



DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE CONCEPCIÓN
CONCEPCIÓN, CHILE.



Mini proyecto 1: Actividades en Arduino

Informe

Profesor: Vincenzo Caro F.

Integrantes: Rodrigo Hernández A.

Javier Herrera I.

12 de abril de 2024

Taller de aplicación TIC II
Ingeniería Civil en Telecomunicaciones

Resumen

El “Mini proyecto 1” propone resolver 2 actividades distintas utilizando un kit de sensores junto con un Arduino UNO.

La primera actividad nos pide crear el juego Whack-A-Mole, añadiéndole distintas dificultades cada una aumentando la velocidad de visualización del topo y un sistema de niveles, este juego ha sido recreado de muchas formas, pero nosotros nos centraremos en los vistos en ferias de diversiones, un concepto simple y divertido de trabajar

La segunda actividad consiste en recrear “The Game of Life” de Conway en Python, mostrar el juego a través de una interfaz gráfica utilizando *PyQt6* y, utilizando comunicación serial por USB, interactuar con el juego a través de *Arduino* para personalizar distintas situaciones de supervivencia al interactuar con sensores y botones a elección. Luego se debe modificar las reglas del juego para agregar condiciones de supervivencia distintas a las originales.

Índice

1. Desarrollo	1
1.1. Actividad 1: Whack-A-Mole!	1
1.1.1. Inicio del programa	1
1.1.2. Juego	2
1.1.3. Final de juego	3
1.2. Actividad 2: The Game of Life	4
1.2.1. Actividad 2.2	7
2. Conclusión	10
3. Referencias	11

1. Desarrollo

1.1. Actividad 1: Whack-A-Mole!

El juego "Whack-a-Mole" tiene sus raíces en los parques de diversiones y ferias, y se ha convertido en un clásico atemporal en la cultura de los juegos. La idea básica del juego es bastante simple: los jugadores tienen que golpear con un mazo a unos "topos" que aparecen de manera aleatoria en agujeros. Cada vez que golpean a un topo, ganan puntos. Sin embargo, los tópicos no permanecen mucho tiempo en la superficie, lo que hace que el juego sea desafiante y rápido.

Utilizando el software Arduino IDE para escribir el código que controlará el juego. El código incluye la lógica para:

1. Mostrar topos de manera aleatoria en los LEDs.
2. Detectar cuándo un jugador golpea un botón.
3. Llevar la cuenta de los puntos del jugador.
4. Controlar el tiempo de juego.

El juego funciona a través de LEDs y botones conectados en una protoboard, los elementos a utilizar son:

1. LEDs KY-009, KY-011, KY-016. Uno de cada tipo
2. 3 Botones de 4 pines
3. 1 buzzer activo
4. Conjunto de cables dupont macho-macho

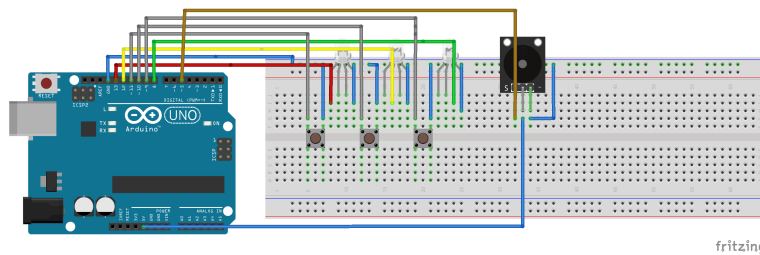


Fig. 1: Sketch de las conexiones

1.1.1. Inicio del programa

Para el inicio del programa se seleccionará entre 3 dificultades por el monitor serial aumentando la velocidad de los topos, pero mantiene el mismo puntaje de inicio, en las dificultades se ocupan los 3 leds.

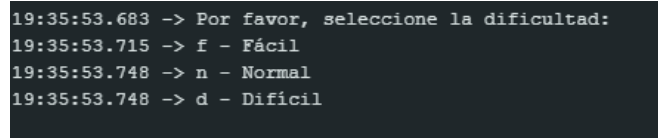
```
void waitForDifficulty() {
  Serial.println("Por favor, seleccione la dificultad:");
  Serial.println("f -- Facil");
  Serial.println("n -- Normal");
  Serial.println("d -- Dificil");

  while (true) {
    if (Serial.available() > 0) {
      char input = Serial.read();
      if (input == 'f') {
        currentDifficulty = FACIL;
        score = 50;
        moleDisplayTime = baseMoleDisplayTime[FACIL]; // Utiliza el tiempo base
                                                         de visualizacion del topo para el nivel facil
        break;
      }
    }
  }
}
```

```

    } else if (input == 'n') {
        currentDifficulty = NORMAL;
        score = 50;
        moleDisplayTime = baseMoleDisplayTime[NORMAL]; // Utiliza el tiempo base
        de visualizacion del topo para el nivel normal
        break;
    } else if (input == 'd') {
        currentDifficulty = DIFICIL;
        score = 50;
        moleDisplayTime = baseMoleDisplayTime[DIFICIL]; // Utiliza el tiempo base
        de visualizacion del topo para el nivel dificil
        break;
    }
}
}
}

```



```

19:35:53.683 -> Por favor, seleccione la dificultad:
19:35:53.715 -> f - Fácil
19:35:53.748 -> n - Normal
19:35:53.748 -> d - Dificil

```

Fig. 2: Selección de dificultad

1.1.2. Juego

Para el juego se pide que el buzzer emita sonido cada vez que se golpea al topo o nos equivocamos, como también se emite el sonido al momento de perder, para esto usamos el siguiente código:

```

int playerHit = waitForInput(); // Espera que el jugador golpee un topo
if (playerHit == activeMole) {
    score += 10; // Incrementa el puntaje si el jugador golpea el topo correcto
    tone(buzzerPin, 1000, 100); // Sonar el buzzer (tono de 1000 Hz por 100 ms)
    cuando golpea bien un topo
} else {
    score -= 5; // Reduce el puntaje en 5 si el jugador golpea el topo
    incorrecto
    if (score < 0) {
        score = 0; // Asegura que el puntaje no sea negativo
    }
    tone(buzzerPin, 500, 200); // Sonar el buzzer (tono de 500 Hz por 200 ms)
    cuando golpea mal un topo
}

```

El juego posee 3 dificultades, las cuales se diferencian en el display de los topos y la velocidad máxima a la cual pueden llegar con el tiempo,

1. La Dificultad "Facil" comienza con tiempos de visualizacion de 1000ms y con un maximo de 350ms
2. La dificultad "Normal" comienza con tiempos de visualizacion de 750ms y con un máximo de 300ms
3. la dificultad "dificil" comienza con tiempos de visualizacion de 500ms y con un maximo de 250ms

cada nivel, se diferencia por el tiempo que lleva el jugador sin llegar a los 0 puntos y se va aumentando la velocidad de visualización del topo

1.1.3. Final de juego

Al finalizar se pide que se vuelva al score de inicio y se reinicie el programa, como también el buzzer de finalización, se realizó de la siguiente manera.

```
// Verificar si el puntaje es igual o menor que cero
if (score <= 0) {
    // Reiniciar el programa
    tone(buzzerPin, 700, 1500);
    delay(1550);
    wdt_enable(WDTO_15MS); // Habilitar el temporizador de reinicio
    while (true) {} // Esperar a que el microcontrolador se reinicie
}
```

Se utilizó la librería " #include avr/wdt.h " para poder acceder al comando para habilitar el reinicio del microcontrolador y se añadió un delay un poco más alto que el tiempo del buzzer para que no se corte al reiniciar el Arduino. El juego indica continuamente el score acumulado del jugador y la dificultad seleccionada

Video del juego en funcionamiento: <https://www.youtube.com/watch?v=-YQUYGXIdIs>

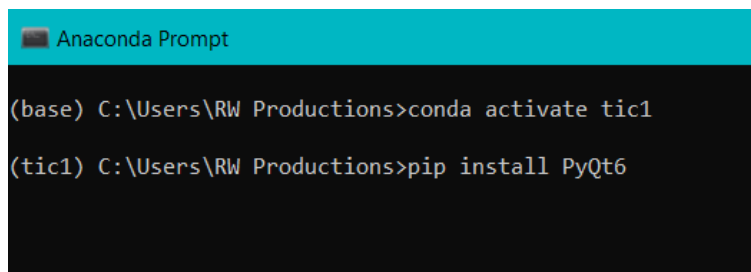
1.2. Actividad 2: The Game of Life

El Juego de la vida es un autómata celular diseñado por el matemático británico John Horton Conway en 1970. Es un juego de cero jugadores, en el que su evolución es determinada por un estado inicial, sin requerir intervención adicional. Se considera un sistema Turing completo que puede simular cualquier otra Máquina de Turing.

Para realizar la actividad se recreó el juego en un programa en Python el cual fue proporcionado por el profesor, el juego se mostraba usando la biblioteca *matplotlib.animation* el cual seguía las reglas originales del juego:

1. Sobrepoblación: Si una célula viva tiene más de tres vecinos vivos, muere debido a la falta de recursos.
2. Subpoblación: Si una célula viva tiene menos de dos vecinos vivos, muere por soledad.
3. Estabilidad: Una célula viva con dos o tres vecinos vivos permanece viva en la siguiente generación.
4. Reproducción: Si una célula muerta tiene exactamente tres vecinos vivos, nace en la siguiente generación debido a la reproducción.

Se pedía mostrar el juego en una interfaz gráfica utilizando *PyQt6* para poder agregar botones y otros elementos que permitieran interactuar con el juego. Para instalar *PyQt6* se debe abrir la consola del intérprete de Python (Anaconda en este caso), se activa la carpeta con los archivos de las bibliotecas (o se crea una en caso de no tener) y se escribe el siguiente comando: ***pip install PyQt6***



```

Anaconda Prompt

(base) C:\Users\RW Productions>conda activate tic1

(tic1) C:\Users\RW Productions>pip install PyQt6

```

Fig. 3: Consola de Anaconda para instalar PyQt6

Una vez instalado, utilizando el programa *Designer* integrado al instalar *PyQt6*, se creó una interfaz gráfica específica para el diseño del juego.

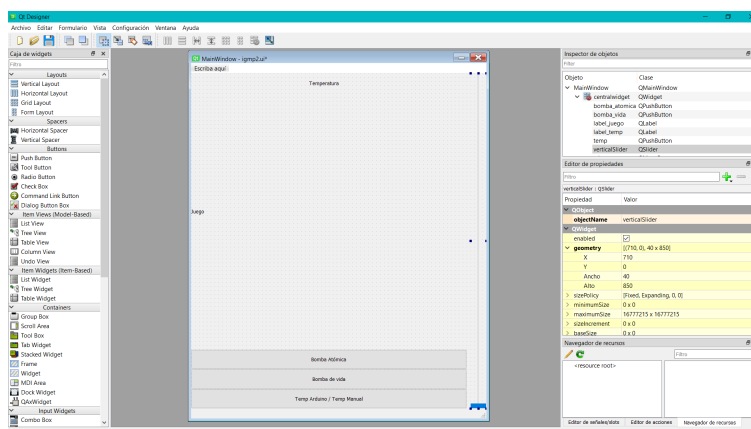


Fig. 4: Diseño de interfaz gráfica en Designer

Para poder implementar el diseño en python, se usó la consola de anaconda para transformar el archivo.ui generado por designer a un archivo .py. Este método consta de importar el archivo .py que contiene la interfaz y añadirlo a nuestra ventana personalizada. El primer paso es transformar el archivo .ui a un archivo .py mediante el comando: `pyuic6 -x test.ui -o gui test.py` Esto generará un archivo .py con la siguiente estructura: una clase

“Ui MainWindow” con un único método llamado “setupUi”

Para convertir el archivo .ui a .py primero abrimos el la consola de anaconda, se activa la carpeta donde están instaladas las libreas con el comando **conda activate** junto con el nombre de la carpeta y nos ubicamos en la carpeta donde se quiere guardar el archivo (el archivo .ui a transformar tienen que estar en la misma carpeta). Para ello usa el comando **cd** junto con la ruta de acceso a la carpeta de interés, luego se ejecuta el comando **pyuic6 -x igmp2.ui -o igmp2.py** para obtener el archivo .py al que llamar como se muestra en la figura 5.

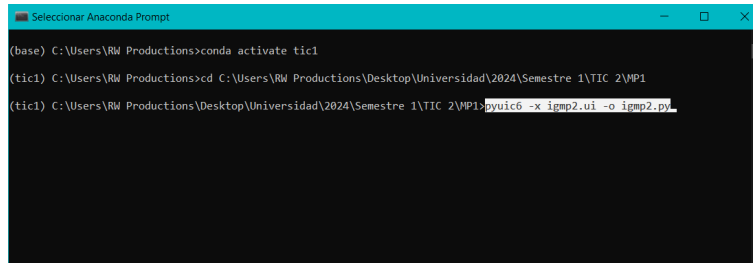


Fig. 5: Anaconda Prompt.

Luego de guardar el archivo .py que contiene el diseño que se generó en designer.exe, se crea un nuevo código el cual llama al archivo .py transformado y este muestra la interfaz gráfica diseñada. El código necesario para llamar al archivo .py es el siguiente:

```

import sys
from PyQt6 import QtWidgets, uic
from igmp2 import Ui_MainWindow

class MainWindow(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self, *args, **kwargs):
        super(MainWindow, self).__init__(*args, **kwargs)
        self.setupUi(self)

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    window.setWindowTitle(" Juego - de - la - Vida - de - Conway")
    app.exec()
  
```

Utilizando el código del juego de la vida que dejó el profesor, se integró a python para correr el juego con la nueva interfaz gráfica y se agregó la función de los botones junto con los sensores para poder controlar el juego de manera externa. el funcionamiento del programa se muestra en la figura 6

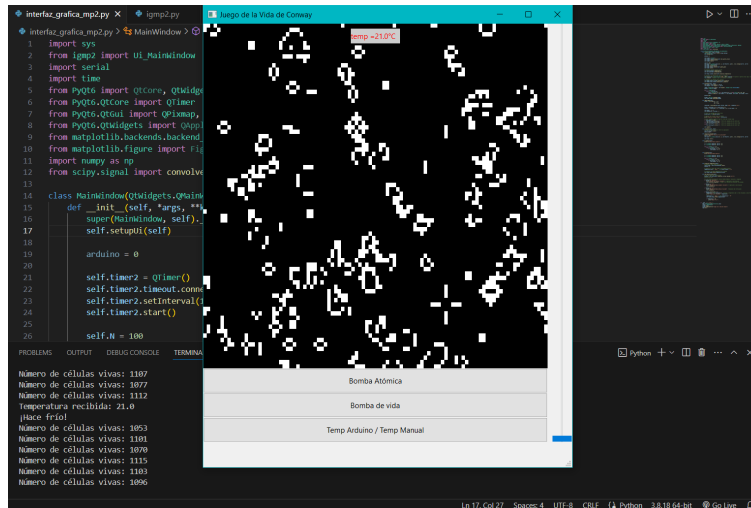


Fig. 6: El juego de la vida mostrado en la interfaz gráfica.

Para poder interactuar con Arduino se usó comunicación serial por USB y se conectaron los siguientes sensores:

1. **KY-004 KEY SWITCH MODULE:** Este switch se utiliza para reiniciar el programa mandando la palabra (Reiniciar) por com serial, luego el programa de en Python lee este comando y reinicia el juego.
2. **KY-009 RGB FULL COLOR LED SMD MODULE:** El led indica la cantidad de células que hay en pantalla. Si hay mas de 1000 células su color es azul y envía un ('E') por com serial, si hay entre 600 y 1000 su color es verde y envía un ('S') por com serial, finalmente si hay menos de 600 su color es rojo y envía un ('A') por com serial.
3. **KY-015 TEMPERATURE AND HUMIDITY SENSOR MODULE:** Este sensor se emplea solamente para medir la temperatura ambiente, enviando cada 3 segundos el dato de la forma 't-xx.xx' donde xx.xx es la temperatura en grados celsius.
4. **KY-017 MERCURY TILT SWITCH MODULE:** Este sensor se emplea para enviar la bomba de vida al juego. Funciona como un switch y envía el dato **b-1** cuando está encendido y **b-0** cuando está apagado.
5. **KY-018 PHOTORESISTOR MODULE:** Este sensor se emplea para enviar la bomba atómica al juego. Funciona como un switch y envía el dato **a-1** cuando hay un cambio de intensidad de luz y **a-0** vuelve a su estado normal. El uso de este sensor tiene sentido poético ya que *con la llegada de la oscuridad viene la destrucción y la muerte.*

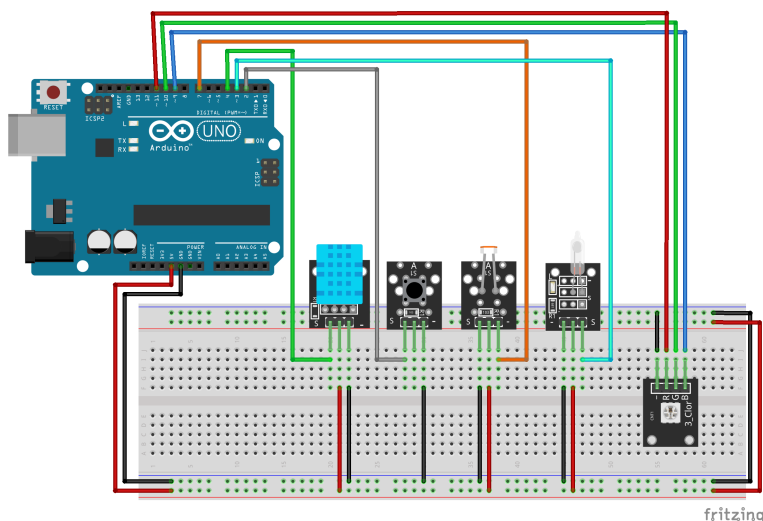


Fig. 7: Diagrama de conexión de los sensores a Arduino.

Para establecer comunicación serial entre Arduino y Python se usó **Serial.println("xx")** en Arduino donde xx es el dato nombrado en el ítem anterior. Esto se lee en python mediante el comando **self.serial_port = serial.Serial('COM4', 9600, timeout=0)** el cual está leyendo constantemente el puerto serial y, con funciones **if** se agregan condiciones para llamar a las respectivas funciones programadas.

Esto es un ejemplo de cómo leer e implementar funciones con el puerto serial:

```
def leer_puerto_serial(self):
    # Leer los datos del puerto serial
    mensaje = self.serial_port.readline().strip().decode('utf-8')

    # Si se recibe el mensaje de reinicio desde Arduino, reiniciar el
    programa
    if mensaje == "Reiniciar":
        print("Mensaje de reinicio recibido") #Impresion de verificacion
        self.reiniciar_programa() # Llamar a la funcion para reiniciar el
        programa

    if mensaje == "b-0":
        print("Mensaje de Bomba sanadora recibido") #Impresion de
        verificacion
        self.b_sanadora()

    if mensaje == "a-0":
        print("Mensaje de Bomba atomica recibido") #Impresion de
        verificacion
        self.b_atmica()
```

Para agregar funcionalidad a los botones puestos en la interfaz gráfica se utiliza la función **clicked.connect** la cual llama a las funciones ya creadas.

A continuación se muestra un ejemplo del uso de la función mencionada:

```
# Llamado de los botones de la interfaz
self.bomba_atmica.clicked.connect(self.b_atmica)
self.bomba_vida.clicked.connect(self.b_sanadora)
```

La dificultad de esta actividad consistía en implementar la comunicación serial entre Arduino y Python y mostrar el juego con **PyQt6** que en un inicio usaba **matplotlib.animation**. Esto se debe a que desde Arduino se tienen que mandar los datos con comillas (" ") y desde Python se debe anteponer una b (**b'x'**).

Otro problema durante la programación del código fue el uso del switch **KY-004**. El principal problema fue que al presionar el botón se mandaban muchos datos en vez de uno. Para solucionarlo se aplicó el método **INPUT_PULLUP** el cual solo detecta cambios en el botón y así manda solo un dato.

1.2.1. Actividad 2.2

Para la presente actividad se pide modificar las reglas del juego de la vida e implementar lo siguiente:

1. Las células ahora tendrán una vida total de 100 puntos, y se verán enfrentadas a condiciones extremas del clima y una guerra contra un planeta vecino.
2. Si una célula viva tiene menos de 2 o más de 3 vecinos vivos, su vida se reduce en 30 puntos en cada iteración del juego.
3. Si una célula muerta tiene exactamente 3 vecinos vivos, renace con una vida de 100 puntos.
4. Si todas las células mueren, reproduzcan un audio (sin buzzer) representando que se perdió la guerra, utilizando para ello la biblioteca playsound u otra similar. Luego, reinicien el juego.

5. Si la temperatura está por debajo del umbral de frío (a definir), aumenta la vida de cada célula en 10 puntos en cada iteración del juego.
6. Si la temperatura está por encima del umbral de calor (a definir), reduce la vida de cada célula en 20 puntos en cada iteración del juego.
7. Si se activa el botón de la bomba nuclear, elimina todas las células dentro de un rango de 21x21 en una posición aleatoria.
8. Si se activa el botón de área de curación, revive las células muertas con 70 puntos de vida y aumenta la vida de las células vivas en 50 puntos (con un máximo de 100) dentro de un rango de 21x21 en una posición aleatoria.

Para realizar la actividad, primero seteamos las células para que tengan un valor de vida de 0 a 100 con la siguiente función:

```
self.N = 100
self.grid = np.random.choice([0, 100], (self.N, self.N), p=[0.8, 0.2])
```

Luego se modifican las condiciones de vecindad para que, si la célula viva tiene menos de 2 o más de 3 vecinos vivos, su vida se reduce en 30 puntos en cada iteración del juego y si la célula muerta tiene exactamente 3 vecinos vivos, renace con una vida de 100.

```
convolved = convolve2d(self.grid > 0, kernel, mode='same', boundary='wrap')
```

```
new_grid = np.where((self.grid > 0) & ((convolved < 2) | (convolved > 3)), self.grid - 30, self.grid)
new_grid = np.where((self.grid == 0) & (convolved == 3), 100, new_grid)
new_grid = np.where((new_grid < 0), 0, new_grid)
```

Luego se setea un rango de temperatura para las condiciones de temperatura, en este caso se eligió 20 para la condición de frío y 27 para la condición de calor:

```
if self.temperatura > 27: # Humbral de calor
    self.grid = self.grid - 34 # debe ser 30
    self.grid = np.where((self.grid < 0), 0, self.grid)

if self.temperatura < 20: # Humbral de frio
    self.grid = self.grid + 10
    self.grid = np.where((self.grid < 0), 0, self.grid)
```

Para poder diferenciar la vida de las células se definieron colores en la célula. Estos son verde si la célula tiene 51 y 100 de vida y rojo si tiene entre 1 y 50 de vida, además si la célula tiene 0 de vida muerta y se transforma en negro.

Los botones de las bombas atómica y de vida solo sufrieron ligeras modificaciones las cuales se ven a continuación:

```
def b_sanadora(self):
    print("Enviando bomba sanadora")

    x = np.random.randint(0, self.N - 21)
    y = np.random.randint(0, self.N - 21)

    for i in range(x, x + 21):
        for j in range(y, y + 21):
            if self.grid[i, j] == 0:
                self.grid[i, j] = 70
            elif 1 <= self.grid[i, j] <= 50:
                self.grid[i, j] = self.grid[i, j] + 50
            else:
```

```

        self.grid[i, j] = 100

def b_atomica(self):
    print("Enviando bomba atomica")

    x = np.random.randint(0, self.N - 21)
    y = np.random.randint(0, self.N - 21)

    for i in range(x, x + 21):
        for j in range(y, y + 21):
            if 0 <= self.grid[i, j] <= 100:
                self.grid[i, j] = 0

```

Finalmente, cuando el número de células vivas llega a 0, se utiliza la biblioteca **playsound** para reproducir un audio.

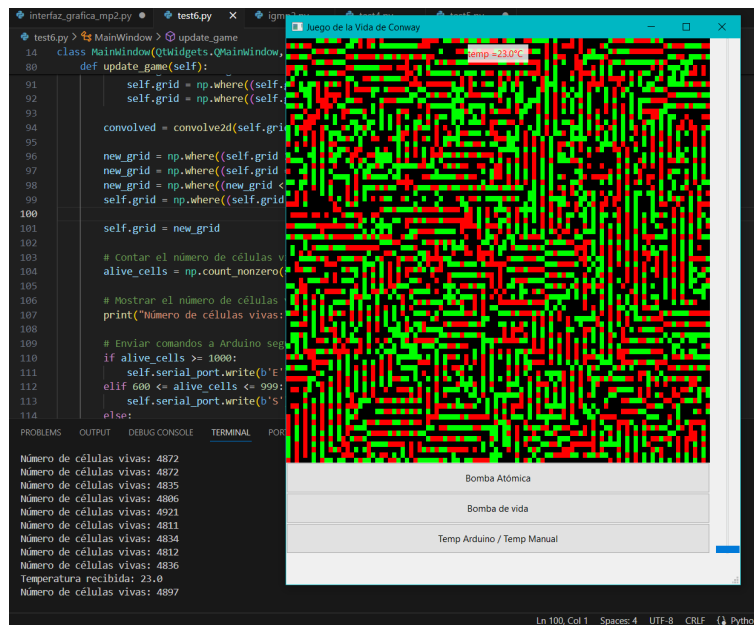


Fig. 8: Juego de la vida modificado.

La implementación de la función de audio generó muchos problemas debido a que no se lograba apagar el audio cuando el número de células cambiaba y al llamar la función de manera normal el audio se reproducía infinitamente y el juego se quedaba pegado. Para solucionar esto se acortó el audio a una duración de 10 segundos y se creó un hilo que llama a la función **Muerte**.

Para evitar problemas se creó una variable con valores `True` y `False` y así se ejecute la función solamente cuando `True` o `False` estén activados.

2. Conclusión

En conclusión, en el **Mini Proyecto 1** se introdujeron los conceptos de comunicación serial junto con la correcta utilización de los sensores y la implementación de estos con Python. Al haber programado ya en Raspberry Pi se hace más fácil saber la lógica de programación de los sensores, además el lenguaje de Arduino es fácil de entender y hay mucha mas información en internet.

Actividad 1: En esta actividad se conectó el Arduino a distintos módulos del kit de sensores entregado a comienzo del ramo, nos ayudó a familiarizarnos con su programación y funcionamiento encontrando dificultades de por medio, a pesar de eso se logró crear un juego funcional de Whack-A-Mole con niveles, un sistema de dificultad y aumentando de a poco la visualización de los topes, no fue fácil lograr que funcionase, pero sí entretenido trabajarlo y solucionarlo.

Actividad 2: En la actividad 2 se conectó Arduino con una interfaz gráfica creada en **Designer**. Al tener ya los conocimientos de cómo crear interfaces esto resultó fácil. Se conectaron distintos sensores a Arduino y a través de comunicación serial con Python se logró controlar un juego tanto física como virtualmente usando **pushButtons**. Modificar el juego de la vida de Conway no fue algo fácil pero fue entretenido y al momento de revivir las células muertas se forman patrones bonitos.

Este proyecto nos proporcionó una valiosa experiencia en la aplicación de conocimientos en telecomunicaciones, la resolución de problemas técnicos y la adquisición, procesamiento y representación de datos de sensores, lo cual enriquece significativamente nuestras habilidades en el campo de la Ingeniería.

3. Referencias

1. colaboradores de Wikipedia. (2024, 29 enero). Juego de la vida. Wikipedia, la Enciclopedia Libre. https://es.wikipedia.org/wiki/Juego_de_la_vida