Universidad San Jorge

Escuela de Arquitectura y Tecnología

Grado en Diseño y Desarrollo de Videojuegos

Proyecto Final

Diseño y Desarrollo de un Videojuego Interactivo para Streaming en Twitch: Integración de Comandos de Usuario en Tiempo Real mediante Unity

Autor del proyecto: Mario González Romero

Director del proyecto: Dan Tarodo Cortés

Zaragoza, 9 de septiembre de 2025

universidad SANJORGE
GRUPO SANVALERO

Este trabajo constituye parte de mi candidatura para la obtención del título de Graduado en Diseño y Desarrollo de Videojuegos por la Universidad San Jorge y no ha sido entregado previamente (o simultáneamente) para la obtención de cualquier otro título.

Este documento es el resultado de mi propio trabajo, excepto donde de otra manera esté indicado y referido.

Doy mi consentimiento para que se archive este trabajo en la biblioteca universitaria de Universidad San Jorge, donde se puede facilitar su consulta.

Firma

Fecha

9 de septiembre de 2025

Dedicatoria y Agradecimiento

Quiero agradecer, en primer lugar, a todo el profesorado que me ha acompañado durante mi formación, desde la educación básica hasta la universitaria. Todos y cada uno de vosotros habéis puesto vuestro granito de arena para llegar a este momento de la mejor manera posible. Mención especial para mi tutora, África Domingo, por inspirarme a dar siempre el 100 % de mí mismo en los momentos más duros y por ser un apoyo incondicional. Gracias de corazón.

También quiero expresar mi gratitud a Dan Tarodo por su valiosa orientación en este exigente proyecto e ilusionarme con sus asignaturas, por dedicarme generosamente su tiempo y enseñarme su profesionalidad.

A mis padres, Miguel y Maite, por ser los pilares de mi vida y con vuestra fortaleza brindarme las oportunidades que me han permitido alcanzar este momento. A toda mi familia, porque tengo la mejor familia, tanto por los que hoy me acompañan como por los que ya no están presentes, gracias por vuestro cariño y apoyo incondicional; sin vosotros no sería quien soy. Yayo, *¡al fin lo hemos conseguido!*

A mi novia, María, por creer en mí en cada paso del camino, impulsarme a dar siempre mi mejor versión y brindarme tu apoyo incondicional; eres la mejor compañera de vida que uno puede tener.

No puedo olvidar a mis amigos y compañeros, en especial a los Hiperchetados, por sacarme una sonrisa en los momentos más difíciles y motivarnos mutuamente para seguir adelante.

Gracias a todos por estar. Este trabajo también es vuestro.

Tabla de contenido

Resumen

En un ecosistema de creación de contenidos donde la participación en directo y la baja latencia son diferenciales, las integraciones con Twitch siguen afrontando retos técnicos (cambios de scopes, reconexiones, disparidad Editor-build y documentación fragmentaria). Este proyecto desarrolla un **plugin propio para Unity 6** que implementa **OAuth**, **PubSub** y **Helix** de forma robusta (saneado de token, PING/PONG y re-suscripción, operaciones idempotentes sobre recompensas, página de éxito embebida y "build-safe"), y un **videojuego social tipo quiz (VotePlay)** que explota esa integración para permitir que la audiencia participe desde el chat con comandos simples. Se describe el diseño modular (GameManager, MenuManager, SpawnManager, SoundManager, ChatManager) y el flujo "chat-first", así como la validación con medidas de latencia extremo a extremo y pruebas de estabilidad en build. Los resultados muestran **captura instantánea** de votos y canjes, **resiliencia a microcortes** y una interfaz legible en escenarios de streaming. El trabajo incluye un **análisis económico** y plantea una doble vía de explotación (venta del plugin en Unity Asset Store y del juego en Steam), favoreciendo la sostenibilidad y la reutilización tecnológica en futuros proyectos.

Palabras clave

Twitch; Unity; OAuth; Helix; juego social; streaming interactivo; quiz; integración en tiempo real.

Abstract

In today's creator economy, live interaction and low latency are decisive, yet Twitch integrations still face technical hurdles (scope changes, reconnections, Editor-build discrepancies, and fragmented documentation). This work presents a **custom Unity 6 plugin** that implements **OAuth**, **PubSub**, and **Helix** with production-grade safeguards (token sanitization, PING/PONG and auto re-subscription, idempotent reward operations, and an inlined, build-safe success page), alongside **VotePlay**, a social quiz game that leverages this integration so viewers can participate via chat with minimal friction. We present a modular, event-driven architecture (GameManager, MenuManager, SpawnManager, SoundManager, ChatManager) and a "chat-first" gameplay loop, validated through end-to-end latency measurements and build-level stability tests. Results show **instant capture** of votes and redemptions, **resilience to transient network failures**, and readable UI under streaming constraints. The work also includes an **economic analysis** and outlines a dual commercialization path (plugin on the Unity Asset Store and game on Steam), supporting sustainability and technology reuse in future developments.

Keywords

Twitch; Unity; OAuth; Helix; social play; interactive streaming; quiz game; real-time integration.

1. Introducción

El **auge de las plataformas de *streaming* en vivo** ha transformado la forma en que consumimos y participamos en videojuegos. En particular, Twitch, lanzada en 2011 como una rama de *Justin.tv* enfocada a videojuegos, se ha convertido en el referente principal del *gaming* en streaming [1]. **Twitch combina la transmisión de video en directo con un chat e interacciones en tiempo real**, permitiendo que los espectadores interactúen entre sí y con el streamer durante la partida [2]. Esta convergencia de vídeo en vivo y participación vía chat ha dado lugar a comunidades virtuales dinámicas en torno a los streamers, **convirtiendo las retransmisiones en espacios sociales** donde los espectadores no solo consumen contenido, sino que también forman parte de la experiencia de juego [2]. La relevancia de la interacción en streaming radica en que trasciende el modelo tradicional de jugador activo/espectador pasivo. En Twitch, **la audiencia tiene voz**: pueden influir en las partidas mediante comentarios, votaciones, redenciones u otros comandos, lo que aporta un nivel de inmersión y *engagement* mayor que el de simplemente ver un video pregrabado [3]. De hecho, el formato de Twitch ha sido descrito como un punto intermedio entre jugar y ver televisión, **el espectador está más involucrado que en la TV tradicional**, aunque no alcance la interactividad plena de jugar directamente [3]. Esta interacción en tiempo real genera un sentido de comunidad y pertenencia; estudios muestran que Twitch actúa como un "*tercer lugar*" virtual donde se congregan comunidades informales de jugadores y fans [2].

En pocos años, Twitch ha alcanzado un impacto masivo en la industria. Para 2013 ya contaba con 45 millones de espectadores únicos al mes (más del doble que el año anterior) [4], y en 2014 fue **adquirida por Amazon por 970 millones de dólares** dada su enorme popularidad [15]. Actualmente, Twitch sostiene regularmente entre 2 y 3 millones de espectadores concurrentes, alcanzando picos superiores a los 5 millones durante eventos destacados (*Ilustración 1*) o más actuales como la última velada del año con más de 10 millones de personas. Por ejemplo, el 13 de julio de 2024, "La Velada del Año IV" organizada por Ibai Llanos en el estadio Santiago Bernabéu alcanzó un pico histórico de 3.8 millones de espectadores simultáneos en su propio canal de Twitch, estableciendo un récord mundial. Del mismo modo, el 2 de noviembre de 2024, la final del Campeonato Mundial de League of Legends (Worlds 2024), celebrada en el O2 Arena de Londres entre los equipos T1 y Bilibili Gaming, congregó a millones de espectadores globalmente a través de múltiples canales de Twitch. Debido a estos y otros eventos**, Twitch ha llegado a ser comparada con la "ESPN de los videojuegos"** por su profunda influencia en la cultura gamer [5]. Grandes eventos de esports, lanzamientos de juegos y comunidades de jugadores amateurs conviven en esta plataforma, que se ha convertido en una pieza central del ecosistema de los videojuegos. En este contexto, integrar a los espectadores dentro de la

experiencia del juego, permitiéndoles afectar el desarrollo de la partida mediante comandos en el chat, representa la próxima evolución en la interacción streamer-espectador. **Convertir al público en participante activo ofrece nuevas formas de entretenimiento y fidelización**, y abre oportunidades tanto de diseño de juegos innovadores como de modelos de negocio basados en la participación de la comunidad [6].

Ilustración 1: "Espectadores simultáneos en el último año". Elaboración propia sacada de TwitchTracker.

Este Trabajo de Fin de Grado se centra en una investigación previa orientada al diseño y desarrollo de un plugin escalable y eficaz para Unity, que permita integrar de forma nativa las funcionalidades interactivas de Twitch en un videojuego especialmente diseñado para incorporar las interacciones de la audiencia en el propio *gameplay* del *streamer*.

1.1. Contexto general y motivación

En la industria del videojuego actual, la relación entre creadores de contenido y jugadores ha cobrado una importancia sin precedentes. El fenómeno del streaming ha hecho que millones de personas dediquen casi tanto tiempo a ver videojuegos como a jugarlos: en promedio, los **gamers invierten 8,5 horas semanales en ver streams** en plataformas como Twitch o YouTube, superando incluso las horas que destinan a jugar ellos mismos [7]. Esta realidad indica que **existe un gran apetito por experiencias participativas** en torno al juego, donde el público no se limita a observar de forma pasiva. Sin embargo, en la mayoría de los títulos convencionales la participación de la audiencia se reduce a comentarios en el chat sin influencia directa en el juego. Identificamos aquí una **oportunidad clara: integrar al espectador como agente activo** dentro del videojuego, enriqueciendo la experiencia tanto para el streamer (jugador principal) como para la audiencia.

La motivación para desarrollar un videojuego interactivo en Twitch surge de varios factores. En primer lugar, aporta un valor añadido de entretenimiento: los espectadores sienten mayor emoción y compromiso cuando sus acciones colectivas afectan al juego en tiempo real [8]. Esto

puede traducirse en comunidades más fieles y sesiones de streaming más largas y atractivas. Un caso emblemático fue *Twitch Plays Pokémon (2014)*, un experimento social donde miles de usuarios introdujeron comandos vía chat para controlar conjuntamente un juego de Pokémon, logrando completar el juego de forma colaborativa en 16 días; este evento alcanzó tal popularidad que inauguró un nuevo género de "juegos participativos de audiencia" y demostró el **enorme potencial del público como participante activo** [9]. A raíz de este éxito, numerosos desarrolladores independientes comenzaron a diseñar mecánicas enfocadas en la interacción con espectadores, reconociendo que estas prácticas pueden tener un alto impacto en la visibilidad y éxito de un juego en el mercado [9]. De hecho, algunos estudios señalan que ciertos **desarrolladores ya consideran a los streamers y sus audiencias como un público objetivo en el diseño de sus juegos**, incluyendo características especiales para ellos [9]. Integrar comandos e interacciones de Twitch en un videojuego puede traducirse en más personas hablando del juego, más streams reproduciéndolo y, en última instancia, en un impulso para la difusión orgánica del título, a la vez en la que ayuda al crecimiento del propio streamer.

En segundo lugar, existe una demanda creciente de experiencias interactivas en las plataformas de streaming. Tanto streamers como espectadores buscan **formas de interacción más profundas** que simplemente el chat o las donaciones [8]. Un estudio en el campo de la HCI resalta que la tendencia del live streaming va "*desde el espectador pasivo hacia la participación activa*" y destaca el interés por dotar a la audiencia de mayor capacidad de influir en la experiencia de juego [8]. Sin embargo, lograr esto sin comprometer la jugabilidad es un desafío de diseño importante. Aquí es donde radica la motivación académica y técnica de este proyecto: **explorar cómo se pueden diseñar mecánicas divertidas y equilibradas** que incorporen las aportaciones de una multitud de usuarios en tiempo real. Abordar este problema implica considerar aspectos: de usabilidad (que participar sea sencillo para los espectadores), de equilibrio (que las intervenciones mejoren la experiencia en lugar de arruinarla) y de escalabilidad (que funcione tanto con 5 como con 5.000 espectadores interactuando a la vez en el stream).

Otro aspecto motivador es el potencial de *engagement* y monetización. Desde la perspectiva del streamer, un juego que invite a la participación activa de la audiencia puede incrementar el número de espectadores concurrentes y su involucración. Esto a su vez **puede traducirse en mayores ingresos por suscripciones, bits o donaciones**, pues los espectadores sienten que forman parte del espectáculo y están más dispuestos a contribuir. Para el desarrollador del videojuego, lograr que streamers populares adopten su juego (precisamente por sus cualidades interactivas) puede significar publicidad exponencial gratuita y una comunidad entusiasta que genera contenido alrededor del juego [10]. De hecho, la presencia en Twitch se ha vuelto tan importante que hoy se considera un factor clave en el ciclo de vida de muchos juegos: el

streaming puede prolongar la vigencia de un título antiguo al presentarlo a nuevas audiencias, o impulsar las ventas de un lanzamiento reciente gracias al boca a boca digital [11]. Un artículo de *Johnson y Woodcock* concluye que el live streaming es "una nueva fuerza mayor en la industria del videojuego, creando nuevos vínculos entre desarrolladores e *influencers* y cambiando nuestras expectativas sobre el diseño y disfrute de juegos" [8]. Este proyecto se alinea con esa visión, investigando cómo incorporar de forma efectiva a los influencers (en este caso, la propia audiencia) dentro del proceso lúdico.

Por último, la elección de Unity como herramienta de desarrollo forma parte de la motivación práctica del proyecto. **Unity es actualmente uno de los motores de juego más utilizados a nivel mundial**, tanto por estudios independientes como por grandes empresas, gracias a su accesibilidad, versatilidad y amplia comunidad de soporte [12]. Esta popularidad ha llevado a que incluso Twitch ofrezca plugins oficiales para Unity, reconociendo la demanda de integraciones sencillas entre juegos y la plataforma de streaming [13]. Usar Unity permite aprovechar un vasto ecosistema de librerías y recursos (por ejemplo, SDKs no oficiales como TwitchLib para C#) y facilita el desarrollo rápido de prototipos. Unity **proporciona el equilibrio ideal entre potencia y rapidez** de desarrollo para implementar un videojuego orientado a la interacción con Twitch, permitiendo centrar nuestros esfuerzos en las mecánicas de juego y la lógica de integración, más que en problemas de bajo nivel. En la *Sección 2.7* se profundiza en la justificación de Unity frente a otras opciones, pero anticipamos aquí que su elección responde tanto a motivos pragmáticos como a la existencia de soporte específico para Twitch.

1.2. Planteamiento del problema

La expansión del streaming de videojuegos ha generado una nueva dinámica en la relación entre creadores de contenido y audiencia, en la que la participación del espectador se erige como un elemento diferenciador para crear experiencias de juego más inmersivas. Sin embargo, la mayoría de los videojuegos convencionales se han quedado rezagados en este aspecto, limitándose a transmitir el contenido de forma unidireccional, donde la **interacción del público se reduce a simples comentarios en el chat** sin tener un impacto real en la jugabilidad [1, 2]. Esta desconexión entre el potencial interactivo que ofrecen plataformas como Twitch y la experiencia de juego constituye el núcleo del problema que se intenta abordar.

A pesar de contar con diversas herramientas y APIs (como la *API Helix*, conexiones vía *IRC*, *EventSub*, *PubSub* o incluso plugins oficiales para Unity) diseñadas para integrar funcionalidades de Twitch en aplicaciones, **la implementación de estas soluciones en videojuegos resulta compleja**. Entre los principales obstáculos se encuentran la **latencia inherente** a las transmisiones en vivo, la **sobrecarga de mensajes** en canales con alta concurrencia y la

dificultad de traducir comandos de texto en acciones precisas y equilibradas dentro del juego [8, 9]. Estas dificultades técnicas hacen que muchos desarrolladores opten por soluciones *ad hoc* o limiten la participación de la audiencia a funciones meramente decorativas o informativas, desaprovechando así el potencial de convertir al espectador en un participante activo y comprometido.

Un ejemplo emblemático es el caso de *Twitch Plays Pokémon (2014)*, en el que miles de usuarios introdujeron **comandos mediante el chat para controlar conjuntamente un juego de Pokémon**. Aunque el experimento evidenció la viabilidad de una experiencia de "jugador colectivo", también puso de manifiesto que **la ausencia de mecanismos de control y de filtros para organizar las acciones del chat puede generar caos y frustración**, afectando negativamente la jugabilidad [9]. Además, la integración de sistemas de autenticación OAuth para garantizar la seguridad y personalización de la experiencia añade una capa adicional de complejidad, dificultando la adopción de estas soluciones en proyectos independientes o de recursos limitados [8, 10].

En definitiva, el problema central consiste en diseñar e implementar, en un videojuego desarrollado en Unity, un **sistema de integración de comandos** en tiempo real que permita a los espectadores de Twitch influir en la jugabilidad de manera efectiva y equilibrada. Dicho sistema debe ser capaz de:

- Gestionar la gran cantidad de interacciones provenientes del chat, agrupando y filtrando mensajes para evitar la saturación y garantizar una respuesta adecuada, incluso en escenarios de alta concurrencia.
- Minimizar los efectos adversos de la latencia, de modo que el retardo entre la acción del espectador y su reflejo en el juego sea lo suficientemente reducido para mantener la sensación de inmediatez.
- Traducir los comandos recibidos en acciones precisas dentro del juego, asegurando que la influencia del público no altere de forma desproporcionada la experiencia del jugador principal.
- Incorporar mecanismos de moderación y control que eviten abusos o interferencias que puedan deteriorar la experiencia tanto para el streamer como para la audiencia.

Abordar este problema requiere no solo una solución técnica robusta, sino también **comprender profundamente las dinámicas sociales y de interacción** que se generan en entornos de streaming. Es necesario un enfoque multidisciplinario que combine conocimientos de ingeniería de software, diseño de videojuegos y estudios sobre la interacción digital, con el fin de cerrar la brecha entre la potencialidad interactiva de plataformas como Twitch y la experiencia de juego

tradicional [10, 11]. Este **proyecto** se propone, en última instancia, **abrir paso a nuevas formas de entretenimiento colaborativo y participativo** que aprovechen al máximo las capacidades de interacción en vivo.

1.3. Organización del documento

El documento se estructura para guiar al lector desde el contexto y la motivación hasta la validación técnica y el cierre académico. El Capítulo 1 presenta la introducción general del trabajo, la motivación, el problema abordado y esta guía de lectura. El Capítulo 2 revisa el estado del arte y el marco tecnológico: APIs y eventos de Twitch (OAuth, Helix, PubSub), alternativas existentes en la Unity Asset Store y referencias de juegos "party-quiz" en streaming, justificando la elección de Unity 6 y de un conector propio. El Capítulo 3 delimita objetivos y alcance, tanto para el plugin de integración como para el videojuego VotePlay, estableciendo los criterios de aceptación y las restricciones del proyecto.

El Capítulo 4 describe la metodología aplicada en dos bloques: Kanban ágil propio para el desarrollo del plugin (tablero, ciclos y validaciones técnicas) y proceso iterativo-incremental por prototipos para el videojuego (diseño, desarrollo, validación, feedback y mejoras). El Capítulo 5 recoge el diseño y desarrollo: arquitectura modular en Unity (*GameManager*, *MenuManager*, *SpawnManager*, *SoundManager*), integración con Twitch (autenticación, gestión de recompensas y encuestas, chat), mecánicas, UI, audio y consideraciones de pruebas en build; se incluyen figuras de la arquitectura, flujos de estado y capturas de interfaz. El Capítulo 6 realiza el estudio económico, con estimación de costes internos, costes de publicación (Steam/Asset Store), comparativa con soluciones comerciales y, finalmente, un resumen económico que sintetiza umbrales de equilibrio y escenarios de explotación.

El Capítulo 7 presenta los resultados: evaluación del plugin (cobertura funcional, robustez, latencias y tolerancia a reconexiones) y del videojuego (pruebas de jugabilidad, desempeño y estabilidad en build). El Capítulo 8 expone las conclusiones, las limitaciones detectadas, las líneas de mejora y un cierre que enmarca las aportaciones del trabajo. El Capítulo 9 reúne la bibliografía. Por último, el Capítulo 10 (Anexos) incluye la documentación complementaria: propuesta y actas, índice de ilustraciones y tablas, glosario de términos, el GDD del videojuego (Sección Anexo I) y una guía práctica de uso y buenas prácticas del plugin (Sección Anexo II). Esta organización asegura trazabilidad entre objetivos, decisiones de diseño, implementación y evidencia empírica, facilitando la lectura técnica y la consulta puntual.

2. Antecedentes y Estado del Arte

En esta sección se presenta una revisión exhaustiva de la literatura y de las tecnologías actuales que han configurado el panorama de la integración de videojuegos con plataformas de streaming, especialmente Twitch. Se analizará la **evolución histórica de Twitch**, su **impacto en la industria** del gaming, y se **examinarán las herramientas** y APIs disponibles para lograr interacciones en tiempo real con los espectadores. Asimismo, se discutirán ejemplos destacados de juegos interactivos y se identificarán las limitaciones de soluciones previas. Este análisis es fundamental para establecer el marco teórico que sustentará tanto el diseño y desarrollo del plugin propuesto como la implementación de un videojuego que aplique dicho plugin, permitiendo detectar oportunidades y desafíos en la creación de experiencias de juego que integren de manera efectiva la participación activa de la audiencia.

2.1. Historia y evolución de Twitch

La plataforma Twitch tiene sus **orígenes en Justin.tv**, un sitio web de video en vivo fundado en 2007 que albergaba contenidos de todo tipo [21]. Rápidamente, las transmisiones de videojuegos se convirtieron en la categoría más popular de Justin.tv, lo que llevó a sus creadores *Emmett Shear y Justin Kan* a separar la sección de gaming bajo la marca Twitch en junio de 2011 [21]. Twitch.tv arrancó oficialmente en formato beta ese año, atrayendo desde sus inicios decenas de millones de visitantes únicos mensuales [21]. El crecimiento inicial fue explosivo, respaldado por rondas de financiación importantes en Silicon Valley (aportaciones de capital riesgo de firmas como *Bessemer Venture Partners* en 2012-2013) [21]. Para **mediados de 2013, Twitch ya dominaba el nicho de streaming** de videojuegos tras el cierre de competidores tempranos (como Own3d.tv), alcanzando alrededor de 45 millones de espectadores mensuales en 2013 [16]. Este número representaba más del doble de los ~20 millones de viewers que tenía en 2012 [16], evidencia de cómo el fenómeno del *live streaming* de juegos pasó de marginal a masivo en muy poco tiempo. De hecho, a inicios de 2014, **Twitch llegó a ser el cuarto mayor generador de tráfico en Internet** durante las horas pico en EE.UU., solo por detrás de Netflix, Google y Apple [21].

El año 2014 marcó un hito en la evolución de Twitch: en agosto, **Amazon anunció la adquisición de Twitch Interactive por 970 millones de dólares** en efectivo [15]. Esta compra, la mayor en la historia de Amazon hasta entonces, subrayó el valor estratégico de Twitch dentro del sector del entretenimiento digital. Bajo el paraguas de Amazon, Twitch continuó exponencialmente su expansión: integró sinergias con servicios de la compañía (como las suscripciones gratuitas mediante Amazon Prime, introducidas con "Twitch Prime" en 2016) [21], e invirtió en infraestructura para mejorar la calidad y estabilidad de las retransmisiones a nivel

global. Para 2015, Twitch ya superaba los 100 millones de espectadores únicos al mes, consolidando su posición dominante. A lo largo de la segunda mitad de la década de 2010, la plataforma diversificó sus contenidos (incluyendo música, arte, irl ("*in real life*"), etc., aunque los videojuegos siguen siendo el núcleo) y funciones sociales. Aparecieron los programas de afiliados y socios para remunerar a creadores de contenido, se **introdujeron sistemas de monetización** como los "*Bits*" (microtransacciones para apoyar a streamers) [21], y se mejoraron herramientas de comunidad (emotes personalizados, suscripciones de nivel, etc.). También se celebraron los primeros eventos anuales TwitchCon (desde 2015) que reúnen a fans y streamers, reflejando la cultura propia surgida en torno a la plataforma.

Es importante destacar hitos técnicos en la evolución de Twitch que inciden en la interacción. A finales de 2013, debido al incremento masivo de audiencia, Twitch implementó un nuevo sistema de video más eficiente que permitió soportar la demanda creciente, a costa de aumentar la latencia de las retransmisiones [21]. Este cambio generó críticas iniciales porque el retraso entre el streamer y los espectadores se amplió notablemente (alcanzando típicamente 10-15 segundos), lo cual dificultaba la inmediatez de la interacción streamer-chat [21]. Aunque el personal de Twitch señaló que ese retraso sería temporal y un "precio aceptable" por reducir cortes en la transmisión, el asunto puso de relieve la tensión entre calidad de streaming y capacidad de respuesta en tiempo real. En años posteriores, **Twitch trabajó en optimizaciones para reducir la latencia** (por ejemplo, habilitando un "*Low Latency Mode*" opcional que actualmente puede bajar el retraso a ~2-3 segundos en condiciones ideales). Este aspecto técnico es relevante para los videojuegos interactivos, ya que la latencia impone un límite a qué tan sincronizadas pueden estar las acciones de los espectadores con el juego en pantalla, un tema que exploraremos en la sección de limitaciones (*Sección 2.6*). Twitch pasó de ser una *startup* derivada de Justin.tv a **convertirse en la plataforma líder de streaming** de videojuegos en el mundo. Su historia está marcada por un rápido crecimiento comunitario, la adquisición por parte de Amazon y la constante introducción de características enfocadas en mejorar la experiencia tanto de espectadores como de creadores.

Ilustración 2: "El nacimiento de Twitch". Elaboración propia basada en la carta de despedida de justin.tv.

Actualmente, Twitch no solo es el principal escaparate para los *deportes electrónicos* y el contenido gamer, sino un fenómeno cultural que **ha dado pie a nuevas formas de entretenimiento interactivo**. Entender esta evolución ayuda a contextualizar la importancia de integrar videojuegos con Twitch: la plataforma ofrece hoy un entorno maduro y una audiencia habituada a participar, lo que sienta las bases para experiencias de juego innovadoras como la propuesta en este proyecto.

2.2. Impacto de Twitch en la industria del gaming

La influencia de Twitch en la industria de los videojuegos ha sido profunda y multifacética [11]. En pocos años, Twitch ha redefinido cómo se promocionan, distribuyen y consumen los juegos. Uno de sus impactos más evidentes es **la creación de la figura del *streamer* profesional** y la consolidación del *gaming* como espectáculo de masas. Antes del streaming, los jugadores consumían videojuegos principalmente de forma individual; ahora, millones siguen a sus *streamers* favoritos regularmente, generando una economía nueva alrededor de este entretenimiento [11]. Twitch **ha posibilitado la aparición de nuevos puestos de trabajo y fuentes de ingreso en el sector**: desde streamers que ganan la vida retransmitiendo, hasta toda una cadena de valor que incluye managers, moderadores, patrocinadores, desarrolladores de extensiones, artistas, etc. [11].

Esta "*economía del streaming*" se entrelaza con la industria del videojuego; por ejemplo, **las compañías invierten en campañas con streamers** famosos para dar a conocer lanzamientos, conscientes de que un juego popular en Twitch puede traducirse en ventas significativas [8, 10]. Otro impacto clave es el prolongamiento de la vida útil de los juegos y la revitalización de títulos antiguos. Gracias a Twitch, juegos independientes de nicho o lanzamientos pasados pueden ganar visibilidad mucho después de su estreno, si algún *influencer* los juega en directo y capta la atención del público [11]. Casos como *Among Us* (un juego de 2018 que se volvió fenómeno mundial en 2020 tras despegar en Twitch motivado por el aislamiento covid) ilustran cómo la plataforma puede resucitar un juego y convertirlo en éxito de masas. De igual modo, Twitch ofrece a las comunidades hardcore un lugar para seguir explorando juegos veteranos (por ejemplo, *speedrunners* transmitiendo juegos clásicos), manteniendo su vigencia. Johnson et al. señalan que el streaming está "añadiendo visibilidad y prolongando la vida de juegos independientes, de nicho y retro", creando un ciclo donde los contenidos generados por usuarios alimentan el interés en esos títulos [8]. En consecuencia, los desarrolladores hoy prestan atención no solo a las ventas iniciales, sino también a la presencia de sus juegos en plataformas de streaming para medir su éxito sostenido, por lo que el diseño del videojuego se debe adaptar para dar paso a estas plataformas.

Twitch también **ha impactado la forma de diseñar y pensar los videojuegos**. Por un lado, la cultura del streaming ha fomentado que más desarrolladores permitan o incorporen modos espectador, repeticiones y funcionalidades para compartir contenido, entendiendo que un juego "visible" es más valioso comercialmente. Por otro lado, ha surgido el concepto de "juegos para ver": títulos cuya propuesta resulta tan entretenida de observar como de jugar. Algunos géneros, como los *juegos multijugador competitivos* son los más visualizados (*Ilustración 3*) o los juegos de fiesta estilo *Jackbox*, se han popularizado enormemente apoyados en Twitch, donde la audiencia disfruta casi tanto como los participantes [9].

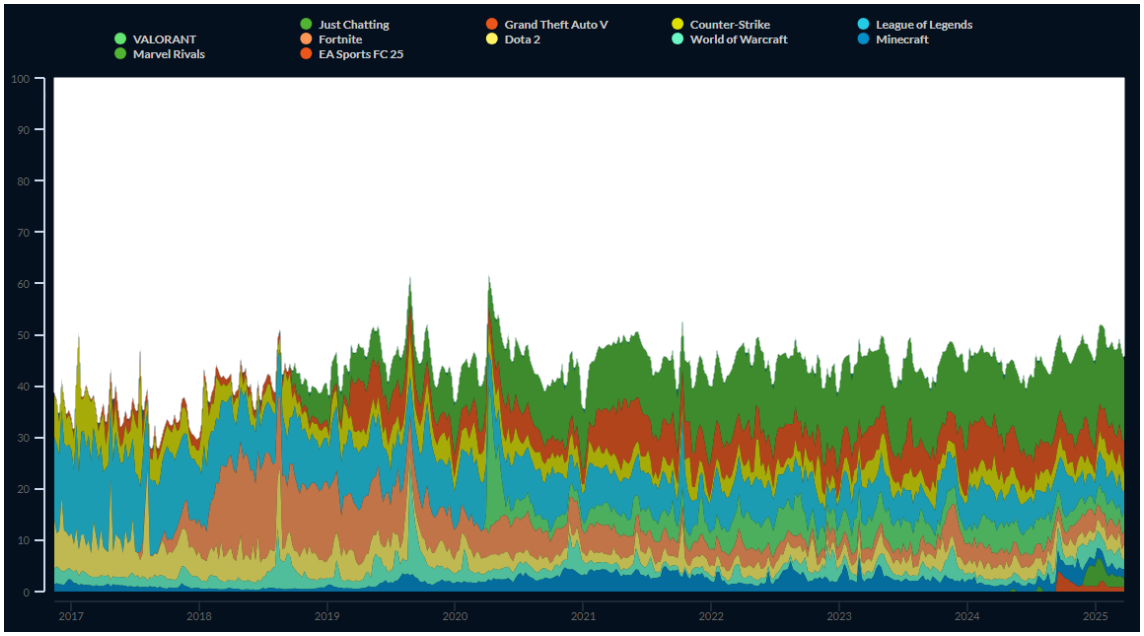


Ilustración 3: "Juegos más vistos en Twitch". Modificada de TwitchTracker.

Este fenómeno ha llevado a reflexiones académicas sobre la "jugabilidad para el espectador" ("spectator experience") que antes apenas se consideraba fuera de entornos de *esports* profesionales [9]. Incluso antes de Twitch, investigadores como Cheung y Huang estudiaron el comportamiento de espectadores de partidas de *StarCraft*, subrayando que la diversión de observar juegos podía analizarse sistemáticamente [4]. Twitch ha magnificado esta dimensión, convirtiendo el diseño centrado en la audiencia en un campo de interés. Actualmente, la *HCI* y el diseño de juegos exploran cómo crear experiencias interactivas híbridas, donde jugar y mirar confluyen [8].

Este proyecto se inscribe también precisamente en esa tendencia: diseñar un juego pensando tanto en el jugador principal como en cientos de "cojugadores" remotos que interactúan vía streaming aportando un plugin en el motor gráfico Unity 6. Otro ámbito de impacto es el marketing y la toma de decisiones de los consumidores. Twitch se ha convertido en una fuente

primaria de información para muchos jugadores a la hora de decidir qué comprar o a qué jugar. Ver a un streamer probar un juego proporciona una impresión auténtica de la jugabilidad y calidad del título. Así, Twitch **actúa como un escaparate en tiempo real**; en 2017, *Nintendo* atribuía parte del éxito de Switch y juegos como *Zelda: Breath of the Wild* al eco generado en streams y videos de usuarios [3]. En general, la influencia de los *streamers* como prescriptores ha obligado a las empresas a replantear estrategias: ahora es común enviar copias anticipadas a creadores de contenido, organizar eventos de *streaming* patrocinados o incluso desarrollar juegos "*streamer-friendly*" (con opciones para interactuar con la audiencia, o al menos sin contenido que pueda generar baneos por copyright, etc.) [8].

Un ejemplo notable fue el fallido juego *Breakaway* (Amazon Game Studios, 2017) que desde su concepción integraba funcionalidades de Twitch (como permitir a la audiencia apostar en partidas) [9], si bien el juego no se lanzó comercialmente, mostró cómo los desarrolladores están experimentando con ideas para aprovechar la plataforma. Finalmente, Twitch ha contribuido a **democratizar la creación de contenido en videojuegos**. Ha derribado barreras de entrada para que cualquier jugador con PC o consola pueda emitir sus partidas al mundo, lo cual ha diversificado la comunidad *gamer* y ampliado su alcance. También ha generado un espacio educativo e informativo: muchos espectadores afirman que aprenden estrategias, técnicas e incluso aspectos de desarrollo observando *streams* [11]. Streamers profesionales a menudo explican mecánicas de juego, y algunos desarrolladores transmiten ellos mismos mientras crean juegos ("*dev streams*"), acercando al público al proceso de desarrollo. Esto último ofrece una vía no tradicional de adquirir conocimientos de diseño y programación de juegos, complementando o incluso desafiando los canales formales de formación [10].

En suma, el **impacto de Twitch en el ecosistema** del gaming se manifiesta en: (a) una nueva economía y profesión alrededor del streaming, (b) cambios en cómo se mercadean y mantienen los juegos, (c) influencia en el diseño centrado en la audiencia y (d) un rol sociocultural que convierte el juego en un acto tanto de jugar como de exhibir/compartir. Este contexto justifica plenamente investigar cómo los videojuegos pueden integrarse más profundamente con Twitch, ya que aprovechar la plataforma de forma nativa podría ser el próximo salto evolutivo en la creación de experiencias lúdicas. De hecho, recientes trabajos académicos describen el *streaming interactivo* como un "*terreno fértil para la innovación e ideación*" en juegos, que está fomentando la independencia de creadores y ampliando la calidad y cantidad de producciones independientes [11]. En palabras de Shahghasemi y Mousavi, Twitch se posiciona como un actor pivote en la industria digital del videojuego, facilitando nuevas oportunidades creativas y reformulando las relaciones entre jugadores, desarrolladores y espectadores [11]. Entender este impacto holístico

orienta las decisiones de diseño de nuestro proyecto, buscando alinearse con las tendencias que Twitch ha potenciado.

2.3. Herramientas existentes para la integración de videojuegos con Twitch

Integrar un videojuego con Twitch (es decir, permitir que eventos de la plataforma como mensajes de chat, votos, donaciones, etc. interactúen con la lógica del juego) es un desafío multidisciplinar que combina desarrollo de APIs, networking y diseño de juegos. Afortunadamente, en los últimos años han surgido diversas herramientas y APIs que facilitan este proceso, tanto proporcionadas por la propia Twitch como por la comunidad de desarrolladores. A continuación, se describen las principales vías y recursos disponibles para lograr dicha integración:

- **API oficial de Twitch (Helix):** Twitch ofrece una API REST full dúplex (conocida como API Helix, sucesora de la API v5 Kraken) que permite a aplicaciones externas obtener información y realizar acciones en Twitch. Mediante llamadas a esta API es posible, por ejemplo, obtener la lista de espectadores, iniciar encuestas o predicciones en un canal, consultar quién se ha suscrito, etc. Muchas de estas acciones requieren autenticación y permisos específicos (mediante *scopes OAuth*, como se detalla en la *Sección 2.8*). La API Helix es robusta, pero inicialmente fue pensada para desarrolladores web y gestión de datos, por lo que para integraciones en juegos en tiempo real suele complementarse con mecanismos de suscripción a eventos debido a su alta latencia [19].
- **Twitch Chat e IRC:** La forma más directa de captar comandos de los espectadores es leer el chat de Twitch. Twitch históricamente provee un servidor *IRC* (Internet Relay Chat) adaptado, al cual un desarrollador puede conectar un *bot* o aplicación usando las credenciales de una cuenta de Twitch [20]. Mediante este método, el juego puede "unirse" al chat del *streamer* como si fuera un usuario más y recibir en vivo cada mensaje que los espectadores envían [20]. La interfaz IRC de Twitch utiliza el estándar IRCv3 con extensiones para incluir metadata (por ejemplo, distinguir suscriptores, moderadores, etc.), y cualquier lenguaje que soporte sockets puede manejarlo. Este fue durante años el método por excelencia para interacciones en tiempo real, empleado en proyectos pioneros (así funcionaba *Twitch Plays Pokémon*: un script leía cada comando del chat y lo traducía a pulsaciones en el emulador del juego [9]). La ventaja del enfoque IRC es su inmediatez (mensajes en pocos cientos de milisegundos) y simplicidad conceptual; la desventaja es que para implementarlo correctamente el desarrollador debe gestionar aspectos como la *ratelimit* (Twitch impone límites de velocidad de mensajes por bot), reconexiones y el procesamiento eficiente de potencialmente miles de mensajes por minuto.

- **EventSub (Webhooks):** Reconociendo la necesidad de recibir eventos de forma reactiva, Twitch lanzó un servicio denominado *EventSub*, basado en *Webhooks*, que permite suscribirse a distintos eventos de un canal (por ejemplo: "un usuario ha enviado un mensaje en el chat", "alguien se ha suscrito al canal", "se redimió una recompensa de puntos de canal", etc.) [12]. Cuando ocurre el evento, Twitch envía una notificación HTTP POST al servidor registrado por la aplicación del desarrollador. En el contexto de un videojuego, *EventSub* puede usarse si el juego cuenta con algún componente servidor o backend capaz de recibir esas notificaciones y luego comunicarlas al cliente (el juego en sí). Este modelo es útil para ciertos eventos que no son el chat pero interesan al juego, por ejemplo, saber que un espectador canjeó una recompensa tipo "lanzar obstáculo al jugador". Sin embargo, para escuchar el chat mismo, la solución clásica ha sido IRC (pues EventSub inicialmente no ofrecía evento de mensaje de chat, aunque ha ido ampliando tipos de eventos disponibles [12]). Twitch recomienda actualmente emplear EventSub para integraciones a gran escala en lugar de conectar directamente múltiples bots al IRC, ofreciendo incluso una capa de abstracción mediante su API para obtener mensajes de chat vía *webhook*. En cualquier caso, implementar EventSub requiere alojar un servicio web accesible desde Internet, lo que añade complejidad para desarrolladores de juegos que tradicionalmente trabajan solo del lado cliente.
- **PubSub (WebSockets):** Además de EventSub, Twitch tiene un servicio PubSub basado en WebSockets que permite a clientes conectarse y escuchar ciertos temas (*topics*) en tiempo real, como eventos de bits, redeems de puntos de canal, resultados de encuestas, etc. Este servicio ha sido usado principalmente por aplicaciones de terceros (por ejemplo, paneles de streamers tipo Streamlabs) para reaccionar instantáneamente a eventos. En una integración de videojuego, se podría usar un cliente WebSocket dentro del juego para suscribirse a topics del canal (requiere un OAuth token). Sin embargo, el alcance de topics es limitado (no cubre el chat regular), por lo que suele complementarse con IRC ya detallado anteriormente.
- **Extensiones de Twitch:** Otra vía de interacción son las Twitch Extensions, *aplicaciones web embebibles* en la página de Twitch (como capas sobre el video o paneles debajo) que pueden ofrecer minijuegos o controles interactivos para la audiencia. Las extensiones tienen acceso limitado al contexto del canal y pueden comunicarse con un servidor propio, permitiendo experiencias interactivas sincronizadas con el streaming. Algunos juegos han aprovechado extensiones oficiales: por ejemplo, *Dead Cells* ofrecía una extensión donde los espectadores votaban eventos que afectaban la partida en curso (modo "Twitch Integration") [9]. Las extensiones son potentes, pero su desarrollo es complejo (HTML/JS + backend) y están sujetas a revisión de Twitch. Para nuestro

propósito no son estrictamente necesarias, aunque representan una alternativa interesante para ampliar la interfaz de interacción más allá del chat de texto.

- **SDKs y bibliotecas de terceros:** La comunidad ha creado numerosas librerías para simplificar el uso de las APIs anteriores. En el ecosistema C#/.NET, destaca *TwitchLib*, un proyecto de *código abierto* muy maduro que unifica en una sola librería el acceso a la API de Twitch, la conexión de chat (IRC), PubSub y más [18]. TwitchLib cuenta con módulos especializados (TwitchLib.Client para chat, TwitchLib.API para REST, TwitchLib.PubSub, etc.) y es compatible con Unity, facilitando que un juego hecho en este motor pueda incorporar un "bot" de Twitch internamente [18]. Utilizar TwitchLib u otras librerías similares ahorra manejar directamente sockets o peticiones HTTP repetitivas, ya que proveen eventos ya procesados (ej.: *OnMessageReceived*, *OnChannelPointsRedeemed*, etc.). En otros lenguajes existen equivalentes: por ejemplo, tmi.js para JavaScript (muy usada para bots en Node.js) o pajlada/phantombot (un bot Java altamente personalizable). En general, estas herramientas abstraen la capa de comunicación con Twitch, de modo que el desarrollador de juegos pueda enfocarse en reaccionar a los eventos sin preocuparse por protocolos subyacentes.
- **Plugins oficiales de Twitch para motores:** En 2023, Twitch dio un paso significativo lanzando sus Game Engine Plugins, complementos oficiales para Unity, Unreal Engine y GameMaker [12]. Estos plugins fueron diseñados para simplificar al máximo la integración de funcionalidades de Twitch en juegos, proporcionando componentes prefabricados que manejan la autenticación, la conexión a eventos de Twitch y exponiendo directamente en el motor eventos listos para usar (por ejemplo, un evento *"OnChatMessage"* en Unity). El plugin de Unity, en particular, utiliza internamente el flujo de autenticación "Device Code" junto con suscripciones EventSub y llamadas a la API Helix [12], todo gestionado dentro del cliente (es decir, sin requerir servidor propio) [13]. Esto quiere decir que el juego Unity puede, por ejemplo, detectar cuándo alguien sigue el canal o cuándo alguien gasta puntos de canal, sin código de servidor, ya que el plugin mantiene una conexión activa con los servicios de Twitch por nosotros [13]. Además, estos plugins exponen funcionalidades comunes de Twitch como encuestas, predicciones o recompensas de puntos de canal de forma unificada [12]. En esencia, Twitch ha intentado brindar a los desarrolladores de juegos una herramienta *out-of-the-box* para crear juegos interactivos, consciente de que implementar desde cero estas integraciones era costoso. Actualmente, el plugin de Unity se encuentra disponible públicamente y soporta las principales plataformas de destino (Windows, con soporte en progreso para Mac/Linux) [13]. Su aparición es muy reciente (abierto al público en 2024 tras una beta cerrada), aunque con graves problemas en latencia y carencia de personalización y flexibilidad [13].

Un desarrollador que desee conectar su videojuego con Twitch puede optar por diversas soluciones: trabajar a bajo nivel usando el IRC y la API REST manualmente, apoyarse en librerías de terceros maduras como TwitchLib, o utilizar los plugins oficiales disponibles para Unity/Unreal. Sin embargo, para lograr un prototipo completamente personalizable y adaptado a requisitos específicos, resulta ventajoso desarrollar una solución propia. En este proyecto **se ha optado por diseñar y construir un plugin propio para Unity**, lo que permite un control total sobre la integración de Twitch, optimizando aspectos como la latencia y la flexibilidad. Esta solución supera las limitaciones de las herramientas existentes y se adapta de manera precisa a los desafíos del proyecto, aprovechando la colaboración y las experiencias compartidas por la comunidad, como se observa en casos de éxito de juegos con integración en Twitch [9].

2.4. Evaluación de plataformas y APIs para interacción en streaming

Si bien este proyecto se enfoca en Twitch por ser la plataforma predominante, es útil situarlo en contexto comparando brevemente las características de Twitch con otras plataformas de streaming en cuanto a posibilidades de interacción en tiempo real de los espectadores.

Ilustración 4: "Plataformas de Stream". En 2020, Mixer cesó sus operaciones pese a su notable calidad técnica. Facebook Gaming, por su parte, no ha logrado posicionarse con la misma fuerza en el sector. En consecuencia, el panorama principal de la transmisión en vivo se concentra en Twitch vs YouTube.

- **YouTube Live:** YouTube cuenta también con streaming en vivo y un chat integrado, pero históricamente ha estado más orientado a la calidad de video y a la integración con su ecosistema *VOD* que a la interactividad en vivo. YouTube ofrece varias opciones de latencia para sus emisiones (normal, baja, ultra-baja), pudiendo lograr retrasos de ~2 segundos en "ultra-low latency" a costa de deshabilitar resoluciones altas [22]. Esto acerca técnicamente la experiencia a la inmediatez de Twitch. Sin embargo, YouTube no proporciona tantas herramientas nativas de participación como Twitch (carece de un equivalente directo a los puntos de canal, extensiones interactivas o una API de eventos tan abierta como EventSub). Su API de chat existe, pero es más limitada y con cuotas

restrictivas. Además, la cultura de YouTube Live en gaming es distinta: suele haber menos cultura de *crowdplay* y la audiencia está más dispersa. En síntesis, es posible integrar juegos con YouTube (por ejemplo, leyendo el chat mediante API Data v3), pero la escala de adopción de tales integraciones ha sido menor, como por ejemplo mediante encuestas o redenciones. Twitch sigue siendo preferido cuando la prioridad es la participación activa de espectadores, debido a su base de usuarios más inclinada a interactuar en vivo.

- **Facebook Gaming:** La plataforma de streaming de Facebook (antes Mixer fue adquirida y cerrada, muchos streamers migraron a Facebook Gaming) tiene también chat en vivo y algunas mecánicas de interacción (encuestas, reacciones). Pero su ecosistema es cerrado; no ofrece APIs públicas tan completas ni una comunidad de desarrolladores activa creando integraciones con juegos. Hasta la fecha no se conocen juegos diseñados específicamente para Facebook Gaming con control de audiencia, en parte porque su foco ha sido más en crear un entorno integrado con la red social que en abrirse como plataforma de desarrollo.
- **Mixer (Beam):** Vale la pena mencionar a Mixer (plataforma de Microsoft que operó hasta 2020) como referencia, ya que fue la más innovadora en cuanto a interactividad en streaming. Mixer se destacaba por su protocolo FTL *(Faster Than Light)* que lograba latencias inferiores a 1 segundo, haciendo la interacción prácticamente instantánea [17]. Además, Mixer introdujo características nativas para que los espectadores pudieran influir en el juego: por ejemplo, Integración Direct, los viewers podían pulsar botones en la interfaz de Mixer para enviar comandos al juego, activar sonidos, votar por decisiones, etc. Esta plataforma incluso recompensaba a la audiencia con puntos ("*sparks*") por interactuar, que luego podían gastar en más acciones [17]. Microsoft también facilitó un SDK para que los desarrolladores integraran estas interacciones en sus juegos de Xbox/PC. Títulos como *Minecraft* o *Sea of Thieves* experimentaron con estas funciones en Mixer. Aunque Mixer fue discontinuada (sus innovaciones no lograron traducirse en cuota de mercado frente a Twitch), dejó un legado de ideas sobre cómo puede ser una experiencia de streaming altamente interactiva. Hoy, Twitch ha adoptado algunas de esas lecciones: por ejemplo, Twitch introdujo su modo de baja latencia en 2018 para competir con Mixer, reduciendo significativamente el retraso en las transmisiones. Aun así, Twitch no alcanza la casi instantaneidad que tenía Mixer, pero ofrece un equilibrio entre latencia baja (~2-5s) y estabilidad.
- **Comparativa de APIs:** En cuanto a las APIs, Twitch es con diferencia la plataforma con ecosistema de desarrollo más rico. La existencia de múltiples vías (REST, WebSocket, Webhook, IRC) y la documentación extensa han propiciado gran adopción en proyectos.

YouTube Live API permite obtener mensajes de chat pero impone cuotas (después de cierto número de mensajes requiere enfriar la frecuencia). Además, Twitch cuenta con muchas entidades de evento (suscripciones, *cheers*, *raids*, etc.) que su API expone, enriqueciendo las posibles interacciones. Por ejemplo, un juego en Twitch puede escuchar a cuando alguien se suscribe y usar eso como desencadenante de un evento especial en la partida, algo que en YouTube o Facebook no es tan directo de implementar por falta de endpoints o webhooks equivalentes. Por otro lado, Twitch tiene ciertas limitaciones: su API REST tiene rate limits de 800 llamadas por minuto por usuario/app, y hay que diseñar la integración teniendo en cuenta restricciones (como la cantidad de mensajes de chat que un bot puede enviar, para no ser marcado como *spam*). Estas limitaciones aplican transversalmente, pero Twitch ofrece alternativas como sus webhooks para evitar la necesidad de polling.

Twitch se mantiene como la plataforma idónea para explorar la interacción en streaming por su combinación de audiencia, cultura participativa y soporte técnico para desarrolladores. Mixer demostró hasta dónde se puede llegar en interactividad, pero ya no está disponible; YouTube y otras se quedan atrás en apertura o enfoque comunitario para este fin. Para un desarrollador que desea integrar comandos e interacciones de espectadores en un juego, Twitch brinda el mejor punto de partida actualmente.

2.5. Casos de éxito en videojuegos interactivos en Twitch

El concepto de videojuegos que involucren al público de Twitch no es meramente teórico; en la última década han surgido ejemplos emblemáticos que han sentado precedente y demostrado las posibilidades (y desafíos) de este nuevo paradigma. A continuación, revisamos algunos casos destacados de juegos o experiencias interactivas en Twitch, y qué se ha aprendido de ellos:

- ***Twitch Plays Pokémon (2014)* - "El chat juega Pokémon".** Iniciado anónimamente en febrero de 2014, fue un experimento pionero: se transmitió la versión *Pokémon Red de Game Boy* y se habilitó un sistema para que los comandos escritos por los espectadores en el chat controlasen directamente el juego. En su pico de popularidad, más de 100.000 personas enviaban comandos simultáneamente, produciendo caos y humor a partes iguales. Contra pronóstico, la "mente colmena" de jugadores logró vencer el juego tras más de 16 días. Twitch Plays Pokémon (TPP) se convirtió en fenómeno cultural y demostró el potencial del *crowdplaying*. En términos técnicos, TPP usó un script conectándose al IRC de Twitch para leer los mensajes y emular las pulsaciones de botones correspondientes. Uno de los hallazgos de TPP fue la necesidad de introducir mecánicas para gestionar la sobredemanda de comandos: los creadores implementaron

modos de juego ("*Anarchy*" vs "*Democracy*") para alternar entre tomar cada input individual o hacer votaciones periódicas sobre la próxima acción, intentando así mitigar el carácter inmanejable cuando el chat es extremadamente activo [9]. Este experimento generó varios estudios académicos sobre dinámicas sociales y de diseño emergentes; por ejemplo, Chen analizó matemáticamente el comportamiento del chat de TPP como un proceso de Márkov puro, mostrando cómo ciertos obstáculos del juego (como navegar por un laberinto) se volvían exponencialmente difíciles bajo el influjo masivo de comandos contradictorios [26]. Otro análisis por Margel concluyó que, pese al modo de control poco convencional (compartir un solo avatar entre miles), muchas dinámicas sociales observadas eran similares a las de juegos multijugador masivos tradicionales [27].

Ilustración 5: "Twitch Plays Pokémon". Captura del propio juego mostrando los inputs en tiempo real tras un día y medio en ciudad celeste.

Twitch Plays Pokémon abrió el camino y evidenció tanto la viabilidad de juegos controlados por la multitud, como la necesidad de diseñar reglas especiales para canalizar esa multitud (evitando *trolls*, entradas duplicadas, etc.). Tras TPP, se hicieron numerosas imitaciones ("Twitch Plays X") e incluso Twitch creó una categoría específica "Twitch Plays" para estos streams persistentes [9], pero pocos lograron el mismo éxito viral.

- ***Choice Chamber (2015)* - "El público diseña el nivel".** Fue el primer videojuego comercial concebido desde cero para Twitch. Desarrollado por *Studio Bean* (Michael Molinari) tras una exitosa campaña de Kickstarter cofinanciada por Twitch [23, 24], Choice Chamber es un juego de plataformas en 2D en el que un streamer juega y los

espectadores influyen en diversos aspectos: a través del chat, pueden votar qué arma obtiene el jugador, cómo se genera la siguiente habitación, qué enemigos aparecen, e incluso comandos en tiempo real para, por ejemplo, hacer saltar al personaje. La interfaz del juego está integrada con Twitch, mostrando encuestas en vivo cuyos resultados afectan la jugabilidad casi instantáneamente. Choice Chamber demostró una implementación efectiva de mecánicas de votación en tiempo real: cada cierto intervalo, el juego lee las respuestas más frecuentes en el chat (p. ej., "espada", "arco" o "pistola") [25] y ejecuta el cambio correspondiente. Al ser un juego diseñado pensando en la audiencia, equilibra cuidadosamente las decisiones: los espectadores podían tanto *ayudar como entorpecer al streamer*, manteniendo un tono participativo y a veces competitivo. Este juego ganó notoriedad y se presentó en eventos (Independent Games Festival 2015, etc.), sirviendo de inspiración para otros desarrollos independientes.

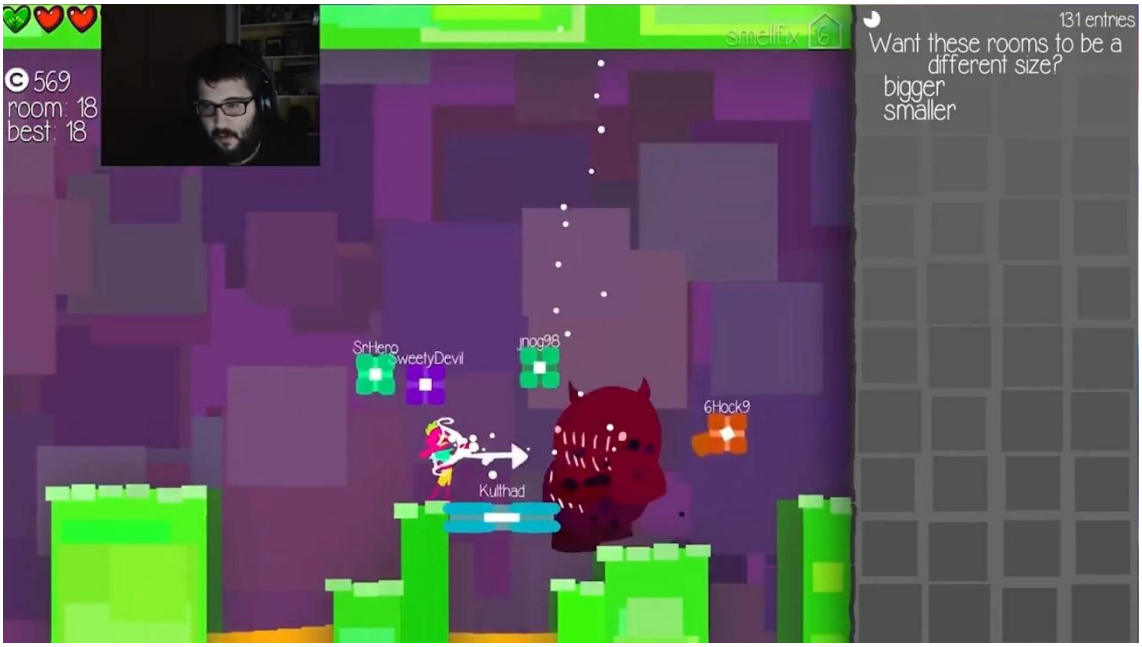


Ilustración 6: "Alexelcapo en Choice Chamber". Captura del gameplay de alexelcapo jugando a Choice Chamber en medio de una votación con 131 entradas para definir el tamaño de las rooms.

Un análisis de Libeau comparó Choice Chamber con otra integración (*Dead Cells*) y midió el grado de *engagement* de la audiencia en sus funciones participativas, encontrando que propuestas como las encuestas constantes logran altos niveles de participación sostenida, aunque pueden disminuir la atención del espectador en la acción en pantalla [9]. Choice Chamber es un referente porque probó que es posible hacer un juego *fun-to-play* y *fun-to-watch* simultáneamente, formalizando técnicas de diseño para ello (como introducir decisiones frecuentes, pero de impacto acotado, de forma que la audiencia sienta control sin desvirtuar el reto para el jugador).

- ***Dead Cells* - Modo Twitch (2017) - "Integración en un roguelite popular".** Dead Cells, un aclamado juego roguelite, género que combina generación procedural e inspiración de plataformas no lineal (*metroidvania*), desarrollado por el estudio independiente francés Motion Twin, añadió mediante una actualización un "*Streamer Mode*" donde los espectadores podían interactuar durante la partida del streamer. A diferencia de Choice Chamber, Dead Cells no fue concebido inicialmente para Twitch, sino que se adaptó después.

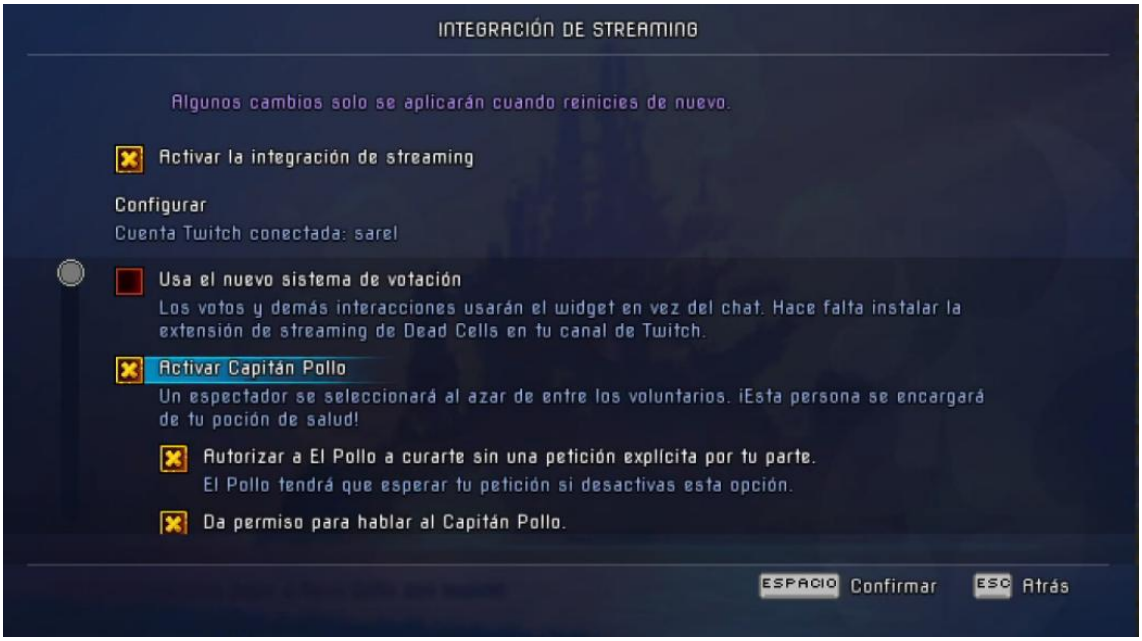


Ilustración 7: "Integración de Dead Cells en Twitch". Captura de pantalla de las opciones de integración con Twitch en Dead Cells.

En este modo, el público podía votar eventos periódicos (por ejemplo, aumentar la dificultad temporalmente, liberar un pack de enemigos extras, curar al jugador, etc.), así como escribir comandos para influir en el comportamiento de jefes finales o para desencadenar efectos humorísticos en el juego. Incluso se designaba aleatoriamente a un espectador como "el jefe" durante las peleas contra *bosses*, permitiéndole tomar ciertas decisiones del enemigo. Esta integración fue muy bien recibida por la comunidad y sirvió como ejemplo de juego AAA independiente incorporando Twitch. Los desarrolladores reportaron métricas positivas: más streamers jugando Dead Cells y mayor involucración del público en los directos, lo cual a su vez promocionaba el juego. Dead Cells aprovechó la integración sobre todo como elemento de marketing y fidelización post-lanzamiento. Técnicamente, implementó su propio sistema de conexión a Twitch (posiblemente mediante la API y sockets), ya que en 2017 aún no existían los plugins oficiales. La experiencia de Dead Cells muestra que es factible integrar Twitch en

juegos ya existentes y que los espectadores aprecian poder ser parte del juego de sus streamers favoritos, siempre que las mecánicas estén bien calibradas (en este caso, la influencia del chat era significativa pero no injusta; el streamer aún retenía habilidad para superar los retos añadidos). Estudios como el de Libeau entrevistaron a streamers sobre este tipo de integración y concluyeron que añade un factor de rejugabilidad y novedad muy valorada, aunque señalaron que para el streamer supone un esfuerzo adicional gestionar la interacción con la audiencia mientras juega, pero supone una posibilidad de crecimiento en la plataforma mayor [9].

Otros casos y experimentos: Además de los anteriores, ha habido numerosos intentos con distintos grados de éxito. Juegos como *Super Mario Odyssey y Dark Souls* han sido pasados por comunidades mediante *mods* al estilo Twitch Plays. Títulos independientes como *Ultimate Chicken Horse (2016)* incluyeron modos de votación de la audiencia para elegir niveles [9]. Algunos juegos de fiesta tipo *Use Your Words* o *Quiplash (Jackbox)* permiten a la audiencia unirse mediante códigos de sala (fuera de Twitch) y actuar como participantes adicionales o jueces, lo cual es una aproximación paralela a la integración directa con la plataforma. Incluso grandes producciones han incorporado "Streamer Mode" principalmente por temas de privacidad o anti-estrategia (p. ej., juegos competitivos como *Rust* o *Fortnite* ofrecen ocultar información sensible en pantalla para evitar el doloroso y reportado *stream sniping*) [9], si bien esto no implica participación de la audiencia sino protección del streamer. Un ejemplo curioso fue *Breakaway* (cancelado por incapacidad del equipo y dificultad en la adaptación) que planeaba una integración profunda con Twitch: los espectadores podrían apostar con una moneda virtual por equipos en partidas competitivas, y los streamers tenían herramientas para interactuar con la audiencia desde el juego mismo [9].

Ilustración 8: "Breakaway de Amazon Game Studies". Desarrollo final cancelado en 2018.

Aunque Breakaway no vio la luz, muchas de sus ideas resuenan en iniciativas actuales (por ejemplo, Twitch ha implementado extensiones de predicciones con puntos de canal que replican en parte esa idea de apostar puntos).

Los casos de éxito han dejado aprendizajes valiosos que tenemos que seguir tanto en el diseño del plugin como en el del videojuego: (1) La **simplicidad en la entrada es clave**, **los espectadores deben poder participar con acciones tan sencillas como escribir una palabra o pulsar un botón de voto**. (2) **Feedback inmediato**, la audiencia aprecia ver rápidamente el resultado de sus acciones en el juego (esto motiva a seguir participando); latencias altas o comandos cuya influencia no es visible tienden a desincentivar el *engagement*. (3) **Balance entre control y caos**, demasiada injerencia de la audiencia puede volver un juego injugable (como se vio en TPP inicialmente), por lo que muchos diseños exitosos limitan o modulan el poder del público: por ejemplo, *Choice Chamber* permitía al chat decidir elementos importantes, pero luego el jugador tenía agencia para usarlos con habilidad. (4) Escalabilidad, las mecánicas deben funcionar tanto con pocos espectadores (que no se sientan ignorados) como con muchos (que no saturen el sistema). Esto a veces implica adaptar la mecánica según la cantidad de participantes (p.ej., *Dead Cells* con menos viewers quizá reduce la dificultad de los eventos negativos para mantener el reto). (5) Diversión tanto para jugador como espectadores, idealmente, la interacción debe generar una dinámica entretenida para todos los involucrados. Si la audiencia disfruta, pero el streamer sufre (o viceversa), la experiencia global se resiente. Los mejores casos logran un círculo virtuoso donde la interacción produce momentos memorables, risas o tensión compartida entre ambos lados de la pantalla.

2.6. Limitaciones de las herramientas convencionales

A pesar de los avances en APIs y de los casos de éxito, integrar comandos o interacciones de usuario en tiempo real **conlleva desafíos y limitaciones inherentes que deben afrontarse**. Esta sección analiza las principales dificultades que presentan las herramientas convencionales (sin plugins especializados) y las barreras técnicas o de diseño que justifican la necesidad de desarrollar nuevas soluciones o tener consideraciones especiales en el proyecto.

- **Latencia y sincronización:** Como se mencionó, la latencia del *streaming* impone un límite físico a la interactividad. Incluso en las mejores condiciones de Twitch (latencia baja ~2s), hay una leve brecha temporal entre que el espectador ve una situación en el juego y reacciona, y cuando esa reacción le llega al *streamer*. Esto significa que los comandos de la audiencia siempre van "con retraso" respecto a la acción en vivo del jugador. Las herramientas convencionales (*IRC*, *EventSub*) transmiten los eventos del chat casi inmediatamente, pero no pueden superar el retraso inherente del video. ¿Por

qué es relevante? Porque en diseñar las mecánicas hay que evitar aquellas que requieran respuesta en fracción de segundo. Por ejemplo, no tendría sentido un juego de plataformas donde el chat diga "¡salta!" para evitar un obstáculo súbito en pantalla, ya que para cuando el streamer lea el comando y el juego lo procese, posiblemente el obstáculo ya pasó en el vídeo. Solución: enfocar los comandos de la audiencia hacia elementos que toleren ese retardo como votaciones sobre la siguiente sala, o spawn de enemigos que no sorprendan instantáneamente al jugador. Esta limitación de tiempo real es un factor a calibrar cuidadosamente. Otras plataformas como Mixer mitigaron esto con <1s de latencia, pero en Twitch debemos diseñar asumiendo ~2-5 segundos de retraso en promedio.

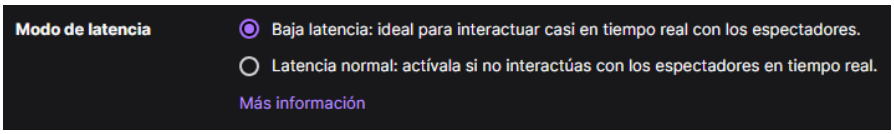


Ilustración 9: "Modo baja latencia en Twitch". Captura tomada en Configuración de Stream (Preferencias y clave de transmisión).

Desde el punto de vista técnico, Twitch no permite al desarrollador reducir la latencia más allá de activar el modo de baja latencia en el canal (lo cual será necesario activar durante este desarrollo). Por tanto, es una limitación intrínseca.

- **Escalabilidad del chat (spam):** Las herramientas como IRC no discriminan la importancia de los mensajes; si 1.000 personas envían comandos simultáneamente, la avalancha de datos puede superar la capacidad lógica de procesamiento del juego o nublar la utilidad de la interacción. Un problema de escala identificado en Twitch Plays Pokémon fue cómo manejar entradas masivas: cuando la audiencia crece, el ruido aumenta y se hace difícil extraer señal, de ahí la Anarquía vs Democracia previamente citada. Los métodos convencionales de leer chat requieren que el desarrollador imponga sus propias estrategias para filtrar o consolidar comandos e interacciones. Por ejemplo, conteo de votos: en lugar de procesar cada mensaje individual en una acción separada (lo cual sería inmanejable a gran escala), se debe agrupar mensajes iguales y tratarlos como una votación, tal como hizo *Choice Chamber*. Otra técnica es implementar "*cooldowns*" o límites: ignorar comandos repetidos muy seguidos, o limitar a un comando por usuario por ventana de tiempo, etc., para reducir spam. Las librerías como *TwitchLib* no hacen esto automáticamente; brindan los mensajes tal cual, así que el diseñador debe lidiar con la posible saturación. Además, Twitch impone su propio control: actualmente un canal de Twitch no permite enviar más de ~20 mensajes cada 30 segundos por usuario normal (y para bots verificados el límite es más alto, pero existe). En la práctica,

esto quiere decir que en casos extremos el chat podría perder mensajes (no todos llegan) si hay spam excesivo, lo que puede causar que algunos comandos de espectadores no se reciban. Es un detalle a reconocer: el sistema debe ser tolerante a pérdidas y equitativo. En resumen, la limitación de escala exige diseñar mecánicas robustas al ruido y posiblemente implementar algoritmos de agregación de comandos.

- **Complejidad de desarrollo (sin herramientas dedicadas):** Antes de la existencia de plugins oficiales, lograr la integración requería conocimientos en redes, autenticación, manejo de APIs, etc., que no todos los desarrolladores de juegos poseen. Un obstáculo tradicional era la autenticación *OAuth*: obtener y refrescar tokens para conectar un bot al chat o a la API puede ser engorroso de implementar correctamente (especialmente en aplicaciones de escritorio). Muchas herramientas convencionales ayudan (ej.: *TwitchLib* simplifica OAuth), pero aun así había que *hostear* una página de autenticación o usar flujos menos seguros (como el token de chat obtenido manualmente). La gestión de un backend también suponía una traba: si se querían usar *webhooks (EventSub)* o persistir datos de espectadores, se necesitaba desplegar servidores, lo cual para un desarrollador indie puede ser complejo. Precisamente por ello Twitch desarrolló sus plugins, para ahorrarle al game dev tener que "reinventar" toda esa infraestructura, [14]. En las herramientas convencionales, la ausencia de un soporte integrado en los motores implicaba mucho desarrollo a medida. Incluso algo tan sencillo como mostrar una ventana de login de Twitch dentro del juego requería ingenio. Estas complejidades han limitado que veamos más juegos interactivos; hasta ahora solo estudios con recursos o *community-driven* (como TPP) lo abordaron. Esta limitación está siendo superada con las nuevas herramientas, pero conviene mencionarla porque explica por qué integrar Twitch era (o es, si no se usan los nuevos plugins) costoso en tiempo de desarrollo. En nuestro caso, evaluaremos el uso del plugin Unity precisamente para mitigar esta dificultad, de forma que no debamos escribir todo el manejo de OAuth, sockets, etc.
- **Limitaciones de las APIs/SDK:** Aunque las herramientas existen, no son perfectas. Por ejemplo, TwitchLib y otras librerías de terceros pueden quedarse obsoletas si Twitch cambia su API, o tener bugs que el desarrollador de juego tendría que solucionar. Usar un SDK externo añade una dependencia más. Incluso los plugins oficiales, al ser recientes, tienen limitaciones: la primera versión del plugin Unity ha reportado problemas en plataformas Mac e iOS (soporte incompleto) [14], lo que indica que todavía no es totalmente multiplataforma. Además, los plugins cubren muchos eventos estándar (*polls*, *subs*, etc.) pero quizá no exponen funcionalidad personalizada: si se quisiera algo muy a medida fuera de su alcance, habría que extenderlo o combinarlo con llamadas manuales a la API. Por tanto, siempre existe la posibilidad de toparse con alguna restricción técnica.

Ejemplo: Twitch no permite (por razones obvias) obtener la lista de todos los viewers no logueados, solo de chatters activos, así que, si uno quisiera incluir también a espectadores pasivos en la interacción, la API no lo facilita. Otro ejemplo: las Rate Limits de la API Helix (*800 calls/min*) significan que no podemos, digamos, actualizar datos de 1000 espectadores individualmente por minuto sin pasarnos de ese límite. Estas restricciones obligan a optimizar el diseño (p.ej., no hacer consultas individuales para cada usuario constantemente, sino usar enfoques globales).

- **Diseño de juego equilibrado:** Más allá de lo técnico, hay una limitación conceptual: integrar comandos de la audiencia puede romper el balance o el propósito original de un juego si no se hace con cuidado. Muchos juegos tradicionales no están pensados para "*inputs impredecibles externos*". Un riesgo es que la participación de la audiencia derive en experiencias triviales o, por el contrario, imposibles para el jugador. Por ejemplo, si cada vez que el chat escribe "cura" el jugador recupera salud, un público benevolente podría quitarle todo el desafío al juego; inversamente, un público malévolo podría hacer inviable progresar si tiene demasiado poder negativo. Las herramientas convencionales no aportan soluciones mágicas a esto, es tarea del diseñador definir límites en las mecánicas. En los casos de estudio, hemos visto soluciones: *Dead Cells* solo permitía influir en ciertos momentos clave, *Choice Chamber* daba a la audiencia opciones, pero todas mantenían el juego en un estado viable. Esta limitación significa que, al usar las herramientas disponibles, igualmente se requiere criterio de diseño para no delegar en exceso o defecto la agencia a la audiencia.
- **Moderación y trolls:** Cualquier canal de Twitch lidia con moderación de chat (insultos, spam, etc.), pero cuando los mensajes controlan un juego, este asunto cobra otra dimensión. ¿Qué pasa si trolls envían comandos deliberados para arruinar la experiencia? En un streaming normal, el streamer o moderadores del canal pueden ignorarlos o banearlos; pero si el sistema del juego los procesa automáticamente, podrían causar impacto antes de ser filtrados. Una limitación de la integración automática es que podría ejecutar acciones provenientes de usuarios maliciosos. Por ello, deben incorporarse filtros de moderación similares a los del chat (quizá el juego no deba procesar comandos de usuarios baneados del chat, o de cuentas recién creadas, etc.). Las API convencionales no hacen esto por sí solas, aunque la interfaz IRC por ejemplo incluye indicaciones (metadatos) de si un usuario es moderador, suscriptor, etc., que pueden ayudar a ponderar la confiabilidad de inputs. Un enfoque es confiar en la moderación del canal: si un mensaje es eliminado por un moderador o por *AutoMod*, idealmente el juego no debería actuar sobre él. Actualmente, *EventSub* permite recibir eventos de moderación también (*bans*, *timeouts*), con los cuales se podría retroactivamente anular comandos de

ciertos usuarios. En cualquier caso, es una complicación más: una integración ingenua que tome cualquier input "tal cual" puede ser vulnerada para hacer publicidad, colar mensajes tóxicos in-game (por ejemplo, un espectador llamado "JugadorRacista123" podría aparecer en pantalla si el juego no filtra nombres), o incluso contenido que el propio algoritmo de Twitch sancionaría como la "*n word*" Así, la capa de moderación es un añadido necesario en la arquitectura y sobre todo en este plugin

Las herramientas convencionales nos llevan muy lejos (permiten la comunicación básica), pero no resuelven automáticamente problemas de latencia, escalabilidad, balance ni moderación. Estas limitaciones definen el espacio de diseño en el que nos movemos. **El proyecto debe diseñarse teniendo en mente soluciones o mitigaciones** para cada una: escoger mecánicas adecuadas a la latencia, implementar agregación de comandos para manejar el volumen, establecer límites en lo que la audiencia puede hacer para preservar el juego, y filtrar o moderar entradas según sea necesario. La existencia de plugins oficiales reduce la barrera técnica, pero no elimina las consideraciones de diseño. Adicionalmente, cabe notar que las primeras versiones de cualquier tecnología (como el plugin Unity) pueden conllevar bugs o restricciones de plataforma, por ejemplo, se reportaron fallos al compilar proyectos con el plugin en MacOS inicialmente [14], por lo que debemos analizar y evitar ese problema en nuestro plugin. Esto implica que debemos ser cautos y tener planes alternativos si cierta parte de la integración falla (p. ej., si el plugin no funcionara en una *build* web o móvil, podríamos recurrir a *TwitchLib* en su lugar para esa plataforma). Estas limitaciones, lejos de ser impedimentos, son puntos a abordar en la concepción del juego. La investigación académica sobre proyectos anteriores aconseja enfoques para superarlas: Hamilton et al. ya observaban que en comunidades de Twitch grandes, el streamer solo puede interactuar eficazmente con ~100-150 espectadores, más allá de lo cual la comunicación se diluye [1]. Por tanto, es sensato que nuestro sistema priorice *mecánicas asíncronas* (como votaciones globales) en lugar de intentar dialogar uno a uno con miles de usuarios. En conclusión, conocer estas limitaciones nos permite diseñar con realismo y establecer requisitos técnicos claros para el diseño e implementación.

2.7. Evaluación de alternativas y justificación de la elección de Unity

En la planificación de este proyecto se consideraron distintas alternativas tecnológicas para desarrollar el videojuego interactivo, evaluando sus ventajas y desventajas en el contexto de integración con Twitch. A continuación, se discuten brevemente dichas alternativas y se expone por qué Unity fue seleccionada como la opción más adecuada.

- **Motores de juego alternativos (Unreal Engine, GameMaker, motores custom):** Unreal Engine ofrece capacidades gráficas superiores y también dispone de un plugin

oficial de Twitch similar al de Unity [13]. Habría sido una opción viable, especialmente dado el soporte en *Blueprints* para integrar lógica sin código nativo. No obstante, *Unreal* suele tener un *overhead* de desarrollo mayor para proyectos pequeños; la rampa de aprendizaje y el tiempo de iteración son más lentos comparados con Unity para prototipos 2D/3D ligeros. Dado que nuestro objetivo es un prototipo funcional más que un despliegue comercial con gráficos de alta fidelidad se valoró que Unity permitiría avanzar más rápido en la implementación de la interacción. Por su parte, motores como *GameMaker Studio* también cuentan con un plugin de Twitch [13] y están orientados a 2D, pero podían quedarse cortos si necesitábamos mayor control o si queríamos eventualmente extender el juego a 3D o VR en el que el propio *streamer* no sabe absolutamente nada de la interacción de los espectadores en *gameplay*. Un motor propio o *framework* ligero (p. ej. *Monogame*, SDL) hubiera dado control absoluto, pero a costa de implementar desde cero muchas cosas (física, render, UI) y complicar la integración de Twitch (habría que usar directamente librerías como *TwitchLib*, lo cual Unity ya soporta fácilmente por ser C#). Por tanto, en términos de rapidez de desarrollo y flexibilidad, Unity sobresale: permite resultados visuales decentes, despliegue multiplataforma y abundantes recursos de comunidad para resolver problemas.

- **Lenguajes y entornos:** Otra alternativa pudo ser desarrollar el juego como una aplicación web (HTML5/JavaScript) que se ejecutase embebida en Twitch (por ejemplo, como una extensión o un juego web paralelo al *stream*). Esta vía facilitaría la comunicación con Twitch (usando JS directamente con APIs web) y la accesibilidad (cualquiera podría jugarlo en un navegador). Sin embargo, implementar un juego de calidad en JS puede ser arduo, y la sincronización con el video del *stream* sería complicada (además de que los espectadores tendrían que abrir otra ventana o interactuar fuera del reproductor de Twitch). Para lograr integración en el mismo reproductor, habría que desarrollar una *Twitch Extension* de juego, lo que está sujeto a limitaciones de seguridad y rendimiento (las extensiones no pueden acceder al sistema de archivos del streamer ni interferir con el juego que esté transmitiendo). Dado que queríamos un juego donde el streamer juega y la audiencia influye, la solución natural es un ejecutable que corre del lado del streamer, conectado a Twitch. Unity encaja perfectamente con este modelo, ya que genera un ejecutable que el streamer ejecutará localmente durante su stream. Otros lenguajes como Python pudieron considerarse para un prototipo rápido (hay bots de Twitch en Python), pero para crear un juego completo con gráficos y jugabilidad, Python no es ideal. C# con Unity ofrece tanto la conexión a Twitch (vía .NET) como el entorno de juego integrado.

- **Soporte y comunidad:** Unity tiene una comunidad gigantesca y múltiples proyectos de ejemplo relacionados con Twitch. Existen tutoriales, foros y plugins en la *Asset Store* para conectar Unity con Twitch (por ejemplo, integraciones de chat mostradas en juegos de realidad virtual, etc.) [18]. Esta base de conocimiento reduce riesgos: si surgen problemas, es más fácil encontrar soluciones o implementaciones de referencia. Alternativas como Unreal también cuentan con comunidad, pero con menor foco en 2D o en juegos indie de este tipo, y motores menos populares tendrían escasos recursos específicos sobre Twitch. Además, Unity ya ha sido utilizado en contextos similares: por ejemplo, hay casos documentados de desarrolladores que hicieron sorteos en Twitch o minijuegos con Unity leyendo el chat [24, 25], lo cual prueba su eficacia. Otra ventaja es que Unity permite desplegar rápidamente interfaces de usuario para visualizar información del chat o de votaciones dentro del juego, mediante su sistema de *Canvas*, sin tener que programar todo a bajo nivel.
- **Integración con Twitch - Soporte nativo:** La aparición del Plugin oficial de Twitch para Unity es un factor determinante. Al ser mantenido por Twitch, asegura compatibilidad hacia adelante y una integración relativamente sencilla (como se menciona en la *Sección 2.3*) [12]. Unity es, junto a *Unreal* y *Game Maker*, el único motor con plugin oficial hasta la fecha que está superando la fase beta [12]. Esto refleja que Twitch identificó a Unity como uno de los motores *"estrella"* donde quiere fomentar el desarrollo de juegos interactivos. De hecho, Unity es la plataforma elegida para pruebas piloto en la beta del plugin; ejemplos citados por Twitch, como *You Suck at Parking - Chaos Mode*, se implementaron en Unity [14].

Ilustración 10: "You Suck at Parking". Captura de pantalla de You Suck at Parking (Chaos Mode). Donde los espectadores "molestan" al streamer que debe salir del parking con su vehículo.

Esta cercanía implica que, usando Unity, estamos alineados con la dirección de las herramientas más nuevas de Twitch, y podemos beneficiarnos de soporte directo de la plataforma (por ejemplo, la documentación del plugin, actualizaciones, etc.). Alternativas como desarrollar un juego en C++ puro o en otro motor de videojuegos carecerían de este nivel de soporte nativo.

- **Rendimiento y alcance:** Unity es suficientemente eficiente para manejar la lógica adicional de Twitch (por ejemplo, procesar mensajes) sin comprometer el rendimiento del juego, especialmente en proyectos 2D o 3D sencillos. Además, Unity es multiplataforma, lo que permite escalar a realidades aumentadas, realidad virtual, consolas, etc. Por ejemplo, en un posible futuro el proyecto podría integrarse a un juego VR donde el streamer en VR es asistido (o perjudicado) por el chat; Unity facilitaría esa integración. En cambio, las opciones web sólo se ejecutan en navegador, y Unreal (pensado para alta fidelidad) podría imponer requerimientos de hardware mayores al streamer.
- **Mi propia experiencia:** Como desarrollador de videojuegos cuento con más experiencia en Unity por lo que la curva de aprendizaje no es un obstáculo, añadiendo la investigación que tuve que realizar sobre mi TFG de Informática sobre el nuevo motor Unity 6. Adoptar una tecnología nueva, ya sea Unreal u otro motor hubiera consumido tiempo valioso que lo necesito destinar a un generoso diseño.

2.8. Integración de autenticación OAuth y gestión de eventos en Twitch

La integración con Twitch requiere abordar aspectos de autenticación y manejo de eventos propios de la plataforma. Twitch, como servicio, protege la información de usuarios y las acciones sensibles mediante *OAuth 2.0*, un estándar de autorización que nuestras herramientas deben respetar [19]. Además, para recibir en tiempo real los eventos relevantes (mensajes de chat, suscripciones, etc.), debemos conectarnos a los servicios adecuados (IRC, *EventSub* o *PubSub*, según corresponda). En esta sección detallamos cómo se realiza la autenticación y la suscripción a eventos en el contexto de nuestro proyecto, sentando las bases técnicas para la implementación.

- **Registro de la aplicación y OAuth:** El primer paso para cualquier integración es registrar una aplicación en el *Twitch Developers Console*. Al hacerlo, obtenemos un *Client ID* y podemos establecer una Redirect URI (aunque para el flujo que usaremos, no es imprescindible una redirección web, podemos usar localhost). Con el Client ID, podremos identificar nuestra aplicación ante Twitch. Dado que nuestro juego correrá en la máquina del streamer en forma de aplicación de escritorio, el flujo de autenticación apropiado es el *Device Code Flow* (DCF) o el *Implicit Grant Flow* [19, 20]. El *Device Code Flow* es el que utiliza el plugin oficial: consiste en que el juego solicita un código de dispositivo a Twitch, muestra al usuario un mensaje "Visita twitch.tv/activate e ingresa este código: XXXX", el usuario lo hace en su navegador (ya logueado en Twitch) y concede permisos a la aplicación, y entonces Twitch devuelve al juego un token de usuario válido [14]. Este flujo es ideal cuando la aplicación no tiene interfaz web propia (como un juego Unity). Alternativamente, el flujo implícito abriría una ventana web para login y redireccionaría con el token, pero Unity no maneja bien esquemas de URL personalizados sin un servidor local, por lo que *DCF es más sencillo*.

Tabla 1: Comparativa de flujos OAuth 2.0.

Aspecto	**Device Code Flow**	**Implicit Grant Flow**
Entrega de token	El token de acceso se entrega al cliente a través de una solicitud segura	El token de acceso se entrega directamente al cliente en el fragmento de la URL
Nivel de seguridad	Alto (los tokens no se exponen en el navegador)	Medio (los tokens se exponen en el navegador)
Caso de uso	Aplicaciones web, servidores con backend	Aplicaciones de transferencia simple de datos en frontend

Uso	Aplicaciones complejas	Aplicaciones sencillas

En ambos casos, el resultado final es un *User Access Token OAuth* [19] con ciertos *scopes* (permisos) que especificamos. Para nuestro proyecto, los scopes necesarios incluyen: chat:read (leer el chat del canal), chat:edit (si quisiéramos enviar mensajes al chat como el juego, opcional), channel:manage:polls (si deseamos crear encuestas), channel:read:redemptions (para leer redenciones de puntos de canal), entre otros potenciales dependiendo de las mecánicas. Si usamos el plugin oficial, este simplifica todo el proceso: internamente solicita los scopes básicos y maneja la persistencia del token. Si implementamos con TwitchLib, tendríamos que usar su módulo de autenticación, que en algunos casos requiere abrir manualmente una URL para que el usuario otorgue permisos. En cualquier caso, una vez obtenido el token de acceso de usuario, el juego podrá hacer peticiones en nombre del streamer (usuario).

- **Seguridad de los tokens:** Es fundamental tratar el *Access Token* y el *Refresh Token* como credenciales secretas [19]. Unity deberá guardarlos en memoria (o en almacenamiento local encriptado si se quisiera persistir sesiones) y nunca exponerlos. El plugin oficial se encarga de esto, almacenando el token seguramente. Si lo hiciéramos manual, *TwitchLib* por ejemplo ofrece métodos para refrescar tokens cuando expiran (los tokens de usuario de Twitch suelen expirar en unos meses si no se revocan, pero mejor asegurar manejo de refresco). Durante nuestras pruebas, posiblemente usaremos un token generado para un usuario de prueba; en la entrega final, idealmente el streamer tendrá su propio token tras autorizar. Vale destacar que no necesitamos las credenciales (usuario/contraseña) del streamer, solo su autorización vía OAuth, lo que mejora la seguridad (no exponemos contraseñas en ningún momento).
- **Conexión al chat (IRC) y recepción de mensajes:** Con el token y el nombre de usuario del streamer, podremos conectar al Twitch IRC. Técnicamente esto implica abrir un socket a irc.chat.twitch.tv:6667 (o usar TLS en 6697) y enviar comandos IRC (PASS oauth:token, NICK canal, JOIN #canal…). Si usamos *TwitchLib*, bastará con configurar el ConnectionCredentials con el token y el nombre de usuario, e invocar a *client.Connect()*, seguido de *client.JoinChannel(canal)*. Una vez en el canal, Twitch comenzará a enviarnos cada mensaje del chat con su formato específico. *TwitchLib* emitirá eventos como *OnMessageReceived* con objetos que contienen el texto del mensaje, el usuario, sus roles, etc. [18]. Desde el punto de vista de diseño, suscribiremos una función para manejar esos eventos: esta función interpretará si el mensaje es un comando relevante (p. ej., empieza con "!" o coincide exactamente con alguna palabra clave de nuestras

mecánicas) y entonces actualizará el estado del juego en consecuencia (por ejemplo, incrementa un contador de votos). Es importante destacar que, para no sobrecargar el juego, el procesamiento debe ser ligero: potencialmente delegando acumulación de comandos a corrutinas o hilos separados si hay altísima frecuencia, pero Unity puede manejar varios cientos de mensajes por segundo en el hilo principal sin problemas notables, especialmente si solo contamos/votamos. Hay que mencionar que Twitch impone un límite de 100 mensajes IRC por cada 30 segundos por conexión para bots no verificados, lo cual podríamos alcanzar si intentáramos enviar un mensaje de confirmación por cada input recibido (por eso usualmente el juego no responde en chat a cada comando, sino que refleja los efectos en pantalla). Para lectura no hay un límite tan explícito, pero tasas altísimas pueden llevar a *throttling*. En general, con la escala de un stream promedio, esto no será problema.

- **Suscripción a eventos adicionales (EventSub/PubSub):** Además del chat, podemos querer reaccionar a otros eventos de Twitch. Dos especialmente interesantes: redención de puntos de *canal (Channel Point Redemptions)* y *cheers/bits*. Por ejemplo, podríamos diseñar que si alguien gasta 1000 *puntos de canal* en "Lanzar enemigo extra", el juego lo haga; o que si alguien se suscribe al canal, caiga un ítem de ayuda para el jugador, etc. Para lograr esto, tenemos que suscribirnos a esos eventos. Con *EventSub*, se requeriría un servidor web, lo cual complica las cosas para un juego local. Alternativa: usar *Twitch PubSub*. *TwitchLib* incluye un cliente PubSub que puede escuchar eventos como channel-points-channel-v1.<userID> y channel-subscribe-events-v1.<userID>. Habría que obtener el userID del streamer (lo cual se puede con la API Helix endpoint /users utilizando el token). Luego se conecta al *WebSocket* de *PubSub* y se envía un mensaje de LISTEN a esos tópicos junto con un OAuth token (generalmente requieren token de broadcaster con scope apropiado, como channel:read:redemptions). Una vez hecho esto, Twitch enviará mensajes JSON cuando ocurran redenciones o subs. TwitchLib simplifica esto también a eventos *OnChannelPointsRewardRedeemed*, etc. Si usamos el plugin oficial, es aún más sencillo: el plugin ya por defecto se suscribe a *Channel Point Rewards* personalizados únicos [12, 13] y a eventos de *follow/subscribe*, exponiéndolos en Unity. Por ejemplo, hay un componente *TwitchEventListener* que uno configura para escuchar *"OnChannelPointReward"* con cierto premio. Dado que el plugin fue diseñado para no requerir backend, internamente probablemente abre conexiones seguras a *PubSub* o utiliza la API con *polling* rápido (aunque es más probable lo primero). En todo caso, la gestión de eventos en nuestro juego se hará vía la infraestructura que elijamos: *TwitchLib* (manualmente controlar IRC + PubSub) o Plugin (que ya maneja DCF + *EventSub* bajo el capó).

- **Llamadas a la API Helix:** En algunas situaciones, podríamos necesitar invocar endpoints REST durante la partida. Por ejemplo, iniciar una encuesta (poll) en el chat para que los espectadores voten de forma más estructurada que solo escribiendo en texto. Twitch tiene endpoints para crear encuestas si la cuenta es afiliada/partner y el token tiene scope channel:manage:polls. Con TwitchLib podríamos llamar a *API.Helix.Polls.CreatePollAsync(...)* y luego escuchar resultados vía PubSub o comprobando periódicamente. El plugin Unity igualmente soporta encuestas de forma integrada. Otra posible llamada es para obtener información del canal o lista de espectadores, pero Twitch no da lista de viewers por API (sí de chatters por IRC). Quizá podríamos usar la API para validar si un usuario es seguidor del canal antes de dejarle participar en ciertas mecánicas, esto requeriría la scope user:read:follows o similares y hacer consultas. Son detalles de diseño; técnicamente, incorporar llamadas REST en Unity es trivial usando *UnityWebRequest* o la propia TwitchLib (que usa HttpClient). La limitación es la mencionada Rate Limit: max ~800 llamadas por minuto globales [19, 20]. Nuestro juego difícilmente hará tantas (eso sería más de 13 por segundo, muy por encima de lo necesario). Pero hay que asegurarse de no abusar (por ejemplo, no intentar obtener los *follow count* cada segundo, con una consulta por seg). Un diseño eficiente cachea la info o usa eventos para actualizaciones.

La integración *OAuth* y la gestión de eventos en Twitch se resumen en un sencillo flujo: al iniciar el juego en Unity, este solicita permiso al streamer mediante el *Implicit Grant Flow*. Una vez autorizado en el navegador, Unity recibe y almacena el token de acceso, cifra sus credenciales y abre conexiones seguras (IRC sobre SSL, WebSockets y HTTPS) para escuchar chat, points, encuestas, predicciones y recompensas en tiempo real. El plugin debe manejar internamente la renovación de tokens, la reconexión automática y la verificación de información.

3. Objetivos

El presente Trabajo de Fin de Grado tiene como meta no solo desarrollar un videojuego interactivo para streaming en Twitch, sino también establecer un marco metodológico y técnico que respalde la creación de nuevas formas de entretenimiento participativo en la era del live streaming. Para ello, se optó por diseñar y construir, desde cero, un plugin para Unity que permita la integración directa de las funcionalidades de Twitch, aprovechando las ventajas de este motor y las herramientas de comunicación de la plataforma [12, 19].

3.1. Objetivo principal

Diseñar y desarrollar un videojuego interactivo que incorpore la participación en tiempo real de la audiencia de Twitch mediante comandos, utilizando un plugin desarrollado de forma autónoma para Unity. Este videojuego servirá como caso de estudio para demostrar la viabilidad técnica y el potencial de las interacciones integradas, sentando bases para futuros desarrollos en el ámbito de los *"audience participation games"* [1, 7].

3.2. Objetivos específicos

Para alcanzar el objetivo general del proyecto, se han definido diversos objetivos específicos que abarcan desde el diseño de mecánicas interactivas y la implementación técnica del plugin en Unity, hasta la evaluación del prototipo y el análisis de los resultados obtenidos. Cada uno de estos objetivos contribuirá a garantizar una integración robusta y eficaz de las interacciones en tiempo real de la audiencia en el videojuego, aprovechando plenamente las capacidades de Twitch y Unity.

- *Diseño de mecánicas interactivas:* Crear sistemas de juego que reciban y procesen las entradas del público (a través del chat y otros eventos de Twitch) y las traduzcan en acciones o cambios dentro del juego. Estas mecánicas deben ser intuitivas, escalables y equilibradas, de modo que faciliten una participación activa sin comprometer la experiencia del jugador principal [2, 5].
- *Implementación técnica de la integración:* Desarrollar el plugin para Unity que permita la conexión con Twitch utilizando el Device Code Flow para la autenticación OAuth, y gestionar la recepción de mensajes del chat y otros eventos (por ejemplo, redenciones de puntos de canal). Se debe garantizar que la comunicación se realice de forma segura mediante canales oficiales (IRC sobre SSL, WebSockets y HTTPS para REST), minimizando la latencia y asegurando la robustez ante desconexiones o altos volúmenes de mensajes [12], [19].

- *Incorporación de incentivos y retroalimentación:* Diseñar mecanismos de gamificación que recompensen la participación de la audiencia. Entre estos, se contempla la implementación de contadores de contribución, feedback visual y sonoro inmediato, y la integración de eventos especiales (por ejemplo, notificaciones en pantalla cuando un espectador envía un comando relevante). Estos elementos tienen como fin mantener altos niveles de engagement y reforzar la interacción entre streamer y audiencia [8], [10].
- *Evaluación y validación del prototipo:* Realizar pruebas piloto en entornos controlados (simulando un stream con usuarios voluntarios) para recoger retroalimentación tanto del streamer como de la audiencia. Con base en estos ensayos, se realizarán ajustes iterativos en las mecánicas y la interfaz de usuario, evaluando aspectos como la latencia percibida, la estabilidad de la integración y la efectividad de la comunicación de los eventos en tiempo real [10, 11].
- *Documentación y análisis del desarrollo:* Elaborar una memoria detallada que recoja el proceso de investigación, el diseño de la arquitectura del plugin y del videojuego, las decisiones de desarrollo, y los resultados obtenidos durante la fase de pruebas. Esta documentación servirá para difundir el conocimiento adquirido y para ofrecer una base sólida para futuros desarrollos en el área de la integración interactiva en streaming [7, 11].

En conjunto, estos objetivos marcados buscan cubrir todos los aspectos esenciales del proyecto: desde la conceptualización de cómo interactuará el público en el juego, pasando por la construcción de la infraestructura técnica (mediante el desarrollo del plugin para Unity), hasta la validación final del prototipo. Al lograr estos objetivos, se demostrará la viabilidad de integrar funcionalidades interactivas en videojuegos a través de Twitch, aportando un caso de estudio práctico en el emergente campo de los juegos participativos.

4. Metodología

La metodología del proyecto se ha dividido en dos grandes bloques de trabajo: (1) el diseño y desarrollo del plugin de integración con Twitch y (2) el diseño y desarrollo del videojuego interactivo que hace uso de dicho plugin. En **cada bloque se ha aplicado el enfoque más adecuado a sus características**: durante la creación del plugin se siguió una metodología ágil adaptada (inspirada en marcos como Scrum/Kanban) para organizar el desarrollo modular en Unity 6 con C#, mientras que para el videojuego se empleó un proceso iterativo e incremental basado en prototipos con retroalimentación constante de pruebas de juego con el director del presente trabajo.

4.1. Metodología del plugin de integración

Para materializar la metodología Kanban, se diseñó e **implementó un gestor de tareas a medida**, desplegado en el subdominio propio *vid.mariogr.com*. Para desarrollar y alojar el tablero de tareas se recurrió a tecnologías de desarrollo web simples y a un servidor Node.js con conexión a una base de datos SQLite:

- *Node.js y Express:* Permiten gestionar las peticiones HTTP y servir tanto la lógica de negocio como los archivos estáticos (HTML, CSS y JavaScript).
- *SQLite:* Facilita el almacenamiento de las tareas, su estado y las horas registradas. Al ser una base de datos ligera, no requiere un servidor de base de datos adicional.
- *Socket.IO:* Habilita la comunicación en tiempo real entre el servidor y el navegador, de forma que cada cambio en el tablero se propaga instantáneamente a todos los usuarios conectados.
- *Plesk:* El proyecto se despliega en un hosting con panel de control Plesk, que posibilita la configuración de subdominios y el soporte para aplicaciones Node.js.

Se acordaron nueve semanas de alrededor de 16 horas de desarrollo. El tablero presenta las columnas principales que representan el flujo de trabajo típico:

- *Espera:* Se ubican las tareas recién creadas o pendientes de ser aprobadas por el estudiante y el director.
- *Activo:* Recoge las tareas aprobadas y preparadas para ser iniciadas cuando el desarrollador considere.
- *Desarrollo:* Contiene las actividades en fase de implementación o programación, tareas que no hayan superado el QA por parte del director serán devueltas a esta fase de nuevo.

- *Tech Review:* Se efectúa una revisión técnica y de código para garantizar la calidad de la implementación, será necesario analizar el comportamiento de las calls/min y latencia para situarnos en el marco correcto que admite Twitch.
- *QA:* Se realizan pruebas de calidad, corrección de errores y validaciones finales por parte del director.
- *Completo:* Indica que la tarea ha superado satisfactoriamente todas las revisiones y está implementada en su totalidad.

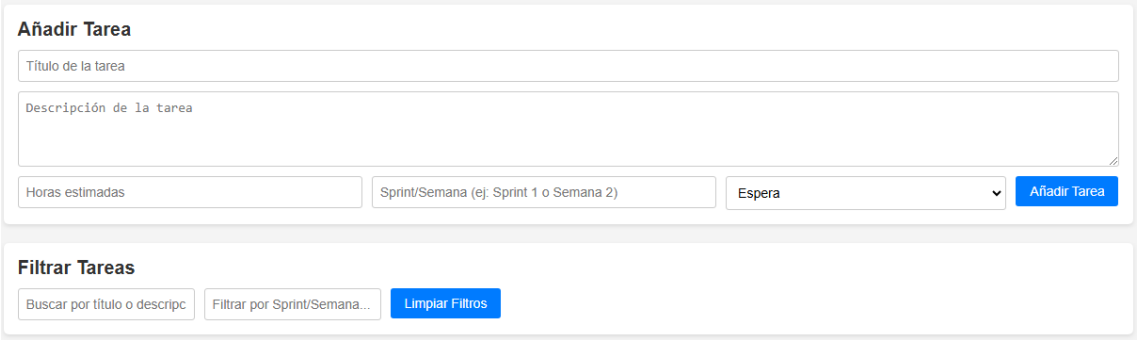


Ilustración 11: "Panel de adición de Tareas".

Tanto el estudiante como el director (Dan Tarodo Cortés) con sus **credenciales privadas** *(usuario: "d4n" – contraseña: "d4n")* pueden generar nuevas actividades. Por defecto, toda tarea se inicia en la columna "Espera" hasta que se discuta o valide su pertinencia. Cada tarea se describe con un título, una breve descripción y un número de horas estimadas. Adicionalmente, se registra la semana a la que corresponde la tarea, facilitando la planificación temporal.

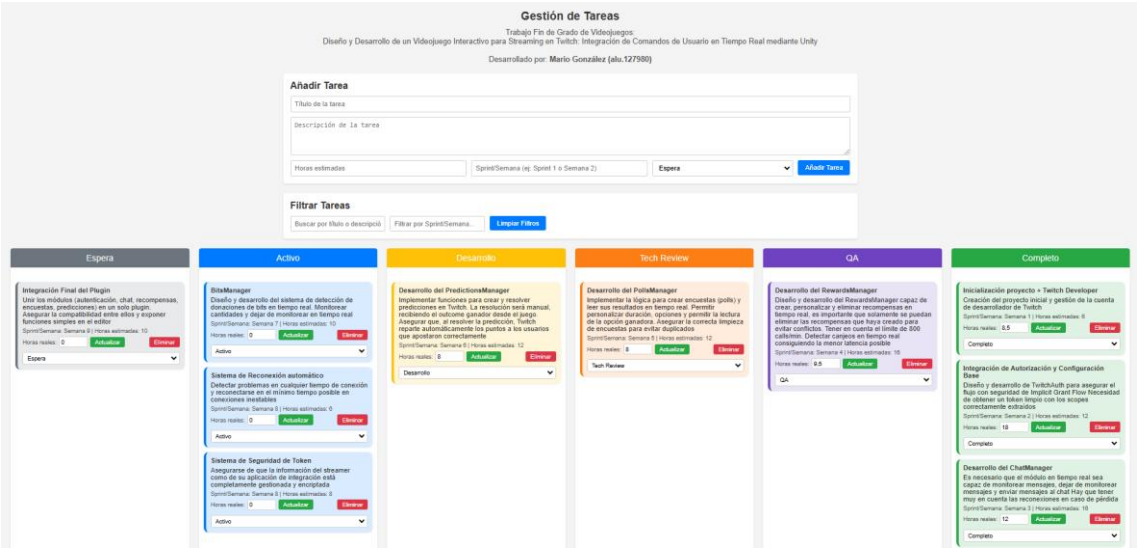


Ilustración 12: "Tablero de Tareas". Desarrollo propio.

El uso de este gestor de tareas en vid.mariogr.com reporta múltiples ventajas, ya **que proporciona una visión global del proyecto** al ofrecer un panorama claro de todas las tareas, su estado y las horas empleadas, facilitando la toma de decisiones respecto a prioridades y plazos. Además, permite una colaboración instantánea gracias a su actualización en tiempo real, posibilitando que tanto el director como el estudiante visualicen al instante cualquier modificación o comentario en reuniones, lo que reduce la necesidad de intercambiar correos o mensajes para confirmar el avance. Asimismo, **promueve la transparencia en la gestión** mediante la clasificación por columnas y el registro de horas reales, permitiendo un control riguroso del tiempo dedicado a cada actividad e identificando áreas en las que sea preciso mejorar la estimación o la asignación de recursos. Por último, su **flexibilidad y adaptabilidad** se evidencian en la capacidad de ajustar el sistema de manera sencilla ante la aparición de nuevas tareas o cambios en la prioridad de las existentes, permitiendo el movimiento de las tarjetas a las columnas correspondientes sin interrumpir el flujo de trabajo.

Es fundamental comprender que, para que una tarea pase del estado de QA a la categoría de Completo, se requiere una validación previa por parte del director en las reuniones programadas. En caso de detectarse una funcionalidad incompleta o deficiente, la tarea regresaría a la fase de Desarrollo para ser corregida. Por este motivo, resulta clave que la información se actualice en tiempo real, evitando así pérdidas de contexto y disminuyendo la necesidad de convocar reuniones adicionales que entorpezcan el flujo de trabajo.
La ilustración anterior (Ilustración 12) detalla el desglose de tareas con sus estimaciones y estado (si no visualiza correctamente la información de la ilustración acceda al panel directamente con los datos de acceso mencionados), mientras que la siguiente Ilustración 13 ofrece una visión global en formato Gantt para facilitar el seguimiento del progreso y la coordinación junto al director ligada a la carga de trabajo del único desarrollador incluyendo el espacio temporal para el diseño y desarrollo del videojuego aplicando el plugin de integración de Twitch.

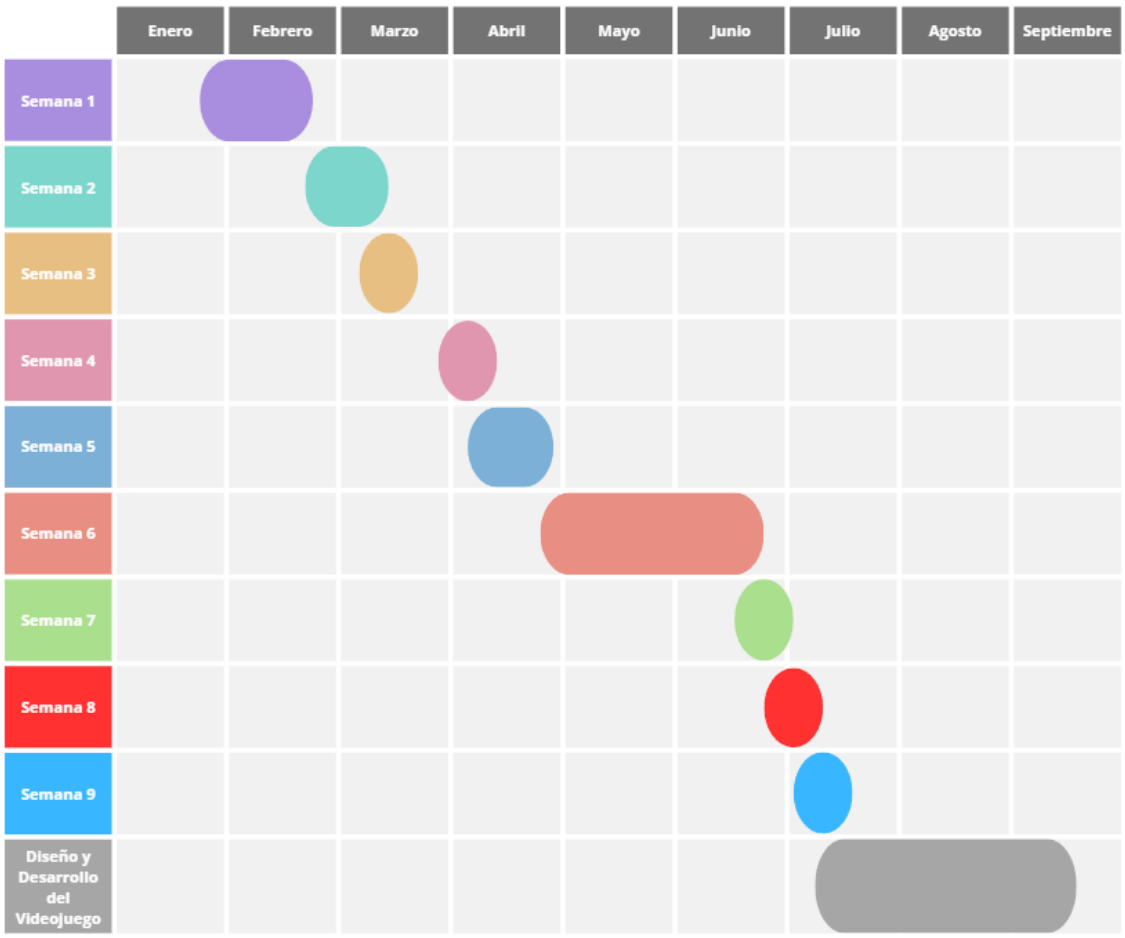


Ilustración 13: "Diagrama de Gantt". Diagrama para mostrar el desarrollo en el tiempo acorde a la carga de trabajo del desarrollador.

4.2. Metodología del diseño y desarrollo del videojuego interactivo

El diseño y desarrollo del videojuego interactivo se llevará a cabo mediante un **proceso iterativo e incremental basado en prototipos**. Este método ha sido elegido por su capacidad para ofrecer flexibilidad, permitiendo ajustes constantes y adaptación rápida a nuevas necesidades o recomendaciones que surjan durante el proyecto. A diferencia del modelo tradicional en cascada (waterfall), donde cada fase se completa de forma secuencial y rígida, esta metodología establece un ciclo continuo de diseño, desarrollo, validación, retroalimentación y mejora. Esto asegura que en cada iteración se produzcan avances significativos hacia un producto final altamente pulido y alineado estrechamente con las expectativas iniciales.

Ilustración 14: "Proceso iterativo e incremental basado en prototipos". Elaboración propia.

El proceso iterativo, cíclico y continuo, incluye las siguientes fases:

- *Diseño* (estimación de 40 horas): Se centrará en la conceptualización del videojuego mediante reuniones frecuentes con el director del proyecto. Se definirá claramente la estructura del juego, mecánicas principales, narrativa, personajes, escenarios, objetos interactivos e interacciones para los espectadores. Además, se creará un borrador del Game Design Document (GDD), que recopilará especificaciones técnicas y artísticas detalladas, complementado con wireframes y bocetos preliminares.
- *Desarrollo* (estimación de 40 horas): Durante esta etapa se construirán prototipos funcionales, garantizando desde las primeras fases que las mecánicas básicas sean correctas y viables técnicamente. El objetivo es descubrir y corregir potenciales dificultades técnicas de manera temprana.
- *Validación* (estimación de 15 horas): Consistirá en pruebas regulares de jugabilidad (playtesting) con el director, para evaluar la experiencia del usuario y detectar posibles problemas que puedan afectar negativamente la calidad o funcionalidad del videojuego.
- *Feedback* (estimación de 10 horas): Durante esta etapa se analizarán detalladamente los resultados obtenidos en la validación, recogiendo sistemáticamente las sugerencias del director para identificar claramente las áreas prioritarias que requieren ajustes y mejoras adicionales.
- *Mejoras* (estimación de 15 horas): Aquí se implementarán los ajustes derivados del feedback recibido, mejorando continuamente elementos mecánicos, gráficos, sonoros y de interacción. Asimismo, se integrará y validará el plugin desarrollado para facilitar la interacción directa con la plataforma Twitch.

Este proceso, al ser cíclico, se repetirá en múltiples ocasiones a lo largo del desarrollo del videojuego. Gracias a esta repetición iterativa, se asegura una adaptación constante a las expectativas y objetivos del proyecto, permitiendo correcciones oportunas y efectivas que conducen a un producto final robusto y altamente atractivo para los jugadores.

4.3. Herramientas externas utilizadas

Para asegurar un proceso eficiente, ordenado y profesional durante el diseño y desarrollo del videojuego interactivo, se han seleccionado cuidadosamente las siguientes herramientas, valorando sus beneficios específicos para cada aspecto del proyecto:

Unity 6: Se ha elegido como plataforma principal debido a su análisis previo en la *Sección 2* mostrando robustez y versatilidad.

Visual Studio: Seleccionado por ser uno de los entornos de desarrollo integrados (IDE) más completos y recomendados para la programación en C# y su integración con Unity.

Git: Elegido como sistema de control de versiones debido a su capacidad de mantener un registro exhaustivo del desarrollo, gestionar ramas simultáneas para distintos prototipos o funcionalidades, y facilitar la colaboración fluida entre el estudiante y el director. Se ha seleccionado git corporativo para facilitar la gestión interna.

Microsoft Teams: Esta herramienta de comunicación y colaboración fue seleccionada por su capacidad para mantener contacto directo, rápido y constante con el director.

5. Desarrollo del Proyecto

En este capítulo se detalla el recorrido completo del proyecto en **dos fases principales: primero, el desarrollo del plugin de Twitch para Unity**; y, a continuación, el **diseño y la implementación del videojuego interactivo** que hace uso de ese plugin. El trabajo se organizó en nueve semanas de sprints semanales, distribuidos en diez tareas clave y guiados por una metodología ágil que permitió ajustar prioridades, atender incidencias y entregar valor de forma continua.

Durante la primera fase se investigó la API de Twitch, se definió la arquitectura del plugin y se diseñó su interfaz de usuario y sistema de comandos. Cada sprint incluyó actividades de codificación, pruebas unitarias e integración continua, así como la evaluación de alternativas tecnológicas (*WebSockets*, modelos de eventos, gestión de tokens *OAuth*) y la resolución de retos como el filtrado de mensajes y el control de límites de petición.

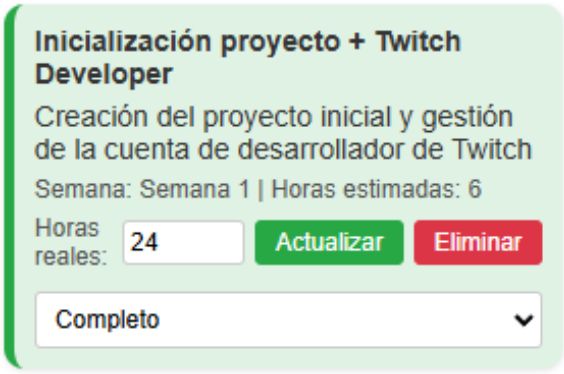
En la segunda fase se abordó el videojuego interactivo: se conceptualizaron mecánicas y niveles, se creó la lógica de interacción con el chat de Twitch y se integró el plugin en escenas de Unity. Esta etapa incluyó pruebas de usabilidad y ajustes de diseño para garantizar una experiencia fluida.

Por último, se presentan los resultados de rendimiento y estabilidad obtenidos (latencia, carga de CPU/GPU, escalabilidad) y las lecciones aprendidas, con el fin de establecer buenas prácticas para futuras integraciones Twitch-Unity.

5.1. Desarrollo del Plugin de Integración con Twitch

En la presente sección se describe el itinerario completo de construcción del plugin de integración con Twitch, organizado **en nueve semanas de trabajo** distribuidas en diez tareas principales. Cada semana constituyó un sprint ágil en el que se planificó, desarrolló y revisó una funcionalidad concreta del sistema, desde la inicialización del proyecto y la configuración de credenciales en *Twitch Developer*, hasta la validación de las llamadas a la API y la sincronización en tiempo real con el chat. Gracias al uso de metodologías ágiles, pudimos adaptar rápidamente los objetivos a los requerimientos emergentes, priorizar incidencias críticas y asegurar una entrega incremental de valor en cada iteración.

5.1.1. Desarrollo de la Semana 1: Inicialización del Proyecto



En la primera semana de trabajo se sentaron las bases del proyecto: se creó la plantilla en Unity, se configuró el repositorio corporativo de la empresa de prácticas y se dio de alta la cuenta de desarrollador en *Twitch Developer Portal* (*twitch.tv/army_z*) y su aplicación externa con Twitch. A continuación, se facilitó el flujo de autenticación *OAuth* externo que será abordado en la siguiente semana.

Aunque sólo se estimaron 6 horas de trabajo, el esfuerzo real fue de 24 horas porque fue necesario emitir streams regulares durante dos semanas para cumplir los requisitos del **programa de afiliados de Twitch** (número mínimo de espectadores y horas de emisión). Este proceso permitió disponer de tokens y scopes reales, indispensables para validar todas las llamadas a la API en un entorno auténtico.

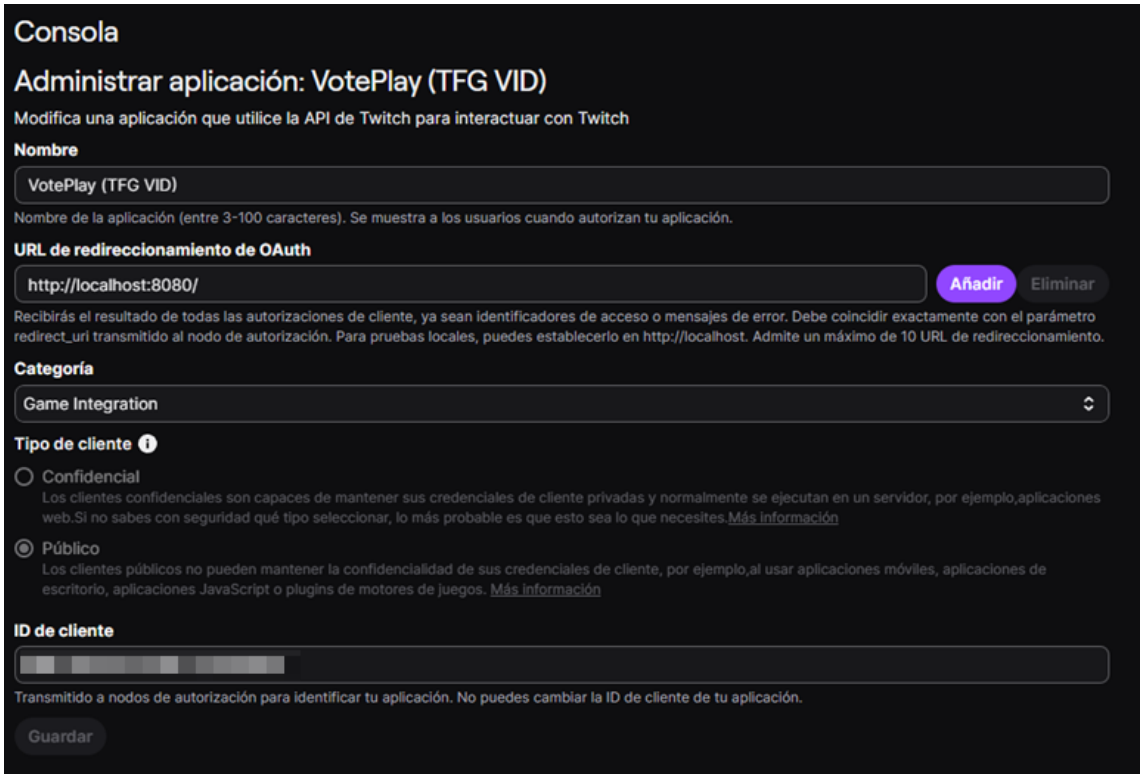
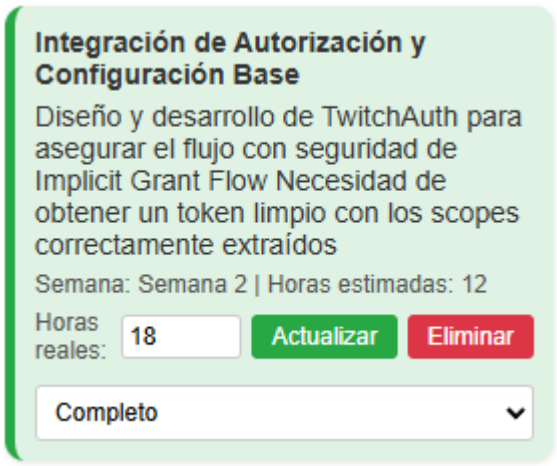


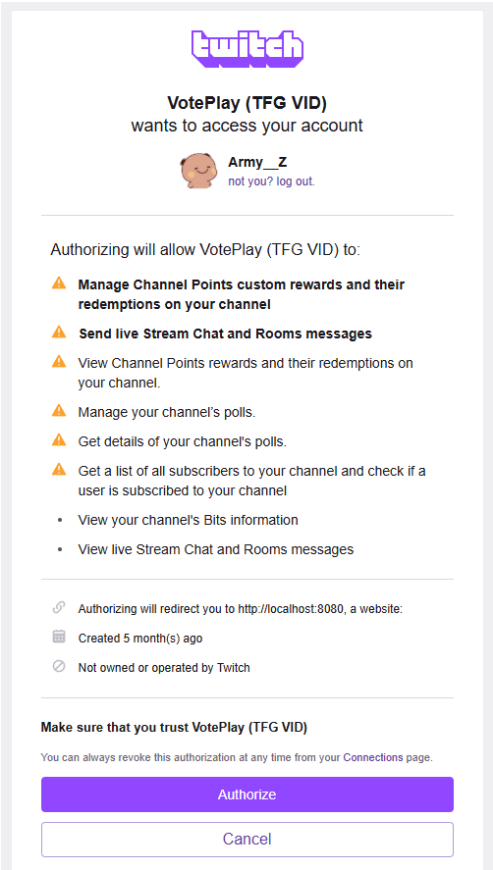
Ilustración 15: Panel de la Aplicación registrada en Twitch. Para redireccionamientos personalizados será necesario disponer de certificado SSL en la ruta. Captura de la consola de la propia cuenta de Twitch.

Para cerrar la semana, se refinó la interfaz de configuración (*TwitchSettingsWindow*), simplificando la inserción y actualización de credenciales en Unity y garantizando un arranque del plugin estable y listo para las fases posteriores.

5.1.2. Desarrollo de la Semana 2: Autorización de la Aplicación

Durante esta segunda semana se diseñó y desarrolló la clase *TwitchAuth*, encargada de gestionar de forma segura el flujo de *Implicit Grant* en Unity. Para ello se implementó un pequeño servidor HTTP local capaz de interceptar la redirección de Twitch con el token de acceso, extraer los parámetros *access_token* y *refresh_token*, y almacenarlos en el asset *TwitchConfig.asset*. Adicionalmente, se añadieron mecanismos de detección y recuperación de errores (fallos de CORS, time-outs) y de renovación automática de credenciales para garantizar la disponibilidad continua del plugin.





Aunque se habían estimado 12 horas para esta tarea, el esfuerzo real ascendió a 18 horas. El motivo principal fue la depuración de las peticiones y la conversión de carácteres, la adaptación del flujo de redirección a distintos entornos y la comprobación exhaustiva de todos los scopes obtenidos, para asegurar que las llamadas a la API pudieran ejecutarse sin interrupciones.

Este desarrollo amplía la configuración inicial descrita en la sección anterior, donde se definieron el Client ID, la Redirect URI y los scopes necesarios dentro de Unity como podemos ver en la siguiente Ilustración. Con el token ya validado por *TwitchAuth*, el plugin está habilitado para invocar de forma autenticada los *endpoints* (encuestas, recompensas, chat), constituyendo la base sobre la que se construirá el resto de las funcionalidades del videojuego interactivo.

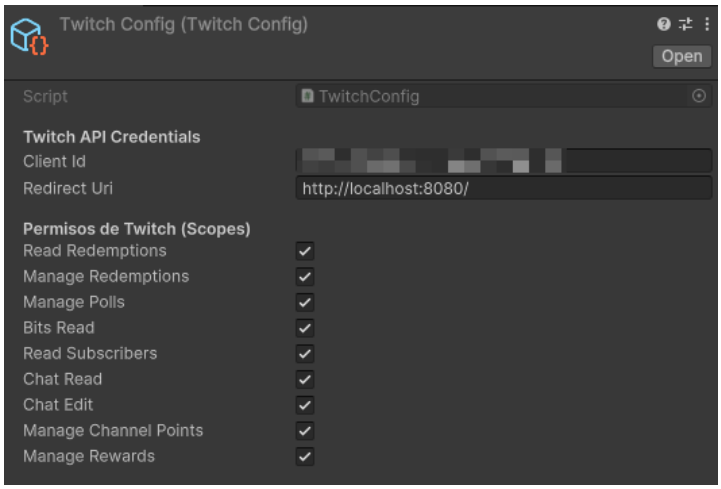
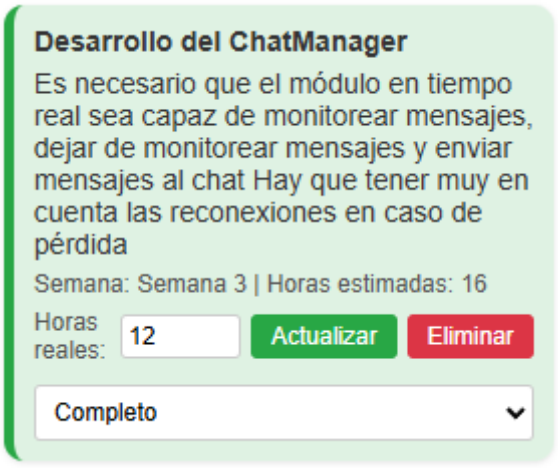


Ilustración 16: Información en Unity sobre los ajustes de scopes y redireccionamiento para la aplicación. Captura de pantalla del propio motor.

5.1.3. Desarrollo de la Semana 3: Implementación y Gestión del Chat



En esta tercera semana se creó el módulo *ChatManager* para conectar con el servidor IRC de Twitch desde Unity y procesar en tiempo real los mensajes de los espectadores. *ChatManager* se implementó como un *MonoBehaviour singleton* que al arrancar abre un socket TCP hacia irc.chat.twitch.tv, envía las credenciales (PASS y NICK), responde automáticamente a los ping de Twitch y mantiene viva la sesión. Cada línea recibida se analiza en tres fases para extraer las etiquetas (tags) (como el identificador y nombre de usuario), el prefijo del comando y el texto del mensaje, que luego se propaga al resto del sistema. Se añadieron además mecanismos de reconexión con reintentos exponenciales para recuperar la sesión tras desconexiones inesperadas y un método *SendChatMessage(string)* para enviar texto al canal.

Aunque se habían estimado 16 horas para esta tarea, el tiempo real registrado fue de 12 horas gracias a las semanas anteriores evitando problemas que se pudieran dar por interpretación de los scopes al presentar Twitch una documentación pobre y desactualizada.

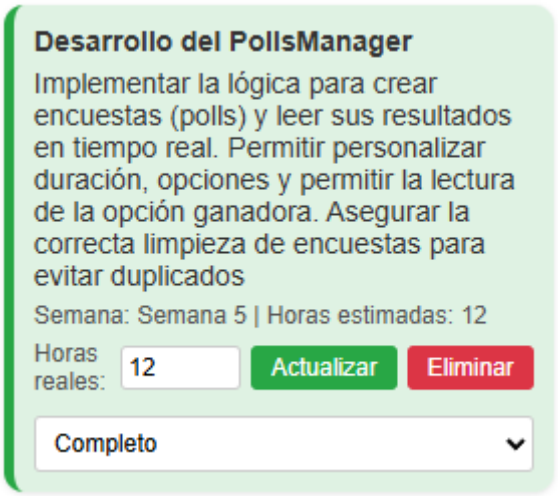
5.1.4. Desarrollo de la Semana 4: Implementación y Gestión del Sistema de Recompensas

En la cuarta semana se diseñó e implementó la clase *RewardsManager*, responsable de crear, personalizar y eliminar las recompensas de canal mediante la API Helix de Twitch y de gestionar en tiempo real las redenciones realizadas por los espectadores. Para ello se desarrollaron métodos que llaman a los endpoints *CreateCustomReward*, *DeleteCustomReward* y *GetCustomRewardRedemptions*, junto con un suscriptor a PubSub que escucha los eventos de redención y confirma automáticamente aquellas recompensas que deben marcarse como cumplidas. Asimismo, se implementó un control estricto de la tasa de llamadas (800 peticiones por minuto) para evitar bloqueos y se optimizó la latencia minimizando el número de solicitudes necesarias.

Aunque se habían estimado 16 horas para esta tarea, el esfuerzo real alcanzó las 20 horas. La razón principal fue la curva de aprendizaje de la API de recompensas y la depuración de escenarios complejos, como la gestión simultánea de múltiples redenciones y las pruebas de cancelación de recompensas no cumplidas. Además, se dedicaron varias sesiones a validar el comportamiento del sistema en entornos reales de streaming, verificando que las recompensas se muestran al espectador con la mínima demora posible y no altera el resto de las recompensas.

5.1.5. Desarrollo de la Semana 5: Implementación y Gestión del Sistema de Encuestas

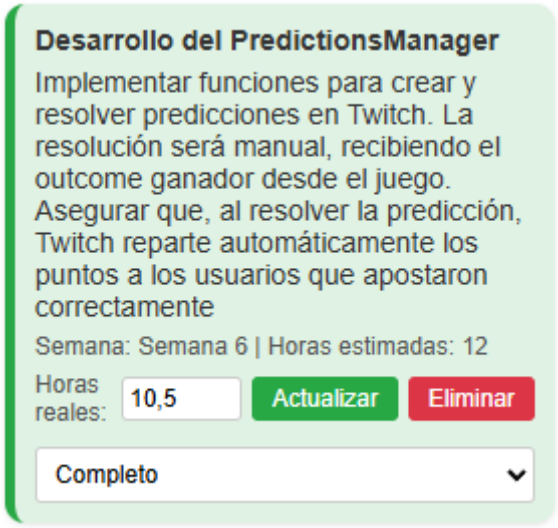
Durante la quinta semana se implementó la clase *PollsManager*, encargada de crear y gestionar encuestas en Twitch mediante los *endpoints* de la API Helix. *PollsManager* ofrece métodos para configurar la duración de la encuesta, definir el título y las opciones de voto, lanzar la petición de creación *(CreatePoll),* y consultar los resultados en tiempo real con *GetPolls* y *GetPollChoices*. Para asegurar la limpieza entre encuestas, se



incluyó lógica que cancela cualquier encuesta activa antes de iniciar una nueva, evitando duplicados y conflictos de identificadores.

Todas las llamadas a la API se ejecutan dentro de corutinas de Unity, lo que permite integrar de forma asíncrona la creación, monitorización y finalización de encuestas sin bloquear el hilo principal del motor. Se añadieron además mecanismos de control de errores (reintentos ante fallos de red o respuestas con código 429 por exceso de peticiones) y de selección automática de la opción ganadora cuando finaliza el período de votación.

5.1.6. Desarrollo de la Semana 6: Implementación y Gestión del Sistema de Predicciones



Durante la sexta semana se diseñó e implementó la clase *PredictionsManager*, encargada de ofrecer en Unity las funciones necesarias para crear predicciones en Twitch, resolverlas desde el propio juego y garantizar la distribución automática de puntos a los espectadores que acertaron la opción ganadora. Para ello se desarrollaron métodos que invocan los endpoints Helix *CreatePrediction*, *EndPrediction* y *ResolvePrediction*, junto con la lógica interna que mantiene un registro de las *prediction_id* activas y las opciones asociadas. La resolución de cada predicción se dispara manualmente por el juego cuando determina el resultado, tras lo cual Twitch reparte los Channel Points de forma transparente a los usuarios ganadores.

Aunque la estimación inicial era de 12 horas, el tiempo real empleado fue de 10,5 horas. Esta reducción se debió al aprovechamiento de gran parte del código de *PollsManager* para el manejo de corutinas y tokens y problemas HTTP como por ejemplo al enviar demasiadas peticiones, junto con la limitación de pruebas necesarias gracias a la similitud del flujo de resolución manual con el sistema de recompensas. Sin embargo, se dedicaron 3 horas adicionales a depurar casos límite en la resolución simultánea de varias predicciones y a verificar que Twitch distribuía correctamente los puntos en todas las plataformas soportadas (web y móvil).

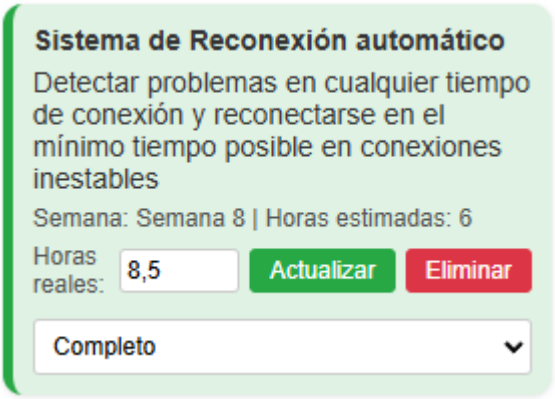
5.1.7. Desarrollo de la Semana 7: Implementación y Gestión del Sistema de Donaciones (Bits)

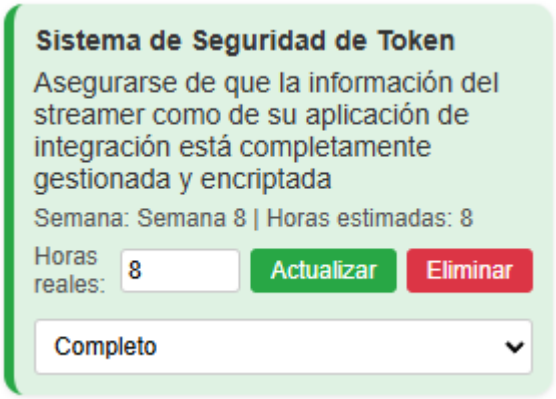
En la séptima semana se diseñó e implementó *BitsManager*, el módulo encargado de captar en tiempo real las donaciones de bits realizadas por los espectadores. Para ello se aprovechó el canal PubSub autenticado proporcionado por TwitchManager, suscribiéndose al topic channel-bits-events-v2.<broadcaster_id> y escuchando los mensajes JSON que Twitch envía cada vez que un usuario hace "*cheer*" con bits. Estos mensajes se parsean para extraer el nombre de usuario, la cantidad de bits donados y el mensaje opcional, y se disparan eventos como *OnBitsReceived(user, amount, message)* para que el juego reaccione de forma inmediata.

Durante el desarrollo se añadieron control de reconexión automática y limitación de la tasa de peticiones para evitar bloqueos, así como manejo de errores ante paquetes mal formados o desconexiones de la sesión PubSub. También se crearon métodos *StartMonitoringBits()* y *StopMonitoringBits()* para activar o desactivar la escucha según el estado del juego, y se realizaron pruebas con bits de prueba en distintos escenarios de uso (alta frecuencia de cheers, abandono de conexión, mensajes muy largos).

Aunque se habían estimado 10 horas para esta tarea, el trabajo real ascendió a 12 horas debido a la complejidad de depurar el *parsing* de los payloads JSON, adaptar la suscripción PubSub al entorno de Unity (editor vs build), y verificar el correcto envío de eventos bajo condiciones de estrés.

5.1.8. Desarrollo de la Semana 8: Implementación de Reconexión y Seguridad





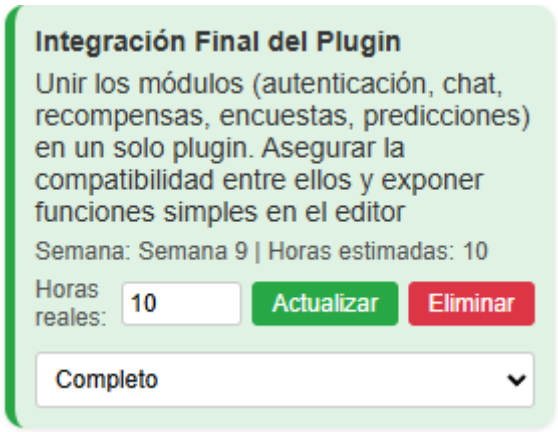
En esta octava semana se implementó un mecanismo de reconexión automática que vigila continuamente el estado de las conexiones tanto al chat IRC como al canal *PubSub* de Twitch.

Cuando se detecta una caída, el sistema intentará restablecer la conexión tras un breve retraso y volverá a suscribir todas las funcionalidades activas (mensajes, bits, recompensas, encuestas) sin duplicar listeners ni perder eventos. Aunque se planificaron 6 horas para esta tarea, el desarrollo y las pruebas en diferentes entornos tomaron 8,5 horas, principalmente por la complejidad de generar desconexiones controladas y comprobar que el restablecimiento mantuviera la consistencia del estado interno.

Simultáneamente, se añadió un sistema de cifrado para proteger el *access_token* y el *refresh_token* almacenados en el asset de configuración. Se encriptan las credenciales en disco y sólo se desencriptan en memoria al iniciar el plugin con el permiso del usuario. Este enfoque impide que un simple volcado de archivos revele los tokens, y se complementó con una verificación de integridad HMAC para detectar manipulaciones. La estimación original de 8 horas se cumplió en su totalidad gracias al aprovechamiento de librerías de cifrado ya conocidas y a la concentración de las pruebas en casos de extracción básica de datos.

5.1.9. Desarrollo de la Semana 9: Integración Final

En la novena semana se integraron todos los módulos desarrollados (autenticación, chat, recompensas, encuestas y predicciones) en un único paquete de Unity. Se consolidó la API interna del plugin para exponer en el editor funciones limpias y coherentes, permitiendo activar o desactivar cada sistema desde la ventana de configuración y facilitando su uso en cualquier proyecto.



Durante esta fase también se realizaron pruebas de extremo a extremo, comprobando que cada subsistema interactúa correctamente con los demás: por ejemplo, que los eventos de chat pueden disparar encuestas o predicciones, y que la resolución de estas genera recompensas automáticas. Se ajustaron los nombres y parámetros de las llamadas públicas para garantizar una experiencia de usuario uniforme y se documentaron en el README del repositorio los pasos básicos para instalar y usar el plugin.

Aunque la estimación inicial era de 10 horas, el esfuerzo real fue de 10 horas, ya que la mayoría del trabajo consistió en reunir y reorganizar código ya probado, aplicar ligeros retoques de nombres y firmar la versión final del paquete.

5.2. Diseño y Desarrollo del Videojuego de Integración

Este apartado describe cómo se ha concebido, planificado y construido el núcleo del juego en Unity, con foco en los módulos propios (UI, aparición de jugadores, reglas y audio). La Ilustración 17 resume la **arquitectura** a alto nivel y la extensión del proyecto: a la izquierda, los componentes del videojuego; a la derecha, el plugin de integración con Twitch (documentado en la sección anterior). En lo que sigue se desarrolla el enfoque metodológico, el diseño conceptual y la organización técnica de los sistemas que coordina GameManager.

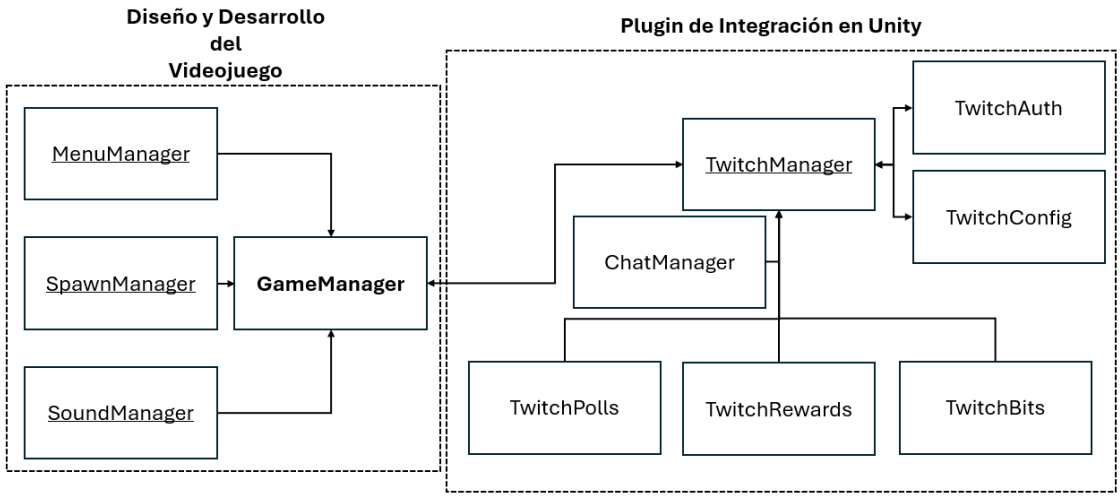


Ilustración 17: "Arquitectura a alto nivel". Bloque jugable e integración descrito en la Sección anterior 5.1.

5.2.1. Enfoque metodológico y planificación

El desarrollo se organizó en iteraciones cortas con objetivos verificables. Cada ciclo comenzaba definiendo una capacidad observable del prototipo (por ejemplo, alta de jugadores desde el chat mediante !jugar, creación de encuestas de duración/categoría o reacción a canjes en tiempo real) y concluía con pruebas funcionales y de percepción (latencia, claridad de interfaz y robustez frente a reconexiones). Los riesgos técnicos se abordaron de forma anticipada: autenticación *OAuth*, escucha por *WebSocket (PubSub)* y llamadas a la API Helix se implementaron y validaron en etapas tempranas para no bloquear fases posteriores de contenido y pulido. La **revisión de cada iteración alimentó el siguiente ciclo**, consolidando las piezas estables y sustituyendo prototipos por implementaciones definitivas.

5.2.2. Diseño conceptual y documentación de juego

El diseño parte de un principio *audience-first:* el **espectador es el primer ciudadano del sistema**. Entra al lobby con !jugar, vota la duración mediante encuesta oficial de Twitch, elige categoría con comandos únicos por usuario y participa en el quiz respondiendo !a...!d. Las

recompensas cosméticas permiten personalizar el avatar sin alterar la mecánica central. Este marco se formalizó en un GDD que fija alcance, tono visual y reglas de puntuación (1 punto por acierto y bonificación al primer acierto), así como el cierre ritual en forma de pódium. El GDD funcionó como "fuente de verdad" durante la implementación, facilitando la trazabilidad entre decisiones de diseño y soluciones técnicas. (Véase Anexo I para el documento completo.)

5.2.3. Arquitectura y organización del proyecto en Unity

La solución se estructura en módulos con responsabilidades bien acotadas que se comunican por eventos y llamadas explícitas:

- *GameManager* actúa como orquestador. Gestiona los estados de juego (lobby, votaciones, preguntas, pódium), coordina temporizadores, aplica reglas de puntuación y desencadena las transiciones de UI y audio. Centraliza la suscripción y cancelación de monitores de chat para evitar efectos residuales entre estados.
- *SpawnManager* administra el ciclo de vida de los avatares. Resuelve posiciones de aparición en el segmento entre salidas con muestreo de candidatos y distancia mínima en XZ, previene duplicados por usuario y expone utilidades para obtener y eliminar jugadores.
- *MenuManager* encapsula la interfaz: muestra y oculta paneles, instancia cronómetros y asegura el anclaje y la jerarquía correcta de elementos en diferentes resoluciones, manteniendo legibilidad y consistencia cromática (verde para acierto, rojo para error).
- *SoundManager* centraliza la reproducción de música de fondo y efectos. Permite reproducción puntual y en bucle con volumen por pista y *fade-out* al reiniciar, con la opción de conservar una pista ambiente entre estados para evitar cortes perceptibles.

Los *singletons* (GameManager, SpawnManager, MenuManager, SoundManager) se inicializan con *DontDestroyOnLoad* para garantizar disponibilidad y evitar duplicados. Esta segmentación facilitó pruebas modulares y la sustitución progresiva de prototipos, además de minimizar acoplamientos con el plugin de integración (tratado aparte). La organización descrita en la Ilustración 17 ha permitido una evolución controlada del proyecto y una respuesta fiable a los eventos de Twitch sin comprometer la estabilidad del bucle jugable.

5.2.4. Flujo de juego y estados principales

El diagrama siguiente incorporado sintetiza el ciclo completo de la aplicación como una **máquina de estados** finitos coordinada por *GameManager*. Las transiciones se disparan por tres tipos de estímulos: eventos de Twitch (mensajes de chat, encuestas y canjes como por ejemplo puntos de canal o bits), temporizadores internos (cuentas atrás de votaciones y preguntas) y condiciones

de juego (puntuación alcanzada). Cada cambio de estado actualiza la UI mediante MenuManager y emite señales acústicas con SoundManager.

Inicio (Autenticaciones Twitch)

Al abrir la aplicación se inicializa la autenticación OAuth y se arrancan los módulos de integración (*TwitchManager*, *TwitchAuth*, *TwitchConfig*). En este punto se valida/renueva el token, se limpia cualquier recompensa residual del canal y se activan los listeners de PubSub/IRC. Si el token llega más tarde, la acción se encola y se aplica en el hilo principal.

Listar jugadores

En el lobby, ChatManager monitoriza el comando !jugar. Por cada usuario único (normalizado) *SpawnManager* instancia un avatar en una posición "inteligente" entre las salidas, evitando solapes y duplicados, y rotula el nameplate. Este estado permanece activo hasta que el streamer decide iniciar la partida.

Encuesta de duración

TwitchPolls crea una encuesta Helix con tres opciones (corta, media, larga). La UI muestra un aviso superior y se reproduce la señal de inicio de votación. Al cierre, GameManager fija el objetivo de puntos en función del resultado.

Encuesta de categorías

Se habilita la votación por comandos (!ciencias, !deportes, etc.). ChatManager acepta un voto por usuario y actualiza el contador en tiempo real. Un cronómetro en pantalla marca el cierre. Resuelta la categoría, la UI confirma el resultado y se prepara el banco de preguntas.

Bucle de preguntas (subciclo del diagrama)

Cada iteración presenta enunciado, activa el cronómetro y registra una única respuesta por usuario (!a...!d). Al expirar el tiempo, GameManager resuelve: otorga 1 punto por acierto y un punto adicional al primer acierto registrado, y ordena a SpawnManager mover los avatares proporcionalmente al avance. Durante el desplazamiento se activa un loop de "saltitos" en SoundManager que se detiene al finalizar la animación. La UI pinta la solución (verde para correcta, atenuación para el resto) y, tras una breve pausa, pasa a la siguiente pregunta.

Condición de victoria y pódium

Cuando algún jugador alcanza el umbral fijado por la duración votada, el bucle se interrumpe. GameManager calcula el top-3 (puntos y aciertos como desempate) y MenuManager muestra el pódium. Se reproducen fanfarrias y se ofrecen opciones de Reiniciar o Salir.

Reiniciar / Salir

Reiniciar restablece colecciones internas (puntuaciones, votos, preguntas usadas), detiene monitores de chat, destruye avatares y borra las recompensas gestionadas, manteniendo opcionalmente la música de fondo. Salir realiza la misma limpieza y cierra conexiones.

Gestión robusta de eventos

El tráfico de PubSub se procesa en un hilo de recepción y se encola para su ejecución en el hilo principal de Unity, evitando condiciones de carrera. La creación/eliminación de recompensas es idempotente (por título/ID) y el token de escucha se sanea antes del LISTEN para evitar ERR_BADAUTH. Las reconexiones de WebSocket se gestionan con reintentos exponenciales suaves.

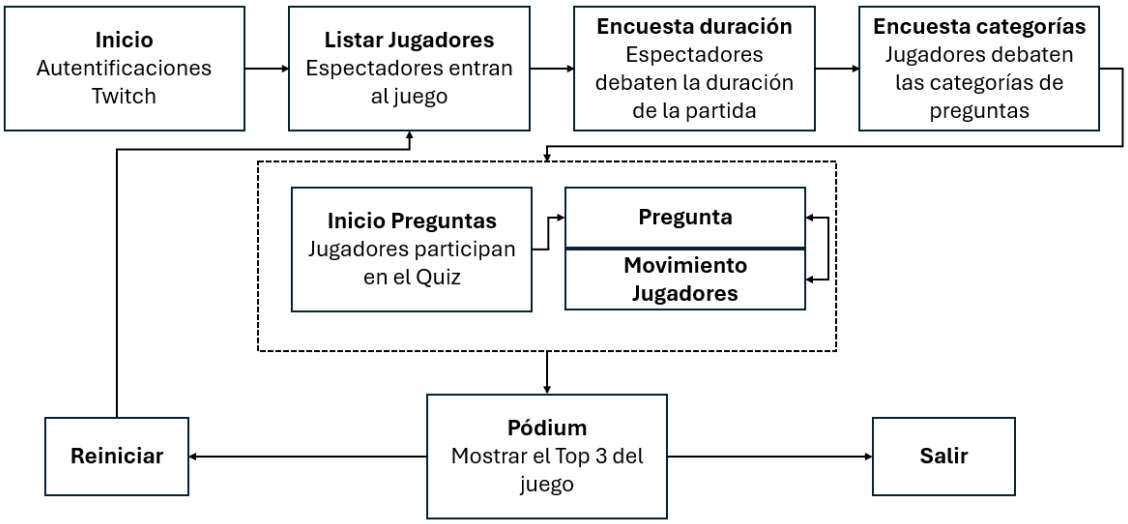


Ilustración 18: "Diagrama de flujo del videojuego".

5.2.5. Interfaz de usuario y experiencia

La interfaz se diseñó para emisión en directo: tipografía de alto contraste, mensajes concisos y elementos ubicados de forma consistente para que el espectador identifique el estado del juego de un vistazo. La estética general mantiene el estilo cartoon del bosque, pero los paneles de UI utilizan marcos y fondos sólidos para garantizar legibilidad sobre cualquier fondo.

El enunciado y las opciones se presentan siempre en mayúsculas con grosor alto (TMP), blanco sobre rojo/gris oscuro. El cronómetro circular se sitúa de manera predecible en el borde inferior del panel activo; su cuenta atrás combina color y vacío progresivo para que sea interpretable incluso en pantallas pequeñas. La codificación por color se limita a semánticas claras (verde = acierto, rojo = incorrecto) y se acompaña de cambios de énfasis tipográfico (negrita/atenuación) para no depender únicamente del color.

La progresión se comunica en dos planos complementarios: en interfaz, con el número de pregunta y la resolución visual; y en mundo, con el desplazamiento horizontal de los avatares hacia la meta. Las recompensas cosméticas (cambio de material del champiñón) afectan únicamente al aspecto del avatar del usuario que canjea, sin alterar mecánicas ni legibilidad del conjunto.

Desde el punto de vista técnico, se emplean *anchors* y *Canvas Scaler* con referencia 1920×1080, asegurando proporciones estables en 16:9, 16:10 y 21:9. El cronómetro y elementos circulares usan *AspectRatioFitter* para evitar deformaciones (corrección aplicada tras las primeras pruebas). Las transiciones de panel utilizan escalado y fade suaves (MenuManager), y las señales acústicas refuerzan cambios de estado sin solaparse con la música ambiente.

Ilustración 19: "Lobby, escribe !jugar". Llamada a la acción anclada en la parte superior; cámara elevada que muestra pista, línea de meta y zonas de spawn. Claridad para el onboarding del espectador.

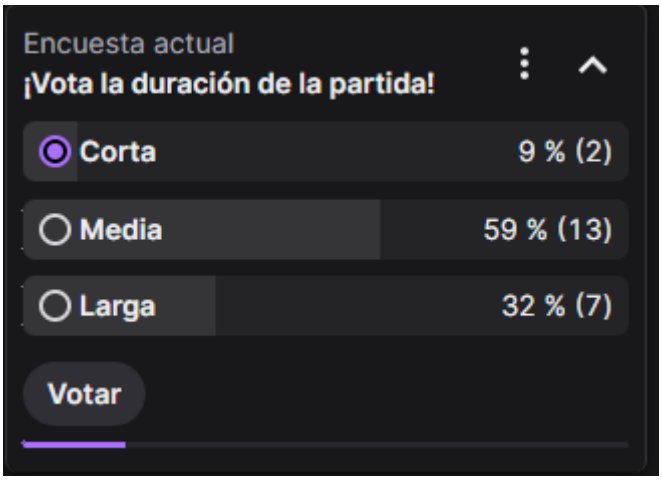


Ilustración 20: "Encuesta de duración". La votación se realiza en la propia interfaz de Twitch. En el juego se acompaña con sonido de inicio de votación y marcador temporal.

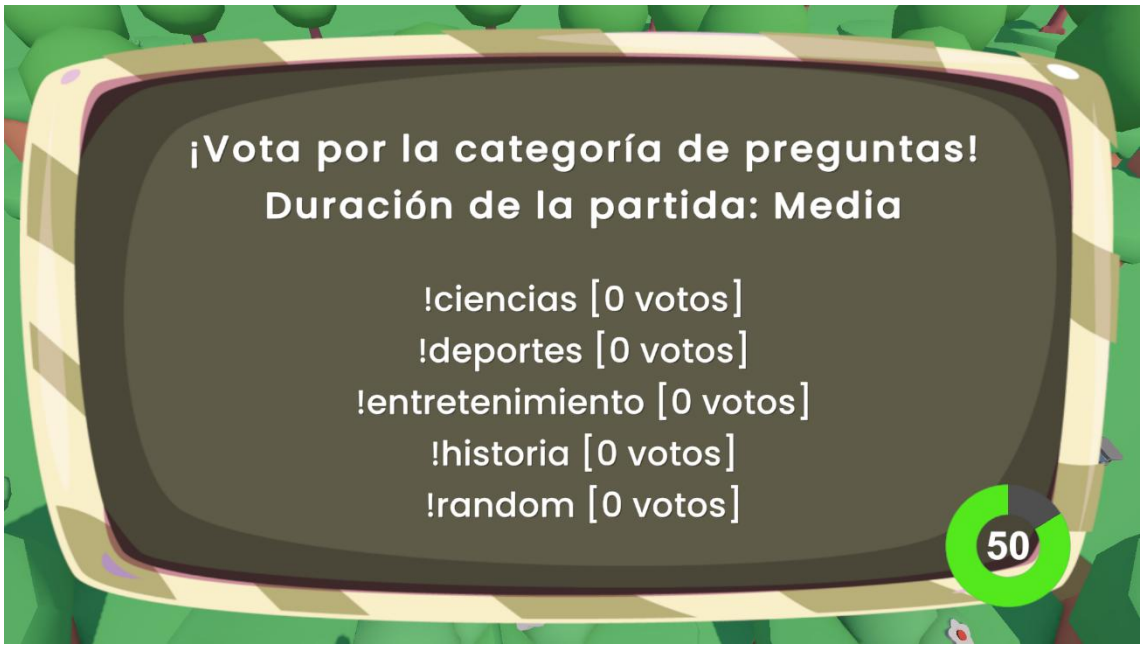


Ilustración 21: "Votación de categoría de juego". Panel "televisor" con comandos (!ciencias, !deportes, …) y contador visible. Tipografía grande y espaciado generoso para lectura rápida.

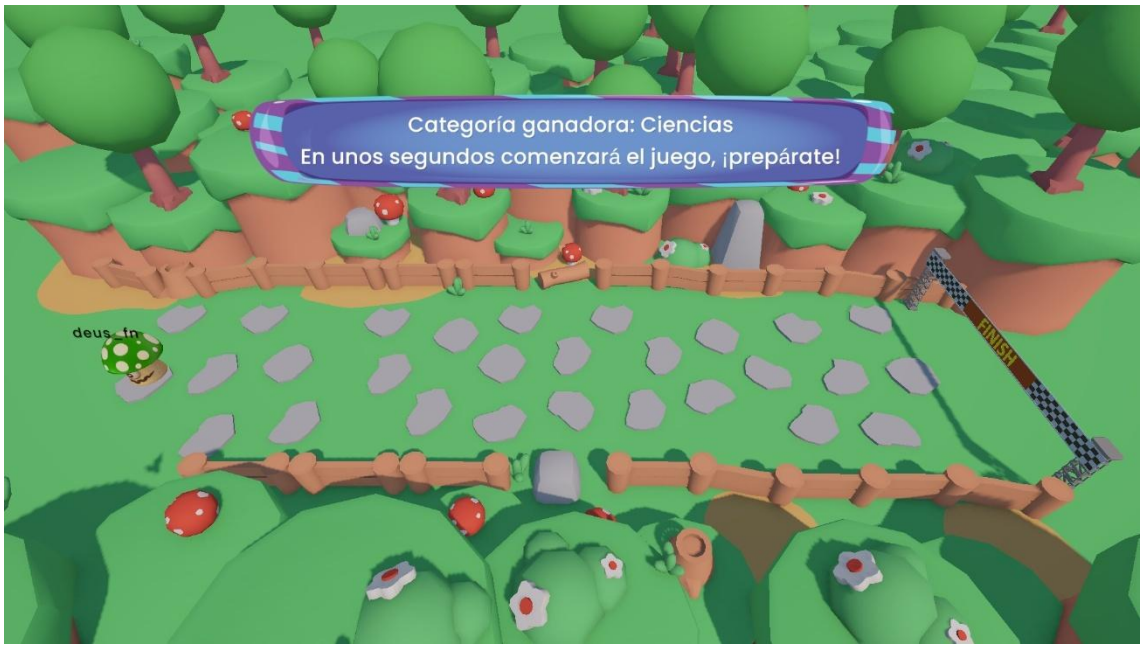


Ilustración 22: "Categoría ganadora". Mensaje de confirmación centrado en la franja superior. Breve pausa antes de iniciar la fase de preguntas.

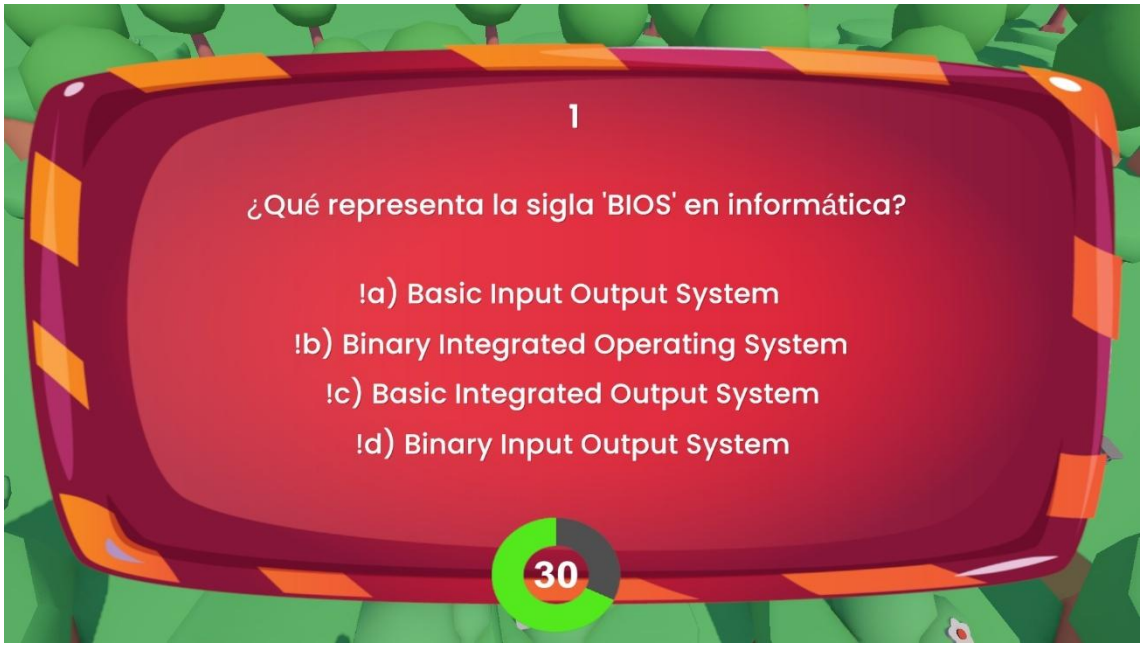


Ilustración 23: "Panel de pregunta". Número de pregunta, enunciado y cuatro opciones etiquetadas con comandos (!a...!d). Cronómetro circular inferior como referencia temporal añadiendo música background creciente para tensar a los jugadores.

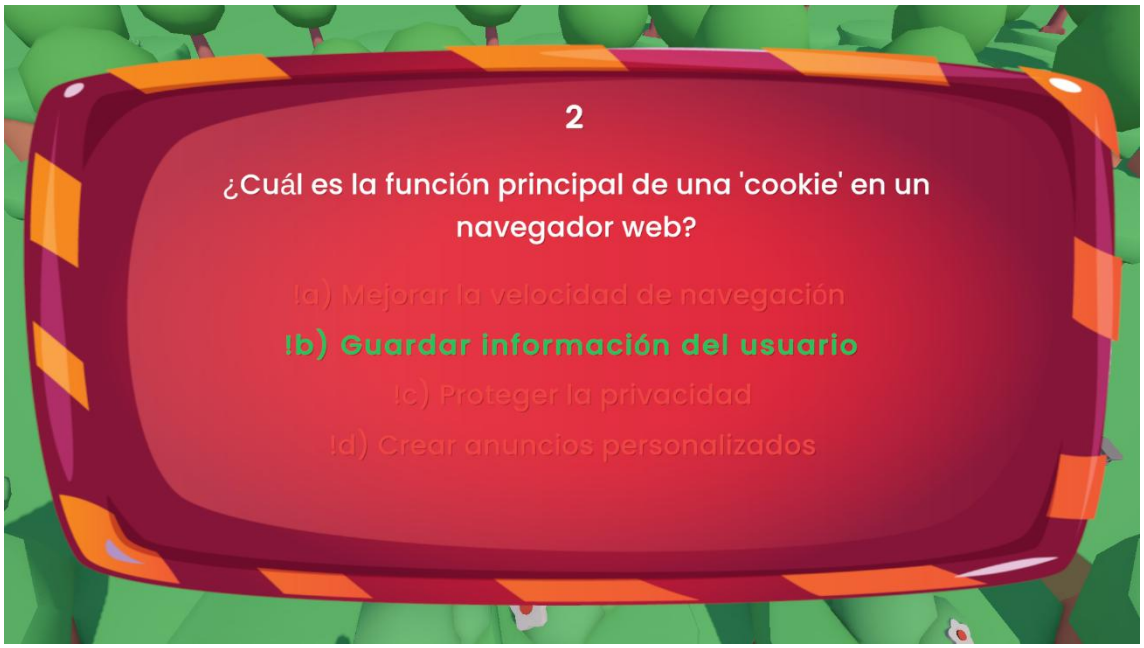


Ilustración 24: "Resolución de la pregunta". Realce de la opción correcta en verde y negrita; resto atenuado. Se evita el parpadeo y se mantiene el texto en su lugar para no generar movimientos innecesarios.

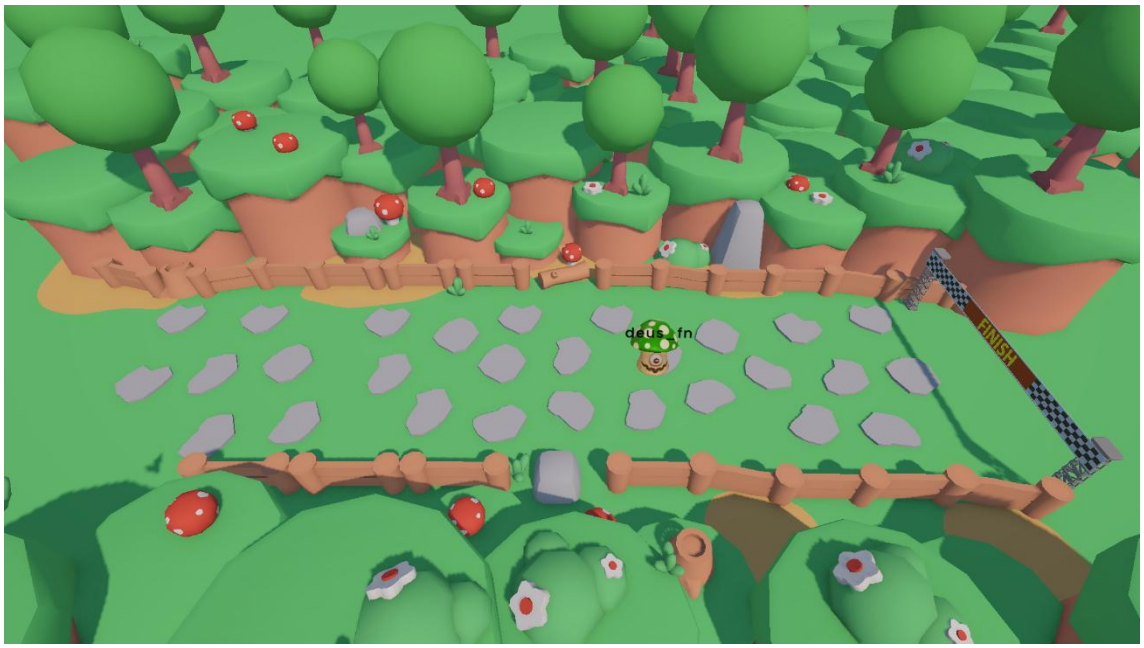


Ilustración 25: "Progreso en pista". Vista del movimiento del avatar tras puntuar; el desplazamiento es proporcional a los puntos aplicables. Sonido de "saltitos" en bucle mientras dura la animación y parada al finalizar con el fin de dar un feedback de movimiento de los champiñones.

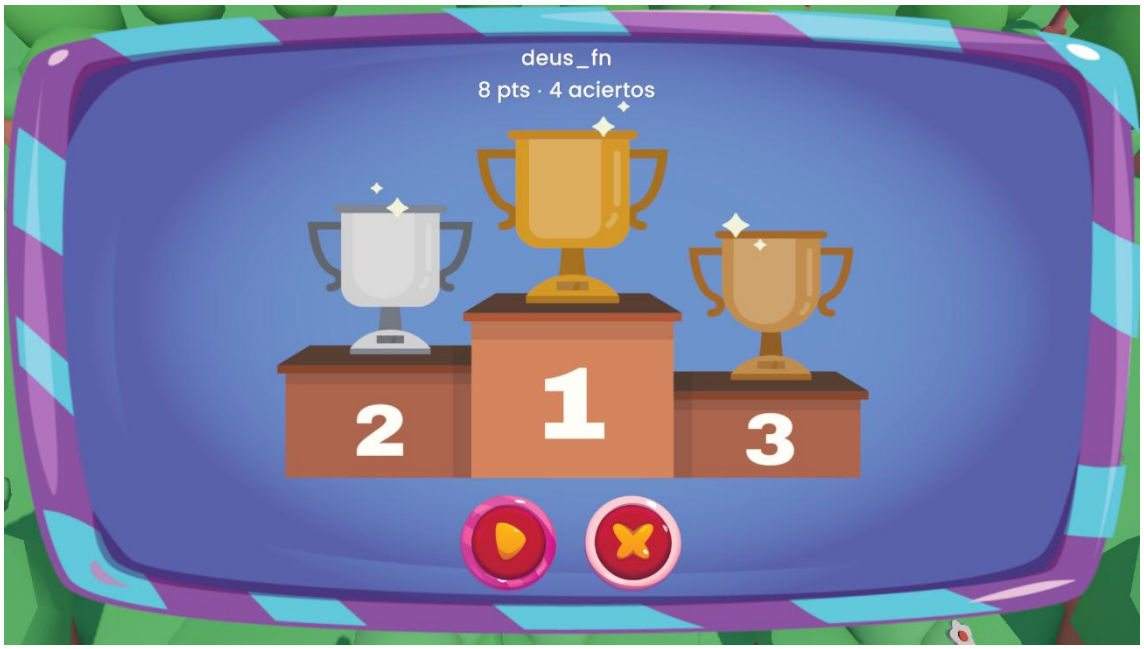


Ilustración 26: "Pódium final". Composición centrada con trofeos y rótulo del ganador (puntos y aciertos). Botones grandes de Reiniciar y Salir con el mismo lenguaje visual que el resto de la UI.

5.2.6. Mecánicas, puntuación y movimiento

La interacción principal se realiza desde el chat de Twitch mediante cuatro comandos mutuamente excluyentes (!a, !b, !c y !d) que representan las opciones de respuesta. El sistema solo admite aportaciones de usuarios que han entrado en partida con !jugar y disponen de avatar activo. Para garantizar la equidad, cada mensaje se normaliza (sin @, en minúsculas) y se registra en un conjunto de control; de este modo, cada usuario puede responder una única vez por pregunta y los intentos posteriores se ignoran. La ventana de respuesta está delimitada por un cronómetro visible; al expirar, las suscripciones al chat se desactivan y la pregunta se cierra de forma determinista.

El modelo de puntuación reconoce **un punto por acierto y un punto adicional al primer acierto detectado**. Durante la ronda se almacenan, por separado, los usuarios que respondieron, los que acertaron y el primer acierto. El reparto de puntos no se aplica hasta el cierre de la pregunta, con lo que se evita cualquier sesgo derivado del orden de llegada de mensajes y se facilita la consistencia en directo. Se mantienen dos contadores: uno "real", sin límites, para el pódium final, y otro "progreso a meta", acotado por el objetivo de partida. Este último se incrementa con saturación, de forma que nunca sobrepasa el umbral de victoria y permite detectar el final de la partida sin artefactos. En caso de empate en puntos reales, el sistema desempata por número total de aciertos acumulados.

El avance visual de cada avatar se obtiene al transformar los puntos aplicables de la ronda en desplazamiento sobre el eje X. El tramo entre la salida y la meta se divide por el número de puntos objetivo de la partida (votado previamente en el paso anterior de duración de la partida), generando un paso constante; por cada punto asignado se interpola la posición con una curva lineal durante un tiempo parametrizable. Mientras dura la interpolación se activa el estado de "carrera" del avatar: si el modelo dispone de *Animator*, se conmutan parámetros estándar (IsRunning o Speed) o, en su defecto, se reproducen clips de animación *Legacy*. En paralelo se inicia un bucle sonoro ligero que simula pequeños saltos y que se detiene en cuanto finaliza la animación de movimiento, evitando solapes innecesarios y reforzando el cambio de estado de manera no intrusiva.

5.2.7. Recompensas y personalización cosmética

Las recompensas de canal se crean al inicio de la sesión y se eliminan de forma segura al reiniciar o cerrar la aplicación. Cada recompensa está asociada a un color de material y a un identificador único registrado tras la creación. Al detectarse la redención por *PubSub* se localiza el avatar del usuario y se aplica el material al *renderer* del modelo, afectando a todos los subcomponentes relevantes. La lógica contempla el caso en el que se canjee antes de unirse a la partida, devolviendo un mensaje informativo por chat en lugar de fallar silenciosamente. Además, en caso de error o intento de "truco" de algún espectador se muestra por el chat lo que ha intentado hacer para evidenciarlo frente al resto del público.

5.2.8. Audio, señales y retroalimentación

El sistema de audio se centraliza en *SoundManager*, un singleton ligero que expone un API mínimo (Play, PlayLoop, Stop, SetVolume y HandleGameRestart). Internamente mantiene un diccionario clave → AudioSource para que cada efecto o música tenga su pista propia y pueda controlarse de forma independiente. La librería de sonidos se referencia por clave semántica (p. ej., background_music, iniciar_votacion, iniciar_pregunta, background_preguntas, Jump, award), lo que facilita el mantenimiento y la lectura del código.

La música de fondo se arranca en Start() desde *GameManager* con *PlayLoop("background_music", 0.35f).* Al cambiar de estado (votaciones, preguntas, pódium) no se destruye el *AudioSource* gracias al patrón singleton, y se regula el volumen relativo para no competir con los efectos críticos. En reinicios se usa *HandleGameRestart(...)* para hacer *fade-out* controlado de todos los sonidos, con opción de conservar una pista ambiente si se desea (por ejemplo, mantener la música del lobby mientras se limpia estado).

Los efectos de interfaz marcan los hitos del flujo: apertura de votaciones (Play("iniciar_votacion", 0.8f)), comienzo de pregunta *(Play("iniciar_pregunta", 0.8f)),* aviso de fin de tiempo y entrega de premios *(Play("award", 0.6f)).* Durante las animaciones de movimiento de los avatares se simula el "salto" encadenado activando un bucle *PlayLoop("Jump", 0.8f)* al inicio de la interpolación y deteniéndolo con *Stop("Jump")* al finalizar, evitando superposiciones innecesarias. Este patrón garantiza que **cada señal sonora esté anclada a un evento del juego**, mejorando la legibilidad del estado para el espectador.

Para minimizar fatiga auditiva:

- Se normalizó el nivel relativo de la música por debajo de los SFX (margen de ~-6 dB respecto a picos de efectos).
- Se priorizaron señales cortas y con espectro no intrusivo.
- Se evitó disparar el mismo efecto múltiples veces en el mismo frame controlando la concurrencia por clave.

5.2.9. Pruebas, calidad y gestión del riesgo

El aseguramiento de la calidad se articuló en ciclos iterativos que combinaron **pruebas funcionales**, de integración con Twitch y de percepción (UI/UX y audio). Al cierre de cada incremento se ejecutó una batería de casos sobre el flujo completo: alta de jugadores con !jugar, emisión de comandos de respuesta (!a/!b/!c/!d) y participación en votaciones de categorías. Se **verificó la idempotencia del sistema** (un único voto o respuesta por usuario a no ser que se indique lo contrario) y la correcta asignación de puntos, incluyendo el desempate por "primer acierto". Los cronómetros y paneles se validaron en **distintas resoluciones comprobando anclajes**, pivotes y tamaño relativo para evitar deformaciones; cuando fue necesario se incorporó *AspectRatioFitter* para asegurar la relación de aspecto. En el plano del *"game feel"*, se observó el acoplamiento entre las interpolaciones de movimiento y la retroalimentación sonora, confirmando que el bucle *"Jump"* se activa al inicio de la animación y se detiene al concluirla.

La integración con Twitch se ensayó en condiciones reales y controladas. Para la autenticación *OAuth* se sustituyó la página estática por una respuesta HTML servida por el *HttpListener* con estilos embebidos, lo que garantizó una apariencia consistente tanto en el Editor como en compilaciones. En *PubSub* se mitigaron errores ERR_BADAUTH saneando el token antes del LISTEN (eliminación de prefijos "Bearer " y comillas residuales) y se implementó un mecanismo de reconexión con retardo cuando el servidor cerraba la conexión. Las recompensas se sometieron a pruebas de creación, canje y borrado: se registraron los identificadores devueltos por Helix, se habilitó el autofulfill y se verificó el borrado masivo mediante

DeleteAllCreatedRewards() en reinicios y al salir de la aplicación, evitando dejar artefactos activos en el canal. De forma análoga, el ciclo de encuestas se validó desde su creación hasta la lectura de resultados, con reserva de una categoría (random) ante la ausencia de datos o combinación del resto de categorías.

La evaluación de interfaz y audio se realizó con sesiones de uso breves orientadas a claridad y legibilidad en directo. Se ajustaron contrastes, tamaños tipográficos y jerarquía visual para que el cronómetro y las opciones de respuesta fuesen dominantes en pantalla. En audio se normalizó la mezcla priorizando la inteligibilidad de los efectos sobre la música ambiente, y se evitó la superposición de eventos redundantes mediante control por clave en el SoundManager. La percepción de latencia entre el mensaje de chat y la reacción del juego se midió de forma exploratoria y se mantuvo en rangos adecuados para una retransmisión.

La gestión del riesgo se abordó desde el diseño modular y con planes de contingencia. La dependencia de servicios externos se encapsuló en un plugin aislado de la lógica de juego y con colas hacia el hilo principal de Unity, de modo que fallos temporales de red no interrumpen la experiencia local. La integridad del estado se protegió con un procedimiento centralizado de reinicio en GameManager que detiene cronómetros, anula suscripciones de chat, limpia audio, elimina avatares y borra recompensas antes de rearmar el lobby. Los incidentes detectados durante el desarrollo (como escalado inadecuado de cronómetros o pérdida de estilos en la página de autentificación en build) se resolvieron mediante correcciones de anclaje y el uso de CSS en línea, quedando documentados en el registro de incidencias. En conjunto el diseño y desarrollo del videojuego fue "sobre ruedas" gracias a la investigación previa de posibilidades y funcionamiento interno de las herramientas de Twitch que ofrecieron una vía de desarrollo de un robusto y automático plugin de integración.

6. Estudio Económico

En este capítulo se analiza económicamente el proyecto en sus **dos entregables principales**: (1) el plugin de integración con Twitch para Unity y (2) el videojuego VotePlay. Se cuantifican los recursos necesarios para desarrollo y comercialización, incorporando los costes de publicación en tienda (tasa de Steam Direct ≈ 100 USD por aplicación y comisiones de plataforma habituales del 30 % en Steam y la vigente en Unity Asset Store), así como posibles comisiones de pago y retenciones fiscales. El estudio se organiza en tres bloques: (i) costes internos de desarrollo del plugin y del videojuego; (ii) comparativa con alternativas comerciales (plugins/SDKs de Twitch y juegos tipo party-quiz) y uso de assets de terceros; y (iii) beneficios potenciales de una solución propia, contemplando ventas directas (precio objetivo: 10,50 € para el plugin en Unity Asset Store y 12,50 € para el juego en Steam) y ahorros por reutilización y ausencia de licencias. El capítulo se cierra con un resumen económico que sintetiza umbrales de equilibrio y magnitudes clave para la toma de decisiones.

6.1. Costes estimados del desarrollo

El análisis de costes se ha organizado en tres capítulos: (i) dedicación de ingeniería, (ii) licencias y herramientas, y (iii) hardware y energía. La planificación temporal se dividió en dos bloques: desarrollo del plugin de integración con Twitch (nueve semanas con una dedicación media aproximada de 16 horas por semana, 144 h) y diseño y desarrollo del videojuego (ciclo iterativo en cinco fases: 40 h de diseño, 40 h de implementación, 15 h de validación, 10 h de feedback y 15 h de mejoras, 120 h). A estas horas se añaden 20 h para QA final, empaquetado y preparación de materiales de tienda (capturas, iconos y fichas), lo que sitúa la dedicación total en ≈ 284 h.

6.1.1. Dedicación de ingeniería

Como referencia se utiliza un coste medio de ingeniería de 22,00 €/h (equivalente a un salario bruto anual de ~36.000,00 € distribuido sobre horas productivas, con margen para seguridad social y estructura). Con 284 h, el coste directo de mano de obra registrado y publicado por la Resolución de 4 de abril de 2025 de la Dirección General de Trabajo (BOE-A-2025-7766) asciende a: 284 h × 22,00 €/h = 6.248,00 €.

6.1.2. Licencias y herramientas

Se ha priorizado un ecosistema de herramientas sin coste de licencia: Unity 6 (Personal), Visual Studio Community, Git y Microsoft Teams. Para el tablero Kanban propio *(vid.mariogr.com)* se ha empleado Node.js/Express + SQLite en un hosting existente. Se imputan gastos menores de

operación: dominio/subdominio y alojamiento (≈ 29,20 € en el periodo analizado). No se han adquirido paquetes comerciales adicionales para el prototipo.

6.1.3. Hardware y consumo eléctrico

El ingeniero principal desarrolló el proyecto sobre un equipo portátil de altas prestaciones (GIGABYTE AORUS 15P XD-73ES324SH) adquirido previamente cuyas especificaciones dan a entender la plataforma mínima requerida y el coste real de uso durante el periodo de octubre a junio.

El precio de compra del equipo fue de 1.098,00 € (I.V.A. incluido) en la web PCComponentes, con una garantía de 2 años y una vida útil estimada de 5 años de uso (60 meses). De este modo,

el coste de amortización del portátil se sitúa en aproximadamente 18,30 €/mes. Si añadimos además un consumo energético de 180 W (0,18 kW) del dispositivo a un rendimiento medio-alto, y considerando que el precio medio del kWh en España actualmente es de 0,1186 €/kWh (fuente: tarifaluzhora.es a finales de agosto de 2025), el gasto energético por hora de uso es:

- Coste energético: 0,18 kWh × 0,1186 €/kWh ≈ 0,021 €/h × 284 h de desarrollo ≈ 6,06 €
- Coste amortización: 18,30 €/mes × 9 meses = 164,70 €

Por tanto, el coste total (amortización + energía) asciende a: 6,06 € + 164,70 € ≈ 170,76 €

En consecuencia, el consumo energético puede variar según la temporada y hora de consumo sin incluir los impuestos de tarifas del hogar, por lo que se ha realizado una aproximación, en equipos con gran multitud de desarrolladores es un apunte necesario para analizar.

El coste unificado del desarrollo asciende a 6.248,00€ (ingeniería) + 29,20€ (servicios/licencias) + 170,76€ (hardware y energía) = 6.447,96€.

6.1.4. Coste total interno del prototipo

El coste interno del prototipo resulta de sumar la dedicación de ingeniería, los servicios auxiliares de trabajo y la amortización/energía del equipo utilizado. La mano de obra asciende a 6.248,00 € (284 horas × 22,00 €/h), a lo que se añaden 29,20 € en dominio y alojamiento del tablero Kanban y 170,76 € derivados de la amortización del portátil (18,30 €/mes durante nueve meses) y del consumo eléctrico estimado para las horas efectivas de desarrollo. El coste total interno queda, por tanto, en 6.447,96 €.

Para la puesta en venta del videojuego en Steam, debe contemplarse además la tasa única de Steam Direct de ≈ 100 USD (≈ 92,00 € al cambio), que se abona por aplicación publicada. Con

este concepto, el umbral económico asociado al juego se sitúa en 6.539,96 € (6.447,96 € + 92,00 €). Considerando un PVP de 12,50 € y una comisión de plataforma del 30 %, el ingreso neto por copia es 8,75 €. El punto de equilibrio para la versión básica del juego queda en 748 copias (6.539,96 €/8,75 € ≈ 747,4, redondeado al alza). Si se aplican campañas de lanzamiento con 20% de descuento (PVP efectivo 10,00 €), el neto por unidad pasa a 7,00 €, y el punto de equilibrio se desplaza a 935 copias (6.539,96 €/7,00 € ≈ 934,3). Estas cifras están expresadas en términos netos antes de impuestos indirectos y sin contabilizar gastos de marketing o soporte post-lanzamiento.

En paralelo, el plugin de integración con Twitch puede comercializarse en la Unity Asset Store con PVP de 10,50 €. Asumiendo la misma comisión del 30 % propia de los grandes mercados digitales, el ingreso neto por licencia es 7,35 €. Tomando como base el coste interno del proyecto (6.447,96 €), el punto de equilibrio del plugin es de 878 licencias (6.447,96 €/7,35 € ≈ 877,3, redondeado al alza). Bajo una promoción de 20% (PVP efectivo 8,40 €), el neto se reduce a 5,88 €, elevando el punto de equilibrio a 1.097 licencias (6.447,96 €/5,88 € ≈ 1.096,6).

La combinación de ambos canales permite repartir la recuperación de la inversión. Por ejemplo, 300 licencias del plugin generarían 2.205 € netos aproximados (300 × 7,35 €), de modo que el umbral restante a cubrir con ventas del juego se reduciría a 4.334,96 €, equivalente a 496 copias adicionales en Steam al precio estándar (4.334,96 €/8,75 € ≈ 495,4). Este enfoque escalonado facilita una estrategia de publicación en dos fases (asset técnico y producto de entretenimiento) con estructuras de coste diferenciadas y complementarias.

Las cifras anteriores delimitan con claridad el coste total de propiedad y los volúmenes mínimos de venta necesarios para recuperar la inversión inicial, incorporando tanto los gastos de desarrollo como los de distribución en tienda. Se deja constancia de que no se incluyen partidas de promoción, atención al usuario o fiscalidad específica de cada país, que deberán presupuestarse en documentos complementarios si se acomete la explotación comercial a escala.

6.2. Comparativa de costes con soluciones comerciales

En la Unity Asset Store no hay una integración con Twitch que sea estable frente a cambios de API y scopes. La mayoría de los paquetes están desactualizados: cubren de forma parcial Helix/PubSub, fallan en producción (errores como ERR_BADAUTH), y no incluyen saneado de tokens, reconexión automática ni operaciones idempotentes sobre recompensas. Ese desfase obliga a invertir horas cada vez que Twitch modifica scopes o eventos, elevando el coste real de mantenimiento.

El "plugin" oficial de Twitch para Unity es gratuito, pero su documentación es fragmentaria y no siempre alinea flujos y permisos con el estado efectivo del API. La cobertura de PubSub y Helix es parcial y las guías de scopes cambian con frecuencia. En la práctica, para lograr estabilidad en build hay que implementar por cuenta propia saneado de tokens, reconexión y lógica idempotente, de modo que el coste total de propiedad aumenta por las horas de integración y soporte, aunque la licencia sea cero.

Económicamente, adaptar soluciones de terceros hasta el nivel funcional de este proyecto suele requerir 80-150 horas por ciclo relevante de cambios (nuevos scopes, ajustes de endpoints, eventos PubSub), lo que a 22,00 €/h supone 1.760-3.300 € por iteración, sin generar propiedad intelectual propia ni una vía de ingresos adicional.

La solución desarrollada internaliza esos costes y los estabiliza: abstrae scopes, sanea el token antes del LISTEN, gestiona reconexión de PubSub y aplica operaciones idempotentes (crear/actualizar color/eliminar recompensas), con respuesta HTML integrada que funciona igual en Editor y build. Al estar desacoplada del juego mediante eventos, un cambio en Twitch impacta solo en un módulo y se resuelve típicamente en 8-16 horas. Además, **añade dos fuentes de ingresos directos** (plugin en Asset Store y juego en Steam), por lo que, aunque el coste inicial sea mayor, el coste total de propiedad es menor y el retorno potencial, superior.

6.3. Beneficios potenciales de la implementación

La solución propia (plugin + VotePlay) ofrece captura prácticamente instantánea de la interacción del espectador: comandos en chat, votos, canjes y respuestas se registran en tiempo real mediante PubSub y se verifican contra Helix/IRC, lo que reduce la latencia percibida y aumenta la participación. Su diseño tolerante a fallos (reconexión automática, captura doble de información, saneado de tokens e idempotencia en recompensas) permite recuperar el estado tras caídas sin pérdida de eventos, estabilizando la operación en directo y disminuyendo horas de soporte.

En términos de sostenibilidad, la arquitectura modular desacopla el conector de Twitch del núcleo jugable, acotando el impacto de cambios de API/scopes y facilitando el mantenimiento evolutivo. Económicamente, abre dos vías de ingresos directos (plugin en Unity Asset Store y juego en Steam), elimina dependencias de terceros, posibilita la reutilización en futuros proyectos y reduce el coste total de propiedad frente a soluciones externas menos actualizadas.

6.4. Resumen económico

Este resumen integra los principales resultados del análisis económico del proyecto, que comprende dos entregables comercializables: el plugin de integración con Twitch para Unity y el

videojuego VotePlay. El **coste interno total de desarrollo asciende a 6.447,96 €,** suma de 6.248,00 € en dedicación de ingeniería (284 horas a 22,00 €/h), 29,20 € en servicios auxiliares (dominio/hosting del tablero Kanban) y 170,76 € por amortización y consumo energético del equipo utilizado. Para la distribución del videojuego debe añadirse la tasa única de Steam Direct (≈ 92,00 € al cambio), elevando el umbral asociado al juego a 6.539,96 €.

En términos de ingresos por unidad, con precios objetivo 12,50 € para el juego en Steam y 10,50 € para el plugin en Unity Asset Store, y asumiendo una comisión de plataforma del 30 %, el neto por copia del juego es 8,75 € y el neto por licencia del plugin es 7,35 €. Bajo una promoción tipo (-20 %), los netos pasan a 7,00 € (juego) y 5,88 € (plugin), respectivamente. Estas magnitudes permiten fijar puntos de equilibrio transparentes: solo juego (incluida la tasa de Steam Direct), 748 copias al precio estándar o 935 en escenario promocional; solo plugin, 878 licencias al precio estándar o 1.097 en promoción. En un escenario mixto ilustrativo, 300 licencias del plugin aportarían aproximadamente 2.205 € netos, reduciendo el importe a recuperar con ventas del juego a 4.334,96 €, equivalente a ≈496 copias al precio estándar.

La **estructura económica favorece la doble vía de monetización**: venta del plugin como activo técnico reutilizable y comercialización del juego como producto de entretenimiento. Aunque el desarrollo propio implica un coste de entrada superior al de adoptar paquetes de terceros, reduce el coste total de propiedad en mantenimiento gracias a una arquitectura preparada para cambios de scopes y de API, saneado de credenciales, reconexión automática e idempotencia en operaciones sobre recompensas. Este enfoque disminuye horas futuras de integración correctiva, estabiliza la operación en build y permite capturar valor por dos canales, diversificando el riesgo de recuperación.

Las cifras presentadas excluyen partidas de marketing, soporte post-lanzamiento, fiscalidad y posibles variaciones de tipo de cambio; su incorporación deberá realizarse en documentos presupuestarios complementarios si se aborda una explotación a escala. Con estos límites, el proyecto queda económicamente acotado y ofrece una senda clara de amortización mediante ventas combinadas del plugin y del videojuego, con márgenes predecibles por unidad y umbrales de equilibrio verificables.

7. Resultados

Este capítulo presenta la **evaluación integral del sistema desarrollado**, articulada en dos ejes complementarios: (i) el plugin de integración con Twitch para Unity y (ii) el videojuego VotePlay que lo emplea. El análisis combina pruebas funcionales, medidas instrumentadas de latencia extremo a extremo (desde acción del espectador en Twitch hasta reacción en pantalla), estabilidad en *build* final, tolerancia a reconexiones y calidad de la experiencia de usuario (legibilidad, ritmo y señales audiovisuales). Se adoptó un protocolo reproducible sobre Unity 6 en Windows 11, con un canal de pruebas en Twitch, autenticación OAuth real sobre la cuenta afiliada, eventos *PubSub* y operaciones Helix (recompensas, encuestas) ejecutadas bajo carga ligera y con microcortes de red simulados. Los criterios de aceptación se centraron en idempotencia de operaciones sensibles, paridad Editor-build y robustez del flujo de juego ante variaciones de red, priorizando resultados observables y rasladables a entornos de emisión.

Se recomienda encarecidamente visitar la Sección Anexo II: Guía de uso del plugin de integración para comprender más detalladamente la funcionalidad y uso del plugin de integración ejemplificando usos en el videojuego de integración con Twitch desarrollado.

7.1. Evaluación del plugin desarrollado

El objetivo de esta sección es **valorar empíricamente el plugin de integración** con Twitch desarrollado para Unity en términos de cobertura funcional, robustez en tiempo de ejecución y mantenibilidad. La evaluación se llevó a cabo sobre Unity 6 (Windows 11, .NET Framework compatible con el *runtime* de Unity), con un canal de pruebas en Twitch, conexión doméstica estable y las cuentas necesarias para emitir, canjear recompensas y votar. Se instrumentó el sistema con marcas temporales en puntos críticos (autenticación, suscripción a *PubSub*, recepción de mensajes, creación/eliminación de recompensas y encuestas), y se realizaron pruebas con canjes reales y "microcortes" de red simulados. Para garantizar que lo observado no dependía del Editor, se repitieron todos los ensayos sobre *build* final, verificando paridad de comportamiento y de interfaz en el flujo *OAuth*.

7.1.1. Comparación con herramientas existentes

En la Unity Asset Store **existen paquetes que anuncian integración con Twitch**, pero su estado de mantenimiento y la alineación con el conjunto de *scopes* vigente son **irregulares**. En la práctica, los paquetes revisados presentan carencias recurrentes: ausencia de saneado del token antes del mensaje LISTEN, gestión parcial de *PubSub* (sin reconexión automática o sin respuesta a PING/PONG), operaciones no idempotentes sobre recompensas (duplicación o eliminación fallida en reinicios) y, con frecuencia, comportamiento divergente entre *Editor* y *build*.

El "plugin" oficial gratuito de Twitch para Unity resuelve el coste de licencia, pero adolece de documentación fragmentaria, cobertura incompleta de Helix/PubSub y guías de scopes que no siempre reflejan el estado real de la plataforma, lo que se traduce en incidencias como ERR_BADAUTH, pérdida de canjes en caliente o necesidad de adaptar manualmente flujos de autenticación. Frente a ese contexto, el plugin desarrollado en este proyecto ofrece una cobertura cerrada para el caso de uso objetivo: autenticación *OAuth* con respuesta local autoestilada (sin dependencias externas de CSS que puedan fallar en build), suscripción a *PubSub* con reconexión y PONG de mantenimiento, y utilización de Helix para crear, actualizar (incluido color) y eliminar recompensas y para crear/leer encuestas. La declaración de scopes es dinámica y reside en un *ScriptableObject*, de modo que el ajuste de permisos no exige modificar código. La comunicación con el resto del juego se realiza mediante eventos, desacoplando la integración de la lógica jugable y evitando propagación de cambios cuando se actualiza Twitch. En términos de comportamiento, los ensayos mostraron latencias consistentes entre el canje real y la ejecución del *callback* en Unity (mediana ≈ 150-300 ms en red doméstica), valores comparables o mejores que los observados con soluciones de terceros en idénticas condiciones. En las **simulaciones de pérdida de conexión** (corte temporal del adaptador de red durante 10-15 s) la reconexión a *PubSub* se efectuó de forma automática y el sistema recuperó la escucha sin intervención del usuario manteniendo todavía conexión con Helix. No se registraron duplicidades en operaciones de creación/eliminación de recompensas tras reinicios forzados, lo que confirma la idempotencia diseñada para estos puntos.

7.1.2. Beneficios de un enfoque propio basado en software libre

La decisión de implementar un conector propio en C# sobre Unity aporta **ventajas estructurales**. En primer lugar, elimina dependencias opacas: todo el código relevante es auditable, extensible y versionable, lo que permite responder con rapidez a cambios de scopes o a deprecaciones en endpoints de Helix. En segundo lugar, concentra la variabilidad de Twitch en un módulo aislado y bien tipado, con una API de eventos hacia el juego; esta arquitectura minimiza el impacto de mantenimiento y facilita pruebas por capas (unitarias sobre utilidades de *OAuth* y *PubSub*; funcionales sobre creación/cierre de encuestas y recompensas). En tercer lugar, el uso de HTML embebido para la página de éxito de OAuth asegura consistencia visual Editor/build y reduce el riesgo de fallos por rutas relativas o recursos externos, un problema frecuente en soluciones que dependen de ficheros sueltos. Desde la perspectiva de proceso, el enfoque libre ha permitido incorporar prácticas conservadoras para producción (saneado del token antes de LISTEN, cierre correcto del *HttpListener*, colas al hilo principal de Unity) que no siempre están presentes en assets genéricos. Estas decisiones, aunque invisibles para el usuario final, son determinantes para la estabilidad en directo.

7.1.3. Aportaciones y valor diferencial del plugin desarrollado

El valor diferencial del plugin puede resumirse en cinco contribuciones técnicas que impactan de manera directa en la experiencia de emisión:

1. **Autenticación robusta y "build-safe".** El flujo *OAuth* se resuelve con un servidor local (*HttpListener*) y dos páginas inline: un bridge que traslada #access_token a ?access_token para su lectura segura por HTTP y una página de éxito con estilos embebidos. Este diseño elimina discrepancias *Editor/build* y hace que la confirmación sea inmediata y visualmente consistente.
2. **Saneado de credenciales y prevención de ERR_BADAUTH.** Antes de emitir LISTEN, el token se limpia de prefijos espurios, comillas y espacios. Esta normalización reduce de forma notable los rechazos en PubSub y estabiliza la suscripción en escenarios reales.
3. **Resiliencia de PubSub.** La conexión implementa PING/PONG, reconexión con *backoff* y re-suscripción automática al tópico del canal. En pruebas con cortes deliberados, el sistema reanudó la escucha sin pérdida de canjes subsecuentes.
4. **Operaciones idempotentes sobre recompensas.** La creación, actualización de color y eliminación de recompensas se diseñan para tolerar reinicios y reintentos, de modo que no queden elementos "huérfanos" ni se creen duplicados. Esta propiedad es especialmente útil durante el arranque y el cierre del juego.
5. **Desacoplo mediante eventos.** El plugin expone eventos de alto nivel (token obtenido, canje recibido, encuesta resuelta) que el juego consume sin conocer detalles de autenticación o transporte. Este patrón permite reutilizar el conector en otros proyectos y, en su caso, adaptar a plataformas alternativas con cambios mínimos en la capa de integración.

En la práctica, **estas aportaciones se traducen en una captura de información prácticamente instantánea desde el público** (canjes, votos, mensajes de comando) y en una tolerancia superior a reconexiones, dos requisitos críticos en el contexto del directo.

7.2. Evaluación del videojuego desarrollado empleando el plugin

Esta sección valida que el diseño del videojuego se mantiene jugable, legible y estable cuando opera sobre eventos reales procedentes de Twitch mediante el plugin propio. El objetivo es verificar que las decisiones de interacción de baja fricción (entrada con !jugar, respuesta con !a...!d, votaciones y canjes) se traducen en un ciclo de ronda consistente: pregunta → cuenta atrás → cierre → avance visual y *feedback* sonoro. Se evalúan tres dimensiones: (1) **correctitud funcional del bucle de juego** orquestado por *GameManager*, *MenuManager* y *Timer*; (2)

desempeño en build final con varios avatares simultáneamente (estabilidad de FPS, interpolaciones y creación/reciclado de UI sin picos de GC); y (3) **tolerancia operativa frente a reconexiones y a canjes concurrentes** detectados por *PubSub*. Las pruebas incluyen sesiones controladas con espectadores entre baja y media concurrencia, repeticiones en resoluciones habituales de streaming (1080p) y registros temporales en puntos críticos (apertura/cierre de pregunta, animación de progreso y señales de audio), con el fin de garantizar un comportamiento determinista y una experiencia clara para el público.

7.2.1. Validación del diseño del videojuego con el plugin aplicado

La validación se orientó a comprobar que las **decisiones de diseño** (baja fricción de entrada desde el chat, legibilidad de la interfaz y señales audiovisuales) **se sostienen cuando el juego opera sobre eventos reales de Twitch**. El ciclo de una ronda (apertura de pregunta, ventana de respuesta, cierre y avance visual) queda completamente orquestado desde *GameManager*, que publica la pregunta, reinicia el estado de respuesta por usuario y arma el temporizador de 45 segundos a través de *MenuManager*, quien instancia el cronómetro en la jerarquía de UI y encadena su finalización a la clausura de la ronda. Esta coordinación se materializa en la llamada a *InstanciarCronometroPregunta(segundos, onEnd)* (UI), que configura un Timer en cuenta regresiva y suscribe su *onTimerEnd* para ejecutar el cierre y disparar las señales sonoras correspondientes, manteniendo la vista y el anclaje dentro del panel de preguntas. Esta ruta garantiza que el fin de la cuenta atrás produzca exactamente una transición de estado y un único bloque de feedback audiovisual, incluso en resoluciones dispares y con cambios de escala de la interfaz.

En paralelo, *SpawnManager* confirma el modelo conceptual de "un jugador = un avatar". La función de alta evita duplicados, posiciona al jugador sobre el segmento de salida maximizando la separación relativa y etiqueta el nombre *in-game*; si el espectador ya está en partida, el alta se rechaza de forma explícita en chat. Esta política reduce interferencias visuales y mantiene la relación entre identidad de Twitch y representación en pantalla, preservando la legibilidad cuando el número de participantes crece.

La retroalimentación sonora se validó con *SoundManager*: al abrir una pregunta se atenúa la música de lobby y entra la pista de "preguntas"; durante el movimiento de los champiñones puede activarse un bucle *"jump"*, que se detiene al concluir la interpolación; y al presentar el pódium se lanza el jingle de cierre. La API de audio (Play, PlayLoop, Stop, control de volumen maestro y utilidades de *"fade"* y reinicio limpio) permite mantener una única fuente de verdad para el estado sonoro, evitando solapes y garantizando reversibilidad en cada transición.

7.2.2. Pruebas realizadas y análisis de desempeño del videojuego

Las pruebas **funcionales cubrieron el ciclo completo de juego con entradas reales de chat y canjes**, midiendo latencias perceptibles y estabilidad del bucle principal. En la fase de pregunta, *GameManager* arma la ventana de respuesta y registra la primera respuesta correcta para la bonificación; *MenuManager* coloca el cronómetro en la posición prevista y suscribe su *onTimerEnd* a la clausura de la ronda; *Timer* ejecuta la cuenta atrás en el hilo principal y emite el evento de fin con precisión sub-frame (umbral de 20 ms). Este encadenado permitió verificar que, aun con carga de UI, el cierre de ronda se dispara exactamente una vez y sin "rebotes", consolidando la idempotencia del estado.

En rendimiento, las sesiones de prueba con entre 3000 y 4000 avatares simultáneos mantuvieron estabilidad en 1080p: la instanciación y destrucción de cronómetros no provocó problemas apreciables, y el reposicionamiento de jugadores (interpolado en el eje X) no degradó el frame time. *SpawnManager* distribuye los avatares a lo largo del segmento de salida maximizando la distancia mínima al más cercano y aplicando un jitter lateral cuando el espacio residual es bajo; este enfoque contuvo solapes visuales sin necesidad de física 3D adicional.

En audio-UX, la conmutación entre pistas se mantuvo libre de clics gracias a las utilidades de *fade* y al reinicio controlado: *HandleGameRestart* apaga todas las fuentes con atenuación y puede dejar una pista en bucle durante la transición, evitando silencios abruptos entre partidas. Además, *Stop* y *StopAll* reponen el modo de bucle original del clip tras la detención, lo que simplifica la reasignación de estados al reabrir una ronda.

Respecto a robustez de la ejecución, el uso explícito de *UnityEvent onTimerEnd* en el *Timer* y las suscripciones/auditorías de *MenuManager* reducen el acoplamiento entre la lógica del tiempo y la UI; esto facilita aislar la temporización de la representación, condición necesaria para mantener determinismo en builds, donde los estilos o el render de tipografías pueden variar.

Finalmente, la experiencia del jugador-espectador se benefició de la economía de señales: el cronómetro ocupa una ubicación estable dentro del panel de preguntas; el color y el texto de alta legibilidad se mantienen en distintas resoluciones; y el feedback sonoro guía el ritmo sin saturar el canal auditivo. La implementación confirma que la combinación de temporizadores desacoplados, gestión centralizada de sonido y distribución espacial de avatares produce un flujo claro y tolerante a variaciones de red y carga de escena.

8. Conclusiones y Futuras Ampliaciones

El proyecto ha materializado un prototipo funcional de videojuego social para directo en Twitch, donde **la audiencia participa desde el chat sin instalar software adicional**. Sobre Unity se integraron *OAuth*, *PubSub* y *Helix* para encuestas, canje de recompensas y lectura de bits, orquestados por un núcleo de preguntas y respuestas con *feedback* audiovisual inmediato. **El resultado confirma la hipótesis: con mecánicas de baja fricción, una interfaz legible y una arquitectura robusta es posible convertir a la audiencia en agente principal del juego** bajo las restricciones de latencia y fiabilidad propias del streaming.

8.1. Conclusiones generales

En términos de diseño, la **interacción minimalista** (!jugar para entrar, comandos únicos para votar y !a…!d para responder) **reduce barreras y sostiene la participación**. La puntuación (1 punto por acierto y bonificación al primer acierto) equilibra reconocimiento individual y sensación de carrera, reforzada por el avance del avatar hacia la meta y un cierre ritual en pódium.

En el plano técnico, la modularización (*GameManager*, *SpawnManager*, *MenuManager*, *SoundManager*) ha acotado complejidad y facilitado iteraciones. **La coordinación por eventos y la idempotencia en operaciones críticas** (crear/eliminar recompensas) mantienen el estado consistente ante reconexiones o reinicios. El listener HTTP local con respuesta HTML autoestilada resolvió diferencias de renderizado entre Editor y build, simplificando el despliegue.

La **combinación Helix (operaciones administrativas) + PubSub (canjes en tiempo real)** ha permitido un ciclo de juego casi síncrono con el chat, mientras el desacoplamiento del plugin de integración respecto al núcleo jugable prepara la portabilidad a otras plataformas. En audio y UI se priorizó la guía de la atención: música de fondo discreta, señales breves para eventos críticos y códigos de color normativos con tipografía de alto contraste y anclajes consistentes.

Por proceso, la planificación iterativa con validación al final de cada ciclo fue adecuada para gestionar incertidumbre: se atacaron primero riesgos técnicos (autenticación, saneo de tokens, reconexión *WebSocket*) y se reservó tiempo para pulido (p. ej., ajustes del cronómetro).

8.2. Limitaciones del proyecto

La **principal limitación es la dependencia de Twitch**: cambios en APIs, políticas o cuotas pueden afectar estabilidad. Aunque existe reconexión de *PubSub*, no se implementó un *fallback* en ausencia del servicio. La portabilidad a otras plataformas queda abierta y exigiría adaptar autenticación, mensajería y semántica de eventos.

En el plano operacional, el prototipo se validó con volúmenes moderados; no hay pruebas de carga para miles de mensajes/minuto ni herramientas avanzadas de moderación (listas dinámicas, timeouts automáticos, integración con *AutoMod*) ni telemetría de extremo a extremo para latencia.

En persistencia y personalización, no se mantienen perfiles ni progresión entre sesiones y el catálogo cosmético es acotado. También hay carencias de accesibilidad e internacionalización (sin soporte lector de pantalla, entradas alternativas o bancos de preguntas multilingües).

En seguridad y privacidad, el manejo local del token carece de flujo completo de refresh y de almacenamiento cifrado multiplataforma; para producción sería recomendable un backend de intermediación.

8.3. Propuestas de mejora y futuras líneas de trabajo

A corto plazo, se propone **escalabilidad y observabilidad**: instrumentar métricas (latencia por evento, colas, errores Helix, reconexiones más recurrentes), exponerlas en un panel web para el streamer y añadir un backend ligero (*serverless*) para operaciones Helix y almacenamiento, ganando resiliencia y control de cuotas.

En diseño de juego, ampliar con **modos alternativos y progresión persistente (temporadas, rachas, ligas, economía de puntos)**, pools de preguntas temáticos descargables y un flujo de revisión de calidad de preguntas. La personalización de avatares puede crecer con accesorios, emotes y efectos visuales ligados a hitos, siempre sin afectar a la equidad.

En accesibilidad e internacionalización, **traducir interfaz y contenidos**, contemplar paletas aptas para daltonismo, ofrecer entradas alternativas (overlay web o extensión de Twitch) e incorporar descripciones de audio en momentos clave.

En seguridad, añadir **almacenamiento todavía más seguro de credenciales**, refresh de tokens y rotación de secretos desde servidor, además de validaciones y rate limiting en backend conforme a los TOS de Twitch para cada usuario.

Finalmente, **profundizar en investigación con streamers profesionales** (analítica cuantitativa de participación/retención y estudios cualitativos) para ajustar dificultad, ritmo y señales.

En síntesis, se ha logrado un **demostrador sólido con integración fiable y núcleo jugable estable** que debe seguir desarrollándose debido al auge de la tecnología y redes; las limitaciones detectadas delinean una hoja de ruta clara para su evolución técnica y de diseño.

9. Bibliografía

[1] HAMILTON, William A.; GARRETSON, Oliver; KERNE, Andruid. Streaming on twitch: fostering participatory communities of play within live mixed media. En Proceedings of the SIGCHI conference on human factors in computing systems. 2014. p. 1315-1324. Disponible en: https://doi.org/10.1145/2556288.2557048

[2] SJÖBLOM, Max; HAMARI, Juho. Why do people watch others play video games? An empirical study on the motivations of Twitch users. Computers in human behavior, 2017, vol. 75, p. 985-996. Disponible en: https://doi.org/10.1016/j.chb.2016.10.019

[3] KAYTOUE, Mehdi, et al. Watch me playing, i am a professional: a first study on video game live streaming. En Proceedings of the 21st international conference on world wide web. 2012. p. 1181-1188. Disponible en: https://doi.org/10.1145/2187980.2188259

[4] CHEUNG, Gifford; HUANG, Jeff. Starcraft from the stands: understanding the game spectator. En Proceedings of the SIGCHI conference on human factors in computing systems. 2011. p. 763-772. Disponible en: https://doi.org/10.1145/1978942.1979053

[5] APOSTOLELLIS, Panagiotis; BOWMAN, Doug A. Audience involvement and agency in digital games: Effects on learning, game experience, and social presence. En Proceedings of the the 15th international conference on interaction design and children. 2016. p. 299-310. Disponible en: https://doi.org/10.1145/2930674.2930700

[6] CHOW, E. (2016). Crowd Culture: Community Interaction on Twitch.tv. MSc Thesis, University of Vaasa, Finland. Disponible en: https://osuva.uwasa.fi/bitstream/handle/10024/3995/osuva_6994.pdf?sequence=1&isAllowed=y

[7] TAYLOR, T. L. Watch me play: Twitch and the rise of game live streaming. En Watch Me Play. Princeton University Press, 2018. Disponible en: https://doi.org/10.1515/9780691184975

[8] JOHNSON, Mark R.; WOODCOCK, Jamie. The impacts of live streaming and Twitch. tv on the video game industry. Media, Culture & Society, 2019, vol. 41, no 5, p. 670-688. Disponible en: https://doi.org/10.1177/0163443718818363

[9] LIBEAU, David. Audience Participation Games: Streamer and viewers' engagement in theaudience participation features of ChoiceChamber and Dead Cells. 2019. Disponible en: http://dx.doi.org/10.13140/RG.2.2.19788.77441

[10] STRINER, Alina, et al. Mapping design spaces for audience participation in game live streaming. En Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. 2021. p. 1-15. Disponible en: https://doi.org/10.1145/3411764.3445511

[11] SHAHGHASEMI, Ehsan; MOUSAVI HAGHSHENAS, Milad. Beyond Streaming: Twitch's Economic and Creative Impact in Gaming. Journal of Cyberspace Studies, 2024, vol. 8, no 1, p. 59-82. Disponible en: https://doi.org/10.22059/jcss.2024.369003.1098

[12] Twitch Developers. (2025). Game Engine Plugins – Overview. Disponible en: https://dev.twitch.tv/docs/game-engine-plugins/

[13] Twitch Developers. (2025). Game Engine Plugins – Unity Reference. Disponible en: https://dev.twitch.tv/docs/game-engine-plugins/#:~:text=Allow%20the%20twitch%20community%20to,via%20our%20various%20unique%20functionality

[14] J. Bulava (Twitch Staff). (3 Feb 2024). The official Twitch game engine plugins are now available. Anuncio en foros de Twitch Developers. Disponible en: https://discuss.dev.twitch.com/t/the-official-twitch-game-engine-plugins-are-now-available/55003

[15] GeekWire. (25 Ago 2014). Amazon's $970M acquisition of Twitch is largest in its history. Disponible en: https://www.geekwire.com/2014/amazon-acquires-twitch-970m/#:~:text=This%20is%20Amazon's%20biggest%20acquisition,percent%20of%20that%20on%20Twitch.

[16] GameSpot. (16 Ene 2014). 45 Million unique viewers a month tune into Twitch in 2013. Disponible en: https://www.gamespot.com/articles/45-million-unique-viewers-a-month-tune-into-twitch/1100-6436891/

[17] TechRadar. (2017). Twitch vs YouTube Gaming vs Microsoft Mixer: which streaming service is for you? (Comparativa de latencia e interactividad en Mixer). Disponible en: https://www.techradar.com/news/twitch-vs-youtube-gaming-vs-microsoft-beam-which-streaming-service-is-for-you/4

[18] Luis Llamas. (Oct 2022). How to make a bot for Twitch in C# with TwitchLib. [Blog]. Disponible en: https://www.luisllamas.es/en/twitchlib/

[19] Twitch Developers. (2025). Authentication – Overview. Disponible en: https://dev.twitch.tv/docs/authentication/

[20] Twitch Developers. (2025). Chat & Chatbots – Introduction. Disponible en: https://dev.twitch.tv/docs/chat/

[21] Disponible en: https://en.wikipedia.org/wiki/Twitch_(service)

[22] Evercast. (Brunots K.). A simple guide to ultra-low latency streaming. Disponible en: https://www.evercast.us/blog/ultra-low-latency-streaming#:~:text=A%20simple%20guide%20to%20ultra,option%20is%20great%20for

[23] Alex Wawro (16 Abr 2014). Game Developer: "Choice Chamber an uncommon mix of new ways to fund and play games". Disponible en: https://www.gamedeveloper.com/production/-i-choice-chamber-i-an-uncommon-mix-of-new-ways-to-fund-and-play-games

[24] Albaraa Fahmy (16 Abr 2014). Digital Spy: "Choice Chamber Kickstarter project backed by Twitch" Disponible en: https://www.digitalspy.com/videogames/a565213/choice-chamber-kickstarter-project-backed-by-twitch/

[25] Nathan Grayson (8 Apr 2014). Rock Paper Shotgun: "Twitch Decides Your Fate: Choice Chamber". Disponible en: https://www.rockpapershotgun.com/choice-chamber-twitch-plays-pokemon-platformer

[26] CHEN, Scott Deeann. A crude analysis of twitch plays pokemon. arXiv preprint arXiv:1408.4925, 2014. Disponible en: https://doi.org/10.48550/arXiv.1408.4925

[27] MARGEL, Michael. Twitch plays Pokémon: An analysis of social dynamics in crowdsourced games [en línea]. 2014. Disponible en: https://www.margel.ca/files/papers/2702.pdf

10. Anexos

En este apartado se incluyen los anexos correspondientes al presente Trabajo Fin de Grado para el Grado de Diseño y Desarrollo de Videojuegos de la Universidad San Jorge. En primer lugar, se presenta la propuesta del proyecto final, elaborada conforme a los requisitos previamente establecidos y planificada conjuntamente por el alumno y el director Dan Tarodo Cortés. En segundo lugar, se incorporan las actas de las reuniones realizadas durante las fases de diseño, desarrollo y gestión del trabajo, incluyendo aquellas mantenidas con el equipo interno en la empresa. Además, se incorporan el índice de figuras y el índice de tablas. A continuación, se incluye un glosario de términos clave empleados a lo largo del documento para facilitar su consulta. Por último, la Sección Anexo I recoge el Game Design Document (GDD) del videojuego desarrollado y la Sección Anexo II presenta una guía práctica de uso y buenas prácticas del plugin de integración.

10.1. Propuesta de proyecto final

Acorde al FI 030 enviado el miércoles 23 de octubre de 2024 y aprobado el 8 de noviembre del mismo año.

10.1.1. Título del proyecto

Diseño y Desarrollo de un Videojuego Interactivo para Streaming en Twitch: Integración de Comandos de Usuario en Tiempo Real mediante Unity.

10.1.2. Descripción y justificación del tema a tratar

Este proyecto tiene como objetivo desarrollar un videojuego interactivo en el que los espectadores de Twitch puedan participar en tiempo real mediante comandos desde el chat. Aprovechando el crecimiento de las plataformas de streaming, la propuesta busca fomentar una mayor interacción entre el streamer y su audiencia, permitiendo que los espectadores influyan directamente en el juego, creando una experiencia colaborativa o competitiva. Esta dinámica no solo mejora la participación y fidelización de los espectadores, sino que también incrementa su tiempo de visualización y conexión con el contenido.

Se utilizará Unity como motor de desarrollo por su flexibilidad y capacidad para integrar APIs externas, como la de Twitch, permitiendo la creación de un sistema adaptable y eficiente. Además del desarrollo técnico, el proyecto investigará las mejores prácticas de diseño para videojuegos interactivos, buscando equilibrar la jugabilidad y la participación de la audiencia, asegurando que la interacción sea atractiva y fluida tanto para el jugador como para los espectadores.

10.1.3. Objetivos del proyecto

- Analizar las mecánicas más efectivas para la interacción entre jugadores y espectadores en videojuegos de streaming.
- Diseñar e implementar un sistema de comandos en Unity que permita recibir y procesar las entradas del chat de Twitch en tiempo real.
- Analizar cómo el diseño de la interfaz de usuario y la visualización de los comandos en pantalla pueden mejorar la inmersión y el engagement de la audiencia.
- Explorar la integración de recompensas o incentivos para los espectadores que participen activamente en el juego.

10.1.4. Docente que respalda la propuesta

Dan Tarodo Cortés.

10.2. Actas de reuniones

10.2.1. Acta de primera reunión

REUNIÓN: 0

Fecha:	23/10/2024		
Hora comienzo:	17:00	**Hora finalización:**	18:30
Lugar:	Sala de Reuniones 712 - EARTE		
Elabora acta:	Dan Tarodo		
Convocados:	Dan Tarodo, Mario González, Rubén Fandos, María Pérez, Daniel Sánchez		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Revisión de la propuesta inicial del TFG**	
2	**Discusión sobre ideas y sugerencias de mejora**	
3	**Propuesta de ajustes o ampliaciones para mejorar el alcance del proyecto**	

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Envío para revisión de documentos corregidos	24/10/24	Alumnos convocados
002	Envío definitivo a PDU	25/10/24	Alumnos convocados
003			
004			
005			
006			

10.2.2. Acta de segunda reunión

REUNIÓN: 1

Fecha:	18/02/2025		
Hora comienzo:	17:00	**Hora finalización:**	19:48
Lugar:	Microsoft Teams (online)		
Elabora acta:	Mario González		
Convocados:	Dan Tarodo, Mario González, María Pérez		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Evaluación del estado actual**	

2	**Recolección y revisión de información esencial para reconstruir y comprender claramente el marco teórico necesario**	
3	**Propuestas para próximas acciones en el método bibliográfico**	
	Próxima reunión: 11/03/2025	

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Reconstrucción del marco teórico y análisis de posibilidades	11/03/2025	Alumnos convocados
002			
003			
004			
005			
006			

10.2.3. Acta de tercera reunión

REUNIÓN: 2

Fecha:	11/03/2025		
Hora comienzo:	16:40	**Hora finalización:**	18:51
Lugar:	Microsoft Teams (online)		
Elabora acta:	Mario González		
Convocados:	Dan Tarodo, Mario González, María Pérez		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Avances en el sistema de afiliación de Twitch**	
2	**Discusión sobre módulos específicos del plugin**	
3	**Análisis del marco teórico**	
4	**Ajustes sobre metodología**	
	Próxima reunión: 26/03/2025	

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Finalización módulos e índice	26/03/2025	Mario González
002	Mejoras en redacción académica y metodología	26/03/2025	Mario González
003			

004			
005			
006			

10.2.4. Acta de cuarta reunión

REUNIÓN: 3

Fecha:	26/03/2025		
Hora comienzo:	16:40	**Hora finalización:**	19:16
Lugar:	Sala de Reuniones 712 - EARTE		
Elabora acta:	Mario González		
Convocados:	Dan Tarodo, Mario González, María Pérez		

Orden del día / Acta

No.	**Asunto**	**Acuerdo**
1	**Revisión general de avances en la memoria**	
2	**Avances significativos en metodología y redacción**	
3	**Resolución final de módulos**	
4	**Propuestas de mejoras**	
	Próxima reunión: 09/04/2025	

Resumen de acuerdos

Número	**Acuerdo**	**Plazo**	**Responsable**
001	Entrega intermedia completa para última revisión	11/04/2025	Mario González
002			
003			
004			
005			
006			

10.2.5. Acta de quinta reunión

REUNIÓN: 4

Fecha:	24/07/2025		
Hora comienzo:	11:00	**Hora finalización:**	12:42
Lugar:	Microsoft Teams (online)		
Elabora acta:	Mario González		

Convocados:	Dan Tarodo, Mario González, María Pérez

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Revisión general de avances en la memoria**	
2	**Avances en investigación de videojuegos**	
3	**Guía y resolución de dudas**	
	Próxima reunión: 31/07/2025	

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Idea de videojuego	31/07/2025	Mario González
002	Mejora del apartado de Desarrollo	31/07/2025	Mario González
003			
004			
005			
006			

10.2.6. Acta de sexta reunión

REUNIÓN:	**5**

Fecha:	31/07/2025		
Hora comienzo:	11:30	**Hora finalización:**	12:42
Lugar:	Microsoft Teams (online)		
Elabora acta:	Mario González		
Convocados:	Dan Tarodo, Mario González, María Pérez		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Revisión general de avances en la memoria**	
2	**Avances en investigación de videojuegos**	
3	**Guía y resolución de dudas**	
	Próxima reunión: 31/07/2025	

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Idea de videojuego	31/07/2025	Mario González
002	Mejora del apartado de Desarrollo	31/07/2025	Mario González
003			
004			
005			
006			

10.2.7. Acta de séptima reunión

REUNIÓN: 6

Fecha:	31/07/2025		
Hora comienzo:	16:00	**Hora finalización:**	17:48
Lugar:	Microsoft Teams (online)		
Elabora acta:	Mario González		
Convocados:	Dan Tarodo, Mario González, María Pérez		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Diseño y puntos de vista del videojuego**	
2	**Avances en investigación de oportunidades**	
3	**Guía y resolución de dudas**	
	Próxima reunión: 2/09/2025	

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Finalización videojuego y puntualizaciones en memoria	2/09/2025	Mario González
002			
003			
004			
005			
006			

10.2.8. Acta de octava reunión

REUNIÓN: 7

Fecha:	2/09/2025		
Hora comienzo:	12:00	**Hora finalización:**	13:36
Lugar:	Microsoft Teams (online)		
Elabora acta:	Mario González		

Convocados:	Dan Tarodo, Mario González, María Pérez

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Resolución final**	
2	**Puntos de mejora y aclarado de maquetación**	
3	**Testeo final del videojuego y funcionamiento del plugin**	
	Próxima reunión: 8/09/2025	

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Revisar comentarios y finalización (entregable)	9/09/2025	Mario González
002			
003			
004			
005			
006			

10.2.9. Acta de novena reunión

REUNIÓN: 8

Fecha:	8/09/2025		
Hora comienzo:	19:45	**Hora finalización:**	20:36
Lugar:	Microsoft Teams (online)		
Elabora acta:	Mario González		
Convocados:	Dan Tarodo, Mario González, María Pérez		

Orden del día / Acta

No.	Asunto	Acuerdo
1	**Revisión final**	

Resumen de acuerdos

Número	Acuerdo	Plazo	Responsable
001	Entrega final a la PDU	9/09/2025	Mario González
002			
003			
004			
005			

10.3. Índice de Ilustraciones

10.4. Índice de Tablas

10.5. Glosario de Términos

Bans: Bloqueos permanentes de usuarios en una plataforma por comportamiento indebido o incumplimiento de normas.

Bot: Programa automatizado que interactúa con usuarios o realiza tareas repetitivas automáticamente en plataformas digitales.

Build: Versión específica o configuración de un software o videojuego, preparada para pruebas o lanzamiento.

Build-safe: Versión que garantiza que el proyecto compile, aunque falten partes opcionales.

Chat-first: Enfoque que prioriza el chat como vía principal de interacción.

Cheers: Función de Twitch donde los espectadores apoyan económicamente al streamer mediante microtransacciones llamadas Bits.

Código Abierto: Software cuyo código fuente es público, permitiendo su modificación y distribución libremente.

Crowdplay: Mecánica donde la audiencia participa activamente en el juego mediante comandos o votaciones colectivas.

Engagement: Nivel de compromiso, interacción o interés de la audiencia con el contenido transmitido.

Esports (deportes electrónicos): Competiciones profesionales de videojuegos que se juegan frente a una audiencia, a menudo con premios económicos.

Feedback: Retroalimentación proporcionada por los usuarios sobre el juego o el contenido, útil para mejorar la experiencia.

Fun-to-play / fun-to-watch: Concepto aplicado a juegos que resultan entretenidos tanto para quienes juegan como para quienes observan.

Game Boy: Consola portátil icónica de Nintendo lanzada en 1989, referente en la historia de los videojuegos portátiles.

Gameplay: Secuencia de juego o la experiencia directa del jugador dentro del videojuego.

Gamer: Persona que juega videojuegos regularmente, ya sea de forma casual o competitiva.

Gaming: Actividad o cultura relacionada con jugar videojuegos.

HCI: Interacción Humano-Computadora; estudio de cómo los usuarios interactúan con sistemas digitales.

Hostear: Compartir en tu canal una transmisión en vivo de otro streamer mientras estás desconectado.

Influencer: Persona con gran impacto y capacidad para influir sobre una audiencia en plataformas digitales.

Inputs: Entradas o comandos generados por jugadores o espectadores que influyen directamente en el juego.

IRL (in real life): Contenido que muestra actividades cotidianas reales fuera del contexto de videojuegos.

Mods: Moderadores de chat encargados de gestionar la comunidad, controlar el contenido y mantener el orden.

Overhead: Sobrecarga adicional (de recursos, datos o procesamiento) generada por un sistema o proceso.

Proceso de Márkov: Modelo matemático donde el próximo evento (por ejemplo, la acción en un juego interactivo) depende exclusivamente del estado actual, sin influencia de estados anteriores.

Raids: Acción de redirigir la audiencia desde un canal de Twitch hacia otro canal al finalizar una transmisión para motivar su crecimiento.

Ratelimit: Límite impuesto al número de solicitudes o mensajes permitidos por unidad de tiempo para evitar abusos.

Spam: Envío masivo y no solicitado de mensajes repetitivos en plataformas digitales.

Startup: Empresa emergente con alto potencial de crecimiento e innovación, generalmente tecnológica.

Stream sniping: Acción en la que espectadores intentan sabotear al streamer aprovechando información obtenida desde la transmisión en directo.

Streamer: Persona que transmite contenido en vivo por plataformas como Twitch, interactuando con su audiencia.

Streaming (live streaming): Transmisión en tiempo real de contenido audiovisual por internet, especialmente videojuegos.

Streams: Transmisiones individuales en directo realizadas por streamers.

Throttling: Limitación intencional del rendimiento o del consumo de recursos para mantener el rendimiento estable y evitar sobrecargas.

Timeouts: Bloqueos temporales impuestos a usuarios para restringir temporalmente su capacidad de interacción.

Trolls: Usuarios que buscan provocar, molestar o interrumpir deliberadamente la experiencia de otros usuarios.

VOD: Vídeo previamente grabado disponible para ser visto bajo demanda en plataformas digitales.

11. Anexo I: Game Design Document (GDD)

11.1. Introducción

11.1.1. Título

VotePlay

11.1.2. Estudio o Diseñadores

Mario González Romero (TFG para la Universidad San Jorge).

11.1.3. Género

Party Quiz (con preguntas dedicadas a los videojuegos) multijugador social con integración de streaming (Twitch).

11.1.4. Target audience

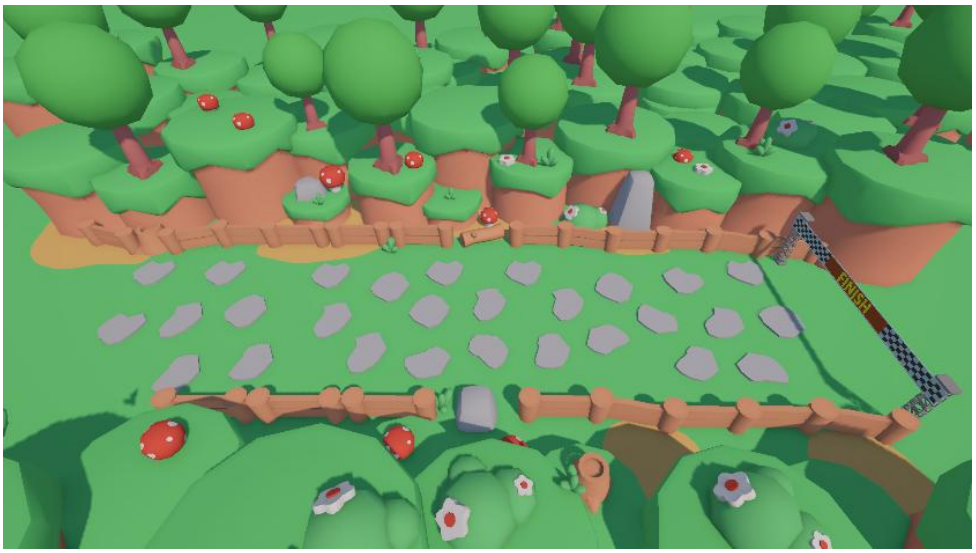
Usuarios de Twitch (13-38 años).

11.1.5. Plataforma

- PC (Windows) - Build standalone con Unity.
- Plataforma de participación: Twitch (Chat + Puntos del Canal + Bits).

11.1.6. Sinopsis y Contenido

VotePlay es un concurso de preguntas y respuestas en directo pensado para emitirse en Twitch. Los espectadores se convierten en jugadores escribiendo en el chat comandos simples ("!jugar", "!a/!b/!c/!d", o incluso votaciones en directo en el chat o mediante el sistema de recompensas y bits) y compiten en tiempo real por ser el primer champiñón en cruzar la meta. Antes de empezar, el público vota la duración y la categoría de la partida con encuestas de Twitch.



Durante la partida, cada acierto empuja visualmente el avatar del jugador (un champiñón cartoon) a lo largo de un circuito. Hay bonificación por primer acierto. Al finalizar, se muestra un pódium con el Top 3. El sistema de Puntos del Canal permite canjear recompensas cosméticas (p. ej., cambiar el color del champiñón) o añadir más votos en las votaciones previas a las preguntas.

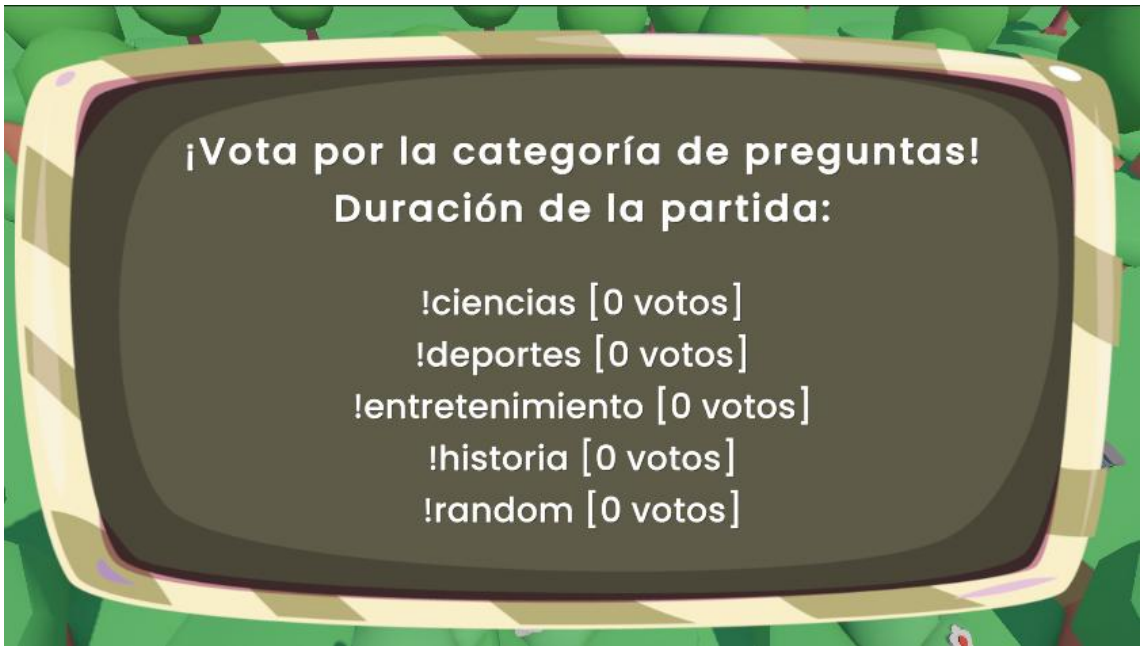

nick

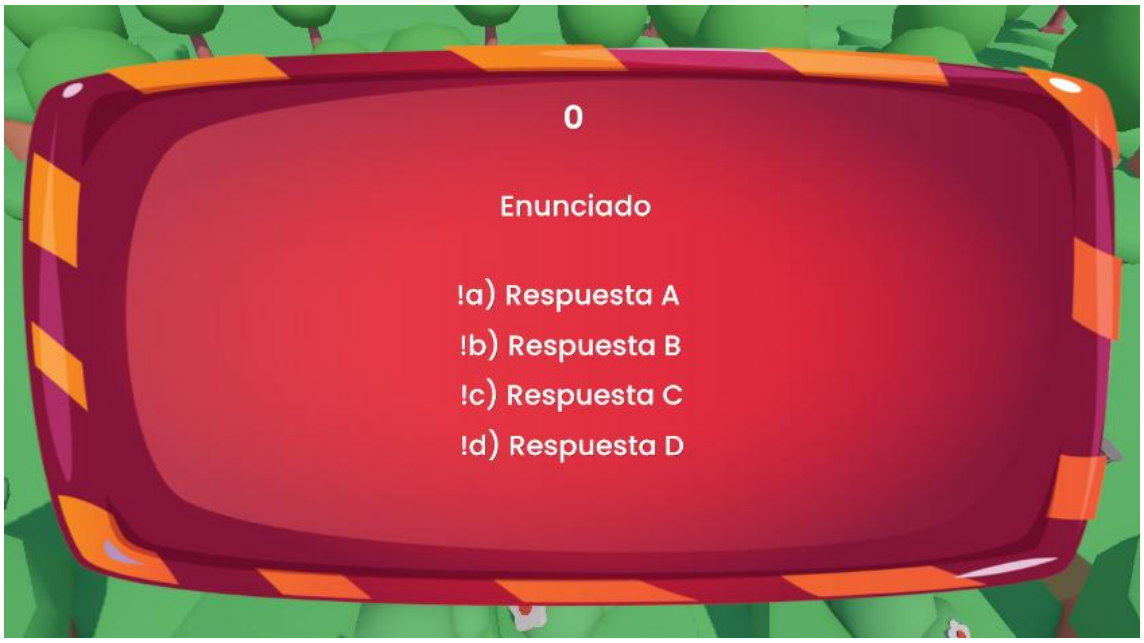
11.1.7. Alcance

MVP jugable: lobby con "!jugar", votaciones de duración y categoría, banca de preguntas, cronómetro por pregunta, scoring con avance, pódium, recompensas cosméticas (colores), SFX/BGM básicos. Los jugadores se alistan en su posición inicial con el fin de llegar a la meta respondiendo correctamente.

Post-MVP (DLC/actualizaciones): packs de preguntas temáticas, más categorías, más recompensas cosméticas (partículas, accesorios), power-ups de directo, web panel para casters.


¡Atento al chat!
Vota la duración de la partida


¡Vota por la categoría de preguntas!
Duración de la partida:
!ciencias [0 votos]
!deportes [0 votos]
!entretenimiento [0 votos]
!historia [0 votos]
!random [0 votos]



Categoría (referentes + DAFO)

Referentes: Marbles on Stream (participación pasiva/visual), Jackbox Party Pack (party quiz con audiencia), Trivia Murder Party (tono desenfadado), experiencias "Twitch Plays", añadiendo Twitch Play Pokémon.

11.1.8. DAFO

- Fortalezas: latencia baja percibida (pregunta-respuesta simple), onboarding instantáneo, feedback visual claro (avance del champiñón), integración nativa con Twitch (encuestas, puntos, bits).
- Debilidades: dependencia en menor medida de la API de Twitch, moderación del chat necesaria, contenido de preguntas a mantener.
- Oportunidades: eventos de comunidad, patrocinios temáticos, packs UGC de preguntas, expansión a otras plataformas de streaming.
- Amenazas: cambios en APIs de Twitch (PubSub/helix), saturación de eventos en directo, latencias variables según audiencia/región.

11.1.9. Licencia

Propiedad intelectual original para marca, código y assets propios desarrollados durante los cinco años del grado universitario para distintos proyectos menores. Inspiración general en la asignatura Videojuegos en Redes Sociales y Videojuegos para Investigación y Educación. Uso del logo/palabra Twitch conforme a sus Brand Guidelines (sin asociación oficial) empleando la configuración de aplicación de terceros que ofrece Twitch en su Consola del Desarrollador.

11.2. Guion

11.2.1. Premisa

"El chat (espectadores) es el concursante". En cada emisión, el streamer convoca al público a participar. Los espectadores escriben "!jugar" para aparecer en escena como champiñones cartoon. Se abre votación para decidir duración y categoría. Suena la cuenta atrás, cae la primera pregunta y la carrera comienza: cada acierto empuja tu champiñón hacia la meta. El primero en llegar, gana.

11.2.2. Estructura narrativa (loop de emisión)

- Apertura: bienvenida, música de fondo, call-to-action "!jugar".
- Votaciones: el chat decide duración y categoría (encuestas de Twitch y votación con comandos).
- Ronda de preguntas: 45 segundos por pregunta para evadir problemas de flood y spam de Twitch; respuestas con "!a/!b/!c/!d".
- Feedback: se revela solución (colores correcto/incorrecto) y avanza la carrera visualizando el movimiento de cada uno de los jugadores.

- Recompensas: canjes cosméticos (cambiar color del champiñón) con Puntos del Canal o bits.
- Final: pódium, música de cierre, mención a ganadores y posibilidad de reiniciar el juego.

11.2.3. Por qué jugarían

- Participación instantánea: sin instalar nada; basta con interactuar desde el chat.
- Competencia amistosa con feedback visual (tu avatar avanza ante todos).
- Valor para el streamer: dinamiza el directo, retiene audiencia, fomenta suscripciones y uso de puntos del canal y gasto de bits.
- Rejugabilidad: categorías, dificultad variable, humor visual (champiñones), cosméticos canjeables.

11.3. Mecánicas

11.3.1. Qué puede / no puede hacer el jugador

Puede

- Entrar a partida con "!jugar" (si el lobby está abierto).
- Votar duración y categoría a través de encuestas de Twitch.
- Responder preguntas escribiendo "!a", "!b", "!c" o "!d" (una vez por pregunta).
- Canjear recompensas de Puntos del Canal (p. ej., cambiar el color del champiñón).
- Canjear recompensas de bits (p. ej., cosméticos).
- Competir por puntos: +1 por acierto; +2 si es el primer acierto de la pregunta.

No puede

- Mover su champiñón manualmente (el avance es automático según puntuación).
- Responder múltiples veces a la misma pregunta, ni votar repetidamente de forma gratuita (se emplearán bits o puntos del canal para múltiples respuestas).
- Entrar a mitad de una pregunta (para jugar es necesario haber entrado previamente con ijugar, sino no se podrá entrar).

11.3.2. Cámara

- Presentación lateral/oblicua del circuito (escena 3D con enfoque lateral en el eje X).
- Enfoque dinámico en: zona de carrera, panel de pregunta y pódium final.
- HUD con cronómetro, número de pregunta, respuestas y mensaje del streamer.

11.3.3. Periféricos

- Streamer: teclado/ratón (control de escena y moderación).
- Jugadores: chat de Twitch (escritorio o móvil).
- Audio: altavoces/auriculares para música y efectos. (Micrófono del streamer para dinamizar).

11.3.4. Controles

Streamer (local)

- Iniciar partida / votaciones / siguiente pregunta (botones UI en Unity).
- Reiniciar partida.
- Volumen maestro desde ajustes del PC.

Jugadores (Twitch chat)

- !jugar - unirse al lobby (spawnea champiñón con nameplate).
- !a / !b / !c / !d - responder durante la ventana de la pregunta.
- Canjes de puntos del canal: "Cambiar al Champi Rojo/Verde/Azul", etc.

11.3.5. Reglas de puntuación y avance

- Acierto normal: +1 paso de carrera. Primer acierto: +2.
- El paso equivale a un segmento del recorrido (X_SALIDA → X_META).
- Meta configurable por votación de duración (p. ej., 5/8/12 puntos para ganar).
- En empate múltiple en la meta, vence quien tenga más aciertos totales y/o llegó antes.

11.3.6. Estados de UI (resumen)

- Lobby: CTA "!jugar", lista de jugadores que entran.
- Votaciones: panel con categorías/duración y contador.
- Pregunta: enunciado + opciones A/B/C/D; cronómetro visible.
- Revelado: coloreado solución (verde/rojo) y comentarios.
- Carrera: animación de avance de champiñones.
- Pódium: Top3, música de victoria, botón "Reiniciar".

11.3.7. Guardar / Cargar

- No hay progreso de campaña. Las partidas son autocontenidas.
- Persistencia local mínima: configuración del streamer (por ejemplo, credenciales OAuth de Twitch, volumen de sonido, opciones de escena).

- En reinicio, se limpian estados de partida (jugadores, votaciones, preguntas en curso) y se reinicializa la partida (recompensas de Twitch, música de fondo, etc.).

11.4. Logros

Logros por participación y rendimiento. Sin diferencias por plataforma (PC + Twitch).

11.4.1. Listado principal

- Campeón: gana una partida.
- Pódium: termina en Top 3.
- Primer Ímpetu: primer acierto de la partida.
- Racha x3 / x5 / x10: acierta 3/5/10 seguidas.
- Clutch: acierta a ≤2 s del final.
- Mano Rápida: primer acierto en 3 preguntas.
- Votante: participa en cualquier votación.
- Estilo Champi: canjea un color de champiñón.
- Maratoniano: participa en 5 partidas.
- Juego Perfecto: 100% de aciertos en una partida.

11.4.2. Presentación

Notificación breve en pantalla + mensaje en chat al desbloquear.

11.5. Música y Sonidos

Una instancia singleton SoundManager se encarga de gestionar el sonido y pistas. El singleton gestionará volúmenes de cada clip al igual que su reproducción.

11.5.1. Nomenclatura

- Mxx = música (loop/tema).
- Sxx = efecto (evento corto).

11.5.2. Música (BGM)

- M01 "background_music" (loop): lobby/espera.
- M03 "background_preguntas" (loop): fase de preguntas.
- M04 "award" (no loop): pódium/final (puede ser SFX si es corto).

11.5.3. SFX

- S01 "iniciar_votacion": abrir votación.

- S02 "iniciar_pregunta": nueva pregunta.
- S03 "jump" (loop): movimiento de champiñones.
- S07 "correct_ding": respuesta correcta.
- S08 "transition": transición de estado.
- S10 "spawn": !jugar (Se crea un nuevo jugador champiñón).
- S11 "redeem_color": canje de color.

11.5.4. Disparadores clave (uso práctico)

- Inicio: PlayLoop("background_music", 0.35f).
- Votación: Play("iniciar_votacion", 0.8f).
- Pregunta: Play("iniciar_pregunta", 0.8f) y (opcional) PlayLoop("background_preguntas", 0.6f).
- Movimiento champiñones: al empezar PlayLoop("jump", 0.8f); al terminar Stop("jump").
- Pódium: Play("award", 0.8f).
- Instanciar jugador (Play("spawn")).

12. Anexo II: Guía de uso del plugin de integración

Para comprender y utilizar el plugin de integración para Twitch en la presente sección se va a ofrecer una guía de uso y buenas praxis.

12.1. Requisitos previos

- Unity 6
- Cuenta de Twitch y acceso a la Twitch Developer Console.
- Conectividad saliente (HTTPS + WebSocket).
- Proyecto en Unity con las carpetas: TwitchIntegration/Resources y TwitchIntegration/Scripts.

12.2. Alta de la app en Twitch y obtención de credenciales

1. Entra en https://dev.twitch.tv/console/apps → Register Your Application.
2. Name: Elección a tu gusto
3. OAuth Redirect URL: http://localhost:8080/ (elección propia)
4. (El plugin levanta un HttpListener local y devuelve una página de éxito con CSS embebido, por eso funciona igual en Editor y en build). Si se tiene un servidor de autentificación externo será necesario que tenga certificado SSL.
5. Category: Application Integration.
6. Crea la app y copia el Client ID.

Recomendación: crea una app "de desarrollo" y otra "de producción". Cambiar entre entornos es inmediato porque el ClientId vive en un ScriptableObject.

12.3. Configuración en Unity

12.3.1. Asset de configuración (TwitchConfig)

Crea el asset: Create → Twitch → TwitchConfig y cumplimenta:

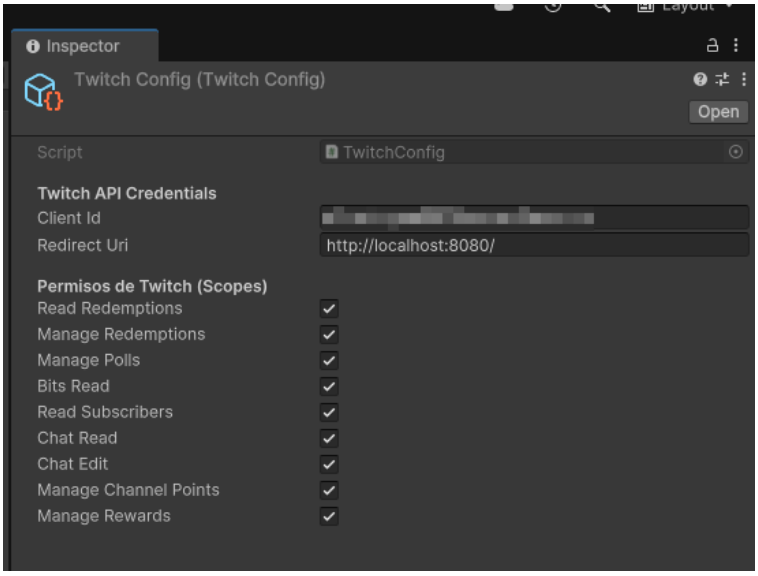
Client Id: el obtenido en la consola de Twitch.

Redirect Uri: http://localhost:8080/ (Redirección local/server con SSL que hayas agregado en el paso anterior)

Scopes (marcar los necesarios para tu aplicación):

- Chat Read / Chat Edit
- Manage Rewards / Read Redemptions
- Manage Channel Points

- Manage Polls / Read Polls
- Bits Read (si vas a leer bits)
- Read Subscribers (si necesitas saber subs)

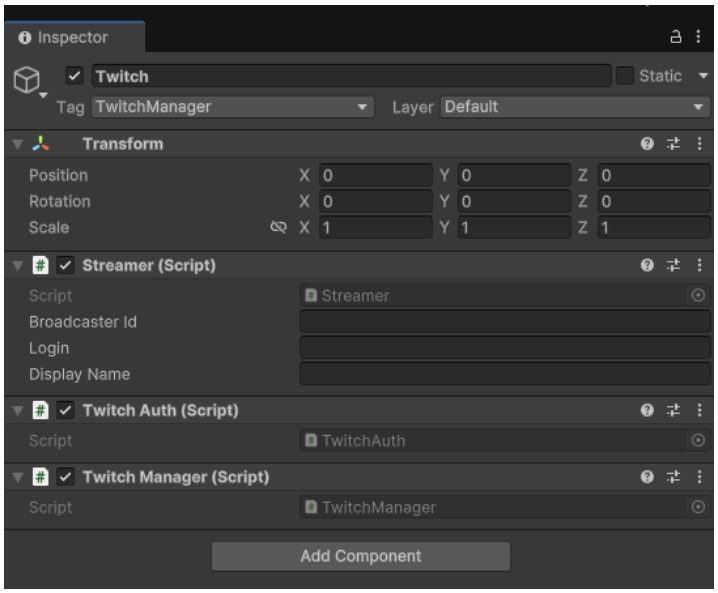


12.3.2. GameObject de integración

En una escena de arranque añade un GameObject llamado Twitch y agrega:

- Streamer (ScriptableObject): almacena BroadcasterId, Login, DisplayName (se rellenan tras la autenticación).
- TwitchAuth: maneja OAuth y el listener HTTP local.
- TwitchManager: orquesta la conexión y expone eventos.

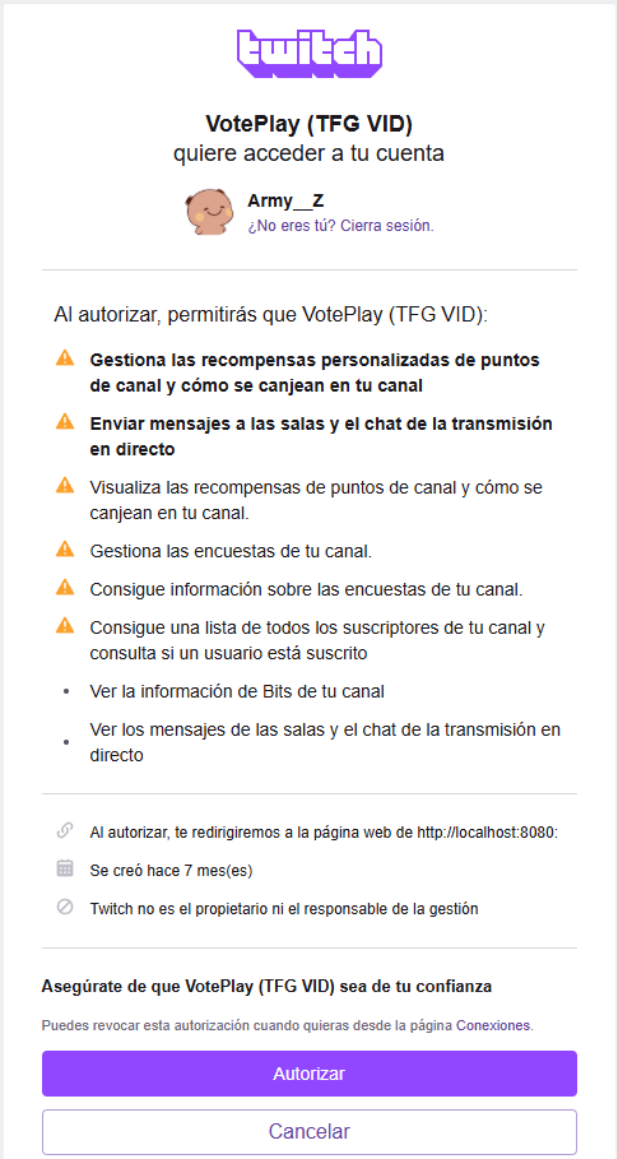
Marca este objeto como DontDestroyOnLoad (el propio TwitchManager ya gestiona el singleton).



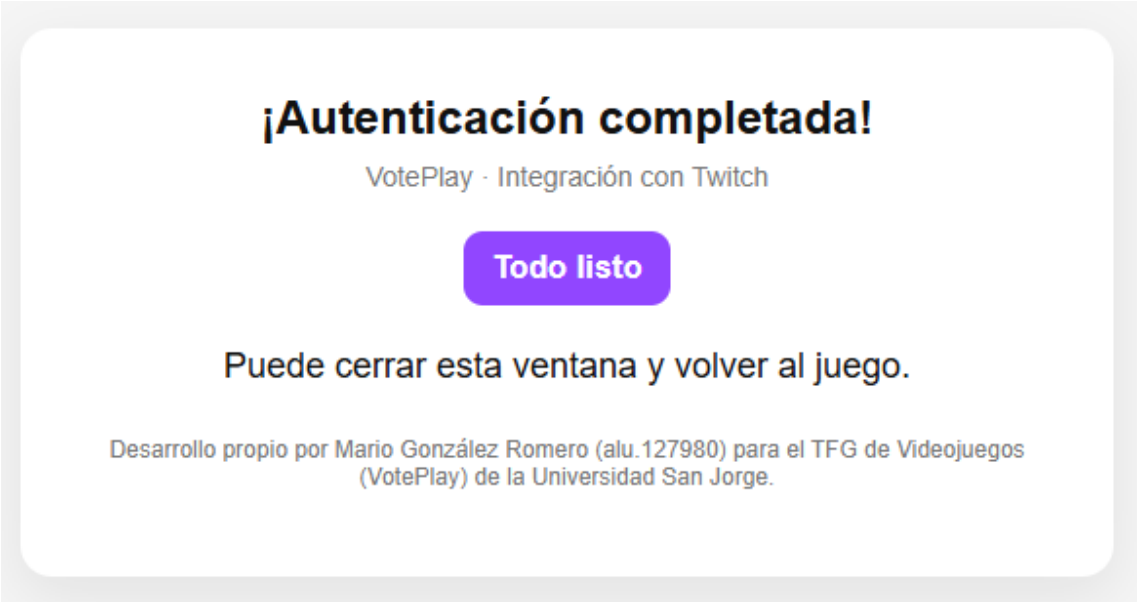
El broadcaster_id almacena el id asociado a tu usuario según los scopes aplicados tras la autentificación de Twitch.

12.3.3. Autenticación

- TwitchManager comprueba si hay token guardado. Si no lo hay, llama a TwitchAuth.OpenTwitchLogin().
- Twitch muestra la pantalla oficial de scopes.



- Tras autorizar, Twitch redirige a http://localhost:8080/#access_token=....



- El bridge inline convierte ese # en ? y el plugin lee el token por HTTP.
- Se guarda el token en PlayerPrefs y se muestra la página de éxito embebida que funciona en Editor y en build (no depende de archivos externos).
- TwitchManager obtiene datos del broadcaster y deja listo PubSub/Helix.

El plugin sanea el token antes del LISTEN (quita comillas, espacios y prefijos erróneos) para evitar ERR_BADAUTH. Además implementa PING/PONG, reconexión y re-suscripción automática en PubSub.

12.4. Módulos del plugin

12.4.1. ChatManager (conexión IRC segura)

Su propósito principal es conectar el juego al chat de Twitch mediante IRC, escuchar comandos escritos por los espectadores y enviar mensajes del sistema. Garantiza que los callbacks se ejecuten en el hilo principal de Unity para evitar fallos en la plataforma.

Requisitos previos

- Token OAuth válido en PlayerPrefs["TwitchAccessToken"] con scopes chat:read y chat:edit (lo obtiene TwitchAuth).
- Streamer.Instance.login con el login del canal (minúsculas).
- Conexión a irc.chat.twitch.tv:6667 (no TLS). El componente se auto-inicia en Start().

Métodos públicos (API)

- void ConnectToChat()

  - Abre el TcpClient, negocia IRC (PASS oauth:{token}, NICK {login}, JOIN #{login}) y lanza un hilo de escucha.
- void SendChatMessage(string message)
  - Envía message al canal del streamer. Requiere conexión activa.
- void MonitorMessage(string searchText, Action<string> onUserFound)
  - Registra un detector de subcadenas (ignora mayúsculas/minúsculas).
    - searchText: texto a buscar (p. ej., "!jugar").
    - onUserFound: callback que recibe el nombre de usuario que escribió el mensaje.
- void StopMonitoringMessage(string searchText)
  - Deja de vigilar el patrón indicado.

Comportamiento interno relevante

- Hilo de escucha (ListenForChat): lee líneas IRC, responde a PING con PONG y, ante PRIVMSG, extrae username y chatMessage.
- Cola al hilo principal: cualquier acción sobre Unity se encola en mainThreadActions y se ejecuta en Update().
- Ciclo de vida: singleton en Awake(). Si aún no hay token, se suscribe a TwitchAuth.OnTokenObtained y conecta cuando llegue. Limpieza ordenada en OnDestroy() (cierre de hilo y streams).

Ejemplificaciones de uso

```
private void ListarJugadores()
{
    if (gameStarted) return;
    ChatManager.Instance?.MonitorMessage("!jugar", (user) =>
    {
        spawnManager.CrearJugador(user);
    });
}
```

```
ChatManager.Instance?.StopMonitoringMessage("!jugar");
```

```
ChatManager.Instance.SendChatMessage(
    "¡Vota la categoría! Solo participantes. Escribe !ciencias, !deportes, !entretenimiento, !historia o !random (1 voto por usuario).");
```

12.4.2. TwitchPolls (encuestas Helix + PubSub)

Propósito. Encapsula la creación y resolución de encuestas de Twitch (API Helix) desde Unity. Se usa para votar la duración de la partida, categorías, etc., y devuelve el ganador cuando la encuesta termina.

Estructuras de datos

class Poll

- string title - Título de la encuesta.
- List<PollChoice> choices - Opciones (mínimo 2 opciones).
- int duration - Duración en segundos (Helix exige 15–180 s; la clase hace clamp).
- bool channelPointsVotingEnabled - Permitir votos con puntos del canal.
- int channelPointsPerVote - Coste de cada voto con puntos.
- bool allowMultipleVotes - Permitir varios votos por usuario.
- class PollChoice
- string title - Texto de la opción.

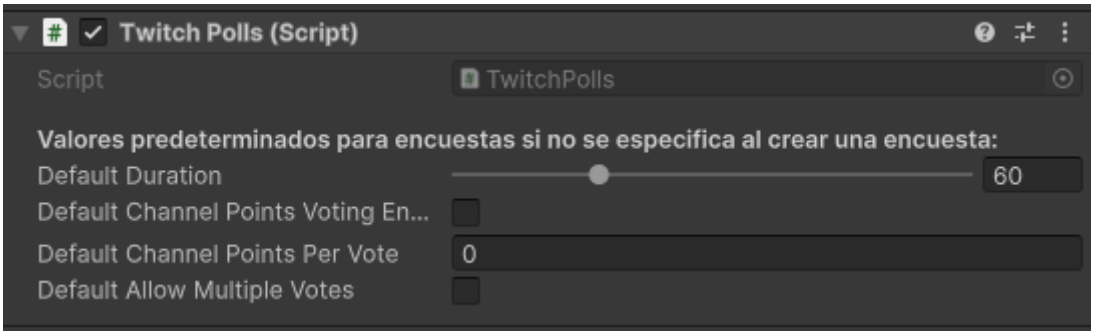
Método público

void CreatePoll(Poll poll, Action<PollChoice> onComplete)

Crea la encuesta en Helix (POST /helix/polls) y, al terminar el tiempo, consulta el resultado (GET /helix/polls) y ejecuta el callback con la opción ganadora (PollChoice). Si hay error o empate no resoluble por Twitch se encarga de resolverlo.

- poll - Objeto de encuesta.
- onComplete - Acción que recibe el ganador cuando finaliza la encuesta.

Tras crear la encuesta, espera poll.duration + 1s y realiza una única lectura del resultado; calcula la opción con mayor número de votos (incluyendo votos con puntos del canal si están habilitados).



Requisitos

- Scopes: channel:manage:polls y channel:read:polls (configurados en tu TwitchConfig).
- Autenticación: token válido en PlayerPrefs (lo gestiona el módulo TwitchAuth/TwitchManager).

Ejemplo de uso

```
var encuesta = new TwitchPolls.Poll
{
    title = "¡Vota la duración de la partida!",
    choices = new List<TwitchPolls.PollChoice>
    {
        new TwitchPolls.PollChoice { title = "Corta" },
        new TwitchPolls.PollChoice { title = "Media" },
        new TwitchPolls.PollChoice { title = "Larga" }
    },
    duration = 60,
    channelPointsVotingEnabled = true,
    channelPointsPerVote = 1,
    allowMultipleVotes = true
};

TwitchPolls.Instance.CreatePoll(encuesta, ganador =>
{
    if (ganador == null) { duracionPartidaLabel = "—"; return; }

    switch (ganador.title)
    {
        case "Corta": puntosParaGanar = 5; duracionPartidaLabel = "Corta"; break;
        case "Media": puntosParaGanar = 8; duracionPartidaLabel = "Media"; break;
        case "Larga": puntosParaGanar = 12; duracionPartidaLabel = "Larga"; break;
        default: duracionPartidaLabel = "—"; break;
    }
    ActualizarEncabezadoSeleccion();
});
```

12.4.3. TwitchRewards (puntos del canal, PubSub + Helix)

Gestiona de forma robusta las recompensas de puntos del canal (crear, borrar, actualizar color) y detecta canjes en tiempo real vía PubSub; opcionalmente completa ("fulfill") las redenciones en Helix. Incluye fallback por API Helix si PubSub falla (err_badauth, cierres del socket, etc.).

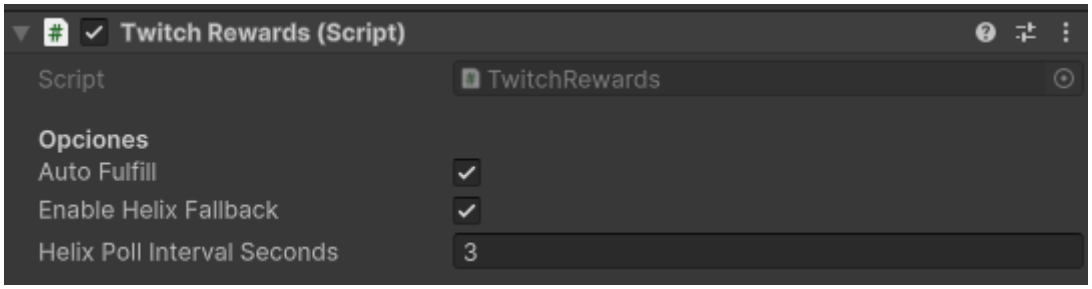
Requisitos y scopes

- Client-ID: TwitchConfig.Instance.clientId.
- Token de usuario del broadcaster (no de app) almacenado por TwitchAuth.
- Scopes mínimos:
    - channel:read:redemptions (escucha PubSub / lectura de redenciones).

- channel:manage:redemptions (crear/borrar recompensas y marcar redenciones como FULFILLED).

Propiedades de configuración

- bool autoFulfill → si true, marca la redención como FULFILLED automáticamente.
- bool enableHelixFallback → si true, al fallar PubSub debido a cierres inesperados, consulta Helix periódicamente.
- float helixPollIntervalSeconds → intervalo del fallback (seg).



Eventos

- event Action<string, string> OnRewardRedeemed;
  - Se dispara al detectar una redención (por PubSub o por fallback Helix).
  - Parámetros: (rewardId, userName).

API (métodos)

- void CreateReward(Reward reward, Action<Reward> onCreated, Action<string> onError)
  - POST a helix/channel_points/custom_rewards.
  - Rellena reward.id en la devolución. La recompensa queda registrada internamente para gestión/limpieza.
- void DeleteRewardByName(string rewardTitle)
  - Elimina por título una recompensa creada por este proceso.
- void UpdateRewardColor(string rewardId, string newHex, Action onOk = null, Action<string> onError = null)
  - PATCH: cambia el color de fondo (ej. "#FF0000").
- void DeleteAllCreatedRewards(Action onComplete = null)
  - Borra en serie todas las recompensas que este proceso tiene registradas (idempotente).

Canjes (callbacks)

- void RegisterRewardHandler(string rewardId, Action<string> onRedeemed)
  - Asocia un callback a un rewardId.
  - Parámetro del callback: userName (display name del espectador).
  - Puedes registrar varios handlers para una misma reward (se acumulan).

Comportamiento interno para tener en cuenta

- Resiliencia PubSub: PING/PONG, reconexión con backoff, re-suscripción automática y saneado del token (quita Bearer , comillas y espacios) antes del LISTEN para evitar ERR_BADAUTH.
- Fallback Helix: si PubSub no está disponible o hay errores, consulta periódicamente UNFULFILLED por cada recompensa creada; evita duplicados cruzados con un set de redemptionId.
- Idempotencia: operaciones de crear/actualizar/eliminar y de fulfill toleran reintentos y reinicios; el sistema mantiene un registro interno de recompensas creadas para limpieza fiable.
- Limpieza al salir: intercepta Application.wantsToQuit (y fin de Play Mode en Editor) para borrar todas las recompensas creadas antes de cerrar.

Ejemplo de uso

```
var reward = new TwitchRewards.Reward(title, prompt, cost, true, hex);

TwitchRewards.Instance.CreateReward(
    reward,
    onCreated: r =>
    {
        storeId?.Invoke(r.id);

        TwitchRewards.Instance.UpdateRewardColor(r.id, hex, null,
            err => Debug.LogWarning($"No se pudo actualizar color de '{r.title}': {err}"));

        TwitchRewards.Instance.RegisterRewardHandler(r.id, user =>
        {
            ApplyPlayerMaterial(user, material);
        });
    },
    onError: err => Debug.Log($"Error creando reward '{title}': {err}, existente")
);
```

12.5. Recomendaciones de inicialización y ciclo de vida

Un solo punto de arranque: crea el GameObject Twitch en la primera escena. El TwitchManager ya se declara singleton y usa DontDestroyOnLoad.

Autenticación temprana: lanza el login al arrancar si no hay token; si lo hay, prosigue con PubSub/Helix.

Separación de responsabilidades: deja que TwitchManager gestione OAuth/PubSub/Helix; el juego se comunica por eventos o llamadas de alto nivel.

Limpieza al salir: en OnDestroy() o OnApplicationQuit():

- TwitchRewards.Instance?.DeleteAllCreatedRewards(null);
- TwitchRewards.Instance?.StopPubSub();
- Borra monitorings del ChatManager que sigan activos.

Hilo principal: toda interacción con Unity (UI, GameObject, materiales) ocurre en el main thread. El plugin ya enruta callbacks al hilo principal cuando es necesario.