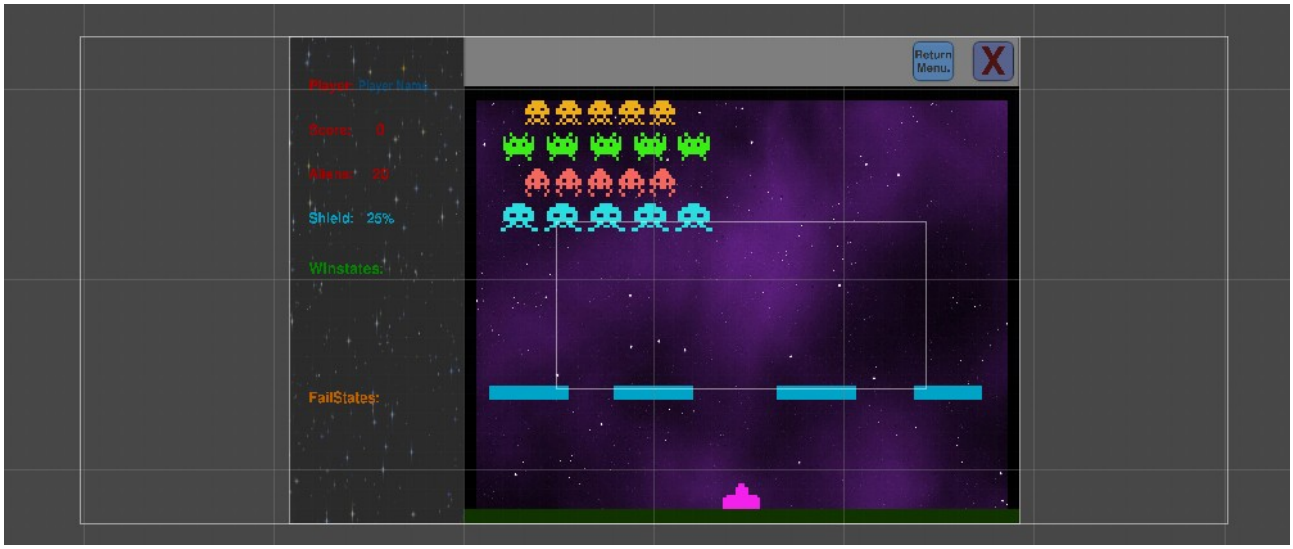
SPACE INVADERS

Un Videojuego Arcade desarrollado con Unity 3



Héctor Gómez Ariño

Ciclo Formativo: DAM

CPIF Bajo Aragón

2020

Sumario

Presentación y propuesta

Propósito

El objetivo del presente documento es plantear el juego a realizar como proyecto de final de ciclo formativo Desarrollo de Aplicaciones Multiplataforma (DAM). En esta presentación o propuesta listaremos sin entrar en detalle el software a utilizar, los componentes generales, los requisitos y la entrada/salida de la aplicación.

En esta sección no haremos un análisis funcional o especificaciones técnicas. Para este detalle referenciamos más adelante en este mismo documento en secciones donde se profundiza en estos aspectos. Simplemente presentamos el concepto previo de la aplicación.

Descripción

Realizaremos un juego con apariencia 2D, basado en una plataforma 3D (Unity3D). Clásico Arcade de máquinas recreativas, ambientado en el espacio, cuyo objetivo es defenderse de las naves de *marcianitos* que nos invaden disparándo con nuestra nave para destruirlas.

Para el desarrollo del videojuego utilizaremos Unity3D una plataforma que permite el desarrollo de videojuegos 3D multiplataforma utilizando librerías y motores gráficos estables y con mucho potencial. Además permite gráficos provistos por la comunidad e importar otros a su sistema de assets.

Género y Trama

El juego se enmarca en los juegos de arcade.

La historia es simple: una invasión alienígena de la que hay que defenderse disparando con nuestra nave o defensas antiaéreas, sin que el fuego devuelto por estas impacte en la nave del jugador.

Mecánicas del Juego

Al arrancar el jugador tendrá la posibilidad de elegir la pantalla y la dificultad, informar su nombre u otras opciones.

El juego podría plantear diferentes dificultades para que el jugador seleccione.

Después de la elección de pantalla el jugador entra a la pantalla del juego e intentará afrontar el reto que le plantea.

Además de la victoria relativa, se mostrará una puntuación al jugador. Con lo que el jugador podrá enfrentar diferentes Winstates o FailStates.

Un WinState es una situación de victoria al vencer un reto pequeño o afrontar con éxito una parte difícil y superarla. En el juego encontraremos varios de estos cuando el jugador haga las cosas bien. Le darán puntos para animarle a seguir.

Un FailState es una situación de derrota, sin que suponga una derrota completa. El jugador podrá "perder" aunque no suponga volver a empezar, por ejemplo, perdiendo los escudos. Supondrá perder puntos en el reto planteado aunque no necesariamente volver a empezar.

La derrota se produce cuando el jugador es destruído por los proyectiles de los alienígenas o cuando estos llegan a la tierra.

El Mundo presentado al jugador será una pantalla con un fondo de estrellas o un sistema planetario que se podría modificar.

Realización

El proyecto está pensado para realizarse en unos 2 meses, tras el análisis previo. Intentaremos usar al máximo librerías libres y Assets gratuitos para facilitar la tarea de diseño del mundo y del entorno.

Tras estos 2 meses de trabajo, la versión beta debería estar completada y se podrá proceder a probar el juego para depurarlo o añadir nuevas funcionalidades. Establecemos esta fase de depuración en un mes como máximo.

Planteamos una primera entrega el 1/6/2020, tras el primer mes de desarrollo, sería una versión devel para acotar objetivos y revisar la programación. En esta entrega se planteará (sin que estén definidos todavía) los análisis funcionales y técnicos para el desarrollo y con posterioridad se escribirán para añadirlos a la memoria. Se detectará si hay requisitos funcionales o técnicos que deberemos abordar para seguir más adelante con el desarrollo y enfocarnos en las debilidades o fortalezas de la aplicación para pulir el resultado final.

Posteriormente se planifica una entrega previa a la presentación del proyecto sobre el 15/6/2020 para ver si se han alcanzado los objetivos y acabar de definir necesidades y detallar el presente documento.

Justificación

¿Porqué del presente?

Marco de desarrollo del proyecto

El proyecto a presentar se enmarca en la formación realizada durante el ciclo formativo Desarrollo de Aplicaciones Multiplataforma.

Los módulos profesionales de este ciclo formativo son los siguientes:

- Sistemas informáticos.
- Bases de Datos.
- Programación.
- Lenguajes de marcas y sistemas de gestión de información.
- Entornos de desarrollo.
- Acceso a datos.
- Desarrollo de interfaces.
- Programación multimedia y dispositivos móviles.
- Programación de servicios y procesos.
- Sistemas de gestión empresarial.
- Proyecto de desarrollo de aplicaciones multiplataforma. Otrora, lo que ahora mismo estamos haciendo todos aquí "reunidos".
- Formación y orientación laboral.
- Empresa e iniciativa emprendedora.
- Formación en centros de trabajo.

Durante el mismo hemos podido aprender:

- A desarrollar aplicaciones multiplataforma, con gestión de Bases de Datos, con conexiones a servidores remotos y con gestión de Assets 3D. Evidentemente bajo criterios de usabilidad, jugabilidad, seguridad y calidad.
- Diferentes lenguajes de programación sobre diferentes entornos de desarrollo (IDE). Gestión de entornos de desarrollo (IDE) y configuración de los mismos adaptada a las particularidades de cada proyecto y aplicación.
- Trabajar con procesos informáticos y gestionar los mismos en tiempo de ejecución.
- Crear y modificar sistemas de gestión de empresarial y manejo avanzado de datos con criterios enfocados a permisos de usuario y acceso (CRUD) en diferentes entornos.
- Interpretar las instrucciones implicadas en las funcionalidades requeridas para poder analizar, programar, probar y documentar las aplicaciones desarrolladas. Proceso resumido como: Análisis, Diseño, Codificación y Pruebas.
- Estas instrucciones han sido analizadas por necesidades de los usuarios y diseños pautados con patrones responsive y de accesibilidad.
- El inglés técnico tan necesario en el mundo de las TIC, en general ha sido reforzado con dos asignatuas. Nos han permitido leer y entender documentaciones o tutoriales en el idioma original para las que muchas veces no hay documentación disponible.

- Configurar y trabajar con sistemas informáticos diferentes, siendo capaces de adaptar la lógica y funcionamiento de los mismos según unos criterios o necesidades finales de uso de las aplicaciones o usuarios.
- Técnicas y procesos para securizar sistemas, servicios y aplicaciones siguiendo planes de seguridad.
- Gestión de Bases de Datos, interpretación de lógicas de diseño, integridad, consistencia, seguridad y acceso a datos.
- Lenguajes, librerías y herramientas que nos permiten cumplir en cada caso unas especificaciones previas.
- Integración de contenidos gráficos y componentes multimedia en las aplicaciones desarrolladas.
- Desarrollo de interfaces gráficos interactivos con criterios de usabilidad, componentes visuales para uso estándar o implementando componentes visuales específicos.
- Desarrollar aplicaciones para teléfonos, PDA y otros dispositivos móviles empleando técnicas y entornos de desarrollo específicos.
- Desarrollar aplicaciones multiproceso y multihilo empleando librerías y técnicas de programación específicas.
- Participar en la implantación de sistemas ERP-CRM evaluando la utilidad de cada uno de sus módulos. Gestionar la información almacenada en sistemas ERP-CRM garantizando su integridad.

Ámbito del proyecto

Gracias a la innovación y creatividad básicas durante toda la Formación Profesional hemos podido implementar proyectos innovadores que nos han permitido adaptarnos a las cambiantes demandas del mercado. El proceso de aprendizaje basado en proyectos nos ha aportado la posibilidad de aprender de manera muy autodidacta conocimientos técnicos y cómo realizar por nuestra cuenta investigación sobre herramientas antes desconocidas y casi sin soporte por parte del profesorado, con norma general. Esto ha supuesto que nos tengamos que enfrentar a retos antes desconocidos y buscar información a solas en foros, manuales y otros para poder solucionar los problemas planteados.

El presente *Proyecto de desarrollo de aplicaciones *multiplataforma* estaba planteado por el expediente académico para llevarse a cabo en **99 horas,** cuando todos sabemos que se han dedicado muchas más y es muy difícil cumplir con esa expectativa irreal. Estimamos que la realización serán aproximadamente unas 250 horas incluyendo documentación. Si pudieramos realizar el videojuego y la documentación trabajando en paralelo con otro compañero, probablemente nos acercaríamos a esas teóricas 99 horas.

Con todo lo expresado anteriormente podemos ser capaces de desarrollar una aplicación con una vertiente técnica potente y que permita integraciones y gestión avanzada. El motor gráfico es razonablemente amigable, aunque el proceso de instalación en algunos entornos se prevee complejo.

Para cumplir con unos mínimos requisitos hemos seleccionado el desarrollo de una aplicación con base 3D que permita la compilación en ejecutables compatibles con diferentes plataformas. Hemos escogido Unity3D que permite satisfacer todos estos requisitos previos. A continuación, en la sección de Objetivos analizaremos los requisitos técnicos y funcionales del proyecto.

En cuanto a nuestro IDE destacamos el siguiente extracto de **wikipedia** sobre el enfoque:

> *El éxito de Unity ha llegado en parte debido al enfoque en las necesidades de los desarrolladores independientes que no pueden crear ni su propio motor del juego ni las herramientas necesarias o adquirir licencias para utilizar plenamente las opciones que aparecen disponibles. El enfoque de la compañía es "democratizar el desarrollo de juegos", y hacer el desarrollo de contenidos interactivos en 2D y 3D lo más accesible posible a tantas personas en todo el mundo como sea posible.*

Este enfoque democratizante nos ha sido de gran ayuda, al permitirnos utilizar assets y librerías abiertas o libres con comodidad y con un motor potente detrás, haciendo que la compilación en diferentes plataformas sea mucho más cómoda.

Objetivos

¿Qué y Cómo? del presente.

Análisis de Requisitos

El presente proyecto pretende exponer de manera condensada el aprendizaje adquirido durante estas más de 1500 horas. Por lo tanto tenemos que desarrollar una aplicación multiplataforma y programación MVC orientada a objetos con gestión de clases, Sprites y Assets; en la parte de diseño de videojuego podemos destacar los colisionadores y las pantallas (escenas).

Requisitos funcionales

No se han establecido unos requisitos funcionales claros antes de iniciar el desarrollo, así que establecemos los siguientes como línea de actuación:

- Juego funcional o funcionalidad primaria: Juego de plataformas en 2D con sistema de disparo y movimiento de objetos en la pantalla de juego. Los objetos (marcianitos, disparos, escudos y nave del jugador) han de interaccionar entre ellos y responder a colisiones (disparos). Pérdida y victoria de la partida.
- Cálculo de puntuación durante la partida.
- Guardar el nombre del jugador.
- Guardar otros campos o valores durante la ejecución de la partida y trabajar con ellos.
- Movimiento de la acción del juego entre pantallas (escenas) y con la aplicación (cerrar, guardar campos).
- Aplicación multi-plataforma-> Exportación de la misma con ejecutables no nativos.

Requisitos técnicos

Tampoco tuvimos claro un listado específico de requisitos técnicos que debería cumplir la aplicación desarrollada. Los hicimos sin demasiado soporte y los establecimos nosotros mismos cuando planificábamos el inicio del proyecto; así que listamos algunos:

- Aplicación ejecutable en multiplataforma (Android APK, GNU/Linux y windows .exe).
- Aplicación con programación en lenguaje orientado a objetos (C#) con uso de clases.
- Programación que gestione colisiones entre objetos, trabaje con ellos (objetos del juego: CRUD creación, eliminación/destrucción, update, cálculo, creación...).
- Utilizar un IDE que incluya tanto emulador como un motor gráfico.
- Motor gráfico que permita utilizar assets libres y sprites abiertos no diseñados ex profeso para esta aplicación, lo que agilizará el desarrollo. Soporte texturas 3D. Soporte de material gráfico de fuentes abiertas.
- Motor gráfico orientado a objetos.

- Navegación: obstáculos dinámicos diseño de rutas.
- Motor gráfico que permita ejecución en pantallas de diferente tamaño/resolución adaptando la ventana de juego.
- Paso de variables en tiempo de ejecución entre las pantallas (por lo menos el nombre del jugador).
- Cálculos durante la partida y print de estos en el output del juego.
- Repositorio de código para generar versiones de la aplicación y poder movernos por el histórico del desarrollo.
- Sistema de ticketing para poder tracear problemas, errores o mejoras y llevar un listado de los que se han desarrollado o los que no.

Metodología de Desarrollo

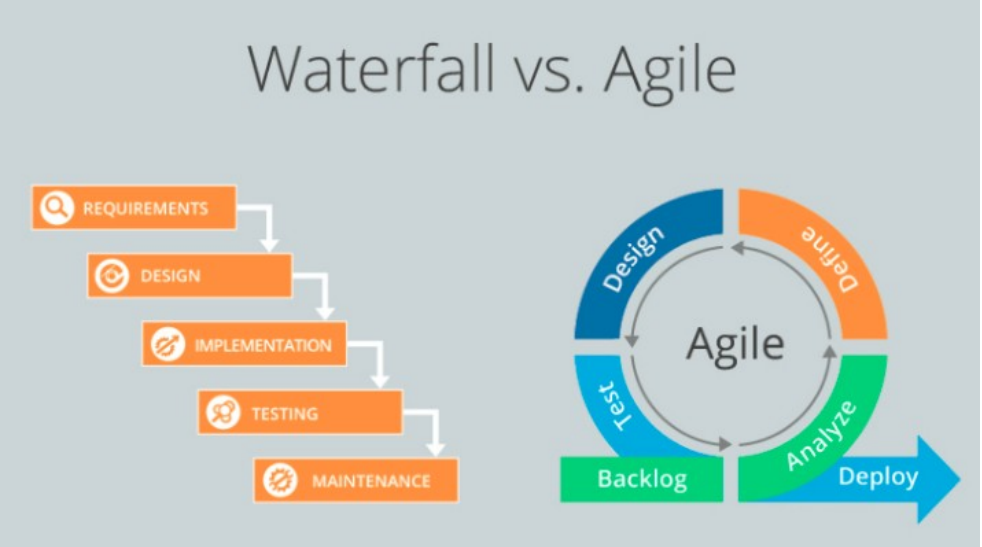
Descripción de la metodologías utilizadas para la implementación técnica

Técnicas utilizadas

Como explicamos más abajo en la sección de planificación, establecemos un modelo de ciclo de vida de software en espiral. Dividimos el trabajo en iteraciones o sprints y cada uno de estos a su vez se divide en:

- Análisis
- Diseño
- Codificación
- Pruebas

Agile



Una metodología de desarrollo ágil lleva desde la toma de decisiones hasta el desarrollo mismo de software y afecta a todas las áreas de trabajo. Se puede incorportar a métodos de desarrollo de ingeniería o a patrones de diseño hasta el mismo testing de las aplicaciones que realicemos.

El enfoque basado en desarrollo iterativo e incremental permite que los requisitos no estén del todo claro al inicio del proceso y se vayan definiendo o incrementando a medida que este progresa adaptándose a las necesidades del proyecto. Los equipos de trabajo se auto-organizan y se pierden los roles marcados de otros procesos de desarrollo, lo que facilita que se agilice mucho la toma de decisiones a corto plazo. El trabajo es autoorganizado en equipos multidisciplinares, como en una pura Anarquía autogestionada.

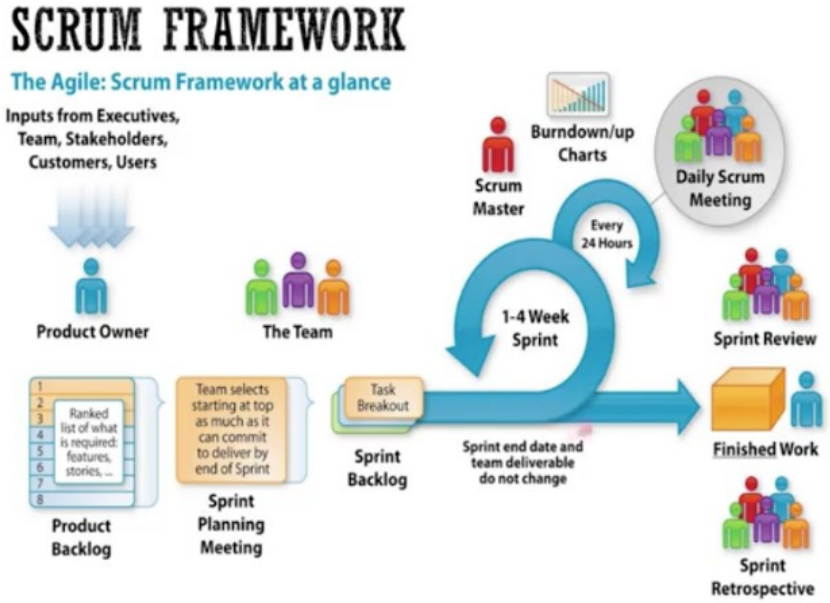
Se enfoca la comunicación cara a cara y se devalua la documentación de los procesos. Una metodología ágil enfatiza que se hagan entregas funcionales con regularidad, pudiendo aportar valor añadido al software con cada iteración. Estas entregas son la primera medida del progreso del software.

Las grandes críticas que se hacen a la metodología ágil es la "indisciplina" y la falta de documentación técnica.

Cada iteración del ciclo de vida incluye planificación, análisis de requisitos, diseño, codificación, pruebas y documentación. Adquiere una gran importancia el concepto de "finalizado" (done), ya que el objetivo de cada iteración no es agregar toda la funcionalidad para justificar el lanzamiento del producto al mercado, sino incrementar el valor por medio de "software que funciona" (sin errores).

Scrum

SCRUM es un proceso de desarrollo de software ágil que incluye una serie de normas o buenas prácticas, definiendo unos roles concretos para realizar las comunicaciones, colaborar en el equipo y finalmente obtener el mejor resultado posible de los proyectos.



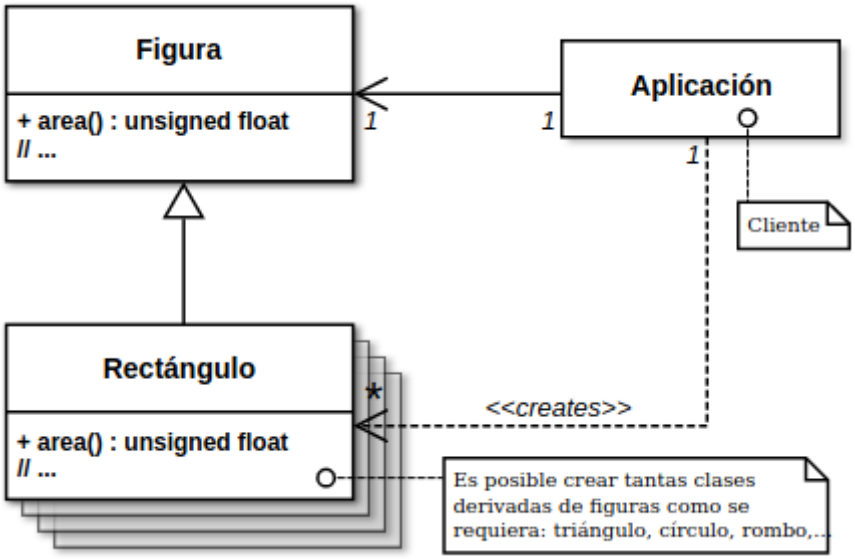
Algunas de sus caracterísitcas principales son:

- Estrategia de desarrollo incremental, en vez de planificación, y ejecución completas de un producto.
- Basar la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que en la calidad de los procesos empleados.
- Solapar las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o en cascada.

Programación orientada a objetos

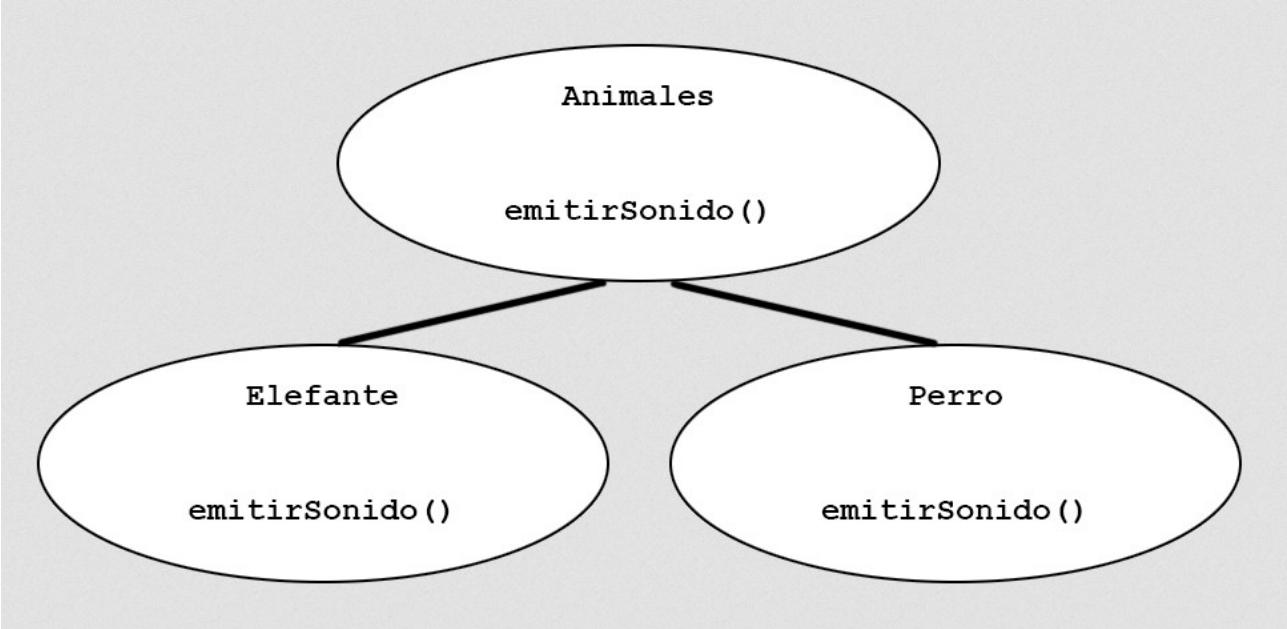
Se conoce como programación orientada a objetos a un paradigma de programación que cambia la manera de obtener los resultados del flujo de datos. Los objetos manipulan los datos de entrada para sacar datos de salida concretos. Cada objeto ha de ser diseñado para una función específica.

Los objetos prediseñados en lenguajes de programación permiten creación de librerías o bibliotecas, también están orientados para que los usuarios creen sus propias bibliotecas.



Las técnicas habituales para trabajar con objetos son:

- Herencia.
- Cohesión.
- Abstracción.
- Acoplamiento.
- Encapsulamiento.
- Polimorfismo, ejemplo en este diagrama:



Tecnologías

Explicación técnica y breve manual de algunas tecnologías aplicadas en el desarrollo. Cómo nos han ayudado y porqué han sido útiles y prácticas.

Plataforma utilizada: *Unity 3D.*

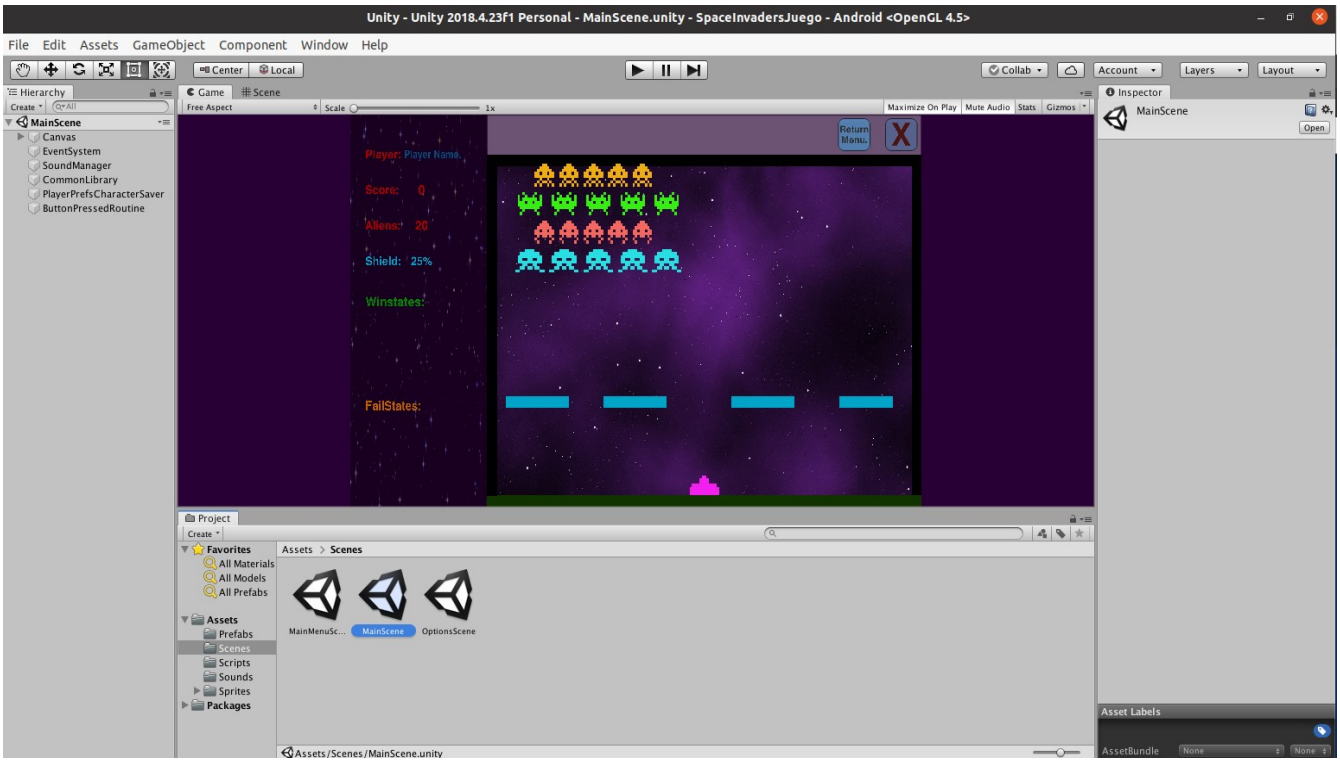
Si bien es cierto que antes de empezar el desarrollo estuvimos dudando con otras plataformas como por ejemplo Monkey SDK, la gran variedad de assets libres en la biblioteca de Unity3D y la cantidad de tutoriales que encontramos en youtube nos hizo decantarnos claramente por Unity3D, a contiuación explicaremos algunas de sus características principales.

¿Qué es Unity?

Unity es una plataforma de creación de videojuegos desarrollada por Unity Tecnologies que proporciona un editor, motor gráfico, un IDE de desarrollo y otros servicios. Entre otras, el motor gráfico cosas engloba:

- Física 2D
- Física 3D
- Audio
- Animación
- Render

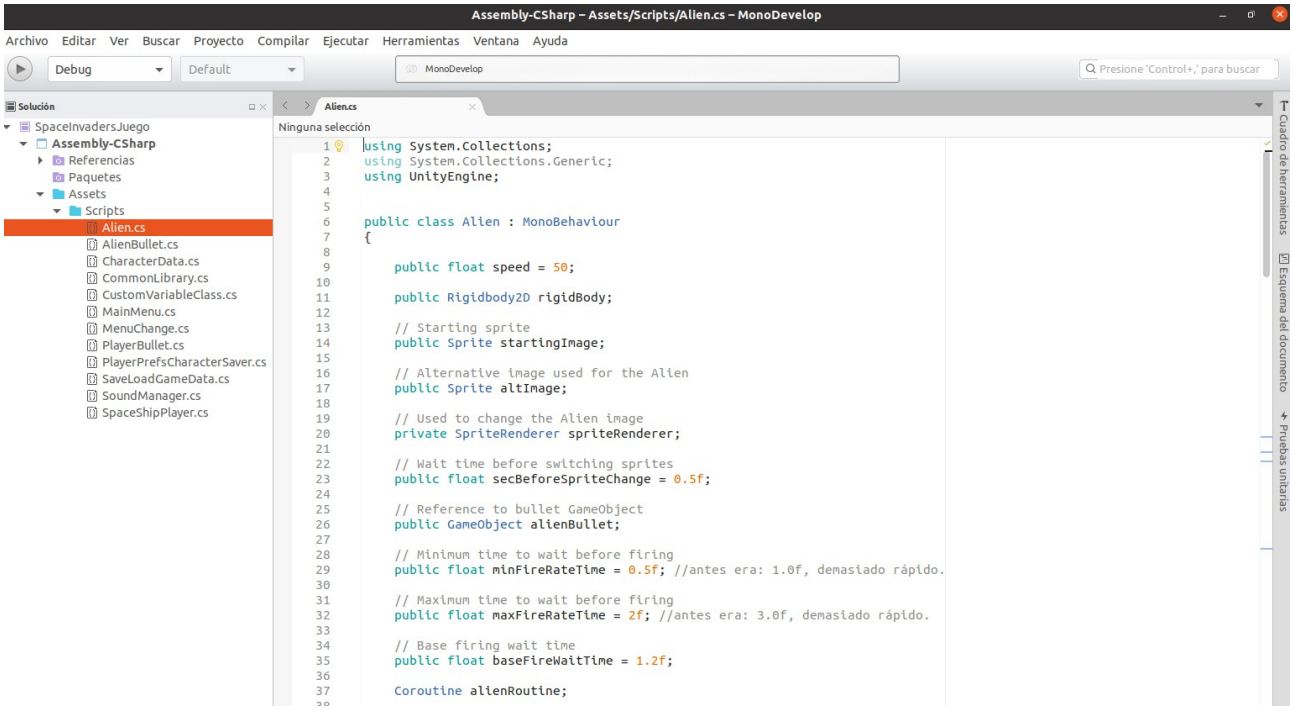
- Herramienta de creación videojuegos
  - https://unity.com/madewith
- Desarrollado por Unity Technologies



¿Por qué Unity?

- Exportar a multitud de plataformas, más de 30 plataformas disponibles para crear nuestro videojuego.
- Servicios: Ads (publicidad), IAP(monetización), Analíticas, Cloud Build (construcción en la nube), Collab (colaboración en equipo)
- Documentación de primera:
  - Manual de usuario. Se actualiza con cada versión de Unity y podemos consultar de manera ordenada las specs de cada versión, errores que podemos encontrar de una versión a otra y todos los módulos de los que se compone (Fisica, 2D, 3D, scripting, multiplayer, audio...). Es una documentación muy completa y que permite reportar errores encontrados.
  - API: También muy actualizada, con referencias de las versiones anteriores de Unity3D, no solo de la última y ofrece la interfaz de programación completa no solo para el motor gráfico sino también para el propio editor. El propio editor de Unity está desarrollado utilizando las herramientas gráficas de Unity.
- Comunidad: Cuenta con una comunidad muy grande y colaboradora. Permite acceder a foros de discusión ordenados por tecnologías, prefabs...
  - Foros
- Programación en C#. Orientado a objetos desarrollado y estandarizado por microsoft como parte de su plataforma .NET. Posteriormente fue aprobado como estándar por la FMA e ISO. Cuenta con su propio IDE estándar utilizado por Unity que es Mono.
  - Aunque nosotros hemos utilizado durante bastante tiempo Visual Code por un tema de compatibilidades con nuestras plataformas, finalmente el propio editor Unity nos ha

"obligado" a pasarnos a MonoDevelop par poder trabar con Unity.



Características principales:

- Unity Hub: Herramienta de gestión de proyectos Unity así como diferentes versiones de Unity para los diferentes proyectos.
- Interfaz de usuario: Paneles, vistas y herramientas de modificación de objetos.
- Asset Store: Tienda de recursos gratuítos o de pago para utilizar, comprar o descargar recursos para la creación de videojuegos. Los recursos provistos por la empresa o la comunidad, de variopinta calidad son excelentes para probar las diferentes características de Unity.
- Prefabs.
- Objetos predefinidos: Game Object, Player, Component, Tags, Capas, Escenas...
- Build Settings: Opciones de contrucción (Build), Pantalla de Splash, ejecutable final...
- Cache Server: Herramienta que ahorra tiempo al trabajar con diferentes plataformas al desarrollar el videojuego.

Curiosidades:

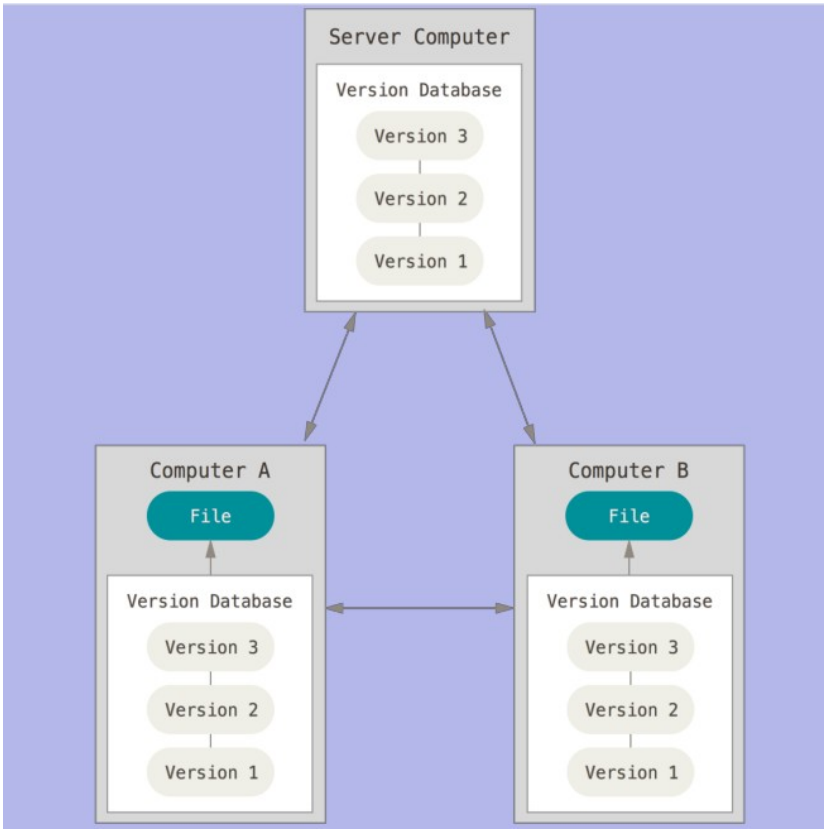
Origen y evolución: En 2004, una empresa lanzó un juego para Mac llamado GooBle que tuvo un éxito comercial pobre, pero rápidamente se vio el potencial que había en las herramientas que habían desarrollado para el videojuego. Ese año fundaron UnityTecnologies y siguieron desarrollando herramientas que permitiera a cualquier persona desarrollar videojuegos a bajo coste. En esa época las herramientas que había eran de licencia con alto precio.

Versiones:

- 2004: GooBall
- 2005 - Unity: Motor exclusivo para OSX
- 2008 - Unity 2.0: Soporte para iOS
- 2010 - Unity 3.0: Soporte para Android
- 2012 - Unity 4.0: DirectX, UGUI (2014 – 4.6)
- 2015 - Unity 5.0: WebGL, iluminación
- 2017 - Unity 2017: Timeline, Cinemachine
- 2018 - Unity 2018: Scriptable Render Pipeline
- 2019 - Unity 2019: Muchos cambios

GIT y GitHub

Hemos decidido utilizar un gestor de versiones para apoyar nuestro desarrollo para permitir mayor escalabilidad, moluralidad y robustez durante el desarrollo. Como repositorio remoto, hemos optado por GitHub, que de manera cómoda nos permite acceder a toda la suite de GIT desde un cómodo entorno gráfico web.

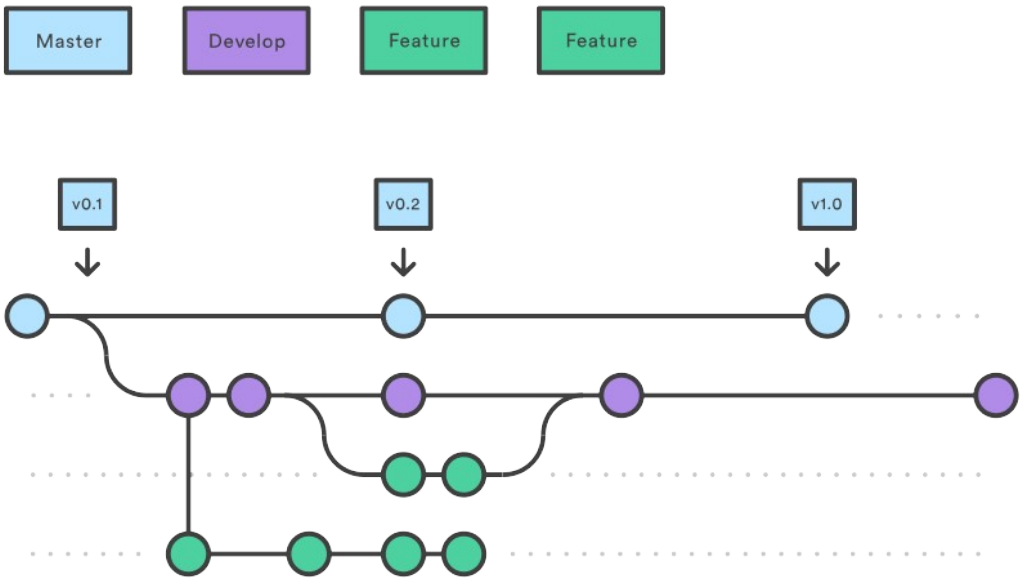


Gestión de versiones: GIT Introducción.

GIT es una potente herramienta de control de versiones software libre, que nos apoyará durante el ciclo de vida del software. Nos permitirá hacer desarrollos incrementales y adaptarnos a las necesidades del cliente de manera modular. Optimiza el desarrollo en paralelo que evolucionan en el tiempo en bloques estables.

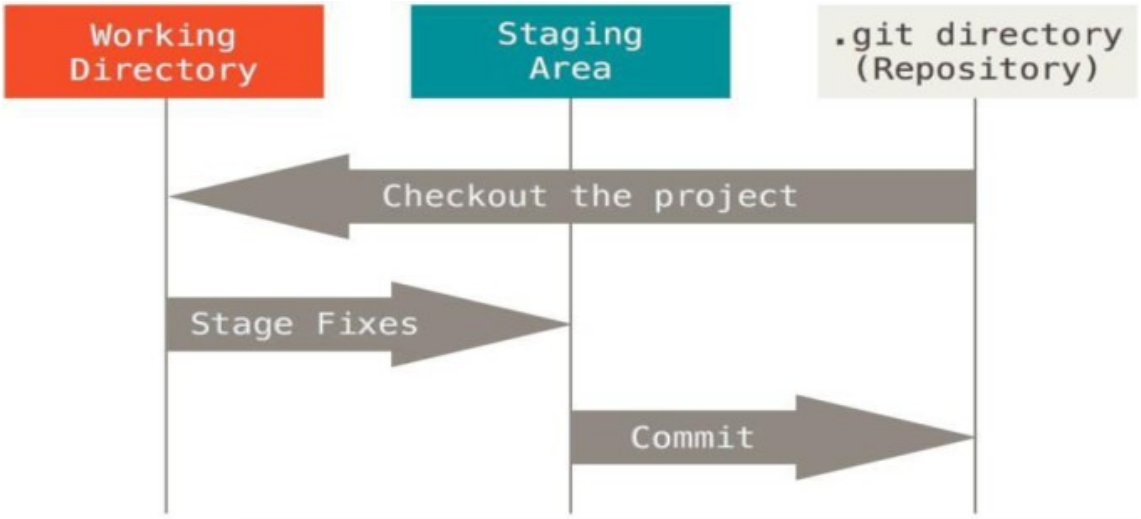
Utiliza un repositorio central de código donde se generan copias de trabajo, permite trabajar sin acceso a la red y solo requiere acceso al servidor en momentos concretos. Se puede trabajar en local con las copias locales del repositorio y conectarnos solo cuando necesitamos actualizar el repositorio remoto. Esto se llama sistema de control de versiones distribuido (DVCS).

Permite tener un historial de revisiones completo y trabajar con Ramas (*branches*) diferentes fusionando (*merge*) ágilmente el código. En cualquier momento puedes acceder al estado de cualquier fichero en cualquier momento del tiempo, no solamente a revisiones puntuales.



Detalle de estados de un fichero en GIT:

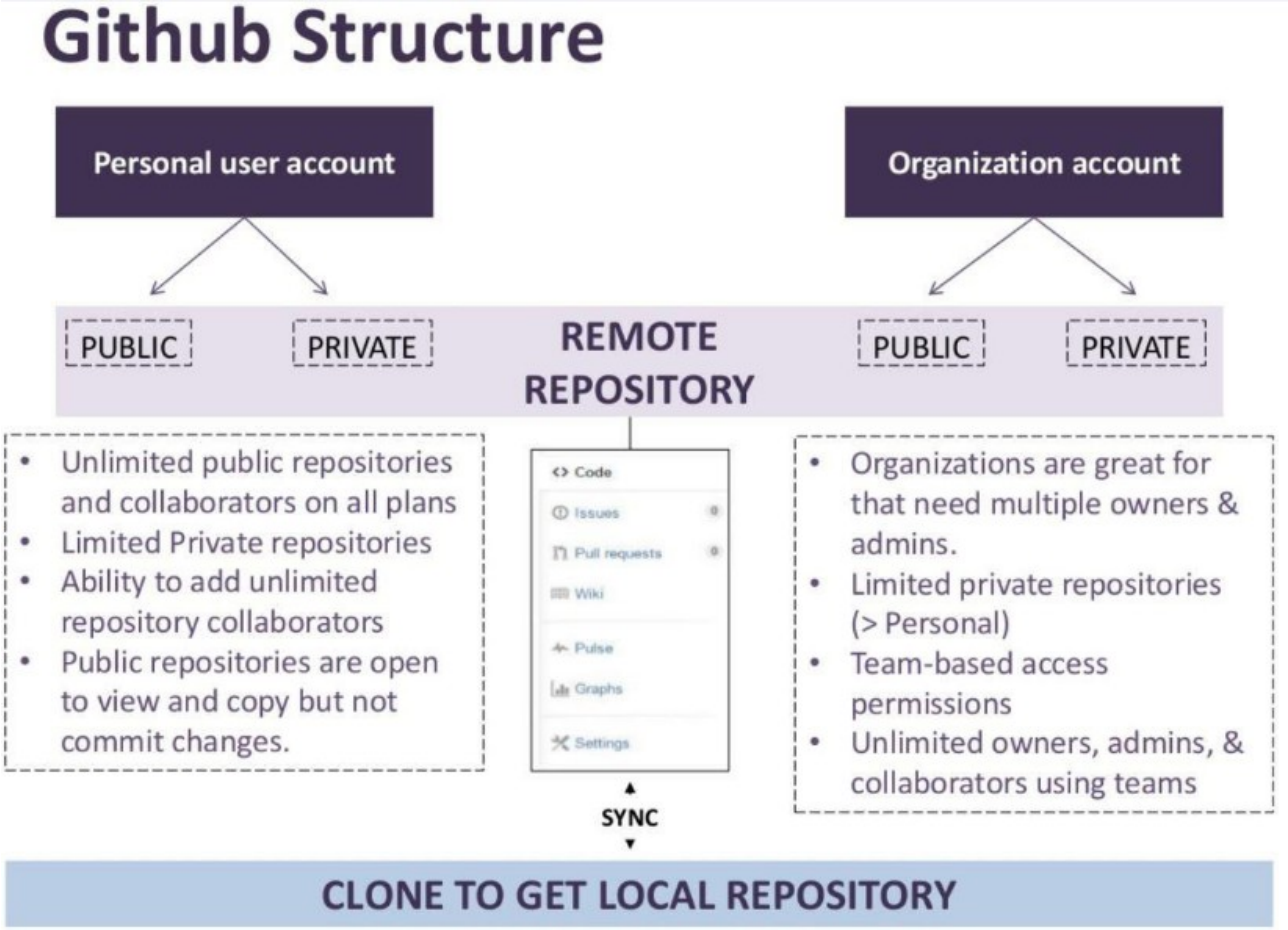
- Confirmado significa que los datos del archivo están almacenados de manera segura en tu base de datos local (*commited*).
- Preparado significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación (*staged*).
- Modificado significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos (*untracked*).



GitHub

Github y Gitlab son un servicios de hosting para GIT:

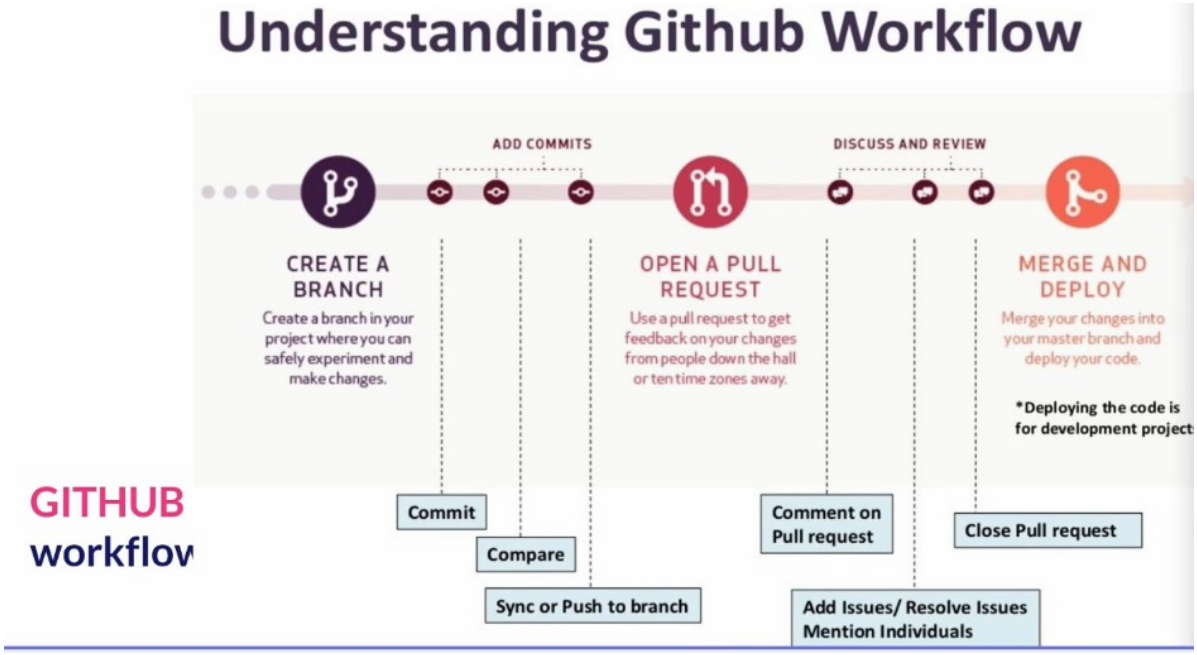
- GitLab además, es software para descargar en local.
- Proporcionan un interfaz web encima de GIT.
- Funcionan como una plataforma social para compartir conocimiento y trabajo
- Proveen de características de control de acceso y herramientas como wikis y gestión de tareas



Conceptos

- Creando un repo/ *Master* en un repositorio: versión final.
- Crear una rama/*branch*:
  - Los cambios de una rama no afectan al *master* a menos que se acepte una petición pull.
  - Los cambios aprobados (*commit*) a una rama, quedan reflejados para seguimiento de versiones.
- Bifurcando (fork) un repositorio:
  - Crea una copia para que trabajes de forma independiente sin cambios para otros.
  - Envía una *pull request* al propietario para que incorpore cambios.

- Pull requests:
  - Inician una discusión sobre tus cambios.
  - Visualizan qué cambios serían fusionados (merged) si se acepta la pull request.
  - Usa el sistema @mention en tu petición de pull para enviar un mensaje que pida feedback a equipos o personas específicas que opinen sobre tu trabajo.
  - Issues (problemas).
  - Resalta bugs o problemas de código que necesitan rectificación.
  - @mention puede usarse para notificar.
- Sintaxis markdown
  - Disponible en descripciones y comentarios de Issues y peticiones de pull. Incluyen @mentions y referencias a issues, pulls, etc.
  - Visionados (watch) y estrellas (star).
  - watch nos notifica de todas las conversaciones sobre tus @mentions, commits, comentarios a discusión...
  - Star demuestra un favorito de una mención, comentario, etc.



GIT, Comandos Básicos.

Aunque diferentes herramientas permiten trabajar en modo gráfico, ejecutaremos GIT utilizando comandos, lo que nos permitirá trabajar en un entorno más parecido a nuestros servidores de trabajo.

Inicializando repositorio local:

- *git init:* Crea un repositorio de trabajo en el directorio actual, permite empezar a trabajar en local con GIT aunque no estemos conectados a repositorio. Crea directorio oculto .git con las configuraciones del repositorio local.

Trabajando con ficheros:

- *git status*: Lista los cambios del repositorio de GIT local. Permite ver el estado de los ficheros y nos muestra si debemos cambiarlos de área.
- *git add*: Añade ficheros al *staging area*, marca como preparados.
- *git commit*: Añade una revisión de los ficheros, marca como aprobados. Es recomendable informar un mensaje con el detalle de los cambios.
- *git rm*: Elimina fichero de repositorio en el *staging area*. Necesario cada vez que se elimina un fichero en *working directory* si se desea eliminar también de repositorio.

Revisando cambios:

- *git log*: Permite ver todas las revisiones del repositorio local, sean pendientes de aprobar o aprobadas. Con --*oneline* muestra un resumen. Ejemplo práctico: *git log --pretty="%h %an %ar - %s"*
- *git diff*: Permite revisar y comparar *working directory* con *staging área*. De nuestro área de trabajo al área de preparado. Con --*staged* compara con area de preparación (*staging area*).

Trabajando con repositorio remoto:

- *git clone*: Clona un repositorio remoto a un nuevo directorio. *git clone <url remota>* Con '.' como último parámetro descarga el repositorio remoto en el path actual. Ejecuta conjuntamente: git init + git remote + git fetch + git pull
- *git remote*: Permite añadir un repositorio remoto. Con *add origin* asignamos a la etiqueta origin el nombre para el nuevo repositorio.
- *git fetch*: Descarga objetos y referencias del repositorio remoto. No descarga ficheros a *working directory*.
- *git pull*: Descarga (tira) desde el repositorio remoto a local. Normalmente: *git pull origin <nombre_rama>*
- *git push: Sube (empuja) desde repositorio local a la fuente (origin) los cambios confirmados (commited). Normalmente: git push origin <nombre_rama>*

Deshaciendo cambios:

- *git checkout*: Cambia de rama o restaura los ficheros locales al *working directory*. Con el nombre de una rama como parámetro cambia el *working directory* a esa rama. Con -- <nombre_fichero> o -- . restaura los ficheros locales al estado del área de commited. Con HEAD o un identificador utiliza la marca temporal especificada. Ejemplo: *git checkout HEAD -- .* actualiza del *working directory* el estado de todos los ficheros del *commit area*.
- *git reset*: Permite recuperar el estado de una marca temporal del *staging area*. Ejemplo: *git reset --hard HEAD~1* comando agresivo que recupera el commit anterior confirmado a nuestro *working directory* y lo elimina del *staging area*.

- *git revert*: Deshace cambios locales. Permite deshacer cambios del *staging area* de manera controlada e informando de la revisión.

Otros comandos útiles:

- *git mv:* renombra o mueve ficheros.
- *git show*: Muestra un objeto de repositorio con sus cambios.
- *git config*: configura parámetros de repositorio como nombre y e-mail de usuario.
- *git branch:* Muestra las *branch* remotas, permite crear nuevas.

Herramientas de trabajo necesarias:

- Cliente GIT. Desde windows *GIT for Windows*, desde GNU/Linux o Mac Git en terminal.
- Comparador gráfico de ficheros*: Meld, Tkdiff, vimDiff* o similar para revisar cambios en paralelo.
- También se puede instalar un gestor más avanzado como *GitKraken* o un plugin para nuestro IDE favorito (*phpStorm* y *Visual Code Studio* incluyen herramientas de ese tipo).

Otros requisitos:

- Acceso a un servidor con repositorio GIT previamente instalado. Puede tratarse de GitLab, GitHub u otros. En nuestro caso se trata de GitHub en esta ubicación accesible en internet: https://github.com/hectorgomezarino/ProyectoDAM
- Credenciales de usuario del mismo con permisos para los proyectos necesarios.

Modelo de Datos

Diseño de Datos. Modelo de Datos utilizado, cómo guarda los datos la aplicación, si es en DB, si es fichero, en tiempo de ejecución, cómo se estructuran... etc.
Objetos, Clases utilizadas.

Para guardar campos del jugador se utiliza el componente (clase) `CharacterData` provisto por el core de Unity en el que se pueden añadir campos e invocarlos durante la ejecución del juego. Estos se guardan mientras dure la partida.

También ha sido muy útil el componente nativo de Unity: `PlayerPrefs`, que ofrece una suite completa de funcionalidades predefinidas con las que guardar variables en campos en tiempo de ejecución del juego para utilizarlas entre pantallas (escenas).

Estructura de directorios de los assets:

- */Assets/*
  - *Prefabs* (aliens, bullets, shields, explosions)
  - *Scripts*
    - */Models*: contiene los objetos (modelos) utilizados.
  - *Scenes*
  - *Sprites* (aliens)
    - */Background*
  - *Sounds* (movement, explosion, fire...)

Pocesos seguidos para el desarrollo de funcionalidades

A continuación vamos a listar, sin entrar en mucho detalle algunos de los procesos que se han llevado a cabo para el desarrollo de algunas de las funcionalidades específicas de la aplicación:

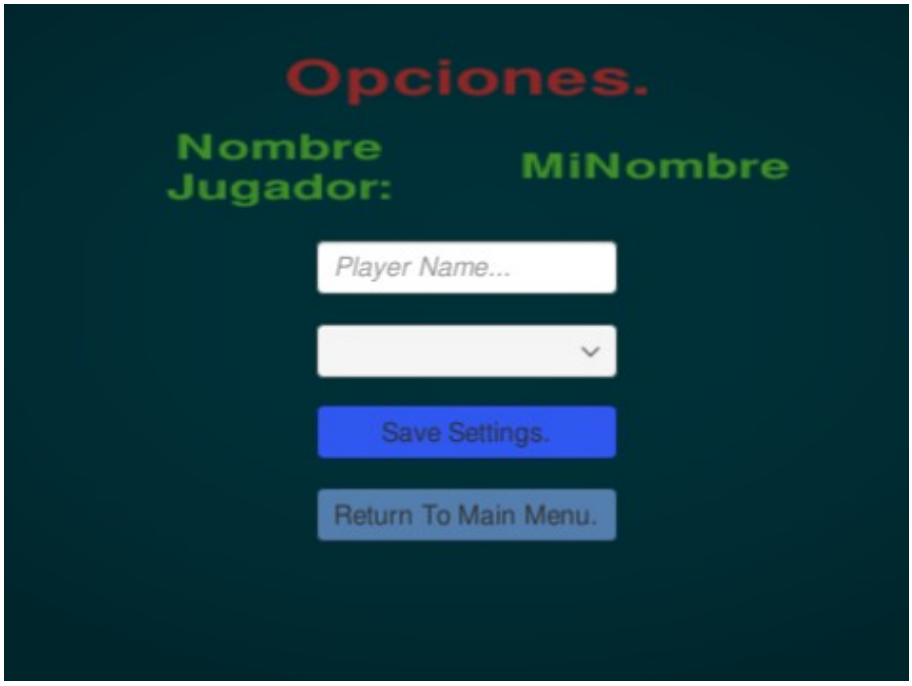
- Crear imágenes con un editor (GIMP) para el juego. Esto nos permite generar nuestros propios assets para utilizar en el juego.
- Colocar las img como sprites para construir los elementos. Utilizando sprites provistos o los anteriormente mencionados, hemos añadido estos a prefabs o componentes del juego.
- Crear un sistema de directorios para los assets diferentes. La estructura se pude ver en la sección anterior.
- Añadir colisionadores en los sprites de las escenas para detectar las colisiones de objetos. Los objetos deben tener la propiedad colider y un manejo de la misma en código.
- Permitir gravedad en los objetos desde los atributos del mismo.
- Controlador de sonido. Hay que añadirlo en las escenas donde se requiere emitir sonidos, además hay que cargar los sonidos en los objetos en los que se requieran.
- Guardar objetos prefabricados. Los prefabs de Unity nos permiten generar un objeto que vamos a utilizar o generar varias veces en nuestro código.
- Instanciar objetos prefabricados. Los objetos prefabricados han de ser instanciados en la escena de Unity para poder invocarlos.
- Destruir Objetos. El método destroyobject permite destruir objetos, también se podrán volver a generar.
- Movimiento entre pantallas. Es básico poder saltar entre pantallas diferentes generando un juego funcional.
- Guardar datos (opciones). Se pueden guardar datos de diferentes maneras en Unity, en nuestro caso solo nos ha hecho falta utilizar el modelo Player, al que se pueden asignar propiedades y características.

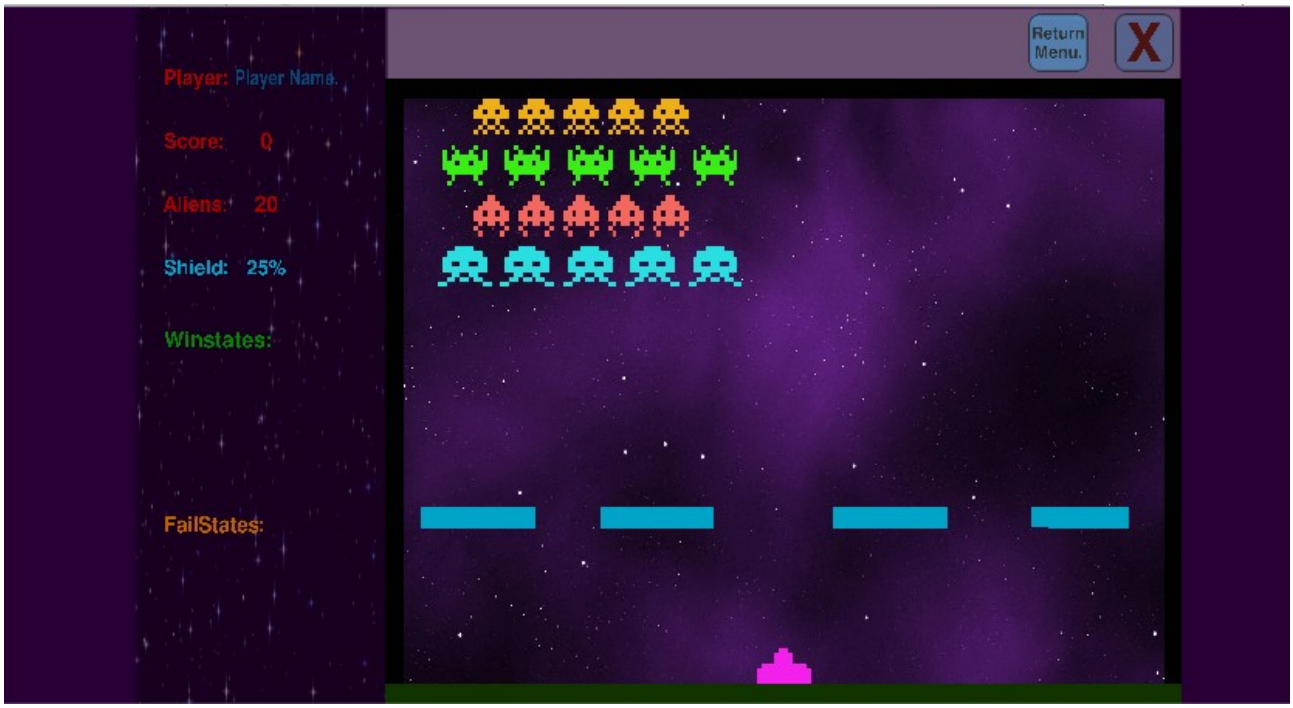
Diseño de la aplicación

Parte visual de la aplicación.

Prototipos (Mook-ups)

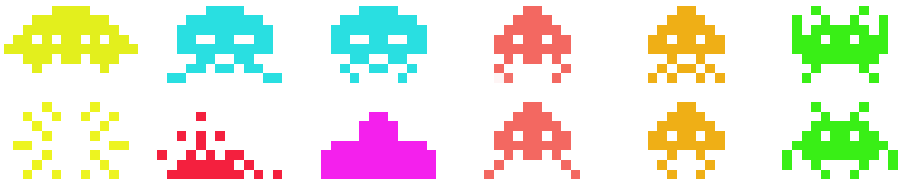
Mook-ups/sprites utilizados para el desarrollo (capturas o paint guarro).
Maquetas para los prototipos de las pantallas.





Otros assets

Principales Sprites utilizados para el videojuego:



Además de estos sprites, hemos utilizado otros gráficos de licencia abierta Copy Left o similar para generar los fondos de pantalla, las transiciones y los splash screen. No los copiamos aquí porque ocuparían demasiado, están en el directorio: */Assets/Sprites/Background* dentro del directorio del videojuego.

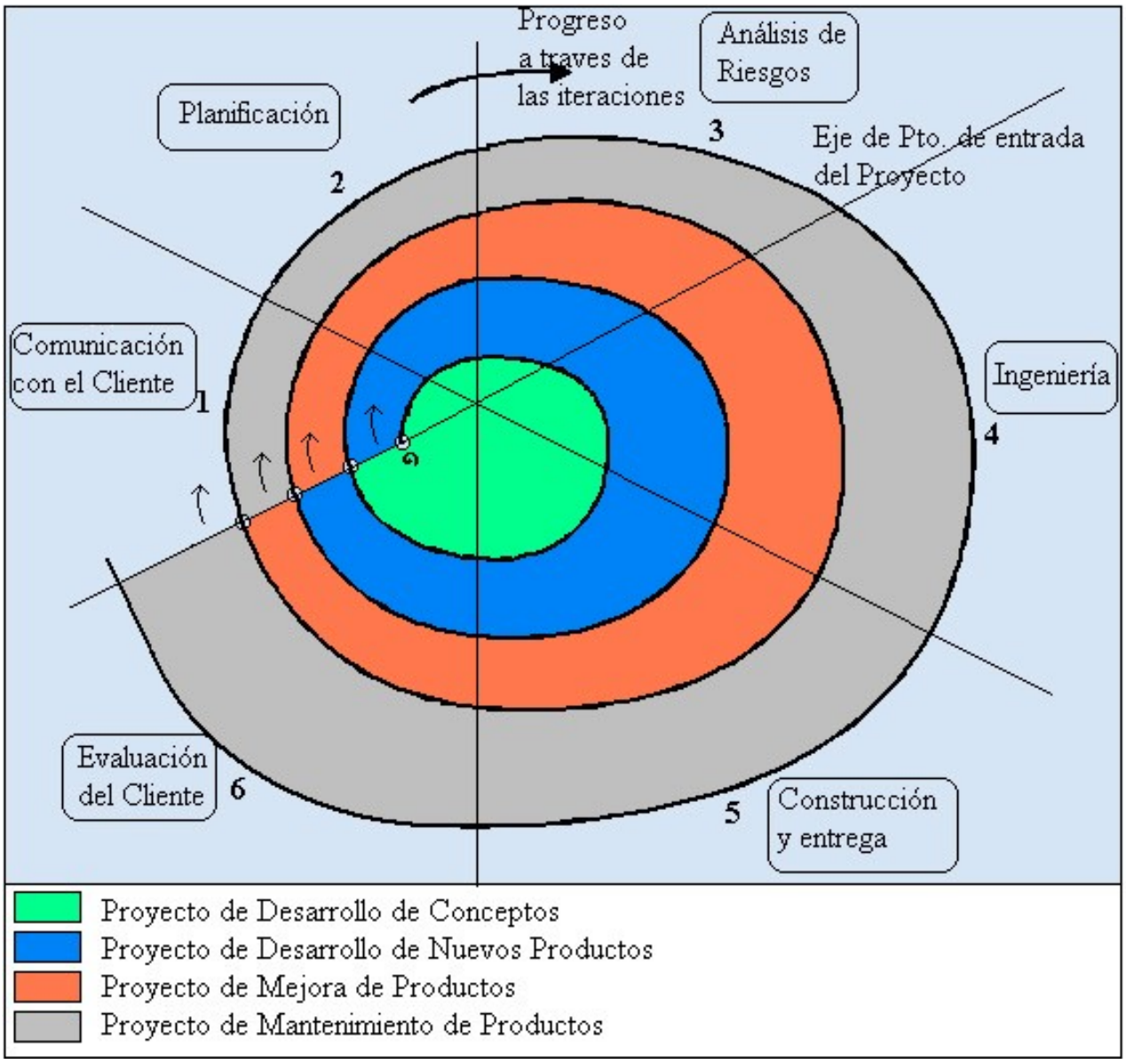
Como es evidente a este manual hemos añadido capturas e imagenes de fuentes libres, han sido referenciadas en la bibliografía, al final de este documento.

Planificación temporal y económica

Planificaremos aquí el tiempo y coste de desarrollo e implementación del presente proyecto.

Planificación temporal inicial

Hemos escogido un ciclo de vida con un modelo en espiral, que es el que mejor nos permite adaptarnos a mejoras de funcionalidades dinámicamente sobre el código que ya tenemos implementado en cada momento. Se preveen 3 iteraciones, cada una más breve que la anterior para poder realizar ligeras mejoras según se acerque la fecha de entrega definitiva. Cada iteración del proyecto nos brinda un prototipo usable mejor que el anterior.



A grandes rasgos cada iteración tiene las siguientes fases:

- Análisis de requisitos.
- Diseño de Software y componentes.
- Programación y codificación de las mejoras.
- Pruebas y validaciones con usuarios.

Cada una de las fases principales tiene un Hito que será una entrega parcial de un ejecutable funcional, adaptándonos a nuestra metodología ágil, podremos dar valor añadido a nuestra aplicación al final de cada iteración, dejando para más adelante otras mejoras pendientes.

A continuación el detalle de las tareas y el tiempo estimado en jornadas de trabajo:

0	**Análisis y Diseño Videojuego**					
0.1	Redacción presentación o propuesta y listado previo requisitos.	Héctor Gómez Ariño.	2/03/2020	2/03/2020	1	0 %
0.2	Análisis Plataformas posibles para el desarrollo.	Héctor Gómez Ariño.	3/02/2020	5/02/2020	3	0 %
0.3	Análisis herramientas, librerías y assets necesarios.	Héctor Gómez Ariño.	5/02/2020	6/02/2020	2	0 %
0.4	Diseño Mook-Ups pantallas y elementos.	Héctor Gómez Ariño.	6/02/2020	6/02/2020	1	0 %
0.5	Redacción requisitos Funcionales	Héctor Gómez Ariño.	9/02/2020	9/02/2020	1	0 %
0.6	Redacción requisitos Técnicos	Héctor Gómez Ariño.	10/02/2020	10/02/2020	1	0 %
1	**Configuración y Desarrollo Aplicación**					
1.1	Configuración repositorio GIT.	Héctor Gómez Ariño	10/02/2020	11/02/2020	2	0 %
1.2	Creación estructura de directorios para el proyecto.	Héctor Gómez Ariño	10/02/2020	13/02/2020	4	0 %
1.3	Configuración entorno Unity3D	Héctor Gómez Ariño	16/02/2020	20/02/2020	5	0 %
1.4	Búsqueda de Assets. Implementación Assets y Sprites	Héctor Gómez Ariño	16/02/2020	20/02/2020	5	0 %
1.5	Investigación Desarrollo requisitos Técnicos.	Héctor Gómez Ariño	16/02/2020	17/03/2020	32	0 %
1.6	Desarrollo Videojuego.	Héctor Gómez Ariño	16/02/2020	15/05/2020	90	0 %
1.7	Configuración librerías compilación multi-plataforma.	Héctor Gómez Ariño	11/05/2020	21/05/2020	11	0 %
1.8	Desarrollo Funcionalidades secundarias.	Héctor Gómez Ariño	21/05/2020	29/05/2020	9	0 %
1.9	**Presentación Versión Beta**	**Héctor Gómez Ariño**	**1/06/2020**	**1/06/2020**	**1**	**0 %**
1.10	Revisión requisitos funcionales	Héctor Gómez Ariño	1/06/2020	3/06/2020	3	0 %
1.11	Revisión requisitos técnicos	Héctor Gómez Ariño	1/06/2020	3/06/2020	3	0 %
2	**Pruebas con Juego y fase de depuración**					
2.1	Depuración aplicación según nuevos requisitos.	Héctor Gómez Ariño	4/06/2020	10/06/2020	7	0 %
2.2	Test previo de la aplicación con los cambios requeridos.	Héctor Gómez Ariño	8/06/2020	12/06/2020	5	0 %
2.3	Pruebas de la implementación de los requisitos avanzados.	Héctor Gómez Ariño	10/06/2020	12/06/2020	3	0 %
3	**Lanzamiento Aplicación y Publicación Repositorio**					
3.1	Preparar juego para compilar en multi-plataforma.	Héctor Gómez Ariño	8/06/2020	12/06/2020	5	0 %
3.2	Investigación librerías de exportación.	Héctor Gómez Ariño	8/06/2020	12/06/2020	5	0 %
3.3	Preparar los ejecutables y emuladores en las diferentes plataformas.	Héctor Gómez Ariño	8/06/2020	12/06/2020	5	0 %
3.4	Subir a GITHub los ejecutables junto con el código.	Héctor Gómez Ariño	8/06/2020	12/06/2020	5	0 %
4	**Seguimiento y mejoras Juego**					
4.1	Crear listado de issues en gitHub con posibles implementaciones futuras.	Héctor Gómez Ariño	1/06/2020	12/06/2020	12	0 %
4.2	Implementación/Corrección de issues.	Héctor Gómez Ariño	1/06/2020	19/06/2020	19	0 %
4.3	Mantenimiento técnico (Actualización de la plataforma)	Héctor Gómez Ariño	18/06/2020	19/06/2020	2	0 %
5	**Fase de lanzamiento Juego**					
5.1	**Presentación previa entrega**	**Héctor Gómez Ariño**	**15/06/2020**	**15/06/2020**	**1**	**0 %**
5.2	Actualización y revisiones finales.	Héctor Gómez Ariño	17/06/2020	19/06/2020	3	0 %
5.3	Redacción Memoria.	Héctor Gómez Ariño	1/06/2020	19/06/2020	19	0 %
5.4	Redacción presentación.	Héctor Gómez Ariño	15/06/2020	19/06/2020	5	0 %
5.5	**Presentación proyecto**	**Héctor Gómez Ariño**	**22/06/2020**	**22/06/2020**	**1**	**0 %**

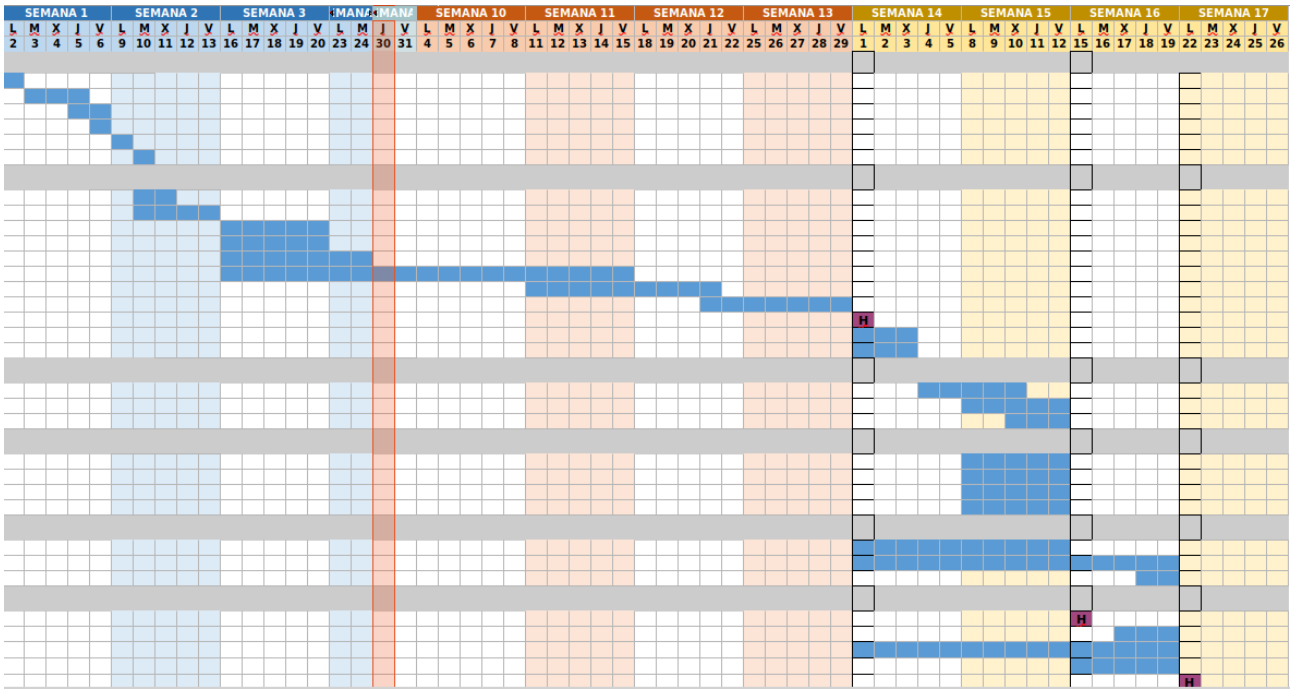
Los hitos temporales serán:

1. Hito 0: 16/3/2020: inicio del desarrollo propiamente dicho.
2. 1/6/2020: Presentación Versión Beta.
3. 15/06/2020: Presentación versión previa.
4. 22/06/2020: Presentación del proyecto.

Aproximadamente el calendario será el siguiente:

- Marzo, dos primeras semanas: Análisis del proyecto, búsqueda de información y planificación del desarrollo.
- Desde mitad de marzo hasta final de mayo: Fase de desarrollo e inicio de documentación.
- Junio: otra iteración, mejora de funcionalidades y documentación. Preparación de la exposición del proyecto.

Este es el detalle de las tareas anteriores calendarizadas, se pueden apreciar los hitos marcados al final de cada iteración. La fase de desarrollo está colapsada para mejor apreciación. **(Ver adjunto Diagrama de Gantt)**



A pesar de que tenemos esta planificación incial, estimamos que habrá cambios y proveeremos una más realista finalizado el proyecto.

Costes del proyecto

Sin tener en cuenta el tiempo personal dedicado y los equipos de trabajo, no se espera

que el proyecto tenga costes materiales. Aún así podemos detallar el siguiente listado de requisitos de los que se podría extraer un coste material teórico:

- GitHub: Nos provee un repositorio de código amigable y abierto de manera gratuíta, será muy práctico para almacenar el software de manera estable.
- Unity3D: Ofrece una versión de uso con licencia personal gratuíta que abaratará los costes del proyecto. También dispone de una biblioteca de Assets con la que podremos trabajar para proveernos de los materiales y sprites necesarios.
- Visual Code Editor: también tiene una versión de licencia gratuíta con integración con Unity.
- Necesitaremos un equipo de trabajo potente, mínimo 8Gb de RAM (con menos Unity se ralentiza en exceso).
- Conexión a la corriente eléctrica e internet.

Riesgos previstos

Listamos a continuación algunos riesgos previstos durante el proyecto:

- Tener que desarrollar una aplicación no nativa y que se pueda ejecutar en diferentes entornos es un riesgo a tener en cuenta.
- El desconocimiento previo de Unity3D y demás herramientas puede ser un riesgo porque requerirá un tiempo de investigación y aprendizajes extra.
- No sabemos como responderán los gráficos de orígenes diferentes. No deberían ser demasiado divergentes con el diseño global del videojuego.

Conclusiones y agradecimientos

Habiendo terminado el proyecto, desglosamos los resultados obtenidos del trabajo realizado estos meses.

Conclusiones técnicas

Al tratarse de un proyecto de fin de ciclo con un caracter muy técnico, se requería presentar un proyecto con despliegue tecnológico avanzado, integrando diferentes herramientas que permitan versatilidad y multi-plataforma.

Los requisitos técnicos y funcionales no estaban claros cuando empezamos con el desarrollo, así que fuimos adaptando el mismo a los objetivos a lo largo del desarrollo.

Para lograr el objetivo inicial de tener una aplicación multi-plataforma recurrimos a Unity3D, aplicación en la que a través de Applets y librerías externas que se pueden integrar en la misma, creando ejecutables para diferentes plataformas. No hubo hitos temporales ni técnicos marcados por los requisitos con lo que fuimos realizando el desarrollo en grandes bloques.

Este proyecto ha permitdio mejorar en el conocimiento de una herramienta novedosa y muy extendida en el desarrollo de videojuegos. Aunque no se han aplicado conocimientos previos del equipo en campos como el desarrollo web, las Bases de Datos y otros. Sí que se han podido aplicar técnicas de desarrollo orientadas a objetos y clases.

Ha sido posible realizar un ejecutable final lógico y coherente con los objetivos iniciales del proyecto y la planificación teórica.

No ha sido posible entregar el proyecto dentro de los plazos iniciales, y hemos tenido que adaptar la planificación teórica (no solamente el gantt realizado) pues no pudimos contactar antes para establecer los plazos con el tutor. En principio teníamos pensado entregar el proyecto a mediados de Mayo, finales como tarde.

Conclusiones personales

En cuanto a las conclusiones personales, podemos decir que ha sido complicado utilizar una herramienta nueva, la inversión de tiempo total ha sido muy grande y ha sido realmente complejo.

Nos gustaría ser más positivos en estas conclusiones, pero a muchos niveles no hemos quedado satisfechos; no tanto con el trabajo realizado sino con los tiempos y plazos establecidos.

Después de meses de trabajo, el resultado no es óptimo, como todo en programación y muchas cosas en esta vida, es susceptible de mejora o nuevas funcionalidades. Pero como dijo un sabio de la antiguedad:

"[...] hacer muchos libros no tiene fin y dedicarse demasiado a ellos es agotador para cualquiera [...]"

Así, que aplicándome estas palabras concluyo que por más que nos esforcemos siempre habrá otro que lo haga mejor, o siempre lo habríamos podido hacer mejor. Con lo que es vital estar contento con lo que se hace y como se hace aunque hay que reconocer humildemente que una mejora es siempre posible y casi siempre beneficiosa.

Una vez más, como nos ha acostumbrado todo el curso de formación a distancia DAM, la ayuda y el soporte ha sido muy escasa. Nos habría gustado más guía al menos unas grandes

líneas funcionales o para la redacción del presente.

Finalmente, no podemos dejar de mencionar que la presente documentación junto con la presentación realizada han sido harto farragosas y hastiantes. De nuevo hemos de recalcar que las 99 horas teóricas que debería haber durado el proyecto han sido sobrepasadas con creces y nos habría faltado tiempo aún para muchas mejoras que han quedado en el tintero, hablamos de ellas a continuación.

Evidentemente devido al confinamiento por la pandemia de COVID-19, a la que no podemos dejar de hacer mención este proyecto será presentado online, no tenemos problema pero habría sido un final más redondo poder hacer la presentación físicamente con todas las consecuencias.

Agradecimientos

En un apartado extra en el que no pretendía entrar, no tengo menos que agradecer a toda la gente que sube tutoriales semi-desinteresadamente a youtube para que podamos utlizar los que estamos empezando con herramientas de desarrollo como Unity.

También he de agradecer la academia online openwebiars.net, gracias a sus cursos hemos podido entender mucho mejor conceptos generales antes de iniciarnos en la programación y nos ha permitido hacer esta memoria mucho más detallada a nivel técnico.

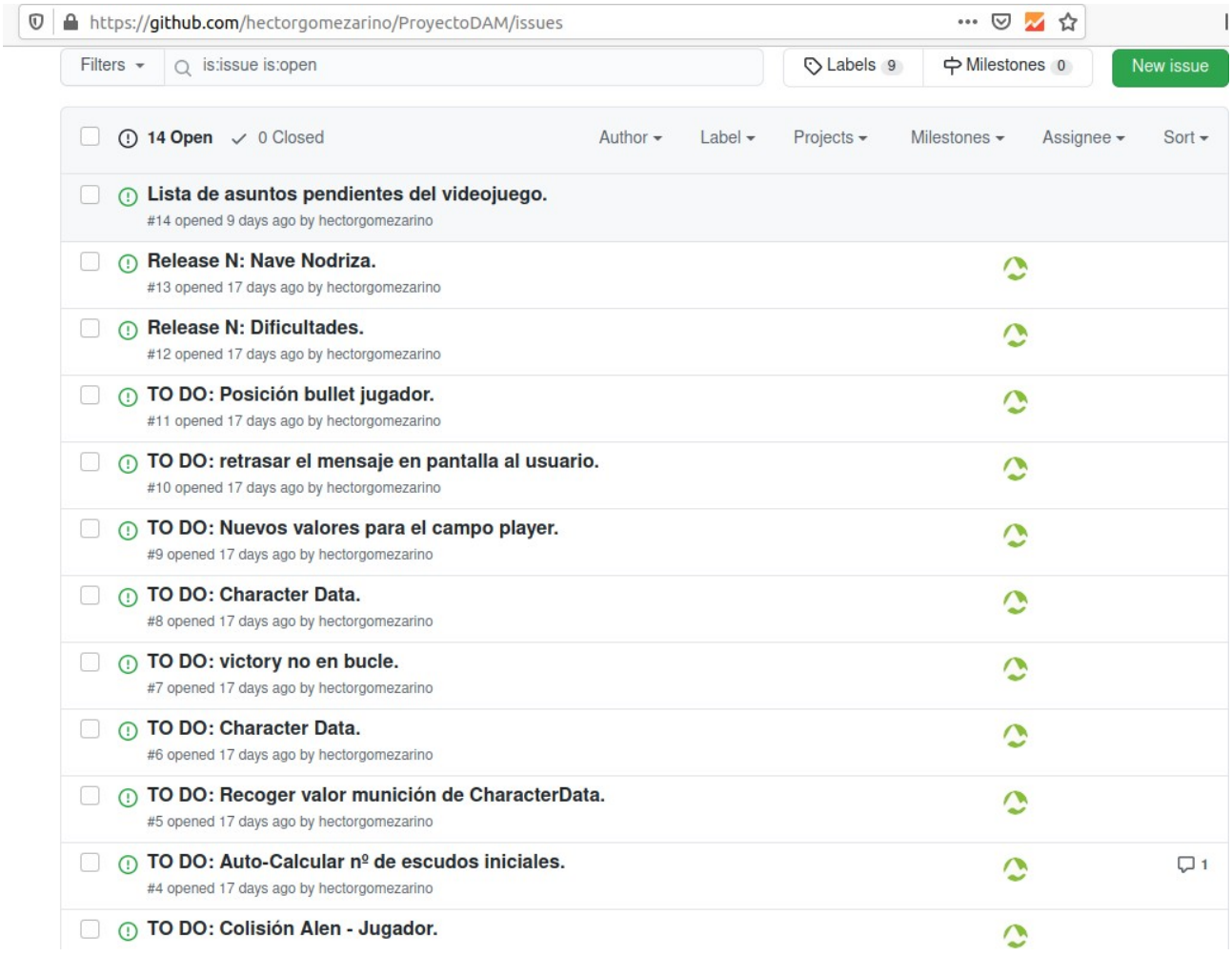
Líneas de investigación y desarrollo futuras.

En caso de que se quisiera proseguir el trabajo, dejamos aquí unos pasos a seguir para continuar con el mismo.

- Se deberían utilizar diferentes branch de GIT para trabajar en paralelo diferentes pasos de la aplicación.
- Incluir en el control de versiones tags con las mismas.
- Liberar el código completamente, tanto a nivel de licencia como de documentación y publicación.
- Añadir un sistema de estadísticas con las puntuaciones de los jugadores dentro del juego.
- Corregir algunos bugs.
- Sería muy interesante poder utilizar la última versión de GIT, pues no ha sido posible que compilara la aplicación tras migrar contenido, assets e imporar librerías.

Por otro lado podemos destacar que gracias al uso de GitHub ya tenemos capacidad para listar errores detectados, dejamos aquí el link con la referencia a las issues detectadas: https://github.com/hectorgomezarino/ProyectoDAM/issues

Dejo un extracto del listado actual, para más detalle consultar el link superior:



Aprendizaje Técnico:

Sería muy interesante a nivel técnico realizar un aprendizaje específico sobre Unity y cómo trabjar con colisiones entre objetos. A nivel específico parece muy interesante, pues hemos visto manuales para generar estelas de movimiento, tracking, pathfinding, y "explosiones" de partículas. Todas estas funcionalidades serían muy interesantes de añadir a nuestro videojuego para mejorar los gráficos. A nivel ténico sería muy bueno tener más tiempo para mejorar aún más el desarrollo, aunque entendemos que el tiempo es limitado, según las especificidades iniciales: 99 horas.

Hipotéticamente se podría plantear subir la aplicación a "mercados" de aplicaciones para facilitar su descarga, pero al tratarse de multi-plataforma no hicimos el planteamiento al inicio y no hemos invertido tiempo en ello.

Bibliografía.

Referencias: libros o artículos citados en el trabajo y Bibliografía complementaria.

Vídeos de Youtube:

- https://www.youtube.com/watch?v=DNLAuV-d4sA
- https://www.youtube.com/watch?v=XFbNTeW1d8
- *https://www.youtube.com/watch?v=Mn6lUik3nyk*
- *https://www.youtube.com/watch?v=OPYPiQziLns*
- *https://www.youtube.com/watch?v=cnfwNzpoIlA*
- *https://www.youtube.com/watch?v=QgEKrJWe8f4*
- *https://www.youtube.com/watch?v=49RmygswEBE*

TestLink:

- https://www.adictosaltrabajo.com/2013/05/07/test-link/
- testlink.com

Unity:

Iput Field Unity:
- https://www.youtube.com/watch?v=u1ht64_abzM
- https://www.youtube.com/watch?v=UnEDxN0DEa4

Curso open webinars de Unity:
- https://openwebinars.net/academia/aprende/unity/5456/

Get Component Unity:
- https://docs.unity3d.com/ScriptReference/GameObject.GetComponent.html

Paso de Variables entre Scipts Unity:
- https://answers.unity.com/questions/699565/how-to-get-a-variable-value-from-another-scriptc.html
- https://docs.unity3d.com/ScriptReference/PlayerPrefs.SetInt.html

Player prefs:
- https://docs.unity3d.com/es/530/ScriptReference/PlayerPrefs.html

Guardar datos del juego (player, model...):
- https://unity3d.com/es/learn/tutorials/topics/scripting/introduction-saving-and-loading

****
********** Guía del videojuego: **********
****

En esta página hemos encontrado diferentes manuales, aunque antiguos nos han servido para desarrollar nuestro videojuego en Unity y nos han provisto de Assets:

- http://www.newthinktank.com/2017/06/make-video-games-4/
- http://www.newthinktank.com/2017/06/make-video-games-5/
- http://www.newthinktank.com/2017/06/make-video-games-6/
- http://www.newthinktank.com/2017/06/make-video-games-7/

Otros:

- Documentación de GIT: https://git-scm.com/docs
- Curso online open webinars de GIT: https://openwebinars.net/cursos/git
- Wikipedia: http://www.wikipedia.org/
- Modelo espiral: https://es.wikipedia.org/wiki/Archivo:Modelo_Espiral_Boehm.jpg
- Metodología Agile: ttps://upload.wikimedia.org/wikipedia/commons/c/c7/Waterfall_Vs_Agile_m%2Cmethod.png
- Polimorfismo, diagrama: https://commons.wikimedia.org/wiki/File:Polimorfismo_diagrama.png
- Polimorfismo, diagrama2: https://es.m.wikipedia.org/wiki/Archivo:Polimorfismo-ES.svg