Feed The Leader

jajerje studios ®

1- Jorge Repullo Serrano - jorgers4@uma.es
2- Artur Vargas Carrión - arturvargas797@uma.es
3- Juan Manuel Valenzuela González - amcgil@uma.es
4- Eduardo González Bautista - edugb@uma.es
5- Rubén Oliva Zamora - rubenoliva@uma.es
6- Javier Toledo Delgado - javier.toledo.delgado@uma.es
7- Eloy González Castillo - eloygonzalez@uma.es

GR1-04
Ingeniería del Software A
Universidad de Málaga

ÍNDICE DE CONTENIDO

1. Introducción

Nuestra idea de proyecto se basa en la creación de un videojuego del género incremental o *clicker*, similar a otros juegos famosos como *Cookie Clicker*. La premisa de este tipo de juegos es muy simple: **asignar una tarea repetitiva al jugador** que le aporte puntos (por ejemplo, hacer clic, de ahí el nombre). Dicha tarea puede parecer repetitiva y aburrida en un principio, pero la idea es ir **avanzando progresivamente** consiguiendo multiplicadores y potenciadores que hagan el avance más satisfactorio.

En otras palabras, en un **juego tipo *clicker***, el jugador tan solo necesita hacer clic de forma repetida o realizar una acción equivalente con el objetivo de **obtener recursos virtuales**. Estos recursos pueden ayudar a mejorar personajes u otros elementos dentro de la temática del juego. A largo plazo, esta repetición constante permite obtener más recursos de manera progresiva.

En nuestro caso vamos a desarrollar un *clicker* llamado ***Feed The Leader***. El líder es un ser supremo que necesita **alimentarse para aumentar su poder**. Nuestra tarea será clicar para darle comida, cuanta más comida le demos, ganaremos más **puntos de fe** (FP) con los que podremos comprar mejores comidas para el líder, potenciadores para ganar más puntos por un tiempo limitado, multiplicadores que aumenten la cantidad de comida que podemos ofrecer al líder, etc.

Nuestro objetivo será ser el **mayor adorador del líder**. Esto se alcanza al llegar a una determinada cantidad de puntos de fe y comprar todas las mejoras. Cuando lleguemos a ese punto podremos **prestigiar**, lo que implica que **empezaremos de nuevo,** pero con uno o varios **potenciadores base,** que acelerarán el progreso.

Al alimentar al líder va **aumentando su peso**, lo que se verá reflejado en su **apariencia física**. Cuando el medidor de saciedad esté lleno significa que el líder está saciado. El líder **prefiere comer cuando no está saciado**, esto implica que la **cantidad de puntos obtenida será mayor** bajo estas condiciones ―y consecuentemente menor si lo sobrealimentamos―. Conforme la cantidad de comida que podamos ofrecer al líder y la frecuencia con la que podamos alimentarlo aumente, necesitaremos una forma de **incrementar la cantidad de comida que puede soportar**. Este problema lo soluciona la **prensa cuántica**, que reduce el tamaño de la comida acercando sus átomos sin que pierdan en el proceso su valor nutricional ni energético. Podremos mejorar la prensa con puntos de fe, generando así una dependencia entre las mejoras a la prensa, a los alimentos y demás.

1.1. ¿Por qué un *clicker*? ¿Por qué *Feed The Leader*?

En la danza constante de clics, en la repetición aparentemente trivial, hallamos un eco de la vida misma. Un videojuego incremental no es solo un entretenimiento; es un **recordatorio**. Como el sol que asoma cada día, como las olas que acarician la orilla sin cesar, la consistencia es la melodía que subyace en nuestra existencia.

En el *clicker*, encontramos la paradoja: lo pequeño se vuelve grande, lo insignificante se torna poderoso. Cada clic, como una gota en el océano, suma y construye. Así también en la vida, nuestros esfuerzos diarios, aunque modestos, moldean nuestro destino. La persistencia, como un hilo dorado, teje la trama de nuestras historias.

Aumentar la moneda de canje, clic tras clic, es un ritual que trasciende lo mundano. En cada pulsación, se esconde la promesa de un mañana más brillante. No es solo un juego, sino una lección: la grandeza no surge de actos grandiosos aislados, sino de la constancia, del compromiso inquebrantable con nuestra causa.

Que este juego sea un faro en la noche, una **invitación a la perseverancia**. Que cada clic sea un voto por la grandeza, un tributo a la persistencia. En el *clicker*, en la vida, la consistencia es la llave que desbloquea puertas insospechadas. Así, clic a clic, tejemos nuestra epopeya personal, recordando que hasta el más pequeño gesto puede resonar en la vastedad de la eternidad.

Además de una **oda a la consistencia**, nuestro juego, *Feed The Leader*, emerge como una ventana a la **crítica** hacia las complejas **estructuras de poder en la sociedad contemporánea**. Al interactuar con la tarea aparentemente simple de alimentar a un ser supremo, el juego nos incita a reflexionar sobre las diferentes capas de autoridad y sumisión que impregnan nuestras vidas. Cada clic en el juego puede interpretarse como un acto de complacencia hacia una figura de autoridad, generando así una oportunidad para examinar críticamente cómo nuestras acciones cotidianas pueden reforzar o desafiar las jerarquías existentes.

2. Roles

Durante el desarrollo del proyecto a cada integrante del grupo se le asignará un rol. Cada uno será el encargado de que ese trabajo salga bien. No obstante, eso no significa que el integrante solo se dedique a su rol y no ayude con los demás.

Entre los roles elegidos encontramos: coordinador, programador, tester, analista y arquitecto.

2.1. Descripción de los roles

El **coordinador** lidera la planificación y ejecución de proyectos, asignando tareas y recursos. Su rol implica mantener la motivación y colaboración del equipo, así como resolver conflictos. Coordinan la comunicación interna y externa, asegurando la alineación con los objetivos organizacionales. En resumen, el coordinador desempeña un papel clave en la gestión efectiva, liderazgo y éxito del proyecto. Necesitamos a personas que sepan tomar buenas decisiones y que soporten la presión.

El **programador** se encargará principalmente de escribir código utilizando diferentes lenguajes de programación. Su función principal es convertir los diseños y especificaciones proporcionados por los analistas y desarrolladores en software funcional. Necesitamos a gente que sea muy hábil con el lenguaje de programación a utilizar.

El **tester** se encarga de probar aplicaciones y sistemas para identificar defectos, asegurando su calidad y rendimiento. Sus responsabilidades incluyen la creación de casos de prueba, ejecución de pruebas manuales o automáticas, documentación de resultados y colaboración con desarrolladores para corregir problemas. Además, participa en la validación de requisitos y contribuye al mantenimiento de estándares de calidad en el ciclo de desarrollo. Su objetivo es garantizar que el software cumpla con los estándares de calidad establecidos antes de su lanzamiento. Necesitamos que los tester tengan muy claro el funcionamiento del producto y que sepan identificar rápidamente de dónde pueden venir los fallos.

El **analista** es responsable de analizar las necesidades del cliente y traducirlas en requisitos técnicos y funcionales para el equipo de desarrollo. Su función implica comprender los procesos comerciales y las necesidades del usuario final, así como documentar y comunicar claramente los requisitos al equipo de desarrollo. Necesitamos a gente que sepa entender las necesidades del cliente y traducirla fácilmente a una implementación en el producto a desarrollar.

El **arquitecto** se encarga de diseñar la estructura y la arquitectura técnica de sistemas y aplicaciones. Sus responsabilidades incluyen la definición de la estructura del software, la selección de tecnologías, la elaboración de patrones de diseño y la garantía de la coherencia y la escalabilidad del sistema. Además, colabora con analistas y otros miembros del equipo para asegurar que la arquitectura cumpla con los requisitos del proyecto. Necesitamos a gente que conozca diversas aplicaciones y técnicas que puedan ser útiles en el desarrollo del proyecto.

2.2. Asignación de roles

Tras evaluar las fortalezas y debilidades de cada integrante, se ha determinado que los roles de cada uno deberían ser los siguientes:

- **Jorge Repullo Serrano:** analista y tester.
- **Artur Vargas Carrión:** analista y coordinador.
- **Juan Manuel Valenzuela González:** tester y programador.
- **Eduardo González Bautista:** programador y tester.
- **Rubén Oliva Zamora:** coordinador y arquitecto.
- **Javier Toledo Delgado:** programador y arquitecto.
- **Eloy González Castillo:** arquitecto y analista.

3. Gestión de Riesgos

Antes de comenzar con el proyecto, debemos ser conscientes de la posibilidad de que no todo el proceso de desarrollo sea en línea recta. Para ello, realizamos una lista con los potenciales riesgos que pueden surgir durante nuestro trabajo, así como su clasificación, la probabilidad de que ocurra, las consecuencias y una posible estrategia para minimizar estas consecuencias.

Movilidad del personal

- Tipo: proyecto.
- Descripción: algún integrante del grupo por motivos personales tenga que abandonar el proyecto.
- Probabilidad: muy baja.
- Efecto del riesgo: serio.
- Estrategia para mitigarlo: mantener informados a los integrantes del grupo de posibles inconvenientes, para poder distribuir la carga de trabajo y prevenir cambios en la planificación.

Subestimación de la dificultad

- Tipo: proyecto y producto.
- Descripción: subestimar la dificultad del desarrollo del software necesario para la realización del proyecto.
- Probabilidad: alta.
- Efecto del riesgo: tolerable.
- Estrategia para mitigarlo: esperar que las tareas nos van a llevar más de lo habitual a la hora de planear cualquier evento, tener en cuenta experiencias de los integrantes del grupo y realizar cualquier trabajo con tiempo de antelación.

Mala planificación con los tiempos de entrega

- Tipo: organización.
- Descripción: mala organización a la hora de dividir la carga de trabajo que provoque una mala compatibilidad a la hora de dedicarle el tiempo al proyecto junto a otras asignaturas.
- Probabilidad: moderada.
- Efecto del riesgo: tolerable.
- Estrategia para mitigarlo: comenzar a trabajar en el proyecto desde el día uno, para que cualquier inconveniente que surja, poder solucionarlo a tiempo mucho antes de la entrega.

Competencia del producto

- Tipo: negocio.
- Descripción: otro grupo realiza un proyecto parecido, que puede provocar comparaciones o incluso que el profesor decida que el proyecto no salga adelante.
- Probabilidad: moderada.
- Efecto del riesgo: tolerable.
- Estrategia para mitigarlo: plantear una idea original o de difícil realización la cual sea complicada de replicar.

Las partes del código que se iban a reutilizar tienen defectos que limitan su funcionalidad

- Tipo: tecnológico.
- Descripción: cuando se trabaja con código, lo normal es reutilizar parte de estos, pues en este caso, suponemos que parte de este código no puede funcionar en todos los casos.
- Probabilidad: alta.
- Efecto del riesgo: insignificante.
- Estrategia para mitigarlo: desarrollar código consistente y coherente con muchos comentarios que ayuden en su comprensión, teniendo en cuenta la posible reutilización de estos escribiéndolos de la forma más genérica posible para que funcionen en un mayor rango de ámbitos.

Pérdida de tiempo en características poco relevantes

- Tipo: organización.
- Descripción: los desarrolladores emplean demasiado tiempo en implementar características alejadas del esqueleto del proyecto y no emplean el tiempo necesario en pulir las partes más importantes del mismo.
- Probabilidad: muy alta.
- Efecto del riesgo: serio.
- Estrategia para mitigarlo: centrarse en desarrollar el esqueleto del proyecto, y hasta que este no esté terminado no a implementar características menos relevantes.

Subestimación de la cantidad de errores en el software del proyecto

- Tipo: tecnológico.
- Descripción: los desarrolladores crean un código con demasiados errores que no permite continuar con otras partes del proyecto.
- Probabilidad: moderada.
- Efecto del riesgo: serio.
- Estrategia para mitigarlo: crear el software poco a poco, con muchos tests para reducir errores potenciales y con numerosas revisiones.

Bajo índice de detección de errores por parte de los tester

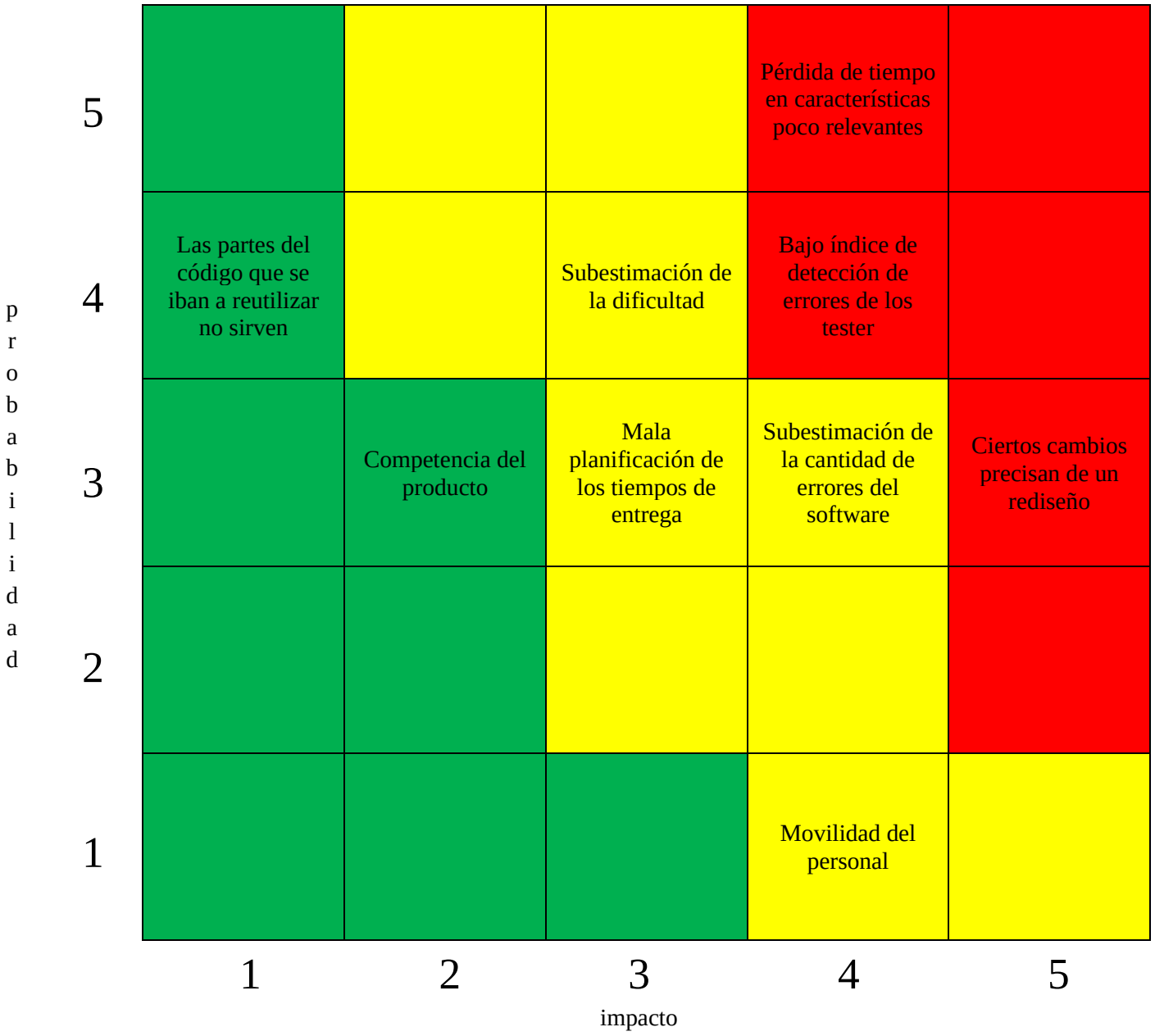
- Tipo: tecnológico.
- Descripción: los desarrolladores crean un código con demasiados errores que los tester no son capaces de encontrar.
- Probabilidad: alta.
- Efecto del riesgo: serio.
- Estrategia para mitigarlo: adoptar una estrategia de detección de errores efectiva por parte de los tester, realizando acciones poco comunes para encontrar el máximo número posible de errores.

Se proponen unos cambios de requisitos que necesitan un importante rediseño

- Tipo: organización.
- Descripción: el profesor o la forma de la estructura del trabajo, nos obliga a cambiar gran parte de lo que llevamos hecho.
- Probabilidad: moderado.
- Efecto del riesgo: catastrófico.
- Estrategia para mitigarlo: Amoldar la forma del trabajo para que los cambios no supongan un cambio de la estructura completa, y organizarnos correctamente los integrantes del grupo con objeto de evitar esto.

3.1. Matriz de riesgos

Para poder visualizar los riesgos, hemos decidido ilustrarlos en una matriz de riesgos:



4. Planificación

4.1. Modelo de proceso de software elegido

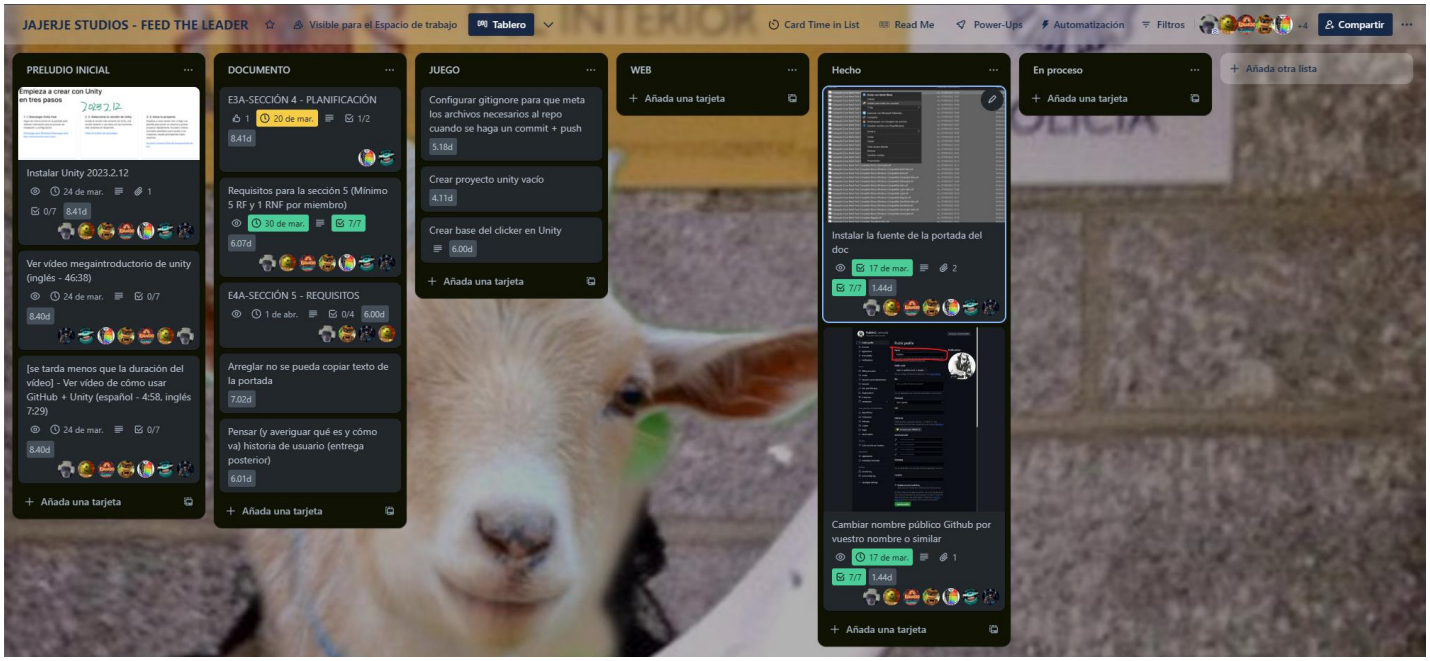
Se ha elegido la metodología ***Scrum*** porque es un modelo que se adecúa perfectamente a nuestras necesidades en el proyecto. Esta **metodología** es **iterativa** e **incremental**. Que sea iterativa quiere decir que el trabajo se divide en iteraciones, llamadas *sprints* en el caso de *Scrum*, que son períodos de tiempo predefinidos durante los cuales se realiza el trabajo planificado. Al final de cada *sprint*, se produce un incremento de software funcional y potencialmente entregable.

Nuestra organización de proyecto se crea gracias a las **reuniones regulares** en las que se decide la mejor organización posible para aumentar la productividad, y son lideradas por el "coordinador principal", llamado en la metodología *Scrum*, *Scrum Master*. Tenemos una comunicación constante y efectiva, con continuas interacciones, lo cual nos permite tener mejores análisis de la situación en cada momento del proyecto.

El uso de este método nos proveerá una mayor auto-organización, pues cada tarea asignada a uno o varios miembros es resuelta por el método que estos consideren. Además de una mayor autonomía y auto-superación, el proyecto se mejora progresivamente. De todo esto, concluimos en un enriquecimiento de todos los miembros del grupo y una transmisión del conocimiento constante.

4.2. Organización mediante tableros

Hemos usado la aplicación web Trello para organizar las tareas mediante tableros y tarjetas.



1 Tablero de Trello

Nuestro tablero de Trello tiene, actualmente, varias columnas. En primer lugar, encontramos la columna ***Preludio inicial***. En esta columna, se observan tareas que no forman parte de ninguna entrega, como

por ejemplo la instalación de *Unity*, ver vídeos para aprender lo básico sobre *Unity* y *GitHub*, instalar fuentes de Word, etc.

A continuación, tenemos la columna ***Documento***. Aquí están todas las tareas relacionadas con la creación del documento del proyecto, como pueden ser la realización las diferentes secciones por entregar cada semana, fallos a corregir de cada sección u otras partes del documento, o "subtareas" que cada integrante debe hacer.

Con respecto a la columna ***Juego***, como su nombre indica, se encuentran las actividades con la realización de nuestro juego. En esta columna, ahora se encuentran todos los **requisitos,** tanto **funcionales** como **no funcionales,** que queremos implementar en nuestro *clicker*. El funcionamiento es sencillo, cuando uno o varios miembros del grupo se pongan de acuerdo en implementar un requisito, se lo autoasignarán, actualizando la tarjeta correspondiente en el Trello. Además, lo comunicarán al resto de integrantes para evitar conflictos.

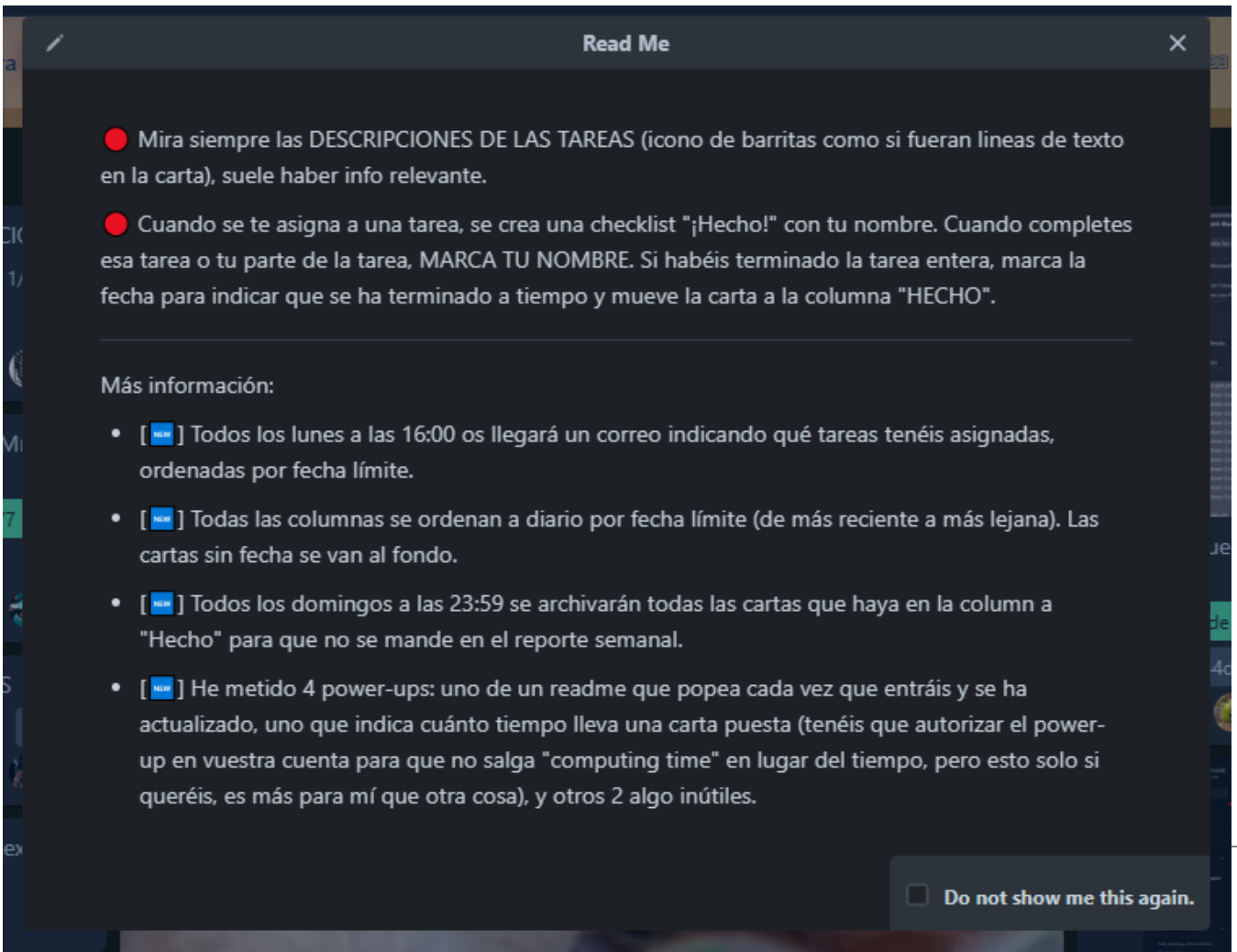
En la columna **En proceso**, encontramos las tareas y requisitos del juego que se están realizando y no han sido terminadas.

Más a la derecha tenemos las columnas ***Web***, aún vacía y ***Hecho***, donde se irán introduciendo aquellas tareas que estén terminadas.

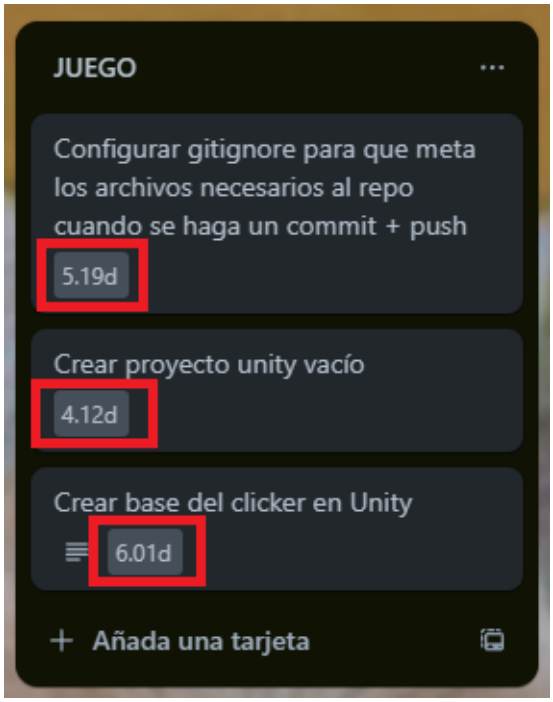
4.3. Power-ups

Se ha decidido añadir los siguientes *power-ups* para conseguir una organización todavía más efectiva:

- **Readme:** usado para dar información general sobre el funcionamiento de este espacio de trabajo.

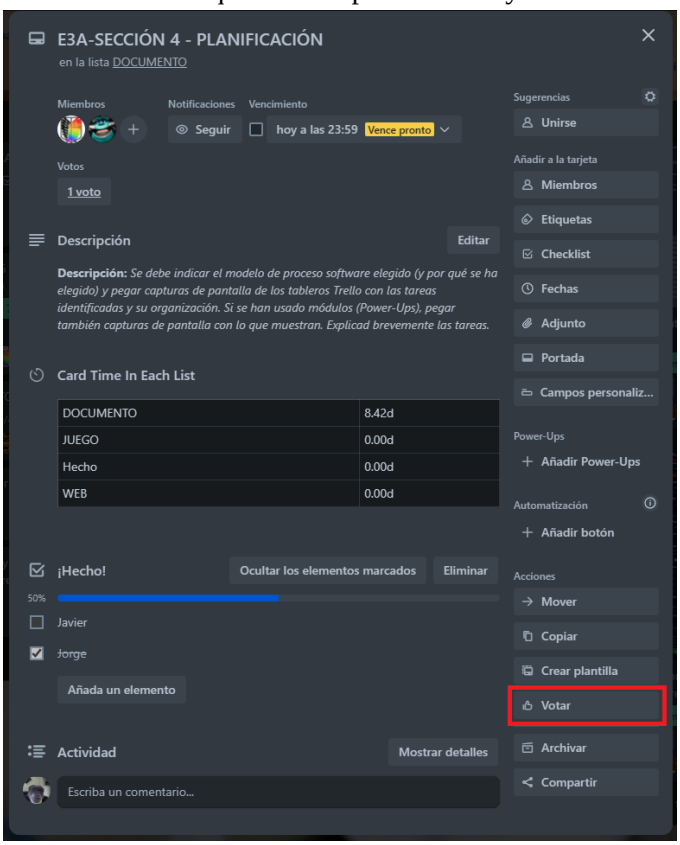


- **Card time:** usado para ver el tiempo que una tarea lleva publicada en el tablero Trello.



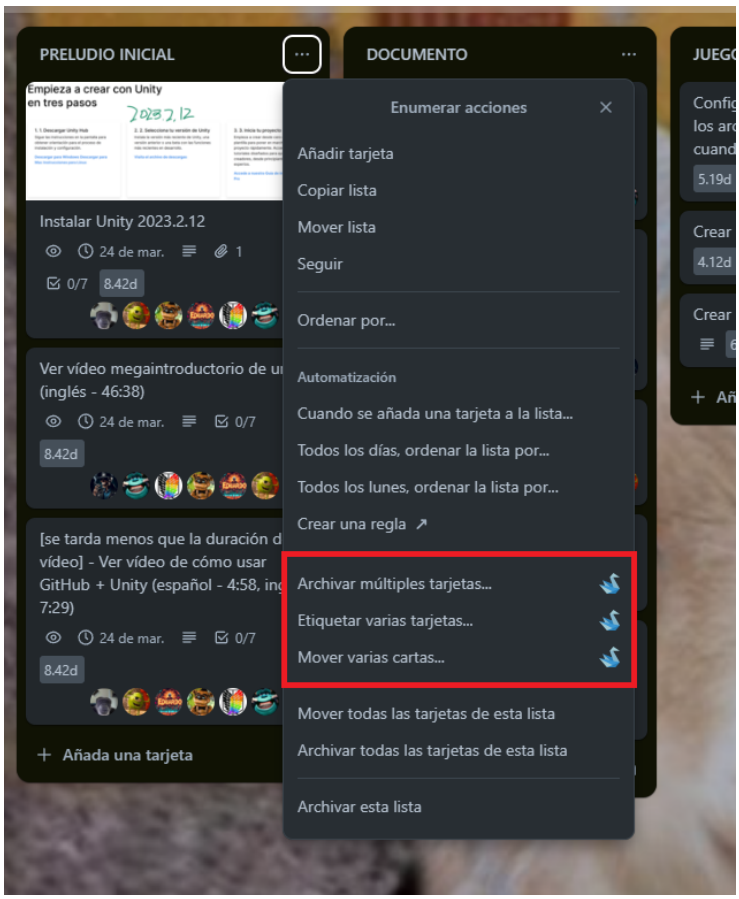
3 Muestra del power-up Card time

Votar: los miembros del grupo puedan votar si una tarea es útil para comunicar de forma efectiva que son conscientes de que forman parte de ésta y están de acuerdo con ella.



4 Muestra del power-up "Votar"

- **Manny:** añadido para ganar eficiencia a la hora de planificar en Trello. Permite mover múltiples tarjetas simultáneamente.

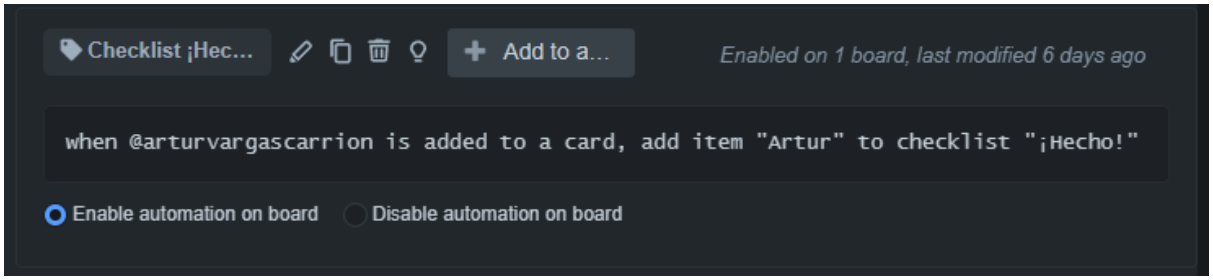


5 Muestra del power-up "Manny"

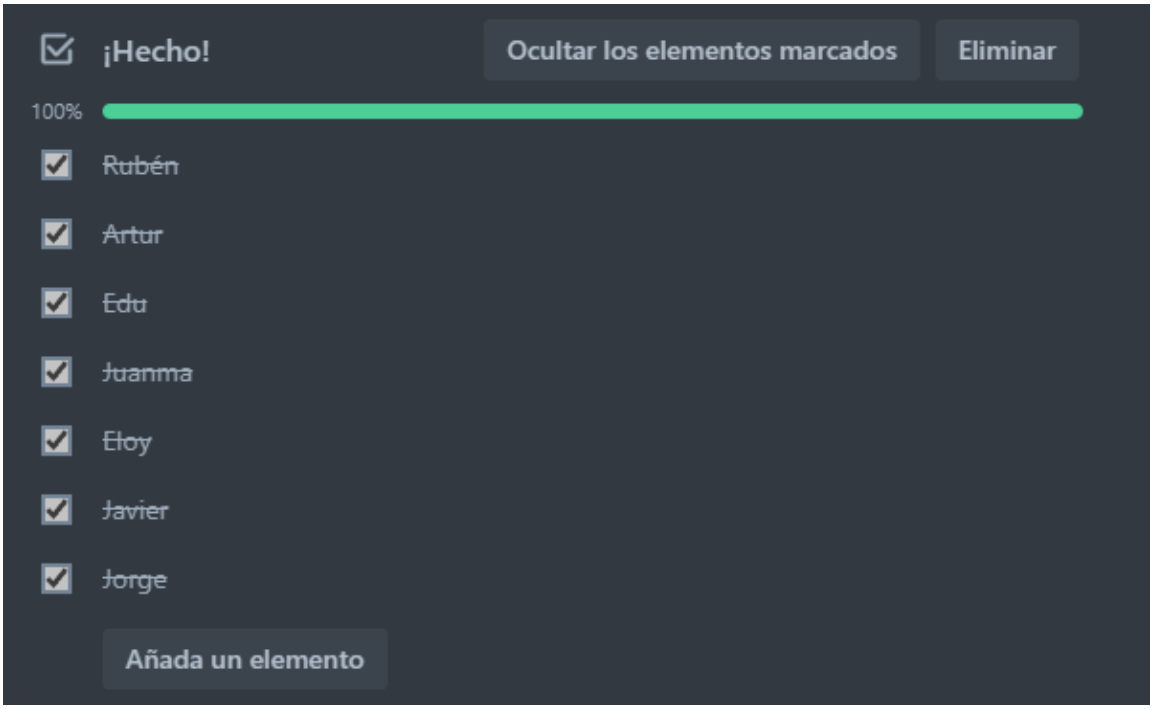
4.4. Automatizaciones

Como ocurre con el power-up Manny, se han creado diferentes automatizaciones con el fin de mejorar la eficiencia a la hora de organizar el proyecto.

- **Checklist:** cuando se añade un miembro a una tarea, automáticamente se crea una lista con el nombre de esa persona. Esta lista sirve para cuando un miembro completa su parte, tacha su nombre de la lista, así se ve visualmente quién ha terminado cada parte.

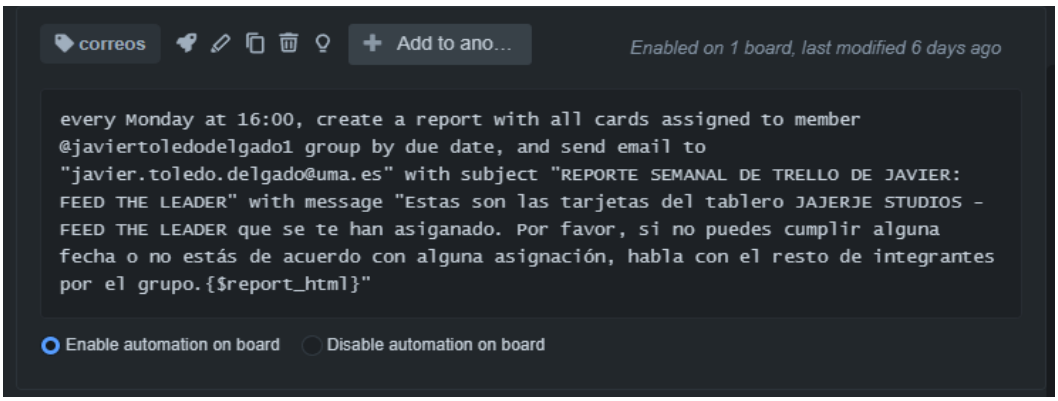


6 Automatización para crear el Checklist ¡Hecho!



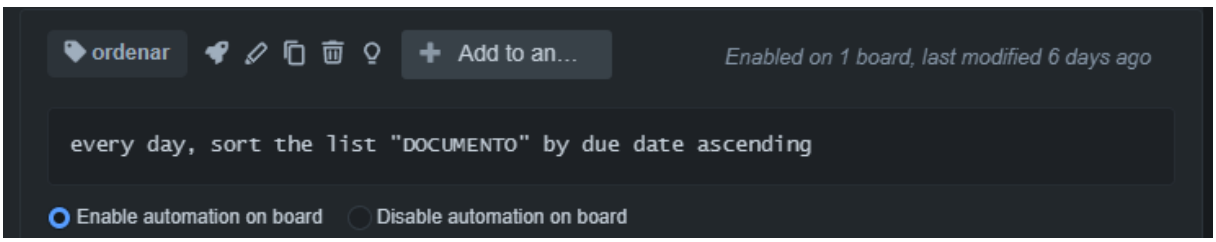
7 Checklist ya creado

- **E-mail semanal:** se ha creado una automatización para que a todos los miembros les llegue un correo semanal en el que se detallan todas las tareas no completadas de las que forma parte.



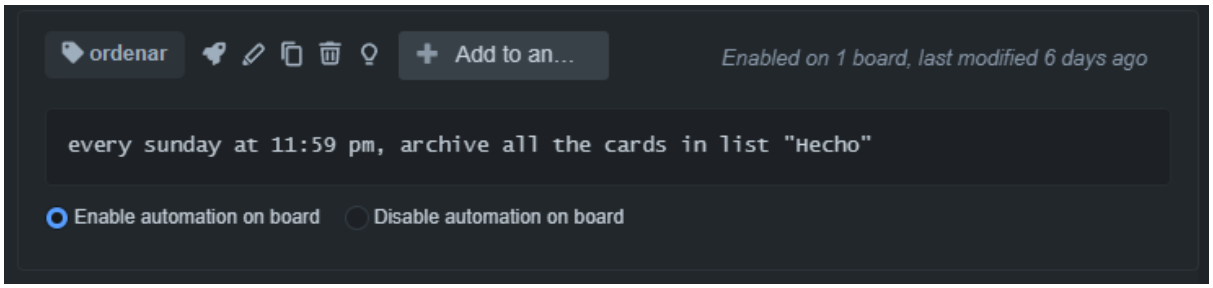
***8** Automatización para enviar correos*

- **Ordenar tarjetas por fecha límite:** cada tarjeta de cada lista se ordena según la fecha de vencimiento diariamente gracias a esta automatización.



9 Automatización para ordenar las listas según su fecha de entrega

- **Archivo automático de tareas:** semanalmente se archivarán automáticamente las tareas que ya hayan sido completadas.

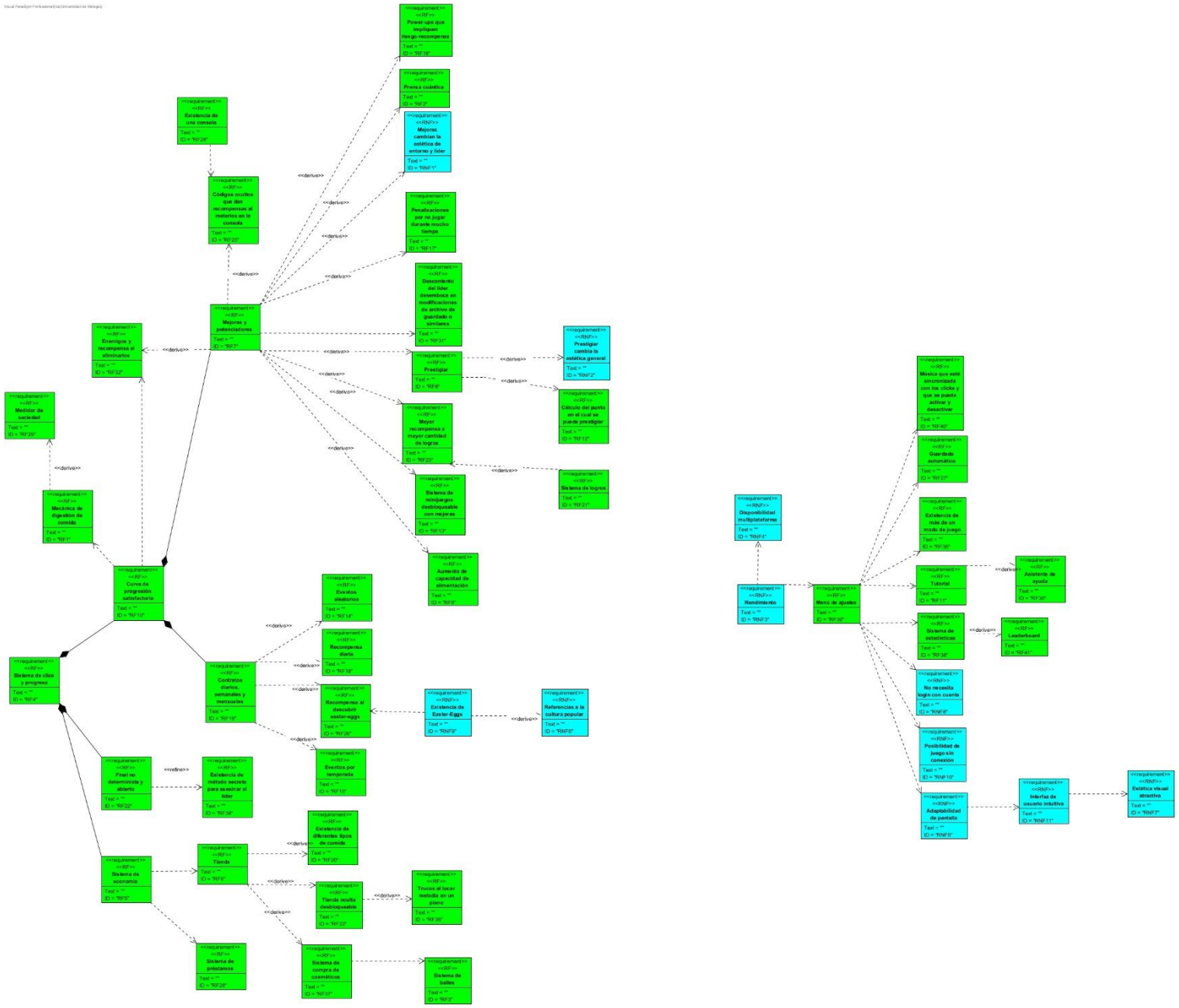


10 Automatización para el archivo de tarjetas

5. Requisitos

En la concepción y desarrollo de cualquier proyecto de software, la comprensión y especificación de requisitos son fundamentales para su éxito. En esta sección, exploraremos en detalle los requisitos que guiarán el diseño y la implementación del videojuego. Abordaremos tanto los requisitos funcionales, que describen las funciones específicas que el sistema debe realizar, como los requisitos no funcionales, que se centran en atributos de calidad como el rendimiento, la seguridad y la usabilidad.

A través de un enfoque sistemático, examinaremos cómo identificar, analizar y documentar estos requisitos para asegurar que el producto final cumpla con las expectativas y necesidades del usuario. Para una comprensión más visual, presentamos un diagrama de requisitos elaborado con Visual Paradigm.



11 Diagrama de requisitos creado en Visual Paradigm

5.1. Requisitos funcionales

RF1: Mecánica de digestión de comida

Como jugador
Quiero que el videojuego tenga mecánicas que me mantengan jugando
Para retener mi atención y conseguir que siga empleando mi tiempo en jugar.

Pruebas de aceptación

- Alimentar al líder con comida debe mostrar una animación o indicador visual que represente la digestión del alimento.
- La cantidad de puntos de fe obtenidos al alimentar al líder debe ser proporcional a la cantidad de comida digerida.
- La barra de saciedad debe incrementarse conforme se alimente al líder.

RF2: Prensa cuántica

Como jugador
Quiero mecánicas interesantes que hagan ameno el progreso
Para querer seguir jugando al juego.

Pruebas de aceptación

- La prensa cuántica debe ser un ítem disponible en la tienda del juego, que además debe poderse mejorar.
- Al comprar la prensa cuántica, la cantidad de calorías que el líder puede consumir debe aumentar significativamente.

RF3: Sistema de bailes

Como jugador
Quiero un sistema de bailes para el líder
Para poder divertirme con una mecánica que ridiculice al personaje principal.

Pruebas de aceptación

- Las animaciones serán fluidas y manteniendo la estética del personaje.
- Los bailes no taparán ningún aspecto importante del juego.
- Habrá un botón debajo del líder que desplegará los bailes disponibles, así como los que se podrán desbloquear.

RF4: Sistema de clics y progreso

Como jugador
Quiero que haya un sistema de progresión
Para querer avanzar en el juego.

Pruebas de aceptación

- Cada clic en el líder debe aumentar los puntos de fe del jugador.
- El progreso en el juego debe estar directamente relacionado con la cantidad de puntos de fe acumulados a través de los clics.

RF5: Sistema de economía

Como jugador
Quiero ver que mis puntos tienen un uso
Para querer emplearlos y seguir consiguiendo más.

Pruebas de aceptación

- El juego debe mostrar claramente la cantidad de puntos de fe disponibles para el jugador en todo momento.
- Con la compra de mejoras, se deben de obtener más puntos de fe por clic.
- Deben existir automatizaciones que permitan obtener puntos de fe sin la participación activa del usuario.

RF6: Tienda

Como jugador
Quiero tener una mecánica de tienda
Para sentirme partícipe de la progresión dando uso a los puntos comprando artículos que puedo ver físicamente en el juego.

Pruebas de aceptación

- Los elementos de la tienda deben estar organizados de manera lógica y fácil de entender para el jugador.
- Al comprar un artículo de la tienda, la cantidad de puntos de fe del jugador debe disminuir correctamente y el artículo debe estar disponible.
- No deben poderse comprar puntos de fe.

RF7: Mejoras y potenciadores

Como jugador
Quiero una variedad de mejoras y potenciadores
Para emplear los puntos y que aumente la eficiencia de la alimentación y el progreso en el juego.

Pruebas de aceptación

- Cada mejora o potenciador adquirido en la tienda debe tener un efecto claro y medible en el progreso del juego.
- Las mejoras y potenciadores deben escalarse de manera que su eficacia aumente con cada compra.

RF8: Prestigiar

Como jugador
Quiero una mecánica de prestigio
Para que el videojuego sea rejugable y además lo haga más interesante volver a completarlo.

Pruebas de aceptación

- Al elegir la opción de prestigiar, el progreso del jugador debe reiniciarse mientras que las mejoras bases se incrementan como se prometió.
- El jugador debe sentir una mejora significativa en la velocidad de progreso tras prestigiar.

RF9: Aumento de capacidad de alimentación

Como jugador
Quiero poder alimentar más al líder
Para que exista progresión en el juego y querer conseguir mejoras.

Pruebas de aceptación

- Las mejoras relacionadas con la capacidad de alimentación deben tener un impacto perceptible en la cantidad de comida que el líder puede consumir, y, por tanto, en los puntos de fe que se pueden conseguir.
- Las mejoras de capacidad de alimentación deben ser equilibradas para evitar romper el juego.

RF10: Curva de progresión satisfactoria

Como jugador
Quiero sentir que el avance en el juego es progresivo y satisfactorio
Para que el avance no sea aburrido, exagerado o inexistente.

Pruebas de aceptación

- La curva de progresión del juego debe diseñarse de manera que los jugadores sientan un aumento constante en el desafío y la recompensa a medida que avanzan, manteniéndolos enganchados con una constante sensación de recompensa.
- La dificultad del juego debe aumentar gradualmente para mantener el interés del jugador sin volverse abrumadora.
- Deben existir incentivos en caso de que el jugador deje de jugar durante un periodo de tiempo considerable (horas o incluso días).

RF11: Tutorial

Como jugador
Quiero tener una noción básica de cómo funciona el juego, preferiblemente mediante un asistente
Para entender la interfaz y las mecánicas.

Pruebas de aceptación

- El tutorial será una mascota que aparecerá cuando el jugador haga clic en algo que no sepa cómo funciona o que añada alguna característica.
- Estará siempre presente en una esquina de la pantalla para que se pueda clicar en caso de que no recuerdes algo.
- Si se clica se abre un menú con opciones de diálogo.
- Si no se clica el personaje se mantiene en una esquina con una animación sencilla.

RF12: Cálculo del punto en el cual se puede prestigiar

Como jugador
Quiero sentir que el prestigio es una mecánica bien implementada
Para que la progresión tenga una recompensa "final".

Pruebas de aceptación

- La curva de progresión debe obligar al jugador en algún punto del juego, a desbloquear las mejoras de prestigio.
- Esto será algo visible para el jugador, que notará que el progreso es demasiado lento.
- A medio camino de necesitarlo, la mascota se comunicará con el jugador para informarle de la existencia de esta opción y cuál es su cometido, de modo que el jugador siempre sepa cuáles son sus opciones disponibles.

RF13: Sistema de minijuegos desbloqueables con mejoras

Como jugador
Quiero minijuegos alternativos al juego base
Para hacer la experiencia y la progresión menos monótona y más divertida.

Pruebas de aceptación

- Ciertas mejoras llevan consigo un desplegable que abre un minijuego.
- Para desbloquear el minijuego, se debe haber mejorado esa mejora hasta cierto nivel. Si está bloqueado, el desplegable aparece bloqueado también, para que se sepa que esa mejora posee un minijuego.
- Durante el minijuego, la producción de puntos no se detiene.
- Se puede cerrar el desplegable en cualquier momento.

RF14: Eventos aleatorios

Como jugador
Quiero que ocurran eventos espontáneamente
Para quedar sorprendido y que se rompa la monotonía del juego con regularidad.

Pruebas de aceptación

- Arbitrariamente aparecen eventos por pantalla, que provocarán animaciones.
- Estos eventos, desencadenan minijuegos que tendrán un efecto notable y claro en el juego.
- Si el jugador no decide hacer lo que se le indique, puede ser castigado.
- Una vez terminado el evento, la animación termina y el juego vuelve a la normalidad (con castigo o recompensa).

RF15: Eventos por temporada

Como jugador
Quiero sentir y notar cada estación de la vida real en el juego
Para sentir al líder y el juego más conectado a mi vida y a la realidad.

Pruebas de aceptación

- Cambian la apariencia de la interfaz.
- Se desbloquean nuevas mejoras y formas de conseguir puntos de fe.
- Una vez terminado, las mejoras de temporada desaparecen, los puntos conseguidos se mantienen y el juego vuelve a la normalidad.

RF16: Power-ups que impliquen riesgo-recompensa

Como jugador
Quiero tener que barajar opciones y tomar decisiones
Para aumentar la intensidad del juego y arriesgarme para conseguir mejores recompensas con el objetivo de disfrutar más de ellas.

Pruebas de aceptación

- Ciertas mejoras llevan consigo un desplegable que abre un menú de power-ups.
- Para desbloquear los power-ups, se debe haber mejorado esa mejora hasta cierto nivel, si está bloqueado el desplegable aparece bloqueado también, para que se sepa que esa mejora posee power-ups.
- Activarlos es gratis, pero pueden provocar tanto efectos positivos como negativos.
- Estos power-ups tienen un tiempo determinado y el jugador no puede decidir eliminarlo siempre que quiera.

RF17: Penalizaciones por no jugar durante mucho tiempo
Como desarrollador
Quiero que el jugador sienta la necesidad de jugar
Para que siempre esté atento al juego y a las actualizaciones que se implementen, así como los eventos y recompensas.

Pruebas de aceptación

- Vienen relacionadas con el tiempo real en la Tierra, ninguno interno del juego.
- El jugador, debe sentir un deber de alimentar al líder en su experiencia de vida.
- Prescindir de alimentar al líder, llevará consigo eventos que provocarán un perjuicio en la producción de puntos de fe.
- El juego avisará al jugador al entrar después del tiempo definido como, "no jugar por mucho tiempo", del impacto que ha tenido no jugar y advertirá de no volver a fallar al líder.

RF18: Recompensas diarias
Como desarrollador
Quiero que el jugador sienta que jugar diariamente le proporcione recompensas
Para asegurar que no se siente obligado a entrar, sino que tiene ganas de hacerlo.

Pruebas de aceptación

- Al entrar al juego por primera vez en el día, se abrirá un menú que informará de una recompensa que obtiene el jugador.
- Este menú también informará de las recompensas que obtendrá si sigue entrando todos los días de aquí a una semana.
- Una vez reclamada la recompensa, este menú se cierra.

RF19: Contratos diarios, semanales y mensuales

Como desarrollador
Quiero un compromiso del jugador con el juego
Para asegurar tiempos más prolongados de juego.

Pruebas de aceptación

- En un lado de la pantalla. habrá un botón que abra una interfaz la cual informará al jugador de las misiones disponibles.
- Este menú, debe de ser claro y grande, indicando sin generar duda, de cuáles son las misiones del día, de la semana y del mes y cuáles son las recompensas por completar cada una.
- Cuanto más duren las misiones, mayores serán los tiempos de compleción, así como las recompensas.
- Al completar un contrato, aparecerá un tic sobre este que podrás clicar para reclamar la recompensa del mismo. Al reclamarla este contrato desaparece.
- Una vez cerrado el menú, se podrá acceder otra vez a él pulsando el botón.

RF20: Existencia de diferentes tipos de comida

Como jugador
Quiero variedad en las opciones del juego
Para no aburrirme y mantener el juego fresco el mayor tiempo posible.

Pruebas de aceptación

- Cada una de estas comidas, tendrá un sprite diferente, que la diferenciará de las demás, así como de un nombre único.
- Al pasar el ratón por encima de la comida, se indicará claramente como prepararla y los beneficios de la misma.
- Al quitar el ratón, la información desaparecerá.
- Estás comidas, van almacenadas en un libro de recetas, que no es más que otro menú desplegable que puedes abrir y cerrar y que se irá rellenando conforme el jugador, obtenga nuevos alimentos.

RF21: Sistema de logros

Como jugador
Quiero objetivos en el juego
Para sentir que progreso y llego a metas.

Pruebas de aceptación

- Cuando el jugador complete un logro, se emitirá un sonido que lo indicará, además se aparecerá una pestaña por pantalla, que informará cuál se ha completado.
- Los logros, cómo conseguirlos y sus beneficios, se podrán consultar en el libro de recetas.
- Estos logros, deben incentivar al jugador a seguir jugando y a conseguirlos a través de sus recompensas, pero sobre todo a aquellos que les guste hacer el 100% de los juegos.

RF22: Final no determinista y abierto

Como jugador
Quiero diferentes formas de acabar mi aventura
Para sentir que yo decido el cómo termina mi historia y la del líder.

Pruebas de aceptación

- Comprobar que, al alcanzar cierto hito o estado en el juego que indica el final, se desbloquea un logro correspondiente y puede ser visible por el jugador.
- Verificar que el juego continúa después del final, permitiendo al jugador seguir interactuando de alguna manera.
- Comprobar que todas las mejoras, puntos y valores se reinician al empezar el juego de nuevo.

RF23: Mayor recompensa a mayor cantidad de logros

Como jugador
Quiero una progresión más rápida y acelerada después de mucho tiempo jugando
Para no sentir que me estanco en el juego, sino que cada vez es más frenético y divertido.

Pruebas de aceptación

- Realizar pruebas para confirmar que a medida que se desbloquean más logros, el porcentaje de recompensa aumenta de manera adecuada.
- Verificar que las mejoras que otorgan los logros no desbalanceen el juego de sobremanera.

RF24: Existencia de una consola

Como tester y jugador
Quiero una forma de controlar el juego con mayor facilidad
Para ahorrarme los largos tiempos de juego y probar cosas concretas, o explorar el juego sin necesidad de disfrutarlo.

Pruebas de aceptación

- Verificar que la consola está presente en la interfaz del juego y se puede interactuar con ella.
- Confirmar que la consola no tiene funcionalidad real en cuanto al funcionamiento o código y solo simula interacción para activar eventos dentro del juego.
- Verificar que los comandos introducidos desencadenen los eventos esperados por los desarrolladores.

RF25: Códigos ocultos que dan recompensas al meterlos en la consola

Como jugador
Quiero recompensas por descubrir secretos
Para que conseguirlos sea algo mágico y satisfactorio, con idea de que lo obtenido, es algo que no todo el mundo tiene y que se siente único.

Pruebas de aceptación

- Probar la inserción de códigos específicos en la consola y verificar que proporcionan las recompensas o eventos esperados.
- Confirmar que los códigos están ocultos y no son fácilmente accesibles sin exploración.
- Comprobar que no se puede acceder a los códigos de ninguna forma que no sea mediante la exploración preparada.

RF26: Recompensas al descubrir *Easter eggs*

Como jugador
Quiero recompensas por descubrir secretos
Para que mi exploración y compromiso con el juego se vea premiada.

Pruebas de aceptación

- Verificar que se pueden buscar *Easter eggs* en el juego y confirmar que al descubrirlos se otorgan recompensas.
- Asegurarse de que los *Easter eggs* están bien ocultos, pero no imposibles de encontrar.
- Implementar alguna mecánica para interactuar con los *Easter eggs*, ya sea hacer clic, meter un comando, etc.

RF27: Guardado automático

Como jugador
Quiero un sistema de guardado sencillo y fácil
Para no perder el tiempo con conceptos externos a la propia acción de jugar.

Pruebas de aceptación

- Verificar que el progreso del juego se guarda automáticamente a intervalos regulares o en puntos clave.
- Confirmar que el guardado automático no interrumpe la experiencia del jugador.
- Comprobar que el guardado no hace conflicto con alguna otra mecánica del juego.
- Opcionalmente agregar algún indicativo visual al jugador mientras esté en progreso el autoguardado.

RF28: Sistema de préstamos

Como jugador
Quiero poder conseguir puntos sin tenerlos
Para acceder antes a mejoras, sin tener que esperar tanto y pagarlo más adelante.

Pruebas de aceptación

- Probar el proceso de solicitar y pagar préstamos en el juego.
- Asegurarse de que el sistema funcione correctamente y que los préstamos se otorguen y paguen según lo previsto.
- Comprobar que existe algún tipo de penalización al no pagar los préstamos en un tiempo o momento determinado.

RF29: Medidor de saciedad

Como jugador
Quiero una mecánica que regule la cantidad de clics que tengo que hacer
Para poder tomarme descansos y utilizar otras mecánicas del juego.

Pruebas de aceptación

- Confirmar que el medidor de saciedad refleje de manera precisa el nivel de satisfacción del líder.
- Verificar que el anti-autoclicker responda adecuadamente a los intentos de automatizar la interacción con el juego.
- Comprobar que alimentar al líder cuando ya esté saciado otorga menos puntuación.
- Opcionalmente agregar algún evento o interacción negativa cuando el líder lleva mucho con el medidor vacío.

RF30: Asistente de ayuda

Como jugador
Quiero un asistente que me guíe
Para saber qué siguiente paso tomar en el juego.

Pruebas de aceptación

- Probar la interacción con el asistente a lo largo del juego para confirmar que proporciona orientación y consejos relevantes.
- Verificar que al final del juego, el asistente se revele contra el jugador y se una al líder de acuerdo con la descripción del requisito.
- Comprobar que el asistente no es demasiado invasivo para el jugador y no perturba demasiado la experiencia del juego base.
- Confirmar que existe una alternativa a que se revele el asistente (relacionado con el final bueno).

RF31: Descontento del líder desemboca en modificaciones de archivo de guardado o similares

Como jugador
Quiero que el juego rompa la cuarta pared
Para hacer la experiencia del juego más divertida y diferente a otros productos del mercado.

Pruebas de aceptación

- Si no le das comida en más de x tiempo (por ejemplo, tres horas usando la aplicación) empieza un evento que pueda cambiar algún archivo de guardado o cerrar el juego.
- Si le das comida en menos del tiempo x, no ocurre nada.
- Cuando ocurra el evento, se notificará al usuario de alguna forma antes de que ocurra.

RF32: Enemigos y recompensa al eliminarlos

Como jugador
Quiero enemigos que penalicen mientras existan y recompensen al eliminarlos
Para romper la monotonía que pueda llegar a tener el juego y tener una mecánica entretenida más.

Pruebas de aceptación

- Si aparecen los enemigos, ocurre una penalización hasta que se eliminan.
- Si no aparecen, no ocurre nada.
- Si los clicas menos veces de las necesarias para eliminarlos, continúan la penalización.
- Si los eliminas, te dan una recompensa y termina la penalización.

RF33: Tienda oculta desbloqueable

Como jugador
Quiero una tienda secreta que dé recompensas especiales
Para aumentar la curva de progresión y añadir contenido interesante al juego.

Pruebas de aceptación

- Para desbloquear la tienda, es necesario usar una característica oculta.
- Si esta característica no se ha usado, no existirá la tienda.
- Al comprar algún objeto, la divisa disminuirá y el objeto se añadirá al inventario.
- Si no se compra un objeto no ocurre nada.
- Una vez desbloqueada, se tendrá que abrir la tienda con un botón en el menú o similar.

RF34: Existencia de método secreto para asesinar al líder

Como jugador
Quiero una forma de derrocar al líder
Para tener una mecánica distinta que permita una historia alternativa.

Pruebas de aceptación

- Una vez realizado el método, se iniciará un evento en el que el líder muera y se cambiará por tu personaje.
- Si este método no se ha realizado, no ocurrirá nada.

RF35: Trucos al tocar melodía en un piano

Como jugador
Quiero trucos desbloqueables con una secuencia de teclas secreta
Para hacer la experiencia menos monótona y más divertida.

Pruebas de aceptación

- Si se toca una cierta melodía, ocurrirá un evento en el juego.
- Si no se toca la melodía designada, o no se toca nada, no ocurre nada.
- Una vez usada una melodía, al volver a tocarla no ocurrirá nada.

RF36: Existencia de más de un modo de juego

Como jugador
Quiero distintos modos de juego
Para romper la monotonía del modo principal y poder dedicarle más horas al juego.

Pruebas de aceptación

- En cada modo debe haber una opción para cambiar a un modo distinto.
- Si no se usa la opción para cambiar el modo, se seguirá en el actual indefinidamente.
- Cada modo tendrá características que lo diferencien de los demás.

RF37: Sistema de compra de cosméticos

Como jugador
Quiero un sistema de compra de cosméticos
Para poder personalizar la interfaz del juego.

Pruebas de aceptación

- Si se compra un cosmético, este debe salir desbloqueado en el menú correspondiente.
- Si no se compra el cosmético, saldrá bloqueado y posiblemente oculto.
- Una vez comprado, el cosmético debe poder activarse y cambiarse desde el menú.

RF38: Sistema de estadísticas

Como jugador
Quiero un sistema de estadísticas
Para saber el progreso que he hecho en el tiempo que he estado jugando.

Pruebas de aceptación

- El sistema de estadística se abrirá solamente pulsando el botón designado.
- Si no se pulsa el botón, no ocurre nada.
- Los valores estadísticos se actualizarán automáticamente.

RF39: Menú de ajustes

Como jugador
Quiero un menú de ajustes
Para personalizar la configuración del juego a mis preferencias.

Pruebas de aceptación

- El menú se abre con un botón en la pantalla principal.
- Si no se pulsa el botón, no ocurre nada.
- Una vez en el menú, debe haber diferentes botones que permitan personalizar los ajustes del juego.

RF40: Música que esté sincronizada con los clics y que se pueda activar y desactivar

Como jugador
Quiero música especial
Para mejorar la experiencia del juego.

Pruebas de aceptación

- La música debe sonar automáticamente sin necesidad de activarla.
- Debe haber posibilidad de desactivarla en el menú de ajustes anterior.
- Si no se hace clic, la música permanecerá en su modo normal.
- Si se hace clic, la música sufrirá un cambio con relación al clic.

RF41: Leaderboard

Como jugador
Quiero una tabla con los mejores jugadores
Para comparar mis estadísticas con las de los mejores jugadores.

Pruebas de aceptación

- En la tabla solo se mostrarán los mejores jugadores y el perfil observador, en caso de que no esté en los mejores valores.
- Cada jugador que ha usado el programa debe poder salir en la leaderboard.

5.2. Requisitos no funcionales

RNF1: Mejoras cambian la estética de entorno y líder

Como jugador
Quiero notar que mis mejoras aparecen de forma visual
Para hacer más amena la progresión y querer seguir mejorando.

Pruebas de aceptación

- Comprando ciertas mejoras cambiará algún aspecto de la pantalla, pero no taparán ninguna información.
- Hay mejoras que son compatibles y se podrán ver las dos por pantalla.
- Hay mejoras que no son compatibles y una sustituirá a la otra.

RNF2: Prestigiar cambia la estética general

Como jugador
Quiero cambios visuales al hacer prestigio
Para notar que merece la pena llegar a ese punto del juego ya que es una mejora difícil.

Pruebas de aceptación

- Llegando a cierto nivel de prestigio aparecerán detalles en la pantalla sin tapar información.
- Si el prestigio vuelve a bajar esos detalles pueden desaparecer.

RNF3: Rendimiento

Como jugador
Quiero que funcione bien el juego
Para que la experiencia sea satisfactoria y divertida.

Pruebas de aceptación

- El juego deberá mantener unos FPS estables en la medida de lo posible.
- Si se detecta que el juego va muy lento se eliminarán algunos efectos visuales de la pantalla que no sean muy perceptibles.
- Si el juego no puede seguir ejecutándose se cerrará y lanzará un mensaje de error.

RNF4: Disponibilidad multiplataforma

Como jugador
Quiero poder jugar en distintos dispositivos
Para llevar mi progreso a todos mis dispositivos y jugar en cualquier parte.

Pruebas de aceptación

- El juego podrá jugarse en las principales plataformas como móvil o tablet, ordenador, consola y también en la web.
- Todas las funcionalidades del juego estarán disponibles y funcionales en cada plataforma compatible.
- Los datos de una partida guardada podrán volver a abrirse en cualquier plataforma.

RNF5: Adaptabilidad a la pantalla

Como jugador
Quiero que el juego se adapte a como tengo puesta la pantalla de forma cómoda y sin errores
Para jugar cómodamente desde cualquier posición en cualquier postura

Pruebas de aceptación

- La interfaz del juego se ajustará dinámicamente para adaptarse a diferentes orientaciones de pantalla (horizontal y vertical).
- Los controles táctiles se escalarán adecuadamente para ser utilizables en pantallas de diferentes tamaños.
- Los elementos de la interfaz no se solaparán en ningún tamaño de pantalla llegando a eliminar alguno si fuese necesario.

RNF6: Referencias a la cultura popular

Como jugador
Quiero sorprenderme con referencias a cosas que conozco
Para divertirme espontáneamente en ciertos momentos del juego, haciendo que me salga una sonrisa o carcajada.

Pruebas de aceptación

- Durante el avance del juego podrán aparecer algunas referencias como algún nombre o algún apartado estético.
- Las referencias de la cultura popular estarán integradas de manera orgánica en el juego.
- Las referencias serán un simple guiño y no deberán afectar a ninguna funcionalidad del juego.

RNF7: Estética visual atractiva

Como jugador
Quiero un entorno cómodo y que me incite a jugar
Para no sentirme abrumado con demasiada información o con animaciones o estética cutre que me saque del juego.

Pruebas de aceptación

- Las animaciones serán suaves y bien ejecutadas, sin artefactos visuales o problemas de sincronización.
- Todo lo que se vea en pantalla debe transmitir información al jugador sobre su progreso en el juego.

RNF8: No necesita login con cuenta

Como jugador
Quiero no tener que iniciar sesión
Para poder jugar sin necesidad de preocuparme tener una cuenta.

Pruebas de aceptación

- No se deberá guardar ninguna información no necesaria del usuario más allá de su progreso.
- Si se cambia de dispositivo se le dará un código al usuario que contenga todo su progreso y que pueda meter en la aplicación o web para seguir con su partida.

RNF9: Existencia de *Easter eggs*

Como jugador
Quiero que haya secretos en el juego
Para mejorar la experiencia y estar atento a todo lo que ocurra.

Pruebas de aceptación

- Los *Easter eggs* no deben estar a la vista y se deberán hacer acciones específicas para que aparezcan.
- Los *Easter Eggs* serán variados y sorprendentes para mantener el interés de los jugadores.
- Los *Easter Eggs* serán estéticos y no afectarán en nada a la jugabilidad.

RNF10: Posibilidad de juego sin conexión

Como jugador
Quiero poder jugar sin conexión a internet
Para poder usar el juego en cualquier situación.

Pruebas de aceptación

- El juego no necesitará internet para poder jugarse.
- Todas las acciones del juego podrán hacerse sin conectarse a la red.

RNF11: Interfaz de usuario intuitiva

Como jugador
Quiero poder entender fácilmente todos los elementos de la interfaz
Para poder interactuar con el juego de manera sencilla.

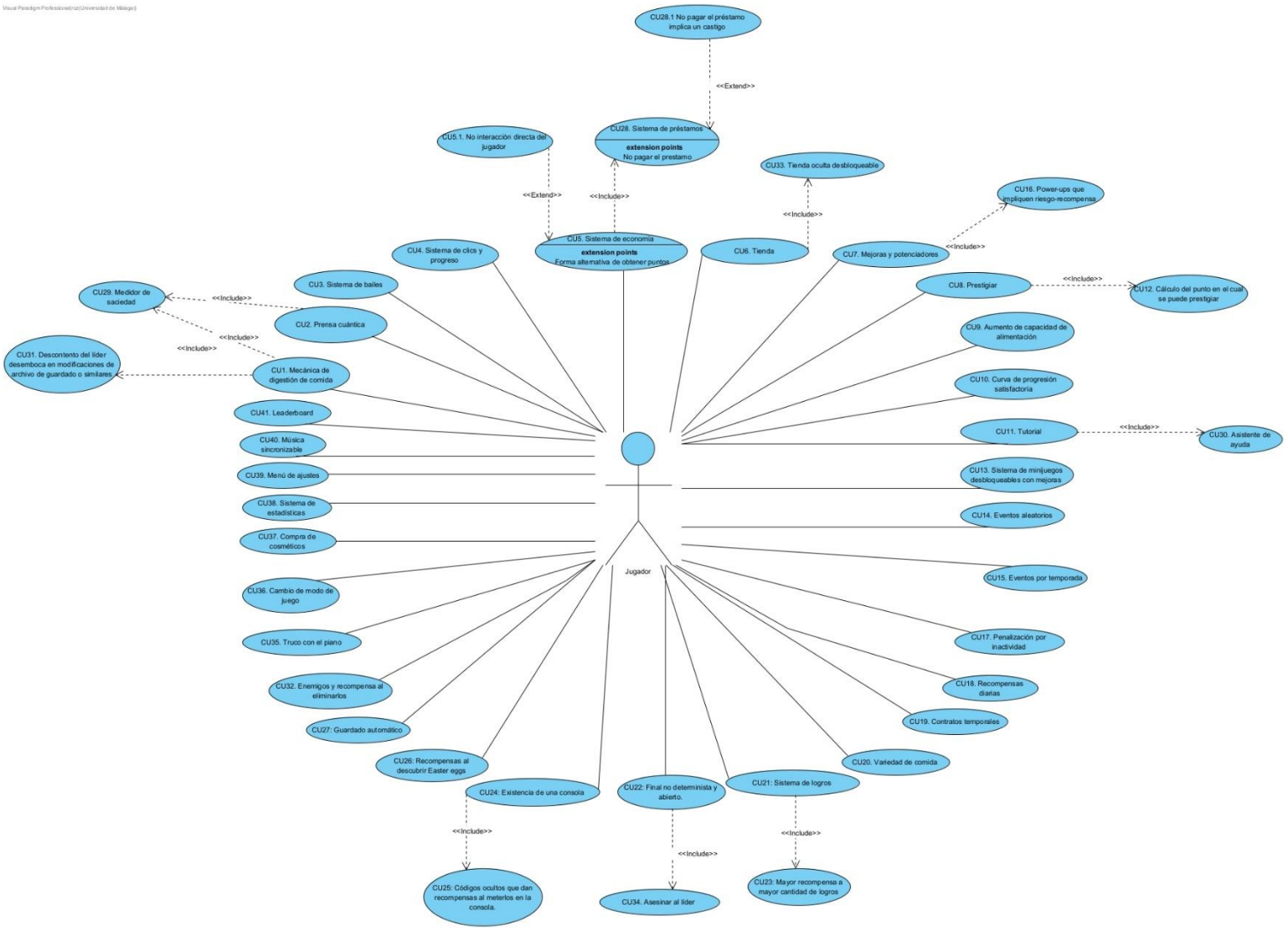
Pruebas de aceptación

- Los botones y controles del juego deben estar claramente etiquetados en el menú de ajustes y ser fácilmente reconocibles por el jugador.
- La navegación entre las diferentes secciones del juego (tienda, menú principal, pantalla de juego, etc.) debe ser intuitiva y fluida.

6. Casos de uso

La sección de *Casos de uso* es una piedra angular en la documentación de sistemas, ya que ofrece una descripción detallada de las interacciones entre los usuarios finales y el sistema en cuestión. En este apartado, se enumeran y describen exhaustivamente las diversas funciones y procesos, lo que permite a los desarrolladores y partes interesadas comprender claramente el flujo de trabajo y las capacidades del sistema. A continuación, mostraremos los casos de uso detallados para la realización de nuestro proyecto.

Como complemento visual, presentamos un diagrama de casos de uso elaborado con Visual Paradigm. Este diagrama ofrece una representación gráfica de cómo los usuarios finales interactúan con el sistema.



12 Diagrama de casos de uso en Visual Paradigm

CU1. Mecánica de digestión de comida.

- **Contexto de uso**
  - El jugador desea participar en una mecánica del juego que le mantenga entretenido mientras juega y le motive a seguir mejorando.
- **Precondiciones y activación**
  - El jugador está activo en el juego y desea mejorar la experiencia en el juego.
- **Garantías de éxito / Postcondición**
  - El jugador disfruta de una mecánica interesante que le incentiva a seguir jugando y a invertir tiempo en la experiencia del juego para una mayor recompensa.
- **Escenario principal**
  - El jugador está interactuando con el juego.
  - El jugador realiza clics sobre el líder.
  - El líder muestra signos de digestión de la comida por cada clic que recibe del jugador.
  - El jugador recibe recompensas por la alimentación exitosa del líder.
- **Escenarios alternativos**
  - El jugador no alimenta al líder con la suficiente frecuencia y no recibe las recompensas esperadas.
  - El líder muestra signos de insatisfacción si no se le alimenta regularmente.

CU2. Prensa cuántica.

- **Contexto de uso**
  - El jugador desea mejorar la capacidad de alimentación del líder para avanzar más rápido en el juego.
- **Precondiciones y activación**
  - El jugador ha acumulado suficientes puntos de fe para comprar la Prensa Cuántica desde la tienda del juego.
  - El jugador está activo en el juego y decide invertir sus recursos en esta mejora.
- **Garantías de éxito / Postcondición**
  - El jugador adquiere la Prensa Cuántica y experimenta un aumento significativo en la cantidad de calorías que el líder puede consumir por cada clic.
- **Escenario principal**
  - El jugador accede a la tienda dentro del juego.
  - El jugador selecciona la opción para comprar la prensa cuántica.
  - El sistema verifica si el jugador tiene los recursos necesarios para la compra.
  - Si el jugador tiene suficientes recursos, se realiza la transacción y se le entrega la Prensa Cuántica.
  - El jugador confirma la adquisición y vuelve al juego principal.
  - Con la Prensa Cuántica en posesión, el jugador realiza clics en el líder.
  - El líder ahora consume una cantidad mayor de calorías por cada clic, gracias a la mejora proporcionada por la prensa cuántica.
- **Escenarios alternativos**
  - Si el jugador no tiene suficientes recursos para comprar la Prensa Cuántica, se muestra un mensaje indicando que la compra no puede completarse.

CU3. Sistema de Bailes.

- **Contexto de uso**
  - El jugador busca una característica divertida y entretenida que le permita interactuar de manera única con el personaje principal del juego.
- **Precondiciones y activación**
  - El jugador está activo en el juego y desea explorar nuevas funcionalidades.
  - El jugador tiene acceso a la opción de activar el Sistema de Bailes para el líder desde la interfaz del juego.
- **Garantías de éxito / Postcondición**
  - El jugador disfruta de la capacidad de hacer que el líder realice diferentes bailes divertidos y ridículos.
- **Escenario principal**
  - El jugador accede al menú de opciones o configuración dentro del juego.
  - El jugador selecciona la opción para activar el Sistema de Bailes para el líder.
  - Una vez activado, el líder comienza a realizar bailes automáticamente en respuesta a los clics del jugador.
  - Cada clic del jugador activa un nuevo baile o una nueva animación del líder.
  - El jugador se divierte viendo al líder realizar diferentes movimientos y bailes ridículos.
- **Escenarios alternativos**
  - Si el jugador decide desactivar el Sistema de Bailes, el líder regresa a su estado normal sin realizar movimientos adicionales.

CU4. Sistema de clics y progreso.

- **Contexto de uso**
  - El jugador busca un sistema que le permita avanzar y progresar en el juego a través de la mecánica de clics.
- **Precondiciones y activación**
  - El jugador está activo en el juego y desea iniciar o continuar su progresión.
  - Se inicia el juego y se permite al jugador interactuar con el líder.
- **Garantías de éxito / Postcondición**
  - El jugador experimenta una sensación de avance y progreso conforme realiza más clics en el juego.
- **Escenario principal**
  - El jugador inicia el juego y comienza a realizar clics sobre el líder.
  - Cada clic realizado por el jugador para alimentar al líder aumenta la cantidad de puntos de fe.
  - El sistema registra el número de puntos de fe y muestra el progreso actual del jugador.
  - A medida que el jugador acumula más puntos, podrá emplearlos para obtener nuevas funcionalidades, mejoras o recompensas en el juego.
  - El jugador experimenta una sensación de satisfacción al ver su progreso reflejado en el juego.
- **Escenarios alternativos**
  - Si el jugador deja de interactuar por un período de tiempo, el sistema puede mostrar notificaciones o recordatorios para alentar al jugador a continuar su progresión, o incluso podrá penalizarlo.

CU5. Sistema de economía.

- **Contexto de uso**
  - El jugador busca una forma clara y motivadora de emplear sus puntos de fe dentro del juego, así como la posibilidad de aumentar su acumulación de puntos y obtener beneficios incluso sin interacción directa.
- **Precondiciones y activación**
  - El jugador ha acumulado puntos de fe durante su interacción con el juego.
  - El jugador está activo en el juego y desea utilizar sus puntos de fe para obtener mejoras.
- **Garantías de éxito / Postcondición**
  - El jugador puede ver claramente sus puntos de fe, disponibles en todo momento, y tiene la capacidad de emplearlos para mejorar su experiencia en el juego.
- **Escenario principal**
  - El juego muestra de manera visible y clara la cantidad de puntos de fe disponibles para el jugador en todo momento, por ejemplo, en una esquina de la pantalla.
  - El jugador accede a la tienda o al menú de mejoras dentro del juego.
  - El jugador explora las diferentes opciones de mejoras disponibles para adquirir utilizando sus puntos de fe.
  - El sistema verifica si el jugador tiene suficientes puntos de fe para realizar la compra.
  - Si el jugador tiene suficientes puntos, se realiza la transacción y se aplica la mejora seleccionada.
  - Con la compra de ciertos ítems/mejoras, el jugador también obtiene una bonificación en la cantidad de puntos de fe que puede ganar por cada clic realizado en el juego.
  - El jugador experimenta un aumento en su capacidad de acumular puntos de fe por cada interacción exitosa con el juego.
- **Escenarios alternativos**
  - Si el jugador no tiene suficientes puntos de fe para realizar una compra, se muestra un mensaje indicando que la compra no puede completarse.
  - El juego ofrece automatizaciones o mejoras que permiten al jugador obtener puntos de fe sin la participación del usuario.

CU6. Tienda.

- **Contexto de uso**
  - El jugador busca una experiencia interactiva y participativa donde pueda emplear sus puntos de fe para adquirir artículos visibles y tangibles dentro del juego.
- **Precondiciones y activación**
  - El jugador tiene acumulados puntos de fe durante su interacción con el juego.
  - El jugador está activo en el juego y desea explorar la tienda para realizar compras.
- **Garantías de éxito / Postcondición**
  - El jugador puede utilizar sus puntos de fe de manera efectiva en la tienda del juego para comprar artículos visibles, experimentando una sensación de progresión y participación en la experiencia del juego.
- **Escenario principal**
  - El jugador accede a la sección de la tienda dentro del juego desde el menú principal o desde un área específica designada.
  - Dentro de la tienda, los elementos están organizados de manera lógica y clara para que el jugador pueda navegar y explorar fácilmente las opciones disponibles.

- El jugador examina los diferentes artículos disponibles en la tienda y decide qué artículo desea comprar utilizando sus puntos de fe.
- Al seleccionar un artículo para comprar, el sistema verifica si el jugador tiene suficientes puntos de fe para realizar la compra.
- Si el jugador tiene suficientes puntos, se realiza la transacción y la cantidad de puntos de fe del jugador se reduce en la cantidad correspondiente.
- Una vez completada la compra, el artículo adquirido se hace disponible inmediatamente en el inventario o en el entorno visible del juego, según el tipo de artículo.
- El jugador puede ver físicamente el artículo adquirido dentro del juego, lo que proporciona una sensación tangible de progresión y participación.

- **Escenarios alternativos**
  - Si el jugador no tiene suficientes puntos de fe para realizar una compra, se muestra un mensaje indicando que la compra no puede completarse.
  - El sistema de la tienda evita que los jugadores compren puntos de fe directamente, limitando las transacciones solo a artículos disponibles en la tienda.

CU7. Mejoras y potenciadores.

- **Contexto de uso**
  - El jugador busca una variedad de mejoras y potenciadores dentro del juego que le permitan aumentar la eficiencia de la alimentación y progresar más rápidamente.
- **Precondiciones y activación**
  - El jugador tiene acumulados puntos de fe durante su interacción con el juego.
  - El jugador está activo en el juego y desea emplear sus puntos de fe para adquirir mejoras y potenciadores.
- **Garantías de éxito / Postcondición**
  - El jugador puede utilizar sus puntos de fe de manera efectiva en la tienda del juego para adquirir mejoras y potenciadores que tengan un efecto claro y medible en el progreso del juego.
- **Escenario principal**
  - El jugador accede a la sección de la tienda dentro del juego desde el menú principal o desde un área específica designada.
  - Dentro de la tienda, el jugador navega por la categoría de Mejoras y Potenciadores.
  - El jugador examina las diferentes opciones de mejoras y potenciadores disponibles, cada uno con un efecto específico en el juego.
  - El jugador selecciona una mejora o potenciador que desea comprar utilizando sus puntos de fe.
  - El sistema verifica si el jugador tiene suficientes puntos de fe para realizar la compra.
  - Si el jugador tiene suficientes puntos, se realiza la transacción y la cantidad de puntos de fe del jugador se reduce en la cantidad correspondiente.
  - Una vez adquirida la mejora o potenciador, su efecto se aplica inmediatamente al juego, aumentando la eficiencia de la alimentación y/o el progreso del jugador.
  - El jugador puede medir claramente el impacto de la mejora o potenciador en su progreso general dentro del juego.
- **Escenarios alternativos**
  - Si el jugador no tiene suficientes puntos de fe para realizar una compra, se muestra un mensaje indicando que la compra no puede completarse.
  - El sistema garantiza que cada mejora o potenciador adquirido tenga un efecto escalable, es decir, su eficacia aumenta con cada compra realizada por el jugador.

CU8. Prestigiar.

- **Contexto de uso**
  - En algún punto de la partida, el jugador podrá reiniciar su progreso en el juego a cambio de una habilidad base que le permita mejorar de una forma notoriamente mejor.
- **Precondiciones y activación**
  - El jugador ha avanzado lo suficiente en el juego y, además, ha conseguido objetos necesarios para que este evento ocurra.
- **Garantías de éxito / Postcondición**
  - El juego se reinicia, es decir, se elimina cualquier objeto que el jugador haya adquirido.
  - Todos los puntos de fe acumulados son eliminados.
  - El jugador tiene un nuevo potenciador base.
- **Escenario principal**
  - El jugador reúne las condiciones necesarias para poder prestigiar.
  - El jugador decide prestigiar.
  - El juego se reinicia y al jugador se le otorga un potenciador base.
- **Escenarios alternativos**
  - El jugador decide no prestigiar y el juego funciona con normalidad.

CU9. Aumento de capacidad de alimentación.

- **Contexto de uso**
  - El jugador puede activar esta mejora para poder alimentar más al líder y obtener así más puntos de fe.
- **Precondiciones y activación**
  - El jugador compra la mejora. Ha de tener puntos de fe suficientes.
- **Garantías de éxito / Postcondición**
  - El valor de la mejora ha sido sustraído de los puntos de fe del jugador.
  - La mejora ha sido asignada al jugador, el líder tiene la capacidad de comer más.
  - El jugador debe poder ahora conseguir más puntos de fe.
- **Escenario principal**
  - El jugador compra la mejora en la tienda.
  - Este cambio se ve reflejado en la tienda.
  - La capacidad de alimentación del líder aumenta, lo que se traduce en que el jugador consigue más rápido puntos de fe.
- **Escenarios alternativos**
  - El jugador no compra la mejora de alimentación, el juego continúa su curso.

CU10. Curva de progresión satisfactoria.

- **Contexto de uso**
  - El juego no puede dar la sensación de que se estanca, es por eso por lo que los objetivos deben ser cada vez más desafiantes y las recompensas más jugosas.
- **Precondiciones y activación**
  - Existencia del juego.
- **Garantías de éxito / Postcondición**
  - Al ir el jugador avanzando la dificultad ha de aumentar en proporción.
- **Escenario principal**
  - El jugador avanza (obtiene puntos de fe y los invierte en la tienda).
  - Nuevos objetos aparecen en la tienda, pero son más difíciles de conseguir.
- **Escenarios alternativos**
  - No hay escenario alternativo.

CU11. Tutorial.

- **Contexto de uso**
  - El jugador tendrá la posibilidad de que un asistente le muestre cómo funcionan los diferentes aspectos del juego la primera vez que interactúe con ellos. Posteriormente podrá acceder al mismo desde una esquina de la pantalla, como si fuera una guía.
- **Precondiciones y activación**
  - Si se añade alguna característica nueva para el jugador al juego, el tutorial se activará automáticamente. Si el jugador decide activar el tutorial, deberá pulsar el botón correspondiente desde la pantalla de juego.
- **Garantías de éxito / Postcondición**
  - El jugador realiza la acción que muestra el tutorial de forma satisfactoria.
- **Escenario principal**
  - Una nueva característica aparece en el juego.
  - El tutorial aparece y explica al jugador cómo funciona.
  - El jugador realiza la acción que se explica.
  - El tutorial se cierra.
- **Escenarios alternativos**
  - El jugador abre el tutorial.
  - El tutorial aparece y explica al jugador cómo funciona la característica que este decide.
  - El jugador realiza la acción que se explica.
  - El tutorial se cierra.

CU12. Cálculo del punto en el cual se puede prestigiar.

- **Contexto de uso**
  - El prestigio debe ser algo que el jugador considere necesario, por eso, debe llegar un punto en el que el jugador se sienta estancado y su única opción sea prestigiar.
- **Precondiciones y activación**
  - El jugador debe estar lo suficientemente avanzado.
- **Garantías de éxito / Postcondición**
  - El jugador ha prestigiado.
- **Escenario principal**
  - El jugador progresa en el juego.
  - Con suficiente antelación, el asistente avisa al jugador de que en algún punto del juego va a necesitar prestigiar.
  - Se le presenta la posibilidad de prestigiar al jugador y este acepta.
- **Escenarios alternativos**
  - El jugador decide no prestigiar y continuar jugando. Debe notar que el progreso en el juego es prácticamente nulo.

CU13. Sistema de minijuegos desbloqueables con mejoras.

- **Contexto de uso**
  - El jugador tiene la posibilidad de jugar ciertos minijuegos relacionados con las mejoras disponibles.
- **Precondiciones y activación**
  - El jugador tiene que haber mejorado una mejora/generador hasta cierto nivel para que el minijuego esté disponible.
- **Garantías de éxito / Postcondición**
  - El jugador podrá terminar el juego cuando desee.
  - Cuando el jugador lo cierra, el juego no se habrá visto afectado, es decir, los puntos que se habrían conseguido con el minijuego cerrado son los mismos que se han conseguido.
- **Escenario principal**
  - El minijuego está disponible y el jugador decide jugarlo.
  - Se abre el desplegable del minijuego, que se encuentra completamente operativo.
  - El jugador cierra el minijuego y vuelve a la parte principal.
- **Escenarios alternativos**
  - El minijuego no está todavía disponible, por lo que si el jugador interactúa con el desplegable no pasa nada.

CU14. Eventos aleatorios.

- **Contexto de uso**
  - Aparece un evento por pantalla. El jugador puede ejecutar o no lo que indica el evento y será premiado o castigado según sus acciones.
- **Precondiciones y activación**
  - El jugador tiene abierto el juego.
- **Garantías de éxito / Postcondición**
  - El jugador ha completado el evento.
  - En su defecto, el jugador no ha terminado el evento y se ha pasado el tiempo establecido.
- **Escenario principal**
  - Un evento es activado aleatoriamente, interrumpiendo la monotonía del juego.
  - El jugador completa el evento y es recompensado.
  - El juego vuelve a la normalidad.
- **Escenarios alternativos**
  - El jugador no completa el evento y es castigado.
  - El juego vuelve a la normalidad.

CU15. Eventos por temporada.

- **Contexto de uso**
  - El jugador desea sentir y notar cada estación de la vida real en el juego para aumentar la conexión con el líder y el juego.
- **Precondiciones y activación**
  - El jugador debe estar dentro del juego. La temporada actual debe estar activa.
- **Garantías de éxito / Postcondición**
  - El jugador experimenta cambios en la apariencia de la interfaz del juego, se desbloquean nuevas mejoras y formas de conseguir puntos de fe.
  - Una vez finalizado el evento de temporada, las mejoras desaparecen, pero los puntos obtenidos se mantienen.
- **Escenario principal**
  - El jugador está dentro del juego.
  - El evento de temporada está activo.
  - El jugador experimenta cambios en la apariencia de la interfaz y se desbloquean nuevas mejoras y formas de conseguir puntos de fe.
  - Una vez finalizado el evento, las mejoras de temporada desaparecen, pero los puntos obtenidos se mantienen.
- **Escenarios alternativos**
  - El jugador no participa en el evento de temporada y no experimenta los cambios correspondientes.

CU16. Power-ups que impliquen riesgo-recompensa.

- **Contexto de uso**
  - El jugador desea tomar decisiones que impliquen riesgos para aumentar la intensidad del juego y obtener mejores recompensas.
- **Precondiciones y activación**
  - El jugador debe haber mejorado ciertas características del juego hasta cierto nivel. Los power-ups deben estar desbloqueados.
- **Garantías de éxito / Postcondición**
  - El jugador puede activar power-ups que proporcionan efectos positivos y negativos durante un tiempo determinado.
- **Escenario principal**
  - El jugador ha mejorado ciertas características del juego hasta cierto nivel.
  - Los power-ups están desbloqueados.
  - El jugador activa un power-up que proporciona efectos positivos y negativos.
  - El power-up tiene efecto durante un tiempo determinado.
- **Escenarios alternativos**
  - El jugador no activa ningún power-up o decide no arriesgarse a usarlos.

CU17. Penalizaciones por no jugar durante mucho tiempo.

- **Contexto de uso**
  - El juego busca incentivar la actividad regular del jugador mediante la imposición de penalizaciones por períodos prolongados de inactividad.
- **Precondiciones y activación**
  - El jugador ha estado inactivo durante un período de tiempo definido por el juego.
- **Garantías de éxito / Postcondición**
  - El jugador recibe una penalización en la producción de puntos de fe como consecuencia de su inactividad prolongada y es notificado al respecto.
- **Escenario principal**
  - El jugador ha estado inactivo durante un período prolongado según el tiempo real del mundo exterior.
  - Al ingresar al juego después de la inactividad prolongada, se le notifica al jugador sobre la penalización en la producción de puntos de fe.
  - Se advierte al jugador sobre las consecuencias de la inactividad prolongada y se le incentiva a no volver a incurrir en ella.
- **Escenarios alternativos**
  - El jugador no recibe la penalización debido a que ha estado activo dentro del período definido por el juego.

CU18. Recompensas diarias.

- **Contexto de uso**
  - El juego ofrece recompensas diarias para fomentar la participación continua del jugador y aumentar su compromiso con el juego.
- **Precondiciones y activación**
  - El jugador ingresa al juego por primera vez en el día.
- **Garantías de éxito / Postcondición**
  - El jugador recibe una recompensa diaria al ingresar al juego por primera vez en el día y se le informa sobre posibles recompensas futuras por su participación continua.
- **Escenario principal**
  - El jugador ingresa al juego por primera vez en el día.
  - Se le presenta al jugador una recompensa diaria por su participación.
  - Se informa al jugador sobre posibles recompensas futuras por su participación continua durante varios días consecutivos.
- **Escenarios alternativos**
  - El jugador decide no reclamar la recompensa diaria y continúa su sesión de juego sin hacerlo.

CU19. Contratos diarios, semanales y mensuales.

- **Contexto de uso**
  - El juego ofrece contratos diarios, semanales y mensuales para comprometer al jugador a participar en actividades específicas durante períodos prolongados de tiempo.
- **Precondiciones y activación**
  - El jugador accede a la sección de contratos temporales dentro del juego.
- **Garantías de éxito / Postcondición**
  - El jugador recibe una lista clara y detallada de los contratos disponibles para completar, junto con las recompensas asociadas a cada uno de ellos.
- **Escenario principal**
  - El jugador accede a la sección de contratos temporales dentro del juego.
  - Se le presenta al jugador una lista de contratos diarios, semanales y mensuales disponibles para completar.
  - Se detallan las recompensas asociadas a cada contrato y se indica la duración de cada uno.
- **Escenarios alternativos**
  - El jugador decide no participar en ninguno de los contratos temporales disponibles.

CU20. Existencia de diferentes tipos de comida.

- **Contexto de uso**
  - El juego ofrece una amplia variedad de comida cada vez más calórica para mantener el interés y la frescura del juego durante períodos prolongados de tiempo.
- **Precondiciones y activación**
  - El jugador accede a la tienda dentro del juego.
- **Garantías de éxito / Postcondición**
  - El jugador puede ver una lista de diferentes tipos de comida disponibles en el juego, cada uno con un Sprite único y beneficios específicos.
- **Escenario principal**
  - El jugador accede al menú de comidas disponibles en la tienda dentro del juego.
  - Se presenta al jugador una lista de diferentes tipos de comida disponibles.
  - Cada tipo de comida puede tener beneficios únicos.
- **Escenarios alternativos**
  - El jugador decide no explorar el menú de comidas disponibles en la tienda y continúa con otras actividades dentro del juego.

CU21. Sistema de logros.

- **Contexto de uso**
  - El sistema registra y muestra los logros alcanzados por el jugador durante el juego.
- **Precondiciones y activación**
  - El jugador realiza acciones específicas en el juego que cumplen los requisitos para desbloquear un logro.
- **Garantías de éxito / Postcondición**
  - El logro desbloqueado se registra en el sistema.
  - El jugador puede visualizar los logros desbloqueados en una sección dedicada.
- **Escenario principal**
  - El jugador cumple los requisitos para desbloquear un logro.
  - El sistema detecta la acción realizada por el jugador.
  - Se desbloquea el logro correspondiente.
  - El logro se registra en el sistema.
  - El jugador puede ver el logro desbloqueado en la sección de logros.
- **Escenarios alternativos**
  - No se desbloquea ningún logro si el jugador no cumple los requisitos necesarios.

CU22. Final no determinista y abierto.

- **Contexto de uso**
  - El juego no tiene un final predeterminado y ofrece múltiples rutas o posibilidades de conclusión.
- **Precondiciones y activación**
  - El jugador progresa en el juego y toma decisiones que afectan el curso de la historia.
- **Garantías de éxito / Postcondición**
  - El juego presenta múltiples finales o caminos de conclusión.
  - La historia se adapta a las decisiones tomadas por el jugador.
- **Escenario principal**
  - El jugador avanza en el juego y toma decisiones importantes.
  - Las decisiones del jugador afectan la trama y el desarrollo del juego.
  - El juego presenta diferentes finales o rutas basadas en las decisiones del jugador.
- **Escenarios alternativos**
  - No hay escenario alternativo, ya que la propia naturaleza del juego es abierta y no determinista.

CU23. Mayor recompensa a mayor cantidad de logros.

- **Contexto de uso**
  - El sistema otorga recompensas adicionales al jugador en función de la cantidad de los desbloqueados.
- **Precondiciones y activación**
  - El jugador desbloquea múltiples logros durante su progreso en el juego.
- **Garantías de éxito / Postcondición**
  - El sistema otorga recompensas adicionales al jugador por cada hito alcanzado en el sistema de logros.
  - Las recompensas pueden ser en forma de potenciadores, descuentos o cualquier otro beneficio dentro del juego.
- **Escenario principal**
  - El jugador desbloquea múltiples logros durante su partida.
  - El sistema registra la cantidad de logros desbloqueados por el jugador.
  - El jugador recibe recompensas adicionales proporcionales a la cantidad de logros desbloqueados.
- **Escenarios alternativos**
  - Si el jugador no desbloquea suficientes logros, no recibe las recompensas adicionales asociadas.

CU24. Existencia de una consola.

- **Contexto de uso**
  - El juego cuenta con una interfaz de consola dentro del juego que permite al jugador acceder a funciones especiales o información adicional.
- **Precondiciones y activación**
  - La consola del juego está disponible para el jugador.
  - El jugador accede a la consola desde el menú principal del juego o desde una combinación de teclas específica.
- **Garantías de éxito / Postcondición**
  - El jugador puede acceder a funciones especiales, información adicional o consigue recompensas por la introducción de códigos en la consola.
- **Escenario principal**
  - El jugador accede al menú principal del juego.
  - Desde el menú, el jugador selecciona la opción para abrir la consola del juego.
  - Se muestra la interfaz de la consola en pantalla.
  - El jugador puede utilizar la consola para acceder a funciones especiales o ingresar códigos.
- **Escenarios alternativos**
  - Si el jugador no accede a la consola o si está desactivada, no se podrá acceder a esta funcionalidad.

CU25. Códigos ocultos que dan recompensas al meterlos en la consola.

- **Contexto de uso**
  - El juego incluye códigos ocultos que, al ser ingresados en la consola del juego, otorgan al jugador recompensas especiales.
- **Precondiciones y activación**
  - El jugador accede a la consola del juego y tiene conocimiento del código que desea ingresar.
- **Garantías de éxito / Postcondición**
  - El jugador ingresa correctamente el código en la consola.
  - El sistema verifica el código y otorga la recompensa correspondiente al jugador.
- **Escenario principal**
  - El jugador accede a la consola del juego desde el menú principal.
  - El jugador ingresa el código oculto en la consola.
  - El sistema verifica el código.
  - Si el código es válido, el sistema otorga la recompensa asociada.
  - El jugador recibe la recompensa en el juego.
- **Escenarios alternativos**
  - Si el código ingresado es incorrecto o no válido, el sistema no otorga ninguna recompensa al jugador.

CU26. Recompensas al descubrir Easter eggs.

- **Contexto de uso**
  - El juego incluye Easter eggs o elementos ocultos que, al ser descubiertos por el jugador, otorgan recompensas especiales.
- **Precondiciones y activación**
  - El jugador realiza acciones específicas o encuentra elementos ocultos que revelan la existencia de un Easter egg.
- **Garantías de éxito / Postcondición**
  - El jugador descubre el Easter egg dentro del juego.
  - El sistema otorga al jugador una recompensa especial por descubrir el Easter egg.
- **Escenario principal**
  - El jugador realiza acciones específicas o encuentra pistas que indican la existencia de un Easter egg.
  - El jugador descubre el Easter egg dentro del juego.
  - El sistema detecta el descubrimiento del Easter egg.
  - El jugador recibe una recompensa especial por descubrir el Easter egg.
- **Escenarios alternativos**
  - Si el jugador no logra descubrir el Easter egg, no recibe la recompensa asociada.

CU27. Guardado automático.

- **Contexto de uso**
  - El juego guarda automáticamente el progreso del jugador en puntos específicos y multiplicador para evitar la pérdida de datos en caso de cierre inesperado o error.
- **Precondiciones y activación**
  - El juego está en ejecución y el jugador realiza acciones que afectan su progreso en el juego.
- **Garantías de éxito / Postcondición**
  - El juego guarda automáticamente el progreso del jugador en intervalos regulares o puntos específicos.
  - En caso de cierre inesperado o error, el jugador puede restaurar su progreso desde el último punto de guardado automático.
- **Escenario principal**
  - El jugador avanza en el juego y realiza acciones que afectan su progreso.
  - El juego guarda automáticamente el progreso del jugador en intervalos regulares.
  - En caso de cierre inesperado o error, el jugador puede restaurar su progreso desde el último punto de guardado automático.
- **Escenarios alternativos**
  - No hay escenario alternativo, ya que el sistema de guardado automático funciona de manera transparente para el jugador.

CU28. Sistema de préstamos.

- **Contexto de uso**
  - El usuario podrá pedir puntos a una especie de banco o entidad teniendo que devolverlos posteriormente (probablemente con intereses).
- **Precondiciones y activación**
  - El usuario interactúa con la opción y solicita la cantidad de puntos de fe que considere.
- **Garantías de éxito / Postcondición**
  - El jugador es provisto con los puntos de fe teniendo que devolverlos en un plazo y probablemente con una comisión.
  - El jugador en principio no puede pedir préstamos si todavía no ha pagado alguno (puede que las mejoras modifiquen este hecho).
  - El jugador puede entrar en deuda con la entidad si no paga los préstamos en el plazo. Esto generará consecuencias negativas en la partida.
- **Escenario principal**
  - El usuario está en el juego.
  - El usuario interactúa con el apartado de préstamos.
  - El usuario pide el préstamo que considere.
  - El usuario administra sus puntos y va pagando el préstamo antes de que acabe el plazo.
- **Escenarios alternativos**
  - El usuario no paga el préstamo a tiempo y es penalizado.
  - El usuario no pide el préstamo y no ocurre nada.

CU29. Medidor de saciedad.

- **Contexto de uso**
  - El jugador podrá ver en la interfaz del juego un medidor que muestre gráficamente la saciedad del líder y no le permita usar un auto-clicker.
- **Precondiciones y activación**
  - El jugador se encuentra en la pantalla de juego (el medidor está siempre activo) y está clicando al líder.
- **Garantías de éxito / Postcondición**
  - El jugador puede ver claramente el nivel de saciedad del líder con el medidor.
  - El jugador no puede usar un auto-clicker ya que al detectar muchos clics simultáneos el juego lo detectará y establecerá un límite humano de clics por segundo, llegando incluso a anular la función de clicar si detecta demasiados clics continuados.
  - El jugador gana una menor cantidad de puntos si el líder está saciado.
  - El jugador puede ser penalizado si el medidor lleva mucho vacío.
- **Escenario principal**
  - El usuario está en el juego.
  - El usuario interactúa con el líder clicando.
  - El usuario observa cómo el medidor de saciedad sube.
- **Escenarios alternativos**
  - El usuario es penalizado si el líder lleva un determinado tiempo con el medidor vacío.
  - El usuario gana menos puntos si sigue alimentando al líder una vez saciado.
  - El usuario comprueba que en caso de usar un auto-clicker no consigue mayor beneficio.

CU30. Asistente de ayuda.

- **Contexto de uso**
  - El jugador podrá interactuar y observar eventos de un asistente que le ayude a entender el juego.
- **Precondiciones y activación**
  - El jugador está en el juego y presiona algún botón u opción que haga que el asistente interactúe con él o el asistente directamente se activa sin interacción bajo ciertas condiciones.
- **Garantías de éxito / Postcondición**
  - El jugador puede interactuar y ver fácilmente los consejos del asistente.
  - El jugador obtiene ayuda que puede aplicar al juego.
  - El jugador observa que tiene una especie de relación virtual simulada con el asistente y éste puede llegar a revelarse contra él.
- **Escenario principal**
  - El jugador está en el juego.
  - El jugador interactúa con el asistente o alguna otra acción o evento lo desencadena.
  - El jugador recibe ayuda del asistente y la aplica al juego.
- **Escenarios alternativos**
  - El jugador ignora los consejos del asistente.
  - El jugador toma una serie de decisiones que llevan a que el asistente se revele.

CU31. Descontento del líder desemboca en modificaciones de archivo de guardado o similares.

- **Contexto de uso**
  - El jugador podrá hacer enfadar al líder con sus acciones y provocar que éste realice cambios en el juego.
- **Precondiciones y activación**
  - El jugador está en el juego y toma una serie de decisiones y acciones que hacen “enfadar” al líder, cuando se llegue a cierto punto éste lo castiga modificando el juego y pudiendo llegar a cambiar archivos de guardado (volver atrás en el tiempo, por ejemplo).
- **Garantías de éxito / Postcondición**
  - El jugador puede ver que sus acciones tienen consecuencias en el líder.
  - El jugador observa cómo puede llegar a hacer enfadar al líder y que éste modifique el juego.
  - El jugador obtiene una sensación de ruptura de la cuarta pared lo cual le genera inquietud.
- **Escenario principal**
  - El usuario está en el juego
  - El usuario realiza ciertas acciones como no alimentar al líder que lo hacen enfadar
  - El usuario nota como el líder se enfada y el juego se modifica
  - El usuario siente que se rompe la cuarta pared lo cual le genera inquietud y hace que disfrute más de la experiencia de juego
- **Escenarios alternativos**
  - El usuario nunca hace enfadar al líder y por tanto no ve estos eventos.

CU32. Enemigos y recompensa al eliminarlos.

- **Contexto de uso**
  - El jugador puede eliminar a una serie de enemigos los cuales si se ignoran generan penalizaciones.
- **Precondiciones y activación**
  - El jugador está en el juego y observa como aparecen enemigos que disminuyen su eficiencia, lo cual genera la decisión de querer eliminarlos.
- **Garantías de éxito / Postcondición**
  - El jugador se percata de los enemigos.
  - El jugador comprueba que la existencia de los enemigos está penalizándolo.
  - El jugador decide si eliminar o no a los enemigos.
- **Escenario principal**
  - El jugador está en el juego.
  - El jugador ve como aparecen enemigos que lo penalizan.
  - El jugador los clica y comprueba que su vida disminuye.
  - El jugador termina por eliminarlos.
  - El jugador observa que las penalizaciones desaparecen.
- **Escenarios alternativos**
  - Como funcionan por probabilidad, puede darse la casualidad de que el usuario nunca vea enemigos.
  - El jugador ignora a los enemigos porque no se percata de que son perjudiciales o por decisión propia.

CU33. Tienda oculta desbloqueable.

- **Contexto de uso**
  - El jugador descubre una tienda paralela a la original la cual contiene mejoras especiales.
- **Precondiciones y activación**
  - El jugador está en el juego y mediante una serie de pistas o por pura casualidad encuentra una característica oculta que puede usar para llegar a la tienda.
- **Garantías de éxito / Postcondición**
  - El jugador encuentra la tienda oculta con relativo esfuerzo, pero no siendo demasiado difícil.
  - El jugador se plantea comprar las mejoras de la tienda secreta porque se diferencian de las de la original.
  - El jugador desbloquea un nuevo apartado para acceder a la tienda.
- **Escenario principal**
  - El usuario está en el juego.
  - El usuario descubre una característica oculta.
  - El usuario comprueba que mediante esa característica se desbloquea una tienda secreta.
  - El usuario accede a la tienda y compra las mejoras especiales.
- **Escenarios alternativos**
  - El usuario nunca llega a desbloquear la característica.
  - El usuario no sabe usar la característica para acceder a la tienda.
  - El usuario decide no comprar ninguna mejora de la tienda oculta.

CU34. Asesinar al líder.

- **Contexto de uso**
  - En un momento de la partida, el usuario tendrá la opción de asesinar al líder.
- **Precondiciones y activación**
  - El usuario ha avanzado suficiente en el juego.
  - El usuario acumula los objetos necesarios para activar el evento.
- **Garantías de éxito / Postcondición**
  - El líder muere y el jugador pasa a ser el líder.
- **Escenario principal**
  - El usuario reúne los requisitos.
  - El usuario inicia la secuencia para matar al líder.
  - El líder muere en una cinemática.
  - El personaje del jugador pasa a ser el nuevo líder.
- **Escenarios alternativos**
  - El usuario falla en la secuencia y el líder se defiende.

CU35. Truco con el piano.

- **Contexto de uso**
  - El jugador podrá acceder a trucas tocando una secuencia de notas en un piano.
- **Precondiciones y activación**
  - El usuario ha desbloqueado el piano.
  - El usuario introduce la secuencia correcta de notas.
- **Garantías de éxito / Postcondición**
  - Se activa un evento y se desbloquea una recompensa especial.
- **Escenario principal**
  - El usuario entra a la interfaz del piano.
  - El usuario introduce la secuencia correcta.
  - El usuario recibe su recompensa.
- **Escenarios alternativos**
  - El usuario introduce una secuencia incorrecta y no pasa nada.

CU36. Cambio de modo de juego.

- **Contexto de uso**
  - El usuario podrá cambiar de modo de juego en la partida.
- **Precondiciones y activación**
  - El usuario ha desbloqueado el modo de juego.
- **Garantías de éxito / Postcondición**
  - El modo de juego se cambia.
- **Escenario principal**
  - El usuario le da al botón de cambiar modo de juego.
  - Se guardan los datos del modo al que esté jugando.
  - Se cambia el modo de juego.
- **Escenarios alternativos**
  - No hay escenario alternativo.

CU37. Compra de cosméticos.

- **Contexto de uso**
  - El usuario tendrá la posibilidad de comprar cosméticos para el líder.
- **Precondiciones y activación**
  - El usuario tiene suficientes puntos como para pagar el cosmético.
- **Garantías de éxito / Postcondición**
  - Los puntos se reducen y se añade el cosmético al inventario.
- **Escenario principal**
  - El usuario compra el cosmético.
  - Los puntos se reducen y se añade el cosmético al inventario.
  - El usuario tendrá la posibilidad de equipar el cosmético al líder.
- **Escenarios alternativos**
  - El usuario no tiene suficientes puntos y no ocurre nada.

CU38. Sistema de estadísticas.

- **Contexto de uso**
  - El usuario podrá acceder a un sistema de estadísticas por un botón en el menú.
- **Precondiciones y activación**
  - El usuario le dará a un botón del menú.
- **Garantías de éxito / Postcondición**
  - El menú se abre y se muestran las estadísticas.
- **Escenario principal**
  - El usuario está en el juego.
  - El usuario pulsa el botón de estadísticas.
  - El menú de estadísticas se abre.
- **Escenarios alternativos**
  - La conexión con la base de datos falla y se muestra un error.

CU39. Menú de ajustes.

- **Contexto de uso**
  - El usuario podrá abrir un menú de ajustes para personalizar el juego.
- **Precondiciones y activación**
  - El usuario le dará a un botón de la interfaz para abrir el menú.
- **Garantías de éxito / Postcondición**
  - El menú se abre y se muestran los ajustes y las opciones personalizables.
- **Escenario principal**
  - El usuario está en el juego.
  - El usuario pulsa el botón para abrir el menú de ajustes.
  - El menú se abre y se muestran las opciones disponibles.
- **Escenarios alternativos**
  - No hay casos alternativos.

CU40. Música que esté sincronizada con los clics y que se pueda activar y desactivar.

- **Contexto de uso**
  - El usuario tendrá la opción de sincronizar la música con sus clics.
- **Precondiciones y activación**
  - El usuario activará la opción y recibirá posibles recompensas por seguir el ritmo de la música.
  - El usuario le dará a un botón de la interfaz para abrir el menú.
- **Garantías de éxito / Postcondición**
  - El usuario es avisado de que está siguiendo el ritmo y recibe su recompensa.
  - El menú se abre y se muestran los ajustes y las opciones personalizables.
- **Escenario principal**
  - El usuario está en el juego.
  - El usuario activa el evento de música sincronizable.
  - El usuario clica siguiendo el ritmo de la música.
  - El usuario es avisado y recibe su recompensa.
- **Escenarios alternativos**
  - El usuario no sigue el ritmo correctamente y es penalizado.

CU41. Leaderboard

- **Contexto de uso**
  - El jugador consulta la leaderboard para compararse con el resto de los usuarios de su equipo y del mundo.
- **Precondiciones y activación**
  - El jugador está en el juego y pulsa un botón que lo lleva a la leaderboard.
- **Garantías de éxito / Postcondición**
  - El jugador encuentra la característica de la leaderboard.
  - El jugador la consulta a menudo para compararse con otros jugadores.
  - El jugador es motivado por esta característica a mejorar y seguir jugando.
- **Escenario principal**
  - El usuario está en el juego.
  - El usuario abre la leaderboard.
  - El usuario la consulta y se compara con el resto de los jugadores.
  - El usuario lucha por escalar puestos.
- **Escenarios alternativos**
  - El usuario nunca llega a interactuar con esta característica.
  - El usuario no entiende o no sabe interpretar la leaderboard.
  - El usuario no es competitivo o simplemente no hace nada en base a la información que le ofrece la leaderboard.

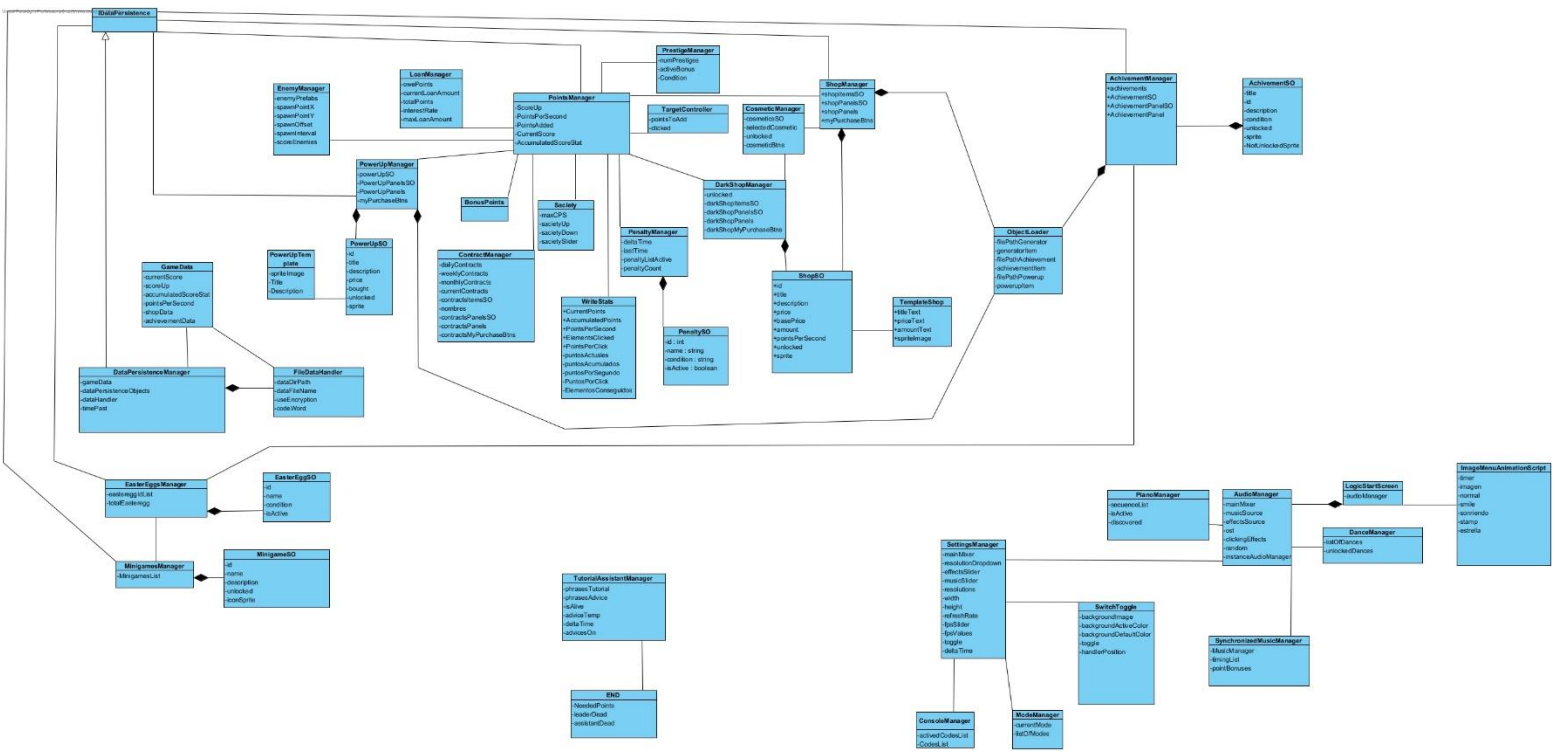
7. Modelo de dominio

El modelo de dominio es una **representación conceptual de las entidades relevantes y las relaciones** entre ellas dentro de un determinado dominio de una aplicación. Es decir, es una forma de visualizar y comprender la estructura del sistema, que se centra en los conceptos claves y sus interacciones.

En este contexto, los diagramas de clase son utilizados comúnmente para representar el modelo, ya que muestran las clases, que representan las entidades en el dominio, junto con sus atributos y relaciones, que reflejan cómo interactúan dichas entidades entre sí.

Esta sección es de gran importancia, pues proporciona una base sólida para el diseño y la implementación del software. Ayuda a los desarrolladores a comprender la naturaleza del problema que se quiere abordar y a identificar las relaciones clave que deben ser consideradas en la construcción del sistema.

Para visualizar mejor esta sección, presentamos un diagrama de clases elaborado con Visual Paradigm en el que se desglosan las relaciones entre los sistemas que conforman nuestro juego.



13 Diagrama de clases para el modelo de dominio en Visual Paradigm

Sistema de tienda

ShopManager

Esta clase se encarga de administrar toda la lógica de la tienda: sus objetos, orden, los que hay disponibles, los que se van comprando, etc…

Atributos:

- shopItemsSO: array con los ítems ScriptableObject de la tienda.
- shopPanelsSO: array con los paneles ScriptableObject de la tienda.
- shopPanels: array con los paneles de la tienda.
- myPurchaseBtns: array con los botones de la tienda.

Relaciones de composición:

- ObjectLoader: es necesaria esta clase para cargar los objetos con sus valores predeterminados (cargar por primera vez) en la tienda.
- ShopSO: cada objeto de la tienda corresponde a un ScriptableObject.

Relaciones de asociación:

- PointsManager: la tienda usa los puntos para comprar sus objetos.
- CosmeticManager: Los cosméticos que administra CosmeticManager pueden ser adquiridos en la tienda que administra ShopManager.
- IDataPersistence — descrita en la clase correspondiente.

ShopSO

Clase del tipo ScriptableObject que define los objetos de la tienda.

Atributos:

- id: id del objeto.
- title: nombre del objeto.
- description: descripción del objeto.
- price: precio del objeto.
- basePrice: precio base del objeto.
- amount: cantidad del objeto de la que se dispone.
- pointsPerSecond: puntos que otorga por segundo el objeto o generador.
- unlocked: booleano que determina si se ha desbloqueado o no el objeto.
- sprite: sprite del objeto para que se muestre en la tienda.

Relaciones de composición:

- ShopManager y DarkShopManager — descritas en las clases correspondientes.

Relaciones de asociación:

- TemplateShop: cada objeto de la tienda usa una plantilla para que se cree y muestre en el juego.

TemplateShop

Plantilla usada para crear los objetos de la tienda.

Atributos:

- titleText: cuadro para que se muestre el texto del nombre cada objeto.
- pirceText: cuadro para que se muestre el texto del precio de cada objeto.
- amountText: cuadro para que se muestre el texto de la cantidad de cada objeto.
- spriteImage: imagen para que se muestre el sprite de cada objeto.

Relaciones de asociación:

- ShopSO — descrita en la clase correspondiente.

DarkShopManager

Similar a ShopManager pero se ocupa de administrar la tienda oculta.

Atributos:

- unlocked: booleano que controla si se ha desbloqueado la tienda oculta o no.
- darkShopItemsSO: array con los ítems ScriptableObject de la tienda.
- darkShopPanelsSO: array con los paneles ScriptableObject de la tienda.
- darkShopPanels: array con los paneles de la tienda.
- darkShopMyPurchaseBtns: array con los botones de la tienda.

Relaciones de composición:

- ShopSO: son los ScriptableObject que usa la tienda oculta para sus ítems (al igual que la normal).

Relaciones de asociación:

- pointsManager: la tienda oculta usa los puntos para comprar sus objetos al igual que la normal.
- CosmeticManager — descrita en la clase correspondiente.

CosmeticManager

Clase que se encarga de administrar el cosmético elegido.

Atributos:

- cosmeticsSO: array con los ScriptableObject para cada cosmético.
- selectedCosmetic: variable que controla el cosmético actualmente seleccionado (en principio se podrá tener solo un cosmético al mismo tiempo).
- unlocked: array de booleanos o similar que determina los cosméticos que están desbloqueados.
- cosmeticBtns: array con los botones para activar o desactivar los cosméticos.

Relaciones de asociación:

- ShopManager: los cosméticos serán objetos que se puedan comprar en la tienda.
- DarkShopManager: algún cosmético especial se comprará en la tienda oculta.

Sistema de mejoras

PowerUpManager

Clase que se encarga de gestionar los diferentes power-ups que aparecen en el juego.

Atributos:

- powerUpSO: array con los power-ups disponibles.
- powerUpPanelsSO: array con los paneles ScriptableObject que contienen los power-ups.
- powerUpPanels: array con los paneles que contienen los power-ups.
- myPurchaseBtns: array con los botones de los power-ups.

Relaciones de asociación:

- PointsManager: dado que los power-ups generan cambios en la producción de puntos, es necesaria esta relación.
- IDataPersistence — descrita en la clase correspondiente.

Relaciones de composición:

- PowerUpSO: Son los Scriptable Objects sobre los cuales trabaja PowerUpManager, encrargándose de realizar todas las tareas relacionadas con ellos.
- ObjectLoader: Los objetos PowerUpSO se cargan desde un archivo csv que carga ObjectLoader en el editor de Unity.

PowerUpSO

Clase del tipo ScriptableObject que genera un objeto power-up.

Atributos:

- id: identificador del power-up.
- title: nombre que recibe el power-up.
- description: variable que contiene la descripción del funcionamiento del power-up.
- price: coste del power-up.
- bought: indica si el power-up ha sido comprado o no.
- unlocked: indica si el jugador puede comprar o no el power-up.
- sprite: imagen del power-up.

Relaciones de asociación:

- PowerUpTemplate — descrita en la respectiva clase.

Relaciones de composición:

- PowerUpManager: los diferentes objetos creados por PowerUpSO son gestionados por esta clase.

PowerUpTemplate

Esta clase gestiona la parte visual de los power-ups.

Atributos:

- spriteImage: imagen del power-up.
- Title: nombre que recibe el power-up.
- Description: variable que contiene la descripción del funcionamiento del power-up.

Relaciones de asociación:

- PowerUpSO: esta clase usa PowerUpTemplate para crear correctamente los power-ups.

Sistema de logros

AchievementManager

Clase que controla y gestiona todo el sistema de logros. Controla la carga, almacenamiento y visualización de los logros obtenidos por el jugador.

Atributos:

- achievements: una lista de logros obtenidos por el jugador.
- AchievementSO: un array de objetos de tipo AchievementSO que representan los logros en formato ScriptableObject.
- AchievementPanelSO: un array de GameObjects que representan los paneles de los logros en formato ScriptableObject.
- AchievementPanel: un array de objetos de tipo AchievementTemplate que representan los paneles de logros en la interfaz de usuario.

Relaciones de asociación:

- ObjectLoader: indica que AchievementManager usa ObjectLoader para cargar los logros desde los archivos especificados en las rutas.
- EasterEggManager: algunos logros son desbloqueados con easter-eggs.
- IDataPersistence — descrita en la clase correspondiente.

Relaciones de composición:

- AchievementSO — descrita en la clase correspondiente.
- ObjectLoader: indica que AchievementManager usa ObjectLoader para cargar los logros desde los archivos especificados en las rutas.

AchievementSO

Esta clase del tipo ScriptableObject representa un logro (achievement) en la aplicación. Contiene información sobre el logro, como su título, descripción, condición para desbloquearlo y su estado de desbloqueo.

Atributos:

- title: el título del logro.
- id: el identificador único del logro.
- description: la descripción del logro.
- condition: la condición que debe cumplirse para desbloquear el logro.
- unlocked: un indicador que señala si el logro ha sido desbloqueado o no.
- sprite: el sprite asociado al logro cuando está desbloqueado.
- NotUnlockedSprite: el sprite asociado al logro cuando no está desbloqueado.

Relaciones de composición:

- AchievementManager: los AchievementSO son gestionados y administrados por AchievementManager.

Sistema de puntuación

PointsManager

Esta clase se encarga de controlar todo el sistema de puntos del juego.

Atributos:

- ScoreUp: puntos a sumar por click.
- PointsPerSecond: puntos a sumar cada segundo.
- PointsAdded: variable auxiliar para actualizar los puntos.
- CurrentScore: puntuación actual de jugador.
- AccumulatedScoreStat: puntos conseguidos por el jugador desde que inició el juego.

Relaciones de asociación:

- TargetController: los eventos aleatorios afectarán a la puntuación.
- ShopManager: cualquier compra modificará la puntuación.
- DarkShopManager: las compras en esta tienda también afectarán a la puntuación.
- PenaltyManager: las penalizaciones restarán puntuación.
- WriteStats: la información para las estadísticas es sacada de PointsManager.
- Satiety: que el líder esté saciado repercutirá en la puntuación.
- ContractsManager: el cumplimiento de contratos dará recompensas que se verán reflejadas en la puntuación.
- BonusPoints: añadirá puntos por la ejecución de una acción.
- PowerUpManager: las mejoras afectarán a la puntuación
- EnemyManager: la eliminación de enemigos o los que nos hagan modificarán la puntuación
- LoanManager: los préstamos afectarán al puntaje.
- PrestigeManager: hacer prestigio afecta a los puntos.
- IDataPersistence — descrita en la clase correspondiente.

TargetController

Esta clase se encarga de controlar la aparición aleatoria y puntaje de distintos elementos por pantalla.

Atributos:

- pointsToAdd: controla cuantos puntos añade al pulsarlo.
- clicked: variable booleana necesaria para saber si el elemento ha sido pulsado.

Relaciones de asociación:

- PointsManager: el clicar el elemento sumará puntos en la partida.

PrestigeManager

Esta clase se encarga de controlar la activación de los prestigios junto con sus modificaciones.

Atributos:

- numPrestiges: variable que almacena el número de prestigios hechos.
- activeBonus[]: Lista de bonus que han sido activados durante el prestigio.
- Condition: condición necesaria para prestigiar.

Relaciones de asociación:

- PointsManager: al prestigiar se resetearán los puntos a cero.

PenaltyManager

Esta clase se encarga de controlar las penalizaciones del jugador por no conectarse con regularidad.

Atributos:

- deltaTime: variable necesaria para calcular de forma precisa el tiempo que lleva sin conectarse un jugador.
- lastTime: variable que almacena la última vez que se conectó el jugador.
- PenaltyListActive: lista con todas las penalizaciones posibles y las activas en ese momento.
- PenaltyCount: contador de las penalizaciones que lleva el jugador.

Relaciones de asociación:

- PointsManager: las penalizaciones afectan a la puntuación.

Relaciones de composición:

- PenaltySO: ScriptableObject que representa una penalización.

PenaltySO

Esta clase es un ScriptableObject de penalizaciones. Con ella se pueden crear todas las penalizaciones necesarias.

Atributos:

- id: identificador.
- name: nombre de la penalización.
- condition: especifica cuándo se activa.
- isActive: especifica si está activa.

Relaciones de composición:

- PenaltyManager: utilizará las penalizaciones que se creen en esta clase.

WriteStats

Esta clase controla toda la parte de las estadísticas.

Atributos:

- CurrentPoints: lleva la cuenta de los puntos actuales.
- AccumulatedPoints: lleva la cuenta de los puntos acumulados.
- PointsPerSecond: lleva la cuenta de los puntos por segundo.
- ElementsClicked: lleva la cuenta de los elementos aleatorios clicados.
- PointsPerClick: lleva la cuenta de los puntos que se consiguen por cada click.
- PuntosActuales: muestra los currentPoints por pantalla.
- PuntosAcumulados: muestra los AccumulatedPoints por pantalla.
- PuntosPorSegundo: muestra los PointsPerSecond por pantalla.
- PuntosPorClick: muestra los PointsPerClick por pantalla.
- ElementosConseguidos: muestra los ElementsClicked por pantalla.

Relaciones de asociación:

- PointsManager: las estadísticas sacan la información de PointsManager.

Satiety

Esta clase controla la saciedad del líder.

Atributos:

- maxCPS: cantidad máxima de clics por segundo que puede hacer el jugador antes de que el líder no acepte más comida.
- sacietyUp: cuánto sube la saciedad por cada clic.
- sacietyDown: cuánto baja la saciedad por cada segundo sin clicar.
- sacietySlider: slider que muestra por pantalla la saciedad del líder.

Relaciones de asociación:

- PointsManager: cuando el líder esté saciado, no sube la puntuación.

ContractManager

Esta clase se encarga de las misiones diarias, semanales y mensuales.

Atributos:

- dailyContracts: lleva la cuenta de los contratos diarios.
- weeklyContracts: lleva la cuenta de los contratos semanales.
- monthlyContracts: lleva la cuenta de los contratos mensuales.
- currentContracts: lleva la cuenta de los contratos activos.
- contractsItemsSO: Scriptable Object para crear contratos.
- nombres: nombre de los contratos.
- contractsPanelsSO: controla el panel donde aparecen los contratos.
- contractsPanels: array con los paneles de los contratos.

- contractsMyPurchaseBtns: controla los botones para reclamar las recompensas.

Relaciones de asociación:

- PointsManager: conseguir realizar contratos sumará recompensas en forma de puntos.

BonusPoints

Esta clase suma un 10% de los puntos actuales a la puntuación cuando se realicen ciertas acciones.

Relaciones de asociación:

- PointsManager: se necesita acceder los puntos actuales y sumar puntos cuando sea necesario.

EnemyManager

Esta clase se encarga de controlar el funcionamiento de los enemigos.

Atributos:

- enemyPrefabs: prefabs para crear enemigos.
- spawnPointX: lugar de aparición en el eje X.
- spawnPointY: lugar de aparición en el eje Y.
- spawnOffset: offset para que los enemigos aparezcan en distintos puntos de la pantalla.
- spawnInterval: controla cada cuanto aparecen enemigos.
- scoreEnemies: controla cuantos puntos da eliminar enemigos.

Relaciones de asociación:

- PointsManager: conseguir eliminar enemigos otorgará recompensas en forma de puntos.

LoanManager

Esta clase se encarga de controlar los préstamos.

Atributos:

- owePoints: guarda los puntos que se deben en ese momento.
- currentLoanAmount: cantidad de puntos pedida.
- totalPoints: guarda los puntos totales de los que se disponen.
- interestRate: interés aplicado al préstamo.
- maxLoanAmount: máxima cantidad de préstamos que se puede pedir.

Relaciones de asociación:

- PointsManager: los préstamos se sumarán a los puntos totales.

Sistema de lógica y audio

LogicStartScreen

Clase que se encarga de la lógica de la pantalla de inicio.

Atributos:

- audioManager: clase que gestiona el audio en el menú de inicio.

Relaciones de asociación:

- AudioManager: LogicStartScreen utiliza un objeto AudioManager para la gestión del audio en esta escena.

Relaciones de composición:

- AudioManager: dentro de la Start Screen está la lógica del audio.

AudioManager

Esta clase gestiona el audio del juego. Controla la reproducción de música de fondo, efectos de sonido y otros elementos relacionados con el audio.

Atributos:

- mainMixer: controlador del mixer principal de audio.
- musicSource: fuente de audio para la música de fondo.
- effectsSource: fuente de audio para los efectos en la escena.
- ost: lista de clips de audios que representan la música original del juego.
- clickingEffects: lista de clips de audios que representan los efectos de sonido al hacer clic.
- random: objeto de la clase random para la reproducción aleatoria de efectos de sonido.
- instanceAudio: objeto para la gestión de la instancia de audio para hacer que AudioManager sea un singleton.

Relaciones de asociación:

- PianoManager: indica que se relaciona con la clase AudioManager para gestionar aspectos relacionados con el piano en el juego.
- SynchronizedMusicManager: indica que se relaciona con la clase AudioManager para gestionar la sincronización de la música en la aplicación.
- DanceManager: indica que se relaciona con AudioManager para gestionar la música que se utilizará en los bailes.
- SettingsManager: en el menú de opciones se controlan opciones relacionadas con el audio, como la música de fondo o los efectos de sonido.

Relaciones de composición:

- LogicStartScreen: el audio es parte de la lógica de la Start Screen.

PianoManager

Esta clase controla la secuencia de notas musicales y otros elementos relacionados con el piano del juego.

Atributos:

- secuenceList: lista de secuencias de notas musicales.
- isActive: Indica si el piano está activo.
- discovered: variable que indica si el piano ha sido o no descubierto.

Relaciones de asociación:

- AudioManager — descrita en la clase correspondiente.

SynchronizedMusicManager

Esta clase controla los puntos de sincronización y otros elementos relacionados con la música.

Atributos:

- MusicManager: gestor de música.
- timingList: lista de sincronización de tiempos.
- pointBonuses: bonificaciones de puntos relacionadas con una sincronización correcta.

Relaciones de asociación:

- AudioManager — descrita en la clase correspondiente.

DanceManager

Clase que controla los bailes que podrá realizar el líder en determinados momentos.

Atributos:

- listOfDances: lista con todos los bailes que se podrán desbloquear.
- unlockedDances: lista con los bailes desbloqueados por el usuario y que podrá realizar el líder.

Relaciones de asociación:

- AudioManager — descrita en la clase correspondiente.

ImageMenuAnimationScript

Esta clase controla las animaciones que realiza la imagen del líder en el menú de inicio del juego. Controla las diferentes poses y animaciones del personaje en el menú.

Atributos:

- timer: tiempo que pasa cada vez que se inicia la animación, es decir, duración de la animación.
- imagen: objeto que representa la imagen del personaje en el menú.
- normal: sprite que represente el estado normal del líder.
- smile: sprite del líder sonriendo.
- sonriendo: indica si el personaje está sonriendo.
- stamp: variable auxiliar que guarda el tiempo que pasa cada vez que se inicia la animación, es decir, duración de la animación.
- estrella: animación con el destello en la sonrisa del personaje.

Relaciones de asociación:

- LogicStartScreen: gestionar el audio relacionado con las animaciones del líder en el menú de inicio.

ObjectLoader

Clase que se encarga cargar, con sus valores predeterminados, los objetos de la tienda (generadores y mejoras) y los logros desde un archivo csv.

Atributos:

- filePathGenerator: ruta hacia el archivo csv con los objetos de la tienda.
- generatorItem: instancia de ShopSO para cargar los ítems para la tienda.
- filePathAchievement: ruta hacia el archivo csv con los logros.
- achievementItem: instancia de AchievementSO para cargar los achievements.
- filaPathPowerups: ruta hacia el archivo csv con los power-ups de la tienda.
- powerupItem: instancia PowerUpSO para cargar los power-ups.

Relaciones de composición:

- ShopManager, AchievementManager y PowerUpManager — descritas en las clases correspondientes.

Sistema de ajustes

SettingsManager

Clase que se encarga de administrar todos los ajustes generales del juego.

Atributos:

- mainMixer: es responsable de ajustar los niveles de volumen, efectos de sonido y otras propiedades relacionadas con el audio del juego.
- resolutionDropdown: lista desplegable de opciones para seleccionar la resolución de pantalla del juego.
- effectsSlider: slider que controla el volumen de los efectos del juego.
- musicSlider: slider que controla el volumen de la música del juego.
- resolutions: lista donde se guardan todas las resoluciones posibles.
- width: variable que almacena el ancho actual de la pantalla.
- height: variable que almacena el alto actual de la pantalla.
- refreshRate: variable que almacena la tasa de refresco actual de la pantalla.
- fpsSlider: slider que restringe la cantidad máxima de FPS que mostrará el juego.
- fpsValues: lista donde se guardan los posibles límites de FPS.
- toggle: toggle que se encarga de activar o desactivar el contador de FPS.
- deltaTime: se utiliza para calcular el tiempo transcurrido entre cada fotograma del juego.

Relaciones de asociación:

- SwitchToggle: para algunos ajustes se usan toggles y estos no existen por defecto en Unity.
- ConsoleManager: los ajustes podrán ser cambiados desde la consola.
- ModeManager: el modo de juego podrá ser cambiado desde la pantalla de ajustes.
- AudioManager: el sonido de volumen y efectos se cambia en los ajustes.

ConsoleManager

Clase que se encarga de controlar y registrar los comandos que son introducidos en la consola del juego.

Atributos:

- activedCodesList: lista donde se almacenan todos los códigos que han sido activados.
- CodesList: lista donde se almacenan todos los posibles códigos que se pueden introducir en la consola.

Relaciones de asociación:

- SettingsManager — descrita en la clase correspondiente.

SwitchToggle

Prefab que genera un toggle que en la pantalla de ajustes se usa para controlar opciones con posibles ajustes binarios.

Atributos:

- backgroundImage: imagen que sirve como fondo al toggle.
- backgroundActiveColor: color al que cambia la imagen de fondo cuando el toggle está activo.
- backgroundDefaultColor: color al que cambia la imagen de fondo cuando el toggle está inactivo.
- toggle: imagen circular situada dentro del fondo que se sitúa a un lado o a otro dependiendo de si el toggle está activo o no.
- handlerPosition: posición de la imagen circular.

Relaciones de asociación:

- SettingsManager — descrita en la clase correspondiente.

ModeManager

Clase que se encarga de registrar cuándo se produce un cambio de modo.

Atributos:

- currentMode: valor que indica el modo actual del juego.
- listOfModes: lista que almacena todos los modos de juego posibles.

Relaciones de asociación:

- SettingsManager — descrita en la clase correspondiente.

Sistema de guardado

IDataPersistence

Esta clase es una interfaz que implementa unos métodos usados en otras clases para llevar a cabo el guardado de datos. No tiene atributos.

Relaciones de asociación:

- ShopManager: se encarga de guardar datos de la tienda.
- AchievementManager; guarda datos de los logros.
- PointsManager: guarda datos de los puntos obtenidos en el juego.
- PowerUpManager: guarda datos de las mejoras.
- EasterEggsManager: guarda datos de los easter-eggs.
- MinigamesManager: guarda datos de los minijuegos.

Relaciones de generalización:

- DataPersistenceManager: la interfaz hace uso de esta clase para poder implementar el guardado en los distintos managers.

DataPersistenceManager

Esta clase se encarga de que todos los datos sobre el progreso de la partida del jugador se guarden correctamente, encriptándolos para evitar su modificación ilegal.

Atributos:

- gameData: clase en la que se guardan todos los datos del juego.
- dataPersistenceObjects: lista con todos los objetos que implementan la interfaz IDataPersistence y deben ser guardados.
- dataHandler: objeto de la clase fileDataHandler.
- timePast: variable que guarda el tiempo transcurrido desde el último guardado automático.

Relaciones de composición:

- FileDataHandler: llama a este objeto para escribir y leer correctamente de los ficheros de guardado.

Relaciones de asociación:

- gameData: objeto que usa el manager para guardar los datos.

Relaciones de generalización:

- IDataPersistence: los métodos de DataPersistenceManager manejan la información que “recoge” la interfaz, implementada en otros managers.

GameData

Esta clase contendrá variables en las que se guardará la información de la partida actual.

Atributos:

- currentScore: guarda la puntuación actual.
- scoreUp: guarda la velocidad a la que se ganan puntos por clic.
- accumulatedScoreStat: guarda la puntuación total obtenida en el juego.
- pointsPerSecond: guarda la cantidad de puntos por segundo obtenidos.
- shopData: diccionario en el que se guarda ID de un ítem de la tienda y la cantidad adquirida.
- achievementData: diccionario en el que se guarda ID de un logro y si está obtenido o no.

Relaciones de asociación:

- DataPersistenceManager: llama a los métodos que usan GameData.
- FileDataHandler: usado para guardar en el archivo de guardado.

FileDataHandler

Esta clase se encarga de guardar, cargar y encriptar los datos del juego.

Atributos:

- dataDirPath: carpeta en la que se encuentra el archivo de guardado.
- dataFileName: nombre del archivo de guardado.
- useEncryption: variable que controla si se encripta o no los datos.
- codeWord: palabra clave usada para encriptar por el algoritmo de XOR.

Relación de composición:

- DataPersistenceManager: es el que llama a sus métodos.

Relación de asociación:

- GameData: usa esta clase para almacenar en un fichero los datos de guardado del juego.

Extras

EasterEggsManager

Esta clase se encarga de controlar la activación de easter-eggs, almacenar los que están activos y sus consecuencias.

Atributos:

- eastereggIdList: lista con todos los easter-eggs.
- totalEasteregg: variable auxiliar que almacena el número de easter-eggs activos.

Relaciones de asociación:

- AchievementManager: algunos easter-eggs desbloquearán logros.
- MinigamesManager: algunos easter-eggs son minijuegos.
- IDataPersistence — descrita en la clase correspondiente.

Relaciones de composición:

- EasterEggSO: objeto que representa un easter-egg.

EasterEggSO

Esta clase del tipo ScriptableObject representa un easter-egg en el juego, con atributos que describen su identificación, nombre, condición para activarse, y su estado de activación.

Atributos:

- id: El identificador único del easter-egg.
- name: El nombre del easter-egg.
- condition: La condición que debe cumplirse para activar el easter-egg.
- isActive: Un indicador que señala si el easter-egg está activo o no.

Relación de composición:

- EasterEggsManager: esta clase contiene instancias de EasterEggSO, lo que implica que tiene la responsabilidad de crear y gestionar las instancias individuales de EasterEggSO.

MinigamesManager

Esta clase se encarga de controlar la activación de minijuegos, almacenar aquellos que están activos y los efectos que se consiguen con cada uno.

Atributos:

- MinigamesList: lista de minijuegos activos.

Relaciones de asociación:

- EasterEggsManager: los minijuegos son easter-eggs.
- IDataPersistence — descrita en la clase correspondiente.

Relaciones de composición:

- MinigameSO: objeto que representa un minijuego.

MinigameSO

Esta clase del tipo ScriptableObject representa un minijuego. Contiene información de este, como su nombre, descripción icono, etc.

Atributos:

- id: identificador del minijuego.
- name: nombre del minijuego.
- description: descripción del funcionamiento del minijuego.
- unlocked: variable booleana que indica si el juego está disponible para jugar.
- iconSprite: guarda el icono del minijuego.

Relaciones de composición:

- MinigamesManager: gestiona todos los objetos tipo MinigameSO.

TutorialAssistantManager

Esta clase se encarga de controlar a la mascota tutorial (Alefito), y a su vez, el propio tutorial en sí mismo.

Atributos:

- phrasesTutorial: lista de frases de tutorial.
- phrasesAdvice: lista de consejos de tutorial.
- isAlive: bool que controla si la mascota sigue viva (puedes matarla).
- adviceTemp: tiempo que tarda en darte una frase de consejo.
- deltaTime: variable auxiliar que necesitamos en Unity para controlar parámetros relacionados con el tiempo.
- advicesOn: booleano que controla si has desactivado los consejos.

Relaciones de asociación:

- END: el asesinato de Alefito tiene repercusión en el final del juego.

END

Esta clase se encarga de controlar el final del juego.

Atributos:

- NeededPoints: puntos necesarios para llegar al final.
- leaderDead: controla si el líder ha muerto.
- assistantDead: controla si Alefito ha muerto.

Relaciones de asociación:

- TutorialAssistantManager: el asesinato de Alefito y la del líder tiene repercusiones en el final.

8. Herramientas software usadas durante la realización del proyecto

Hasta la fecha, las herramientas software usadas en la realización del proyecto son las siguientes:

- **Microsoft Word (Office 365)**: elaboración de documentos compartidos en línea para trabajo cooperativo.
- **WhatsApp**: comunicación asíncrona.
- **Discord**: comunicación síncrona y compartición de documentos.
- **Google Drive**: compartición de documentos.
- **Photopea**: edición de imágenes para el logo del grupo.
- **GIMP**: edición de imágenes para el logo del proyecto.
- **tensor.art**: elaboración de logo con estilo de arte tipo *pixel-art*.
- **8bit Painter**: elaboración de logo con estilo de arte tipo *pixel-art*.
- **remove.bg**: eliminación del fondo de pantalla de imágenes.
- **Copilot (anteriormente Bing AI)**: creación de imágenes, elaboración de texto y búsquedas en internet.
- **ChatGPT**: resumen y elaboración de textos.
- **GitHub**: repositorio y control de versiones.
- **GitHub Desktop**: aplicación gratuita de código abierto que ayuda a trabajar con código hospedado en GitHub.
- **Trello**: organizador de tareas.
- **Unity**: motor de videojuegos multiplataforma usado para crear el videojuego *Feed The Leader*.
- **Microsoft Visual Studio Community**: entorno de desarrollo integrado usado junto con Unity.
- **C#**: lenguaje de programación multiparadigma usado en el desarrollo del videojuego en Unity.
- **Visual Paradigm**: software de diagramas y solución de gráficos empleado en el apartado de *Requisitos*.
- **Aseprite**: herramienta de creación y animación de *pixel-arts*.
- **Freepik**: banco de imágenes en línea, generalmente imágenes gratuitas o sin copyright. También contiene un asistente de dibujo por IA.
- **Audacity**: aplicación informática multiplataforma libre que se puede usar para grabación y edición de audio.
- **Pixabay**: sitio web internacional para el intercambio de fotos de alta calidad de forma gratuita.
- **YouTube**: sitio web dedicado a compartir vídeos. Usado para ver tutoriales y buscar información concreta.