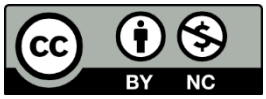
**Descubre el proceso completo de desarrollo de un videojuego utilizando el entorno gráfico JavaFX.**

# Proyecto

## Desarrollo de Aplicaciones Multiplataforma

Alberto Gutierro Jiménez
Catalin Trandafir Trandafir

ALBERTO GUTIERRO JIMÉNEZ CATALIN TRANDAFIR TRANDAFIR






## Datos del proyecto y resumen

### Resumen

Este proyecto hecho por dos estudiantes de desarrollo de aplicaciones multiplataforma en el instituto el Puig Castellar consiste en crear un videojuego utilizando la plataforma JavaFx.

Dicho videojuego, es un juego de naves en 2D con diversos modos de juego que pretenden poner a prueba las habilidades del usuario ya sea jugando solo o con varios jugadores.

### Abstract

This is a proyect developed by two students of multiplatform application development at the Puig Castellar Institute, consist on create a videogame using JavaFx platform.

This videogame is a 2D ship Game with diverse game modes that find test the user skill playing alone or with others players.

### Palabras Clave

Java FX, Videojuego 2D, Juego en Red, Programación multihilo.

# Índice

## Introducción

Para este proyecto de final de curso, los integrantes del proyecto van a realizar un juego de naves en 2D.

## Contexto y justificación del proyecto.

Como programadores y amantes de los videojuegos consideramos que el desarrollo de un videojuego puede ser una experiencia enriquecedora, la cual, nos permitirá poner a prueba todos los conocimientos que hemos adquirido durante todo el ciclo de desarrollo de aplicaciones multiplataforma.

## Objetivos

Además de la propia elaboración del juego, nos hemos propuesto una serie de objetivos para perfeccionar y aumentar las dimensiones del propio juego.

- Elaborar un modo de un solo jugador.
- Elaborar un modo de varios jugadores en red local.
- Hacer diferentes salas dentro de un servidor en red local.
- Optimizar el juego.
- Hacer un propio ejecutable del juego.
- Establecer Interfaz gráfica al servidor.

## Alcance

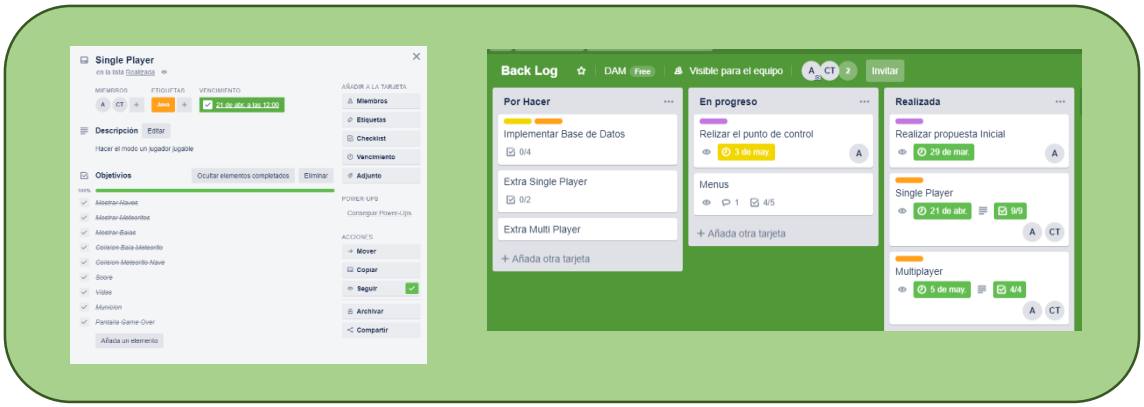
### Posibles Obstáculos

Probablemente el principal obstáculo que se pueda encontrar en la realización de este proyecto es la falta de tiempo para el desarrollo, debido a la falta de experiencia a la hora de desarrollar un proyecto de dichas dimensiones por parte de ambos integrantes del equipo encargado del desarrollo del proyecto.

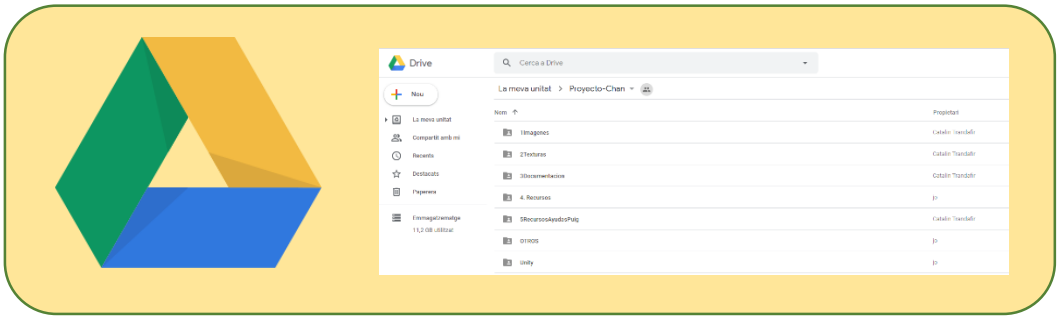
### Metodología de Trabajo

Para gestionar las dimensiones del proyecto, hemos utilizado una serie de programas para facilitarnos el trabajo.

- Trello: Trello es un software de navegador que te permite organizar todas las diferentes tareas utilizando una pizarra. (Igual que en la metodología Scrum). Además de poder colocar las diferentes tareas en una pizarra, tienes unos “checkbox” en cada tarea para ver los avances que se van haciendo.




- Google Drive: La herramienta de google drive se utiliza para poder tener diferentes archivos del proyecto en la nube, ya sean imágenes o el propio ejecutable del juego para poder descargar.



# Presupuesto

## Elementos de la estimación inicial y justificación

Para realizar el presupuesto inicial, se decidió dividirlo en tres partes: **Equipo, Instalaciones y Personal.**

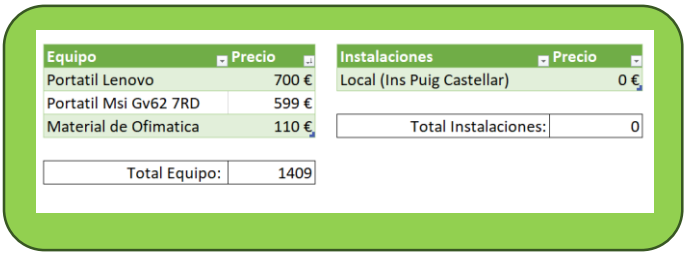
En la Parte de equipo, hemos introducido todo el material de trabajo necesario para llevar a cabo el proyecto, tanto software como hardware.

El apartado de instalaciones se compone los gastos generados por los locales y lugares de trabajo que se han utilizado durante la elaboración del proyecto.

Por último los gastos de personal donde se estima lo que cobrara los empleados por las horas de proyecto. Para hacer la estimación de lo que cobrara cada desarrollador hemos realizado una búsqueda por diferentes webs de empleo. Y hemos realizado una media de lo que deberá cobrar por todo el trabajo.

## Valoración de la viabilidad económico-financiera

Después de ver los siguientes resultados obtenidos en la elaboración del presupuesto para el proyecto, consideramos que hacer un proyecto de estas dimensiones solo es viable si los desarrolladores son lo que impulsan la idea y ya poseen los recursos necesarios para hacerlo posible.



| Equipo | Precio |
|---|---|
| Portatil Lenovo | 700 € |
| Portatil Msi Gv62 7RD | 599 € |
| Material de Ofimatica | 110 € |
| Total Equipo: | 1409 |

| Instalaciones | Precio |
|---|---|
| Local (Ins Puig Castellar) | 0 € |
| Total Instalaciones: | 0 |

| Personal | Precio*100 horas |
|---|---|
| Java Developer Junior | 928 € |
| Java Developer Junior | 928 € |
| Total Personal: | 1856 |

| Apartado | Precio |
|---|---|
| Equipo | 1409 |
| Instalaciones | 0 |
| Personal | 1856 |
| Total proyecto: | 3265 |

# Planificación temporal

## Planificación inicial

### *Introducción*

En este trabajo de síntesis, se ha decidido elaborar un videojuego utilizando el motor de videojuegos Unity.

### *Objetivos.*

Inicialmente se ha planteado una serie de objetivos para llevar a cabo el producto final de este proyecto, que será la realización de un videojuego utilizando el motor de juegos Unity.

- Formar al equipo de desarrollo para crear un juego en Unity.
- Elaborar el propio videojuego.
- Utilizar una base de datos para almacenar los datos recogidos en el propio juego.
- Desarrollar un modo de vario jugadores mediante red local.

### *Tecnologías*

Para desarrollar el propio juego utilizaremos el propio motor Unity acompañado por Visual Studio Code para la elaboración de los diferentes Scripts que requieran programación. Unity, además de ser el motor de juegos, incluye una nube utilizada como repositorio del proyecto.

*Logo Visual Studio Code*


*Logo Unity*

## Punto de control

Es siguiente Punto de control se realizó el día 03-05-2019.

### *Cambios respecto la planificación inicial*

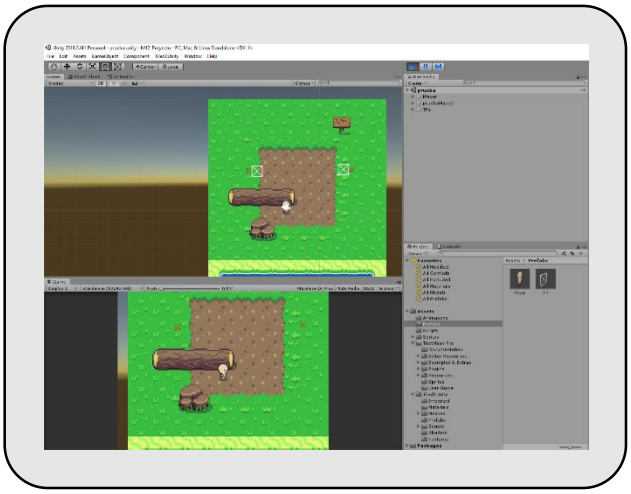
Desde que se inició el proyecto hasta el primer punto de control se estuvo desarrollando en la plataforma Unity, pero tras ver que era una plataforma que no ofrecía lo que buscaba el equipo en el inicio del proyecto, se decidió cambiar la plataforma donde se desarrollara y el tipo de juego que se elaborara.

### *Consecuencias de los cambios respecto los objetivos y desarrollo del proyecto*

Debido a este cambio, las primeras semanas del proyecto donde nos centramos en la formación de Unity y la primera parte del videojuego (la cual consistía en un entorno de pruebas donde se testearían todas las funcionalidades que podrías hacer dentro del juego en Unity) puede que haya generado unos retrasos en el desarrollo en la nueva plataforma, debido al cambio tan radical de rumbo que ha llevado el proyecto.

### *Situación del proyecto en el punto de control*

Dentro de Unity hemos elaborado un espacio de pruebas donde se testean las diferentes funcionalidades que hubiese tenido el juego si se continuase desarrollando en Unity. Dentro de este espacio podemos ver nuestro personaje moverse por un mapa previamente creado, se puede ver la animación de caminar, las colisiones con los diferentes objetos y un "teletransporte" para moverse entre los 2 cuadros que se aprecian en la siguiente imagen.



Actualmente el videojuego de JavaFx, posee un modo “singlePlayer” donde el jugador debe destruir diferentes meteoritos para que no colisionen con la nave y un modo “multijugador” (por red), donde desde diferentes ordenadores dentro de la misma red local pueden luchar entre ellos en un modo de todos contra todos.

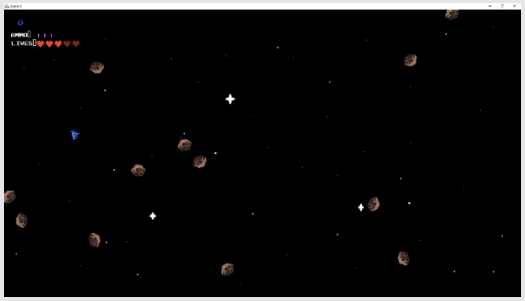


*Imagen tomada durante una partida en modo un jugador*

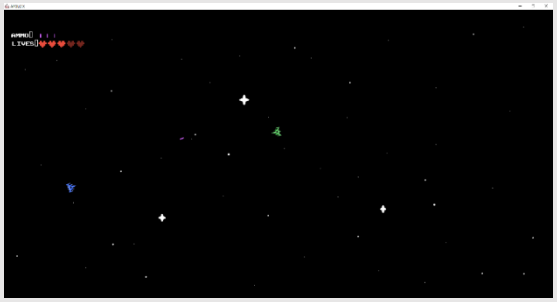


*Imagen tomada durante una partida en modo multijugador*

## Planificación final

Finalmente, el proyecto sigue el rumbo que se estableció en el punto de control que se hizo el día 03-05-2019. Por lo tanto, finalmente el proyecto será en JavaFx y se va a hacer un juego de naves en 2D.

## Fases del proyecto

El desarrollo del proyecto contara con 5 Fases de desarrollo diferentes.

- Análisis: En la Fase de análisis buscamos establecer las dimensiones del proyecto y definir las diferentes funcionalidades que tendrá.

- Diseño: Dentro de la fase de diseño estableceremos como se estructurara el código que desarrollemos además de establecer cómo se realizaran las diferentes pruebas. Además de la creación de algunos diagramas de clases y estado “básicos” para entender a que nos enfrentamos.

- Desarrollo: En la fase de desarrollo se explicara todo lo relacionado con la lógica del juego.

- Pruebas: En la fase de pruebas, es donde se ven reflejadas las diferentes pruebas que se han elaborado del programa

# Análisis y Diseño

## Funcionalidades

Como usuario, dentro del juego vas a tener diferentes funcionalidades.

Primero de todo y las dos funciones más básicas que tienes es jugar solo al propio juego o con varias personas.

Dentro de la funcionalidad que es jugar, tenemos diferentes funcionalidades. Estas funcionalidades consisten en las funciones que puede hacer la nave. La nave que se utiliza para jugar, tiene las siguientes funcionalidades:

- Moverse:
  - W o ↑: te permite mover la nave hacia arriba.
  - A o ←: te permite mover la nave hacia la izquierda.
  - S o ↓: te permite mover la nave hacia abajo.
  - D o →: te permite mover la nave hacia la derecha.
- Disparar
  - Soltando cualquier botón del ratón podemos disparar la nave.

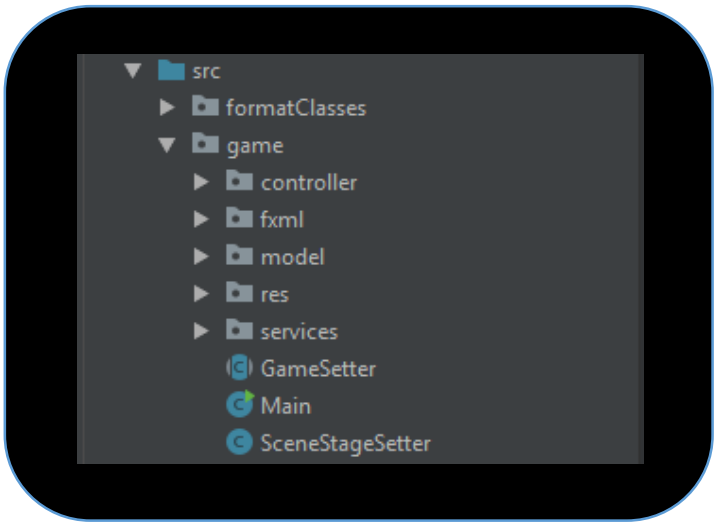
Otra función que contiene es buscar/crear salas, esta función la utilizamos únicamente en el multijugador para buscar o esperar a otros jugadores que estén conectados en el servidor para jugar.

Por parte de los administradores, existe una funcionalidad adicional que consiste en la gestión del servidor, utilizada para ejecutar o parar el servidor.

## Arquitectura

### *Descripción general*

La arquitectura del proyecto se puede resumir en que utiliza la arquitectura MVC, también conocida como la arquitectura **Modelo-Vista-Controlador.**




La arquitectura MVC, consiste en la división del proyecto en 3 ramas.

El **Modelo** representa la información con la cual se opera en el programa, en nuestro caso, los diferentes objetos que hay en el juego (Nave, Bala, Meteorito, etc.).

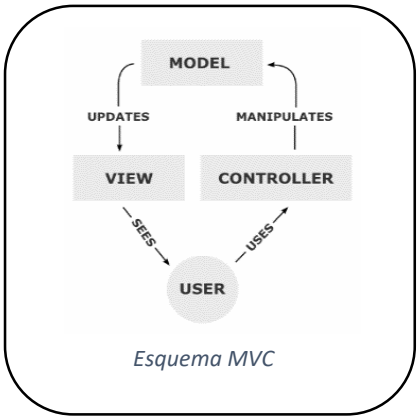
El **controlador** es el encargado de manipular los diferentes objetos, ya sea para obtener datos o modificarlos.

La **Vista** es la interfaz gráfica. Se encarga de mostrar los objetos (situados en el modelo), y te permite interactuar con ellos.

Además de utilizar esta arquitectura, hemos añadido 2 carpetas que complementan al modelo y el controlador, llamadas **Res** y **Services.**

Dentro de **Res** se almacena todo el contenido multimedia que contiene el juego.

Por último, en **Services** se guardan los diferentes servicios utilizados en el controlador para gestionar las naves y los meteoritos.



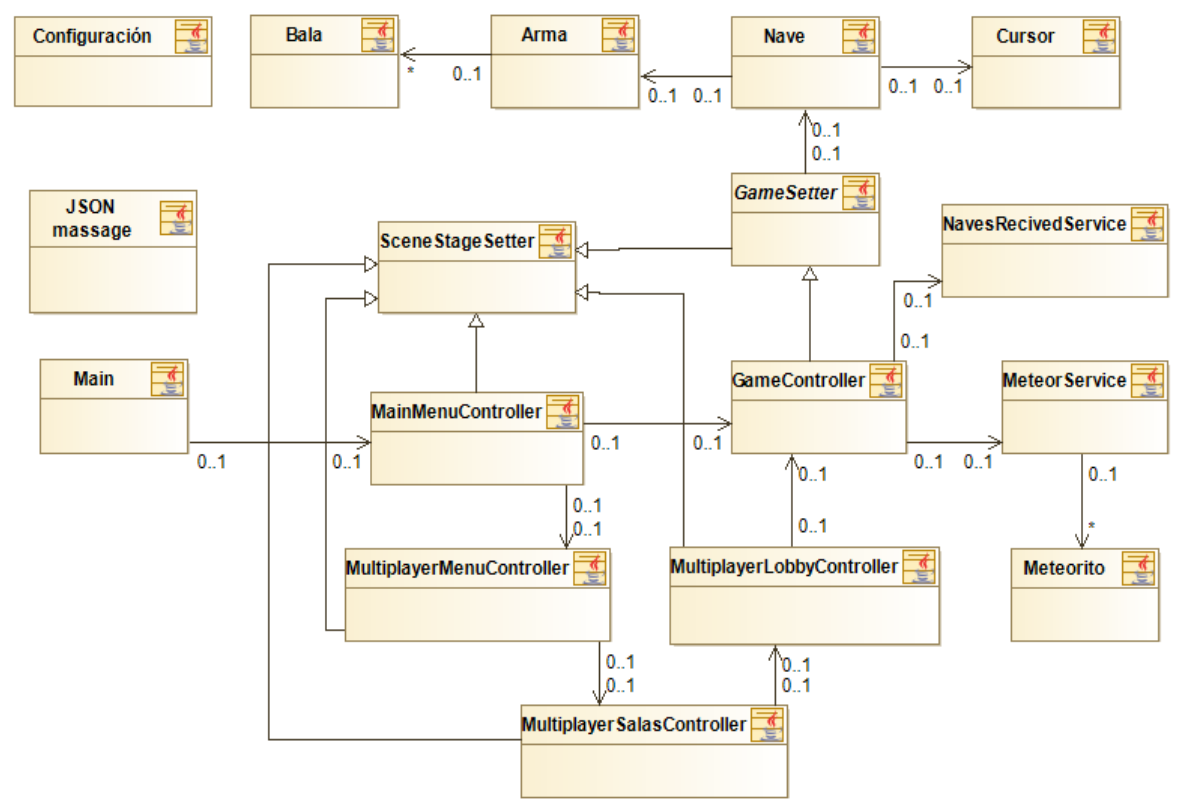
*Esquema MVC*

### *Diagramas*

Los siguientes diagramas son los utilizados para estructurar el código del videojuego.

Primero de todo tenemos dos diagramas de clases, uno que muestra la estructura del juego y otro la estructura del servidor. También, se ha creado un diagrama de estados para el juego.

## Diagramas Aplicación



*Mapa conceptual de la aplicación*

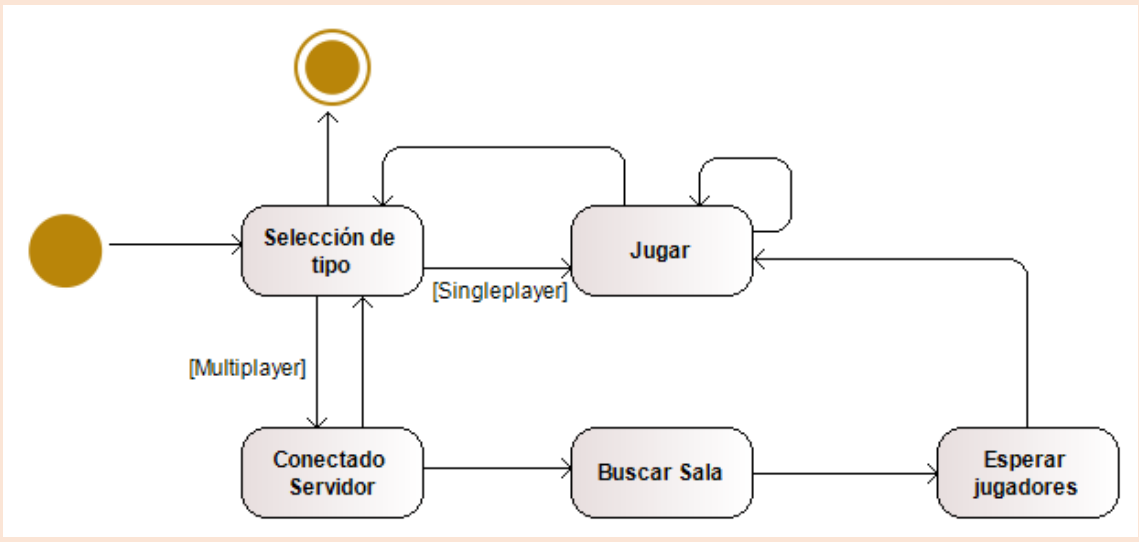



*Diagrama estado Juego*

## Diagrama estado nave

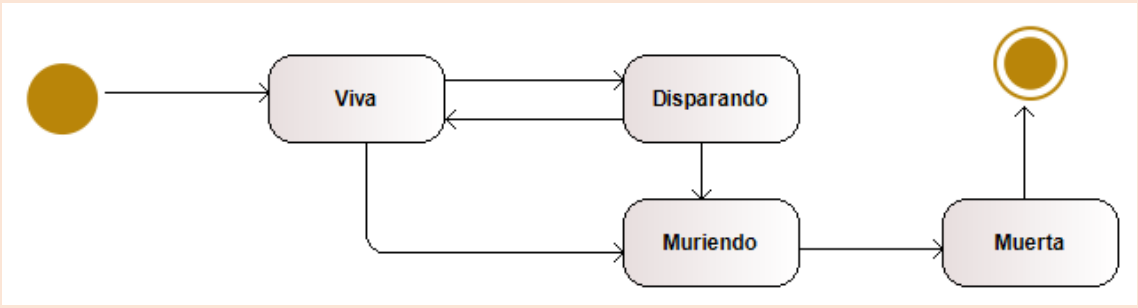



*Diagrama estado Nave*

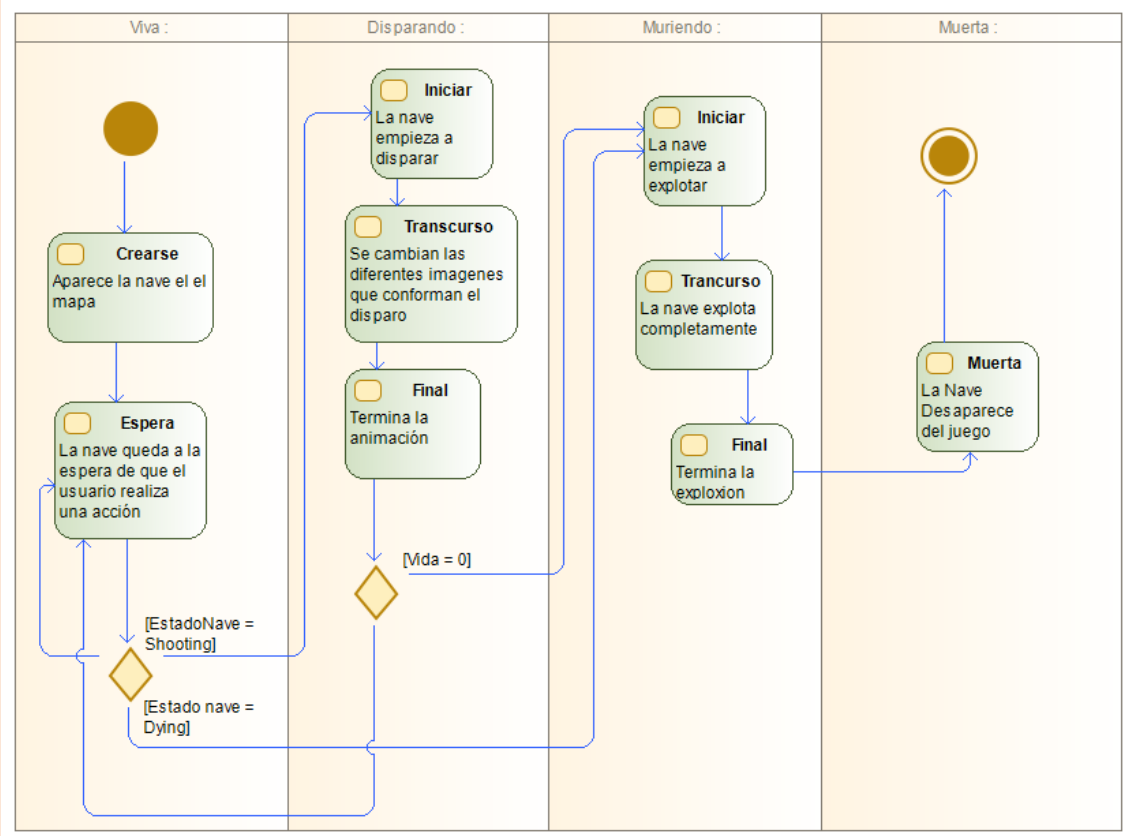



*Diagrama de actividad de una nave*

## Diagrama de meteoritos

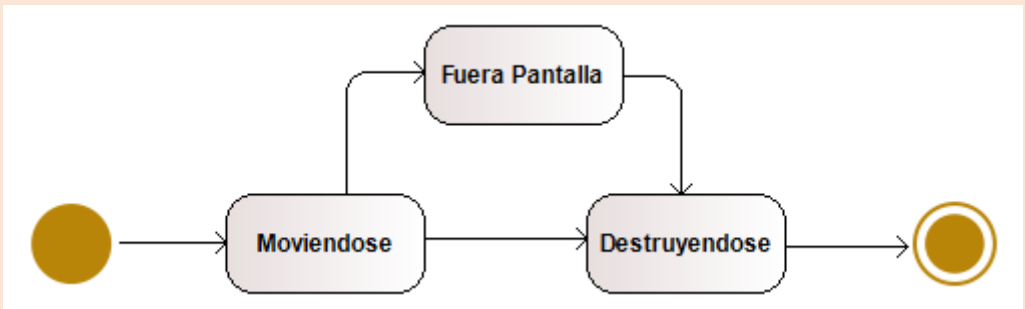



*Diagrama de estados del meteorito*

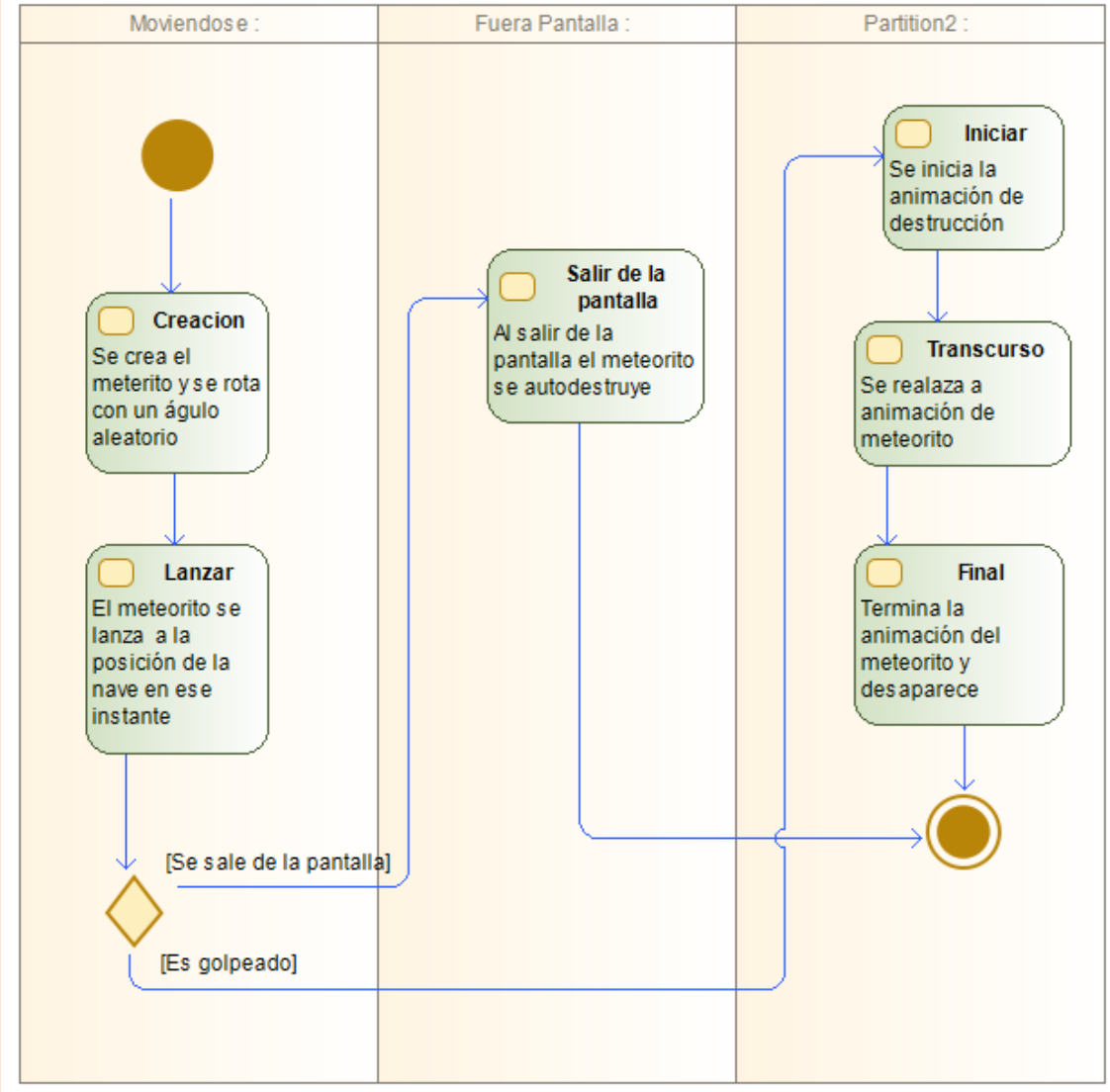
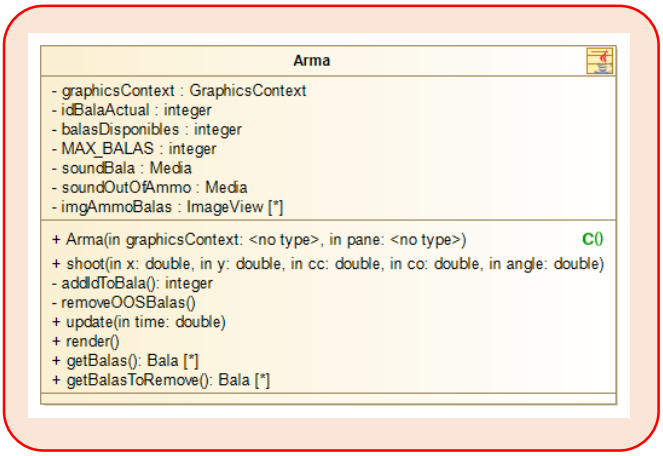



*Diagrama de actividad del meteorito*

## Seguridad

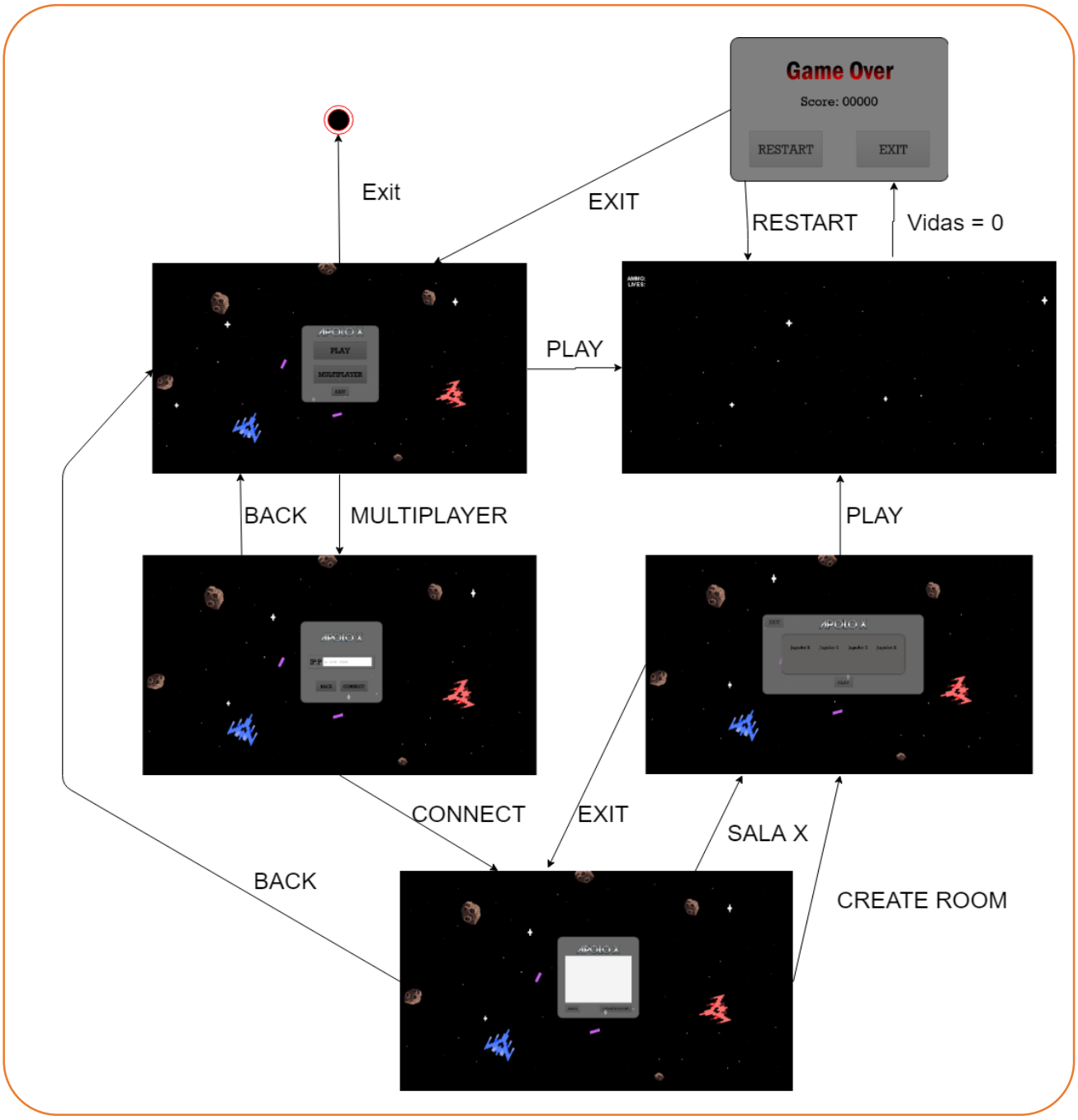
Para lograr hacer que el código del juego tenga los mínimos fallos de seguridad y para que los desarrolladores no puedan modificar campos fuera de los objetos sin pasar antes por un método “setter”, la gran mayoría de los campos se han declarado como privados.

Gracias a declarar los campos como privado, únicamente podemos modificarlos dentro del objeto.



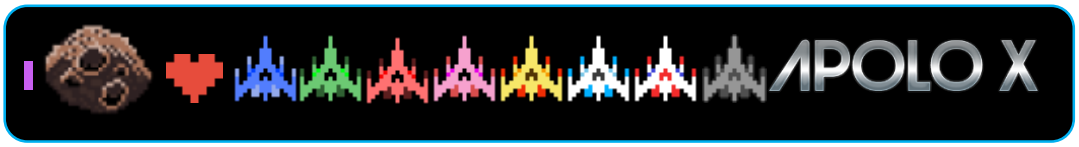

## Interfaz

La forma más sencilla de explicar el funcionamiento de la interfaz es utilizando un mock up de la aplicación. Los mock Ups son fotomontajes que permiten a los diseñadores gráficos mostrar a cliente como quedara el diseño de la aplicación. En este caso, el mock up mostrara las diferentes pantallas de la aplicación y como se relacionaran entre ellas.


Game Over
Score: 00000
RESTART
EXIT
Exit
EXIT
RESTART
Vidas = 0
PLAY
PLAY
BACK
MULTIPLAYER
PLAY
CONNECT
EXIT
SALA X
BACK
CREATE ROOM

## Modelos.

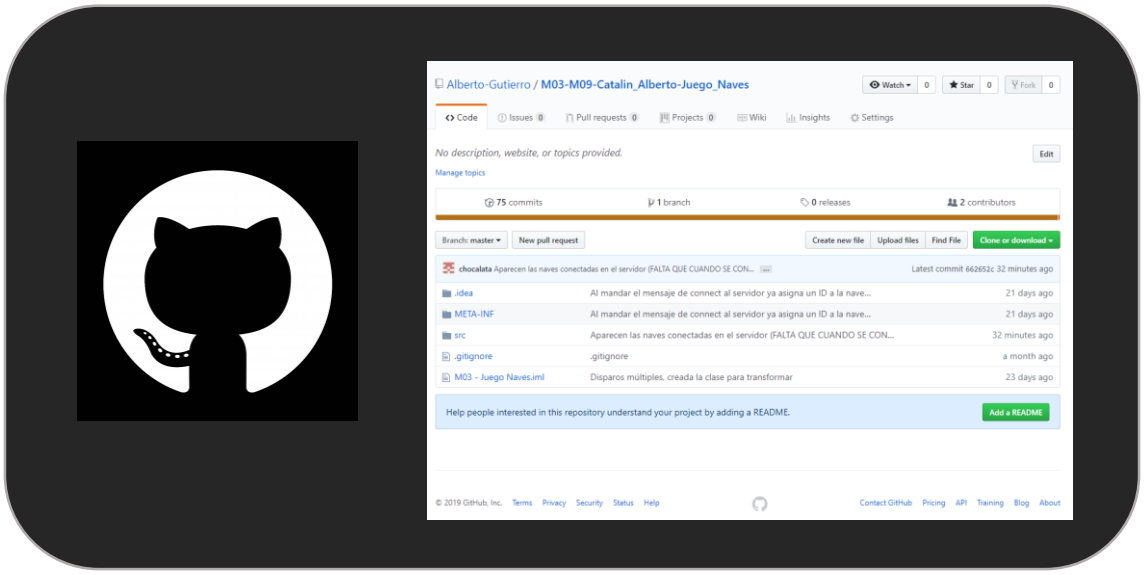
Las texturas utilizadas dentro del juego (sin contar animaciones) son las siguientes.



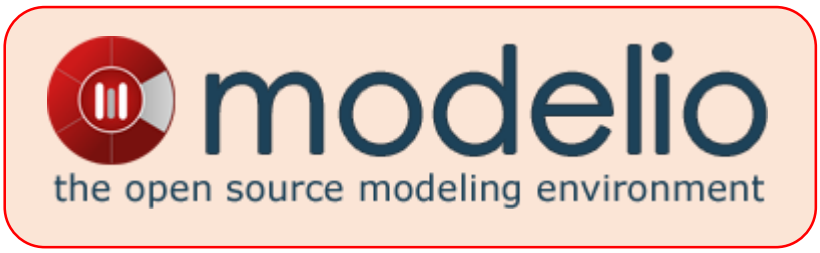

## Tecnología

Para realizar este videojuego, se han utilizado las siguientes herramientas de desarrollo.

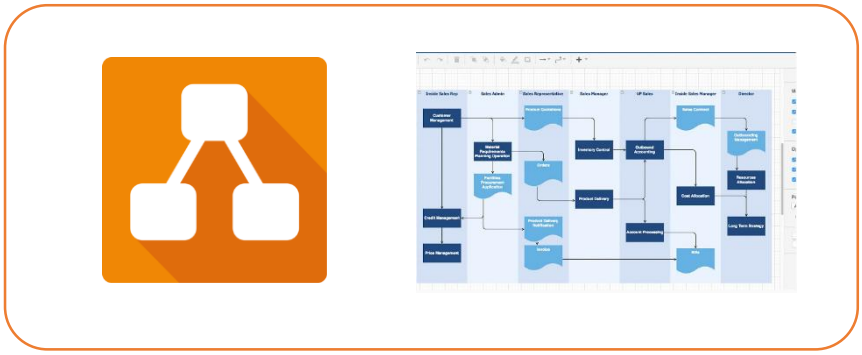
- GitHub: GitHub es el repositorio web utilizado para guardar el código con el que se está trabajando. Dentro de tus repositorios, puedes tener diferentes Ramas de producción, ayudándote por ejemplo a dividir tu proyecto en los entornos DES – PRE – PRO, para así ayudante a tener mayor control sobre tu código fuente.
  Además, GitHub ofrece muchas facilidades y muchos programas tienen soporte para esta herramienta.
  Por último, funciona con la tecnología GIT, por lo que se pueden utilizar comandos para gestionar tus repositorios.



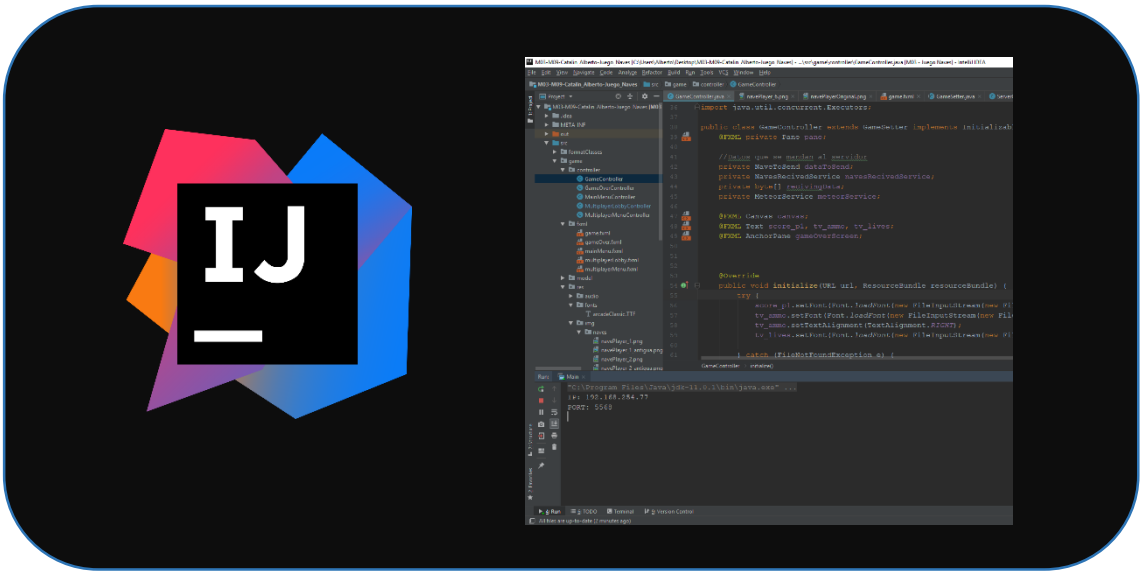

- Modelio: Modelio es una herramienta utilizada para el desarrollo de diagramas UML. Es un software de código abierto. Principalmente se utiliza en las primeras fases de desarrollo para establecer la arquitectura que tendrá tu programa. Dentro de este proyecto se utilizara para hacer diagramas de clases y estado.



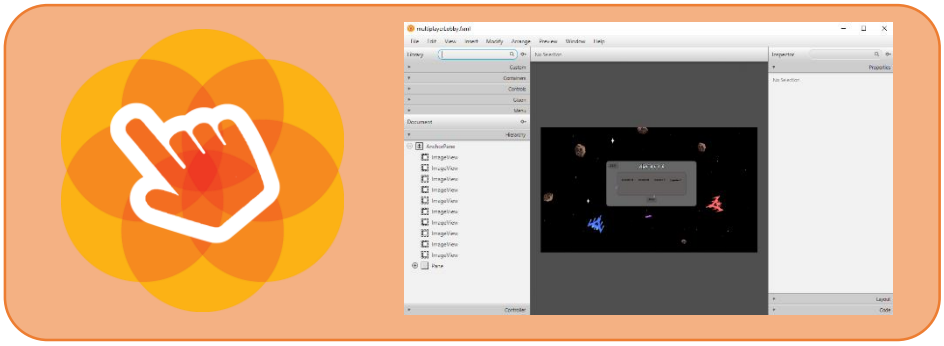

- Draw.io: Draw.io es la herramienta google que te permite dibujar todo tipo de diagramas, ya sean diagramas UML o un dibujo cualquiera. Dentro del proyecto esta herramienta la utilizaremos para hacer el mock up de la aplicación.



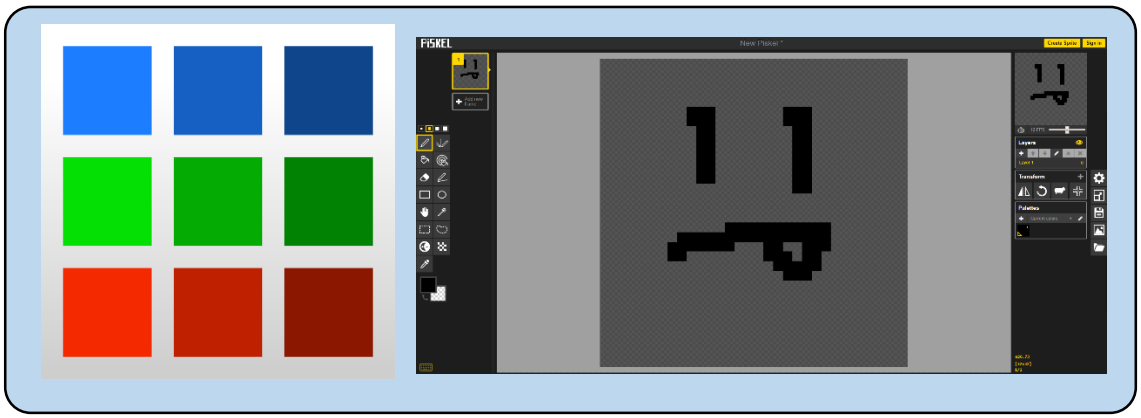
- IntelliJ IDEA: Es la plataforma donde se ha desarrollado todo el código generado, IntelliJ es el programa donde se ha trabajado durante todo el curso, este programa incluye su propio terminal, facilidades de configuración, compatibilidad con GitHub y una maravillosa interfaz oscura.



- SceneBuilder: SceneBuilder es la herramienta que utilizamos para modificar los archivos fxml del juego. Gracias a esta herramienta podemos modificar las diferentes pantallas del juego mediante una interfaz gráfica.



- PiskelAPP: PiskelAPP es la herramienta que se ha utilizado para elaborar las diferentes texturas que se utilizan dentro del juego. Esta interesante herramienta de navegador ofrece crear tus propias animaciones en formato pixelArt.



- Exe4J: Exe4j es la aplicación que utilizamos para convertir nuestro ejecutable en .jar un un archivo ejecutable .exe

# Desarrollo

## Juego

### *Controlador*

Los controladores son las clases de java que se están relacionadas con las diferentes pantallas del juego y son las encargadas de dar lógica.

Dentro del juego, cada controlador tiene una función diferente.

Desde el controlador del menú principal podemos acceder al juego o ir al menú del multijugador. Para poder acceder entre pantallas tenemos que definir un Stage y un scene.

El Stage es toda la ventana del programa mientras que el scene es todo lo que está dentro del Stage.

Dentro del proyecto se utiliza una clase SceneStageSetter que es extendida por todos los controladores. Dentro de esta clase únicamente guardamos el Stage y el scene para así optimizar el código.

Para acceder a otra pantalla primero de todo debemos cargar el nuevo fxml, después instanciamos su controlador y la mostramos.

Desde el MultiplayerMenuControler te conectas al servidor, para hacer eso posible se tiene que introducir una IP y un Puerto. Con estos datos enviaremos un paquete a esa IP y puerto, si los datos son correctos, entra al menú de salas del servidor.

Debido a que se pueden ocasionar errores al intentar acceder al servidor, se han colocado diferentes alertas para tratarlos.

Dentro de nuestro controlador creamos un proceso que envía al servidor un paquete para preguntar cuántas salas hay creadas, al recibir las salas creadas enviamos una petición al proceso principal para que dibuje las Salas. Esto se debe a que un proceso que no sea el principal no puede modificar el fichero fxml. En el caso de que queramos entrar a una sala, se enviara un paquete entrar en la sala, dependiendo el estado de la sala podremos acceder a ella o nos mostrara una alerta por el cual no podemos entrar. También como usuario se puede crear una sala. Al crear la sala envías al servidor un paquete para crearla, recibes el id y entras en ella.

Una vez que accedemos a la sala se enviara cada cierto tiempo un paquete para recibir los jugadores que hay actualmente. Al recibir los datos, dibujamos a cada jugador dentro de la sala. Por último pulsando el botón “Play” empiezas una partida en el modo multiplayer.

GameControler es el controlador más importante de todo el programa. Dentro de este controlador está el funcionamiento de una partida en modo un jugador, multijugador y además un modo espectador que te permite seguir viendo la partida una vez hayas perdido. Al iniciase, comprueba que tipo de partida se va a jugar.

Si la partida es de un solo jugador creamos los meteoritos, después simplemente lo que hace es actualizar los datos del meteorito y de la nave, comprobar sus colisiones y dibujar ambos objetos. Al morir te muestra pantalla de fin de partida, te muestra tu puntuación y te permite reiniciar la partida o volver al menú de inicio.

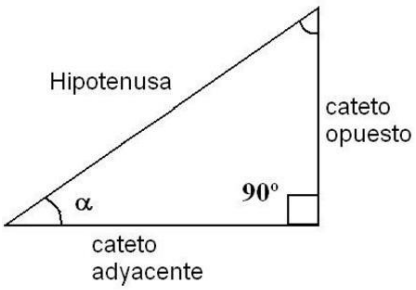
En el caso de que la partida este en el modo multijugador, dibuja tu nave, las naves de los enemigos, sus balas y en el caso de ser destruida una nave, su animación. Lo explicado anteriormente funciona de la siguiente manera:

- Transformamos los datos necesarios para la comunicación de nuestra nave en formato JSON.
- Mandamos esos datos al servidor y luego recibimos otro JSON con los datos de todas las naves jugando en esa sala.
- Una vez tenemos esos datos dibujamos las naves y balas en sus respectivos lugares.
- Luego, en el caso de no haber perdido la partida, actualizamos los datos de nuestra nave, los dibujamos y los volvemos a mandar al servidor.
- En el caso de haber perdido, te quedas en el modo espectador recibiendo solo los datos de las demás naves.
- Una vez acaban todos los jugadores recibimos un paquete del servidor decidiendo que la partida se ha terminado y nos envía al lobby.

### *Modelos*

El juego posee principalmente dos modelos, la nave y el meteorito.

La Nave, está compuesta por su arma y el cursor del usuario. La clase Cursor lo utilizamos para saber en qué posición de la pantalla se encuentra nuestro cursor y así poder hacer que la nave siempre este apuntando a la posición que se encuentra el Cursor. Para hacer eso posible utilizamos las coordenadas X e Y de la Nave y el cursor. Con estos datos podemos formar la hipotenusa de un triángulo rectángulo, a partir de ahí y aplicando la fórmula de la arco tangente sacamos el ángulo que nos interesa para poder rotar la imagen.



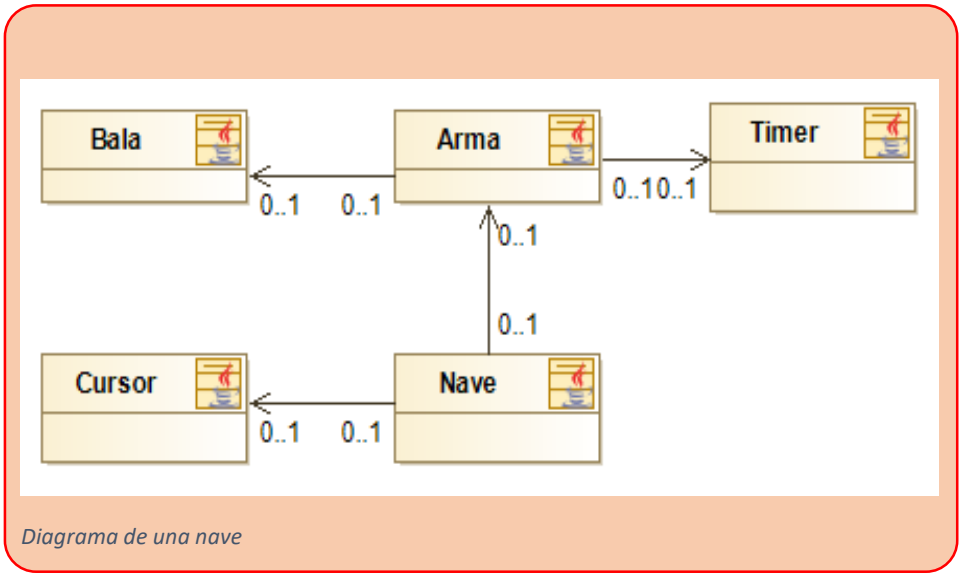




*Diagrama de una nave*

El Meteorito tiene toda la información que está enfocada al meteorito. En este caso, a diferencia de la nave, cada meteorito que se crea dentro del juego se crea con un ángulo aleatorio y se rota la imagen con dicho ángulo.

Para que aparezca el meteorito en diferentes bordes la pantalla generamos cuatro campos que representaran cada borde de la pantalla, sus valores son: 0, 1, 2 y 3. A partir de ahí sacamos un número aleatorio entre el 0 y el 3, dependiendo el número, aparecerá en un borde u otro de la pantalla. Una vez tenemos el borde modificaremos las los ejes X e Y del meteorito para que comience su recorrido desde ahí.

Además, dichos meteoritos siempre saldrán hacia la dirección en la que se encuentra la nave justo en el momento en el que se lanzan. Para hacerlo posible, utilizamos la misma fórmula trigonométrica que utilizamos para que las balas se muevan (explicado en el siguiente párrafo).

Por último, las balas. Cada nave tendrá como máximo 3 balas que se irán reponiendo cada 0,75 segundos. Dichas balas se crearan con el mismo ángulo que tiene la nave en el momento de dispararla. Para mover las balas en el eje X se utiliza la siguiente fórmula: $\cos\left(\frac{cc}{\sqrt{cc^2+co^2}}\right) * v$ para mover la Bala en el eje Y utilizaremos la siguiente fórmula: $\sin\left(\frac{co}{\sqrt{cc^2+co^2}}\right) * v$ donde “cc” significa cateto Continuo, “co” significa cateto opuesto y “v” significa la velocidad de la bala.

Para comprobar las colisiones tenemos una clase CollisionRectangle propia que extiende de la clase Rectangle de Java que lo único que contiene es un método para asignar todas las posiciones necesarias y así optimizar el código.

### *Vista*

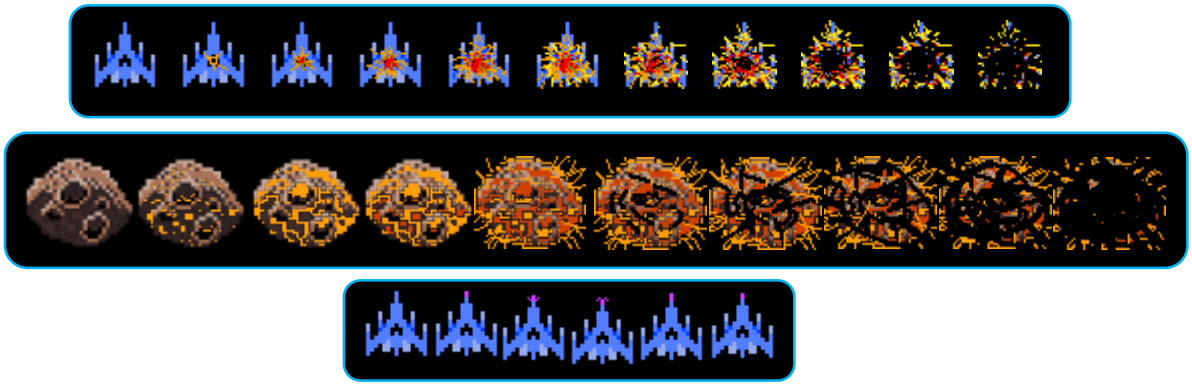
Todas las vistas que tienes el juego dependen de un controlador, el controlador es una clase de Java que se encarga desde darle lógica a los elementos cómo de crear y modificar elementos que pantalla. El controlador que tiene tu vista la defines con un atributo en el nodo raíz.

Las vistas simplemente son documentos .fxml. Dentro de las vistas hay creados paneles que contienen todos los elementos que poseen los distintos menús que aparecen en el juego. Para elaborar el fondo, simplemente se ha establecido un fondo de pantalla y se han añadido diferentes imágenes de naves meteoritos y balas. Java, a la hora de leer el archivo sigue un orden secuencial, por lo tanto, los primero que añades será lo primero en dibujarse. Al seguir un orden secuencial primero se ha dibujado el fondo de pantalla, posteriormente los elementos que decoran el fondo (naves, meteoritos, balas) y por último los menús correspondientes.

### *Animaciones.*

Dentro del juego, se han creado 3 animaciones diferentes. La animación de disparo de una nave, la animación de muerte y la destrucción del meteorito. Gracias a estas animaciones podemos obtener un mayor realismo y una mejor sensación de juego para el usuario.

Para hacer que funcionen las animaciones, se han creado diferentes estados a las naves y a los meteoritos. Además de una clase Animaciones que se encarga de cambiar las imágenes para dar el efecto de animación. Una vez que acaba la animación de la nave o el meteorito pasa a su siguiente estado.



## Servidor

El tipo de servido que se ha implementado es UDP (User Datagram Protocol), que se basa en el intercambio de información con el cliente (recibe datos y a partir de esos datos devuelve una respuesta u otra).

### *Interfaz*

La interfaz que tiene el servidor es solo para encender y parar el servidor con un simple botón.

### *Funcionamiento*

- El servidor lo primero que hace es ponerse a la espera de un nuevo paquete.
- Una vez lo ha recibido, mediante el uso de expresiones regulares se trata de distinta forma dependiendo de los datos que contenga. A continuación el valor entre comillas será el dato que puede contener el paquete aplicando expresiones regulares (en todos los lugares que se utilice ".+" será el id que haga referencia a una sala).
    - "Connect" o "Rooms": Esto quiere decir que el cliente se quiere conectar al servido o recibir salas (En estos dos casos se devolverá las salas).
    - "Create": Esto quiere decir que el usuario quiere crear una sala (Si la puede crear recibirá los datos necesarios para entrar en ella).
    - "^Room:.+$": En el caso de que el paquete comience por "Room:" querrá decir que el cliente se quiere unir a una sala (En este caso se devolverá la sala a la que se quiere unir).
    - "^Waiting:.+$": Cuando el mensaje comienza por "Waiting:" el servidor sabrá que hay que devolver las naves que se encuentren en la sala.
    - "^Exit:.+$": Si el servidor recibe la señal que comienza por "Exit:" hace que el cliente que lo haya mandado se salga de la sala.

- "Start": Con esta señal el servidor sabrá que la partida va a comenzar.
- "^Dead:.+$": Cuando el servidor recibe un paquete que comienza por “Dead:” comprueba si la nave se ha quitado del juego, si no se ha quitado se borran los datos de la nave para que los demás clientes dejen de dibujarla y devuelve al cliente muerto los datos de las demás naves, en el caso de que ya se haya quitado se devuelve simplemente los datos de las demás naves.
- En el caso de que no se haya cumplido ninguna de las anteriores quiere decir que se está jugando la partida y envía los datos correspondientes de la sala en la que se encuentra el jugador.

## Pruebas

### Funcional

Antes de la versión final se han hecho diferentes pruebas para comprobar que todas las funcionalidades del juego funcionan correctamente. A continuación dejo un listado de las pruebas que se han realizado.

- Test funcionamiento animaciones modo un jugador.
- Test funcionamiento salas del multiplayer.
- Test funcionamiento partida multiplayer con 2 jugadores.
- Test funcionamiento partida multiplayer con 3 jugadores.
- Test funcionamiento partida multiplayer con 4 jugadores.

### Corrección de errores

Respecto al rendimiento, se descubrió una serie elementos que generaban problemas de recursos y se optimizaron.

Primero de todo, a la hora de comprobar las colisiones se gastaban muchos recursos, haciendo que se ralentice el juego.

Este problema se debió a que para comprobar las colisiones se generan rectángulos con el tamaño de las diferentes naves balas y meteoritos. En total por cada segundo de podrían llegar a crear y sobrescribir 540 rectángulos dentro del multijugador y dentro del modo un jugador hasta 840. Para solucionar el problema, se modificó el código para siempre tener 2 rectángulos creados los cuales se les cambia la posición por la que tiene el objeto con el que se realiza la comprobación.

Otro problema de rendimiento era cuando recibías las vidas, había un método que miraba todas las naves y si era la nuestra nos asignaba las vidas que hayan llegado desde el servidor. Pues resulta que ya se hace una búsqueda para todas las naves en otro lugar, con añadir ahí que a la nuestra le asigne las vidas ya estaría.

El tercer problema que se encontró fue que al dibujar las naves que recibimos del servidor creamos nuevas imágenes por cada nave en la partida. Para arreglar ese fallo de rendimiento, lo que hacemos es tener las imágenes ya creadas anteriormente para que cuando se necesiten dibujar lo único que se hará será coger la misma imagen y modificarle la posición y lo mismo para las balas.

También, otro error que había era que anteriormente se había añadido una forma de poner reducir la velocidad de reproducción de la animación y en lugar de modificar la imagen cada fotograma, modificarla cada más dependiendo de cómo se ajustara mejor. Pero nos dimos cuenta de que al ejecutar la animación cada fotograma se veía de una manera óptima.

El último problema que tuvo el juego fue con su ejecutable. Al cerrar la aplicación, había procesos que seguían corriendo. Para solucionarlo, desde la primera clase al recibir una petición de cierre se ejecuta una función que cierra todas las ventanas y la máquina virtual de java que está corriendo.

## Conclusiones

- Elaborar un videojuego en un entorno gráfico como JavaFx, el cual no trae ninguna ventaja a la hora de elaborar un juego es un perfecto reto que puede hacer un programador una vez en la vida.

## Post Proyecto

Debido a la que el proyecto únicamente tiene en total 200 horas de trabajo hemos planteado futuras implementaciones que se irán añadiendo a lo largo del tiempo.

- Añadir un BD al juego.
- Mejorar interfaz gráfica servidor. (Visual y funcionalidades)
- Añadir un selector de naves.
- Añadir un menú de opciones.
- Añadir diferentes armas.
- Añadir pruebas unitarias al servidor
- Un chat en la salas diferentes salas

## Web grafía

Wikipedia: Modelo-Vista Controlador

https://ca.wikipedia.org/wiki/Model-Vista-Controlador

Wikipedia: Servidor UDP

https://ca.wikipedia.org/wiki/User_Datagram_Protocol

Creative Commons (Licencias Copyright)

https://creativecommons.org/licenses/by-nc/4.0/

StackOverFlow (Debido a la cantidad de veces que se ha tenido que utilizar, resulta inviable poner todo los enlaces)

https://stackoverflow.com/

Nuestro Diagrama de Trello

https://trello.com/b/5lazn3aw/back-log

Repositorio GitHub

https://github.com/Alberto-Gutierro/Catalin_Alberto-Juego_Naves