



INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y
TECNOLOGÍAS AVANZADAS

Ingeniería Mecatrónica

U.A. Programación Avanzada
Prof. Jose Luis Cruz Mora

Práctica 6 ”Comunicacion serie”

Grupo: 2MV7

Integrantes:

Barrrios Mendez Jose Alberto

Cordero García Pablo Daniel

Hernandez Tapia Jared Hassan

Sanchez Velazquez Luis Enrique

Fecha: 7 de mayo de 2024

Objetivo

Crear aplicaciones que se comuniquen por un puerto USB con un dispositivo externo.

Introducción

La comunicación serie entre Python y Arduino es una herramienta esencial en el mundo de la electrónica y la informática, donde la interacción entre el software y el hardware es fundamental para el desarrollo de una amplia gama de proyectos. Python, conocido por su versatilidad y facilidad de uso, se convierte en un aliado perfecto para el procesamiento de datos, mientras que Arduino, con su capacidad para interactuar con el mundo físico a través de sensores y actuadores, se convierte en el corazón de muchos proyectos electrónicos.

Una de las formas más comunes de conectar Python y Arduino es a través de la comunicación serie. Este método establece una conexión punto a punto entre los dos dispositivos, permitiendo la transmisión bidireccional de datos byte a byte. Esta comunicación serie puede realizarse mediante un cable USB o de forma inalámbrica, dependiendo de los requisitos del proyecto.

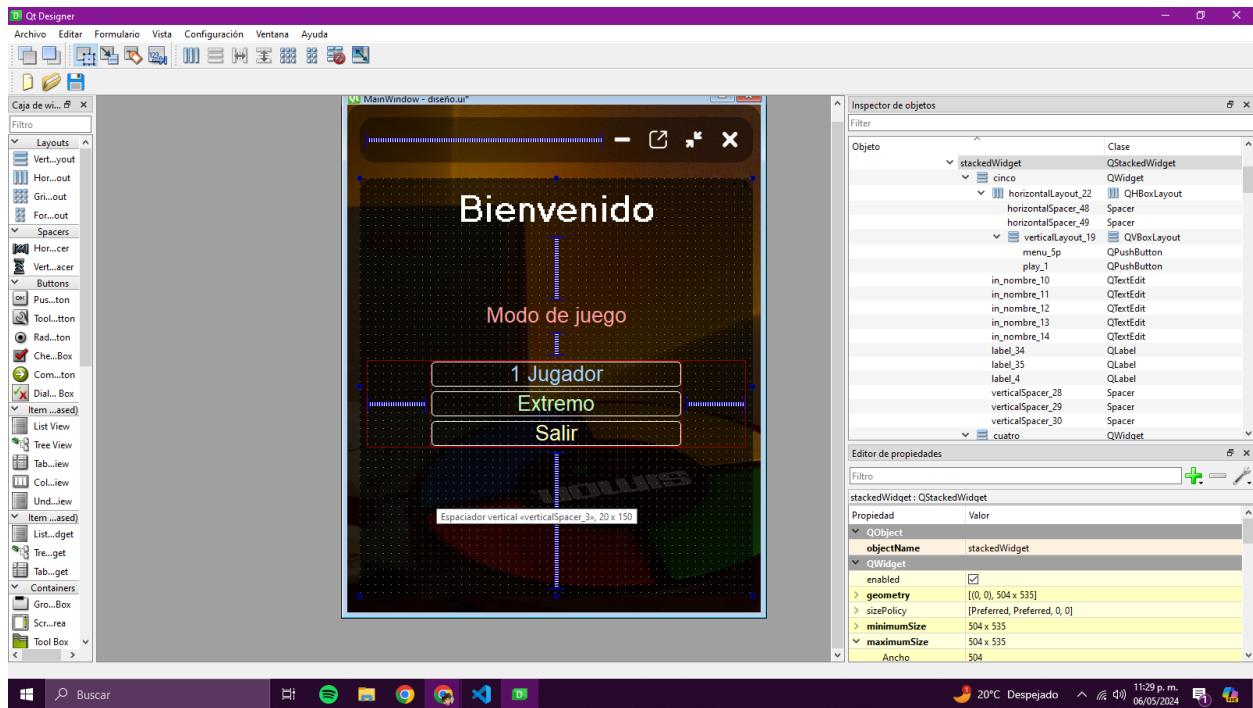
Para facilitar esta comunicación desde el lado de Python, existen diversas bibliotecas y herramientas. Una de las opciones más populares es PyFirmata, una biblioteca que proporciona una interfaz de programación de aplicaciones (API) para controlar Arduino desde Python utilizando el protocolo Firmata. Firmata es un protocolo de comunicación que se carga en la placa Arduino y le permite ser controlado por un software externo a través de un puerto serie. PyFirmata simplifica la interacción con Arduino al proporcionar una API sencilla y fácil de usar para enviar y recibir datos.

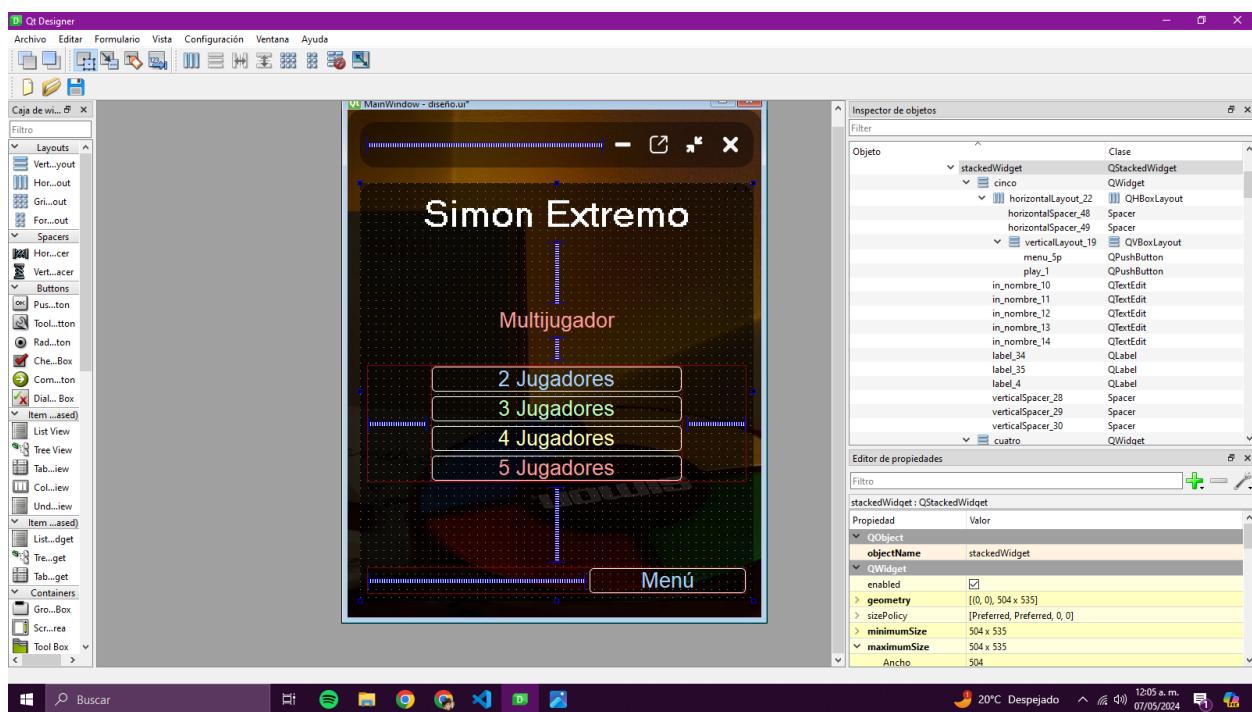
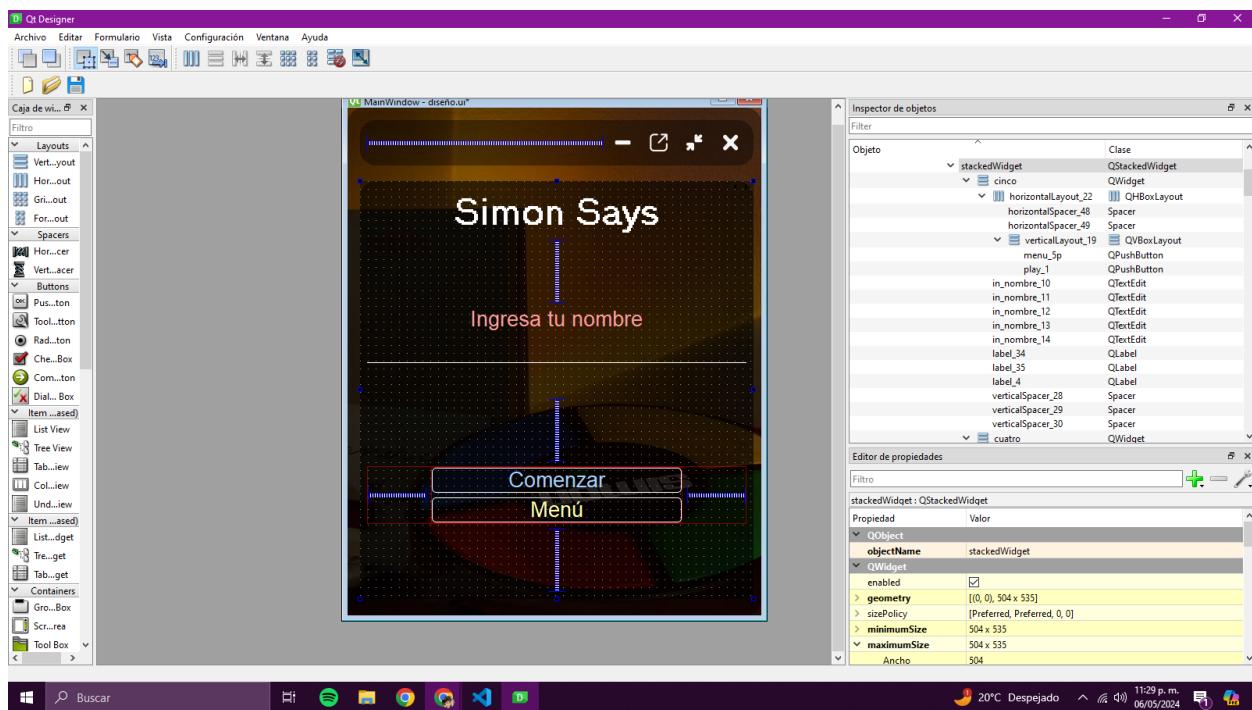
Una vez establecida la comunicación entre Python y Arduino, las posibilidades son casi ilimitadas. Desde controlar luces y motores hasta recopilar datos de sensores y enviarlos a una aplicación Python para su procesamiento y análisis, la comunicación serie abre un mundo de posibilidades para la creación de proyectos interactivos y conectados.

Desarrollo

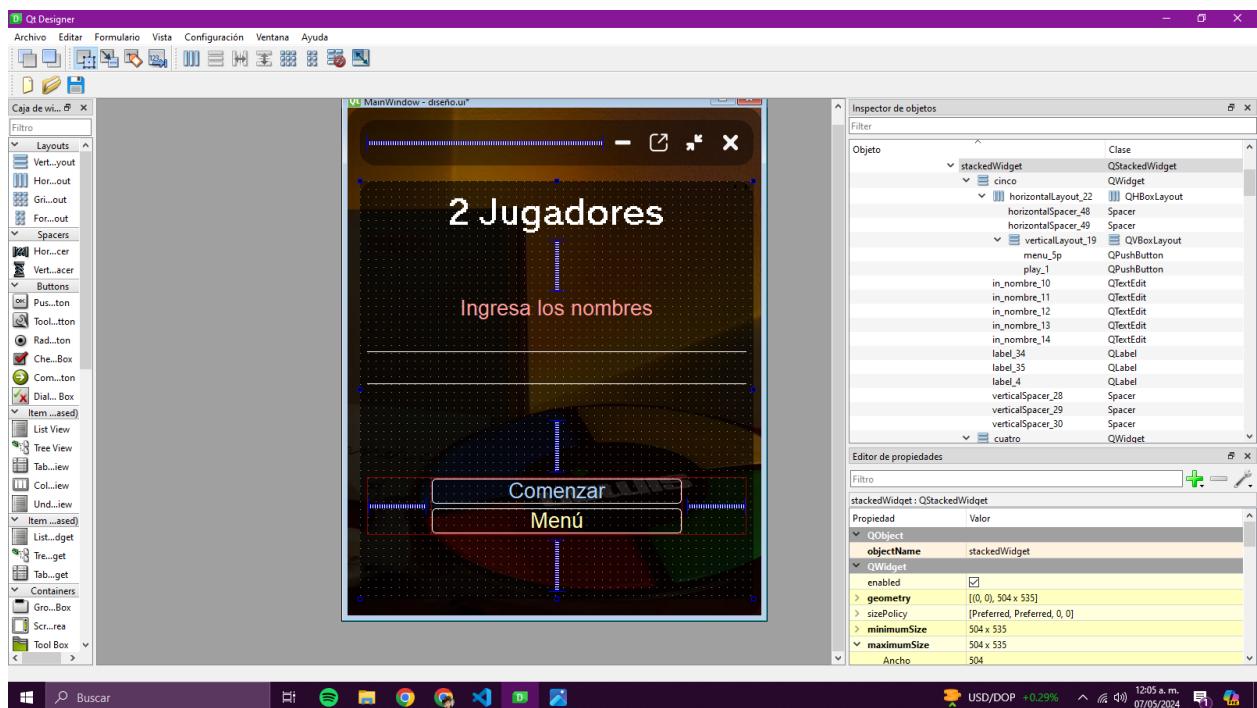
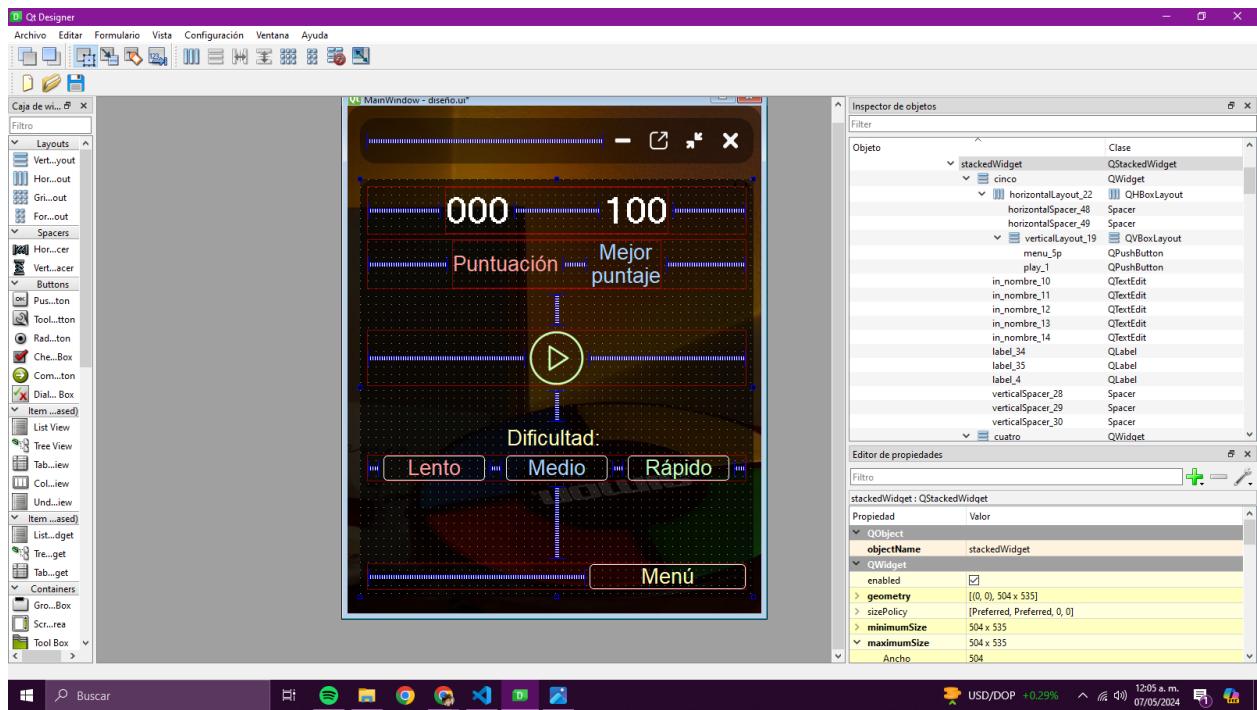
Elaboracion de la GUI

Para que nuestro código opere desde una GUI, primero debemos de estructurar la gui, hacerla de manera que podamos entenderla y manipularla facilmente, como podemos observar, existen 3 opciones en las cuales cada una representa un botón, 1 jugador y el extremo que sera multijugador, la ultima opción es salir.



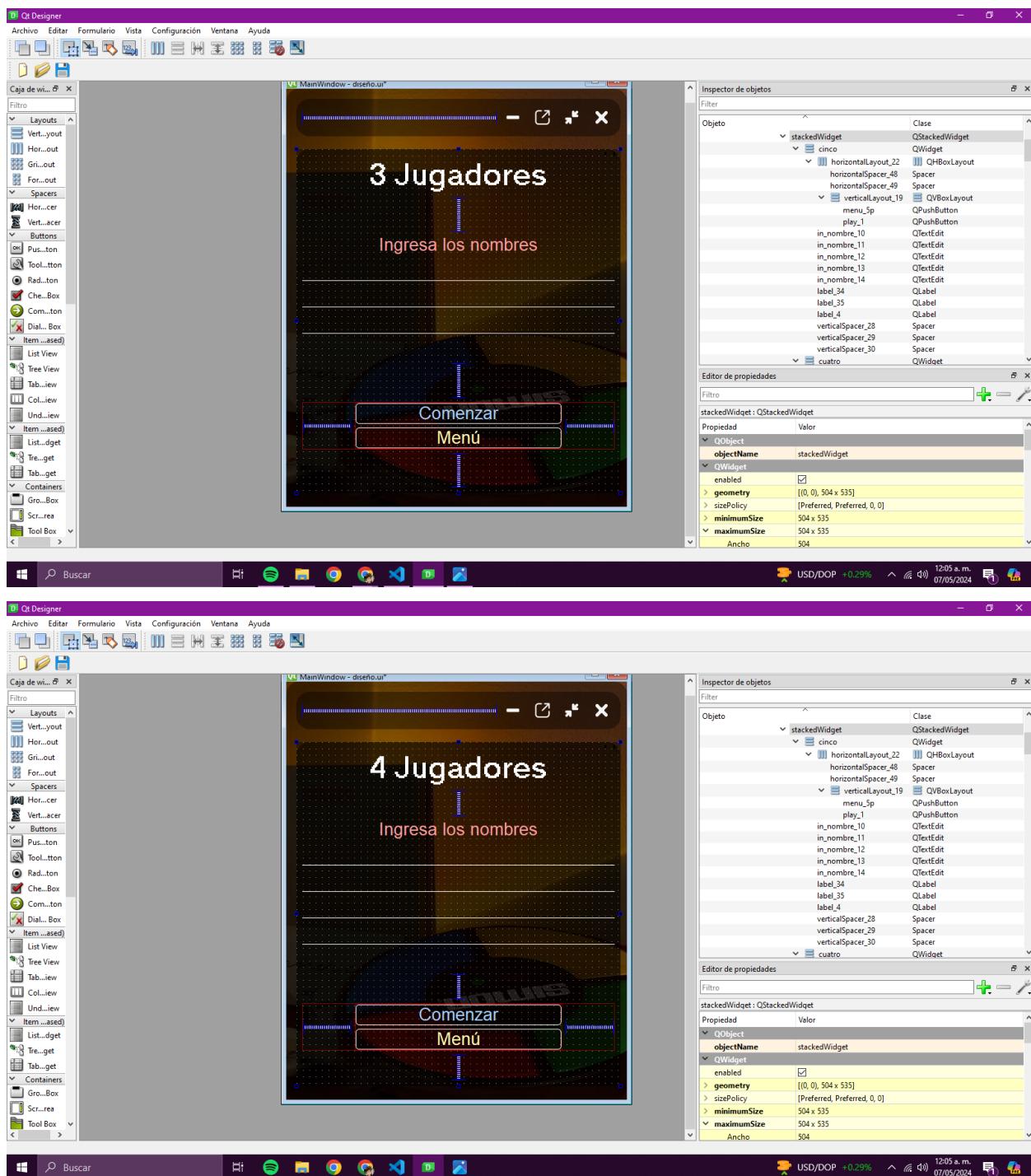


El stackWidget de simon says es para la modalidad de un solo jugador, la cual permite al usuario registrar su nombre y poder comenzar o regresar al menu, pero no funcionara hasta que se le ponga play en la siguiente pagina de ese StackWidget

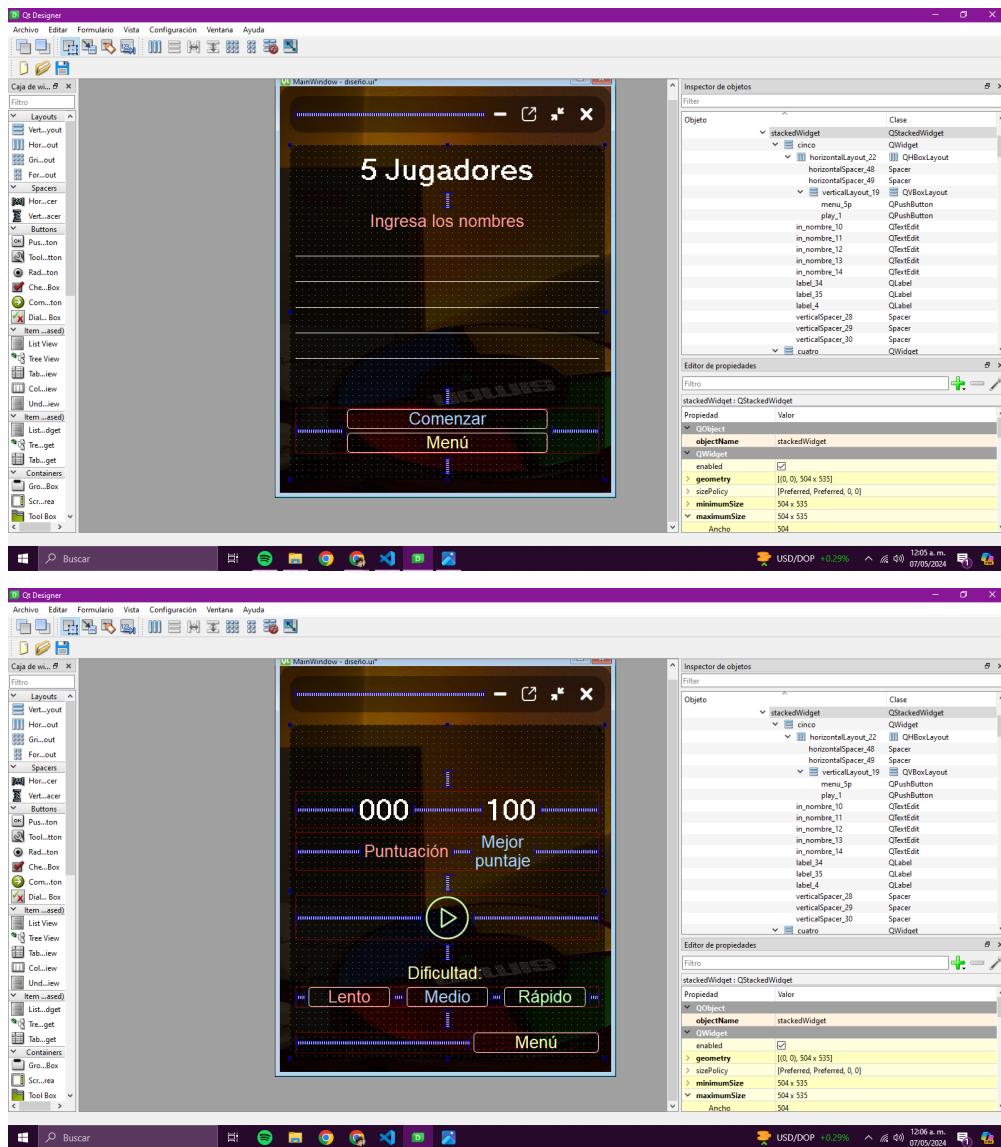


La página de la puntuación básicamente es la segunda principal, ya que nos indica el mejor puntaje , iniciar el juego poniéndole play y la dificultad del juego

En la otra página tenemos la versión de multijugador de 2 jugadores



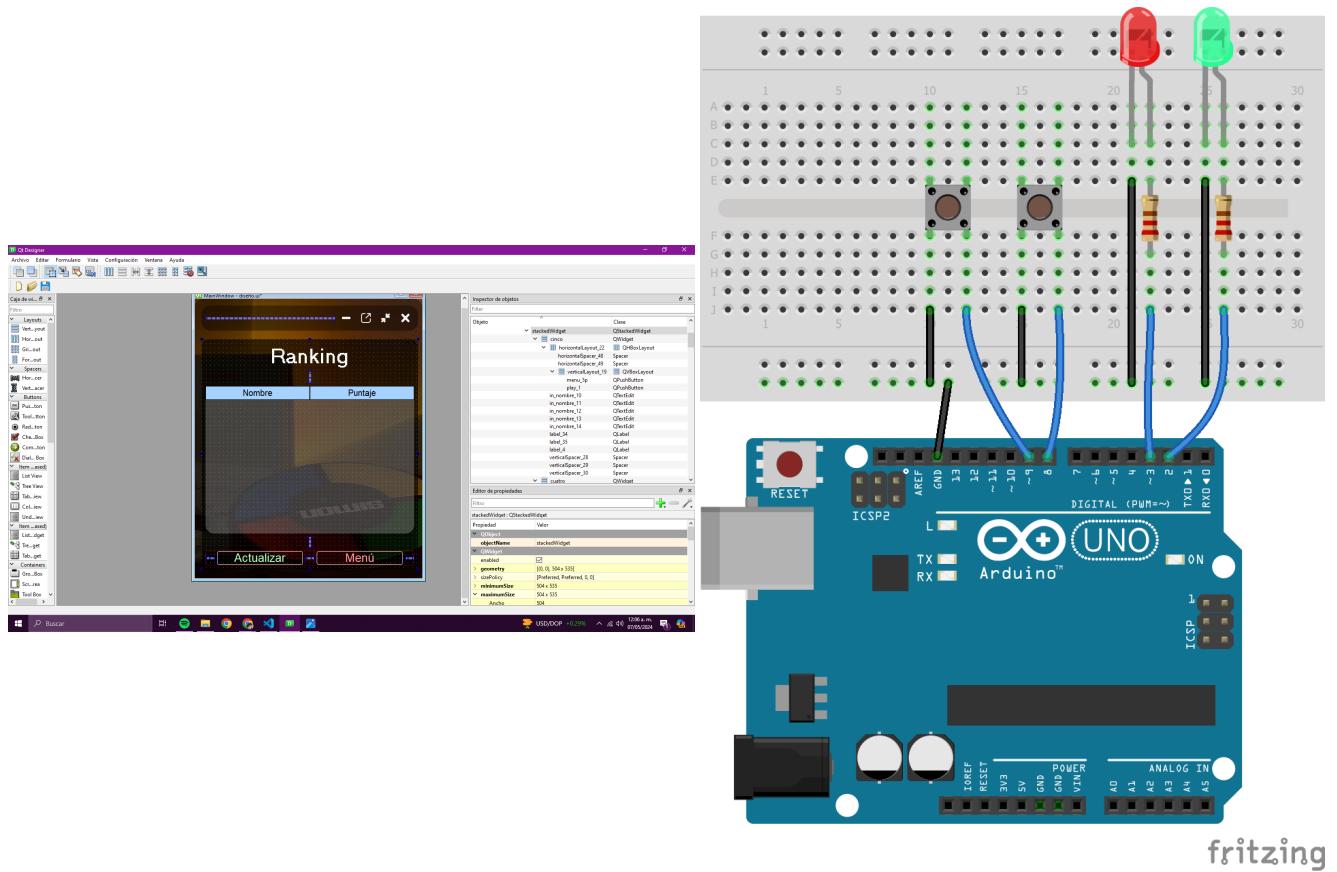
Aquí se pueden observar las páginas para 3 y 4 jugadores, donde nos pide en cada una que registremos los nombres de los respectivos números de jugadores, aparte de que después nos sacará a la página de play para comenzar el juego, cuando obviamente presionemos el botón comenzar, menu nos volverá al inicio, recordemos que para que exista un perdedor, simplemente al equivocarse en la secuencia podremos eliminar a cada uno y así sucesivamente hasta que prevalezca uno.



Tenemos nuevamente el modo multijugador pero en esta ocasión para 5 personas, con los mismo botones que las paginas anteriores, simplemente que aqui nos pedira los nombres de los 5 participantes. Y finalmente le damos la vuelta para que las pagianas nos muestren los resultados y el puntaje mas alto registrado. Cabe destacar que el servo quirara hacia el lado del jugador que le toque, como solo son 180 grados, por lo tanto seran esos 180 sobre el numero de jugadores que estan.

De nuestro lado izquierdo inferior podemos observar la ultima pagina, las cual nos dice los puntajes y demas valores para cuando se juega extremo (multijugador)

Del lado derecho podemos observar el diagrama de nuestro circuito en arduino, obviamente con mas botones y leds. pero es la misma funcionalidad, se prende un led, usted o el jugador debera de presionarlo para que la secuencia que se mando al arduino , sea la misma y de esa manera no pierda tal jugador y asi sucesivamente



The screenshot shows the PyCharm IDE interface with the following details:

- File Structure (Left):** Shows the project structure with files like `main_simon_dice.py`, `UI_diseño.py`, `UI_diseño.ui`, `ext.qrc`, `fondo.jpeg`, and `scores.txt`.
- Code Editor (Center):** Displays Python code for a dice game application. The code includes imports for `sys`, `random`, `time`, `pyfirmata`, `PyQt6.uic`, `QKeyEvent`, `QMouseEvent`, `QMainWindow`, `QApplication`, `QHeaderView`, `QTableWidgetItem`, `QMessageBox`, `QtCore`, and `QtWidgets`. It defines two classes: `JuegoThread` and `JuegoThread2`, which inherit from `QThread`. The `JuegoThread` class has a constructor that takes a `mainwindow` parameter and a `run` method that enters a loop to call `jugar_varios_integrantes` and check for a response. The `JuegoThread2` class has a similar constructor and `run` method.
- Bottom Status Bar:** Shows the current file as `main_simon_dice.py`, the line number as 22, the column number as 19, the space count as 4, the encoding as UTF-8, the file type as Python, the version as 3.12.2, and the bit width as 64-bit.

JuegoThread maneja el juego con múltiples jugadores, mientras que JuegoThread2 se encarga del juego para un solo jugador. Ambos subprocessos ejecutan un bucle infinito donde llaman a métodos específicos del objeto mainwindow, como jugarvariosintegrantes() o unsolojugador(), para gestionar la lógica del juego. Estos subprocessos se detienen cuando la secuencia generada por mainwindow no coincide con la respuesta esperada, indicando un error en el juego.

The screenshot shows a Python IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Toolbar:** Standard icons for file operations.
- Explorer:** Shows a tree view of files and folders. The current folder is 'PRACTICA6'. Files listed include: main_simon_dice.py, U1_diseño.py, diseno.ui, ext.qrc, fondo.jpeg, main_simon_dice.py (selected), prueba1.py, prueba2.py, prueba3.py, scores.txt, simon_dice_final.py, simon_dice.py, and U1_diseño.py.
- Editor:** Displays the code for `main_simon_dice.py`. The code defines a `JuegoThread2` class that extends `QThread`. It also defines a `MainWindow` class that extends `QMainWindow`. The `MainWindow` class sets its window flags to be frameless and has a grip size of 10. It contains several buttons and a signal-slot connection for each button's click event. A variable `self.numero_jugadores=6` is set. The code ends with a warning about the `inspect` module's lack of `getargspec`.
- Terminal:** Shows the command line with the path `E:\PITA\24-2(Tercer semestre)\Programación Avanzada\Práctica6` and the message `AttributeError: module 'inspect' has no attribute 'getargspec'. Did you mean: 'getargs'?`
- Bottom Bar:** Includes icons for file operations, a search bar, and system status information like temperature (20°C), battery level (Despejado), and date/time (12:34 a.m., 07/05/2024).

Establecemos la ventana principal de la aplicación y define su interfaz de usuario. Configura la ventana sin bordes y con una opacidad completa. Luego, carga la interfaz de usuario desde un archivo externo y configura los botones para realizar diferentes acciones, como cambiar entre menús o iniciar juegos. Además, maneja eventos como minimizar, expandir o cerrar la ventana. También inicializa variables para almacenar nombres y puntajes de jugadores, junto con la velocidad del juego. Finalmente, conecta señales de otros widgets a funciones que realizan diversas acciones, como actualizar el ranking o iniciar juegos multijugador.

The screenshot shows a Python IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Toolbar:** Back, Forward, Refresh, Home, Save, Run, Stop, Help.
- Explorer:** Shows a file tree with files like `main_simon_dice.py`, `U_idiseño.py`, and `UI_diseño.py`.
- Editor:** Displays the code for `UI_diseño.py`. The code defines a class `MainWindow` that inherits from `QMainWindow`. It initializes a window with a title bar, a menu bar, and a central area. It connects various buttons to methods like `ir_nombres`, `extremo.clicked.connect(self.ir_nombres)`, and `actualizar.clicked.connect(self.act_ranking)`. It also handles button clicks for play levels (play_1 to play_5) and speeds (lento, rápido, medio).
- Terminal:** Shows the command line with the message: `AttributeError: module 'inspect' has no attribute 'getargspec'. Did you mean: 'getargs'?`
- Status Bar:** Shows the path `E:\UPIITA\24-2(Tercer semestre)\Programación Avanzada\Práctica6`, line count (Ln 22), column count (Col 19), spaces (Spaces 4), encoding (UTF-8), CRLF, Python version (3.12.2 64-bit), and system information (20°C Despejado, 12:34 a.m., 07/05/2024).

De igual manera seguimos conectando botones a la interfaz

conecta las señales emitidas por los botones iniciarmultijugador e iniciar1jugador a las funciones iniciarjuego e iniciarpara1jugador, respectivamente. Además, inicializa la comunicación con un dispositivo Arduino conectado al puerto COM1. Configura el movimiento de la ventana a través del evento mouseMoveEvent y comienza un iterador para comunicarse con la placa Arduino. Luego, inicializa los pines de entrada y salida del Arduino para controlar botones y LEDs. Finalmente, crea instancias de las clases JuegoThread y JuegoThread2 para manejar los hilos del juego. Las funciones minimizar, reducir y expandir controlan el comportamiento de la ventana cuando se minimiza, reduce o expande, respectivamente

The screenshot shows a Python development environment with the following details:

- File Explorer:** Shows files in the main_simon_dice project, including `main_simon_dice.py`, `UI_diseño.py`, and `UI_diseño.ui`.
- Code Editor:** Displays the `main_simon_dice.py` file. The code implements a dice game application with a GUI. It includes functions for ranking scores, handling window events, and managing mouse interactions.
- Terminal:** Shows the command `PS E:\UIPIITA\24-2(Tercer semestre)\Programación Avanzada\Práctica6`.
- Bottom Status Bar:** Provides information about the current line (Line 22), column (Col 19), and other settings like spaces, tabs, and encoding.

La función actranking actualiza el registro de puntajes en un archivo de texto. Primero, toma el nombre ingresado por el usuario y el puntaje actual. Luego, escribe esta información en un archivo llamado 'scores.txt'. Después, lee el contenido de este archivo y lo muestra en una tabla para que el usuario pueda ver el ranking actualizado. Las otras funciones (resizeEvent, mousePressEvent, moverventana) están relacionadas con la manipulación y el comportamiento de la ventana

La función ‘iniciarpara1jugador’ inicia el juego para un solo jugador si el hilo correspondiente no está en ejecución.

La función ‘unsolojugador’ ejecuta el juego para un solo jugador, donde se genera una secuencia de juego, se envía al jugador, se espera su respuesta, se actualiza el puntaje y se verifica si la respuesta es correcta. Si la respuesta es incorrecta, el juego se detiene y se reinicia.

The screenshot shows a Python IDE interface with the following details:

- File Explorer:** Shows files in the current workspace, including `main_simon_dice.py`, `Ui_diseño.py`, `main_simon_dice.py` (with a warning), `UI_juego0thread.py`, and several UI files like `diseño.ui` and `ext.qrc`.
- Code Editor:** Displays the `main_simon_dice.py` file. The code implements a Simon Says game with a thread for handling player input. It includes functions for starting the game, handling player turns, and checking answers.
- Terminal:** Shows the command line output and errors. An `AttributeError` is present, indicating a problem with the `inspect` module.
- Status Bar:** Provides information about the current session, including the line and column numbers, spaces used, encoding, and the Python version being used.

La función ‘iniciarpara1jugador’ inicia el juego para un solo jugador si el hilo correspondiente no está en ejecución.

La función ‘unsolojugador’ ejecuta el juego para un solo jugador, donde se genera una secuencia de juego, se envía al jugador, se espera su respuesta, se actualiza el puntaje y se verifica si la respuesta es correcta. Si la respuesta es incorrecta, el juego se detiene y se reinicia.

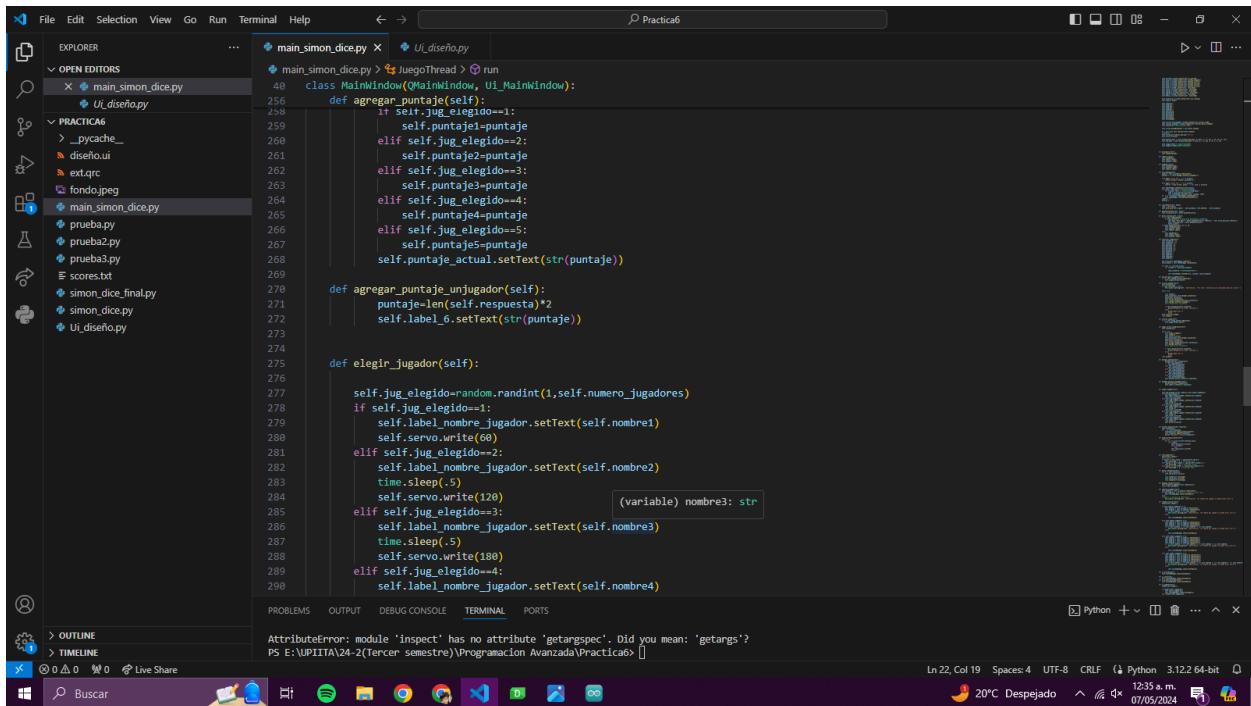
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like `main_simon_dice.py`, `UI_diseño.py`, and `simon_dice_final.py`.
- Code Editor:** Displays the `main_simon_dice.py` file containing Python code for a game. The code includes functions for initializing the game, playing with multiple players, and calculating scores.
- Terminal:** Shows the command prompt with an `AttributeError` message about the `inspect` module.
- Status Bar:** Provides information about the current file (22 lines, 19 spaces), the terminal (Python), and the system (Windows 10, 20°C).

La función ‘jugarvariosintegrantes’ ejecuta el juego para múltiples jugadores, donde se elige un jugador aleatorio, se genera una secuencia de juego, se envía al jugador, se espera su respuesta, se actualiza el puntaje del jugador y se verifica si la respuesta es correcta. Si la respuesta es incorrecta, el juego se detiene y se reinicia.

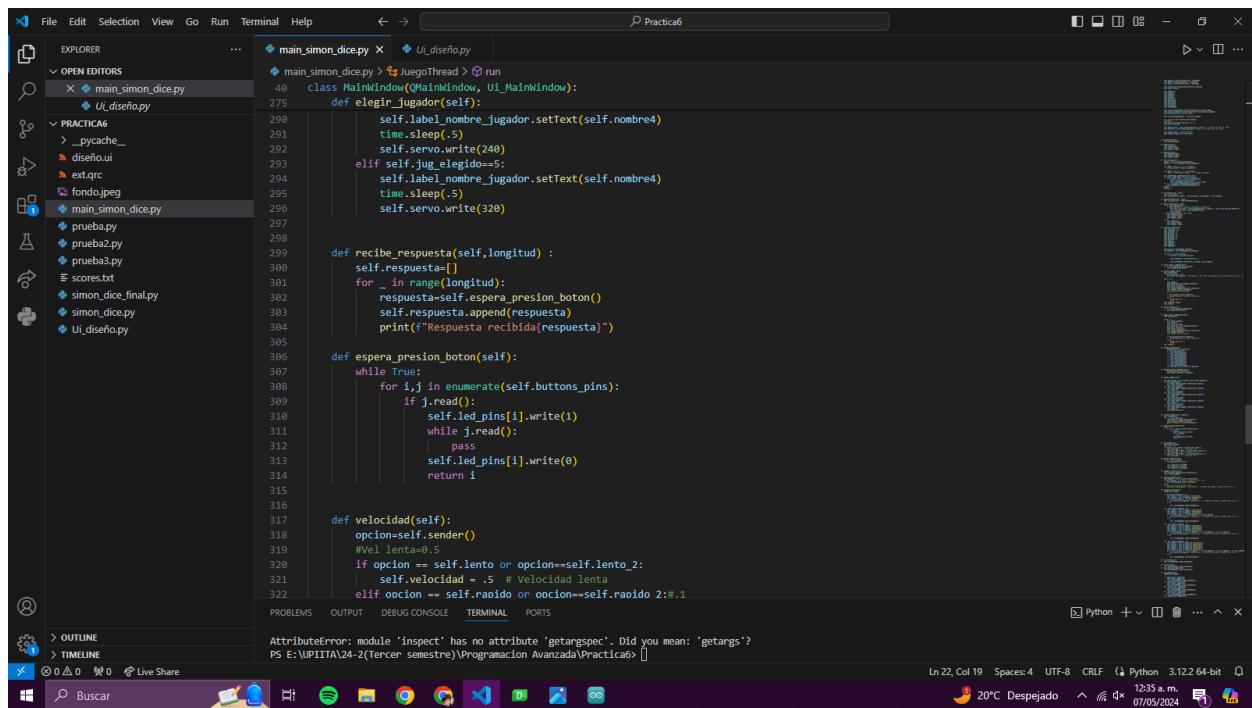
La función ‘agregarpuntaje’ asigna el puntaje correspondiente al jugador seleccionado y lo muestra en la interfaz.

La función ‘agregarpuntajeunjugador’ asigna el puntaje al jugador único y lo muestra en la interfaz.



```
File Edit Selection View Go Run Terminal Help ← → 🔍 Practica6
OPEN EDITORS
  main_simon_dice.py x  U_diseño.py
    U_diseño.py
      PRACTICA6
        _pycache_
        diseno.ui
        ext.qrc
        fondo.jpeg
      main_simon_dice.py
        prueba1.py
        prueba2.py
        prueba3.py
      scores.txt
      simon_dice_final.py
      simon_dice.py
      U_diseño.py
  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
AttributeError: module 'inspect' has no attribute 'getargspec'. Did you mean: 'getargs'? PS E:\PITTA\24-2(Tercer semestre)\Programación Avanzada\Práctica6 []
Ln 22 Col 19 Spaces:4 UTF-8 CRLF Python 3.12.2 64-bit 20°C Despejado 12:35 a.m. 07/05/2024
```

La función ‘elegirjugador’ elige aleatoriamente a uno de los jugadores disponibles y actualiza la interfaz para mostrar su nombre. Además, envía una señal al Arduino para controlar un servo que indica al jugador seleccionado. El servo se mueve a diferentes ángulos según el jugador elegido, lo que podría estar relacionado con algún tipo de indicador físico en el juego.



```
File Edit Selection View Go Run Terminal Help ← → 🔍 Practica6
EXPLORER OPEN EDITORS
main_simon_dice.py x U_diseño.py
  main_simon_dice.py > JuegoITread > run
    40 class MainWindow(QMainWindow, ui_MainWindow):
    275     def elegir_jugador(self):
    290         self.label_nombre_jugador.setText(self.nombre4)
    291         time.sleep(.5)
    292         self.servo.write(248)
    293         elif self.jug_elegido==5:
    294             self.label_nombre_jugador.setText(self.nombre4)
    295             time.sleep(.5)
    296             self.servo.write(328)

    297     def recibe_respuesta(self,longitud):
    298         self.respuesta=[]
    299         for _ in range(longitud):
    300             respuesta=self.espera_presion boton()
    301             self.respuesta.append(respuesta)
    302             print("Respuesta recibida[respuesta]")
    303
    304     def espera_presion boton(self):
    305         while True:
    306             for i,j in enumerate(self.buttons_pins):
    307                 if j.read():
    308                     self.led_pins[i].write(1)
    309                     while j.read():
    310                         pass
    311                     self.led_pins[i].write(0)
    312             return i
    313
    314
    315
    316     def velocidad(self):
    317         opcion=self.sender()
    318         #Vel lenta=0.1
    319         if opcion == self.lento or opcion==self.lento_2:
    320             self.velocidad = .5 # Velocidad lenta
    321         elif opcion == self.raido or opcion==self.raido_2:#.1
    322             self.velocidad = 1 # Velocidad rápida

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
AttributeError: module 'inspect' has no attribute 'getargspec'. Did you mean: 'getargs'?
PS E:\PITA\24-2(Tercer semestre)\Programación Avanzada\Práctica6> []
Ln 22, Col 19 Spaces:4 UTF-8 CRLF Python 3.12.2 64-bit
20°C Despejado 12:35 a.m. 07/05/2024
```

La función ‘reciberespuesta’ espera y registra la respuesta del jugador presionando los botones. Utiliza la función ‘esperapresionboton’ para esperar hasta que un botón sea presionado y luego registra el índice del botón presionado. Mientras tanto, enciende el LED correspondiente al botón presionado durante la espera. Una vez que se presiona un botón, el LED correspondiente se apaga y se devuelve el índice del botón presionado. Esto permite registrar la secuencia de botones presionados como respuesta del jugador.

The screenshot shows a Python IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Terminal:** Shows the command PS E:\UPIITA\24-24\Tercer semestre\Programación Avanzada\Práctica6.
- Code Editor:** The main editor window displays the file `main_simon_dice.py`. The code implements a Simon Says game with a graphical user interface (UI) designed in Qt using `PyQt5`.
- UI Code:** The UI code is located in `UIdiseño.py`, defining a `JuegoThread` class that runs the game logic.
- Project Structure:** The left sidebar shows the project structure with files like `main_simon_dice.py`, `UIdiseño.py`, `__pycache__`, `diseño.ui`, `ext.qrc`, `fondo.jpeg`, and `scores.txt`.
- Output and Status:** The bottom status bar indicates the current line (Ln 22), column (Col 19), spaces (Spaces: 4), encoding (UTF-8), and file type (Python). It also shows the date and time (07/05/2024, 12:35 a.m.).

```

File Edit Selection View Go Run Terminal Help ← → J:\Practica6
OPEN EDITORS
  main_simon_dice.py x  U_idiseño.py
  U_idiseño.py > JuegoThread > run
  40 class MainWindow(QMainWindow, Ui_MainWindow):
    def ventana_iniciar(self):
        347     if not self.nombre1 or not self.nombre2:
        348         QMessageBox.warning(self, "Advertencia", "El nombre del jugador no puede estar vacío.")
        349     else:
        350         self.stackedWidget.setCurrentIndex(8)
        351
        352     elif self.numero_jugadores == 3:
        353         self.nombre1 = self.in_nombre_3.toPlainText()
        354         self.nombre2 = self.in_nombre_4.toPlainText()
        355         self.nombre3 = self.in_nombre_5.toPlainText()
        356     if not self.nombre1 or not self.nombre2 or not self.nombre3:
        357         QMessageBox.warning(self, "Advertencia", "El nombre del jugador no puede estar vacío.")
        358     else:
        359         self.stackedWidget.setCurrentIndex(8)
        360
        361     elif self.numero_jugadores == 4:
        362         self.nombre1 = self.in_nombre_6.toPlainText()
        363         self.nombre2 = self.in_nombre_7.toPlainText()
        364         self.nombre3 = self.in_nombre_8.toPlainText()
        365         self.nombre4 = self.in_nombre_9.toPlainText()
        366     if not self.nombre1 or not self.nombre2 or not self.nombre3 or not self.nombre4:
        367         QMessageBox.warning(self, "Advertencia", "El nombre del jugador no puede estar vacío.")
        368     else:
        369         self.stackedWidget.setCurrentIndex(8)
        370
        371     elif self.numero_jugadores == 5:
        372         self.nombre1 = self.in_nombre_10.toPlainText()
        373         self.nombre2 = self.in_nombre_11.toPlainText()
        374         self.nombre3 = self.in_nombre_12.toPlainText()
        375         self.nombre4 = self.in_nombre_13.toPlainText()
        376         self.nombre5 = self.in_nombre_14.toPlainText()
        377
        378
        379
        380
        381
        382
        383
        384
        385
  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
  OUTLINE TIMELINE
  Live Share
  Buscar
  Python + v 12:35 a.m. 20°C Despejado 07/05/2024
  Ln 22 Col 19 Spaces:4 UTF-8 CRLF Python 3.12.2 64-bit

```

La función ‘velocidad’ ajusta la velocidad del juego según la opción seleccionada por el usuario. Luego, la función ‘enviarsecuencia’ enciende y apaga los LEDs correspondientes a la secuencia generada, con una pausa de tiempo determinada por la velocidad. ‘agregarsecuencia’ genera una nueva secuencia de botones aleatorios.

Las funciones ‘ventanaljugador’ y ‘ventanainiciar’ manejan el flujo de la interfaz de usuario para iniciar el juego con uno o varios jugadores, respectivamente. Verifican si se ha ingresado el nombre del jugador y muestran un mensaje de advertencia si está vacío. Dependiendo del número de jugadores seleccionados, establecen el índice actual de la pila de widgets para avanzar en la interfaz de usuario.

```

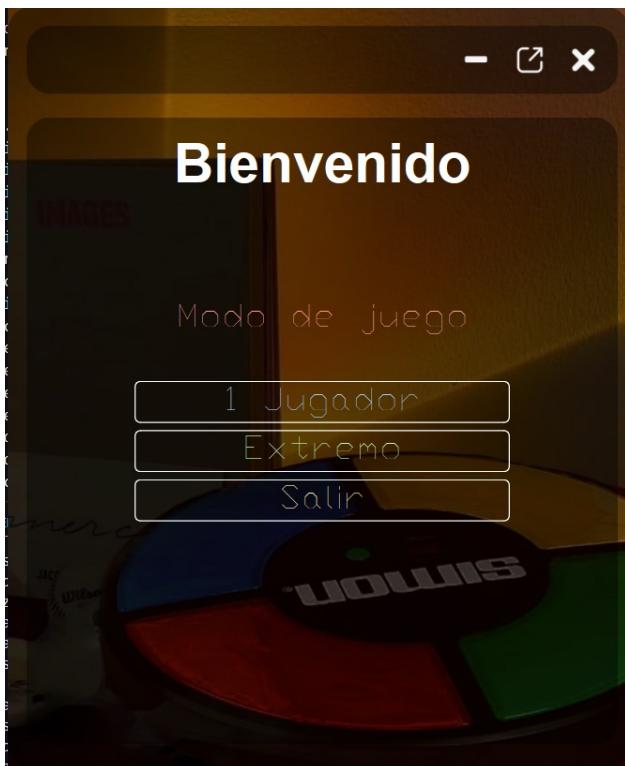
File Edit Selection View Go Run Terminal Help ← → 🔍 Practica6
EXPLORER OPEN EDITORS
  main_simon_dice.py x  U_diseño.py
  U_diseño.py
  PRACTICA6
  _pycache_
  diseno.ui
  ext.qrc
  fondo.jpeg
  main_simon_dice.py
  prueba1.py
  prueba2.py
  prueba3.py
  scores.txt
  simon_dice_final.py
  simon_dice.py
  U_diseño.py
  40   class MainWindow(QMainWindow, Ui_MainWindow):
  347       def ventana_iniciar(self):
  387           if self.sender() == self.pushButton_iniciar:
  388               QMessageBox.warning(self, "Advertencia", "El nombre del jugador no puede estar vacío.")
  389           else:
  390               self.stackedWidget.setCurrentIndex(8)
  391
  392       def ir_extremo(self):
  393           self.stackedWidget.setCurrentIndex(2)
  394
  395       def ir_rank(self):
  396           self.stackedWidget.setCurrentIndex(9)
  397       def ir_a_menu_principal(self):
  398           self.stackedWidget.setCurrentIndex(0)
  399
  400       def ir_nombres(self):
  401           sender=self.sender()
  402
  403           if sender==self.jugadores1:
  404               self.numero_jugadores=1
  405               self.stackedWidget.setCurrentIndex(1)
  406           elif sender==self.jugadores2:
  407               self.numero_jugadores=2
  408               self.stackedWidget.setCurrentIndex(4)
  409           elif sender==self.jugadores3:
  410               self.numero_jugadores=3
  411               self.stackedWidget.setCurrentIndex(5)
  412           elif sender==self.jugadores4:
  413               self.numero_jugadores=4
  414               self.stackedWidget.setCurrentIndex(6)
  415           elif sender==self.jugadores5:
  416               self.numero_jugadores=5
  417               self.stackedWidget.setCurrentIndex(7)
  418
  419
  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
  AttributeError: module 'inspect' has no attribute 'getargspec'. Did you mean: 'getargs'?
  PS E:\PITA\24-2(Tercer semestre)\Programación Avanzada\Práctica6> []
  Ln 22, Col 19  Spaces: 4  UTF-8  CRLF  Python 3.12.2 64-bit
  20°C Despejado  12:35 a.m.  07/05/2024
  Buscar
  
```

Las funciones ‘irextremo’, ‘irrank’ e ‘iramenuprincipal’ cambian el índice actual de la pila de widgets para redirigir la interfaz de usuario a diferentes pantallas: la pantalla de juego extremo, la pantalla de ranking y la pantalla del menú principal, respectivamente.

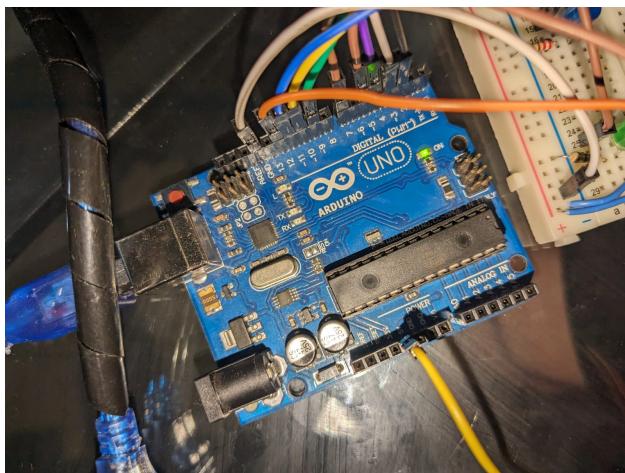
La función ‘irnombres’ determina el número de jugadores seleccionados según el botón de jugador presionado y redirige la interfaz de usuario a la pantalla correspondiente para que los jugadores ingresen sus nombres.

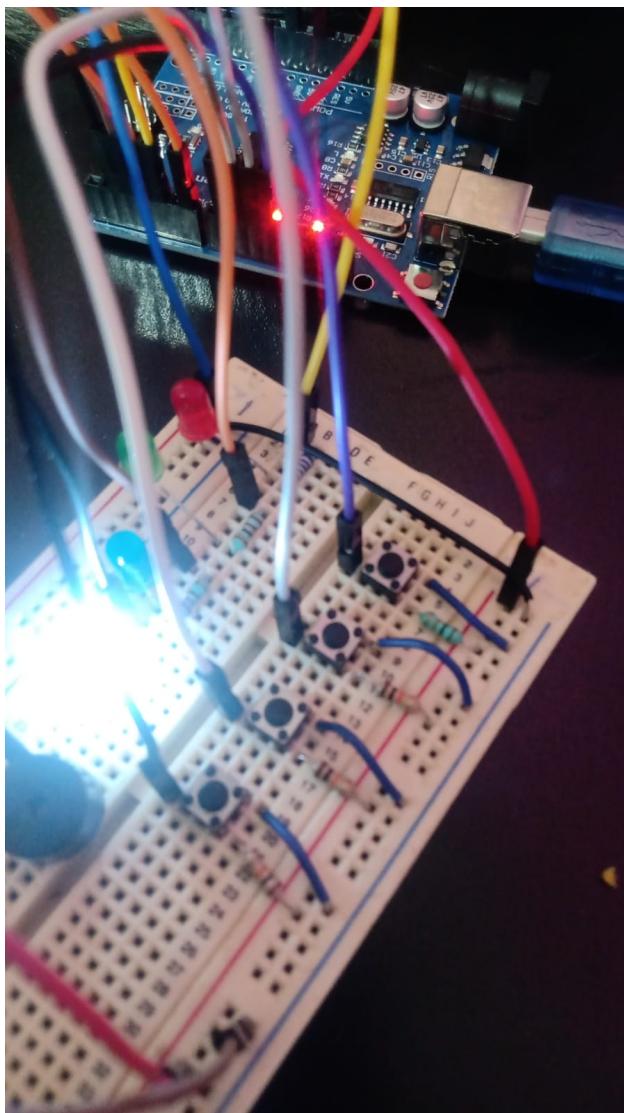
Ejecución

Cuando ejecutamos nuestro código, nos parece la pantalla principal de simon, y se despliega como se indica, todo aquí depende de la interacción que se tenga con el usuario



Podemos observar las conexiones del arduino y la conexión del puerto serial, ya que sin este no puede cargar el programa ya que se le asignó una conexión serial





Finalmente observamos el encendido y apagado de los leds de la secuencia que se mando a imprimir...

Conclusiones

Usar una interfaz gráfica de usuario (GUI) en Python junto con Arduino ha sido una experiencia fascinante para nosotros. Ha ampliado significativamente las posibilidades de lo que podemos lograr con nuestros proyectos, agregando una capa visual y de interacción que antes no teníamos.

La interactividad mejorada es uno de los aspectos más destacados. La combinación de Python y Arduino nos ha permitido crear interfaces de usuario que responden a la entrada del usuario y a las lecturas de sensores en tiempo real. Esto ha mejorado enormemente la experiencia de usuario y ha hecho que nuestros proyectos sean más atractivos y fáciles de usar.

Además, la capacidad de visualizar datos recopilados por los sensores de Arduino en una GUI ha sido muy útil. Podemos representar gráficamente los datos de manera clara y comprensible, lo que facilita el monitoreo y análisis de los mismos, especialmente en proyectos relacionados con la monitorización del medio ambiente o la domótica.

La posibilidad de control remoto y automatización es otro aspecto importante. Integrar una GUI con Arduino nos ha permitido controlar dispositivos físicos y sistemas embebidos de forma remota a través de la red. Esto ha abierto la puerta a la automatización de tareas y procesos, mejorando la eficiencia y comodidad en una variedad de situaciones.

El prototipado rápido también se ha beneficiado enormemente de esta combinación. Podemos crear interfaces de usuario rápidamente para probar y demostrar conceptos antes de invertir tiempo en el desarrollo de una solución completa. Esto ha sido invaluable para iterar y refinar nuestras ideas de manera rápida y efectiva.

En términos de flexibilidad y escalabilidad, Python y Arduino son herramientas altamente versátiles. Esta combinación nos permite adaptar nuestros proyectos a una amplia variedad de aplicaciones y requisitos, ya sea trabajando en un proyecto simple o en uno más complejo.

Referencias

ditronikx. (2022, June 5). comunicación y control con python y arduino mediante pyserial y visual code studio [Video]. YouTube. <https://www.youtube.com/watch?v=RuCE6PLyyym0>

Julian Ruiz. (2021, January 14). Como conectar PYTHON con ARDUINO [2022] — Julian Ruiz [Video]. YouTube. <https://www.youtube.com/watch?v=fCuFDW1RoxI>

etwork Warriors. (2021, September 27). Comunicación Python - Arduino (Pycharm) pyserial — Enviar y Recibir Datos [Video]. YouTube. https://www.youtube.com/watch?v=_DMDjNYNVI