UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

549305-1 – Taller de aplicacion TIC ll

Profesor Vincenzo Caro Fuentes

Mini proyecto 1

Bastián Baeza Retamal

Pablo Olguin Molina

Concepción, 24 de mayo de 2023

Resumen

Se presentan los resultados trabajados durante la primera parte del curso, que fueron condensados en el Miniproyecto I, donde nos centramos en 2 actividades, el juego interactivo “Whack-A-Mole”, que consiste en en golpear los "topos" iluminados en el momento justo para ganar puntos y avanzar niveles, esto se logró mediante la utilización de leds, que representaban a los topos alzándose y, botones, que al golpearlos simulaban, un golpe al topo correspondiente. El desarrollo de esta actividad fue de tanta dificultad como la actividad 2, pero supuso un desafío, debido a nuestro desconocimiento con la programación en C y arduino.

Por otra parte, la segunda actividad constó de la programación de el reconocido “The Game of Life”, juego creado por John Conway como un modelo matemático de un sistema celular con el fin de explorar la idea de la vida y la muerte en un entorno abstracto. Como era de esperarse, por su naturaleza lógica, el proceso de programación fue bastante difícil, además, tuvimos que volver a usar las interfaces gráficas y el entorno de python, uniendo estos 2 lenguajes para que “trabajasen” en armonía. Sin duda, el desarrollo de la 2 etapa esta actividad, fue la que más desafíos nos presentó, habiendo bastantes que no pudimos superar, a pesar de esto, la lógica del juego y todas sus funciones, de no ser por una pocas funcionan como fue estipulado.

Para estas 2 actividades se describe la metodología de trabajo, los aciertos y los fallos que hemos enfrentado en el desarrollo de miniproyecto, además de los detalles detrás de su implementación y enfoque. Describiendo el desarrollo del entorno gráfico, la construcción del circuito junto con el diseño de funciones, además del mecanismo de funcionamiento de cada uno de estos juegos.

Creemos que en esta actividad se busca expandir nuestro conocimiento mediante la interacción de dos lenguajes, arduino y python debido a que para el desarrollo de esta actividad es absolutamente necesario que ambos dispositivos “interactúen” entre sí, arduino enviando señales que son mecánicas mediante el pulso de botones, ejecutando las órdenes, y también recibiendo mensajes desde python los cuales debe interpretar según el escenario que se programó.

Tabla de Contenidos

1

Descripción en de los implementos utilizados

1.1 Sensores y dispositivos implementados (actividad 1)

Para el desarrollo de las actividades, utilizaremos el arduino R3 board, los leds RGB KY-016 y KY-009 y un led amarillo KY-011, un buzzer pasivo KY-006 y los respectivos botones, que fueron entregados en clases por el profesor.

1.2 Sensores y dispositivos implementados (actividad 2)

Actividad n°1. Estilo de Títulos para Actividades

Ítem 1.1

1.1.a) Enunciado

Esta primera actividad consiste en implementar el juego interactivo “Whack-A-Mole” utilizando una placa Arduino, LEDs como "topos" y botones como martillos. El juego consistirá en golpear los "topos" iluminados en el momento justo para ganar puntos, definiendo distintos niveles de dificultad asociados a los tiempos de duración del encendido de los LEDS y el tiempo total para cada ronda del juego.

1.1.b) Esquemático Circuitos

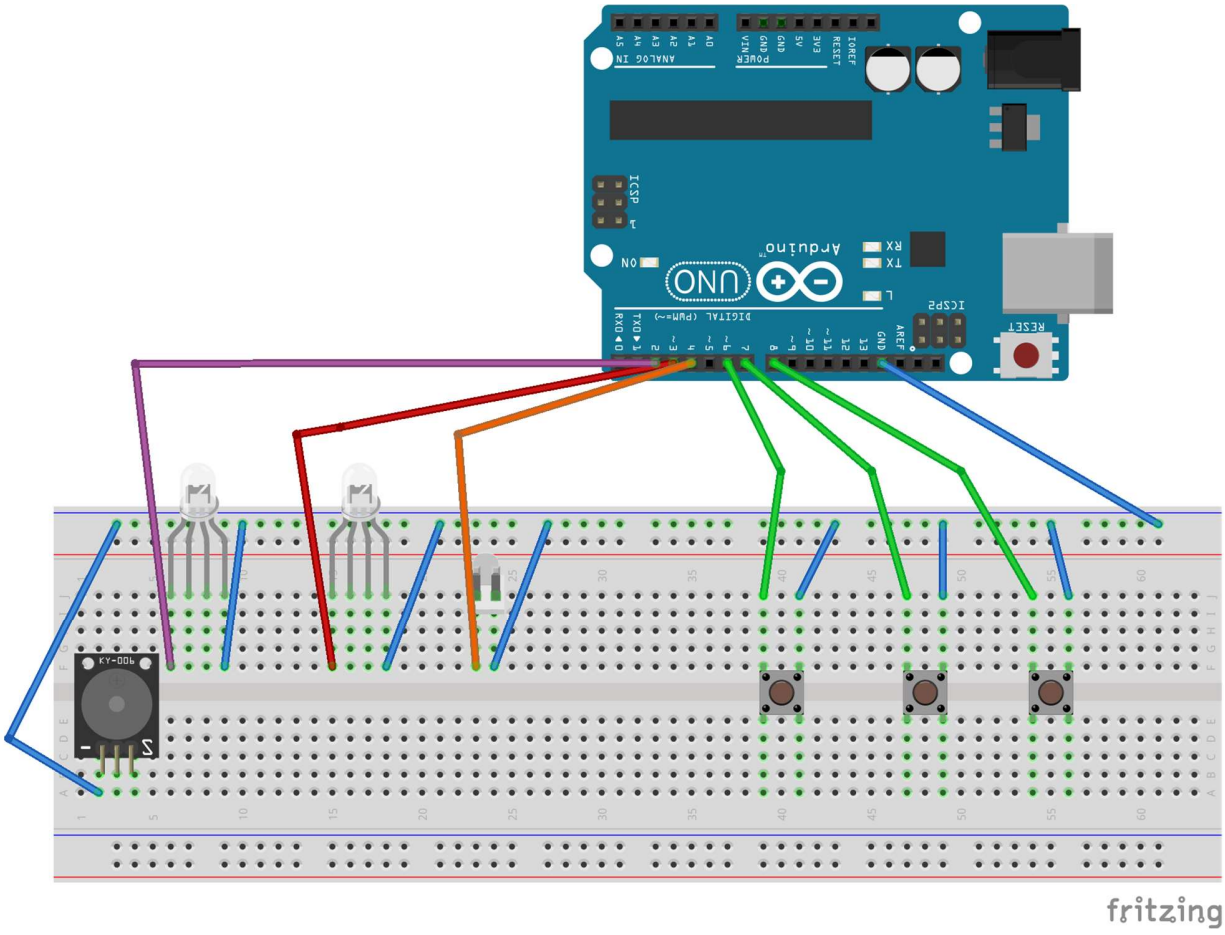


Figura 1.1 Conexión de 3 leds, un buzzer pasivo y 3 botones con una placa Arduino.

1.1.c) Resultados y Comentarios

Antes de explicar el funcionamiento del juego, se deben aclarar algunos cambios que se hicieron con respecto a los requerimientos estipulados en la pauta.

a) La elección de dificultad, no afecta a la cantidad de leds que se utilizaran en el juego
b) Cuando el juego se termina, la dificultad no es restaurada, sigue siendo la misma que se escogió desde un principio, esto es debido a que definimos esa parte del código en la función de setup, por lo tanto, si se quiere escoger una nueva dificultad, se debe reiniciar el código.
c) En nuestro juego, el jugador comienza con 90 puntos, con cada acierto sumando 10 y cada fracaso 30.

01.Definición de Pines y variables

```
// Definición de pines
const int ledPins[] = {2, 3, 4};  // Pines para los LEDs
const int buttonPins[] = {6, 7, 8}; // Pines para los botones
const int buzzerPin = 10; // Pin para el buzzer

// Variables de juego
int score = 90; // Puntuación inicial del jugador
int level = 1; // Nivel de dificultad inicial
int successfulHits = 0; // Número de golpes exitosos
int missedHits = 0;
int racha = 0; // contador de racha
int tiempoEncendidoLED = 0;
int tmax = 3000;
```

Primero que nada, definimos los pines que utilizaremos, siendo el 2, 3 y 4 los pines para los leds, el 6, 7 y 8 para los botones y el 10 para el buzzer. Los pines de los leds y botones tienen la siguiente relación con respecto a qué botón corresponde a cada led: **led pin 2 - botón pin 6, led pin 3 - botón pin 7 y led pin 4 - botón pin 8.**

Las función de cada variable que fue definida en el inicio es la siguiente:

score: Puntuación inicial del juego

level: nivel inicial del juego

successfulHits: Número de golpes exitosos

missedHits: Número de golpes erróneos

racha: Contador de golpes exitosos consecutivos

tiempoEncendidoLED: Tiempo que se mantendrá encendido el led (milisegundos)

tmax: Tiempo en el que se podrá presionar el botón (milisegundos)

02.Función void setup

```
18  void setup() {
19    Serial.begin(9600); // Inicialización del monitor serial
20    randomSeed(analogRead(0)); // Solicitar al usuario que elija la dificultad
21    Serial.println("Por favor, elija la dificultad introduciendo el numero correpondiente:");
22    Serial.println("1. Fácil (Los topos se mostraran por solo 0,5 s)");
23    Serial.println("2. Medio (Los topos se mostraran por solo 0,3 s)");
24    Serial.println("3. Difícil (Los topos se mostraran por solo 0,1 s)");
25    Serial.println("4. Extremo (Los topos se mostraran por solo 0,1 s y tendras menos tiempo para golpearlos!)");
26    while (!Serial.available()) {
27      // Esperar hasta que se ingrese algo en el monitor serial
28    }
```

Inicializamos el monitor serial, donde emitimos el mensaje "Por favor, elija la dificultad introduciendo el número correspondiente:".

```
30    // Leer la opción de dificultad del usuario
31    int dificultad = Serial.parseInt();
32
33    // Configurar el tiempo de acuerdo a la dificultad elegida
34    switch (dificultad) {
35      case 1:
36        tiempoEncendidoLED = 500; // Fácil
37        break;
38      case 2:
39        tiempoEncendidoLED = 300; // Medio
40        break;
41      case 3:
42        tiempoEncendidoLED = 100; // Difícil
43        break;
44      case 4:
45        tiempoEncendidoLED = 100; // Extremo
46        tmax = 1500;              // tiempo para golpear reducido a la mitad
47        break;
48      default:
49        Serial.println("Dificultad no válida. Se usará la dificultad predeterminada.");
50        tiempoEncendidoLED = 500; // Por defecto, se utiliza la dificultad fácil
51        break;
```

En esta instancia el jugador deberá escoger entre las 4 dificultades, las 3 más fáciles modifican la variable de tiempo que dicta cuantos milisegundos se podrá ver el led encendido y la última dificultad, acorta el tiempo en el que el jugador puede reaccionar y golpear el botón, antes de que “salga el otro topo”.

```
52    }// Inicializar pines de LEDs como salida
53    //for (int i = 0; i < sizeof(ledPins); i++) {
54      // pinMode(pin, OUTPUT);
55    pinMode(2, OUTPUT);
56    pinMode(3, OUTPUT);
57    pinMode(4, OUTPUT);
58
59    // Inicializar pines de botones como entrada
60    for (int i = 0; i < sizeof(buttonPins); i++) {
61      pinMode(buttonPins[i], INPUT_PULLUP);
62    }
63
64    // Inicializar pin de buzzer como salida
65    pinMode(buzzerPin, OUTPUT);
66  }
```

Ya escogida la dificultad, se inicializan los leds, botones y el buzzer. El código comentado fue parte de unas pruebas, no se debe tomar en cuenta.

03.Función void loop

Nuestra función void loop contiene la parte más compleja del código, anteriormente nos dedicamos a definir variables y preparar las variables para poder jugar, pero ahora debemos dedicarnos a la lógica del juego.

```
68  void loop() {
69    // Lógica del juego
70    while (score > 0) {
71      // Generar secuencia aleatoria de LEDs a encender
72      int lednumber = random(2, 5);
73      digitalWrite(lednumber, HIGH);
74      delay(tiempoEncendidoLED); // Tiempo que permanece encendido el LED
75      digitalWrite(lednumber, LOW);
76
77      // Esperar a que el jugador golpee el LED
78      unsigned long startTime = millis();
79      int buttonPressed = -1;
80      while (buttonPressed == -1 && millis() - startTime < tmax) {
81        for (int i = 0; i < sizeof(buttonPins)/sizeof(buttonPins[0]); i++) {
82          if (digitalRead(buttonPins[i]) == LOW) {
83            buttonPressed = i + 2;
84            break;
85          }
86        }
87      }
```

La primera parte de la función empieza con un ciclo while que estará activo mientras la puntuación del juego sea mayor a 0, dentro de este ciclo definimos la variable **lednumber** como un número aleatorio entre 2, 3 o 4, que corresponde a el led que se encenderá.

En "***unsigned long startTime = millis();***" obtenemos el tiempo actual para poder hacer un conteo desde el momento en que se encendió el led, luego definimos "**buttonPressed**"como la variable que almacenará el botón presionado. En el ciclo while empieza una cuenta regresiva durante la cual se podrá presionar el botón, y el código verifica si se ha presionado alguno de los botones, si es que se presiona un boton, la variable "**buttonPressed**" guarda el número del botón presionado.

```
89      // Comprobar si el jugador golpeó correctamente
90      if (buttonPressed == lednumber) {
91        score += 10;
92        successfulHits++;
93        racha += 1;
94        tone(buzzerPin, 1000, 200); // Tono para golpe exitoso
95      } else {
96        score -=30;
97        missedHits++;
98        racha = 0;
99        tone(buzzerPin, 200, 500); // Tono para golpe errado
100     }
```

Con el botón ya presionado, (y en caso de que ninguno fuese presionado, se guarda como el valor -1, que dará como valor erróneo al momento de la verificación) se compara con un "**if**" si el botón corresponde al led encendido, si este es el correcto, el jugador gana 10 puntos y anota una +1 en el contador de rachas y si es incorrecto, se le restan 30 puntos y el contador de rachas vuelve a ser 0. Además de esto se reproduce un sonido característico para los aciertos y los fracasos.

```
    // Actualizar monitor serial
    Serial.print(lednumber);
    Serial.print("Score: ");
    Serial.print(score);
    Serial.print(" | Level: ");
    Serial.print(level);
    Serial.print(" | Successful Hits: ");
    Serial.print(successfulHits);
    Serial.print(" | Missed Hits: ");
    Serial.println(missedHits);
    Serial.print(" | Successful Hits consecutivos: ");
    Serial.println(racha);

    // Aumentar nivel si es necesario
    if (racha % 10 == 0 && racha > 0) {
    level++;
    }
    if (racha == 0) {
    level = 0;
    }
  }

  // Reproducir melodía de fin de juego
  for (int i = 1000; i < 2000; i += 100) {
    tone(buzzerPin, i, 100);
    delay(50);
  }

  // Reiniciar variables de juego
  score = 50;
  level = 1;
  successfulHits = 0;
  missedHits = 0;
  racha = 0;
}
```

Los valores son reportados después de cada golpe en el serial monitor y se verifica si se debe subir de nivel o no. Cuando el juego termina, osea, el jugador perdió, se reproduce una melodía y se reinician las variables del juego, para así volver a jugar.

Actividad n°2

Enunciado: The game of life fue creado por Conway como un modelo matemático de un sistema celular con el fin de explorar la idea de la vida y la muerte en un entorno abstracto. Conway buscaba un sistema con reglas simples,pero que pudiera producir comportamientos complejos y variados

P 2.1:

- Cada un intervalo de 10 segundos, el programa de Python debe enviar hacia Arduino la cantidad de células vivas que se encuentran actualmente en la cuadrícula. Luego, definan en Arduino 3 intervalos para evaluar las condiciones de estabilidad, subpoblación y sobrepoblación de las células; vinculando cada estado con un color particular de un módulo LED. En caso de ocurrir un cambio de estado, la transición entre cada color de cada estado debe realizarse de forma paulatina, utilizando las salidas PWM de Arduino.
- Desde Arduino, implementen un botón pulsador que sea capaz de reiniciar desde 0 el juego en Python. Para ello, cada vez que se presione el botón, envíen un carácter o un mensaje hacia Python para que la interfaz reaccione a esta acción.

Codigo en arduino:

Comenzamos definiendo los pines que vamos a utilizar, tanto los de los leds como los de los botones.

Y también establecemos los límites que vamos a establecer para cada caso, sub-poblacion, sobrepoblación y estabilidad.

Posteriormente en la función void setup que se ejecuta una vez al inicio del programa, la que contiene la configuración de los pines 9,10,11 que corresponden a los pines de salida que envían señal a los

leds este ajuste servirá posteriormente para controlar la intensidad de los colores.

```
const int ledred = 9;
const int ledgre = 10;
const int ledblu = 11;
const int BUTTON_PIN = 2;
int alive_cells;

// Definir los intervalos de condiciones
volatile bool Reiniciar_flag = false;

// Prototipo de la función de interrupción
void Reiniciarjuego();

void setup() {
  // Configurar pines como salida
  pinMode(ledred, OUTPUT);
  pinMode(ledgre, OUTPUT);
  pinMode(ledblu, OUTPUT);

  // Configurar pin del botón como entrada con pullup
  pinMode(BUTTON_PIN, INPUT_PULLUP);
```

También se especifica la velocidad a la que va a trabajar nuestro puerto seria la cual será 9600.

En la segunda línea if (Serial.available() >0) verificará si hay datos disponibles para la lectura en el puesto serial, luego si encuentra datos disponibles se define el número de células vivas como int (numero entero) para enviar la cantidad desde python a arduino.

después se establecen una serie de condiciones para determinar el estado de nuestra “colonia” de células si es que se encuentran sobre o bajo los márgenes previamente establecidos para así determinar el color de nuestros leds o rgb.

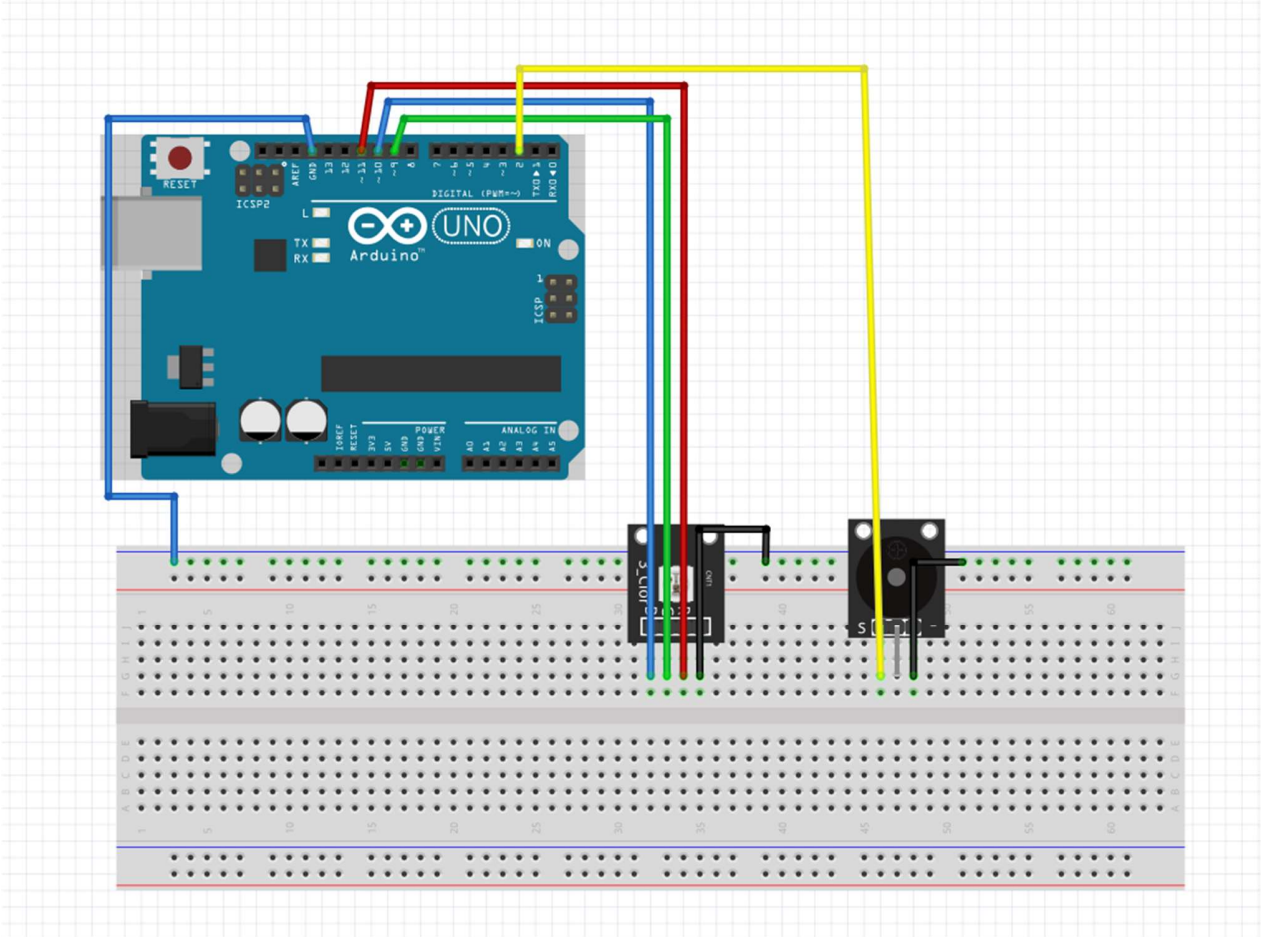
```
  // Inicializar comunicación serial
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    // Leer el número de células vivas desde Python
    alive_cells = Serial.read();

    if (alive_cells == 'S') {       ///blue es para sobrepoblacion
      digitalWrite(ledblu, HIGH);
      digitalWrite(ledgre, LOW);
      digitalWrite(ledred, LOW);
    }
    if (alive_cells == 'C') {            ////green es subpoblacion
      digitalWrite(ledblu, LOW);
      digitalWrite(ledgre, HIGH);
      digitalWrite(ledred, LOW);
    }
    if (alive_cells == 'E') {               ////red es para estabilidad
      digitalWrite(ledblu, LOW);
      digitalWrite(ledgre, LOW);
      digitalWrite(ledred, HIGH);
    }
  }

  // Detectar si se presionó el botón
  if (Reiniciar_flag) {
    // Reiniciar juego
    Serial.println("Reiniciar");
    Reiniciar_flag = false; // Restablecer la bandera de reinicio
    delay(100); // Espera para evitar la detección múltiple del botón
  }
```

En la primera parte se verifica si la variable de reinicio es verdadera, tal que si es verdadera se ha presionado el botón y el juego debe reiniciarse, luego se utiliza un delay para evitar que detecciones falsas del botón

```
// Definición de la función de interrupción
void Reiniciarjuego() {
 Reiniciar_flag = true; // Activar la bandera de reinicio
}
```

La función set color nos ayudará a establecer el color del led rgb conectado a arduino y recibe tres parámetros, red, green y blue, los cuales representarán la intensidad que le damos a nuestros leds, también se utiliza la funciona analogWrite para enviar señales a los pines de los les y al final tenemos nuestro void reiniciarJuego la que tiene como fin actuar cuando se establece que reiniciar flag es verdadero, tal que el programa debe reiniciarse hasta el void loop del programa.

Esquema de arduino 2.1



Parte 2.2

P2.2 Arduino.

```arduino
#include <DHT.h>

#define DHTPIN 5
#define DHTTYPE DHT11
const int ledred = 9;
const int ledgre = 10;
const int ledblu = 11;
const int BUTTON_PIN = 2;
const int buton2 = 6;
const int pinjoystick = 3;

int alive_cells;
DHT dht(DHTPIN, DHTTYPE);

// Definir los intervalos de condiciones
volatile bool Reiniciar_flag = false;

// Prototipo de la función de interrupción
void Reiniciarjuego();

void setup() {
  // Configurar pines como salida
  pinMode(ledred, OUTPUT);
  pinMode(ledgre, OUTPUT);
  pinMode(ledblu, OUTPUT);
  dht.begin();
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  pinMode(buton2, INPUT_PULLUP);
  pinMode(pinjoystick, INPUT_PULLUP);

  // Inicializar comunicación serial
  Serial.begin(9600);
}
```

En esta parte del código definimos 2 botones adicionales para poder enviar a través del serial print mediante pulso los mensajes por el serial para que python los reciba utilizamos input_pullup que asegura la lectura apropiada del pin (botón) cuando el interruptor está abierto.

Luego arduino recibe los caracteres s,e,c, con los cuales se determina que color del rgb va a ser encendido según el estado de las células en nuestro juego de la vida.

```arduino
if (Serial.available() > 0) {



  if (alive_cells == 'S') {       //blue es para sobrepoblacion
    digitalWrite(ledblu, HIGH);
    digitalWrite(ledgre, LOW);
    digitalWrite(ledred, LOW);
  } else if (alive_cells == 'C') { //green es subpoblacion
    digitalWrite(ledblu, LOW);
    digitalWrite(ledgre, HIGH);
    digitalWrite(ledred, LOW);
  } else if (alive_cells == 'E') { //red es para estabilidad
    digitalWrite(ledblu, LOW);
    digitalWrite(ledgre, LOW);
    digitalWrite(ledred, HIGH);
  }
}
```

finalmente (en arduino) utilizamos la función digitalRead en cada uno de nuestros botones cuando su estado == LOW es decir cuando fue presionado para que el serial monitor envíe los caracteres, h_1, b_1 que significan heal 1 y bomba 1 respectivamente, utilizamos un delay de 100 para evitar que el botón se presione múltiples veces y obtener resultados no deseados

```
if (digitalRead(buton2) == LOW) {
  //
  Serial.println("n_1");
  //
  delay(100);
}


if (digitalRead(pinjoystick) == LOW) {
  //
  Serial.println("h_1");
  // delay parar evitar multples cliclos
  delay(100);
}


if (digitalRead(BUTTON_PIN) == LOW) {
  //
  Serial.println("Reiniciar");
  //
  delay(100);
}
}
```

2.2 Python.

Cuando nuestro arduino envía los caracteres, h_1 y b_1, nuestro código en python debe ejecutar las instrucciones que se le atribuyen a cada carácter, en este caso h_1 y b_1 tienen como instrucción revivir o eliminar células en un espacio randomizado de 21x21

```
    def leer_puerto(self):
    # Leer datos del puerto serial de manera no bloqueante
        while self.serial_port.in_waiting > 0:
            mensaje = self.serial_port.readline().decode().strip()

        if mensaje == "Reiniciar":
            print("reiniciando")
            self.reset_game()

        elif mensaje == "h_1":
            print("Mensaje de Bomba sanadora recibido")
            self.h_1()
        elif mensaje == "n_1":
            print("Mensaje de Bomba atómica recibido")
            self.n_1()
```

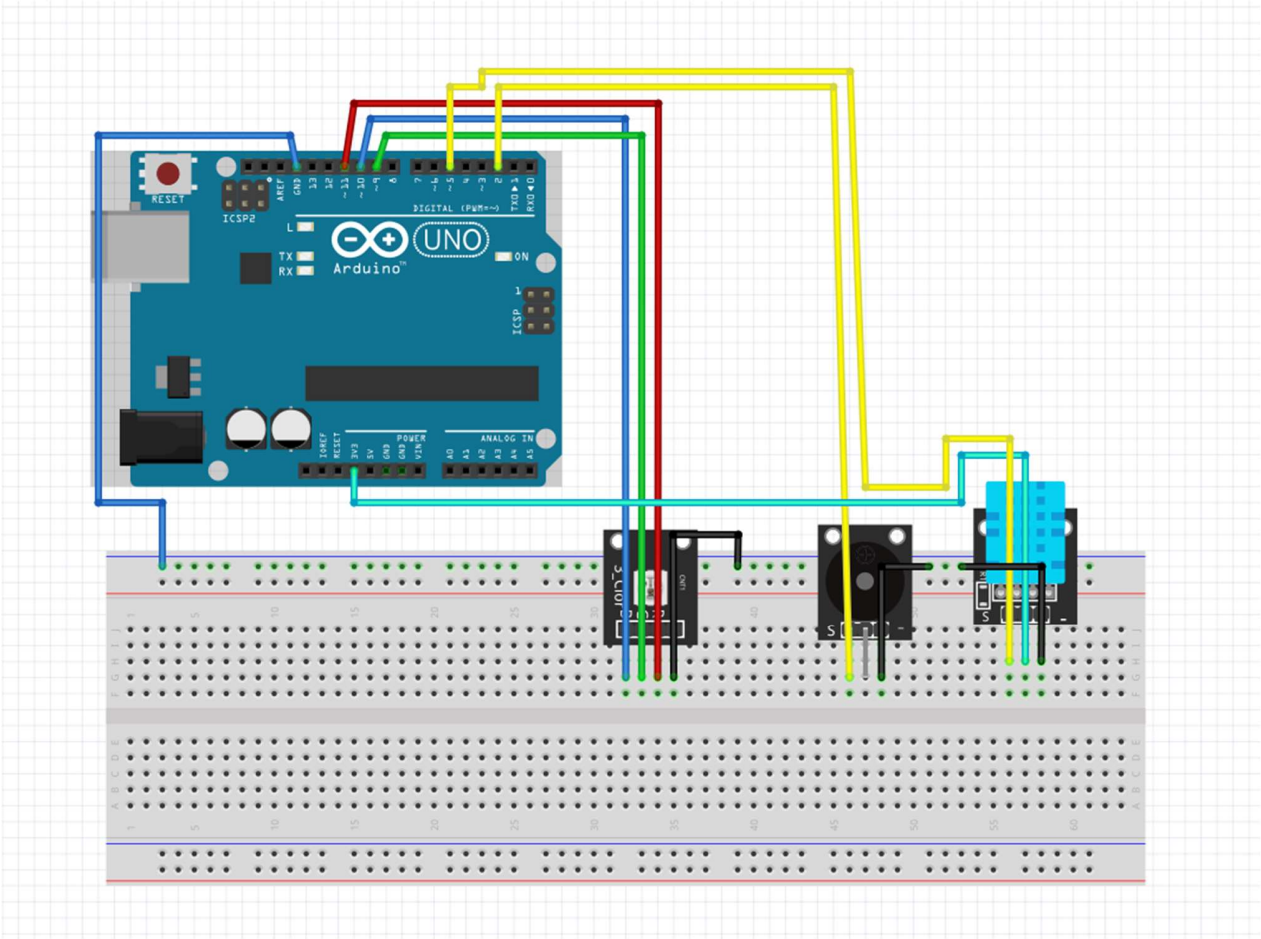
```
    def update(self, frame, img):
        kernel = np.array([[1, 1, 1],
                           [1, 0, 1],
                           [1, 1, 1]])

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


    # Aplicar las modificaciones específicas adicionales
        nuevo_grid = np.where((self.grid > 0) & ((convolved < 2) | (convolved > 3)), self.grid - 30, self.grid)

# Regla para renacimiento de células muertas
        nuevo_grid = np.where((self.grid == 0) & (convolved == 3), 100, nuevo_grid)
        nuevo_grid = np.where((nuevo_grid < 0), 0, nuevo_grid)
        self.grid = np.where((self.grid > 100), 100, nuevo_grid)

        self.grid = nuevo_grid
        img.set_data(nuevo_grid)
        img.autoscale()
```

También aplicamos las nuevas reglas del juego de la vida, donde nuestro nuevo _grid tiene reglas diferentes por ejemplo en la primera línea decimos que sí self.grid>0 y si una célula tiene menos de dos vecinos o más de 3 vecinos le restamos 30 de vida ect.

También definimos una serie de QpushButton para las funciones heal, nuke, reset, start y stop las cuales nos permiten presionas las distintas acciones en pantalla.

```
    self.start_button = QPushButton("Start")
    self.start_button.clicked.connect(self.start_game)
    self.layout.addWidget(self.start_button)

    self.stop_button = QPushButton("Stop")
    self.stop_button.clicked.connect(self.stop_game)
    self.layout.addWidget(self.stop_button)

    self.reset_button = QPushButton("Reset")
    self.reset_button.clicked.connect(self.reset_game)
    self.layout.addWidget(self.reset_button)

    self.temperature_textedit = QTextEdit()
    self.layout.addWidget(self.temperature_textedit)

    self.figure = plt.figure()
    self.canvas = FigureCanvas(self.figure)
    self.layout.addWidget(self.canvas)

    self.n_1_button = QPushButton("Nuke")
    self.n_1_button.clicked.connect(self.n_1)
    self.layout.addWidget(self.n_1_button)

    self.h_1_button = QPushButton("Heal")
    self.h_1_button.clicked.connect(self.h_1)
    self.layout.addWidget(self.h_1_button)

    self.n_1_button.clicked.connect(self.n_1)
    self.h_1_button.clicked.connect(self.h_1)

def setup_game(self):
```

Fritzing 2.2


RESET
ICSP2
DIGITAL (PWM~)
UNO
Arduino
TX
RX
ON
ICSP
POWER
ANALOG IN