Universidad Católica Andrés Bello
Escuela de Ingeniería Informática
Arquitectura del Computador
Profesor: Ramón Porras

MAQUINA DE ESTADOS FINITA

Proyecto de Arquitectura del Computador 2020

Integrantes:

Ricardo Salvatorelli (26.967.602)
José Manuel Ramírez (26.902.002)

Caracas, 12 de Julio del 2020.

Proyecto Arquitectura del Computador 2020 – Maquina de Estados Finita

Introducción

Para la realización de este proyecto decidimos desarrollar un juego el cual consta de 8 objetos, 6 objetos llamados “inocentes” (denotados con la letra “I”), 1 objeto llamado jugador (denotado con la letra “J”) y 1 objeto llamado objetivo (denotado con la letra “O”). El juego consta de 5 estados los cuales son “No matar” (0), “Mata inocente” (1), “Mata objetivo” (2), “Gana” (3), “Pierde” (4). El objetivo del juego es que el jugador mate al objetivo (esto solo lo podrá lograr siempre y cuando se encuentre en el estado indicado) y de esa forma asegura la victoria, si el jugador se encuentra con el objetivo y no se encuentra en el estado indicado para poder “matarlo” entonces el jugador perderá.

Diagrama de estado para el jugador:

El jugador cuenta con 5 posibles estados.

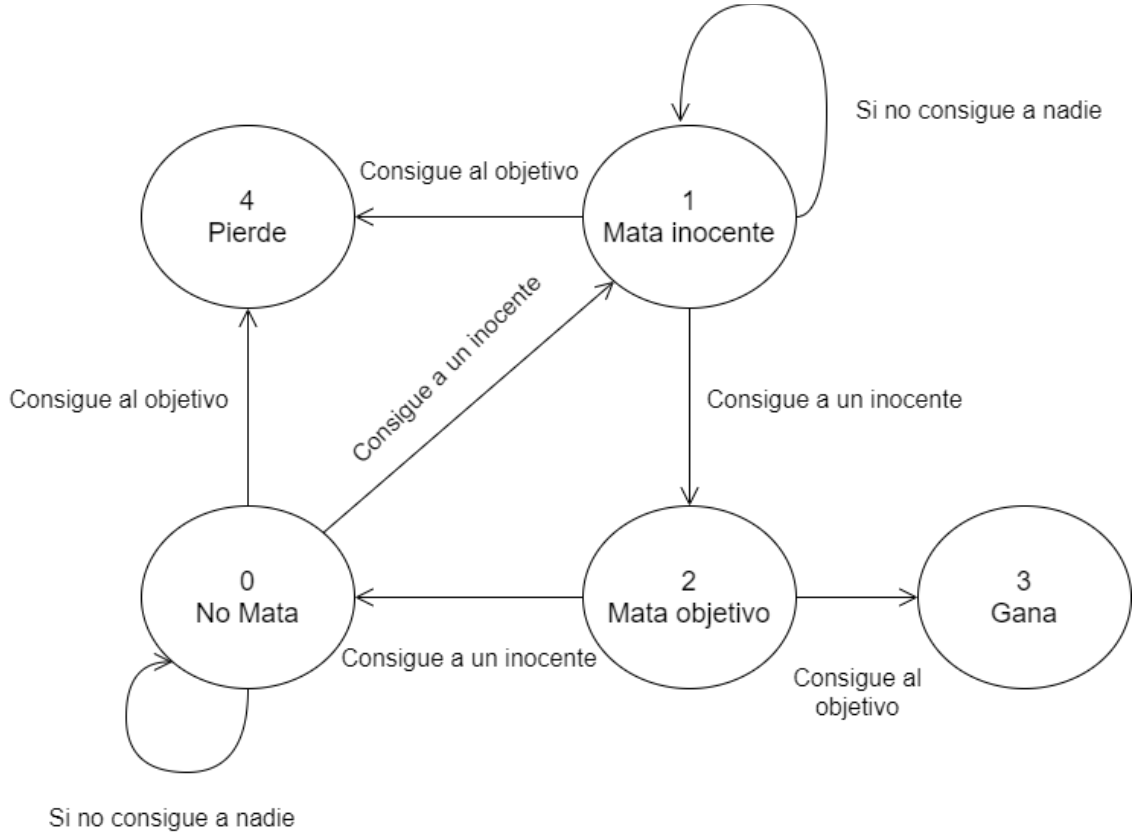

Imagen #1 Diagrama de estados

Como se puede observar en la imagen #1, los estados 4 y 3 no cuentan con transiciones, por lo tanto, se podrían considerar como estados finales.

- Si el jugador se encuentra en el estado “No Mata” el estado siguiente será “Mata inocente” en el caso de que consiga a un inocente (ya que esta es la condición de transición) o el estado siguiente sería el mismo estado “No Mata” si no consigue a nadie.

- Si el jugador se encuentra en el estado “Mata inocente” el estado siguiente será “Mata objetivo” si no consigue a nadie, será “Pierde” si consigue al objetivo o será el mismo estado “Mata inocente” si no consigue a nadie.

- Si el jugador se encuentra en el estado “Mata objetivo” el estado siguiente será “Gana” si el jugador consigue al objetivo, o será “No Mata” si el jugador consigue a un inocente.

- Si el jugador se encuentra en el estado “Gana” o en el estado “Pierde” ya no hay más estados siguientes por lo tanto se termina el juego.

Lo explicado anteriormente se resume en la tabla de estados que se presenta a continuación.

Tabla de estados:

Estado Actual / Nuevo Estado	No Mata	Mata inocente	Mata objetivo	Gana	Pierde
No mata	Si no consigue a nadie	Consigue a un inocente	-	-	Consigue al objetivo
Mata inocente	-	Si no consigue a nadie	Consigue a un inocente	-	Consigue al objetivo
Mata objetivo	Consigue a un inocente	-	-	Consigue al objetivo	-
Gana	-	-	-	-	-
Pierde	-	-	-	-	-

Tabla #1 Tabla de estados y transiciones

Desarrollo de la Máquina de estados finitos:

Usando como base el diagrama de estados y la tabla de estados procedemos a iniciar el desarrollo del programa para ello el primer paso es crear una aplicación en el lenguaje de desarrollo “C#” con el nombre de “Proyecto ADC”.

Crearemos la clase en donde desarrollaremos todo lo relacionado a los estados de la máquina de estados finitos, dicha clase se llamará “Maquina”, comenzaremos creando una estructura llamada “objeto” donde podremos crearemos una variable llamada “x” donde después se almacenará la coordenada horizontal y una variable llamada “y” donde después se almacenará la coordenada vertical de dicho objeto, también crearemos una variable denominada “activo” en donde se almacenará el estado de dicho objeto, es decir, si el objeto se encuentra activo o no, y por último creamos una variable llamada “estado” a la cual se le asignará el estado en que se encuentra dicho objeto.

```
/*
 * Declaracion de la estructura para
 * cada objeto utilizado
*/
public struct objeto
{
    public bool activo;
    public int x, y;
    public int estado;
}
```

Imagen #2 definición de la estructura “objeto”

En la misma clase “Maquina” creamos un arreglo de tipo “objeto” que llamaremos “jugs” y tendrá una longitud de 8 posiciones las cuales representarán los 8 objetos que se muestran en el programa (6 “inocentes”, 1 jugador y 1 objetivo).

```
/*
 * Clase que contendrá la lógica de los estados
 * y los movimientos de los objetos utilizados
*/
public class Maquina
{

    // Creación de los 8 objetos utilizados
    public objeto[] jugs = new objeto[8];
```

Imagen #3 creación del arreglo “jugs”

Luego seguimos trabajando en la clase “Maquina” y creamos una función para determinar el estado siguiente la cual tiene como nombre “determinarEstado” y recibe como parámetros las coordenadas “x” y “y” siguientes a donde se deberá mover el jugador.

```
public void determinarEstado(int sigx, int sigy, int i)
{
    if (jugs[0].estado == 0)
    {
        if (sigx == 800 && sigy == 400)
        {
            jugs[0].estado = 4;
            jugs[0].activo = false;
        }
        else if(jugs[i].activo == true) jugs[0].estado = 1;
    }
    else if (jugs[0].estado == 1)
    {
        if (sigx == 800 && sigy == 400)
        {
            jugs[0].estado = 4;
        }
        else
        {
            if (jugs[i].activo == true)
            {
                jugs[0].estado = 2;
                jugs[i].activo = false;
            }
        }
    }
    else
    {
        if (sigx == 800 && sigy == 400)
        {
            jugs[0].estado = 3;
            jugs[7].activo = false;
        }
        else jugs[0].estado = 0;
    }
}
```

Imagen #4 creación de la función “determinarEstado”

Continuando con el desarrollo, procedimos a crear una función llamada “moverObjetos” la cual es la encargada de mover el objeto “jugador” a las coordenadas correspondientes dependiendo del estado además en esta función se determinará el estado siguiente del jugador.

```
public void moverObjetos()
{
    Random ver = new Random(Guid.NewGuid().GetHashCode());
    Random hor = new Random(Guid.NewGuid().GetHashCode());

    int mver = ver.Next(1, 3);

    if (mver == 2)
    {
        // movimiento vertical hacia abajo
        if (jugs[0].y == 200)
        {
            for (int i = 0; i <= 7; i++)
            {
                if (jugs[i].x == jugs[0].x && jugs[i].y == 400)
                {
                    determinarEstado(jugs[i].x, jugs[i].y, i);
                    jugs[0].y = 400;
                    if(i != 7)
                        jugs[i].y = 200;
                    return;
                }
            }
        }
```

Imagen #5 creación de la función “moverObjetos”

```
        // movimiento vertical hacia arriba
        else
        {
            for (int i = 0; i <= 7; i++)
            {
                if (jugs[i].x == jugs[0].x && jugs[i].y == 200)
                {
                    determinarEstado(jugs[i].x, jugs[i].y, i);
                    jugs[0].y = 200;
                    if (i != 7)
                        jugs[i].y = 400;
                    return;
                }
            }
        }
    }
    else
    {
        int mhor = hor.Next(1, 3);
        // movimiento horizontal hacia la derecha
        if ((jugs[0].x == 200) || (jugs[0].x != 800 && mhor == 2))
        {
            for (int i = 0; i <= 7; i++)
            {
                if (jugs[0].y == jugs[i].y && jugs[0].x + 200 == jugs[i].x)
                {
                    determinarEstado(jugs[i].x, jugs[i].y, i);
                    jugs[0].x += 200;
                    if (i != 7)
                        jugs[i].x -= 200;
                    return;
                }
            }
        }
```

Imagen #6 continuación de la creación de la función “moverObjetos”

```
        // movimiento horizontal hacia la izquierda
        else if ((jugs[0].x == 800) || (jugs[0].x != 200 && mhor == 1))
        {
            for (int i = 0; i <= 7; i++)
            {
                if (jugs[0].y == jugs[i].y && jugs[0].x - 200 == jugs[i].x)
                {
                    determinarEstado(jugs[i].x, jugs[i].y, i);
                    jugs[0].x -= 200;
                    if (i != 7)
                        jugs[i].x += 200;
                    return;
                }
            }
        }

    }
}
```

Imagen #7 continuación de la creación de la función “moverObjetos”

Por último, definimos el constructor para determinar los valores iniciales que deberán tener de los 8 objetos (6 “inocentes”, 1 jugador y 1 objetivo) creados previamente, los cuales se encuentran en el arreglo llamado “jugs”, dichos valores iniciales son las coordenadas “x” y “y”, el estado respectivo (el cual será 0) y si los objetos se encuentran activos o no, en este caso el valor inicial de la variable activo para cada objeto será true, esto es así para que se muestren todos los objetos en pantalla.

```
    /*
     * Constructor de la clase, realiza la asignacion
     * de atributos iniciales de los objetos
     */
    public Maquina()
    {
        int x = 200;
        int y = 200;

        for (int i = 0; i <= 7; i++)
        {
            jugs[i].x = x;
            jugs[i].y = y;
            jugs[i].estado = 0;
            jugs[i].activo = true;

            if (i == 3)
            {
                x = 200;
                y = 400;
            }
            else x += 200;
        }
    }
```

Imagen #8 creación del constructor de la clase “Maquina”

Y creamos el método Main desde donde se inicia y hace que se muestre la interfaz del juego para poder observar todos los objetos, si están activos o no, sus coordenadas y sus estados.

```
/*
 * Punto de inicio de la aplicación,
 * se encarga de llamar a la creación de
 * la interfaz gráfica
*/
[STAThread]
static void Main()
{
    Application.Run(new Form1());
}
```

Imagen #9 creación del método Main desde donde se inicia el programa

Demostración del programa:

El juego comienza con 8 objetos de los cuales 6 son los “inocentes”, 1 es el jugador (J) y 1 es el objetivo (O). Solamente se dibujan los objetos que tengan un estado activo, es decir, que el valor de la variable activo sea “true”.

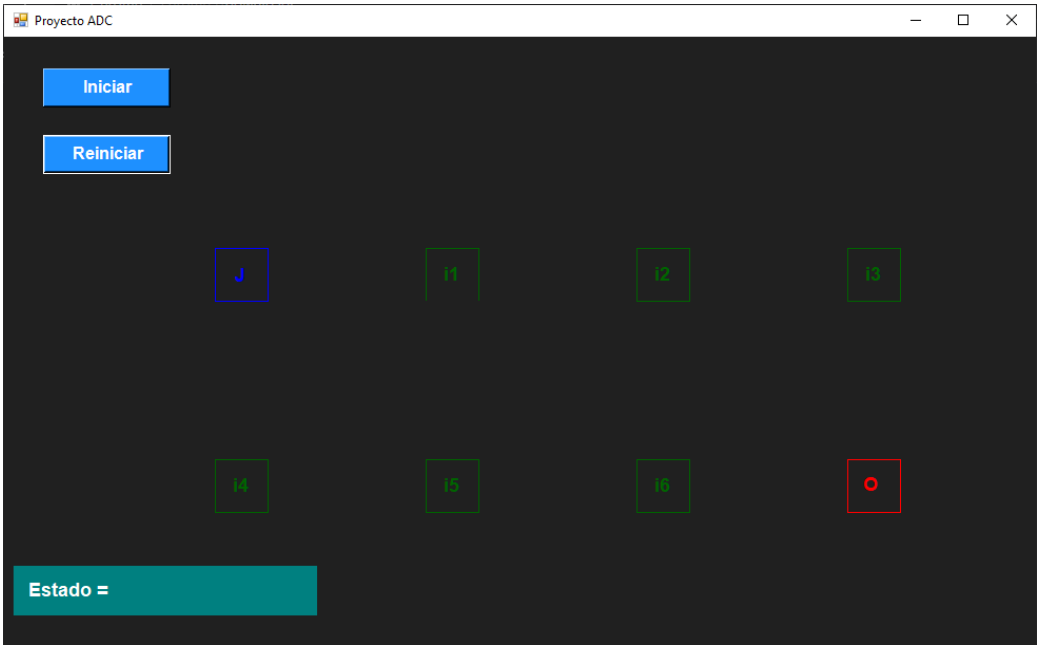

Imagen #10 inicio del programa

Al momento de seleccionar iniciar comienza el juego, mostrándose los estados en la parte inferior izquierda.

Imagen #11 Estado actual del juego

El juego va avanzando y al final se muestra el resultado del mismo, los cuales indican que el jugador ha ganado o que el jugador ha perdido.

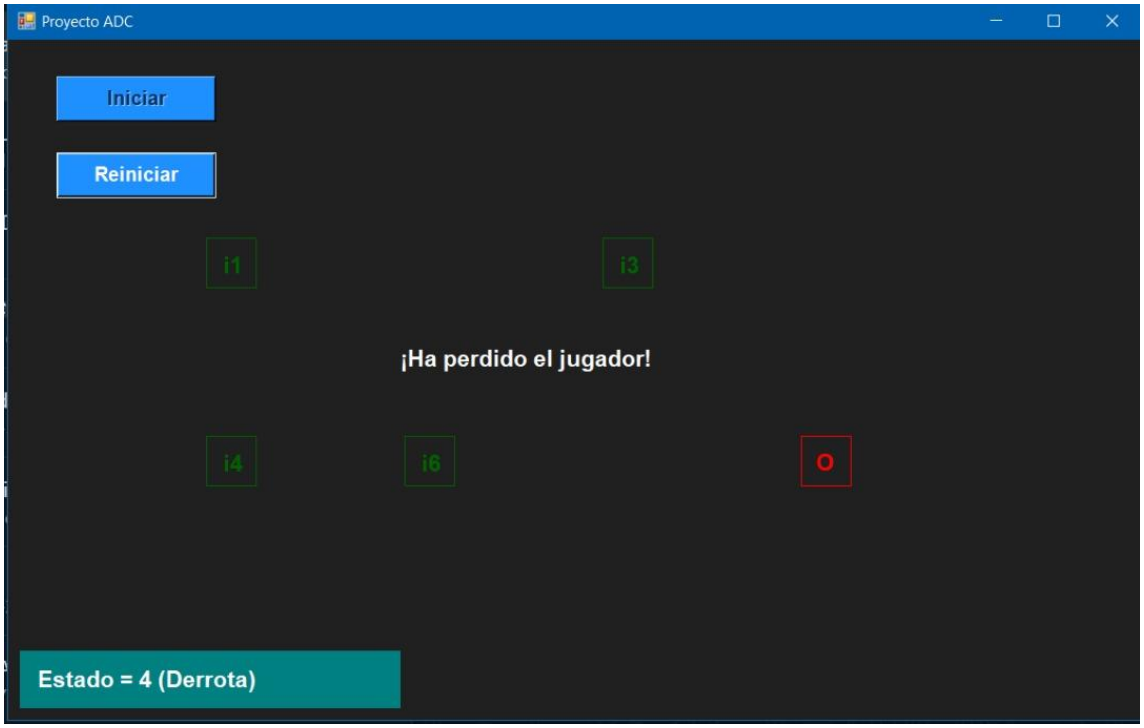

Imagen #12 Mensaje de derrota

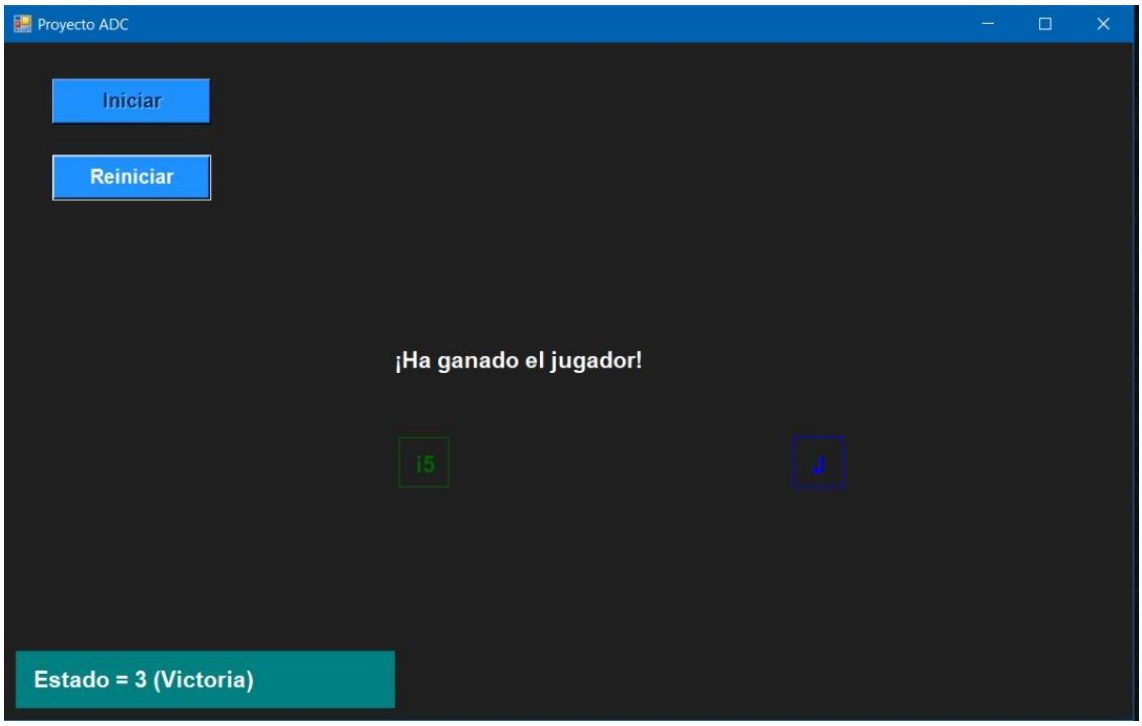

Imagen #12 Mensaje de victoria

Conclusión

Luego del desarrollo de este programa podemos concluir que el juego se comporta como una máquina de estados finitos, ya que el juego tiene una cantidad finita de estados y los estados van cambiando siempre y cuando se cumpla la condición de transición, se puede observar que cuando se llega al estado 3 (“Gana”) o al estado 4 (“Pierde”) el juego termina debido a que estos son los últimos estados y de estos estados no se puede ir a otros porque no existe una condición de transición que lo permita. Para finalizar también se observa que el resultado del juego (victoria o derrota) dependerá del estado actual a la hora de que el jugador se encuentre con el objetivo.