TECNOLÓGICO NACIONAL DE MÉXICO

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

INSTITUTO TECNOLÓGICO DE CIUDAD GUZMÁN

EMPRESA:

INSTITUTO TECNOLÓGICO DE CD. GUZMÁN

REPORTE FINAL DE RESIDENCIA PROFESIONAL

TÍTULO: MINI JUEGOS QUE APOYEN EN LA EDUCACIÓN NUTRIMENTAL DE NIÑOS (PARTE 2)

CARRERA: INGENIERÍA EN SISTEMAS COMPUTACIONALES

PLAN DE ESTUDIOS: ISC-2010-224

15290933 DANIEL ULISES VELÁZQUEZ PONCE

10° SEMESTRE

CREDITOS: 250

ASESORA:

DRA. ROSA MARÍA MICHEL NAVA

Cd. Guzmán, Jalisco, a 22 de Junio del 2020

1. Preliminares

Agradecimientos

Quiero agradecer a todas las personas que me brindaron su apoyo durante el desarrollo de este proyecto, comenzando con mi asesora, la cual hizo posible que fuera parte de este proyecto, además de siempre brindarme a mí y a mis compañeros, libertad y facilidades para desarrollar el proyecto, sin mencionar que siempre estuvo ahí para revisar nuestros avances y tomar en cuenta nuestras sugerencias hacia el proyecto.

Quiero agradecer a mis padres que me apoyaron durante mi estancia como universitario y me brindaron todo lo necesario para poder llevar esta esta etapa de mi vida a cabo. A mis amigos y compañeros que compartieron este camino conmigo y finalmente a mis compañeros de equipo de en este proyecto. Les agradezco a todos de corazón.

Resumen

Nutritional Games es un proyecto que consiste en apoyar a una estudiante de Maestría en Ciencias Computacionales con su tesis. Este proyecto busca desarrollar una aplicación en la cual los niños puedan aprender sobre la educación nutricional a través de minijuegos.

La creación de este proyecto es de suma importancia ya que acorde a la Encuesta Nacional de Salud y Nutrición (ENSANUT) los niños en México de entre 5 y 11 años presentaron signos de sobrepeso y obesidad, con índices de 34.4%, 19.8% para sobrepeso y 14.6% para obesidad, los cuales forman parte del público objetivo.

Para el desarrollo de este proyecto se utilizó uno de los motores de videojuegos más populares, Unity, el cual trabaja a la par con una API que enlaza una base de datos, la cual contiene información nutrimental sobre alimentos que serán usados en la aplicación.

Como resultado principal se desarrolló una aplicación con un estilo llamativo hacia los niños que los encamina a alimentarse sanamente.

i. Índice General

ii. Índice de Figuras

iii. Índice de Tablas

2. Generalidades del proyecto

Introducción

Como se mencionó anteriormente el proyecto se enfoca a dar apoyo a los niños en su alimentación nutrimental. Para ello se diseñó una aplicación educativa disfrazada de videojuego.

Este documento está conformado de varios capítulos que presentan divisiones de la información de este proyecto.

En el capítulo de marco teórico se desarrolla la teoría del proyecto dándole fundamentación. En el capítulo de Desarrollo se describen las actividades realizadas en este proyecto, así como los papeles llevados a cabo por el autor. En el capítulo de resultados se describe lo que se obtuvo con el desarrollo, los cuales van de la mano con la fase de pruebas dentro de la metodología. En el capítulo de Conclusiones se presenta de manera breve lo que concluye el autor tras realizar el proyecto, así como sus experiencias. Finalmente se presenta la sección de competencias desarrolladas a lo largo del proyecto, así como una bibliografía de referencias que dan sustento a la información llevada a cabo en el documento.

Descripción de la empresa u organización y del puesto o área del trabajo el estudiante

El Instituto Tecnológico de Ciudad Guzmán es una institución que ofrece servicios de educación superior tecnológica en el Sur de Jalisco en áreas del desarrollo cultural, técnico y económico desde 1972. El puesto desarrollado dentro del proyecto puede definirse como: Game Developer, Game Designer, Diseñador, Ilustrador. Las responsabilidades dentro del puesto constan de desarrollar y programar parte de los videojuegos propuestos en el proyecto, así como diseñar y dibujar ilustraciones que forman parte de la aplicación.

Problemas a resolver priorizándolos

Uno de los principales problemas que se presentaron fue el diseñar un videojuego educacional, ya que normalmente un niño no suele asociar el estudio o aprendizaje con la diversión, por lo que se plantearon ideas para poder enmascarar el videojuego para que no cayera en este prejuicio. Dichas ideas a lo largo de juntas eventualmente dieron luz a la aplicación que se tiene hoy, un videojuego que ayuda al aprendizaje nutricional para niños.

Un problema que se presentó durante el desarrollo de los minijuegos fue la manera en cómo funcionaban las mecánicas, ya que en algunos casos era posible equivocarse fácilmente, tras un par de reuniones se rediseñó visualmente algunos aspectos de los minijuegos, tras los cuales se evitaron problemas similares que podrían ocurrir en el futuro.

Objetivos (General y Específicos)

El objetivo general del proyecto es:

Desarrollar dos mini juegos (rompecabezas y banda transportadora de comida) que apoyen en la educación nutrimental de niños con sobrepeso u obesidad, identificando qué y cuánto seleccionan para alimentarse.

Con base en lo anterior, se plantearon los siguientes objetivos:

- Analizar y diseñar cada mini juego para que contemple la revisión del tipo y la cantidad de alimentos que seleccionan como parte de su dieta, los niños que presentan sobrepeso u obesidad.

- Desarrollar los mini juegos, de tal manera que permitan almacenar el seguimiento del niño en cada sesión.
- Poner a prueba cada mini juego, revisando que cumpla con los requerimientos establecidos y con las características de diseño centrado en el usuario.
- Verificar que se almacenan correctamente los datos del tipo y cantidad de alimentos que niños con sobrepeso u obesidad seleccionan una vez que utilizan los mini juegos.
- Documentar los resultados obtenidos de los mini juegos, redactando también el informe final de la residencia profesional.

Justificación

Según la Encuesta Nacional de Salud y Nutrición (ENSANUT, 2012) en México los niños en edad escolar (ambos sexos), de 5 a 11 años, presentaron una prevalencia nacional combinada de sobrepeso y obesidad de 34.4%, 19.8% para sobrepeso y 14.6% para obesidad. En Jalisco, las prevalencias de sobrepeso y obesidad fueron 23.9 y 15.7%, respectivamente (suma de sobrepeso y obesidad, 39.6%). La prevalencia de sobrepeso en localidades urbanas aumentó de 2006 a 2012 de 20.0 a 24.1% y para las rurales pasó de 31.0 a 23.1%, respectivamente.

Ante el problema planteado, que implica cambios de comportamientos y establecer hábitos alimenticios saludables, es necesario mejorar la educación nutrimental desde la infancia para evitar la prevalencia de sobrepeso u obesidad en la edad adulta.

Por consiguiente, se propuso mediante este proyecto, que a través de mini juegos, los niños aprendan a seleccionar qué y cuántos alimentos deben consumir, para que puedan tomar decisiones saludables dentro del entorno donde se desenvuelven.

3. Marco teórico

En este capítulo del documento se describirán aquellos conceptos y metodologías necesarias para desarrollar el proyecto desde un aspecto teórico.

Marco teórico (fundamentos teóricos)

Scrum

Scrum es un framework que sirve como herramienta para equipos de trabajo que desean abordar problemas complejos, mientras se producen y crean productos de alta calidad de manera rápida.

Scrum, por sí mismo es un framework simple para emplear buenas prácticas, comúnmente usadas en desarrollo de software, para trabajo en equipo. Scrum, como metodología ágil ofrece una forma de trabajo en eventos conocidos como Sprint. Durante cada Sprint se asignan roles y metas específicas y a su finalización se evaluará el propio Spring por el equipo para realizar ajustes según conveniencia.

En Scrum, un proyecto se ejecuta en ciclos temporales cortos y de duración fija (iteraciones que normalmente son de 2 semanas, aunque en algunos equipos son de 3 y hasta 4 semanas, límite máximo de feedback de producto real y reflexión). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente (Product Owner) prioriza los objetivos balanceando

el valor que le aportan respecto a su coste (que el equipo estima considerando la Definición de Hecho) y quedan repartidos en iteraciones y entregas.

Las actividades que se llevan a cabo en Scrum son las siguientes (los tiempos indicados son para iteraciones de 2 semanas):

Planificación de la iteración

El primer día de la iteración se realiza la reunión de planificación de la iteración. Tiene dos partes:

Selección de requisitos (2 horas). El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que prevé que podrá completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.

Planificación de la iteración (2 horas). El equipo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos seleccionados. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se autoasignan las tareas, se autoorganizan para trabajar incluso en parejas (o grupos mayores) con el fin de compartir conocimiento (creando un equipo más resiliente) o para resolver juntos objetivos especialmente complejos.

Ejecución de la iteración

Cada día el equipo realiza una reunión de sincronización (15 minutos), normalmente delante de un tablero físico o pizarra (Scrum Taskboard). El equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la iteración,

obstáculos que pueden impedir este objetivo) para poder hacer las adaptaciones necesarias que permitan cumplir con la previsión de objetivos a mostrar al final de la iteración. En la reunión cada miembro del equipo responde a tres preguntas:

- ¿Qué he hecho desde la última reunión de sincronización para ayudar al equipo a cumplir su objetivo?
- ¿Qué voy a hacer a partir de este momento para ayudar al equipo a cumplir su objetivo?
- ¿Qué impedimentos tengo o voy a tener que nos impidan conseguir nuestro objetivo?

Durante la iteración el Facilitador (Scrum Master) se encarga de que el equipo pueda mantener el foco para cumplir con sus objetivos.

- Elimina los obstáculos que el equipo no puede resolver por sí mismo.
- Protege al equipo de interrupciones externas que puedan afectar el objetivo de la iteración o su productividad.

Durante la iteración, el cliente junto con el equipo refina la lista de requisitos (para prepararlos para las siguientes iteraciones) y, si es necesario, cambian o replanifican los objetivos del proyecto (10%-15% del tiempo de la iteración) con el objetivo de maximizar la utilidad de lo que se desarrolla y el retorno de inversión.

Inspección y adaptación

El último día de la iteración se realiza la reunión de revisión de la iteración. Tiene dos partes:

- Revisión (demostración) (1,5 horas). El equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo. En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.
- Retrospectiva (1,5 horas). El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad. El Facilitador se encargará de eliminar o escalar los obstáculos identificados que estén más allá del ámbito de acción del equipo.

En un proyecto de Scrum, la principal tarea de el equipo que lo conforma es desarrollar un software de calidad. Por lo que es esencial definir las características que debe tener el producto a desarrollar, de esta manera se evitaran obstáculos que podrían interrumpir el flujo de desarrollo para el equipo.

El equipo Scrum está formado por los siguientes roles:

- Scrum Master: persona que lidera al equipo guiándolo para que cumpla las reglas y procesos de la metodología. Gestiona la reducción de impedimentos del proyecto y trabaja con el Product Owner para maximizar el ROI (Return of Investment/Retorno de la Inversión).
- Product Owner (PO): representante de los accionistas y clientes que usan el software. Se focaliza en la parte de negocio y él es responsable del ROI (Return of

Investment/Retorno de la Inversión) del proyecto (entregar un valor superior al dinero invertido). Traslada la visión del proyecto al equipo, formaliza las prestaciones en historias a incorporar en el Product Backlog (una lista de nuevas características, cambios a las características existentes, correcciones de errores y cambios de infraestructura) y las prioriza de forma regular.

- Team: grupo de profesionales con los conocimientos técnicos necesarios y que desarrollan el proyecto de manera conjunta llevando a cabo las historias a las que se comprometen al inicio de cada Sprint.

En la *Figura 1* se puede apreciar una representación del ciclo de vida de Scrum con los procesos anteriormente mencionados.

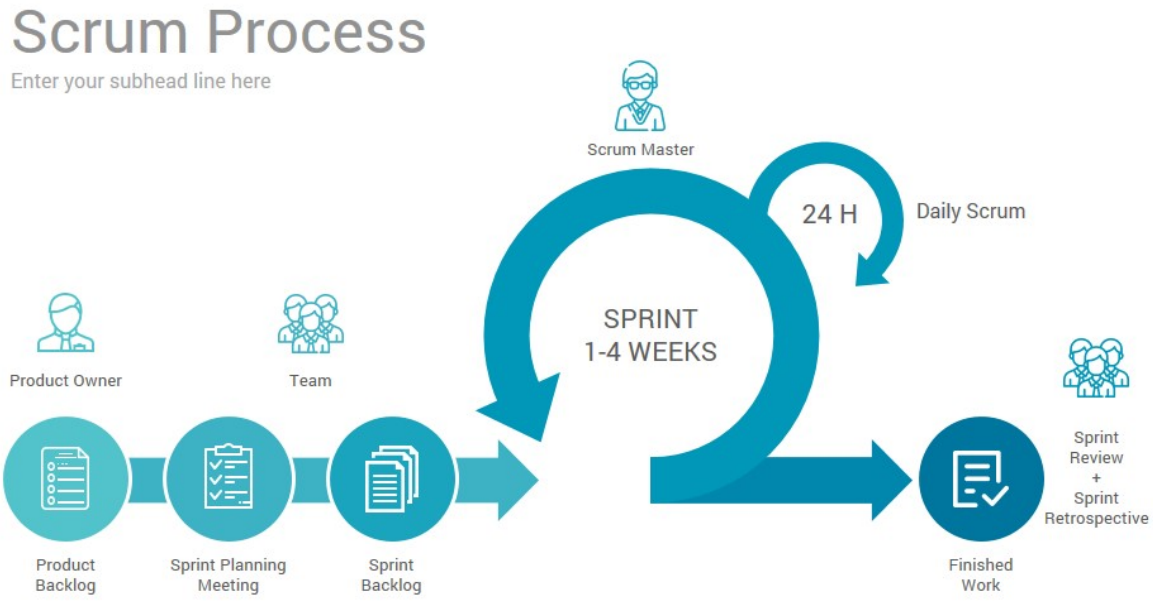

Figura 1: Proceso de Scrum

Fases del ciclo de vida de desarrollo de software

A continuación, se expone la teoría que sustenta cada una de las fases del ciclo de vida del proyecto.

Fase de análisis

En la fase de análisis se definieron las premisas básicas, por lo que la teoría de diseño de videojuegos sustenta el proceso por el cual se diseñó la plataforma.

Definición y premisas de diseño de videojuegos

Las fundaciones de diseño del videojuego son las siguientes:

Fundación

Los títulos siempre deben de comenzar con el jugador en su estado más vulnerable. Una situación clásica es que el jugador comience con mejoras parciales o completas de sus habilidades, y después de un evento, se le arrebaten.

Uno de los principales errores al diseñar con enfoque al progreso, es hacer el inicio del juego muy tedioso y frustrante para mostrar la evolución del personaje en escenarios posteriores, esto puede llegar a incluir el arrebato de habilidades como correr, saltar e incluso atacar.

Es errado querer que los jugadores tengan inmediatamente una experiencia frustrante al empezar el juego. Si se va a comenzar sin habilidades esenciales, se requieren incluir de manera muy rápida, para que el jugador pueda comenzar a experimentar las mecánicas básicas. El núcleo de tu título debe de ser sólido desde el primer minuto.

Los títulos deben comenzar controlando responsivamente al personaje y mejorar a partir de ahí.

Diseño orientado al progreso

Una de las mecánicas más importantes en los títulos de este subgénero, son las que permiten la mejora o el progreso. La mejora de habilidades es un estándar dentro de los juegos de acción, sin embargo, en este caso se pretende ir un paso más allá, pues las habilidades van a modificar de manera fundamental cómo se juega.

La mejora de habilidades es la parte más importante en el diseño, puesto que de aquí depende el éxito del videojuego. Las mejoras en las habilidades del personaje van desde aumento de salud, daño y el resto de las variables básicas, hasta permitir al jugador teletransportarse, saltar de manera infinita, evadir y contrarrestar ataques enemigos.

Habilidades como llaves

Un concepto básico que hay que concretar es el de habilidades como llaves. Una llave es un objeto que permite al jugador acceder a un objeto, sección o mecánica. Es común dentro del diseño de videojuegos que se diseñe una habilidad como llave, alterando el entorno en el que se mueve el jugador y permitiéndole acceder a secciones que anteriormente no podía. Un buen diseño permitirá al jugador cambiar la ruta por la que se mueve a través del juego.

Diseño ambiental

El diseño ambiental o de escenarios está diseñado para incentivar y recompensar la exploración. Generalmente, el ambiente es un mundo interconectado, dividido en diversas secciones o escenarios. Esto permite al jugador tomar diversas rutas para llegar a zonas importantes.

A medida que el jugador avanza dentro de la historia del juego, los obstáculos deben cambiar para reflejar las habilidades que el personaje principal ha adquirido hasta el momento. El ejemplo más prominente tiene que ver con la movilidad. A medida que el jugador progresa, se vuelve más móvil, por lo que el diseño de escenarios comenzará a colocar plataformas más altas o separadas una de otra, para sacar provecho y forzar al jugador a implementar sus habilidades acrobáticas.

Un buen diseño de habilidades conlleva un buen diseño de escenarios, dado que puede funcionar como fuente de inspiración a la hora de construir los niveles de la mitad o final del juego.

Coleccionables y exploración

Como se dijo anteriormente, el género posee un enfoque muy prominente hacia la exploración y recompensa al jugador por hacerlo. Es muy importante que el videojuego posea secretos, los cuales generalmente vienen en dos presentaciones.

La primera presentación son los secretos que el jugador puede encontrar con las habilidades y mecánicas básicas. El otro caso, son los que requieren una habilidad en específico. De aquí, parte la siguiente premisa de diseño.

Backtracking

Backtracking es el término que se le otorga a regresar a áreas previamente completadas en busca de secretos, coleccionables y caminos a los que previamente no se tenía acceso, para tentar al jugador y otorgarle un sentimiento de progreso, al ver la diferencia en jugabilidad desde la última vez que pasó por la zona.

Existe un peligro inminente al diseñar las zonas con backtracking, el cual consta en volverlo tedioso al diseñar la zona muy larga, que tome mucho tiempo regresar, o sobrexplotar la mecánica.

Una de las técnicas más utilizadas para minimizar este riesgo, es el crear estaciones de fast travel. Una estación de fast travel es una mecánica en la cual se interactúa con un objeto del juego que permite viajar a zonas previamente finalizadas, esto vuelve el backtracking más corto y simple de implementar.

Planeación de mecánicas de sistema

Para construir las mecánicas básicas, hay que seguir los siguientes pasos:

1. Definir la premisa del videojuego y qué es lo que lo hace único.

    - ¿El videojuego será 3D o 2D?
    - ¿Cómo funciona la cámara? (Side scroller, overhead, controlada por el jugador, tercera persona, basada en spline, programada, estática, etcétera).
    - ¿Qué es lo que controla el jugador? (Un personaje, vehículo, objeto, etcétera).
    - ¿De qué estilo es el juego? (Sci-fi, fantasía, realista, cartoony, exageración, o una combinación de estilos).
    - ¿Qué es lo que define al juego y lo hace único? (Mecánicas, premisa, trabajo visual, trabajo de audio, historia, personajes).

2. Definir las mecánicas básicas del jugador.

    - ¿Cómo se mueve el personaje? (Vuela, camina, corre, se desliza, etcétera).
    - ¿Qué hace el jugador para interactuar con el mundo? (Salta, dispara, corta, besa, agarra, toca).

3. Definir cuáles son las habilidades, armas, herramientas, equipamiento que el personaje va a poder adquirir.

- ¿De cuántas maneras se puede mejorar la movilidad del personaje? (Salta más alto, corre, se agacha, se desliza, nada, gira, etc).
- ¿De cuántas maneras se puede transformar el personaje? (Cambia su tamaño, se convierte en otro personaje, altera su estado, sus habilidades). Todas las transformaciones modifican el movimiento, propiedades, colisiones y vulnerabilidades, por lo que requieren limitaciones y mecánicas únicas.

- ¿De cuántas maneras puede crecer? (Armas, proyectiles, magia, tipos de daño elemental, velocidad).

- ¿De cuántas maneras puede ver el mundo y cómo el mundo lo ve a él? (Opciones de visibilidad, invisibilidad, luces, dimensiones, secretos).

- ¿Cuántos tipos de obstáculos y hazards hay en el juego y de qué maneras pueden prevenirse? (Armadura, sombreros, trajes, amuletos para prevenir daño de hazards como el fuego, lava, agua, enemigos, electricidad, vacío, etc).

- ¿Cuáles habilidades se pueden adquirir que le permitan alterar el ambiente? (Detener el tiempo, cambiar la gravedad, fast travel, teletransportación).

- ¿El personaje tiene salud, requiere recargar munición, combustible o alguna otra variable? ¿Se recargan por sí mismas? (Con cuánto comienza y cuál es la capacidad máxima).

Implementación de sistemas

Una vez que se planearon y conceptualizaron las mecánicas de sistema, es momento de implementarlas. Un buen diseño debe implementar las mecánicas de manera versátil, ofreciendo múltiples usos para la misma habilidad. Las mecánicas crean nuevas oportunidades en el campo de la jugabilidad e incluyen limitaciones que creen retos para el jugador. En esta área es en la que fallan la gran mayoría de títulos.

Es recomendable colocar una lista de las habilidades a ser adquiridas por el jugador. Podría lucir de la siguiente manera:

- Habilidades por defecto: caminar, cortar, saltar.
- Upgrades a habilidades: salto alto, proyectil de fuego, transformación en gato, traje de veneno, visión nocturna, gravedad invertida, correr.

Estándares de habilidades

Es recomendable tener estándares de construcción que muestren los rangos en los que se desempeñarán las habilidades. Esto ayuda mucho al momento de probar y balancear el juego. Por ejemplo, que la altura máxima por defecto de salto sea de 2 metros y la altura máxima del salto alto sea de 5 metros. Siempre es buena práctica tener ilustraciones y diagramas en los que se represente visualmente el estándar.

Diseño de pruebas de secuencia y progresión

Una vez que se implementan las mecánicas, se necesita diseñar un entorno de pruebas que te permita probar los estándares de habilidades previamente definidos y codificados.

El entorno de pruebas es una colección de cuartos que permitirá diseñar la secuencia en la que las habilidades van a desbloquearse y presentarse al jugador. Los objetivos del entorno de pruebas son los siguientes:

- Permitir al jugador entrar al cuarto y ver una representación gráfica de la habilidad que van a desbloquear.
- Forzar al jugador a una situación en la que no opción y debe de tomar el ítem.
- Permitir al jugador el interactuar con el objeto para adquirir la nueva habilidad.
- Permitir al jugador utilizar la habilidad adquirida para poder entrar al siguiente cuarto.

Cada uno de los cuartos deben encapsular la experiencia, no necesitan ser muy grandes y no necesitan verse bien. Toda la información debe ser provista de inmediato en la cámara y debe de representar el uso más general de la habilidad. La situación debe de ser muy simple de comprender. Una vez que los cuartos se encuentran diseñados, hay que colocarlos en un orden en el que sea interesante adquirirlos.

Una vez respondidas las preguntas, hay que mejorar el diseño modificando el orden y los estándares de las habilidades, repitiendo la cuantificación y pruebas hasta que exista una completa satisfacción.

Pruebas ambientales, de objetos y enemigos

Hay que generar otro entorno de pruebas para ajustar el ambiente, los objetos y los enemigos a implementar dentro del juego. Algunos de los elementos más importantes a colocar dentro de este entorno son los siguientes:

- Puertas: las puertas separan los diversos escenarios y áreas del juego, además de ser una excelente manera de bloquear el progreso hasta que se adquiera la llave correspondiente.
- Destructibles: son objetos con los que se puede interactuar al destruirlos, romperlos o dañarlos. Cada uno debe servir un propósito y ser ajustado para ser superados con éxito tras adquirir la herramienta.
- Obstáculos: son todos los objetos que puedan dañar al jugador. Todos deben de servir un propósito y ser superados con facilidad con la herramienta correcta.
- Mecanismos y objetos que se mueven: son todo lo que requiera el uso de habilidades para ser activadas. Ejemplos de este tipo son las ziplines, los elevadores, plataformas móviles, engranes y pistones.

Enemigos

Todos los enemigos deben de ser divertidos y suponer un reto, sin importar con qué habilidad se les enfrente. Se deben de planear de tal manera que exista por lo menos un enemigo por habilidad, o que sea vulnerable a la misma. Es importante tener enemigos que supongan un reto, inclusive cuando el jugador ha llegado al punto máximo de crecimiento. Son una parte muy importante en la ecuación que asegura el éxito del juego.

Diseño de niveles

Los niveles deben de ser diseñados de tal manera que supongan un reto e incentiven el uso de las habilidades adquiridas por el jugador, para otorgarle un sentimiento de progreso y recompensa. Al diseñar un nivel, hay que crear un mapa y una simbología que represente las diversas partes que lo conforman, como lo son las puertas, rutas, nuevas habilidades, rutas, jefes y secretos. Se deben de etiquetar los bloques o secciones de un nivel en base a su propósito. Algunos de los propósitos generales que puede llegar a tener una sección del nivel, son los siguientes:

Exploración

Asegurar el uso de una habilidad

- Ruta alternativa.
- Almacenar un secreto.
- Cuarto de combate.
- Cuarto de introducción.
- Cuarto de transición.
- Cuarto de puzzle.
- Cuarto de jefe.

- Cuarto de transición.
- Salida.
- Salida alternativa.

Diseño

La fase de diseño está soportada por la teoría de la arquitectura general del proyecto y los módulos que la integran. Además, se escogieron las herramientas necesarias para llevar a cabo el proyecto, incluyendo los lenguajes de programación y el entorno de desarrollo.

Unity3D

Unity 3D o Unity es una plataforma de desarrollo de videojuegos la cual ofrece un motor de videojuegos multiplataforma. Este motor ofrece un editor en el cual podremos manipular diversos contenidos de multimedia, animaciones, interfaces y su estelar, videojuegos.

En la *Figura 2* puede apreciarse el logotipo de Unity.

Figura 2: Logotipo de Unity3D

Adobe Photoshop

Adobe Photoshop o Photoshop es un editor gráfico utilizado en diferentes áreas como la fotografía, el dibujo y la edición. Este editor posee una variedad de herramientas las cuales simulan a un taller de fotos, así como múltiples funciones para la edición digital. Una característica de este editor es que trabaja alrededor de un espacio por capas las cuales facilitan el orden y acomodo de un proyecto.

En la *Figura 3* se puede apreciar el logotipo de Photoshop.

Figura 3: Logotipo de Adobe Photoshop

4. Desarrollo

El proyecto surge para apoyar a una estudiante de Maestría en Ciencias de la Computación en su tesis, la cual involucra una investigación sobre el impacto que tiene el uso de videojuegos educativos en los hábitos alimenticios de los niños. Al igual que el marco teórico, se presentará la información en el mismo orden que soporta la metodología de Scrum.

Análisis

Durante la fase de análisis se realizó una serie de reuniones al inicio del proyecto. En ellas, se reunió con la asesora externa, la cual proporcionó la información necesaria para poder trabajar alrededor de ella. Una vez que se tuvo conocimiento sobre la alimentación nutrimental de los niños, se propusieron y definieron los minijuegos que formarían parte de la aplicación; así como los roles que tomaría cada miembro del equipo y las fechas correspondientes para las juntas Sprint.

Diseño

En la fase de diseño se definió la arquitectura general del proyecto y de cada uno de los módulos que la integrarían. El trabajo realizado incluye el diseño de las ilustraciones que conforman parte de los sprites (imágenes que pueden ser utilizadas para representar visualmente a personajes y el entorno de un videojuego, así como también pueden ser utilizadas para animación) e interfaces de la aplicación; el diseño de los minijuegos, así como su programación y el enlace con la API.

Para ello se crea una plataforma que consta de dos partes:

1. Una API que simplifica la conexión entre la base de datos y los videojuegos con el fin de escalar el proyecto de manera más simple.
2. Una aplicación que contiene una serie de videojuegos que reta al niño para que refuerce sus conceptos en el área nutricional mientras se divierte.

Para el desarrollo del proyecto en el rol de Game Developer, se definieron durante las reuniones de Sprint las tareas que se llevarían a cabo para construir los minijuegos de la aplicación. Como rol de Ilustrador y Diseñador se mostraría al final de cada Spring el avance de los diseños de la aplicación a los integrantes del equipo para verificar que dichos diseños estén acorde a lo establecido durante las juntas.

Durante las juntas se propusieron diversas opciones, temáticas y estilos de juego para atraer al niño a la aplicación y en su parte camuflar la alimentación nutrimental con un diseño atractivo para las edades objetivo.

Como propuesta se planteó la idea de una “mascota” que representaría a la aplicación, tras algunas juntas se propuso de diseño de un fantasma apodado “Blanky” (un nombre que deriva de sabanita en inglés) el cual con una pequeña historia de trasfondo alentaría a los niños a tomar un estilo de vida nutrimental a través del juego.

En la *Figura 4* se puede apreciar los primeros diseños de la mascota, Blanky:

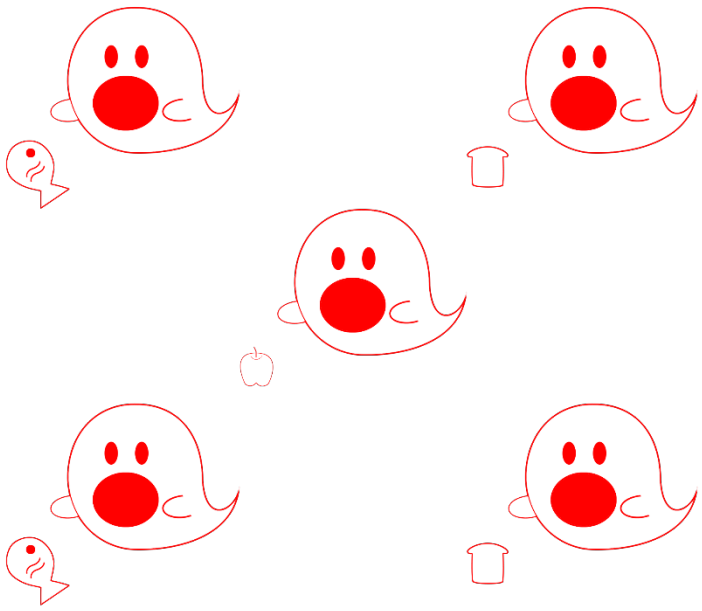

Figura 4: Primer diseño de la mascota

Desarrollo del Minijuego: Rompecabezas sobre el plato del buen comer

Una de las tareas llevadas a cabo fue el desarrollo de un minijuego el cual simularía a un rompecabezas incorporando las bases del plato del buen comer.

Las reglas constan de lo siguiente: al iniciar este minijuego se mostrarían 6 alimentos aleatorios de la base de datos, los cuales, posteriormente tendrán que ser arrastrados con el mouse al grupo correspondiente del plato del buen comer.

En un inicio se planteó utilizar un diseño plato del buen comer clásico para este minijuego, pero dado su diseño se era muy fácil cometer un error involuntariamente, ya que las porciones

de cada grupo están demasiado cercas y a menos de que se colocara el alimento en una sección del grupo alejada de los demás grupos este error era muy posible.

En la *Figura 5* se puede apreciar el diseño inicial para el plato del buen comer.

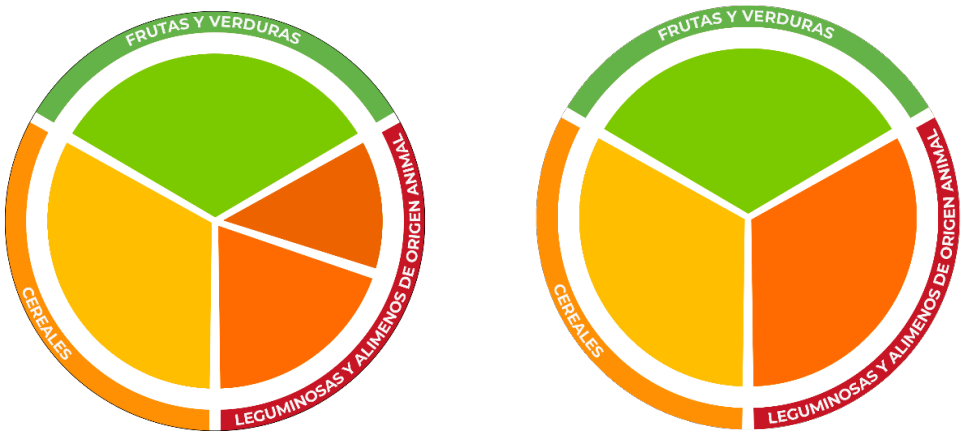

Figura 5: Diseño del plato del buen comer

Ante este problema se propuso reemplazar el plato del buen comer clásico por platos separados que representaran a cada grupo, lo cual mejoro drásticamente la experiencia de usuario.

En la *Figura 6* se puede apreciar el diseño alternativo para el plato del buen comer.

Figura 6: Diseño alternativo al plato del buen comer

Desarrollo del Minijuego: Banda de comida

Como segundo minijuego se optó por el desarrollo de un minijuego que simularía a una banda que transportara continuamente comida de la cual se tendrá que seleccionar los alimentos que correspondan a un grupo en particular en diferentes porciones de tiempo.

En la *Figura 7* se puede apreciar el diseño de la banda de comida.

Figura 7: Diseño de la banda de comida

Las reglas constan de los siguiente: Al iniciar el juego se mostrará a Blanky seguido de un letrero y una banda de comida la cual transportará alimentos continuamente durante la duración del minijuego. Además, cada cierto tiempo el letrero mostrara el nombre de un grupo de alimento diferente del plato del buen comer. Como objetivo el niño tendrá que alimentar a Blanky con un alimento que corresponda al grupo que se muestra en el letrero en su debido tiempo.

En la *Figura 8* se puede apreciar el diseño del letrero.

Figura 8: Diseño del letrero

Ya que la aplicación maneja grupos de alimentos del plato del buen comer se optó por agregar un glosario el cual le proporcionaría el conocimiento necesario para poder interactuar sin problemas con los minijuegos de la aplicación.

Codificación

En la fase de codificación se realizó el desarrollo de la programación pillar de los minijuegos, la cual incluye el enlace con la API.

Como primer paso se agregaron los "atlas de sprites" para posteriormente con la ayuda del editor dividirlos y utilizarlos individualmente.

En la *Figura 9* se pueden apreciar los sprites de la aplicación en el inspector de Unity. En la *Figura 10* se puede apreciar un atrás de sprites desglosado.

Figura 9: Sprites del proyecto en Unity

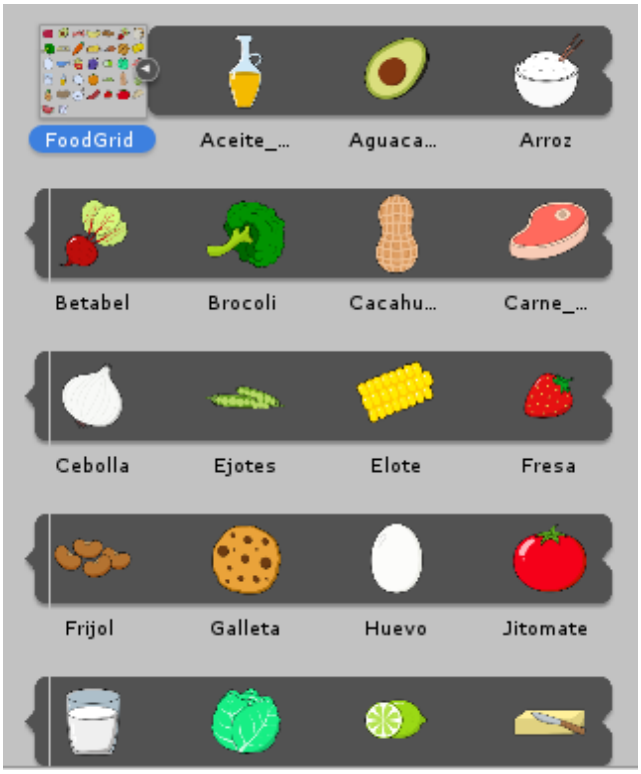

Figura 10: Ejemplo de sprites individuales

Ya que todos los minijuegos requieren de la acción de arrastrar y soltar (drang and drop) objectos se creó un script llamado DragDrop, el cual contendría las funciones necesarias para hacer uso de esta acción.

En la *Figura 11* se puede apreciar la estructura del código en el script DragDrop.

```
public class DragDrop : MonoBehaviour
{
    private bool isDragging;
    Vector2 originalPos;

    public void OnMouseDown()
    {
        isDragging = true;
    }

    public void OnMouseUp()
    {
        isDragging = false;
    }

    void Start()
    {
        originalPos = transform.position;
    }

    private void OnTriggerStay2D(Collider2D collision)
    {
        if (collision.gameObject.tag != gameObject.tag)
        {
            transform.position = originalPos;
            // Debug.Log("Wrong food placement");
        }
    }

    void Update()
    {
        if(gameObject.tag != "Undraggable")
        {
            if (isDragging)
            {

                GetComponent<CircleCollider2D>().isTrigger = false;
                Vector2 mousePosition = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;
                transform.Translate(mousePosition);
            }
            else
            {
                GetComponent<CircleCollider2D>().isTrigger = true;
            }
        }
        else
        {
            GetComponent<CircleCollider2D>().isTrigger = false;
        }

    }
}
```

Figura 11: Estructura del script DragDrop

Se hizo uso de la API para poder spawnear (generar) los alimentos como objetos en la escena del minijuego. Para ello, se creó el script spawnPoint, el cual mediante el inspector se manipula una serie de listas para almacenar el id correspondiente de cada grupo de alimentos acorde a la base de datos.

En la *Figura 12* se puede apreciar las listas de alimentos en el inspector de Unity.

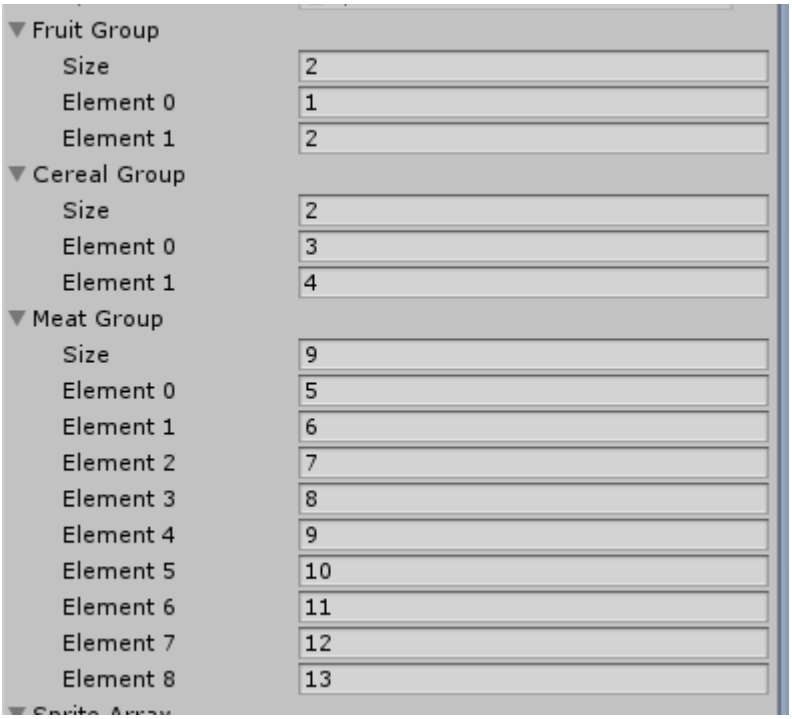

Figura 12: Listas de grupos de alimentos en el inspector

Adicionalmente se añadió un arreglo que contendrá cada Sprite correspondiente de cada alimento, como puede apreciarse en la *Figura 13* la cual muestra un arreglo que contiene los sprites de los alimentos en el inspector.

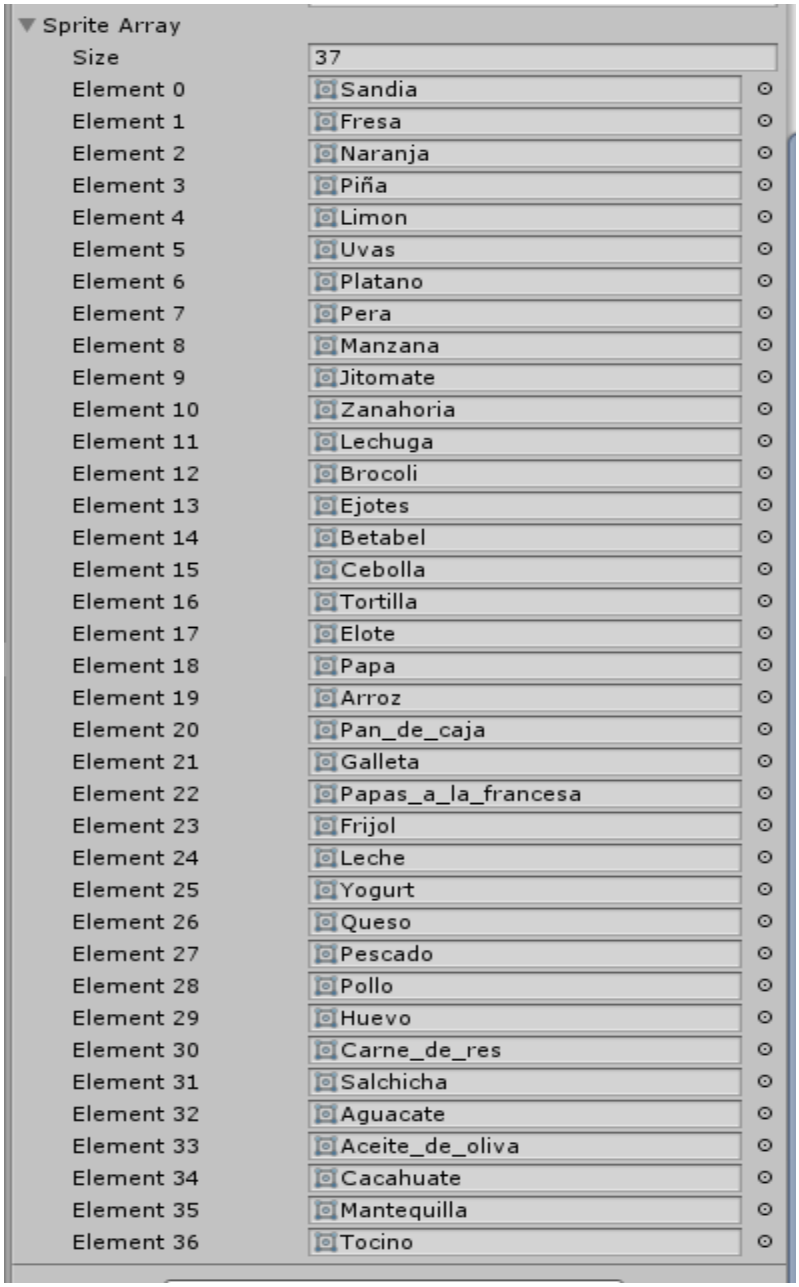

Figura 13: Array que contiene los sprites de alimentos en el inspector

Ya que se requiere spawnear un objeto en una posición en especifica se guardó la posición original de las coordenadas del objecto en la escena, seguido de la ejecución del método Spawner(), el cual conseguía un alimento aleatorio a través de la API.

En la *Figura 14* se puede apreciar la ejecución del método Spawner(). En la *Figura 15* puede apreciarse la estructura del método Spawner().

```
Vector2 spawnPos;

 void Start()
{
    spawnPos = transform.position;

    Spawner();
}
```

Figura 14: Gravado de la posicion del spawn point y llamada al metodo Spawner()

```
public void Spawner()
{

    List<Food> list = APIConnection.GetAllFood();

    int index = Random.Range(0, list.Count);
    var currentFood = list[index];
    GameObject newObject = new GameObject(currentFood.group_id.ToString());

    setFoodGroutTag(newObject, currentFood.group_id);

    newObject.layer = 9; // Foods layer
    newObject.transform.localScale += new Vector3(-0.65f, -0.65f, 1); // to make every newObject smaller
    newObject.transform.position = spawnPos;
    SpriteRenderer renderer = newObject.AddComponent<SpriteRenderer>();
    renderer.sprite = spriteArray[currentFood.food_id - 1];
    newObject.GetComponent<SpriteRenderer>().sortingLayerName = "Food";
    newObject.GetComponent<SpriteRenderer>().sortingOrder = 1;
    newObject.AddComponent<CircleCollider2D>();
    newObject.GetComponent<CircleCollider2D>().isTrigger = true;
    newObject.AddComponent<DragDrop>();

    linkedFood = newObject;
}

public void setFoodGroutTag(GameObject nO, int g_id)
{
    if (fruitGroup.Contains(g_id)) {
        nO.tag = "Fruit";
    }
    if (cerealGroup.Contains(g_id))
    {
        nO.tag = "Cereal";
    }
    if (meatGroup.Contains(g_id))
    {
        nO.tag = "Meat";
    }
    //nO.tag = "Draggable";
}
```

Figura 15: Estructura del metodo Spawner()

Se creó un objeto PiecesSpawner el cual contendrá objetos hijo llamados SpawnPoints los cuales hacen uso del script spawnPoint.

En la *Figura 16* se puede apreciar la jerarquía de PiecesSpawner en el proyecto.

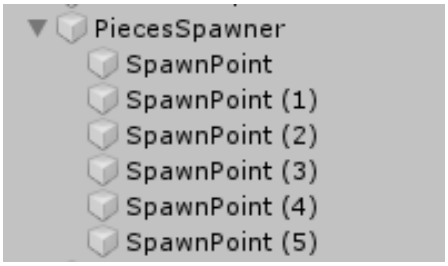

Figura 16: Jerarquia de los objectos spawn en el proyecto

Además, se enlazan los objetos generados con su respectivo SpawnPoint Padre para posteriormente generar nuevos alimentos en la misma posición como se muestran en la *Figura 17*. En conjunto con el código en la *Figura 17* que manda a llamar al método Spawner() siempre y cuando se cumpla la condición para generar de nuevo alimentos (respawneo) como se aprecia en la *Figura 18*.

```
private GameObject linkedFood;
```

```
linkedFood = newObject;
```

Figura 17: Estructura de linked Food

```
void Update()
{
    if(linkedFood.tag == "Undraggable")
    {
        Spawner();
    }
}
```

Figura 18: Estructura de respawneo

En el caso del minijuego del plato del buen comer, se requirió la creación de un script llamado GroupCollider, el cual comprobaría si el alimento fue soltado dentro de su grupo de comida correspondiente.

En la *Figura 19* puede apreciarse la estructura del script GroupCollider.

```
public class GroupCollider : MonoBehaviour
{
    public static int totalScore;

    private void OnTriggerStay2D(Collider2D other)
    {
        if (other.gameObject.tag == gameObject.tag)
        {
            other.tag = "Undraggable";
            totalScore += 10;
            // Debug.Log("Score: " + totalScore);
        }


    }
}
```

Figura 19: Estructura del script Group Collider

En el caso del minijuego la banda de comida, se creó una serie de variantes de scripts previamente mostrados que cumplían la misma función dentro de este minijuego, ya que dada la mecanica que requiere este minijuego causaba conflictos si se reutilizaban completamente dichos scripts. Como primer script se creó DDband, el cual cumple las mismas funciones que DragDrop con ligeras modificaciones.

En la *Figura 20* puede apreciarse la estructura del script DDBand.

```
public class DDBand : MonoBehaviour
{
    public static bool animDrag;
    private bool isDragging;
    private bool touchingGround;
    private string currentCollision;

    public void OnMouseDown()
    {
        isDragging = true;
        animDrag = isDragging;
    }

    public void OnMouseUp()
    {
        isDragging = false;
        animDrag = isDragging;
    }


    void Update()
    {

        transform.Translate(Vector3.left * Time.deltaTime, Space.World);

        // Check if the gameObject is draggable, if so, the mouse will drag it untill released
        if (gameObject.tag != "Undraggable")
        {
            if (isDragging)
            {
                Vector2 mousePosition = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;
                transform.Translate(mousePosition);
                GetComponent<Rigidbody2D>().gravityScale = 0.0f;
            }
            else
            {
                GetComponent<Rigidbody2D>().gravityScale = 1f;
            }
        }
        else
        {
            GetComponent<BoxCollider2D>().isTrigger = false;
        }

    }
}
```

Figura 20: Estructura del script DDBand

Se creó un objecto Spawner el cual contendrá el script foodSpawner en el cual se puede manipular el intervalo en el que aparece cada alimento, se especifica el tipo de alimento y se asignan los sprites correspondientes.

En la *Figura 21* puede apreciarse la estructura del script foodSpawner en el inspector de Unity.

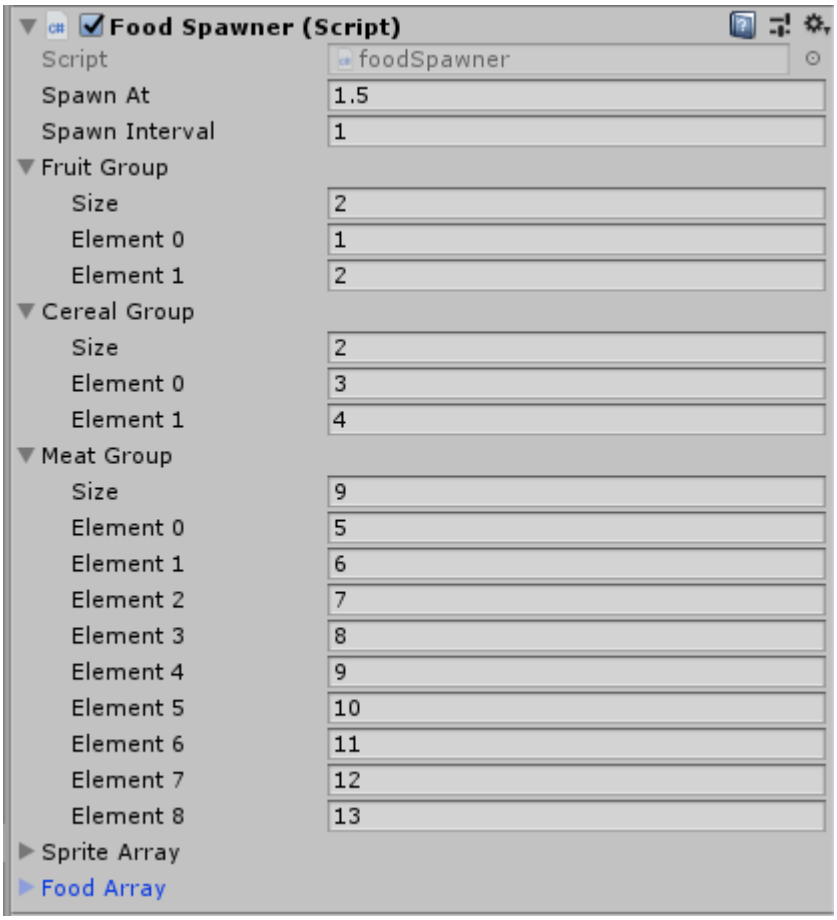

Figura 21: Estructura del script foodSpawner en el insperctor

En la *Figura 22* y *Figura 23* puede apreciarse la estructura del script foodSpawner.

```
public class foodSpawner : MonoBehaviour
{
    public float spawnAt;
    public float spawnInterval;
    public List<int> fruitGroup;
    public List<int> cerealGroup;
    public List<int> meatGroup;
    public Sprite[] spriteArray;
    public GameObject[] foodArray;
    private int currentFood = 0;
    Vector2 spawnPos;

    GameObject clone;

    void Start()
    {
        spawnPos = transform.position;

        List<Food> list = APIConnection.GetAllFood();
        foreach (Food food in list)
        {


            GameObject newObject = new GameObject(food.group_id.ToString());

            setFoodGroutTag(newObject, food.group_id);

            newObject.layer = 9; // Foods layer
            newObject.transform.localScale += new Vector3(-0.65f, -0.65f, 1);
            newObject.transform.position = spawnPos;
            SpriteRenderer renderer = newObject.AddComponent<SpriteRenderer>();
            renderer.sprite = spriteArray[food.food_id-1];
            newObject.GetComponent<SpriteRenderer>().sortingLayerName = "Food";
            newObject.GetComponent<SpriteRenderer>().sortingOrder = 1;
            newObject.AddComponent<BoxCollider2D>();
            newObject.AddComponent<Rigidbody2D>();
            newObject.GetComponent<Rigidbody2D>().freezeRotation = true;
            newObject.AddComponent<DDBand>();
            newObject.AddComponent<avoidFood>();

            newObject.SetActive(false);
            foodArray[currentFood] = newObject;

            currentFood++;
        }

        InvokeRepeating("Spawner", spawnAt, spawnInterval);
    }
```

Figura 22: Estructura del script foodSpawner 1/2

```
public void Spawner()
{
    int rand = Random.Range(0, foodArray.Length);
    clone = Instantiate(foodArray[rand]);
    clone.SetActive(true);
    clone.name = foodArray[rand].name;
}

public void setFoodGroutTag(GameObject nO, int g_id)
{
    if (fruitGroup.Contains(g_id))
    {
        nO.tag = "Fruit";
    }
    if (cerealGroup.Contains(g_id))
    {
        nO.tag = "Cereal";
    }
    if (meatGroup.Contains(g_id))
    {
        nO.tag = "Meat";
    }
}
```

Figura 23: Estructura del script foodSpawner 2/2

Además, se creó un objecto llamado Stop el cual contiene el script bandStop, el cual detiene la caída de los alimentos tras aparecer y hace que se muevan constantemente hacia la izquierda para simular el comportamiento de una banda transportadora.

En la *Figura 24* puede apreciarse la estructura del script bandStop.

```
public class bandStop : MonoBehaviour
{

    private Rigidbody2D rb;
    private float moveSpeed = 2f;

    private void OnCollisionStay2D(Collision2D other)
    {
        // Moves a gameOjbect to T H E  V O I D when it collides with the band
        other.transform.Translate(Vector3.left * moveSpeed * Time.deltaTime, Space.World);
    }

    private void OnCollisionEnter2D(Collision2D other)
    {
        //Debug.Log("It enters");
    }
    private void OnCollisionExit2D(Collision2D other)
    {
        //Debug.Log("It exits");
    }
}
```

Figura 24: Estructura del script bandStop

Para ambos minijuegos se crearon “Preafabs” los cuales actúan como platillas que contienen GameObject que pueden ser reutilizados en distintas escenas de la aplicación. El primer prefab creado, Timer como se muestra en la *Figura 25*, este prefab muestra un contador en la escena que disminuye cada segundo y al llegar a cero, se carga la escena que contiene la puntuación final, este prefab contiene el script Timer.

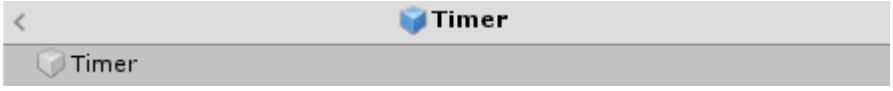

Figura 25: Jerarquia del prefab Timer

En la *Figura 26* puede apreciarse la estructura del script Timer.

```
public class Timer : MonoBehaviour
{

    float currentTime = 0f;
    public float startingTime;
    int timeLeft;

    void Start()
    {
        currentTime = startingTime;
    }
    void Update()
    {
        currentTime -= 1 * Time.deltaTime;
        timeLeft = (int)currentTime;
        GetComponent<TextMesh>().text = "Tiempo: " + timeLeft;
        if(currentTime < 0)
        {
            SceneManager.LoadScene("LevelComplete"); //Scene 3
        }
    }
}
```

Figura 26: Estructura del script Timer

El segundo prefab, GameScore muestra la puntuación actual de cada minijuego y una vez que termine la partida cargar la puntuación obtenida a la escena que contiene la puntuación final. Este prefab contiene el script scoreControl el cual maneja el score de cada minijuego como se aprecia en la *Figura 27*.

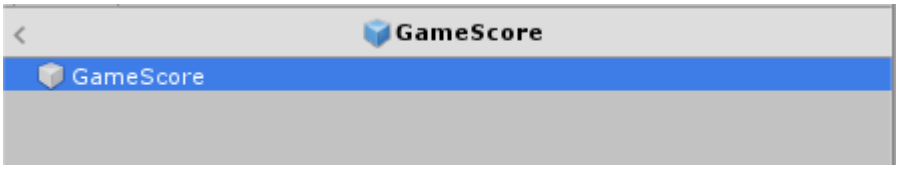

Figura 27: Jerarquia del prefab GameScore

En la *Figura 28* puede apreciarse la estructura del script scoreControl.

```
public class scoreControl : MonoBehaviour
{
    public string levelCompleteScore;

    void Start()
    {
        levelCompleteScore = "0";
    }

    void Update()
    {
        levelCompleteScore = ((Mouth.score) + (GroupCollider.totalScore)).ToString();


        GetComponent<TextMesh>().text = "Puntuación: " + levelCompleteScore;
    }
}
```

Figura 28: Estructura del script scoreControl

Pruebas

En esta fase se puso a prueba el funcionamiento de la aplicación con el fin de encontrar y depurar todas las posible fallas y errores que podría contener de manera que después de ser solucionadas dichas fallas, la aplicación se encuentre completamente funcional y pueda ser entregada en su totalidad para que sea usada por el usuario final.

En la *Tabla 1* se puede apreciar es comportamiento obtenido al interactuar con la escena de menú principal.

Escena: Menú Principal			
Acción	**Resultado esperado**	**Resultado**	**Porcentaje de cumplimiento**
Presionar opción minijuegos	Acceder la ventana: submenú minijuegos	Correcto	100%
Presionar opción glosario	Acceder a la escena: glosario	Correcto	100%
Presionar opción salir	Salir de la aplicación	Correcto	100%

Tabla 1: Verificación del menú principal

En la *Tabla 2* se puede apreciar el comportamiento obtenido al interactuar con la ventana sub menú de minijuegos.

Ventana: Submenú Minijuegos			
Acción	**Resultado esperado**	**Resultado**	**Porcentaje de cumplimiento**
Presionar opción rompecabezas	Acceder la ventana: instrucciones para el rompecabezas	Correcto	100%
Presionar opción banda de comida	Acceder la ventana: instrucciones para la banda de comida	Correcto	100%
Presionar opción atrás	Regresar al menú principal	Correcto	100%

Tabla 2: Verificación del sub menú videojuegos

En la *Tabla 3* se puede apreciar el comportamiento obtenido al interactuar con la ventana instructiva al minijuego de rompecabezas.

Ventana: Instructivo rompecabezas			
Acción	**Resultado esperado**	**Resultado**	**Porcentaje de cumplimiento**
Presionar botón comenzar	Acceder a la escena: minijuego rompecabezas	Correcto	100%
Presionar botón aquí	Acceder a la escena: glosario	Correcto	100%
Presionar botón	Regresar al submenú de	Correcto	100%

regresar	minijuegos		

Tabla 3: Verificación del instructivo para el minijuego del rompecabezas

En la *Tabla 4* se puede apreciar el comportamiento obtenido al interactuar con las opciones de la ventana instructiva al minijuego de banda de comida.

Ventana: Instructivo banda de comida			
Acción	**Resultado esperado**	**Resultado**	**Porcentaje de cumplimiento**
Presionar botón comenzar	Acceder a la escena: minijuego banda de comida	Correcto	100%
Presionar botón aquí	Acceder a la escena: glosario	Correcto	100%
Presionar botón regresar	Regresar al submenú de minijuegos	Correcto	100%

Tabla 4: Verificación del instructivo para el minijuego de la banda de comida

En la *Tabla 5* se puede apreciar el comportamiento obtenido al interactuar con la escena de glosario.

Escena: Glosario			
Acción	**Resultado esperado**	**Resultado**	**Porcentaje de cumplimiento**

Presionar botón regresar	Regresar a la escena: menú principal	Correcto	100%
Arrastrar barra de desplazamiento	Desplazarse por el contenido de la escena	Correcto	100%

Tabla 5: Verificación del glosario

En la *Tabla 6* se puede apreciar el comportamiento obtenido al interactuar con la escena de rompecabezas.

Escena: Minijuego de rompecabezas			
Acción	**Resultado esperado**	**Resultado**	**Porcentaje de cumplimiento**
Presionar Esc/Escape	Mostrar la ventana: menú pausa. El minijuego se pausa	Correcto	100%
Agarrar un alimento con el cursor del mouse	El alimento sigue el movimiento del mouse	Correcto	100%
Soltar el alimento	El alimento deja de seguir el movimiento del mouse y mantiene su posición	Correcto	100%
Soltar el alimento en un	En alimento mantiene su posición e incrementa la	Correcto	100%

grupo correcto	puntuación. Un nuevo alimento aparece en la escena		
Soltar el alimento en un grupo incorrecto	En alimento regresa a su posición original y la puntuación no es afectada	Correcto	100%
Contador llega a cero	Acceder a escenario: puntuación final	Correcto	100%

Tabla 7: Verificación del minijuego de rompecabezas

En la *Tabla 8* se puede apreciar el comportamiento obtenido al interactuar con la escena de banda de comida.

Escena: Minijuego de banda de comida			
Acción	**Resultado esperado**	**Resultado**	**Porcentaje de cumplimiento**
Presionar Esc/Escape	Mostrar la ventana: menú pausa. El minijuego se pausa	Correcto	100%
Agarrar un alimento con el cursor del mouse	El alimento sigue el movimiento del mouse	Correcto	100%
Soltar el alimento	El alimento deja de seguir el movimiento del mouse y cae hasta tocar	Correcto	100%

	la banda transportadora		
Soltar el alimento en un grupo correcto	En alimento desaparece e incrementa la puntuación. Se muestra una animación con un gesto positivo	Correcto	100%
Soltar el alimento en un grupo incorrecto	En alimento desaparece y la puntuación no es afectada. Se muestra una animación con un gesto negativo	Correcto	100%
Contador llega a cero	Acceder a escenario: puntuación final	Correcto	100%

Tabla 8: Verificación del minijuego de la banda de comida

En la *Tabla 9* se puede apreciar el comportamiento obtenido al interactuar con la ventana del menú de pausa.

Ventana: Menú de pausa			
Acción	**Resultado esperado**	**Resultado**	**Porcentaje de cumplimiento**
Presionar Esc/Escape	Ocultar la ventana: menú pausa. El minijuego se reanuda	Correcto	100%
Presionar opción	Ocultar la ventana: menú	Correcto	100%

reanudar	pausa. El minijuego se reanuda		
Presionar opción salir	El juego se reinicia y se retorna a la escena: menú principal	Correcto	100%

Tabla 9: Verificación de la ventana menú de pausa

En la *Tabla 10* se puede apreciar el comportamiento obtenido al interactuar con la escena de puntuación final.

Escena: Puntuación final			
Acción	**Resultado esperado**	**Resultado**	**Porcentaje de cumplimiento**
Presionar opción regresar al menú principal	Regresar a la escena: menú principal	Correcto	100%

Tabla 10: Verificación de puntuación final

5. Resultados

El primer resultado fue la presentación de los diseños que conformarían la aplicación como los alimentos, como se aprecia en la *Figura 29*. Además del diseño final mascota de la aplicación, como puede apreciarse en la *Figura 30*.

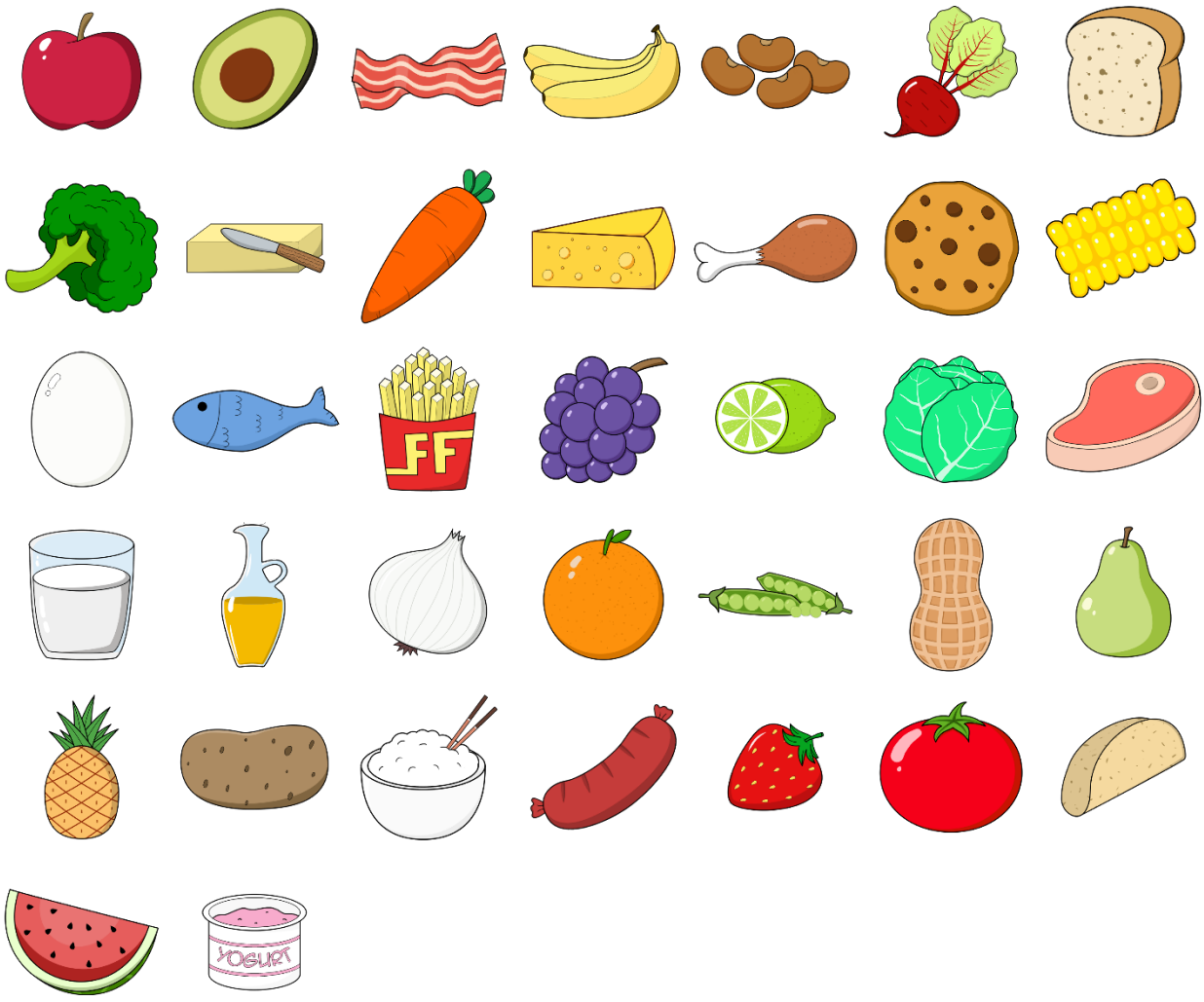

Figura 29: Sprites de Alimentos para la aplicación

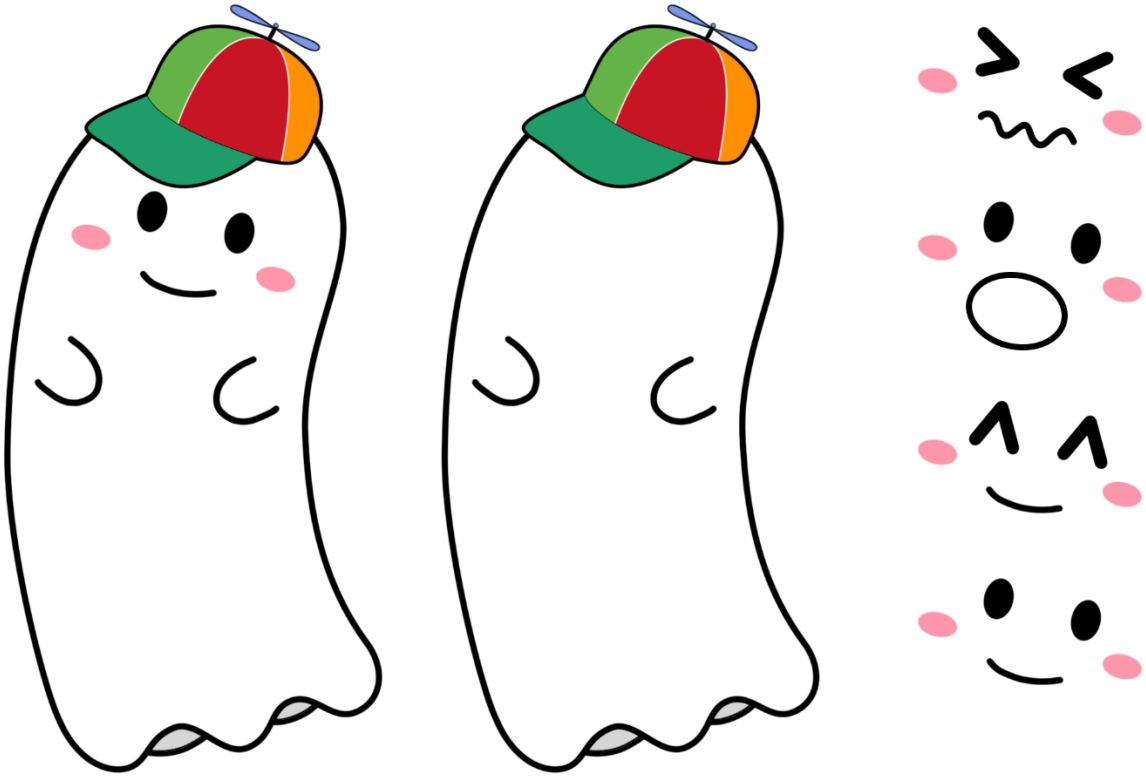

Figura 30: Diseño final de la mascota de la aplicacion

Como segundo resultado se diseñaron los escenarios del menú principal y de los minijuegos, los cuales pasarían a formar parte de un "atlas de sprites" (conjunto de sprites en una sola imagen para posteriormente separarlos en sprites individuales), que se utilizaran en el proyecto.

En la *Figura 31* puede apreciarse el atlas de sprites del menú principal. En la *Figura 32* y *Figura 33* pueden apreciarse los atlas de sprites de cada minijuego.

Figura 31: Atas de sprites para el menu principal

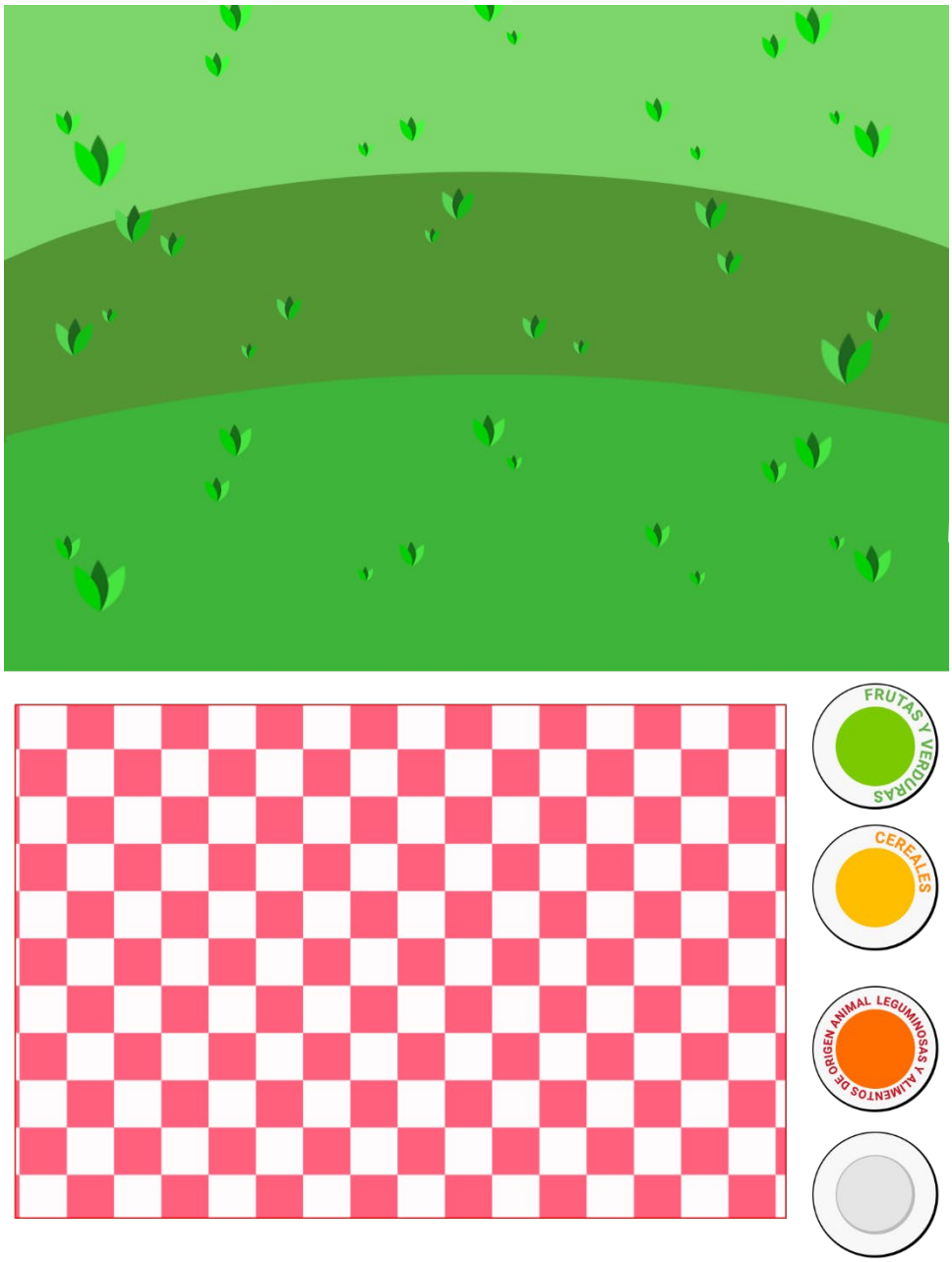

Figura 32: Atas de sprites para el minijuego de rompecabezas

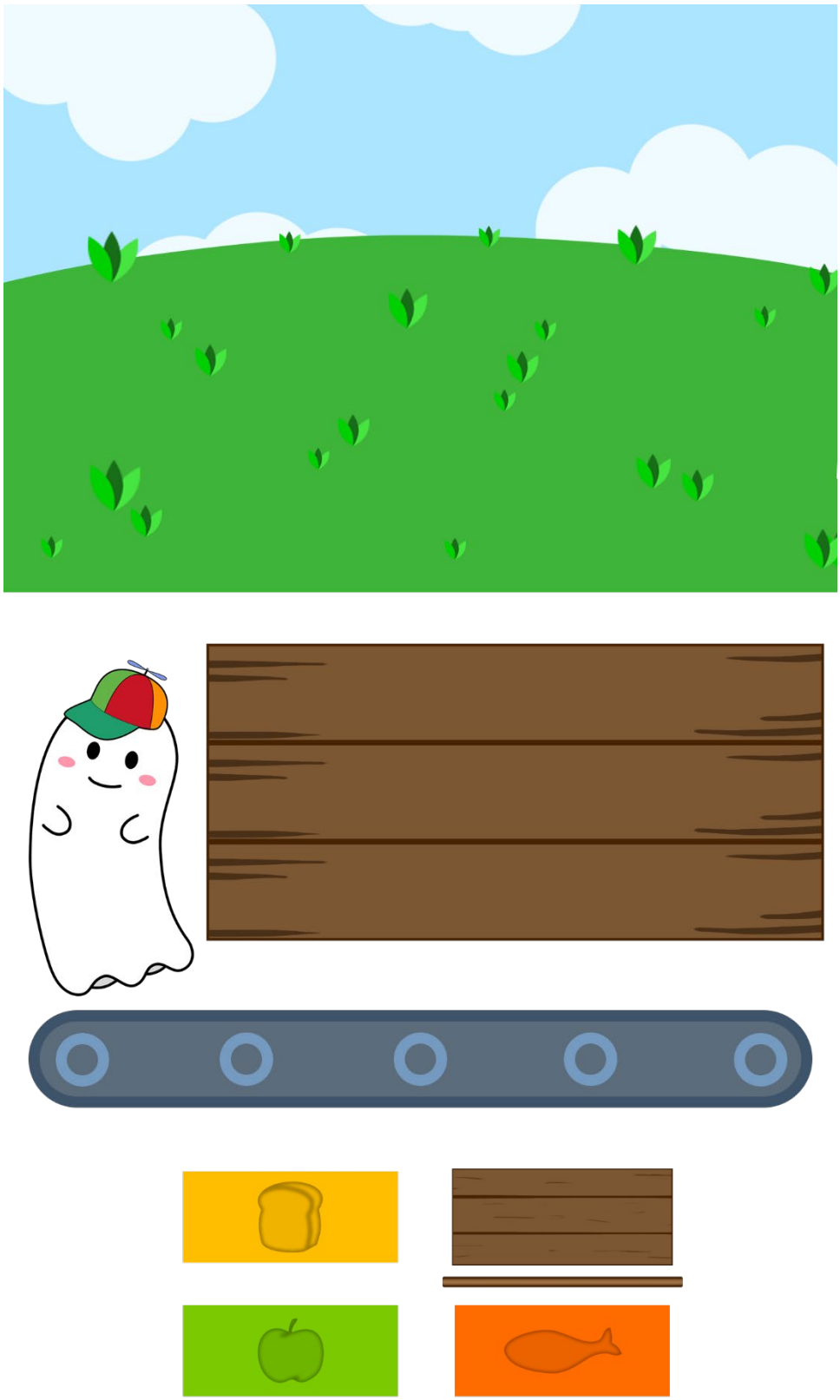

Figura 33: Atlas de sprites para el minijeugo de la banda de comida

Posteriormente se optó por utilizar uno de los sprites para complementar la escena del glosario como se aprecia en la *Figura 34.*

Figura 34: Ventana principal del glosario de la aplicación

El tercer resultado fue el desarrollo del rompecabezas del plato del buen comer, el cual como se muestra en la *Figura 35*, cumple con las expectativas de un juego con estilo infantil manteniendo así el objetivo del proyecto.

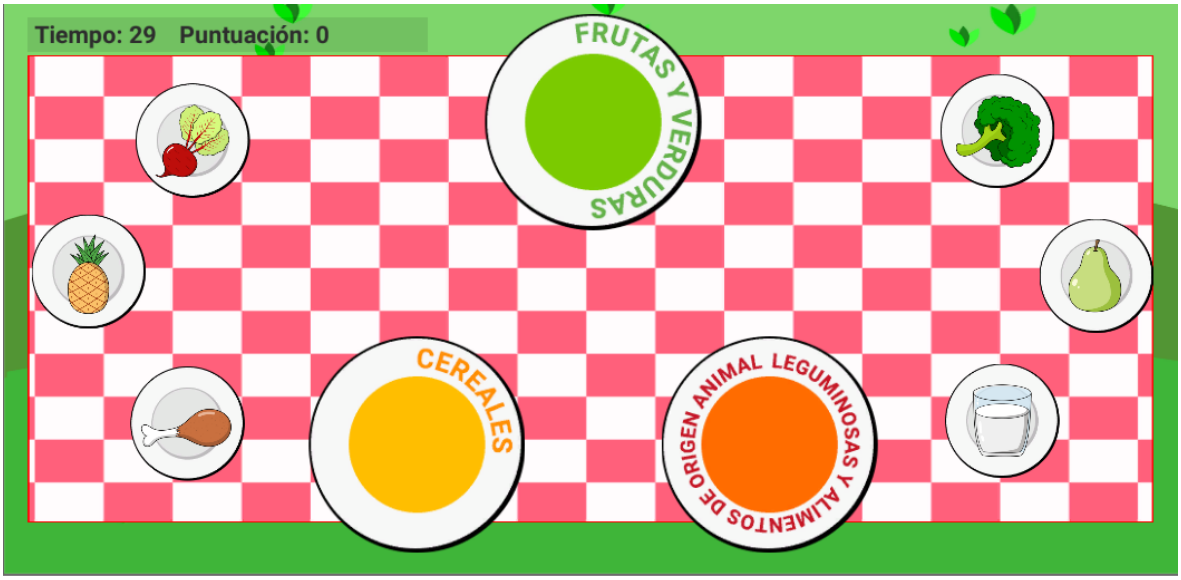

Figura 35: pantalla principal del rompecabezas del plato del buen comer

Como era de esperarse el diseño alternativo al plato del buen comer original resolvió uno de los primeros problemas de jugabilidad en el cual era muy fácil equivocarse. Posteriormente a los cambios el juego se ejecutaba como era debido como puede apreciarse en la *Figura 36* a la vez que se muestra el tiempo a contra reloj y el score correspondiente a los aciertos *Figura 37*.

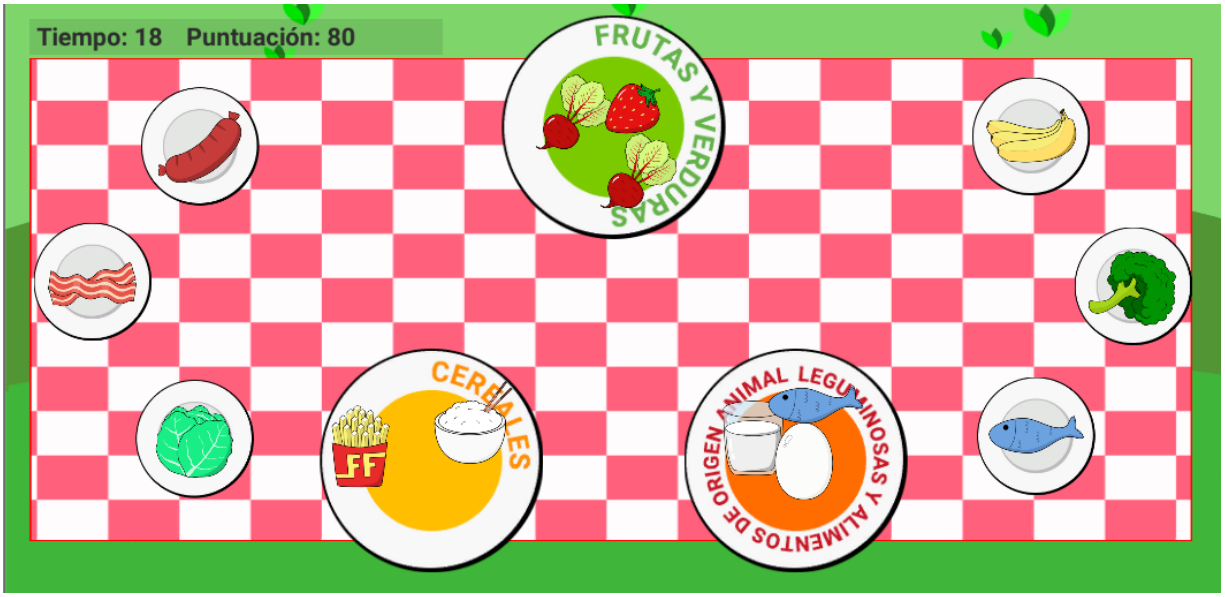

Figura 36: ejecución del rompecabezas del plato del buen comer

Figura 37: Tiempo restante y Score

El cuarto resultado fue el desarrollo de la banda de comida, el cual cumplió su objetivo como un videojuego distinto al rompecabezas pero que sirviera como complemento para el aprendizaje del niño.

En la *Figura 38* puede apreciarse la ejecución de la pantalla principal del minijuego de la banda de comida.

Figura 38: Pantalla principal del la banda de comida

El quinto resultado fue el desarrollo de la interfaz principal de la aplicación, como puede apreciarse en la *Figura 39*. Esta interfaz les dará continuidad al submenú que muestra las opciones para seleccionar minijuegos y al glosario como se muestra en la *Figura 40*.

Figura 39: Pantalla principal de la aplicación

Figura 40: Submenu Minijuegos

El sexto resultado fue el desarrollo de interfaces introductorias que le darán instrucciones al niño sobre como jugar cada minijuego.

En la *Figura 41* puede apreciarse la interfaz introductoria al minijuego del rompecabezas.

Figura 41: Pantalla instructiva al minijuego rompecabezas del plato del buen comer

En la *Figura 42* puede apreciarse la interfaz introductoria al minijuego de la banda de comida.

Figura 42: Pantalla instructiva al minijuego de banda de comida

El séptimo resultado fue el desarrollo una interfaz de pausa para el juego como se aprecia en la *Figura 43*, además del desarrollo de la escena que muestra la puntuación final al finalizar una partida como se aprecia en la *Figura 44*.

Figura 43: Interfaz de pausa

Figura 44: Escena de puntuación

6. Conclusiones

Debo decir que trabajar en este proyecto me permitió experimentar de primera mano lo que es trabajar en un ambiente laboral y estoy agradecido de poder formar parte de esta experiencia que me ayudó a mejorar como desarrollador.

Al realizar mi papel como diseñador me ayudó a pulir mis habilidades como artista ya que comencé en el dibujo como un hobby, pero al emplear mis habilidades en un proyecto profesional de verdad me ayudó a impulsar mi gusto por este hobby, que ahora, busco hacerlo algo más en el futuro.

Como todo amante de los videojuegos, desde joven el trabajar como desarrollador de videojuegos es una fantasía que si bien parecía algo fuera de mi alcance siempre fue un sueño que tuve durante muchos años, pero gracias a proyectos pasados y mi participación en este proyecto me han ayudado a tomar este sueño enserio; y me da inspiración de continuar con mis metas y pasiones.

Si no lo he repetido ya muchas veces, quiero agradecer una vez mas a todas aquellas personas que compartieron una parte de mi vida en todo este camino, a pesar de que inesperadamente el mundo se detuvo por la pandemia y nos diera problemas a todos, en especial a la organización de nuestro equipo estoy feliz de haber trabajado con él, ya que hizo posible que el proyecto pudiera continuar y estoy feliz con los resultados tras su finalización.

7. Competencias desarrolladas

En este capítulo se muestran las competencias y habilidades obtenidas y aplicadas durante el desarrollo del proyecto.

Competencias Instrumentales

Son las competencias que funcionan como instrumentos para lograr algún objetivo. Las siguientes competencias instrumentales fueron las necesarias para que el residente realizara el proyecto.

- Habilidades de gestión de información, utilizada con el objetivo de controlar, almacenar y recuperar información de forma eficaz y productiva, proporcionado una mayor seguridad para la toma de decisiones.
- Solución de problemas, ya que en cualquier proyecto los problemas surgirán, aunque se haya hecho el análisis, la planeación y la modelación de manera correcta. Dicha habilidad fue necesaria para tener la capacidad de analizar y solucionar problemas de manera eficaz.
- Toma de decisiones, siendo esta habilidad la que permite elegir la mejor entre varias opciones, haciendo uso de diferentes técnicas para ello.

Competencias Interpersonales

Son las competencias que incluyen las competencias para el trabajo en equipo y las relativas al compromiso con el trabajo.

- Trabajo en equipo, ya que, por la naturaleza del proyecto, el software se dividió en módulos, siendo necesario el trabajo en equipo, desarrollando la habilidad de tratar con personas para lograr una serie de metas específicas, haciendo que la buena comunicación entre los integrantes del equipo y la toma de decisiones grupales algo de vital importancia.
- Habilidad de trabajar en un ambiente laboral: la realización de este proyecto proporcionó una experiencia muy cercana a la laboral por lo cual fue necesario desarrollar esta habilidad, permitiendo una adaptación más natural en futuros proyectos.
- Compromiso ético: esta habilidad fue utilizada para respetar los compromisos con la asesora interna y los trabajos realizados por otras personas.

Competencias Sistemáticas

Estas hacen referencia a la integración de capacidades cognitivas, destrezas prácticas y disposiciones.

- Habilidades de investigación: esta habilidad permitió una forma más eficaz de obtener, verificar y utilizar información.
- Capacidad de generar nuevas ideas: esta habilidad permitió generar las ideas necesarias para desarrollar el proyecto, diseñar su funcionalidad, interfaz gráfica y la solución de problemas.

Competencias por asignaturas

- Programación Orientada a Objetos: Paradigma de programación que utiliza objetos para la manipulación de entrada de datos para la obtención de datos de salida específicos.

- Fundamentos de Ingeniería de Software: base teórica para la definición de requerimientos del sistema.
- Gestión de proyectos de software: base teórica del ciclo de desarrollo de un software.
- Ingeniería de Software: correcta ejecución del desarrollo del software, así como la implementación de pruebas y trato con el cliente.
- Graficación: bases y principios de diseño con vectorización, transformación de pixeles, diseño de interfaces y animación.

Tecnologías utilizadas

El siguiente listado son todas las tecnologías que se necesitaron para poder desarrollar el software para apoyar al proyecto.

- Unity Engine: Utilizado como motor principal para el desarrollo de la aplicación.
- C#: Utilizado para manejar eventos de la aplicación.
- Photoshop: Utilizado para diseñar sprites e interfaces de la aplicación.
- Git/ Bitbucket: Para administrar el versionamiento de código del proyecto.

8. Fuentes de información

Adobe. (s.f.). *Guía del usuario de Photoshop*. Recuperado el 10 de Marzo de 2020, de Adobe: https://helpx.adobe.com/mx/photoshop/user-guide.html

Instituto Tecnologico de Ciudad Guzmán. (s.f.). *Antecedentes Históricos*. Recuperado el 10 de Marzo de 2020, de itcg: http://www.itcg.edu.mx/index.php?opc=historia

Proyectos Agiles. (s.f.). *Cómo Funciona Scrum*. Recuperado el 10 de Marzo de 2020, de Proyectos Agiles: https://proyectosagiles.org/como-funciona-scrum/

Scrum. (10 de Marzo de 2020). *What is Scrum?* Obtenido de Scrum.org: https://www.scrum.org/resources/what-is-scrum

UNITY. (s.f.). *Unity Manual*. Recuperado el 10 de Marzo de 2020, de Unity3D: https://docs.unity3d.com/Manual/UnityManual.html

Wikipedia. (s.f.). *Adobe Photoshop*. Recuperado el 10 de Marzo de 2020, de Wikipedia: https://es.wikipedia.org/wiki/Adobe_Photoshop

Wikipedia. (s.f.). *Unity (motor de videojuego)*. Recuperado el 10 de Marzo de 2020, de Wikipedia: https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)