primeros días de la puesta en marcha del proyecto, se configuraron alrededor de 15 o 16 Chromebooks con el objetivo de instalar un sistema operativo ligero (Lubuntu), se establecieron usuarios con privilegios y otros como invitados, y se configuraron direcciones IP estáticas. Además, colaboré en la construcción del armario en el que se mantendrán los portátiles enchufados usando un taladro para realizar los agujeros pertinentes.

4.2.2 - CREACIÓN DEL FONDO DEL TERRENO DE JUEGO

A la hora de empezar la partida, los jugadores tendrán que elegir entre cuatro mapas diferentes, dos de ellos de estética “retro” y dos de estética realista:

1. Los mapas de estética *retro* han sido creados usando un [programa](#) gratuito de generación de fondos del espacio de la página itch.io.
2. Los mapas de estética realista consisten en, uno de ellos, una foto de la Luna y, el otro, en una captura de pantalla de un fotograma de la película de Star Wars “*El retorno del Jedi*” en el cual se ve la *estrella de la muerte* destruida sobre *Endor*.

4.2.3 - DISTRIBUCIÓN DEL FONDO EN LAS PANTALLAS

Dado que el juego se visualiza en 12 pantallas diferentes a modo de cuadrícula, los fondos se han dividido en 12 partes iguales y cada una de esas partes se muestra en su pantalla correspondiente mediante el uso de una clase que gestiona la distribución de los fondos llamada *BackgroundManager*. Esta clase se llama desde la clase encargada de pintar los

elementos en la pantalla y devuelve el fondo a pintar en base al mapa escogido y el ID de la pantalla en el que se va a dibujar el fondo.

4.2.4 - CARGA Y GESTIÓN DE RECURSOS VISUALES

Para evitar consumir recursos de manera innecesaria, se ha implementado una clase, de nombre *AssetsManager*, que se encarga de cargar todos los recursos visuales de la parte visual del proyecto una vez instanciada y así, en lugar de tener que cargar el archivo a dibujar cada vez que se necesite, se obtiene el archivo mediante los *getters* de esta clase, evitando así consumir recursos adicionales.

4.2.5 - APOYO A COMPAÑEROS DEL EQUIPO

Para conseguir una repartición equitativa de la carga de trabajo de la parte visual del proyecto, proporcioné apoyo en la implementación de diversas clases del proyecto. Entre esas clases, principalmente colaboré en la creación de las clases de objetos visuales dinámicos. Estos objetos son todos aquellos que se ven representados de manera gráfica y poseen algún tipo de movimiento. Dichos objetos requieren una serie de atributos (tales como su posición, estado, vida, ángulo, etc.) que se usarán tanto para la parte gráfica y física como para el resto de secciones del proyecto global.

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

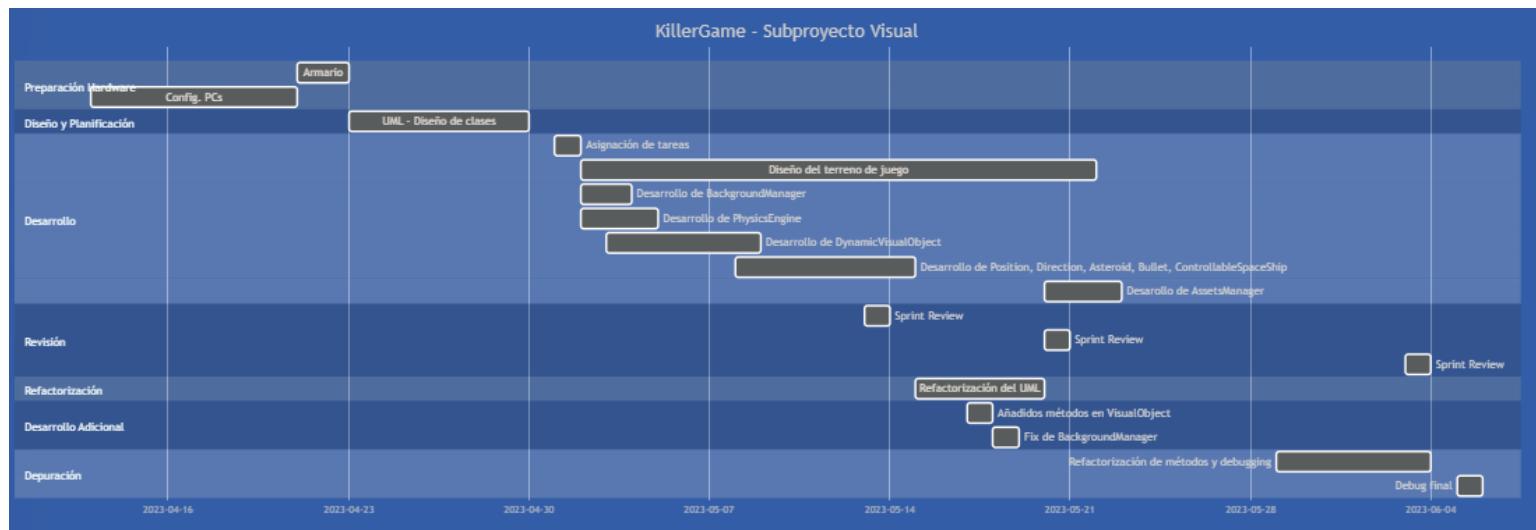


Figura 5.1 : Diagrama de Gantt subprojeto visual.

6 - DETALLE DE LAS DEDICACIONES Y TAREAS REALIZADAS

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

Por tanto, se hace referencia clara a las funciones clave desempeñadas durante el desarrollo del proyecto global y las responsabilidades adquiridas durante el desarrollo del mismo.

6.1 - PREPARACIÓN DE LOS CHROMEBOOKS

Durante los primeros días de la puesta en marcha del proyecto, se llevó a cabo una tarea que salió del alcance de la parte visual principalmente asignada. En concreto, se configuraron alrededor de 15 o 16 Chromebooks con el objetivo de instalar un sistema operativo ligero (Lubuntu). Además, se establecieron usuarios con privilegios y otros como invitados, y se configuraron direcciones IP estáticas. Además, colaboré en la construcción del armario en el que se mantendrán los portátiles enchufados usando un taladro para realizar los agujeros pertinentes.

6.2 - CREACIÓN DEL FONDO DEL TERRENO DE JUEGO

El objetivo de este punto es la creación de los cuatro terrenos de juego sobre los que los jugadores podrán desarrollar su partida. De estos cuatro mapas, dos de ellos son de estética “retro” y dos de estética realista:

1. Los mapas de estética *retro* han sido creados usando un [programa](#) gratuito de generación de fondos del espacio de la página itch.io.
2. Los mapas de estética realista consisten en, uno de ellos, una foto de la Luna y, el otro, en una captura de pantalla de un fotograma de la película de Star Wars “*El retorno del Jedi*” en el cual se ve la *estrella de la muerte* destruida sobre *Endor*.

6.3 - DISTRIBUCIÓN DEL FONDO EN LAS PANTALLAS

Este punto consiste en la implementación de una clase que distribuya correctamente el terreno de juego, previamente dividido en partes iguales, entre las doce pantallas del proyecto para su correcta y lógica visualización.

6.3 - CARGA Y GESTIÓN DE RECURSOS VISUALES

Este punto consiste en la implementación de una clase que se ocupe de cargar todos los archivos visuales del proyecto una vez instanciada para que el resto de secciones del proyecto accedan a los mismos mediante el uso de *getters* para evitar el consumo innecesario de recursos.

6.4 - DESARROLLO OBJETOS DINÁMICOS VISUALES

El objetivo de este punto trata de desarrollar e implementar los objetos que se mostrarán en pantalla y tendrán movimiento. Estos objetos son los mencionados en el anterior punto.

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 esta 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 esta 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.

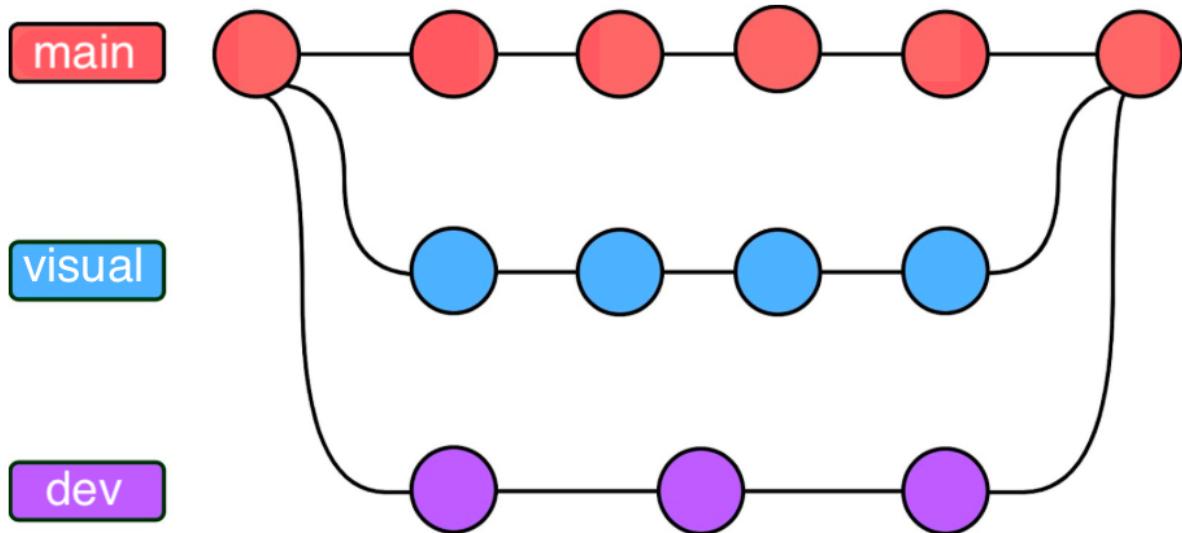


Figura 7.1 : Diagrama de ramas visual

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 este apartado se describe la arquitectura del subproyecto asignado a nivel específico, donde se incluyen los diagramas y los módulos en los que se colabora.

El subproyecto se encuentra dentro del apartado visual, más específicamente en la parte de gestión de recursos visuales y en la parte de los objetos visuales dinámicos. Por lo que podemos representar nuestra arquitectura con un diagrama UML sobre las clases Java que se han implantado para representar objetos visuales dinámicos y el motor de físicas.

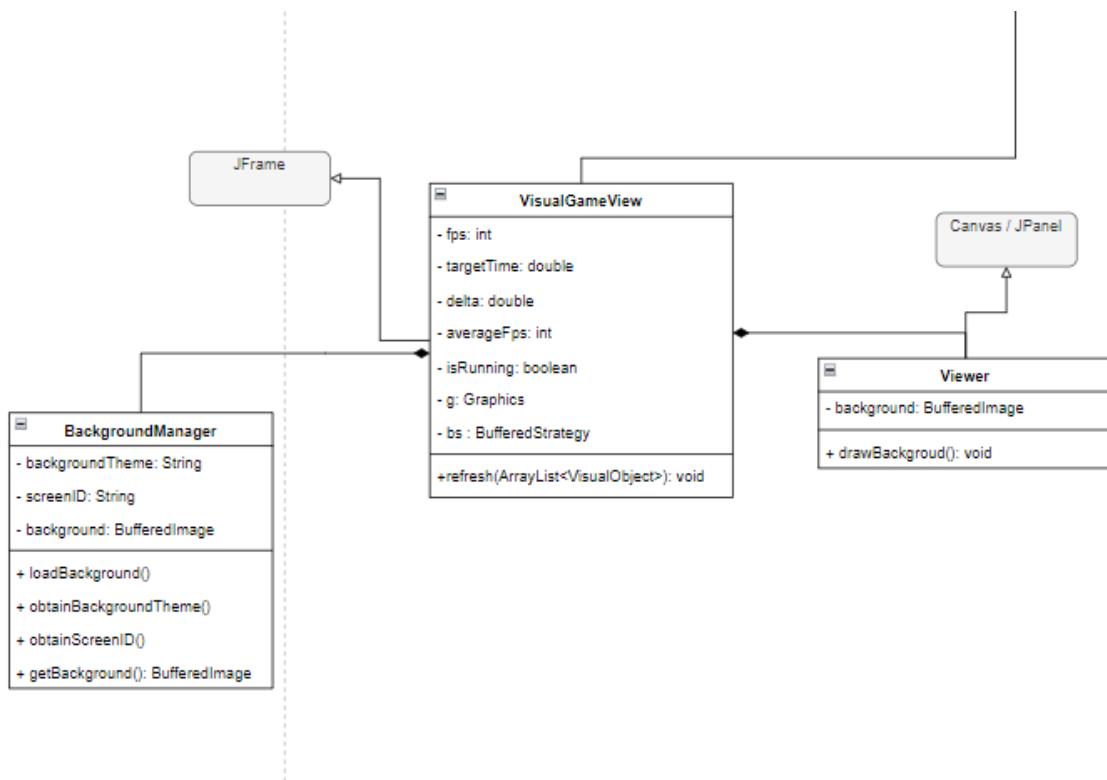


Figura 8.1 Diagrama UML gestor de fondos

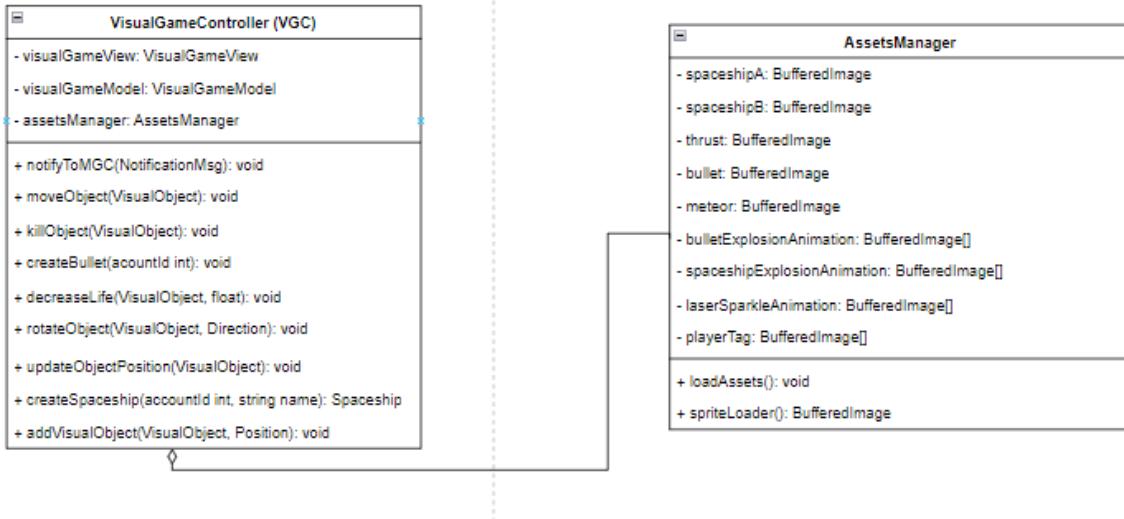


Figura 8.2 Diagrama UML gestor de recursos visuales

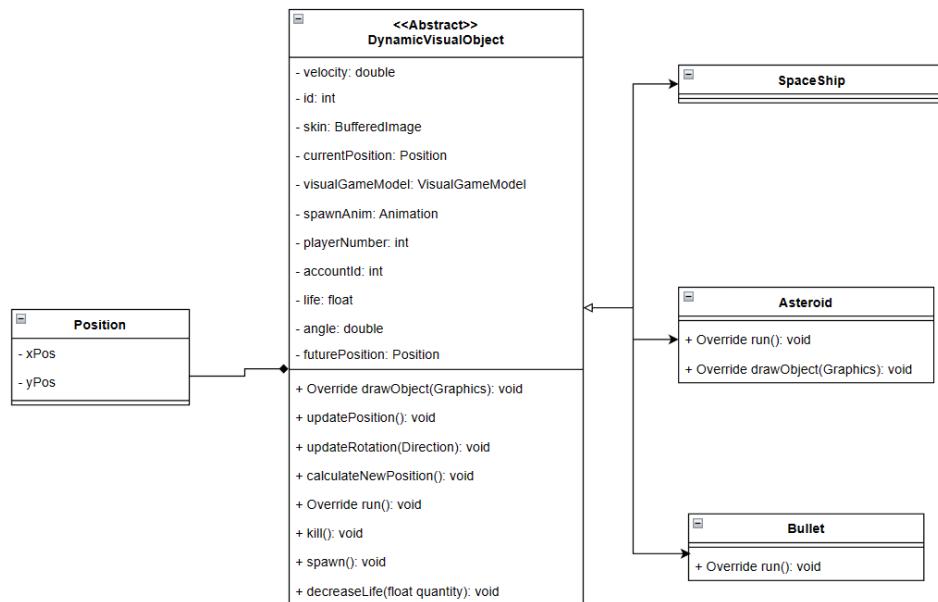


Figura 8.3 Diagrama UML objetos visuales dinámicos

La clase de gestión de fondos (*BackgroundManager*) es usada por la clase **VisualGameView** a la hora de la instanciación de la clase **Viewer**, a la que se le pasa como parámetro *background* el fondo gestionado por **BackgroundManager**.

La clase de gestión de recursos visualies (**AssetsManager**) es usada por la clase **VisualGameController**, que tiene un atributo de tipo **AssetsManager** mediante el cual el resto de secciones de la parte visual del proyecto acceden a los archivos de imagen.

Nuestros objetos visuales dinámicos empiezan por la clase **DynamicVisualObject**, una clase abstracta que representa todos los objetos visuales dinámicos. Contiene los atributos

necesarios tanto como objeto visual dinámico como objeto visual en sí, además de los métodos necesarios para su tratamiento. De esta clase heredan otras 3 clases, que representan nuestros 3 objetos dinámicos visuales del proyecto; SpaceShip (nave), Asteroid (Asteroide), Bullet (Bala).

9 - DESCRIPCIÓN DE LOS MÓDULOS O ÍTEMS DESARROLLADOS.

En este módulo, al tratarse de un subproyecto de coordinación describe el desarrollo del software BPM que se ha usado para este proyecto.

9.1 - BackgroundManager

```
public class BackgroundManager {

    // ! Attributes
    private VisualGameView visualGameView;
    private String backgroundTheme;
    private Maps mapName;
    private String screenID;
    private BufferedImage background;

    // ! Constructor
    public BackgroundManager(VisualGameView visualGameView) {
        this.visualGameView = visualGameView;
        obtainBackgroundTheme();
        obtainScreenID();
        loadBackground();
    }

    // ! Methods

    private void loadBackground() {
        try {
            background =
ImageIO.read(Objects.requireNonNull(getClass().getResource("/visual/img/backgrounds/" + backgroundTheme + "/background" + screenID + ".png")));
        } catch (IOException e) {
            System.err.println("Error en la obtención del fondo");
        }
    }

    private void obtainBackgroundTheme() {
```

```

        mapName =
visualGameView.getVisualGameController().getMainGameController().getBackgroundTheme
();
    switch (mapName) {
        case MAP_1:
            backgroundTheme = "theme1";
            break;

        case MAP_2:
            backgroundTheme = "theme2";
            break;

        case MAP_3:
            backgroundTheme = "theme3";
            break;

        case MAP_4:
            backgroundTheme = "theme4";
            break;
    }

}

private void obtainScreenID() {
    screenID =
String.valueOf(visualGameView.getVisualGameController().getMainGameController().get
ConfigurationFileDialog());
}

// ! Getters and Setters

public BufferedImage getBackground() {
    return background;
}
}

```

La clase "BackgroundManager" es responsable de gestionar y cargar imágenes de fondo en el proyecto. Contiene los atributos "visualGameView" (la vista del juego), "backgroundTheme" (tema de fondo), "mapName" (nombre del mapa), "screenID" (ID de pantalla) y "background" (la imagen de fondo en sí, representada como un objeto `BufferedImage`).

El constructor de la clase toma un objeto "VisualGameView" como parámetro y se encarga de inicializar los atributos necesarios. Llama a los métodos "obtainBackgroundTheme", "obtainScreenID" y "loadBackground" para obtener el tema de fondo, el ID de pantalla y cargar la imagen de fondo correspondiente, respectivamente.

El método "loadBackground" utiliza la clase "ImageIO" para cargar la imagen de fondo desde un archivo PNG ubicado en una ruta específica basada en el tema de fondo y el ID de pantalla.

Los métodos "obtainBackgroundTheme" y "obtainScreenID" obtienen el tema de fondo y el ID de pantalla llamando a los métodos correspondientes del controlador principal del juego.

El método getter "getBackground" devuelve la imagen de fondo cargada como un objeto `BufferedImage`.

9.2 - AssetsManager

```
public class AssetsManager {

    public BufferedImage spaceShipA;
    public BufferedImage spaceShipB;
    public BufferedImage bullet;
    public BufferedImage meteor;

    // Animations
    public BufferedImage[] bulletExplosionAnimation = new BufferedImage[5];
    public BufferedImage[] spaceShipExplosionAnimation = new BufferedImage[18];
    public BufferedImage[] laserSparkleAnimation = new BufferedImage[5];

    public AssetsManager() {
        loadAssets();
    }

    public void loadAssets() {

        spaceShipA = spriteLoader("/visual/img/assets/spaceShipA.png");
        spaceShipB = spriteLoader("/visual/img/assets/spaceShipB.png");
        bullet = spriteLoader("/visual/img/assets/laser.png");
        meteor = spriteLoader("/visual/img/assets/meteor.png");

        for(int i = 0; i < bulletExplosionAnimation.length; i++)
            bulletExplosionAnimation[i] =
        spriteLoader("/visual/img/animations/bulletExplosion/"+i+".png");

        for(int i = 0; i < spaceShipExplosionAnimation.length; i++)
            spaceShipExplosionAnimation[i] =
        spriteLoader("/visual/img/animations/spaceShipExplosion/"+i+".png");

        for(int i = 0; i < laserSparkleAnimation.length; i++)
            laserSparkleAnimation[i] =
        spriteLoader("/visual/img/animations/laserSparkle/"+i+".png");

    }
}
```

```

public BufferedImage spriteLoader(String path) {
    System.out.println(path);
    try {
        return ImageIO.read(
            Objects.requireNonNull(
                AssetsManager.class.getResource(path)
                    )));
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}

public BufferedImage getSpaceShipA() {
    return spaceShipA;
}

public BufferedImage getSpaceShipB() {
    return spaceShipB;
}

public BufferedImage getBullet() {
    return bullet;
}

public BufferedImage getMeteor() {
    return meteor;
}

public BufferedImage[] getBulletExplosionAnimation() {
    return bulletExplosionAnimation;
}

public BufferedImage[] getSpaceShipExplosionAnimation() {
    return spaceShipExplosionAnimation;
}

public BufferedImage[] getLaserSparkleAnimation() {
    return laserSparkleAnimation;
}
}

```

La clase "AssetsManager" es la clase responsable de cargar y almacenar los recursos gráficos, como imágenes estáticas y animaciones, utilizados en el proyecto.

La clase contiene los atributos "spaceShipA", "spaceShipB", "bullet" y "meteor", que son imágenes representadas como objetos `BufferedImage`.

También incluye atributos de animaciones como "bulletExplosionAnimation", "spaceShipExplosionAnimation" y "laserSparkleAnimation", que son matrices de objetos `BufferedImage` que representan los frames de una animación.

El constructor "AssetsManager" se encarga de llamar al método "loadAssets()" para cargar los recursos gráficos.

El método "loadAssets()" utiliza el método "spriteLoader()" para cargar las imágenes estáticas y animaciones a partir de las rutas proporcionadas.

El método "spriteLoader()" carga una imagen utilizando la clase "ImageIO" y la ruta proporcionada. Devuelve la imagen cargada como un objeto BufferedImage.

La clase también proporciona métodos getter para acceder a los recursos gráficos cargados.

9.3 - DynamicVisualObject

```
package visual;

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 representa un objeto visual dinámico en el proyecto.

El método updatePosition actualiza la posición del objeto mediante la asignación de futurePosition a position. Luego, establece futurePosition a null.

El método updateRotation actualiza la rotación del objeto en función de una dirección específica. Utiliza el método setDirection de VisualGameModel (presumiblemente una instancia de PhysicsEngine) para calcular el nuevo ángulo de dirección del objeto.

El método calculateNewPosition calcula la nueva posición futura del objeto utilizando el método calculatePosition de VisualGameManager (también presumiblemente una instancia de PhysicsEngine). Utiliza la posición actual, el ángulo actual y la velocidad del objeto.

El método run se ejecuta en un hilo mientras el objeto está vivo. Verifica si futurePosition no es nulo y notifica a VisualGameManager mediante una instancia de NotificationMsg que la posición del objeto debe actualizarse.

9.3.1 - SpaceShip

```
package visual;
import java.awt.image.BufferedImage;
import maincontroller.gameinfo.Team;

public class SpaceShip extends DynamicVisualObject {
    // * Constructor
    public SpaceShip(int id, BufferedImage skin, Position position,
float life, int accountId, VisualGameManager visualGameManager,
        int playerNumber, Animation deadAnim, Animation spawnAnim,
Team team, double velocity, double angle) {
        super(id, skin, position, life, accountId, visualGameManager,
playerNumber, deadAnim, spawnAnim, team, velocity, angle);
    }

    // * Methods
    @Override
    public void run() {
    }
}
```

La clase SpaceShip es una subclase de DynamicVisualObject y representa una nave espacial en el proyecto.

La clase SpaceShip hereda de la clase DynamicVisualObject y no agrega nuevos atributos a los ya existentes.

El constructor de la clase invoca al constructor de la clase padre DynamicVisualObject para inicializar los atributos heredados.

La clase no proporciona una implementación para el método run. Esto significa que utiliza la implementación heredada del método de la clase DynamicVisualObject. El método run de la clase base está vacío, por lo que en este caso, la nave espacial no realiza ninguna acción específica durante la ejecución del hilo.

En resumen, la clase SpaceShip representa una nave espacial en el juego y hereda las funcionalidades y características de un objeto visual dinámico de la clase

DynamicVisualObject. Puede tener su propia lógica y comportamiento al sobrescribir los métodos de la clase base según sea necesario.

9.3.2 - Bullet

```
package visual;

import java.awt.image.BufferedImage;
import maincontroller.gameinfo.Team;

public class Bullet extends DynamicVisualObject {

    // * Constructor
    public Bullet(int id, BufferedImage skin, Position position, float life, int accountId, VisualGameModel vgm, int playerNumber, Animation deadAnim, Animation spawnAnim, Team team, double velocity, double angle) {
        super(id, skin, position, life, accountId, vgm, playerNumber, deadAnim, spawnAnim, team, velocity, angle);
    }

    // * Methods
    @Override
    public void run(){
        while(getIsAlive()){
            if(getFuturePosition() == null){
                calculateNewPosition();
                getVisualGameModel().notifyToVGC(new
NotificationMsg(NotificationType.positionUpdate, this));
            }
        }
    }
}
```

La clase Bullet es una subclase de DynamicVisualObject y representa una bala en el proyecto.

La clase Bullet hereda de la clase DynamicVisualObject y no agrega nuevos atributos a los ya existentes.

El constructor de la clase invoca al constructor de la clase padre DynamicVisualObject para inicializar los atributos heredados.

La clase sobrescribe el método run. Dentro del método, se ejecuta un bucle mientras la bala esté viva (getIsAlive() sea verdadero). En cada iteración del bucle, verifica si futurePosition es nulo. Si es nulo, significa que la bala aún no ha calculado su nueva posición. Luego,

llama al método calculateNewPosition para calcular la nueva posición futura de la bala utilizando el motor de física del juego. Después de calcular la posición, se utiliza getVisualGameModel() para notificar al VisualGameModel que la posición de la bala debe actualizarse mediante una instancia de NotificationMsg.

En resumen, la clase Bullet representa una bala en el juego y hereda las funcionalidades y características de un objeto visual dinámico de la clase DynamicVisualObject. El método run de la clase sobrescrita se encarga de calcular y actualizar continuamente la posición de la bala en el juego y notificar al VisualGameModel sobre los cambios en la posición.

9.3.3 - Asteroid

```
package visual;
public class Asteroid extends DynamicVisualObject {
    private double rotationAngle;
    // * Constructor
    public Asteroid(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, team, velocity, angle);
        rotationAngle = 0;
    }

    // * Methods
    @Override
    public void drawObject(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        AffineTransform at = g2d.getTransform();
        g2d.translate(getPosition().getxPos(), getPosition().getyPos());
        g2d.rotate(rotationAngle);
        g2d.drawImage(getSkin(), at, null);
    }

    @Override
    public void run(){
        while(getIsAlive()){
            if(getFuturePosition() == null){
                calculateNewPosition();
                rotationAngle += VisualConstants.rotationVelocity;
                getVisualGameModel().notifyToVGC(new
NotificationMsg(NotificationType.positionUpdate, this));
            }
        }
    }
}
```

```
    }  
}
```

La clase Asteroid es una subclase de DynamicVisualObject y representa un asteroide en el proyecto.

La clase Asteroid hereda de la clase DynamicVisualObject y agrega un nuevo atributo llamado rotationAngle que representa el ángulo de rotación del asteroide.

El constructor de la clase invoca al constructor de la clase padre DynamicVisualObject para inicializar los atributos heredados. Además, inicializa el atributo rotationAngle a 0.

La clase sobrescribe el método drawObject. Dentro del método, se obtiene el contexto gráfico Graphics2D y se guarda la transformación actual en una variable at. Luego, se realiza una serie de transformaciones en el contexto gráfico para posicionar y rotar el asteroide según su posición y rotationAngle. Finalmente, se dibuja la imagen (skin) utilizando el contexto gráfico actualizado.

La clase sobrescribe el método run. Dentro del método, se ejecuta un bucle mientras el asteroide esté vivo (getIsAlive() sea verdadero). En cada iteración del bucle, verifica si futurePosition es nulo. Si es nulo, significa que el asteroide aún no ha calculado su nueva posición. Luego, llama al método calculateNewPosition para calcular la nueva posición futura del asteroide utilizando el motor de física del juego. Después de calcular la posición, se actualiza rotationAngle agregando VisualConstants.rotationVelocity para lograr un efecto de rotación. Luego, se utiliza getVisualGameModel() para notificar al VisualGameModel que la posición del asteroide debe actualizarse mediante una instancia de NotificationMsg.

En resumen, la clase Asteroid representa un asteroide en el juego y hereda las funcionalidades y características de un objeto visual dinámico de la clase DynamicVisualObject. El asteroide se dibuja en la pantalla con una imagen rotada según el ángulo de rotación rotationAngle. Durante la ejecución del hilo, el asteroide calcula continuamente su posición, actualiza el ángulo de rotación y notifica al VisualGameModel sobre los cambios en la posición.

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 - APPLICACIÓN 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

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

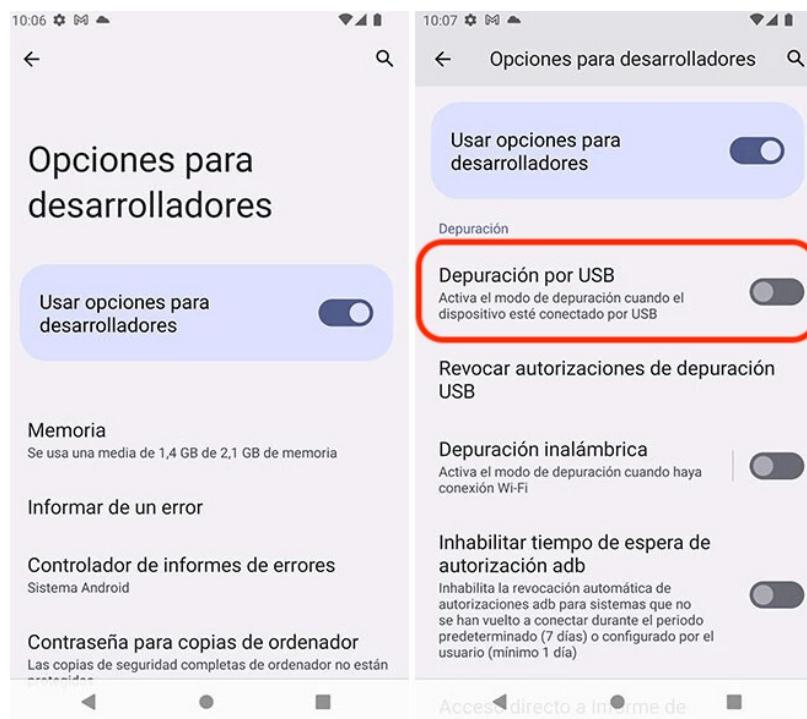


Figura 10.3 - Activación depuración por USB

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.

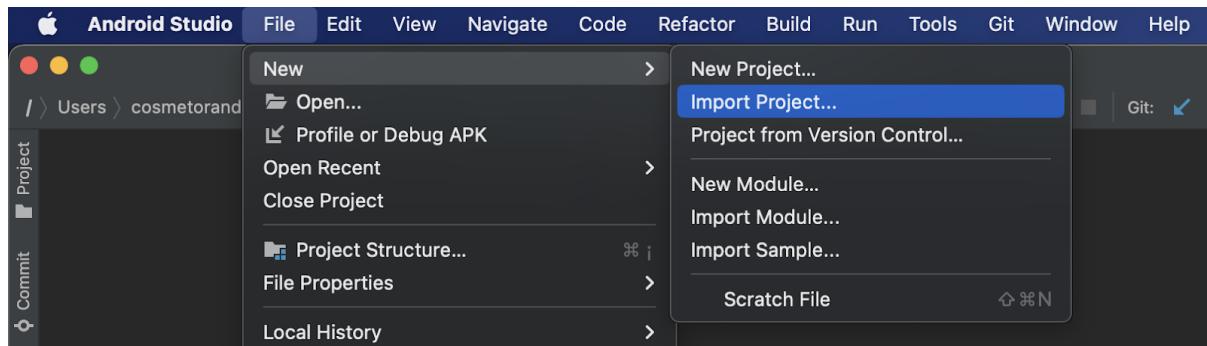


Figura 10.4 - Importar proyecto android 1

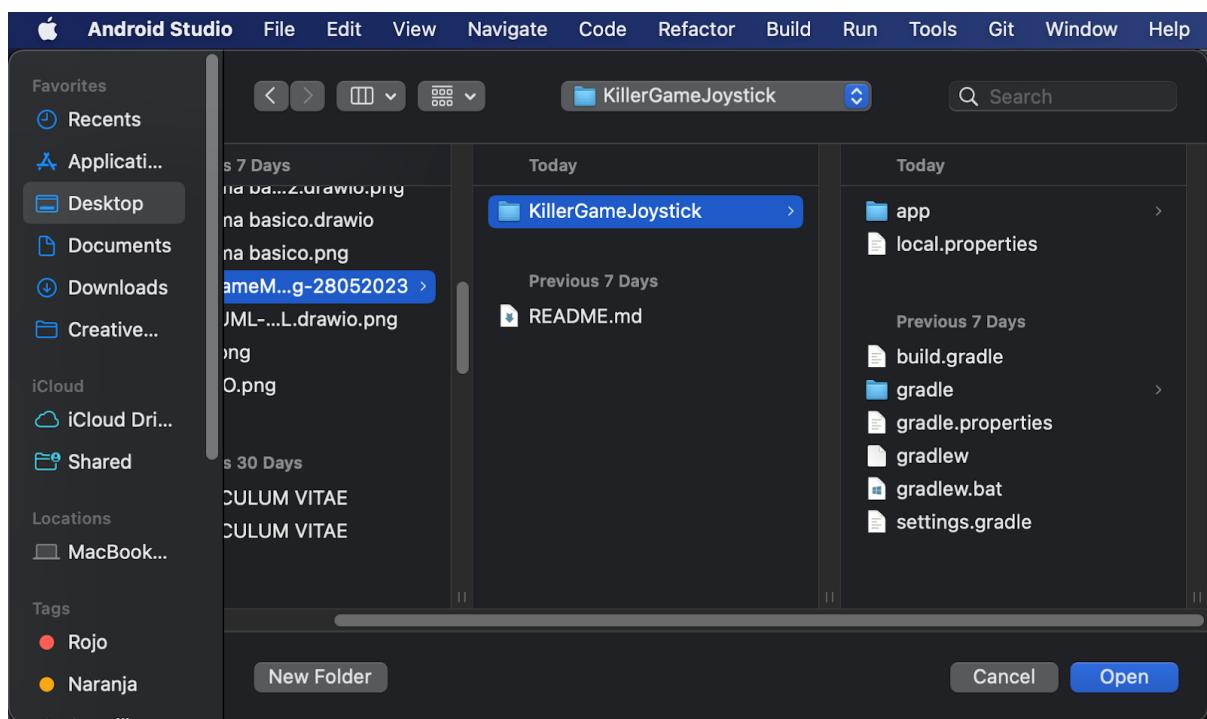


Figura 10.5 - Importar proyecto android 2

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”.

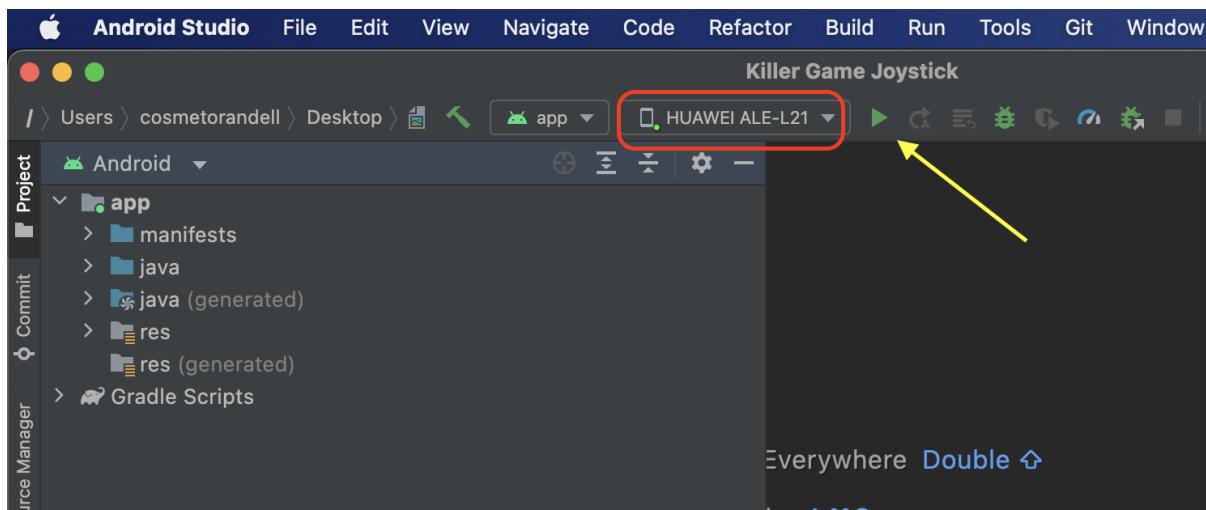


Figura 10.6 - Instalación aplicación en dispositivo

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

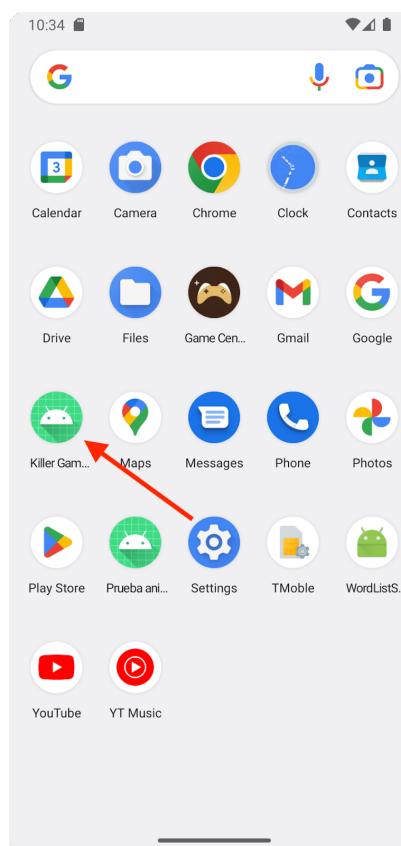


Figura 10.7 - Inicio de la aplicación móvil

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

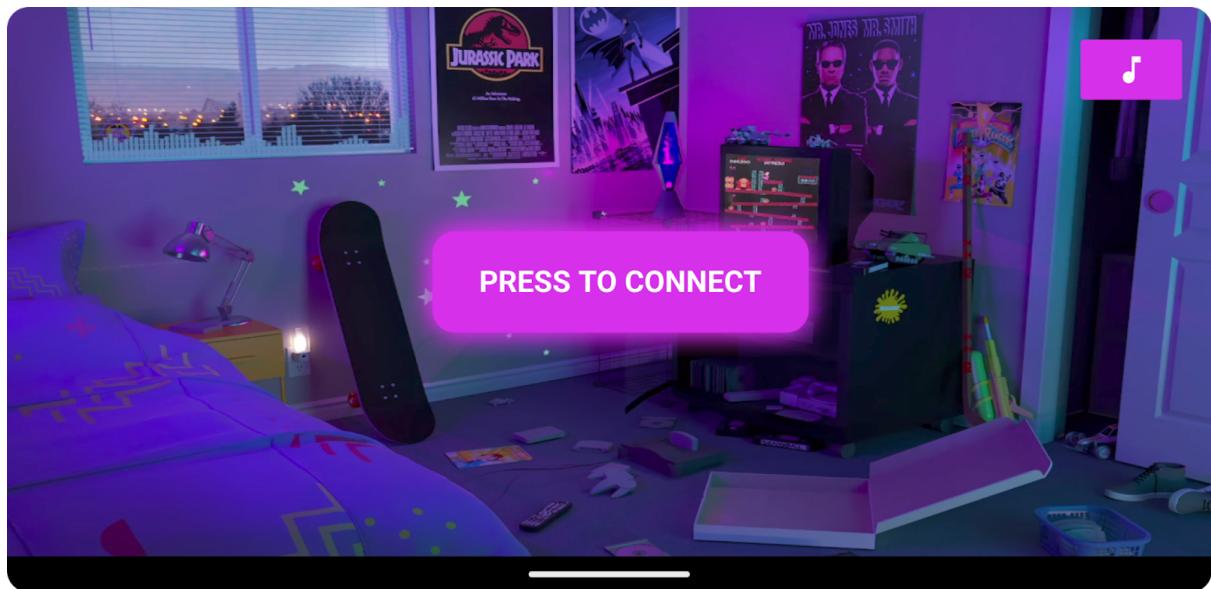


Figura 10.8 - Pantalla inicial del mando de juego

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 interactivos	

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 detallan las conclusiones del subproyecto de gestión de distribución del terreno de juego, gestión de recursos visuales del proyecto y desarrollo de objetos visuales dinámicos. El punto más importante estará en el desarrollo de la parte visual del proyecto.

11.1.2.1 - EXPECTATIVAS Y MEJORAS

Debido a la falta de objetivos concretos y la excesiva abstracción de los que teníamos, el proyecto del apartado visual y, a su vez el general, no pudo ser completamente desarrollado como se esperaba. La falta de una dirección clara y la existencia de numerosas ideas sin concretar generaron una falta de eficiencia en el trabajo realizado.

Inicialmente, el objetivo del proyecto visual era desarrollar e implementar un sistema que mostrara los objetos visuales en diversas pantallas y les permitiera moverse e interactuar. Se establecieron físicas similares a las del juego Asteroid y se planificó la inclusión de más objetos visuales, como agujeros negros y "power-ups", además de diversas mecánicas más avanzadas. Sin embargo, a medida que el proyecto avanzaba, nos dimos cuenta de que las expectativas iniciales eran difíciles de alcanzar y, por lo tanto, se tuvo que ajustar el enfoque.

A lo largo del tiempo de desarrollo, el rendimiento del equipo fue lento al principio debido al desconcierto sobre el proyecto concreto que debíamos desarrollar. La falta de claridad en los objetivos contribuyó a la falta de eficiencia en el trabajo. Conforme pasaba el tiempo, el equipo comenzó a adaptarse y rendir mejor, pero aún así no fue suficiente para alcanzar las expectativas iniciales del proyecto.

En última instancia, la falta de objetivos concretos y la falta de concreción en las ideas planteadas impactaron negativamente en el desarrollo del proyecto. Para futuros proyectos, es fundamental establecer metas claras y definir de manera precisa las funcionalidades y requisitos específicos, lo que permitirá un enfoque más eficiente y una mayor posibilidad de éxito en su implementación.

.