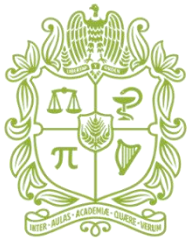
DOCUMENTACIÓN PROYECTO FINAL

Ingeniería de Software I



UNIVERSIDAD
NACIONAL
DE COLOMBIA

Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial

Presentado por:

Nicolas Quezada Mora - nquezada@unal.edu.co

Sharick Yelixa Torres Monroy - shtorres@unal.edu.co

Laura Sofia Vargas Rodriguez - lavargasro@unal.edu.co

Jeronimo Bermudez Hernandez- jebermudez@unal.edu.co

Docente:

Oscar Eduardo Alvarez Rodriguez - oalvarezr@unal.edu.co

9 de febrero, 2025

1. Levantamiento de requerimientos:

SCRUM'S Castle es un videojuego de simulación y estrategia ambientado en la época medieval, donde los jugadores gestionan un reino aplicando la metodología Scrum, los jugadores asignan tareas a diferentes personajes (caballeros, herreros, campesinos, escribas) para completar proyectos como la construcción de herramientas, el desarrollo de tecnologías o la defensa del reino.

La idea de SCRUM'S Castle surge al identificar dificultades comunes para aplicar metodologías ágiles como Scrum en proyectos reales, se decidió crear un videojuego de simulación y estrategia medieval que enseñara Scrum de manera interactiva y divertida, tras reuniones y votaciones, el equipo eligió esta idea y asignó roles para su desarrollo, el juego busca resolver problemas como la falta de experiencia práctica y la gestión ineficiente de tareas, ofreciendo una experiencia gamificada donde los jugadores gestionan un reino aplicando Scrum, los usuarios esperan un sistema educativo y entretenido, mientras que nosotros, como equipo, esperamos aprender nuevas tecnologías, mejorar nuestras habilidades de trabajo en equipo y contribuir con una herramienta innovadora para la comunidad.

¿Qué esperamos del desarrollo del proyecto?

- **Sharick Torres:**
  Espero que en el desarrollo de Scrum's Castle podamos poner en práctica los conceptos teóricos que hemos aprendido en clase en cuanto a la gestión de proyectos en la vida real y, además, nos permita plantear una manera interesante de aprender Scrum de manera práctica. Por otro lado, dado que no he realizado proyectos en el área de los videojuegos, considero esta una oportunidad de conocer nuevas herramientas y adquirir más habilidades.

- **Jeronimo Bermudez:**
  Para mí, este proyecto representa una oportunidad única para explorar cómo la gamificación puede ser una herramienta poderosa para la educación, espero que Scrum's Castle no solo sea divertido para los jugadores, sino que también logre transmitir de manera efectiva los principios de Scrum, personalmente, estoy interesado en profundizar en el desarrollo de mecánicas de juego que reflejen la metodología ágil, y espero que este proyecto nos permita innovar en la forma en que se enseña

- **Nicolas Quezada:**
  Lo que más espero del desarrollo de Scrum's Castle es poder combinar mi interés por los videojuegos con el aprendizaje de metodologías ágiles, creo que este proyecto tiene un gran potencial para ser una herramienta educativa innovadora, y estoy emocionado por contribuir en la creación de un juego que sea tanto entretenido como instructivo, también espero que este proceso nos permita aprender nuevas tecnologías y mejorar nuestras habilidades de programación y diseño.

- **Sofia Vargas:**
  Lo que más espero del desarrollo de Scrum's Castle es poder aplicar los conocimientos que hemos adquirido en la universidad a un proyecto real, me interesa especialmente la parte de la gestión de tareas y la asignación de roles, ya que creo que es fundamental para el éxito de cualquier proyecto, espero que este juego nos permita experimentar con mecánicas innovadoras que hagan que el aprendizaje de Scrum sea más accesible y atractivo.

Especificación de requerimientos

Requerimiento	Datos requeridos	Procesos asociados
Movimiento del Jugador: Permitir al jugador controlar el movimiento de su personaje o equipo dentro del mundo del juego	- Dirección del movimiento (arriba, abajo, izquierda, derecha). - Velocidad de movimiento. - Animaciones de movimiento (caminar, correr, etc.).	- Implementar controles intuitivos para el movimiento (teclado, ratón, mando, pantalla táctil). - Asegurar que el movimiento sea fluido y responsivo.

Interacción con el Entorno: Permitir al jugador interactuar con objetos, personajes y elementos del entorno.	- Objetos interactuables (recursos, herramientas, edificios). - Acciones disponibles (recolectar, construir, hablar, etc.).	- Implementar un sistema de detección de interacciones (por ejemplo, al acercarse a un objeto o personaje). - Mostrar indicadores visuales de interacción (iconos, mensajes). - Permitir al jugador realizar acciones específicas al interactuar (por ejemplo, recolectar madera al interactuar con un árbol).
Interacción con el Equipo: Permitir al jugador interactuar con los miembros de su equipo para asignar tareas y gestionar recursos.	- Miembros del equipo (herrero, granjero, alquimista, arquero). - Tareas asignables a cada miembro del equipo. - Estado de cada miembro del equipo (ocupado, disponible, cansado)	- Implementar un menú de gestión del equipo. - Permitir al jugador asignar tareas específicas a cada miembro del equipo. - Mostrar el estado actual de cada miembro del equipo (por ejemplo, ocupado recolectando madera).
Control de Tareas: Supervisar la gestión de tareas en tiempo real**.**	- Nombre de la tarea. - Descripción breve de la tarea. - Estado de la tarea (Por hacer, En progreso, Hecho). - Tiempo estimado para completar la tarea. - Personaje asignado (herrero, granjero, alquimista, arquero).	- Asignar tareas específicas a cada personaje. - Generar alertas cuando una tarea está cerca de exceder el tiempo estimado (notificaciones en pantalla). - Visualizar el estado de las tareas en un tablero Kanban sencillo y filtrable (por estado, personaje asignado, etc.).

Progreso: Supervisar el avance del equipo y del reino	- Tareas completadas en el sprint. - Tareas no completadas y razones. - Recursos utilizados y disponibles. - Tiempo total del sprint. - Objetivos del reino alcanzados y pendientes.	- Generar un resumen visual del sprint - Permitir al jugador identificar qué funcionó bien, qué no funcionó y cómo mejorar. - Ofrecer recomendaciones automáticas basadas en el desempeño del jugador. - Mostrar el progreso general del reino en un menú específico, con indicadores claros de avance.
Control de Recursos: Supervisar la disponibilidad de recursos en tiempo real.	- Nombre del recurso (madera, hierro, comida, oro). - Cantidad disponible en stock. - Descripción breve del recurso. - Icono o imagen del recurso (opcional).	- Actualizar automáticamente el inventario de recursos cuando se completan tareas. - Generar alertas cuando un recurso esté cerca de agotarse (notificaciones en pantalla). - Visualizar el estado de los recursos en un formato sencillo y filtrable (por tipo, cantidad, etc.). - Permitir al jugador agregar, modificar o eliminar recursos desde un formulario.
Retrospectivas: Permitir al jugador revisar y mejorar su gestión después de cada sprint.	- Tareas completadas en el sprint. - Tareas no completadas y razones. - Recursos utilizados y disponibles. - Tiempo total del sprint.	- Generar un resumen visual del sprint - Permitir al jugador identificar qué funcionó bien, qué no funcionó y cómo mejorar. - Ofrecer recomendaciones

		automáticas basadas en el desempeño del jugador.
Personalización del Equipo: Permitir al jugador gestionar y mejorar su equipo de personajes.	- Nombre del personaje (herrero, granjero, alquimista, arquero). - Habilidades del personaje. - Nivel de experiencia. - Equipamiento actual (herramientas, armas, armaduras).	- Permitir al jugador asignar tareas específicas a cada personaje. - Ofrecer opciones para mejorar habilidades y equipamiento mediante recompensas. - Generar alertas cuando un personaje está cerca de subir de nivel.
Sistema de Recompensas: Motivar al jugador con recompensas por completar tareas y objetivos	- Tipo de recompensa (monedas, recursos, mejoras). - Cantidad de recompensa. - Condiciones para obtener la recompensa (completar tareas, alcanzar objetivos).	- Asignar automáticamente recompensas al completar tareas o objetivos. - Permitir al jugador visualizar las recompensas obtenidas en un menú específico. - Ofrecer opciones para canjear recompensas por mejoras o recursos adicionales.
Castigos Medievales: Aplicar consecuencias por el incumplimiento de tareas y objetivos.	- Tipo de castigo (pasar una noche en el calabozo, realizar una tarea adicional, perder recursos, etc.). - Condiciones para aplicar el castigo (incumplimiento de tareas, agotamiento de recursos críticos). - Personaje o equipo afectado por el castigo.	- Aplicar automáticamente castigos cuando no se cumplen las tareas o se agotan los recursos críticos. - Mostrar una animación o escena que represente el castigo (por ejemplo, el personaje siendo encerrado en el calabozo).

		- Notificar al jugador cuando se aplica un castigo, explicando la razón y las consecuencias.
Mejoras y Actualizaciones: Aplicar Permitir al jugador mejorar las habilidades y equipamiento del equipo	- Tipo de mejora (herramientas, armas, habilidades). - Costo de la mejora (recursos, monedas). - Beneficios de la mejora (aumento de eficiencia, reducción de tiempo en tareas).	- Ofrecer opciones de mejora en un menú específico. - Permitir al jugador adquirir mejoras con los recursos obtenidos. - Aplicar automáticamente los beneficios de las mejoras al equipo.
Fabricación de Herramientas: Permitir al jugador fabricar herramientas para mejorar la eficiencia del equipo.	- Tipo de herramienta (hacha, pico, martillo, etc.). - Materiales necesarios para fabricar cada herramienta. - Tiempo necesario para fabricar cada herramienta. - Beneficios de cada herramienta (aumento de eficiencia en tareas específicas).	- Ofrecer opciones de fabricación en un menú específico. - Permitir al jugador asignar personajes para fabricar herramientas. - Aplicar automáticamente los beneficios de las herramientas al equipo.
Gestión de Inventario	- Capacidad de almacenamiento del inventario. - Cantidad de cada material y herramienta en el inventario. - Ubicación del inventario (almacén central, edificios específicos).	- Generar alertas cuando el inventario esté cerca de su capacidad máxima. - Permitir al jugador expandir la capacidad del inventario construyendo almacenes. - Visualizar el estado del inventario en un formato sencillo y filtrable (por tipo de material, cantidad, etc.).

2. Análisis de requerimientos:

Identificación y organización utilizando MoSCoW

Must have (imprescindibles):

- **Movimiento del Jugador (2 días):** Implementa el desplazamiento fluido del personaje principal por el entorno medieval, se utilizarán las herramientas de Godot para programar controles intuitivos y animaciones responsivas, garantizando una buena experiencia de juego desde el primer contacto
- **Interacción con el Entorno (3 días):** Permite al jugador interactuar con elementos del escenario, como puertas, palancas y objetos decorativos, lo que desencadena eventos o abre nuevas áreas, la detección de colisiones y señales en Godot se aprovechará para hacer estas interacciones naturales y coherentes con la ambientación.
- **Control de Tareas (3 días):** Desarrolla un tablero de gestión estilo Scrum donde se asignan, visualizan y controlan las tareas de los personajes (caballeros, herreros, campesinos, escribas), esta interfaz, diseñada en Godot, ayudará a los jugadores a planificar y seguir el progreso de los proyectos del reino.
- **Castigos Medievales (5 días):** Integra mecánicas de penalización inspiradas en la época medieval que se activan ante el incumplimiento o retraso en las tareas, estos castigos no solo añaden desafío y realismo, sino que también refuerzan la importancia de la organización y el cumplimiento de plazos, siendo un componente central en la simulación Scrum
- **Interacción con el Equipo (2 días):** Habilita la comunicación y coordinación entre los jugadores y/o personajes, simulando reuniones y votaciones típicas de Scrum. Se implementará un sistema básico de mensajería y señales visuales en Godot, fundamental para fomentar la colaboración y el trabajo en equipo.

Should have (importantes):

- **Progreso (2 días):** Presenta indicadores visuales (barras, gráficos o semáforos) que muestren el avance de los proyectos y tareas, esta funcionalidad brinda feedback inmediato al jugador sobre el rendimiento del equipo y la evolución del reino, ayudando a priorizar las acciones en cada sprint.
- **Control de Recursos (2 días):** Permite la gestión y asignación de recursos (como oro, materiales y energía) necesarios para la construcción y mejoras del

reino, la implementación en Godot incluirá un sistema sencillo de inventario que incentive decisiones estratégicas en tiempo real.

- **Retrospectivas (2 días):** Implementa una fase post-sprint donde se revisa el desempeño del equipo, identificando aciertos y áreas de mejora, esta herramienta, inspirada en la práctica Scrum, ayuda a ajustar estrategias y a aprender de la experiencia en cada ciclo, mejorando la eficiencia del equipo.
- **Sistema de Recompensas (3 días):** Crea un mecanismo de incentivos que premie la culminación exitosa de tareas y proyectos, las recompensas pueden incluir mejoras, bonificaciones o desbloqueos de contenido extra, aumentando la motivación y aportando un elemento lúdico al aprendizaje de la metodología.

Could have (deseables, opcionales o para futuras iteraciones):

- **Personalización del Equipo (5 días): s**e permitirá al jugador personalizar los personajes del equipo, modificando aspectos visuales, nombres y habilidades básicas. Esta funcionalidad aportará un toque personal y aumentará la inmersión en la ambientación medieval.
- **Mejoras y Actualizaciones (3 días):** Se desarrollará un sistema que permita la evolución y mejora de personajes y estructuras del reino, basándose en logros y experiencia acumulada. Esto agregará profundidad a la simulación y motivará la progresión del juego.
- **Fabricación de Herramientas (5 días):** Se implementará una mecánica de crafting que permita crear y actualizar herramientas o armas a partir de recursos recolectados. Esta característica aportará una capa adicional de estrategia y personalización al juego
- **Gestión de Inventario (3 días):** Se creará un sistema avanzado de administración de recursos y objetos, facilitando el manejo de materiales y equipamiento. Esta herramienta ayudará al jugador a optimizar el uso de sus activos dentro del reino.

Won’t have (no ahora):

- **Eventos Aleatorios Complejos (13 días):** Aunque se contempló incluir eventos dinámicos que alteren el curso del juego, su complejidad supera el alcance del sprint actual y se pospone para una futura expansión.
- **Misiones Especiales (8 puntos):** Se había considerado incorporar misiones secundarias con objetivos únicos para diversificar la experiencia, pero se descartó en esta versión para mantener el enfoque en las funcionalidades esenciales.

Escala de Fibonacci y argumentos de estimación

Requerimiento	Prioridad	Estimación (Días)	**Justificación de estimación**
Movimiento del Jugador	MUST	2	La implementación se apoya en las funcionalidades nativas de Godot para detección de entradas y físicas básicas, lo que la hace relativamente directa y de bajo nivel de complejidad.
Interacción con el Entorno		3	Requiere coordinar detección de colisiones y activación de eventos mediante señales, añadiendo una capa de lógica sobre la base del movimiento, lo que incrementa moderadamente la dificultad.
Control de Tareas		3	Involucra el desarrollo de una interfaz dinámica para asignar y seguir tareas, basándose en patrones de diseño de UI y lógica de negocio comunes, lo que implica un nivel de complejidad moderado.
Castigos Medievales		5	La integración de reglas condicionales, efectos visuales y sonoros, y la coordinación de penalizaciones en función del desempeño del jugador conlleva un mayor desafío en la implementación, justificando un nivel intermedio-alto.
Interacción con el Equipo		2	Se trata de una funcionalidad básica de comunicación y coordinación mediante interfaces simples y señales visuales, con una complejidad mínima al aprovechar herramientas estándar de Godot.

Progreso	SHOULD	2	La representación visual de indicadores de avance se basa en componentes UI preexistentes, lo que la hace una tarea de baja complejidad.
Control de Recursos		2	Consiste en implementar una lógica sencilla para el seguimiento y asignación de recursos, utilizando una interfaz simple que no demanda una integración compleja.
Retrospectivas		2	La funcionalidad se centra en recopilar y mostrar información básica sobre el rendimiento, lo que implica una integración mínima en términos de lógica y diseño visual.
Sistema de Recompensas		3	Al involucrar la configuración de incentivos y la integración con la progresión de tareas, añade cierta complejidad extra en comparación con simples visualizaciones, ubicándose en un nivel moderado.
Personalización del Equipo	COULD	5	Implica desarrollar múltiples opciones de personalización (visual, nombres, atributos) y crear interfaces flexibles, lo que añade una capa significativa de complejidad en la integración de elementos gráficos y lógicos.
Mejoras y Actualizaciones		3	Aunque involucra la evolución de personajes y estructuras, se apoya en patrones de mejora ya existentes y es menos complejo que sistemas de crafting o inventario, ubicándose en un nivel moderado.
Fabricación de Herramientas		5	La mecánica de crafting requiere la definición de recetas, combinación de recursos y lógica de creación, lo que añade un desafío considerable en términos de diseño e

			integración.
Gestión de Inventario		3	Consiste en la organización y administración de múltiples recursos mediante una interfaz avanzada; si bien es más compleja que una simple visualización, se basa en patrones comunes de inventarios en videojuegos.
Eventos Aleatorios Complejos	WON´T	13	La integración de eventos que alteren de manera significativa el curso del juego implica una lógica dinámica y condicional compleja, coordinación de efectos visuales y cambios en la narrativa, lo que eleva su nivel de dificultad a uno muy alto.
Misiones Especiales		8	Desarrollar misiones secundarias con objetivos únicos requiere la creación de narrativas alternativas, diálogos y sistemas de recompensa específicos, lo que demanda un esfuerzo considerable en diseño y programación, ubicándola en un nivel de complejidad elevado.

3. Análisis gestión de software

Triada de Gestión de Proyectos

- **Tiempo**
  Para organizar el desarrollo del videojuego, dividiremos el trabajo en tres fases principales: **diseño, desarrollo y pruebas**. A continuación, detallamos cada una de estas fases junto con su estimación de tiempo:
  Fase 1: Diseño (5 días)
  - **Días 1-3:** Definición de mecánicas principales, flujo del juego y diseño de interfaz (UI/UX).
  - **Días 3-5:** Creación de prototipos y wireframes, documentación detallada de requerimientos.

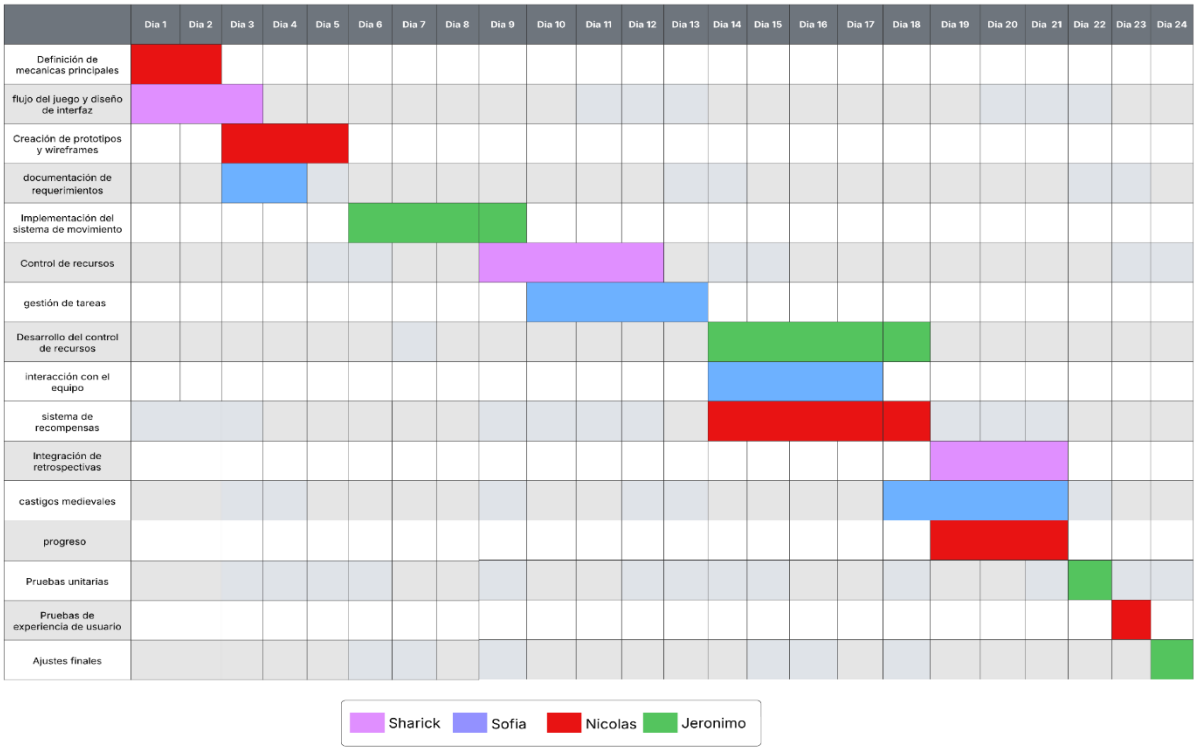
Fase 2: Desarrollo (16 días)

- **Días 6-9:** Implementación del sistema de movimiento del jugador e interacción con el entorno.
- **Días 9-13:** Control de recursos y gestión de tareas
- **Días 14-18:** Desarrollo del control de recursos, interacción con el equipo y sistema de recompensas.
- **Días 18-21:** Integración de retrospectivas, castigos medievales y progreso.

Fase 3: Pruebas y Ajustes (3 días)

- **Día 22:** Pruebas unitarias y de integración en cada módulo, corrección de bugs criticos
- **Día 23:** Pruebas de experiencia de usuario (UX) y balanceo de mecánicas.
- **Día 24:** Ajustes finales y optimización del rendimiento.

Duración total estimada: 24 días.



- **Costos**

  Equipo: Para determinar los costos, hemos considerado un equipo reducido, asumiendo que trabajaremos con desarrolladores junior y utilizaremos

herramientas gratuitas siempre que sea posible. La siguiente tabla resume los costos estimados bajo el siguiente análisis:

Mínimo
Promedio
Máximo

Rol/Especialidad	Salario promedio (COP) Mensual	Salario promedio (COP) horas	Fuentes
Desarrollador Junior	1.318.734	12.717	- https://co.indeed.com/career/programador-web/salaries - https://www.dolarapp.com/es-CO/blog/freelancer-tips/cuanto-gana-un-programador - https://talently.tech/herramientas/colombia/salario
	2.314.413	16.689	
	2.685.791	33.894	
UX/UI Designer	2.800.000	15.385	- https://co.talent.com/salary?job=ux+ui - https://www.glassdoor.com.ar/Sueldos/colombia-ux-designer-sueldo-SRCH_IL.0,8_IN54_KO9,20.htm - https://talently.tech/herramientas/colombia/salario/developer/ui-designer
	4.000.000	21 978	
	6.000.000	32.967	
Tester	1.876.750	10.312	- https://co.talent.com/salary?job=tester - https://co.indeed.com/career/tester/salaries - https://www.glassdoor.com.ar/Sueldos/bogota-colombia-test-automation-engineer-sueldo-SRCH_IL.0,15_IM1064_KO16,40.htm
	2.592.862	25.892	
	5.250.000	28.846	

Aplicación en nuestro proyecto:

Concepto	Cantidad	Costo Mensual (COP)	Duración	Costo Total (COP)
Desarrollo				
Desarrollador Junior	2	$2.314.413	1 mes	$4.628.826
UX/UI Designer	1	$4.000.000	1 mes	$4.000.000
Tester	1	$2.592.862	1 mes	$2.592.862
Costo total aproximado				

Tecnologías y herramientas:

Herramienta	Categoría	Descripción	Costo	Sitio Web
Godot	Motor de juego	Motor de juego de código abierto ideal para desarrollar juegos 2D y 3D, perfecto para proyectos que se ejecutan localmente.	Gratis	godotengine.org
GitHub	Control de versiones	Plataforma para gestionar repositorios de código y colaborar en el desarrollo.	Gratis	github.com
Trello	Gestión de proyectos	Herramienta visual para la organización de tareas y seguimiento de sprints.	Gratis	trello.com
Notion	Documentación/ organización	Plataforma colaborativa para	Gratis (Plan	notion.so

		gestionar documentos, tareas y bases de conocimiento.	Personal)	
Figma	Diseño UI/UX	Herramienta colaborativa para diseño de interfaces y prototipos de aplicaciones.	Gratis (Plan Starter)	figma.com
Aseprite	Diseño UI/UX	Editor de gráficos rasterizados, diseñado específicamente para crear y editar Sprites y animaciones píxel art.	Gratis (Trial Version)	aseprite.org

Estimación del costo total del proyecto en cada fase:

Fase	Tiempo estimado (días)	Elemento	Costo total $	Herramienta
Fase 1: Diseño	5	1 - UX/UI Designer	833.333	Trello Notion Aseprite Godot Github Figma
		2 - Desarrollador Junior	964.340	
Subtotal de fase			$1.797.673	
Fase 2: Desarrollo	16	1 - UX/UI Designer	1.000.000	
		2 - Desarrollador Junior	3.085.568	
Subtotal de fase			$4.085.568	
Fase 3: Pruebas y ajustes	3	1 - Tester	324.108	
		2 - Desarrollador Junior	578.604	

Subtotal de fase			902.712	
Total			6.785.953	

- **Alcance**

  Entregaremos un prototipo jugable centrado en las mecánicas esenciales: movimiento fluido del personaje, gestión estratégica de recursos (como alimentos y herramientas), y un sistema de recompensas vinculado a tareas diarias en un entorno medieval. La interfaz será intuitiva, con menús accesibles y retroalimentación visual clara (ejemplo: barra de progreso de tareas). Incluiremos 3 escenarios de castigos medievales (como la picota o multas en oro) y una curva de dificultad ajustada tras pruebas con usuarios reales. No incluirá música original ni diálogos complejos para priorizar el MVP (Producto Mínimo Viable).

  Incluidas en el MVP:

  - Movimiento del jugador.
  - Interacción con el entorno.
  - Control de tareas.
  - Control de recursos.

  Importantes, pero no imprescindibles para el MVP:

  - Interacción con el equipo.
  - Progreso del reino.
  - Retrospectivas.
  - Sistema de recompensas.
  - Castigos medievales.

  Opcionales (podrían añadirse en futuras actualizaciones):

  - Personalización del equipo.
  - Mejoras y actualizaciones.
  - Fabricación de herramientas.
  - Gestión de inventario.

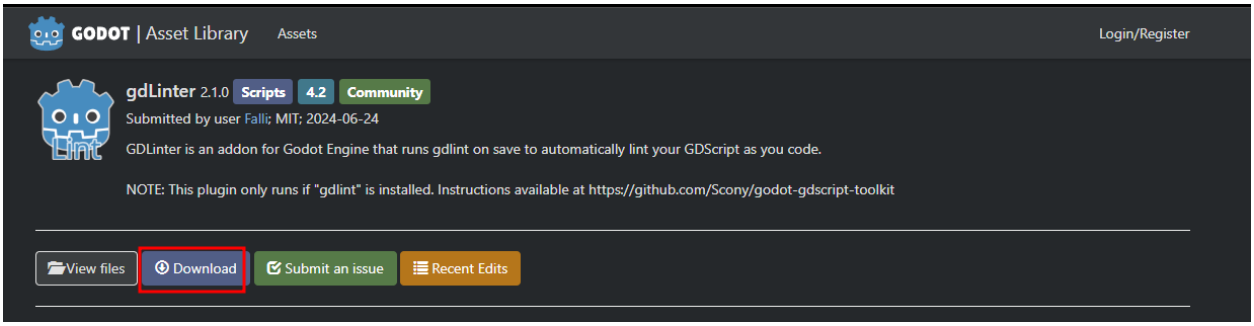
  Excluidas en esta fase:

- Eventos aleatorios complejos.
- Misiones especiales.

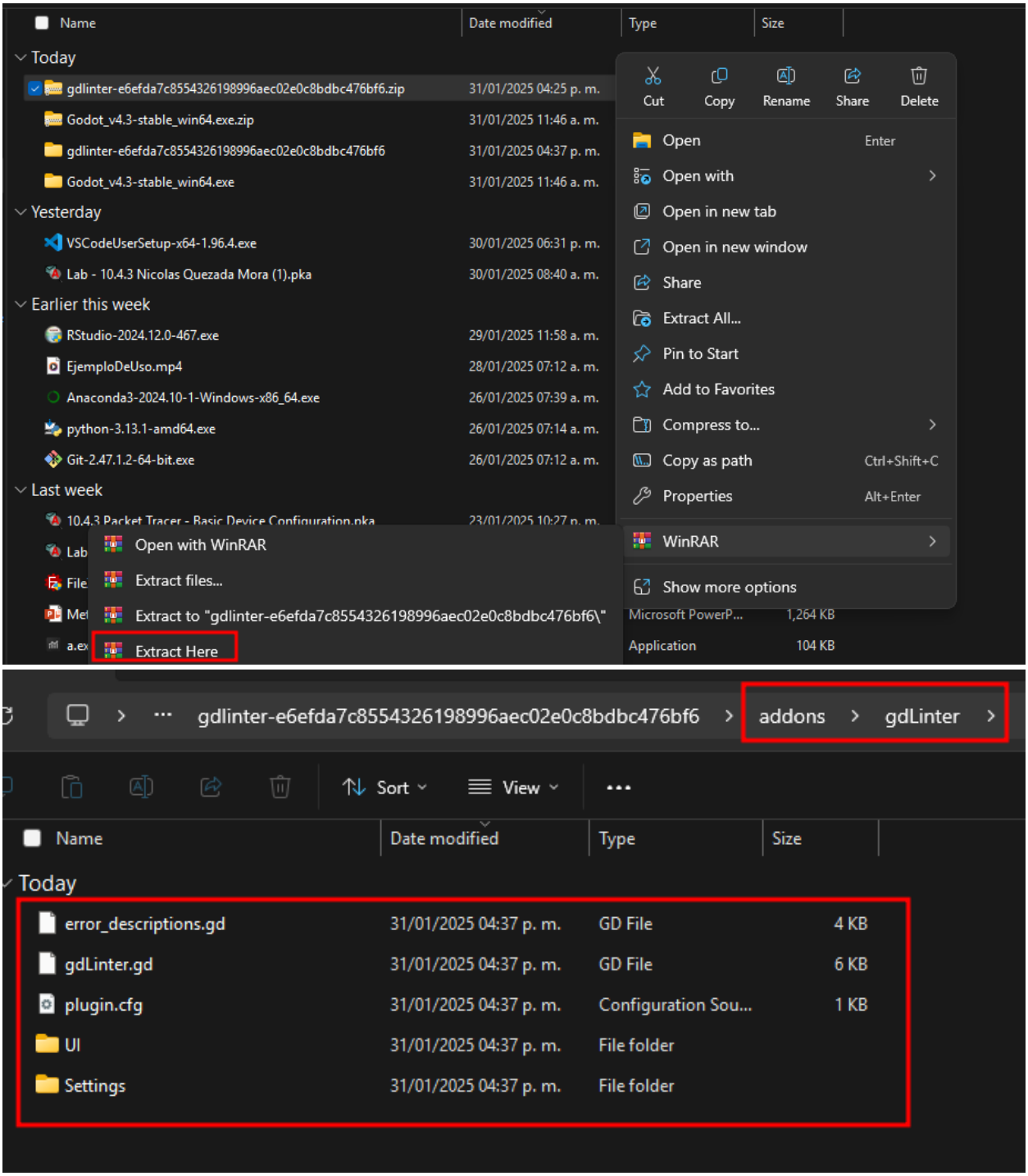
4. Clean code:

Para el linter de nuestro programa, seleccionamos el plugin para Godot denominado gdLinter, el cual se encarga de analizar y depurar el código. Este plugin revisa el código en busca de inconsistencias, errores de sintaxis y áreas de mejora, lo que facilita la optimización del rendimiento y la adherencia a las buenas prácticas de programación.

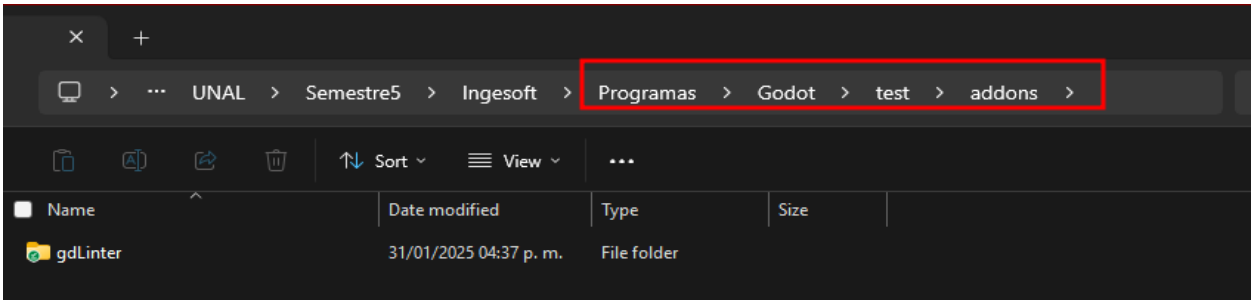
En el primer paso, lo descargamos:



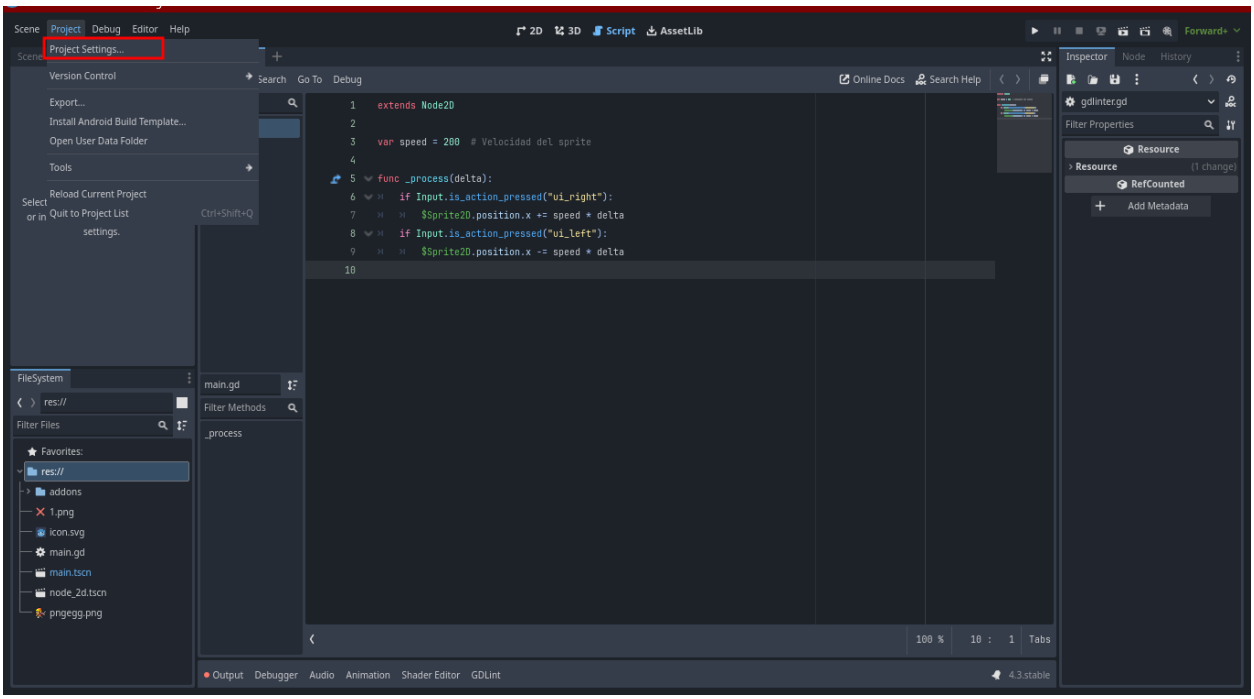
Luego se extrae:

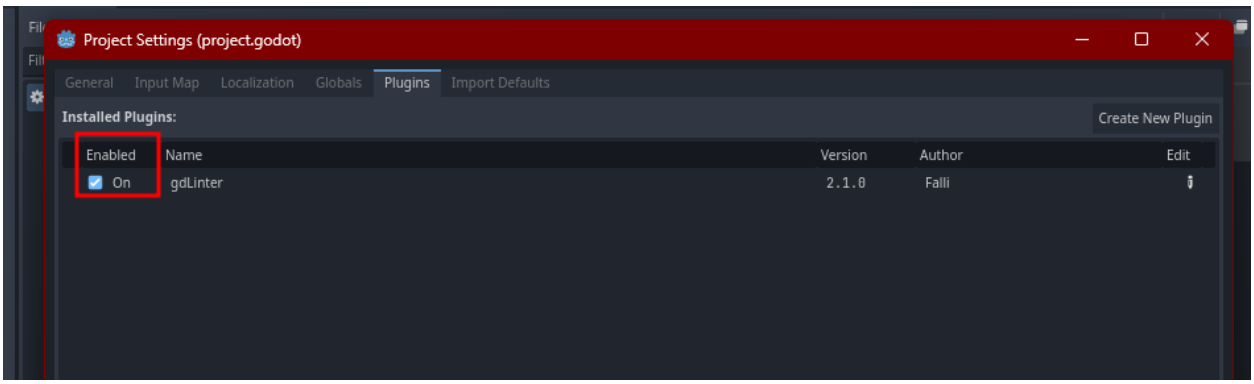


Después se incorpora en la carpeta de addons del proyecto en el que estamos trabajando:

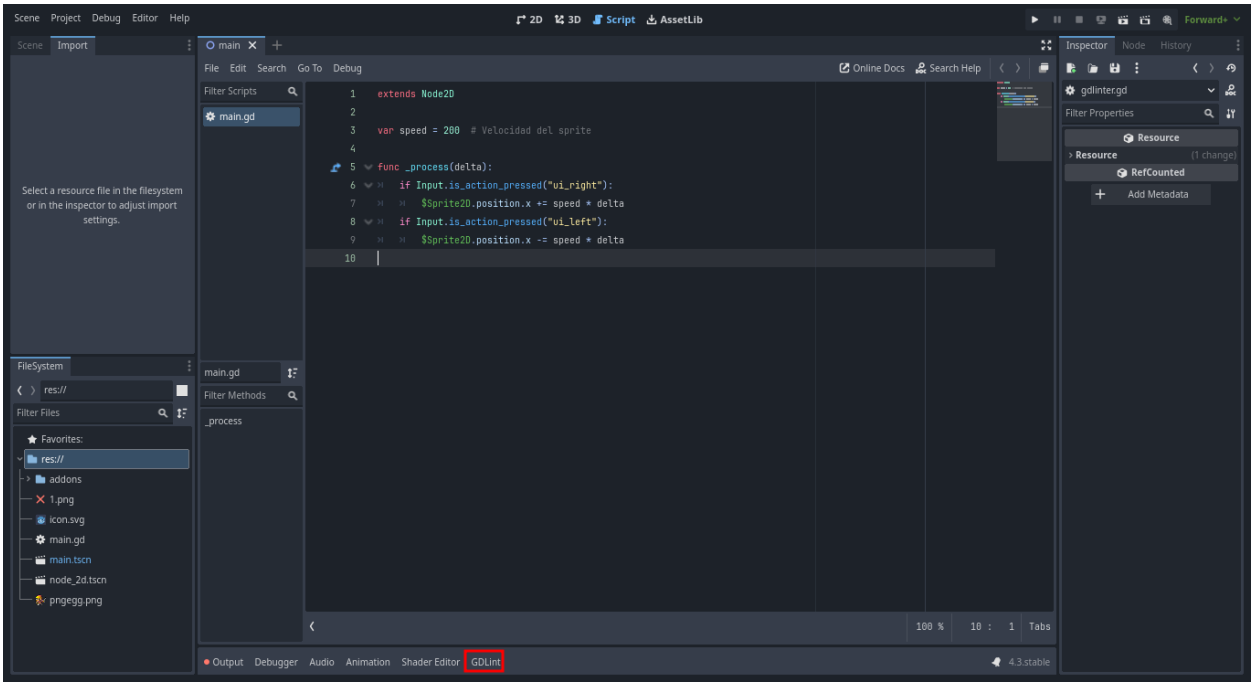


Una vez realizado este procedimiento, lo activamos desde la pestaña de plugins:

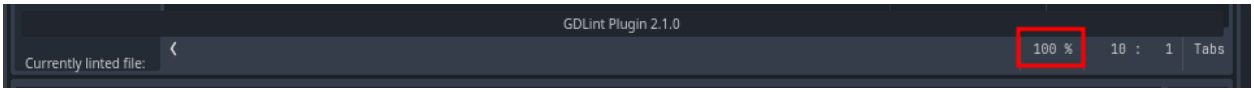




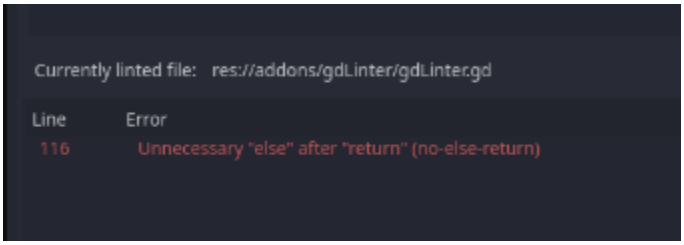
Se selecciona en la parte de abajo del entorno:



Como se puede observar, en el código inicial no se detectaron errores en el código inicial que tenemos:



A continuación, se muestra una imagen que ilustra cómo se vería el entorno en caso de que se presentara algún error:



Comentarios acerca del código en equipo:

- **Sharick Torres:**

  Considero que el código general refleja un esfuerzo coordinado, se nota que cada uno puso de su parte siendo coherentes con los aportes de los demás, aunque aún falta parte del código, hasta el momento hemos tratado de mantener un estándar común que facilite la comprensión del resto y futuras modificaciones. Sin embargo, creo que podríamos hacer mejor uso de los comentarios.

- **Jeronimo Bermudez:**

  Desde mi perspectiva, el código demuestra una buena organización y es fruto de un excelente trabajo en equipo, cada uno tenía claro su rol y eso permitió que cada parte se desarrol¿lara de forma autónoma sin confusiones, la división clara de funciones hizo que el avance fuera ordenado y acorde a lo que queríamos lograr

- **Nicolas Quezada:**

  En general, considero que la estructura lograda es uniforme y funcional, y se ajusta a los estándares que acordamos, me gustó mucho lo claro y legible que resultó el código, cada segmento cumple una función específica, estas características hicieron que fuera sencillo entender qué hace cada parte.

- **Sofia Vargas:**

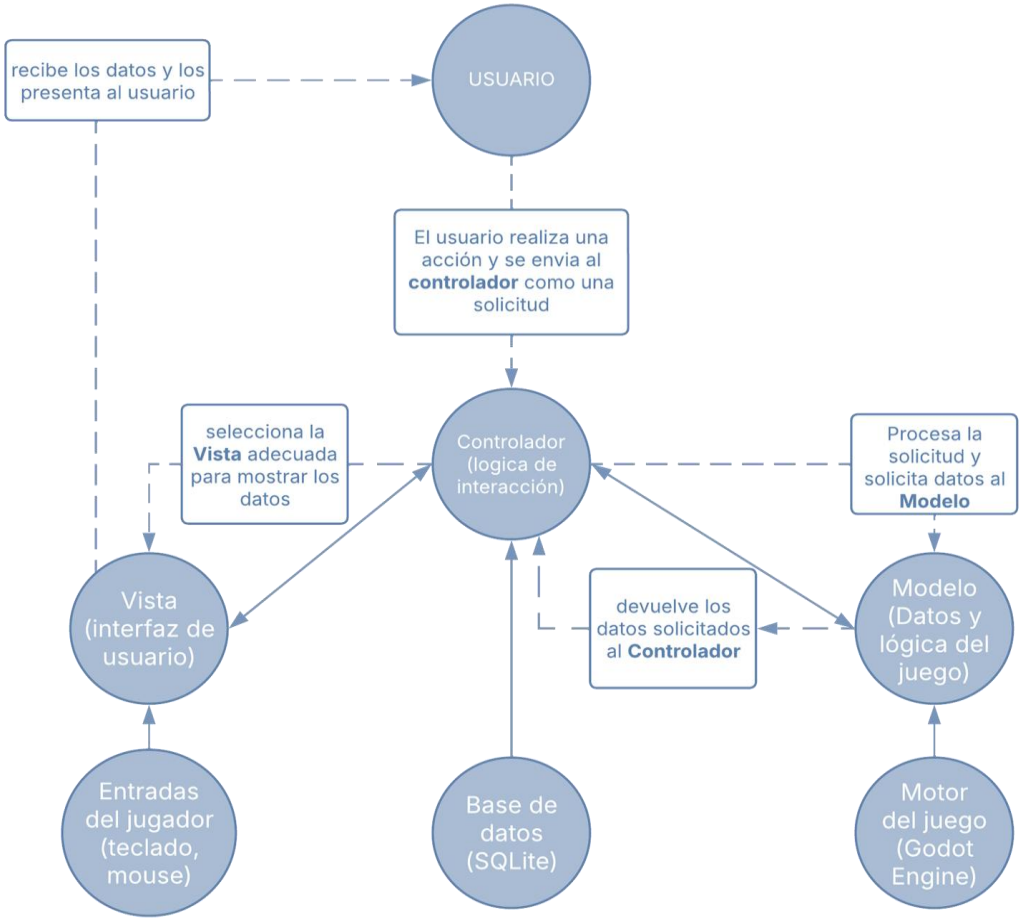
  A mi parecer, el código que hemos desarrollado resulta muy claro y sigue buenas prácticas de programación, lo que se nota en la organización y calidad de cada sección, además, he aprendido mucho al ver el trabajo de mis compañeros, sus enfoques y técnicas me han ayudado a mejorar mi propia forma de programar, la experiencia compartida ha sido muy enriquecedora para mí.

5. Diseño y arquitectura

Identificación de la Arquitectura

Para Scrum Castle, se ha seleccionado una arquitectura monolítica con un enfoque MVC (Modelo-Vista-Controlador), esta arquitectura es ideal para videojuegos single-player, ya que permite una clara separación de responsabilidades entre la lógica del juego, la interfaz de usuario y la gestión de datos, sin la necesidad de un servidor externo.

A continuación, se presenta un diagrama de arquitectura en formato textual que describe cómo interactúan los distintos componentes del sistema:



Componentes Principales

- Vista (Interfaz de Usuario):
    - Muestra la información al jugador (HUD, menús, tablero Kanban).
    - elementos visuales como animaciones, iconos y mensajes de interacción.
- Controlador:
    - Gestiona las interacciones del jugador (movimiento, clics, selecciones).
    - Recibe las entradas del jugador (teclado, ratón) y las procesa para actualizar el Modelo.
- Modelo:

    - Representa los datos del juego (estado del jugador, tareas, recursos, moral).
    - Contiene la lógica del juego, como la gestión de tareas, recursos, y eventos aleatorios.
- Base de Datos (SQLite):
    - Almacena los datos persistentes del juego (progreso del jugador, inventario, tareas).
    - Se comunica con el Modelo para guardar y recuperar información.
- Motor del Juego (Godot Engine):
    - Gestiona la renderización del juego, la física, y las animaciones.
    - El Controlador se comunica con el motor para actualizar el estado del juego.

Justificación

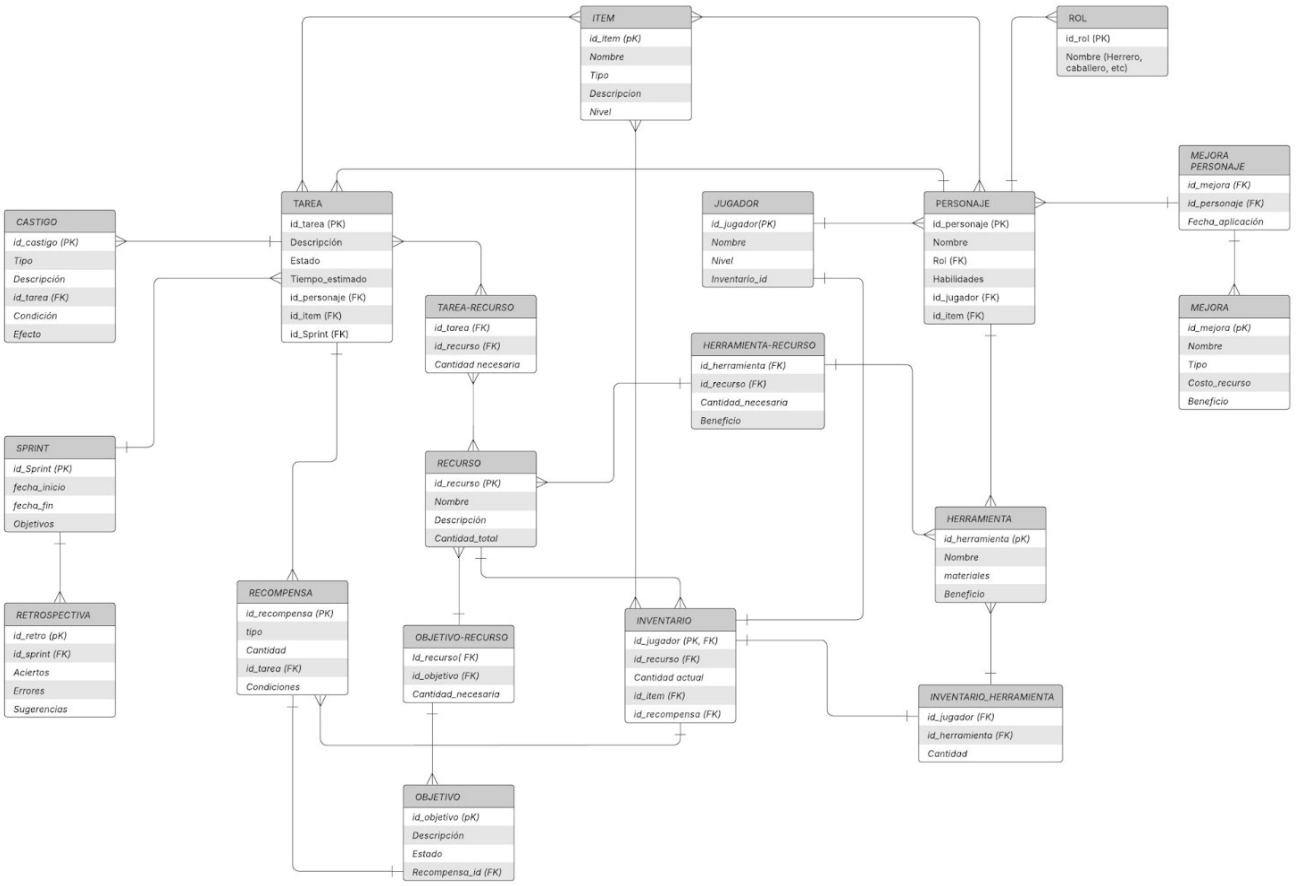
- El enfoque MVC permite una clara separación entre la lógica del juego (Modelo), la interacción con el jugador (Controlador), y la presentación visual (Vista), esto facilita el desarrollo y el mantenimiento del código**.**

- Al ser un juego single-player, no es necesario un servidor externo, toda la lógica del juego y los datos se gestionan localmente, lo que simplifica la arquitectura y reduce la complejidad.

- Aunque es una arquitectura monolítica, el uso de MVC permite escalar el juego añadiendo nuevas funcionalidades sin afectar el resto del sistema, por ejemplo, se pueden añadir nuevas mecánicas de juego modificando solo el Modelo, sin tocar la Vista o el Controlador.

- Godot Engine es compatible con MVC y proporciona herramientas para implementar esta arquitectura de manera eficiente, además, SQLite es una base de datos ligera y fácil de integrar, ideal para un proyecto de videojuegos.

Base de Datos Relacional (SQL):



Se eligió una base de datos relacional (SQLite) debido a la naturaleza estructurada de los datos del juego, las relaciones entre las entidades (jugador, tarea, recurso) son claras y bien definidas, lo que hace que un modelo relacional sea la opción más adecuada.

La elección de SQLite como base de datos relacional (SQL) se fundamenta en su capacidad para manejar datos estructurados y relaciones complejas de manera eficiente, características críticas para un juego de simulación estratégica como Scrum's Castle, al ser un proyecto single-player, SQLite ofrece ventajas clave: ligereza (no requiere servidor externo), portabilidad (almacena datos en un único archivo fácil de distribuir) y transacciones ACID (garantiza integridad al guardar el progreso del jugador), su esquema relacional permite modelar entidades como tareas, recursos, personajes y sprints con relaciones claras (ej: un sprint contiene múltiples tareas que consumen recursos), facilitando consultas complejas ("¿Cuánta madera se necesita para las tareas pendientes?") y evitando redundancias mediante normalización, además, SQLite se integra sin

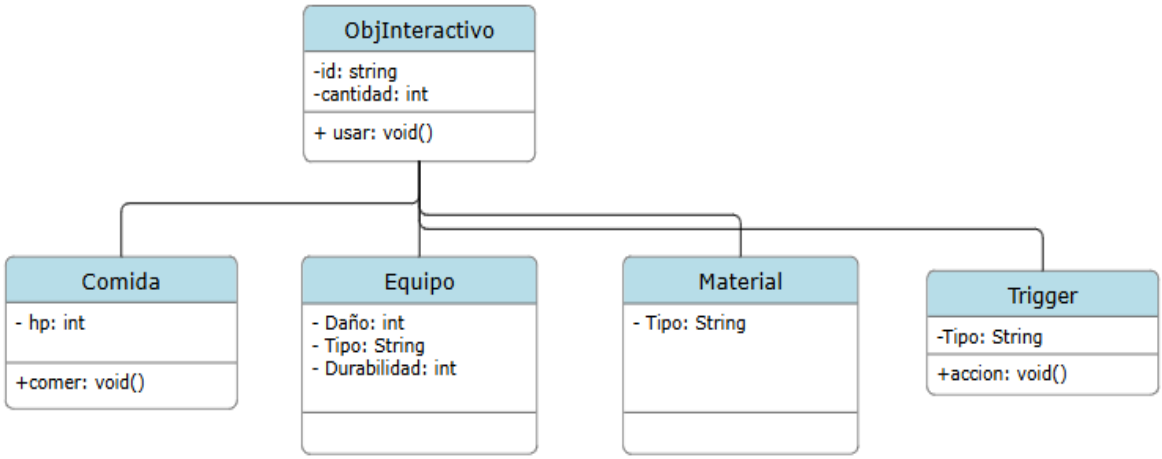
fricciones con Godot Engine, simplificando el desarrollo y permitiendo futuras expansiones (ej: añadir misiones secundarias) sin comprometer la estructura existente. Estas características lo hacen ideal para un proyecto que prioriza la gestión ordenada de datos, la escalabilidad modular y una experiencia de usuario fluida.

6. Patrones de diseño

Strategy

- Resuelve el problema de los objetos, ya que son muchos y cuentan con diferentes tipos de respuestas al interactuar con ellos, pero son muy parecidos en lo más básico, ya que son objetos con un sprite que permiten al usuario interactuar con ellos.

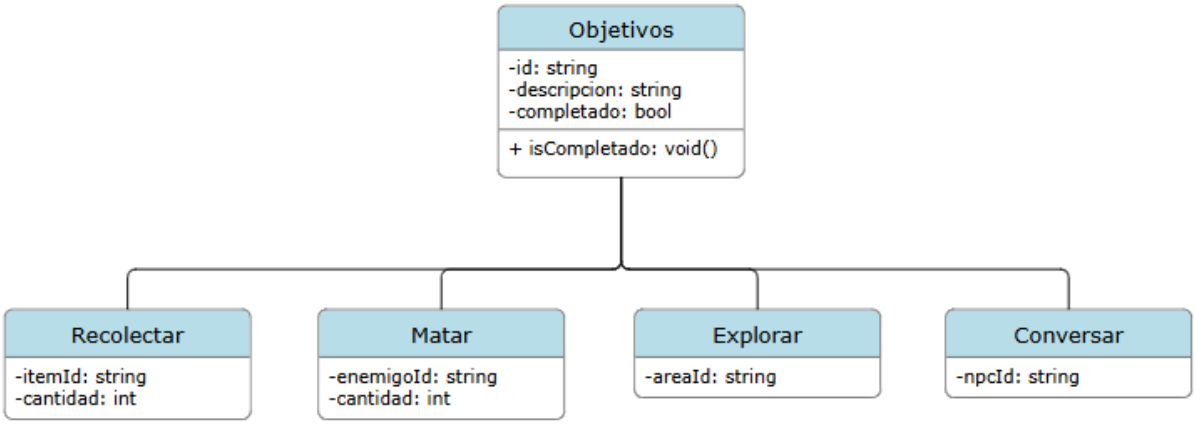
- Es necesario en el proyecto para facilitar la programación de los objetos interactivos, ya que su naturaleza es la misma.



Factory:

- Similar al strategy, resuelve el problema con las misiones y los objetivos, permitiendo hacer solo una clase grande que los contenga a todos a pesar.

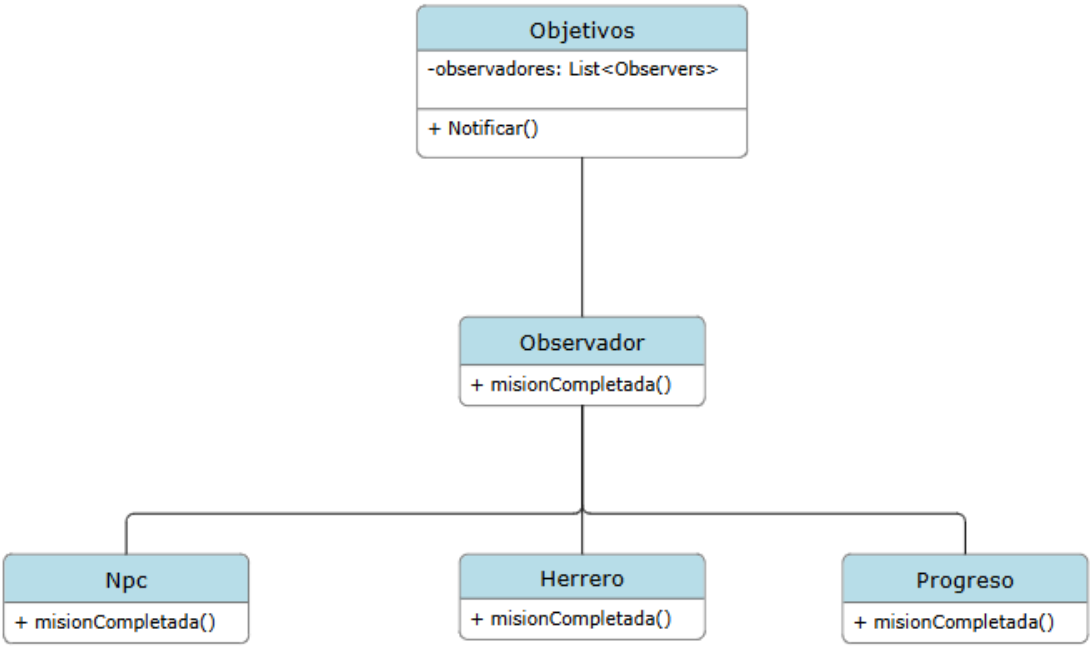
- Es necesario para reducir la cantidad de código a utilizar. En este caso poniendo una plantilla para todos los tipos de misiones que se van a hacer, cambiando para cada caso el tipo de misión.



Observer:

- El problema son los eventos que afectan a todo el juego, como las misiones principales, donde hay que notificar a varias clases que algo ha cambiado.

- Es necesario para reducir la dependencia entre clases.


Objetivos
-observadores: List<Observers>
+ Notificar()
Observador
+ misionCompletada()
Npc
+ misionCompletada()
Herrero
+ misionCompletada()
Progreso
+ misionCompletada()