VISUALIZACIÓN ANIMADA DEL JUEGO DE LA VIDA

JOSÉ SEBASTIÁN IBARRA ARREGUI

ÍNDICE

INTRODUCCIÓN

El Juego de La Vida de Conway consiste en un sistema de complejidad de movimiento de “células” en una cuadrilla bidimensional, a partir de reglas muy simples que se asemejan a las de los seres vivos. Donde cada celda tomaba el estado de vivo o muerto, y ese estado dependía en cada iteración del estado de las celdas vecinas, cumpliendo un comportamiento complejo que evoluciona y puede tener una combinación de patrones. Como sistemas, pueden desencadenar e imitar nuevos comportamientos, una configuración en la que pueden haber movimientos hacia una dirección, o autómatas que permitan grandes reacciones en cadena. Existen ejemplos como naves, colisiones, puertas AND, OR, trenes, osciladores, e incluso autómatas de mayor tamaño de simulaciones de un universo entero. Más allá de todos los complejos comportamientos, el juego nos da la impresión de la vida misma al notar que somos un eslabón más dentro de una cadena de simulaciones que hacen funcionar otras simulaciones y forman el mundo entero, como una función recursiva.

El Juego de la Vida nos enseña cómo tras reglas muy simples de vecindad en las que dependen si vives o mueres en el siguiente instante, pueden ser suficientes para emerger comportamientos que simulan cómo funciona nuestra inteligencia, nuestra biología, la sociedad, la vida misma. Coincidiendo con eventos incontrolables, con el caos y con eventos que notan la facilidad en la que con un simple movimiento, se crean mayores y complejas reacciones en cadena que hacen que cambie todo.
Es por ello que el juego recibe este nombre, porque al usuario le da la reflexión de cómo la interacción con nuestro entorno más cercano crea un impacto y desencadenamiento de eventos que no podríamos predecir ni controlar a futuro. Finalmente concluyendo que con la simplicidad de crear un evento, también este puede simplemente acabar.

El proyecto del Juego de la Vida también fue escogido por tributo y en memoria al matemático John Horton Conway, creador de este juego, quien debido a la situación pandémica actual de este año, falleció por Covid-19 en el pasado mes de Abril.

ENUNCIADO DEL PROYECTO

Visualización Animada del Juego de la Vida (Conway's Game of Life).

Usted debe de implementar una aplicación desktop usando el módulo Tkinter. Está en la libertad de incluir los componentes que considere necesario. Trate que vuestro trabajo esté orientado a mostrar a los estudiantes el funcionamiento de los algoritmos de iteración y recursión. La implementación debe de usar Programación Orientada a Objetos.

La aplicación como mínimo debe de permitir:

- Iniciar el juego
- Mostrar las soluciones
- Reiniciar el juego

Figura 1. Visualización del Juego de la Vida de John Conway.

ANÁLISIS DEL EJERCICIO

Objetivos

- Permitir mediante el módulo Tkinter, realizar una implementación de interfaz gráfica del Juego de la Vida, realizar la animación y permitir el reinicio y algoritmo del juego.

- Usar efectivamente los elementos aprendidos en clase de interfaz gráfica para complementar el juego y su estructura (Labels, Buttons, Frames, Radiobuttons, StringVar, OptionMenu, IntVar, pack(), grid(), colores, etc).

- Asegurar que la implementación del algoritmo y el juego en general sea usando Programación Orientada a Objetos. Tratar de establecer varias funciones, iteraciones, y si hay posibilidad, usar recursividad.

- Aparte de lograr desarrollar el algoritmo y mostrar las soluciones, permitir que un usuario pueda manipular el juego y las células a su disposición.

- Finalmente, lograr una solución completa y sin errores de la interfaz gráfica del juego, con una animación interactiva y visualización de ejemplos de patrones de movimiento.

Análisis general

El juego de la vida resume, como fue mencionado en la introducción, la vida misma tras eventos de vida y muerte en distintos momentos tras reglas muy sencillas. La interfaz en general del juego no es complicado hasta el momento en el que se debe implementar el algoritmo y fraccionar los patrones y la continuidad de instantes en una cuadrilla de juego. El juego consta de 3 simples reglas para las llamadas “células” dentro del juego.

1. **Nacer.**

   Se debe definir los espacios vacíos como aquellos espacios donde las células nacerán y posteriormente morirán. Para nacer, un espacio vacío debe contar con 3 células vivas alrededor de él y así, en el siguiente instante nacerá una célula en ese espacio previamente vacío.

2. **Sobrevivir.**

   Una vez la célula existe, esta debe sobrevivir en los momentos posteriores, y para ello la célula debe presentar vecindad con otras 2 o 3 células vivas. Solo esa cantidad, ni más ni menos.

3. **Morir**

   Ahora, qué sucede si la célula viva no cumple con las regla anterior, en caso de que tenga más de 3 células vecinas, esta morirá por sobrepoblación. Si la célula presenta menos de 2 vecinos, también morirá en el próximo instante por soledad.

Ahora, por la teoría vista del juego, si implementamos estas 3 reglas en una cuadrilla 2D, el comportamiento debería variar y simular varios movimientos, pero esto no necesariamente pasa con los cuadrados que uno dibuje.
Existen figuras que al ser inicializadas desencadenan patrones interesantes y no desaparecen en poco tiempo, incluso se mantienen oscilando en distintos tiempos, la idea de implementar el juego sería configurar también estas figuras predeterminadas para contemplar mejor la simulación de la solución.

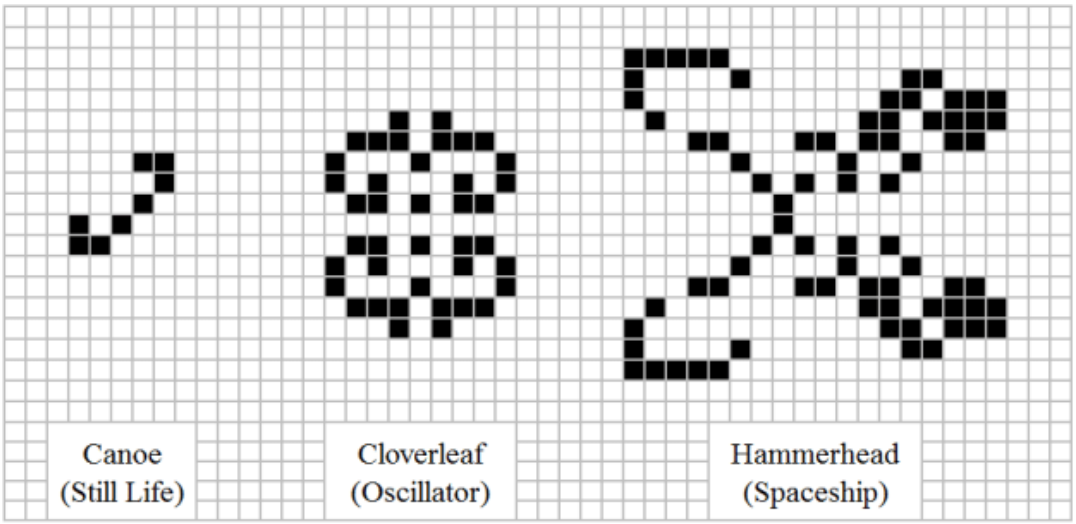

Figura 2. Ejemplos de patrones predeterminados para simular

Finalmente, parte de los objetivos es crear una GUI que facilite la visualización y manipulación de la aplicación del juego a un usuario. Es por ello que se necesita considerar usar todos los recursos necesarios de Tkinter aprendidos en el curso para realizar una aplicación y Menú instructivo, entendible y con buena estética.

EXPLICACIÓN DEL PROYECTO

La explicación del código será estructurada en diferentes partes. La explicación requiere imágenes y parte del código para un mayor entendimiento.

Estructura

La explicación del proyecto se divide en 4 secciones:

1. Menú de Inicio
2. Instrucciones del juego
3. Frame del juego y el algoritmo
4. Funciones dentro del juego

1. Menú de Inicio

Como se indicó en el Enunciado del Proyecto, el trabajo se realiza mediante Tkinter y solo eso bastaba para implementar todo el juego.

```
2. from tkinter import *
3. from PIL import Image,ImageTk
```

También se importó PIL para agregar la imágen de John Conway en el Menú de Inicio como se muestra en la figura 3.
La implementación de la clase padre general será del frame principal, porque de él dependen todas las demás funciones posteriormente inicializadas.

```
1.     class GameofLife(Frame):
2.         def __init__(self):
3.             Frame.__init__(self)
4.             self.pack(fill=BOTH,expand=1)
5.             self.master.title("THE GAME OF LIFE")
6.             self.configure(bg="darkslategray")
7.             self.continuous = 1
8.             self.frame = Frame(self, bg="darkslategray")
9.             self.frame.pack(side=TOP,expand=1)
10.            self.label = Label(self.frame,text="THE GAME OF LIFE",
11.           font=("System","20","bold"), fg="WHITE",bg="gray8", width=30,
12.       height=5, relief = SUNKEN)
13.            self.label.pack(side=TOP, fill=X,expand=1)
```

La apreciación del menú inicial fue con labels, 2 buttons y frames distribuidos específicamente en ciertas secciones para tener un orden establecido cuando el usuario lo aprecie.

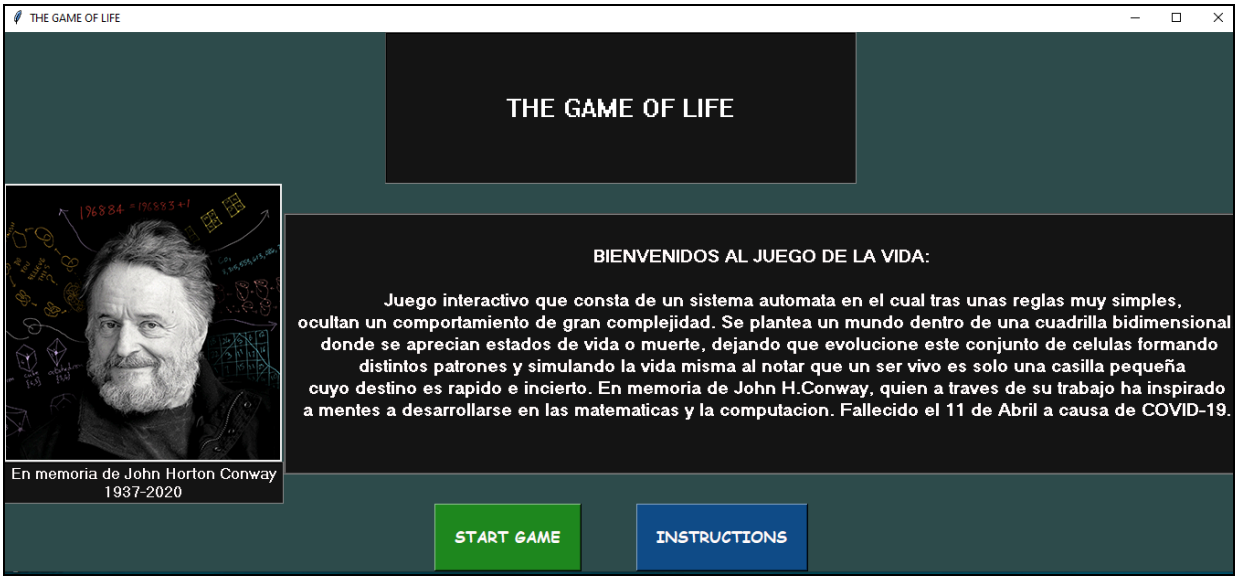

Figura 3. Menú inicial del juego de la vida

En el código se aprecia que la imagen fue importada mediante la función ImageTk.PhotoImage, en donde se adjunta el archivo de imagen, en este caso el retrato de Conway, situado en un label dentro del segundo frame del menú principal. IMPORTANTE: La imagen importada, es adjuntada en la carpeta del trabajo final para que pueda ser reconocida al momento de iniciar la simulación.

```
self.frame2 = Frame(self.framemid,bg="darkslategray",relief=GROOVE)
self.frame2.pack(side=LEFT)
self.ima = ImageTk.PhotoImage(Image.open("Conwayy.JPG"))
self.lab=Label(self.frame2,image=self.ima)
self.lab.pack()
self.lab2=Label(self.frame2,fg="snow",bg="gray8",font=("MS Sans Serif",
"13","bold"),relief = SUNKEN,text="En memoria de John Horton Conway\n
1937-2020")
self.lab2.pack(side=BOTTOM,ipadx=5)
```

Las demás opciones construidas dentro del menú principal como los 2 botones, el color, el texto, y cada frame para cada espacio del menú se aprecian en el código y no requiere mucha explicación. Se ha propuesto diferentes estilos de Fuente para diferentes partes, en los títulos, para mantener la estética binaria, se usó la fuente System, los botones, de una manera más llamativa, se usó la fuente Comic Sans; y por último, para los textos se usó el MS Sans Serif.

Dentro del texto fueron ignorados los signos de tildes y otras puntuaciones por el error que suele formarse cuando las interfaces son implementadas a partir de textos en tkinter.

2. Instrucciones del juego

En esta parte tampoco es necesaria mucha explicación, fueron a través de labels, frames y posicionados por pack(), mostrando diferentes colores y la explicación de cómo funciona el juego. Para este recuadro fueron necesarios 6 labels distintos posicionados geométricamente en orden, se muestra como ejemplo del código al primer label, donde va el título de Instructions.

```
def Instructions(self):
        root = Tk()
        root.title("Instructions")
        root.geometry('960x600')

        l1 = Label(root, text = "INSTRUCTIONS",font=("System","20","bold"),
bg = 'gray8',fg="snow", relief = SUNKEN,height=3,width=15)
        l1.pack(fill = X)
```

Cabe mencionar que este recuadro nuevo es abierto siempre en 2 ocasiones: Cuando se presiona INSTRUCTIONS en el Menú Principal y cuando se presiona HELP dentro del juego mismo, eso se explicará más adelante.

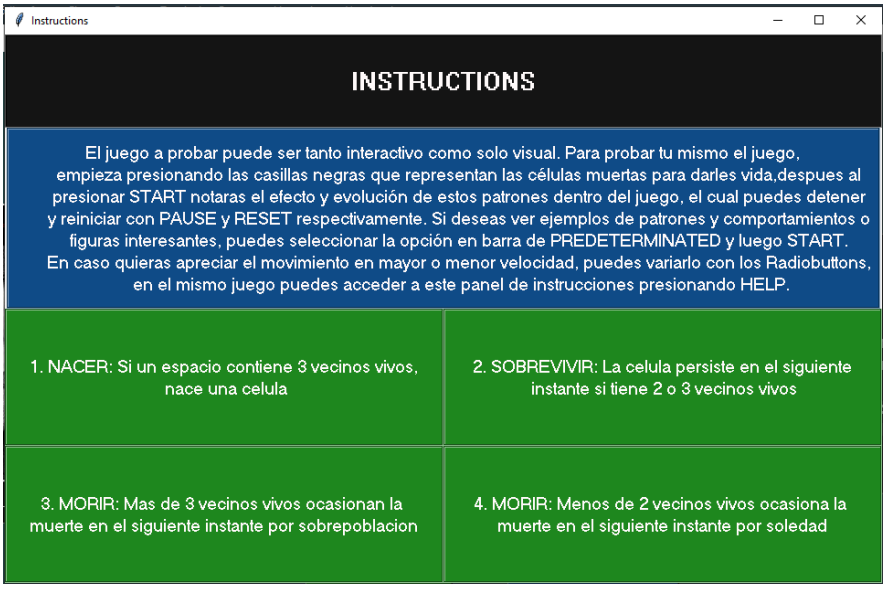

Figura 4. Instrucciones del juego.

3. Frame del juego y algoritmo

En esta sección, definimos la cuadrilla y el juego en esencia, para ello fue necesario crear una matriz de botones, en este caso de 20x40 para que entre el espacio al ser iniciada la simulación. Esta ventana, como se puede apreciar en el código, es inicializada dentro del mismo Frame inicial del menú del juego de la vida. Es decir, se conserva el mismo background inicial padre, y los demás frames son eliminados con destroy() y se genera primero la cuadrícula, que será llamada self.general_grid y dentro de ella se crearán todos los botones que simularan el juego completo.

```
    def GridGUI(self):

        self.btnstart.destroy()
        self.ins.destroy()
        self.label.destroy()
        self.frame.destroy()
        self.framemid.destroy()
        self.frame1.destroy()
        self.frame2.destroy()
        self.frame3.destroy()

        self.general_grid = Frame(self, bg="black")
        self.general_grid.grid(padx=(10,10),pady=(8,8),row=2,column=1)
```

Después, como se aprecia en la figura 5, hay botones, 1 menú desplegable y 3 radiobuttons que son acompañamientos necesarios para el usuario que desee probar el juego de la vida.

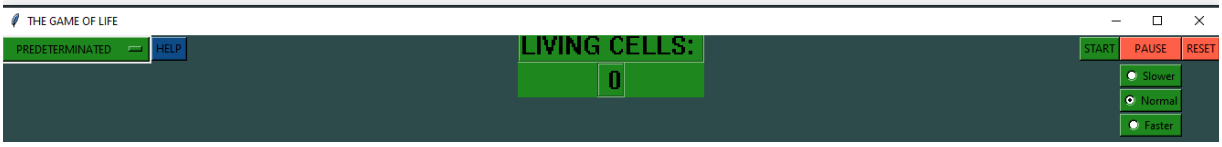

Figura 5. Botones, labels, menú y radio botones del frame del juego.

El label central, que es el más llamativo, tiene la finalidad de contar cuantas celdas “vivas” hay dentro de la cuadrícula que representa el universo donde se vive y muere. Para ello fue necesario implementar un código donde al crear un frame y 2 labels donde se menciona LIVING CELLS: y el número variable de las celdas se varía con la opción de self.code, el cual es una variable definida en el inicio y varía con respecto al cambio de colores de la cuadrícula, en una función explicada más adelante.

En el lado izquierdo hay un menú desplegable que dice PREDETERMINED, y un botón que dice HELP, el botón HELP lleva al usuario nuevamente a INSTRUCTIONS en caso no haya

comprendido cómo manipular el juego, para ello nada más vuelve a llamar a la funcion Instructions() en el comando de ese botón.

En PREDETERMINED, la barra de Menú fue diseñada para escoger los diferentes patrones predeterminados para simular los comportamientos complejos dentro del juego. La manera en la que son implementados es explicada más adelante con la función Predetermined().

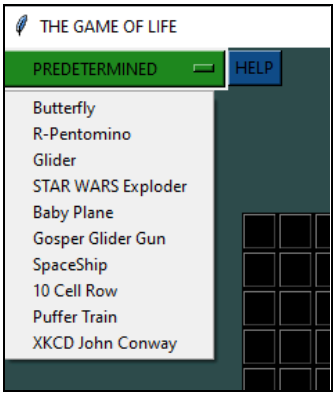

Figura 6. Opciones de PREDETERMINED para el juego.

Los botones de la derecha son intuibles, se entiende que START permite iniciar la simulación seguida y los radiobuttons cambian la velocidad de iteración de cada movimiento de células, el botón PAUSE efectivamente pausa el movimiento, y RESTART permite reiniciar todo el comportamiento, en teoría mata a todas las células para empezar el juego de nuevo.

Después, la vista general de la cuadrícula de 20x40 se aprecia en la figura 7. La parte de la creación mediante una lista/matriz de 20x40 botones se inició, donde la opción *command= lambda i=i, j=j: self.SwitchColor(i,j))* es la que permite cambiar de color al hacer click en un botón en específico dentro de la cuadrícula. La función SwitchColor básicamente permite cambiar el color de blanco a negro, y al final del código para cada cambio, sea negro o blanco, se disminuye o aumenta, respectivamente, el valor de self.code para contar el número de células vivas. Para SwitchColor se ha definido las variables self.coloralive como blanco y self.colordead como negro.

Entonces, la cuadrícula hasta este punto puede cambiar de estado vivo a muerto al hacer click dentro de cualquier botón que se desee.

```
for i in range(20):
            row = []
            self.cells.append(row)
            for j in range(40):
                self.cells[i].append(
Button(self.general_grid,bg=self.colordead,width=2,height=1))
                self.cells[i][j].configure(command=lambda i=i,
j=j:self.SwitchColor(i,j))
```

```
                self.cells[i][j].grid(row=i,column=j,padx=1,pady=1,sticky=N+S)
```

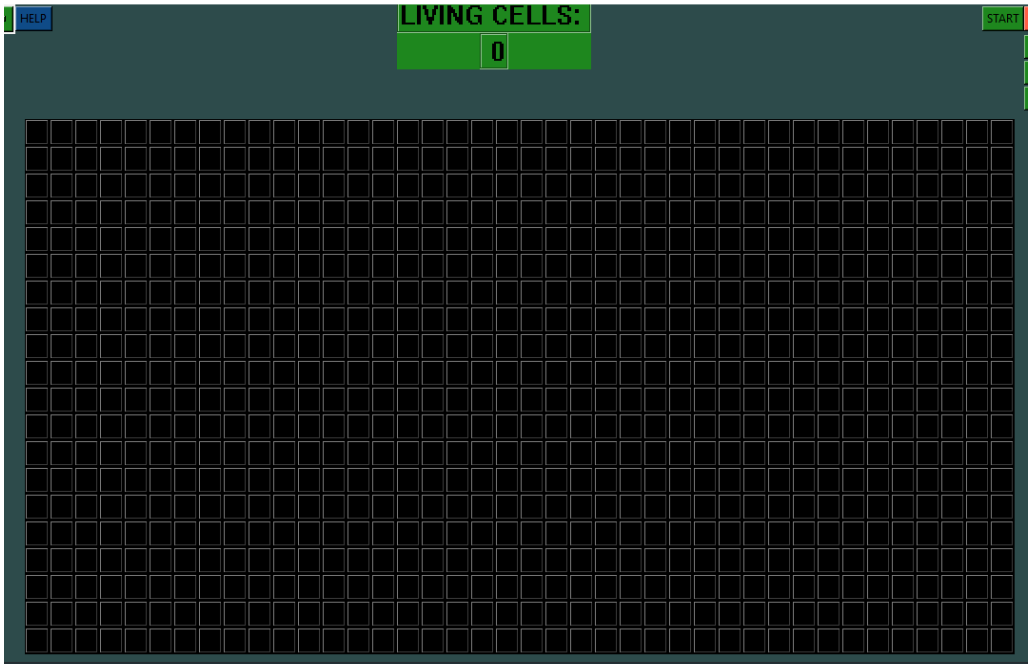

Figura 7. Cuadrícula general donde se visualiza el juego

Ahora, para definir el comportamiento y algoritmo de los seres vivos una vez inicializados dentro de la cuadrícula, se procede con el siguiente código.

```
def GameAlgorithm(self):
        #Se crea la lógica de movimiento del juego de la vida
        if self.continuous == 1:
            self.continuous = 2
        self.behavior = []
        for i in range(20):
            for j in range(40):
                bwcoord = (i,j)
                #Cuando una celula muerta tiene 3 vecinos vivos y nace:
                if self.cells[i][j]['bg'] == self.colordead and
self.neighbourbehavior(i,j) == 3:
                    self.behavior.append(bwcoord)
                #Cuando una célula viva no contiene 2 o 3 vecinos vivos y muere:
                elif self.cells[i][j]['bg'] == self.coloralive and
self.neighbourbehavior(i,j) != 3 and self.neighbourbehavior(i,j) != 2:
                    self.behavior.append(bwcoord)

        for a in self.behavior:
            self.SwitchColor(a[0],a[1])

        if len(self.behavior) != 0 and self.continuous == 2:
            self.after(self.rate,self.GameAlgorithm)

        else:
```

```
            self.continuous = 1
```

Donde, self.neighbourbehavior(i,j) es otra función en la que se define el cambio de color de los vecinos de una celda en específico.

En la función GameAlgorithm() se continúa la variable self.continuous como bandera para empezar el juego; dentro de la cuadrícula, en la lista vacía self.behavior se almacenan las celdas que cumplen con las 2 condiciones mostradas que son las reglas de supervivencia para cada célula. Después, para cada término dentro del self.behavior, se le aplicará la función SwitchColor, ya que todos los que cumplen self.behavior cumplen el algoritmo de vida o muerte, y los que no, se mantienen en el color original. Es así como se define el estado próximo del Juego de La Vida.

Ahora, procedemos a usar **recursividad,** en un bucle for determinamos 2 condiciones que siempre cumplen una vez GameAlgorithm() fue iniciado, una es la de self.continuous == 2 y la otra es que nunca self.behaviour no tenga valor alguno dentro, entonces ahí es donde se vuelve a emplear la funcion de GameAlgorithm(), porque la variable self.rate fue inicializada con un valor de 500 para el after. Entonces, podremos notar que una vez al presionar START podremos presenciar una secuencia de diferentes movimientos seguidos, por lo que presenciaremos la complejidad de la secuencia de los movimientos y reglas del juego de la vida.

4. Funciones dentro del juego

Para empezar, las funciones vistas previamente para GameAlgorithm, que son SwitchColor y neighbourbehavior, dentro de SwitchColor como ya se mencionó, permite el cambio de color de los botones y contabiliza las células vivas con self.code. Y respecto a neighbourbehavior, a y b representan a las coordenadas dentro de la cuadrícula de los vecinos que cumplen con las condiciones en GameAlgorithm, que al final devuelve c que es el contador de vecinos que cumplen la condición para permitir el siguiente estado de la célula que rodean.

```
def SwitchColor(self,i,j):
        if self.cells[i][j]["bg"] == self.coloralive:
            self.cells[i][j].configure(bg=self.colordead)
            self.code-=1

        else:
            self.cells[i][j].configure(bg=self.coloralive)
            self.code+=1

        self.scorelabel.configure(text=" "+str(self.code))

    def neighbourbehavior(self,i,j):
        c=0
        for a in range(i-1,i+2):
            for b in range(j-1,j+ 2):
                if a >=0 and b >=0 and a<20 and b<40:
                    if a == i and b == j:
                        pass
                    elif self.cells[a][b]["bg"]== self.coloralive:
                        c+=1
```

```
        return c
```

Las siguientes funciones corresponden a los botones iniciales del cuadro del juego, en cuanto a StopMotion es el comando que se activa al ser presionado PAUSE, como se nota, self.continuos valdrá 3, ya no será el estado inicial 1 ni el estado 2 de GameAlgorithm(), eso quiere decir que en ese comando activado, se detendrá la iteración del comportamiento de las células.
Restart es para "matar" a todas las células y volver a empezar el juego, esto mediante el uso de SwitchColor para los valores que tengan el self.coloralive.
En cuanto a Speed, este se varía con la variable self.varrad de tipo IntVar que determina la velocidad a variar, dependiendo de los Radiobuttons de velocidad, tanto 1000 para slower, 500 como Normal que equivale a self.rate, que es el predeterminado; y Faster con 100.

```
def StopMotion(self):
        self.continuous = 3

    def Restart(self):
        for i in range(20):
            for j in range(40):
                if self.cells[i][j]["bg"] == self.coloralive:
                    self.SwitchColor(i,j)


    def Speed(self):
        self.rate = self.varrad.get()
```

La función que falta definir es la de Predetermined(self,event) en la que responde al comando del Menú desplegable en la esquina superior izquierda. Para ello se definieron 10 listas de coordenadas diferentes que corresponden a todos los patrones predeterminados que permiten visualizar complejos e interesantes patrones del juego de la vida.

```
if self.varop.get() == "SpaceShip":
            for a in comms1:
                self.SwitchColor(a[0],a[1])
        elif self.varop.get() == "Butterfly":
            for a in comms2:
                self.SwitchColor(a[0],a[1])
        elif self.varop.get() == "Gosper Glider Gun":
            for a in comms3:
                self.SwitchColor(a[0],a[1])
        elif self.varop.get() == "10 Cell Row":
            for a in comms4:
                self.SwitchColor(a[0],a[1])
        elif self.varop.get() == "Glider":
            for a in comms5:
                self.SwitchColor(a[0],a[1])
        elif self.varop.get() == "Baby Plane":
            for a in comms6:
                self.SwitchColor(a[0],a[1])
        elif self.varop.get() == "XKCD John Conway":
```

```
            for a in comms7:
                self.SwitchColor(a[0],a[1])
        elif self.varop.get() == "STAR WARS Exploder":
            for a in comms8:
                self.SwitchColor(a[0],a[1])
        elif self.varop.get() == "R-Pentomino":
            for a in comms9:
                self.SwitchColor(a[0],a[1])

        elif self.varop.get() == "Puffer Train":
            for a in comms10:
                self.SwitchColor(a[0],a[1])
```

Como es mostrado, en cada uno de estos comandos, de comms1 a comms10 se definieron las coordenadas para que al seleccionar por ejemplo “R-Pentomino” en la cuadrícula del juego se dibuje esa forma y al presionar START el usuario pueda visualizar el patrón que sigue esa figura con las reglas del juego de la vida, un ejemplo se ve en la figura 8.
Esto se realiza gracias a self.varop que es un StringVar en el que están definido como PREDETERMINED y todos los valores guardados en self.shapes que fue inicializado al crear la barra de menú desplegable.
Esta fue la última función dentro de la clase padre GameOfLife(), la cual se inicializa en la última para probar la interfaz gráfica final.

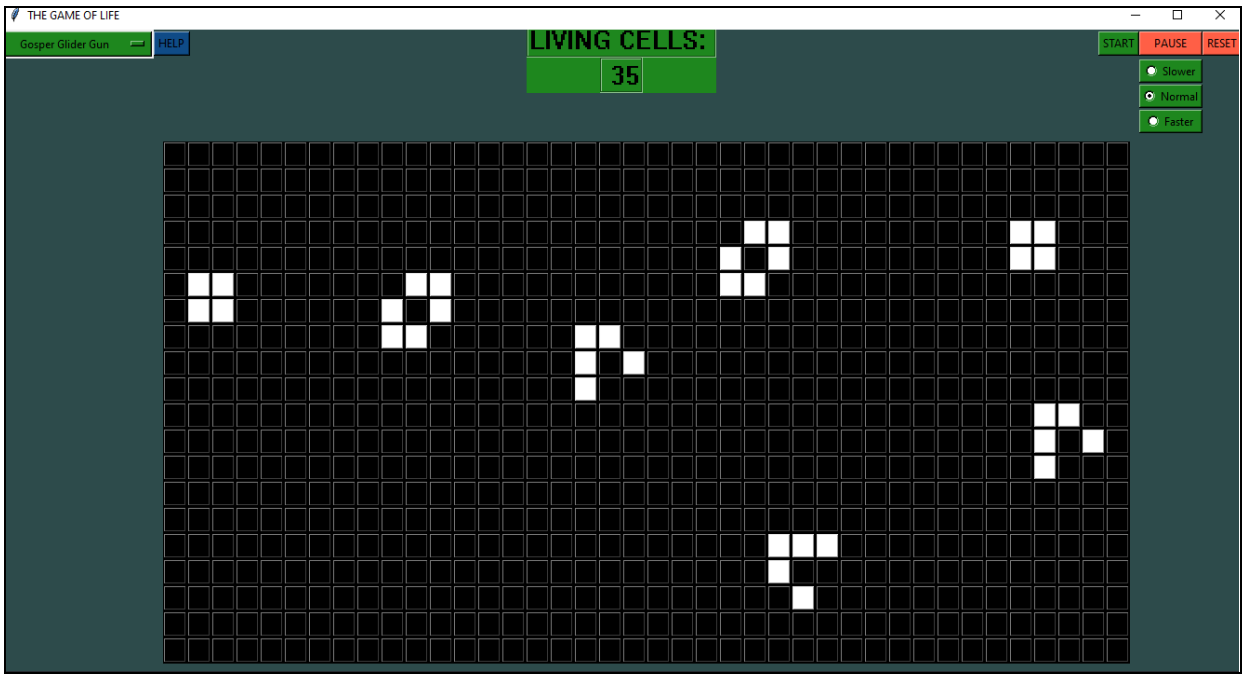

Figura 8. Selección de patrón de Gosper Glider Gun

CONCLUSIONES

- Como conclusión principal, se puede afirmar que el proyecto ha sido exitosamente realizado, cumpliendo con todas las indicaciones más detalles extra. El objetivo de realizar este proyecto con Tkinter y todas las variables y funciones de interfaz que se nos enseñó en el curso de Programación Avanzada también fue cumplido satisfactoriamente.

- Se usó programación orientada a objetos, Tkinter, recursividad, iteraciones y funciones propias, evidenciando el uso de los temas enseñados en el curso para cumplir con el trabajo.

- El proyecto, tiene como base estética que sea fácil de entender, como un producto, para que cualquier usuario sepa interpretarlo y pueda interactuar y jugar con la aplicación de manera que entienda la interfaz gráfica.

- El presente proyecto tomó tiempo de dedicación, sin embargo, sirvió para saber organizar y economizar código para la programación, instruyó bastante en la parte de arquitectura de software, de dónde y cómo empezar a crear una aplicación.

- Al simular el código, en caso encontrar el error FileNotFoundError: [Errno 2] No such file or directory: 'Conwayy.JPG'. Es por la imagen que está siendo adjuntada, asegurarse que deba estar en el mismo directorio de descarga que el código.

- El proyecto de el juego de la vida permitió que pueda reflexionar y tener cierto acercamiento a lo que es el estrecho vínculo entre las matemáticas y la informática, ayudando a la formación y elección de una especialidad dentro de lo que es la carrera de Ingeniería Biomédica.