PROYECTO FINAL: SCRUM'S Castle

Universidad Nacional de Colombia

Facultad de Ingeniería

Departamento de Industrial y Sistemas

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

Profesor:

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

Viernes 31 de Enero

2025

2. 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 Castle nació 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.

1. **Movimiento del Jugador**
    - Permitir al jugador controlar el movimiento de su personaje o equipo dentro del mundo del juego
    - **Datos a registrar:**
        - Dirección del movimiento (arriba, abajo, izquierda, derecha).
        - Velocidad de movimiento.
        - Animaciones de movimiento (caminar, correr, etc.).
    - **Procesos asociados:**
        - Implementar controles intuitivos para el movimiento (teclado, ratón, mando, pantalla táctil).
        - Asegurar que el movimiento sea fluido y responsivo.
2. **Interacción con el Entorno**
    - Permitir al jugador interactuar con objetos, personajes y elementos del entorno.
    - **Datos a registrar:**
        - Objetos interactuables (recursos, herramientas, edificios).
        - Acciones disponibles (recolectar, construir, hablar, etc.).

- **Procesos asociados:**
  - 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).

3. **Interacción con el Equipo**
   - Permitir al jugador interactuar con los miembros de su equipo para asignar tareas y gestionar recursos.
   - **Datos a registrar:**
     - Miembros del equipo (herrero, granjero, alquimista, arquero).
     - Tareas asignables a cada miembro del equipo.
     - Estado de cada miembro del equipo (ocupado, disponible, cansado)
   - **Procesos asociados:**
     - 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).
4. **Control de Tareas**
   - Supervisar la gestión de tareas en tiempo real**.**
   - **Datos a registrar:**
     - 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).
   - **Procesos asociados:**
     - 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.).
5. **Progreso**
   - Supervisar el avance del equipo y del reino
   - **Datos a registrar:**
     - Tareas completadas en el sprint.

        - Tareas no completadas y razones.
        - Recursos utilizados y disponibles.
        - Tiempo total del sprint.
        - Objetivos del reino alcanzados y pendientes.
    - **Procesos asociados:**
        - 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.
6. **Control de Recursos**
    - Supervisar la disponibilidad de recursos en tiempo real.
    - **Datos a registrar:**
        - Nombre del recurso (madera, hierro, comida, oro).
        - Cantidad disponible en stock.
        - Descripción breve del recurso.
        - Icono o imagen del recurso (opcional).
    - **Procesos asociados:**
        - 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.
7. **Retrospectivas**
    - Permitir al jugador revisar y mejorar su gestión después de cada sprint.
        - Datos a registrar:
        - Tareas completadas en el sprint.
        - Tareas no completadas y razones.
        - Recursos utilizados y disponibles.
        - Tiempo total del sprint.
    - **Procesos asociados:**
        - 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.

8. **Personalización del Equipo**
    - Permitir al jugador gestionar y mejorar su equipo de personajes.
    - **Datos a registrar:**
        - Nombre del personaje (herrero, granjero, alquimista, arquero).
        - Habilidades del personaje.
        - Nivel de experiencia.
        - Equipamiento actual (herramientas, armas, armaduras).
    - **Procesos asociados:**
        - 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.
9. **Sistema de Recompensas**
    - Motivar al jugador con recompensas por completar tareas y objetivos
    - **Datos a registrar:**
        - Tipo de recompensa (monedas, recursos, mejoras).
        - Cantidad de recompensa.
        - Condiciones para obtener la recompensa (completar tareas, alcanzar objetivos).
    - **Procesos asociados:**
        - 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.
10. **Castigos Medievales**
    - Aplicar consecuencias por el incumplimiento de tareas y objetivos.
    - **Datos a registrar:**
        - 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.

- **Procesos asociados:**
  - 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.

11. Mejoras y Actualizaciones

- Aplicar Permitir al jugador mejorar las habilidades y equipamiento del equipo
- **Datos a registrar:**
  - Tipo de mejora (herramientas, armas, habilidades).
  - Costo de la mejora (recursos, monedas).
  - Beneficios de la mejora (aumento de eficiencia, reducción de tiempo en tareas).
- **Procesos asociados:**
  - 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.

12. **Fabricación de Herramientas**

- Permitir al jugador fabricar herramientas para mejorar la eficiencia del equipo.
- **Datos a registrar:**
  - 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).
- **Procesos asociados:**
  - 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.

13. Gestión de Inventario

- Supervisar el almacenamiento y uso de materiales y herramientas
- **Datos a registrar:**
  - Capacidad de almacenamiento del inventario.
  - Cantidad de cada material y herramienta en el inventario.

        - Ubicación del inventario (almacén central, edificios específicos).
    - **Procesos asociados:**
        - 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.).

---

3. Análisis de requerimientos:

1. **Must have (imprescindibles):**
    - Movimiento del Jugador
    - Interacción con el Entorno
    - Control de Tareas
    - Castigos Medievales
    - Interacción con el Equipo
2. **Should have (importantes):**
    - Progreso
    - Control de Recursos
    - Retrospectivas
    - Sistema de Recompensas
3. **Could have (deseables):**
    - Personalización del Equipo
    - Mejoras y Actualizaciones
    - Fabricación de Herramientas
    - Gestión de Inventario
4. **Won’t have (no ahora):**
    - Eventos Aleatorios Complejos
    - Misiones Especiales

Estimación de esfuerzo en desarrollo con secuencia de Fibonacci:

	Jerónimo	Sofia	Sharick	Nicolas	Total
Movimiento del Jugador	1	2	2	1	2
Interacción con el Entorno	2	3	2	3	2
Control de tareas	3	5	3	3	3
Castigos medievales	5	5	3	5	5
Interacción con el equipo	3	3	3	5	5
Progreso	3	3	3	3	3
Control de recursos	5	8	5	5	5
Retrospectivas	3	2	3	3	2
Sistema de recompensas	1	3	2	2	2
					29

---

4. Análisis gestión de software

1. Triada de Gestión de Proyectos

1.1 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 (10 días)

- **Días 1-4:** Definición de mecánicas principales, flujo del juego y diseño de interfaz (UI/UX).

- **Semana 5-10:** Creación de prototipos y wireframes, documentación detallada de requerimientos.

Fase 2: Desarrollo (30 días)

- **Días 10-18:** Implementación del sistema de movimiento del jugador, interacción con el entorno y gestión de tareas.

- **Días 18-29:** Desarrollo del control de recursos, interacción con el equipo y sistema de recompensas.

- **Días 30-40:** Integración de retrospectivas, castigos medievales y progreso.

Fase 3: Pruebas y Ajustes (8 días)

- **Días 41-43:** Pruebas unitarias y de integración en cada módulo.

- **Días 42-47:** Pruebas de experiencia de usuario (UX) y balanceo de mecánicas.

- **Días 41-48:** Ajustes finales y optimización del rendimiento.

Duración total estimada: 48 días.

1.2 Costo

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:

Concepto	Cantidad	Costo Mensual (COP)	Duración	Costo Total (COP)
Desarrollo				
Desarrollador Junior	2	$2.500.000	1 mes	$5.000.000
UX/UI Designer	1	$2.000.000	1 mes	$2.000.000
Tester	1	$1.800.000	1 mes	$1.800.000

Total Aproximado				$8.800.000

Nota: Estamos asumiendo que utilizamos motores y servidores gratuitos como **Godot o Unity (versión gratuita)**.

1.3 Alcance

El alcance del proyecto se define en función de las características imprescindibles para la primera versión del videojuego (**MVP - Minimum Viable Product**). Hemos clasificado las funcionalidades en base a su prioridad:

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.

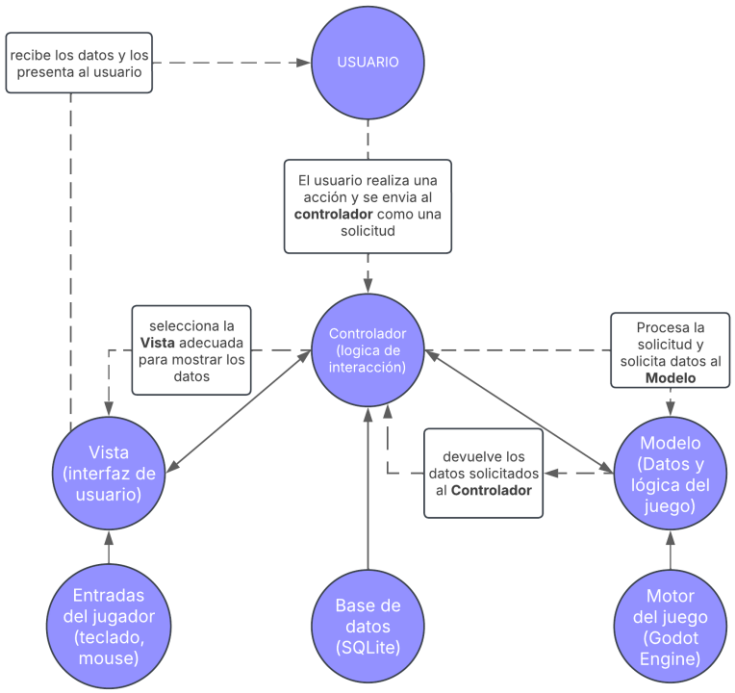
Excluidas en esta fase:

- Eventos aleatorios complejos.
- Misiones especiales.

8. 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).
    - Incluye 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.

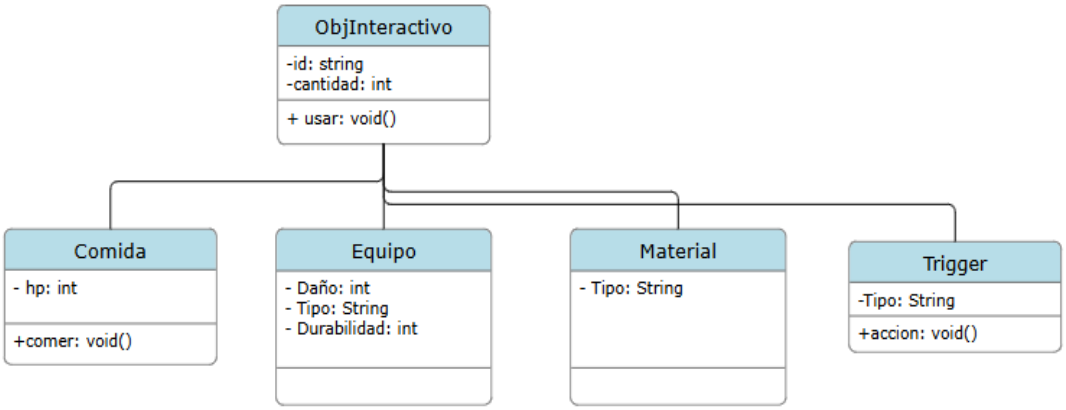
---

9. 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

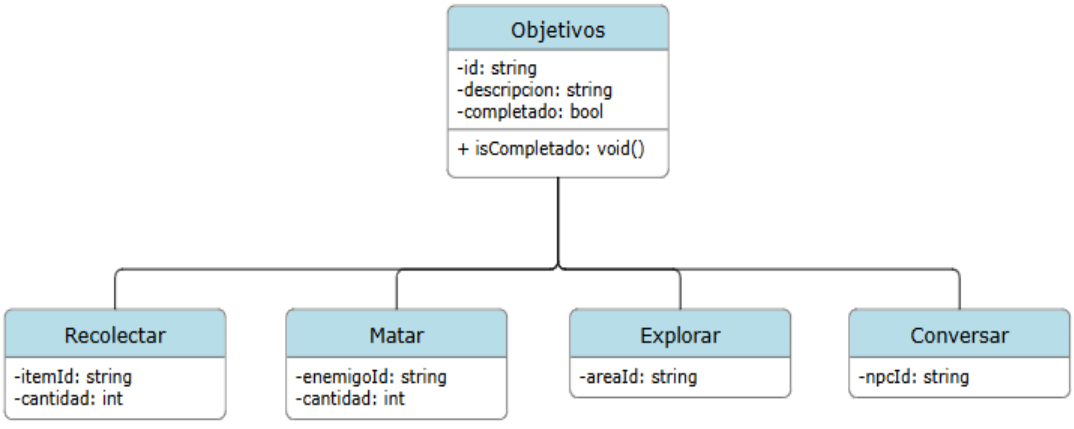
- Es necesario en el proyecto para facilitar la programación de los objetos interactivos, ya que su naturaleza es la misma.



➢ **Factory:**

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

- 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.