UNIVERSIDAD LUTERANA SALVADOREÑA

FACULTAD DE CIENCIAS DEL HOMBRE Y LA NATURALEZA

LICENCIATURA EN CIENCIAS DE LA COMPUTACIÓN

Proyecto final juego Snake

Integrantes

- José Alexander Rodríguez González
- Carolina Jazmín Martínez
- Jonathan Esaú Magaña Rodríguez

Docente: Inga. Lesbia Mancía

Materia: Ingeniería de Software

San Salvador, 28 de noviembre de 24

Contenido

Introducción

En el curso de Ingeniería de Software, se nos ha encomendado la tarea de desarrollar un proyecto grupal que ponga en práctica los principios, metodologías y herramientas aprendidas a lo largo del semestre. Para este propósito, hemos elegido recrear el clásico juego arcade Snake utilizando Python.

Snake es un juego que ha perdurado en el tiempo debido a su mecánica simple pero desafiante. El jugador controla una serpiente que debe moverse por la pantalla en busca de comida, aumentando su tamaño a medida que la consume. El desafío radica en evitar colisiones con los bordes de la pantalla o con el propio cuerpo de la serpiente, lo que hace del juego una prueba constante de habilidad y estrategia.

Nuestro enfoque en este proyecto no solo se centrará en la recreación del juego, sino también en la implementación de un sistema de puntuación que permitirá a los jugadores ver su puntaje en tiempo real. Además, seguiremos una metodología ágil, dividiendo el trabajo en sprints para asegurar una entrega incremental y continua de valor, con pruebas y ajustes constantes a lo largo del desarrollo.

La confiabilidad y seguridad del software serán aspectos cruciales de nuestro proyecto. Nos aseguraremos de que el juego funcione correctamente bajo condiciones específicas y que esté protegido contra posibles amenazas, ofreciendo una experiencia de usuario segura y satisfactoria. Este proyecto no solo representa una oportunidad para aplicar nuestros conocimientos teóricos en un entorno práctico, sino también para rendir homenaje a un clásico del mundo de los videojuegos.

Al concluir el proyecto, esperamos haber desarrollado un producto que no solo capture la esencia del juego original, sino que también cumpla con los estándares modernos de desarrollo de software. La integración de pruebas exhaustivas, manejo adecuado de errores y medidas de seguridad robustas serán claves para garantizar la calidad del juego. Así, podremos ofrecer una experiencia de juego que no solo sea divertida y desafiante, sino también confiable y segura, logrando así un equilibrio perfecto entre nostalgia y modernidad.

En este documento te presentamos toda la información necesaria para recrear este juego, desde los requisitos de hardware y software hasta los pasos detallados de implementación. Incluimos código fuente comentado y estrategias de prueba para asegurar que puedas seguir cada etapa del desarrollo sin inconvenientes.

Objetivo general

El objetivo de este proyecto final es aplicar los principios y metodologías de Ingeniería de Software para diseñar, desarrollar y entregar una versión funcional, confiable y segura del clásico juego Snake utilizando el lenguaje de programación Python. Este proyecto tiene como propósito demostrar el uso efectivo de tecnologías de desarrollo ágil, promoviendo la colaboración en equipo para crear software de alta calidad. Además, se enfocará en identificar y mitigar posibles vulnerabilidades, garantizando que el software funcione correctamente y esté protegido contra amenazas. Este proyecto no solo servirá como una práctica técnica en programación, sino que también permitirá al equipo comprender mejor el ciclo de vida del desarrollo de software y asegurar que el producto final cumpla con los estándares de calidad y seguridad establecidos.

Objetivos específicos

- Diseñar un diagrama de casos de uso que describa cómo el jugador interactúa con las funciones del juego Snake, y un diagrama de clases que represente la estructura del juego, incluyendo clases como la serpiente, la comida y las direcciones, con sus atributos y métodos.

- Listar y definir los requerimientos que aseguren la confiabilidad y seguridad del software, garantizando que el juego funcione de manera consistente y sin errores, y protegiendo el sistema contra vulnerabilidades.

- Identificar posibles vulnerabilidades tecnológicas que puedan surgir durante el desarrollo del proyecto y establecer mecanismos efectivos para asegurar la seguridad del juego.
- Diseñar y desarrollar una interfaz gráfica utilizando la biblioteca Turtle, que muestre el juego incluyendo la serpiente, la comida y el sistema de puntuación, asegurando una visualización clara y atractiva.

- Implementar funciones para que los jugadores puedan mover la serpiente utilizando las teclas de flecha y crear un sistema que registre y muestre el puntaje actual y el puntaje más alto alcanzado durante la sesión, actualizando estos valores en tiempo real.

- Crear la lógica necesaria para la generación aleatoria de comida y el manejo de colisiones, realizar pruebas unitarias y funcionales para asegurar que todas las características del juego funcionen correctamente y documentar el proceso de desarrollo detalladamente, incluyendo las decisiones tomadas y las lecciones aprendidas.

Descripción General del Proyecto: Juego Snake en Python

Nombre del Proyecto:

Desarrollo del Juego Snake en Python

Descripción Funcional:

Snake es un juego donde el jugador controla una serpiente que se desplaza por la pantalla. La serpiente debe comer "comida" para crecer, y el jugador debe evitar que la serpiente choque contra los bordes de la pantalla o contra sí misma. A medida que la serpiente come, crece en longitud y el juego se vuelve más desafiante. El objetivo es acumular la mayor cantidad de puntos posible antes de que ocurra una colisión.

Alcance del Proyecto:

Este proyecto abarca el desarrollo completo del juego Snake, implementando las siguientes funcionalidades:

- **Movimiento de la serpiente**: La serpiente se mueve continuamente en una dirección, controlada por las teclas de flecha del teclado.
- **Colisiones con los bordes**: Si la serpiente toca el borde de la pantalla, el juego se reinicia y se restablece el puntaje.
- **Comida aleatoria**: La comida aparece en posiciones aleatorias en la pantalla. Cada vez que la serpiente la consume, crece en longitud y el puntaje aumenta.
- **Crecimiento de la serpiente**: Cada vez que la serpiente come, un nuevo segmento se agrega a su cuerpo.
- **Colisiones con el propio cuerpo**: Si la serpiente choca con alguna parte de su propio cuerpo, el juego se reinicia.
- **Sistema de puntuación**: Se muestra el puntaje actual del jugador y la puntuación más alta lograda durante la sesión de juego.
- **Velocidad dinámica**: La velocidad de la serpiente aumenta conforme el jugador acumula más puntos.

Justificación

El videojuego Snake es una excelente opción para un proyecto de desarrollo por razones como que las reglas del juego son simples y fáciles de entender, lo cual lo hace accesible para jugadores de todas las edades, así como es un proyecto ideal para desarrolladores principiantes, ya que no requiere conocimientos avanzados de programación. Desarrollar Snake permite a los programadores aprender a practicar conceptos fundamentales como estructuras de datos (listas), control de flujo (bucles y condicionales), y manejo de eventos. Snake es un juego reconocido mundialmente, lo que puede aumentar el interés y la motivación tanto de los desarrolladores como de los usuarios. Para muchos, Snake evoca recuerdos de los primeros días de los videojuegos móviles, lo que puede atraer a una audiencia más amplia.

Proporciona un entorno controlado para practicar técnicas de pruebas y depuración, esenciales para cualquier proyecto de software.

La simplicidad del juego permite obtener feedback inmediato sobre el funcionamiento del código, facilitando el proceso de aprendizaje.

En resumen, el desarrollo de un videojuego Snake no solo es un proyecto divertido y nostálgico, sino también una valiosa herramienta educativa que puede ayudar a los desarrolladores a adquirir y perfeccionar habilidades esenciales en programación y desarrollo de software.

Listado de requerimientos

Requerimientos Funcionales

1. El jugador puede mover la serpiente en cuatro direcciones (arriba, abajo, izquierda, derecha) usando las flechas del teclado.

1.1 El sistema deberá permitir el acceso a las cuatro flechas (arriba, abajo, izquierda, derecha)

2. El jugador observa que la serpiente aumente de tamaño al consumir el elemento que aparece simbolizando una manzana y esta aleatoriamente en el mapa.

2.1 El sistema debe agregar otro elemento es decir una manzana esta aparecerá en cualquier posición valida del mapa, luego que la serpiente se haya comido la anterior.

2.2 El sistema debe agregar un segmento al cuerpo de la serpiente y actualizar su longitud luego de cada manzana consumida.

2.3 El sistema deberá desaparecer la manzana una vez que la serpiente la consuma.

3. El jugador perderá y termina el juego si la cabeza de la serpiente choca con las paredes del mapa o surge colisiones entre la cabeza de la serpiente y su propio cuerpo.

3.1 Cuando el juego termina por una colisión, se muestra la puntuación final y se reinicia el juego.

3.2 El sistema debe detectar colisiones entre la cabeza de la serpiente y las paredes del mapa y finalizar el juego si ocurre una colisión.

4. El jugador gana puntos por cada elemento consumido, es decir por cada manzana que ha consumido obtiene un cierto puntaje.

4.1 El sistema debe incrementar la puntuación del jugador en una cantidad predefinida (por ejemplo, 1 punto) cada vez que la serpiente consume una manzana.

4.2 El sistema debe actualizar y almacenar la puntuación actual en tiempo real para reflejar el número de elementos consumidos.

4.3 El sistema debe mostrar la puntuación actual del jugador en una parte visible de la interfaz de usuario mientras el juego está en curso.

4.4 Una vez que se detecta el fin del juego, el sistema debe mostrar una pantalla con la puntuación final del jugador

4.5 Después de mostrar la puntuación final, el sistema debe reiniciar automáticamente el juego, restableciendo la serpiente a su tamaño inicial y la puntuación a 0.

5. El jugador se moverá automáticamente a una velocidad constante.

5.1 El sistema debe mover la serpiente automáticamente en la dirección indicada por el jugador a una velocidad constante.

Requerimientos no funcionales

1. La dificultad del juego debe aumentar de manera gradual para mantener el interés del jugador sin hacerlo demasiado frustrante.

2. Aunque el juego tradicionalmente es para un solo jugador, si se planea agregar modos multijugador, el sistema debe ser escalable para manejar múltiples jugadores.

3. Implementar medidas básicas para proteger el juego contra trampas y modificaciones no autorizadas.

4. El diseño gráfico debe ser atractivo, pero no distractor. La estética debe ser coherente con el estilo del juego y mejorar la experiencia del jugador.

5. La música y los efectos de sonido deben ser agradables y no molestos, con opciones para ajustar el volumen o desactivarlos si el jugador lo prefiere.

Desarrollo ágil

Fase 1: Definición del Proyecto y Planificación del Backlog

Objetivos:
Crear una versión funcional del juego Snake utilizando Python y el módulo Turtle, donde el jugador controla una serpiente que debe comer para crecer y evitar chocar con los bordes de la pantalla o con su propio cuerpo.

*Definir claramente los requisitos del juego (funcionalidades y aspectos visuales).

*Crear un backlog de tareas con las características más importantes, priorizando las entregas iterativas.

*Definir las tareas de cada sprint y asignar tiempo a cada uno.

Requisitos funcionales:

1. Movimiento de la serpiente: La serpiente debe moverse en las direcciones indicadas por las teclas.
2. Comida y puntuación: La serpiente debe "comer" la comida y aumentar su tamaño, con el puntaje incrementándose.
3. Colisiones con bordes: La serpiente debe reiniciarse si toca los bordes.
4. Colisiones con el cuerpo: Si la serpiente choca con su propio cuerpo, el juego debe reiniciarse.
5. Interfaz gráfica básica: Pantalla de juego con serpiente y comida.

Backlog inicial:

- Implementar el movimiento básico de la serpiente.
- Crear la comida y detectar colisiones con ella.
- Implementar la detección de colisiones con los bordes de la pantalla.
- Implementar la detección de colisiones con el propio cuerpo.
- Actualizar la puntuación y mostrar la puntuación más alta

Historias de Usuario

- Como jugador, quiero mover la serpiente usando las teclas de flecha para comer comida.
- Como jugador, quiero ver mi puntaje actual y el puntaje más alto en la pantalla.
- Como jugador, quiero que el juego se reinicie si la serpiente choca con los bordes o consigo Misma.

Fase 2:

Sprint 1 - Configuración básica del juego (historias de usuario 1-2)

Nombre de la tarea	Responsable	Fecha de inicio	Fecha final	Días	Estado
1 El jugador puede mover la serpiente en cuatro direcciones (arriba, abajo, izquierda, derecha) usando las flechas del teclado.					
El sistema deberá permitir el acceso a las cuatro flechas (arriba, abajo, izquierda, derecha)	Carolina Martínez	1/09/2024	3/09/2024	3	Finalizado
Nombre de la tarea	**Responsable**	**Fecha de inicio**	**Fecha final**	**Días**	**Estado**
2.El jugador observa que la serpiente aumente de tamaño al consumir el elemento que aparece simbolizando una manzana y esta aleatoriamente en el mapa.					
El sistema debe agregar otro elemento es decir una manzana esta aparecerá en cualquier posición valida del mapa, luego que la serpiente se haya comido la anterior.	Jonathan Rodríguez	3/09/2024	4/09/2024	1	Finalizado
El sistema debe agregar un segmento al cuerpo de la serpiente y actualizar su longitud luego de cada manzana consumida.	Jonathan Rodríguez	4/09/2024	07/09/2024	4	Finalizado
El sistema deberá desaparecer la manzana una vez que la serpiente la consuma.	Jonathan Rodríguez	5/09/2024	10/09/2024	5	Finalizado

Duración: 2 semanas

Objetivos:

- Crear la ventana gráfica y los objetos básicos: la serpiente y la comida.
- Configurar el teclado para que controle el movimiento de la serpiente.
- Crear el bucle principal del juego.

Tareas:

1. **Importación de módulos**: Asegurarse de que los módulos turtle, time y random estén importados correctamente.
2. **Crear la ventana del juego**: Definir la ventana gráfica y sus propiedades (tamaño, título, color).
3. **Inicializar la serpiente**: Crear la cabeza de la serpiente con los parámetros de inicio.
4. **Inicializar la comida**: Colocar la comida en la pantalla de manera aleatoria.
5. **Configurar los controles de movimiento**: Vincular las teclas de flecha para controlar la dirección de la serpiente.

Revisión:

- Asegurarse de que la serpiente se mueve correctamente en las direcciones indicadas.
- Verificar que la ventana del juego funcione sin errores y que los gráficos básicos sean visibles.

Fase 3: Sprint 2 - Lógica de juego básica (historias de usuario 3-4)

Nombre de la tarea	Responsable	Fecha de inicio	Fecha final	Días	Estado
3 El jugador perderá y termina el juego si la cabeza de la serpiente choca con las paredes del mapa o surge colisiones entre la cabeza de la serpiente y su propio cuerpo.					
Cuando el juego termina por una colisión, se muestra la puntuación final y se reinicia el juego.	José Rodríguez	20/09/2024	20/04/2020	10	Pendiente
El sistema debe detectar colisiones entre la cabeza de la serpiente y las paredes del mapa y finalizar el juego si ocurre una colisión.	José Rodríguez	20/09/2024	21/09/2024	2	Pendiente
Nombre de la tarea	**Responsable**	**Fecha de inicio**	**Fecha final**	**Días**	**Estado**
4 El jugador gana puntos por cada elemento consumido, es decir por cada manzana que ha consumido obtiene un cierto puntaje.					
El sistema debe incrementar la puntuación del jugador en una cantidad predefinida (por ejemplo, 1 punto) cada vez que la serpiente consume una manzana.	José Rodríguez	21/09/2024	30/09/2024	9	Pendiente
El sistema debe actualizar y almacenar la puntuación actual en tiempo real para reflejar el número de elementos consumidos.	José Rodríguez	1/10/2024	10/10/2024	10	Pendiente
El sistema debe mostrar la puntuación actual del jugador en una parte visible de la interfaz de usuario mientras el juego está en curso.	Carolina Martínez	10/10/2024	15/10/2024	5	Pendiente
Una vez que se detecta el fin del juego, el sistema debe mostrar una pantalla con la puntuación final del jugador	Carolina Martínez	15/10/2024	23/10/2024	8	Pendiente
Después de mostrar la puntuación final, el sistema debe reiniciar automáticamente el juego, restableciendo la serpiente a su tamaño inicial y la puntuación a 0.	Carolina Martínez	23/10/2024	31/10/2024	9	Pendiente

Duración: 6 semanas

Objetivos:

- Implementar las colisiones básicas: con los bordes y con la comida.
- Incrementar la longitud de la serpiente al comer.
- Aumentar el puntaje.

Tareas:

1. **Detectar colisiones con los bordes**: Implementar la lógica para reiniciar el juego cuando la serpiente toque el borde.
2. **Detectar colisiones con la comida**: Implementar la lógica para que la comida desaparezca y reaparezca en una posición aleatoria.
3. **Crecer la serpiente**: Añadir segmentos al cuerpo de la serpiente al comer.
4. **Actualizar el puntaje**: Incrementar el puntaje cada vez que la serpiente coma comida y mostrar la puntuación actual.

Revisión:

- Verificar que la serpiente crece al comer comida.
- Comprobar que el puntaje se actualiza correctamente y que las colisiones con los bordes reinician el juego.

Fase 4: Sprint 3 - Mejoras y optimización del juego (continuación de la historia de usuario 4-5)

Nombre de la tarea	Responsable	Fecha de inicio	Fecha final	Días	Estado
5 El jugador se moverá automáticamente a una velocidad constante.					
El sistema debe mover la serpiente automáticamente en la dirección indicada por el jugador a una velocidad constante.	Jonathan Rodríguez	1/11/2024	10/11/2024	10	Pendiente

Duración: 1 semana

Objetivos:

- Implementar la detección de colisiones con el propio cuerpo de la serpiente.
- Optimizar la velocidad de la serpiente según el puntaje.
- Actualizar la puntuación más alta.

Tareas:

1. **Colisiones con el cuerpo**: Detectar si la serpiente choca consigo misma y reiniciar el juego en ese caso.
2. **Controlar la velocidad del juego**: Modificar el tiempo de pausa entre iteraciones (posponer) para aumentar la velocidad conforme la serpiente crece.
3. **Mantener la puntuación más alta**: Almacenar y mostrar la puntuación más alta lograda.

Revisión:

- Comprobar que la serpiente colisiona consigo misma y que el juego se reinicia correctamente.
- Verificar que la velocidad del juego aumenta conforme el jugador avanza.
- Asegurarse de que la puntuación más alta se actualiza correctamente y se muestra al jugador.

Fase 5: Sprint 4 - Testing y corrección de errores

Duración: 1 semana

Objetivos:

- Realizar pruebas exhaustivas para asegurar que todas las funcionalidades del juego se comportan como se espera.
- Corregir cualquier error o bug encontrado.

Tareas:

1. **Pruebas de colisiones**: Asegurarse de que las colisiones con los bordes y con el cuerpo de la serpiente se detectan correctamente.

2. **Pruebas de crecimiento de la serpiente**: Verificar que la serpiente crece correctamente y que los segmentos siguen a la cabeza.
3. **Pruebas de rendimiento**: Asegurarse de que el juego funciona sin retardos, incluso cuando la serpiente es muy grande.

Revisión:

- Confirmar que no hay errores o comportamientos inesperados en el juego.
- Realizar pruebas de usuario y recopilar retroalimentación.

Tecnologías para ocupar en el proyecto

Lenguaje de Programación

- **Python**: Es el lenguaje principal para desarrollar el juego, conocido por su simplicidad y facilidad de uso. La biblioteca turtle se utilizará para la creación de gráficos y la interfaz del juego.

Bibliotecas

- **Turtle**: Esta biblioteca integrada en Python permite crear gráficos y manejar eventos de manera sencilla, ideal para juegos 2D simples como Snake.
- **Random**: Utilizada para generar posiciones aleatorias para la comida dentro del juego.
- **Time**: Para manejar pausas en el juego, permitiendo controlar la velocidad de la serpiente.

Entorno de Desarrollo

- **IDE (Entorno de Desarrollo Integrado)**: Usar un buen IDE te ayudará a escribir y depurar el código de manera más eficiente. Algunas opciones populares incluyen:
    - **PyCharm**: Uno de los IDE más completos para Python, con herramientas de depuración y gestión de entornos virtuales.
    - **VS Code**: Ligero y fácil de usar, con muchas extensiones que facilitan el desarrollo en Python.

Metodologías Ágiles

- **Scrum** : Esta metodología pueden ser adoptadas para gestionar el desarrollo del proyecto. Scrum implica trabajar en sprints cortos, se enfoca en visualizar el flujo de trabajo

Control de Versiones

- **Git**: Para el control de versiones del código fuente. Permite a los desarrolladores colaborar eficazmente y mantener un historial de cambios.

Pruebas y Depuración

- **Unittest** o **PyTest**: Son dos marcos de pruebas para Python que puedes utilizar para asegurarte de que las funciones principales del juego (como las colisiones, el crecimiento de la serpiente, y la actualización de puntuaciones) funcionan correctamente.
- **PDB** (Python Debugger): Herramienta de depuración que te permite inspeccionar y ejecutar tu código paso a paso para detectar errores.

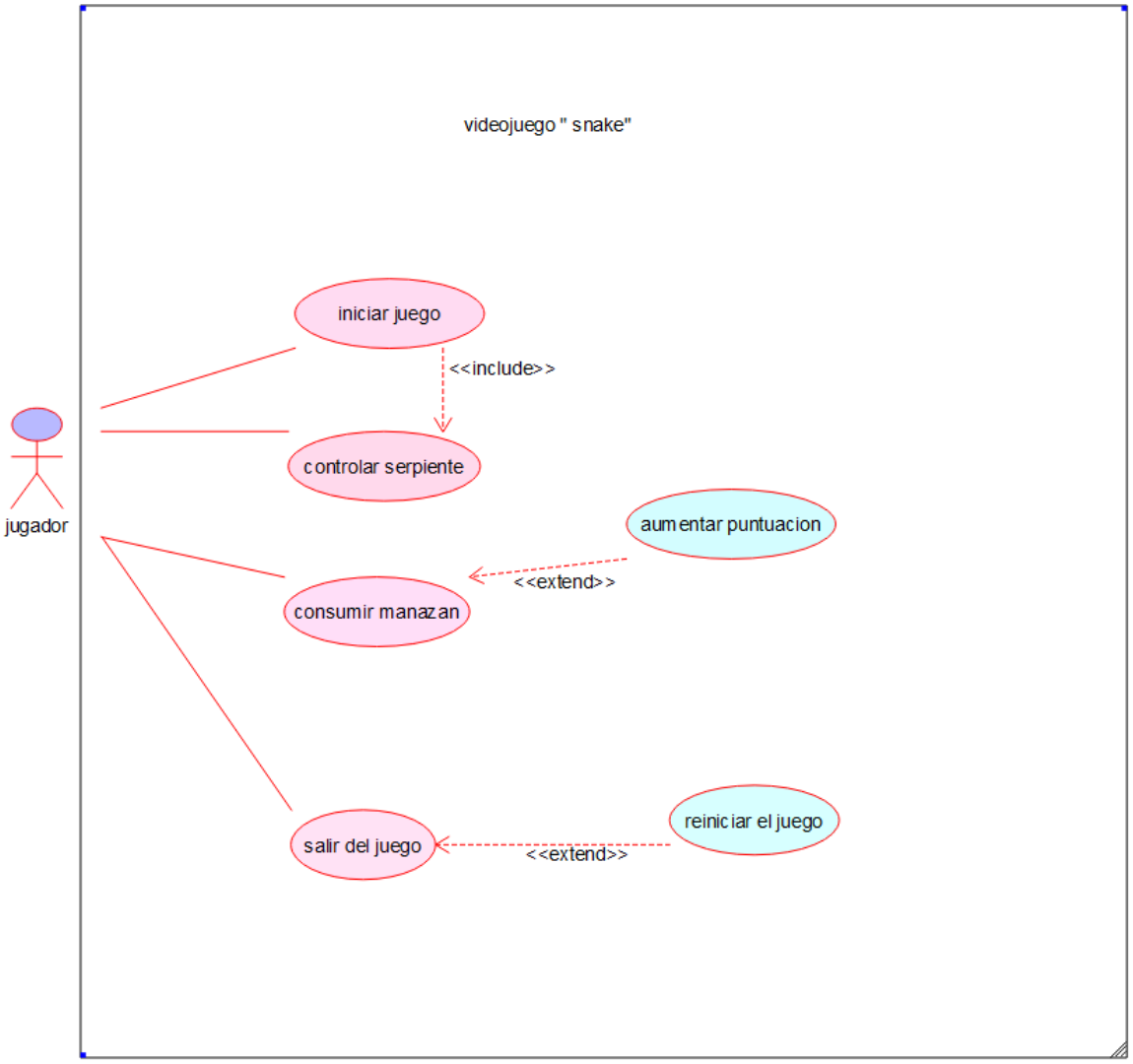
Distribución del Proyecto

- **PyInstaller**: Si quieres distribuir tu juego como un ejecutable para usuarios que no tienen Python instalado, puedes usar PyInstaller. Esta herramienta empaqueta el código en un archivo ejecutable para Windows, macOS o Linux.

Administración de Dependencias

- **pip**: Para gestionar los paquetes externos que necesites (aunque en este proyecto solo se usa la biblioteca estándar de Python, pip es útil para instalar otros paquetes si es necesario).

Diagrama de caso de uso



Descripción de casos de uso

Caso de uso	**Iniciar juego**
Objetivo	Comenzar una nueva partida de Snake
Actor principal	Jugador
Personal involucrado	N/A
Precondición	El juego está instalado y operativo
Garantías de éxito	El juego se inicia correctamente y la serpiente comienza a moverse
Escenario principal	1. El jugador "Inicia el Juego". 2. La pantalla de juego aparece, la serpiente se genera en su posición inicial, y la comida aparece en un lugar aleatorio
Flujos alternativos	Si ocurre un error técnico (fallo de hardware o software), el sistema muestra un mensaje de error.
Requisitos especiales	Debe cargar rápidamente, y la comida debe generarse en una posición aleatoria que no choque con la serpiente.
Frecuencia	Al inicio de cada partida

Caso de uso	**Controlar serpiente**
Objetivo	Permitir al jugador mover la serpiente por la pantalla
Actor principal	Jugador
Personal involucrado	N/A
Precondición	El juego está en progreso
Garantías de éxito	La serpiente responde correctamente a las direcciones dadas por el jugador
Escenario principal	1. El jugador usa las flechas del teclado para controlar la dirección de la serpiente. 2. La serpiente se mueve de acuerdo a la dirección seleccionada.
Flujos alternativos	Si la serpiente choca contra un borde o contra sí misma, se termina el juego.
Requisitos especiales	La serpiente no debe poder moverse en dirección opuesta a su trayectoria actual (para evitar colisiones inmediatas).
Frecuencia	Continuamente durante el juego
Caso de uso	**Consumir manzana**
Objetivo	Hacer que la serpiente consuma la comida y crezca en tamaño

Actor principal	Jugador
Personal involucrado	N/A
Precondición	La serpiente debe tocar la comida
Garantías de éxito	La serpiente crece en longitud y aparece una nueva manzana en una ubicación aleatoria
Escenario principal	1. La serpiente entra en contacto con la comida. 2. La comida desaparece y la serpiente aumenta su longitud. 3. Aparece una nueva manzana en otro lugar de la pantalla
Flujos alternativos	Si la serpiente no toca la manzana, la manzana permanece en la pantalla hasta que sea consumida
Requisitos especiales	Si la serpiente no toca la manzana, la manzana permanece en la pantalla hasta que sea consumida
Frecuencia	Ocasional, cada vez que la serpiente consume una manzana

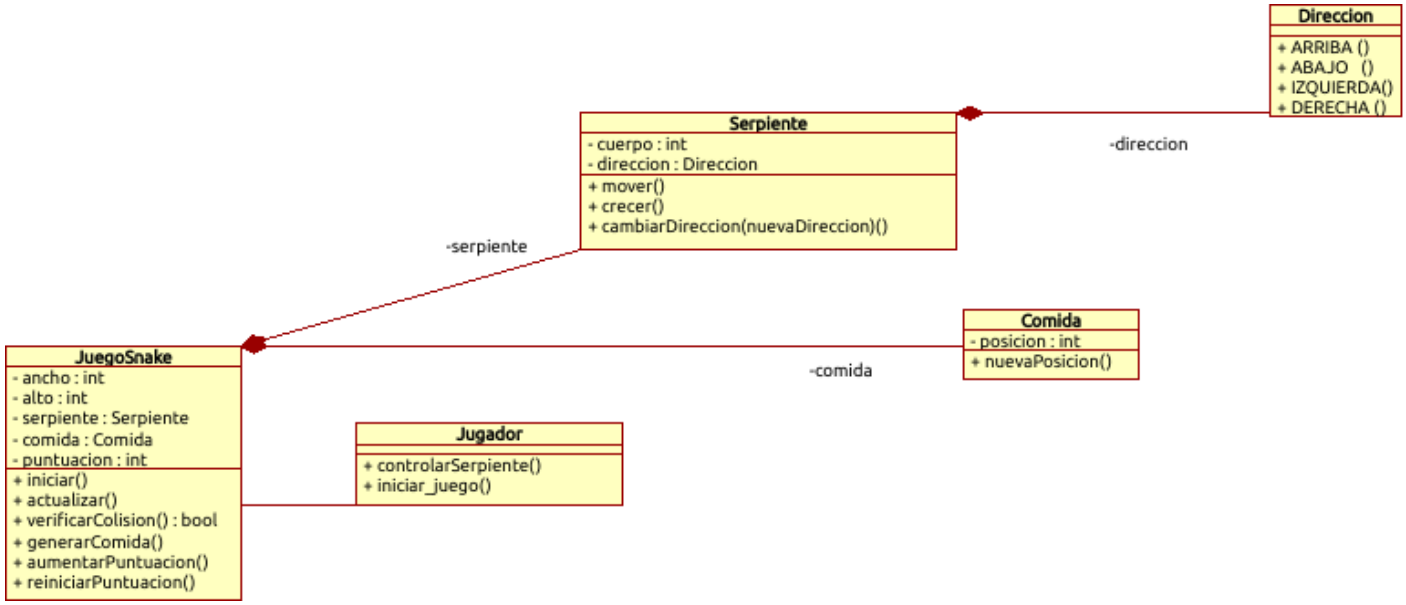
Caso de uso	**Aumentar puntuación**
Objetivo	Incrementar la puntuación del jugador cada vez que come
Actor principal	Jugador
Personal involucrado	N/A
Precondición	La serpiente consume una manzana
Garantías de éxito	La puntuación del jugador se incrementa correctamente
Escenario principal	1. Cada vez que la serpiente consume una manzana, el sistema incrementa la puntuación del jugador. 2. La nueva puntuación se muestra en pantalla.
Flujos alternativos	Si no se consume la manzana, la puntuación permanece igual.
Requisitos especiales	La puntuación debe reflejarse en tiempo real, y puede mostrarse en una parte de la pantalla.
Frecuencia	Ocasional, cada vez que se consume una manzana

Caso de uso	**Salir del juego**
Objetivo	Permitir al jugador cerrar el juego
Actor principal	Jugador

Personal involucrado	N/A
Precondición	El juego está en curso
Garantías de éxito	El juego se cierra correctamente
Escenario principal	El juego se detiene y el sistema lo cierra
Flujos alternativos	Si el jugador elije "Salir", el juego se sale y reinicia desde cero.
Requisitos especiales	El sistema debe guardar las puntuaciones o estadísticas antes de salir, si en caso aplicara.
Frecuencia	Ocasional, al final o intermedio del juego

Caso de uso	**Reiniciar el juego**
Objetivo	Empezar una nueva partida desde cero
Actor principal	Jugador
Personal involucrado	N/A
Precondición	El juego debe estar en estado de "Game Over" o pausado
Garantías de éxito	El juego se reinicia correctamente y la serpiente aparece en su posición inicial
Escenario principal	1. Después de un "Game Over", el jugador puede reiniciar el juego 2. El juego vuelve a su estado inicial con una nueva serpiente y una nueva comida en pantalla.
Flujos alternativos	Si el jugador no selecciona desea reiniciar, puede salir del juego.
Requisitos especiales	El sistema debe reiniciar todos los valores (puntuación, longitud de la serpiente) y generar una nueva comida aleatoria.
Frecuencia	Después de cada partida (frecuente)

Diagrama de clases



Requerimientos de confiabilidad y seguridad

Listado de Requerimientos para Garantizar Confiabilidad y Seguridad del Software

1 movimiento de la Serpiente

1.1. El sistema no debe permitir que la serpiente se mueva en direcciones inválidas o conflictivas.

Requerimiento de Seguridad: El sistema no debe aceptar entradas de usuario no válidas que puedan generar movimientos indeseados o erróneos de la serpiente.

2 crecimiento de la Serpiente y Generación de Manzanas

2.1. El sistema no debe generar una manzana en una posición que esté ocupada por la serpiente o fuera del área válida del mapa.

2.2. El sistema no debe dejar de actualizar correctamente la longitud de la serpiente tras consumir una manzana.

2.3. El sistema no debe mantener una manzana en el mapa una vez que ha sido consumida por la serpiente.

Requerimiento de Confiabilidad: El sistema no debe permitir que la manzana se genere en una posición donde se superponga con la serpiente o en posiciones no válidas.

Requerimiento de Seguridad: El sistema no debe permitir que la generación de nuevos segmentos de la serpiente provoque errores de memoria o desbordamientos.

3 detección de Colisiones y Terminación del Juego

3.1. El sistema no debe ignorar la finalización del juego cuando ocurra una colisión con una pared o el cuerpo de la serpiente.

3.2. El sistema no debe fallar en la detección de colisiones entre la cabeza de la serpiente y las paredes del mapa.

Requerimiento de Confiabilidad: El sistema no debe generar falsos positivos o negativos en la detección de colisiones.

Requerimiento de Seguridad: El sistema no debe permitir que la detección de colisiones provoque fallos en la estabilidad o vulnerabilidades explotables.

4 sistema de Puntuación

4.1. El sistema no debe fallar en incrementar la puntuación cuando la serpiente consuma una manzana.

4.2. El sistema no debe perder la puntuación actual ni dejar de actualizarla en tiempo real.

4.3. El sistema no debe omitir la visualización de la puntuación actual mientras el juego está en curso.

4.4. El sistema no debe omitir la visualización de la puntuación final una vez que el juego ha terminado.

4.5. El sistema no debe mantener la puntuación anterior tras el reinicio del juego.

Requerimiento de Confiabilidad: El sistema no debe dejar de actualizar o almacenar correctamente la puntuación en tiempo real.

Requerimiento de Seguridad: El sistema no debe permitir la manipulación o alteración indebida de la puntuación por parte de usuarios no autorizados.

5 movimiento Automático de la Serpiente

5.1. El sistema no debe detener o ralentizar el movimiento automático de la serpiente sin motivo.

Requerimiento de Confiabilidad: El sistema no debe permitir que el movimiento automático de la serpiente sea inconsistente o irregular.

Requerimiento de Seguridad: El sistema no debe ser susceptible a interrupciones o ataques que comprometan el control del movimiento automático.

Vulnerabilidad asociada con las opciones de tecnología

En el desarrollo del juego clásico Snake utilizando Python, es crucial identificar y mitigar posibles vulnerabilidades que puedan comprometer la seguridad y confiabilidad del software. A continuación, se presentan algunas de las vulnerabilidades más comunes relacionadas con Python y cómo pueden afectar al desarrollo de Snake, junto con estrategias para abordarlas:

1.Validación de Entrada

•Vulnerabilidad: La entrada del usuario no validada puede llevar a inyecciones de código o comportamientos inesperados.

2.Manejo de Errores

•Vulnerabilidad: Errores no gestionados adecuadamente pueden causar que el juego se bloquee o exponga datos sensibles.

3.Dependencias Desactualizadas

•Vulnerabilidad: El uso de bibliotecas y módulos desactualizados puede exponer el juego a vulnerabilidades conocidas.

4.Acceso a Datos

•Vulnerabilidad: Acceso no controlado a datos del usuario, como la puntuación, puede llevar a manipulación indebida.

5.Traversión de Directorios

•Vulnerabilidad: Manejo incorrecto de rutas de archivos puede permitir a un atacante acceder a archivos fuera del directorio permitido.

6.Paquetes Maliciosos

•Vulnerabilidad: La inclusión de paquetes maliciosos en el proyecto puede introducir comportamientos no deseados y comprometer la seguridad del juego.

Tipo de software avanzado en el proyecto

1. Reutilización de Software

Justificación:

- **Uso de componentes reutilizables**: La biblioteca **Turtle** es un ejemplo directo de reutilización de software, ya que proporciona herramientas preexistentes para manejar gráficos y eventos de manera eficiente.
- **Abstracción de funcionalidades**: Aunque el proyecto es pequeño, se puede **modularizar el código** creando funciones o clases específicas para manejar partes del juego, como la serpiente, la comida y la puntuación. Esto facilita la reutilización de las clases o módulos en otras partes del juego o incluso en otros proyectos.
- **Escalabilidad**: Si en el futuro se desea añadir más funcionalidades al juego, como un sistema de niveles o integrarlo con otros sistemas, tener un enfoque de reutilización hará que sea más fácil hacerlo sin reescribir el código desde cero.

Características:

- **Uso de componentes existentes**: Se aprovechan módulos, bibliotecas o clases ya desarrolladas para evitar duplicar esfuerzos.
- **Abstracción y modularidad**: Las funcionalidades se estructuran de forma independiente, lo que facilita su reutilización.
- **Escalabilidad**: A medida que el proyecto crece, los componentes reutilizables pueden integrarse fácilmente para añadir nuevas funcionalidades.

Ventajas:

- **Ahorro de tiempo y esfuerzo**: Al reutilizar código existente, se reducen los tiempos de desarrollo.
- **Reducción de errores**: Los componentes reutilizados ya están probados, lo que disminuye la probabilidad de errores.
- **Facilita el mantenimiento**: Los componentes reutilizables son más fáciles de mantener y actualizar sin afectar otras partes del sistema.

Desventajas:

- **Dependencia de componentes externos**: Si un componente reutilizado tiene un bug o se actualiza de manera incompatible, puede generar problemas en el proyecto.

Limitación en la personalización: Puede ser difícil adaptar componentes reutilizados a necesidades muy específicas o complejas del proyecto.

2. Ingeniería de Software Basada en Componentes

Justificación:

- **Modularización del código**: El proyecto puede beneficiarse de una **arquitectura de componentes**, donde diferentes aspectos del juego se dividen en módulos independientes que interactúan entre sí. Por ejemplo, se puede tener:
    - **Componente gráfico**: Para gestionar los elementos visuales (serpiente, comida, fondo).
    - **Componente de lógica de juego**: Para controlar las reglas del juego, la detección de colisiones y la actualización de la puntuación.
    - **Componente de entrada**: Para gestionar la entrada del usuario (teclas para mover la serpiente).
- **Facilidad de mantenimiento**: Dividir el proyecto en componentes facilita la **modificación y actualización del código**, si deseas cambiar la forma en que se maneja la puntuación o añadir nuevas características (como nuevos tipos de comida o efectos visuales), puedes hacerlo de forma independiente sin afectar otras partes del código.
- **Mejora de la calidad del código**: La separación de responsabilidades, como tener funciones específicas para cada tarea (por ejemplo, una función para mover la serpiente, otra para verificar colisiones), sigue los principios de la **ingeniería de software basada en componentes**, lo que mejora la organización y la legibilidad del código.

Características:

- **Modularización**: El sistema se divide en partes independientes (componentes) que manejan funcionalidades específicas.
- **Desarrollo independiente**: Cada componente se puede desarrollar, probar y mantener por separado.
- **Interoperabilidad**: Los componentes interactúan entre sí a través de interfaces bien definidas, permitiendo su integración.

Ventajas:

- **Flexibilidad**: La modularidad permite modificar o actualizar partes del sistema sin afectar el todo.
- **Mantenibilidad**: El código es más fácil de mantener y mejorar, ya que cada componente tiene una responsabilidad específica.
- **Escalabilidad**: Se pueden agregar nuevos componentes sin interrumpir el sistema completo.

Desventajas:

- **Complejidad de integración**: Integrar y hacer que los componentes trabajen juntos puede ser desafiante si no se gestionan adecuadamente las interfaces.
- **Rendimiento**: El uso de múltiples componentes puede introducir sobrecarga, especialmente si no están optimizados.
- **Requiere buena planificación**: La correcta división en componentes requiere una planificación detallada para evitar dependencias o conflictos entre ellos

Tipos de riesgos

Tipo de riesgos	Riesgos posibles
Tecnológico	• La biblioteca **Turtle** tiene limitaciones en rendimiento cuando se manejan muchos elementos gráficos (serpiente larga o múltiples elementos en pantalla). • **Turtle** puede no soportar características avanzadas, como gráficos complejos que pueden ser necesarios para mejorar la experiencia visual. • La biblioteca puede presentar diferencias de comportamiento en distintos sistemas operativos, especialmente en ventanas gráficas.
Personal	• Falta de experiencia del equipo con la biblioteca **Turtle**, lo que podría generar demoras en la implementación de funcionalidades clave. • Problemas de disponibilidad de tiempo para cumplir con el cronograma debido a otros compromisos académicos.
De organización	• Cambios en los requerimientos funcionales durante el desarrollo, como ajustes en las reglas del juego o diseño gráfico, que podrían requerir rediseños significativos.
Herramientas	• Dificultad para integrar Turtle con módulos adicionales, como almacenamiento de datos en archivos para guardar puntuaciones • Falta de acceso a recursos adicionales, como imágenes personalizadas para mejorar la experiencia visual.
Requerimientos	• Dificultad para implementar el aumento de dificultad progresiva de manera equilibrada para mantener el interés del jugador.
Estimación	• Subestimación del tiempo necesario para implementar funcionalidades clave, como el sistema de colisiones o el crecimiento dinámico de la serpiente. • Subestimación de la complejidad técnica para optimizar el juego, especialmente en el manejo de memoria o rendimiento con serpientes largas.
Calidad del producto	• Errores en la detección de colisiones entre la serpiente y las paredes o su propio cuerpo, lo que podría llevar a un juego inconsistente.

Riesgos

Riesgos	Probabilidad	Efecto
• La biblioteca **Turtle** tiene limitaciones en rendimiento cuando se manejan muchos elementos gráficos (serpiente larga o múltiples elementos en pantalla). • **Turtle** puede no soportar características avanzadas, como gráficos complejos que pueden ser necesarios para mejorar la experiencia visual. • La biblioteca puede presentar diferencias de comportamiento en distintos sistemas operativos, especialmente en ventanas gráficas.	Alta	Medio
• Falta de experiencia del equipo con la biblioteca **Turtle**, lo que podría generar demoras en la implementación de funcionalidades clave. • Problemas de disponibilidad de tiempo para cumplir con el cronograma debido a otros compromisos académicos.	Media	Alto
• Cambios en los requerimientos funcionales durante el desarrollo, como ajustes en las reglas del juego o diseño gráfico, que podrían requerir rediseños significativos.	Alta	Medio
• Dificultad para integrar Turtle con módulos adicionales, como almacenamiento de datos en archivos para guardar puntuaciones • Falta de acceso a recursos adicionales, como imágenes personalizadas para mejorar la experiencia visual.	Media	Medio
• Dificultad para implementar el aumento de dificultad progresiva de manera equilibrada para mantener el interés del jugador.	Media	Alto
• Subestimación del tiempo necesario para implementar funcionalidades clave, como el sistema de colisiones o el crecimiento dinámico de la serpiente. • Subestimación de la complejidad técnica para optimizar el juego, especialmente en el manejo de memoria o rendimiento con serpientes largas.	Alta	Alto
• Errores en la detección de colisiones entre la serpiente y las paredes o su propio cuerpo, lo que podría llevar a un juego inconsistente.	Media	Alto

Estrategias para gestionar los riesgos del proyecto

Tipo de riesgos	Riesgos posibles	Estrategias para gestionar
Tecnológico	• La biblioteca **Turtle** tiene limitaciones en rendimiento cuando se manejan muchos elementos gráficos (serpiente larga o múltiples elementos en pantalla). • **Turtle** puede no soportar características avanzadas, como gráficos complejos que pueden ser necesarios para mejorar la experiencia visual. • La biblioteca puede presentar diferencias de comportamiento en distintos sistemas operativos, especialmente en ventanas gráficas.	• Implementar técnicas para optimizar el uso de recursos gráficos, como reducir la frecuencia de actualización de la pantalla y simplificar los gráficos cuando sea posible. • Considerar el uso de otras bibliotecas gráficas que puedan complementar o reemplazar Turtle si las necesidades superan sus capacidades • Realizar pruebas en distintos sistemas operativos para identificar y solucionar problemas específicos de cada plataforma.
Personal	• Falta de experiencia del equipo con la biblioteca **Turtle**, lo que podría generar demoras en la implementación de funcionalidades clave. • Problemas de disponibilidad de tiempo para cumplir con el cronograma debido a otros compromisos académicos.	• Organizar sesiones de entrenamiento y compartir recursos educativos sobre la biblioteca Turtle para aumentar la competencia del equipo. • Crear un cronograma flexible y priorizar tareas críticas, distribuyendo el trabajo equitativamente entre los miembros del equipo.
De organización	• Cambios en los requerimientos funcionales durante el desarrollo, como ajustes en las reglas del juego o diseño gráfico, que podrían requerir rediseños significativos.	• Mantener una documentación clara y actualizada de los requerimientos y comunicar cualquier cambio de manera efectiva. • Identificar y asegurar los recursos necesarios desde el inicio del proyecto, buscando alternativas gratuitas o de bajo costo cuando sea posible.
Herramientas	• Dificultad para integrar Turtle con módulos adicionales, como almacenamiento de datos en archivos para guardar puntuaciones • Falta de acceso a recursos adicionales, como imágenes personalizadas para mejorar la experiencia visual.	• Investigar y probar soluciones de integración compatibles con Turtle antes de implementarlas en el proyecto. • Aprovechar bibliotecas y recursos gratuitos disponibles en línea para complementar las funciones necesarias.
Requerimientos	• Dificultad para implementar el aumento de dificultad progresiva de manera equilibrada para mantener el interés del jugador.	• Implementar un diseño modular que permita ajustes fáciles en la mecánica del juego sin afectar otras partes.
Estimación	• Subestimación del tiempo necesario para implementar funcionalidades clave, como el	• Dividir las tareas en componentes más pequeños y realizar estimaciones más precisas

	sistema de colisiones o el crecimiento dinámico de la serpiente. • Subestimación de la complejidad técnica para optimizar el juego, especialmente en el manejo de memoria o rendimiento con serpientes largas.	• Usar métodos de gestión ágil para revisar y ajustar continuamente las estimaciones y prioridades del proyecto.
Calidad de producto -	• Errores en la detección de colisiones entre la serpiente y las paredes o su propio cuerpo, lo que podría llevar a un juego inconsistente.	• Implementar pruebas unitarias para cada función de detección de colisiones y pruebas funcionales del juego completo.

Conclusión

El desarrollo del juego Snake ha sido una experiencia enriquecedora y desafiante que nos ha permitido aplicar de manera práctica los principios y metodologías de la Ingeniería de Software aprendidas a lo largo del curso. Este proyecto no solo ha servido como una plataforma para mejorar nuestras habilidades de programación en Python, utilizando la biblioteca Turtle, sino que también ha puesto a prueba nuestra capacidad para trabajar en equipo, gestionar riesgos y cumplir con los estándares de calidad y seguridad del software.

Hemos desarrollado diagramas de casos de uso y clases detallados que representan cómo el jugador interactúa con el juego y cómo se estructuran las clases principales del sistema. Siguiendo un enfoque ágil, dividimos el trabajo en sprints, lo que nos permitió iterar rápidamente y ajustar el desarrollo según las necesidades emergentes. Identificamos y gestionamos una variedad de riesgos tecnológicos, personales, de organización, de herramientas, de requerimientos y de estimación, aplicando estrategias específicas para cada uno. Implementamos pruebas unitarias y funcionales para garantizar que la detección de colisiones y otras funcionalidades críticas funcionaran correctamente, asegurando una experiencia de juego consistente y fluida. Además, generamos una documentación detallada que incluye todas las fases del desarrollo, decisiones técnicas y lecciones aprendidas, facilitando el mantenimiento y futuras mejoras del juego.

Este proyecto ha resaltado la importancia de la planificación, la comunicación efectiva y la flexibilidad en el desarrollo de software. Nos enfrentamos a diversos desafíos, desde limitaciones tecnológicas hasta la gestión de tiempo y recursos, pero logramos superarlos mediante el trabajo en equipo y la aplicación de buenas prácticas de ingeniería. Aunque hemos alcanzado muchos de nuestros objetivos, siempre hay espacio para mejoras. Algunas posibles mejoras futuras incluyen ampliar la funcionalidad del juego, añadir nuevas características y modos de juego para mantener el interés de los jugadores, mejorar la interfaz gráfica utilizando bibliotecas gráficas más avanzadas y optimizar el rendimiento afinando el código para manejar mejor los recursos y mejorar la fluidez del juego.

Manual del usuario

Documento

https://drive.google.com/file/d/1HVO4YUJ9TftN9XqaWpwg7vDq-ru8Wuut/view?usp=sharing

Enlace de videotutorial

https://youtu.be/sD6EwnPdl7I?si=wlAhVPhnJV7KtCNQ

Bibliografía

- *Campus Virtual | ULS: Entrar al sitio*. (s. f.). http://campus.uls.edu.sv/pluginfile.php/20840/mod_resource/content/1/0.%20Libro_Somerville_9.pdf
- Aqua Security. (2024, 23 julio). *Python Security: 6 common Risks & What you can do About them*. Aqua. https://www.aquasec.com/cloud-native-academy/application-security/python-security/?form=MG0AV3
- Ciberseg. (2022, 16 febrero). *Python y ciberseguridad, todo lo que debes saber*. Ciberseguridad.https://ciberseguridad.com/guias/prevencion-proteccion/python/?form=MG0AV3

Anexos

Autoevaluación y coevaluación

Autoevaluación		
Nombre del Estudiante: José Alexander Rodríguez González		
N°	**Aspecto a Evaluar**	**Rúbrica (Nota)**
1	Colaboré con las tareas que me fueron asignadas	10
2	Participé en forma activa en el trabajo de equipo	10
3	Mantuve comunicación con el equipo	9
4	Cumplí a tiempo con las actividades designadas	9
5	Aporté ideas de calidad	9
Total		47

Coevaluación		
Nombre del Evaluador: José Alexander Rodríguez González		
Nombre del Evaluado: Carolina Jazmín Martínez		
N°	Aspecto a Evaluar	Rúbrica (Nota)
1	Demostró compromiso y responsabilidad con el grupo	10
2	Demostró respeto y tolerancia hacia las opiniones de los demás	10
3	Aportó al desarrollo del proyecto	10
4	Propicia un clima de trabajo agradable	10
5	Antes de entregar la tarea, fue revisado por el evaluado	10
Total		50

Autoevaluación		
Nombre del Estudiante: Jonathan Esaú Magaña Rodríguez		
N°	**Aspecto a Evaluar**	**Rúbrica (Nota)**
1	Colaboré con las tareas que me fueron asignadas	9
2	Participé en forma activa en el trabajo de equipo	9
3	Mantuve comunicación con el equipo	9
4	Cumplí a tiempo con las actividades designadas	9
5	Aporté ideas de calidad	9
Total		45

Coevaluación		
Nombre del Evaluador: Jonathan Esaú Magaña Rodríguez		
Nombre del Evaluado: José Alexander Rodríguez González		
N°	Aspecto a Evaluar	Rúbrica (Nota)
1	Demostró compromiso y responsabilidad con el grupo	9
2	Demostró respeto y tolerancia hacia las opiniones de los demás	8
3	Aportó al desarrollo del proyecto	8
4	Propicia un clima de trabajo agradable	9
5	Antes de entregar la tarea, fue revisado por el evaluado	9
Total		43

Autoevaluación		
Nombre del Estudiante: Carolina Jazmín Martínez		
N°	**Aspecto a Evaluar**	**Rúbrica (Nota)**
1	Colaboré con las tareas que me fueron asignadas	10
2	Participé en forma activa en el trabajo de equipo	9
3	Mantuve comunicación con el equipo	9
4	Cumplí a tiempo con las actividades designadas	10
5	Aporté ideas de calidad	10
Total		48

Coevaluación		
Nombre del Evaluador: Carolina Jasmín Martínez		
Nombre del Evaluado: Jonathan Esaú Magaña Rodríguez		
N°	Aspecto a Evaluar	Rúbrica (Nota)
1	Demostró compromiso y responsabilidad con el grupo	10
2	Demostró respeto y tolerancia hacia las opiniones de los demás	10
3	Aportó al desarrollo del proyecto	10
4	Propicia un clima de trabajo agradable	10
5	Antes de entregar la tarea, fue revisado por el evaluado	10
Total		50

Código fuente

```
from PIL import Image

import turtle

import time

import random

import tkinter as tk

from tkinter import messagebox


# Redimensionar las imágenes antes de cargarlas en el juego

def redimensionar_imagen(ruta_imagen, nuevo_ancho, nuevo_alto):

    imagen = Image.open(ruta_imagen)

    imagen = imagen.resize((nuevo_ancho, nuevo_alto))

    imagen.save(ruta_imagen)


# Redimensionar las imágenes '

redimensionar_imagen("serpiente (1).gif", 30, 30)

redimensionar_imagen("serpiente (2).gif", 30, 30)

redimensionar_imagen("manzana(1).gif", 30, 30)


posponer = 0.1


# Marcador

puntuación = 0

puntuación_alta = 0
```

```
# Configuración de la ventana
wn = turtle.Screen()
wn.title("Juego Snake")
wn.setup(width=800, height=600)
wn.bgcolor('black')

# Cargar imágenes
try:
    wn.addshape("serpiente (1).gif")
    wn.addshape("serpiente (2).gif")
    wn.addshape("manzana(1).gif")
except turtle.TurtleGraphicsError:
    exit()

cabeza = turtle.Turtle()
cabeza.shape('serpiente (1).gif')
cabeza.penup()
cabeza.goto(0, 0)
cabeza.direction = "stop"

# Comida
comida = turtle.Turtle()
comida.speed(0)
```

```
comida.shape("manzana(1).gif")
comida.penup()
comida.goto(0, 100)
comida.direction = "stop"

# Segmentos
segmentos = []

# Texto
texto = turtle.Turtle()
texto.speed(0)
texto.color("white")
texto.penup()
texto.hideturtle()
texto.goto(0, 260)
texto.write("Puntuación: 0     Puntuación alta: 0", align="center", font=("Courier", 24, "normal"))

# Movimiento de la serpiente
def mov():
    if cabeza.direction == "up":
        y = cabeza.ycor()
        cabeza.sety(y + 15)

    if cabeza.direction == "down":
```

```
        y = cabeza.ycor()
        cabeza.sety(y - 15)

    if cabeza.direction == "left":
        x = cabeza.xcor()
        cabeza.setx(x - 15)

    if cabeza.direction == "right":
        x = cabeza.xcor()
        cabeza.setx(x + 15)

# Funciones de dirección
def arriba():
        cabeza.direction = "up"

def abajo():
        cabeza.direction = "down"

def izquierda():
        cabeza.direction = "left"

def derecha():
        cabeza.direction = "right"
```

 # Teclado

```
wn.listen()
wn.onkeypress(arriba, "Up")
wn.onkeypress(abajo, "Down")
wn.onkeypress(izquierda, "Left")
wn.onkeypress(derecha, "Right")
```

 # Función para reiniciar el juego

```
def reiniciar_juego():
    global puntuación
    time.sleep(1)
    cabeza.goto(0, 0)
    cabeza.direction = "stop"

    for segmento in segmentos:
        segmento.goto(1000, 1000)

    segmentos.clear()

    puntuación = 0
    texto.clear()
```

```
    texto.write(f'Puntuación: {puntuación}          Puntuación alta: {puntuación_alta}', align="center", font=("Courier", 24, "normal"))


# Función para reiniciar o salir
def manejar_colision():
    root = tk.Tk()
    root.withdraw()
    respuesta = messagebox.askretrycancel("Game Over", "¡Has perdido! ¿Quieres reiniciar el juego o salir?\n\nReintentar: \nCancelar:")
    root.destroy()

    if respuesta:
        reiniciar_juego()
    else:
        wn.bye()

# Bucle principal del juego
while True:
    wn.update()

    # Colisiones con el borde de la ventana
    if cabeza.xcor() > 380 or cabeza.xcor() < -380 or cabeza.ycor() > 280 or cabeza.ycor() < -280:
        manejar_colision()
        continue
```

```
    # Colisiones con la comida
    if cabeza.distance(comida) < 20:
        x = random.randint(-380, 380)
        y = random.randint(-280, 280)
        comida.goto(x, y)

        nuevo_segmento = turtle.Turtle()
        nuevo_segmento.speed(0)
        nuevo_segmento.shape("serpiente (2).gif")
        nuevo_segmento.penup()
        segmentos.append(nuevo_segmento)

        # Aumenta la puntuación
        puntuación += 1
        if puntuación > puntuación_alta:
            puntuación_alta = puntuación

        texto.clear()
        texto.write(f'Puntuación: {puntuación}  Puntuación alta: {puntuación_alta}', align="center", font=("Impact", 24,))

    # Movimiento del cuerpo de la serpiente
    totalSeg = len(segmentos)
    for index in range(totalSeg - 1, 0, -1):
```

```
    x = segmentos[index - 1].xcor()
    y = segmentos[index - 1].ycor()
    segmentos[index].goto(x, y)

if totalSeg > 0:
    x = cabeza.xcor()
    y = cabeza.ycor()
    segmentos[0].goto(x, y)

mov()

# Colisiones con el cuerpo
for segmento in segmentos:
    if segmento.distance(cabeza) < 15:
        time.sleep
        cabeza.goto(0,0)
        cabeza.direction = "stop"
        manejar_colision()
        continue

        #Esconder los segmentos
        for segmento in segmentos:
            segmento.goto(1000,1000)
```

```
        segmentos.clear()


        puntuación = 0
        texto.clear()
        texto.write(f'Puntuación {puntuación}     Puntuación alta: {Puntuación_alta}',align = "center", font
=("Impact", 24,))


    time.sleep(posponer)


cabeza.speed(0)


turtle.done()
```

Enlace de carpeta juego snake

https://drive.google.com/drive/folders/1y2AR_6i_sKhe4-INj-XDlIj11GcykCWz?usp=sharing