Universidad ORT Uruguay
Facultad de Ingeniería

Not an oasis

Videojuego de extracción, acción y aventura

Proyecto colaborativo Ingeniería en sistemas, Licenciatura en sistemas y Licenciatura en animación y videojuegos

Entregado como requisito para la obtención del título de Licenciado en Sistemas

Daniel Komés - 225841

Tutora: Helena Garbarino

2025

Declaración de autoría

Nosotros, Javier de Mattos y Daniel Komés, declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el Proyecto Final de Carrera;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Javier de Mattos
10/04/2025

Daniel Komés
10/04/2025

Abstract

Este documento presenta el desarrollo de *Not an Oasis*, un videojuego de extracción, acción y aventura ambientado en un desierto post apocalíptico. El objetivo principal del juego es explorar un mundo hostil en busca del oasis, y recogiendo recursos para construir y desbloquear mejoras que potencien al jugador y a su acompañante, un robot autónomo que asiste en la aventura con habilidades especiales como disparar, excavar, hackear e interactuar con el entorno. Estas habilidades pueden personalizarse e intercambiarse mediante la instalación de diferentes mods, lo que introduce un componente estratégico y de personalización en la jugabilidad.

El proyecto fue desarrollado con el motor Unity y empleó metodologías ágiles adaptadas a un equipo pequeño. Se documentaron todas las etapas del desarrollo, incluyendo diseño, implementación, actividades de SQA y gestión de riesgos. A lo largo del proceso se priorizó la iteración y el testeo constantes, permitiendo un avance sostenido y controlado. El resultado es un prototipo funcional que combina exploración, combate y toma de decisiones tácticas, con una narrativa centrada en la supervivencia y la búsqueda del oasis oculto.

Palabras clave

Unity, videojuego, C#, metodologías ágiles, proyecto colaborativo, jugabilidad, pruebas con usuarios, desierto, post apocalíptico, 3D, estilizado

Índice

1. Introducción..... 10
1.1. Integrantes del equipo..... 10
1.2. Objetivos..... 10
1.2.1. Objetivos académicos..... 10
1.2.2. Objetivos del proyecto..... 11
1.2.3. Objetivos del juego..... 11
1.3. Hitos..... 11
1.4. Análisis del mercado..... 12
1.5. Propuesta de valor..... 14
1.6. Descripción del problema..... 14
1.7. Descripción de la solución..... 15
1.7.1. Flujo del juego..... 15
2. Ingeniería de requerimientos..... 16
2.1. Introducción..... 16
2.2. Proceso de ingeniería de requerimientos..... 16
2.3. Documentación..... 17
2.4. Validación..... 17
2.4.1. Relevancia con respecto a la visión del juego..... 17
2.4.2. Viabilidad técnica y temporal..... 17
2.4.3. Coherencia con la milestone actual..... 17
2.4.4. Feedback del docente y del game designer..... 17
2.4.5. Impacto sobre la experiencia del usuario..... 18
2.4.6. Pulido vs. nuevas funcionalidades..... 18
2.5. Requerimientos funcionales..... 18
2.5.1. Menú de inicio..... 18
2.5.2. Menú de pausa..... 19
2.5.3. Gobi..... 19
2.5.3.1. Movimiento..... 19
2.5.4. TA-2..... 20
2.5.4.1. Habilidades de TA-2..... 20
2.5.5. Mods..... 21
2.5.6. Terminal de mods..... 21
2.5.7. Enemigos..... 22
2.5.7.1. Comportamiento de enemigos..... 23
2.5.8. Economía..... 24
2.5.9. Niveles..... 25
2.5.10. Armas..... 26

2.6. Requerimientos no funcionales..... 28
2.6.1. RNF1 - Diseño atractivo e intuitivo..... 28
2.6.2. RNF2 - Rendimiento aceptable..... 28
2.6.5. RNF5 - Requerimientos de hardware mínimos y recomendados..... 28
3. Arquitectura de la solución..... 30
3.1. Desarrollo C#..... 30
3.1.1. Estructura y módulos..... 30
3.1.2. Clean Code..... 37
3.1.2.1. Beneficios..... 38
3.1.2.1.1. Legibilidad..... 38
3.1.2.1.2. Mantenibilidad..... 38
3.1.2.1.3. Profesionalismo..... 38
3.1.2.1.4. Integración..... 38
3.1.2.2. Prácticas utilizadas..... 38
3.1.2.2.1. Comentarios..... 38
3.1.2.2.2. Nombres de variables y funciones descriptivos..... 39
3.1.2.2.3. Funciones pequeñas y con una sola responsabilidad..... 39
3.1.2.2.4. Evitar “números mágicos”..... 40
3.1.2.2.5. Evitar duplicación de código..... 40
3.1.2.2.6. Evitar funciones con demasiados parámetros..... 41
3.1.2.2.7. Manejo correcto de excepciones y errores..... 41
3.1.2.2.8. Uso adecuado de estructuras de control..... 42
3.2. Decisiones de diseño..... 42
3.2.1. Utilización del patrón State..... 42
3.2.2. Utilización del patrón Singleton..... 43
3.2.3. Utilización de herencia..... 44
3.3. Herramientas y tecnologías de desarrollo..... 44
3.3.1. Herramientas y tecnologías..... 44
3.3.2. Justificación de herramientas y tecnologías..... 53
4. Jugabilidad..... 55
4.1. Introducción..... 55
4.2. Atributos de jugabilidad..... 55
4.2.1. Satisfacción..... 55
4.2.2. Aprendizaje..... 55
4.2.3. Efectividad..... 55
4.2.4. Inmersión..... 55
4.2.5. Motivación..... 55
4.2.6. Emoción..... 55

4.2.7. Socialización..... 56
4.3. Correlación entre atributos de jugabilidad..... 56
4.4. Factores y atributos de calidad en videojuegos..... 57
4.4.1. Efectividad..... 57
4.4.2. Eficiencia..... 57
4.4.3. Seguridad y prevención..... 57
4.4.4. Satisfacción..... 57
4.5. Facetas de la jugabilidad..... 57
4.5.1. Jugabilidad intrínseca..... 57
4.5.2. Jugabilidad mecánica..... 57
4.5.3. Jugabilidad interactiva..... 57
4.5.4. Jugabilidad artística..... 58
4.5.5. Jugabilidad intrapersonal..... 58
4.5.6. Jugabilidad interpersonal..... 58
4.6. Métricas jugabilidad..... 58
4.6.1. Efectividad..... 58
4.6.2. Eficiencia..... 59
4.6.3. Seguridad y prevención..... 59
4.6.4. Satisfacción..... 59
4.7. Análisis de métricas jugabilidad..... 61
4.7.1. Efectividad..... 61
4.7.2. Eficiencia..... 62
4.7.3. Seguridad y prevención..... 63
4.7.4. Satisfacción..... 63
4.7.5. Conclusión..... 67
5. Gestión de la calidad..... 68
5.1. Introducción..... 68
5.2. Objetivos de calidad del producto..... 68
5.3. Objetivos de calidad del proceso..... 68
5.4. Objetivos de calidad académicos..... 69
5.5. Aseguramiento de la calidad (SQA)..... 69
5.6. Actividades principales de SQA..... 70
5.7. Definición de métricas..... 73
5.7.1. Introducción..... 73
5.7.2. Métricas del producto..... 74
5.7.3. Métricas de proceso..... 75
5.7.4. Métricas de proyecto..... 76
5.8. Análisis y desarrollo de métricas..... 77

5.8.1. Resultado de métricas del producto..... 77
5.8.2. Resultado de métricas del proceso.....81
5.8.3. Resultado de métricas del proyecto.....86
5.9. Conclusión de plan de calidad..... 88
6. Gestión del proyecto..... 89
6.1. Proceso de desarrollo..... 89
6.2. Roles..... 90
6.2.1. Product owner.....91
6.2.2. Project manager.....91
6.2.3. Scrum master.....91
6.2.4. Ingeniero de requerimientos..... 91
6.2.5. Responsable de SQA..... 91
6.2.6. Responsable de SCM..... 91
6.2.7. Game designer..... 91
6.2.8. Director de arte..... 91
6.2.9. Artista técnico..... 92
6.2.10. Artista.....92
6.2.11. Diseñador UX / UI..... 92
6.2.12. Desarrollador..... 92
6.3. Artefactos..... 92
6.4. Ceremonias..... 93
6.5. Medición del trabajo..... 93
6.6. Definición de backlog..... 94
6.7. Estimación y planificación..... 95
6.8. Gestión de riesgos..... 98
6.8.1. Categorización de riesgos..... 98
6.8.2. Definición de riesgos.....99
6.8.3. Conclusiones de riesgos..... 103
6.9. Gestión de la comunicación..... 103
6.10. Resultado de sprints.....105
6.10.1. Sprint 1 (21-08-2024).....105
6.10.2. Sprint 2 (04-09-2024).....106
6.10.3. Sprint 3 (18-09-2024).....107
6.10.4. Sprint 4 (02-10-2024).....108
6.10.5. Sprint 5 (16-10-2024).....109
6.10.6. Sprint 6 (30-10-2024).....109
6.10.7. Sprint 7 (13-11-2024)..... 110
6.10.8. Sprint 8 (27-11-2024).....111

6.10.9. Sprint 9 (11-12-2024)..... 112
6.10.10. Sprint 10 (25-12-2024)..... 112
6.10.11. Sprint 11 (08-01-2025)..... 113
6.10.12. Sprint 12 (22-01-2025)..... 113
6.10.13. Sprint 13 (05-02-2025)..... 114
6.10.14. Sprint 14 (19-02-2025)..... 114
6.10.15. Sprint 15 (05-03-2025)..... 115
6.10.16. Sprint 16 (19-03-2025)..... 115
7. Gestión de la configuración..... 116
7.1. Control de versiones..... 116
7.2. Estructura de ramas..... 116
8. Conclusiones..... 117
9. Referencias bibliográficas..... 118
10. Anexo..... 119
10.1. Descarga del juego..... 119
10.2. GDD (Game Design Document)..... 119
10.3. Lista de tareas..... 119
10.4. Encuestas..... 119
10.5. Trailer..... 121

1. Introducción

Not an Oasis es un juego de aventuras en tercera persona con arte 3D estilizado, donde el jugador asume el rol de Gobi, un sobreviviente que, junto a su compañero robótico TA-2, busca recursos para asegurar su supervivencia y respuestas acerca del origen del mundo desolado en el que viven.

Combina agilidad y combate estratégico para superar terrenos hostiles y enfrentar enemigos carroñeros. TA-2 será tu mayor aliado, ya que su apoyo puede marcar la diferencia entre la vida y la muerte.

La narrativa te transporta a un entorno árido y peligroso, donde cada batalla es un desafío por la supervivencia. Regresa a la Aldea para descansar, mejorar tus habilidades y las de tu compañero robótico y prepararte para la próxima incursión.

Este juego promete acción constante y la emoción de sobrevivir en un mundo donde incluso los lugares más desolados esconden sorpresas.

1.1. Integrantes del equipo

Javier de Mattos, de Ingeniería en Sistemas.

Daniel Komés, de Licenciatura en Sistemas.

Franco Baretto y Martina Abascal, de Licenciatura en Animación y Videojuegos.

1.2. Objetivos

1.2.1. Objetivos académicos

Como proyecto final de carrera, los objetivos académicos deben representar los conocimientos adquiridos por cada integrante a lo largo de sus carreras.

Algunos de los objetivos propuestos fueron:

- Coordinar un equipo de trabajo multidisciplinario, donde los integrantes no se conocían previamente, con diferentes niveles de experiencia en las distintas herramientas usadas, y diferente disponibilidad horaria.
- Adaptarse a un proyecto existente, definiendo métodos para integrar nuevos requerimientos y sus criterios de aceptación.
- Desarrollar código de forma correcta, extensible, prolija y en equipo.

- Producir documentación completa y comprensible, siguiendo los estándares establecidos por la Universidad.

1.2.2. Objetivos del proyecto

- Crear una experiencia inmersiva, limpia, estética.
- Implementar mecánicas interesantes de movimiento, combate y gestión de recursos.
- Diseñar situaciones que requieran estrategia.
- Diseñar armas variadas.
- Desarrollar una historia interesante.
- Aplicar arte estilizado con temática tribal y desértica.
- Lograr una vista 3D en tercera persona.

1.2.3. Objetivos del juego

- Lograr un mundo estilizado, con aspectos únicos mezclando una temática tribal con un mundo post-apocalíptico.
- Desarrollar un conjunto de mecánicas que permitan una movilidad fluida.
- Toma de decisiones guiada por el jugador dando variedad a cada incursión.
- Desarrollar un conjunto de armas que hagan interesante la rejugabilidad.
- Implementar una gestión del proyecto eficiente y una comunicación constante y clara.
- Aplicar buenas prácticas para desarrollar un producto extensible y mantenible.

1.3. Hitos

20/07/2024: Sprint 0: se comienza el trabajo preliminar de preparación.

20/08/2024: Comienzo de Sprint 1.

13/11/2024: Presentación mutua con los proyectos de animación.

11/12/2024: Betatesting

13/02/2025: Informe de avance de Licenciatura.

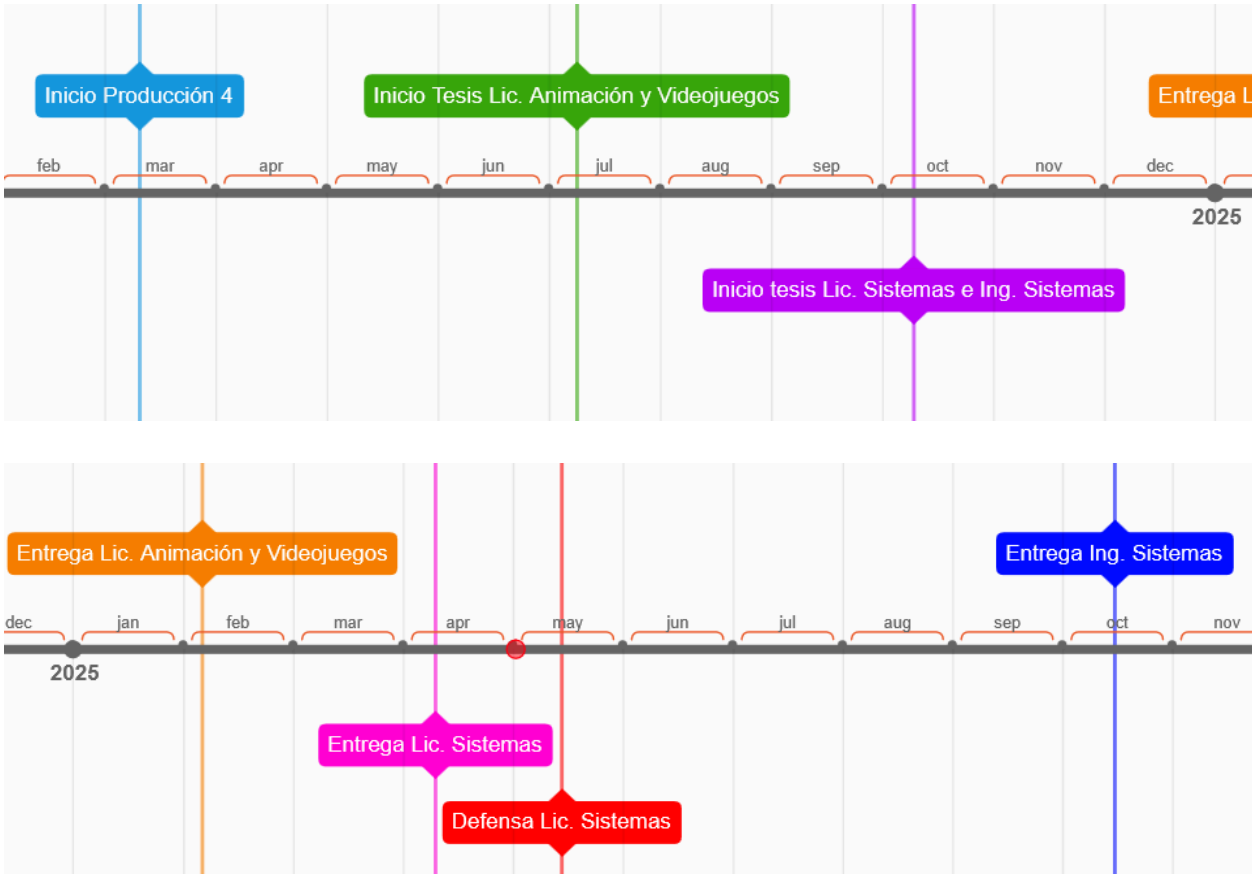
04/03/2025: Revisión de proyecto.

10/04/2025: Entrega final de Licenciatura.

10/04/2025: Informe de avance de Ingeniería.

11/08/2025: Tercera revisión de Ingeniería.

16/10/2025: Entrega final de Ingeniería.



1.4. Análisis del mercado

Se analizaron proyectos similares en el mercado como inspiración y referencia.

Sable:

Analizado por su diseño del entorno y mundo, estilo de arte y flujo general de juego.

Developer: Shedworks

Publisher: Raw Fury

First Release Date: 22 de septiembre, 2021

165.000 de unidades vendidas

$2.8 millones en ventas.

Mad Max:

Analizado por la ambientación y temática de la historia.

Developer: Avalanche Studios

Publisher: Warner Bros. Interactive Entertainment, Warner Bros. Games

First Release Date: 1 de octubre, 2015

3 millones de unidades vendidas

$71.8 millones en ventas

Conclusión:

Estos juegos destacan por sus mecánicas en común:

Supervivencia: desafíos y amenazas que presentan obstáculos en el camino del jugador.

Personalización: capacidad de mejorar herramientas, armas y otras habilidades para cambiar su comportamiento y poder adaptarlas a la situación según se desee.

Historia: narrativa inmersiva que el jugador va descubriendo a medida que juega, haciendo todo el proceso más divertido y atrapante.

Ambiente inmersivo: el mundo da la sensación de estar vivo y ser independiente, y permitir al jugador interactuar y tomar ventaja de él como parte de su estrategia.

1.5. Propuesta de valor

Teniendo en cuenta el análisis del mercado realizado, se concluyó que la propuesta de valor de este proyecto sería la historia, estética, mecánicas y flujo del juego.

Para la historia se planificó una narrativa de misterio y supervivencia, y que el jugador fuera descubriéndola al explorar, en pequeños fragmentos.

Para la estética, se pretendía crear arte estilizado con estilo cartoon.

Para las mecánicas y flujo del juego, se pretendía crear algo único, entretenido, que empujara al jugador a planear y probar distintas estrategias.

1.6. Descripción del problema

Se concluyó que el género más adecuado para la situación es el de extracción, donde el jugador se prepara en su base y luego viaja y explora el nivel principal con un límite de tiempo, con el objetivo de visitar distintas áreas en cada exploración.

Además, un factor importante es la rejugabilidad, por lo que se debían idear mecánicas y flujos de juego que hicieran que el jugador encuentre valor en repetir los niveles para conseguir más recursos opcionales, y también repetir el juego entero, para encontrar diferentes estrategias posibles.

1.7. Descripción de la solución

La solución se enfocó principalmente en la rejugabilidad, para lo que se diseñó un sistema de mejoras modulares instalables en TA-2, el robot que acompañaría constantemente al jugador, permitiéndole enfrentar distintas situaciones con las distintas herramientas que decidiera instalar.

1.7.1. Flujo del juego

El objetivo del jugador es explorar el mundo descubriendo su historia, derrotar enemigos, y recoger recursos para construir mejoras que le permitan explorar nuevas zonas.

El juego consta de dos mapas: la aldea para preparar el equipo, comprar mejoras y descansar sin peligro, y el desierto, donde los enemigos son frecuentes y la supervivencia es difícil.

El bucle del juego es:

Desde la aldea, prepararse para la siguiente misión equipando las armas y herramientas deseadas, y cambiando las habilidades de TA-2.

Ir al nivel principal en el desierto, derrotar enemigos, conseguir recursos, explorar.

Volver a la aldea, desbloquear mods que permitirán explorar nuevas zonas y prepararse nuevamente.

Repetir.

2. Ingeniería de requerimientos

2.1. Introducción

El proceso de ingeniería de requerimientos fue un esfuerzo conjunto entre las áreas de arte y programación

2.2. Proceso de ingeniería de requerimientos

El proceso de ingeniería de requerimientos en este proyecto se estructuró de forma iterativa, integrando elementos del enfoque ágil y adaptándose a la realidad del equipo, el alcance progresivo del proyecto y las recomendaciones del docente a cargo. En una primera instancia, se tomó como base el juego ya existente, desarrollado por Franco y Alejo en la materia que para Ingeniería en Sistemas y Licenciatura en Sistemas corresponde a Desarrollo de Videojuegos 2, que si bien presentaba un esqueleto funcional, carecía de claridad en sus objetivos y tenía mecánicas limitadas. A partir de esta base, y a través de un proceso colaborativo entre los miembros del nuevo equipo y Franco (el diseñador original del proyecto, quien permaneció como parte activa del equipo), se redefinió la dirección del juego.

Durante las primeras semanas, se generó una visión general de los objetivos del proyecto, priorizando aspectos de jugabilidad, claridad, coherencia narrativa y rejugabilidad. Esta visión fue la base para derivar los primeros requerimientos funcionales y no funcionales. A partir de allí, y siguiendo la estructura de trabajo sugerida por el docente, se dividió el desarrollo en milestones de duración variable, cada uno enfocado en una temática o conjunto de mecánicas específicas, las que definimos fueron: prototipo inicial, combate básico, combate avanzado, preparación de entrega final, entrega final y entrega para licenciatura. Los requerimientos que surgían se asociaban directa o indirectamente a estos bloques de trabajo, permitiendo que cada iteración del desarrollo estuviera alineada con un conjunto de objetivos claros.

La generación de requerimientos fue incremental. A medida que se implementaban nuevas funcionalidades, se detectaban oportunidades de mejora, problemas de claridad, errores de implementación o necesidades estéticas que derivaban en nuevos requerimientos. Asimismo, se recibían sugerencias tanto desde el equipo de desarrollo como desde las instancias de revisión con el docente (Álvaro) y el game designer (Franco), quienes cumplían un rol crucial en la validación de ideas. Esto permitió mantener una coherencia estética, técnica y de alcance a lo largo del proceso, también un factor clave en el añadido de requerimientos fueron las opiniones dadas por los beta testers en la instancia de Beta test ocurrida en diciembre.

2.3. Documentación

2.4. Validación

Los requerimientos eran aceptados o descartados en función de una combinación de criterios técnicos, funcionales, estéticos y estratégicos, que se fueron consolidando como parte del proceso interno del equipo.

La validación de los requerimientos aceptados se realizaba a través de la planificación de tareas por sprint, el seguimiento mediante tableros colaborativos, y las instancias de revisión en sprint reviews. Cada requerimiento debía tener una implementación concreta que pudiera ser evaluada por el equipo y validada por el docente, ya sea a través de pruebas de jugabilidad o de presentaciones realizadas al docente.

Estos criterios de aceptación fueron los que se detallan a continuación.

2.4.1. Relevancia con respecto a la visión del juego

Solo se aceptaban requerimientos que aportaran valor claro a la experiencia de juego, ya sea mejorando la jugabilidad, la comprensión del jugador, la estética general o la rejugabilidad. Ideas que se alejaban de esta visión, aunque técnicamente viables, eran descartadas o reformuladas.

2.4.2. Viabilidad técnica y temporal

Uno de los criterios más importantes fue la estimación de complejidad y tiempo de desarrollo. Si una funcionalidad representaba un riesgo de sobrecarga para el equipo, o bien era difícil de integrar sin comprometer otras partes del proyecto, se decidía postergarla o descartarla, priorizando el pulido de lo ya implementado.

2.4.3. Coherencia con la milestone actual

Cada milestone tenía un enfoque temático definido (por ejemplo, combate básico, combate avanzado, etc.). Los requerimientos eran priorizados en función de su relación directa con los objetivos de la milestone. Esto evitaba dispersión y mantenía al equipo enfocado, si el requerimiento no encajaba con la milestone actual pero cumplía con los otros criterios de aceptación, este requerimiento entraba al backlog en forma de tareas para ser agregado luego.

2.4.4. Feedback del docente y del game designer

La opinión de Álvaro (docente a cargo) y Franco (diseñador del juego) funcionaba como filtro decisivo. Si cualquiera de los dos consideraba que una propuesta no sumaba valor, rompía con la estética, o simplemente no era viable por cuestiones de tiempo o alcance, se discutía con el equipo para intentar reformularla. Si no era posible justificarla dentro del marco general del proyecto, se descartaba.

2.4.5. Impacto sobre la experiencia del usuario

Se priorizaban requerimientos que facilitaran la comprensión del jugador, que dieran más claridad sobre las acciones posibles dentro del juego o que mejoraran la forma en que se presentaban las mecánicas principales (por ejemplo, mejorar la visibilidad del robot, mostrar mejor los estados de los enemigos, etc.).

2.4.6. Pulido vs. nuevas funcionalidades

En varias instancias del desarrollo, especialmente en la milestone relacionada al combate avanzado, se priorizó el refinamiento de sistemas ya implementados por encima de agregar nuevas mecánicas. Esto respondía a una estrategia consciente de mejorar la calidad general del producto en lugar de expandirlo sin control, para esto tomamos la recomendación de Juan Rodríguez (quien integra la organización de Jam2Jam en Uruguay y cuenta con amplia experiencia en el desarrollo de videojuegos), quien fue nuestro mentor temporalmente, y nos recomendó priorizar el mejorar funcionalidades ya desarrolladas por sobre funcionalidades nuevas.

2.5. Requerimientos funcionales

2.5.1. Menú de inicio

La pantalla que se muestra al iniciar el juego. Con opciones para iniciar la partida, borrar el progreso, y salir del juego.

2.5.2. Menú de pausa

Menú que aparece al presionar la tecla correspondiente (Escape) durante el juego. El juego se pausa (el tiempo se detiene) mientras está visible. Tiene opciones para continuar, ver los controles, cambiar opciones, y salir al menú principal.

2.5.3. Gobi

El personaje principal controlado por el jugador. Tiene distintas mecánicas de movimiento e interacciones para moverse por el mundo.

2.5.3.1. Movimiento

Caminar: presionando las teclas de movimiento (WASD) se puede mover lentamente en la dirección presionada respecto a la dirección que la cámara apunta.

Correr: sosteniendo shift izquierdo mientras se camina se puede mover rápidamente con la misma lógica que caminar.

Saltar: presionando espacio se puede impulsar hacia arriba, despegándose del suelo. Se puede controlar el movimiento en el aire con las teclas de movimiento.

Trepar: presionando espacio mientras Gobi está en el aire se puede alcanzar bordes cercanos en el entorno: se ejecuta una pequeña animación de trepado durante la que se impide cualquier input del jugador, y al terminar, Gobi estará posicionado sobre el borde al que trepó.

Rodar: presionando y soltando shift izquierdo se puede rodar mientras se está en el suelo: Gobi ejecuta una pequeña animación de rodar mientras se desplaza rápidamente una distancia corta sobre el suelo.

2.5.4. TA-2

Compañero de Gobi. Un pequeño robot que ayuda al jugador durante el combate y con distintas interacciones en el entorno.

2.5.4.1. Habilidades de TA-2

Seguir: TA-2 sigue a Gobi. Si Gobi está quieto, cada cierto tiempo se reposiciona en un área cercana, para darle una sensación de autonomía.

Moverse a ubicación: el jugador puede indicar una posición en el terreno apuntando la cámara en la dirección deseada y presionando el botón de dar órden (número “1” del teclado). No se pueden indicar posiciones demasiado lejanas a Gobi. Una vez que TA-2 llega a la posición indicada se mantiene allí sin moverse.

Atacar: el jugador puede marcar enemigos para ser atacados por TA-2 (click izquierdo del ratón). Los enemigos marcados tienen un contorno blanco para indicarlo. Si Gobi está cerca del objetivo, TA-2 se posicionará entre Gobi y el objetivo y lo atacará. Si Gobi se aleja del objetivo, TA-2 lo seguirá con su comportamiento de seguir. Si Gobi se aleja aún más del objetivo, será desmarcado. Sólo puede haber un objetivo marcado. Si ya había uno, el anterior se desmarcará.

Cavar: TA-2 cava montículos en la arena, y luego de su animación, aparece encima el objeto que estaba enterrado. Si TA-2 está siguiendo a Gobi (no tiene ninguna orden manual) automáticamente encontrará y cavará montículos que tenga cerca en un pequeño radio a su alrededor. El jugador puede impedir esto dándole una orden de posicionarse en otro lugar. También el jugador puede ordenar manualmente que cave, en caso de que TA-2 esté priorizando otra acción, apuntando la cámara al montículo deseado y presionando el botón de moverse a ubicación (número “1” del teclado).

Hackear: el jugador puede ordenar a TA-2 interactuar con una terminal apuntándole con la cámara y presionando el botón de moverse a ubicación (número “1” del teclado). TA-2 se dirigirá a ella y, luego de unos segundos, la terminal ejecutará su acción y TA-2 volverá a su comportamiento de seguir a Gobi.

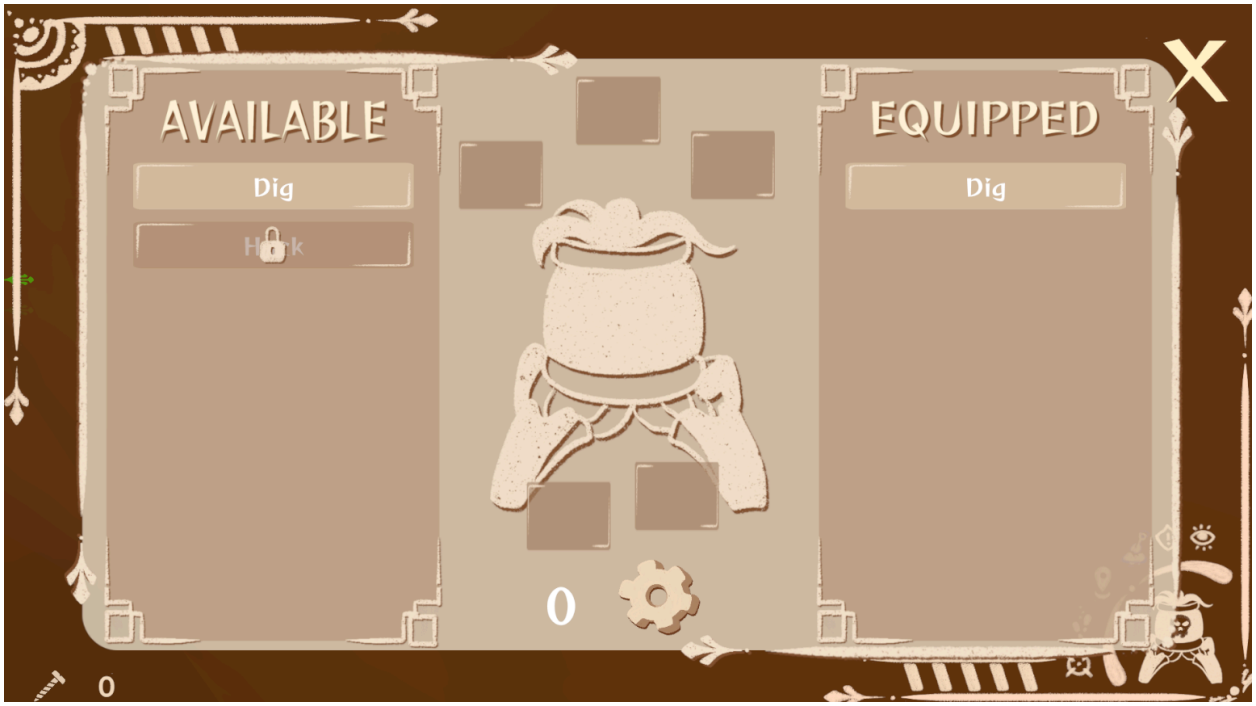
2.5.5. Mods

Objetos recolectables que se guardan en el inventario del jugador cuando Gobi los toca. Son usados para instalar comportamientos en TA-2 usando una terminal de mods. Algunos mods pueden estar bloqueados y requieren recursos para desbloquear.

2.5.6. Terminal de mods

Una computadora con la que Gobi puede interactuar. Al usar, se muestra una pantalla donde se listan los mods en el inventario del jugador, los mods instalados en TA-2, y los recursos en el inventario del jugador para desbloquear mods. Haciendo clic en los mods en la lista del inventario, se instalan en TA-2, dándole el comportamiento correspondiente. Haciendo clic en los mods de la lista de instalados, se desinstalan y TA-2 pierde el comportamiento correspondiente.

Haciendo clic en un mod bloqueado, si el jugador tiene en su inventario los recursos suficientes, se desbloquea y se descuentan los recursos del inventario usados.



2.5.7. Enemigos

Los enemigos son robots hostiles hacia Gobi y TA-2 con los que se puede entrar en combate. Son la principal amenaza para el jugador. Los enemigos tienen un campo de visión limitado en ángulo y distancia que les da un comportamiento dinámico.

Los enemigos en combate alertarán a otros enemigos cercanos para que investiguen la posición donde fue avistado el jugador.

Los enemigos investigarán a su alrededor si reciben daño.

Los personajes en la misma facción no pueden dañarse entre sí (los enemigos no pueden dañar a otros enemigos, y el jugador y TA-2 no pueden dañarse entre ellos).

Los enemigos son marcados por un indicador en la pantalla, que apunta en la dirección donde están, con un color que representa su estado (comportamiento) actual.

Los enemigos tienen varios estados con comportamientos diferentes.

2.5.7.1. Comportamiento de enemigos

Desactivado: un estado inicial especial, sólo encontrado en algunos enemigos en el nivel. El enemigo está inmóvil, ciego, no se muestra su indicador en la pantalla y no se puede seleccionar como objetivo. Una vez fuera de este estado, no regresan a él. En este estado su modelo 3D tiene una posición especial parecida a dormir sentado.

Patrullar: estado por defecto. indicador color verde. El enemigo sigue una ruta predefinida en el mapa. Se usa este estado cuando el enemigo está “tranquilo” (no se cumplen las condiciones para ningún otro estado).

Distraído por TA-2: indicador color violeta. Si un enemigo ve a TA-2 pero no ve al jugador, el enemigo sigue inofensivamente a TA-2. El propósito de este estado es permitir trazar estrategias de sigilo distrayendo a los enemigos utilizando los comandos de movimiento de TA-2.

Combate: indicador color rojo. Un enemigo puede tener distintos ataques, dependiendo de sus condiciones. Si tiene un arma del tipo indicado a continuación, puede usar el ataque asociado.

Cuerpo a cuerpo: el enemigo puede golpear a su objetivo, impidiéndole actuar por unos segundos y empujándolo hacia atrás. Durante el combate, el enemigo puede decidir usar este ataque, deteniendo otros ataques y corriendo hacia su objetivo para golpearlo cuando esté suficientemente cerca. También puede usar este ataque cuando su objetivo se acerca demasiado mientras está usando otros ataques. Mientras está ejecutando su animación de golpear, el enemigo no puede caminar ni girar.

A distancia: el enemigo usa su arma a distancia en la dirección de su objetivo. Este es el ataque más frecuente que los enemigos realizan. Las armas a distancia pueden ser la pistola láser, u objetos lanzables que puedan encontrarse cerca en el entorno.

Buscar arma a distancia: indicador color naranja. Si el enemigo no tiene equipada un arma a distancia (pistola láser u objeto lanzable) buscará en el entorno, a intervalos regulares, un arma a distancia. Si la encuentra, se dirigirá hacia ella, ignorando a sus objetivos hasta que la alcance y equipe.

Investigar: indicador color amarillo. Cuando un enemigo que está en combate pierde de vista a sus objetivos, entra en este estado, donde se dirigirá a la última posición donde vio a su objetivo. Al llegar, se detendrá y observará a su alrededor, buscando. Luego de unos segundos, si no encuentra objetivos, volverá a patrullar.

2.5.8. Economía

Durante el juego se pueden encontrar recursos necesarios para avanzar desbloqueando mods. Estos recursos son agregados al inventario del jugador cuando Gobi los toca. Se pueden obtener haciendo que TA-2 excave en montículos en la arena, y derrotando enemigos. Los enemigos tienen un 50% de probabilidades de soltar una unidad del recurso.

2.5.9. Niveles

Aldea: donde se inicia el juego. Una zona pacífica, sin peligros, donde el jugador puede prepararse para la siguiente incursión. Tiene una zona de práctica de tiro y una terminal de mods para preparar las habilidades de TA-2.

Incursión: el nivel principal, donde el jugador puede combatir con los enemigos, recolectar recursos y resolver puzzles para terminar la misión.

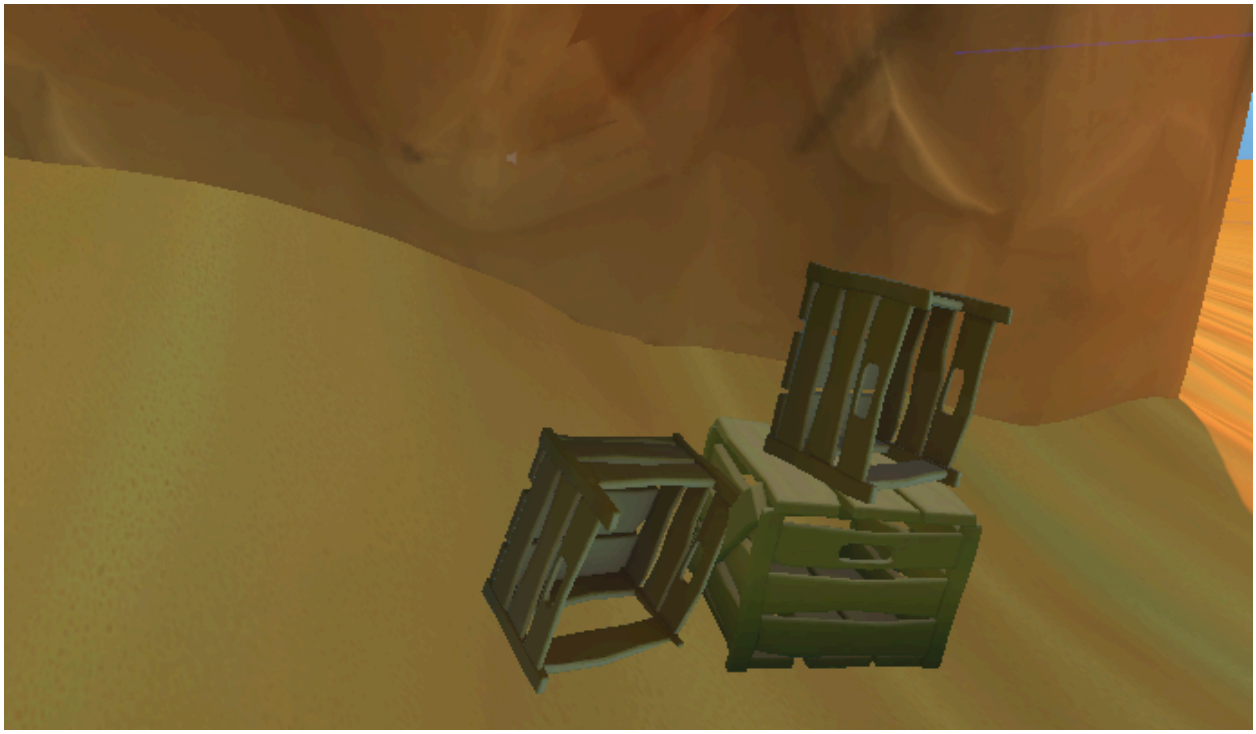
2.5.10. Armas

Existen armas de distintos tipos usadas por los enemigos y TA-2.

Pistola láser: arma a distancia. Dispara una bala que viaja en línea recta a cierta velocidad hasta que impacta con el terreno o un objetivo, o viaja la distancia máxima permitida. Es el arma de TA-2 y de algunos enemigos.

Melee: arma cuerpo a cuerpo. Empuja a su objetivo si está a su alcance luego de unos momentos de carga. Impide el movimiento mientras carga. Debe recargarse durante unos momentos luego de usarse. Algunos enemigos tienen esta arma.

Cajas lanzables: arma a distancia. Empuja a su objetivo si lo golpea. Los enemigos sin arma a distancia pueden buscar, recoger y lanzar estas cajas. Usan el motor de físicas de Unity (son afectadas por la gravedad y por colisiones del entorno).



2.6. Requerimientos no funcionales

2.6.1. RNF1 - Diseño atractivo e intuitivo

El juego debe contar con una interfaz visual clara y estética, acompañada de una disposición lógica de los elementos interactivos, de forma que el jugador pueda comprender rápidamente cómo jugar sin necesidad de instrucciones extensas. La experiencia visual y la navegación dentro del juego deben seguir principios de diseño de interfaces centradas en el usuario, manteniendo la coherencia estética con la temática general del proyecto

2.6.2. RNF2 - Rendimiento aceptable

El juego debe ser capaz de ejecutarse de forma fluida en los equipos objetivo, manteniendo una tasa de cuadros por segundo (FPS) estable en situaciones normales de juego. No deben producirse congelamientos, caídas abruptas de rendimiento ni tiempos de respuesta elevados ante las acciones del jugador. El umbral mínimo considerado aceptable es de 30 FPS sostenidos, con un objetivo recomendado de 60 FPS.

2.6.3. RNF3 - Carga rápida

Los tiempos de carga del juego, tanto al iniciar como al realizar transiciones entre escenas o niveles, deben mantenerse dentro de márgenes razonables para no afectar la inmersión del jugador. Idealmente, el tiempo de carga total desde la apertura de la aplicación hasta la jugabilidad no debe superar los 10 segundos en equipos con las especificaciones mínimas.

2.6.4. RNF4 - Uso aceptable de memoria RAM

Durante la ejecución, el uso de memoria RAM del juego debe mantenerse dentro de un rango eficiente y controlado, optimizando el uso de texturas, modelos y efectos visuales. El consumo de memoria no debe generar problemas de estabilidad, especialmente en dispositivos que solo cuentan con los recursos mínimos. Se considera un máximo aceptable de 6 GB de RAM en uso activo.

2.6.5. RNF5 - Requerimientos de hardware mínimos y recomendados

Componentes	Mínimos	Recomendados
Sistema Operativo	Windows 10/11 (64 bits)	Windows 10/11 (64 bits)
Procesador	Intel Core i5-6400 / AMD Ryzen 3 1200	Intel Core i7-9700K / AMD Ryzen 5 5600X

Memoria RAM	6 GB	12 GB
Almacenamiento	15 GB en HDD	15 GB en SSD
Tarjeta gráfica (GPU)	Intel HD Graphics 4000 o superior	NVIDIA GTX 1050 / AMD RX 560 o superior
DirectX	Versión 11	Versión 12

3. Arquitectura de la solución

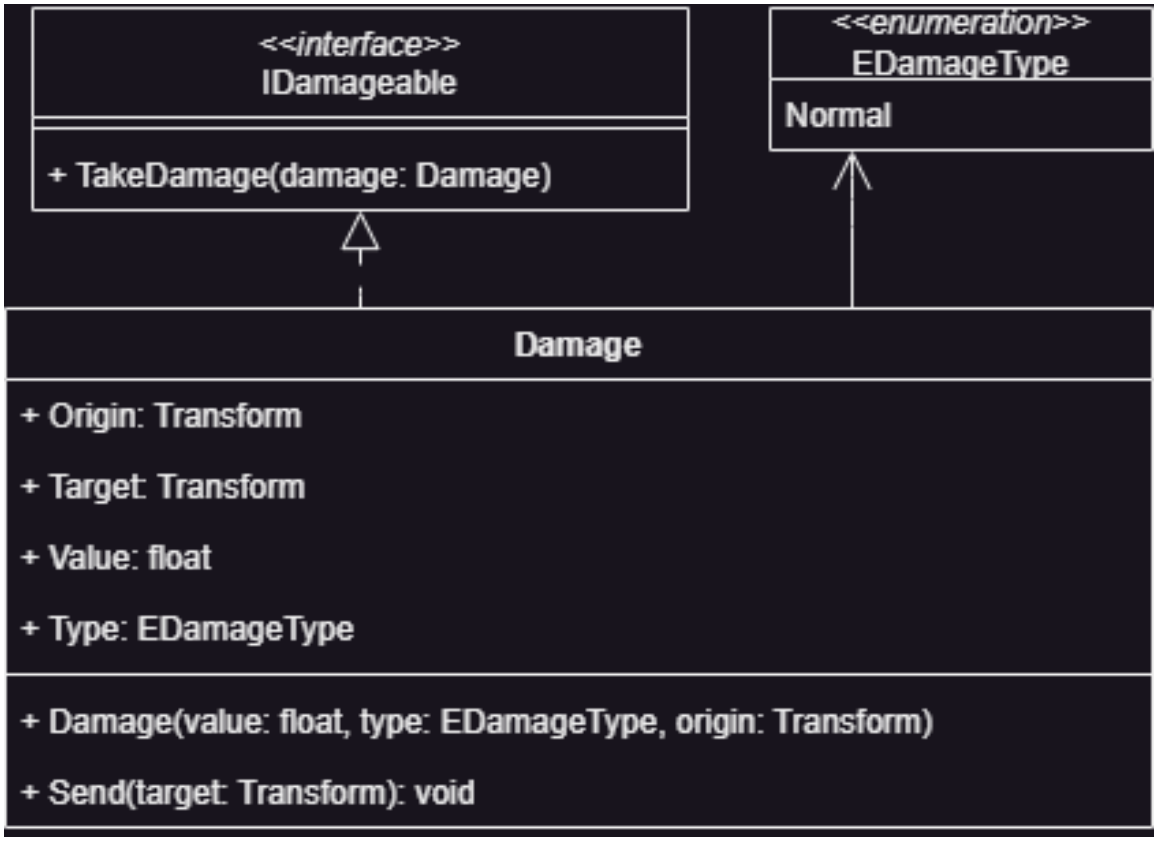
A diferencia de los sistemas empresariales tradicionales, que suelen estar compuestos por una arquitectura distribuida con componentes bien definidos como frontend, backend, bases de datos y servicios intermedios, este proyecto no responde a ese esquema. El videojuego fue concebido como una aplicación autónoma y autocontenida, sin requerimientos de conectividad a servidores externos ni integración con sistemas de terceros. Su arquitectura está centrada exclusivamente en la lógica del juego, y aunque cuenta con una funcionalidad de guardado de partida para conservar el progreso del jugador, no incorpora una base de datos relacional ni un backend persistente. Todo el procesamiento y almacenamiento se realiza de forma local.

3.1. Desarrollo C#

3.1.1. Estructura y módulos

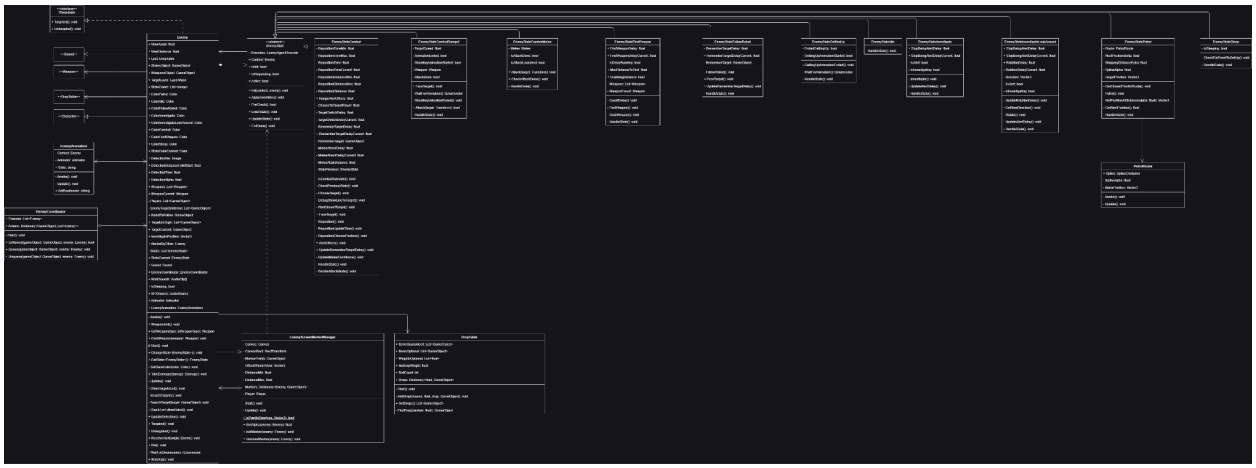
3.1.1.1. Damage

Contiene elementos usados para el cálculo y aplicación de daño. Creada en anticipación a futuros planes de agregar tipos de daño y efectos a los personajes.



3.1.1.2. Enemy

Clases relacionadas a los enemigos y su lógica, comportamiento, animaciones, y otros elementos relacionados.

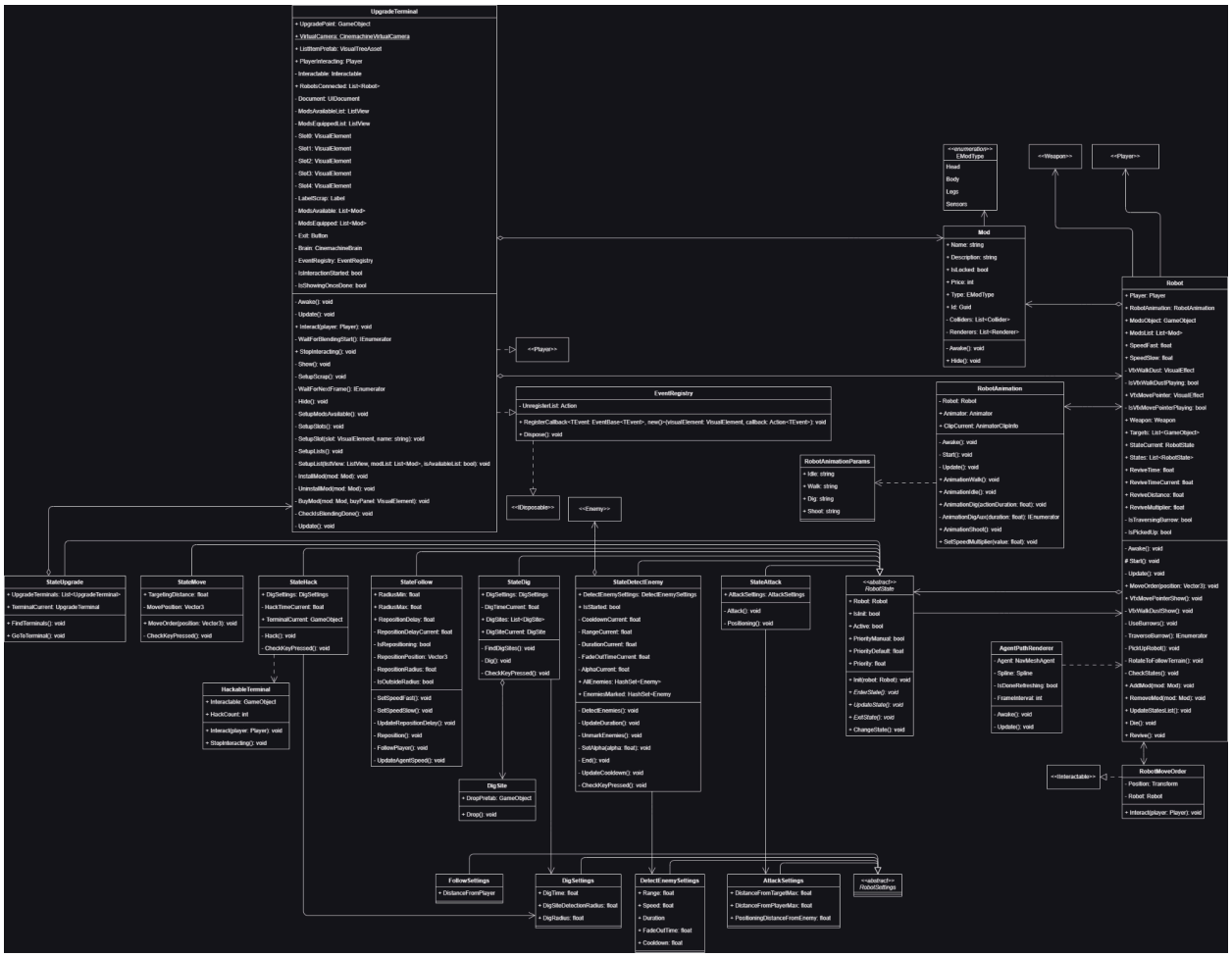


3.1.1.3. Menu

Clases relacionadas a la UI: HUD, menú principal y menú de pausa.

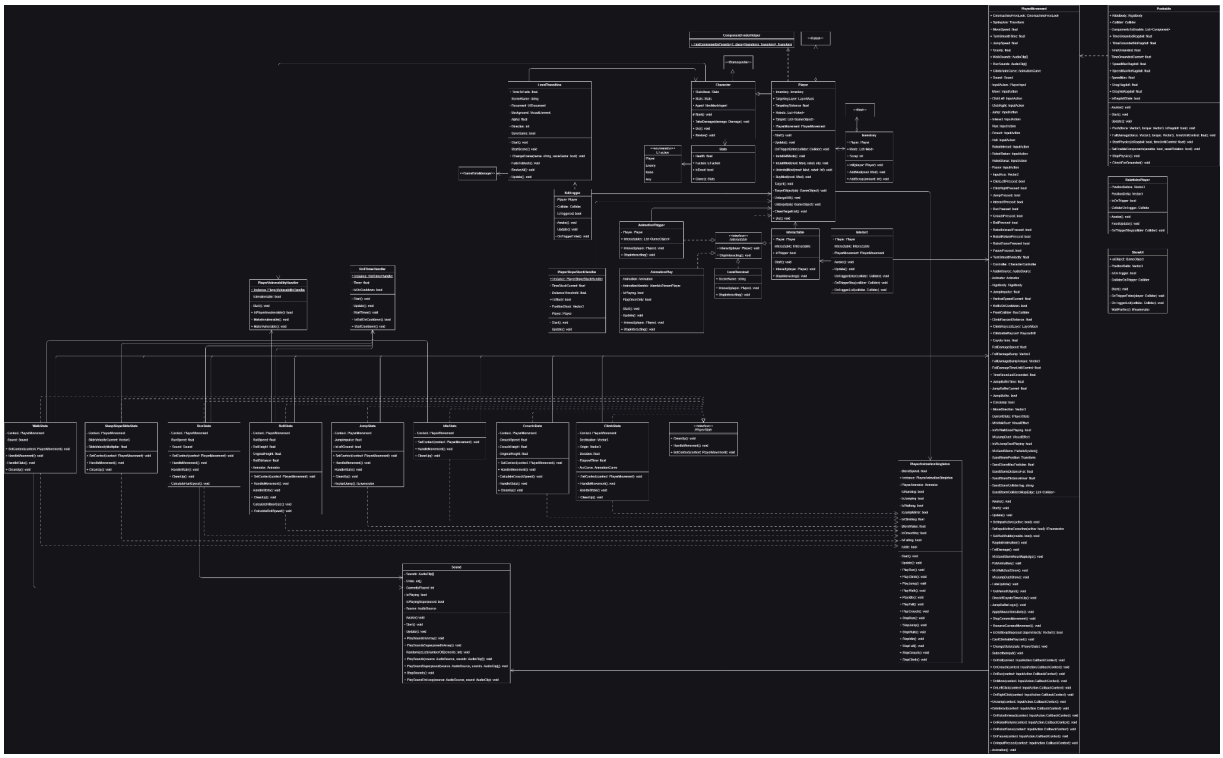
3.1.1.4. Robot

Clases relacionadas a TA-2 y su lógica, comportamiento, configuraciones, animaciones, mods, y otros elementos relacionados.



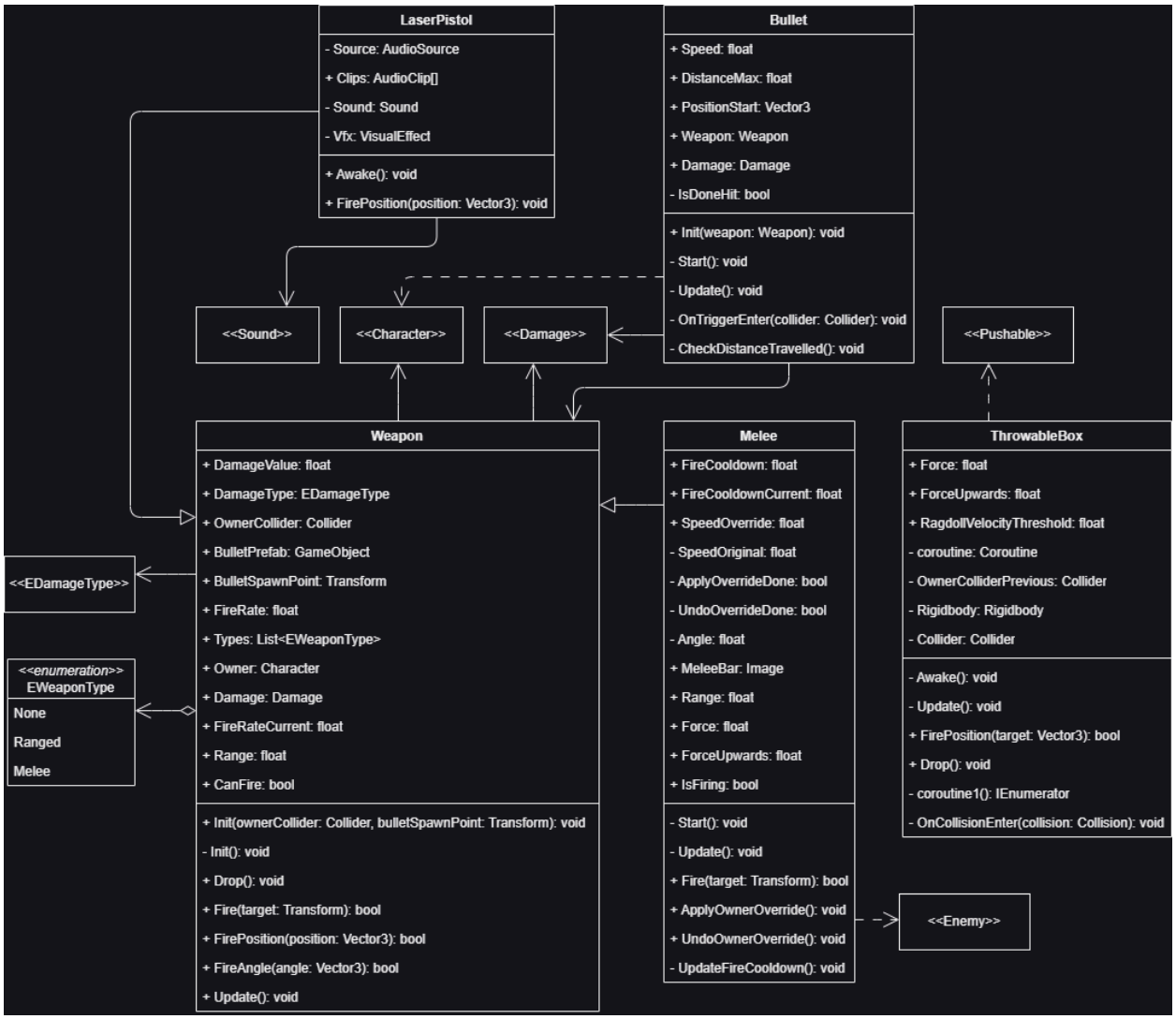
3.1.1.5. Player

Clases relacionadas a Gobi y su lógica, comportamiento, acciones, animaciones, interacciones, y otros elementos relacionados.



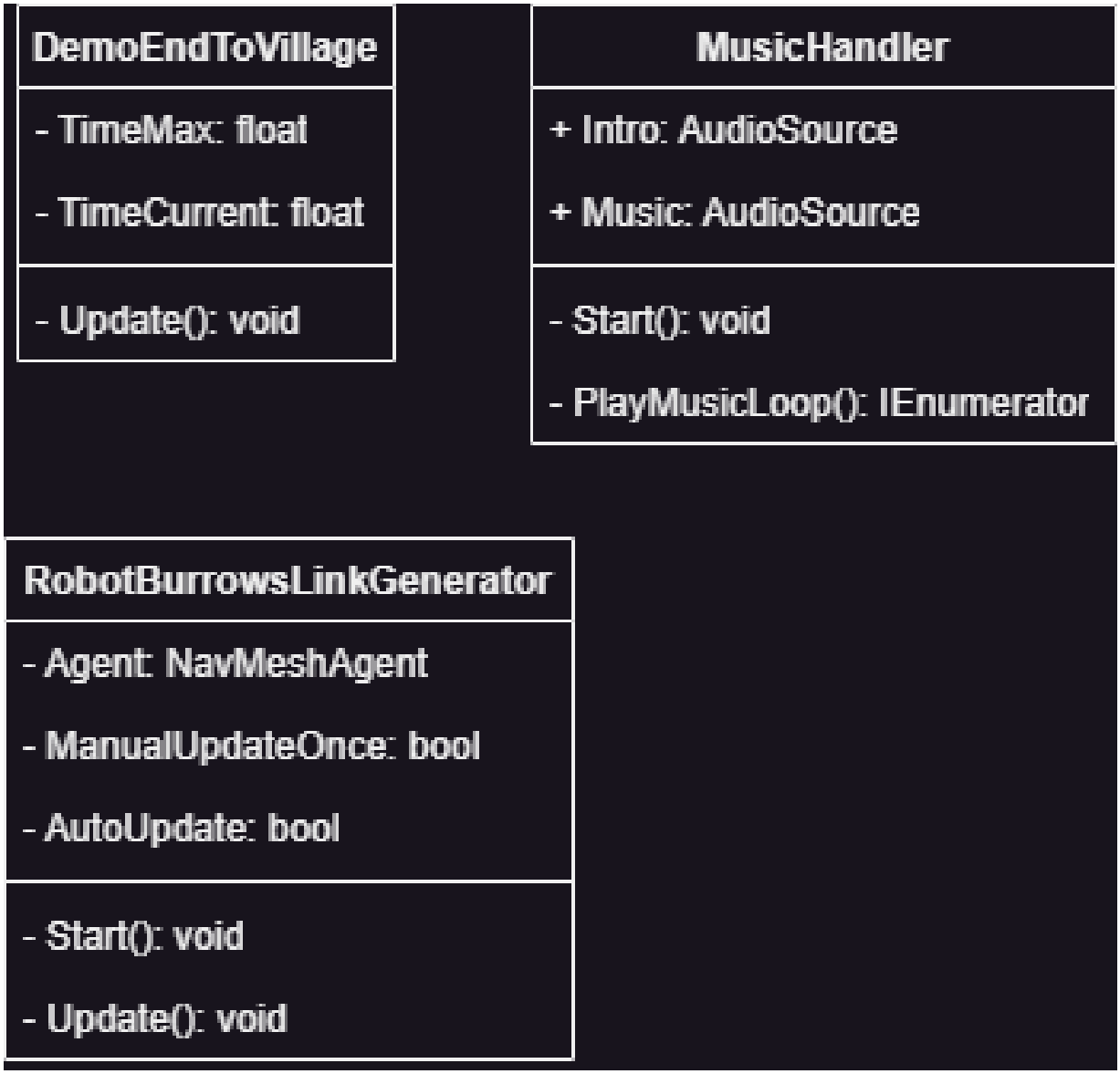
3.1.1.6. Weapon

Clases relacionadas a las armas y su lógica, tipo de arma, tipo de daño, tipo de proyectil, entre otros elementos relacionados.



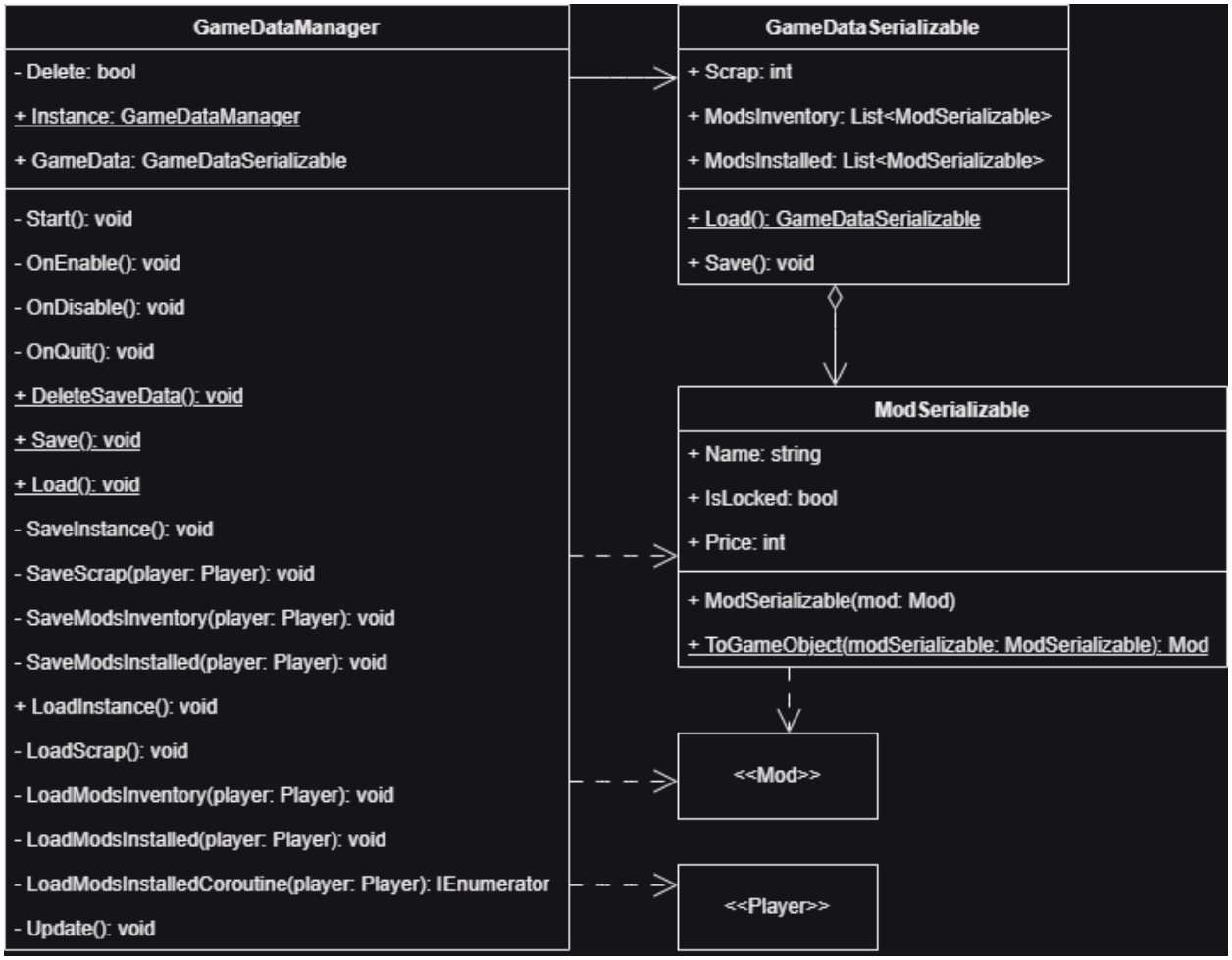
3.1.1.7. Utils

Clases de utilidades, como el handler para volver a la aldea al terminar el juego, el handler para reproducir música y sonidos, y el generador de caminos para la navegación de TA-2 en el terreno.



3.1.1.8. Persistence

Clases relacionadas a la persistencia de los datos, como clases serializables y deserializables para registrar eventos ocurridos en el juego, y una clase manager usada como interfaz para acceder a la lógica de guardado y cargado.



3.1.2. Clean Code

En el desarrollo de software, Clean Code (popularizado por Robert C. Martin en su libro *Clean Code: A Handbook of Agile Software Craftsmanship*) [4] se refiere a un conjunto de principios y buenas prácticas cuyo fin es que el código escrito sea claro, mantenible y eficiente. Esto se logra haciendo que el código sea fácil de entender, modificar y escalar con el tiempo. Dada la naturaleza del proyecto, el reducido tamaño del equipo (2 programadores), y su extendida duración, se pretendió cumplir con los lineamientos de Clean Code para evitar retrasos o errores a causa de no comprender el código escrito por otro integrante o uno mismo, un problema conocido en la industria, cuyo riesgo aumenta a medida que pasa el tiempo.

Se procuró hacer el código lo más legible posible para que fuera autoexplicativo. Aunque las ventajas que proporciona el uso de Clean Code son múltiples se decidió utilizarlo principalmente por los siguientes beneficios.

3.1.2.1. Beneficios

3.1.2.1.1. Legibilidad

Un proyecto con código legible mejora la mantenibilidad, reduce tiempos de comprensión, tanto de otros integrantes como del mismo autor, y facilita en general la creación de futuro código.

3.1.2.1.2. Mantenibilidad

Dada la naturaleza evolutiva del proyecto y los posibles cambios en los requerimientos basados en feedback de los potenciales usuarios, interesados, testers y opinión de personas con experiencia en el área, es crucial en el desarrollo que el código esté escrito de forma prolija, a fin de favorecer la mantenibilidad a futuro, ya que un código poco prolijo se torna difícil de entender y, por lo tanto, de modificar.

3.1.2.1.3. Profesionalismo

Importante de mantener en general, se nota su utilidad especialmente en caso de integrar nuevos colaboradores al proyecto. Además, ayuda a generar la costumbre de mantener la calidad.

3.1.2.1.4. Integración

Es una realidad que un proyecto de esta índole requiere muchas veces la integración con diversas herramientas de terceros, y facilita enormemente el trabajo que el código se encuentre organizado y lo más simple posible. Además esto proporciona una mayor seguridad de que un error en la integración de una nueva herramienta en una sección del código, no afecte a otras secciones del código.

3.1.2.2. Prácticas utilizadas

A continuación describiremos las principales prácticas de Clean Code utilizadas y proporcionaremos ejemplos demostrando su aplicación en el código.

3.1.2.2.1. Comentarios

Durante el desarrollo del proyecto se evitó el uso de comentarios innecesarios, por lo que los comentarios utilizados en el código solamente se realizaron en ocasiones donde era necesario añadir un mayor contexto a una función o método existente.

```
        }
        //allow jump if stuck in slope
        if (Context.JumpPressed && PlayerSlopeStuckHandler.Instance.IsStuck)
        {
            Context.VerticalSpeedCurrent = -2;
            IPlayerState jumpState = new JumpState();
            Context.ChangeState(jumpState);
        }
```

3.1.2.2.2. Nombres de variables y funciones descriptivos

A la hora de nombrar variables y funciones, se procuró en todo momento que tuvieran nombres autodescriptivos y lo más comprensibles posible, para que, de esta forma, no necesiten ningún tipo de comentario extra.

```
public class EnemyStateInvestigateLookAround : EnemyState
{
    1 reference
    [field: SerializeField] public float StopBeingAlertDelay { get; set; } = 6;
    3 references
    public float StopBeingAlertDelayCurrent { get; set; }
    2 references
    [field: SerializeField] public float RotationDelay { get; set; } = 2;
    4 references
    private float RotationDelayCurrent { get; set; }
    2 references
    private Vector3 Direction { get; set; }
    2 references
    private bool IsAlert { get; set; }
```

3.1.2.2.3. Funciones pequeñas y con una sola responsabilidad

Se procuró que cada función del juego tuviera una sola responsabilidad y que no se tornaran demasiado extensas, en los casos en que las funciones constaban de demasiadas líneas de código, se dividió en funciones más pequeñas con responsabilidades únicas, para que no hubieran funciones difíciles de mantener o modificar, o que, por ser demasiado grandes, facilitaran la duplicación de código.

```
1 reference
private void SaveInstance()
{
    // DeleteSaveData();
    Player player = FindObjectOfType<Player>();
    SaveScrap(player);
    SaveModsInventory(player);
    SaveModsInstalled(player);
    GameData.Save();
}
```

3.1.2.2.4. Evitar “números mágicos”

En el caso de los valores invariables utilizados, se intentó que estuvieran contenidos dentro de una constante con un nombre explicativo de ellas, para que, de esta forma se entendiera su significado y no cause que un lector externo, o el mismo autor del código, desconozca su origen en un futuro, en el caso que requiera revisión sobre el código que involucra esos valores.

```
1 reference
private void SetSpeedFast()
{
    Robot.Agent.speed = Robot.SpeedFast;
}

1 reference
private void SetSpeedSlow()
{
    Robot.Agent.speed = Robot.SpeedSlow;
}
```

3.1.2.2.5. Evitar duplicación de código

En ocasiones se notó que código resultaba necesario en múltiples sitios, por lo que en lugar de copiar y pegarlo en los lugares donde se necesitara, se consolidó en funciones reutilizables, reduciéndolo y haciéndolo más legible, además de ahorrar tiempo, al utilizar llamadas a las funciones en lugar de reprogramar una función ya existente en múltiples sitios.

```
2 references
void SetupSlots()
{
    SetupSlot(Slot0, "Attack");
    SetupSlot(Slot1, "Move");
    SetupSlot(Slot2, "Dig");
    SetupSlot(Slot3, "Hack");
    SetupSlot(Slot4, "Detect");
    // SetupSlot(SensorsSlot, EModType.Sensors); //TODO add 4th slot
}
5 references
void SetupSlot(VisualElement slot, string name)
{
```

3.1.2.2.6. Evitar funciones con demasiados parámetros

En los casos en que una función tomaba demasiados parámetros, si era posible, se creaban clases contenedoras para englobar los parámetros y evitar que se pasaran demasiados parámetros a una función.

```
public override void TakeDamage(Damage damage)
{
    if (StateCurrent is not EnemyStateCombat)
    {
        InvestigatePosition = damage.Origin.transform.position;
        ChangeState<EnemyStateInvestigate>();
    }
    if (!IsSleeping)
    {
        base.TakeDamage(damage);
    }
}
```

3.1.2.2.7. Manejo correcto de excepciones y errores

En cada oportunidad que se detectara que una excepción podría ocurrir, se capturó y lanzó un mensaje claro y concreto detallando la razón de su ocurrencia, evitando errores críticos en la ejecución, y manejando correctamente los errores que puedieran surgir.

```
void OnValidate()
{
    string prefixMsg = $"DropTable: {name}: ";
    string errorMsg = $"{prefixMsg}";
    string warningMsg = $"{prefixMsg}";
    bool error = ItemsOptional.Count != WeightsOptional.Count;
    bool warning = RollCount <= 0;
    errorMsg += $"Optional table lengths must match\n";
    warningMsg += $"Roll count <= 0. No optional loot will drop";
    if (!error) Debug.Log($"{prefixMsg}validation OK");
    else Debug.LogError(errorMsg);
    if (warning) Debug.LogWarning(warningMsg);
}
```

3.1.2.2.8. Uso adecuado de estructuras de control

Finalmente, en las ocasiones donde fue necesario el uso de estructuras de control, se procuró evitar, en la medida de lo posible, el uso de estructuras “if” anidadas, ya que esto puede aumentar la probabilidad de provocar errores lógicos. Se prefirió utilizar “early return” siempre que fuera posible, provocando que la lógica no se encuentre contenida dentro de bloques “if” que pudieran provocar los ya mencionados errores lógicos.

```
private void UpdateCooldown()
{
    if (CooldownCurrent >= DetectEnemySettings.Cooldown) return;
    CooldownCurrent += Time.deltaTime;
}
```

3.2. Decisiones de diseño

A lo largo del proyecto se tomaron diferentes decisiones de diseño con el fin corregir errores, prevenirlos, o facilitar futuros cambios a clases (o conjuntos de ellas) que lo requieran por su naturaleza. A continuación enumeraremos los más relevantes.

3.2.1. Utilización del patrón State

El patrón State es un patrón de diseño de comportamiento muy utilizado en la industria de videojuegos, ya que está estrechamente relacionado con el concepto de máquina de

estados finitos, donde en un momento dado, un programa puede encontrarse en un número finito de estados, variando su comportamiento según el estado en el que se encuentre, esto se ve con mucha frecuencia en los videojuegos donde los personajes, objetos o cualquier componente del juego, puede cambiar constantemente su comportamiento debido a estímulos o interacciones con los distintos elementos de su entorno o comandos del jugador.

Un juego en sí puede considerarse una máquina de estados desde cierta perspectiva, así que con frecuencia se consideraron apropiadas máquinas de estados para manejar de forma correcta, ordenada y eficiente los comportamientos de ciertos elementos.

En los sitios donde fue mayormente utilizado este patrón de diseño fue en las clases de comportamiento de los distintos personajes. Se utilizó en el caso de los enemigos, ya que podían realizar una amplia gama de acciones que determinaban comportamientos diferentes según los estímulos que recibieran. En este caso fue particularmente útil a la hora de agregar nuevos comportamientos a los enemigos, sin afectar los comportamientos ya añadidos en iteraciones anteriores, además de permitir manejar de manera adecuada las diferentes animaciones para cada acción y los sonidos emitidos al realizarlas.

En el caso de Gobi fue particularmente útil a la hora de manejar el movimiento, permitiendo que acciones como trepar que fueron añadidas más tarde en el desarrollo, no interfirieran con el resto de acciones agregadas previamente.

Finalmente, en el caso de TA-2 también se utilizó este patrón de diseño a la hora de implementar las diversas acciones a las que tiene acceso de acuerdo a los mods que tenga equipados en el momento, además, al ser acciones independientes entre sí, ya que cada acción es un estado diferente, ayudó a que, por ejemplo, la acción “Seguir” pueda habérsele añadido comportamiento extra sin detrimento o riesgo de afectar las otras habilidades de TA-2.

3.2.2. Utilización del patrón Singleton

En el caso de este patrón creacional, su utilidad y aplicabilidad viene derivada de la necesidad de garantizar la existencia de una instancia única de una clase, y proporcionar un único punto de acceso a dicha clase una vez creada.

En el caso de los videojuegos este patrón se torna particularmente útil en el caso de que existan instancias que deban ser accesibles por diversos elementos a lo largo del juego, pero que no se deban crear cada vez ya que todos los elementos deban acceder a esa misma instancia única.

Este patrón fue especialmente notable con el caso de las pistas de audio, donde se consideró apropiada la creación de un singleton que manejara la pista de audio base, porque en caso contrario, si se creara un nuevo objeto cada vez, las pistas podrían sobreponerse entre sí provocando una disonancia en los sonidos. Por otro lado, aunque no fue utilizado por el momento, fue considerado que resulta también particularmente útil para manejar el pausado y resumido de las pistas.

También se utilizó para manejar al personaje principal, ya que teníamos valores que debían persistirse al realizar transiciones entre las distintas escenas, como por ejemplo los contadores de scrap del inventario.

Por último se utilizó un Singleton con el fin de administrar el flujo principal del juego y las transiciones entre escenas.

3.2.3. Utilización de herencia

La herencia es un patrón básico y vital en la programación orientada a objetos. En este proyecto fue usada para introducir comportamiento diferente a versiones similares de un mismo objeto, es decir que comparten lógica o características similares, aunque no se utilizaron versiones demasiado variadas de un mismo objeto. En los casos en que se detectó que podrían existir variantes a futuro o que tenían al menos dos versiones que compartían un comportamiento base, se crearon las clases de forma que heredaran de un padre en común, por lo que clases como las armas de los personajes, que comparten el comportamiento de realizarle daño a un objeto de una facción enemiga, son clases hijas de un padre “Weapon” común, que contiene funciones comunes como por ejemplo “Fire” que corresponde al disparo del arma, con las clases hijas sobreescribiendo la implementación para ejecutar su funcionalidad individual. También se crearon las clases de personajes de modo que todos ellos (Enemigos, TA-2 y Gobi) hereden de un padre en común “Character” ya que funciones tales como recibir daño o morir son comunes a todos ellos.

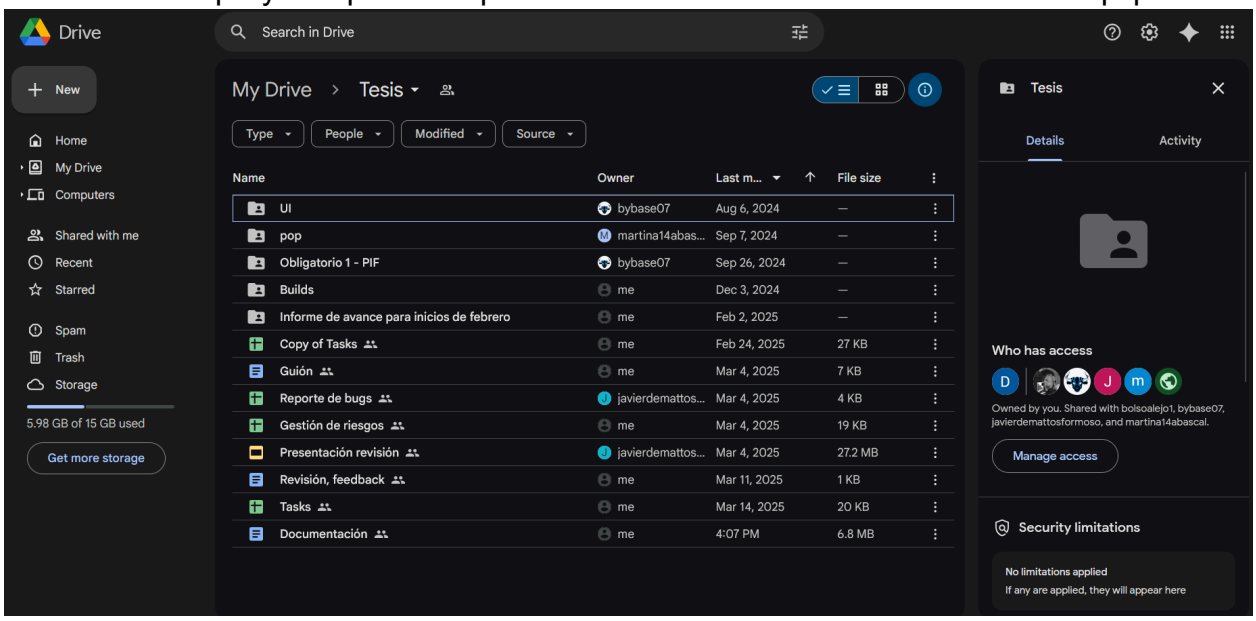
3.3. Herramientas y tecnologías de desarrollo

3.3.1. Herramientas y tecnologías

3.3.1.1. Documentación

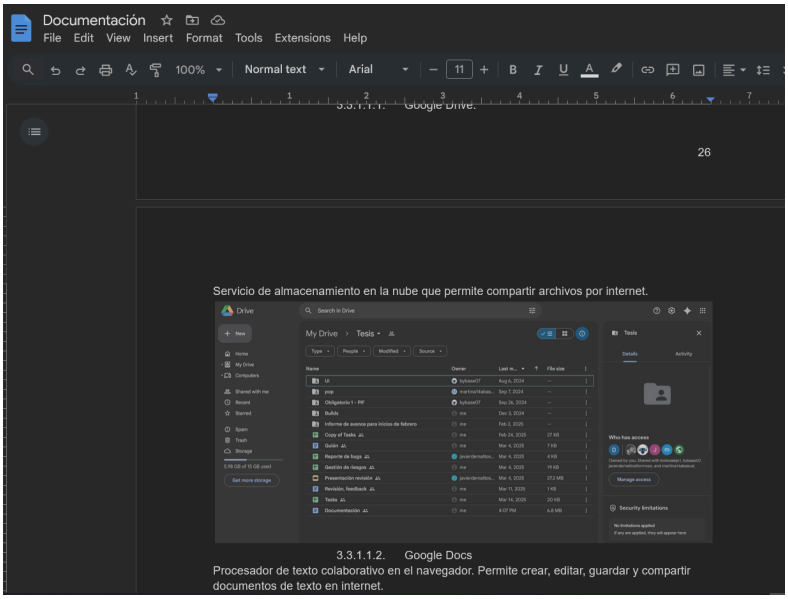
3.3.1.1.1. Google Drive

Servicio de almacenamiento en la nube que permite compartir archivos por internet. Usado en este proyecto para compartir e intercambiar documentos entre el equipo.



3.3.1.1.2. Google Docs

Procesador de texto colaborativo en el navegador. Permite crear, editar, guardar y compartir documentos de texto en internet. Usado en este proyecto para registrar documentación de forma colaborativa.

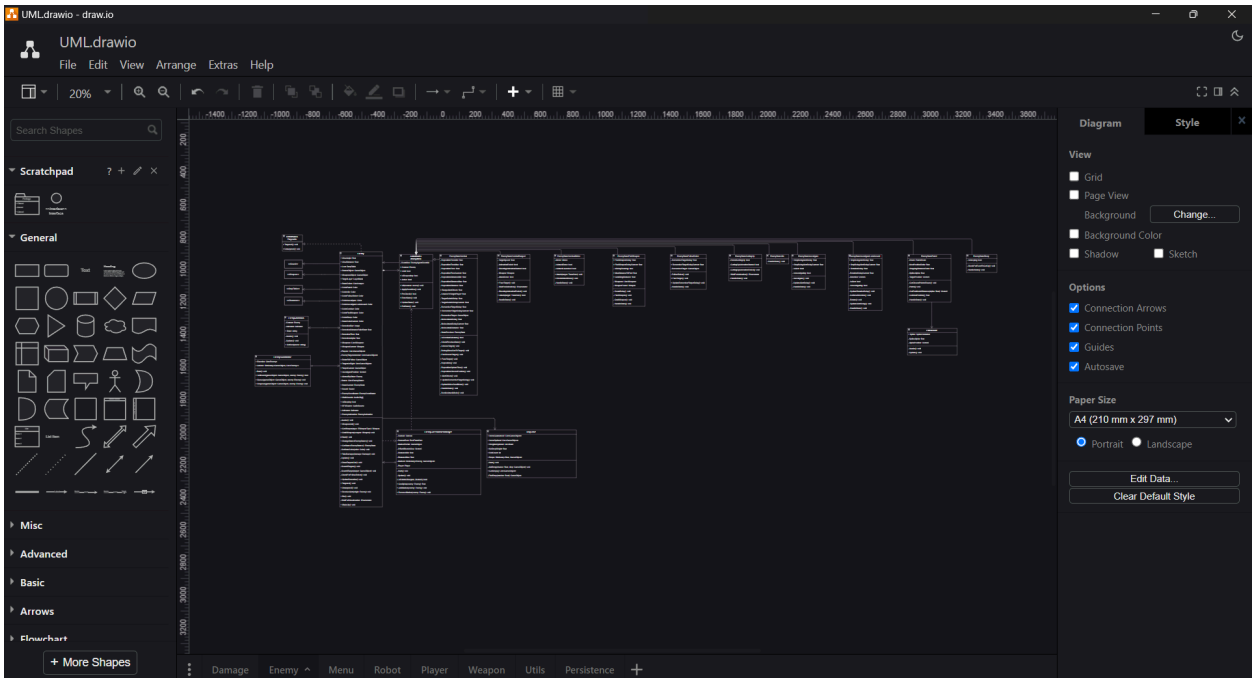


3.3.1.1.3. Google Sheets

Aplicación de hojas de cálculo colaborativa en el navegador. Permite crear, editar, guardar y compartir hojas de cálculo en internet. Usadas en este proyecto principalmente para registrar tareas y hacer cálculos, estimaciones y gráficas.

3.3.1.1.4. Draw.io

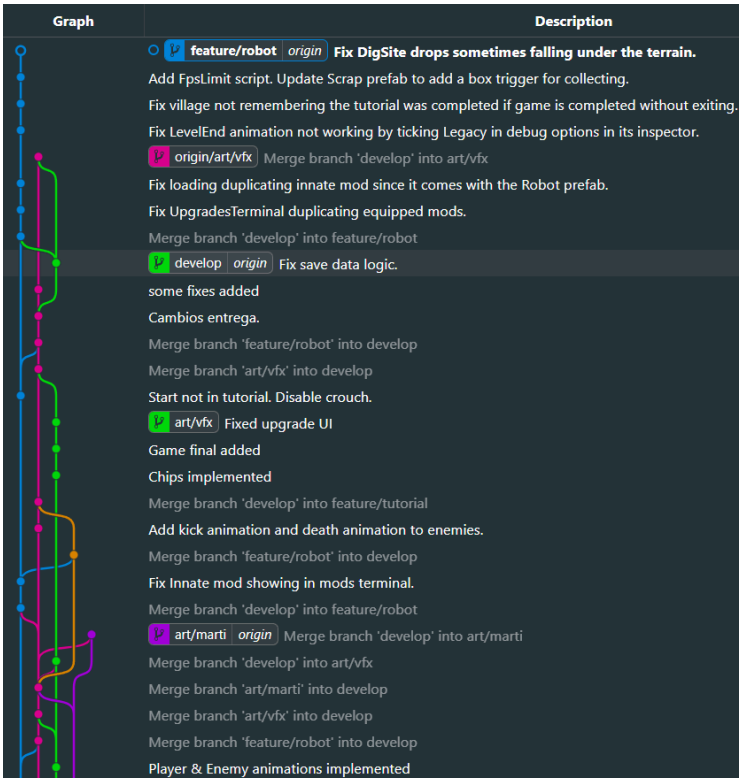
Aplicación colaborativa para construir diagramas. Usada en este proyecto para estructurar la lógica del juego y diagramar la estructura de clases del lenguaje de programación usado (C#).



3.3.1.2. Gestión del equipo

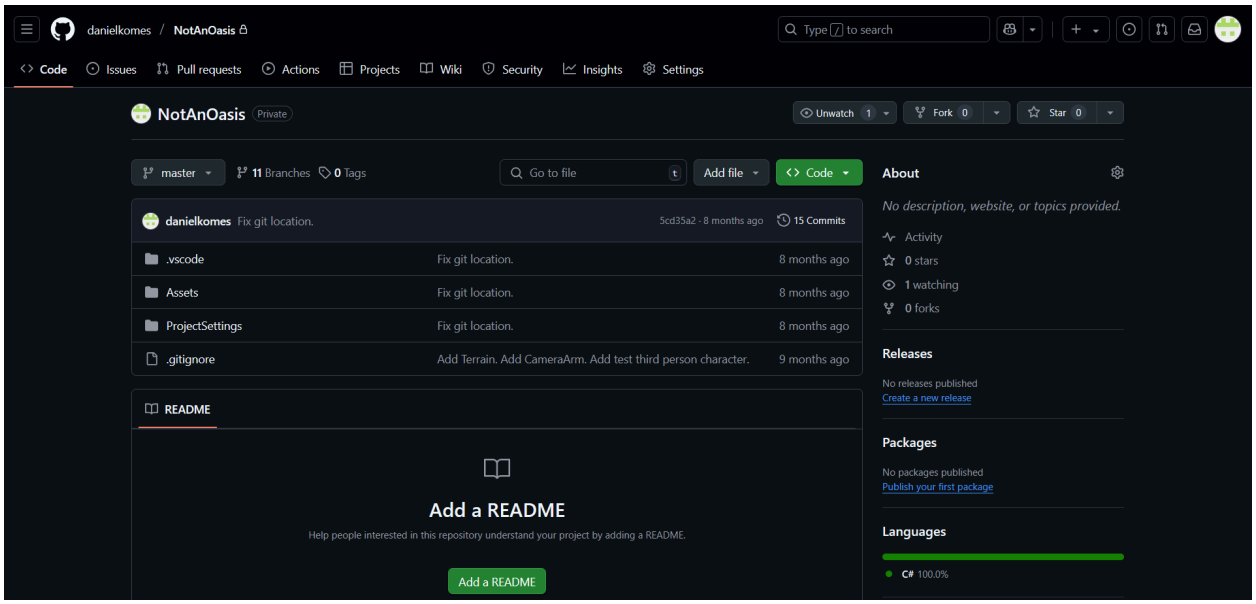
3.3.1.2.1. Git

Sistema de control de versiones de archivos. Usado en este proyecto para guardar e intercambiar versiones del proyecto entre los integrantes.



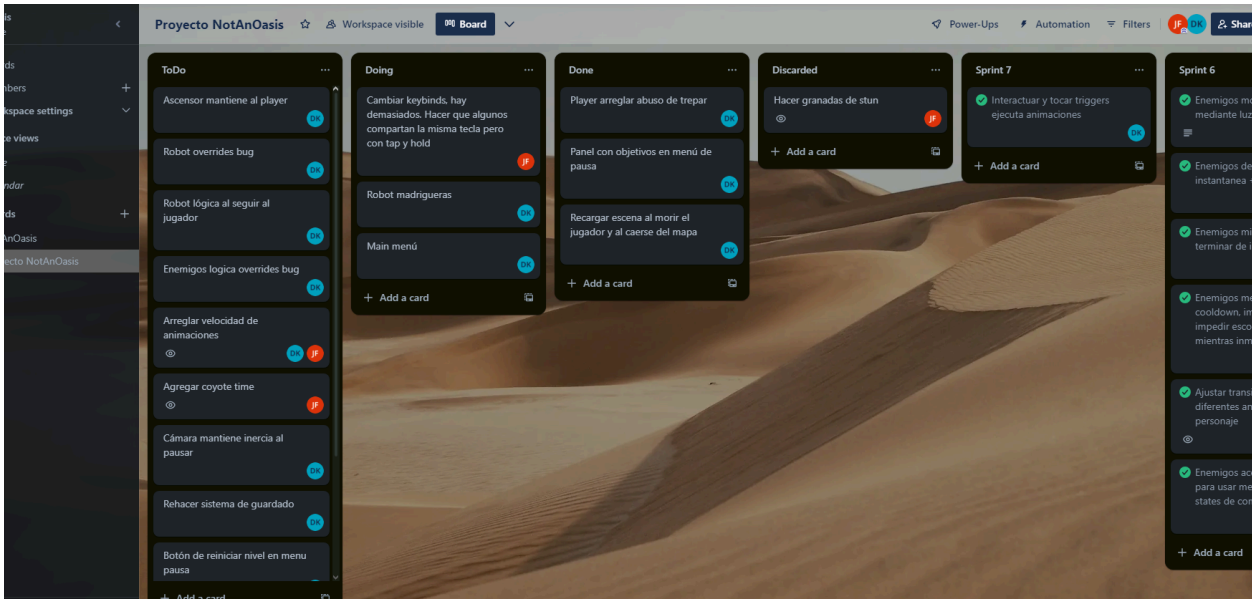
3.3.1.2.2. Github

Plataforma web que usa Git para compartir proyectos principalmente de programación. Usada en este proyecto para distribuir actualizaciones entre los integrantes.



3.3.1.2.3. Trello

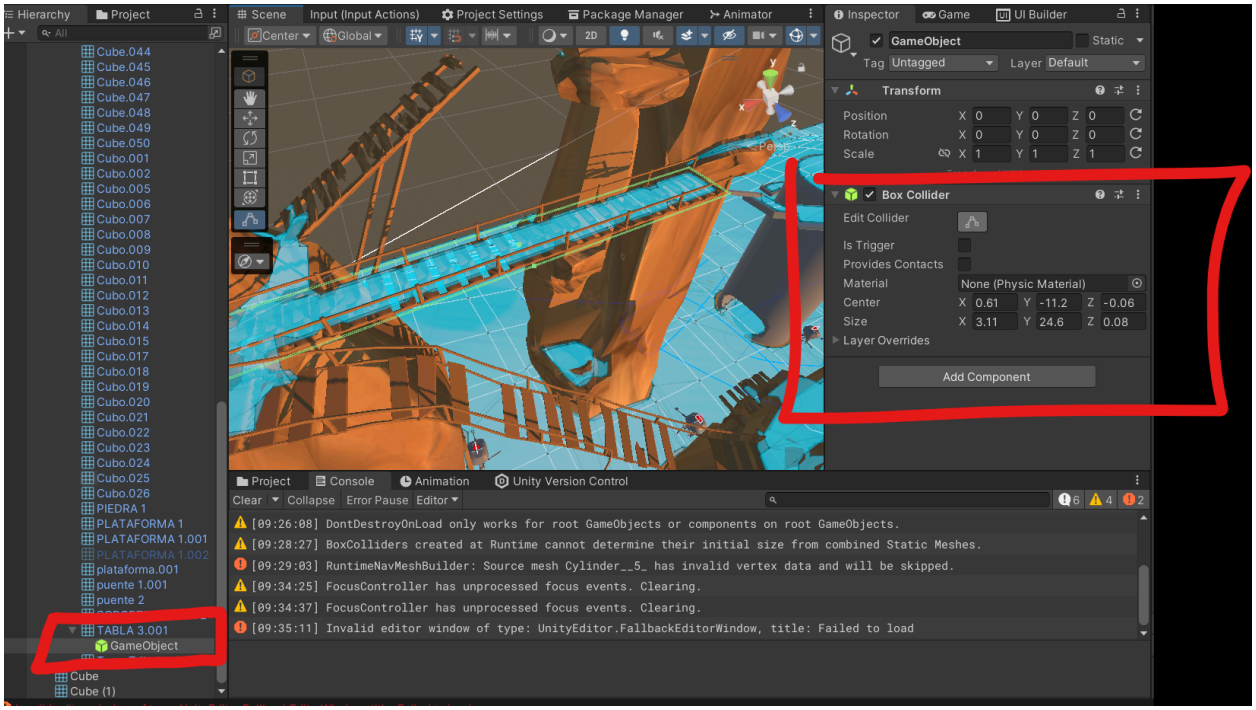
Herramienta de gestión de proyectos colaborativa en el navegador, con interfaz visual, con estilo Kanban, para organizar, planear y colaborar en proyectos usando tablas, listas y cartas, facilitando el seguimiento de progreso y la administración de flujo de trabajo.



3.3.1.3. Desarrollo del juego

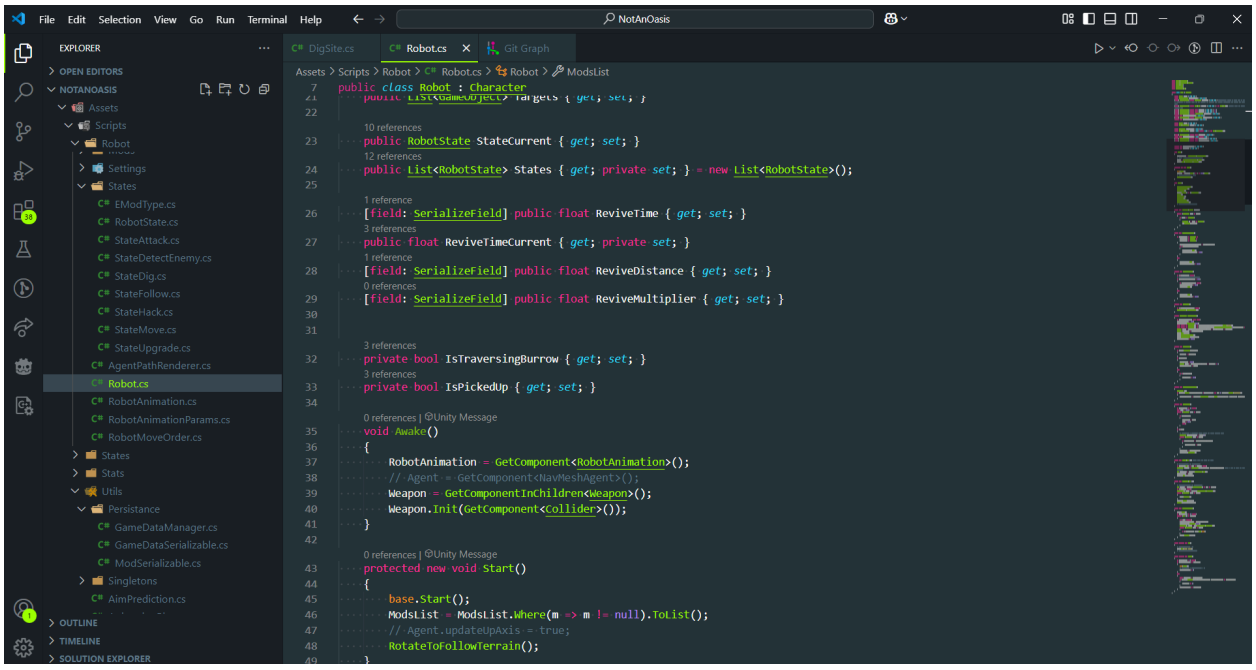
3.3.1.3.1. Unity 2022

Plataforma de desarrollo de aplicaciones interactivas en 2D y 3D multiplataformas, como juegos, simulaciones, entre otros, popular por su versatilidad y accesibilidad. Usado en este proyecto como motor principal en el diseño y desarrollo del juego.



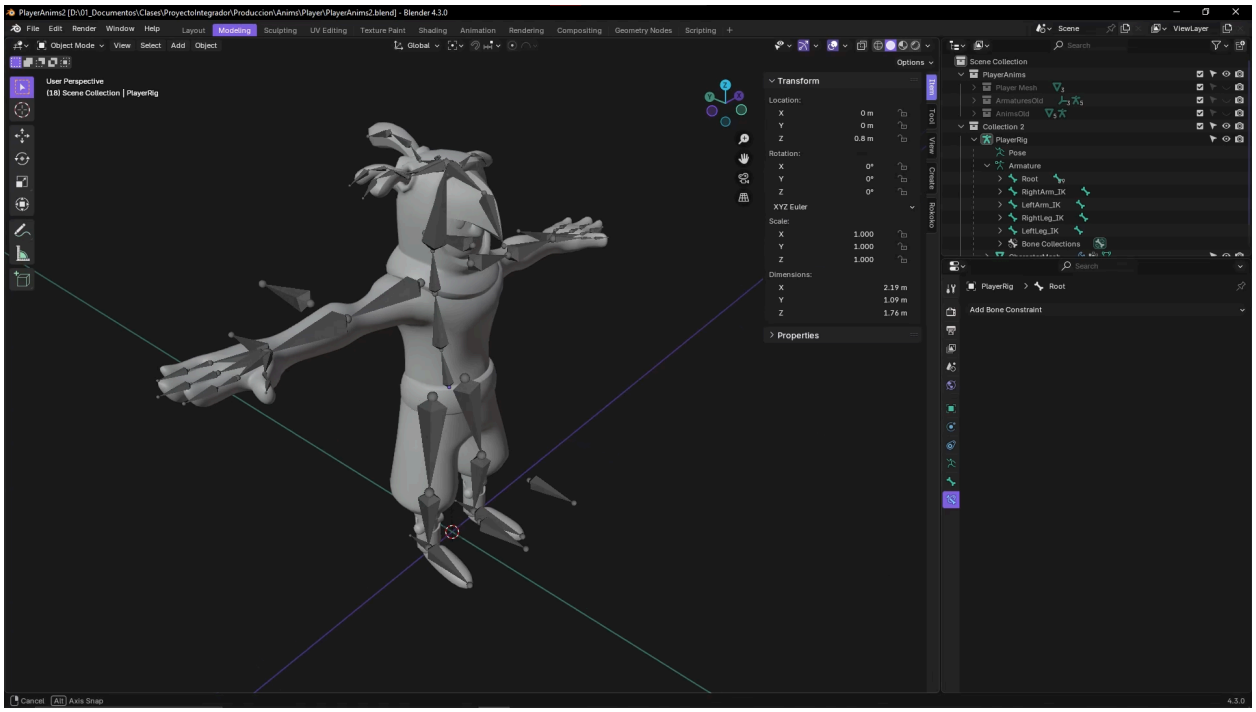
3.3.1.3.2. Visual Studio Code

Editor de código fuente ligero y extensible. Usado en este proyecto para desarrollar el código del juego.



3.3.1.3.3. Blender

Aplicación de creación de modelos 3D, animación, edición de video, entre otros. Usado en este proyecto para diseñar y animar los modelos 3D, y diseñar y modelar el ambiente de los niveles.

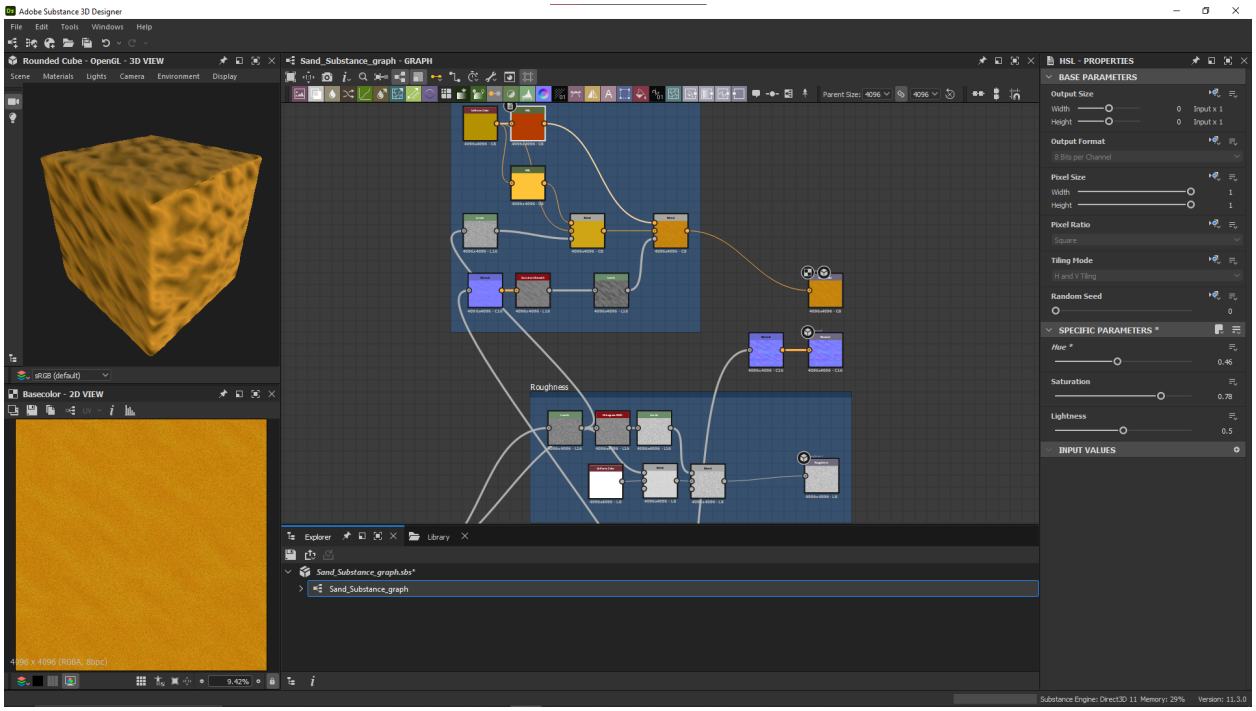


3.3.1.3.4. Substance Painter

Aplicación de pintado y texturizado de modelos 3D.

3.3.1.3.5. Substance Designer

Aplicación de diseño por nodos para crear materiales, texturas, patrones, entre otros archivos 3D, con gran control y flexibilidad.



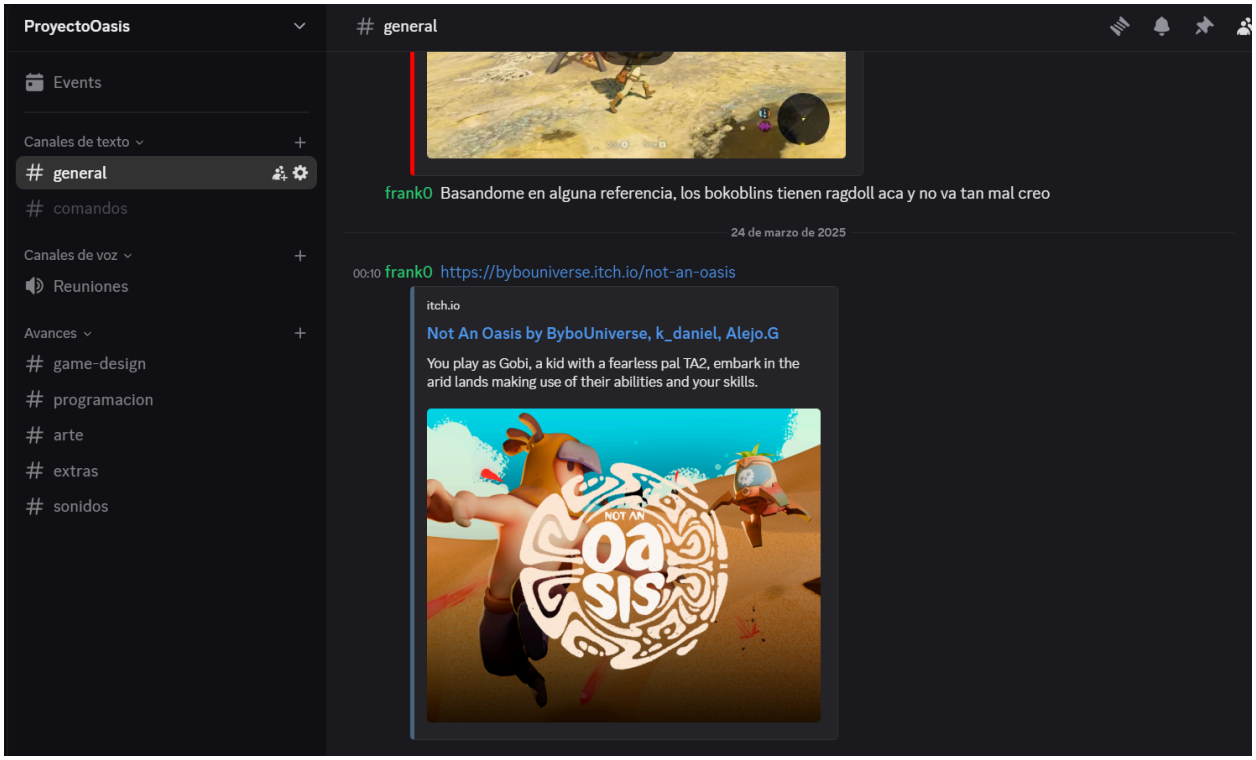
3.3.1.3.6. Photoshop

Aplicación de edición de imágenes y creación de arte digital.

3.3.1.4. Comunicación del equipo

3.3.1.4.1. Discord

Plataforma de comunicación por voz, video y texto organizada en comunidades. Usada en este proyecto para comunicación y organización del equipo, mostrar avances, compartir soluciones a problemas comunes con las herramientas, y coordinar reuniones entre el equipo.



3.3.1.4.2. Whatsapp

Plataforma de mensajería instantánea multiplataforma. Permite enviar texto, audio, imágenes, videos, documentos, ubicaciones, hacer llamadas de voz y video. Usado en este proyecto para coordinar temas rápidos y mostrar actualizaciones simples.

3.3.2. Justificación de herramientas y tecnologías

3.3.2.1. Documentación

3.3.2.1.1. Google Apps (Drive, Docs, Sheets)

Elegidas por ser gratuitas, estándar en la industria, permiten trabajar en equipo simultáneamente, y el equipo estaba familiarizado con ellas.

3.3.2.1.2. Draw.io

Elegida por ser gratuita, simple pero completa, rápida, disponible online y descargable, y el equipo estaba familiarizado con ella.

3.3.2.2. Gestión del equipo

3.3.2.2.1. Git

Elegida por ser el estándar de la industria, y el equipo estaba familiarizado con ella.

3.3.2.2.2. Trello

Elegida por ser gratuita, simple pero completa, permite trabajar en equipo simultáneamente, y el equipo estaba familiarizado con ella.

3.3.2.3. Desarrollo del juego

3.3.2.3.1. Unity 2022

Elegida por ser gratuita, estándar en la industria, completa, tiene requerimientos de hardware aceptables para el equipo, el rendimiento resultante era aceptable para el equipo, y el equipo estaba familiarizado con ella.

3.3.2.3.2. Blender

Elegida por ser gratuita, estándar en la industria, completa, tiene una comunidad grande con gran cantidad de tutoriales, y el equipo estaba familiarizado con ella.

3.3.2.3.3. Substance Painter

Elegida por tener un buen flujo de desarrollo y combinarse bien con las demás herramientas, personalizable, versátil, y el equipo estaba familiarizado con ella.

3.3.2.3.4. Substance Designer

Elegida por permitir buen control sobre el desarrollo del diseño de los personajes y el entorno, y el equipo estaba familiarizado con ella.

3.3.2.3.5. Photoshop

Elegida por ser estándar en la industria, completa, y el equipo estaba familiarizado con ella.

3.3.2.4. Comunicación del equipo

3.3.2.4.1. Whatsapp

Elegida por ser gratuita, fácil de usar, y el equipo estaba familiarizado con ella.

3.3.2.4.2. Discord

Elegida por ser gratuita, fácil de usar, personalizable, con opciones de organización aceptables para el proyecto, y el equipo estaba familiarizado con ella.

4. Jugabilidad

4.1. Introducción

En un videojuego, la jugabilidad es uno de los aspectos más importantes, ya que determina cómo se desarrolla la interacción entre el jugador y el entorno del juego.

4.2. Atributos de jugabilidad

Estos atributos se tuvieron en cuenta al implementar las funcionalidades en el proyecto, permitiendo evaluarlas según criterios más tangibles que meras suposiciones de lo que hace divertido al videojuego. Los atributos que establecidos para caracterizar la jugabilidad son:

4.2.1. Satisfacción

Agrado o complacencia del jugador ante el videojuego completo o en algunos aspectos concretos de éste, como mecánicas, gráficos, sistema interactivo, historia, etc.

4.2.2. Aprendizaje

Facilidad para comprender y dominar el sistema y la mecánica del videojuego, es decir, los conceptos definidos en el Gameplay/Game Mechanic del juego: objetivos, reglas y formas de interactuar con el videojuego.

4.2.3. Efectividad

Tiempo y recursos necesarios para ofrecer diversión al jugador mientras éste logra los objetivos propuestos en el videojuego y alcanza la meta final de éste.

4.2.4. Inmersión

Capacidad para creerse lo que se juega e integrarse en el mundo virtual mostrado en el juego.

4.2.5. Motivación

Característica del videojuego que mueve a la persona a realizar determinadas acciones y persistir en ellas para conseguir sus objetivos.

4.2.6. Emoción

Impulso involuntario originado como respuesta a los estímulos del videojuego que induce sentimientos y que desencadena conductas de reacción automática.

4.2.7. Socialización

Atributos y elementos del juego que fomentan el factor social o la experiencia en grupo, lo que provoca apreciar el videojuego de distinta manera, gracias a las relaciones que se entablan con otros jugadores o con otros personajes y que complementan las acciones a realizar y los retos a resolver en el juego.

4.3. Correlación entre atributos de jugabilidad

Para realizar un análisis de la correlación entre los atributos de jugabilidad, nos basamos en una Tesis Doctoral realizada en la Universidad de Granada por el ahora Doctor José Luis González Sánchez quien la realizó sobre Jugabilidad [1].

Como analizaba en su Tesis, con la tabla que se incluye a continuación extraída de esta, el favorecer una experiencia positiva en algún atributo de jugabilidad implica un aumento positivo en el resto de atributos. Esto implica, que a la vez que favorecemos o mejoramos un atributo de jugabilidad, el resto de atributos también presentan una mejoría, consiguiendo en consecuencia una mejora a la experiencia global del jugador.

En la tabla que se presenta a continuación se expresa con un signo de “+” que el atributo presenta “un alto grado de” y el signo de “-” representa un “bajo grado” de un determinado atributo de la jugabilidad.

Atributos		Satisfacción	Aprendizaje	Eficiencia	Inmersión	Motivación	Emoción	Socialización
Satisfacción	+		+	+	+	+	+	+
	-		-	-	-	-	-	-
Aprendizaje	+	-		-	-	-	-	-
	-	+		+	+	+	+	+
Efectividad	+	+	+		+	+	+	+
	-	-	-		-	-	-	-
Inmersión	+	+	+	+		+	+	+
	-	-	-	-		-	-	-
Motivación	+	+	+	+	+		+	+
	-	-	-	-	-		-	-
Emoción	+	+	+	+	+	+		+
	-	-	-	-	-	-		-
	-	-	-	-	-	-	-	

Al encontrarse tan estrechamente relacionados estos atributos, se vuelve crucial en el desarrollo conocer cada aspecto de la jugabilidad para poder monitorearlo adecuadamente.

4.4. Factores y atributos de calidad en videojuegos

4.4.1. Efectividad

Grado en que el jugador puede lograr las metas propuestas con precisión y completitud en el videojuego.

4.4.2. Eficiencia

Grado en que el jugador puede lograr las metas propuestas invirtiendo una cantidad adecuada de recursos en relación a la efectividad lograda en el juego. Este factor es determinado por la facilidad de aprendizaje y la inmersión.

4.4.3. Seguridad y prevención

Grado aceptable de riesgo para la salud del jugador, o sus datos, en el contexto del videojuego.

4.4.4. Satisfacción

Grado en el que los usuarios están satisfechos con el videojuego. Algunos atributos que se pueden considerar en este factor son: agrado, atracción, placer, confort, confiabilidad, motivación, emoción y sociabilización.

4.5. Facetas de la jugabilidad

Las facetas acotan el análisis de la jugabilidad a los distintos elementos de un videojuego, permitiendo hacer un análisis a nivel interactivo, intrínseco, entre otros.

4.5.1. Jugabilidad intrínseca

El videojuego como juego, la esencia que lo caracteriza: sus mecánicas, objetivos, rejugabilidad, recompensas y penalizaciones, balance, dificultad, libertad de acciones y estrategias, entre otros.

4.5.2. Jugabilidad mecánica

El videojuego como software, sus elementos y módulos: el motor y su efectividad, la correctitud de la apariencia visual, las correcciones de errores ante acciones del jugador, instrucciones y ayudas dinámicas ante retos nuevos, la interacción con los elementos del mundo y sus reacciones, la inmersión del sistema de guardado, la posibilidad de elección de estrategias, la capacidad de la cámara de captar correctamente los eventos del juego, entre otros.

4.5.3. Jugabilidad interactiva

El videojuego como sistema interactivo, los elementos de interfaz entre el juego y el jugador: la inmersión del sistema de control, menús y diálogos, la forma de los

controles de seguir el estándar del género, la satisfacción y facilidad del aprendizaje de controles, la personalización de los controles, la forma del juego de mostrar el estado al jugador en todo momento, entre otros.

4.5.4. Jugabilidad artística

El videojuego como elemento artístico: los gráficos, sonidos, historia, la capacidad de los objetos en el juego de ser reconocidos y relacionados con objetos del mundo real por el jugador, el atractivo visual de los elementos, el atractivo de los sonidos, la capacidad del jugador de elegir la parte de la historia que desea descubrir, entre otros.

4.5.5. Jugabilidad intrapersonal

La visión personal del jugador sobre el videojuego al jugarlo, basado en sus experiencias pasadas: el tiempo invertido por el jugador concretamente en ciertas áreas, retos o metas, la voluntad del jugador por repetir ciertos retos, el deseo del jugador por conseguir ciertas recompensas, entre otros.

4.5.6. Jugabilidad interpersonal

Aspectos que aparecen al jugar acompañado por otros jugadores: la existencia de nuevos retos y reglas en modo multijugador, el agrado por los nuevos retos y reglas, la similitud general del flujo del juego con los nuevos retos y reglas, la capacidad de la historia por ser coherente y completa para todos los jugadores, la capacidad de los jugadores de compartir recursos, la existencia de variaciones en la gestión de recursos, la correcta implementación de mecanismos de comunicación, entre otros.

4.6. Métricas jugabilidad

4.6.1. Efectividad

4.6.1.1. Del objetivo

Se desea evaluar el porcentaje de jugadores que logran completar el juego.

Deseado: 75% o más

Aceptable: 50% - 75%

No deseado: 50% o menos

4.6.1.2. Movilidad

Se desea evaluar si el jugador logró dominar las mecánicas de movimiento de Gobi y TA-2.

Deseado: 3 - 4

Aceptable: 2

No deseado: 1

4.6.2. Eficiencia

4.6.2.1. Del objetivo

Si el jugador logró llegar al final del juego en un tiempo razonable o cuánto tiempo tardó en rendirse. Esta métrica fue registrada personalmente por los miembros del equipo que supervisaron a los jugadores que probaron el juego.

Deseado: 0 - 5 minutos

Aceptable: 5 - 10 minutos

No deseado: 10 minutos o más

4.6.3. Seguridad y prevención

Se desea evaluar la cantidad y severidad de bugs encontrados y algún otro comportamiento inesperado.

Deseado: 25% o menos

Aceptable: 25% - 50%

No deseado: 50% o más

4.6.4. Satisfacción

4.6.4.1. Arte

Se desea evaluar el grado de satisfacción del aspecto visual del juego: gráficos, animaciones, modelos, texturas, efectos, etc.

Deseado: 4 - 5

Aceptable: 3

No deseado: 1 - 2

4.6.4.2. Movimiento

Se desea evaluar el grado de satisfacción de las mecánicas de movimiento: correr, saltar, rodar, trepar, de Gobi.

Deseado: 4 - 5

Aceptable: 3

No deseado: 1 - 2

4.6.4.3. Interacción

Se desea evaluar el grado de satisfacción de las mecánicas de interacción: hackear, cambiar mods, cavar, dar órdenes a TA-2.

Deseado: 4 - 5

Aceptable: 3

No deseado: 1 - 2

4.6.4.4. Sonido

Se desea evaluar el grado de satisfacción del aspecto de sonido: efectos, música, inmersión, coordinación, etc.

Deseado: 4 - 5

Aceptable: 3

No deseado: 1 - 2

4.6.4.5. Diseño de nivel

Se desea evaluar el grado de satisfacción del diseño del nivel: posicionamiento de obstáculos, posicionamiento de enemigos, recorrido esperado, interacciones necesarias, capacidad de explorar distintas rutas, etc.

Deseado: 4 - 5

Aceptable: 3

No deseado: 1 - 2

4.6.4.6. Dificultad

Se desea evaluar el grado de satisfacción de la dificultad: balance de recompensas, dificultad del combate, dificultad de efectuar mecánicas, penalización por errores, etc.

Deseado: 4 - 5

Aceptable: 3

No deseado: 1 - 2

4.6.4.7. Manejo de TA-2

Se desea evaluar el grado de satisfacción del manejo de TA-2: dar órdenes, percepción de tiempo de reacción, percepción de utilidad de TA-2, etc.

Deseado: 4 - 5

Aceptable: 3

No deseado: 1 - 2

4.6.4.8. Combate

Se desea evaluar el grado de satisfacción del combate: comprensión de comportamiento de enemigos, reacción ante distintos ataques, percepción de dificultad, diversión, etc.

Deseado: 4 - 5

Aceptable: 3

No deseado: 1 - 2

4.6.4.9. General

Se desea evaluar el grado de satisfacción general del juego. Con esta métrica se pretende analizar cualquier incongruencia entre las demás respuestas, y en caso de notar la ausencia de alguna métrica importante.

Deseado: 4 - 5

Aceptable: 3

No deseado: 1 - 2

4.7. Análisis de métricas jugabilidad

4.7.1. Efectividad

4.7.1.1. Del objetivo

Pregunta	¿Pudiste completar el juego?	
Respuesta	Jugadores	Valoración
Sí	4	(1 x 4) = 4
No	5	(0 x 5) = 0
Total	9	4
Resultado	44.44%	No deseado

4.7.1.2. Movilidad

4.7.1.2.1. De Gobi

Pregunta	¿Qué te pareció la movilidad de Gobi?	
Respuesta	Jugadores	Valoración
1	0	(1 x 0) = 0
2	0	(2 x 0) = 0
3	5	(3 x 5) = 15
4	4	(4 x 4) = 16
Total	9	31
Resultado	3.444444444	Deseado

4.7.1.2.2. De TA-2

Pregunta	¿Qué te pareció la movilidad de TA-2?	
Respuesta	Jugadores	Valoración
1	0	(1 x 0) = 0
2	3	(2 x 3) = 6
3	3	(3 x 3) = 9
4	3	(4 x 3) = 12
Total	9	27
Resultado	3	Deseado

4.7.2. Eficiencia

4.7.2.1. Del objetivo

	Medición de tiempo hasta completar el juego	
Minutos	Jugadores	Valoración
3	1	(3 x 1) = 3
6	2	(6 x 2) = 12
7	1	(7 x 1) = 7
Total	4	22
Resultado	5.5	Aceptable

4.7.3. Seguridad y prevención

Pregunta	¿Te encontraste con algún bug?	
Respuesta	Jugadores	Valoración
Sí	3	(1 x 3) = 3
No	6	(0 x 6) = 0
Total	9	3
Resultado	33.33%	Aceptable

4.7.4. Satisfacción

4.7.4.1. Arte

Pregunta	¿Cuánto te gustó el juego respecto al arte?	
Respuesta	Jugadores	Valoración
1	0	(1 x 0) = 0
2	0	(2 x 0) = 0
3	2	(3 x 2) = 6
4	1	(4 x 1) = 4
5	6	(5 x 6) = 30
Total	9	40
Resultado	4.444444444	Deseado

4.7.4.2. Movimiento

Pregunta	¿Cuánto te gustó el juego respecto al movimiento?	
Respuesta	Jugadores	Valoración
1	0	(1 x 0) = 0
2	1	(2 x 1) = 2
3	4	(3 x 4) = 12
4	1	(4 x 1) = 4
5	3	(5 x 3) = 15
Total	9	33
Resultado	3.666666667	Aceptable

4.7.4.3. Interacción

Pregunta	¿Cuánto te gustó el juego respecto la interacción?	
Respuesta	Jugadores	Valoración
1	0	(1 x 0) = 0
2	3	(2 x 3) = 6
3	2	(3 x 2) = 6
4	2	(4 x 2) = 8
5	2	(5 x 2) = 10
Total	9	30
Resultado	3.333333333	Aceptable

4.7.4.4. Sonido

Pregunta	¿Cuánto te gustó el juego respecto al sonido?	
Respuesta	Jugadores	Valoración
1	1	(1 x 1) = 1
2	0	(2 x 0) = 0
3	6	(3 x 6) = 18
4	1	(4 x 1) = 4
5	1	(5 x 1) = 5
Total	9	28
Resultado	3.111111111	Aceptable

4.7.4.5. Diseño del nivel

Pregunta	¿Cuánto te gustó el juego respecto al diseño del nivel?	
Respuesta	Jugadores	Valoración
1	0	(1 x 0) = 0
2	0	(2 x 0) = 0
3	4	(3 x 4) = 12
4	3	(4 x 3) = 12
5	2	(5 x 2) = 10
Total	9	34
Resultado	3.777777778	Aceptable

4.7.4.6. Dificultad

Pregunta	¿Cuánto te gustó el juego respecto a la dificultad?	
Respuesta	Jugadores	Valoración
1	0	(1 x 0) = 0
2	1	(2 x 1) = 2
3	3	(3 x 3) = 9
4	3	(4 x 3) = 12
5	2	(5 x 2) = 10
Total	9	33
Resultado	3.666666667	Aceptable

4.7.4.7. Manejo de TA-2

Pregunta	¿Cuánto te gustó el juego respecto al manejo de TA-2?	
Respuesta	Jugadores	Valoración
1	0	(1 x 0) = 0
2	2	(2 x 2) = 4
3	4	(3 x 4) = 12
4	2	(4 x 2) = 8
5	1	(5 x 1) = 5
Total	9	29
Resultado	3.222222222	Aceptable

4.7.4.8. Combate

Pregunta	¿Cuánto te gustó el juego respecto al combate?	
Respuesta	Jugadores	Valoración
1	1	(1 x 1) = 1
2	3	(2 x 3) = 6
3	2	(3 x 2) = 6
4	2	(4 x 2) = 8
5	1	(5 x 1) = 5
Total	9	26
Resultado	2.888888889	No deseado

4.7.4.9. General

Pregunta	¿Cuánto te gustó el juego en general?	
Respuesta	Jugadores	Valoración
1	0	(1 x 0) = 0
2	1	(2 x 1) = 2
3	1	(3 x 1) = 3
4	5	(4 x 5) = 20
5	2	(5 x 2) = 10
Total	9	35
Resultado	3.888888889	Aceptable

4.7.5. Conclusión

En general las métricas de satisfacción son aceptables, excepto la de combate, con un resultado de “No deseado”. Con esto se concluye que el diseño del combate requiere ser reevaluado.

Otra métrica que resultó en “No deseado” fue la de efectividad del objetivo. Menos personas de lo aceptable fueron capaces de terminar el juego. Con esto se concluye que el diseño general del juego debe ser reevaluado, tal vez agregando más ayudas al jugador, o reduciendo el tamaño y tiempo requeridos para completarlo.

5. Gestión de la calidad

5.1. Introducción

En esta sección se detallan las acciones y lineamientos adoptados por el equipo para asegurar la calidad tanto del proceso de desarrollo como del producto final. El Aseguramiento de la Calidad del Software (SQA, por sus siglas en inglés) desempeña un rol fundamental en proyectos interactivos como los videojuegos, donde la experiencia del usuario, el rendimiento y la estabilidad son factores clave.

Para alcanzar los niveles de calidad deseados, se establecieron estándares específicos que guiaron el trabajo del equipo durante las distintas etapas del proyecto. Además se aplicaron prácticas de evaluación y control con el objetivo de detectar temprano posibles desviaciones, errores o incumplimientos.

Esta sección también describe las herramientas utilizadas para medir la calidad y las estrategias implementadas para su mejora continua. Finalmente, se presentan los objetivos de calidad definidos al inicio del proyecto.

5.2. Objetivos de calidad del producto

- Garantizar la estabilidad del juego a lo largo del desarrollo, minimizando la presencia de errores críticos que afecten la jugabilidad.
- Validar la experiencia de los usuarios durante el desarrollo para que sea lo más fluida posible, esto logrado mediante uso de controles responsivos y mecánicas intuitivas.
- Asegurar la coherencia estética y narrativa del juego.
- Verificar el funcionamiento correcto de las mecánicas principales del juego.

5.3. Objetivos de calidad del proceso

- Asegurar una gestión efectiva y eficiente de errores mediante un sistema de reporte y seguimiento con prioridades asignadas.
- Reducir la tasa de retrabajo aplicando revisiones por pares y control de versiones.
- Promover la mejora continua trabajando en ciclos iterativos y evolutivos.
- Controlar los cambios implementados pudiendo gestionarlos adecuadamente, utilizando herramientas de control de cambios como Git.

5.4. Objetivos de calidad académicos

- Aplicar buenas prácticas y conocimientos adquiridos a lo largo de la carrera en un proyecto real.
- Integrar elementos ágiles y tradicionales de QA adaptados al contexto educativo.
- Realizar una documentación adecuada a los estándares establecidos por el centro educativo.
- Analizar y evaluar métricas obtenidas y tomar decisiones en base a sus resultados.
- Realizar una presentación clara y correctamente estructurada del proyecto.

5.5. Aseguramiento de la calidad (SQA)

Parte del aseguramiento de calidad consistió en registrar y evaluar el esfuerzo dedicado a ciertos tipos de tareas clave para entender cómo se desarrolló el proyecto y qué partes mejorar.

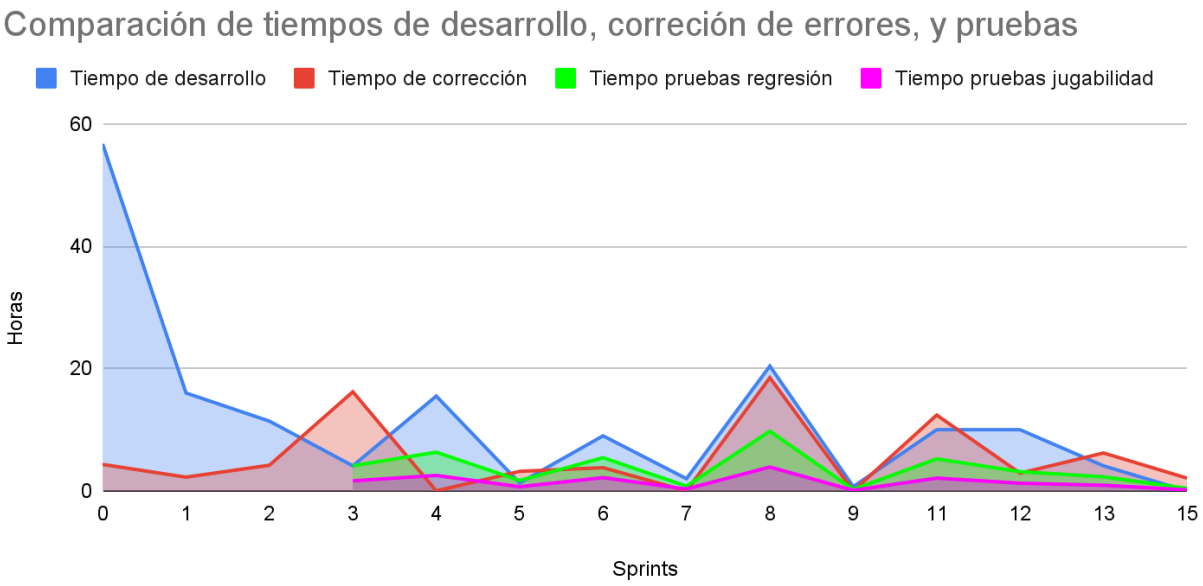
En esta tabla se muestra una comparación de tiempo invertido en tareas de ingeniería (no incluye las tareas de arte): tiempo invertido en desarrollo (de nuevas funcionalidades o modificaciones planeadas), tiempo invertido en corrección (de errores de funcionalidades ya existentes), tiempo invertido en pruebas de regresión (pruebas de otras funcionalidades diferentes a la implementada), y tiempo invertido en pruebas de jugabilidad (pruebas de integración para confirmar que el juego es completable).

El sprint 0 fue un período especial previo al inicio formal del proyecto y su materia. Tuvo una duración mucho más larga que los otros sprints y se realizaron tareas de preparación para cambiar el motor de Unreal a Unity.

Se puede apreciar cómo el tiempo de desarrollo es superado por el tiempo invertido en corrección de errores en el sprint 3. Esto impulsó la realización de pruebas de regresión y jugabilidad periódicas, para evitar acumular demasiadas funcionalidades sin probar, y que empiecen a dar errores al agregar y modificar otras.

El sprint 7, donde hay poca actividad, corresponde a fechas donde el foco se centró en el desarrollo de elementos de arte y planificación, y no tanto en desarrollo de código.

El sprint 9, donde no hay actividad, corresponde al período de receso de diciembre-enero.



5.6. Actividades principales de SQA

Para asegurar el cumplimiento de los estándares de calidad requeridos en un proyecto de este tipo (videojuego) se realizaron las siguientes actividades que responden a objetivos de calidad como se puede ver en la siguiente tabla, luego de esto se realiza una descripción a detalle de cada una de ellas:

Actividad	Descripción	Principales objetivos relacionados
Revisión por pares	Validación cruzada de código, diseño y contenido	Objetivos 1 y 2 del proceso.
Pruebas de usabilidad	Evaluación interna y externa de la experiencia de usuario	Objetivos 2 y 3 del producto
Pruebas de integración	Verificación de completitud del nivel ante nuevas funcionalidades	Objetivo 1 del producto Objetivo 2 del proceso
Pruebas de regresión	Corroborar el correcto funcionamiento de funcionalidades existentes luego del agregado de	Objetivos 1 y 4 del producto Objetivos 2 y 4 del

	nuevas funcionalidades	proceso
Pruebas de jugabilidad internas	Revisar internamente de forma periódica el *look and feel* del juego	Objetivos 1, 3 y 4 del producto
Gestión de errores	Uso de una tabla de gestión de incidencias.	Objetivo 1 del producto Objetivo 1 del proceso
Control de versiones	Gestionar el código y permitir la colaboración en simultáneo mediante el uso de Git	Objetivo 4 del proceso
Uso de prototipos	Validación temprana de ideas y ajustes sobre bosquejos, niveles de prueba, whiteboxing, etc..	Objetivo 3 del producto Objetivo 3 del proceso

Revisión por pares

Con el fin de mejorar la calidad de cada funcionalidad o elemento de arte introducido durante el desarrollo, se procuró que, antes de su introducción, estos debían ser revisados por al menos un integrante del equipo diferente al dueño del cambio a introducir; preferentemente, perteneciendo este a la misma área del dueño del cambio, siendo así los cambios introducidos por un artista, validados por el otro artista del grupo, y los cambios introducidos por un desarrollador, validados por el otro desarrollador del equipo. De esta forma, cada cambio o nueva funcionalidad, tenía una segunda opinión sobre sí, permitiendo la detección de errores por parte de otros miembros del equipo.

Esta práctica se realiza habitualmente en proyectos que utilizan metodologías ágiles como forma de trabajo ya que permite la detección temprana de errores.

Pruebas de usabilidad

Para evaluar la experiencia de usuario y la facilidad de uso de los distintos controles y claridad de los mismos, se realizaron pruebas de uso directamente con potenciales usuarios finales del juego, de esta forma se obtuvo valiosa retroalimentación acerca de distintos aspectos del juego, tanto al escuchar la opinión de los usuarios, como observar su desempeño y mediante la realización de una encuesta donde pudieron expresar sus opiniones acerca de puntos como facilidad de los controles, opinión

acerca del arte, satisfacción respecto al juego en general, al uso del personaje, al uso del robot, a la dificultad tanto mecánica como del combate con los enemigos, y los aspectos artísticos (visuales y de sonido), entre otros.

En base a estas opiniones y la observación realizada de la interacción de los usuarios con el juego, se implementaron cambios atacando los diversos aspectos a mejorar que consideramos relevantes.

Estas pruebas se centran en el usuario y permitieron identificar y corregir barreras de accesibilidad, comprensión y satisfacción con el estado del juego.

Pruebas de integración

Dado que el desarrollo se realizó de forma iterativa, cada nueva funcionalidad desarrollada debía integrarse correctamente con el resto del juego. Para asegurar que un nuevo cambio no comprometa la conclusión correcta de una incursión, o volviera el nivel imposible de superar de alguna manera, se procuró que, en caso de que el cambio afecte el diseño o recorrido del nivel, quien haya introducido el cambio debía realizar un recorrido completo del nivel antes de integrar el cambio al flujo principal de trabajo (realizar el merge a develop). De esta forma se evitó que un cambio o agregado de nuevo contenido provoque la introducción de errores que comprometa la jugabilidad.

Pruebas de regresión

A partir de cierto punto en el desarrollo, conforme el juego fue creciendo, se volvió necesario realizar pruebas de las funcionalidades previamente agregadas al introducir nuevas, ya que el agregado de una nueva funcionalidad podía provocar el mal funcionamiento de una previamente agregada. Aunque se procuró siempre el apoyo en patrones de diseño para asegurar la facilidad del agregado de nuevas mecánicas sin comprometer el correcto funcionamiento de las anteriores, se consideró necesario probar las ya existentes.

Esta práctica suele ser necesaria en proyectos que crecen iterativamente, como en un desarrollo ágil.

Pruebas de jugabilidad

Además de las pruebas de usabilidad en las que se les proporcionó el juego a testers ajenos al desarrollo, también se realizaron pruebas de jugabilidad internas, donde los miembros del equipo probaban el juego y daban sus opiniones acerca de aspectos a mejorar, detectaban errores y verificaban que la experiencia se sintiera fluida y fuera

satisfactoria, apuntando a verificar el balance, fluidez y satisfacción general del gameplay, además de la detección temprana de errores

Manejo de control de versiones y gestión de merge

Durante el desarrollo se usó Github para tener un control correcto de versiones y resolver o revertir cualquier problema detectado con facilidad. Además internamente y para mayor comodidad del equipo de arte, se acordó que el “merge” de una rama de arte al flujo principal de trabajo sea realizado siempre con un miembro del equipo de desarrollo presente, ya que al comienzo del desarrollo surgieron algunos problemas con el uso de la herramienta que provocaron retrabajo por parte de los artistas.

Esto es una parte crucial en el control de calidad del proceso de desarrollo ya que redujo considerablemente el riesgo de errores en el repositorio.

Gestión de errores y bugs

Se contaba durante el desarrollo con una lista de errores conocidos a corregir, y según la gravedad de los mismos, se les asignaba una prioridad, si esta era alta, el error se convertía en una tarea a realizar con la mayor antelación posible, si era media se tomaba entre las tareas a realizar pero cuando el desarrollador lo considerara conveniente, y si su prioridad era baja se postergaba hasta el momento en que se pudiera realizar, priorizando todo el resto de tareas y errores a corregir.

Adaptado mediante una tabla de gestión de incidencias nos permitió gestionar y responder de forma eficiente ante fallos críticos.

Uso de prototipos

Se usó la técnica whiteboxing para prototipar el nivel antes de la versión definitiva esto permitió validar la experiencia del juego antes de invertir tiempo y esfuerzo en el trabajo de arte final, lo cual es muy valorado en SQA, ya que asegura que no existan defectos que afecten negativamente la experiencia del usuario, y en la gestión del proyecto, puesto que previene la inversión de horas en retrabajo.

5.7. Definición de métricas

5.7.1. Introducción

Con la finalidad de garantizar el cumplimiento de los objetivos de calidad planteados y para medir las actividades de SQA realizadas, se definió y aplicó un conjunto de métricas para mejorar y asegurar la calidad tanto del producto como del proceso de

desarrollo. Las métricas de software constituyen medidas cuantitativas que permiten analizar atributos tanto del proceso de desarrollo como del producto final, facilitando la toma de decisiones y la mejora continua. Según Pressman en su libro *Software Engineering: A Practitioner's Approach* [3], las métricas permiten comprender, evaluar y controlar distintos aspectos del software, lo cual es fundamental en entornos donde la calidad es un factor crítico, como en el desarrollo de videojuegos.

Antes de comenzar, algunos puntos a considerar son los siguientes:

- El sprint 0 ocurrió previo al comienzo de la tesis, ya que el equipo determinó cambiar de motor de Unreal Engine a Unity y, por lo tanto, se debió volver a programar todo el juego en Unity (recordar que el proyecto es una continuación de un juego comenzado previamente).

- El sprint 7 comprende una fecha en la que se realizó una muestra en conjunto con el curso de animación de la universidad, por lo que ese sprint estuvo enfocado en tareas de arte principalmente.

- El sprint 9 corresponde a la etapa de final de semestre y comprende el beta testing, por lo que no hubo grandes cambios sobre el proyecto sino ajustes sobre lo que ya se tenía de cara a poner a punto el juego para el beta testing.

- El sprint 10 comprende el 25 y 1 de enero fechas en las cuales resultaba difícil coordinar para el equipo (más aún teniendo en cuenta que las reuniones con el equipo eran los miércoles y que ambas festividades ocurrieron un miércoles), por lo que el equipo consideró sensato realizar una reunión para analizar los resultados del beta testing pero se planeó un número muy reducido de tareas.

- A partir del sprint 13, el equipo de arte (Martina y Franco) dejaron de formar parte activa del proyecto, ya que realizaron su entrega final.

5.7.2. Métricas del producto

A continuación se presenta una tabla que contiene las métricas utilizadas para medir la calidad en el producto desarrollado, estas métricas fueron monitoreadas regularmente con el fin de encontrar problemas de calidad, rendimiento, estabilidad y jugabilidad/usabilidad.

Métrica	Descripción	Fórmula/Método	Valor esperado
Tasa de fallos críticos en pruebas de regresión	Porcentaje de errores de impacto alto introducidos al introducir nuevas	(Cantidad fallos críticos/total de pruebas de	< 10%

	funcionalidades	regresión) * 100	
Errores encontrados por usuarios	Cantidad de bugs no conocidos detectados por testers	Cantidad de errores únicos detectados solamente por testers	≤ 1 bug reportado por sesión
Rendimiento promedio (FPS)	Cuadros por segundo medidos durante el gameplay	FPS medido por unity	30 (> 60 caso ideal)
Estabilidad	Cantidad de sesiones de prueba con cierre inesperado del juego	(Cantidad de cierres inesperados + cantidad de fallos críticos/sesiones de prueba totales) * 100	< 1%
Satisfacción general del usuario	Opinión subjetiva de los testers mediante encuesta	Promedio escala Likert (1-5)	3.5/5

5.7.3. Métricas de proceso

La siguiente sección corresponde a las métricas del proceso, cuyo fin fue el de evaluar la eficiencia y efectividad del proceso de desarrollo, estas métricas fueron cruciales para conocer el estado del proceso en todo momento.

Métrica	Descripción	Fórmula/Método	Valor esperado
Número de bugs reportados por sprint	Total de errores detectados por iteración	Observación directa de la tendencia del gráfico	Disminución progresiva
Tiempo promedio de tareas por sprint	Tiempo medio para completar tareas por sprint	Suma de tiempos reales/cantidad de tareas	Estable entre sprints

Porcentaje de retrabajo vs desarrollo	Proporción del trabajo que debió rehacerse	(Horas de retrabajo/horas totales) * 100	< 10%
Velocidad del equipo	Qué tan rápido avanza el equipo por sprint	SP cerrados por sprint	Constante a lo largo del desarrollo o creciente
Porcentaje de tareas completadas en fecha	Cumplimiento correcto de milestones	(Tareas a tiempo/Tareas totales) * 100	≥ 90%

5.7.4. Métricas de proyecto

A continuación se presentan las métricas definidas para evaluar el proyecto, tanto desde una perspectiva de gestión como de desempeño general. Estas métricas permitieron monitorear correctamente el avance del proyecto, identificar desviaciones respecto a los plazos y objetivos iniciales, y tomar decisiones clave durante su desarrollo.

Métrica	Descripción	Fórmula/Método	Valor esperado
Precisión de estimaciones	Diferencia entre tiempo estimado y real	(Tiempo real/Tiempo estimado) * 100	80 - 120%
Avance del proyecto en porcentaje	Porcentaje de avance respecto al cronograma	(Tareas finalizadas / Tareas totales del backlog) * 100	≥ 90% al final del proyecto (con funcionalidades claves totalmente implementadas)
Desviación de cronograma	Porcentaje de desviación del cronograma	(Duración real - duración planificada)	±10% de diferencia máximo
Story Points por sprint	Cantidad de story points realizados por sprint por	SP realizados por cada miembro del equipo en cada	13 SP por sprint por cada miembro

	miembro del equipo	sprint	del equipo

5.8. Análisis y desarrollo de métricas

A continuación se presentan los distintos resultados obtenidos de las mediciones realizadas, en conjunto con el análisis de cada una de ellas, y las conclusiones y decisiones que tomó el equipo en base a los resultados obtenidos.

5.8.1. Resultado de métricas del producto

5.8.1.1. Tasa de fallos críticos en pruebas de regresión

A partir del sprint 3, debido al crecimiento del juego y para no afectar el funcionamiento de partes del proyecto que ya se encontraban funcionando correctamente al introducir nuevos cambios, se comenzaron a realizar pruebas de regresión para corroborar su funcionamiento correcto previo al agregado de nuevos cambios.

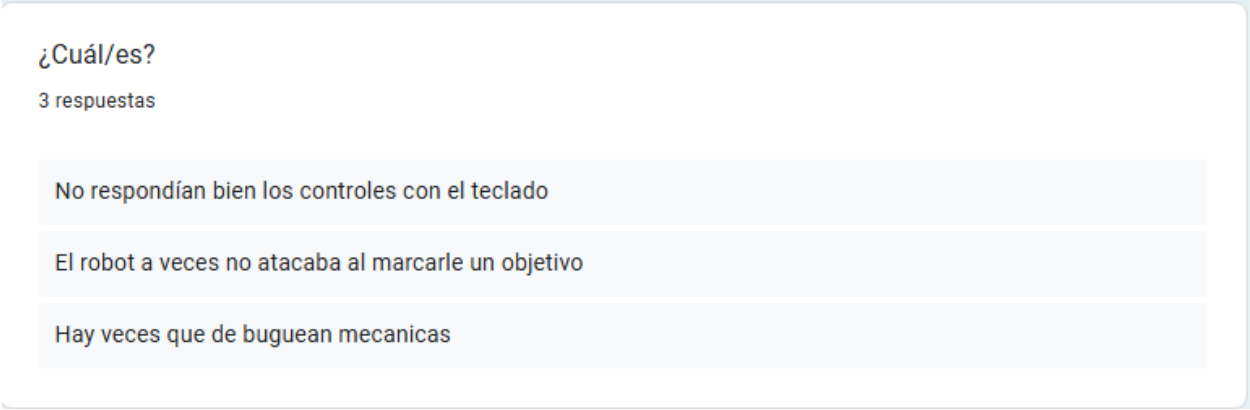
Para esta métrica se definió como un porcentaje aceptable un valor menor al 10% utilizando este cálculo (# fallos críticos/total de pruebas de regresión) * 100, donde fallos críticos correspondían a fallos que impidieran el correcto funcionamiento del juego o de funcionalidades ya implementadas, y el total de pruebas de regresión corresponde al total de pruebas de regresión realizadas al introducir nuevas funcionalidades. El valor del 10% se tomó como aceptable dado que un porcentaje superior podría ser indicador de que las clases del proyecto dependen demasiado entre sí y que, por lo tanto, el juego sería poco mantenible y extensible a futuro, ya que cualquier cambio incluido afectaría muchas otras clases y se tornaría difícil el agregar nuevos requerimientos.

Como se puede ver en el gráfico a continuación, ese porcentaje de fallos críticos encontrados tuvo un valor estimado del 4%, por lo que se mantuvo por debajo del índice esperado, los datos utilizados para el cálculo fueron un total de 24 pruebas de regresión a lo largo del proyecto con solo una de ellas conteniendo errores críticos. En el caso de error fue dado por desconocimiento de las clases sobre las que se trabajaba, por lo que el error, al ser analizado y resuelto se determinó que no fue por problemas respecto a la independencia de las clases, ya que fue dado por una introducción de un nuevo estado a los estados de los enemigos, por lo que estando dentro de los estándares deseados no se realizaron acciones al respecto.

5.8.1.2. Errores encontrados por usuarios

Esta sección corresponde a los errores encontrados por usuarios dentro de las instancias en que se les dio a probar el juego a personas externas al proyecto, como lo fue el beta testing de diciembre de 2024. En este caso obtuvimos 3 respuestas de casos de error cuando nuestro resultado esperado fue que los errores reportados fueran uno o ninguno, por lo que se analizaron los errores reportados para saber dónde ocurrían y poder actuar en consecuencia, de esa forma podríamos saber si la calidad del producto estaba siendo la esperada o no.

Los errores reportados se pueden ver en la siguiente imagen:



Respecto al primer error reportado por los beta testers, se determinó que no se trataba de un error en el juego, sino que estaba relacionado a una limitación del hardware utilizado, que se da en teclados antiguos o de oficina. En ellos, al presionar múltiples teclas de manera simultánea (por ejemplo varios comandos de movimiento, tecla de correr y tecla de salto), algunas de esas pulsaciones no son registradas correctamente debido al Key Rollover (KRO), esto varía según el teclado, en algunos casos solo permiten registrar 2 o 3 teclas a la vez, en caso de que ese límite se supere, las teclas que se presionen luego no son detectadas, provocando que le juego no responda.

Sobre el segundo bug, se determinó que tampoco era un bug sino que era falta de claridad, ya que a cierta distancia máxima entre el jugador y los enemigos, TA-2 deja de atacar, para esto se acordó trabajar en una mayor claridad en las acciones del robot.

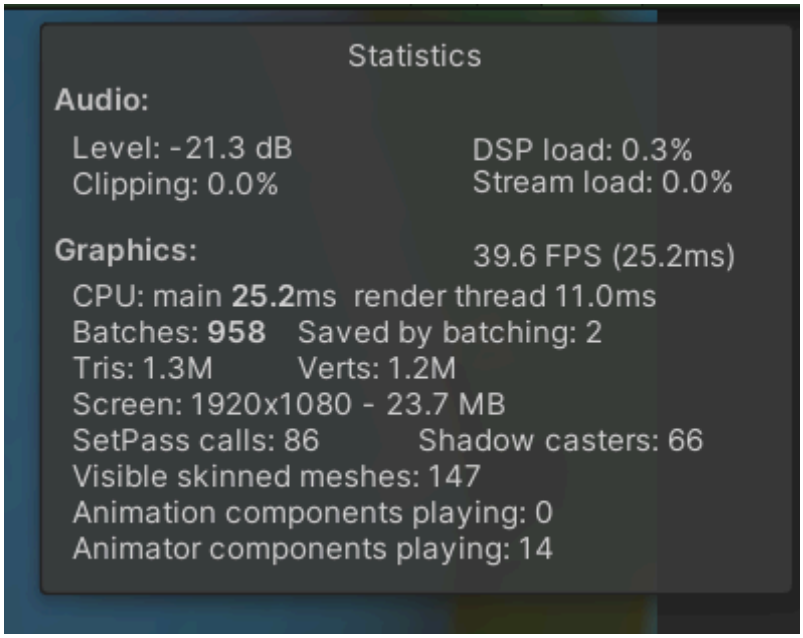
Finalmente, analizando el tercer error reportado, aunque fue poco detallado el reporte, se marcó una tarea para reevaluar las mecánicas y se planeó realizarles algunas mejoras, así como también un par de mejoras al terreno, ya que en ocasiones las mecánicas parecían estar fallando cuando en verdad el problema era el nivel en sí. Este error aún así si fue contado como tal, pues se realizaron tareas correctivas al respecto luego de realizar pruebas.

En conclusión, si bien el indicador se halló por encima del valor esperado, los errores reportados fueron catalogados como errores equivocadamente, por lo que el indicador dio dentro de los valores esperados, aún así, queda clara la importancia de brindar una buena experiencia de usuario y se observaron varias oportunidades de mejora sobre el nivel y las mecánicas implementadas.

5.8.1.3. Rendimiento promedio (FPS)

Al comienzo del proyecto se detectaron problemas de FPS en computadoras de la facultad, lo cual no ocurría en equipos más potentes. Visto que los FPS se encontraban por debajo de lo esperado se tomaron las medidas relacionadas al rendimiento detalladas en el plan de gestión de riesgos con el fin de solucionar estos problemas de rendimiento por debajo de lo esperado aplicando técnicas de occlusion culling y LODs.

Actualmente los valores se encuentran dentro de lo esperado en un equipo con los requerimientos mínimos como se puede ver en la captura que se encuentra a continuación.



5.8.1.4. Estabilidad

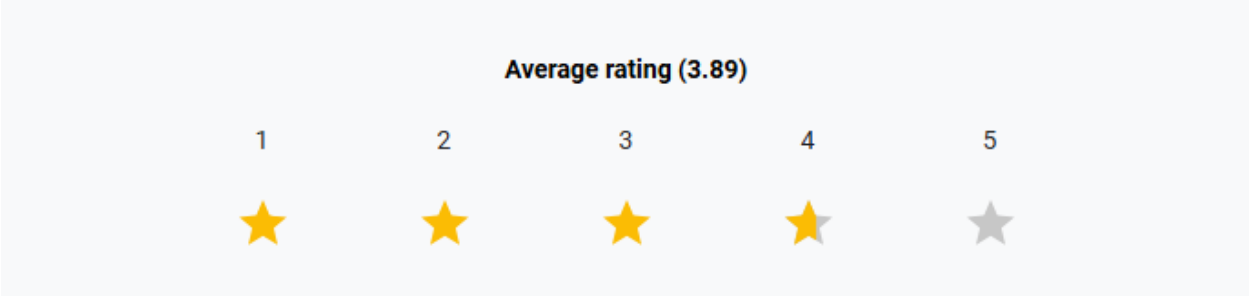
El total de sesiones de prueba de jugabilidad, regresión, integración y usabilidad fue 237, de las cuales 7 tuvieron cierres inesperados y fallos críticos. Realizando el cálculo, esto representa un 0,029% de sesiones de prueba con cierres inesperados, por lo que se considera que el juego se encontraba estable salvo casos puntuales. Algunos de estos casos fueron la inclusión por error de un objeto duplicado causando un cierre forzado de la aplicación, y problemas ocurridos a la hora de mezclar cambios sobre las escenas o prefabs corruptos. En todos los casos se solucionaron los errores antes de hacer el merge a rama principal.

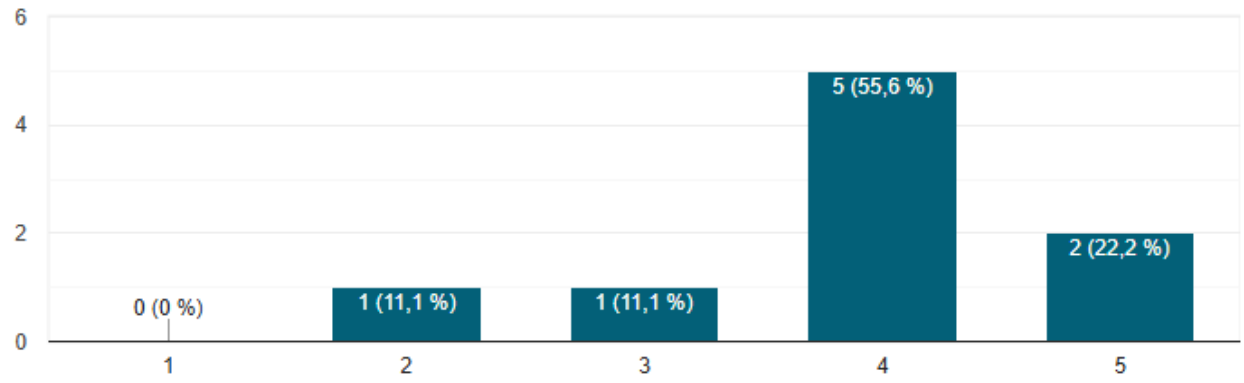
Si nos limitamos a contar errores críticos en pruebas con beta testers, este indicador es 0.

5.8.1.5. Satisfacción general del usuario

Para medir esto se realizó una encuesta a los beta testers utilizando la escala de Likert (del 1-5) sobre diferentes aspectos del juego que se encuentran detallados en profundidad en la sección de jugabilidad, también se les solicitó una calificación general del juego. En esta calificación se esperaba una valoración entre 3.5 y 5 como aceptable para etapas tempranas del proyecto y se planeó que para el siguiente hito, en octubre, la calificación supere el 4.0 en puntaje.

La puntuación obtenida en este punto fue de 3.89, que está dentro de los valores deseados y no muy lejanos a la próxima meta. De la desambiguación en distintos aspectos particulares de las respuestas de los usuarios se tomaron medidas para seguir mejorando las áreas con puntajes más bajos, sin descuidar nuestros puntos fuertes.





5.8.2. Resultado de métricas del proceso

5.8.2.1. Número de bugs reportados por sprint

Lo esperable en un proyecto de este tipo es que, a medida que el proyecto va avanzando, se vuelva cada vez más estable, por lo que decidimos graficar la cantidad de bugs reportados por sprint, esperando una tendencia a descender conforme pasaran los sprints. Esto fue lo que ocurrió efectivamente como se puede observar en la siguiente gráfica. Para realizarla se tomó como referencia la tabla de reporte de incidencias y se contó la cantidad de errores nuevos reportados en cada sprint.

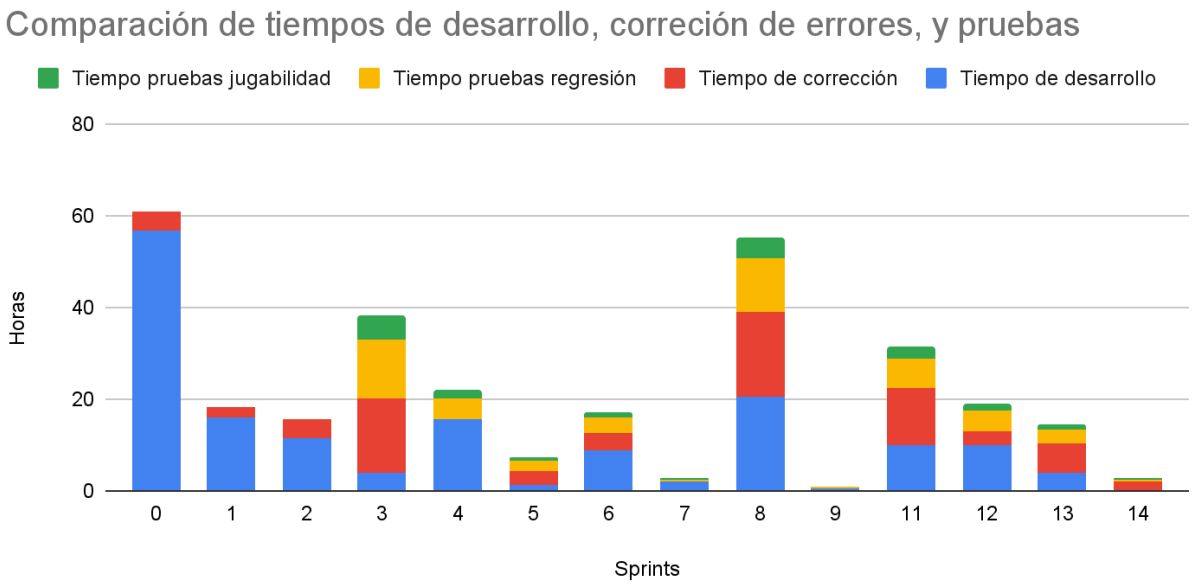
Como se puede observar la gráfica presenta una clara tendencia a la baja, mostrando así que a medida que transcurrió el tiempo, el juego se volvió cada vez más estable.

5.8.2.2. Tiempo promedio de tareas por sprint

Esta métrica fue utilizada con la finalidad de saber qué tan bien se estaban dividiendo las tareas en los sprints. El resultado esperado era que fuera una gráfica lo más estable posible, ya que en caso de que presentara resultados demasiado altos y alejados de la media, significaría que las tareas podrían haberse subdividido en tareas más pequeñas. En caso de que los resultados hubieran dado demasiado bajos, para un número bajo de tareas sería indicador de que quizás el número de tareas por sprint se podría incrementar, ya que en promedio las tareas asignadas tardaron poco tiempo en cumplirse. Como cota superior se propuso 3 horas de todas maneras, ya que se consideró que en promedio una tarea simple no debería tardar más que eso y como cota inferior 1 hora. En caso de que los promedios estuvieran alejados de las cotas se analizó la razón para saber si se estaban tomando correctamente las tareas o no.

5.8.2.3. Porcentaje de retrabajo vs desarrollo

Este indicador muestra en porcentajes qué cantidad de tiempo fue invertido en retrabajo comparado con tiempo invertido en desarrollo de nuevas funcionalidades. El valor esperado fue que el tiempo dedicado al retrabajo se mantuviera por debajo del 10% del tiempo total dedicado al desarrollo.



Como se puede observar en la gráfica, los primeros sprints la tasa de retrabajo fue bastante baja, pero para el sprint 3 esa tasa creció considerablemente debido a la introducción de bugs a medida que se iban añadiendo nuevas funcionalidades que muchas veces entraban en conflicto con otras funcionalidades existentes. En ese caso se tomó la medida de incluir nuevos tipos de pruebas para disminuir la cantidad de errores introducidos en nuevas versiones sobre funcionalidades ya implementadas, disminuyendo la tasa de retrabajo en futuros sprints.

Otro detalle a destacar es que, aún con la cota propuesta, este indicador no fue determinante para indicar si se estaba introduciendo una cantidad excesiva de errores que ocupaba tiempo de desarrollo. Esto a veces el Game Designer tenía una cantidad pequeña de tareas que asignar (sobre todo de en fechas cercanas a hitos en el cronograma, donde se debía mantener el juego en el mejor estado posible), y se ocupaba el tiempo que ocuparía el desarrollo en corregir errores. Considerando esto, aunque este indicador nos resultó muy útil al comienzo, no fue determinante para demostrar que el método estaba fallando, ya que la distribución del tiempo variaba según la inserción de nuevos requerimientos al backlog, lo cual estaba dado por el Product Owner y el Game Designer.

5.8.2.4. Velocidad del equipo

Para analizar la velocidad del equipo durante el desarrollo del proyecto, se utilizó la métrica de Story Points completados por sprint. Los Story Points fueron asignados a cada tarea en función de su complejidad técnica, volumen de trabajo e incertidumbre, utilizando una escala inspirada en la secuencia de Fibonacci (1, 2, 3, 5, 8, 13, …). Esta escala permitió una evaluación relativa del esfuerzo requerido para completar cada tarea.

Además se incluyeron en el análisis las tareas planificadas pero luego descartadas, lo que permitió calcular tanto la productividad como la capacidad de planeación del equipo.

Un detalle a considerar es que, por la forma del dictado de la materia de los artistas del equipo, el rol del Game Designer era quien determinaba las tareas asignadas a los desarrolladores, y, a su vez cada parte (desarrolladores y artistas) se encargaban del análisis y asignación de sus tareas. Las tareas que se listan debajo corresponden al equipo de programación.

Otra consideración es que el sprint 10 que se encuentra sin tareas corresponde a fin de año, el sprint 14 corresponde a la preparación de la revisión de la tesis y el sprint final, el 16, se trabajó únicamente en la documentación.

A continuación se presenta la tabla con los story points para cada sprint del equipo de desarrollo.

Sprint	Story Points planeados	Story Points finalizados	Cantidad de miembros
1	20	20	2
2	17	17	2
3	30	30	2
4	9	9	2
5	6	3	2
6	16	8	2
7	3	3	2
8	54	54	2
9	6	6	2
10	0	0	2
11	23	23	2
12	23	23	2
13	28	17	2
14	0	0	2
15	6	6	2

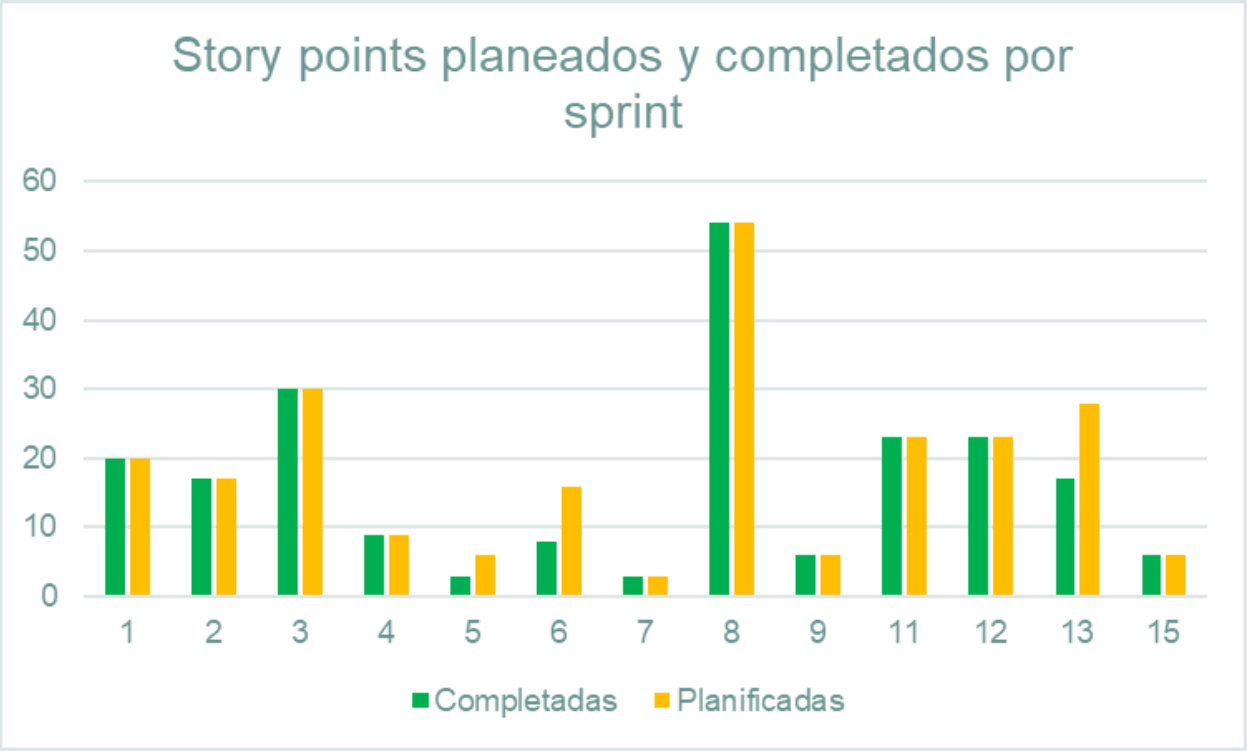
16	0	0	2

Como se puede ver no se consiguió mantener la métrica definida al comienzo. Hubo sprints en que, al depender de los requerimientos solicitados por el equipo de arte, esa cota no se alcanzaba, y en otros en que la cota se superaba. Otro detalle es que la cota fue creada en base al sprint 0, el cual duró 2 meses (tiempo aproximado de 4 sprints) en que se creó el juego de cero en el nuevo motor y ese sprint fue de 103 story points, así que se realizó un aproximado de la norma en los sprints realizando el siguiente cálculo:

(total de Story Points/cantidad de sprints)/cantidad de integrantes = (103/4)/2 ≈ 12,9.

Sin embargo es lógico que el avance al comienzo fuera más rápido que más adelante, ya que en la primera parte, todos los requerimientos ya estaban definidos de antemano y no se añadían nuevos, ya que el juego debía ser igual al que ya había creado en el otro motor. Más adelante dependía de la cantidad de requerimientos nuevos que se añadieran.

A continuación se presentan estos datos en forma de gráfico para mostrar de manera más simple los datos.



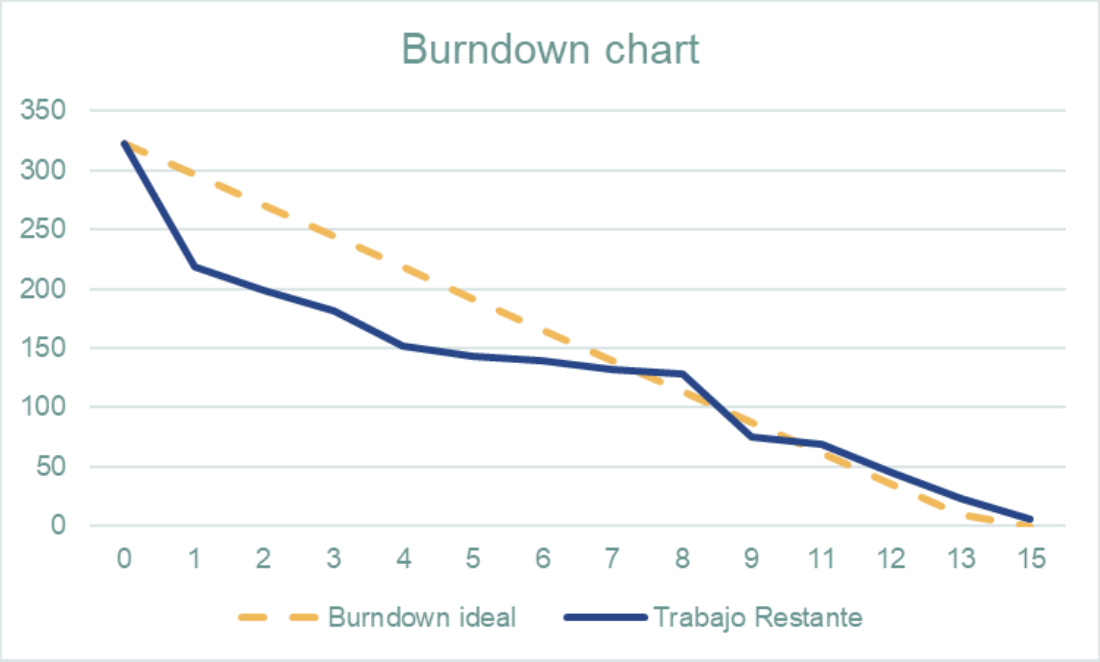
Comparando Story Points planeados y completados, se observa una diferencia en algunos de los sprints porque los requerimientos eran variables por la naturaleza del proyecto, por lo que hubo casos donde un requerimiento que se había planeado se

descartaba luego, siguiendo los criterios definidos para aceptación o descarte de requerimientos.

5.8.2.5. Porcentaje de tareas completadas en fecha

Finalmente la última métrica del proceso utilizada se realizó con el fin de observar el seguimiento correcto del cronograma definido, garantizando el cumplimiento de plazos. Se procuró que en ningún momento la cantidad de tareas sin completar superara el 10% de las tareas totales del sprint. Hubo un caso en que uno de los integrantes del equipo percibió un posible atraso en el cronograma, por lo que, para seguir cumpliendo con los estándares de calidad, invirtió horas extra tomándose días libres del trabajo.

En el burndown chart del proyecto que se presenta a continuación se puede ver en los sprints 4 al 8 donde el flujo de tareas empezó a ser inferior al esperado. En el sprint 8 se utilizó el plan de contingencia, dedicándole más horas al proyecto con el fin de compensar las tareas que comenzaban a acumularse. Esta burndown chart no fue posible realizarla desde el comienzo debido a los cambios en el backlog y en los requerimientos.



5.8.3. Resultado de métricas del proyecto

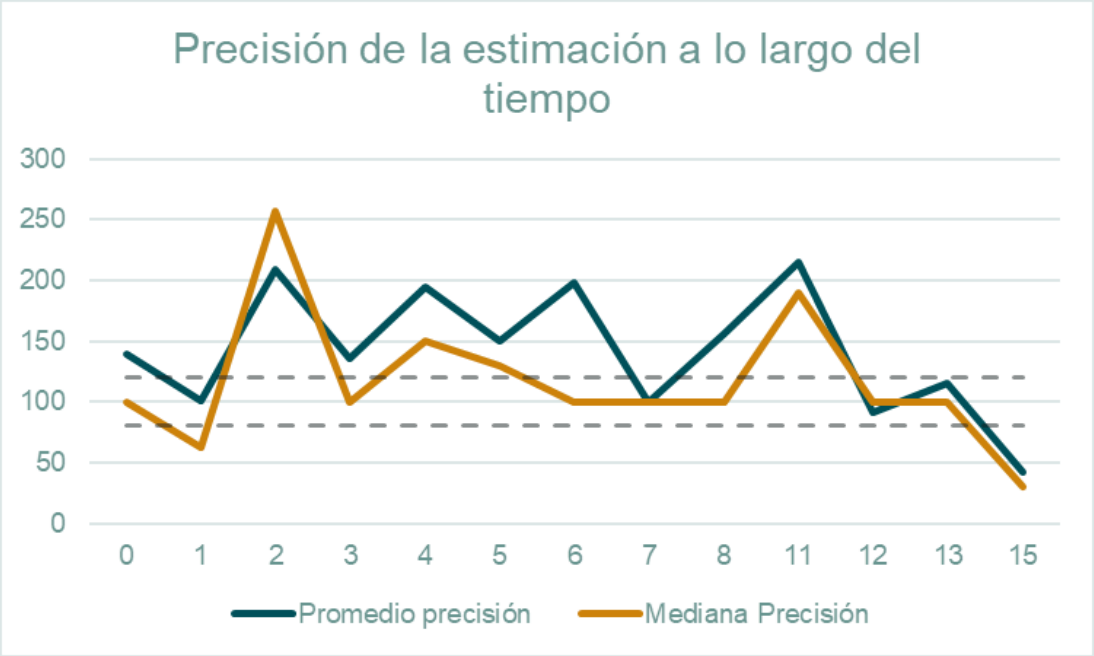
5.8.3.1. Precisión de estimaciones

Una métrica usada para confirmar la precisión de las estimaciones de tareas para fue verificar si la tasa de estimación de tareas se encontraba dentro de los valores esperados (entre 80% y 120% de precisión en la estimación). Esta métrica fue crucial durante el desarrollo ya que estaba muy relacionada a los criterios de aceptación de requerimientos. Si las estimaciones no eran demasiado imprecisas, existía el riesgo de

descartar funcionalidades que podían ser positivas para el juego para no retrasar el cronograma cuando en realidad lo que ocurría era que no se estaban estimando correctamente los tiempos.

En el gráfico a continuación se puede ver cómo esta estimación fue mejorando a lo largo del tiempo. Observando la mediana se puede ver como en los últimos sprints, la gráfica se fue manteniendo cada vez más en el rango de las cotas.

Se excluyen en la gráfica los sprints 10, 14 y 16 enfocados en documentación y el sprint 9 ya que contuvo tareas de agregado de audio solamente, las cuales tomaron más tiempo del estimado porque se tuvo que trabajar sobre las pistas de audio editándolas (en lo cual no se tenía experiencia y tampoco se previó que sería necesario) y buscar y agregar nuevos sonidos apropiados a los cambios recientes de estilismo y ambientación.



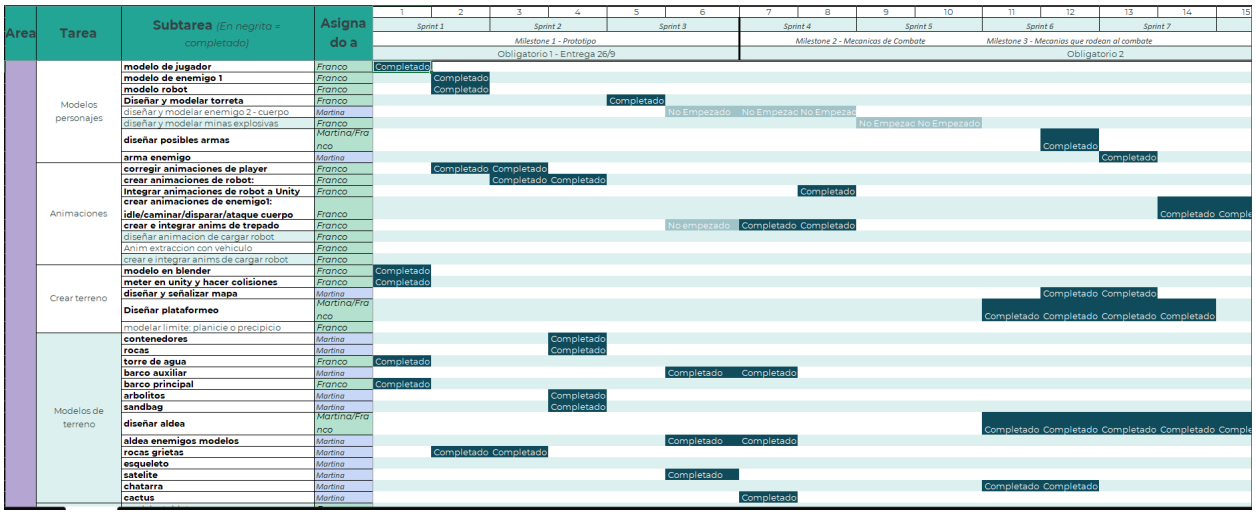
5.8.3.2. Avance del proyecto en porcentaje

Esta métrica a la fecha actual se encuentra con un 100% de completitud de las tareas estimadas para realizarse a día de hoy, por lo que se concluye que se cumplió con la meta planteada. Por otra parte aún quedan tareas en el backlog ya que el proyecto continuará con un estudiante de ingeniería. Sin embargo las funcionalidades clave planeadas para la fecha se encuentran completamente implementadas y se completaron todas las tareas planeadas para esta fase del desarrollo del proyecto.

5.8.3.3. Desviación del cronograma

Como se puede ver en la siguiente imagen, se contaba con un cronograma durante el proyecto en el que se anotaban las tareas correspondientes a cada milestone y del que

se podía obtener el cumplimiento con plazos. En el único momento que la desviación del cronograma fue mayor al 10%, se resolvió invirtiendo más horas de trabajo con el fin de volver a los plazos establecidos. Esto ocurrió en el sprint 8.



5.8.3.4. Story Points por sprint

Se procuró que la asignación de tareas para cada sprint fuera lo más equitativa posible, a fin de no sobrecargar a miembros del equipo con más tareas que a otros, aunque en ocasiones, sujeto a disponibilidad, miembros del equipo se ofrecían a realizar más trabajo a fin de cubrir a los otros miembros del equipo con menos disponibilidad horaria o que estuvieran en fechas comprometidas por otras entregas o carga de trabajo. Esto fue algo totalmente voluntario y que surgió de la buena comunicación y coordinación en el equipo.

5.9. Conclusión de plan de calidad

Definir un plan de calidad correctamente fue una pieza fundamental en el desarrollo del juego, ya que nos permitió garantizar, no solo un producto que cumpla con los estándares de calidad deseados y las expectativas puestas en él, sino que permitió, mediante las métricas corregir diversos errores y gestionar adecuadamente el proceso de desarrollo y el proyecto en general.

Se observa la importancia de monitorear constantemente las métricas, detectando en fases tempranas falencias en la calidad y sus razones y permitiéndonos tomar acciones correctivas con el fin de mejorar la calidad y reducir la tasa de retrabajo por defectos introducidos, ya que un error no detectado en fases tempranas del desarrollo puede provocar un sistema defectuoso y difícil de corregir a futuro.

Además, permitió gestionar adecuadamente el cumplimiento de plazos conforme al cronograma y ayudó con los criterios de aceptación o rechazo de requerimientos, definiendo muchas veces el rumbo del proyecto de manera realista.

6. Gestión del proyecto

6.1. Proceso de desarrollo

El proceso de desarrollo del proyecto se estructuró tomando como base el enfoque conocido como Dual Track Scrum, aunque adaptado a las necesidades y limitaciones concretas del equipo. Dual Track Scrum es una metodología ágil que propone dividir el trabajo en dos flujos paralelos: uno enfocado en la exploración y validación de ideas (discovery track) y otro centrado en la implementación técnica de las funcionalidades (delivery track). Esta separación tiene como objetivo evitar que se desarrollen funcionalidades innecesarias, permitiendo validar tempranamente el valor real de lo que se planea construir. En el desarrollo de videojuegos, esta metodología resulta especialmente útil, ya que muchas decisiones clave, como la claridad de las mecánicas, la experiencia de usuario, o el impacto de ciertas interacciones, sólo pueden validarse una vez que se prototipan o prueban directamente con el jugador.

Este modelo sirvió como marco conceptual, pero fue adaptado a las características específicas del proyecto. El equipo estaba compuesto por cuatro personas (dos desarrolladores y dos artistas), por lo que mantener dos flujos de trabajo formales en paralelo habría implicado una sobrecarga operativa considerable. En lugar de separar los roles en discovery y delivery de forma estricta, el equipo optó por una única línea de trabajo integrada, en la cual las ideas, funcionalidades y decisiones de diseño se discutían colectivamente, se validaban con el docente a cargo y con el Game Designer, y luego se incorporaban directamente al desarrollo según la prioridad del momento.

A pesar de no contar con una estructura formal de discovery, muchos de los principios del Dual Track Scrum sí estuvieron presentes. Por ejemplo, existía una validación constante de funcionalidades antes de su implementación definitiva. Esta validación se daba a través de discusiones con el Game Designer, experimentación directa en el juego, revisión entre pares, y sobre todo, por medio del feedback recibido en las sprint reviews con el docente. Estas instancias cumplían el rol de control de calidad y alineación con la visión general del proyecto, evitando avanzar con desarrollos que no aportaran valor o que pudieran generar problemas en etapas futuras o atrasos en el cronograma.

A lo largo del proceso también se mantuvo una alta flexibilidad en la priorización del backlog, permitiendo que ciertas ideas o ajustes surgieran en función de la evolución del juego, y no necesariamente desde una planificación rígida previa. Este enfoque iterativo y adaptativo es coherente con la lógica de descubrimiento continuo que propone el Dual Track Scrum, pero llevado a cabo de forma simplificada y orgánica.

La decisión de no aplicar la metodología en su totalidad respondió a motivos concretos. En primer lugar, el tamaño del equipo hacía inviable la existencia de dos tracks paralelos con roles diferenciados. En segundo lugar, el tiempo limitado por fechas de entregables y estrictamente marcado en el caso de los artistas por el cronograma del curso y las fechas marcadas de los entregables hacía necesario optimizar cada

instancia de trabajo, evitando duplicar procesos o generar documentación que no aporte valor inmediato. Finalmente, la comunicación directa con los stakeholders (el docente y el Game Designer) permitía validar rápidamente ideas o prototipos sin necesidad de procesos de investigación separados.

En definitiva, el proyecto se desarrolló a través de un enfoque ágil adaptado, basado en la filosofía del Dual Track Scrum, pero ajustado a la escala y dinámica del equipo. Se priorizó la colaboración constante, la validación continua y la flexibilidad en la toma de decisiones, manteniendo una estructura lo suficientemente ágil para sostener el desarrollo del juego sin sacrificar la calidad ni la visión de diseño.

6.2. Roles

Para asignar los roles se tomó en cuenta la preferencia personal de cada miembro del equipo. Aunque cada uno se centró principalmente en su rol acordado, el equipo se permitió cierta flexibilidad en los roles a la hora de enfrentar desafíos y tareas inesperadas, sólo en situaciones determinadas, por tiempo limitado, y si era estrictamente necesario para cumplir con el cronograma.

Product owner: Álvaro Azofra (profesor y tutor de Licenciatura en Animación y Videojuegos).

Project manager: Javier de Mattos.

Scrum master: Javier de Mattos.

Ingeniero de requerimientos: todos.

Responsable de SQA: Javier de Mattos.

Responsable de SCM: Daniel Komés.

Game designer: Franco Barreto.

Director de arte: Franco Barreto.

Artista técnico: Franco Barreto.

Artista: Franco Barreto, Martina Abascal.

Diseñador UX / UI: Franco Barreto, Martina Abascal.

Desarrollador: Daniel Komés, Javier de Mattos.

6.2.1. Product owner

Supervisa el avance del proyecto y la gestión general del equipo. Ayuda a establecer los objetivos del producto. Participa en la priorización de tareas del backlog.

6.2.2. Project manager

Establece la gestión del proyecto. Supervisa los procesos, objetivos y métricas. Es el principal encargado de la gestión de riesgos. Ajusta la planificación de acuerdo al cronograma, alcance e impacto.

6.2.3. Scrum master

Facilita reuniones, supervisa el cumplimiento del proceso, garantiza la comunicación efectiva y gestiona adaptaciones al marco de trabajo.

6.2.4. Ingeniero de requerimientos

Investiga y coordina con el Product owner estándares existentes para especificar requerimientos funcionales y no funcionales del proyecto, validando su necesidad e impacto.

6.2.5. Responsable de SQA

Desarrolla el plan de calidad, define los estándares a cumplir y coordina actividades para asegurar la calidad del proyecto.

6.2.6. Responsable de SCM

Administra las herramientas, establece estándares y capacita al equipo en el uso de las herramientas para garantizar su uso efectivo en el proyecto.

6.2.7. Game designer

Planifica y diseña la experiencia de juego, idea y define las mecánicas, narrativa, diseño de ambiente y niveles, apariencia y personalidad de personajes, entre otros. Coordina con los otros roles para asegurar la experiencia del producto final, balanceo de la dificultad y recompensas. Analiza el feedback de pruebas con usuarios para ajustar el diseño y mejorar la experiencia de usuario final.

6.2.8. Director de arte

Dirige el aspecto visual y estético, gestionando el estilo y paleta de colores. Participa en la decisión del diseño del entorno, niveles y efectos visuales. Lidera y coordina a otros miembros del equipo de arte para lograr un resultado atractivo y coherente.

6.2.9. Artista técnico

Artista gestionado por el director de arte. Diseña texturas, materiales, modelos, entre otros elementos necesarios para el aspecto visual. Usa su experiencia técnica con otros programas usados por el proyecto para integrar archivos diferentes en un entorno común para permitir que el resto del equipo los use.

6.2.10. Artista

Artista gestionado por el director de arte. Diseña texturas, materiales, modelos, entre otros elementos necesarios para el aspecto visual. Produce la mayoría de elementos visuales a usar.

6.2.11. Diseñador UX / UI

Diseña la estructura y aspecto visual de las interfaces de usuario para mantener una experiencia intuitiva, conveniente, fácil de usar y coherente con el estilo acordado por el director de arte.

6.2.12. Desarrollador

Desarrollar la implementación técnica, resolver problemas complejos de programación y garantizar la calidad del código.

6.3. Artefactos

Los artefactos que usados en nuestra metodología ágil, con el fin de gestionar correctamente todos los aspectos del proyecto, fueron los siguientes.

Product Backlog: lista de tareas desprendidas de los requerimientos aceptados según los criterios de aprobación.

Reporte de incidencias: lista de reporte de errores e incidencias con prioridad asignada, basada en su gravedad y urgencia por ser corregida.

Sprint Backlog: era un subconjunto del product backlog, donde se marcaban las tareas propias de cada sprint.

Board de tareas (Trello): se contaba con un trello donde los miembros del equipo movían sus tareas entre las distintas columnas “ToDo”, “Doing” y “Done” según su estado actual.

Burndown chart: gráfico que mostraba el progreso del proyecto, mostrando el trabajo restante comparado con el tiempo restante.

6.4. Ceremonias

Durante el desarrollo del proyecto, el equipo aplicó un enfoque ágil adaptado a su estructura y dinámica de trabajo. Si bien no se implementaron todas las ceremonias de Scrum de forma estricta, se llevaron a cabo aquellas instancias que resultaron clave para organizar y sostener el progreso continuo del desarrollo. Se realizaron sprints de 2 semanas donde se realizaba una sprint review al finalizar cada uno. Esta instancia se utilizaba para mostrar al docente el avance del proyecto y recibir feedback necesario para seguir mejorando el producto.

Luego de cada sprint también se realizaba una Sprint Retrospective donde el equipo analizaba el funcionamiento del trabajo en conjunto a lo largo de ese sprint. En estas reuniones se compartían experiencias sobre lo que había funcionado bien, lo que podía mejorarse y las dificultades que surgieron. Esto permitió ajustar dinámicas internas, mejorar la comunicación entre arte y desarrollo, y mejorar la calidad global del producto ya que en estas reuniones se tomaban o comunicaban las decisiones basadas en los resultados de los resultados de las métricas de SQA que derivaban en actividades nuevas de SQA en algunos casos como por ejemplo la inclusión de pruebas de regresión del sprint 3.

Todos los miércoles el equipo se reunía de forma sincrónica para revisar el estado general del proyecto, coordinar los avances del sprint en curso, y resolver dudas o bloqueos específicos. Estas reuniones reemplazaron el formato de “daily meetings” tradicional, ya que la frecuencia semanal resultó más adecuada por ser un equipo reducido y multidisciplinario y el volumen de trabajo que manejaban, la mayor parte de las comunicaciones de coordinación se realizaban mediante charlas informales por los canales de comunicación definidos. Esto ocurría ya que, a pesar de ser un equipo en su totalidad, internamente existían 2 equipos autogestionados que se encontraban en constante comunicación (desarrollo y diseño). De todas formas, en caso de que hubiera alguna deliberación importante que tomar que involucrara a todo el equipo se realizaba una reunión donde todos pudieran estar presentes.

6.5. Medición del trabajo

Dado el tamaño reducido del equipo y la variedad de tiempo disponibles de los integrantes, experiencia con distintas herramientas y al ser un equipo multidisciplinario por sobre todo, se optó por, en lugar de realizar la medición del trabajo mediante horas trabajadas, utilizar como unidad de medida los Story Points planeados para cada miembro del equipo por sprint. Estos Story Points dependían de las siguientes características de una tarea.

- Complejidad técnica o artística: se evaluaba cuán difícil era implementar o realizar una tarea, considerando conocimientos previos, herramientas a utilizar, integración con otras partes del sistema, o el nivel de detalle artístico necesario.
- Tiempo estimado de dedicación: si bien no se expresaba el esfuerzo en horas concretas, sí se consideraban rangos aproximados de duración para estimar el

tamaño de cada tarea. Esta estimación se basaba en experiencia previa o referencias de tareas similares.
- Riesgo e incertidumbre: tareas que implicaban exploración técnica, posibles bloqueos, o poca claridad en sus requerimientos eran puntuadas más alto, ya que representaban una mayor carga mental y riesgo de retrabajo.
- Dependencias: se tenía en cuenta si una tarea dependía de la entrega o avance de otro miembro del equipo. Las tareas con muchas dependencias requerían mayor coordinación y, por lo tanto, solían considerarse de mayor esfuerzo relativo.
- Impacto en la jugabilidad o en la entrega: aquellas tareas que eran clave para una entrega específica (por ejemplo, una milestone) o que impactaban directamente en la experiencia del jugador, eran cuidadosamente ponderadas, tanto en tiempo como en prioridad.

Para mantener consistencia, los Story Points asignados seguían una escala basada en la serie de Fibonacci modificada (1, 2, 3, 5, 8, 13, 21, 34). Esta elección permitió representar con más claridad la diferencia entre tareas pequeñas, medianas y grandes, evitando una linealidad artificial en el esfuerzo estimado.

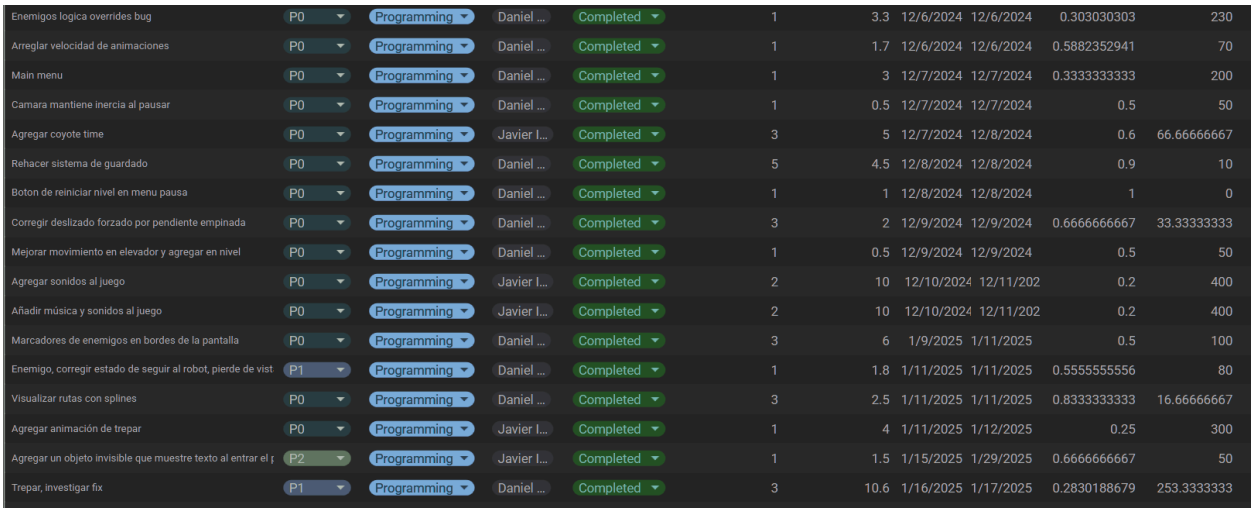
6.6. Definición de backlog

El backlog es la lista de tareas pendientes. Estas tareas están sujetas a modificaciones y revisiones, sobre todo cuando ha pasado cierto tiempo entre su definición y su realización, ya que el proyecto podría haber avanzado en direcciones inesperadas que causaran que haya que replantear la tarea.

Al definir la tarea se registraba su descripción, tipo de actividad (programación, arte, UI, game design), horas estimadas de trabajo, prioridad, y opcionalmente, responsable.

Al terminar la tarea se completaban los datos de su fecha de inicio y fin de realización y horas de trabajo.

Tener este documento fue extremadamente útil para organizar las tareas y entender cómo avanzaba el proyecto en el tiempo.



Enemigos logica overrides bug	P0	Programming	Daniel ...	Completed	1	3.3	12/6/2024	12/6/2024	0.303030303	230
Arreglar velocidad de animaciones	P0	Programming	Daniel ...	Completed	1	1.7	12/6/2024	12/6/2024	0.5882352941	70
Main menu	P0	Programming	Daniel ...	Completed	1	3	12/7/2024	12/7/2024	0.3333333333	200
Camara mantiene inercia al pausar	P0	Programming	Daniel ...	Completed	1	0.5	12/7/2024	12/7/2024	0.5	50
Agregar coyote time	P0	Programming	Javier I...	Completed	3	5	12/7/2024	12/8/2024	0.6	66.66666667
Rehacer sistema de guardado	P0	Programming	Daniel ...	Completed	5	4.5	12/8/2024	12/8/2024	0.9	10
Boton de reiniciar nivel en menu pausa	P0	Programming	Daniel ...	Completed	1	1	12/8/2024	12/8/2024	1	0
Corregir deslizado forzado por pendiente empinada	P0	Programming	Daniel ...	Completed	3	2	12/9/2024	12/9/2024	0.6666666667	33.33333333
Mejorar movimiento en elevador y agregar en nivel	P0	Programming	Daniel ...	Completed	1	0.5	12/9/2024	12/9/2024	0.5	50
Agregar sonidos al juego	P0	Programming	Javier I...	Completed	2	10	12/10/2024	12/11/202	0.2	400
Añadir música y sonidos al juego	P0	Programming	Javier I...	Completed	2	10	12/10/2024	12/11/202	0.2	400
Marcadores de enemigos en bordes de la pantalla	P0	Programming	Daniel ...	Completed	3	6	1/9/2025	1/11/2025	0.5	100
Enemigo, corregir estado de seguir al robot, pierde de vist	P1	Programming	Daniel ...	Completed	1	1.8	1/11/2025	1/11/2025	0.5555555556	80
Visualizar rutas con splines	P0	Programming	Daniel ...	Completed	3	2.5	1/11/2025	1/11/2025	0.8333333333	16.66666667
Agregar animación de trepar	P0	Programming	Javier I...	Completed	1	4	1/11/2025	1/12/2025	0.25	300
Agregar un objeto invisible que muestre texto al entrar el p	P2	Programming	Javier I...	Completed	1	1.5	1/15/2025	1/29/2025	0.6666666667	50
Trepar, investigar fix	P1	Programming	Daniel ...	Completed	3	10.6	1/16/2025	1/17/2025	0.2830188679	253.3333333

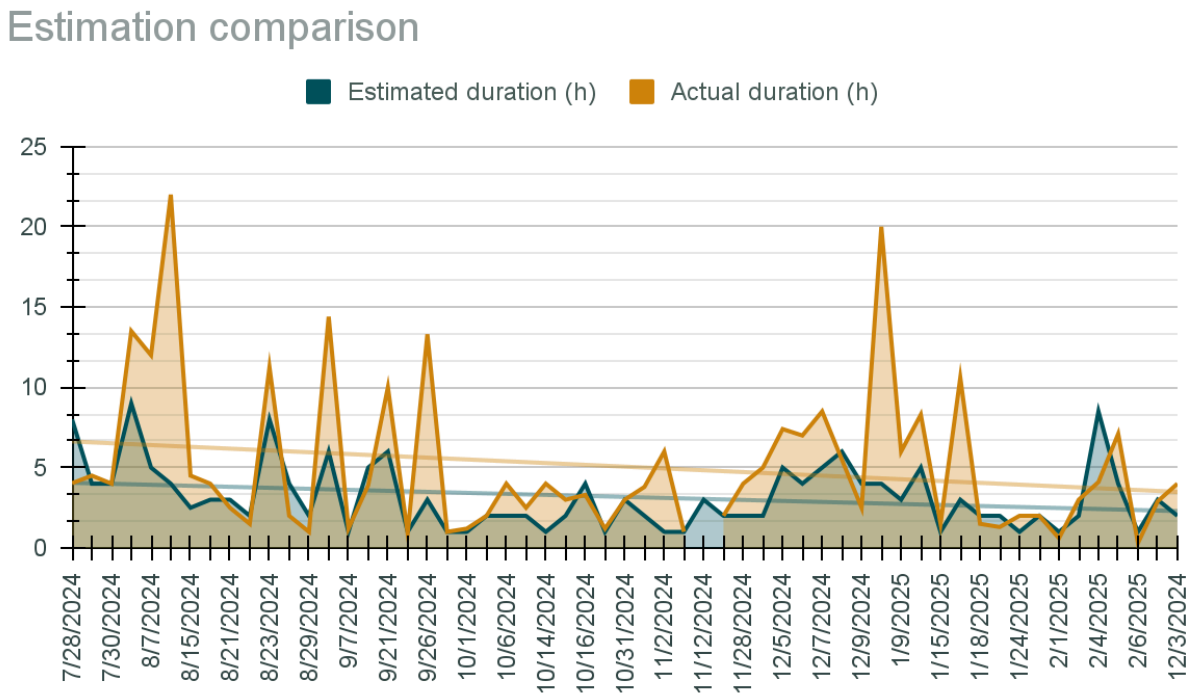
6.7. Estimación y planificación

El equipo del proyecto se compuso por dos artistas y dos desarrolladores, y la supervisión de dos tutores, uno para cada facultad: Facultad de Ingeniería y Facultad de Diseño y Animación.

Se estableció que los sprints serían de 2 semanas cada uno, lo que resultó en 16 sprints hasta la primera entrega final (de Licenciatura).

Debido a la incertidumbre de tiempos, se fueron definiendo milestones a medida que avanzaba el proyecto.

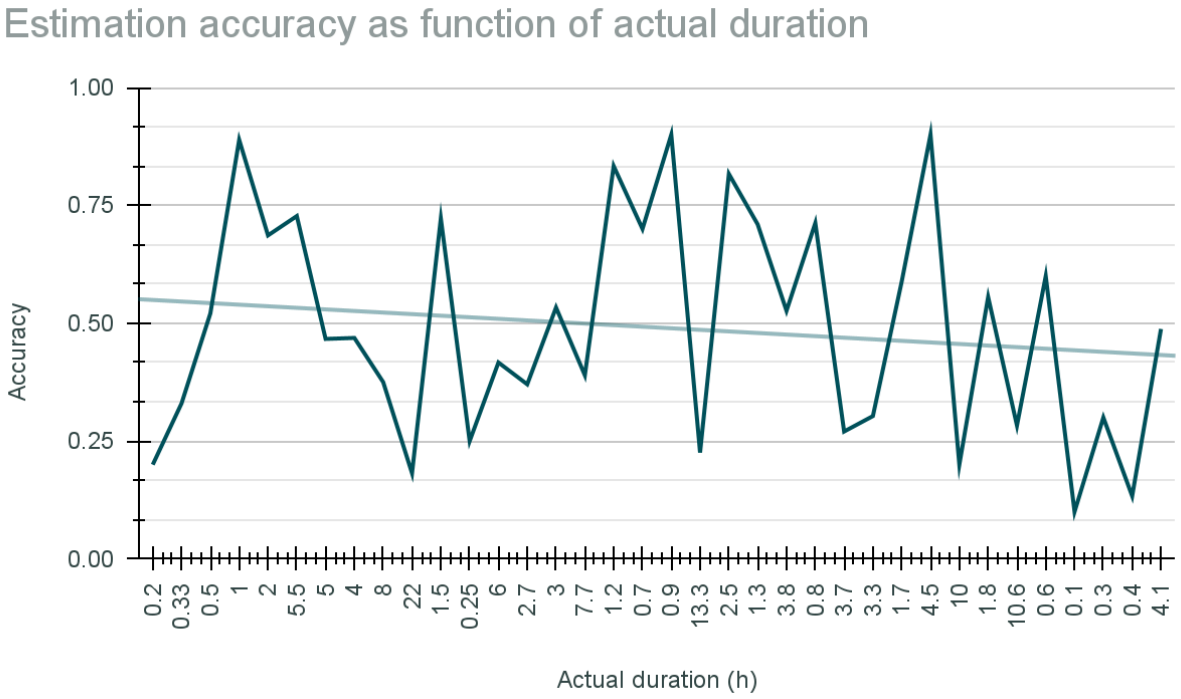
A continuación se muestra una comparación entre estimaciones de duración de tareas y duración real de tareas.



Las líneas de trend descendientes significan que ambas, las estimaciones y la realidad, fueron descendiendo con el paso del tiempo. Esto significa que a medida que avanzaba el proyecto fuimos capaces de separar tareas en fracciones más pequeñas.

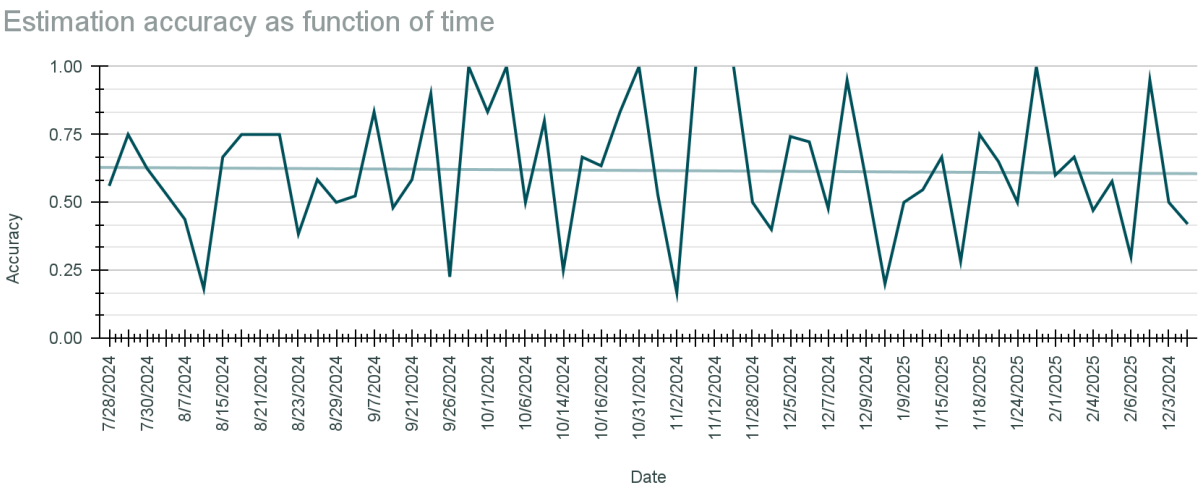
Además, ambas líneas de trend se acercan una a la otra con el paso del tiempo. Esto significa que fuimos aumentando, aunque sea ligeramente, nuestra precisión al estimar la duración de las tareas.

La siguiente es una comparación entre la precisión de la estimación con respecto a la duración real de cada tarea.



La precisión de las estimaciones desciende cuando la duración real aumenta. Esto significa que nuestras estimaciones fueron más precisas al estimar tareas que terminaron requiriendo menos esfuerzo.

La siguiente gráfica refleja la precisión de las estimaciones en función del tiempo.



La precisión de las estimaciones decrece ligeramente cuando la fecha de inicio de la tarea aumenta. Esto significaría que a medida que avanzaba el proyecto, la precisión de las estimaciones bajó. Sin embargo, es posible que errores de redondeo u otras circunstancias causen que el trend descienda.

6.8. Gestión de riesgos

Durante el desarrollo del proyecto se registraron y categorizaron varios riesgos que deberían ser monitorizados.

6.8.1. Categorización de riesgos

En esta tabla se muestran los riesgos identificados, su tipo, impacto, probabilidad, estrategia general a usar y fecha de identificación.

Tipo	Título	Impacto	Probabilidad	Estrategia	Fecha de identificación
Técnicos	Compatibilidad con hardware	Medio	Muy baja	Aceptar	11/4/2024
	Rendimiento	Muy alto	Media	Reducir	11/4/2024
	Fallo de integración	Alto	Muy baja	Evitar	11/4/2024
Desarrollo	Cambio de tecnología	Crítico	Muy baja	Evitar	11/4/2024
	Retraso en el cronograma	Alto	Alta	Reducir	11/4/2024
	Falta de personal o recursos clave	Muy alto	Muy baja	Aceptar	11/4/2024
	Problemas de comunicación y	Alto	Baja	Reducir	11/4/2024
	Problemas de comunicación con tutor	Alto	Muy baja	Reducir	11/4/2024
Calidad	Problemas de usabilidad	Medio	Media	Reducir	11/4/2024
	Bugs	Medio	Alta	Reducir	11/4/2024
	Problemas de balance del juego y experiencia	Alto	Media	Reducir	11/4/2024
	Features incrementales	Alto	Alta	Reducir	11/4/2024

6.8.2. Definición de riesgos

6.8.2.1. Técnicos

6.8.2.1.1. Compatibilidad con hardware

Problemas de compatibilidad con distintos dispositivos.

Impacto: los dispositivos afectados tendrían una mala experiencia de juego o incluso podrían no lograr ejecutarlo.

Medidas de prevención: investigar las configuraciones que ofrece Unity para estos casos.

6.8.2.1.2. Rendimiento

Requerimientos de hardware demasiado altos para la mayoría de dispositivos objetivo.

Impacto: los dispositivos afectados tendrían una mala experiencia de juego o incluso podrían no lograr ejecutarlo.

Medidas de prevención: aplicar occlusion culling, una técnica que facilita Unity que desactiva el renderizado de objetos que la cámara no capta, ya que siguen consumiendo recursos aunque no sean visibles. Probar regularmente el juego en dispositivos con el hardware objetivo para asegurar que el rendimiento es aceptable.

Plan de contingencia: si el problema se presenta en demasiados dispositivos, hacer más eficiente las partes del código (donde sea posible) que se ejecutan más

frecuentemente, y/o reducir su frecuencia. Aplicar LOD, una técnica que reduce la resolución de objetos lejanos a la cámara, ya que no se pueden apreciar sus detalles, pero siguen consumiendo recursos.

6.8.2.1.3. Fallo de integración

Al usar un motor de terceros (Unity) pueden ocurrir fallos de integración que requieran soluciones adicionales. Esto requeriría investigar interacciones externas con el motor.

Impacto: un cambio de tecnología requiere cambiar el motor, lo que afectaría enormemente al proyecto, hasta el punto de potencialmente cancelarlo por completo.

Medidas de prevención: investigar funcionalidades sobre las que el equipo tiene poca experiencia de implementación. Descartar funcionalidades difíciles o imposibles.

6.8.2.2. Desarrollo

6.8.2.2.1. Cambio de tecnología

El equipo decide o necesita cambiar de motor.

Impacto: se requeriría rehacer por completo ciertas partes de diseño, cambiar el código para conectar con la API del nuevo motor, o incluso reescribir todo el código en otro lenguaje si el nuevo motor no es compatible con el lenguaje actual (C#).

Medidas de prevención: reducir alcance para que el juego sea realizable en el motor actual.

6.8.2.2.2. Retraso en el cronograma

Problemas para cumplir plazos y tiempos establecidos.

Impacto: reduciría la calidad del resultado final. Forzaría una reducción del alcance.

Medidas de prevención: reuniones periódicas de informes de avances con el equipo para mantener actualizado el cronograma y expectativas.

Plan de contingencia: Reducir el alcance y/o aumentar la carga horaria de los miembros del equipo.

6.8.2.2.3. Falta de personal o recursos clave

Los miembros del equipo abandonan el proyecto, o surge la carencia de algún recurso vital (hardware o software específico usado).

Impacto: el proyecto sería retrasado y tendría menos calidad y funcionalidades.

Medidas de prevención: monitorear actualizaciones del software usado para detectar cambios de compatibilidad con hardware disponible. Reuniones periódicas con el equipo para mantenerse al tanto de cambios relevantes.

Plan de contingencia: reducir alcance. Justificar reducción de calidad.

6.8.2.2.4. Problemas de comunicación y coordinación del equipo

Malentendidos, poca coordinación, errores de interpretación, trabajo duplicado, inconsistencias, etc.

Impacto: bajaría la calidad del resultado final, posiblemente causaría retrasos en el cronograma, y por lo tanto, reducción en el alcance.

Medidas de prevención: reuniones periódicas para asegurar un buen clima de trabajo. Informes de avances en el equipo para mantener la coordinación. Mantener una lista de tareas asignadas con responsable designado y plazos definidos.

6.8.2.2.5. Problemas de comunicación con tutor

Problemas de comunicación entre el tutor y los miembros del equipo que causan conflicto entre las partes y entorpecen el avance del proyecto.

Impacto: reduciría la calidad de la documentación y procesos referentes a la tesis.

Medidas de prevención: mantener comunicación periódica. Interacción abierta y respetuosa. Reuniones semanales de informe de avances para evitar malentendidos.

Plan de contingencia: recurrir a coordinadores de la cátedra de Software Factory.

6.8.2.3. Calidad

6.8.2.3.1. Problemas de usabilidad

No cumplir las expectativas de facilidad de uso y percepción de calidad.

Impacto: causaría una mala experiencia de uso. Si se intenta corregir usando feedback de testing, podría causar reducción del alcance.

Medidas de prevención: especificar requerimientos de calidad. Reuniones y testing periódico de funcionalidades e integración para opinar en equipo posibles mejoras.

Plan de contingencia: dedicar tiempo en mejorar el producto, a cambio de reducir el alcance. O mantener el alcance aceptando la calidad subóptima.

6.8.2.3.2. Bugs

Errores y comportamiento no esperado durante el juego.

Impacto: reduciría la experiencia del usuario dependiendo de la gravedad de los comportamientos inesperados.

Medidas de prevención: pruebas frecuentes al implementar cada funcionalidad y pruebas de integración con el resto de funcionalidades. Durante el desarrollo, dar acceso al proyecto a personas de confianza externas para que lo prueben. Supervisar y registrar errores encontrados.

Plan de contingencia: priorizar corregir y registrar bugs graves (que impidan el funcionamiento de la aplicación o impidan de forma evidente el progreso del juego). Dedicar tiempo en mejorar el producto, a cambio de reducir el alcance, si los bugs son críticos. O mantener el alcance aceptando la calidad subóptima, si los bugs no son críticos.

6.8.2.3.3. Problemas de balance del juego y experiencia

Incorrecto balance en el juego. Ej: recompensas demasiado escasas o demasiado abundantes. Combate demasiado fácil o difícil. Mecánicas demasiado simples o complejas. Enemigos demasiado pasivos o agresivos. Exploración demasiado tediosa, aburrida y/o inútil.

Impacto: afectaría la comprensión, inmersión y/o diversión general del jugador.

Medidas de prevención: pruebas con terceros seleccionados que opinen sobre su experiencia de juego durante el desarrollo. Considerar hacer cambios basados en opiniones repetidas entre los usuarios.

Plan de contingencia: dedicar tiempo en mejorar el producto, a cambio de reducir el alcance. O mantener el alcance aceptando la calidad subóptima.

6.8.2.3.4. Features incrementales

Funcionalidades que cambian una vez que el balance del juego tiene una estabilidad aceptable.

Impacto: potencialmente causarían retrasos y descoordinaciones en el cronograma.

Medidas de prevención: monitorear constantemente las funcionalidades agregadas. Evaluar el costo de cambio de cada una.

Plan de contingencia: aceptar la funcionalidad, pero requiere más tiempo para balancear el juego. O rechazar los cambios, pero una funcionalidad potencialmente valiosa se descarta.

6.8.3. Conclusiones de riesgos

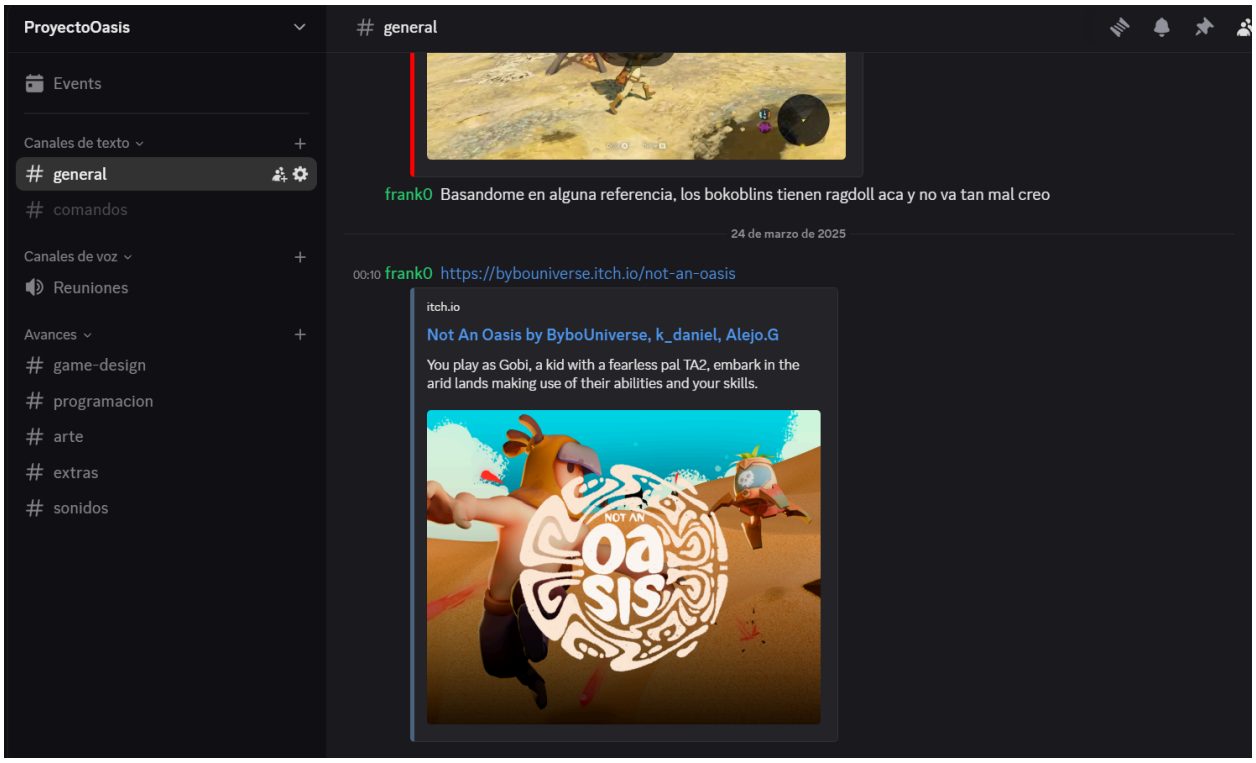
El análisis de riesgos presentado permite anticipar los principales factores que podrían afectar negativamente el desarrollo y la calidad del proyecto, agrupándolos en categorías técnicas, de desarrollo y de calidad.

Identificar estos riesgos desde etapas tempranas facilita la planificación de medidas preventivas y planes de contingencia apropiados, lo que incrementa las posibilidades de éxito del proyecto. Si bien es imposible eliminar por completo la incertidumbre, el enfoque adoptado busca minimizar el impacto de los eventos adversos mediante una gestión proactiva, coordinada y realista.

A medida que avanzó el proyecto, se observó la utilidad de la gestión de riesgos, ya que algunos de ellos efectivamente se presentaron. Gracias al monitoreo constante y a la aplicación de las estrategias previstas, fue posible abordarlos de manera oportuna, evitando que tuvieran un impacto negativo significativo sobre el desarrollo.

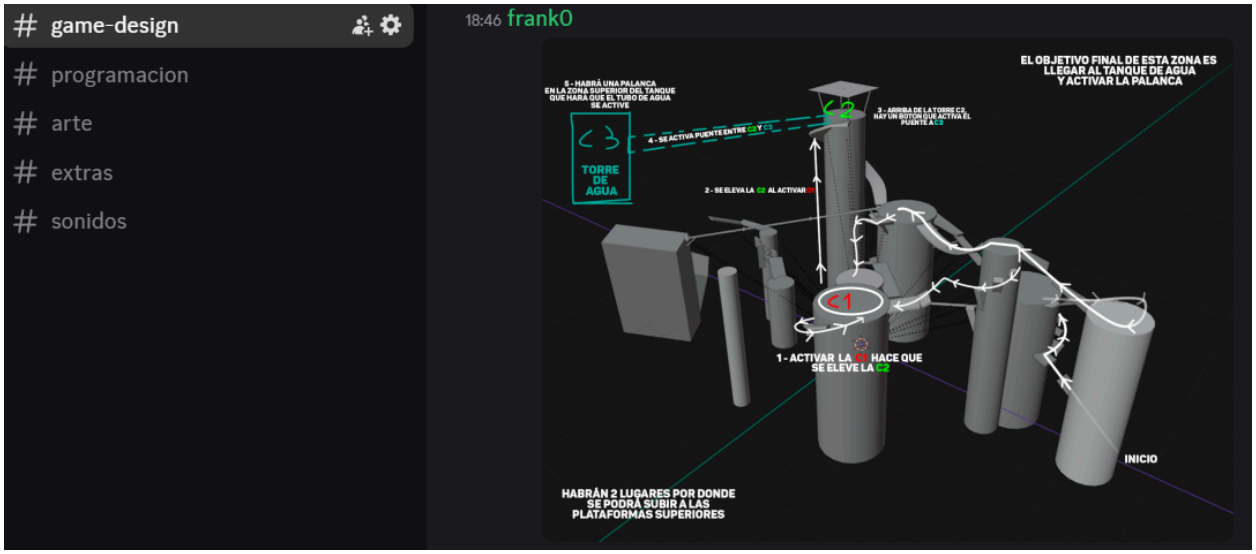
6.9. Gestión de la comunicación

El principal medio de comunicación durante el proyecto fue Discord, debido a sus funcionalidades de organización por “canales”. Los canales de texto en la sección “Avances” fueron usados para discutir los temas correspondientes.



A continuación se describen los canales usados en el proyecto.

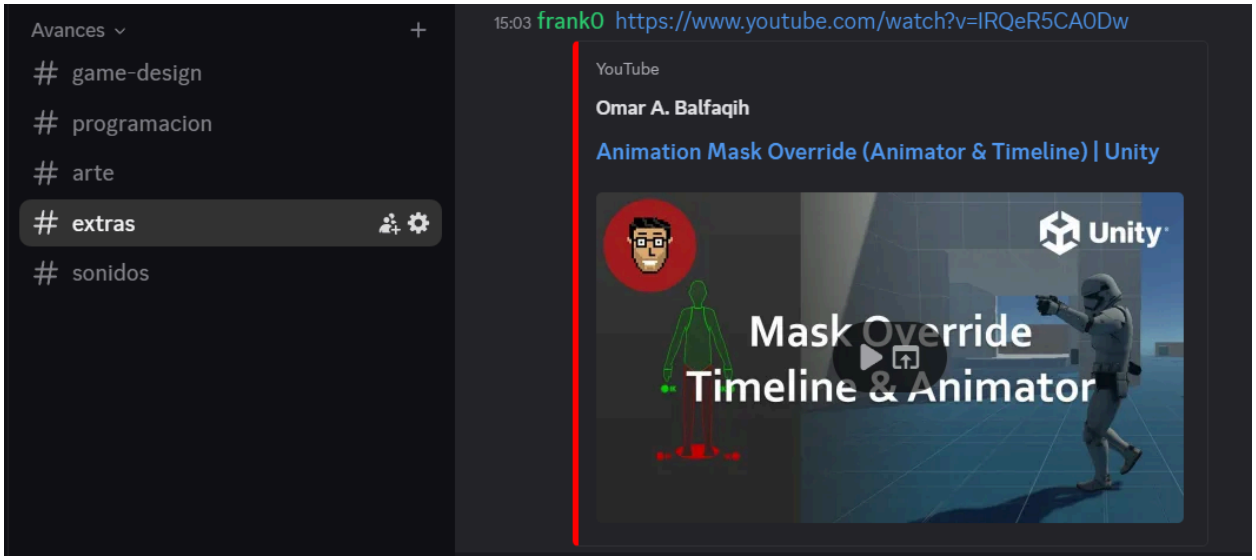
En “game-design” se discutían temas de diseño del juego.



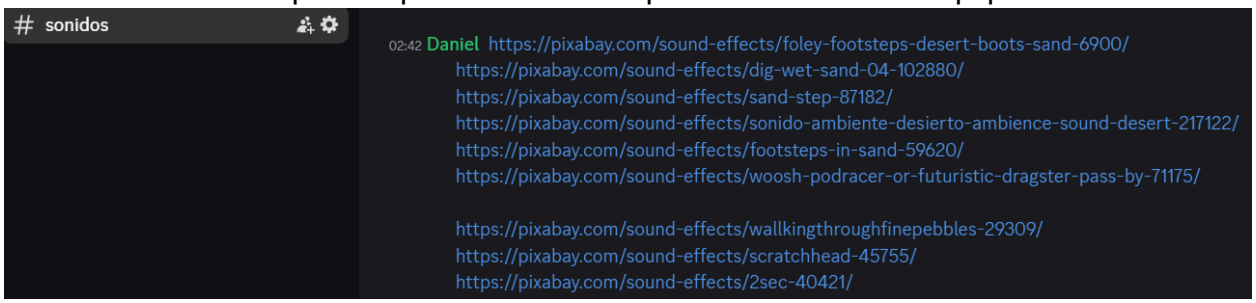
En “programacion” se mostraban avances y discutían temas de la programación.

En “arte” se mostraban avances y discutían temas de arte.

En “extras” se compartían plugins, y tutoriales útiles de temas varios.



En “sonidos” se compartían posibles sonidos para evaluar con el equipo.



6.10. Resultado de sprints

En esta sección realizaremos un listado de los distintos sprints y sus principales hitos.

6.10.1. Sprint 1 (21-08-2024)

En el primer sprint se organizaron los canales de comunicación para el equipo en Discord y Whatsapp.

Milestone 1: prototipo.

Tareas realizadas:

- Se diseñaron los modelos de Gobi, TA-2 y enemigo.
- Se creó el terreno en blender y se configuró para importar en Unity. Se agregaron las colisiones y algunos modelos de objetos del ambiente (torre de agua, barco destruido, rocas).
- Se texturizaron algunos modelos.

- Se crearon varios efectos especiales de chispas al disparar las armas láser .
- Se programaron algunos comportamientos del enemigo y TA-2, y algunos comportamientos de los proyectiles.

Sprint Review: En esta primera versión, de la aplicación (recordemos que el proyecto había comenzado a construirse en el semestre anterior al comienzo de clases), si bien la opinión del docente fue positiva, se indicaron algunas correcciones a las animaciones, así como se hizo mención a algunas de ellas faltantes, además se solicitó una versión del recorrido del nivel principal para poder brindar feedback.

Como era el primer sprint no se realizó una Sprint Retrospective.

6.10.2. Sprint 2 (04-09-2024)

Milestone 1: prototipo.

Tareas realizadas:

- Se mejoraron las animaciones de Gobi.
- Se crearon las animaciones de TA-2 de idle, cavar y caminar.
- Se modelaron decoraciones varias (rocas, contenedores, árboles).
- Se texturizaron los modelos de Gobi, TA-2, mochila de Gobi, barco.
- Se creó el material para la arena.
- Se modelaron la casa y algunas decoraciones de la aldea.
- Se presentó un esquema del recorrido del nivel principal para feedback.
- Se implementó la terminal de mods para cambiar mods de TA-2.
- Se agregaron comportamientos al enemigo: investigar, seguir a TA-2, ataque melee al acercarse.
- Se agregaron comportamientos a TA-2: hackear, cavar.
- Se modificó la lógica de TA-2 para que sus comportamientos sean modulares e independientes.
- Se implementó la lógica de interactuar de Gobi con otros objetos interactuables.

Sprint Review: En este sprint el docente nos brindó feedback acerca del recorrido del nivel principal, así como solicitó un “white boxing” del nivel principal para así poder tener un prototipo jugable sobre el cual continuar trabajando.

Sprint Retrospective: el equipo se encuentra conforme con el flujo actual de tareas, ninguno de los integrantes del equipo expresa encontrarse sobrecargado de trabajo. Sin embargo, por algunas complicaciones a la hora de realizar cambios en el repositorio por parte del equipo de arte, se llega a la decisión en conjunto de que siempre haya un desarrollador presente a la hora de realizar la integración del trabajo de los artistas en git, evitando de esta forma que puedan tener algún tipo de problema que les implique realizar retrabajo, además se determinó necesario realizar pruebas de regresión a fin de que nuevos cambios no entren en conflicto con cambios ya integrados. Se discutieron ajustes finales para la entrega del primer obligatorio.

6.10.3. Sprint 3 (18-09-2024)

Milestone 1: prototipo.

Tareas realizadas:

- Se modeló la torreta (descartada).
- Se planificó el modelo del enemigo 2 con temática cuerpo a cuerpo (descartado).
- Se planificó la animación de trepar de Gobi.
- Se modeló el barco secundario.
- Se modelaron algunos enemigos para la aldea.
- Se modeló la antena parabólica del nivel.
- Se modelaron algunas decoraciones para la casa en la aldea.
- Se modeló el vehículo de Gobi (descartado).
- Se crearon efectos visuales de tormenta de arena.
- Se trabajó en el “white boxing” del nivel.
- Se discutió el alcance preliminar del juego.
- Se creó el menú de pausa para la primera entrega.
- Se agregó comportamiento en los enemigos para que alerten a enemigos cercanos cuando entran en combate.
- Se cambió la lógica de TA-2 al seguir al jugador para que no se ponga en el camino del jugador, entorpeciendo el movimiento.
- Se cambió la lógica de marcado de enemigos para que se desmarquen al alejarse demasiado, e impedir marcar desde demasiado lejos.

- Se realizaron pruebas de regresión sobre el movimiento y mecánicas del jugador.

- Se creó el ejecutable para entregar en el primer obligatorio.

Sprint Review: se mostró el “white boxing” del nivel al profesor, el cual realizó sugerencias y aspectos a mejorar sobre el prototipo presentado.

Sprint Retrospective: Se notó una mejora positiva respecto a los problemas con la integración de nuevas características a la hora de generar nuevos errores, aunque se esperó ver en un futuro cuando se tuvieran más datos si realmente el cambio fue positivo. Se coordinaron horarios de disponibilidad de los integrantes para tener reuniones. Se planteó como objetivo la inclusión de nuevos comportamientos en los enemigos y Gobi.

6.10.4. Sprint 4 (02-10-2024)

Milestone 2: combate básico.

Tareas realizadas:

- Se crearon e implementaron las animaciones de trepar de Gobi.

- Se agregaron algunas animaciones de TA-2.

- Se modelaron algunas decoraciones del terreno.

- Se texturizaron algunos objetos decorativos y la mochila de Gobi y terminal de mods.

- Se creó el material general para las rocas.

- Se crearon efectos visuales de la habilidad Sonar de TA-2, de impacto de balas en enemigos y polvo al saltar.

- Se agregaron comportamientos al enemigo: patrullar siguiendo rutas predefinidas, atacar melee cuando el objetivo está suficientemente cerca, buscar y lanzar cajas del entorno, probabilidad de priorizar atacar a Gobi en vez de TA-2, armas modulares de forma que cambien su comportamiento según el arma que tengan, capacidad de buscar y equipar armas del suelo.

- Se implementó la capacidad de agacharse de Gobi (descartado).

Sprint review: El docente indicó que el combate no se sentía bien.

Sprint retrospective: Se comentó acerca de un problema que surgió a la hora de integrar cambios en la estructura del nivel, el problema surgido fue que, con las nuevas texturas, resultó perjudicado el correcto recorrido del nivel, es decir, el completar el nivel se volvió demasiado complejo. Visto esto se determinó la necesidad de incluir

pruebas de integración, de esta forma, cada vez que se introducía un cambio o ajuste que impactara directamente sobre el nivel, se debía realizar una prueba de todo el nivel (es decir jugarlo completo o al menos hasta el sitio del cambio en caso de que este sea de algún modelo del terreno) a fin de garantizar que dicho cambio no perjudicara la experiencia del jugador a la hora de completar el nivel o que directamente se lo impidiera. Se planificó para el próximo sprint una forma de esquivar los ataques, y tomando en cuenta los comentarios del docente, se realizaron cambios a TA-2 y a Gobi para intentar mejorar el combate.

6.10.5. Sprint 5 (16-10-2024)

Milestone 2: combate básico.

Tareas realizadas:

- Se planificó el modelado de minas explosivas (descartado).
- Se texturizó la casa de la aldea y algunas decoraciones.
- Se creó el material para una variante de la arena (descartado).
- Se planificó el modelado del faro para la aldea (descartado).
- Se creó el efecto visual de polvo al correr y se modificó el efecto visual de tormenta de arena.
- Se hicieron pruebas de integración y se corrigieron errores encontrados.
- Se modificó el comportamiento de TA-2 para que se sienta más dinámico e independiente.
- Se implementó la lógica de rodar de Gobi.

Sprint review: El comportamiento de TA-2 se sentía mejor pero aún así el combate no se sentía del todo bien, el jugador sentía no tener el control o se sentía directamente inútil, se plantea la posibilidad de plantear al jugador la posibilidad de tomar una ruta alternativa para limitar los encuentros con enemigos, una de los principales problemas del combate era el sentirse abrumado debido a la cantidad de enemigos y la incapacidad de atacar, por lo que se planeó que en la ruta alternativa, los encuentros con enemigos estuvieran limitados a combates contra solo un enemigo a la vez, la idea era que la verticalidad del nivel lograra esto, manteniendo los enemigos en la parte inferior. En base a eso se declaró como necesario implementar una mecánica que permitiera trepar a superficies.

6.10.6. Sprint 6 (30-10-2024)

Milestone 3: combate avanzado.

Tareas realizadas:

- Se discutió el diseño de posibles armas.
- Se trabajó en el diseño del nivel, creando señalizaciones y diseñando el flujo de plataformas.
- Se diseñó el terreno de la aldea.
- Se modeló la chatarra usada como recurso.
- Se planificó la creación del material para arena movediza (descartada).
- Se trabajó en el whiteboxing de la aldea.
- Se implementó la mecánica de trepar de Gobi.
- Se planificó la mecánica de granadas explosivas (descartado).

Sprint review: Si bien los cambios implementados se sentían mejor aún se sentía mal la mecánica de trepar del jugador, además era notoria la falta de modelos y animaciones.

Sprint retrospective: El equipo de arte, considerando el feedback proporcionado y que la siguiente iteración contenía una muestra del juego a una clase de la facultad de animación, se propone agregar las animaciones ausentes, en las mejoras a la fluidez del recorrido del nivel, y en los modelos faltantes, también comentan que la mecánica de trepar se siente inconsistente y es fácilmente utilizable para escalar superficies verticales, lo cual hace que se pueda ignorar todas las plataformas, por lo que, por su parte el equipo de desarrollo se propone pulir dicha mecánica.

6.10.7. Sprint 7 (13-11-2024)

Milestone 3: combate avanzado.

Tareas realizadas:

- Se modeló el arma de los enemigos.
- Se crearon las animaciones del enemigo de idle, caminar, disparar, ataque cuerpo a cuerpo.
- Se continuó trabajando en el flujo de plataformas del nivel y en el diseño de la aldea.
- Se diseñó el sketch de la portada.
- Se trabajó en mejorar la mecánica de trepar.

- Se planificó la creación de granadas de repulsión usables por Gobi y un ataque en área para TA-2 (descartados).

Sprint review: En este sprint se recibió feedback importante por parte de docentes del área de arte y de estudiantes avanzados en este fin en una jornada impulsada por parte del docente, la mayor parte de este feedback fue acerca del arte presente en el juego, las mejoras propuestas fueron sobre el aspecto del personaje, las texturas del mapa y el agregado de más assets para que el mapa se vea más vivo.

Sprint retrospective: En base al feedback el equipo de arte se propuso implementar las mejoras propuestas además de agregar la UI correspondiente a las mejoras del robot. El equipo de desarrollo se propuso mejorar los controles para controlar a TA-2 e implementar la plataforma móvil que conduce al final del nivel.

6.10.8. Sprint 8 (27-11-2024)

Milestone 4: preparación de entrega final.

Tareas realizadas:

- Se trabajó en el arte de UI.
- Se texturizó la estructura del objetivo final del nivel.
- Se planificó la creación del efecto visual para los ojos de TA-2 (descartado).
- Se diseñó la portada.
- Se mejoraron los controles para dar órdenes de interacción a TA-2.
- Se implementó la lógica para plataformas móviles.
- Se creó la animación de uso de terminal.
- Se crearon objetivos secundarios en el nivel (puntos de excavación).

Sprint review: El docente realizó algunas correcciones de cara al *beta testing* que ocurrió el 11 de diciembre de 2024 en Universidad ORT, como la inclusión de todas las animaciones al juego, el acabado de texturizados, las pantallas de pausa e inicio, mejoras a los controles.

Sprint retrospective: Se definen las tareas de cara al *beta testing*, el enfoque se aboca mayormente en corregir errores y pulir las partes que no contaban con el acabado deseado, también se añaden instrucciones a modo de *placeholder* para que los jugadores comprendan los controles.

6.10.9. Sprint 9 (11-12-2024)

Milestone 4: preparación de entrega final.

Tareas realizadas:

- Se texturizó el modelo de las terminales.
- Se creó el material para caminos de piedra en el suelo del terreno.
- Se diseñaron iconos para comportamientos de TA-2.
- Se creó la pantalla con la explicación de controles en el menú de pausa.
- Se creó la animación de caída de Gobi.
- Se continuó trabajando en los objetivos secundarios del nivel.

Sprint review: Esta semana se realizó el beta testing, aunque las opiniones fueron bastante variadas, la opinión fue bastante positiva, los puntos más destacables a corregir fueron mayor claridad en cuanto a las acciones de TA-2, mayor claridad general acerca de qué ocurre, y mejoras generales al movimiento.

Sprint retrospective: En cuanto a las actividades para el siguiente sprint, considerando la indisponibilidad de los miembros del equipo, se planeó un número muy reducido de tareas, entre estas agregar los indicadores de los estados de los enemigos.

6.10.10. Sprint 10 (25-12-2024)

Milestone 5: entrega final.

Tareas realizadas:

- Se planificó crear la animación de Gobi cargando a TA-2 (descartada).
- Se diseñó el indicador de los enemigos en la UI.
- Se planificó la implementación de la mecánica de Gobi de empujar cajas (descartada).
- Se planificó el diseño del ataque cuerpo a cuerpo de Gobi (descartado).
- Se planificó el diseño de ataque en sigilo de Gobi (descartado).
- Se planificó la mecánica de arena movediza (descartada).
- Se continuó trabajando en los objetivos secundarios del nivel.

Sprint review: Dado que el curso de los artistas finalizó y teniendo en cuenta las fechas no se dio en este sprint.

Sprint retrospective: Se discutió acerca del feedback obtenido en el *beta testing* y se planearon actividades del lado del equipo de arte para mejorar la claridad de lo que ocurría en todo momento en el nivel, incluyendo un HUD (basándose en el feedback del beta testing), y maniquíes de práctica para la zona inicial del juego, es decir, la aldea.

6.10.11. Sprint 11 (08-01-2025)

Milestone 5: entrega final.

Tareas realizadas:

- Se planificó la cinemática de llegada y salida al nivel (descartada).
- Se planificó la integración de la animación de Gobi cargando a TA-2 (descartada).
- Se planificó el modelo de la tablet de Gobi (descartada).
- Se creó el material para las montañas.
- Se planificó la creación del material para arena movediza (descartada).
- Se modelaron los maniquíes de prueba de la aldea.
- Se diseñó el arte de las barras de vida en el HUD.
- Se planificó el diseño del contador de recursos en la tablet (descartado).
- Se planificó la integración de la animación de ataque cuerpo a cuerpo de Gobi (descartada).
- Se planificó la implementación de la mecánica de ataque cuerpo a cuerpo de Gobi (descartada).

Sprint review: No se realizó dadas las fechas y la ausencia de miembros del equipo.

Sprint retrospective: Siendo que no se encontraba todo el equipo presente, se propuso un número más reducido de tareas para completarlas cumpliendo los tiempos, se habló de corregir varias cosas, crear un nivel de tutorial en la zona inicial y de las actividades necesarias para la entrega del equipo de arte.

6.10.12. Sprint 12 (22-01-2025)

Milestone 5: entrega final.

Tareas realizadas:

- Se mejoró la mecánica y animación de trepar de Gobi.
- Se diseñó el nivel tutorial.
- Se planificó el diseño de variante de tutorial (descartado).
- Se trabajó en la presentación del stand en ORT.

Sprint review: Se mostró el juego a la tutora y nos recomendó coordinar una reunión con el docente, realizó comentarios acerca de tener *feedback* en la jugabilidad previos a la entrega, además de remarcar la importancia del *feedback* del docente.

Sprint retrospective: El equipo conversó acerca de los puntos a abarcar de cara a la entrega final de los artistas, se comentó la importancia del tutorial, se comentó acerca de la importancia de agregar un nuevo comportamiento a los enemigos.

6.10.13. Sprint 13 (05-02-2025)

Milestone 5: entrega final.

Tareas realizadas:

- Se implementó el final de la demo.
- Se integraron las animaciones del enemigo ataque cuerpo a cuerpo, despertarse, muerte.
- Se integró la animación de ragdoll de Gobi.
- Se mejoró el nivel tutorial agregándole formas de probar las habilidades.
- Se creó la animación del tanque de agua del final del nivel.
- Se grabaron videos sobre el juego para la entrega de artistas.
- Se trabajó en la presentación del stand en ORT.
- El equipo de desarrollo trabajó en el informe de avance de la carrera de Licenciatura que se entregó el 13/02/2025.

6.10.14. Sprint 14 (19-02-2025)

Milestone 6: entrega final licenciatura.

Este sprint se utilizó para la preparación de la revisión, por lo que no se realizaron tareas, se trabajó en la documentación, en la presentación para la revisión y se ensayó la presentación.

6.10.15. Sprint 15 (05-03-2025)

Milestone 6: entrega final licenciatura.

- Se corrigió la detección de si el jugador tocó la zona que lleva al final del nivel, que no funcionaba.
- Se arregló un error que hacía que el juego repitiera el tutorial si se ganaba sin salir de la aplicación.
- Se creó un script para limitar fps manualmente para pruebas.
- Se revisó el duplicado de mods y scrap al agarrar con bajos fps.
- Se corrigió un error que provocaba que los drops de montículos excavables cayeran bajo el terreno.

6.10.16. Sprint 16 (19-03-2025)

Milestone 6: entrega final licenciatura.

Se trabajó en la documentación.

7. Gestión de la configuración

Para el proyecto se usó git y GitHub como repositorios ya que son confiables, fácil de usar, gratuitos y el equipo tenía cierto nivel de familiaridad con su uso.

Para intercambiar archivos relevantes por fuera de git se usó Google Drive, en situaciones donde los integrantes querían planear la realización de alguna funcionalidad entre todos. Esto permitía mantener prolijo el repositorio. Principalmente se usó para almacenar e intercambiar modelos, texturas, imágenes, builds y documentos a entregar.

7.1. Control de versiones

Se usó versionado semántico para numerar las versiones: a.b.c

a: número de release. Será 0 durante todo el desarrollo. Se cambiará a 1 cuando se distribuya la primera versión estable completa.

b: milestone. Cambia para representar el milestone actual.

c: bug fix. Incrementa al corregir bugs menores.

Cada número se reinicia a cero cuando el número anterior cambia.

7.2. Estructura de ramas

Para organizar el repositorio se comenzó con el estándar master y develop, y estableció que se crearían ramas cuando fuera necesario para cada funcionalidad, además de otras ramas intermedias para cada miembro del equipo, de forma que pudieran explorar la herramienta y verificar sus cambios sin temor a crear conflictos con otras ramas.

Las ramas de funcionalidades se separaron en dos categorías, feature/, conteniendo cambios de código, y art/, conteniendo cambios de modelos, animaciones y texturas.

En los casos donde surgían conflictos al unir ramas, el equipo se reunía y se discutía entre todos la mejor forma de resolverlos.

Se estableció que las ramas nacidas de feature y art debían volcarse en develop cuando se completara un cambio relevante y fuera necesario que los demás integrantes actualizaran el proyecto. Desde develop se volcaría hacia master cuando se requiriera entregar una versión estable.

8. Conclusiones

Este proyecto fue una experiencia de aprendizaje muy valiosa para todos los integrantes. La oportunidad de colaborar con estudiantes de otras áreas presentó un desafío interesante y refleja la realidad de la industria fuera de lo académico, por lo que esta oportunidad de interactuar es extremadamente educativa.

El desarrollo de *Not an Oasis* permitió aplicar de manera práctica conocimientos adquiridos a lo largo de la carrera, integrando programación, diseño de videojuegos, trabajo en equipo y gestión de proyectos. A lo largo del proceso se enfrentaron desafíos tanto técnicos como organizativos, que fueron abordados mediante planificación iterativa, documentación constante y una gestión activa de riesgos.

El enfoque de jugabilidad basado en la relación entre el jugador y su compañero robótico resultó ser un elemento diferenciador clave, que enriqueció las posibilidades estratégicas y de exploración del juego. Además, la implementación de mejoras modulares aportó profundidad a la personalización y rejugabilidad, consolidando una experiencia lúdica coherente con el entorno postapocalíptico planteado.

El uso de herramientas como Unity y metodologías ágiles resultó adecuado para el tamaño y necesidades del equipo, permitiendo mantener un ritmo de desarrollo sostenible y adaptable. A pesar de limitaciones de tiempo y recursos, se logró construir un prototipo funcional que refleja la visión original del proyecto y sienta una base sólida para su posible expansión futura.

Finalmente, este proyecto no solo permitió desarrollar un producto interactivo con valor lúdico y técnico, sino que también fortaleció competencias clave para el trabajo profesional, como la resolución de problemas, la toma de decisiones en contexto real, y la colaboración efectiva en equipo.

9. Referencias bibliográficas

[1] J.L.G. Sanchez. “Jugabilidad: Caracterización de la experiencia del jugador en videojuegos”, M.S. tesis, Dept. Lenguajes y Sistemas Informáticos, Universidad de Granada, Granada, España, 2010.

[2] Project Management Institute, Mohammed Ahmad S Al-Shamsi, Project Management Body of Knowledge (2017).

[3] Pressman, R. S. (2010). Software Engineering: A Practitioner’s Approach (7th ed.). McGraw-Hill.

[4] R. Martin, and J. Coplien. *Clean code: a handbook of agile software craftsmanship*. Prentice Hall, Upper Saddle River, NJ etc., (2009)

10. Anexo

10.1. Descarga del juego

Página de itch.io donde el juego será actualizado en el futuro.

Link: https://bybouniverse.itch.io/not-an-oasis

Build original, producida al momento de la entrega de este documento: Build

10.2. GDD (Game Design Document)

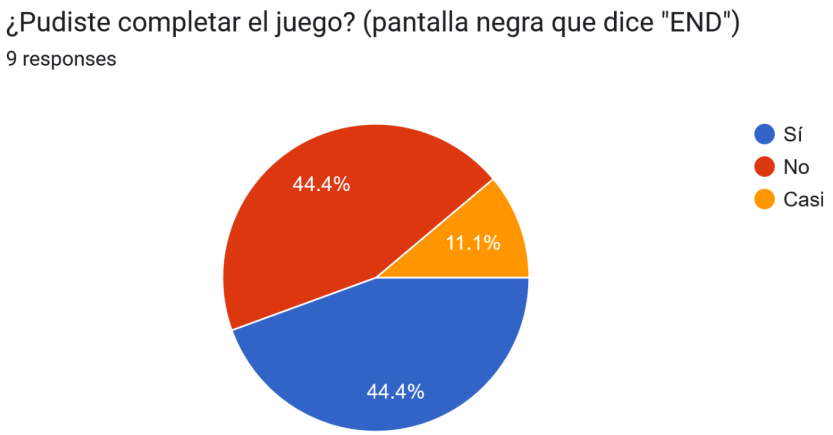
El GDD es un documento presentado por el equipo de arte en su entrega final. Contiene toda la información relacionada al juego, su historia, arte conceptual, planes pasados, descripción de mecánicas, etc.

Link: NotAnOasisGDD.pdf

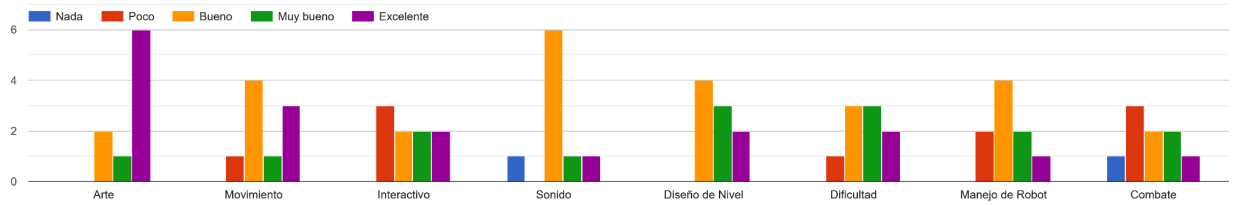
10.3. Lista de tareas

Link: Tasks

10.4. Encuestas

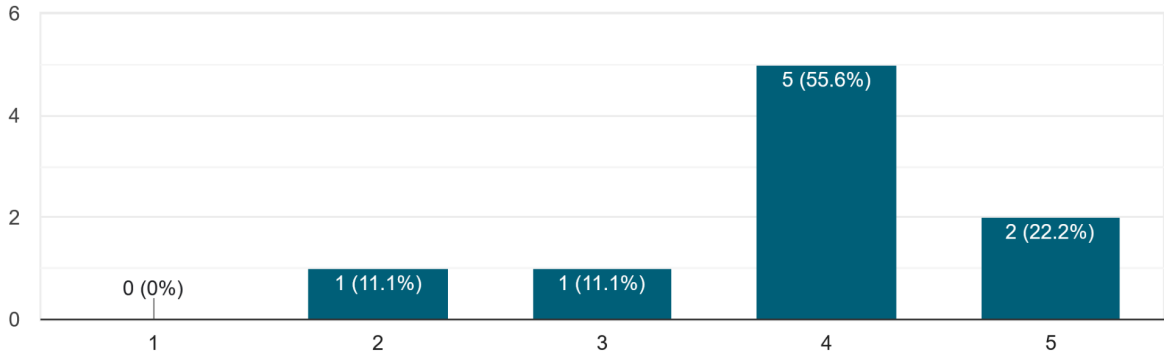


¿Cuánto te gustó el juego con respecto a...?



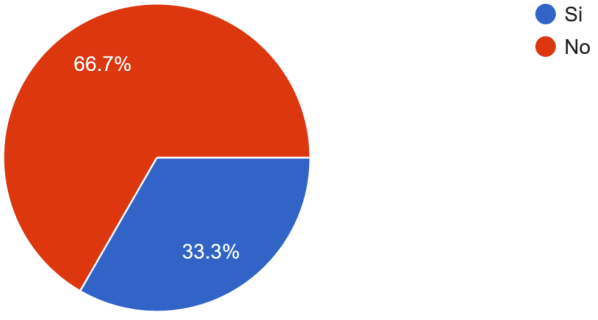
Calificación General (hay más formulario después de esto, si quieres dar más detalles)

9 responses

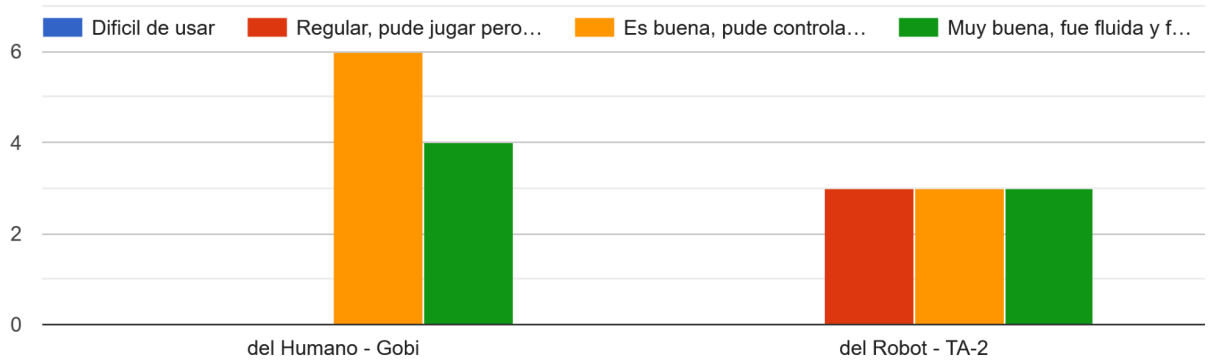


¿Te encontraste con algún error en el juego?

9 responses



¿Que te pareció la movilidad?



10.5. Trailer

Trailer NotAnOasis.mp4