UNIVERSIDAD NACIONAL DE SAN JUAN

FACULTAD DE CIENCIAS EXACTAS FÍSICAS Y NATURALES

DEPARTAMENTO DE INFORMÁTICA

LICENCIATURA EN CIENCIAS DE LA COMPUTACIÓN

Evaluación de la viabilidad técnica de un framework basado en el patrón Entidad-Componente-Estado.

Caso de estudio: videojuego multijugador

AUTOR
Tec. Martín Facundo Gómez Peralta

CO - ASESOR
Mg. Cintia Ferrarini Oliver

ASESOR
Mg. Emilio Ormeño

SAN JUAN

2026

Agradecimientos

A mi madre

A mi pareja

A mis directores

A mis compañeros

Resumen

En este trabajo final se propuso evaluar un framework backend basado en patrones de componentes y estados para el desarrollo de videojuegos, facilitando la separación entre servidor y frontend. La evaluación se apoya en una investigación aplicada con enfoque experimental y exploratorio, utilizando como estrategia principal el prototipado funcional de un videojuego multijugador. Esta propuesta nace de la necesidad de contar con una solución unificada para la arquitectura completa de un videojuego, que le permita a los desarrolladores competir en un mercado que requiere productos de software en ciclos cortos, evitando caer en la generación de código desestructurado y errores en las versiones iniciales. Se realizó una revisión de trabajos previos e integración de modelos como Entidad-Componente-Sistema (ECS), patrón Entidad-Componente y patrón de diseño estado, identificando la falta de soluciones unificadas para la arquitectura completa de un videojuego. Se optó por una metodología cuantitativa para la evaluación de los aspectos técnicos del framework, en cuanto a seguridad, escalabilidad y mantenimiento. Se considera que la propuesta contribuirá significativamente al avance tecnológico y metodológico en el sector global de la industria de videojuegos.

Índice General

Índice de Imágenes, tablas y figuras

Introducción

Este trabajo final de Licenciatura[1] se llevó a cabo en el marco del Proyecto de Investigación CICITCA denominado "Framework USIM para el desarrollo de Aplicaciones Web", aprobado según resolución 2902-CS-UNSJ, en el Gabinete de Gamificación del Instituto de Informática de la Facultad de Ciencias Exactas, Físicas y Naturales de la Universidad Nacional de San Juan.

Se trata de una investigación aplicada, que aporta a la industria de los videojuegos. Industria que en las últimas dos décadas ha tenido un gran crecimiento y se ha convertido en un negocio muy rentable. Por lo tanto, el desarrollo de software de videojuegos se ha vuelto una de las más importantes y prominentes profesiones en el mercado (Barakat, 2019).

Una de las dificultades en el desarrollo de videojuegos se da en la creciente presión de los mercados en continua expansión y la intensa competencia entre compañías. Estas dinámicas han reducido significativamente los tiempos necesarios para lanzar un producto al mercado, generando ciclos de desarrollo más rápidos, como consecuencia, las versiones iniciales de un videojuego presentan muchos errores (Barakat, 2019).

En el desarrollo de videojuegos, en general, se pone más énfasis en la creatividad que en la consistencia técnica, dado que los requerimientos funcionales cambian con mucha frecuencia (Borowa, Zalewski & Saczko, 2021). Debido a esto, muchos productos carecen de un diseño arquitectónico que pueda unificar la lógica de todo el videojuego, lo que conlleva a que los desarrolladores generen un código sin estructura, difícil de mantener y actualizar; y propenso a errores.

En esta propuesta, se considera que las dificultades mencionadas pueden ser mitigadas desde el comienzo del proyecto de desarrollo de videojuegos si se diseña un framework que incorpore patrones de estados y componentes. Por tal motivo, se busca evaluar un framework generado en el marco del proyecto mencionado mediante el diseño de un prototipo de videojuego multijugador.

Este trabajo se ha organizado en cinco capítulos, de la siguiente manera: en el capítulo Nº 1 se presentan los antecedentes, el problema, la justificación y los objetivos; en el capítulo N°2 se expone el marco teórico; en el capítulo N°3 se indica la metodología aplicada tanto para la investigación como para el desarrollo de la propuesta; en el capítulo N°4 se presenta la propuesta de evaluación del framework; y en el capítulo N°5 se exponen las conclusiones y los trabajos futuros.

[1] Requisito para obtener el título de Licenciado en Ciencias de la Computación, FCEFyN, UNSJ.

Capítulo Nº 1: Antecedentes, Problema, Justificación y Objetivos

1. 1 Antecedentes

A continuación se presentan algunos trabajos previos relacionados con la propuesta.

En (Blanch & Beaudouin-Lafon, 2006) se propone un kit de herramientas (toolkit) para desarrollar interfaces de usuario altamente interactivas utilizando el patrón de diseño de estado junto con jerarquías de máquinas de estados, obteniendo herencia de comportamientos (behaviours) que resultan ser fáciles de reutilizar y modificar. Si bien este kit de herramientas permite realizar interfaces interactivas fácilmente, tiene la desventaja de que dificulta el reporte de errores o no permite hacerlos explícitos.

En (Raffaillac & Huot, 2019) se propone un framework de diseño de interfaces interactivas basados en el modelo Entidad-Componente-Sistema (ECS) facilitando también el manejo de comportamientos y promoviendo su reutilización. A través del patrón de composición permite generar nuevos elementos y comportamientos haciendo uso de componentes. Este trabajo muestra la potencia de ECS en cuanto a herencia de comportamiento, sin embargo no menciona el uso del framework en el área del desarrollo de videojuegos.

En (Sintaro, Latumakulita, Bernard, Surahman & Islam, 2023) se aplica el patrón de diseño estado para implementar el encapsulamiento del comportamiento de un personaje de un videojuego y utiliza una máquina de estados finita para facilitar la transición de estados causada por las entradas de usuario o el cambio de valores en sus variables. Sin embargo en lo que respecta al diseño de videojuegos, se centra en solucionar solamente la interactividad del personaje pero no del videojuego en su conjunto.

En (Mizutani & Kon, 2020) se propone una arquitectura de referencia para economizar la implementación de mecánicas de videojuegos utilizando el modelo ECS y el patrón de diseño estado pero limita a este último al manejo de las “escenas” tal como se conciben en Unity.

Según InvestGame, una empresa que se dedica a la inteligencia de negocios en la industria de videojuegos, los motores de videojuegos Unreal Engine y Unity dominaron las ventas del 2025 en la plataforma Steam tanto para videojuegos indie como videojuegos de gran complejidad (AAA).

Estos motores implementan la arquitectura GameObject-Component, o dicho de otra manera Entidad-Componente. Esta arquitectura propone que un videojuego se define como Entidades que interaccionan entre sí, donde estas interacciones están definidas por los Componentes que estas poseen.

Otro de los motores de videojuegos mencionados es GameMaker, el cual implementa una arquitectura similar a la de Entidad-Componente, pero las interacciones entre Entidades quedan definidas por los Eventos que pueden manejar y no por sus Componentes.

Unity también permite implementar máquinas de estado finito FSM para modelar el comportamiento de los GameObjects, en particular de los personajes (Pavan,2024), pero se limita solo a estos y no permite el modelado de todo el videojuego a través del uso de estados.

El trabajo de (Oliveira, 2024) propone un motor de videojuego programado en Typescript donde se implementan GameObjects que se encargan de gestionar toda la lógica que puede incluirse con Componentes pero sin utilizar esta abstracción. A su vez, existe una implementación de niveles o etapas llamada Stage, en la cual se almacenan todos los GameObject. Esto último podría pensarse como una sencilla implementación del patrón de Estado, pero el autor describe las Stage de manera similar a como funcionan las Scenes de Unity, es decir, contenedores que permiten agrupar GameObjects según algún criterio, en este caso división de niveles.

Una implementación del patrón Entidad-Componentes se encuentra en el motor de videojuegos Gear2D, en donde además se implementa con el patrón Composite y Adaptive Object-Model la posibilidad de agregar Componentes a los GameObjects en tiempo de ejecución.(Freitas, 2012)

(Macedo,2015) realiza una implementación de un simulador de carreras de autos utilizando una máquina de estados finitos FSM para dividir los distintos comportamientos de los vehículos en estados.

Otro enfoque respecto al uso de estados para manejar el ciclo de vida de una partida se realiza en el trabajo de (Bezditnyi, 2024), aunque el uso del Patrón estado solo se aplica a los tres estados básicos de todo partida, es decir, a la inicialización, gameloop y finalización. Si bien esta es una clara implementación del patrón estado para gestionar el ciclo de vida, este no se aplica a cada uno de los estados intermedios que se presentan en el gameloop, por lo que no resuelve lo que se intenta realizar con GameCore.

Después de la revisión de la literatura, no se encontraron artículos en los que se proponga usar el Patrón de diseño Estado junto con el patrón de Entidad-Componente para definir una arquitectura que permita el diseño y desarrollo completo de un videojuego. Esta propuesta pretende aportar en este sentido.

1.2 Formulación del Problema y Justificación

En la industria de los videojuegos, una de las dificultades que se produce en el desarrollo de videojuegos es la necesidad de generar productos con ciclos de vida de desarrollo más rápidos, los tiempos para ubicar un producto en el mercado se han reducido, debido a que el mercado de los videojuegos está en continua expansión, y a la competencia entre compañías. Especialmente en las pequeñas empresas, que al tener que responder con entregas rápidas descuidan aspectos como la calidad del software, ocasionando muchos errores (bugs) en las primeras versiones de los videojuegos (Racheva, Barakat, 2019).

En los proyectos de desarrollo de videojuegos es más importante el proceso de control de calidad y diseño (playtesting) que el proceso de verificación y validación (testing) del software propiamente dicho, por lo que para detectar errores o cambiar alguna funcionalidad, siempre se opta por escuchar a quienes realizan el playtesting sin contemplar buenas prácticas de desarrollo de software (Marklund, Engström, Hellkvist, Backlund, 2019). Además los proyectos pueden tener cambios en todas las etapas del proceso ya que las fuentes de ideas pueden estar en cualquiera de

los involucrados en el desarrollo del proyecto. Por lo tanto, los requerimientos de desarrollo son altamente subjetivos, impredecibles y flexibles, por lo que el proceso de desarrollo de un videojuego no se puede comparar con un proceso de desarrollo de software tradicional.

Otra dificultad es que en la actualidad, los motores de videojuegos centran el desarrollo de productos en aspectos visuales y auditivos (creativos), imponiendo así en los desarrolladores un flujo de trabajo por defecto que va en perjuicio de la lógica del producto, en cuanto a su consistencia técnica (Borowa et al., 2021).

Otro aspecto a considerar surge cuando los desarrolladores necesitan incorporar características específicas de la lógica del videojuego, como la gestión de niveles o el registro de partidas. Esta tarea se vuelve compleja, ya que requiere múltiples modificaciones en el código.

La mayoría de los motores de videojuegos están diseñados como soluciones completas que residen en el frontend, limitando la posibilidad de brindar soporte a la implementación de aplicaciones que aprovechen la separación de responsabilidades entre backend y frontend. Otros obstáculos o limitaciones que se presentan tienen que ver con la falta de frameworks orientados a videojuegos que soporten una arquitectura distribuida entre backend y frontend, con la ausencia de implementación de patrones como Estados y Componentes en frameworks backend específicos para motores de videojuegos y con los desafíos en la persistencia de estados complejos de GameObjects y su sincronización en tiempo real con el frontend.

La propuesta pretende evaluar como el desarrollo de un Framework de Backend para el desarrollo de Videojuegos basado en Patrones de Estados y Componentes facilitando el desarrollo de videojuegos robustos y fácilmente escalable, beneficiando a la industria de videojuegos. Se consideró que la exploración de los patrones mencionados podrían mejorar la gestión de partidas y objetos de juego, reduciendo la complejidad del frontend. Así como también el desarrollo de este trabajo ha contribuido a la formación científico-tecnológica de su autor.

Por lo tanto, en este trabajo se plantea el siguiente interrogante: ¿Qué aspectos debe considerar una evaluación de viabilidad técnica de un framework basado en el patrón Entidad-Componente-Estado, para el desarrollo de videojuegos multijugador?

1.3 Objetivos

1.3.1 Objetivo General

- Evaluar la viabilidad técnica de un framework basado en el patrón Entidad-Componente-Estado, mediante la implementación de un prototipo de videojuego multijugador.

1.3.1.1 Objetivos Específicos

- Analizar patrones de diseño de Estados y Componentes aplicados al desarrollo de videojuegos.
- Identificar los requerimientos funcionales y no funcionales, así como las especificaciones técnicas del videojuego multijugador.

- Diseñar un prototipo de videojuego multijugador basado en el Framework.
- Validar el prototipo desarrollado.

Capítulo Nº 2: Marco Teórico

2.1. Enfoque epistemológico

La presente propuesta se inscribe en el paradigma tecnocrático (Eden, 2007) y adopta un enfoque de investigación aplicada (OCDE–Manual de Frascati, 2015), en tanto se orienta al desarrollo de herramientas tecnológicas y soluciones concretas para la industria del desarrollo de videojuegos. Este tipo de investigación prioriza la aplicación práctica del conocimiento teórico existente con el objetivo de resolver problemas específicos, profundizando y operacionalizando conceptos propios de la ingeniería de software y el diseño de sistemas interactivos.

En este marco, el desarrollo de videojuegos se concibe como un dominio de aplicación compleja y multidisciplinaria. Tal como señala Marklund (2019), esta actividad integra saberes provenientes de la programación, el diseño gráfico, la narrativa interactiva, la experiencia de usuario, la ingeniería de software y hardware, así como de la gestión, los negocios y el marketing, entre otros. Esta convergencia de áreas confiere a cada videojuego características singulares y exige enfoques arquitectónicos flexibles.

2.2. Frameworks y motores de videojuegos

Desde una perspectiva conceptual, un framework puede entenderse como un puente programático entre conceptos abstractos y sus implementaciones concretas de bajo nivel. Taylor (2018, citado en Politowski et al., 2021) define a los frameworks como estructuras que permiten mapear estilos arquitectónicos en implementaciones reutilizables, proporcionando una base para el diseño de arquitecturas de software.

En el ámbito del desarrollo de videojuegos, los motores de videojuegos (game engines) constituyen frameworks especializados. Según Lewis y Jacobson (2002, citados en Politowski et al., 2021), un motor de videojuegos es una colección de módulos de código de simulación que no define directamente la lógica del juego ni los datos de nivel, sino que provee los servicios fundamentales para su construcción. Motores como Unity, Unreal Engine y Godot integran funcionalidades esenciales gráficos, física, audio y scripting y ofrecen una plataforma sólida para el desarrollo de proyectos interactivos.

2.3. Paradigmas de diseño en el desarrollo de software

El desarrollo de software contemporáneo, particularmente en motores de videojuegos y sistemas de simulación en tiempo real, se sustenta en patrones arquitectónicos que priorizan la flexibilidad y la modularidad por sobre estructuras jerárquicas rígidas siempre que sea posible (Nystrom, 2014). En este contexto, resulta central analizar la dicotomía entre herencia y composición como mecanismos de reutilización y extensión del comportamiento.

2.3.1. Herencia versus composición

En la Programación Orientada a Objetos (POO) tradicional, la reutilización de código se logra comúnmente mediante la herencia de clases. Sin embargo, en sistemas complejos como los videojuegos, las jerarquías de clases profundas tienden a generar rigidez, alto acoplamiento y problemas estructurales como el denominado problema del diamante.

Como respuesta a esto, existe el principio de "Composición sobre Herencia" (Gamma et al., 1994), el cual establece que el comportamiento de un objeto debe construirse a partir de la agregación de componentes (relación "tiene un"), favoreciendo la extensibilidad y el desacoplamiento.

2.4. Arquitectura Entidad-Componente (EC)

El principio de composición sobre herencia se implementa en el patrón arquitectónico Entidad-Componente (Entity-Component, EC), ampliamente adoptado en el desarrollo de videojuegos. Este modelo propone la concepción de los objetos no como instancias de clases monolíticas sino como agregaciones dinámicas de comportamientos.

Según Nystrom (2014), la arquitectura EC, (nombrada como patrón de componentes) desacopla la identidad de un objeto de su funcionalidad mediante dos elementos fundamentales:

- Entidad (Entity): actúa como un contenedor abstracto, sin lógica ni datos específicos del dominio.
- Componente (Component): actúa como un módulo que contiene datos y lógica tales como renderizado, físicas o audio.

Teniendo en cuenta esto, un objeto del juego (GameObject) es definido entonces por un conjunto de componentes que tiene en un tiempo dado, lo que permite una alta reutilización de código y permite la mutabilidad en tiempo de ejecución.

2.5. Implementación del patrón EC en Unity

El motor Unity, en su versión clásica, implementa una variación del patrón Entidad-Componente que constituye el núcleo de su arquitectura de scripting y gestión de escenas.

2.5.1. GameObject

En Unity, la entidad se materializa en la clase GameObject, la cual funciona como un contenedor que posee una transformación espacial obligatoria (Transform) y una lista de componentes. De esta manera, un personaje o un enemigo se define a través de sus componentes específicos asociados, como por ejemplo, su componente de físicas, de movimiento, de renderización, entre otros.

2.5.2. MonoBehaviour

Los componentes en Unity heredan de la clase MonoBehaviour. A diferencia de un ECS puro, estos componentes encapsulan tanto:

- Datos: variables serializables visibles en el Inspector.
- Lógica: métodos de ciclo de vida (Awake, Start, Update) invocados por el motor mediante callbacks.

Este enfoque integra comportamiento y estado dentro del mismo componente, característica distintiva del modelo EC clásico.

Unity permite la manipulación dinámica de entidades a través de la habilitación o deshabilitación de componentes, de esta manera se puede modificar la lógica del juego sin destruir los objetos, optimizando el rendimiento y la gestión de memoria.

2.6. Diferenciación entre EC y ECS (Data-Oriented Design)

Desde una perspectiva taxonómica, es fundamental distinguir entre el modelo Entidad-Componente (EC) y el Entidad-Componente-Sistema (ECS) orientado a datos.

EC (modelo clásico de Unity): orientado a objetos; los componentes contienen datos y lógica, con posibles penalizaciones de rendimiento por dispersión en memoria.

ECS (Unity DOTS): orientado a datos; las entidades son identificadores, los componentes contienen sólo datos y los sistemas encapsulan la lógica de procesamiento.

Aunque Unity avanza hacia ECS para maximizar el rendimiento, la arquitectura basada en GameObjects y MonoBehaviours corresponde formalmente al patrón EC (Fabian, 2018; Sepela, 2024). En Unreal Engine, una arquitectura conceptualmente similar se implementa mediante Actors y Blueprints.

2.7. Backend y videojuegos multijugador

En los videojuegos multijugador, el backend constituye una capa fundamental encargada de gestionar la lógica y los datos de la aplicación (Pérez Ibarra et al., 2021). Estos sistemas deben garantizar coherencia, baja latencia y seguridad en la transmisión de datos (Kim & Zhao, 2020, citado en Ormeño & Ferrarini Oliver, 2025).

Una problemática central es la sincronización del estado de los objetos del juego. En arquitecturas cliente-autoritativas, el control distribuido del estado facilita la aparición de trampas. Como respuesta, se propone el uso de arquitecturas servidor-autoritativas, donde el estado global de la partida es mantenido por el servidor y los clientes se limitan a renderizar la información recibida. En este contexto, GameCore se presenta como un framework orientado a integrar estos requerimientos sin comprometer escalabilidad ni extensibilidad.

2.8. Patrones de diseño aplicados a videojuegos

Los patrones de diseño constituyen soluciones reutilizables a problemas recurrentes en la arquitectura de software. Gamma et al. (1994) destacan el patrón Estado, fundamental para gestionar cambios dinámicos de comportamiento. Por su parte, Nystrom (2014) y Härkönen (2019) describen los patrones Componente y ECS como enfoques centrales para el desarrollo flexible y desacoplado de videojuegos modernos.

2.9. Marco legal

En Argentina, la Ley 26.043 (2005) regula aspectos vinculados a la clasificación etaria y advertencias sanitarias de los videojuegos, sin abordar su desarrollo como industria. Gala (2019) señala la ausencia de una normativa específica que los tipifique como obras protegidas dentro del régimen de propiedad intelectual, lo que genera una dualidad entre su consideración como bienes culturales y productos de la industria del software, dificultando su protección jurídica y desarrollo estratégico.

Capítulo Nº 3: Metodología

En este capítulo se explicitan métodos, estrategias metodológicas, procedimientos realizados en este trabajo final.

3.1. Metodología de la Investigación

Este trabajo responde a una investigación aplicada con enfoque experimental y exploratorio, utilizando como estrategia principal el prototipado funcional de un videojuego multijugador. La lógica de la investigación fue cuantitativa, se analizaron métricas de rendimiento, escalabilidad y tiempos de desarrollo.

De acuerdo a la finalidad de la investigación, se aplicó un diseño exploratorio-descriptivo, dado que el autor poseía escasos conocimientos sobre el tema y se quería determinar las propiedades o características del fenómeno bajo estudio, para luego describirlas. En cuanto a su naturaleza temporal, se aplicó una investigación transeccional dado que se realizó una sola evaluación de la propuesta (Yuni & Urbano, 2014).

3.1.1. Instrumentos de recolección, procesamiento y análisis de la información.

Se aplicaron los siguientes instrumentos:

- Logs y Registros de Servidor: lo que permitió la captura de comportamiento del usuario, clics y errores en aplicaciones. Los usuarios en este caso fueron los asesores.
- Software de base: Compiladores, editores y depuradores que manipulan los datos a bajo nivel.

3.2. Metodología del Desarrollo del prototipo.

En el ámbito del desarrollo de videojuegos no hay una metodología estándar de desarrollo ni se mantienen los requisitos funcionales y no funcionales de manera constante durante el desarrollo (Murkland, 2019). Es por esto que se decidió investigar metodologías acordes al desarrollo que se propuso y adoptar finalmente una que permita gestionar el trabajo en proyectos individuales.

Agile Puzzle es una metodología basada en Scrum que permite adaptarse rápidamente a los cambios de requisitos y gestionar las tareas en el orden que se desee, debido a que estas no deben estar atadas estricamentente a un momento específico del desarrollo (Loor-Cobeña & Alcívar-Cevallos, 2025). A continuación se describe.

3.2.1. Fases del Desarrollo del Sistema/Producto.

Esta metodología contempla cuatro fases: Planificación, Desarrollo, Pruebas y Lanzamiento. A continuación se detallan las tareas que se realizan en cada etapa.

Componentes de la Metodología Agile Puzzle	
Fases	**Tareas**
Planificación	Creación del GameLog
	Creación del Backlog
	Sprint Planning
Desarrollo	Daily Scrum
	Desarrollo Continuo
	Integración Continua
	Sprint Retrospective
	Refinamiento del Backlog
Pruebas	Pruebas Internas
	Beta Testing
	Corrección de Errores
Lanzamiento	Preparación para el Lanzamiento
	Marketing y Publicidad
	Lanzamiento del Juego

Fuente: Loor-Cobeña & Alcívar-Cevallos (2025)

Capítulo Nº 4: Propuesta de evaluación del framework mediante el prototipado de videojuego

En este capítulo se presenta en primer lugar el framework basado en Backend y luego el prototipado de videojuego multijugador generado a partir de este para la evaluación de su viabilidad técnica.

4.1 Framework GameCore

El Framework GameCore implementa una arquitectura backend robusta utilizando PHP/Laravel como núcleo tecnológico, con las siguientes características distintivas:

- Patrón de Componentes: mediante la implementación de elementos, estructuras o assets prefabricados y estandarizados (ejemplo: paredes, suelos o módulos funcionales) que se ensamblan entre sí para crear escenarios, personajes o sistemas complejos de forma eficiente (GameObjects modulares) con Componentes reutilizables
- Gestión de Estados: Componentes de estado (State Components) que permiten máquinas de estados complejos.
- Sistema de Eventos: mediante la comunicación desacoplada entre elementos del framework.
- Servicios Persistentes: con la incorporación de GameServices para modelar la lógica transversal de juego.
- Sistema de Renderizado: View System y Render System para gestionar la sincronización cliente-servidor
- Prefabs: Plantillas (Templates) para instanciación eficiente de estructuras complejas y reutilización de código.

El framework GameCore responde a los siguiente requisitos funcionales y no funcionales.

4.1.1 Requisitos funcionales y no funcionales de Game Core

Requisitos funcionales

- **RF-01**: El sistema debe proporcionar una API REST para que los clientes comuniquen eventos y señales a fin de actualizar el frontend.
- **RF-02**: El sistema debe gestionar GameObjects persistentes organizables en jerarquías padre-hijo.
- **RF-03**: El sistema debe implementar un patrón de Componentes, donde estos son asociados a los GameObject para definir su comportamiento.
- **RF-04**: El sistema debe soportar máquinas de estados mediante State Components que se activen/desactiven según el estado del GameObject.
- **RF-05**: El sistema debe permitir implementar Prefabs como plantillas reutilizables para instanciar estructuras complejas de GameObjects.
- **RF-06**: El sistema debe permitir la definición de múltiples GameApps (videojuegos) independientes dentro del mismo framework.

- **RF-07**: El sistema debe implementar un sistema de eventos con suscripción automática para comunicación entre componentes de GameObjects.
- **RF-08**: El sistema debe gestionar ciclos de vida de componentes mediante callbacks (Start, Update, View, Enter, OnExit).
- **RF-09**: El sistema debe manejar recursos multimedia (imágenes, sonidos) con generación automática de placeholders cuando no existan.
- **RF-10**: El sistema debe construir vistas combinando documentos JSON generados por los componentes habilitados.
- **RF-11**: El sistema debe renderizar y enviar vistas actualizadas al cliente cuando ocurran eventos relevantes.
- **RF-12**: El sistema debe persistir automáticamente GameObjects, Componentes y sus atributos en la base de datos.

Requisitos no funcionales

A. Requisitos del Producto

- **RNF-P01 (Eficiencia)**: El sistema debe mantener la consistencia de estado entre múltiples clientes conectados simultáneamente.
- **RNF-P02 (Fiabilidad)**: El sistema debe permitir recuperar el estado del juego durante fallos del servidor, automáticamente al reiniciarse.
- **RNF-P03 (Portabilidad)**: El backend debe ser independiente de la tecnología de frontend utilizada.
- **RNF-P04 (Mantenibilidad)**: Las modificaciones en un componente no deben requerir cambios en otros componentes no relacionados.

B. Requisitos Organizacionales

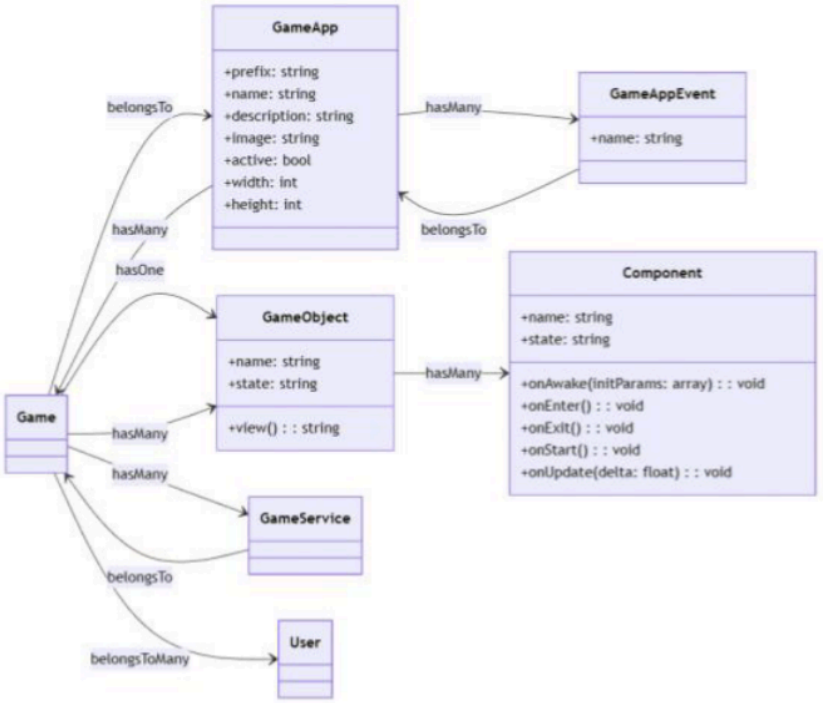
- **RNF-O01 (Implementación)**: El sistema debe implementarse utilizando PHP 8.0+ y el framework Laravel.
- **RNF-O02 (Entregable)**: La documentación técnica debe incluir ejemplos funcionales para todos los patrones soportados.
- **RNF-O03 (Estructura)**: Los juegos deben organizarse en directorios siguiendo la convención app/GameApps/[prefijo]/.

C. Requisitos Externos

- **RNF-E01 (Interoperabilidad)**: El sistema debe exponer interfaces REST compatibles con estándares HTTP/JSON.
- **RNF-E02 (Legislativos)**: El sistema debe permitir configurar edad mínima por juego para cumplir regulaciones de clasificación por edad.
- **RNF-E03 (Éticos)**: El sistema no debe exponer datos de usuarios no autenticados a través de la API.
- **RNF-E04 (Compatibilidad)**: El sistema debe soportar conexiones desde motores de juego comerciales (Unity, Unreal, Godot).

4.1.3 Diagrama arquitectónico de GameCore

GameCore se implementa utilizando el patrón Entidad-Componente y Estado. Cada GameObject contiene Componentes que se encargan de brindarles funcionalidades. Los estados son implementados como Componentes que se añaden al GameObject para brindarles la capacidad de gestionar el cambio de vista y comportamiento según los eventos propios de la interacción del usuario. A su vez existe un objeto GameObject que contiene toda la aplicación, llamado GameApp, la cual tiene componentes-estado que guían toda la ejecución del videojuego.



Fuente: Ormeño & Ferrarini Oliver (2025).

A continuación se presenta el videojuego multijugador cuyo prototipo se implementó con el framework GameCore. Cabe aclarar que se seleccionó este juego porque favorece la capacidad de argumentación y expresión oral de sus jugadores, se realizaron ajustes para su implementación en una aula de clases de nivel medio.

4.2 Descripción del videojuego basado en Werewolves game

4.2.1 Narrativa Werewolves game

El videojuego propone una variación del juego de mesa The Werewolves of Miller's Hollow (Los hombres lobo de CastroNegro). Este juego consiste en una serie de rondas donde aldeanos deben descubrir, a través de la argumentación, a hombres hombres lobo que los asesinan por las noches y que están presentes entre ellos. Los hombres lobo cada noche se encargan de eliminar a los aldeanos tratando de no ser descubiertos mientras que los aldeanos cada día intentan descubrir a los hombres lobo para así eliminarlos y ganar la partida.

A continuación se presenta la narrativa original (reducida) del juego de mesa, entre las consideraciones simplificadas se eliminaron a algunos personajes clave con el fin de generar un videojuego sencillo y que a su vez mantenga a los jugadores atentos. Luego se presenta la adaptación de esa narrativa para ser utilizada en las aulas en una clase o varias, dependiendo de la planificación del docente.
Posteriormente se muestran los diagramas y pantallas que describen el flujo del videojuego.

Narrativa original(reducida)

En esta sección se indican los roles y los estados por los que transitan los jugadores. Estos se encuentran delimitados por intervalos de tiempo fijos que son pequeños para favorecer a que los jugadores se concentren plenamente en la partida. También se muestra un cuadro que detalla la cantidad de jugadores y su relación con la cantidad de hombres lobo del juego. Por último, se muestra cada uno de los roles indicando sus acciones clave.

Estado: Reparto de roles (2'), Noche(3'), Día(2'), Votación(10'), Fin del juego(1')

En el siguiente cuadro se presenta la cantidad de participantes, estos son a razón de 1 lobo cada 3 o 4 participantes.

Cantidad de participantes:

Jugadores Totales	Nº de Hombres Lobo	Nº de Aldeanos
5 - 6	1	4 - 5
7 - 8	2	5 - 6
9 - 11	3	6 - 8

Roles: Moderador, Hombre Lobo, Aldeano

Moderador: Dirige la partida.

Hombre Lobo: Durante la noche puede eliminar a un aldeano.

Aldeano: Durante el día puede eliminar al aldeano sospechoso de ser hombre lobo.

Relato introductorio: En un pueblo muy lejano están sucediendo asesinatos misteriosos. Los aldeanos piensan que un habitante por las noches se transforma en hombre lobo y comete estos crímenes. Es por esto que cada día todos se reúnen después del mediodía en la plaza central para eliminar al presunto hombre lobo.

Relato de la primera noche: La luz de la luna llena está haciendo estragos, aldeanos que eran pacíficos hace unas horas se transforman en criaturas violentas y sedientas de sangre. Se oyen aullidos y rugidos feroces. La leyenda se ha hecho realidad, un hombre lobo ha nacido…

Relato del día: Amanece en el pueblo, los aldeanos están muy asustados. Se oyeron gritos en la madrugada, la noche se cobró una nueva víctima. El hombre lobo continúa con sus crímenes. Todos se han reunido en la plaza central para debatir y votar quien es el hombre lobo, al cual van a eliminar.

Relato de la noche: Los aldeanos convencidos de haber eliminado al hombre lobo, se van tranquilos a dormir. En ese momento, a la luz de la luna llena un aldeano se transforma en hombre lobo y busca a su próxima víctima.

Flujo de la partida

1. El moderador reparte las cartas(roles) a cada uno de los participantes. (Se tiene en cuenta la cantidad de Hombres lobo y aldeanos para que la partida sea óptima, según se indicó en el cuadro anterior). Cada uno de los participantes observa su carta sin mostrarla a los demás y adopta el rol que le asignaron.

   INICIA LA PARTIDA
2. El moderador comienza el relato inicial.
3. El moderador indica que ya es de noche.
    a. Todos los participantes son privados de la vista momentáneamente.
    b. El moderador indica que los hombres lobo pueden recuperar la vista. (En este momento se reconocen entre ellos)
    c. El moderador indica que deben elegir a un aldeano para eliminar de la partida.
    d. Los hombres lobo en completo silencio(usando gestos o cualquier otra señal que sea difícil de percibir para los aldeanos) se comunican entre ellos para quedar de acuerdo a que aldeano deben eliminar del juego. (Puede que entren en votación)
    e. Los hombres lobo indican en silencio al moderador quien es el aldeano elegido.
4. El moderador indica que ya es de día.
    a. Todos los participantes recuperan la visión.
    b. El moderador comienza el relato y luego indica quien es el aldeano que fue eliminado.
    c. El aldeano eliminado debe mostrar su carta y salir de la partida. (Debe mantenerse en completo silencio para no perturbar el juego)
    d. El moderador indica que los participantes deben argumentar para eliminar al aldeano sospechoso de ser hombre lobo.
    e. Cada participante debe argumentar sobre quien cree que es un hombre lobo. (Los hombres lobo tienen que convencer a los aldeanos de que no eliminen a sus compañeros sin poner en descubierto su rol).
5. El moderador indica que los participantes deben realizar la votación.
    a. Cada participante emite un voto

   b. El moderador cuenta los votos y elimina al participante más votado.
6. Si se eliminan todos los hombres lobo, la partida termina y ganan los aldeanos. (Continúa "Fin de partida").
7. Si la cantidad de hombres lobo es mayor o igual a la cantidad de aldeanos, la partida termina y ganan los hombres lobo. (Continúa "Fin de partida").
8. Si existen más aldeanos que hombres lobo, se comienza un nuevo ciclo noche/día.
   FINALIZA LA PARTIDA
9. Fin de partida
   a. Cada personaje muestra su carta (rol).
   b. El moderador narra el final de la partida

Adaptaciones realizadas para el desarrollo de un videojuego

Se presentan distintas adaptaciones a la narrativa original teniendo en cuenta que el videojuego

El videojuego propondrá videos cortos con textos para favorecer el involucramiento de los participantes y que estos puedan mantener su atención durante el desarrollo de la partida.

En una primera instancia, se propone que las partidas se desarrollen en forma presencial en el aula, con la posibilidad de adaptarse para un entorno a distancia a través de videoconferencia.

El videojuego incluirá temporizadores y votaciones aleatorias a fin de que la partida no exceda un tiempo mayor al previsto por el docente.

En la etapa de votación nocturna, se propone que los aldeanos jueguen a un minijuego a fin de mantener su enfoque en la pantalla para favorecer el clima de la partida permitiendo que la votación nocturna no ponga en evidencia a los hombres lobo. Estas actividades generan un ranking de puntuación de aldeanos, que se tiene en cuenta cuando se elimina un aldeano si al finalizar el temporizador los hombres lobo se encuentran en un empate o no realizan ninguna votación.

Opción 1: Debido a que puede darse el caso donde los hombres lobo no lleguen a votar o su votación resulte en un empate al finalizar el temporizador de la noche, se eliminará automáticamente al aldeano que obtuvo menor puntuación en el minijuego.

Opción 2: El minijuego tiene dos razones, una de ellas es distraer a los aldeanos para que los hombres lobo pueda elegir al aldeano y la otra es eliminar a un aldeano automáticamente en el caso de que al finalizar el temporizador los hombres lobo no hayan escogido ningún aldeano o la votación haya resultado en un empate. De esta manera los aldeanos deben jugar obligadamente ya que si no corren el riesgo de ser eliminados de la partida.

En cuanto a lo tecnológico, también se contempla un pre-registro de los participantes donde ingresen su nombre y una foto/avatar para que puedan identificarse unívocamente entre ellos.

Por último, en la narrativa adaptada se resaltan los aspectos esenciales a ser implementados.

Modalidad: Presencial o Virtual Sincrónico

Estados: Creación de la partida, Informa el código a los jugadores, Espera a que se unan, Cierre de partida, Reparto de roles, momentos: Noche, Día, Votación, Fin del juego, Cancelación de partida.

Cantidad de participantes: La cantidad de hombres lobo sigue una proporción de 1 cada 3 o 4 aldeanos. En la tabla siguiente se muestra a modo de ejemplo la proporción indicada.

Jugadores Totales	Nº de Hombres Lobo	Nº de Aldeanos
4 - 5	1	3 - 4
5 - 6	2	4 - 5
20 - 25	5	15 - 20

Roles: Moderador, Hombre Lobo, Aldeano

Moderador: Dirige la partida.

Hombre Lobo: Durante la noche puede votar para eliminar a un aldeano.

Aldeano: Durante el día puede votar para eliminar al aldeano sospechoso de ser hombre lobo.

Relatos:

- **Relato introductorio:** En un pueblo medieval muy lejano están sucediendo asesinatos misteriosos durante la luna llena. Los aldeanos creen en una antigua leyenda de hombres lobo, por lo que sospechan que uno de sus vecinos por las noches se transforma en lobo y comete estos crímenes. Es por esto que cada día todos se reúnen después del mediodía en la plaza central para eliminar al aldeano que se transforma en lobo.

- **Relato de la primera noche:** La luz de la luna llena está haciendo estragos, aldeanos que eran pacíficos hace unas horas se transforman en criaturas violentas y sedientas de sangre. Se oyen aullidos y rugidos feroces. La leyenda se ha hecho realidad, un hombre lobo ha nacido…

- **Relato del día:** Amanece en el pueblo, los aldeanos están muy asustados. Se oyeron gritos en la madrugada, la noche se cobró una nueva víctima (Se muestra la víctima). El hombre lobo continúa con sus crímenes. Todos se han reunido en la plaza central para debatir y votar quien es el hombre lobo, al cual van a eliminar.

- **Relato de la noche:** Los aldeanos convencidos de haber eliminado al hombre lobo, se van tranquilos a dormir. En ese momento, a la luz de la luna llena un aldeano se transforma en hombre lobo y busca a su próxima víctima.
- **Relato final:**
    - Aldeanos ganan: Fue una lucha tortuosa y agotadora pero por fin los aldeanos pudieron deshacerse de la plaga de hombres lobo…. A partir de ahora estarán atentos a cada luna llena.
    - Hombres lobos ganan: El hambre insaciable de los voraces hombres lobo no se ha calmado, ya eliminaron a todo un pueblo…seguirán en camino en busca de más víctimas.

Restricciones de una partida:

- Cantidad de ciclos: Un ciclo comienza con la etapa de la noche y finaliza con la votación del hombre lobo que se realiza en el día. No se pueden superar los 3 ciclos, aunque esto depende de la planificación del docente y la cantidad de participantes.
- Temporizador: Cada etapa del juego tiene un temporizador para evitar la dispersión de los jugadores.
- Eliminaciones automáticas: Si finaliza el temporizador de una etapa y no se ha realizado una votación el sistema elimina a un aldeano automáticamente, esto puede ser teniendo en cuenta un sistema de puntos del minijuego o aleatoriamente.
- Participantes registrados: los participantes deberán registrarse en la plataforma con un nombre y una foto/avatar para poder ser identificados unívocamente.
- Partida sincrónica: La partida exige que haya sincronicidad entre los participantes ya sea presencial o a distancia.

Estados:

CREACIÓN DE LA PARTIDA

1. El moderador presiona un botón para crear una partida.
2. El sistema crea la partida y retorna el código para compartirla.
3. El moderador debe elegir si desea aceptar individualmente las peticiones de los participantes o permitir que se unan automáticamente.
4. El sistema entra en espera hasta que se complete el cupo de participantes o el moderador inicie la partida.
5. El moderador comparte el código de la partida a los distintos participantes

   a. Cada participante ingresa el código de la partida en el juego y presiona el botón unirse a la partida
   b. Cada participante debe ingresar su nombre y foto de perfil/avatar
6. El sistema informa cuando se ha completado el cupo mínimo de participantes
7. El sistema muestra cada participante que ingresa para sea autorizado
8. El moderador presiona el botón comenzar partida
9. El sistema cierra la partida para impedir que se unan nuevos participantes

INICIO DE LA PARTIDA

10. El moderador presiona el botón de "Comenzar relato inicial".
11. El sistema reproduce un video con el relato inicial.
12. El sistema reparte los roles a cada uno de los participantes (siguiendo una ecuación que define la cantidad de Hombres lobo para que la partida sea óptima)
13. El sistema muestra a cada participante en su pantalla su rol asignado junto con las recomendaciones para interpretarlo.

NOCHE

14. El moderador hace clic en el botón "Comenzar noche"
    a. El sistema muestra un video con el relato de noche
    b. El sistema inicia el temporizador de noche
    c. El sistema muestra a los aldeanos una pantalla para indicar de noche,
        i. Los aldeanos deben realizar distintas acciones en un minijuego para mantenerse atentos a su pantalla.
    d. El sistema muestra a los hombres lobo una pantalla de votación donde aparecen todos los aldeanos. Dentro de la pantalla también los hombre lobo pueden ver quienes son sus compañeros.
        i. Cada hombre lobo presiona en la foto/avatar de un aldeano para seleccionar a su víctima.(Solo tiene 1 voto que puede cambiar mientras dure el temporizador)
        ii. Si el temporizador lleva más de la mitad de tiempo transcurrido, el sistema selecciona un aldeano al azar para acelerar la elección.
    e. Cuando finaliza el temporizador de noche el sistema selecciona al aldeano eliminado teniendo en cuenta lo siguiente:
        i. Se selecciona el aldeano con mayoría de votos.
        ii. Si no, el primer aldeano que algún hombre lobo seleccionó.
        iii. Si no se registran votos, el sistema escoge un aldeano teniendo en cuenta su puntuación en el minijuego.
15. El sistema muestra al moderador el aldeano seleccionado por los hombres lobo.

DIA

16. El moderador indica que ya es de día y hace clic en el botón “Comienza el día”.
    a. El sistema muestra un video del relato del día.
    b. El moderador presiona el botón Revelar aldeano.
    c. El sistema muestra a todos los participantes el aldeano eliminado, indicando si es un hombre lobo o no.
        i. El aldeano eliminado sale de la partida y se le muestra una pantalla de espectador. (Debe mantenerse en completo silencio para no perturbar el juego)
17. El sistema verifica que existen más aldeanos que hombres lobo para continuar la partida, si no Continúa “Fin de Partida”:
18. El moderador indica que los participantes deben argumentar para eliminar al próximo aldeano sospechoso haciendo clic en el botón “Iniciar Argumentación”.
    a. El sistema inicia un temporizador de argumentación
    b. Cada participante debe argumentar sobre quien cree que es un hombre lobo. (Los hombres lobo tienen que convencer a los aldeanos para que no eliminen a sus compañeros).
19. El moderador hace clic en el botón “Iniciar Votación”, indicando a los aldeanos que deben realizar la votación.
    a. El sistema inicia un temporizador de votación.
    b. Cada aldeanos emite un voto *(Pantalla de votación)*
    c. Si finaliza el temporizador o todos los aldeanos emiten su voto, el sistema cuenta los votos y elimina al participante más votado.
20. Si existen más aldeanos que hombres lobo, la partida continúa.
    a. El moderador repite los pasos desde el estado NOCHE.

CONDICIONES DE FINALIZACIÓN

Si se eliminan todos los hombres lobo, la partida termina y ganan los aldeanos.

Si existen más hombres lobo que aldeanos, la partida termina y ganan los hombres lobo.

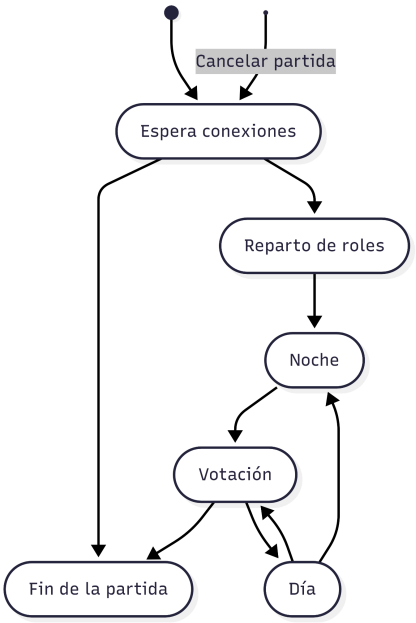
Si se excede el tiempo del temporizador la partida termina.

FINALIZA LA PARTIDA

Fin de partida*(Pantalla de finalización)*

a. El sistema muestra el rol de todos los aldeanos.
b. El moderador presiona el botón de “Comenzar relato final”
c. Se finaliza la partida

Diagramas de Estados



Fuente: Elaboración propia

4.2.3 Vistas

La siguiente sección muestra las vistas específicas de los estados mencionados anteriormente.

Estado Inicial

El estado inicial contempla la pantalla de bienvenida según el rol del usuario.

Pantalla inicial moderador

Pantalla inicial jugadores

Estado Esperando conexiones

Pantalla Moderador

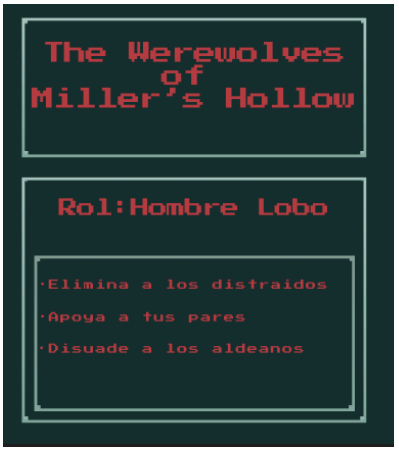
Pantalla Usuarios

Estado Reparto de roles

Si bien el estado indica el reparto de roles, este se hace en segundo plano mientras los usuarios se encuentran en la pantalla de introducción. Una vez que se termina la introducción se muestra la pantalla de roles según el rol que se le haya asignado al usuario.

Pantalla de relato inicial usuarios

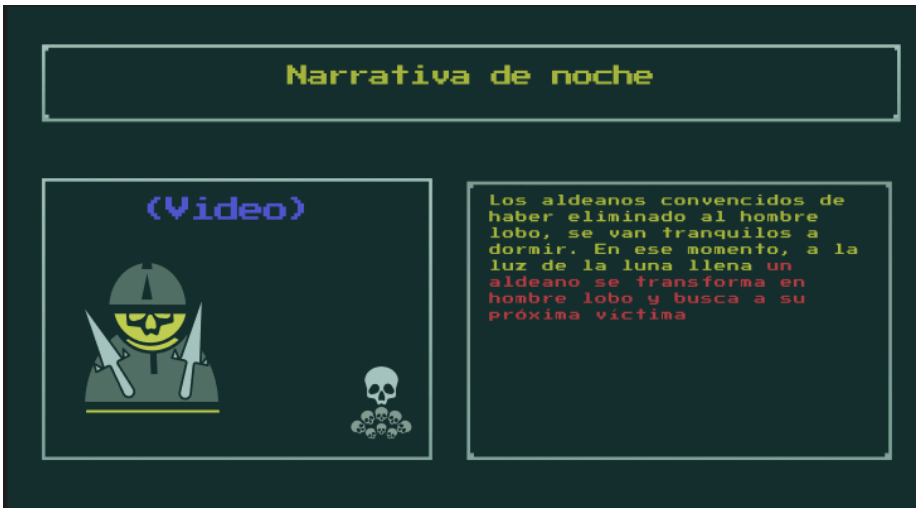
Pantallas de Roles



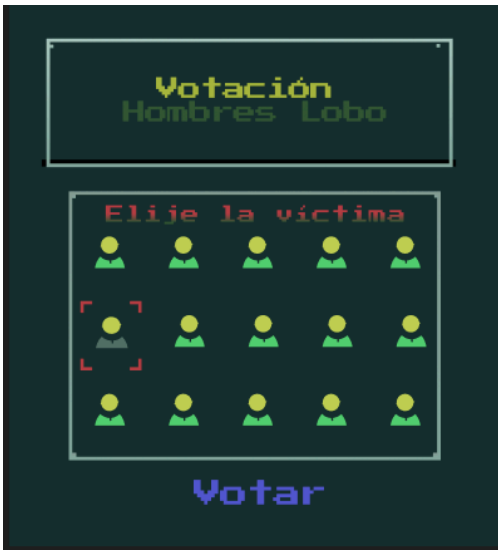
Estado Noche

Una vez que se inicia la noche se muestra una pantalla que muestra un relato para luego pasar a la pantalla correspondiente al rol asignado al usuario.

Pantalla de relato de noche



Pantalla de Hombre Lobo y Aldeanos



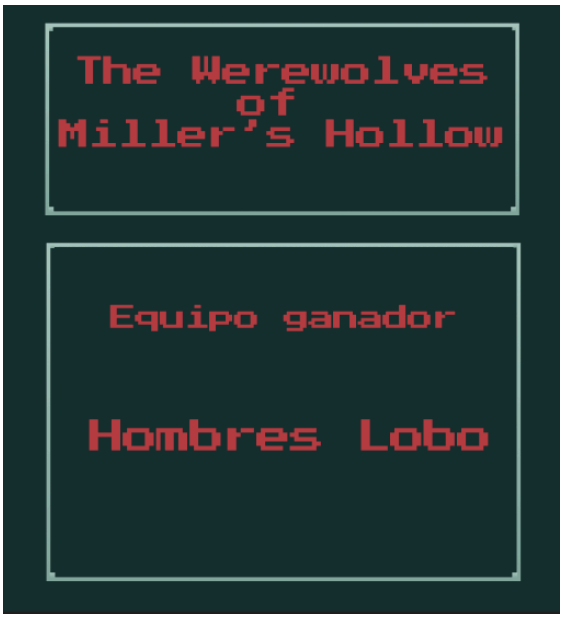
Estado Día

Pantallas de día para ambos roles

Estado de Votación

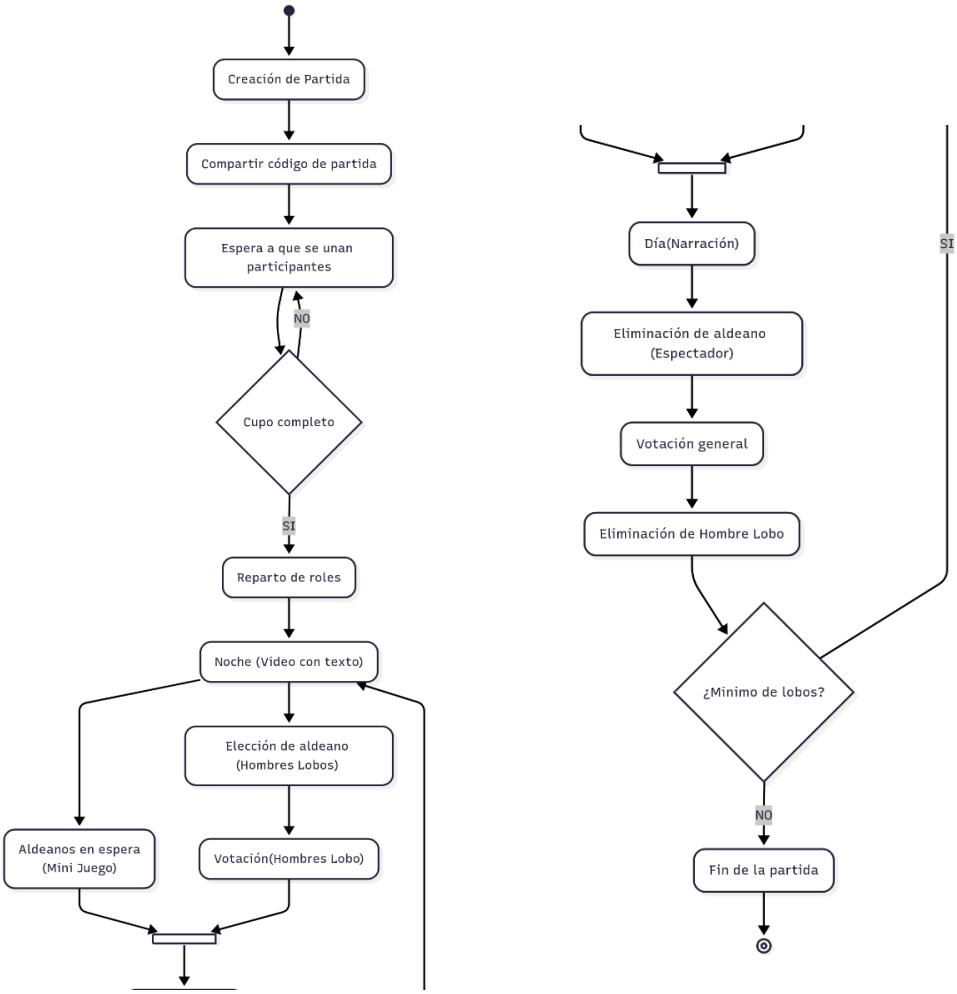
Pantalla de votación general

Estado de Fin de la partida



Diagramas de Actividades

Este diagrama muestra el flujo general del videojuego, donde se indican cada unos de los pasos que siguen los usuarios hasta finalizar la partida.



Fuente: Elaboración propia

4.3. Criterios e indicadores de evaluación

Se evaluó la viabilidad técnica a partir de los siguientes criterios: Correctitud arquitectónica (Gamma, 1994), (Nystrom, 2014) y (Buschmann, 1996); Rendimiento y eficiencia (Gregory, 2021)(Fabian, 2018); Escalabilidad y extensibilidad (Bass, Clements y Kazman, 2013), (Pressman y Maxim, 2020) y (Gamma et al.,1994) ; Coherencia Multijugador (Claypool y Claypool, 2006), (Gregory, 2021),(Kim y Zhao, 2020).

A continuación se presentan los componentes de cada uno de los criterios.

4.3.1. Correctitud arquitectónica

- Implementación fiel del patrón ECE.
- Bajo acoplamiento entre componentes.
- Claridad en la gestión de estados.

4.3.2. Rendimiento

- Estabilidad del sistema con múltiples entidades.
- Ausencia de degradación crítica al aumentar la carga.
- Uso eficiente de recursos.

4.3.3. Escalabilidad y extensibilidad

- Facilidad para agregar nuevos componentes y estados.
- Impacto mínimo en código existente.
- Reutilización de componentes.

4.3.4. Coherencia multijugador

- Consistencia del estado compartido.
- Manejo adecuado de desincronizaciones.
- Robustez frente a latencias variables.

A modo de cierre de este capítulo se puede indicar que la viabilidad técnica del framework propuesto se evaluó mediante el prototipado de un videojuego multijugador funcional, concebido como un artefacto experimental. El prototipo permitió analizar la implementación del patrón Entidad–Componente–Estado, la coherencia del modelo servidor autoritativo y el comportamiento del sistema bajo condiciones controladas de ejecución, a partir de métricas técnicas y análisis arquitectónico.

Capítulo Nº 5: Análisis, presentación y discusión de los Resultados

Análisis e interpretación de resultados

Los datos recolectados se analizaron en función de: los criterios definidos en el capítulo anterior, los objetivos del framework GameCore, comparaciones cualitativas con motores o frameworks existentes (Unity/EC clásico).

Se identifican: fortalezas, limitaciones técnicas, oportunidades de mejora.

Criterios	GameCore	Unity Clásico	Unreal Engine
Fortalezas	- Facilidad de uso - Código abierto - Gestión de estados por defecto	- Facilidad de uso - Posee muchos componentes	- Alta calidad en gráficos 3D - Programación visual con bloques
Limitaciones Técnicas	No posee muchos componentes por encontrarse en desarrollo	Gestión de memoria deficiente para muchos objetos	- Requiere hardware potente - Curva de aprendizaje alta
Oportunidades de mejora	Creación de comunidad	Programación visual con bloques	Facilidad de uso

Este trabajo contribuye al avance tecnológico y metodológico en el sector del desarrollo de videojuegos mediante:

- La integración innovadora de patrones arquitectónicos Componentes y State
- La presentación de Framework open-source reutilizable para la comunidad académica y profesional
- La propuesta de una metodología de validación replicable para frameworks similares

Se espera que GameCore facilite la democratización del desarrollo de videojuegos multijugador, reduciendo barreras técnicas y promoviendo la innovación en el sector, especialmente beneficiando a desarrolladores independientes y equipos académicos con recursos limitados.

Se presentan a continuación posibles líneas de investigación y trabajo futuras

Posibles líneas de investigación futuras

- Optimización y Escalabilidad Distribuida. En el desarrollo de algoritmos para distribución automática de GameObjects a través de múltiples servidores y técnicas de clustering para juegos multijugador masivos, incluyendo predicción de carga mediante Machine Learning.

- Inteligencia Artificial Basada en Componentes. Mediante la integración de IA adaptativa en Componentes para comportamientos inteligentes de NPCs y sistemas de decisión colaborativa entre GameObjects mediante arquitecturas multi-agente.
- Seguridad y Sistemas Anti-Cheat (antitrampas) Avanzados. Mediante investigaciones sobre la validación autoritativa que permitan detectar anomalías en tiempo real y exploración de tecnologías blockchain para garantizar integridad de estados críticos del juego.
- Edge Computing y Compensación de Latencia. Desarrollo de estrategias para migrar Components a edge servers según proximidad geográfica y algoritmos predictivos para compensación automática de latencia de red.
- Análisis de Datos y Player Behavior Analytics. Implementación de sistemas de métricas en tiempo real embebidos en Components y aplicación de técnicas de big data para análisis avanzado de comportamiento de jugadores y optimización de mecánicas de juego.

Capítulo Nº 6: Conclusiones y Trabajos Futuros

Este trabajo contribuye al avance tecnológico y metodológico en el sector del desarrollo de videojuegos mediante:

- La integración innovadora de patrones arquitectónicos Componente y Estado
- La presentación de Framework open-source reutilizable para la comunidad académica y profesional
- La propuesta de una metodología de validación replicable para frameworks similares

Se espera que GameCore facilite la democratización del desarrollo de videojuegos multijugador, reduciendo barreras técnicas y promoviendo la innovación en el sector, especialmente beneficiando a desarrolladores independientes y equipos académicos con recursos limitados.

Se identifican posibles líneas de investigación futuras en cuanto a la optimización y Escalabilidad Distribuida. Desarrollo de algoritmos para distribución automática de GameObjects a través de múltiples servidores y técnicas de clustering para juegos multijugador masivos, incluyendo predicción de carga mediante Machine Learning; La Inteligencia Artificial Basada en Componentes. Mediante la integración de IA adaptativa en Componentes para comportamientos inteligentes de Personajes No Jugadores (NPCs) y sistemas de decisión colaborativa entre GameObjects mediante arquitecturas multi-agente. La Seguridad y Sistemas Anti-Cheat (anti trampas) Avanzados. Mediante investigaciones sobre la validación autoritativa que permitan detectar anomalías en tiempo real y exploración de tecnologías blockchain para garantizar integridad de estados críticos del juego.

Referencias Bibliográficas

Argentina. (2005). Ley 26.043. Clasificación y control de videojuegos. https://www.argentina.gob.ar/normativa/nacional/ley-26043-107891/texto

Bafandeh Mayvan, B., Rasoolzadegan, A., & Ghavidel Yazdi, Z. (2017). The state of the art on design patterns: A systematic mapping of the literature. Journal of Systems and Software, 125, 93–118. https://doi.org/10.1016/j.jss.2016.11.030

Barakat, N. H. (2019, April). A framework for integrating software design patterns with game design framework. In Proceedings of the 8th International Conference on Software and Information Engineering (pp. 47–50).

Beecrowd.com. (n.d.). Framework: Descubre qué es y para qué sirve. Retrieved from https://beecrowd.com/es/blog-posts/framework-3/

Berg Marklund, B., Engström, H., Hellkvist, M., & Backlund, P. (2019). What empirically based research tells us about game development. The Computer Games Journal, 8, 179–198.

Bezditnyi, V., & Chebanyuk, O. (2024). Software Engineering Fundamentals to Design Application for Modern Game Engines.

Blanch, R., & Beaudouin-Lafon, M. (2006). Programming rich interactions using the hierarchical state machine toolkit. En Proceedings of the Working Conference on Advanced Visual Interfaces (pp. 51–58).

Borowa, K., Zalewski, A., & Saczko, A. (2021). Living with technical debt—a perspective from the video game industry. IEEE Software, 38(6), 65–70.

Claypool, M., & Claypool, K. (2009, April). Perspectives, frame rates and resolutions: it's all in the game. In *Proceedings of the 4th International Conference on Foundations of Digital Games* (pp. 42-49).

de Freitas, L. G., Reffatti, L. M., de Sousa, I. R., Cardoso, A. C., Castanho, C. D., Bonifácio, R., & Ramos, G. N. (2012, May). Gear2d: an extensible

component-based game engine. In Proceedings of the International Conference on the Foundations of Digital Games (pp. 81-88).

Dockins, K. (2017). Design Patterns in PHP and Laravel (pp. 100–140). Apress.

Eden, A. H. (2007). Three paradigms of computer science. Minds and Machines, 17(2), 135–167. https://doi.org/10.1007/s11023-007-9060-8

Eden, Amnon. (2007). Three Paradigms of Computer Science. Minds and Machines. 17. 135-167. 10.1007/s11023-007-9060-8.

Fabian, S. (2018). Entity component system architecture. En Game Programming Patterns in Practice. https://gameprogrammingpatterns.com/

FECYT – Fundación Española para la Ciencia y la Tecnología. (2018). Manual de Frascati 2015: Guía para la recopilación y presentación de información sobre la investigación y el desarrollo experimental.

Ferreira, M. D. O. (2024). Um motor de Jogos 2D em Typescript (Bachelor's thesis).

Gala, R. P. (2019). Aproximación a los aspectos legales, sindicales y comerciales de la industria de videojuegos en Argentina.

GameMaker Manual (LTS). (2024). GameMaker.io. Recuperado de https://manual.gamemaker.io/lts/en/

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of reusable object-oriented software. Addison-Wesley.

Gregory, J. (2018). Game Engine Architecture (3rd ed.). CRC Press.

Härkönen, J. (2019). Entity-component-system architecture for game engines. Journal of Game Development, 13(1), 22–35.

Johnson, R. E. (1997). Frameworks = (components + patterns). Communications of the ACM, 40(10), 39–42.

Kim, J., & Zhao, M. (2020). Secure and low-latency architectures for multiplayer online games. IEEE Transactions on Games, 12(3), 245–256. https://doi.org/10.1109/TG.2019.2954672

Lewis, M., & Jacobson, J. (2002). Game engines in scientific research. Communications of the ACM, 45(1), 27–31. https://doi.org/10.1145/502269.502274

Limura, T., Hazeyama, H., & Kadobayashi, Y. (2004). Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. Proceedings of ACM SIGCOMM 2004 Workshops on NetGames '04 Network and System Support for Games - SIGCOMM 2004 Workshops. ACM SIGCOMM 2004 workshops, Portland, Oregon, USA. https://doi.org/10.1145/1016540.1016549

Macedo, B. H., Araujo, G. F., Silva, G. S., Crestani, M. C., Galli, Y. B., & Ramos, G. N. (2015, November). Evolving finite-state machines controllers for the simulated car racing championship. In 2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames) (pp. 160-172). IEEE.

Malatesta, F. (2015). Learning Laravel 's Eloquent. Packt Publishing.

Manual de Frascati 2015: Guía para la recopilación y presentación de información sobre la investigación y el desarrollo experimental.

Marklund, B. B. (2019). Game development research: Methods, techniques and tools. Springer. https://doi.org/10.1007/978-3-030-16228-6

Mizutani, W. K., & Kon, F. (2020, March). Unlimited rulebook: A reference architecture for economy mechanics in digital games. In 2020 IEEE International Conference on Software Architecture (ICSA) (pp. 58–68). IEEE.

Nystrom, R. (2014). Game programming patterns. Genever Benning. https://gameprogrammingpatterns.com

Organisation for Economic Co-operation and Development. (2015). Frascati manual 2015: Guidelines for collecting and reporting data on research and experimental development. OECD Publishing. https://doi.org/10.1787/9789264239012-en

Ormeño, E. G., & Ferrarini Oliver, C. (2025). GameCore: un framework backend para videojuegos autoritativos. In XXVII Workshop de Investigadores en Ciencias de la Computación (Mendoza, 10 y 11 de abril de 2025).

Pavan, P. Y., Gour, S., & Bhaiya, L. K. P. (2024). Designing custom state machine instead of animation state machine in unity for developing 3rd person action game. *International Journal of Computer Engineering and Technology*, *15*(3), 48-64.

Pérez Ibarra, J., Quispe, M., Mullicundo, L., & Lamas, M. (2021). Arquitecturas backend para videojuegos multijugador en tiempo real. Revista Latinoamericana de Ingeniería de Software, 9(2), 55–68.

Pérez, M. (2021). Modelos de gestión de estados en videojuegos autoritativos. Computer Graphics Forum, 31(1), 89–103.

Politowski, C., Petrillo, F., & Guéhéneuc, Y.-G. (2021). Game development frameworks: A systematic literature review. Journal of Systems and Software, 171, 110824. https://doi.org/10.1016/j.jss.2020.110824

Racheva, S. (2019). Testing practices in indie game development from a software engineering perspective: An exploratory study.

Ravaillac, T., & Huot, S. (2019). Polyphony: Programming interfaces and interactions with the entity-component-system model. Proceedings of the ACM on Human-Computer Interaction, 3(EICS), 1–22.

Schmidt, D. C., Stal, M., Rohnert, H., & Buschmann, F. (2013). Pattern-oriented software architecture: Patterns for concurrent and networked objects. John Wiley & Sons.

Sepel'a, B. M. Data-Oriented Design in Game Development.

Sepela, J. (2024). Understanding Unity’s GameObject and MonoBehaviour architecture. Game Engine Architecture Review, 6(1), 1–12.

Sintaro, S., Salaky, D. T., Latumakulita, L. A., Bernard, B., Surahman, A., & Islam, N. (2023). Implementation and comparison in using state pattern on main character movement (Case study: Pocong Jump video game version 1.0). BAREKENG: Jurnal Ilmu Matematika dan Terapan, 17(2), 955–968.

Taylor, R. N. (2018). Software architecture foundations, theory, and practice. En Software Architecture in Practice (4th ed.). Addison-Wesley.

The Big Game Engine Report of 2025. (2025). Investgame.net. https://investgame.net/wp-content/uploads/2025/02/The_Big_Game_Engines_Report_of_2025.pdf

Unity Technologies. (2023). Unity Scripting API Reference. Unity Documentation.

Unity User Manual. (2017). Unity documentation. Recuperado de https://docs.unity3d.com/Manual/index.html

Unreal engine Terminology. (2025). Epicgames.com. Recuperado de https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-terminology

Yuni, J. U., & Urbano, A. C. (2014). Técnicas para investigar.

Apéndice A: GDD Werewolf

1. VISIÓN GENERAL

1.1. Tema / Ambientación / Género

- **Tema Principal:** Engaño, deducción social y supervivencia
- **Ambientación:** Pueblo medieval gótico bajo una maldición lunar
- **Género:**
    - **Primario:** Juego social de deducción (Social Deduction)
    - **Secundario:** Party Game, Multijugador en línea
    - **Estilo:** Casual competitivo

1.2. Mecánicas Principales (Resumen)

- **Ciclo Día/Noche:** Sistema por turnos que alterna entre fases de acción secreta y debate público
- **Votación Asimétrica:**
    - Votación nocturna secreta (Hombres Lobo)
    - Votación diurna pública (Todos los jugadores)
- **Minijuegos Nocturnos:** Actividades interactivas para aldeanos durante la noche
- **Roles Especiales:** Cada jugador tiene habilidades únicas según su rol
- **Comunicación Integrada:** Chat de texto
- **Sistema de Desempate Automático:** Basado en puntuación de minijuegos

1.3. Plataformas Objetivo

- **Plataforma Primaria:** Web (HTML5/WebGL)
- **Plataformas Secundarias:**
    - Windows (PC)
    - macOS
- **Tecnología:** Desarrollado en GameCore con PHP y Laravel.

1.4. Alcance del Proyecto

- **Duración del Proyecto:** 3 meses
- **Tiempo de Juego:** 10-20 minutos por partida
- **Equipo de Desarrollo:** 1 persona
- **Jugadores por Partida:** 5-20 jugadores
- **Licencias:**
    - Motor de juego: MIT
    - Assets: licencias libres

- Música/SFX: Licencias libres

1.5. Influencias

- **Among Us**
  - **Medio:** Videojuego
  - **Influencia:** Popularización del género de deducción social digital, mecánicas de votación y acusación simplificadas, éxito en streaming
- **Los Hombres Lobo de Castronegro (Juego de Mesa)**
  - **Medio:** Juego de mesa físico
  - **Influencia:** Sistema de roles asimétricos, ciclo día/noche, interacción social pura, narrativa de pueblo medieval

1.6. Elevator Pitch

Werewolves Game es un juego de deducción social multijugador donde unos aldeanos deben descubrir quiénes son los hombres lobo que los atacan mediante el debate y la deducción, mientras estos intentan eliminarlos en secreto durante la noche.

1.7. Descripción del Proyecto (Resumen)

Werewolves game es una adaptación digital del clásico juego social de deducción, diseñada específicamente como trabajo final de carrera. El juego enfrenta a dos facciones (aldeanos y hombres lobo) en una batalla de engaños y estrategia, donde la comunicación y la observación son clave para la victoria.

Implementado en 2D con un estilo artístico amigable, el juego incluye todas las mecánicas clásicas del género más innovaciones como minijuegos interactivos durante las fases de espera y un sistema de desempate automático basado en rendimiento.

1.8. Descripción del Proyecto (Detallada)

El proyecto consiste en desarrollar un videojuego multijugador en línea del género "social deduction", basado en el clásico juego de mesa "Los Hombres Lobo de Castronegro". El objetivo principal es crear una experiencia digital completa que mantenga la esencia social del original mientras aprovecha las ventajas del medio digital.

Componentes Técnicos Principales:

1. **Backend Multijugador:** Servidor en GameCore
2. **Cliente Web:** Interfaz desarrollada en HTML5/Canvas o WebGL
3. **Base de Datos:** MySQL para persistencia de cuentas y estadísticas
4. **Chat en Tiempo Real:** Texto

Características de Jugabilidad:

- Partidas rápidas de 10-20 minutos
- Sistema de roles balanceado (aldeanos, lobos)
- Interfaz intuitiva diseñada para nuevos jugadores
- Sistema de tutorial integrado

Objetivos Educativos del Proyecto:

- Demostrar la viabilidad técnica de GameCore
- Documentar el proceso de desarrollo

2. Originalidad

2.1. Mecánicas de Juego Principales (Detalladas)

2.1.1. Sistema de Minijuegos Nocturnos

Detalles: Durante la fase nocturna, mientras los hombres lobo deliberan en secreto, los aldeanos no permanecen inactivos. Participan en minijuegos simples pero atractivos que generan una puntuación. Esta puntuación no solo mantiene a los jugadores comprometidos, sino que también sirve como mecanismo de desempate automático.

Cómo Funciona:

1. Cada aldeano ve un minijuego diferente cada noche (variedad para evitar monotonía)
2. Los minijuegos son rápidos (30-60 segundos) y requieren atención
3. La puntuación se calcula basándose en precisión, velocidad o combinación de ambas
4. Si los hombres lobo no votan o empatan en su votación, el sistema elimina automáticamente al aldeano con menor puntuación. Esto incentiva a todos los jugadores a participar activamente en todo momento

Ejemplos de Minijuegos:

- **Reacción Rápida:** Pulsar botones que aparecen aleatoriamente
- **Memoria:** Recordar secuencias de símbolos
- **Precisión:** Detener un indicador en el área correcta
- **Ritmo:** Seguir un patrón de pulsaciones

2.1.2. Sistema de Debate con Herramientas Integradas

Detalles: Durante la fase de debate diurno, los jugadores no solo hablan, sino que tienen herramientas digitales para presentar "evidencia" y argumentar de manera más efectiva.

Cómo Funciona:

1. **Sistema de Notas:** Cada jugador puede tomar notas privadas durante el juego
2. **Votación de Pruebas:** Jugadores pueden nominar "pruebas circunstanciales" para votación

3. **Marcadores de Comportamiento:** Sistema para marcar jugadores como sospechosos/confiables

2.1.4. Modos de Juego Variables

Detalles: El juego incluye diferentes configuraciones de partida que cambian fundamentalmente cómo se juega, ofreciendo variedad y rejugabilidad.

Cómo Funciona:

1. **Modo Clásico:** Roles tradicionales, reglas básicas
2. **Modo Caos:** Roles aleatorios cada noche, reglas cambiantes
3. **Modo Tormenta:** Tiempos reducidos, presión constante
4. **Modo Silencio:** Chat limitado, más énfasis en observación
5. Cada modo tiene balance diferente y atrae a distintos tipos de jugadores

3. HISTORIA Y JUGABILIDAD

3.1. Historia (Resumen)

En el pueblo medieval de Hollowbrook, una antigua maldición se activa cada luna llena, transformando a algunos habitantes en bestias sedientas de sangre. Los aldeanos, aterrorizados, deben descubrir quiénes son los infectados antes de que sea demasiado tarde. Cada día trae nuevos sospechosos, cada noche trae nuevas víctimas. La desconfianza crece mientras la comunidad se desmorona.

3.2. Historia (Detallada)

El Pueblo de Hollowbrook:

Hollowbrook fue fundado hace siglos en un valle apartado, donde sus habitantes vivían en paz hasta que descubrieron las ruinas de un antiguo templo lunar. Al profanarlo, liberaron una maldición que había estado contenida durante milenios.

La Maldición Lunar:

Cada noche de luna llena, la maldición se activa aleatoriamente en algunos habitantes. Los afectados (Hombres Lobo) mantienen su forma humana durante el día pero pierden el control por la noche, transformándose en bestias que atacan a sus vecinos.

La Dinámica Social:

La paranoia se ha instalado en Hollowbrook. Nadie confía plenamente en nadie. Las amistades de toda la vida se rompen por sospechas infundadas. Las familias se dividen. El pueblo está al borde del colapso social mientras intentan identificar a los infectados.

Los Personajes:

- **Aldeanos Comunes:** Ciudadanos que intentan sobrevivir y proteger a sus seres queridos
- **Hombres Lobo:** Infectados que luchan contra sus instintos o los abrazan

Narrativos por Partida:

Cada partida cuenta una historia emergente basada en:

1. Las muertes que ocurren
2. Las acusaciones que se hacen
3. Las traiciones que se descubren
4. El resultado final

Elementos de Lore (expandible):

- Orígenes de la maldición
- Antiguas familias con secretos
- Artefactos protectores
- Rituales de curación
- Profecías sobre el fin de la maldición

3.3. Jugabilidad (Resumen)

Los jugadores son asignados aleatoriamente a una de dos facciones: Aldeanos o Hombres Lobo. El juego se desarrolla en ciclos de Día y Noche. Durante la Noche, los Hombres Lobo eligen una víctima en secreto, mientras los Aldeanos participan en minijuegos. Durante el Día, todos debaten y votan para ejecutar a un sospechoso. El juego termina cuando una facción alcanza sus condiciones de victoria.

3.4. Jugabilidad (Detallada)

3.4.1. Flujo Completo de una Partida

[LOBBY] → [ASIGNACIÓN DE ROLES] → [RELATO INICIAL] → [NOCHE 1] → [DÍA 1] → [VOTACIÓN 1] →

(Repetir ciclo Noche/Día hasta victoria) → [FIN DE PARTIDA] → [ESTADÍSTICAS]

3.4.2. Fase de Noche (Detallada)

Para Hombres Lobo:

1. Se revela la identidad de los compañeros lobo
2. Tienen 2 minutos para deliberar en chat privado
3. Cada lobo selecciona una víctima haciendo clic en su avatar
4. El sistema muestra votos en tiempo real solo a los lobos
5. Si hay empate o no hay votos, se usa el sistema de desempate automático
6. La víctima seleccionada es eliminada al amanecer

Para Aldeanos:

1. Ven pantalla negra con el mensaje "Es de noche..."
2. Después de 10 segundos, aparece el minijuego
3. Juegan activamente durante 1-2 minutos
4. Reciben puntuación basada en rendimiento
5. La puntuación afecta desempates y posiblemente otros sistemas

3.4.3. Fase de Día (Detallada)

Revelación Matutina:

1. Transición visual noche → día
2. Aparece el cuerpo de la víctima en la plaza
3. Se revela el rol de la víctima
4. La víctima se convierte en espectador (sin hablar)

Debate Público:

1. 5 minutos de tiempo para discutir
2. Chat de texto público para todos
3. Temporizador visible para presión

Proceso de Acusación:

1. Cualquier jugador puede acusar a otro
2. Las acusaciones aparecen destacadas en el chat
3. Se puede votar para "poner en el banquillo" a alguien
4. Máximo 3 jugadores pueden ser nominados simultáneamente

3.4.4. Fase de Votación (Detallada)

Votación Diurna:

1. Interfaz con avatares de todos los jugadores vivos
2. Cada jugador selecciona un sospechoso
3. Votación secreta hasta que termina el tiempo
4. Revelación pública de resultados
5. Si hay un claro ganador, es ejecutado inmediatamente
6. Si hay empate, se discute y revota o se muere el aldeano con menos puntos

Ejecución:

1. Animación de ejecución (estilo cartoon, no gráfico)
2. El ejecutado revela su rol
3. Se convierte en espectador

3.4.5. Condiciones de Victoria

Victoria Aldeanos:

- Eliminar a TODOS los Hombres Lobo
- No importa cuántos aldeanos queden vivos
- Se reproduce animación de celebración aldeana

Victoria Hombres Lobo:

- Número de lobos ≥ número de aldeanos
- Se reproduce animación de transformación y ataque
- Los lobos sobrevivientes celebran

3.4.6. Sistema de Roles

Roles Base (siempre disponibles):

1. **Aldeano:** Sin habilidades especiales
2. **Hombre Lobo:** Mata cada noche, puede ver a sus compañeros

Roles Especiales (desbloqueables/probabilísticos):

1. **Cazador:** Al morir, elige a alguien para llevar consigo

Proporciones Recomendadas:

- 5-6 jugadores: 1 Lobo, 4-5 Aldeanos (0-1 especial)
- 7-8 jugadores: 2 Lobos, 5-6 Aldeanos (1-2 especiales)
- 9-11 jugadores: 3 Lobos, 6-8 Aldeanos (2-3 especiales)

3.4.7. Interfaz de Usuario (UI)

Pantallas Principales:

1. **Pantalla de Login/Registro:** Simple, con opción de guest
2. **Lobby Principal:** Lista de salas, crear partida, unirse
3. **Sala de Espera:** Chat previo, configuración, lista jugadores
4. **Pantalla de Juego Principal:**
    - Sección central: Plaza del pueblo con avatares
    - Chat a la derecha
    - Información de juego arriba (tiempo, fase, vivos/muertos)
    - Acciones abajo (votar, usar habilidad, notas)
5. **Pantalla de Espectador:** Para jugadores eliminados
6. **Pantalla de Resultados:** Estadísticas finales

Elementos de UI Clave:

- Iconos claros para cada rol
- Indicador de fase (día/noche) prominente
- Temporizador siempre visible
- Indicador de tu propio rol (discreto pero accesible)
- Historial de eventos colapsable

4. ACTIVOS NECESARIOS

4.1. Gráficos 2D

Sprites de Personajes:

- **Avatares Base (16x16 o 32x32):**
    - Aldeano masculino (varias poses)
    - Aldeano femenino (varias poses)
    - Hombre Lobo (forma humana y forma bestia)
    - Roles especiales (Vidente, Médico, etc.)
- **Expresiones Faciales:**
    - Neutral
    - Sospechoso
    - Inocente
    - Asustado
    - Enfadado
    - Alegre (victoria)
- **Animaciones por Sprite:**
    - Idle (respiración sutil)
    - Seleccionado
    - Votado
    - Ejecutado
    - Transformación lobo

Interfaz de Usuario (UI):

- **Botones:**
    - Botón primario (normal, hover, pressed)
    - Botón secundario
    - Botón de votación
    - Botones de chat
- **Iconos:**
    - Iconos de roles (15-20 iconos)
    - Iconos de acciones
    - Iconos de estado
    - Iconos de temporizador
- **Elementos de UI:**

    - Barras de progreso
    - Marcadores de selección
    - Fondos de chat
    - Ventanas modales
    - Barras de salud/indicadores
- **Fuentes:**
    - Fuente principal (legible, estilo medieval)
    - Fuente para números/temporizadores
    - Varios tamaños pre-renderizados

Escenarios y Fondos:

- **Fondo de Pueblo:**
    - Plaza central (día y noche)
    - Casas individuales
    - Bosque circundante
    - Cielo (día, noche, atardecer)
- **Pantallas de Menú:**
    - Pantalla de título
    - Fondo de lobby
    - Fondo de sala de espera
    - Fondo de resultados
    - Fondo de pantalla de votación
- **Elementos Decorativos:**
    - Árboles, piedras, pozos, bancos
    - Elementos de iluminación (linternas, velas)
    - Efectos atmosféricos (niebla, luciérnagas, estrellas)

Efectos Visuales:

- **Transiciones:**
    - Día → Noche (fade a azul)
    - Noche → Día (fade a naranja)
    - Muerte/desaparición
- **Efectos de Habilidad:**
    - Transformación lobo
    - Ejecución
- **Efectos de Interfaz:**
    - Selección hover
    - Voto confirmado
    - Notificación nueva

Texturas y Patrones:

- Texturas de madera para UI
- Texturas de piedra para fondos

- Patrones medievales para bordes
- Gradientes para fondos

4.2. Sonido

Efectos de Sonido (SFX):

- **Interfaz:**
  - Clic botón
  - Hover botón
  - Cambio de pantalla
  - Notificación nueva
- **Juego:**
  - Votación emitida
  - Votación recibida
  - Cambio día/noche
  - Muerte/eliminación
  - Ejecución
  - Transformación lobo
- **Ambiente:**
  - Pueblo de día (pájaros, viento)
  - Pueblo de noche (grillos, búhos, aullidos)
  - Lluvia (ocasional)
  - Viento fuerte
- **Minijuegos:**
  - Sonidos de éxito/error
  - Sonidos de progreso
  - Ticks temporizador

Música:

- **Menú Principal:** Tema medieval misterioso
- **Lobby:** Música tensa pero suave
- **Día:** Música de pueblo tranquilo
- **Noche:** Música tensa, misteriosa
- **Debate:** Música de suspenso
- **Votación:** Música dramática, crescendo
- **Victoria Aldeanos:** Música alegre, liberadora
- **Victoria Lobos:** Música oscura, amenazante

Voces/Narración:

- **Narrador (opcional):**
  - Introducción a la partida
  - Anuncio día/noche
  - Anuncio de muerte

    - Final de partida
- **Efectos de Voz:**
    - Susurros (lobos comunicándose)
    - Gritos/gemidos (muerte)

4.3. Código

Cliente (Frontend):

- **Sistema de Red:** Conexión WebSocket, reconexión automática
- **Minijuegos:** 3-5 minijuegos implementados
- **Localización:** Sistema básico para múltiples idiomas
- **Configuración:** Gestor de opciones y ajustes

Servidor (Backend):

- **Servidor de Juego:** GameCore
- **Gestor de Salas:** Creación, unión, cierre de partidas
- **Lógica del Juego:** Resolución de acciones, cálculos, reparto de roles
- **Base de Datos:** MySQL para usuarios y estadísticas
- **API REST:** Para registro, login, estadísticas, gestión de estados, eventos
- **Sistema de Chat:** Mensajería en tiempo real
- **Validación:** Anti-cheat básico, validación de acciones

Scripts Específicos:

- **Script de Rol:** Comportamiento específico por rol
- **Script de Votación:** Gestión de votos, conteo, resultados
- **Script de Minijuego:** Lógica individual para cada minijuego
- **Script de Animación:** Control de animaciones por estado
- **Script de Sonido:** Gestión contextual de audio
- **Script de Efectos:** Partículas y efectos visuales

Herramientas de Desarrollo:

- **Editor de Configuraciones:** Para balancear roles y tiempos
- **Herramientas de Debug:** Overlays para desarrollo
- **Logger Extendido:** Registro detallado de eventos
- **Simulador:** Para probar partidas sin jugadores reales

4.4. Animaciones

Animaciones de Personajes:

- **Idle:** Respiración sutil, parpadeo ocasional

- **Selección:** Efecto de brillo o resaltado
- **Votación:** Animación de mano levantada o señalando
- **Ejecución:**
    - Aldeano: Desvanecimiento o caída
    - Lobo: Transformación y huida
- **Transformación Lobo:** Metamorfosis humana→bestia
- **Emociones:** Cambios faciales temporales

Animaciones de Interfaz:

- **Transiciones de Pantalla:** Fade in/out, slides
- **Botones:** Animación de press, hover states
- **Notificaciones:** Slide in/out, bounce
- **Temporizadores:** Barra de progreso animada
- **Votación:** Conteo animado de votos
- **Resultados:** Revelación progresiva

Efectos Especiales:

- **Día/Noche:** Ciclo gradual de iluminación
- **Habilidades:** Efectos visuales para cada habilidad
- **Muerte:** Efectos de partículas (polvo, luz)
- **Victoria:** Efectos de celebración (confeti, fuegos)

Animaciones de Minijuegos:

- **Reacción:** Botones que aparecen/desaparecen
- **Memoria:** Cartas que se voltean
- **Precisión:** Indicador que se mueve
- **Ritmo:** Notas que fluyen

Apéndice B: Publicación en CACIC 2025

Framework para Backend de Videojuegos basado en Patrones de Componentes y Estados

Tec. Martín Gómez[1], Mg. Emilio Ormeño[2], Mg.Cintia Ferrarini Oliver[1]

1.Departamento de Informática, 2.Instituto de Informática
Facultad de Ciencias Exactas, Físicas y Naturales, Universidad Nacional de San Juan, San Juan, Argentina
martinsj0811@gmail.com, eormeno@gmail.com, ferrarinicintia@gmail.com

Resumen. En este artículo se presenta una propuesta en proceso de un framework backend para videojuegos multijugador, realizado en el marco del Proyecto CICITCA "Framework USIM para el desarrollo de Aplicaciones Web", cuyo objetivo es optimizar el desarrollo de videojuegos mediante la integración de los patrones Entidad-Componente-Sistema (ECS) y Patrón Estado, a fin de lograr una arquitectura modular, escalable y mantenible. La implementación se está realizando en PHP/Laravel, e incluye sistemas de eventos, renderizado, gestión de estados y prefabs reutilizables. Se está aplicando una metodología cuanti-cualitativa para la validación del framework. Se espera que esta propuesta reduzca las barreras técnicas para desarrolladores independientes y académicos de la industria de los videojuegos.

Palabras Clave: Framework, Patrón Entidad Componente Sistema, Patrón de Diseño Estado.

1 Introducción

Este artículo presenta una propuesta de Trabajo Final de Licenciatura en Ciencias de la Computación en proceso. Se está llevando a cabo en el marco del Proyecto de Investigación CICITCA denominado "Framework USIM para el desarrollo de Aplicaciones Web", aprobado según resolución 2902-CS-UNSJ, en el Gabinete de Gamificación del Instituto de Informática de la Facultad de Ciencias Exactas, Físicas y Naturales de la Universidad Nacional de San Juan. La propuesta surge en respuesta a la creciente demanda de soluciones tecnológicas que optimicen el desarrollo de videojuegos multijugador, reduciendo significativamente los tiempos de desarrollo y que permitan disminuir errores en las versiones iniciales. La industria del videojuego presenta desafíos únicos en términos de sincronización en tiempo real, escalabilidad y mantenimiento de la integridad del estado del juego[1]. Este trabajo final de grado tiene los siguientes objetivos: desarrollar componentes clave del Framework GameCore[2], basado en la integración de patrones arquitectónicos Entidad Componente Sistema[3] y Patrón Estado[4], y validar el framework completo aplicando una metodología cuanti-cualitativa. Cabe aclarar que el framework mencionado ya se viene desarrollando en el proyecto señalado. A continuación se presenta la arquitectura tecnológica propuesta, la metodología de investigación que se está aplicando, los aportes de este trabajo al área disciplinar, las posibles líneas de acción y las referencias básicas consultadas.

2 Arquitectura Tecnológica Propuesta

El Framework GameCore implementa una arquitectura backend robusta utilizando PHP/Laravel como núcleo tecnológico, con las siguientes características distintivas:

- Patrón de Componentes: mediante la implementación de GameObjects modulares con Componentes reutilizables
- Gestión de Estados: State Components que permiten máquinas de estados complejas[5][6].
- Sistema de Eventos: mediante la comunicación desacoplada entre elementos del framework
- Servicios Persistentes: con la incorporación de GameServices para lógica transversal de juego
- Sistema de Renderizado: View System y Render System para sincronización cliente-servidor

- Prefabs: Templates para instanciación eficiente de estructuras complejas

3 Metodología de investigación

Se adopta un enfoque de investigación aplicada con una metodología cuali-cuantitativa. Dado que, por un lado, se analizarán desde un enfoque cuantitativo, métricas de rendimiento, escalabilidad y tiempos de desarrollo. Y por otro, se evaluará la experiencia de usuario (UX) de desarrolladores mediante encuestas y entrevistas semi-estructuradas.

De acuerdo al Manual de Frascati[8], este trabajo se clasifica como investigación aplicada orientada a la adquisición de nuevos conocimientos dirigidos hacia un objetivo práctico específico. Se espera que el framework proporcione seguridad, escalabilidad y mantenibilidad para la industria global del desarrollo de videojuegos.

4 Aportes del trabajo

Este trabajo contribuye al avance tecnológico y metodológico en el sector del desarrollo de videojuegos mediante:

- La integración innovadora de patrones arquitectónicos ECS y Estado[7]
- La presentación de Framework open-source reutilizable para la comunidad académica y profesional
- La propuesta de una metodología de validación replicable para frameworks similares

Se espera que GameCore facilite la democratización del desarrollo de videojuegos multijugador, reduciendo barreras técnicas y promoviendo la innovación en el sector, especialmente beneficiando a desarrolladores independientes y equipos académicos con recursos limitados.

5 Posibles líneas de investigación futuras

- Optimización y Escalabilidad Distribuida. Desarrollo de algoritmos para distribución automática de GameObjects a través de múltiples servidores y técnicas de clustering para juegos multijugador masivos, incluyendo predicción de carga mediante Machine Learning.
- Inteligencia Artificial Basada en Componentes. Mediante la integración de IA adaptativa en Componentes para comportamientos inteligentes de Personajes No Jugadores (NPCs) y sistemas de decisión colaborativa entre GameObjects mediante arquitecturas multi-agente.
- Seguridad y Sistemas Anti-Cheat (antitrampas) Avanzados. Mediante investigaciones sobre la validación autoritativa que permitan detectar anomalías en tiempo real y exploración de tecnologías blockchain para garantizar integridad de estados críticos del juego.

Referencias

1. Berg Marklund, B., Engström, H., Hellkvist, M., Backlund, P.: What empirically based research tells us about game development. The Computer Games Journal 8, 179–198 (2019).
2. Johnson, R.E.: Frameworks = (components + patterns). Communications of the ACM 40(10), 39–42 (1997).
3. Ravaillac, T., Huot, S.: Polyphony: Programming interfaces and interactions with the entity-component-system model. Proceedings of the ACM on Human-Computer Interaction 3(EICS), 1–22 (2019).
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of reusable object-oriented software. Addison-Wesley (1995).
5. Blanch, R., Beaudouin-Lafon, M.: Programming rich interactions using the hierarchical state machine toolkit. In: Proceedings of the working conference on Advanced visual interfaces, pp. 51–58 (2006).
6. Pérez, M.: Modelos de Gestión de Estados en Videojuegos Autoritativos. Comput. Graphics Forum 31(1), 89–103 (2021).
7. Bafandeh Mayvan, B., Rasoolzadegan, A., Ghavidel Yazdi, Z.: The state of the art on design patterns: A systematic mapping of the literature. J. Syst. Softw. 125, 93–118 (2017).
8. FECYT-Fundación Española para la Ciencia: Manual de Frascati 2015: Guía para la Recopilación y Presentación de Información sobre la Investigación y el Desarrollo Experimental (2018).