**Segundo proyecto de Taller de Programación**

**Instituto Tecnológico de Costa Rica**

**Sede San Carlos**

**Unidad de Computacion**

**Proyecto:**

"Virus Spread Challenge"

**Estudiantes:**

Andrés Valerio Salas

Jean Carlos Huertas Piedra

Mauricio Barrantes Lopéz

**Profesor:**

José Leonardo Viquez Acuña

**Fecha de inicio:**

29/04/25

**Fecha de entrega:**

20/05/25

## INTRODUCCIÓN: SOBRE USO DE MATRICES Y ARCHIVOS BINARIOS.

En el desarrollo de videojuegos, una correcta gestión de los datos y una representación eficiente del entorno de juego son esenciales para garantizar una experiencia fluida y dinámica. En este sentido, los conceptos de matrices y archivos binarios juegan un papel crucial en la organización, manipulación y almacenamiento de los datos del juego. Esta introducción aborda cómo se aplican estos dos principios en la creación de un juego en el que un virus se mueve aleatoriamente a través de un tablero representado por una matriz, y el jugador debe bloquear su avance con muros. Además, se explica el uso de archivos binarios para guardar el estado del juego y permitir la persistencia de los datos entre sesiones.

En este proyecto, la matriz se utiliza como la estructura de datos central que representa el tablero del juego. Cada celda de la matriz corresponde a un espacio en el juego que puede estar vacío, ocupado por un muro o por un virus. Las matrices, al ser estructuras bidimensionales, permiten una representación clara y ordenada del espacio de juego, donde el virus puede moverse aleatoriamente y el jugador puede colocar muros para bloquear su avance. Este enfoque facilita la manipulación eficiente de los objetos dentro del juego, ya que se pueden realizar operaciones sobre la matriz para actualizar el estado del juego, como mover el virus o colocar un muro en una celda específica. Además, las matrices permiten un acceso rápido y directo a cada celda, lo que optimiza el rendimiento del juego.

El uso de archivos binarios es fundamental para garantizar la persistencia del estado del juego. En lugar de utilizar archivos de texto, que suelen ocupar más espacio y ser más lentos de procesar, los archivos binarios almacenan la información de manera más eficiente en términos de espacio y tiempo de acceso. Este tipo de archivo permite guardar datos complejos, como el estado de la matriz, el nivel del juego y otros parámetros relevantes, de manera compacta y rápida. Los archivos binarios son ideales para juegos que requieren un rendimiento constante, ya que permiten leer y escribir datos con mayor rapidez. En este proyecto, se utiliza un sistema de guardado y carga de partidas que permite al jugador almacenar su progreso en diferentes ranuras y continuar la partida desde el mismo punto en el que la dejó. De este modo, los archivos binarios permiten una experiencia más fluida, ya que el jugador puede reanudar su juego en cualquier momento sin perder su avance (Python Guides, 2023).

La combinación de matrices y archivos binarios en este proyecto permite una gestión eficiente tanto de los datos del juego como de su almacenamiento. A continuación, el análisis del problema donde se estudia cómo solucionar problemas generales que podrían surgir de la realización del presente proyecto.

## ANÁLISIS DEL PROBLEMA.

En este proyecto, los estudiantes desarrollarán un juego relacionado a la propagación de un virus dentro de una matriz, enfrentándose a desafíos relacionados con la estrategia de bloqueo del avance del virus. La naturaleza del juego implica una combinación de técnicas fundamentales de

programación, como el manejo de matrices, la manipulación de archivos binarios y el diseño de interfaces gráficas. Este análisis tiene como propósito identificar los aspectos clave del problema, describir las dificultades inherentes y explorar las estrategias que los estudiantes deberán considerar al abordar cada uno de esos desafíos.

*Simulación de la Propagación del Virus:*

En el núcleo del juego, el virus se desplaza de una celda infectada a una celda adyacente en la matriz, propagándose en una de las cuatro direcciones posibles: arriba, abajo, izquierda o derecha. La lógica que rige este comportamiento plantea varios retos, sobre todo en términos de la aleatoriedad del movimiento del virus y su propagación controlada. El virus puede ocupar una celda libre en la matriz, pero su avance puede ser detenido mediante la colocación de barreras por parte del jugador.

En este contexto, los estudiantes deben implementar algoritmos que gestionen de manera eficiente los movimientos del virus y permitan verificar si una celda está bloqueada de manera que sea irremediable para el avance del virus.

*Desafíos de la Colocación de Barreras:*

El jugador tiene la capacidad de colocar barreras para detener el avance del virus, pero existen restricciones claras sobre cómo deben colocarse: solo se puede colocar una barrera por turno, y esta debe estar en una celda libre. Sin embargo, una de las reglas más importantes es la de impedir la creación de islas. Esto significa que el jugador no puede bloquear áreas de la matriz que sean completamente inaccesibles para el virus, lo que podría hacer que el juego se volviera injusto.

Para evitar esta situación, los estudiantes deberán programar una lógica que valide si la colocación de una barrera afectará la conectividad entre las celdas. Una estrategia para lograr esto podría ser el uso de un algoritmo que simule la propagación del virus desde sus puntos de infección para verificar que todas las celdas siguen siendo alcanzables.

*Almacenamiento de Datos y Archivos Binarios:*

Una parte clave del proyecto es la implementación de un sistema de almacenamiento para guardar el estado del juego en archivos binarios. La matriz de juego se debe guardar utilizando una codificación especial, donde cada fila de la matriz se convierte a base 3 (libre = 0, virus = 1, barrera = 2), y luego ese número se convierte en un valor hexadecimal. Este enfoque de almacenamiento binario tiene ventajas significativas en cuanto a la eficiencia y el manejo de los datos, permitiendo que el juego sea guardado y cargado rápidamente, con el mínimo uso de espacio.

Aunque la codificación en base 3 y su conversión a hexadecimal es relativamente simple, la dificultad radica en la correcta estructuración de la información para que sea fácilmente accesible

cuando se cargue una partida. Una estrategia clave aquí será la correcta implementación de funciones que guarden y carguen el estado del juego sin corromper los datos y permitiendo la posibilidad de guardar y continuar múltiples partidas en diferentes ranuras.

*Sistema de Niveles y Escalabilidad del Juego:*

El juego tiene niveles de dificultad que van desde el nivel 1, con un solo punto de infección, hasta niveles más altos con múltiples puntos de infección. Cada nivel añade complejidad al juego, lo que incrementa el desafío tanto para el jugador como para los desarrolladores al crear un sistema que gestione estos cambios de manera dinámica. El sistema de niveles debe evolucionar para ofrecer desafíos progresivos y mantener el interés del jugador.

## SOLUCIÓN DEL PROBLEMA.

Las soluciones para abordar el proyecto incluyen varios aspectos clave y técnicas de programacion como matrices, aleatorización, validación de estados y almacenamiento en archivos binarios.

*Representación y Gestión del Entorno mediante Matrices:*

La matriz bidimensional de NxN es utilizada para representar el campo de juego, donde cada celda tiene un valor que indica si está libre (0), ocupada por un virus (3) o bloqueada por una barrera (2). Este enfoque de matriz facilita la manipulación eficiente de los datos, ya que permite acceder y modificar las celdas de forma rápida durante la simulación del juego.

La función crear_matriz(filas, columnas, elemento) permite inicializar la matriz con un tamaño determinado por el jugador, donde el valor predeterminado de cada celda es None o un valor específico proporcionado como argumento. Esta estructura es esencial para gestionar la disposición de los virus y barreras a medida que avanza el juego.

*Colocación Aleatoria de los Virus:*

La función agregar_virus(m, cantidad, nivel) simula la aparición de virus en posiciones aleatorias dentro de la matriz. En cada nivel del juego, la cantidad de virus aumenta progresivamente, lo que introduce una dificultad adicional.

El proceso para agregar los virus implica los siguientes pasos:

- Se identifica la lista de celdas disponibles (es decir, aquellas que están libres, representadas por el valor 0).
- Según el nivel, la cantidad de virus que aparecerán se ajusta: en el nivel 1 aparece 1 virus, en el nivel 2 aparecen 2 virus, y en el nivel 3 aparecen 3 virus.
- Se seleccionan posiciones aleatorias de la lista de celdas libres y se colocan los virus en esas ubicaciones.

Este proceso se lleva a cabo mediante el uso de la función r.sample(), que selecciona de manera aleatoria las posiciones de los virus, garantizando que el juego sea impredecible y dinámico.

*Propagación del Virus:*

La propagación del virus es el núcleo de la dinámica del juego. La función avanzar_virus(matriz) simula el movimiento del virus desde su celda actual hacia una de las celdas adyacentes. Para esto, se consideran solo las celdas válidas para la expansión, es decir, aquellas que están libres y dentro de los límites de la matriz. La función obtener_vecinos_validos(m, f, c) ayuda a determinar estas celdas vecinas válidas para la propagación, asegurando que el virus solo se mueva a lugares que no estén bloqueados por barreras ni fuera de los límites del tablero.

Cada vez que el virus avanza, se elige una celda vecina aleatoriamente usando la función r.choice(), que asegura que el virus se propague de manera impredecible, lo que añade un elemento de desafío al juego. La propagación del virus se detiene en cada turno después de que se haya realizado el movimiento, lo que permite al jugador colocar barreras y controlar el avance del virus.

*Almacenamiento y Carga de Partidas usando Archivos Binarios:*

El estado del juego se guarda y carga utilizando archivos binarios. La función guardar_partida(nombre_archivo, matriz, nivel) almacena el estado actual del juego en un archivo con una codificación especial. La matriz se guarda en formato binario, y cada fila se convierte a base 3 antes de ser codificada en hexadecimal para su almacenamiento eficiente.

Al cargar una partida con cargar_partida(nombre_archivo), se lee el archivo binario y se reconstruye la matriz y el nivel del jugador. Este enfoque permite que el juego sea fácilmente guardado y continuado en futuras sesiones, proporcionando una experiencia de juego continua y persistente.

## ANÁLISIS DE RESULTADOS.

| TAREA | ESTADO | OBSERVACIONES |
|---|---|---|
| Interfaz grafica | Completo | - |
| Propagacion de virus | Completo | - |
| Colocacion de barreras | Completo | - |
| Guardado y carga de partidas | Completo | - |
| Prevencion de islas | - | No se implementó. |
| Sistema de niveles | Completo | - |
| Multijugador local | - | No se implementó debido a falta de tiempo |
| Documentacion externa | Completo | - |

**PARTICIPACIÓN EN RESULTADOS.**

| ACTIVIDAD | JEAN | ANDRÉS | MAURICIO |
|---|---|---|---|
| Interfaz gráfica. | Realizado en totalidad (RT) | - | - |
| Propagación de virus. | Adaptación a main.py e interfaz gráfica del código. | Creación de la lógica de la propagación. | - |
| Colocación de barreras. | Adaptado. | Creado. | - |
| Sistema de niveles | - | Creado junto a la def agregar_virus | - |
| Aplicacion y utilizacion de archivos binarios | - | - | RT |
| Multijugador (tarea extra) | - | - | - |
| Documentacion interna | Documentó su parte (main.py) | Documentó su parte (virus.py) | Documentó su parte (multijugador.py y guardado) |
| Arreglos generales y pruebas | Participó de los mantenimientos finales y sirvió de lider en el proceso de programacion y arreglos finales. | Participó “” | Participó “” |
| Documento externo | - | RT | Colaboración con información sobre archivos binarios. |

## CONCLUSIONES.

El proyecto "Virus Spread Challenge" ha permitido la integración de conceptos clave en programación, como el uso de matrices, la manipulación de datos mediante archivos binarios, y la simulación de comportamientos aleatorios dentro de un entorno controlado. La solución a los problemas presentados, como la representación del juego, la propagación del virus y la colocación de barreras, ha sido eficaz gracias a la implementación de técnicas de programación orientadas a la eficiencia en el uso de memoria y el control del flujo del juego.

La utilización de matrices como la estructura principal para representar el campo de juego ha facilitado la organización y manipulación de los elementos del juego. Esto ha permitido gestionar de manera eficiente la propagación del virus y la colocación de barreras, elementos clave para la jugabilidad. La codificación de la matriz en base 3 y su almacenamiento en archivos binarios ha optimizado el uso del espacio de almacenamiento, asegurando que los datos se guarden de manera compacta y eficiente sin sacrificar la velocidad de carga y guardado.

En términos generales, el proyecto ha logrado combinar conceptos de programación fundamentales con una estructura de juego que fomenta la toma de decisiones estratégicas y la optimización de recursos, permitiendo a los estudiantes experimentar con técnicas avanzadas de programación en un contexto práctico y educativo.

## RECOMENDACIONES.

Algunas mejoras accesibles que mejorarían la experiencia del juego:

- Optimización de la Propagación del Virus: Aunque el avance del virus se realiza de manera aleatoria, se recomienda explorar métodos más complejos para la propagación, como la propagación en múltiples direcciones o la implementación de diferentes velocidades de propagación según la cercanía del virus al borde de la matriz. Esto añadiría un nivel adicional de desafío al juego.
- Escalabilidad del Juego: Se sugiere agregar un sistema de puntuación basado en la rapidez con la que el jugador detiene la propagación del virus, lo que podría fomentar la competitividad y la rejugabilidad.
- Mejoras en la Interfaz Gráfica: Si bien el proyecto utiliza herramientas como PyQt para la interfaz gráfica, se recomienda explorar la posibilidad de agregar elementos visuales más sofisticados, como animaciones del virus o efectos visuales al colocar barreras, para mejorar la experiencia del usuario y hacer el juego más atractivo.
- Persistencia de datos mejorada: Para mejorar la experiencia del jugador, se puede considerar agregar la posibilidad de guardar más detalles sobre la partida, como el tiempo transcurrido, el número de barreras utilizadas o la cantidad de virus eliminados. Esto podría ser útil no solo para la carga de partidas, sino también para ofrecer estadísticas que enriquezcan la jugabilidad.

## REFERENCIAS.

Python Guides. (2023). *Read a binary file into a byte array in Python*. Recuperado de https://pythonguides.com/read-a-binary-file-into-a-byte-array-in-python/