Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Diseño y Desarrollo de Videojuegos

Desarrollo del videojuego: “Scavenger Rover”

Trabajo fin de estudio presentado por:	David Cazar Marco González Washington Quillupangui
Tipo de trabajo:	Videojuego 3D
URL del repositorio de código	https://drive.google.com/drive/folders/1MvoLklytuEGyAOyYYrwcnro519p2B3gO?usp=sharing
URL del juego	https://drive.google.com/drive/folders/16CMCloANItAqYzI6-dq4oHrheiamwVHY?usp=sharing
URL del tráiler de presentación (si procede)	https://www.youtube.com/watch?v=yrb3wOjAbQs
Director/a:	FRAILE NARVÁEZ, MARCELO
Fecha:	2021/09/15

Resumen

Este trabajo busca presentar al lector el proceso seguido para el desarrollo del video juego “Scavenger Rover”. Para ello se ha ejecutado a propuesta acorde al tiempo establecido para culminar el presente Master.

El propósito es determinar las bases de conocimiento artístico y técnico necesario para el desarrollo del mismo. Mediante el ejercicio práctico expuesto en los siguientes apartados, se desea transmitir la experiencia y aprendizaje obtenidos durante el master.

Para la ejecución del proyecto se ha utilizado metodologías relacionadas al diseño de videojuegos generando la documentación necesaria, lo que ha permitido realizar la planificación y seguimiento de actividades, cumpliendo con cada hito planteado en el proyecto.

“Scavenger Rover” está soportado por el motor Unreal Engine 4, tomando ventaja en la experiencia de los autores, la facilidad para crear proyectos y su potencia gráfica. Como resultado, este trabajo presenta los diferentes recursos construidos durante el desarrollo del proyecto como su código fuente, texturas, modelos, etc. Todo disponible para el entendimiento del lector.

El desarrollo de videojuegos es una aventura artística y técnica muy compleja que podría parecer simple. Se propone exponer los retos y desafíos a los que se enfrentan los desarrolladores de videojuegos, por más altas las expectativas en el videojuego a desarrollar, se debe mantener los pies en la tierra.

Palabras clave: Exoplaneta, Transporte, Meca, Parásitos, videojuego

Abstract

This paper seeks to present to the reader the process followed for the development of the video game "Scavenger Rover". For this, the proposal has been executed according to the time established to complete this Master.

The purpose is to determine the bases of artistic and technical knowledge necessary for its development, through the practical exercise exposed in the following sections, to transmit the experience and learning during the master.

For the execution of the project, methodologies related to the design of video games have been used, generating the necessary documentation, which has allowed the planning and monitoring of activities, complying with each milestone set in the project.

"Scavenger Rover" is supported by the Unreal Engine 4 engine, taking advantage of the authors' experience, the ease of creating projects and its graphical power. As a result, this work presents the different resources built during the development of the project such as its source code, textures, models, etc. All components are available for the reader's understanding.

Game development is a very complex artistic and technical adventure that might seem simple. It is proposed to expose the challenges that video game developers face, no matter how high the expectations in the video game to be developed, you must keep your feet on the ground.

Keywords: Exoplanet, Transport, Mecha, Parasites, videogame

Índice de contenidos

1. Introducción 14
1.1. Ficha técnica 14
1.2. Público Objetivo 14
1.3. Objetivos del juego 14
1.4. Referencias 15
2. El concepto del videojuego 18
2.1.1. Ambientación 18
2.1.2. Mecánicas principales 18
2.1.3. Dinámicas principales 18
2.1.4. Diseño artístico 19
2.1.5. Resumen de la Historia 19
3. Planificación del proyecto 20
3.1. Preproducción 21
3.2. Producción 21
3.3. Postproducción / Mantenimiento 22
3.4. Roles en el desarrollo 23
3.4.1. Arte 23
3.4.2. Programación 23
3.4.3. Diseño 24
3.4.4. Producción 24
3.4.5. Actividades 24
3.4.6. Temporización / Cronograma de actividades 26
4. Diseño arquitectónico del software 28
4.1. gravedad de un exoplaneta 29

4.1.1. Instalación del Plugin 29
4.1.2. Arquitectura del plugin 30
4.1.3. Explicación del funcionamiento del componente gravitatorio 30
4.1.4. Usabilidad del plugin 34
4.2. Arquitectura del proyecto 36
4.3. Gameplay Framework Classes 44
4.4. Implementación 45
4.4.1. Implementación de movimientos básicos del MECHA 45
4.4.2. Implementación del sistema de disparo de mecha 48
4.4.3. Implementación de ia para movimiento de enemigos 50
4.4.4. Implementación de sistema de orientación (vista aérea) 52
4.4.5. Implementación de linterna para buscar minerales 53
4.4.6. Implementación de minerales para ser visto por la linterna 54
4.4.7. De creación de tiendas mineras para su extracción 55
4.4.8. Implementación de diálogos 57
4.4.9. Creación de sistema de tutoriales: vista aérea, enemigos y extracción de minerales 58
5. GDD de alto nivel 59
5.1. Alcance del juego 59
5.2. Diseño visual 60
5.3. Gameplay y mecánicas 64
5.4. Control 65
5.5. Cámara 65
5.6. Personajes 65
5.6.1. El capitán "Matthews Gathely" 66

5.6.2. Inteligencia artificial Betty 68
5.6.3. Cápsula de Minado XLM Tipo Taladro 69
5.6.4. El robot (Mecha) cuadrúpedo K2CP (KONG BOT) 71
5.7. Enemigos 73
5.7.1. General 73
5.7.2. Los Krongs "Bettle" 74
5.8. Niveles 76
5.8.1. Resumen 76
5.8.2. Elementos 76
5.9. Objetos e Ítems 78
5.9.1. Faro 78
5.9.2. Salud 78
5.10. Screenflow 79
5.10.1. Diseño de la UI 79
5.11. Narrativa 80
6. Conclusiones 80
7. Postmorten del juego 81
8. Referencias bibliográficas y de otros videojuegos 83

Índice de figuras

Figura 1. Helldivers (Helldivers Saldrá a La Venta El 3 de marzo En PS4, PS3 y PS Vita, n.d.) .15

Figura 2. Mario Galaxy (Free Super Mario Galaxy Wallpaper | Super Mario Galaxy, Galaxy Wallpaper, Super Mario, n.d.) 16

Figura 3. Deiland (Análisis - Deiland — Legión de Jugadores, n.d.) 17

Figura 4. Starcraft 2 (Star Craft 2 - 1366_2000.Jpg (1050×760), n.d.) 17

Figura 5. Imagen del plugin en el store de EPIC 2 (Elaboración propia) 29

Figura 6. Imagen del plugin en el store de EPIC 3 (Elaboración propia) 30

Figura 7. Contenido general del plugin (Elaboración propia)..... 30

Figura 8. Carpeta de Characters del plugin (Elaboración propia) 31

Figura 9. Lógica del blueprint donde setea la gravedad (Elaboración propia)..... 31

Figura 10. Lógica del blueprint donde setea la gravedad 2 (Elaboración propia)..... 32

Figura 11. Lógica del blueprint donde setea la gravedad 3 (Elaboración propia)..... 32

Figura 12. Componente Set Gravity Direction (Elaboración propia)..... 32

Figura 13. Código C++ para la función SetGravityDirection (Elaboración propia) 33

Figura 14. Código C++ para la función GetGravityDirection (Elaboración propia)..... 33

Figura 15. Código C++ para la función PhysFalling (Elaboración propia) 33

Figura 16. Código C++ de donde se obtiene la velocidad para la gravedad (Elaboración propia) 34

Figura 17. Código C++ de donde se muestra la aplicación de la gravedad (Elaboración propia) 34

Figura 18. Lógica del blueprint donde setea la gravedad 6 (Elaboración propia)..... 35

Figura 19. Personaje creado con características de gravedad hacia el centro (Elaboración propia) 35

Figura 20. Arquitectura general del proyecto (Elaboración propia) 36

Figura 21. Contenido de la carpeta Blueprints (Elaboración propia) 36

Figura 22. Contenido de la carpeta Characters (Elaboración propia) 37

Figura 23. Contenido de la carpeta Core (Elaboración propia) 37

Figura 24. Contenido de la carpeta Developers (Elaboración propia) 38

Figura 25. Contenido de la carpeta Enum (Elaboración propia) 38

Figura 26. Contenido de la carpeta Font_CajadeTexto (Elaboración propia) 38

Figura 27. Ejemplo de visualización de la fuente (Elaboración propia) 39

Figura 28. Contenido de la carpeta HUD (Elaboración propia) 39

Figura 29. Contenido de la carpeta Maps (Elaboración propia) 39

Figura 30. Contenido de la carpeta Megascans (Elaboración propia) 40

Figura 31. Contenido de la carpeta MiniMap (Elaboración propia) 40

Figura 32. Comparación mini mapa v/s vista aérea (Elaboración propia) 41

Figura 33. Contenido de la carpeta MSpresets (Elaboración propia) 41

Figura 34. Contenido de la carpeta PointCloudsMorphing (Elaboración propia) 42

Figura 35. Vista general de nubes (Elaboración propia) 42

Figura 36. Contenido de la carpeta Realistic_Starter_VFX_Pack_Niagara (Elaboración propia) 42

Figura 37. Ejemplo de partículas de misiles del MECHA (Elaboración propia) 42

Figura 38. Contenido de la carpeta Sequences (Elaboración propia) 43

Figura 39. Contenido de la carpeta Textures_Planet (Elaboración propia) 43

Figura 40. Contenido de la carpeta Weapons (Elaboración propia) 44

Figura 41. Framework Class Relationships (Unreal Architecture | Unreal Engine Documentation, n.d.) 45

Figura 42 Inputs de movimiento (Elaboración propia) 45

Figura 43. Configuración manual de Inputs de movimiento (Elaboración propia) 46

Figura 44. Código del Macro del input (Elaboración propia) 46

Figura 45. Archivo donde guarda los inputs (Elaboración propia) 47

Figura 46. Configuración de inputs por código (Elaboración propia) 47

Figura 47. Rotación de cámara (Elaboración propia) 47

Figura 48. Configuración manual de Inputs de cámara (Elaboración propia) 48

Figura 49. Adjunta un arma al MECHA (Elaboración propia) 48

Figura 49. Creación de los spawner de las balas (Elaboración propia) 48

Figura 51. Creación de las balas (Elaboración propia) 49

Figura 52. Interfaz de comunicación con Niagara (Elaboración propia) 49

Figura 53. Sistema de partículas de disparo con Niagara (Elaboración propia) 49

Figura 54. Disparo del arma (Elaboración propia) 50

Figura 55. Lógica de sobrecalentamiento de las armas (Elaboración propia) 50

Figura 56. Lógica del movimiento de los enemigos hacia su base (Elaboración propia) 51

Figura 57. Lógica del ataque de los enemigos al MECHA (Elaboración propia) 51

Figura 58. Lógica de los enemigos al recibir daño (Elaboración propia) 51

Figura 59. Lógica de los enemigos al morir (Elaboración propia) 51

Figura 60. Lógica de las animaciones del enemigo Beetle (Elaboración propia) 52

Figura 61. Visualización de la cámara aérea desde el Viewport (Elaboración propia) 52

Figura 62. Lógica del cambio de cámara (Elaboración propia) 52

Figura 63. Visualización de la implementación de la cámara aérea (Elaboración propia) 53

Figura 64. Lógica de la búsqueda de minerales con la linterna (Elaboración propia) 53

Figura 65. Visualización de la linterna en el Viewport (Elaboración propia) 54

Figura 66. Ejemplo de la implementación de la linterna en el juego (Elaboración propia) 54

Figura 67. Lógica de cómo se muestran o esconden los minerales (Elaboración propia) 55

Figura 68. Implementación de las bases mineras (Elaboración propia) 55

Figura 69. Lógica de la extracción de minerales parte 1 (Elaboración propia) 56

Figura 70. Lógica de la extracción de minerales parte 2 (Elaboración propia) 56

Figura 71. Lógica de la extracción de minerales parte 3 (Elaboración propia) 57

Figura 72. Implementación del widget de diálogos (Elaboración propia) 57

Figura 73. Lógica de ejemplo de creación de diálogos (Elaboración propia) 58

Figura 74. Lógica de los tutoriales (Elaboración propia) 58

Figura 75. Conceptos iniciales del juego. (Elaboración propia) 60

Figura 76. Propuestas iniciales del juego. (Elaboración propia) 61

Figura 77. Moodboards y propuestas de arte. (Elaboración propia) 62

Figura 78. Diseño, texturas y materiales de Kong-Bot. (Elaboración propia) 63

Figura 79. KONG BOT. (Elaboración propia) 64

Figura 80. KONG BOT. Vista isométrica (Elaboración propia) 64

Figura 81. Primer boceto del capitán Gathely. (Elaboración propia) 66

Figura 82. Matt. (Elaboración propia) 67

Figura 83. Diseño terminado de Matt exportado en UE4 (Elaboración propia) 68

Figura 84. Betty. (Elaboración propia) 68

Figura 85. Cápsula de minado. (Elaboración propia) 69

Figura 86. Proceso de minado. (Elaboración propia) 70

Figura 87. KONG BOT. (Elaboración propia) 71

Figura 88. KONG BOT dispara. (Elaboración propia) 72

Figura 89. Krongs Vistas isométricas. (Elaboración propia) 73

Figura 90. Los Krongs "Bettle" importados en Unreal Engine4. (Elaboración propia) 74

Figura 91. Diseño de nivel. (Elaboración propia) 76

Figura 92. Moodboard de planetoide. (Elaboración propia) 77

Figura 93. Screen Flow. (Elaboración propia) 79

Figura 94. HUD. (Elaboración propia).....80

Índice de tablas

Tabla 1. Gantt del Proyecto 26

1. INTRODUCCIÓN

1.1. FICHA TÉCNICA

Videojuego de exploración y disparos, basado en una era postapocalíptica, el personaje principal es un mercenario retirado que dispone de una nave para viajar a los exoplanetas, un robot (Mecha) de exploración equipado con armas y detectores de yacimientos de minerales, el personaje también dispone de equipo minero para la extracción y recolección de minerales.

- **Género,** shooter de doble stick y exploración en planetoides
- **Número de jugadores,** un jugador
- **Plataformas,** ordenadores (PC)
- **Tipo de cámara**, juego en tercera persona
- **Modelo de negocio,** Pay to Play
- **Target objetivo,** hombres entre 15 y 20 años
- **Motor de desarrollo,** Unreal Engine 4
- H**erramientas de modelado y texturizado,** Autodesk Maya, Zbrush y Sustance

1.2. PÚBLICO OBJETIVO

El público objetivo considera a hombres y mujeres entre 25 a 35 años que tengan preferencia por los juegos acción y exploración. Se incluyen de tipo mid-core y hardcore

1.3. OBJETIVOS DEL JUEGO

El objetivo del presente trabajo es crear un prototipo de un videojuego que pueda tener mecánicas bases y escalables para poder desarrollar un nivel de juego, por otro lado, incorporar un estilo artístico con gráficos 3D acordes a las exigencias del mercado.

Para esto se plantearon los siguientes objetivos específicos:

- Creación de mecánica de exploración de un exoplaneta en busca de yacimientos de minerales, buscados con un detector.
- Extracción de minerales, creando una base minera.
- Eliminar enemigos que amenacen la extracción de minerales.

- Generación de tutoriales para explicación de historia y mecánicas principales.

1.4. REFERENCIAS

Dentro de la investigación, se analizaron numerosos videojuegos existentes en el mercado que sirvieron como inspiración en distintos ámbitos, rescatando los aspectos más relevantes de cada uno, analizando la forma de como resolvieron mecánicas similares al proyecto en cuestión, a continuación, se muestran las referencias más importantes:

- **Helldivers**, es un videojuego del género shooter de doble stick, cooperativo y para jugadores expertos, desarrollado por la empresa Arrowhead Game Studios y publicado por PlayStation Mobile, Inc, lanzado en la plataforma Steam el 7 de diciembre de 2015 (*Helldivers Saldrá a La Venta El 3 de marzo En PS4, PS3 y PS Vita*, n.d.). Lo que más se rescata de este juego es el sistema de combate, por la relevancia que tiene este proyecto. Otro de los aspectos importantes de Helldivers, es que el jugador viaja entre planetas distintos mostrando distintos biomas en su ambientación, resolviendo misiones y derrotando distintos enemigos característicos de cada uno de estos planetas, la mecánica de combate es muy dinámica y sin duda es una gran inspiración para el proyecto.

Figura 1. Helldivers (Helldivers Saldrá a La Venta El 3 de marzo En PS4, PS3 y PS Vita, n.d.)

- **Mario Galaxy**, es un videojuego de plataformas en planetoides desarrollado y publicado el 1 de noviembre de 2007 en Japón por la empresa Nintendo. (*Super Mario*

Galaxy (Wii) Game Profile | News, Reviews, Videos & Screenshots, n.d.). Este videojuego destaca por la jugabilidad en planetoides pequeños, en donde se desenvuelve Mario Bros, el personaje principal de la saga. Si bien no es la primera vez que aparecen planetoides en la saga de Mario, en esta entrega se observa diversidad en atmósfera y mecánicas, que permiten explorar ideas para ser consideradas en Scavenger Rover.

Figura 2. Mario Galaxy (Free Super Mario Galaxy Wallpaper | Super Mario Galaxy, Galaxy Wallpaper, Super Mario, n.d.)

- **Deiland**, es un videojuego de aventuras combinando elementos de RPG y mecánicas de Sandbox, desarrollado por la empresa española Chibig lanzado el 12 de febrero de 2016 en múltiples plataformas. (*Deiland*, n.d.) Deiland es un juego en el cual se puede observar las mecánicas de construcción y cultivo, además de combate con enemigos, al ser un planetoide tiene dinámicas muy distintas a otros tipos de juegos similares, por esa razón es digno de análisis para el proyecto.

Figura 3. Deiland (Análisis - Deiland — Legión de Jugadores, n.d.)

- **Starcraft 2,** es un videojuego de ciencia ficción de estrategia en tiempo real desarrollado y publicado por la empresa Blizzard Entertainment el 27 de Julio de 2010 en la plataforma Windows y MacOS. (*StarCraft II: Wings of Liberty - Wikipedia*, n.d.) Una de las mecánicas principales y de inspiración, es la recolección de recursos, la cual deben crear vehículos que busquen minerales o gas y transportarlos a una base. Principalmente se intentaron hacer mecánicas similares, pero por motivos de tiempo se tuvo que simplificar.

Figura 4. Starcraft 2 (Star Craft 2 - 1366_2000.Jpg (1050×760), n.d.)

2. EL CONCEPTO DEL VIDEOJUEGO

2.1.1. Ambientación

La ambientación está basada en un futuro postapocalíptico en donde la población humana está concentrada en el único planeta estable y seguro de la galaxia, sin embargo, ya no quedan materias primas, y es por eso que han tenido que viajar a pequeños planetoides infestados de enemigos con el fin de conseguirlas y prosperar en el tiempo. Aún existe vegetación, pero la mayoría de los planetas son desérticos, que será la ambientación del planeta que usaremos para este prototipo. Es un mundo hostil, desértico con pasajes y rocas altas, se debe sentir una aparente calma porque todo está bajo tierra tanto los recursos como la mayoría de enemigos, la idea es que el jugador se sienta libre de investigar y buscar recursos, pero con el peligro de ser atacado en cualquier momento.

2.1.2. Mecánicas principales

- Caminar: El personaje puede caminar en cualquiera de las direcciones que desee en 360 grados, con una velocidad constante en todas sus direcciones.
- Saltar: El personaje puede saltar levemente, sin superar el doble de su altura, esto sirve para subirse a rocas pequeñas, llegar a lugares que le permitan resguardase de los peligros.
- Disparar: El personaje puede disparar de dos formas, una con un arma tipo ametralladora (Munición infinita), con una limitante al sobrecalentarse el cañón, y otra con una carga de misiles teledirigidos (finitos).
- El personaje puede interactuar con tecnología para dar información de eventos

2.1.3. Dinámicas principales

- Explorar el planeta
- Exterminar las criaturas para limpiar el área de minado
- Exterminar las criaturas para proteger el equipo de extracción
- Identificar yacimientos de minerales.
- Marcar los yacimientos para que el equipo de extracción haga su trabajo

2.1.4. Diseño artístico

A continuación, se define las características de diseño y arte escogidos para el proyecto. Scavenger Rover es un juego basado en una historia postapocalíptica en un futuro distante, sus principales componentes a definir bajo este aspecto es el personaje principal "Matthews Gathely" el capitán, un robot (Mecha) llamado K2CP o mejor conocido como KONG BOT, y la maquinaria usada para extraer minerales de los yacimientos. También se revisará el arte de los enemigos alienígenas de tipo insecto llamados "The Krongs" los cuales habitan el exoplaneta. El planetoide XDESRt de tipo desértico, aunque lleno de recursos, está infestado por los enemigos alienígenas. Finalmente hay diferentes artes, que aparecerán en cinemáticas o diálogos dando apoyo a la historia del videojuego.

El personaje jugable es el robot (Mecha) cuadrúpedo K2CP (KONG BOT) similar a un gorila, tiene aproximadamente 7 metros de altura, tiene una capacidad para 6 tripulantes, 4 en la cabina principal y 2 en la cabina posterior, se podría decir que esta bestia mecánica es un veterano de guerra, ha luchado en las guerras antiguas y actualmente es la pieza más importante en la minería de planetoides, para el diseño se necesitaba mostrar esta fortaleza, tanto en forma como en materiales, que aunque gastados y sucios se nota su buena calidad, es un robot (Mecha) pesado pero con una fuerza impresionante, aplasta a cualquier enemigo que se cruce en su camino, a pesar de su aspecto por sus gigantes brazos delanteros puede moverse ágilmente, está preparado para explorar cualquier planeta y enfrentar cualquier amenaza.

En base a lo mencionado se detalla (puede ser en los anexos) las características usando moodboars de cada elemento y después el diseño final, cabe destacar que todo el arte conceptual y la creación 3D de modelados y texturas han sido realizadas desde cero y en exclusiva para el proyecto. En cuanto a los assets y materiales del planeta, se usan recursos de la biblioteca de Megascans.

2.1.5. Resumen de la Historia

En un futuro distante la Tierra había adoptado gobernaciones regionales para cada continente, la Unión Europea, la Unión Americana y la unión africana con el objetivo de eliminar las fronteras entre países convirtiendo cada continente en una gran nación. A pesar de alcanzar su objetivo los recursos naturales comenzaron a escasear cada vez más. La lucha

por territorio y recursos mantuvo la guerra de los 100 años. Durante ese periodo la humanidad desarrollo herramientas y tecnología para la extracción de recursos y armamento de combate. En todo el globo la tierra se volvió estéril, se agotaron los minerales y, en consecuencia, animales y humanos sufrieron hambruna y muerte.

Para el año 2100 se celebra el acuerdo de paz y sostenibilidad entre las gobernaciones continentales, permitiendo a la humanidad darse un respiro para buscar fuera de la galaxia los recursos necesarios para sobrevivir, creando la tecnología de teletransportación llamada "Teleport 3". Durante 20 años la humanidad fue reduciéndose rápidamente, en consecuencia, las estructuras de gobierno desaparecieron dando paso a pequeñas tribus. Ya para el año 2120 ha desaparecido del 99% de la población.

El personaje principal es un veterano de guerra de 58 años llamado "Matthews Gathely" conocido como Matt, proveniente del extremo sur de la región americana, ha decidido aventurarse fuera de la galaxia en busca de recursos para su tribu, llamada "Almma 7" en honor a los 7 grandes fundadores, de los cuales solo queda uno vivo, conocido como "El gran Toki".

Matt lleva años manteniendo su equipo de la guerra, que consta de una nave que le permite llegar a los exoplanetas, un robot (Mecha) K2CP llamado KONG BOT de combate y rastreo, y un equipo de extracción de recursos. Toda la tecnología es controlada con la ayuda de la Inteligencia Artificial "Abi" compañera inseparable de Matt.

3. PLANIFICACIÓN DEL PROYECTO

Para este apartado se ha puesto mucha atención a los roles, fases y métodos de gestión, cabe destacar que si bien es una introducción a la metodología utilizada cada uno de los ítems serán ampliados en los siguientes apartados.

Parte importante de la metodología de desarrollo es identificar claramente los roles del equipo de desarrollo, para ello se toma en cuenta los roles básicos que por lo general se encuentran en el proceso de producción (Llansó García, 2014, p. 11):

- Artista
- Programador
- Diseñador

En cuanto a las fases, para el proceso de desarrollo depende de cada proyecto, aunque independiente del tamaño se puede distinguir las fases (Chandler, 2009):

3.1. PREPRODUCCIÓN

En esta etapa se han establecido los integrantes del equipo de desarrollo y sus roles. Debido al tamaño del equipo se comparten roles y responsabilidades.

Para este proyecto también se establecen las mecánicas principales sobre las cuales se construirá el videojuego.

La dirección de arte es uno de los principales puntales en los cuales se desarrolla este proyecto y se ha soportado con la experiencia de uno de los autores. Con el fin de dar una ambientación artística acorde a los objetivos planteados para el proyecto.

Se ha elaborado la propuesta de videojuego, que se detalla en el apartado de introducción, así también las diferentes líneas artísticas y narrativas.

Por último, la elección de tecnologías en las cuales se desarrolla el proyecto y herramientas tecnológicas que acompañan el desarrollo del mismo.

3.2. PRODUCCIÓN

Para esta etapa se ha planteado el uso de las herramientas de gestión tratadas en este trabajo. También el desarrollo y uso de los documentos de diseño necesario para el desarrollo de video juegos.

Por otro lado, la ejecución de actividades y tareas, detalladas en el apartado de planificación, se han distribuido entre los diferentes integrantes del equipo. El uso de un cronograma permite la identificación de retrasos o errores inesperados.

Una vez identificado los roles y las fases es importante seleccionar una metodología de gestión que permita garantizar entregas funcionales y jugables en el menor tiempo posible, este tipo de requerimiento, se ajusta a las metodologías basadas en iteraciones cortas que hasta 2001 se las denominaba métodos livianos, a partir de este año se creó el "manifiesto ágil" que sintetiza los valores y principios de los métodos livianos (Keith, 2010):

- Individuos e interacciones sobre procesos y herramientas
- Software funcional sobre documentación completa

- Colaboración con el cliente sobre la negociación de contratos
- Responde al cambio sobre el plan en ejecución

Estos valores dieron paso a diferentes frameworks como Scrum. Lean y XP (Keith, 2010).

En cuanto a Scrum como metodología podemos resumir que (Llansó García, 2014):

- La planificación se realiza en base a historias que definen una necesidad del juego.
- Estas historias se reparten entre los integrantes del equipo.
- Se puede mejorar cualquier parte del desarrollo.
- Se mantiene integración continua sobre los contenidos.
- En lo desarrollado, no debe existir errores que interfieran con el progreso del equipo.
- Estas historias se agrupan en iteraciones llamadas Sprints
- Una agrupación de hitos más grande se denomina Release.
- Existe una lista completa de las historias necesarias o Backlog
- Cada Sprint puede detallar una Historia en un subconjunto de tareas.
- Se realizan dos tipos de reuniones, diarias y al final del Sprint.

En base a lo mencionado en los ítems anteriores Scrum permite un control sobre pequeñas versiones del videojuego las cuales al ser entregadas en cada Sprint inmediatamente pueden ser testeadas, corregidas o cambiadas dando visibilidad continua del avance del videojuego.

Para el presente proyecto se ha optado por usar Scrum como metodología de gestión.

Durante esta etapa se han encontrado diferentes problemas al momento del desarrollo, cuando los integrantes han propuesto incluir modificaciones o nuevos desarrollos que tienen un costo en tiempo y en recursos no estimado.

El control de la etapa de producción es muy crítico para el cumplimiento con la entrega del videojuego y para el presente caso el prototipo.

3.3. POSTPRODUCCIÓN / MANTENIMIENTO

Para el proyecto que se ha presentado en este trabajo no se han considerado de manera formal estas etapas, debido a las limitaciones y riesgos encontrados en la etapa de producción por lo que será importante considerar estas etapas de manera teórica y no practica dentro de este trabajo.

3.4. ROLES EN EL DESARROLLO

Actualmente el equipo está conformado por tres miembros, cabe destacar que es un equipo multidisciplinario lo que ha facilitado asignar los roles necesarios para el desarrollo del videojuego.

3.4.1. Arte

Los artistas crean el arte conceptual, las animaciones, los modelos 3D, las texturas 2D y cualquier otro elemento gráfico del juego (Chandler, 2020).

Dado que el equipo de desarrollo es pequeño, los artistas del equipo pueden ser generalistas, lo que significa que tienen la capacidad de crear diferentes tipos de assets artísticos (Chandler, 2020).

Este rol es ocupado por David Cazar, y fue el encargado de hacer todo el apartado artístico, desde elegir la ambientación y línea artística del juego, modelar los personajes, enemigos, la nave, la creación del mapa y la UI, además también hizo las animaciones y efectos especiales del MECHA.

3.4.2. Programación

Son responsables de crear la tecnología para cada aspecto del juego, incluida la física, el rendimiento, la inteligencia artificial, los gráficos, las herramientas de secuencias de comandos, el audio, la iluminación, el movimiento del jugador, etc. (Chandler, 2020)

Para este rol el equipo dispone de dos personas: Marco González y Washington Quillupangui, los cuales crearon las mecánicas del juego, el disparo, el movimiento del MECHA, la creación de cinemáticas, el comportamiento de los enemigos, y todo lo relacionado a los tutoriales dentro del juego. Se utilizó principalmente el lenguaje de Blueprints de Unreal Engine 4, con el conocimiento que se obtuvo en uno de los ramos impartidos en este Master.

Además, antes de llegar a las mecánicas actuales, anteriormente se hicieron pruebas de mecánicas de naves y movimientos en planetoides de distinto tamaño, con el objetivo de ir iterando las ya mencionadas mecánicas.

3.4.3. Diseño

La función principal del diseñador es crear una experiencia de juego atractiva e inmersiva. Son responsables de crear todos los "verbos" en el juego, es decir, las cosas que un jugador puede hacer e interactuar. Esto incluye el esquema de control, los sistemas de juego (combate, comercio, subir de nivel, etc.), narrativa e historia, antecedentes y personalidades de los personajes, misiones y objetivos, diseños de niveles, etc. (Chandler, 2020).

Para este rol y el giro que tiene el Máster se ha tomado en cuenta a los 3 integrantes del equipo, en las cuales al tener todos visiones distintas fueron varias ideas las que se fueron probando para que se sintiera bien las mecánicas del personaje y pueda tener relación a la historia del juego, algunas cosas que se desestimaron fue por ejemplo un radar o mini mapa que estaba en el lado derecho del juego, el cual fue reemplazado por una vista aérea, simulando la visión de la nave que deja al MECHA en el planeta, el cual fue una decisión de diseño. Otra mecánica que se diseñó, pero no se implementó fue el sistema de extracción de minerales de tipo puzle, ya que sintió era muy engorroso para el jugador, el cual debería ir descubriendo donde están los minerales.

3.4.4. Producción

La producción, como cualquier otra disciplina, es un oficio que se puede aprender y mejorar con la experiencia. Algunas personas pueden ser más adecuadas para el rol que otras porque la producción requiere una fuerte combinación de habilidades organizativas y de liderazgo (Chandler, 2020).

En sí, la producción engloba todas las actividades que permitan al equipo lograr cada hito y llegar a la meta por lo que este rol es de vital importancia para coordinar y gestionar las actividades y los diferentes desafíos que tendrá el equipo durante el desarrollo del videojuego.

En este rol se ha designado a Washington Quillupangui como productor.

3.4.5. Actividades

3.1.5.1 Preproducción:

- Definición del estilo artístico moodboard general
- Creación de boceto y moodboards de arte para MECHA
- Creación de boceto y moodboards de arte para 4 distintos enemigos

- Creación de boceto y moodboards para Matt
- Creación de boceto y moodboards para Betty
- Creación de boceto y moodboards para nave principal
- Creación de boceto y moodboards para planetoide
- Creación de boceto y moodboards para Interfaz gráfica
- Diseño de mecánicas de movimiento del MECHA
- Diseño de mecánicas de disparo de MECHA
- Diseño de movimiento de enemigos (Beetle)
- Diseño de sistema de orientación
- Diseño de sistema de rastreo de minerales
- Diseño de sistema de extracción de minerales o "mining" (Linterna)
- Diseño de sistema de diálogos
- Diseño de tutoriales

3.1.5.1 Producción:

- Moodboard de escenarios, definición de estilo grafico
- Modelado, textura, animaciones, shading y rigging de MECHA (KongBot)
- Modelado, textura, animaciones, shading y rigging de un enemigo
- Modelado, textura, animaciones, shading y rigging de Matt
- Modelado, textura, animaciones, shading y rigging de Betty
- Modelado, textura, animaciones, shading y rigging para nave principal
- Modelado, textura y shading para planetoide
- Ilustración vectorial para Interfaz gráfica
- Definición de cantidad de planetoides y definición de biodiversidad
- Creación de level designer (primera etapa)
- Creación de partículas y FX como nubes, gas o disparos
- Creación y programación e implementación de prototipo del primer nivel (Blockout)
- Creación y programación e implementación de mecánicas principales para prototipo del primer nivel (Blockout)
- Montaje de primer nivel con assets gratuitos del store de Epic Game
- Creación y programación e implementación de cinemática inicial en UE4

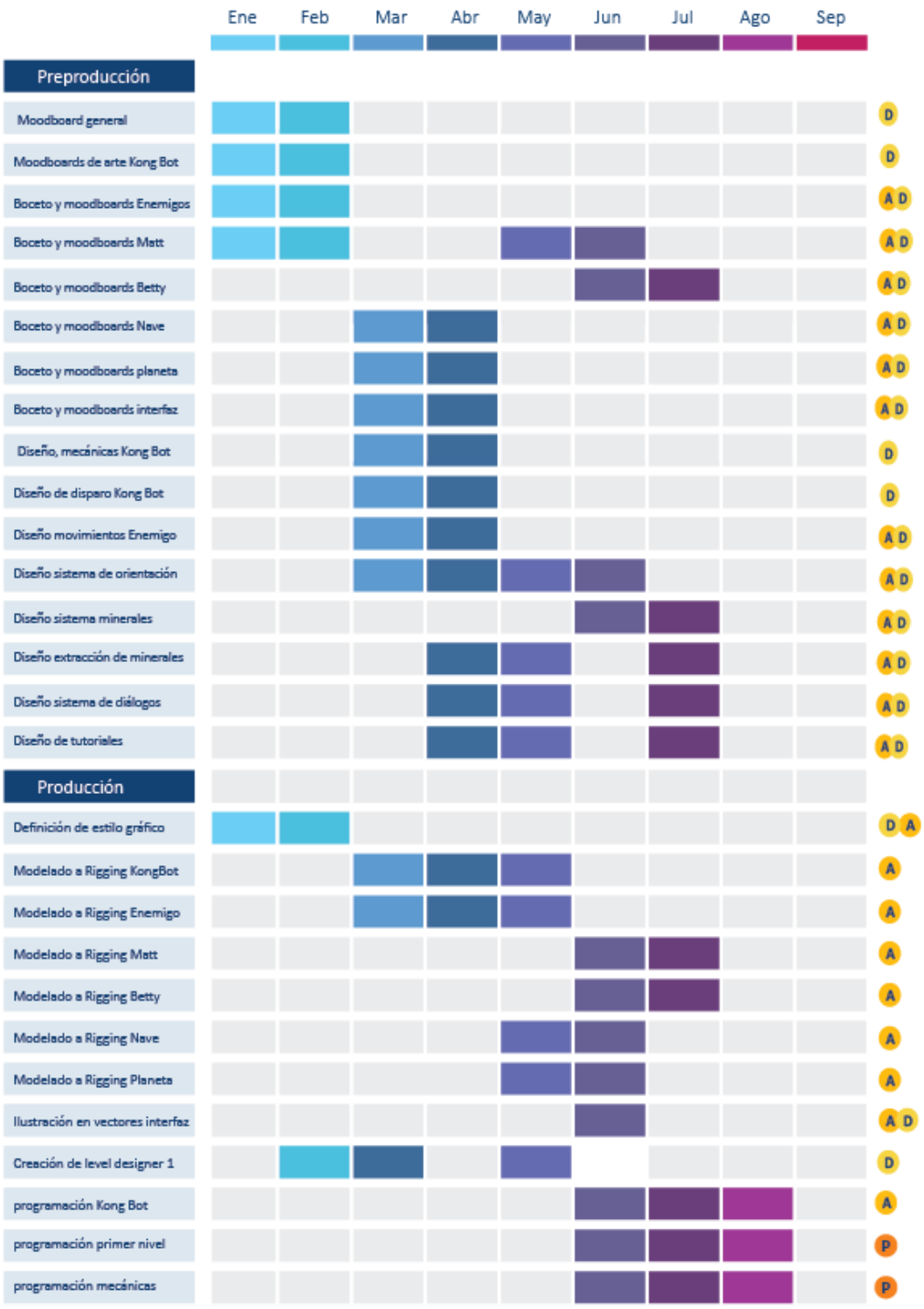
- Creación y programación e implementación de Mecánica de movilidad básica del MECHA para el demo
- Implementación de linterna para buscar minerales
- Implementación de minerales para ser visto por la linterna
- Implementación de creación de tiendas mineras para su extracción
- Implementación de diálogos

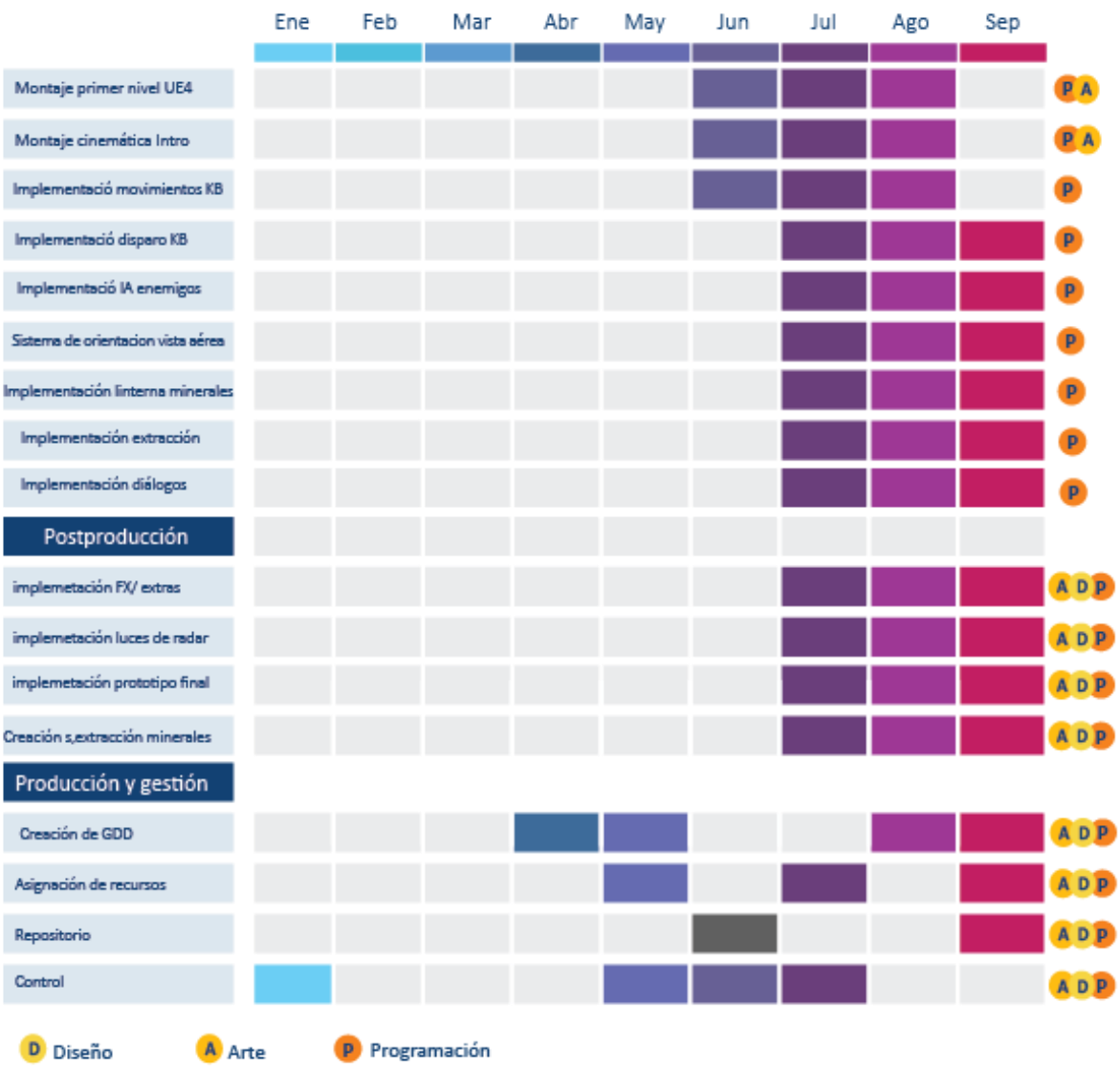
3.1.5.3 Postproducción:

- programación e implementación de y FX como disparos y explosiones
- Creación y programación e implementación de un enemigo con movimiento básico para Vertical Slice
- Creación, programación balance e implementación del prototipo final jugable
- Creación de sistema de tutoriales: Vista aérea, enemigos y extracción de minerales

3.4.6. Temporización / Cronograma de actividades.

Tabla 1. Gantt del Proyecto


Ene
Feb
Mar
Abr
May
Jun
Jul
Ago
Sep
Preproducción
Moodboard general
D
Moodboards de arte Kong Bot
D
Boceto y moodboards Enemigos
A D
Boceto y moodboards Matt
A D
Boceto y moodboards Betty
A D
Boceto y moodboards Nave
A D
Boceto y moodboards planeta
A D
Boceto y moodboards interfaz
A D
Diseño, mecánicas Kong Bot
D
Diseño de disparo Kong Bot
D
Diseño movimientos Enemigo
A D
Diseño sistema de orientación
A D
Diseño sistema minerales
A D
Diseño extracción de minerales
A D
Diseño sistema de diálogos
A D
Diseño de tutoriales
A D
Producción
Definición de estilo gráfico
D A
Modelado a Rigging KongBot
A
Modelado a Rigging Enemigo
A
Modelado a Rigging Matt
A
Modelado a Rigging Betty
A
Modelado a Rigging Nave
A
Modelado a Rigging Planeta
A
Ilustración en vectores interfaz
A D
Creación de level designer 1
D
programación Kong Bot
A
programación primer nivel
P
programación mecánicas
P



4. DISEÑO ARQUITECTÓNICO DEL SOFTWARE

Para la implementación de los diferentes componentes a nivel de software es necesario entender la arquitectura con la cual está desarrollado el núcleo de Unreal y su framework para el desarrollo.

Los Actores son instancias de clases que se derivan de la clase AActor; la clase base de todos los objetos de juego que se pueden colocar en el mundo. Los objetos son instancias de clases que heredan de la clase UObject; la clase base de todos los objetos en Unreal Engine, incluidos Actors. Entonces, en realidad, todas las instancias en Unreal Engine son Objetos; sin embargo, el término Actores se usa comúnmente para referirse a instancias de clases que derivan de AActor en su jerarquía. (*Unreal Architecture | Unreal Engine Documentation*, n.d.)

Aquí en la terminología de Unreal se hace una distinción entre Actores y Objetos, para un real la terminología de objetos se refiere a las diferentes clases que no hereden de la clase AActor. Estas clases se pueden denominar especializadas. (*Unreal Architecture | Unreal Engine Documentation*, n.d.)

4.1. GRAVEDAD DE UN EXOPLANETA

El primer desafío fue crear un exoplaneta con gravedad hacia el centro, la intención es que el jugador pueda dar la vuelta completa a un planeta, con el objetivo de que pueda explorar buscando minerales, es por eso por lo que se debía crear un sistema de gravedad distinto al que Unreal Engine 4 trae por defecto. Se optó finalmente por usar un plugin de gravedad gratuito de la Epic Store llamado “Directional & Planet Gravity”.

4.1.1. Instalación del Plugin

Para su instalación, primero se debe abrir el Launcher de Epic games e ir a la pestaña bazar tal como se muestra en la siguiente imagen y buscar el plugin gratuito llamado “Directional & Planet Gravity” y presionar en el botón “Instalar en el Motor”,

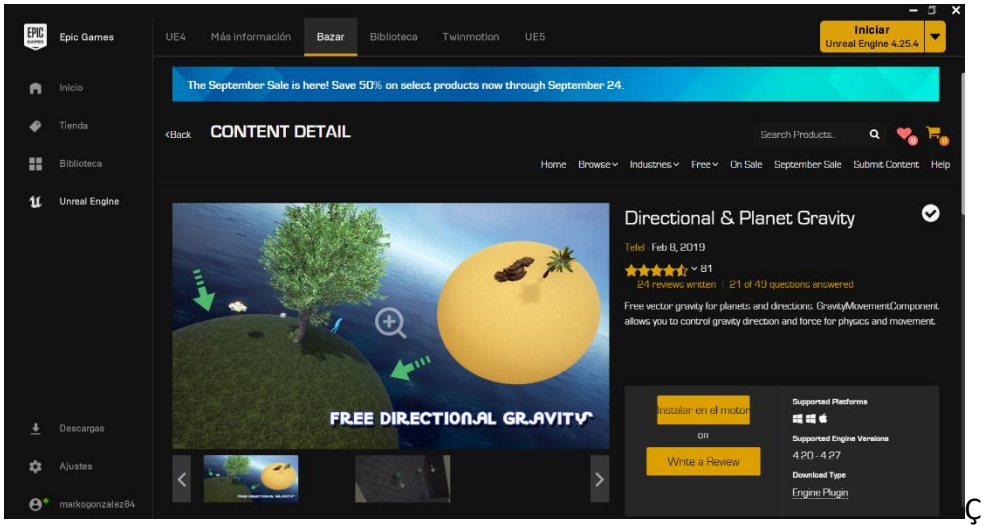


Figura 5. Imagen del plugin en el store de EPIC 2 (Elaboración propia)

y nos preguntará en que versión se quiere instalar.

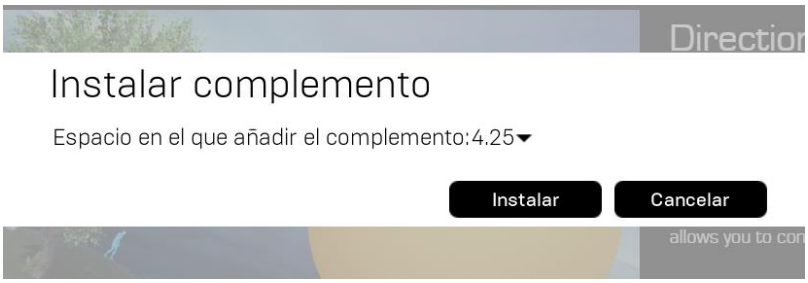


Figura 6. Imagen del plugin en el store de EPIC 3 (Elaboración propia)

4.1.2. Arquitectura del plugin

Ya teniendo el plugin instalado en el motor se puede usar en el proyecto, la arquitectura de este plugin al ser externo no se profundizará, ya que lo importante es conocer su funcionamiento, la arquitectura general del plugin se puede ver en la siguiente imagen:

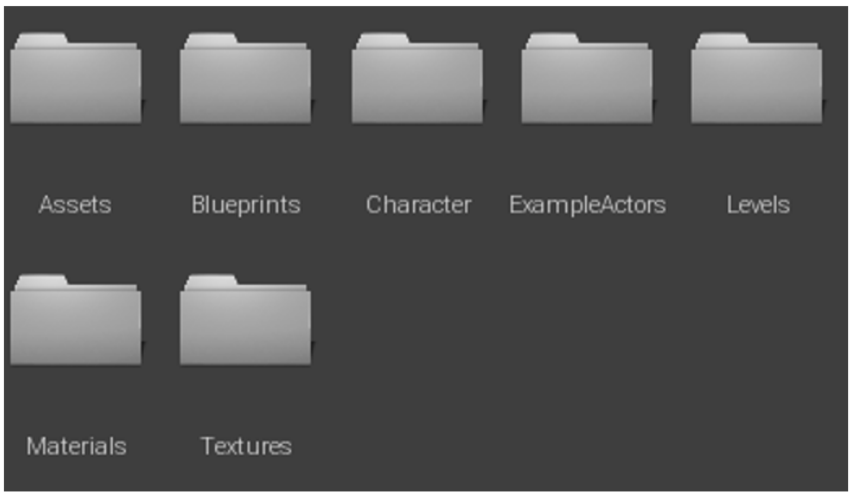


Figura 7. Contenido general del plugin (Elaboración propia)

4.1.3. Explicación del funcionamiento del componente gravitatorio

En la carpeta Character se encuentra el Blueprint llamado "BP_HumanGravityCharacter" del cual hereda el personaje principal y funciona perfectamente con la gravedad del planetoide.

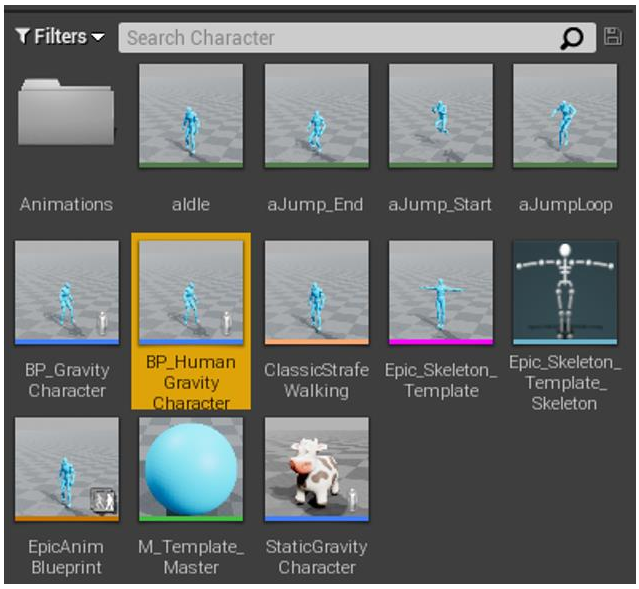


Figura 8. Carpeta de Characters del plugin *(Elaboración propia)*

Este Blueprint hereda de “BP_GravityCharacter” el cual llama a la lógica de “Set Gravity Direction” por vía Blueprint tal como se muestra a continuación:

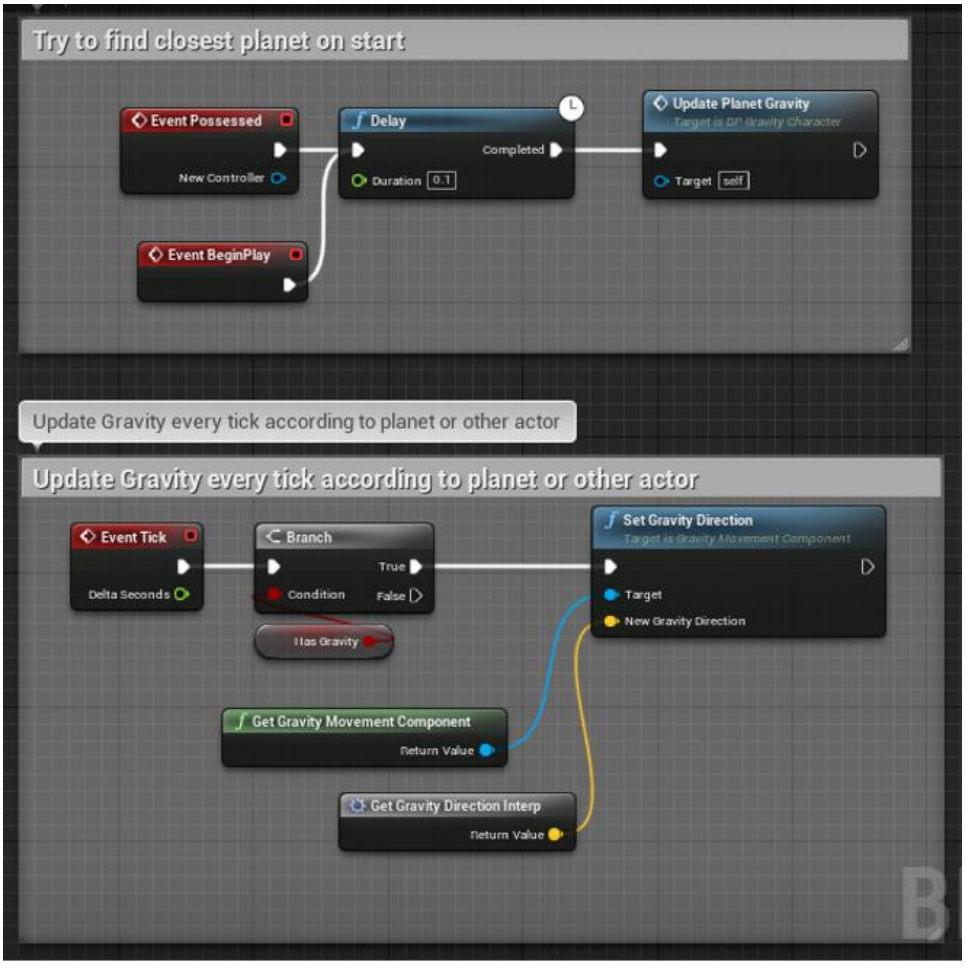


Figura 9. Lógica del blueprint donde setea la gravedad (Elaboración propia)

Y finalmente este Blueprint hereda de la clase en C++ GravityCharacter que es la que hace toda la lógica de la gravedad hacia el centro.

El funcionamiento de esto, se ve primero en la macro "Get Gravity Direction Interp" en donde busca un vector direccional desde el personaje hacia el centro,

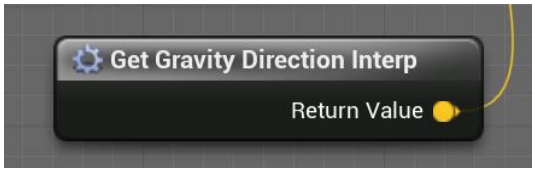


Figura 10. Lógica del blueprint donde setea la gravedad 2 (Elaboración propia)

Tal como se ve en el siguiente Blueprint, en donde calcula el vector desde el personaje hacia el centro y va haciendo una interpolación por cada "Delta Second" para que la rotación a medida que avanza el personaje se vea fluida.

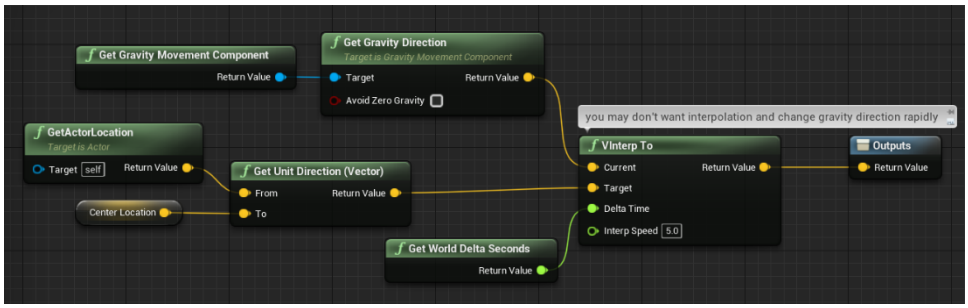


Figura 11. Lógica del blueprint donde setea la gravedad 3 (Elaboración propia)

Luego de obtener este vector se setea la dirección en cada tick,

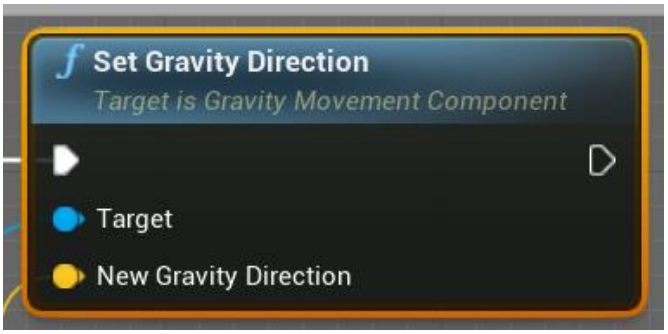


Figura 12. Componente Set Gravity Direction (Elaboración propia)

Este es nos lleva al código C++ del componente de gravedad que es el núcleo del plugin, primero setea la variable CustomGravityDirection

```
69  void UGravityMovementComponent::SetGravityDirection(FVector NewGravityDirection)
70  {
71      CustomGravityDirection = NewGravityDirection.GetSafeNormal();
72  }
```

Figura 13. Código C++ para la función SetGravityDirection (Elaboración propia)

```
40  FVector UGravityMovementComponent::GetGravityDirection(bool bAvoidZeroGravity) const
41  {
42      // Gravity direction can be influenced by the custom gravity scale value.
43      if (GravityScale != 0.0f)
44      {
45          if (!CustomGravityDirection.IsZero())
46          {
47              return CustomGravityDirection * ((GravityScale > 0.0f) ? 1.0f : -1.0f);
48          }
49
50          const float WorldGravityZ = Super::GetGravityZ();
51          if (bAvoidZeroGravity || WorldGravityZ != 0.0f)
52          {
53              return FVector(0.0f, 0.0f, ((WorldGravityZ > 0.0f) ? 1.0f : -1.0f) * ((GravityScale > 0.0f) ? 1.0f : -1.0
54          }
55      }
56      else if (bAvoidZeroGravity)
57      {
58          if (!CustomGravityDirection.IsZero())
59          {
60              return CustomGravityDirection;
61          }
62
63          return FVector(0.0f, 0.0f, (Super::GetGravityZ() > 0.0f) ? 1.0f : -1.0f);
64      }
65
66      return FVector::ZeroVector;
67  }
```

Figura 14. Código C++ para la función GetGravityDirection (Elaboración propia)

Esta es última función se usa en la función "PhysFalling"

```
1025  void UGravityMovementComponent::PhysFalling(float deltaTime, int32 Iterations)
1026  {
```

Figura 15. Código C++ para la función PhysFalling (Elaboración propia)

Que contiene el código donde se calcula la velocidad real que se le debe dar para que nuestro personaje este siempre impulsado hacia el centro del planeta, es importante aclarar, tal como aparece en la documentación que esta velocidad está siendo afectada por el componente de Gravedad en el Movement Component de nuestro Character, es decir que se puede manipular de igual forma directamente de este componente.

```
1081          // Compute Velocity.
1082          {
1083              // Acceleration = FallAcceleration for CalcVelocity(), but we restore it after using it.
1084              TGuardValue<FVector> RestoreAcceleration(Acceleration, FallAcceleration);
1085
1086              Velocity = FVector::VectorPlaneProject(Velocity, GravityDir);
1087              CalcVelocity(TimeTick, FallingLateralFriction, false, BrakingDecelerationFalling);
1088              Velocity = FVector::VectorPlaneProject(Velocity, GravityDir) + OldVelocityZ;
1089
1090
1091          }
```

Figura 16. Código C++ de donde se obtiene la velocidad para la gravedad (Elaboración propia)

Finalmente, es aplicada con el siguiente código

```
1100        // Apply gravity.
1101        const FVector Gravity = GetGravity();
1102        Velocity = NewFallVelocity(Velocity, Gravity, TimeTick);
1103        VelocityNoAirControl = NewFallVelocity(VelocityNoAirControl, Gravity, TimeTick);
1104        const FVector AirControlAccel = (Velocity - VelocityNoAirControl) / TimeTick;
1105
1106        if (bNotifyApex && CharacterOwner->Controller && ((Velocity | GravityDir) * -1.0f) <= 0.0f)
1107        {
1108            // Just passed jump apex since now going down.
1109            bNotifyApex = false;
1110            NotifyJumpApex();
1111        }
1112
1113        // Move now.
1114        FHitResult Hit(1.0f);
1115        FVector Adjusted = 0.5f * (OldVelocity + Velocity) * TimeTick;
1116
1117        SafeMoveUpdatedComponent(Adjusted, PawnRotation, true, Hit);
```

Figura 17. Código C++ de donde se muestra la aplicación de la gravedad (Elaboración propia)

4.1.4. Usabilidad del plugin

Ya se conoce como instalar el plugin y su funcionamiento interno, ahora se mostrará la forma que se utilizó en el proyecto. Primero se debe crear un character heredado del objeto ya mencionado en la sección anterior llamado “BP_HumanGravityCharacter”

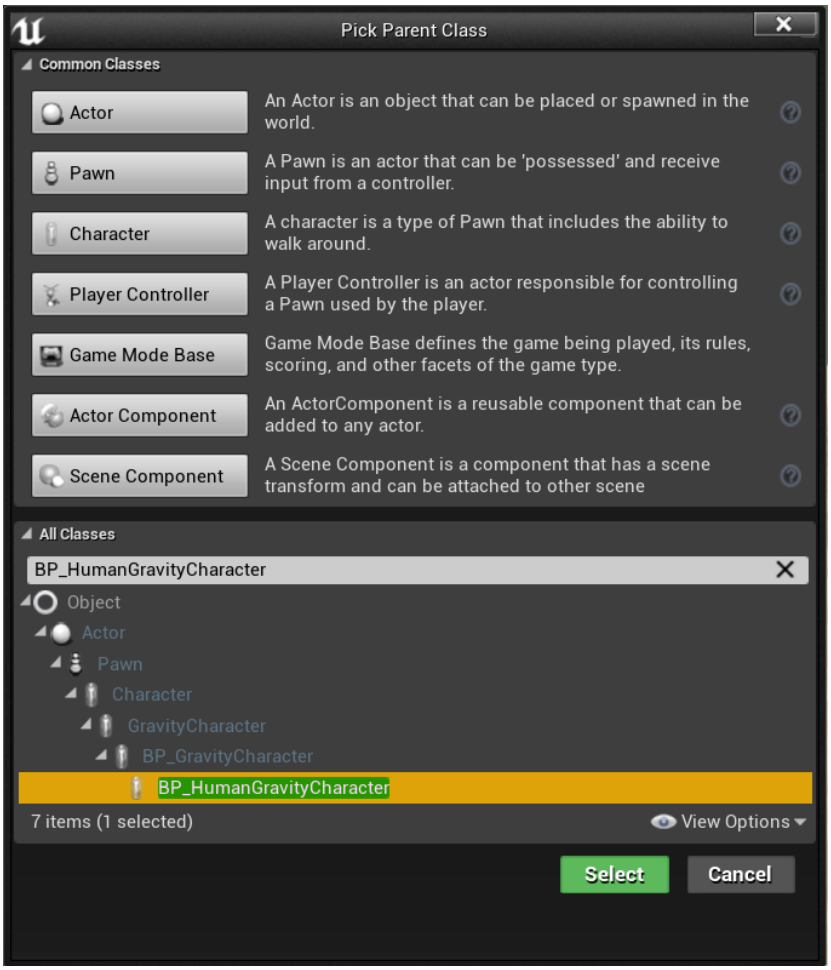


Figura 18. Lógica del blueprint donde setea la gravedad 6 (Elaboración propia)

Y con esto, ya se tienen todas las características de la gravedad en nuestro personaje y se encuentra listo para modificarlo y programar sus demás características

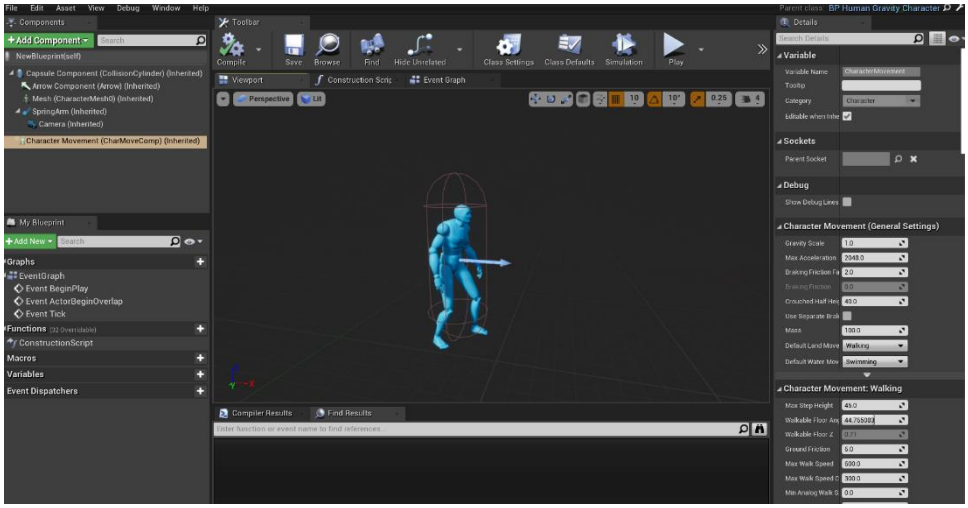


Figura 19. Personaje creado con características de gravedad hacia el centro (Elaboración propia)

4.2. ARQUITECTURA DEL PROYECTO

La Arquitectura del proyecto está separada por carpetas, el detalle de cada una se nombra a continuación:

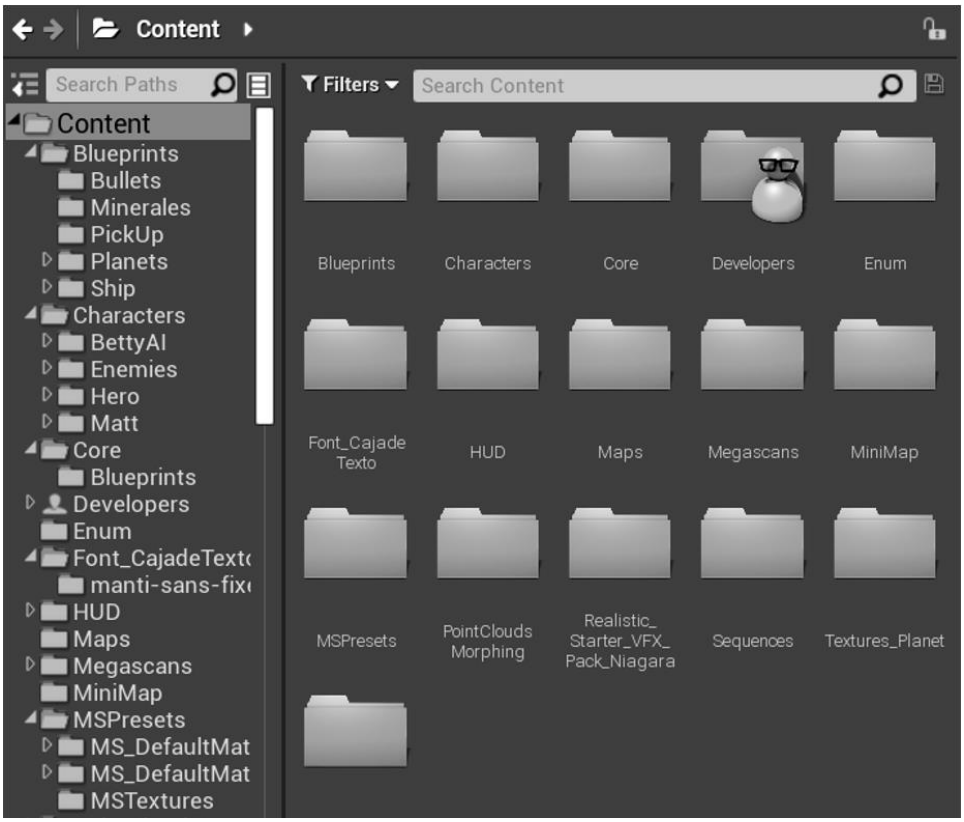


Figura 20. Arquitectura general del proyecto (Elaboración propia)

- **Carpeta Blueprints**

Contiene los objetos interactuables más importantes del juego, como balas, minerales, baterías, centros mineros, los planetas y la nave.

Figura 21. Contenido de la carpeta Blueprints (Elaboración propia)

- **Carpeta Characters**

Contiene todos los personajes que existen en el juego, Betty, los enemigos (para este demo solo los Beetle), el Hero (que es el MECHA) y Matt que es el personaje que controla el MECHA, para este demo no lo utilizaremos, pero la intención es que en ciertos lugares se pueda bajar de este.

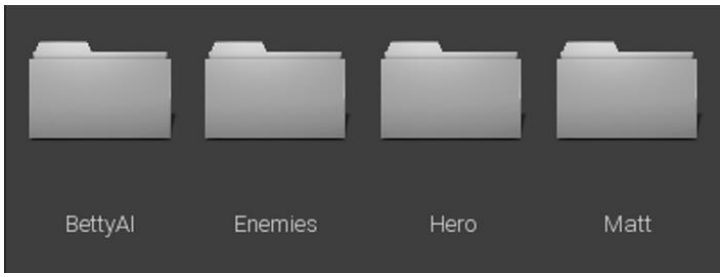


Figura 22. Contenido de la carpeta Characters (Elaboración propia)

- **Carpeta Core**

Como el nombre lo indica son los blueprints que son los más importantes del juego, el GameMode donde están las bases del juego, el BP_Player que es nuestro personaje principal, se separa de la carpeta Characters, ya que acá está toda la lógica y es más fácil encontrarlo. Finalmente, la interfaz ClimbInterface, sirve para detectar en que lugares el personaje puede trepar, pero esta característica no está considerada para esta entrega.

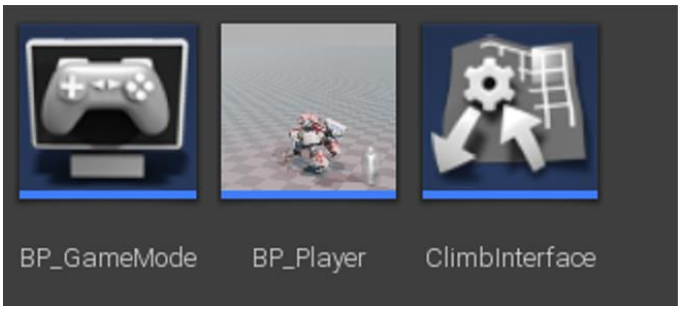


Figura 23. Contenido de la carpeta Core (Elaboración propia)

- **Carpeta Developers**

Esta carpeta tiene subcarpetas de todos los integrantes del team, sirve para hacer pruebas antes de que queden en producción si es necesario (en la carpeta content), el proyecto al buildearse no las considera, ya que por defecto Unreal entiende que son para testing.

Figura 24. Contenido de la carpeta Developers (Elaboración propia)

- **Carpeta Enum**

Contiene todos los enumeradores del juego, actualmente solo contiene el estado de una mina, y nos sirve para saber el estado de las minas con el fin de poner íconos distintos para ser vistos de la vista aérea (que es la vista de la nave) y para un eventual mini mapa. Para esta versión hemos dejado de lado el mini mapa, ya que, si bien está implementado, por concetos de jugabilidad decidimos dejarlo fuera, ya que con la vista aérea es suficiente, y le damos también menos información al jugador para que tenga que explorar el planeta por su cuenta.

Figura 25. Contenido de la carpeta Enum (Elaboración propia)

- **Carpeta Font_CajadeTexto**

Contiene la fuente ocupada en los textos del juego

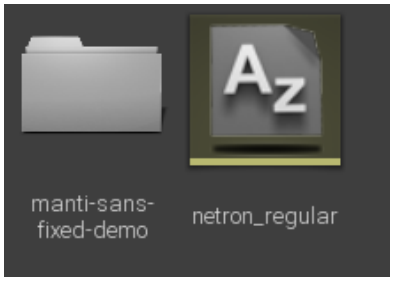


Figura 26. Contenido de la carpeta Font_CajadeTexto (Elaboración propia)

Figura 27. Ejemplo de visualización de la fuente (Elaboración propia)

- **Carpeta HUD**

Administra todas las interfaces del juego, diálogos introductorios, tutoriales, acciones dentro del juego, eventos, etc.

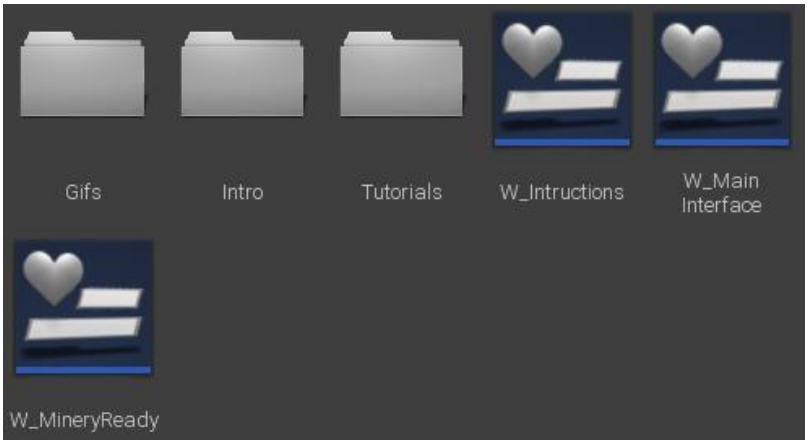


Figura 28. Contenido de la carpeta HUD (Elaboración propia)

- **Carpeta Maps**

Guarda todos los mapas del juego, principalmente estábamos trabajando con un mapa llamado Main, que era un mapa que estábamos probando que cosas podíamos hacer, y luego hicimos un nuevo diseño de mapa llamado Main_New_Design en donde implementamos el demo, además guarda información del build de luces (objetos terminados en Build_Data).

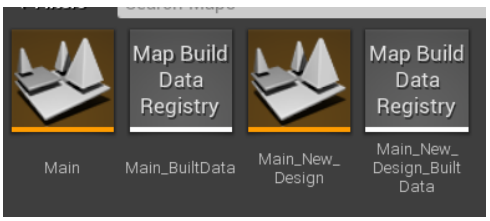


Figura 29. Contenido de la carpeta Maps (Elaboración propia)

- **Carpeta Megascans**

Set de rocas sacados de la librería de Megascan, que nos sirve para ayudar a ambientar los escenarios, existen varios tipos ya que queremos hacer varios biomas representados en cada planetoide, para este trabajo se usaron rocas y materiales más desérticos.

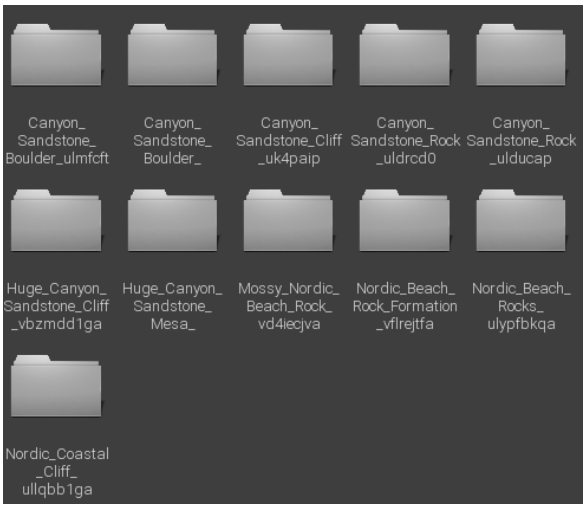


Figura 30. Contenido de la carpeta Megascans (Elaboración propia)

- **Carpeta MiniMap**

En la carpeta MiniMap se encuentra todo lo relacionado al minimapa del juego, si bien se logró implementar de forma preliminar, como se mencionó antes por una decisión de diseño se decidió sacarlo ya que la vista aérea era suficiente para orientar al jugador

Figura 31. Contenido de la carpeta MiniMap (Elaboración propia)

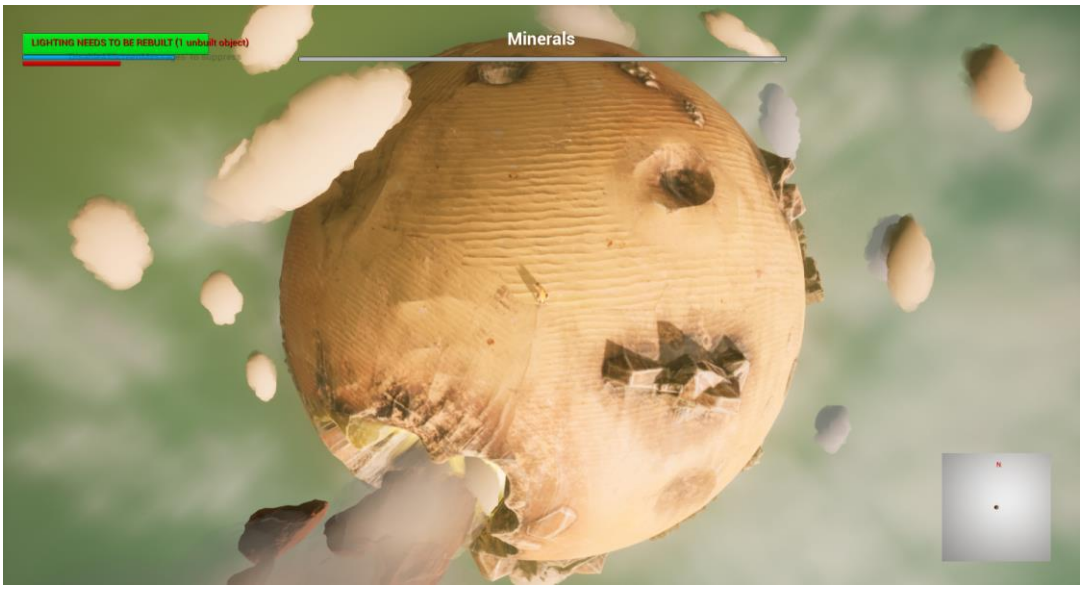


Figura 32. Comparación mini mapa v/s vista aérea (Elaboración propia)

- **Carpeta MSPresets**

Texturas sacadas del store de Epic ocupadas para materiales ambientales

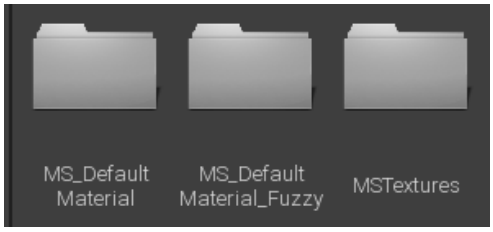


Figura 33. Contenido de la carpeta MSpresets (Elaboración propia)

- **Carpeta PointCloudsMorphing**

Unreal Engine viene con un plugin para la creación de nubes de forma simple, este lo hemos ocupado en nuestro juego y esta carpeta se crea automáticamente al instalar el ya mencionado plugin.

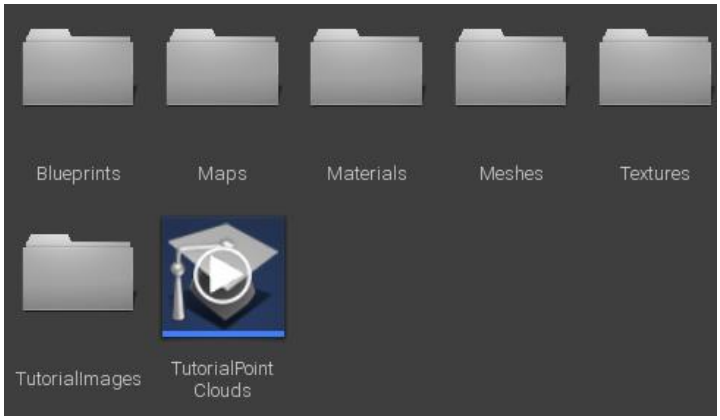


Figura 34. Contenido de la carpeta PointCloudsMorphing (Elaboración propia)

Se muestra una visualización general de las nubes

Figura 35. Vista general de nubes (Elaboración propia)

- **Carpeta Realistic_Starter_VFX_Pack_Niagara**

Paquete inicial de efectos especiales con Niágara, usados para el disparo del personaje

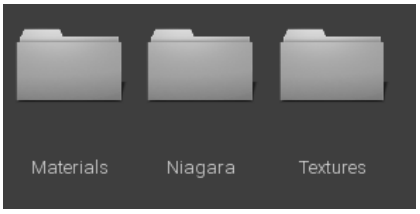


Figura 36. Contenido de la carpeta Realistic_Starter_VFX_Pack_Niagara (Elaboración propia)

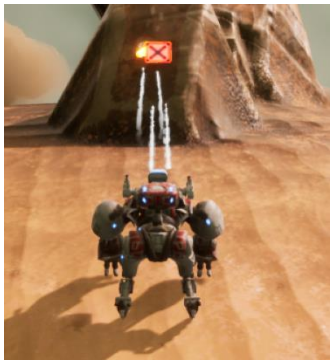


Figura 37. Ejemplo de partículas de misiles del MECHA (Elaboración propia)

- **Carpeta Sequences**

Carpeta donde se guardan todas las cinemáticas del juego

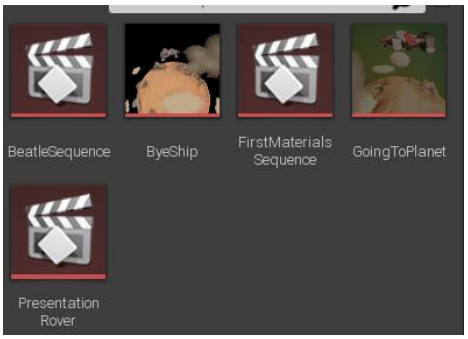


Figura 38. Contenido de la carpeta Sequences (Elaboración propia)

- **Carpeta Textures_Planet**

Carpeta donde se guardan todos los materiales finalmente ocupados en el juego

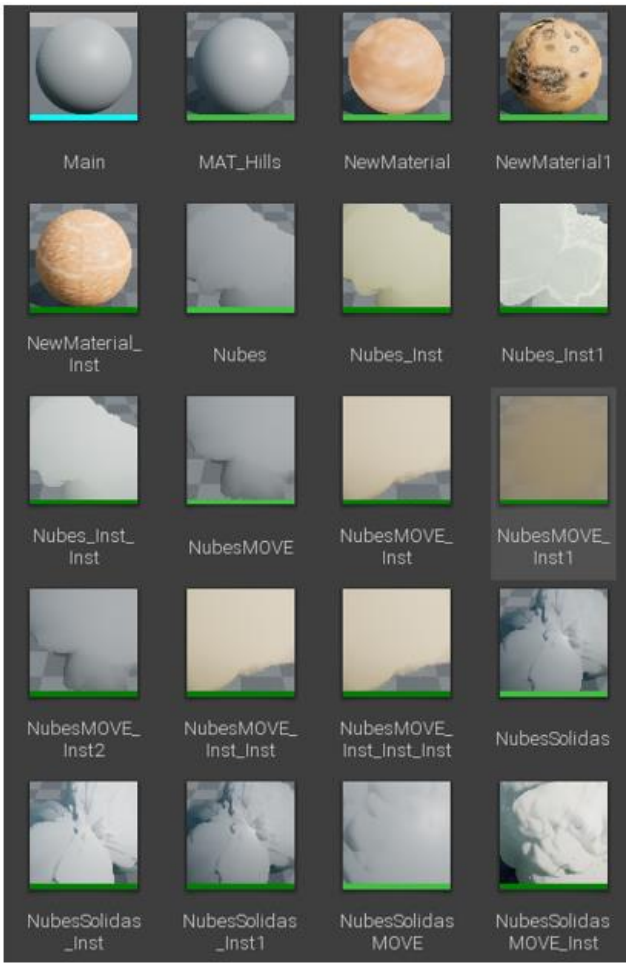


Figura 39. Contenido de la carpeta Textures_Planet (Elaboración propia)

- **Carpeta Weapons**

En un principio se estructuró el juego para que tuviera varias armas que estuvieran por separado y que pueda atacharse al personaje, sin embargo, esto quedo de lado, ya que la funcionalidad de poder escoger armas no quedó en el demo final.

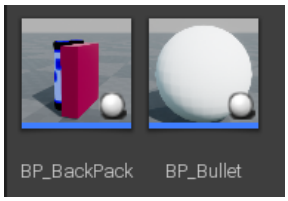


Figura 40. Contenido de la carpeta Weapons (Elaboración propia)

4.3. GAMEPLAY FRAMEWORK CLASSES

Las clases de juego básicas incluyen funciones para representar jugadores, aliados y enemigos, así como para controlar estos avatares con la entrada del jugador o la lógica de la IA. También hay clases para crear pantallas de visualización y cámaras para jugadores. Finalmente, las clases de juego como GameMode, GameState y PlayerState establecen las reglas del juego y hacen un seguimiento del progreso del juego y de los jugadores. (*Unreal Architecture | Unreal Engine Documentation*, n.d.)

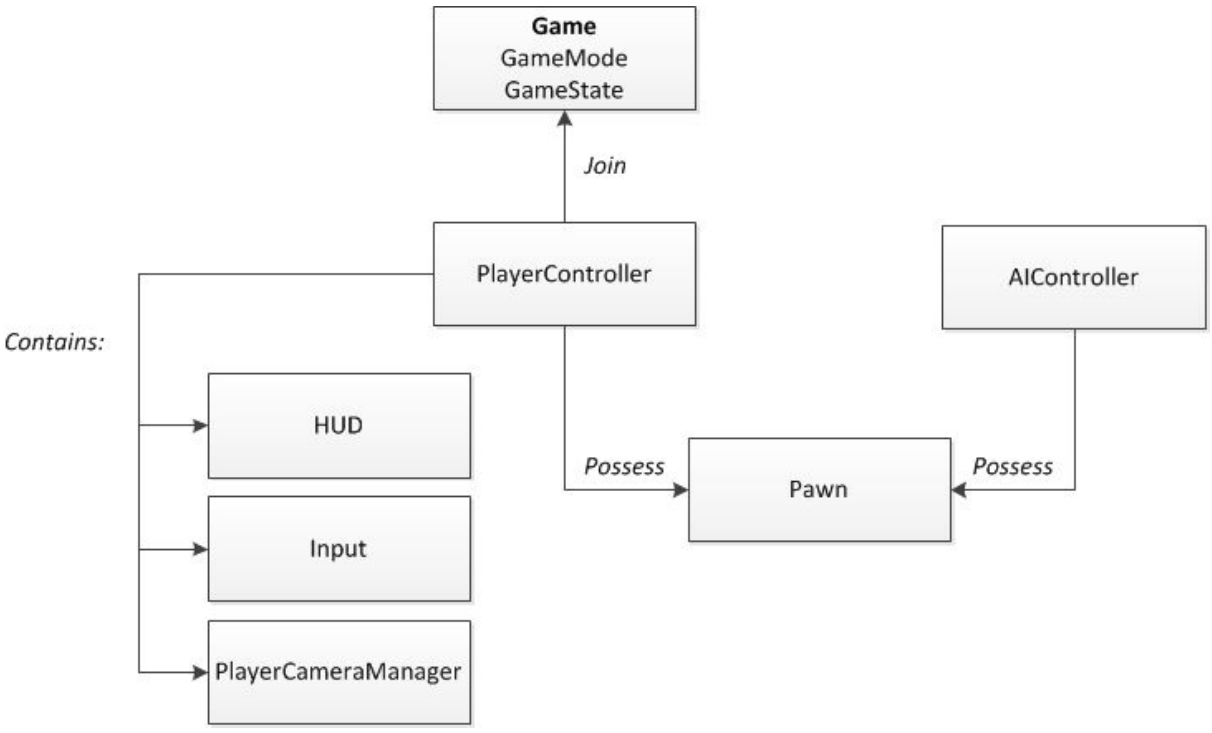


Figura 41. Framework Class Relationships (Unreal Architecture | Unreal Engine Documentation, n.d.)

4.4. IMPLEMENTACIÓN

4.4.1. Implementación de movimientos básicos del MECHA

El movimiento básico del MECHA viene implementado desde el padre de la clase BP_Player, llamada BP_HumanGravityCharacter, este movimiento permite mover al personaje en las 4 direcciones, adelante, atrás, derecha e izquierda con las letras WASD y mover la vista con el mouse, usado además para apuntar con la mira.

A continuación, se muestra el código de los movimientos:

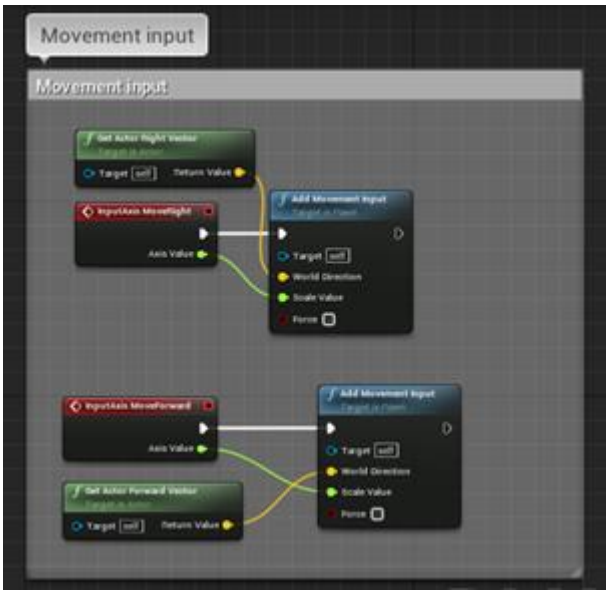


Figura 42 Inputs de movimiento (Elaboración propia)

Como se puede observar se tiene un input axis derecho (InputAxis MoveRight) que añade movimiento dependiendo del valor del vector derecha (Get Actor Right Vector) y se lo añade al add movement Input en la dirección correspondiente. Lo mismo pasa con el input axis hacia adelante (InputAxis MoveForward) que obtiene el vector dirección hacia delante de GetActor Forward Vector y se lo añade al Add Movement Input para moverlo hacia adelante o hacia atrás.

Estos valores en los axis están seteados de forma manual en el BeginPlay por el plugin de la siguiente manera:

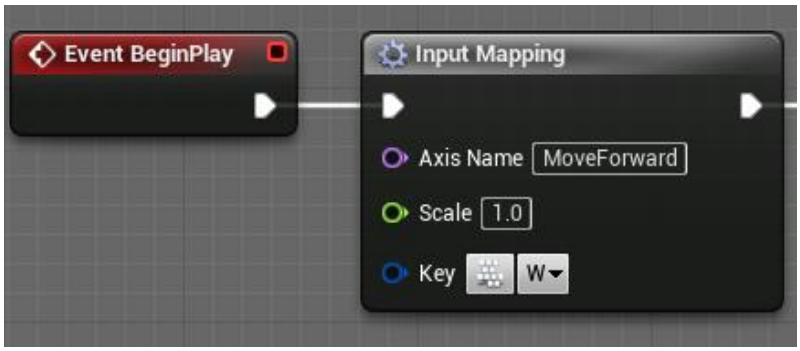


Figura 43. Configuración manual de Inputs de movimiento (Elaboración propia)

Esta Macro se ejecuta la primera vez que el personaje está en escena, y está compuesto por3 inputs que son:

- **Axis Name,** que como el nombre lo dice es el nombre del axis que se ocupará para gatillar el evento
- **Scale,** representa la escala del movimiento, este valor puede estar entre –1 y 1.
- **Key,** representa a la tecla que se usará para mover el input.

Luego de esto, podemos ver cómo se comporta el Macro:

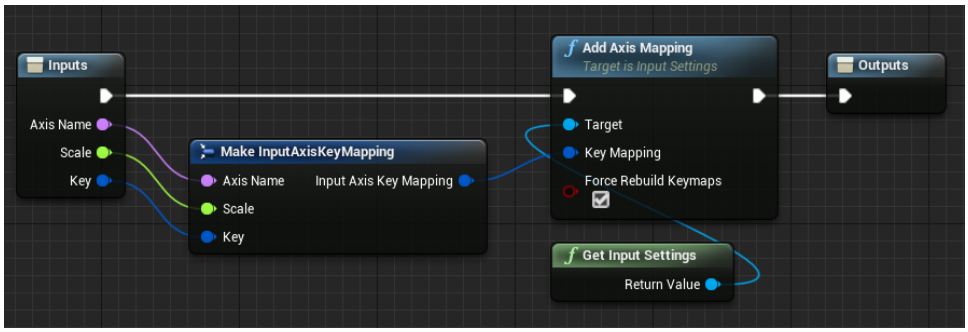


Figura 44. Código del Macro del input (Elaboración propia)

Con estos parámetros, se crea un inputAxisKeyMapping, y se le adhiere a al Axis mapping con la función "Add Axis Mapping" obtenida desde los inputs Settings, finalmente esta información queda grabada en el archivo DefaultInput.ini, localizada en la carpeta Config del proyecto

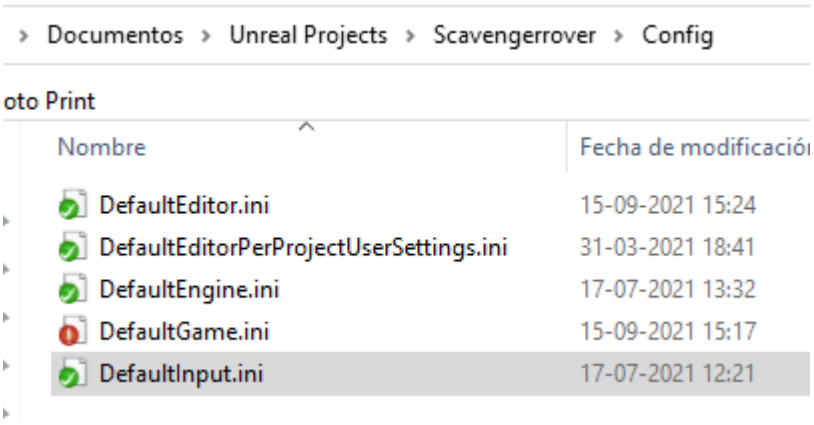


Figura 45. Archivo donde guarda los inputs (Elaboración propia)

El resto de los inputs corresponde al movimiento en los 4 axis:

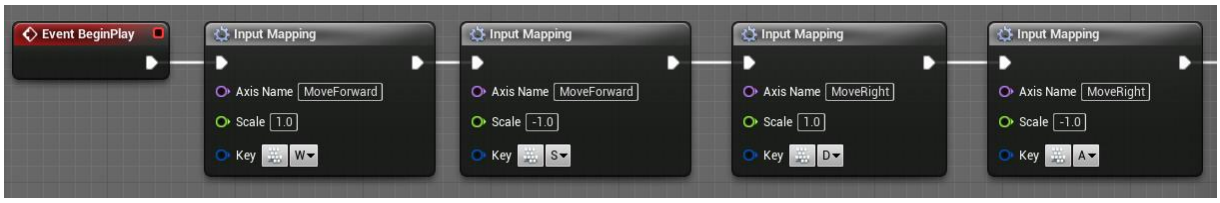


Figura 46. Configuración de inputs por código (Elaboración propia)

De los cuales usa el MoveForward de la siguiente forma: si avanza con W es positivo, si retrocede es negativo, y usa el MoveRight de la siguiente forma: Si va a la derecha es positivo y a la izquierda si es negativo.

Por otra parte, el movimiento de la rotación de la cámara y el apuntado, va a depender de la rotación del personaje, ya que al ser un juego donde la gravedad va hacia el centro del planetoide tendrá una rotación constante respecto a las coordenadas del mundo, lo cual lo hace más complejo, es por lo que hace un cálculo cada tick, como se muestra a continuación:

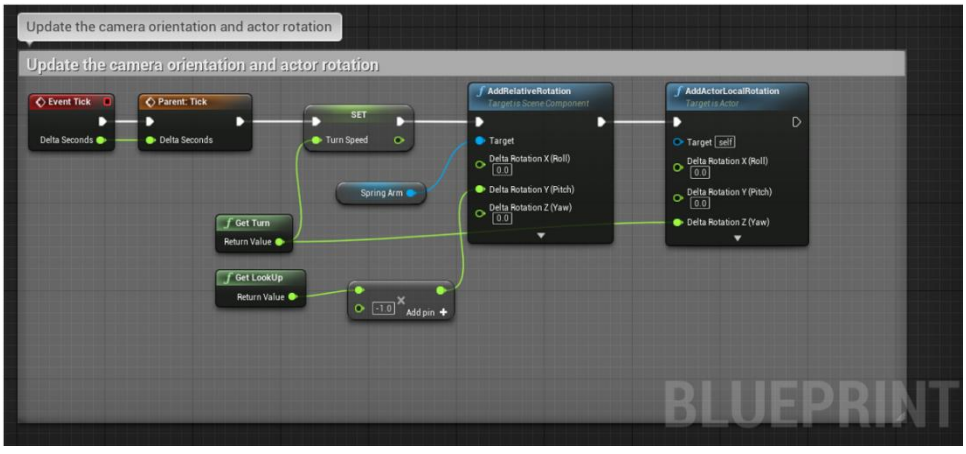


Figura 47. Rotación de cámara (Elaboración propia)

El sistema de inputs es similar al de los axis, está definido de forma manual como se muestra a continuación:

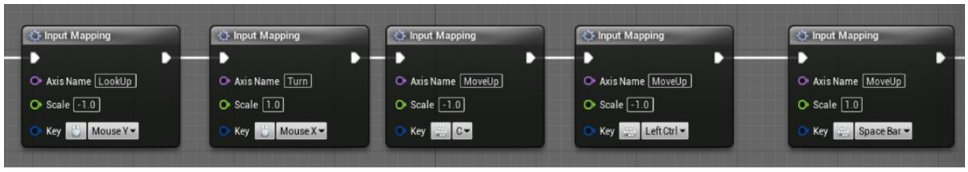


Figura 48. Configuración manual de Inputs de cámara (Elaboración propia)

4.4.2. Implementación del sistema de disparo de mecha

El sistema de disparos es implementado con la idea de que el personaje principal pueda cambiar de armas de forma simple en un futuro, con un sistema de compras u obtención de estas con monedas internas del juego. Es por eso que las armas no están en el personaje, sino que son atachadas al momento de la creación del personaje, en el Begin Play, como se ve a continuación:

Figura 49. Adjunta un arma al MECHA (Elaboración propia)

Lo que hace el evento "Attach Gun", es crear un Spawneador de balas, que es atachado a un socket, y seteado en una variable llamada Gun1 que nos servirá para trabajar con ella al momento de disparar, como el personaje tiene dos armas que disparan una en cada lado del MECHA, este proceso se hace dos veces, esta vez con la variable Gun2.

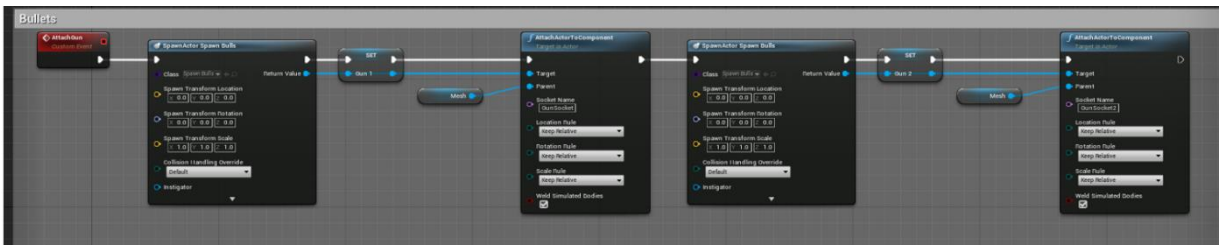


Figura 50. *Creación de los spawner de las balas (Elaboración propia)*

Dicho esto, el spawneador de balas está listo para crear las balas en el momento del disparo, para esto vemos que en el blueprint llamado SpawnBulls crea con un "For Each Loop", todas las balas creadas por el sistema de partículas de Niagara:

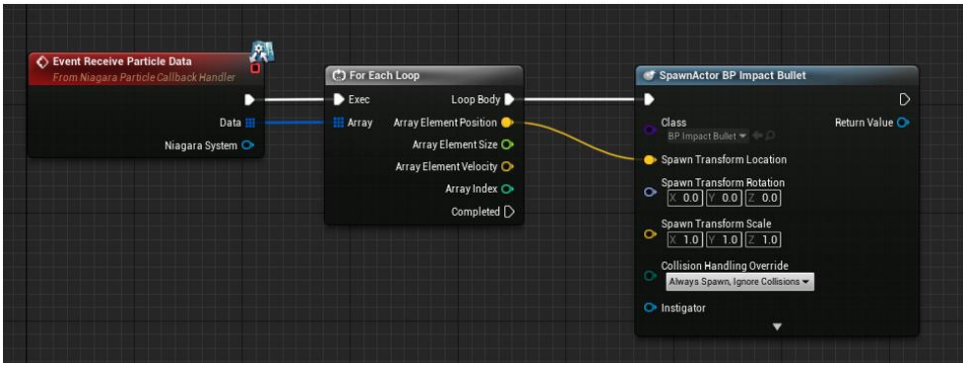


Figura 51. Creación de las balas (Elaboración propia)

Este sistema de partículas se comunica mediante una Interfaz que crea el evento y se los pasa a "Data" para su posterior creación.

Figura 52. Interfaz de comunicación con Niagara (Elaboración propia)

Toda la lógica de comportamiento de la partícula está en las configuraciones en Niagara, y estas nos dan el evento que nos indica cuando una partícula colisiona con un objeto, solo tomamos esta información y la utilizamos.

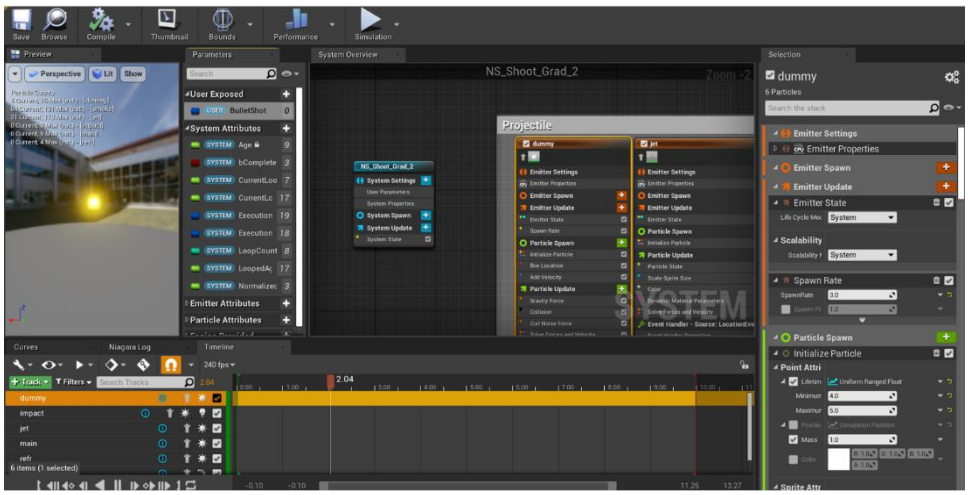


Figura 53. Sistema de partículas de disparo con Niagara (Elaboración propia)

Luego, solo queda llamar al Input de disparo para que active las partículas, estas se ven en la siguiente figura, que se activa el evento Fire

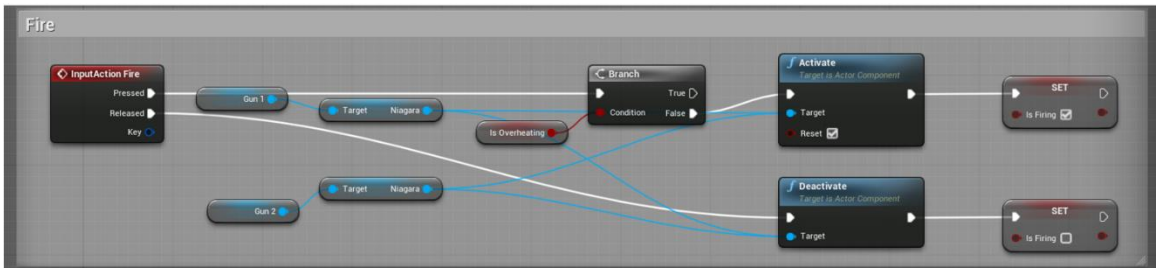


Figura 54. Disparo del arma (Elaboración propia)

Finalmente tenemos una función llamada UpdateGun Everheating que pregunta periódicamente si se está disparando para hacer controlar la barra de calentamiento si se calienta por completo el jugador no podrá disparar en unos segundos.

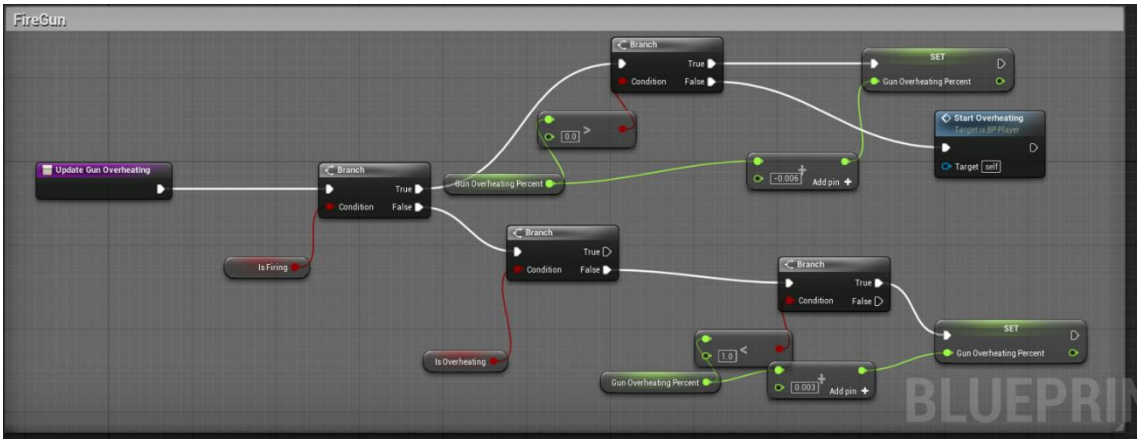


Figura 55. Lógica de sobrecalentamiento de las armas (Elaboración propia)

4.4.3. Implementación de ia para movimiento de enemigos

Los enemigos están diseñados para tener comportamientos simples, como se puede apreciar a continuación el enemigo puede ir a un target, con un sistema de movimiento similar al usado por el player, ya que en este caso al tener un sistema de gravedad en un planetoide no se puede ocupar la IA de Unreal:

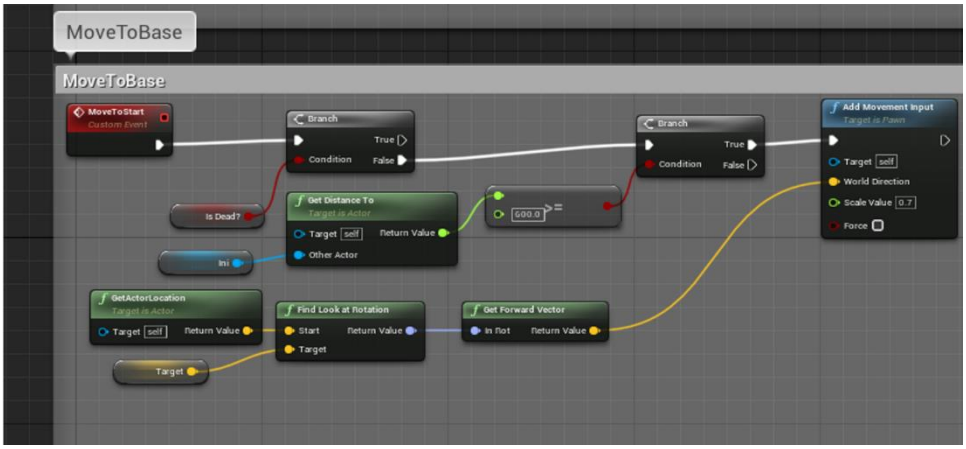


Figura 56. Lógica del movimiento de los enemigos hacia su base (Elaboración propia)

Atacar al player, una vez que lo detecta va inmediatamente y si lo toca lo ataca.

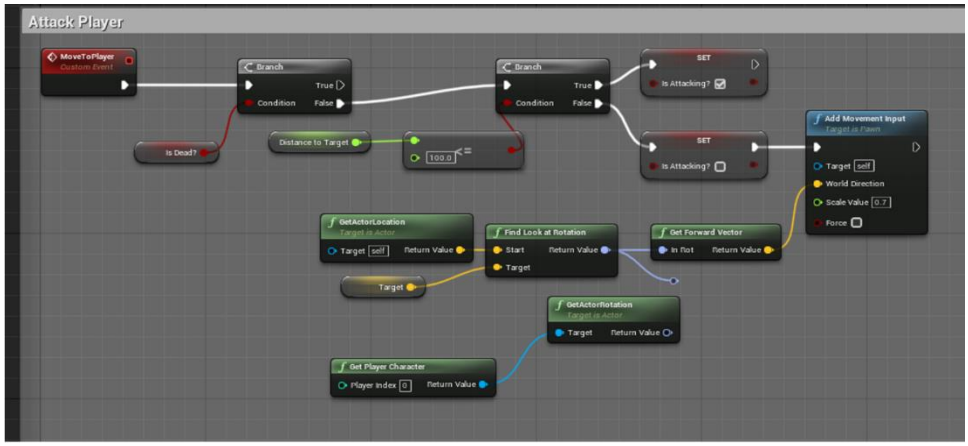


Figura 57. Lógica del ataque de los enemigos al MECHA (Elaboración propia)

También puede ser dañado por el MECHA, en este caso se disminuirá una barra de energía que tienen en su cabeza

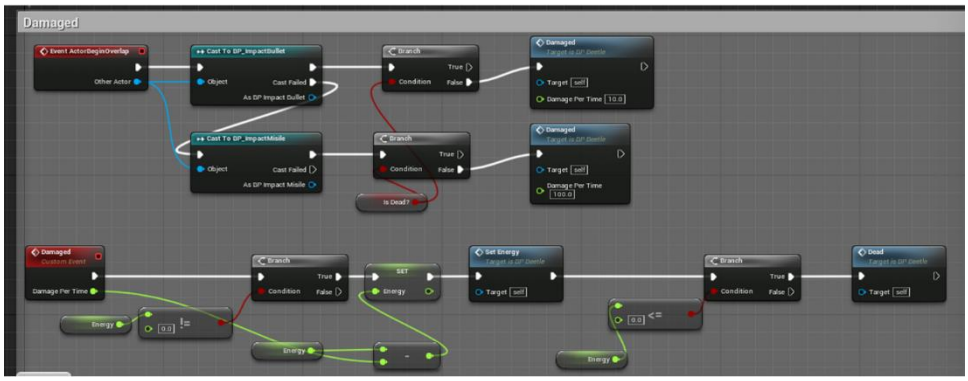


Figura 58. Lógica de los enemigos al recibir daño (Elaboración propia)

Si muere, tiene una animación de muerte y desaparece después de unos segundos.

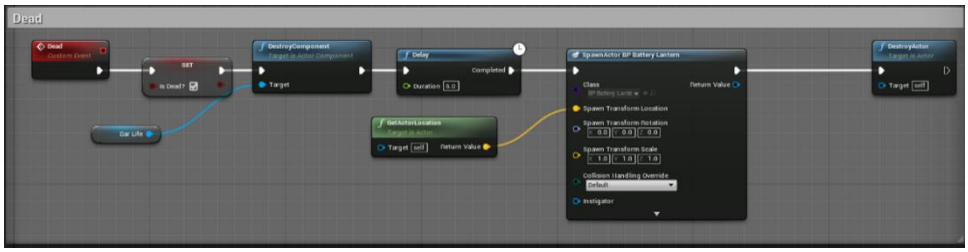


Figura 59. Lógica de los enemigos al morir (Elaboración propia)

Finalmente se ve una representación del sistema de control de las animaciones:

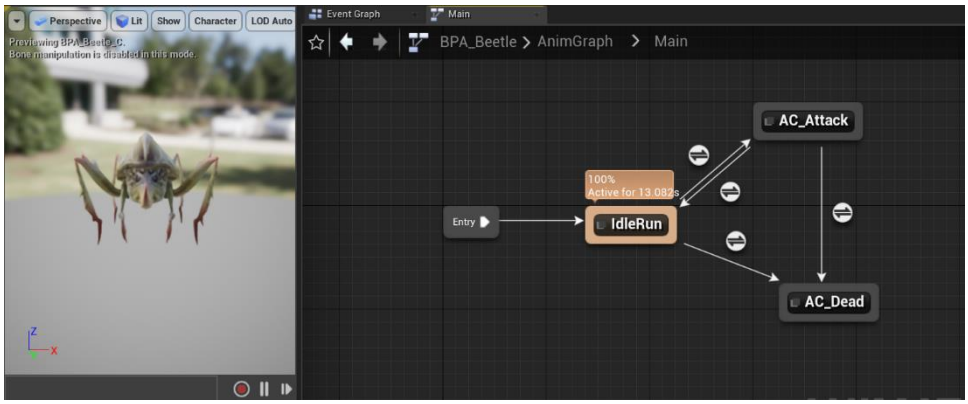


Figura 60. Lógica de las animaciones del enemigo Beetle (Elaboración propia)

4.4.4. Implementación de sistema de orientación (vista aérea)

El jugador tiene 2 cámaras, la primera es la que viene por defecto que se ve detrás del jugador, y la otra cámara llamada CameraFarMap, es una cámara que nos permita tener una visión de la nave que deja a nuestro MECHA, como se muestra a continuación

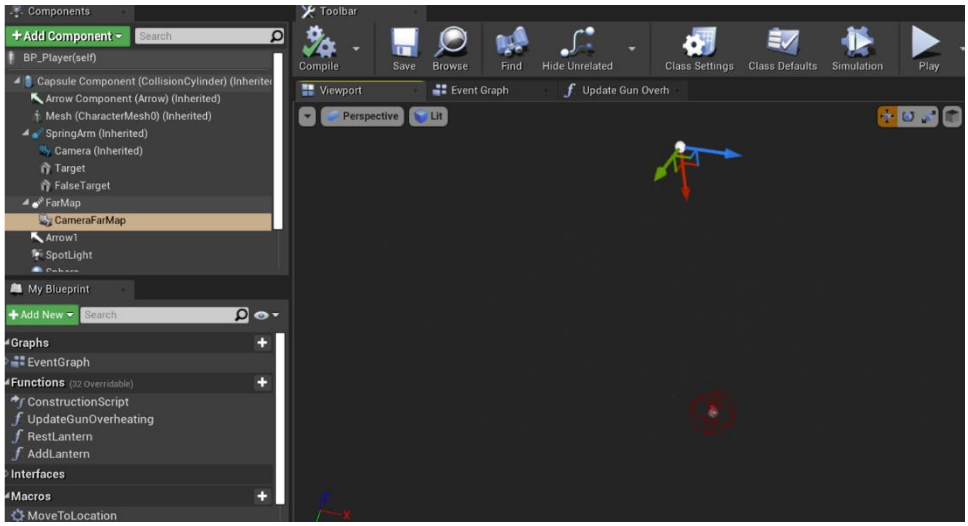


Figura 61. Visualización de la cámara aérea desde el Viewport (Elaboración propia)

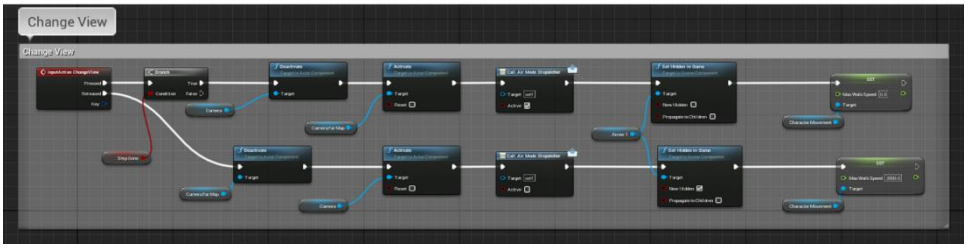


Figura 62. Lógica del cambio de cámara (Elaboración propia)

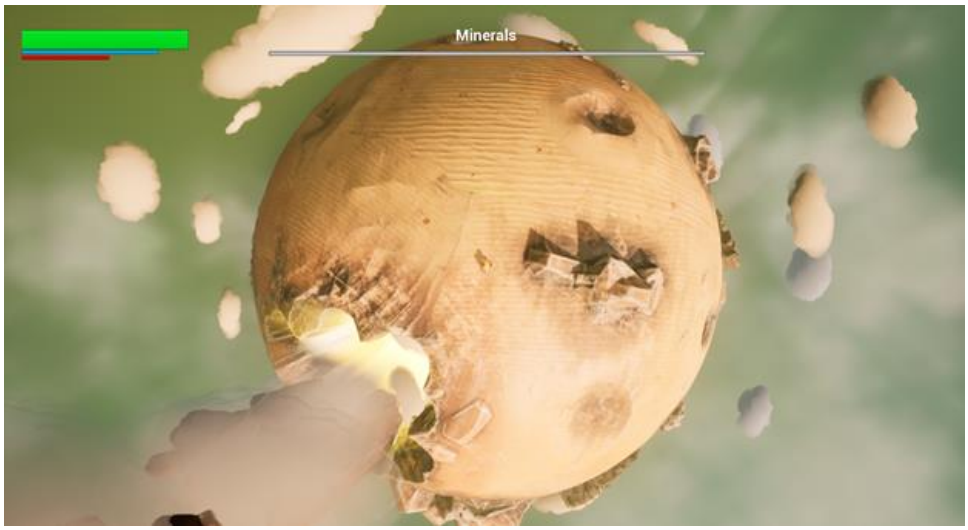


Figura 63. Visualización de la implementación de la cámara aérea (Elaboración propia)

4.4.5. Implementación de linterna para buscar minerales

La implementación de la linterna consiste en crear un radio visible alrededor del MECHA en el cual pueda buscar minerales, estos minerales están escondidos a simple vista, y el contacto con la linterna hará que estos se vean. La limitante que tiene es que al ser una linterna tiene batería que se va descargando al usarla, por lo que el jugador deberá ser consciente que no puede tenerla prendida todo el tiempo. Sin embargo, la energía que necesita es una energía orgánica que se obtiene al derrotar a los enemigos. El código se muestra en la siguiente imagen:

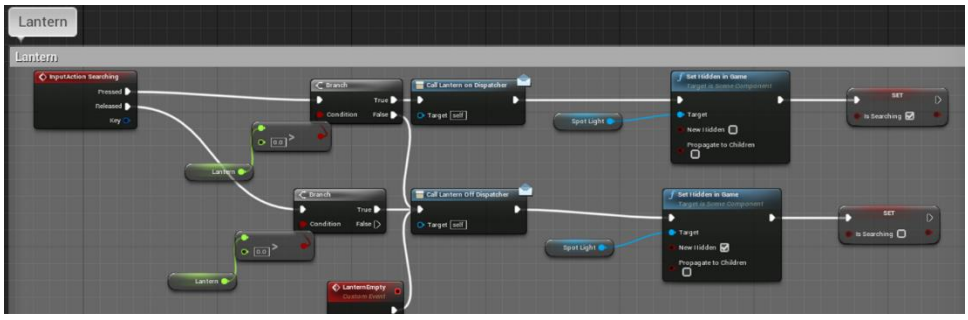


Figura 64. Lógica de la búsqueda de minerales con la linterna (Elaboración propia)

A continuación, podemos ver la construcción de la luz desde el viewport

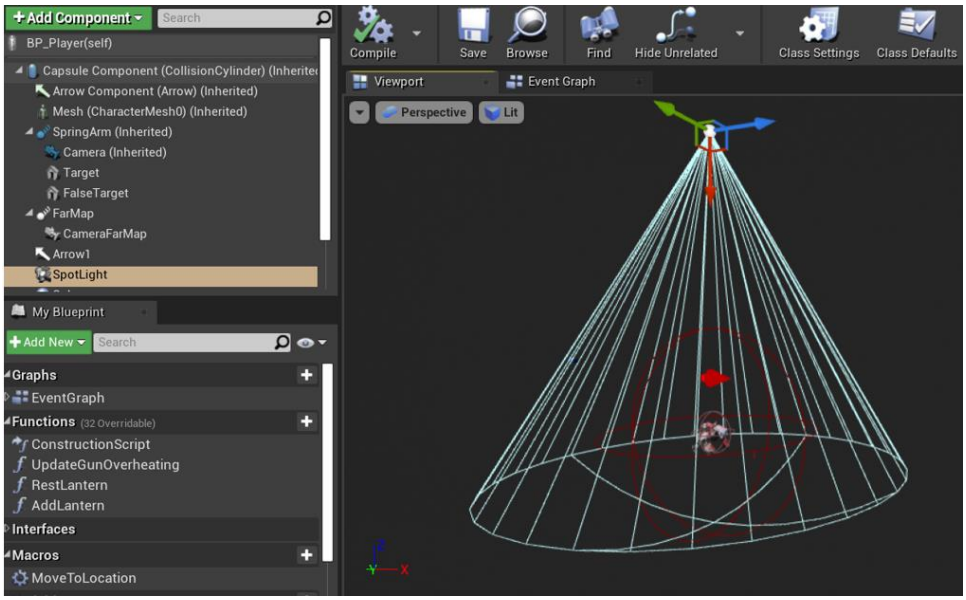


Figura 65. Visualización de la linterna en el Viewport (Elaboración propia)

Y finalmente como queda implementado en el juego

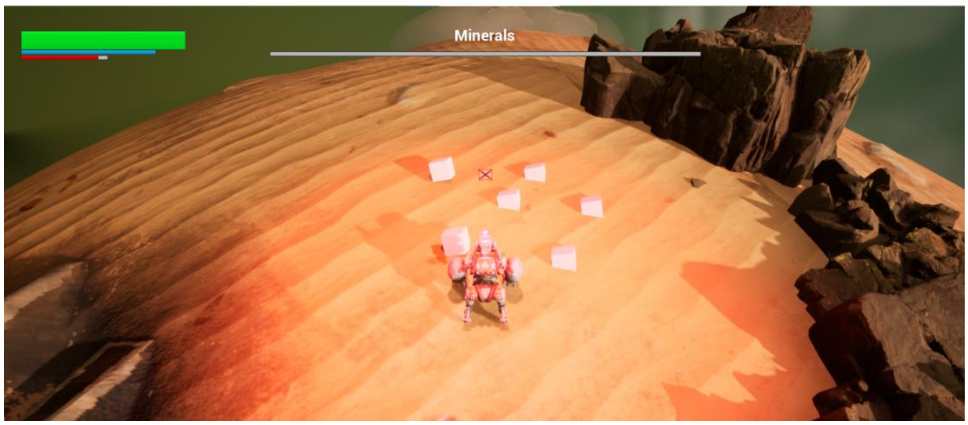


Figura 66. Ejemplo de la implementación de la linterna en el juego (Elaboración propia)

4.4.6. Implementación de minerales para ser visto por la linterna

Los minerales están escondidos por todo el escenario, sin embargo, al tener contacto con la luz del MECHA aparecen, a continuación, la lógica y el código implementado.

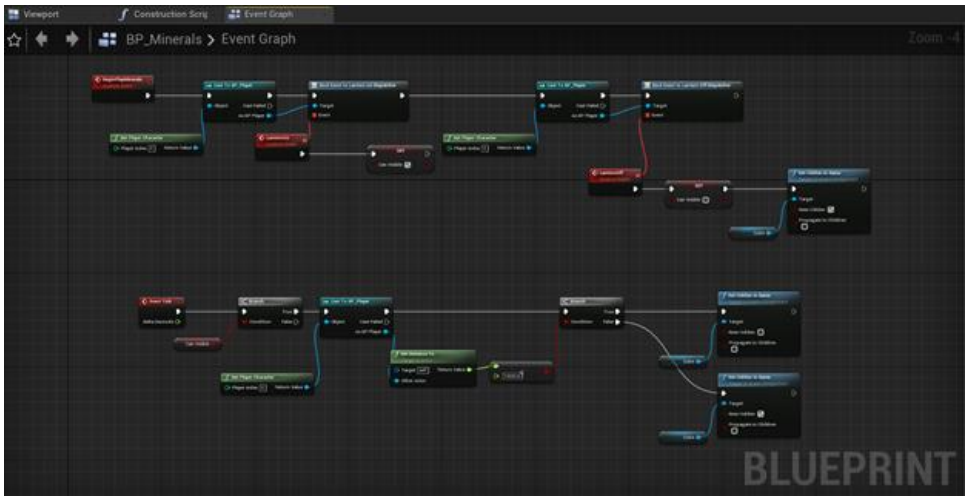


Figura 67. Lógica de cómo se muestran o esconden los minerales (Elaboración propia)

4.4.7. De creación de tiendas mineras para su extracción

La implementación del campamento minero es una semiesfera que se crea en un lugar del planeta en el cual se comienza a minar, una vez terminado, todos los minerales que están entre esa esfera se extraen y se aumenta la barra de minerales como se muestra a continuación:

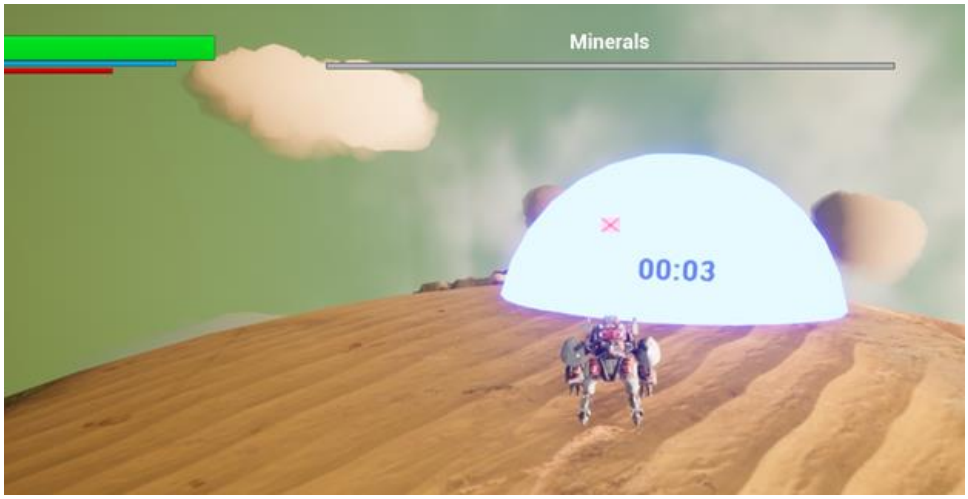


Figura 68. Implementación de las bases mineras (Elaboración propia)

El código de esta mecánica está representado en el siguiente Blueprint:

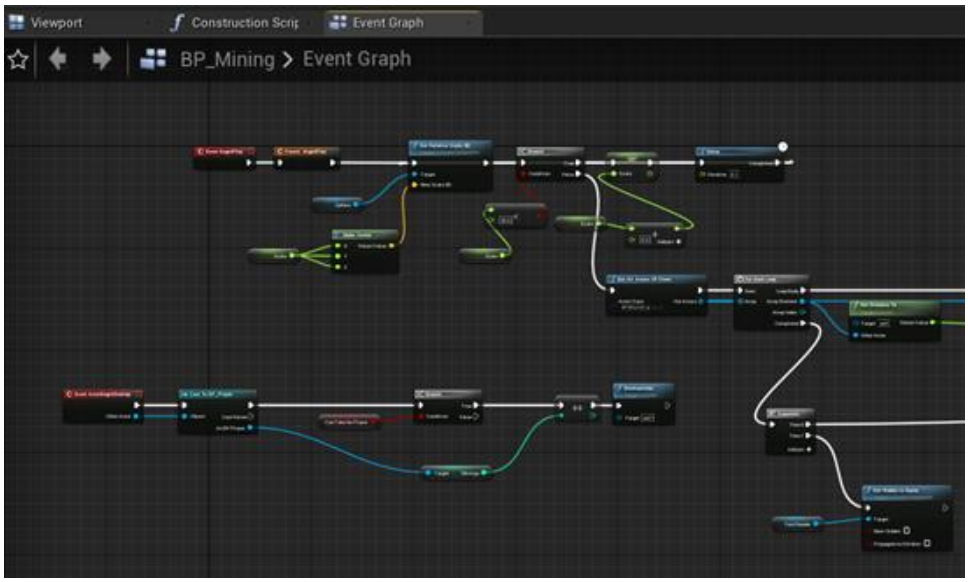


Figura 69. Lógica de la extracción de minerales parte 1 (Elaboración propia)

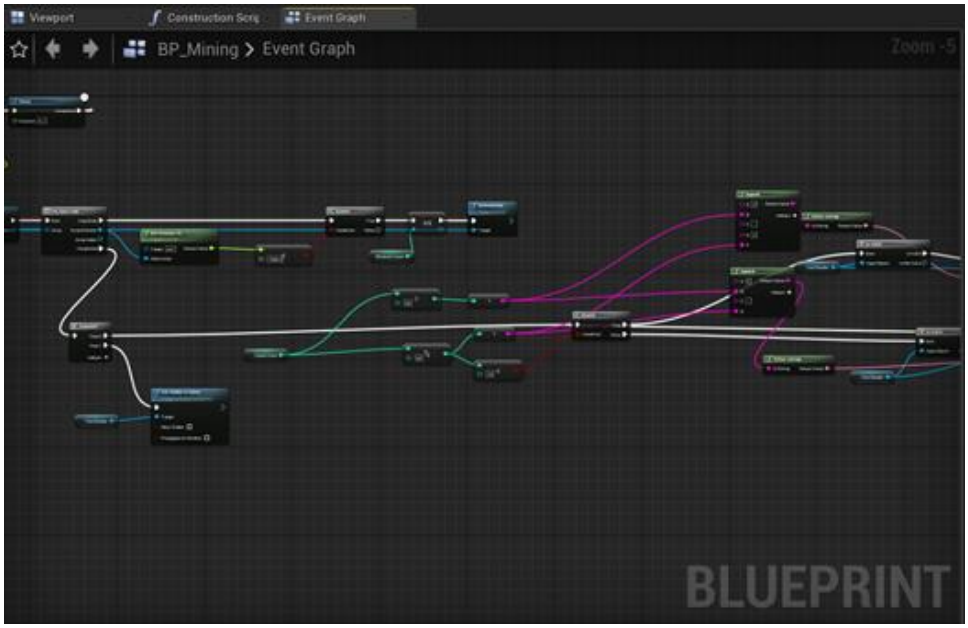


Figura 70. Lógica de la extracción de minerales parte 2 (Elaboración propia)

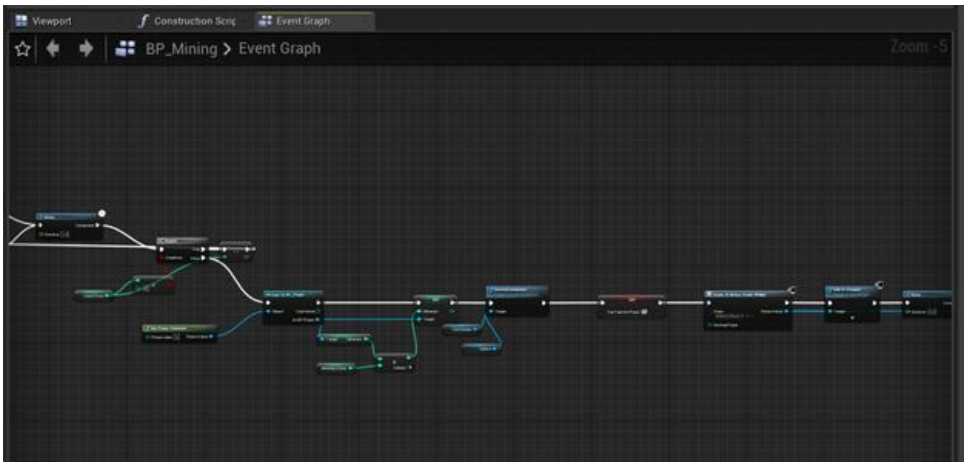


Figura 71. Lógica de la extracción de minerales parte 3 (Elaboración propia)

En el cual se implementa no solo la lógica de la minería sino además el reloj está programado a mano, este reloj indica cuanto tiempo le queda para que esté minando.

4.4.8. Implementación de diálogos

Se ha creado un sistema de diálogos, del cual está compuesto por un Blueprint tipo Widget en el que siempre están los dos personajes y un texto vacío, con el objetivo de que estos personajes se vayan visibilizando o escondiendo a medida que vayan hablando, además el texto en blanco nos permite estar constantemente cambiando el texto sin estar creando un Widget por cada uno, a continuación, se muestra el sistema gráfico y un ejemplo del código dentro del widget

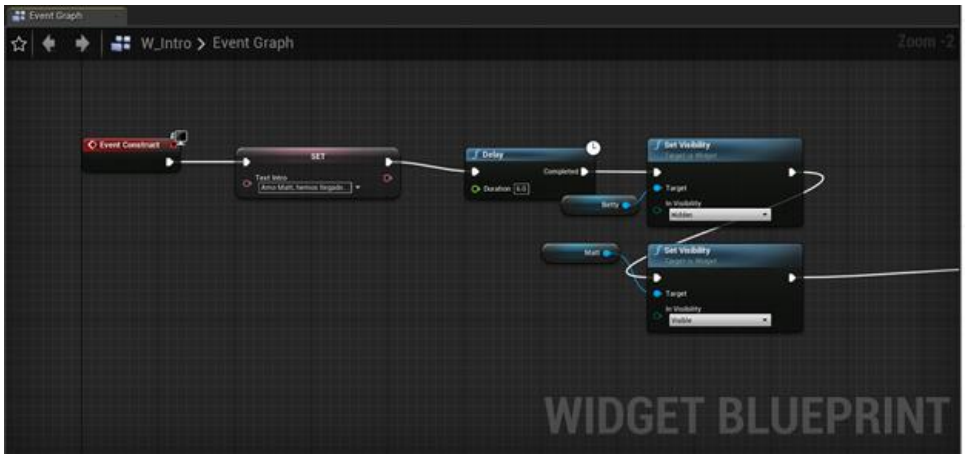


Figura 72. Implementación del widget de diálogos (Elaboración propia)

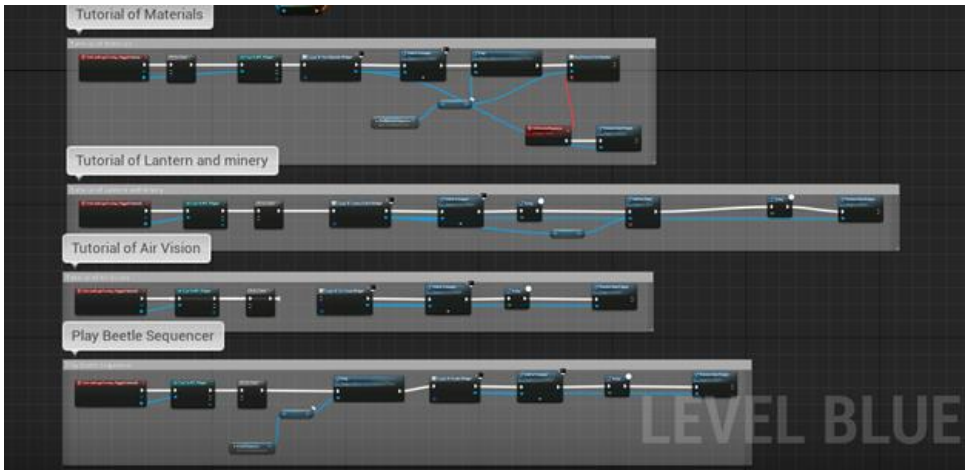


Figura 73. Lógica de ejemplo de creación de diálogos (Elaboración propia)

4.4.9. Creación de sistema de tutoriales: vista aérea, enemigos y extracción de minerales

Se ha creado un sistema de tutoriales, en el cual consiste en que cuando el personaje toque alguno de los triggers dentro del planeta se activará una cinemática con sus respectivos diálogos:

Figura 74. Lógica de los tutoriales (Elaboración propia)

5. GDD DE ALTO NIVEL

5.1. ALCANCE DEL JUEGO

El alcance del juego consta de 10 niveles, representados en exoplanetas con una misión principal y dos secundarias en cada uno. Los exoplanetas tienen diferentes Biomas y climas, aunque tienen enemigos básicos en común, cada planeta tendrá un único enemigo, hay exoplanetas desérticos, con vegetación, con hielo o nieve, con lava y rocas, y también varían de tamaño, a medida que el nivel aumenta los recursos minerales se vuelven más escasos, y los enemigos más abundantes.

En cuanto a las mecánicas serán siempre las mismas, es decir que no existe un sistema de progresión directa del personaje durante el juego. A medida que visitemos más planetas podremos adquirir mejoras para las armas, tanto en potencia como en tiempo de recarga, así mejoras para las capsulas minadoras.

El juego podrá vender contenido adicional (DLC) así como expansiones basadas en más planetas, enemigos, galaxias, armas y mejoras.

Finalmente, una segunda parte del juego según su éxito de ventas se consideraría invertir en escenas con diálogos hablados, traducción a diferentes idiomas, poder navegar en la nave por el espacio, e incluso poder manejar a nuestro personaje en la nave durante los viajes a la siguiente misión, y la posibilidad de migrar a la nueva versión del motor Unreal Engine 5, con el cual podremos trabajar en personajes con cargas poligonales mucho más detallas gracias a sus nuevas opciones como la tecnología “Nanite virtualized geometry” y también a su nuevo sistema de iluminación “Lumen” podremos simular ray tracing haciendo que la calidad grafica del producto sea superior.

5.2. DISEÑO VISUAL

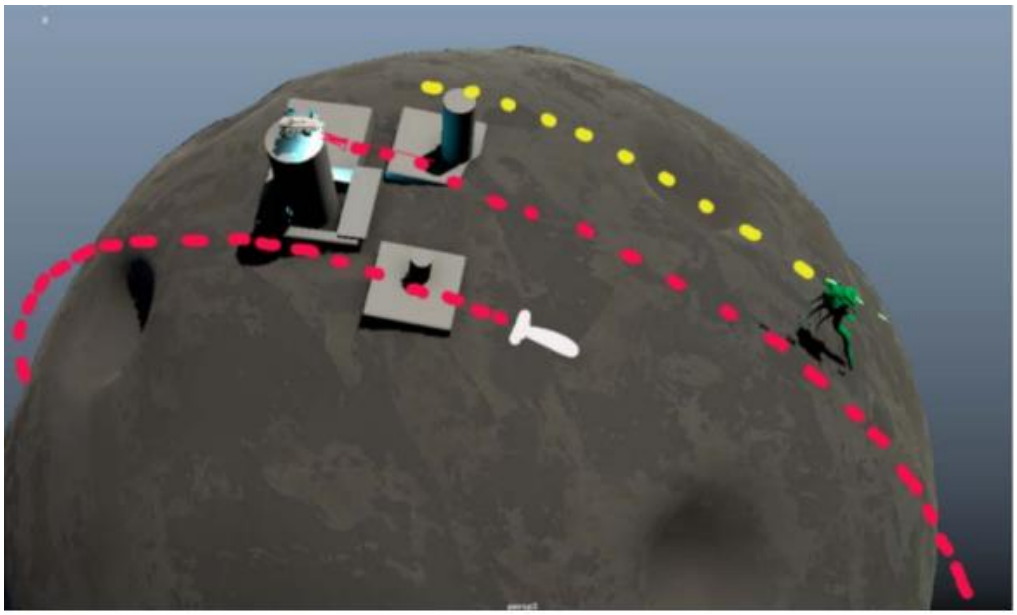


Figura 75. Conceptos iniciales del juego. (Elaboración propia)

Scavenger Rover nace de nuestra primera lluvia de ideas, llegamos a la conclusión de hacer un videojuego de ciencia ficción, naves, robots y algo que determinaba la idea era el mapa en pequeños planetoides y sobre eso plantear mecánicas en las que haya extracción de recursos, diálogos y looping, en un inicio el juego era demasiado ambicioso, teníamos un sistema de planetas en el cual se movería nuestra nave, visitamos esos planetas en busca de recursos, llegaríamos a estaciones en las cuales podríamos tener diálogos y obtener más misiones

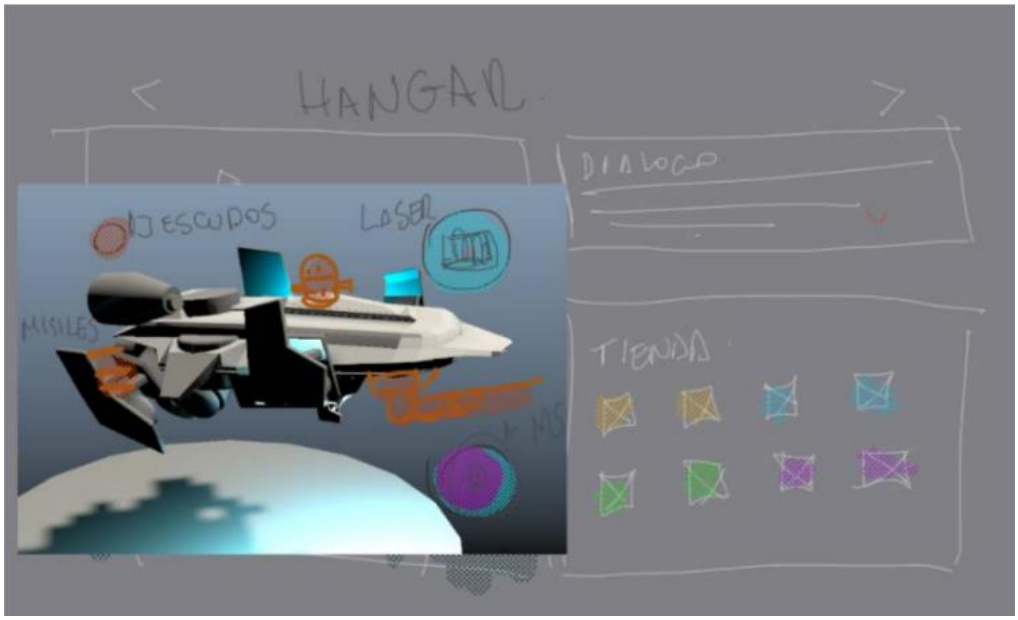


Figura 76. Propuestas iniciales del juego. (Elaboración propia)

En un segundo punto se propuso controlar la nave para viajar de planeta en planeta y después para explorar los planetas se usarían un robot terrestre que se encargaría de enfrentarse a enemigos, pero como un personaje secundario, finalmente decidimos que el viaje será usado en cinemáticas, para mostrar el objetivo y los diálogos, nos centraremos solo en la mecánica del robot en el planetoide.

A nivel de arte conceptual y diseño de niveles se trabajó en los primeros moodboards en los que se intentaba definir los primeros diseños de naves, de enemigos y de ambientes aquí algunos ejemplos de lo que serían los primeros prototipos: aunque después se descartaron algunos y se excluyeron otros se debía escoger el material que se usaría para el vertical slice

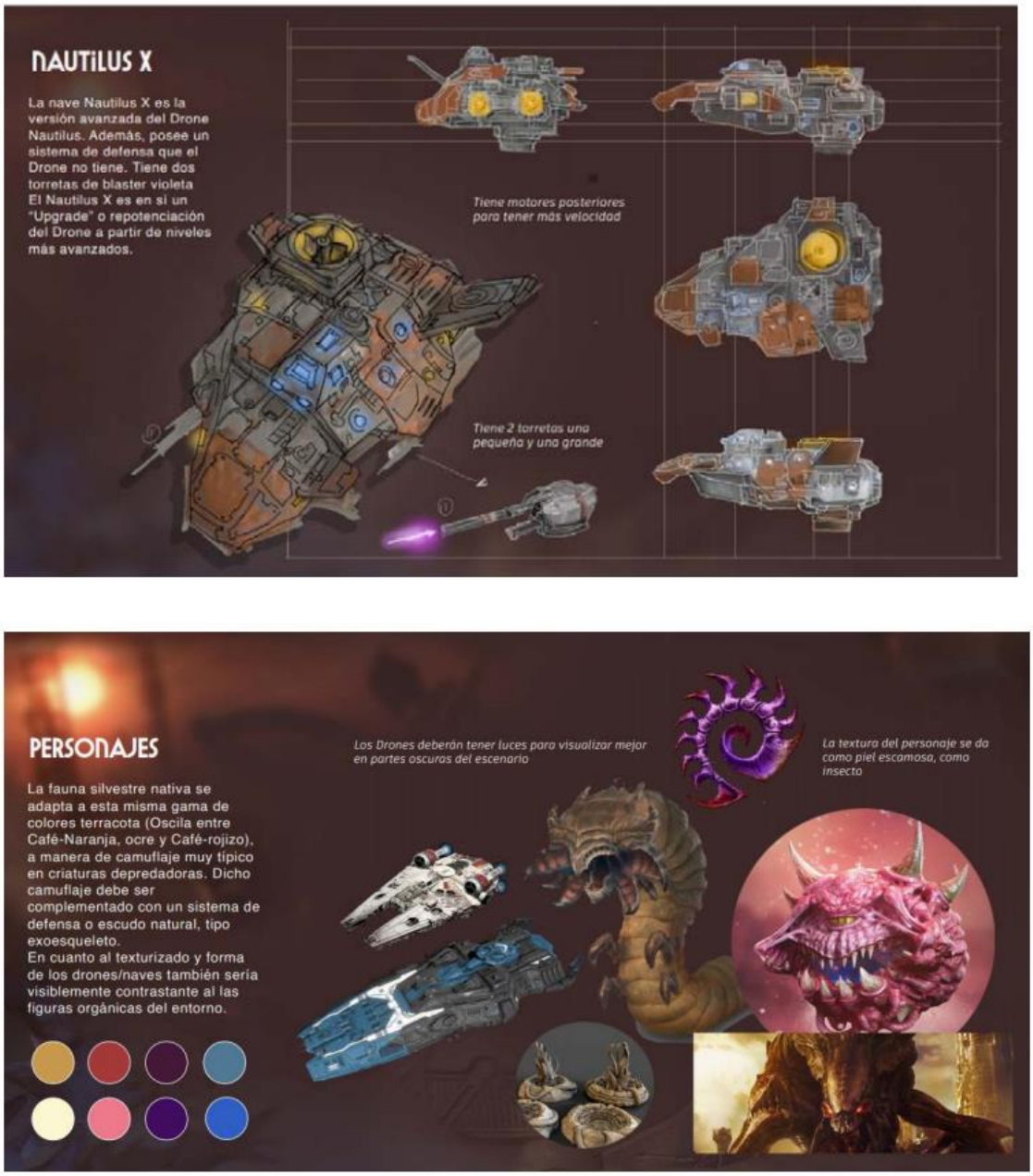


Figura 77. Moodboards y propuestas de arte. (Elaboración propia)

Después de algunas reuniones definimos el alcance del vertical slice, nos centraremos solo en un planeta, un enemigo y la nave principal quedaría solo para los viajes planetarios y pasamos directamente al diseño de mecanicas en el control del robot terrestre, tenía que ser un robot que sea todo terreno, muy fuerte que pueda dirigir este tipo de expediciones y enfrentarse a cualquier amenaza y al mismo tiempo administrar los recursos de la minería, así nace la idea del robot Kongbot, un meca cuadrúpedo similar a un gorila gigante y que será asistido por cápsulas espaciales que desciende al planeta desde la nave que se encargaran de la extracción de recursos.

Figura 78. Diseño, texturas y materiales de Kong-Bot. (Elaboración propia)

Finalmente llegamos a el diseño final de Scavenger Rover, es un juego basado en una historia postapocalíptica, por tanto, todos los componentes se definen con tecnología futurista. Su personaje principal, un robot (Mecha) llamado K2CP o mejor conocido como KONG BOT. La maquinaria usada para minar, cápsulas Minadoras.

El planetoide XDESRt de tipo desértico, aunque lleno de recursos, está infestado por estas criaturas.

Los enemigos que anidan este planetoide, alienígenas de tipo insecto llamados: ”The Krongs”.

Todo el arte conceptual, la creación 3D de modelados y texturas, se realizan desde cero para el videojuego, con excepción de assets y materiales del planeta, se emplean recursos escaneados de la biblioteca de Megascans.

Figura 79. KONG BOT. (Elaboración propia)

5.3. GAMEPLAY Y MECÁNICAS

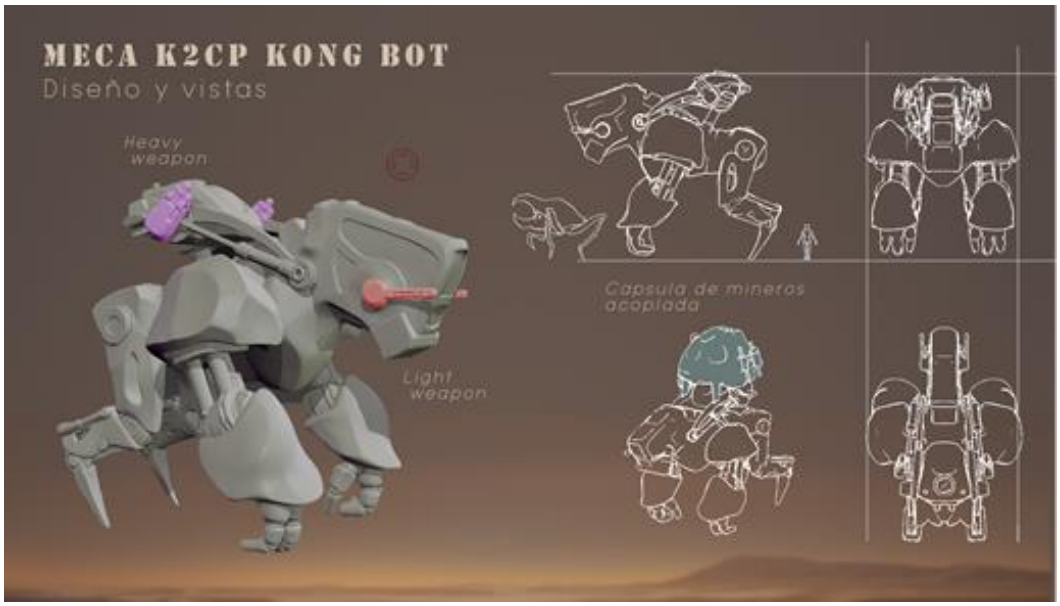


Figura 80. KONG BOT. Vista isométrica (Elaboración propia)

- Caminar/Correr en cualquier dirección: El personaje por su movimiento camina siempre viendo hacia adelante, se desplazará hacia los lados

- Saltar, su propia altura: aunque es un bot bastante pesado en movimiento puede saltar para salir de irregularidades del terreno
- Disparar ráfagas de cohetes: Ráfagas de pequeños misiles que no son muy potentes, pero son rápidos y con mucha precisión
- Disparar artillería pesada de tipo mortero: en la plataforma de la espalda tiene cañones de artillería pesada, son misiles de alto potencia, pero poca precisión
- Escanear yacimientos de minerales: puede escanear el terreno mientras se mueve
- Minar materiales de los yacimientos: despliega la estación minera la cual es autónoma, el personaje puede seguir explorando

5.4. CONTROL

- Correr con las letras WASD
- Caminar presionando la tecla Shift
- Mostrar mapa con la tecla Tab
- Mover la cámara con el Mouse
- Disparar balas con botón izquierdo del mouse
- Disparar misiles con botón derecho del mouse
- Saltar con la tecla espacio
- Escanear minas con la tecla R
- Botón de acción con la tecla E
- Acercar la cámara con el Scroll del Mouse

5.5. CÁMARA

- Cámara en tercera persona cenital
- Plano general
- Personaje siempre visible de cuerpo entero
- Se puede acercar y alejar la cámara

5.6. PERSONAJES

Para esta sección detallaremos al personaje que complementa la narrativa, el personaje "Matthews Gathely" capitán de Kong Bot, el personaje principal, Betty, la inteligencia articial

de la nave como personaje secundario, el MECHA cuadrúpedo K2CP (KONG BOT) y las Cápsulas de Minado XLM.

5.6.1. El capitán "Matthews Gathely"

Para nuestro personaje principal buscamos un perfil de un veterano de guerra, solitario, con experiencia en tratar y asimilar experiencias bélicas y hacer arriesgados trabajos, aunque su tiempo de gloria ya paso tiene que ganarse la vida, así viaja por la galaxia haciendo dinero minando en estos planetas infestados de peligrosos alienígenas, su compañero de trabajo es su robot Kong-Bot, su compañero, un robot único que él ha mejorado desde sus tiempos de guerra

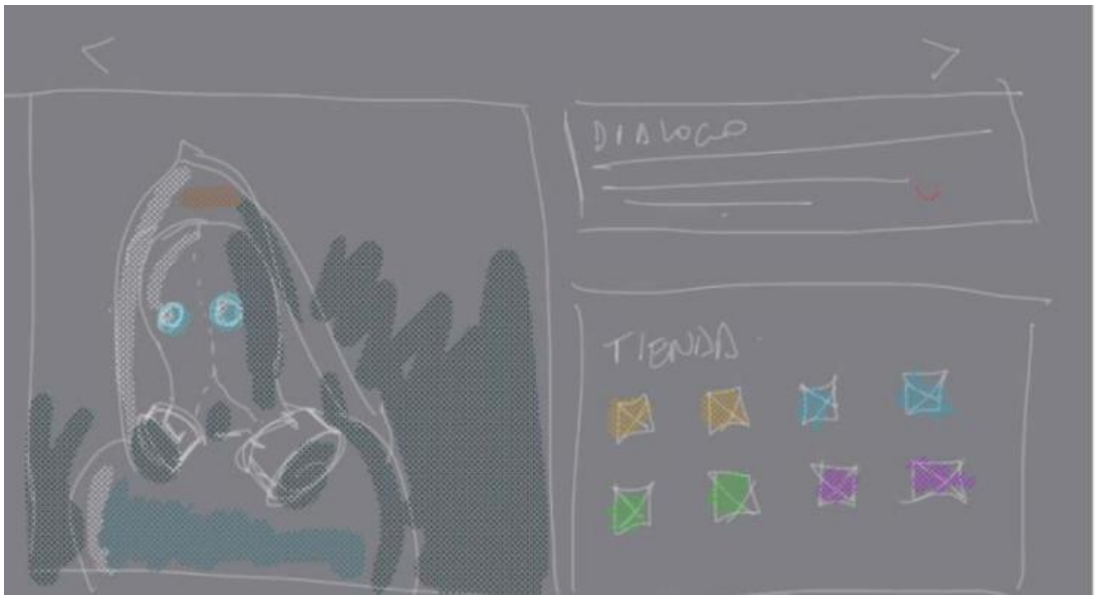


Figura 81. Primer boceto del capitán Gathely. (Elaboración propia)

Conocido como Matt, al mando de Kong Bot, dirige toda la operación de extracción de minerales. Aunque no es un personaje jugable directamente, su rol es apoyar la narrativa. A nivel visual, se buscó un estilo futurista cyber-punk para el atuendo, algo relacionado con su pasado militar y a nivel de textura que se vea bastante gastado por el uso y por el clima expuesto en estos planetas

Figura 82. Matt. (Elaboración propia)

Se elaboro un primer modelo con escultura digital que fue posteriormente compuesto digitalmente con fotografías para conseguir el primer boceto, finalmente y una vez aprobado el diseño por todo el equipo se creó un modelo que estuviera optimizado para poder funcionar en Unreal Engine 4, tanto su carga poligonal como el peso de las texturas, esto se repetirá para todos los personajes.

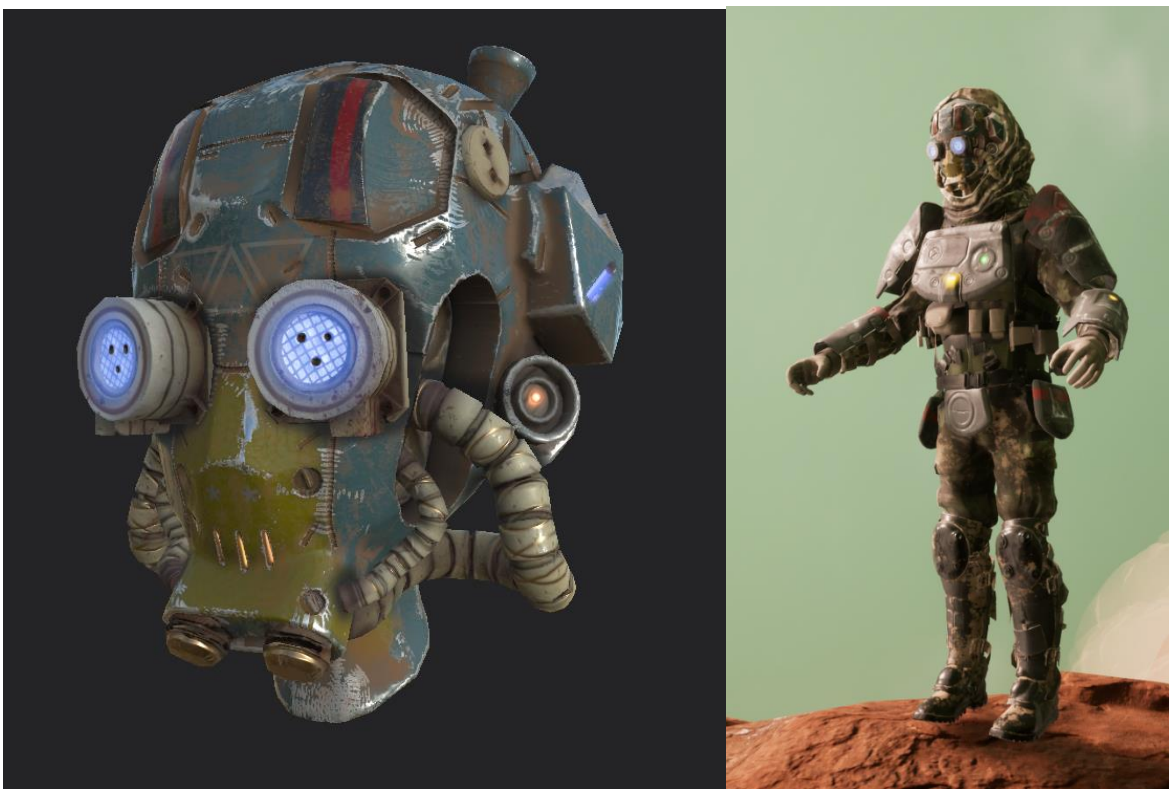


Figura 83. Diseño terminado de Matt exportado en UE4 (Elaboración propia)

5.6.2. Inteligencia artificial Betty

Betty es un personaje secundario, es una Inteligencia artificial almacenada en un antiguo androide incompleto físicamente, que está encargada de manejar y administrar los recursos de la nave, es la asistente y única compañía de Matt, sus algoritmos de pensamiento y razonamiento son muy avanzados, es un modelo único, muy cotizado en toda la galaxia, Matt la pudo obtener en una apuesta en el mercado negro, al inicio del juego se da a entender que su memoria fue formateada y reseteada, de esta manera podremos usar la mecánica de diálogos entre Matt y Betty para entender mucho sobre este mundo, el pasado y la historia en general, ya que Betty hace preguntas de todo tipo para entender y aprender cómo funciona el mundo

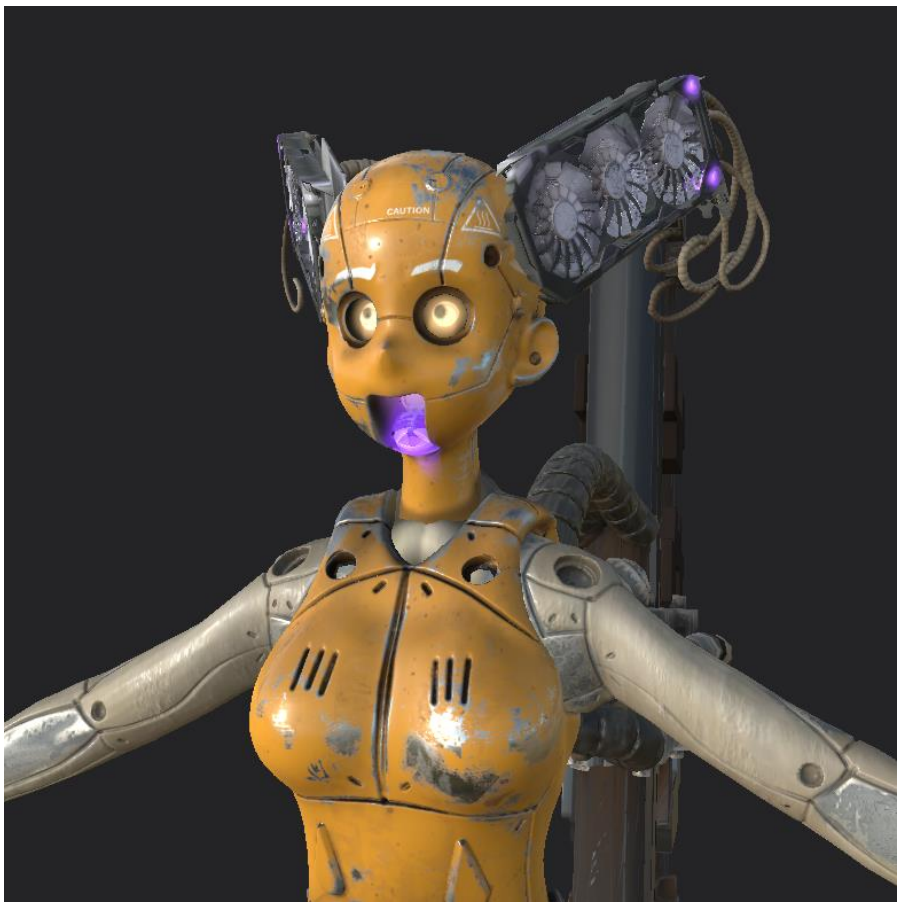


Figura 84. Betty. (Elaboración propia)

5.6.3. Cápsula de Minado XLM Tipo Taladro

Son NPCs que descienden al planeta desde la nave principal para minar las zonas donde Kong Bot ha puesto Faro de minado.

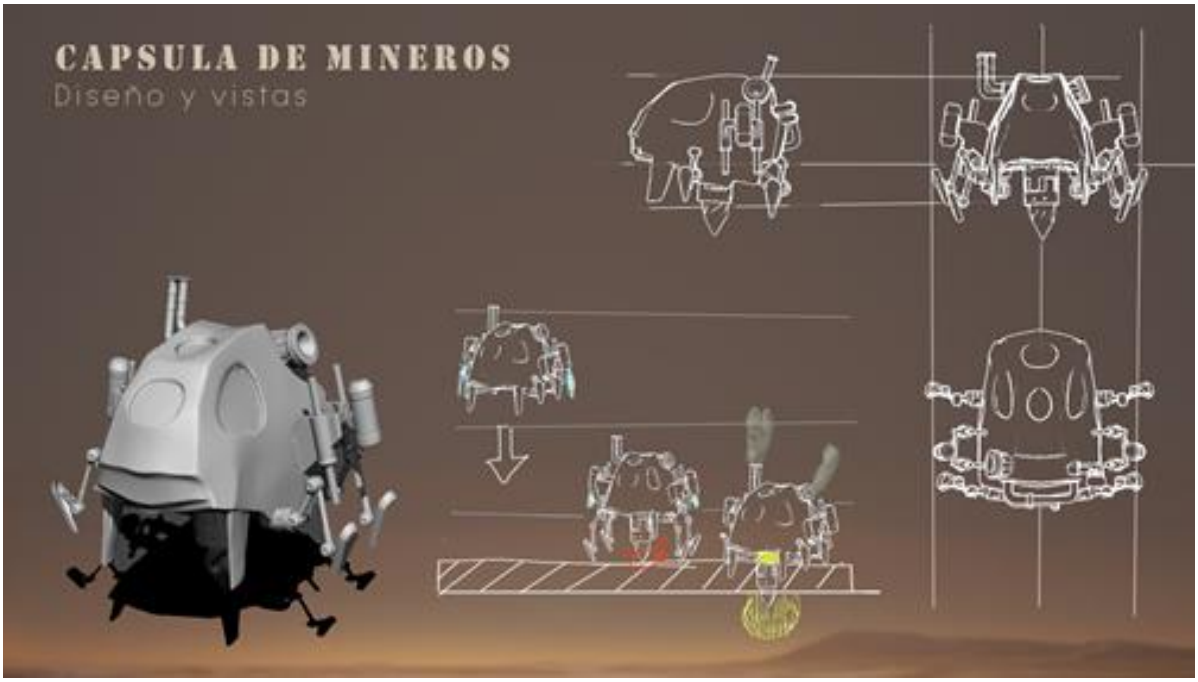


Figura 85. Cápsula de minado. (Elaboración propia)

5.6.3.1. Conceptos Básicos simplificados para el prototipo jugable

La Cápsula XLM necesita parámetros básicos: cantidad de vida, movimiento, velocidad, etc.

- Vida: Se representa de manera continua y pueden sufrir daño por parte de los enemigos, la cantidad disponible es de 25.
- Velocidad: Valor que define con qué velocidad extraen los minerales.

5.6.3.2. Comportamiento

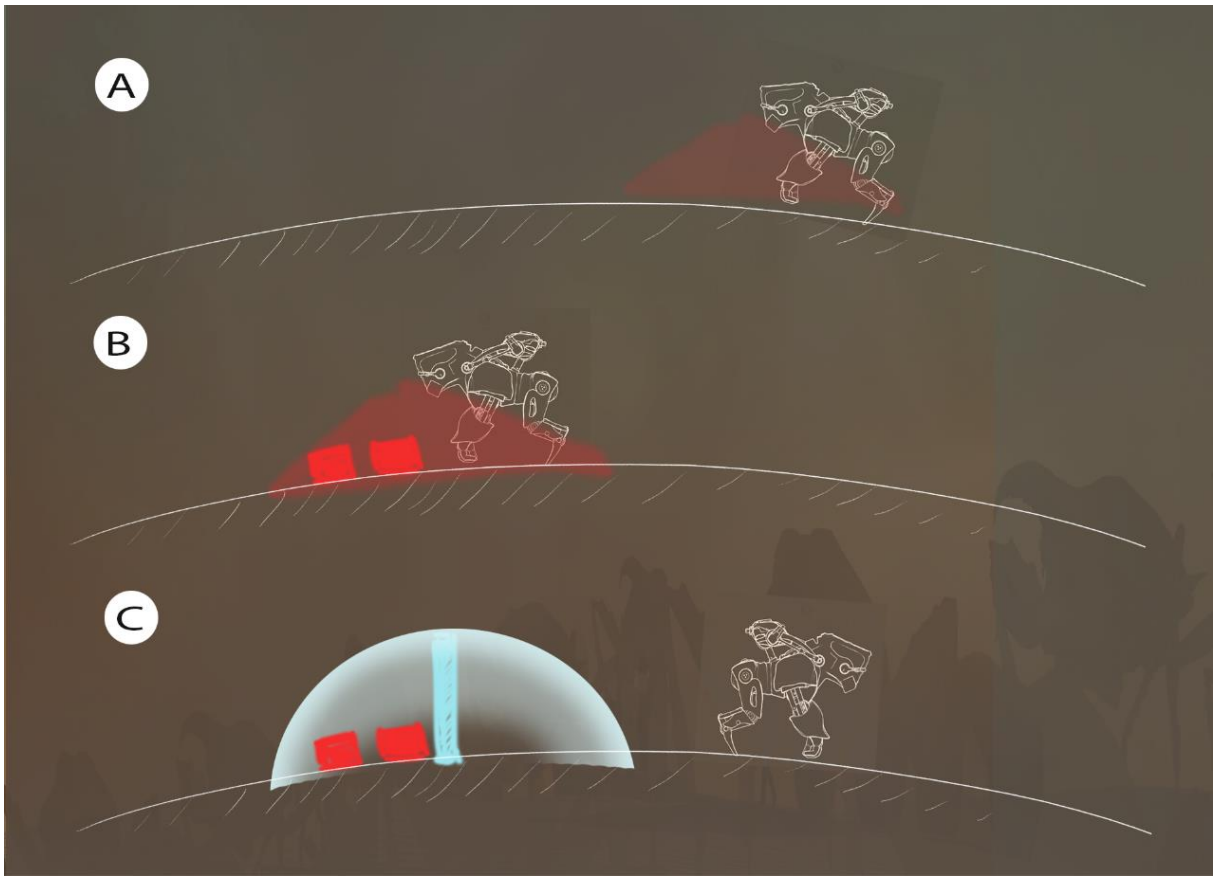


Figura 86. Proceso de minado. (Elaboración propia)

Para definir la mecánica de minado se simplifico gráficamente el proceso especialmente para el prototipo jugable, se dividió en 3 etapas:

- Fase A: El personaje Kong Bot busca y explora los recursos en el mapa con su radar representado con la luz roja en el juego, Kong bot como parte de su equipamiento, lleva un sensor que detecta minas al activar el scanner y apuntar sobre el territorio del nivel.
- Fase B: Los recursos invisibles se hacen visibles solo con el radar de luz, Kong Bot dispone de un muelle que despliega las cápsulas mineras una vez marcado el área de minado.
- Fase C: puedes mandar a despliega la mina representada por la columna azul y despliega un escudo mientras está minando, el jugador puede avanzar hacia otro punto, al terminar de terminar se envía un mensaje al jugador

5.6.4. El robot (Mecha) cuadrúpedo K2CP (KONG BOT)

Es el personaje principal, tiene aproximadamente 7 metros de altura, una capacidad para 6 tripulantes, 4 en la cabina principal y 2 en la cabina posterior.

Figura 87. KONG BOT. (Elaboración propia)

5.6.4.1. Conceptos Básicos

Kong Bot necesita parámetros básicos: cantidad de vida, movimiento y velocidad.

- Movimiento: Kong Bot puede moverse hacia adelante, hacia atrás, izquierda a derecha, para esto se usan los controles de movimiento.
- Apuntar: Kong Bot posee un punto de mira con el cual apunta sus armas a los enemigos, para esta funcionalidad se usa el control de puntería.
- Saltar: Kong Bot puede dar saltos para esquivar obstáculos o alcanzar zonas altas. Para su salto se aplica cierta cantidad de aceleración, para esto disponemos el control de salto.

Kong Bot también carga módulos mineros, municiones y armas.

5.6.4.2. Combate

La idea del combate es simple, disparar al enemigo con su arma principal o secundaria. Esquivar a los enemigos con los movimientos básicos. Infringir daño a los enemigos dependiendo del arma utilizada, recibir daño al ser atacado por un enemigo.

5.6.4.3. Armas

- Arma Principal: Esta arma dispara una ráfaga de balas por minuto, no hay límite para la cantidad de munición, y puede ser una ráfaga continua.
- Arma secundaria: Esta arma dispara rondas de morteros por segundo, esta arma necesita segundos de recarga para volver a ser utilizada, no existe límite de munición.

Figura 88. KONG BOT dispara. (Elaboración propia)

5.6.4.4. Vida

La vida de Kong Bot se representa de manera continua con valores de 0 a 100. Para que Kong Bot vaya recuperando su vida después de enfrentarse a los enemigos, debe consumir ítems de vida encontrados en el nivel o que son arrojados por los enemigos al ser derrotados.

- Item de vida en el nivel: Estos items de vida se ubican aleatoriamente en el mapa. Y restauran un 50% de vida de Kong Bot.
- Item de vida de enemigos: Estos items de vida aparecen cuando se eliminan una cantidad de enemigos. Restauran el 25% de vida de KONG BOT.

5.7. ENEMIGOS

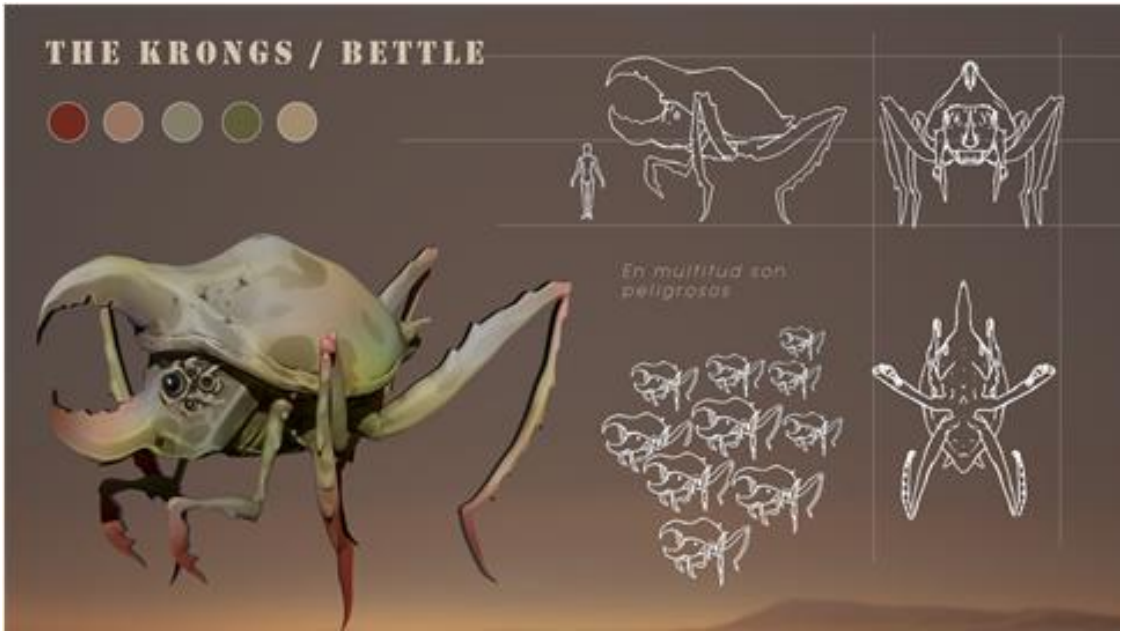


Figura 89. Krongs Vistas isométricas. (Elaboración propia)

Para este apartado se revisan los enemigos que atacarán a Kong Bot en el mapa. De tipo insecto llamados Krongs atacaran a KONG BOT en diferentes puntos del mapa y en un número determinado de enemigos por cada grupo de ataque.

5.7.1. General

5.7.1.1. Parámetros

Probablemente todos los enemigos posean los siguientes parámetros:

- Vida: Cantidad de daño que sufre el enemigo antes de ser eliminado, representada de manera continua.
- Velocidad: Medida con la cual los enemigos se movilizan puede ser afectada por diferentes comportamientos.
- Soltar: Los enemigos sueltan ítems de vida.
- Distancia: Medida con la que los enemigos detectan a KONG BOT

Otros parámetros se podrán añadir durante el desarrollo.

5.7.1.2. Navegación

- Los enemigos necesitan una capacidad de navegación básica. Por lo que deben evitar.
- Los enemigos no deben traspasar objetos en el mapa.

- Los enemigos no deben romper las físicas del mapa.
- Los enemigos no deben quedarse congelados.
- Los enemigos no deben solaparse entre sí.

Lo más importante es cómo perseguir a Kong Bot para atacar de la manera más coherente posible.

5.7.2. Los Krongs "Bettle"

Son los enemigos más comunes de encontrar en este planetoide.

Figura 90. Los Krongs "Bettle" importados en Unreal Engine4. (Elaboración propia)

5.7.2.1. Comportamientos

A continuación, se detallan una serie de comportamientos que deben ser mejorados durante el desarrollo.

Patrullaje

Los enemigos pertenecen al planetoide en sí es su territorio.

- Caminan desde su punto de spawn y las zonas de posible minado.
- De vez en cuando deben parar y hacer animaciones de manada.
- Si estaba acercándose a Kong Bot si no fueron asesinados y le perdieron la pista vuelven al punto de spawn.

Perseguir

Los enemigos detectan a Kong Bot en base a una distancia definida como parámetro (30 metros). Y el parámetro velocidad define qué tan rápido se acercarán a KONG BOT

Ataque

Para definir el ataque que efectúan los enemigos sobre Kong Bot se necesitan los siguientes parámetros.

- Daño: Cantidad de vida que será restada a Kong Bot
- Fuerza: Medida que indica cuánto efecto físico sufre Kong Bot cada Ataque
- Retraso: El tiempo que el enemigo le toma en recargar para atacar de nuevo.
- Oportunidad: Cantidad de enemigos que pueden atacar simultáneamente, este parámetro se debe contrastar con el parámetro de retraso.
- Rango: Medida de distancia que indica el inicio del ataque también permite sincronizar con la animación

Melee

Es el único tipo de ataque que poseen los enemigos Bettle, Una vez que los enemigos han alcanzado el valor de rango atacan a KONG BOT con sus tenazas requiere una animación y aplicar el valor de daño a Kong Bot.

Spawning

Los enemigos aparecen en los puntos de spawn designados en el mapa. Para generar más enemigos se deberán cumplir ciertas condiciones.

Los enemigos emergen a la superficie mediante animación, esto siempre y cuando estén en el rango de visión de KONG BOT, caso contrario solo aparecen.

Desaparecer

Los enemigos desaparecen solo cuando Kong Bot los elimina.

- Una vez que se termina la vida del enemigo se muestra la animación de muerte y un efecto de sonido.
- Ciertos enemigos sueltan ítems de vida, solo cuando Kong Bot ha eliminado una cierta cantidad de enemigos consecutivos sin perder vida.
- Cada muerte debe informarse a la lógica de nivel.

5.8. NIVELES

5.8.1. Resumen

Para este apartado se utilizan elementos gráficos y textuales.

5.8.1.1. Diseño de nivel

Considerando que el mapa de nivel es un planetoide se ha diseñado de manera esférica en base a esta premisa. Y en este diseño se muestra la ubicación de los puntos de spawn para los enemigos, la ubicación de las minas y el punto de spawn de KONG BOT.

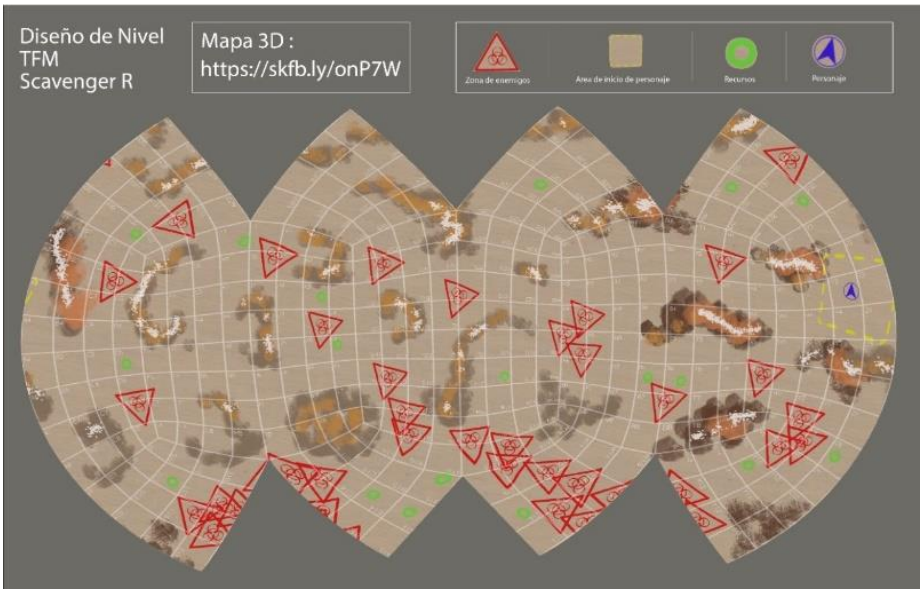


Figura 91. Diseño de nivel. (Elaboración propia)

5.8.1.2. 3D y Cámara

A tomar en cuenta en este nivel acerca de la cámara

- Usar la funcionalidad del motor gráfico elegido
- El seguimiento de la cámara al personaje debe ser fluido
- Evitar puntos ciegos o fallas en el seguimiento.

5.8.2. Elementos

5.8.2.1. Estética

Hace referencia a la experiencia sensorial, en base a elementos visuales y de sonido

- 3D: El juego usa los recursos 3D para mostrar este planetoide y las posibilidades de recorrer en todas las direcciones.

Figura 92. Moodboard de planetoide. (Elaboración propia)

- Música: Se puede usar fondos de música electrónica o de sintetizadores para comunicar el tiempo futurista con interludios de viento y silencio. En cuanto a los momentos de ataque se subirá la intensidad con música electrónica, incluyendo guitarra eléctrica y batería.
- Sonido: Los efectos de sonido incluyen sonidos de motor para Kong Bot, sonidos de insectos para los enemigos, chillidos en ataque o persecución, sonidos de disparos y explosiones para el ataque de Kong Bot.

5.8.2.2. General

- Colisiones: Es necesario usar el sistema de físicas del motor gráfico elegido, en ese sentido usar las mismas físicas para saltos, caídas, empujo de ataque
- Zonas de minado: En el mapa se deben ubicar este tipo de áreas que Kong Bot detecta para implantar un faro de minado.

5.8.2.3. Personajes

- Kong Bot: Es el personaje principal tiene un punto de spawn entre otros, ampliado en el apartado de personajes.
- Enemigos: Aparecen en los puntos de spawn para enemigos, se amplía en el apartado enemigos.
- Cápsulas: Cápsulas mineras básicamente NPC que cumplen las órdenes que da Kong Bot se amplía en el apartado personajes

5.8.2.4. Interacción

- Minas: Están ubicadas en diferentes puntos del mapa y para identificarlas es necesario que Kong Bot las detecte. Posteriormente marque las minas con el Faro de Minado para que las cápsulas mineras hagan el resto.
- Ítems: Estos elementos de salud están ubicados en diferentes puntos del mapa y cuando se eliminan enemigos.

5.8.2.5. Eventos

- Triggers: Ocurren en ciertos momentos durante la partida, como cuando Kong Bot elimina a todos los enemigos de una zona, se envía cápsulas de minado, se comienza a minar, o se termina de minar.
- Cinemáticas: Habrá cinemáticas en los momentos más importantes del juego, presentación de niveles, mecánicas y hechos de la historia que se quieran destacar durante el juego. Se hará hincapié sobre todo en el primer nivel para explicar el contexto y las mecánicas principales.

5.9. OBJETOS E ÍTEMS

5.9.1. Faro

Objeto con el cual Kong Bot marca el área a minar, inicia la animación desplegar mineros que irán desde el muelle de Kong Bot a la ubicación del faro.

5.9.2. Salud

Objeto que restaura la salud de Kong Bot, para ello existen dos tipos los que están en el mapa y los que sueltan los enemigos al morir.

5.10. SCREENFLOW

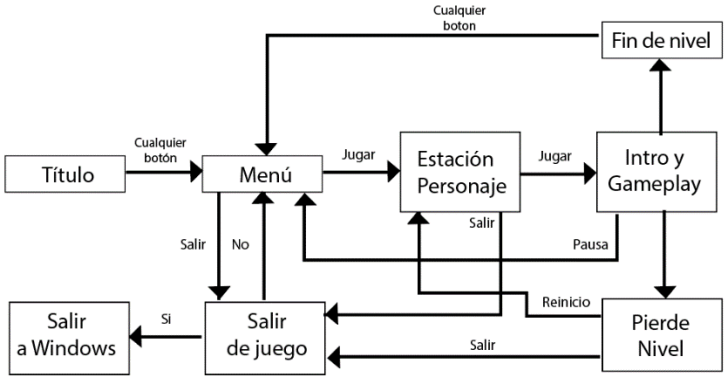


Figura 93. Screen Flow. (Elaboración propia)

5.10.1. Diseño de la UI

En el título al iniciar el juego la presentación de KONG BOT en la estación de abastecimiento con una pequeña animación cíclica, al presionar cualquier botón la cámara se aleja sobre la misma escena por el lado izquierdo entra el menú, al iniciar el juego se encuentran las opciones de personalización del personaje, al salir de la nave con el meca iniciamos el nivel en el planetoide, la interfaz es muy simple y minimalista, en la parte inferior izquierda la barra de salud y las municiones, se ubica un mapa que se activa y se despliega con la tecla M, y la linterna escáner que al presionar muestra la información en forma de hologramas sobre los recursos disponibles en el mapa, en la parte inferior derecha se encuentra el número de capsulas de extracción de minerales disponibles y la cantidad de recursos que se han recolectado.

Figura 94. HUD. (Elaboración propia)

5.11.NARRATIVA

El veterano de guerra de 58 años llamado "Matthews Gathely" conocido como Matt, proveniente del extremo sur de la región americana, ha decidido aventurarse fuera de la galaxia para buscar recursos minerales para su tribu, llamada "Almma 7" en un mundo post apocalíptico devastado por la guerra y la explotación desmedida de recursos.

Acompañado de la Inteligencia Artificial "Abi" pilotea su Mecha K2CP, un vehículo robot en forma de primate, puede derrotar a los enemigos y peligros que va encontrando en cada uno de los planetas para lograr su objetivo.

6. CONCLUSIONES

La conclusión que se obtiene al finalizar este proyecto es que se cumplieron los objetivos principales en general, el cual considera la construcción de un prototipo de videojuego con las especificaciones mencionadas en los objetivos del proyecto. Este prototipo incluye mecánicas consideradas base y escalables, que significa que puedan seguir desarrollándose en un futuro para transformarse en un videojuego comercial con un estilo artístico acorde a los videojuegos actuales, si bien llegamos a un apartado artístico muy bueno, faltaron algunas mejoras en cuanto a animaciones e interfaz de usuario que nos faltó mejorar.

Para poder completar este proyecto se tuvo que hacer un análisis de las mecánicas planteadas tempranamente, para conocer su factibilidad y alcance, crear un documento de diseño y planificación, para finalmente hacer su construcción desarrollada en el Motor de Unreal Engine 4.

El principal inconveniente que se tuvo fue que la lógica de gravedad no funcionó como se esperaba, ya que se hicieron cálculos que permitían hacer un impulso a nuestro personaje en cada instante del juego hacia el centro del planetoide, este no era preciso, ya que el terreno del nivel es muy irregular, y el hecho de hacer colisionar al jugador con tantos objetos hacía que el personaje o quedara en el aire, o que tuviera problemas de colisión con otros objetos. La solución a este problema es usar un plugin gratuito que está en la tienda de Unreal.

Otra problemática que se tuvo fue el abandono temprano de uno de los integrantes del proyecto por lo que el aspecto técnico de programación tuvo que tomarlo una sola persona, y que en su inicio estaba contemplado para dos.

Si bien se cumplieron los objetivos principales, hubo aspectos que no quedaron completos, como por ejemplo mejorar el sistema de combate, ya que queríamos incorporar al menos dos armas y solo alcanzamos a crear una, incorporar más enemigos, poner música, sonido o menú de juego que por limitantes de tiempo y técnicas se alcanzó a realizar.

7. POSTMORTEN DEL JUEGO

Durante el transcurso del proyecto siempre se intentó hacer un videojuego que resaltara a simple vista y que pudiera seguir desarrollándose como un proyecto real pensando siempre en PC y consolas. Con cada prototipo que se generaba tentaba a los autores a pensar en nuevas ideas y creaba inconformismo sobre el resultado de cada iteración. Sin embargo, con coordinación y esfuerzo se logró finalizar un prototipo del proyecto. Aun entre los autores queda el sin sabor de que el proyecto tiene mucho potencial, debido a que las mecánicas implementadas crean una sólida base para seguir desarrollando el proyecto y convertirlo en un juego comercial.

En un principio se pensó hacer un juego de naves que permitiera volar en la superficie planetaria e ir eliminando los enemigos que se presentaban, y buscando los yacimientos de minerales, el prototipo que se hizo se sentía bien, pero la decisión del equipo fue que un MECHA que pudiera escalar rocas podría ser más atractivo, además que la historia que se fue desarrollando entusiasmó mucho al equipo, luego de crear al mencionado MECHA en 3D se incorporó al juego ya se sabía que era la idea que se quería seguir, sin embargo la mecánica de escalado no llegó a puerto, ya que la dificultad de que pudiera posicionarse y crear las animaciones correspondientes eran muy complicadas, así que se desestimó.

Otro aspecto que se quiso realizar y no se pudo lograr fue la extracción de minerales, lo primero que se le ocurrió al equipo pensando además la referencia de Star Craft 2, fue el hecho de que mini robots fueran minando los yacimientos y transportando los minerales extraídos a la nave, mientras eran escoltados por el MECHA. Si bien se hicieron prototipos de esto, no se logró llegar a una solución que se sintiera bien.

También relacionado con los yacimientos, es que luego de desestimar la idea anterior, se pensó que hubiera una especie de puzle para la extracción muy parecido al juego "busca minas" (*Buscaminas - ¡juega Ahora!*, n.d.), pero se estimó que la idea no llegó a diseñarse correctamente, es por eso que no se implementó y se decidió por hacer algo más simple.

El proyecto, carece de ciertos detalles, como un menú de juego, o música y sonido que no se puedo implementar para esta entrega, o el resultado al obtener todos los minerales, sin embargo, el resultado final es satisfactorio, ya que se lograron las mecánicas esperadas, la ambientación y el estilo artístico.

Otro aspecto importante dentro del prototipo es que no se alcanzó a hacer un grupo de control para chequear que el juego funcione y que sea atractivo para el público por motivos de tiempo, lejanía de los integrantes y también considerar las dificultades del COVID. Esto es uno de los aspectos que más se lamenta no haber podido hacer, ya que la realimentación con los usuarios es fundamental, sobre todo en esta etapa del desarrollo.

Finalmente hay que destacar el trabajo en equipo que, si bien al principio costó echarlo a andar, se llegó a un consenso colectivo y a una dinámica de trabajo que permito obtener los resultados presentados, reconociendo siempre que al proyecto le faltaron algunos aspectos para que pudiera presentarse como un demo y no solamente un prototipo.

8. REFERENCIAS BIBLIOGRÁFICAS Y DE OTROS VIDEOJUEGOS

Análisis - Deiland — Legión de Jugadores. (n.d.). Retrieved July 22, 2021, from https://legiondejugadores.com/all-review-list/deiland/

Buscaminas - ¡juega ahora! (n.d.). Retrieved September 21, 2021, from https://buscaminas.eu/

Chandler, H. M. (2009). *The game production handbook* (Second Edi). Infinity Science Press LLC 2009.

Chandler, H. M. (2020). The Game Production Toolbox. In *The Game Production Toolbox*. CRC Press. https://doi.org/10.1201/9780429440021

Deiland. (n.d.). Retrieved September 21, 2021, from https://store.playstation.com/es-cl/product/UP3868-CUSA12406_00-D000000000001001/

Free Super Mario Galaxy Wallpaper | Super mario galaxy, Galaxy wallpaper, Super mario. (n.d.). Retrieved July 21, 2021, from https://www.pinterest.cl/pin/169588742197119342/

Helldivers saldrá a la venta el 3 de marzo en PS4, PS3 y PS Vita. (n.d.). Retrieved July 21, 2021, from http://www.desconsolados.com/2015/02/14/helldivers-saldra-a-la-venta-el-3-de-marzo-en-ps4-ps3-y-ps-vita/

Keith, C. (2010). *Agile Game Development with Scrum (Addison-Wesley Signature Series* (1st ed.). Addison-Wesley Professional. http://www.amazon.com/dp/0321618521

Llansó García, D. (2014). *Metodología ontológica para el desarrollo de videojuegos*. 290.

Star Craft 2 - 1366_2000.jpg (1050×760). (n.d.). Retrieved September 21, 2021, from https://i.blogs.es/27e2a2/starcraft-2-gratis-5/1366_2000.jpg

StarCraft II: Wings of Liberty - Wikipedia. (n.d.). Retrieved September 21, 2021, from https://en.wikipedia.org/wiki/StarCraft_II:_Wings_of_Liberty

Super Mario Galaxy (Wii) Game Profile | News, Reviews, Videos & Screenshots. (n.d.). Retrieved September 21, 2021, from https://www.nintendolife.com/games/wii/super_mario_galaxy

Unreal Architecture | Unreal Engine Documentation. (n.d.). Retrieved August 16, 2021, from https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/ProgrammingWithCPP/UnrealArchitecture/