Juego de la Vida Proyecto Programación I

1st Sergio Alejandro Sagastume González
Facultad de Ingeniería
Universidad Mesoamericana
Sede Quetzaltenango, Guatemala, C.A
carné 202508005
sergiosagastume4@umes.edu.gt

2nd Annelís Juany Sacalxot Chojolán
Facultad de Ingeniería
Universidad Mesoamericana
Sede Quetzaltenango, Guatemala C.A.
carné 202508040
annelisjuany_sacalxotchojolan@umes.edu.gt

***Abstract*—Este proyecto presenta el desarrollo del "Juego de la Vida" por John Conway, el cual representa una simulación de células en un cuadro bidimensional. de 20x40 de manera predeterminada, y elegir entre cuatro patrones iniciales: Glider, Blinker, Toad y Oscillator, todos representativos de comportamientos clásicos en autómatas celulares. La aplicación utiliza matrices estaticas, ciclos de control y estructuras condicionales para evaluar el estado de cada célula en función de sus vecinos, aplicando las reglas clásicas de nacimiento, supervivencia y muerte. El juego se manejo en consola por lo que los símbolos utilizados son represantaciones de las celulas, creando una experiencia visual y dinámica.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introducción

El "Juego de la Vida", propuesto por John Conway en 1970, es un autómata celular que simula la evolución de patrones biológicos simples en un entorno discreto. Su funcionamiento se basa en reglas locales que determinan el estado futuro de cada célula en una cuadrícula bidimensional, dependiendo del estado de sus vecinas. A pesar de su simplicidad, el sistema puede generar comportamientos complejos.

Este trabajo presenta una implementación del Juego de la Vida en lenguaje C, la cual incorpora un menú interactivo que permite al usuario seleccionar entre cuatro patrones iniciales predefinidos: *Oscillator*, *Glider*, *Blinker* y *Toad*. Estos patrones representan estructuras comunes dentro del sistema, cada una con características evolutivas particulares, tales como movimiento, oscilación o estabilidad.

La aplicación fue desarrollada con un enfoque modular, priorizando la claridad en la lógica de evolución de las generaciones y la manipulación eficiente de la matriz de estados. Este enfoque permite al usuario observar el comportamiento del sistema a través de múltiples iteraciones y comprender la dinámica de los patrones.

II. Diseño del Juego

A. Generalidades

La simulación del Juego de la Vida fue implementada utilizando una matriz bidimensional de tamaño 40×20, en la cual cada celda representa el estado de una célula (viva o muerta). Esta matriz se visualiza en consola dentro de un marco delimitador construido con caracteres ASCII, lo que permite al usuario identificar claramente los límites del entorno simulado y proporciona una representación visual ordenada del sistema celular.

Debajo del marco de la cuadrícula, se incorporó un contador informativo que muestra en tiempo real la cantidad de células vivas, células muertas y el número de generación actual. Esta información permite al usuario seguir de forma clara la evolución del sistema y observar patrones recurrentes o cambios significativos en el comportamiento celular a lo largo de las generaciones.

La evolución del sistema se ejecuta en un bucle que actualiza la matriz en función de las reglas del Juego de la Vida, con un intervalo de tiempo entre generaciones para facilitar la visualización. El usuario puede pausar la simulación presionando la tecla `Enter`, momento en el cual se despliega un menú interactivo que le permite decidir si desea regresar al menú principal de selección de patrones o continuar hacia una pantalla de créditos. En esta última sección se presentan los nombres de los desarrolladores del programa, reconociendo su contribución al proyecto.

III. DESARROLLO E IMPLEMENTACIÓN DETALLADA DEL CODIGO EN C

A continuacion se presenta de manera detallada todas las funciones (modulos) que componen el programa del Juego de la Vida en

A. Librerías y Definiciones Utilizadas

El desarrollo del programa se apoyó en diversas librerías estándar del lenguaje C, junto con definiciones que establecen el tamaño de la matriz del juego. A continuación se muestra el bloque correspondiente de código fuente:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include <string.h>
#include <conio.h>
#include <windows.h>

#define GAME_WIDTH  40
#define GAME_HEIGHT 20
```

Identify applicable funding agency here. If none, delete this.

Estas librerías permiten realizar operaciones de entrada/salida, manipulación de memoria, manejo de tiempo y entrada por teclado, así como la utilización de funciones específicas para sistemas Windows, como el control del cursor y la limpieza de pantalla.

B. Variables Globales

El programa utiliza algunas variables globales que permiten controlar el estado de la simulación y su presentación en pantalla. A continuación se muestra la declaración de dichas variables:

```
static bool generationColorToggle = false;
static int generation = 0;
```

La variable `generationColorToggle` permite alternar entre distintos colores en la visualización por generaciones, mientras que `generation` lleva el conteo de las generaciones simuladas desde el inicio del patrón seleccionado.

C. Control de Posición del Cursor en Consola

Para facilitar la impresión controlada de los elementos gráficos dentro de la consola, se implementó la función `gotoxy`, que permite mover el cursor a una posición específica de la pantalla utilizando coordenadas `(x, y)`. Esto es fundamental para actualizar únicamente las áreas necesarias del marco sin redibujar toda la interfaz.

```
void gotoxy(int x, int y) {
   COORD coord;
   coord.X = x;
   coord.Y = y;
   SetConsoleCursorPosition(
      GetStdHandle(STD_OUTPUT_HANDLE), coord);
}
```

La función utiliza estructuras y funciones proporcionadas por la API de Windows (`windows.h`), lo cual limita su portabilidad a sistemas operativos compatibles con dicha interfaz.

D. Visualización de Créditos Finales

Al finalizar la simulación, se presenta una animación que muestra los nombres de los desarrolladores del programa, alternando colores de fondo y desplazando el texto verticalmente en la consola. Esto proporciona un cierre visual animado al juego.

```
void mostrarSalida() {
   const char* linea1 =
      "Sergio Alejandro Sagastume Gonzalez "
      "Programacion I 202508005";
   const char* linea2 =
      "Annelis Juany Sacalxot Chojolan "
      "Programacion I 202508040";
   int y = 0;
   int alternadorColor = 0;

   while (y < 26) {
      if (alternadorColor % 2 == 0)
         system("color 5F");
      else
         system("color 2F");
      system("cls");
      gotoxy(10, y);
      printf("%s", linea1);
      gotoxy(10, y + 2);
      printf("%s", linea2);

      Sleep(200);
      y++;
      alternadorColor++;
   }

   exit(0); // Termina el programa
}
```

E. Evaluación del Estado de las Células

La función `isAlive` determina si una célula estará viva o muerta en la siguiente generación del Juego de la Vida. Se basa en las reglas de Conway, evaluando las ocho celdas vecinas de una posición dada en la matriz del juego.

```
bool isAlive(int game[GAME_WIDTH][GAME_HEIGHT
    ],
          int x, int y) {
   int alive = 0;
   for (int dx = -1; dx <= 1; dx++) {
      for (int dy = -1; dy <= 1; dy++) {
         if (dx == 0 && dy == 0) continue;
         int nx = x + dx;
         int ny = y + dy;

         if (nx >= 0 && nx < GAME_WIDTH &&
            ny >= 0 && ny < GAME_HEIGHT) {
            if (game[nx][ny] == 1) alive++;
         }
      }
   }

   if (game[x][y] == 1)
      return (alive == 2 || alive == 3);
   else
      return (alive == 3);
}
```

F. Dibujo y Visualización del Estado del Juego

La función `draw` se encarga de imprimir el estado actual de la matriz de células en la consola. Presenta un marco delimitador usando caracteres ASCII, colorea las células vivas alternando colores por generación y muestra un resumen con conteos y la generación actual.

```
void draw(int game[GAME_WIDTH][GAME_HEIGHT]) {
   system("cls");

   const char* color = generationColorToggle
                  ? "\x1b[35m" : "\x1b[34m";

   int totalAlive = 0;

   for (int x = 0; x < GAME_WIDTH; ++x) {
      for (int y = 0; y < GAME_HEIGHT; ++y) {
         if (game[x][y]) totalAlive++;
      }
   }
```

```
    int totalCells = GAME_WIDTH * GAME_HEIGHT;
    int totalDead = totalCells - totalAlive;

    putchar(201);
    for (int x = 0; x < GAME_WIDTH; ++x)
        putchar(205);
    putchar(187);
    printf("\n");

    for (int y = 0; y < GAME_HEIGHT; ++y) {
        putchar(186);
        for (int x = 0; x < GAME_WIDTH; ++x) {
            if (game[x][y])
                printf("%s%c\x1b[0m", color, 254);
            else
                putchar(' ');
        }
        putchar(186);
        putchar('\n');
    }

    putchar(200);
    for (int x = 0; x < GAME_WIDTH; ++x)
        putchar(205);
    putchar(188);
    putchar('\n');

    printf("\n\x1b[33mResumen:\x1b[0m\n");
    printf(" Celulas vivas : %d\n", totalAlive)
        ;
    printf(" Celulas muertas: %d\n", totalDead)
        ;
    printf(" Generacion : %d\n", ++generation);

    generationColorToggle = !
        generationColorToggle;
}
```

G. Inserción de Patrones Clásicos en el Juego de la Vida

Para inicializar configuraciones típicas del Juego de la Vida, se implementaron funciones que insertan patrones conocidos en posiciones específicas de la cuadrícula. Estos patrones son fundamentales para el estudio y demostración del comportamiento dinámico del sistema.

```
void insertGlider(int grid[GAME_WIDTH][
    GAME_HEIGHT], int x, int y) {
    if (x + 2 < GAME_WIDTH && y + 2 <
        GAME_HEIGHT) {
        grid[x + 1][y] = 1;
        grid[x + 2][y + 1] = 1;
        grid[x][y + 2] = 1;
        grid[x + 1][y + 2] = 1;
        grid[x + 2][y + 2] = 1;
    }
}

void insertBlinker(int grid[GAME_WIDTH][
    GAME_HEIGHT], int x, int y) {
    if (x + 2 < GAME_WIDTH && y < GAME_HEIGHT)
        {
        grid[x][y] = 1;
        grid[x + 1][y] = 1;
        grid[x + 2][y] = 1;
    }
}

void insertToad(int grid[GAME_WIDTH][
    GAME_HEIGHT], int x, int y) {
    if (x + 3 < GAME_WIDTH && y + 1 <
        GAME_HEIGHT) {
        grid[x + 1][y] = 1;
        grid[x + 2][y] = 1;
        grid[x + 3][y] = 1;
        grid[x][y + 1] = 1;
        grid[x + 1][y + 1] = 1;
        grid[x + 2][y + 1] = 1;
    }
}
```

H. Título Intermitente para la Selección de Patrones

La función `printBlinkingTitle` muestra un mensaje intermitente en la consola para destacar la sección de selección de patrones iniciales. Alterna el color del texto entre rojo y azul, utilizando secuencias de escape ANSI y limpiando la pantalla en cada iteración para crear el efecto parpadeante.

```
void printBlinkingTitle() {
    const char* red = "\x1b[31m";
    const char* blue = "\x1b[34m";
    const char* reset = "\x1b[0m";

    for (int i = 0; i < 6; i++) {
        system("cls");
        if (i % 2 == 0)
            printf("%sSelecciona un patron
                inicial:%s\n\n", red, reset);
        else
            printf("%sSelecciona un patron
                inicial:%s\n\n", blue, reset);
        Sleep(300);
    }
}
```

I. Función Principal y Control del Flujo del Juego

La función `main` es el punto de entrada del programa y controla el flujo principal del juego de la vida. Implementa un ciclo infinito que presenta un menú para que el usuario seleccione un patrón inicial, genera un estado aleatorio de células, inserta patrones específicos según la opción, y ejecuta la simulación iterativa mostrando las generaciones en la consola. La simulación avanza automáticamente, actualizando la visualización cada medio segundo, y permite salir a través de la tecla Enter para regresar al menú o terminar el programa.

```
int main() {
    while (1) {
        int display[GAME_WIDTH][GAME_HEIGHT] =
            {0};
        int swap[GAME_WIDTH][GAME_HEIGHT] = {0};
        int opcion;

        printBlinkingTitle();
        printf("1. Oscillator (solo aleatorio)\n
            ");
        printf("2. Glider\n");
        printf("3. Blinker\n");
```

```
        printf("4. Toad\n");
        printf("Opcion: ");
        scanf("%d", &opcion);
        while (getchar() != '\n');

        srand((unsigned int)time(NULL));
        for (int i = 0; i < GAME_WIDTH; ++i) {
            for (int j = 0; j < GAME_HEIGHT; ++j)
                {
                display[i][j] = (rand() % 6 == 0)
                    ? 1 : 0;
            }
        }

        if (opcion == 2) {
            insertGlider(display, rand() % (
                GAME_WIDTH - 3), rand() % (
                GAME_HEIGHT - 3));
        } else if (opcion == 3) {
            insertBlinker(display, rand() % (
                GAME_WIDTH - 3), rand() %
                GAME_HEIGHT);
        } else if (opcion == 4) {
            insertToad(display, rand() % (
                GAME_WIDTH - 4), rand() % (
                GAME_HEIGHT - 2));
        }

        generation = 0;

        while (1) {
            for (int i = 0; i < GAME_WIDTH; ++i)
                {
                for (int j = 0; j < GAME_HEIGHT;
                    ++j) {
                    swap[i][j] = isAlive(display, i
                        , j) ? 1 : 0;
                }
            }

            draw(swap);
            memcpy(display, swap, sizeof(display)
                );
            Sleep(500);

            if (_kbhit() && _getch() == 13) break
                ; // Enter para salir
        }

        char volver;
        printf("\nVolver al menu? (s/n): ");
        scanf(" %c", &volver);
        while (getchar() != '\n');

        if (volver != 's' && volver != 'S') {
            system("cls");
            mostrarSalida();
        }
    }
    return 0;
}
```

J. FUNCIONAMIENTO DEL JUEGO

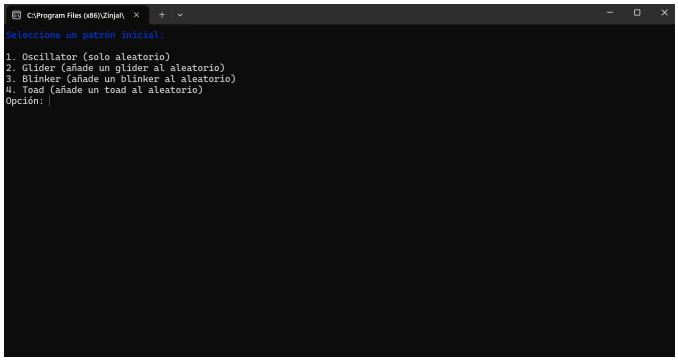

Fig. 1. Menú de selección del patrón por parte del usuario.

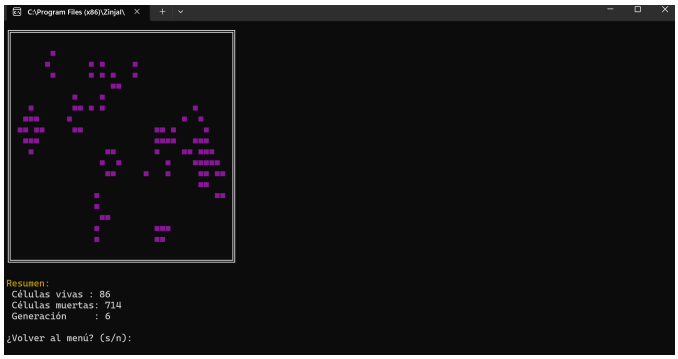

Fig. 2. Evolución de un patrón aleatorio oscilador.

Fig. 3. Ejemplo de un glider (planeador) desplazándose en la cuadrícula.

Fig. 4. Patrón Blinker: un oscilador de período 2.

Fig. 5. Patrón Toad: oscilador que alterna entre dos configuraciones.

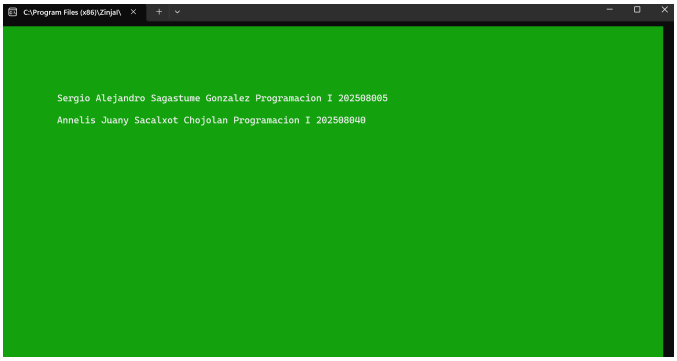

Fig. 6. Créditos del programa con fondo titilante entre verde y morado.

K. Variables utilizadas para el funcionamiento del juego

Para controlar ciertos aspectos visuales y de seguimiento dentro del programa, se utilizan las siguientes variables globales:

- `static bool generationColorToggle = false;`
  Esta variable booleana alterna su valor en cada generación con el fin de cambiar el color de las células vivas en la interfaz, lo que permite distinguir visualmente entre generaciones pares e impares.
- `static int generation = 0;`
  Esta variable entera lleva un conteo del número de generaciones transcurridas desde el inicio del juego. Se incrementa automáticamente con cada actualización del tablero.

CONCLUSIONES

El desarrollo del proyecto “Juego de la Vida” en lenguaje C representó una oportunidad efectiva para aplicar conceptos clave de la programación estructurada, la lógica algorítmica y la gestión visual en consola. A través de una implementación se logró simular un autómata celular capaz de representar el comportamiento emergente de sistemas dinámicos en una cuadrícula de 20 filas y 40 columnas.

Una de las principales fortalezas del programa es su interfaz interactiva y visual, donde se presentan generaciones sucesivas de células vivas y muertas en un entorno gráfico basado en caracteres ASCII. Este entorno permite observar claramente los cambios generacionales a partir de reglas simples pero poderosas. Además, se implementó una selección dinámica de patrones iniciales, lo que proporciona al usuario la posibilidad de elegir entre cuatro configuraciones representativas: *Glider*, *Blinker*, *Toad* y Oscillator. Cada una de estas opciones genera comportamientos particulares y refuerza el entendimiento de cómo patrones básicos pueden derivar en sistemas complejos.

Este proyecto se orienta más hacia el fortalecimiento de la lógica estructurada y el manejo visual de datos en entornos modernos, sin requerir programación a nivel de hardware o interrupciones.

Finalmente, el código entregado demuestra un dominio sólido del lenguaje C, con uso correcto de funciones, control de flujo, manipulación de arreglos bidimensionales, interacción con el usuario y animaciones por consola. Proyectos como este reafirman que la programación no solo se aprende escribiendo código, sino observando cómo este genera sistemas vivos que evolucionan en el tiempo.

VIDEO DEMOSTRATIVO

Para complementar la comprensión del proyecto Juego de la Vida de Conway en lenguaje C, se ha preparado un video demostrativo que presenta la funcionalidad general del programa, desde la selección del patrón inicial hasta la visualización dinámica de las generaciones celulares.

El video inicia con la pantalla de menú, donde el usuario puede seleccionar uno de los cuatro patrones disponibles:

1) Oscillator (aleatorio)
2) Glider
3) Blinker
4) Toad

Una vez seleccionada la opción, el sistema genera una cuadrícula de 40 columnas por 20 filas y simula automáticamente la evolución de las células vivas y muertas en dicha matriz, conforme a las reglas del autómata celular. El comportamiento se visualiza mediante una interfaz en consola, utilizando caracteres especiales para representar células vivas y muertas , alternando colores para identificar cada generación.

Durante la ejecución, el video muestra cómo los patrones se desplazan, oscilan o se estabilizan, dependiendo de su configuración inicial, lo que permite apreciar la riqueza del modelo de Conway incluso con reglas simples.

El video concluye mostrando la opción para reiniciar el programa o salir, y presenta una animación final con los créditos de los autores, alternando colores en pantalla, como parte de un toque creativo y visual que refuerza el aprendizaje técnico con elementos gráficos llamativos.

Link of the video demostrativo: https://bit.ly/3SRaZMZ