PROYECTO T

Desarrollo de un videojuego de entrenar y cuidar a tu mascota virtual

Nombre del alumno: Alejandro Leal Laserna

Curso académico: 2024-2025

Tutor del proyecto: Cardador

ABSTRACT

Este Trabajo de Fin de Grado presenta el desarrollo completo de 'Proyecto T', un videojuego junta las mecánicas clásicas de mascotas virtuales como Tamagotchi y Digimon con sistemas ya presentes de juegos de rol (RPG) y minijuegos de entrenamiento. Desarrollado en Godot 4.5, el proyecto ha abarcado un periodo de cuatro meses.

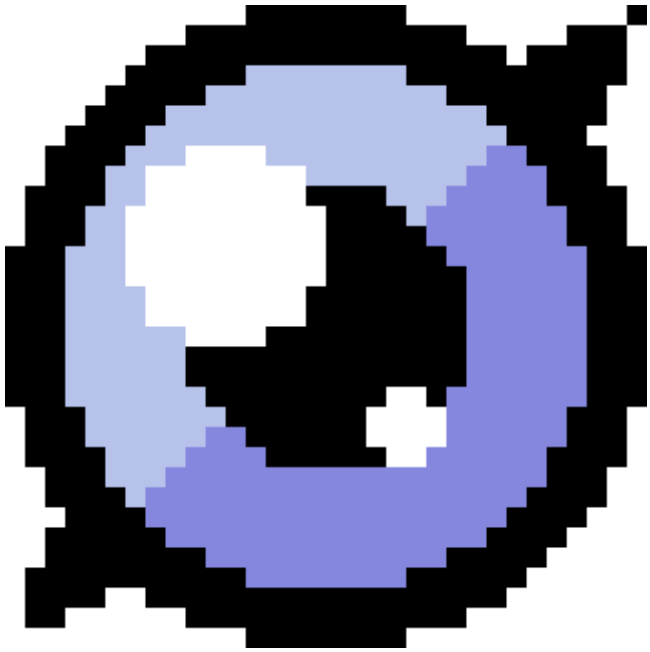
El juego introduce un sistema de cuidado y evolución donde el jugador debe mantener el bienestar de su mascota virtual 'T' mediante indicadores de hambre y felicidad, similar a los dispositivos Tamagotchi originales de los años 90.

El trabajo no crea nada nuevo. Si no que se inspira en mecánica de otros juegos.

La elección de Godot como motor de desarrollo se fundamenta en su código abierto, su especialización en juegos 2D, y su accesibilidad para desarrolladores independientes. El proyecto implementa arquitecturas de código nodular utilizando GDScript, el lenguaje nativo de Godot con sintaxis similar a Python, que permite un desarrollo ágil y mantenible. Todos los recursos visuales del juego, incluyendo sprites de personajes, interfaces de usuario y elementos de entorno, fueron creados manualmente utilizando LibreSprite, con estética Pixel Art en la mayoría de lo posible.

El proyecto demuestra la posibilidad de combinar múltiples géneros de videojuegos intentando mantener una coherencia. El sistema de evolución reacciona dinámicamente a las decisiones del jugador en cuanto al cuidado y entrenamiento, las mecánicas de combate proporcionan desafíos estratégicos basados en las estadísticas entrenadas, y los minijuegos ofrecen una jugabilidad entretenida.

Este TFG detalla el proceso completo de desarrollo, incluyendo la lógica del código, sus funciones y su arquitectura. El TFG concluye que existe un nicho viable en el mercado actual para experiencias que combinen la nostalgia de los juguetes virtuales clásicos con mecánicas de juego ya existentes en dispositivos de la actualidad.



ÍNDICE

1.JUSTIFICACIÓN DEL PROYECTO

1.1 Motivación personal

La motivación de este proyecto empieza de una profunda conexión emocional con los juguetes electrónicos de mi infancia, específicamente con dispositivos como el Tamagotchi y el Digivice de Digimon. Son dispositivos de bolsillo que representaban una gran curiosidad en los niños de Aquella época.

El Tamagotchi, lanzado por Bandai en 1996, revolucionó el concepto de mascota virtual al crear un vínculo emocional real entre el jugador y un simple conjunto de píxeles en una pantalla monocromática. La necesidad de atender constantemente a la mascota virtual, alimentarla, jugar con ella

, limpiarla y cuidar de su salud, generaba un sentido de responsabilidad genuino.

En cambio, el Digivice llevó este concepto un paso más allá al introducir elementos de combate y evolución más complejos. Los Digimon no solo requerían cuidado básico, sino también entrenamiento para batallas contra otros Digimon. La mezcla del cuidado y el entrenamiento competitivo creaba una experiencia mejor y más entretenida.

La ilusión de poder crear mi propio videojuego basado en estos conceptos que marcaron mi infancia representa un desafío técnico dado mi nivel de conceptos en programación, sino también la posibilidad de mejorarlo y en un futuro lanzarlo al público

1.2 Identificación del problema

A pesar del éxito masivo que experimentaron las mascotas virtuales durante su época dorada en los años 90, el mercado actual de videojuegos dejó prácticamente abandonado este género. Mientras que géneros como las plataformas 2D o los RPG clásicos han experimentado un resurgimiento significativo con títulos indie modernos, las mascotas virtuales han quedado relegadas a reimplementaciones directas de los dispositivos originales sin innovación significativa.

Este vacío en el mercado representa una oportunidad desaprovechada. Existen varios factores que contribuyen a esta situación:

Falta de evolución del género: No existe una app de mascota virtual desde la app de POU. No existe un videojuego que de este estilo que tenga un recorrido a la larga.

Limitaciones de diseño autoimpuestas: Los desarrolladores que han intentado revivir el género tienden a mantenerse extremadamente fieles a las limitaciones técnicas de los dispositivos originales. No digo que este proyecto haga algo diferente. Pero si es el principio o tiene las bases para evolucionar a algo más desarrollado.

Ausencia de integración con otros géneros: Hay un potencial para desarrollar algo más complejo mezclándose con mecánicas ya existentes

Proyecto T busca llenar este vacío del mercado ofreciendo una experiencia que honra las mecánicas fundamentales que hicieron exitosos a los originales (cuidado constante, evolución basada en decisiones, vínculo emocional) mientras introduce sistemas adicionales que aprovechan las posibilidades de las plataformas modernas para crear una experiencia más rica, variada y regulable.

2. INTRODUCCIÓN

Proyecto T es un videojuego 2D desarrollado en Godot Engine 4.5 que no Re imagina el concepto clásico de mascota virtual para plataformas modernas. En el juego el jugador asume el rol de cuidador y entrenador de 'T', una criatura digital que requiere atención constante y guía para evolucionar y convertirse en un combatiente.

En su núcleo, el juego mantiene la filosofía fundamental de las mascotas virtuales: crear y mantener un vínculo emocional entre el jugador y la criatura digital mediante la responsabilidad del cuidado constante.

Sistema de Cuidado donde el jugador debe gestionar los indicadores de hambre y felicidad de T. Estos indicadores disminuyen de forma pasiva con el tiempo, requiriendo intervención regular. La felicidad está directamente vinculada al nivel de hambre, creando una relación de cascada donde descuidar la alimentación afecta negativamente el estado emocional de la mascota.

Sistema de Entrenamiento con Tres minijuegos especializados que permiten al jugador mejorar las estadísticas de combate de T. Cada minijuego se enfoca en una estadística específica (fuerza, defensa o evasión) y presenta mecánicas arcade únicas que requieren diferentes habilidades del jugador. La puntuación obtenida en cada minijuego determina cuántos puntos de estadística gana T, incentivando el dominio de las mecánicas y la práctica continua.

Sistema de Combate: Batallas aleatorias contra enemigos que ponen a prueba el desarrollo de T. El sistema de combate implementa una mecánica de piedra-papel-tijera expandida donde cada opción (fuerza, defensa, evasión) contrarresta a otra. Las estadísticas desarrolladas mediante entrenamiento influyen directamente en el resultado de estos enfrentamientos, creando una progresión satisfactoria.

La presentación visual del juego tiene una estética pixel art inspirada en los sprites de 16-bit, estableciendo una conexión con la era dorada de las mascotas virtuales y claridad en la comunicación de estados del juego. La interfaz de usuario implementa un diseño minimalista y nostálgica.

Aunque el juego está en fase beta, está diseñado para funcionar en sesiones cortas repetidas. Una sesión típica podría consistir en verificar el estado de T, alimentarla si es necesario, jugar uno o dos minijuegos para mejorar estadísticas específicas, y potencialmente enfrentar un combate aleatorio en principio. Este diseño respeta el tiempo del jugador mientras mantiene un sentido de progresión constante y recompensas por el esfuerzo invertido.

3. OBJETIVOS

3.1 Objetivo general

Desarrollar un videojuego en fase beta funcional que integre de manera unida las mecánicas clásicas de cuidado de mascotas virtuales con sistemas modernos de combate RPG y minijuegos de entrenamiento, demostrando técnicas de desarrollo de software, diseño de juegos, y creación de arte digital de un estudiante de DAM.

Este objetivo implementa técnicas del juego, sino también demostrar las mis capacidades como alumno: que me ha sido posible crear una experiencia entretenida y rejugable que honre el legado de las mascotas virtuales en la mayoría de lo posible. El proyecto busca servir como prueba de la finalización de mis estudios y como evolución como programador amateur.

3.2. Objetivos específicos

Para alcanzar el objetivo general, se establecieron los siguientes objetivos específicos que guiaron cada fase del desarrollo algo errático:

Implementación de un sistema de estadísticas dinámicas e interconectadas Degradando el hambre de forma temporal pasiva y precisa, vinculando la felicidad al nivel de hambre penalizando el descuido, manteniendo los valores dentro de rangos válidos mediante funciones “clamp” apropiadas, persiguiendo los valores entre sesiones de juego mediante un sistema de guardado y comunicando claramente los cambios en estadísticas al jugador mediante interfaz visual victorias y derrotas de las diferentes mecánicas. Diseño e implementación tres minijuegos de entrenamiento diferenciados.

Creación de tres experiencias de minijuego distintivas para demostrar el aprendizaje de lógica distinta:

Minijuego de Fuerza (Strength Game): Mecánica de botones alternos que requiere velocidad. El jugador debe presionar los botones izquierdo y derecho en la secuencia correcta durante X segundos. La puntuación (número de pulsaciones correctas) determina directamente las ganancias de fuerza según rangos X.

Minijuego de Defensa (Defence Game): Sistema de timing y bloqueo de proyectiles. Los proyectiles se mueven desde la derecha hacia el jugador con velocidad incremental. El jugador debe presionar el botón de bloqueo cuando el proyectil está en una de tres zonas. Bloquear fuera de las zonas o demasiado tarde resulta en game over.

Minijuego de Evasión (Evasion Game): Endless runner típico juego del dinosaurio de Google con obstáculos procedurales. El jugador controla un personaje que debe saltar sobre obstáculos que aparecen aleatoriamente. La velocidad de desplazamiento del mundo aumenta progresivamente con cada obstáculo superado. El juego termina cuando el jugador colisiona con un obstáculo.

Cada minijuego proporciona feedback visual inmediato (animaciones del personaje, efectos visuales) y auditivo durante el juego que a día de hoy está sin desarrollar, incluyendo puntuación obtenida y estadísticas ganadas.

Desarrollo de un sistema de combate estratégico tipo piedra-papel-tijera expandido

Implementar un sistema de combate por turnos que: Siga la lógica: Fuerza > Evasión > Defensa > Fuerza, genere enemigos con estadísticas aleatorias dentro de rangos balanceados, visualice los turnos mediante proyectiles animados que colisionan en el centro de la pantalla, haga que las estadísticas desarrolladas mediante entrenamiento influyan en el resultado de cada turno, implemente un sistema de puntos de vida (HP) para jugador y enemigo (10 HP cada uno), calcule el daño basado tanto en la elección de acción como en la comparación de estadísticas relevantes y comunique claramente el estado del combate mediante sprites animados de HP (Corazones) y etiquetas de estadísticas actualizadas

Creación de un sistema de evolución visual del personaje que implemente un sistema que verifique condiciones de evolución basadas en estadísticas acumuladas, ejecute solo una vez por sesión de juego la verificación de evolución (al cargar el juego, reproduzca una secuencia de animación visual cuando se cumplan las condiciones, pause el mundo del juego durante la secuencia de evolución para crear un momento especial, cambie el sprite del personaje para reflejar visualmente la nueva etapa evolutiva, mantenga el estado evolutivo para que se mantenga entre sesiones y minijuegos y actualice el sprite en todas las escenas (mundo principal, minijuegos, combate) para mantener consistencia visual

Implementación un sistema de gestión de escenas robusto para desarrollar la infraestructura para transaccionar suavemente entre múltiples escenas y Proporcionar botones de navegación claros para retornar a escenas previas

Diseño de los recursos visuales del juego y creación manualmente de un conjunto completo de assets visuales incluyendo: Sprites animados del personaje principal con múltiples estados (caminar, estar de pie, pelear, feliz, enojado) para cada etapa evolutiva, sprites de enemigos para el sistema de combate con animaciones equivalentes, elementos de interfaz de usuario (botones, barras de progreso, iconos, paneles informativos),sprites de proyectiles o magias para el sistema de combate y el minijuego de defensa, obstáculos y elementos de entorno para los minijuegos y sprites de suelo, arbustos decorativos, y elementos de fondo (nubes) para crear profundidad visual mediante “parallax”

He intentado mantener que todos los assets deben mantener un estilo visual cohesivo de pixel art con paleta de colores limitada inspirada en la estética de 16-bit, asegurando legibilidad y claridad visual incluso en resoluciones pequeñas.

Implementación visual: Quería asegurar que cada acción del jugador reciba confirmación visual inmediata mediante animaciones del personaje que reaccionan a eventos (alimentación, entrenamiento exitoso, daño recibido) con efectos de partículas o sprites temporales para acciones importantes (golpes exitosos, evolución)

Estados visuales claros para botones y pantallas de resultado al finalizar minijuegos que muestran claramente logros y recompensas.

Estos objetivos específicos me proporcionaron una hoja de ruta clara para el desarrollo, permitiendo medir progreso concreto y asegurar que cada componente del juego ayudara efectivamente a la experiencia global.

El cumplimiento exitoso de estos objetivos ha resultado en un producto final que ofrece una experiencia de juego coherente y satisfactoria dentro de mis posibilidades actuales de esta beta.

4. DESARROLLO

4.1. Cronología del desarrollo

El proyecto se desarrolló durante un periodo de cuatro meses (septiembre 2025 - diciembre 2025) con dedicación de aproximadamente 15-20 horas semanales, totalizando cerca de 280 horas de trabajo.

La distribución temporal siguió estas fases:

Mes 1 - Prototipado y mecánicas (septiembre 2025):

Semana 1-2: Creación inicial de proyecto en Godot, configuración de control de versiones Git, investigación de documentación de Godot.

Semana 3-4: Implementación de sistema de hambre y felicidad con timers, creación de UIPlayer para visualización de stats, desarrollo del primer minijuego (Strength Game).

Mes 2 - Expansión de sistemas (octubre 2025):

Semana 1-2: Sistema de estadísticas persistentes mediante GameData singleton, implementación de minijuegos de Defensa y Evasión.

Semana 3-4: Desarrollo del sistema de combate con mecánica piedra papel y tijera, creación de algoritmo de generación de enemigos aleatorios, implementación de sistema de HP y determinación de ganador.

Mes 3 - Sistema de evolución y arte noviembre 2025):

Semana 1-2: Diseño e implementación de sistema de evolución condicional, creación de múltiples sprites para diferentes etapas evolutivas, animaciones de transición.

Semana 3-4: Producción completa de arte en LibreSprite: sprites de personaje principal (walk, standing, fight, happy, angry), UI elements (botones, barras, fondos de pantalla). Desarrollo de memoria del este TFG.

Mes 4 - Pulido, testeo y documentación (diciembre 2025):

Semana 1: Corrección de bugs, refinamiento de animaciones, adición de transiciones suaves entre estados, optimización de rendimiento.

4.2. Metodología de desarrollo

En lugar de intentar completar el diseño completo antes de implementación, el proyecto se dividió en características discretas que podían desarrollarse, testearse y refinarse independientemente:

- Iteración 1: Sistema de hambre y felicidad básico
- Iteración 2: Primer minijuego (fuerza) con mecánica física.
- Iteración 3: Sistema de estadísticas y persistencia
- Iteración 4: Segundo minijuego (defensa)
- Iteración 5: Sistema de combate
- Iteración 6: Tercer minijuego (evasión)
- Iteración 7: Sistema de evolución

4.3 Justificación de la elección de aplicaciones

Godot 4.5

La selección de Godot Engine como plataforma de desarrollo para Proyecto T fue resultado de un análisis de las opciones disponibles entre las opciones actuales de motores de juego. Esta decisión se fundamenta en tres pilares principales: consideraciones técnicas, filosóficas y prácticas.

Godot es completamente gratuito y de código abierto bajo licencia MIT, lo que ofrece ventajas significativas para un proyecto académico y para el desarrollo independiente en general. A diferencia de alternativas comerciales como Unity o Unreal Engine que operan bajo modelos freemium con restricciones de ingresos o royalties, Godot permite libertad total para publicar y monetizar el juego sin costes de licencia ni porcentajes de ingresos, modificar el motor mismo si surge la necesidad de funcionalidades específicas no cubiertas por la implementación base cosas que por mi conocimiento no sabría pero que existe una comunidad que se encarga de mejorarlo constantemente, garantiza la sostenibilidad a largo plazo del proyecto sin dependencia de cambios en políticas corporativas o términos de servicio al ser de código abierto.

Esta libertad es perfecta en el contexto educativo, donde las restricciones de licenciamiento pueden complicar la presentación y distribución de trabajos académicos. Además, la transparencia del código fuente de Godot facilita el debugging y la comprensión de problemas técnicos complejos.

Aunque Godot es menos popular que Unity o Unreal, su comunidad es activa y la calidad de sus recursos educativos es bastante fácil. Para la realización del proyecto me he encontrado documentación oficial con ejemplos de código para cada API, tutoriales oficiales paso a paso para diferentes tipos de juego en Youtube mayormente, comunidades activas en Reddit, Discord, y foros oficiales donde obtener ayudarme y abundancia de tutoriales en video en YouTube tanto en inglés como en español.

Estos recursos fueron instrumentales durante el desarrollo de Proyecto T, especialmente al enfrentar desafíos técnicos específicos como la sincronización de animaciones o la gestión eficiente del cambio entre escenas.

Libre Sprite

LibreSprite (fork open-source de Aseprite) fue la herramienta principal para creación de todos los assets visuales. Esta aplicación especializada en pixel art ofreció:

Sistema de frames y capas: Fundamental para crear animaciones frame-por-frame donde cada frame es una capa independiente que puede editarse sin afectar los demás.

“Onion Skinning”: Visualización semitransparente de frames adyacentes facilitando la creación de animaciones fluidas al poder ver el frame anterior y siguiente mientras se dibuja el actual.

Paletas de colores personalizadas: Creación y gestión de paletas limitadas que aseguran consistencia visual en todo el proyecto. La paleta se limitó a 16 colores para mantener una estética cohesiva.

Exportación de Spritesheets: Generación automática de hojas de sprites optimizadas que Godot puede importar directamente como SpriteFrames resources.

Herramientas de Selección Pixel-Perfect: Precisión absoluta necesaria para trabajo en escalas pequeñas (sprites de 32x32 a 64x64 píxeles).

Previsualización de Animación en Tiempo Real: Visualización inmediata de cómo se verá la animación completa, permitiendo ajustar timing y transiciones iterativamente.

Git y GitHub

Aunque el proyecto fue desarrollado individualmente, se implementó control de versiones mediante Git con repositorio remoto en GitHub:

Comentarios frecuentes con mensajes descriptivos después de cada modificación completada o bug significativo corregido

Ramificaciones separadas para experimentación con modificación riesgosas antes de “merge” a “main”

Tags para marcar hitos importantes.

Guardado seguro automático en la nube mediante pushes y ocommits regulares con posibilidad de revertir cambios problemáticos sin pérdida de progreso

4.4. Mecánicas implementadas

Esta sección detalla el funcionamiento interno de cada sistema del juego, explicando no solo qué hace cada mecánica, sino cómo está programada y por qué se tomaron las decisiones de diseño específicas.

4.4.1. Sistema de hambre y felicidad

El sistema de hambre y felicidad constituye el “core loop” más básico del juego.

Degradación por Tiempo: A diferencia de Tamagotchis reales que usaban tiempo real incluso cuando apagados, Proyecto T solo degrada hambre durante sesiones activas. Cada X minutos se el estado de hambre descenderá y la felicidad se verá afecta cuando el hambre alcanza ciertos porcentajes

Esta implementación escalonada evita que la felicidad caiga demasiado rápido, dando al jugador tiempo de reaccionar. La función “clamp” asegura que los valores nunca excedan máximos, previniendo bugs.

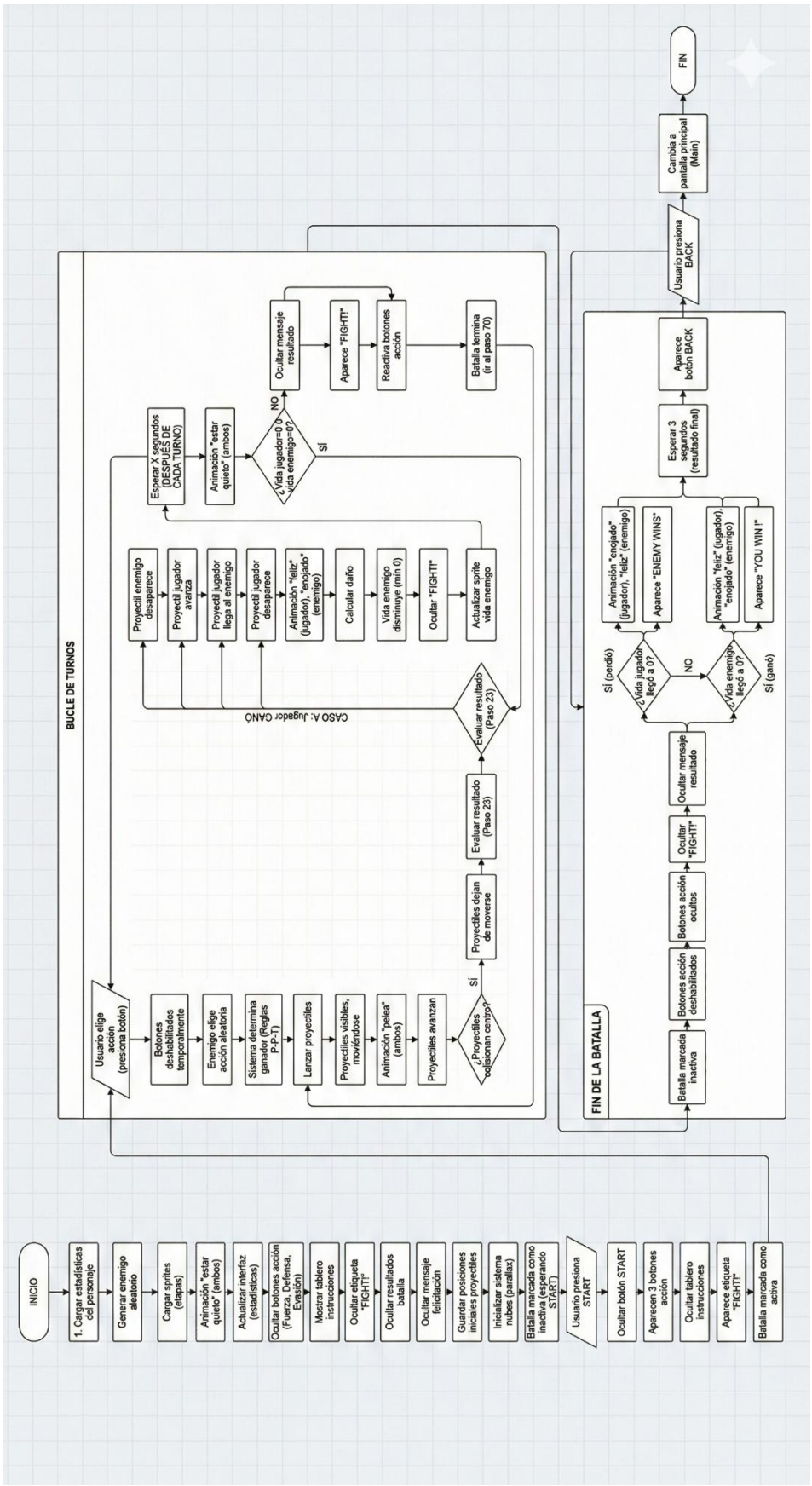
El sistema de hambre y felicidad constituye el núcleo jugable más básico del proyecto, funcionando como el “core loop” que mantiene la interacción constante del jugador con su criatura. A diferencia de los Tamagotchis tradicionales, que reducían las estadísticas incluso cuando el dispositivo estaba apagado, en Proyecto T la degradación solo ocurre mientras la sesión está activa. Con el paso del tiempo, cada cierto intervalo, el nivel de hambre disminuye de forma automática y, cuando alcanza determinados umbrales, la felicidad también se ve afectada de manera gradual. Esta estructura escalonada permite que la criatura no caiga bruscamente en estados críticos, lo que ofrece al jugador un margen razonable para reaccionar. Además, gracias al uso de funciones como *clamp*, se asegura que los valores nunca superen los límites establecidos, evitando inconsistencias y posibles errores.

Cuando el jugador entra a la pantalla principal, la interfaz presenta las acciones disponibles, entre ellas los botones **Feed** y **Trainer**. Si el usuario decide alimentar a la criatura, la interfaz envía la orden correspondiente al personaje, que ejecuta la animación o proceso de alimentación y aumenta automáticamente su felicidad en una cantidad fija. A continuación, tanto la barra de hambre como la de felicidad se actualizan en pantalla para reflejar los nuevos valores. Estos cambios no solo se muestran visualmente, sino que también se registran de inmediato en el sistema de guardado. De esta forma, el progreso queda almacenado sin que el jugador tenga que realizar ninguna acción adicional, y el juego vuelve de manera natural al estado inicial, listo para que el usuario decida su siguiente movimiento.

4.4.2. Sistema de combate

El combate es un sistema estructurado por turnos con una separación clara entre la lógica de decisión y la representación visual. Inicialmente, se cargan estadísticas y se genera un enemigo aleatorio. El flujo se divide en estados: "Inactivo" (esperando start) y "Activo" (bucle de pelea). En cada turno, el jugador selecciona una acción (Fuerza, Defensa, Evasión), bloqueando la interfaz para evitar *spam*. El enemigo selecciona su acción aleatoriamente y el sistema determina el ganador internamente usando reglas tipo "Piedra-Papel-Tijera" antes de mostrar nada.

Una vez decidido el resultado, comienza la "fase visual": ambos lanzan proyectiles que colisionan en el centro. Dependiendo del cálculo previo, un proyectil destruye al otro y golpea al oponente, restando vida. El sistema verifica tras cada impacto si la vida de alguno llegó a cero. Si ambos siguen vivos, se reinicia el turno. Si hay un ganador, se termina el combate, se muestran las animaciones finales (victoria/derrota) y mensajes correspondientes, retornando finalmente al menú principal.


INICIO
1. Cargar estadísticas del personaje
Generar enemigo aleatorio
Cargar sprites (etapas)
Animación "estar quieto" (ambos)
Actualizar interfaz (estadísticas)
Ocultar botones acción (Fuerza, Defensa, Evasión)
Mostrar tablero instrucciones
Ocultar etiqueta "FIGHT!"
Ocultar resultados batalla
Ocultar mensaje felicitación
Guardar posiciones iniciales proyectiles
Inicializar sistema nubes (parallax)
Batalla marcada como inactiva (esperando START)
Usuario presiona START
Ocultar botón START
Aparecen 3 botones acción
Ocultar tablero instrucciones
Aparece etiqueta "FIGHT!"
Batalla marcada como activa
BUCLE DE TURNOS
Usuario elige acción (presiona botón)
Botones deshabilitados temporalmente
Enemigo elige acción aleatoria
Sistema determina ganador (Reglas P-P-T)
Lanzar proyectiles
Proyectiles visibles, moviéndose
Animación "pelea" (ambos)
Proyectiles avanzan
¿Proyectiles colisionan centro?
Sí
Proyectiles dejan de moverse
Evaluar resultado (Paso 23)
Evaluar resultado (Paso 23)
CASO A: Jugador GANÓ
Proyectil enemigo desaparece
Proyectil jugador avanza
Proyectil jugador llega al enemigo
Proyectil jugador desaparece
Animación "feliz" (jugador), "enojado" (enemigo)
Calcular daño
Vida enemigo disminuye (mín 0)
Ocultar "FIGHT!"
Actualizar sprite vida enemigo
Esperar X segundos (DESPUÉS DE CADA TURNO)
Animación "estar quieto" (ambos)
¿Vida jugador=0 o vida enemigo=0?
NO
Sí
Ocultar mensaje resultado
Aparece "FIGHT!"
Reactiva botones acción
Batalla termina (ir al paso 70)
FIN DE LA BATALLA
Batalla marcada inactiva
Botones acción deshabilitados
Botones acción ocultos
Ocultar "FIGHT!"
Ocultar mensaje resultado
¿Vida jugador llegó a 0?
Sí (perdió)
NO
¿Vida enemigo llegó a 0?
Sí (ganó)
Animación "enojado" (jugador), "feliz" (enemigo)
Aparece "ENEMY WINS"
Animación "feliz" (jugador), "enojado" (enemigo)
Aparece "YOU WIN !"
Esperar 3 segundos (resultado final)
Aparece botón BACK
Usuario presiona BACK
Cambia a pantalla principal (Main)
FIN

4.4.3. Sistema de Evolución

El sistema de evolución actúa como el mecanismo central de recompensa a largo plazo, diseñado para transformar la dedicación del jugador en un cambio tangible y permanente en la mascota. A diferencia de un crecimiento basado meramente en el tiempo transcurrido, esta mecánica requiere un **cuidado activo**: la evolución debe "ganarse" alcanzando umbrales estadísticos específicos mediante minijuegos y manteniendo el bienestar general. Actualmente, el sistema gestiona la transición de la etapa inicial ("Baby") a la etapa juvenil ("Young"), aunque su arquitectura modular está preparada para escalar hacia formas adultas y finales en el futuro.

Lógica de Verificación y Disparadores Técnicamente, el sistema opera mediante una validación en dos tiempos para no interrumpir el flujo de juego.

El primer momento ocurre de forma invisible tras finalizar un entrenamiento: cuando el sistema otorga puntos de estadística (Fuerza, Defensa o Evasión), compara inmediatamente los nuevos valores con los requisitos de evolución. Si se cumplen, simplemente marca una "bandera" interna indicando que la evolución está pendiente y guarda el estado, permitiendo que el jugador continúe en el menú de entrenamiento sin interrupciones abruptas.

El segundo momento es la ejecución visual, que sucede al regresar a la pantalla principal (Main). Durante la inicialización de la escena, el sistema consulta la bandera de evolución (verificando solo una vez por sesión para evitar bucles lógicos). Si detecta que las condiciones están cumplidas, bloquea el comportamiento estándar de la mascota e inicia la secuencia cinematográfica.

Secuencia de Transformación (La Ceremonia) La evolución es un evento dramático estructurado en fases cronometradas que alteran el estado del mundo:

1. Pausa y Preparación: El personaje emite una señal que detiene el tiempo en el entorno (Main) y desactiva su propio motor de físicas, congelándose en una animación de espera para centrar la atención del usuario.
2. Transformación de Datos: Tras una pausa dramática y efectos visuales, el sistema realiza el cambio crítico en el backend: actualiza la variable de etapa evolutiva, guarda la partida automáticamente para asegurar el progreso y, lo más importante, descarga el sprite antiguo para cargar y renderizar la nueva apariencia de la mascota.
3. Celebración y Retorno: Una vez visible la nueva forma, la mascota ejecuta una animación de "felicidad" o victoria. Finalmente, tras esperar un total de 5 segundos desde el inicio del evento, el sistema reactiva las físicas, reanuda el tiempo del mundo y devuelve el control a la IA de la mascota, que comienza a caminar e interactuar luciendo sus nuevas características.

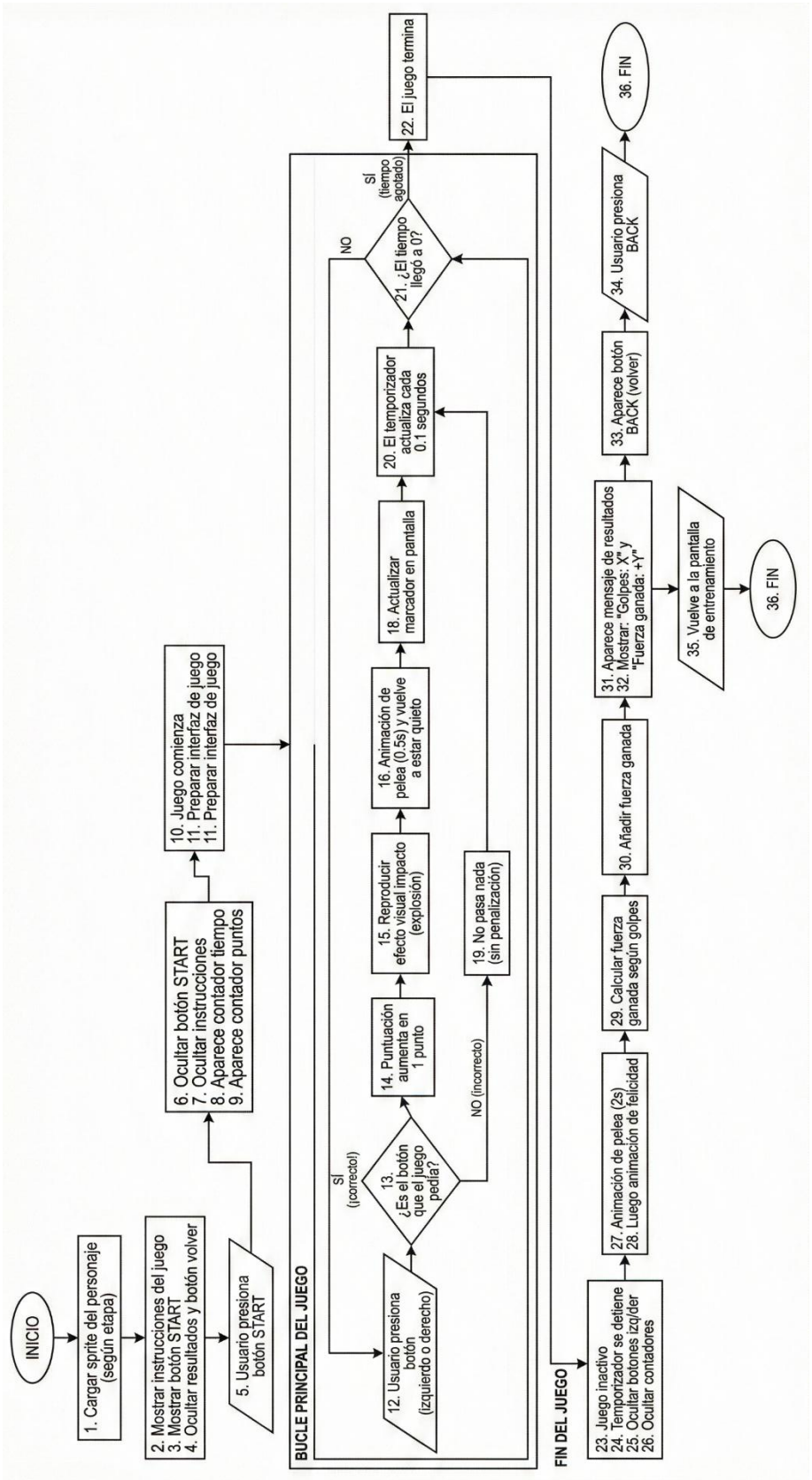
4.4.4. Minijuegos de entrenamiento

Los tres minijuegos siguen una estructura similar, pero con mecánicas diferenciadas para entrenar stats específicas.

Minijuego de Fuerza (Strength Game)

Este sistema se plantea como un desafío de agilidad mental y reflejos bajo presión de tiempo. Tras la inicialización del sprite evolutivo y la interfaz, el bucle principal comienza activando un temporizador regresivo. La mecánica central consiste en solicitar al jugador un input específico (izquierda o derecha), obligándolo a reaccionar correctamente para sumar puntos. Si el input coincide con la solicitud, se premia al usuario con feedback visual (impacto/explosión), una animación de ataque rápida y la alternancia inmediata del botón objetivo, generando un ritmo frenético. Los errores no penalizan restando puntos, pero consumen tiempo valioso al no avanzar el marcador.

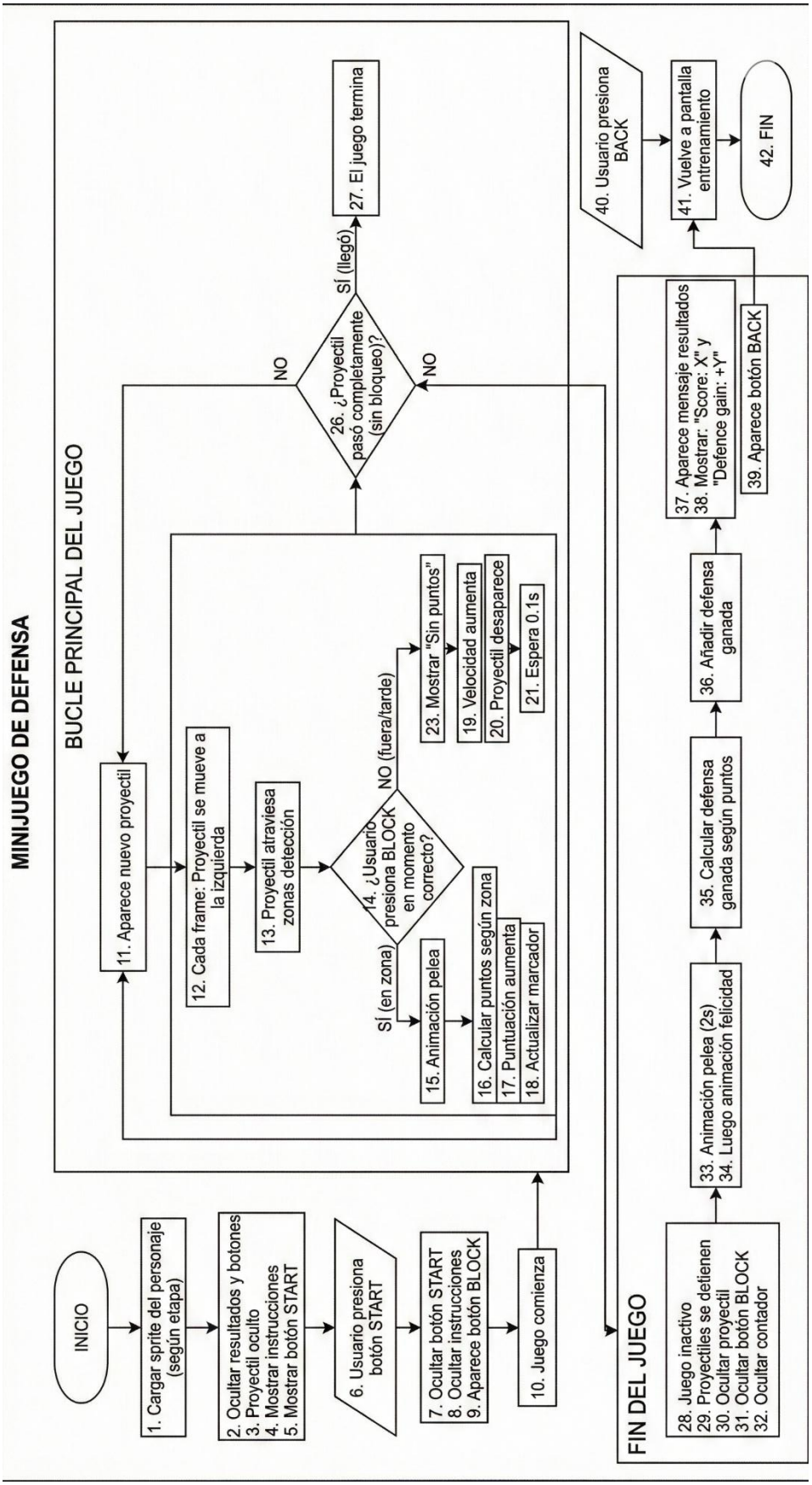
El ciclo se mantiene verificando cada décima de segundo el estado del temporizador. Al llegar a cero, el sistema bloquea los controles, finaliza la sesión y transiciona a una fase de recompensa. Aquí, el personaje celebra visualmente y se ejecuta una fórmula matemática que convierte la cantidad de golpes acertados en puntos de estadística de "Fuerza" permanentes, cerrando el flujo con un botón de retorno al menú principal.


INICIO
1. Cargar sprite del personaje (según etapa)
2. Mostrar instrucciones del juego
3. Mostrar botón START
4. Ocultar resultados y botón volver
5. Usuario presiona botón START
6. Ocultar botón START
7. Ocultar instrucciones
8. Aparece contador tiempo
9. Aparece contador puntos
10. Juego comienza
11. Preparar interfaz de juego
11. Preparar interfaz de juego
BUCLE PRINCIPAL DEL JUEGO
12. Usuario presiona botón (izquierdo o derecho)
13. ¿Es el botón que el juego pedía?
SÍ (correcto!)
NO (incorrecto)
14. Puntuación aumenta en 1 punto
15. Reproducir efecto visual impacto (explosión)
16. Animación de pelea (0.5s) y vuelve a estar quieto
18. Actualizar marcador en pantalla
19. No pasa nada (sin penalización)
20. El temporizador actualiza cada 0.1 segundos
21. ¿El tiempo llegó a 0?
NO
SÍ (tiempo agotado)
22. El juego termina
FIN DEL JUEGO
23. Juego inactivo
24. Temporizador se detiene
25. Ocultar botones izq/der
26. Ocultar contadores
27. Animación de pelea (2s)
28. Luego animación de felicidad
29. Calcular fuerza ganada según golpes
30. Añadir fuerza ganada
31. Aparece mensaje de resultados
32. Mostrar: "Golpes: X" y "Fuerza ganada: +Y"
33. Aparece botón BACK (volver)
34. Usuario presiona BACK
35. Vuelve a la pantalla de entrenamiento
36. FIN
36. FIN

Minijuego de Defensa (Defense Game)

Diseñado como una prueba de "supervivencia" basada en el timing, este módulo pone al jugador en una posición estática frente a amenazas entrantes. Al iniciar, se ocultan los elementos innecesarios y se presenta un único botón de acción: "Bloquear". La lógica del juego genera proyectiles que se desplazan horizontalmente hacia el personaje, atravesando zonas de detección invisibles. El núcleo del gameplay reside en presionar el botón de bloqueo exactamente cuando el proyectil se encuentra dentro de estas zonas de validación.

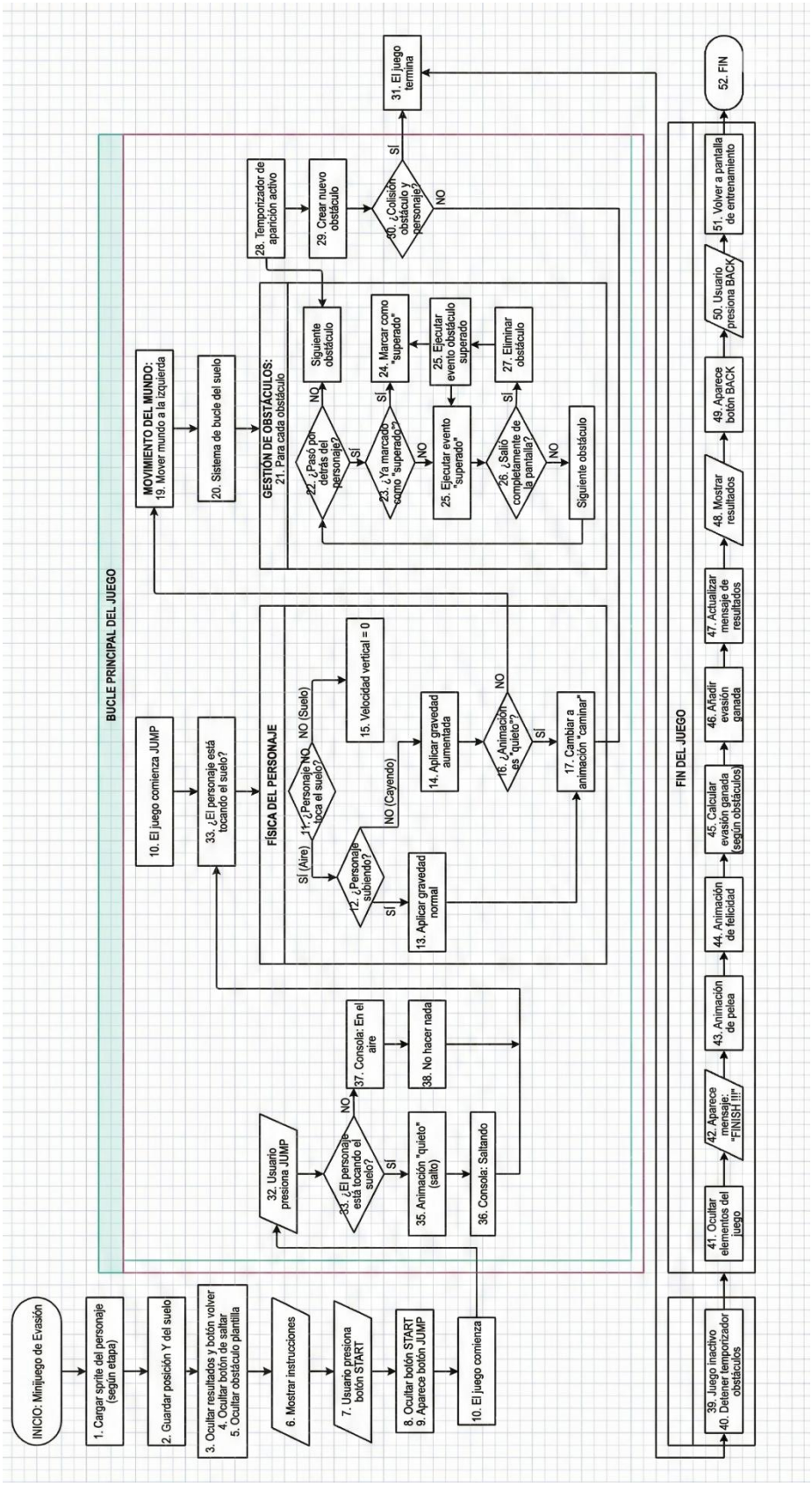
El sistema de dificultad es progresivo: cada bloqueo exitoso no solo suma puntos, sino que incrementa la velocidad de los siguientes proyectiles, exigiendo mayor concentración. Si el jugador falla el timing o no presiona el botón, y el proyectil cruza la pantalla impactando al personaje, el juego termina inmediatamente (condición de derrota). Al finalizar, se calcula la estadística de "Defensa" ganada en función del puntaje acumulado antes del fallo, y tras una animación de victoria/derrota, se permite volver al entrenamiento.


MINIJUEGO DE DEFENSA
INICIO
1. Cargar sprite del personaje (según etapa)
2. Ocultar resultados y botones
3. Proyectil oculto
4. Mostrar instrucciones
5. Mostrar botón START
6. Usuario presiona botón START
7. Ocultar botón START
8. Ocultar instrucciones
9. Aparece botón BLOCK
10. Juego comienza
BUCLE PRINCIPAL DEL JUEGO
11. Aparece nuevo proyectil
12. Cada frame: Proyectil se mueve a la izquierda
13. Proyectil atraviesa zonas detección
14. ¿Usuario presiona BLOCK en momento correcto?
SÍ (en zona)
NO (fuera/tarde)
15. Animación pelea
16. Calcular puntos según zona
17. Puntuación aumenta
18. Actualizar marcador
19. Velocidad aumenta
20. Proyectil desaparece
21. Espera 0.1s
23. Mostrar "Sin puntos"
26. ¿Proyectil pasó completamente (sin bloqueo)?
SÍ (llegó)
NO
NO
27. El juego termina
FIN DEL JUEGO
28. Juego inactivo
29. Proyectiles se detienen
30. Ocultar proyectil
31. Ocultar botón BLOCK
32. Ocultar contador
33. Animación pelea (2s)
34. Luego animación felicidad
35. Calcular defensa ganada según puntos
36. Añadir defensa ganada
37. Aparece mensaje resultados
38. Mostrar: "Score: X" y "Defence gain: +Y"
39. Aparece botón BACK
40. Usuario presiona BACK
41. Vuelve a pantalla entrenamiento
42. FIN

Minijuego de Evasión (EvasionGame)

Este minijuego implementa una lógica de *endless runner* (corredor infinito) con un sistema de física simplificado. El entorno se mueve hacia la izquierda para simular el avance, mientras el personaje gestiona su posición vertical mediante gravedad y fuerzas de salto. El bucle lógico evalúa constantemente si el personaje está en el suelo (permitiendo saltar y animando "caminar") o en el aire (aplicando gravedad y animando "salto"). La dificultad radica en la generación procedimental de obstáculos que entran por la derecha.

El sistema monitorea la posición de cada obstáculo relativo al jugador. Si un obstáculo es superado (pasa por detrás del personaje sin tocarlo), se incrementa el contador y eventualmente se recicla el objeto. Si existe una colisión (superposición de *hitboxes*), el juego se detiene al instante. Al igual que los otros módulos, al detectar la colisión, se congelan los temporizadores, se reproduce la secuencia de celebración del personaje y se convierte la distancia recorrida u obstáculos superados en puntos de estadística de "Evasión".


INICIO: Minijuego de Evasión
1. Cargar sprite del personaje (según etapa)
2. Guardar posición Y del suelo
3. Ocultar resultados y botón volver
4. Ocultar botón de saltar
5. Ocultar obstáculo plantilla
6. Mostrar instrucciones
7. Usuario presiona botón START
8. Ocultar botón START
9. Aparece botón JUMP
10. El juego comienza
BUCLE PRINCIPAL DEL JUEGO
10. El juego comienza JUMP
33. ¿El personaje está tocando el suelo?
FÍSICA DEL PERSONAJE
11. ¿Personaje toca el suelo?
SÍ (Aire)
NO (Suelo)
15. Velocidad vertical = 0
12. ¿Personaje subiendo?
NO (Cayendo)
SÍ
13. Aplicar gravedad normal
14. Aplicar gravedad aumentada
16. ¿Animación es "quieto"?
NO
SÍ
17. Cambiar a animación "caminar"
MOVIMIENTO DEL MUNDO:
19. Mover mundo a la izquierda
20. Sistema de bucle del suelo
GESTIÓN DE OBSTÁCULOS:
21. Para cada obstáculo
22. ¿Pasó por detrás del personaje?
NO
SÍ
Siguiente obstáculo
23. ¿ Ya marcado como "superado"?
SÍ
NO
24. Marcar como "superado"
25. Ejecutar evento "superado"
25. Ejecutar evento obstáculo superado
26. ¿Salió completamente de la pantalla?
SÍ
NO
27. Eliminar obstáculo
Siguiente obstáculo
28. Temporizador de aparición activo
29. Crear nuevo obstáculo
30. ¿Colisión obstáculo y personaje?
SÍ
NO
31. El juego termina
32. Usuario presiona JUMP
33. ¿El personaje está tocando el suelo?
SÍ
NO
35. Animación "quieto" (salto)
36. Consola: Saltando
37. Consola: En el aire
38. No hacer nada
FIN DEL JUEGO
39. Juego inactivo
40. Detener temporizador obstáculos
41. Ocultar elementos del juego
42. Aparece mensaje: "FINISH !!!"
43. Animación de pelea
44. Animación de felicidad
45. Calcular evasión ganada (según obstáculos)
46. Añadir evasión ganada
47. Actualizar mensaje de resultados
48. Mostrar resultados
49. Aparece botón BACK
50. Usuario presiona BACK
51. Volver a pantalla de entrenamiento
52. FIN

4.5. Cumplimiento de objetivos

Proyecto T ha alcanzado exitosamente todos los objetivos establecidos al inicio del desarrollo:

Objetivo General: Crear un videojuego funcional que combine mecánicas de cuidado virtual con sistemas de combate RPG cumpliendo completamente que el juego integre un sistema de hambre/felicidad con minijuegos de entrenamiento y combate estratégico.

Objetivos Específicos:

Implementar sistema de estadísticas dinámicas donde las acciones del jugador afectan a la progresión.

Diseñar minijuegos de entrenamiento interactivos que mejoren stats con tres minijuegos distintos (fuerza, defensa, evasión) completamente funcionales.

Crear sistema de evolución basado en cuidado y entrenamiento cumpliendo la evolución condicional de stats con transformación visual completa.

Combate implementa Pi-Pa-Ti con proyectiles animados donde stats determinan efectividad.

Todos los sprites dibujados manualmente en LibreSprite con paleta predefinida y animaciones fluidas.

El proyecto no solo cumplió objetivos, sino que demostró que la visión inicial es viable técnicamente y entretenida.

4.6 Testeo

Mi forma de hacer el testeo ha sido principalmente manual mientras jugaba porque, sinceramente, era la manera más natural de detectar fallos en tiempo real. Al interactuar directamente con el proyecto podía ver al momento qué cosas funcionaban bien y cuáles necesitaban un retoque. Además, este tipo de prueba me permitió ponerme en el lugar del usuario final, algo que a veces es complicado transmitir solo con pruebas automatizadas.

Otro motivo es que el proyecto todavía estaba en plena construcción, y muchas de las mecánicas estaban cambiando constantemente. Hacer tests automáticos en esa etapa habría supuesto invertir mucho tiempo en actualizarlos cada vez que modificaba una función o añadía una característica nueva. En cambio, probándolo yo mismo mientras avanzaba, podía ajustar sobre la marcha sin frenar el ritmo de desarrollo.

También debo decir que jugar el proyecto mientras lo probaba me ayudó a entender mejor la experiencia general, desde la fluidez hasta los pequeños detalles que normalmente pasan desapercibidos. Este tipo de testeo me permitió descubrir errores visuales, problemas de sensaciones y pequeños fallos lógicos que no habría visto tan fácil de otra forma.

Aunque en un futuro quiero complementar este proceso con tests más formales y automatizados, por ahora el testeo manual ha sido la opción más práctica, directa y eficaz para asegurar que todo funcionara de manera coherente y agradable.

5. REFLEXIÓN FINAL

Desarrollo de Proyecto T ha sido muy personal. Transformar concepto abstracto ("quiero hacer un Tamagotchi moderno") en producto jugable requirió centenares de horas de aprendizaje, programación, diseño, testeo, frustración, y satisfacción.

El proyecto validó mi pasión por videojuegos, combinada con dedicación para aprender herramientas modernas y voluntad para iterar hasta alcanzar calidad, puede resultar en creaciones que no solo cumplen requisitos académicos, sino que tienen potencial real para los jugadores.

Más allá de lo técnico, Proyecto T validó el potencial de revitalizar géneros nostálgicos mediante un cuidado que respeta funciones básicas. Las mascotas virtuales tienen lugar en mercado moderno, no como reliquias del pasado, sino como experiencias que pueden evolucionar para audiencias sin perder presencia.

El feedback positivo de playtesters ("Me quedó bonito", "Funciona", "Tiene futuro") valida que meses de trabajo resultaron en algo emocionante. Este reconocimiento externo, combinado con satisfacción personal de ver ideas abstractas convertirse en producto jugable, hace que cada hora invertida haya valido pena.

Mirando hacia adelante, Proyecto T es solo comienzo. Las líneas futuras de desarrollo trazadas representan roadmap ambiciosa pero alcanzable para transformar el prototipado actual en producto comercialmente viable.

Este proyecto ha demostrado que pasión por videojuegos, combinada con dedicación para aprender herramientas modernas y voluntad para iterar hasta alcanzar calidad, puede resultar en creaciones que contribuyen al medio del entretenimiento interactivo.

Proyecto T vive, respira digitalmente, y está listo para crecer.

6. LÍNEAS DE INVESTIGACIÓN FUTURAS

El prototipo actual del Proyecto T constituye una base técnica muy sólida. Sin embargo, para transformarlo en un producto comercialmente viable y competitivo, se propone la expansión de sus características en varias áreas clave, detalladas a continuación:

1. Profundidad en la Progresión.
2. Sistema de Evolución: Se sugiere implementar un sistema de evolución más estratificado y significativo.
3. Etapas Avanzadas: Introducir una etapa "Adulto", accesible al dominar todas las estadísticas, y una "Forma Final" como máximo hito, obtenida tras un número destacable de batallas victoriosas.
4. Evoluciones Ramificadas: Las decisiones del jugador influirían en el desarrollo de la mascota. Por ejemplo, priorizar el entrenamiento de la Fuerza derivaría en una evolución de tipo físico, mientras que centrarse en la Evasión conduciría a una forma más ágil.
5. Ampliación del gameplay: colección y estrategia.
6. Sistema de Múltiples Mascotas: Los jugadores podrían capturar o desbloquear hasta 10 especies diferentes, cada una con un árbol evolutivo único.
7. Tipos y Ventajas: Incorporar tipos elementales (agua, fuego, planta) con un sistema de ventajas y desventajas añadiría una capa estratégica tanto al cuidado como al combate.
8. Mecánicas de Interacción Diversificadas: Minijuegos e Ítems
9. Minijuegos Variados: Incluir un minijuego de azar (ej. una máquina tragaperras) y otros diseñados para entrenar estadísticas compuestas (ej. Fuerza + Defensa). La dificultad progresaría con el nivel de la mascota.
10. Sistema de Ítems Integral: Añadiría profundidad al incluir comidas especiales, equipales que aumenten stats de forma permanente o temporal, e ítems específicos para desencadenar evoluciones.
11. Compromiso a Largo Plazo: Personalización y Eventos.
12. Personalización: Permitir nombrar a la mascota, elegir su paleta de colores, decorar su hábitat y desbloquear skins cosméticas fomenta el apego emocional.

13. Dinámica Mundial: Un ciclo día/noche que afecte al comportamiento de la mascota y eventos especiales estacionales (Halloween, Navidad) o aleatorios mantendrían el mundo vivo y el juego fresco.

14. Experiencia de Usuario Compleja: Audio y Persistencia.

15. Audio Inmersivo: Es fundamental contar con una banda sonora adaptada a cada escena, efectos de sonido para todas las acciones y voice clips para la mascota que refuercen su personalidad.

16. Persistencia Robusta: Un sistema de guardado en múltiples slots, guardado en la nube para sincronización entre dispositivos y auto-save son esenciales para una experiencia de usuario moderna y fiable.

17. Estrategia Comercial: Multijugador, multiplataforma y Localización.

18. Características Sociales: El combate PvP (local y online), un sistema de intercambio de mascotas (trading) y tablas de clasificación globales son clave para la competitividad y longevidad.

19. Expansión Multiplataforma: Exportar el proyecto a Android/iOS y Steam (PC) maximizaría su alcance, adaptando los controles para una experiencia touch-friendly en móviles.

20. 20. Localización: Traducir la interfaz y los textos al inglés es el primer paso para una audiencia global, con potencial expansión a otros idiomas como francés, alemán o japonés.

7. BIBLIOGRAFÍA Y REFERENCIAS

[1] Godot Engine documentation (2024). "Introduction to Godot's Editor". Disponible en: https://docs.godotengine.org/en/stable/

[2] GDScript Reference (2024). "GDScript Basics". Godot documentation. Disponible en: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/

[3] LibreSprite documentation (2023). "Creating animations and spritesheets". Disponible en: https://libresprite.github.io/docs/

[4] Git Documentation (2024). "Getting started - Version control". Disponible en: https://git-scm.com/doc

[5] Godot forum (2024). Múltiples threads sobre implementación de sistemas de stats y persistencia. https://forum.godotengine.org/

[6] Reddit r/godot (2024). Community discussions sobre best practices en GDScript.

[7] Discord: Godot Engine Community Server. Conversaciones con developers experimentados sobre arquitectura de código para games.

[8] Godot Engine 4.3 (2024). Motor de videojuegos open-source. Godot foundation.

[9] LibreSprite v1.0 (2023). Software de creación de pixel art. Open-source fork de Aseprite.

[10] Visual Studio Code v1.85 (2024). Editor de código. Microsoft.

AGRADECIMIENTOS

Quiero expresar mi profundo agradecimiento a todas las personas que hicieron posible la realización de este proyecto:

A mi tutor, el profesor Cardador, por su guía durante el desarrollo del proyecto, por sus correcciones constructivas, y por permitirme la libertad creativa de explorar un concepto que me apasionaba genuinamente.

A la comunidad de Godot Engine, especialmente los usuarios en Discord y foros oficiales que respondieron pacientemente mis preguntas de principiante y compartieron su experiencia generosamente.

A los creadores de contenido educativo en YouTube (HeartBeast, GDQuest, Brackeys) cuyas lecciones formaron la base de mi conocimiento como desarrollador.

A Bandai, por crear los Tamagotchis y Digimon originales que inspiraron este proyecto y marcaron la infancia de una generación completa.

Y finalmente, a mí mismo del pasado que, a pesar de no saber nada de Godot, los cuatro meses, tuvo el coraje de empezar.

Gracias a todos.