UNIVERSIDAD AUTÓNOMA "GABRIEL RENÉ MORENO"

FACULTAD DE INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN Y TELECOMUNICACIONES

JUEGO ARCADE: "FLAPPY BIRD"

PROYECTO : #36

ÁREA – CATEGORÍA: PROGRAMACIÓN – BÁSICO

MATERIA: PROGRAMACIÓN I

DOCENTE: ING. ALBERTO MOLLO MAMANI

INTEGRANTES:	**CARRERA:**
EDDY MARCELO TOLEDO QUIROGA	-ING. SISTEMAS
RODRIGO LLANOS VINAYA	-ING. SISTEMAS

SANTA CRUZ – BOLIVIA

1 – 2024

ÍNDICE

1. INTRODUCCIÓN

Los juegos de arcade son juegos electrónicos que se originaron en salones recreativos (conocidos popularmente como arcades) durante las décadas de 1970 y 1980.

Estos juegos se caracterizan por su enfoque en la acción rápida, una jugabilidad sencilla y la competencia por puntuaciones altas.

Generalmente este tipo de juegos tienen controles muy simples que los hacen accesibles para una amplia audiencia de jugadores. Son conocidos por su capacidad para brindar una diversión rápida y emocionante en sesiones de juego cortas.

“Flappy Bird”, es un videojuego de este estilo. Consiste en controlar un pequeño ave evitando obstáculos para llegar a marcar una puntuación más alta con el tiempo.

El objetivo al desarrollar "Flappy Bird" es mostrar su diseño y mecánicas de juego, proporcionando un modelo para aquellos que deseen entender cómo está diseñado y estructurado un juego de este tipo. Con este proyecto queremos dar un ejemplo de cómo lograr el desarrollo un videojuego simple pero divertido y adictivo.

Para este proyecto, utilizaremos herramientas de desarrollo como el lenguaje de programación C#, un entorno de desarrollo integrado (IDE) como Visual Studio, y Unity, una plataforma avanzada para el desarrollo de juegos. Unity nos permitirá implementar funciones, características, gráficos y diseños de niveles de manera eficiente y cómoda.

2. CONTEXTO

Las máquinas arcade surgieron en los años 70 y 80 y aún hasta el día de hoy siguen siendo muy populares alrededor del mundo.

Estás máquinas fueron pioneras en el uso de los gráficos y sonidos digitales que marcaron el inicio de la industria de los videojuegos.

Las primeras máquinas arcade eran simples juegos de mesa electrónicos, que se introdujeron en la década de 1960. Eran juegos muy básicos y se limitaban en la reproducción de sonidos y luces. No fue hasta la década de 1970 que las máquinas arcade comenzaron a evolucionar y a incluir juegos más sofisticados.

En 1971, Atari lanzó el primer juego arcade exitoso, “Pong”, que rápidamente se convirtió en un fenómeno mundial. Era un juego simple de tenis en el cual los jugadores debían de golpear una pelota de un lado a otro de la pantalla.

Con el éxito rotundo de este primer juego, la industria del videojuego comenzó a despegar, dando lugar a la evolución de los videojuegos arcade.

A medida que la tecnología avanzaba, los juegos de arcade se volvieron más avanzados e incluían sonidos más realistas. En 1980, “Pac-Man”, se convirtió en el juego arcade más exitoso de todos los tiempos y se volvió un ícono de los videojuegos.

En los años 80, los juegos arcade se volvieron aún más populares y la industria creció rápidamente. Muchos juegos de arcade se convirtieron en franquicias exitosas y se crearon versiones para consolas de videojuegos y ordenadores personales.

Fue así que la comunidad de juegos arcade se hizo muy popular durante varios años, abriendo el mercado de juegos de este estilo en otras plataformas, hasta el punto de llegar a ser desarrollados incluso para móviles.

La llegada de “Flappy Bird” al mercado de los juegos arcade.

A finales de 2012, un desarrollador vietnamita, Dong Nguyen, mostraba al mundo por primera vez, una imagen de un juego muy llamativo, al que primeramente llamó “Flap Flap”, mediante la red social Twitter.

En 24 de mayo de 2013, lanzaba al mercado de móviles un juego simple y difícil llamado “Flappy Bird”.

La mecánica del mismo era simple, un pájaro que debíamos controlar tocando el móvil para que éste no chocara con unas tuberías presentes a lo largo del escenario del juego.

Durante sus primeros meses tuvo una relevancia moderada hasta que con el pasar de los meses, el juego fue adquiriendo mayor relevancia, logrando posicionarse en el puesto 1469 en la categoría de juegos familiares en EEUU.

El juego fue popularizándose a lo largo de los siguientes meses, hasta que en enero de 2014 llegó a convertirse en el juego número 1 de la AppStore en EEUU y fue adquiriendo más popularidad en alrededor del mundo.

El juego fue conocido por su "dificultad demoníaca", pero en lugar de desanimar a los jugadores, logró que el juego llegue a convertirse en un ícono de la industria de los juegos móviles, dejando un legado que perdura hasta el día de hoy.

3. DESCRIPCIÓN

Flappy Bird es un juego donde los jugadores controlan un pequeño pájaro que debe volar a través de un curso de obstáculos intercalados aleatoriamente a lo largo de todo el escenario del juego.

El juego consiste en volar lo más lejos posible sin chocar con los obstáculos. El pájaro vuela automáticamente hacia adelante, y los jugadores pueden hacer que el pájaro salte con el “Click Izquierdo del ratón”, lo que le permite esquivar los obstáculos y ganar puntos por cada obstáculo esquivado.

El objetivo principal es obtener la mayor puntuación posible antes de chocar y terminar el juego.

Los componentes más importantes que usaremos para desarrollar este juego son los vectores y puntos. Estos nos permitirán calcular de forma precisa la posición y velocidad del pájaro, así como la posición de obstáculos en el juego.

De esta manera, podremos gestionar adecuadamente el diseño del juego e interpretar mejor las colisiones entre el pájaro y los obstáculos.

Usaremos funciones para generar aleatoriamente los obstáculos en el juego, actualizar el estado del juego y manejar la lógica del propio juego.

Para cumplir con nuestro objetivo, utilizaremos el entorno de desarrollo integrado (IDE) de Visual Studio, que nos permitirá organizar una estructura sólida para el juego. Implementaremos clases para dividir el código en módulos o componentes fácilmente modificables.

Adicionalmente, utilizaremos Unity, una reconocida plataforma de desarrollo de juegos, que nos permitirá crear un diseño atractivo para 'Flappy Bird'.

Unity ofrece una amplia gama de herramientas y recursos que simplifican la implementación de mecánicas de juego, la gestión de gráficos y el diseño del juego**.**

4. FORMULACIÓN DE OBJETIVOS

4.1. OBJETIVO GENERAL

Desarrollar un juego arcade inspirado en el clásico 'Flappy Bird' utilizando Unity y C#, con la incorporación de nuevos niveles y gráficos mejorados para actualizar la experiencia de juego.

4.2. OBJETIVOS ESPECÍFICOS

- Diseñar una interfaz de usuario intuitiva que permita al jugador acceder fácilmente a todas las funciones principales del juego, permitiéndole elegir entre distintos niveles, ver su puntaje más alto y navegar por el menú del juego.
- Implementar la estructura vectorial para gestionar las posiciones de los elementos del juego, como el pájaro, los tubos y otros objetos adicionales presentes en el diseño del juego.

- Implementar algoritmos que detecten el choque entre el pájaro y algún obstáculo para la finalización de la sesión de juego.
- Implementar estructuras de archivos para guardar y cargar datos de sesiones de juego anteriores, dando a conocer el puntaje más alto alcanzado hasta ese momento.
- Utilizar archivos para gestionar el diseño del juego, de modo que mediante el uso de archivos consigamos implementar texturas, efectos de sonido e imágenes para una mejor experiencia al momento de jugar.
- Integrar Unity como la plataforma principal de desarrollo, para aprovechar sus características avanzadas en la creación de gráficos, física y gestión de recursos del juego.

5. DESARROLLO DEL PROYECTO

5.1. USO DE ESCENAS (EDITOR DE UNITY)

Implementamos el uso de escenas en Unity, para estructurar y gestionar los diferentes niveles del juego y el menú principal.

- **Escenas “Nivel X”**

Cada nivel está representado por una escena dedicada denominada "Nivel" y el número de nivel correspondiente.

Cada escena contiene los elementos del nivel del juego, cómo el fondo del escenario, los obstáculos de cada nivel y el diseño del mismo.

Además, cada una de las escenas está configurada para invocar las funciones y clases definidas en el código de Visual Studio. Estas funciones manejan los aspectos del juego, como la inicialización de cada partida y la interacción con los elementos del nivel.

- **Escena “Menu”**

Con el fin de mejorar la experiencia del usuario, hemos implementado una escena dedicada a la interfaz del juego. Esta incluye un menú principal que permite a los jugadores iniciar sesiones de juego, seleccionar niveles, salir del juego y realizar otras acciones.

Las características principales de la escena “Interfaz” son las siguientes:

- **Imagen de Fondo:** Visualiza el escenario del juego.
- **Título del Juego:** Muestra el título "Flappy Bird" en la interfaz.
- **Botón "Play":** Oculta la escena de la interfaz y muestra la escena principal del juego (Escena “Game”), permitiendo al jugador iniciar una sesión de juego.
- **Botón “Niveles”:** Permite ver y elegir el nivel a jugar.
- **Botón “Volver”:** Se visualiza en la sección 'Niveles' y permite al jugador regresar al menú principal.
- **Botón “Reinicio”:** Se muestra luego de la colisión del pájaro con un obstáculo. Permite al jugador comenzar una nueva sesión de juego sin necesidad de volver al menú principal.
- **Botón “Salir del Juego”:** Cierra la aplicación y termina la ejecución del juego, regresando al usuario al escritorio de Windows.

- **Escena “MenuNiveles”**

Muestra los niveles del juego a través del menú principal y está integrada a la escena “Menu”.

Permite al jugador seleccionar entre los diferentes niveles del juego antes de iniciar una partida.

5.2. IMPLEMENTACIÓN DE SCRIPTS (C#)

Se implementaron scripts en C# para manejar el código del juego y controlar las funcionalidades del mismo, cada uno con sus definiciones y respectivas características:

- **Instancias de Clases:** Se definen varias clases para el desarrollo ordenado del juego. Implementamos las clases Pájaro, Obstaculo, Generador, Puntaje, AreaPuntaje, Empezar, ControladorEscena.
- **Declaración de Variables:** Variables para controlar los parámetros de cada elemento del juego, como el puntaje del jugador, las propiedades de los objetos (pájaro y obstáculos), entre otros aspectos necesarios para las mecánicas del juego.
  Definimos las variables en cada clase, según la lógica que desarrollamos en cada una.
- **Métodos Implementados**: Hacemos uso de métodos para desplazar el pájaro y los obstáculos, manejamos eventos que permitan dar fin al juego al colisionar con un obstáculo o para iniciar una sesión de juego.
- **Gestión de Archivos:** Utilizamos métodos de Archivos para dos funciones principales del juego:
  - Guardar y cargar el progreso del jugador, permitiendo al jugador conocer su puntuación más alta alcanzada en sus sesiones de juego.
  - Gestionar los elementos de diseño del juego, como las imágenes para el escenario, el diseño de obstáculos, niveles y del pájaro. Como también los sonidos y efectos de sonido (SFX).

5.3. IMPLEMENTACIÓN DE CLASES

- **Clase PÁJARO:** La clase ‘’ pájaro ‘’ representa la entidad del pájaro en el juego. Contiene propiedades de estado, como la altura y la gravedad.
  Esta clase se utiliza para controlar el movimiento del pájaro en el juego
- **Clase OBSTÁCULO:** La clase “obstáculo” maneja el movimiento de los obstáculos a lo largo del escenario del juego. Calcula el desplazamiento de los obstáculos a través de un algoritmo de posicionamiento.

- **Clase GENERADOR:** La clase “generador” se encarga de generar obstáculos por todo el escenario del juego.
- **Clase PUNTAJE:** Esta clase maneja las puntuaciones del jugador, las almacena y las muestra en pantalla. Registra la puntuación actual de la sesión de juego y la puntuación más alta alcanzada, y las muestra en pantalla.
- **Clase AREAPUNTAJE:** La clase “AreaPuntaje” establece y verifica el área del juego donde se incrementa el puntaje al pasar por un obstáculo, es de vital importancia para asegurar que el puntaje se incremente solo en áreas correctas y evitar puntos marcados incorrectamente.
- **Clase EMPEZAR:** Este método se invoca para iniciar una sesión de juego. Se encarga de ocultar los elementos de la interfaz para comenzar una sesión de juego.
- **Clase CONTROLADORESCENA:** Maneja la lógica principal del juego y la interfaz, como el inicio, reinicio del juego, ajustes, y las demás funciones. Es la clase responsable de coordinar la interfaz y la lógica del juego.

6. MARCO TEÓRICO

6.1. PROGRAMACIÓN ORIENTADA A OBJETOS

La Programación Orientada a Objetos (POO) es un paradigma de programación, es decir, un modelo o un estilo de programación que nos da unas guías sobre cómo trabajar con él. Se basa en el concepto de clases y objetos. Este tipo de programación se utiliza para estructurar un programa de software en piezas simples y reutilizables de planos de código (clases) para crear instancias individuales de objetos.

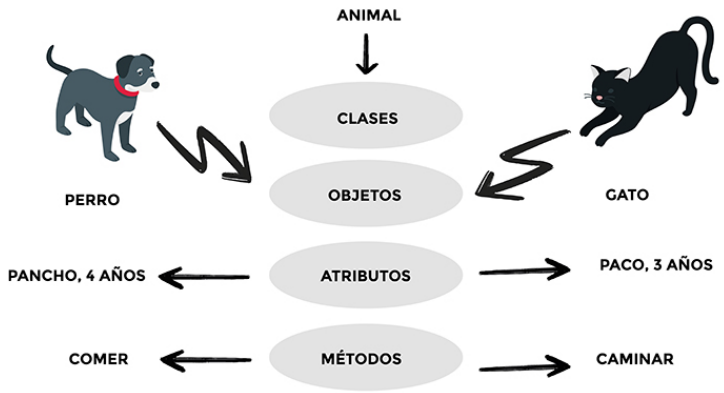
Con el paradigma de Programación Orientado a Objetos lo que buscamos es dejar de centrarnos en la lógica pura de los programas, para empezar a pensar en objetos, lo que constituye la base de este paradigma. Esto nos ayuda muchísimo en sistemas grandes, ya que, en vez de pensar en funciones, pensamos en las relaciones o interacciones de los diferentes componentes del sistema.

Un programador diseña un programa de software organizando piezas de información y comportamientos relacionados en una plantilla llamada clase. Luego, se crean objetos individuales a partir de la plantilla de clase. Todo el programa de software se ejecuta

haciendo que varios objetos interactúen entre sí para crear un programa más grande. (Martínez M., 2020)

- **Clase, Objetos e Instancias**

  Una clase es una plantilla. Define de manera genérica cómo van a ser los objetos de un determinado tipo. Por ejemplo, una clase para representar a animales puede llamarse ‘animal’ y tener una serie de atributos, como ‘nombre’ o ‘edad’ (que normalmente son propiedades), y una serie con los comportamientos que estos pueden tener, como caminar o comer, y que a su vez se implementan como métodos de la clase (funciones).



Con una clase se pueden crear instancias de un objeto, cada uno de ellos con sus atributos definidos de forma independiente. Con esto podríamos crear un “gato” llamado Paco, con 3 años de edad, y otro animal, este tipo “perro” y llamado Pancho, con una de edad de 4 años. Los dos están definidos por la clase animal, pero son dos instancias distintas. Por lo tanto, llamar a sus métodos puede tener resultados diferentes. Los dos comparten la lógica, pero cada uno tiene su estado de forma independiente.

- **Principios de la Programación Orientada a Objetos**

  - **Encapsulación**

La encapsulación contiene toda la información importante de un objeto dentro del mismo y solo expone la información seleccionada al mundo exterior.

Esta propiedad permite asegurar que la información de un objeto esté oculta para el mundo exterior, agrupando en una Clase las características o atributos que cuentan con un acceso privado**, y los comportamientos o métodos que presentan un acceso público.**

- **Abstracción**

La abstracción es cuando el usuario interactúa solo con los atributos y métodos seleccionados de un objeto, utilizando herramientas simplificadas de alto nivel para acceder a un objeto complejo.

- **Herencia**

La herencia define relaciones jerárquicas entre clases, de forma que atributos y métodos comunes puedan ser reutilizados. Las clases principales extienden atributos y comportamientos a las clases secundarias. A través de la definición en una clase de los atributos y comportamientos básicos, se pueden crear clases secundarias, ampliando así la funcionalidad de la clase principal y agregando atributos y comportamientos adicionales.

- **Polimorfismo**

El polimorfismo consiste en diseñar objetos para compartir comportamientos, lo que nos permite procesar objetos de diferentes maneras. Es la capacidad de presentar la misma interfaz para diferentes formas subyacentes o tipos de datos.

- **Beneficios de la Programación Orientada a Objetos**

- **Reutilización** del código.
- Convierte cosas complejas en **estructuras simples reproducibles**.
- Evita la **duplicación de código**.
- Permite **trabajar en equipo** gracias al encapsulamiento ya que minimiza la posibilidad de duplicar funciones cuando varias personas trabajan sobre un mismo objeto al mismo tiempo.
- Al estar la clase bien estructurada permite la **corrección de errores** en varios lugares del código.
- **Protege la información** a través de la encapsulación, ya que solo se puede acceder a los datos del objeto a través de propiedades y métodos privados.
- La abstracción nos permite **construir sistemas más complejos** y de una forma más sencilla y organizada.

- **Implementación de POO en C#**
    - **Definición de clases:** Utiliza la palabra clave «class» para definir clases y especificar sus atributos y métodos.
    - **Herencia:** Utiliza los operadores de acceso «public», «private» y «protected» para establecer la visibilidad de los miembros heredados.
    - **Interfaces:** Define contratos que las clases deben implementar mediante el uso de interfaces.
    - **Polimorfismo:** Utiliza la palabra clave virtual para métodos en la clase base y «override» en las subclases.
    - **Modificadores de acceso:** Utiliza «public», «private» y «protected» para lograr encapsulación y controlar el acceso a los miembros de una clase.

6.1.1.VECTORES

Los vectores son estructuras de datos similares a los arreglos, pero más desarrollados, ya que, entre otras cosas, crecen y decrecen dinámicamente, según se necesite. Un vector almacena objetos. La sintaxis y forma de acceder es exactamente igual a la de vectores de datos simples. También utilizaremos la clase Vector la forma de declararlo es: vector nombreVariable: los objetos que forma el arreglo pertenecen a la clase Pasajero, definida previamente.

Estructura de un Vector – Ilustración

									Elemento ▼
Vector	22	5	45	74	12	8	10	64	14
Posición	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

6.1.2. MÉTODO ARCHIVOS

Es un conjunto de información fuente de datos que se guarda físicamente en un medio de almacenamiento (disco duro) contenido dentro de un computador. Es decir, queda almacenada permanentemente en disco a elección del usuario. (Mollo A.,2016)

6.2. TEORÍA Y DISEÑO DEL JUEGO

6.2.1. ¿CÓMO ESTÁ COMPUESTO UN JUEGO DE ESTILO “ARCADE”?

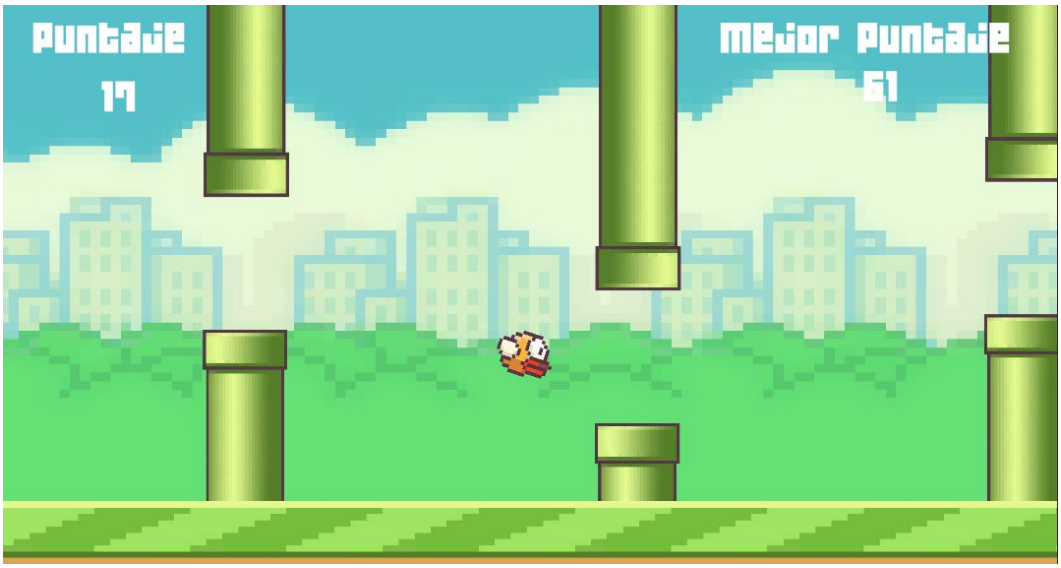
Los juegos arcades son conocidos por su simplicidad, accesibilidad y capacidad de brindar entretenimiento rápido y adictivo. Generalmente este tipo de juegos tienen características clave que incluyen:

- **Simplicidad de Control:** Los juegos de arcade generalmente tienen controles muy simples y directos. En el caso de "Flappy Bird", el control se reduce a una sola acción: tocar o clickear en la pantalla para que el pájaro aletee. Esta simplicidad permite que los jugadores se enfoquen en la jugabilidad en lugar de concentrarse en aprender controles muy complejos.
- **Dificultad Progresiva:** Una característica esencial de los juegos de arcade es el incremento progresivo de la dificultad a medida que el jugador avanza. En 'Flappy Bird', este efecto se logra mediante la disposición de obstáculos en diferentes posiciones a lo largo del escenario del juego. Cada salto requiere precisión y reflejos para esquivar estos obstáculos.
- **Reacción inmediata:** Los juegos de arcade ofrecen una reacción inmediata a las acciones del jugador. En “Flappy Bird”, por ejemplo, cuando el pájaro choca con un obstáculo, el juego finaliza de inmediato, enfatizando la importancia de cada acción y manteniendo el interés de los jugadores.
- **Puntuaciones y Competencia**: La competencia por alcanzar puntuaciones altas es un elemento motivador en muchos juegos de arcade. En “Flappy Bird”, se registra la puntuación más alta del jugador, incentivándolo a superar su propio récord y competir con amigos y otros jugadores.
- **Repetición y Rejugabilidad:** La jugabilidad de los juegos de arcade está diseñada para ser repetitiva pero entretenida. En “Flappy Bird”, cada intento es rápido y el jugador puede reiniciar el juego inmediatamente, lo que proporciona una alta rejugabilidad y diversión continua.

6.2.2. MECÁNICA DEL JUEGO Y DISEÑO DE NIVELES

La mecánica de "Flappy Bird" es muy simple. El jugador controla un pájaro que se desplaza autómaticamente hacia la derecha de la pantalla. Cada click o tecleo del jugador hace que el pájaro aletee y suba ligeramente, mientras que la gravedad lo hace descender cuando no se clickee. El objetivo es evitar chocar con los obstáculos colocados en a través del nivel de juego.

Diseño de Niveles: Cada nivel del juego presenta un diseño sencillo que incluye un fondo atractivo, obstáculos distribuidos a lo largo del escenario y la visualización del puntaje del jugador en pantalla.



1 - Ilustración del Diseño del juego.

Dificultad: La colocación de los obstáculos es aleatoria, lo que demanda mucha precisión y reflejos rápidos por parte del jugador para esquivarlos.

6.3. UNITY Y SU APLICACIÓN EN EL DESARROLLO DE VIDEOJUEGOS

Unity es una plataforma de desarrollo de videojuegos 2D y 3D, creada por Unity Technologies.

Permite el desarrollo de videojuegos para múltiples plataformas (PC, dispositivos móviles y consolas). A pesar de ser utilizado mayormente para el desarrollo de videojuegos,

también ofrece la posibilidad de crear y operar contenido interactivo en 2D y 3D, renderizar imágenes o crear pistas de audios, lo cual es ideal para artistas, arquitectos, diseñadores, cineastas, y demás profesionales.

Unity proporciona un entorno de desarrollo integral que incluye herramientas para diseño gráfico, gestión de física, programación en C#, y muchas más herramientas ideales para el desarrollo 2D y 3D.

6.3.1.BENEFICIOS DE USAR UNITY

Motor gráfico avanzado: Ideal para desarrollar juegos en 2D y 3D con gráficos detallados.

Desarrollo multiplataforma: Facilita el desarrollo del juego para varias plataformas, incluyendo PC, consolas y dispositivos móviles.

Facilidad de Aprendizaje: No se requieren conocimientos avanzados en programación, ya que el margen de aprendizaje es sencillo.

Herramientas de Desarrollo: Unity permite la importación de proyectos desde otras plataformas como Photoshop o Blender, facilitando el desarrollo al utilizar estas herramientas. Además, ofrece integración con Visual Studio para facilitar el desarrollo a los desarrolladores que trabajan con este IDE..

7. CONCLUSIONES Y RECOMENDACIONES

7.1. CONCLUSIONES

- El juego 'Flappy Bird' se desarrolló siguiendo una estructura coherente y eficiente, implementando los principios de Programación Orientada a Objetos (POO). Esto facilitó la organización del código y aseguró un orden en cada aspecto del juego, desde la interfaz de usuario hasta la lógica del juego, mediante la implementación de diversas clases y funciones.
- Hemos logrado realizar nuestra visión de crear un juego entretenido y accesible para los jugadores, ofreciendo un desafío que pone a prueba sus reflejos y concentración.

- Destacamos la simplicidad del juego, señalando que no se requiere un diseño ni mecánicas complejas para lograr un juego entretenido. La simplicidad puede contribuir a su diversión y alta rejugabilidad.
- El desarrollo del juego en Unity utilizando Visual Studio (C#) nos permitió mejorar el diseño y estructura del juego de manera significativa.
- Las pruebas desempeñaron un papel fundamental durante el desarrollo del juego. Permitieron identificar errores y asegurar el correcto funcionamiento del juego, garantizando así una experiencia satisfactoria para los usuarios finales.

7.2. RECOMENDACIONES

Con el objetivo de apoyar a otros usuarios y desarrolladores en el futuro, se presentan las siguientes recomendaciones para mejorar el juego:

Ampliación del juego: Existe un oportunidad considerable para expandir el juego mediante la adición de más niveles, variedad en los obstáculos y posiblemente nuevas mecánicas, manteniendo siempre la simplicidad del juego.

Recompensas adicionales: Para incentivar a los jugadores a alcanzar puntuaciones altas, se recomienda implementar más recompensas por cumplir objetivos dentro del juego. Estas podrían incluir desbloquear skins para el pájaro, nuevos niveles o cambios en la música de los niveles.

Optimización del juego: Debido al uso de Unity como motor principal, es fundamental optimizar el uso de recursos del juego. Se recomienda implementar algoritmos más eficaces para garantizar la compatibilidad con una mayor variedad de dispositivos.

Integración de tablas de puntuaciones en línea: Con el objetivo de fomentar la competencia entre los jugadores, se sugiere la implementación de una tabla de puntuaciones en línea. Esto permitirá a los jugadores comparar sus puntajes con otros jugadores del mundo, motivándolos a alcanzar puntuaciones más altas.

8. BIBLIOGRAFIA

I. *"Historia de los videojuegos arcade" (Blog de "Arcades RetroAL). Obtenido de: https://arcades-retroal.com/historia-de-los-videojuegos-arcades*

II. *Mashable (2014) "La verdadera historia de Flappy Bird, su popularidad y desaparición". Obtenido de: https://iphoneros.com/40162/la-verdadera-historia-de-flappy-bird-su-popularidad-y-desaparicion*

III. *Martínez M. (2020) "¿Qué es la Programación Orientada a Objetos?". Obtenido de: https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/*

IV. *JLPM (2021): "Re: Tutorial COMPLETO Unity 2D desde Cero". Obtenido de: https://youtu.be/GbmRt0wydQU*

V. *Alberto, M. M. (2016). Algoritmos de Programación. Santa Cruz de la Sierra-Bolivia.*

VI. *Martínez, R. (2015). Teoría y práctica del videojuego. McGraw-Hill.*

VII. *Glo Martínez (2023). "¿Qué es Unity y para qué sirve?". Obtenido de: https://ebac.mx/blog/que-es-unity-y-para-que-sirve*

VIII. *Unity Technologies. (2024). Unity - Manual de usuario. Obtenido de https://docs.unity3d.com/Manual/index.html*

IX. *Brave Pixel G (2021): "Re: Cómo crear animaciones 2D en Unity". Obtenido de: https://youtu.be/I2D_nx7MbbQ*

X. *Dev RItch (2024): "Re: Cómo poner música y sonido en Unity". Obtenido de : https://youtu.be/vI992L2gYPM*

9. ANEXOS

Ilustración 2 - Editor de Escenas (Unity)

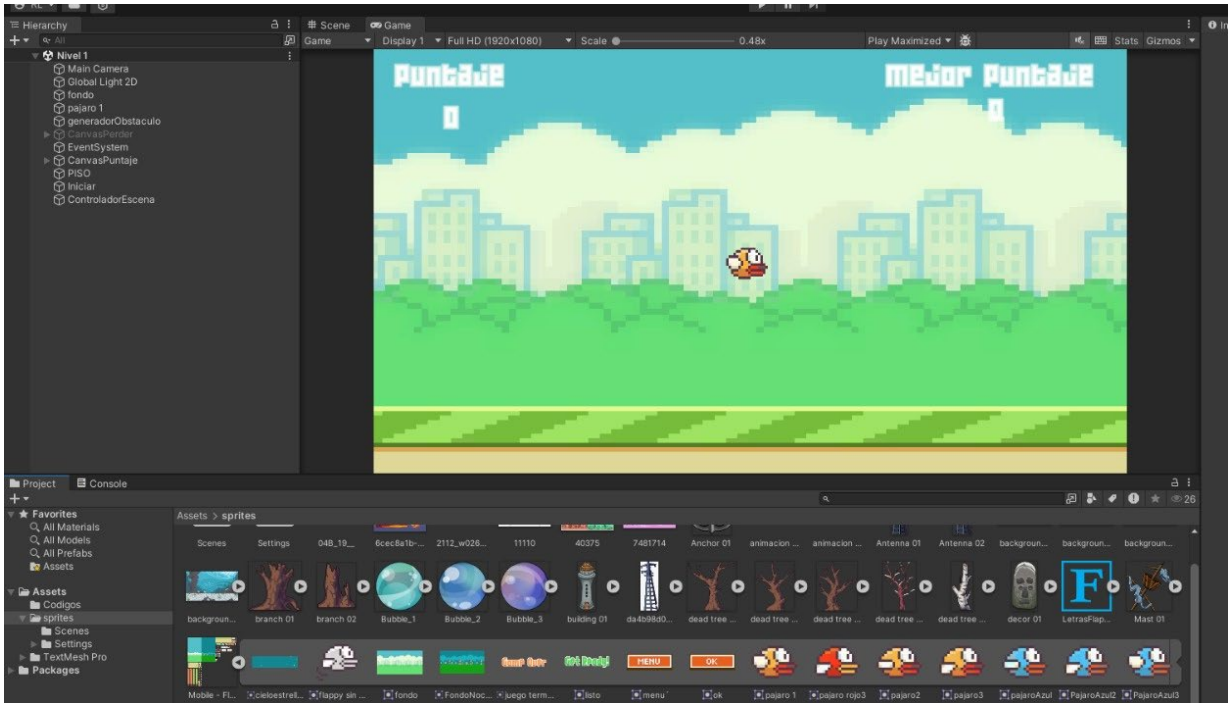


Ilustración 3 - Clase Empezar

```
Script de Unity (3 referencias de recurso) | 0 referencias
public class Empezar: MonoBehaviour
{

    0 referencias
    public void menuNiveles()
    {

        SceneManager.LoadScene(2);

    }
    0 referencias
    public void VolverMenu()
    {
        SceneManager.LoadScene(1);
    }
    0 referencias
    public void nivel1()
    {

    SceneManager.LoadScene(0);
    }
    0 referencias
    public void nivel2()
    {

    SceneManager.LoadScene(3);
    }
    0 referencias
    public void nivel3()
    {
        SceneManager.LoadScene(4);
    }
    0 referencias
    public void nivel4()
    {
        SceneManager.LoadScene(5);
    }
    0 referencias
    public void nivel5()
    { SceneManager.LoadScene(6);
    }
    0 referencias
    public void nivel6()
    {
        SceneManager.LoadScene(7);
    }
    0 referencias
    public void nivel7()
    {

    SceneManager.LoadScene(8);
    }
    0 referencias
    public void nivel8()
    {
        SceneManager.LoadScene(9);
    }
    0 referencias
    public void nivel9()
    {
        SceneManager.LoadScene(10);
    }
    0 referencias
    public void nivel10()
    {
        SceneManager.LoadScene(11);
    }
    0 referencias
    public void Salir()
    {
        Application.Quit();
    }
```

Ilustración 4 - Clase Pajaro

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class NewBehaviourScript : MonoBehaviour
{

    public float velocity = 1;
    public float rotacion= 20;
    private Rigidbody2D rb;
    public ControladorEscena controladorEscena;
    Mensaje de Unity | 0 referencias
    void Start()
    {
        rb=GetComponent<Rigidbody2D>();
    }
    Mensaje de Unity | 0 referencias
    void Update()
    {
        if (Input.GetMouseButtonDown(0)) {
            rb.velocity = Vector2.up* velocity;
        }
    }
    Mensaje de Unity | 0 referencias
    private void FixedUpdate()
    {
        transform.rotation= Quaternion.Euler(0,0, rb.velocity.y* rotacion);
    }
   Mensaje de Unity | 0 referencias
   private void OnCollisionEnter2D(Collision2D collision)
    {
        controladorEscena.Perdiste();
    }
```

Ilustración 5 - Clase Obstaculo

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class Obstaculo : MonoBehaviour
{
    public float velocidad;
    Mensaje de Unity | 0 referencias
    void Start()
    {



    }

    Mensaje de Unity | 0 referencias
    void Update()
    {
        transform.position += Vector3.left * velocidad * Time.deltaTime;
    }
}
```

Ilustración 6 - Clase Generador

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class Generador : MonoBehaviour
{
    public float TiempoMax = 1;
    private float TiempoInicial = 0;
    public GameObject obstaculo;
    public float altura;
    public float tiempoObstaculo = 10f;

    Mensaje de Unity | 0 referencias
    void Update()
    {
        if (TiempoInicial > TiempoMax)
            {
                GameObject obstaculoNuevo = Instantiate(obstaculo);
                obstaculoNuevo.transform.position = transform.position + new Vector3(0, Random.Range(-altura,altura), 0);
                Destroy(obstaculoNuevo,tiempoObstaculo);
            TiempoInicial = 0;
            }
           else
            {
            TiempoInicial += Time.deltaTime;
            }
    }
}
```

Ilustración 7 - Clase ControladorEscena

```
8   public class ControladorEscena : MonoBehaviour
9
10  {
11      public GameObject CanvasPerder;
        Mensaje de Unity | 0 referencias
12      private void Start()
13      {
14          Time.timeScale = 1;
15      }
        1 referencia
16      public void Perdiste()
17      {
18          CanvasPerder.SetActive(true);
19          Time.timeScale = 0;
20      }
        0 referencias
21      public void ReiniciarNivel1()
22      {
23          SceneManager.LoadScene(0);
24          Time.timeScale = 1;
25
26      }
        0 referencias
27      public void ReiniciarNivel2()
28      {
29          SceneManager.LoadScene(3);
30      }
        0 referencias
31      public void ReiniciarNivel3()
32      {
33          SceneManager.LoadScene(4);
34      }
```

```
        0 referencias
35      public void ReiniciarNivel4()
36      {
37          SceneManager.LoadScene(5);
38      }
        0 referencias
39      public void ReiniciarNivel5()
40      {
41          SceneManager.LoadScene(6);
42      }
        0 referencias
43      public void ReiniciarNivel6()
44      {
45          SceneManager.LoadScene(7);
46      }
        0 referencias
47      public void ReiniciarNivel7()
48      {
49          SceneManager.LoadScene(8);
50      }
        0 referencias
51      public void ReiniciarNivel8()
52      {
53          SceneManager.LoadScene(9);
54      }
        0 referencias
55      public void ReiniciarNivel9()
56      {
57          SceneManager.LoadScene(10);
58      }
        0 referencias
59      public void ReiniciarNivel10()
60      {
61          SceneManager.LoadScene(11);
62      }
        0 referencias
63      public void Menu()
64      {
65          SceneManager.LoadScene(1);
66      }
        0 referencias
67      public void salir()
68      {
69
70          Application.Quit();
71      }
72
73  }
```

Ilustración 8 - Clase AreaPuntaje

```
Script de Unity (5 referencias de recurso) | 0 referencias
public class LogicaAreaPuntaje : MonoBehaviour
{
    private Puntaje puntajeScript;

    Mensaje de Unity | 0 referencias
    private void Start()
    {
        puntajeScript = FindObjectOfType<Puntaje>();

    }

    Mensaje de Unity | 0 referencias
    private void OnTriggerEnter2D(Collider2D collision)
    {

        puntajeScript.IncrementarPuntaje();

    }
}
```

Ilustración 9 - Clase Puntaje

```
Script de Unity (5 referencias de recurso) | 2 referencias
public class Puntaje : MonoBehaviour
{
    public TextMeshProUGUI puntajeTexto;
    public TextMeshProUGUI mejorPuntajeTexto;
    public int puntaje = 0;
    public int mejorPuntaje = 0;

    Mensaje de Unity | 0 referencias
    private void Start()
    {

        CargarMejorPuntaje();
        ActualizarTextoPuntaje();
        ActualizarTextoMejorPuntaje();
    }

    1 referencia
    public void IncrementarPuntaje()
    {
        puntaje+=1;
        ActualizarTextoPuntaje();
        if (puntaje > mejorPuntaje)
        {
            mejorPuntaje = puntaje;
            GuardarMejorPuntaje();
            ActualizarTextoMejorPuntaje();
        }
    }

    2 referencias
    private void ActualizarTextoPuntaje()
    {
        puntajeTexto.text =  puntaje.ToString();
    }

    2 referencias
    private void ActualizarTextoMejorPuntaje()
    {
        mejorPuntajeTexto.text = mejorPuntaje.ToString();
    }

    1 referencia
    private void GuardarMejorPuntaje()
    {
        PlayerPrefs.SetInt("MejorPuntaje", mejorPuntaje);
        PlayerPrefs.Save();
    }

    1 referencia
    private void CargarMejorPuntaje()
    {
        if (PlayerPrefs.HasKey("MejorPuntaje"))
        {
            mejorPuntaje = PlayerPrefs.GetInt("MejorPuntaje");
        }
    }

}
```

Nuestro proyecto es de código abierto, ¡Visítanos!