

UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE INGENIERIA ELÉCTRICA Y ELECTRÓNICA



SUDOKU CON PYTHON
EN ORIENTADO A OBJETOS

INTEGRANTES:

- BUENO OTAÑO, Luis Sebastián (20232247J)
- SANCHEZ FLORES, Sandro Steven (20231403H)
- VELASQUE PUMA, Brayan (20230411G)

SECCIÓN:

Q

DOCENTE:

Tello Canchapoma, Yury Oscar

Índice

Introducción	3
Objetivos	4
Antecedentes	4
Diagrama UML	6
Identificación de clases	7
Relaciones entre clases	7
Flujo general	7
Estructuración del código	7
Paso 1: Estructuración de las clases métodos y atributos	7
Paso 2: Rellenar las clases	12
Paso 3: Código listo para jugar Sudoku	15
Pruebas automatizadas	22
Conclusión	25
Referencias bibliográficas	26

Introducción

El “Sudoku” es uno de los juegos de lógica más populares y desafiantes a nivel mundial, conocido por su capacidad para desarrollar habilidades cognitivas como la resolución de problemas, la lógica deductiva y el pensamiento estratégico. El objetivo del juego es completar una cuadrícula de 9x9 con números del 1 al 9, de manera que cada fila, cada columna y cada subcuadrícula 3x3 contengan los números sin repetir ningún valor. Si bien resolver un Sudoku manualmente puede ser una tarea entretenida, la automatización de este proceso utilizando métodos computacionales abre nuevas oportunidades para explorar algoritmos de inteligencia artificial y optimización.

Esta monografía tiene como propósito explorar la creación de un juego Sudoku utilizando el lenguaje de programación Python, con un enfoque en el uso del paradigma orientado a objetos (POO) y técnicas de aprendizaje por refuerzo (Reinforcement Learning) para resolver el rompecabezas de manera autónoma. El paradigma orientado a objetos permite estructurar el código de manera modular, organizando los elementos del Sudoku como objetos con atributos y comportamientos específicos, lo que facilita la implementación, mantenimiento y expansión del programa.

El uso de Reinforcement Learning introduce un enfoque más avanzado, permitiendo que el algoritmo aprenda de la experiencia y mejore su capacidad para resolver el Sudoku a través de la interacción con su entorno. En lugar de seguir una serie de pasos predefinidos, el algoritmo busca obtener recompensas por tomar las decisiones correctas en cada paso del juego, optimizando su desempeño en la resolución del rompecabezas con el tiempo.

Objetivos

- Realizar un juego de Sudoku ayuda a la estimulación mental, es decir, mejora la agilidad mental y lógica. Además, este juego también contribuye con la reducción de estrés
- Cuando se realice el código uno de los objetivos es obtener un código modular para que sea fácil de mantener.
- Mejorar las habilidades cognitivas de los jugadores para desarrollar competencias en la planificación y toma de decisiones. También para fomentar a la perseverancia y paciencia de los usuarios.
- Crear un código funcional con algoritmos robustos para asegurar la eficiencia del programa.
- Someter el programa a una serie de pruebas con el fin de asegurar la calidad del mismo y facilitar futuras actualizaciones y mantenimiento.

Antecedentes

Angarita (2021), Universidad Nacional de Colombia, trabajo de investigación titulado "Videojuego para el aprendizaje de lógica de programación". El objetivo principal del trabajo de investigación es desarrollar un videojuego para dispositivos móviles que facilite el aprendizaje de la lógica de programación en estudiantes de básica secundaria. Se busca utilizar el videojuego como una herramienta didáctica que refuerce los conceptos de lógica y estructuras computacionales, haciendo el aprendizaje más atractivo y accesible para los estudiantes. Las conclusiones del estudio indican que el videojuego desarrollado facilita el aprendizaje de los conceptos de programación estructurada de una manera divertida y con un grado de dificultad alto. La validación realizada con un grupo de estudiantes mostró un amplio interés en la programación a través de videojuegos, considerándolo un recurso didáctico efectivo. Además, se sugiere que el uso de juegos serios en la educación puede hacer que la programación sea más accesible y menos intimidante para los estudiantes, lo que puede contribuir a mejorar su comprensión de conceptos complejos de programación en un contexto lúdico.

Pérez (2008) realizó una investigación sobre juego didáctico del sudoku y su importancia en el proceso del razonamiento lógico matemático, en alumnos del 6° y 7° grado de educación básica en la "Escuela mixta de parroquia Eloy Alfaro de Manta". La población estudiada estuvo conformada por 23 alumnos entre varones y mujeres. Con una investigación de carácter experimental. En donde llegó a esta conclusión: "los profesores en su labor pedagógica no incluyen los juegos sudokus como herramienta educativa". Existe gran deficiencia en los profesores sobre el conocimiento del juego sudoku y su aplicación, emplean con menor frecuencia los juegos sudoku en el aula.

Arias (2019). Realizó una de investigación sobre: “El desarrollo del juego sudoku y su relación con el pensamiento lógico matemático en la Institución educativa Pedro Sánchez Gavidia de Huánuco, periodo 2017”. Este estudio estuvo constituido por una muestra representativa con 23 escolares, siendo un trabajo de clase aplicada y explicativa, con diseño experimental. De este trabajo se consigue: con la prueba Z, como métodos estadísticos de comprobación. Se logró concluir que el uso de este juego incide positivamente en los resultados de Razonamiento en matemática, obteniéndose como resultados el valor $Z = 11,86$ puntos, el mismo que es superior a la Z crítica de 1,64 puntos, lo cual significa que con el uso de este juego y el análisis posterior a la investigación nos permite estimular y potenciar las habilidades de razonamiento creativo en el alumnado, que fue el objeto de estudio.

Diagrama UML

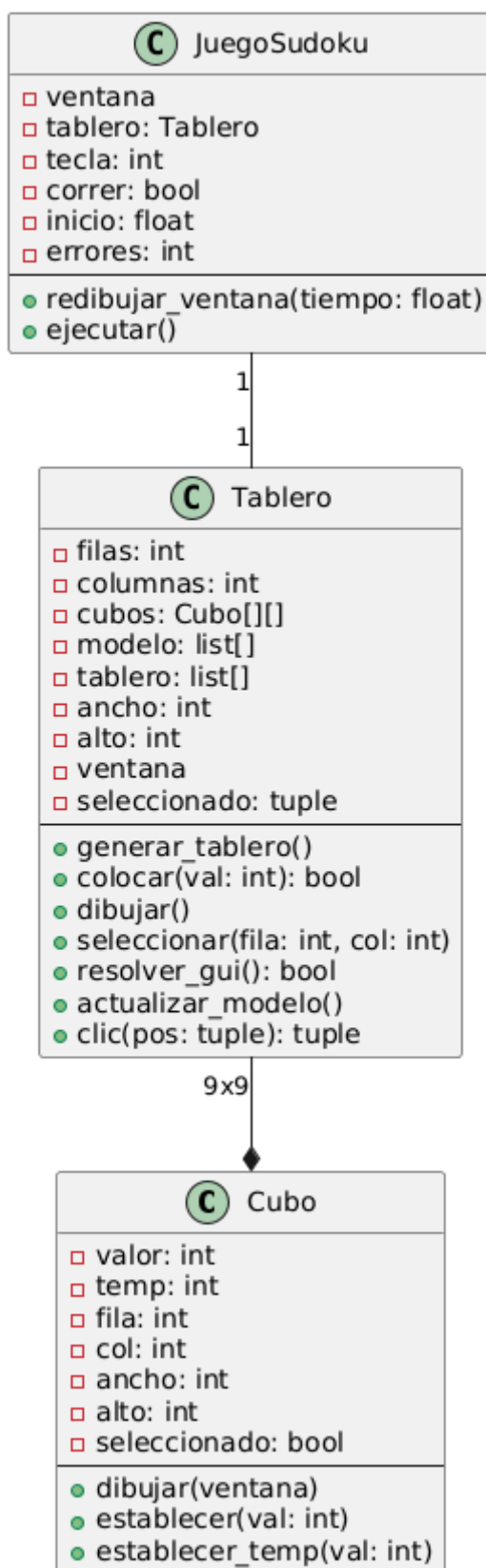


Imagen 1: Diagrama UML del juego Sudoku

El diagrama UML fue creado a partir del análisis del código del proyecto de Sudoku desarrollado en Python. Se utilizó un enfoque orientado a objetos para identificar las clases principales y sus relaciones.

- **Identificación de Clases:**

- Se identificaron tres clases principales:
 - **Cubo**: Representa cada celda individual del tablero.
 - **Tablero**: Gestiona la lógica del Sudoku y contiene una matriz de objetos **Cubo**.
 - **JuegoSudoku**: Administra la interfaz gráfica y el flujo general del juego.

- **Relaciones entre Clases:**

- **JuegoSudoku** ↔ **Tablero**: Asociación directa; **JuegoSudoku** administra el juego y utiliza el **Tablero** como base lógica.
- **Tablero** ↔ **Cubo**: Composición; el **Tablero** está compuesto por una matriz fija de 81 **Cubo**, cada uno modelando una celda del Sudoku.

- **Flujo General:**

1. **JuegoSudoku** inicializa un objeto de la clase **Tablero**.
2. **Tablero** crea 9x9 instancias de la clase **Cubo** y gestiona sus valores y estados.
3. **JuegoSudoku** interactúa con **Tablero**, que a su vez actualiza los **Cubo** y refleja los cambios en la interfaz gráfica.

Estructuración del juego:

Paso 1: Estructuración de las clases, métodos y atributos

Como primer paso al realizar este proyecto y como estamos utilizando el paradigma orientado a objetos en Python, debemos estructurar de manera eficiente sus clases, métodos y atributos siguiendo la estructuración del diagrama UML, para poder mantener el código de forma ordenada.

Código:

```
import pygame
import random
import time
pygame.font.init()

class Cubo:
    filas = 9
    columnas = 9

    def __init__(self, valor, fila, col, ancho, alto):
        self.valor = valor
        self.temp = 0
        self.fila = fila
        self.col = col
        self.ancho = ancho
        self.alto = alto
        self.seleccionado = False

    def dibujar(self, ventana):
        pass

    def establecer(self, val):
        pass

    def establecer_temp(self, val):
        pass
```

Imagen 2: Creación de clases, métodos y atributos de la clase Cubo

Como se puede observar en la imagen 1, se empiezan a crear los métodos y atributos de la clase Cubo. A continuación, veremos una pequeña leyenda de lo que se va a encargar de realizar esta clase en general:

Leyenda:

- Método `__init__`: Este método inicializa los atributos del cubo y se utiliza para establecer su estado inicial.

- Método dibujar: Este método dibuja el cubo en la ventana usando la biblioteca Pygame.
- Método establecer: Este método asigna un valor definitivo al cubo. Modifica el atributo valor del cubo con el valor que se pase como argumento.
- Método establecer_temp: Este método asigna un valor temporal al cubo.

```
class Cuadro:
    def __init__(self, filas, columnas, ancho, alto, ventana):
        self.filas = filas
        self.columnas = columnas
        self.cubos = [[Cubo(0, i, j, ancho, alto) for j in range(columnas)] for i in range(filas)]
        self.ancho = ancho
        self.alto = alto
        self.modelo = None
        self.generar_tablero()
        self.seleccionado = None
        self.ventana = ventana

    def llenar_tablero(
        self: Self@Cuadro,
        tablero: Any
    ) -> None

    def llenar_tablero(self, tablero):
        pass

    def eliminar_numeros(self, tablero):
        pass

    def actualizar_cubos(self):
        pass

    def actualizar_modelo(self):
        pass

    def colocar(self, val):
        pass

    def bocetar(self, val):
        pass

    def dibujar(self):
        pass
```

Imagen 3: Creación de clases, métodos y atributos de la clase Cuadro

```

def seleccionar(self, fila, col):
    pass

def limpiar(self):
    pass

def clic(self, pos):
    pass

def esta_terminado(self):
    pass

def resolver(self):
    pass

def resolver_gui(self):
    pass

```

Imagen 4: Creación de clases, métodos de la clase Cuadro

Como se puede observar en la imagen 2 y 3, se empiezan a crear los métodos y atributos de la clase Cuadro. A continuación, veremos una pequeña leyenda de lo que se va a encargar de realizar esta clase en general:

Leyenda:

- Método `__init__`: Este método inicializa los atributos del cubo y se utiliza para establecer su estado inicial.
- Método `generar_tablero`: Genera el tablero de Sudoku.
- Método `llenar_tablero`: Este método usa un algoritmo de backtracking (búsqueda recursiva) para llenar el tablero.
- Método `eliminar_numeros`: Elimina entre 40 a 50 números aleatorios del tablero para crear un desafío jugable.
- Método `actualizar_cubos`: Sincroniza los valores de los cubos con los valores del tablero.
- Método `actualizar_modelo`: Actualiza el modelo del tablero, que es la representación interna de los números del Sudoku.
- Método `colocar`: Intenta colocar un valor val en la celda seleccionada. Verifica si la colocación es válida.
- Método `bocetar`: Dibuja un valor provisional en la celda seleccionada (el valor temporal) sin modificar permanentemente el tablero.
- Método `dibujar`: Dibuja las líneas del tablero (para crear una cuadrícula) y luego dibuja todos los cubos en la ventana de Pygame.
- Método `seleccionar`: Marca una celda como seleccionada. Esto permite al jugador ver qué celda está activa y puede ser editada.

- Método limpiar: Limpia el valor temporal de la celda seleccionada.
- Método clic: Determina en qué celda del tablero se ha hecho clic.
- Método esta_terminado: Verifica si el tablero está completo, es decir, si todas las celdas tienen un valor distinto de 0.
- Método resolver: Resuelve el tablero utilizando un algoritmo de backtracking. Llama a encontrar_vacio() y luego intenta llenar las celdas vacías recursivamente.
- Método resolver_gui: Similar a resolver(), pero se ejecuta en una interfaz gráfica.

```
class JuegoSudoku:
    def __init__(self):
        pass

    def redibujar_ventana(self, tiempo):
        pass

    def formatear_tiempo(self, segs):
        pass

    def ejecutar(self):
        pass
```

Imagen 5: Creación de clases, métodos de la clase JuegoSudoku

Como se puede observar en la imagen 4, se empiezan a crear los métodos de la clase JuegoSudoku. A continuación, veremos una pequeña leyenda de lo que se va a encargar de realizar esta clase en general:

Leyenda:

- Método __init__: El constructor inicializa la ventana de Pygame, configura el tablero de Sudoku y otros atributos.
- Método redibujar_ventana: Redibuja la ventana de Pygame con el tablero de Sudoku, el tiempo de juego transcurrido y el número de errores cometidos.
- Método formatear_tiempo: Convierte el tiempo transcurrido en segundos en un formato de "minutos:segundos".
- Método ejecutar: El bucle principal del juego. Escucha los eventos de Pygame (clics del ratón, presiones de teclas) y actualiza la ventana en consecuencia.

Paso 2: Rellenar las clases

Los primeros intentos para crear el código de proyecto empezaron por un objetivo más sencillo, la de mostrar el tablero sudoku en pantalla. Este es un buen ejercicio para familiarizarse con las herramientas que nos brinda librerías como pygame.

```
import pygame
import sys

pygame.init()

BLANCO = (255, 255, 255)
NEGRO = (0, 0, 0)

ANCHO_PANTALLA = 600
ALTO_PANTALLA = 600
TAM_CUADRICULA = 450 # Tamaño de la cuadrícula de Sudoku
TAM_CELDA = TAM_CUADRICULA // 9 # Cada celda será un cuadrado de 50x50

# Configurar pantalla
pantalla = pygame.display.set_mode((ANCHO_PANTALLA, ALTO_PANTALLA))
pygame.display.set_caption("Menú del Juego")

fuente = pygame.font.Font(None, 40)

opciones = ["Jugar", "Ajustes", "Salir"]
indice_seleccionado = 0
```

Las primeras líneas del código contenían información sobre las características generales que aparecerían en la pantalla del juego. Además de mostrar las opciones que se mostrarían en el menú.

Imagen 6: Primer código para gestionar la ventana del juego

Método Celda.dibujar se encarga de justamente de dibujar el tablero de un sudoku. Se utiliza pygame.draw.line para dibujar las líneas negras del tablero.

```
class Celda:
    def dibujar(self, ventana: pygame.Surface):
        """
        Dibuja una cuadrícula vacía de Sudoku en la ventana proporcionada.
        """
        ventana.fill(BLANCO) # Fondo blanco para la cuadrícula

        # Dibujar líneas horizontales y verticales
        for fila in range(10): # 9 líneas + 1 adicional para el borde
            # Líneas gruesas cada 3 filas o columnas
            grosor = 4 if fila % 3 == 0 else 1
            # Dibujar líneas horizontales
            pygame.draw.line(ventana, NEGRO,
                             (self.pos_inicio[0], self.pos_inicio[1] + fila * self.tam_celda),
                             (self.pos_inicio[0] + self.tam_cuadrícula, self.pos_inicio[1] + fila * self.tam_celda),
                             grosor)
            # Dibujar líneas verticales
            pygame.draw.line(ventana, NEGRO,
                             (self.pos_inicio[0] + fila * self.tam_celda, self.pos_inicio[1]),
                             (self.pos_inicio[0] + fila * self.tam_celda, self.pos_inicio[1] + self.tam_cuadrícula),
                             grosor)

        pygame.display.flip() # Actualizar la pantalla con la cuadrícula dibujada
```

Imagen 7: Método Celda.dibujar. Dibuja el tablero.

Otras líneas de código provisionales fueron aquellas definían las funciones `dibujar_menu` y `ejecutar_opcion`. Estas funciones se encargaban de animar el menú y de gestionar las opciones que se elegirían desde el menú. Sin embargo, estas funciones fueron dejadas de lado para ayudar a centrar el proyecto en la creación del juego.

```
# Función para dibujar el menú en la pantalla
def dibujar_menu():
    pantalla.fill(NEGRO)
    for i, opcion in enumerate(opciones):
        if i == indice_seleccionado:
            texto = fuente.render(opcion, True, (0, 255, 0)) # Opción seleccionada en verde
        else:
            texto = fuente.render(opcion, True, BLANCO) # Otras opciones en blanco
        pantalla.blit(texto, (ANCHO_PANTALLA // 2 - texto.get_width() // 2, 150 + i * 50))

# Función para manejar la selección de opciones
def ejecutar_opcion(opcion):
    if opcion == "Jugar":
        print("Mostrando cuadrícula vacía de Sudoku...")
        tabla_sudoku = Celda(TAM_CUADRICULA, TAM_CELDA) # Crear una instancia de la clase Tabla
        jugando = True
        while jugando:
            tabla_sudoku.dibujar(pantalla) # Llamar al método dibujar pasándole la ventana (pantalla)
            for evento in pygame.event.get():
                if evento.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()
                if evento.type == pygame.KEYDOWN:
                    if evento.key == pygame.K_ESCAPE: # Salir de la cuadrícula al presionar "Esc"
                        jugando = False
                        pantalla.fill(NEGRO) # Volver al menú
                        break
            elif opcion == "Ajustes":
                print("Abrir menú de ajustes...")
                # Aquí podrías abrir la pantalla de ajustes
            elif opcion == "Salir":
                print("Salir del juego.")
                pygame.quit()
                sys.exit()
```

Imagen 8: Antiguas funciones `dibujar_menu` y `ejecutar_opcion`

```

# Bucle principal
corriendo = True
while corriendo:
    # Dibujar el menú en pantalla
    dibujar_menu()

    # Actualizar la pantalla
    pygame.display.flip()

    # Manejar eventos
    for evento in pygame.event.get():
        if evento.type == pygame.QUIT:
            corriendo = False
            pygame.quit()
            sys.exit()

        # Navegar por las opciones del menú
        if evento.type == pygame.KEYDOWN:
            if evento.key == pygame.K_DOWN:
                indice_seleccionado = (indice_seleccionado + 1) % len(opciones) # Mover hacia abajo
            elif evento.key == pygame.K_UP:
                indice_seleccionado = (indice_seleccionado - 1) % len(opciones) # Mover hacia arriba
            elif evento.key == pygame.K_RETURN: # Seleccionar opción con "Enter"
                ejecutar_opcion(opciones[indice_seleccionado])

pygame.quit()

```

Para ejecutar todos los métodos y funciones definidos, se dispuso de este bucle while. Nótese que en todas estas líneas de código mostradas, todavía no se incorporaba una funcionalidad para poder jugar con el mouse.

Imagen 9: Bucle principal, cuando se salga del bucle el juego termina



Imagen 10: Antiguo Menú

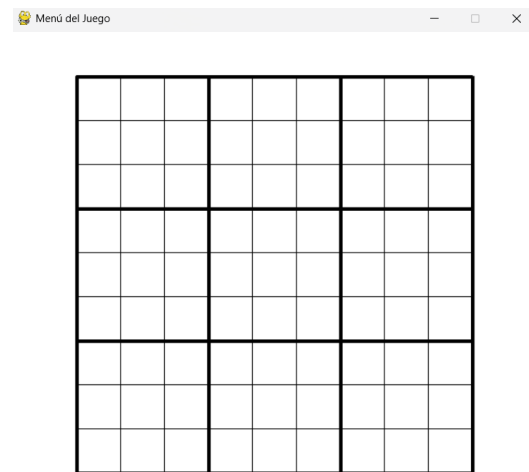


Imagen 11: Primer Tablero

Paso 3: Código listo para jugar Sudoku

Una vez ya realizado correctamente la creación de las clases y métodos; de acuerdo al diagrama UML, con sus respectivos algoritmos, el programa final es el siguiente:

```
import pygame #Creacion ventana de juegos
import random #Creacion de numeros aleatorios
import time #Manejo del tiempo
pygame.font.init()

class Cubo:
    filas = 9
    columnas = 9

    def __init__(self, valor, fila, col, ancho, alto):
        self.valor = valor
        self.temp = 0
        self.fila = fila
        self.col = col
        self.ancho = ancho
        self.alto = alto
        self.seleccionado = False

    def dibujar(self, ventana):#dibujar el cubo en la ventana
        fuente = pygame.font.SysFont("comicsans", 40) #Fuente del texto
        espacio = self.ancho / 9
        x = self.col * espacio
        y = self.fila * espacio #calcula el espacio ocupado por cada cubo y su posición (x, y) en la ventana.

        if self.temp != 0 and self.valor == 0: #Dibuja el temp si no es cero y el valor es cero, mostrando el número en gris.
            texto = fuente.render(str(self.temp), 1, (128, 128, 128))#Color gris, cadena renderizar
            ventana.blit(texto, (x + 5, y + 5))
        elif self.valor != 0:
            texto = fuente.render(str(self.valor), 1, (0, 0, 0))
            ventana.blit(texto, (x + (espacio / 2 - texto.get_width() / 2), y + (espacio / 2 - texto.get_height() / 2)))

        if self.seleccionado:#Si el cubo está seleccionado, se dibuja un borde rojo alrededor del cubo.
            pygame.draw.rect(ventana, (255, 0, 0), (x, y, espacio, espacio), 3)

    def establecer(self, val):#Asigna un valor al cubo
        self.valor = val

    def establecer_temp(self, val): #Asigna un valor temporal al cubo
        self.temp = val
```

Imagen 12: Importación de bibliotecas y código de la clase Cubo

```

class Tablero:#Tablero del Sudoku
    def __init__(self, filas, columnas, ancho, alto, ventana):
        self.filas = filas
        self.columnas = columnas
        self.cubos = [[Cubo(0, i, j, ancho, alto) for j in range(columnas)] for i in range(filas)]
        self.ancho = ancho
        self.alto = alto
        self.modelo = None
        self.generar_tablero()
        self.seleccionado = None
        self.ventana = ventana

    def generar_tablero(self):
        tablero = [[0 for _ in range(9)] for _ in range(9)]
        self.llenar_tablero(tablero)
        self.eliminar_numeros(tablero)
        self.tablero = tablero
        self.actualizar_cubos()

    def llenar_tablero(self, tablero):#Llena el tablero utilizando un algoritmo de backtracking (todas las combinaciones de una solución).
        def es_valido(num, fila, col):
            for i in range(9):
                if tablero[fila][i] == num or tablero[i][col] == num:
                    return False

            fila_caja = fila // 3 * 3#Division exacta
            col_caja = col // 3 * 3
            for i in range(3):
                for j in range(3):
                    if tablero[fila_caja + i][col_caja + j] == num:
                        return False
            return True #Verifica si el número también es válido en su caja 3x3.

        def resolver(): #Comprobación del backtracking- recursividad
            for i in range(9):
                for j in range(9):
                    if tablero[i][j] == 0:
                        numeros_aleatorios = list(range(1, 10))
                        random.shuffle(numeros_aleatorios) #Mezcla elementos de una lista
                        for num in numeros_aleatorios:
                            if es_valido(num, i, j):
                                tablero[i][j] = num
                                if resolver():
                                    return True
                                tablero[i][j] = 0
                        return False
            return True

        resolver()

```

Imagen 13: Código de la clase Tablero


```

def eliminar_numeros(self, tablero):
    intentos = random.randint(40, 50) #Elimina entre 40 a 50 numeros
    while intentos > 0:
        i = random.randint(0, 8)
        j = random.randint(0, 8)
        if tablero[i][j] != 0:
            tablero[i][j] = 0
            intentos -= 1

def actualizar_cubos(self): #Sincroniza los valores de los cubos con los del tablero.
    for i in range(self.filas):
        for j in range(self.columnas):
            self.cubos[i][j].establecer(self.tablero[i][j])

def actualizar_modelo(self):
    self.modelo = [[self.cubos[i][j].valor for j in range(self.columnas)] for i in range(self.filas)]

def colocar(self, val): #Coloca el numero y verifica si va o no.
    fila, col = self.seleccionado
    if self.cubos[fila][col].valor == 0:
        self.cubos[fila][col].establecer(val)
        self.actualizar_modelo()

        if valido(self.modelo, val, (fila, col)) and self.resolver():
            return True
        else:
            self.cubos[fila][col].establecer(0) #Vuelvo a poner 0= blanco
            self.cubos[fila][col].establecer_temp(0)
            self.actualizar_modelo()
            return False

def bocetar(self, val):
    fila, col = self.seleccionado
    self.cubos[fila][col].establecer_temp(val)

def dibujar(self): #Dibuja las lineas de separacion y numeros en la ventana pygame
    espacio = self.ancho / 9
    for i in range(self.filas + 1):
        grueso = 4 if i % 3 == 0 and i != 0 else 1
        pygame.draw.line(self.ventana, (0, 0, 0), (0, i * espacio), (self.ancho, i * espacio), grueso)
        pygame.draw.line(self.ventana, (0, 0, 0), (i * espacio, 0), (i * espacio, self.alto), grueso)

    for i in range(self.filas):
        for j in range(self.columnas):
            self.cubos[i][j].dibujar(self.ventana)

```

Imagen 14: Código de los métodos de la clase Tablero

```

def seleccionar(self, fila, col): #Seleccionar el cuadro
    for i in range(self.filas):
        for j in range(self.columnas):
            self.cubos[i][j].seleccionado = False

    self.cubos[fila][col].seleccionado = True
    self.seleccionado = (fila, col)

def limpiar(self):#Limpia el cuadro seleccionado
    fila, col = self.seleccionado
    if self.cubos[fila][col].valor == 0:
        self.cubos[fila][col].establecer_temp(0)

def clic(self, pos): #Donde se hizo el click
    if pos[0] < self.ancho and pos[1] < self.alto:
        espacio = self.ancho / 9
        x = pos[0] // espacio
        y = pos[1] // espacio
        return (int(y), int(x))
    else:
        return None

def esta_terminado(self):
    for i in range(self.filas):
        for j in range(self.columnas):
            if self.cubos[i][j].valor == 0:
                return False
    return True

def resolver(self): #Verificar el algoritmo backtracking
    li, pos, flag = encontrar_vacio(self.modelo)

    if flag == 0:
        return True

    if not pos:
        return False

    fila, col = pos

    for i in li:
        if valido(self.modelo, i, (fila, col)):
            self.modelo[fila][col] = i

            if self.resolver():
                return True

            self.modelo[fila][col] = 0

    return False

```

Imagen 15: Código de los métodos de la clase Tablero

```

def resolver_gui(self):
    li, pos, flag = encontrar_vacio(self.modelo)

    if flag == 0:
        return True

    if not pos:
        return False

    fila, col = pos

    for i in li:
        if valido(self.modelo, i, (fila, col)):
            self.modelo[fila][col] = i
            self.cubos[fila][col].establecer(i)
            self.cubos[fila][col].dibujar(self.ventana)
            self.actualizar_modelo()
            pygame.display.update()
            pygame.time.delay(100)

            if self.resolver_gui():
                return True

            self.modelo[fila][col] = 0
            self.cubos[fila][col].establecer(0)
            self.actualizar_modelo()
            self.cubos[fila][col].dibujar(self.ventana)
            pygame.display.update()
            pygame.time.delay(100)

    return False

```

Imagen 15: Código de los métodos de la clase Tablero

```

class JuegoSudoku:
    def __init__(self): #Se inicializa la ventana de Pygame (instanciando)
        self.ventana = pygame.display.set_mode((650, 650))
        pygame.display.set_caption("Sudoku")
        self.tablero = Tablero(9, 9, 540, 540, self.ventana)
        self.tecla = None
        self.correr = True
        self.inicio = time.time()
        self.errores = 0

    def redibujar_ventana(self, tiempo): #Actualiza la interfaz grafica
        self.ventana.fill((255, 255, 255))
        fuente = pygame.font.SysFont("comicsans", 40)
        texto = fuente.render("Tiempo: " + self.formatear_tiempo(tiempo), 1, (0, 0, 0))
        self.ventana.blit(texto, (540 - 160, 560))
        texto = fuente.render("X " * self.errores, 1, (255, 0, 0))
        self.ventana.blit(texto, (20, 560))
        self.tablero.dibujar()

    def formatear_tiempo(self, segs): #Añade el tiempo
        seg = segs % 60
        minuto = segs // 60
        hora = minuto // 60
        return f" {minuto}:{seg}"

```

Imagen 16: Código de la clase JuegoSudoku y sus métodos

```

def ejecutar(self):
    while self.correr:
        tiempo_juego = round(time.time() - self.inicio)

        for evento in pygame.event.get():
            if evento.type == pygame.QUIT: #Salir de la interfaz
                self.correr = False
            if evento.type == pygame.KEYDOWN: #Si se presiona tecla
                if evento.key in (pygame.K_1, pygame.K_2, pygame.K_3, pygame.K_4, pygame.K_5, pygame.K_6, pygame.K_7, pygame.K_8, pygame.K_9):
                    self.tecla = evento.key - pygame.K_0
                if evento.key == pygame.K_DELETE:
                    self.tablero.limpiar()
                    self.tecla = None

                if evento.key == pygame.K_RETURN: #Se obtiene el numero preseleccionado
                    i, j = self.tablero.seleccionado
                    if self.tablero.cubos[i][j].temp != 0:
                        if self.tablero.colocar(self.tablero.cubos[i][j].temp):
                            print("Éxito")
                        else:
                            print("Incorrecto")
                            self.errores += 1
                    self.tecla = None

                    if self.tablero.esta_terminado():
                        print("Juego terminado")

            if evento.type == pygame.MOUSEBUTTONDOWN: #clic del mouse(posicion)
                pos = pygame.mouse.get_pos()
                clicado = self.tablero.clic(pos)
                if clicado:
                    self.tablero.seleccionar(clicado[0], clicado[1])
                    self.tecla = None

        if self.tablero.seleccionado and self.tecla is not None:
            self.tablero.bocetar(self.tecla)

        self.redibujar_ventana(tiempo_juego) #Actualizar la ventana

        pygame.display.update()

    pygame.quit()

```

Imagen 17: Código de los métodos de la clase JuegoSudoku

```

def completo(tablero, i, j):
    excluidos = {0}
    for fila in range(len(tablero)):
        excluidos.add(tablero[fila][j])

    for col in range(len(tablero)):
        excluidos.add(tablero[i][col])

    i -= i % 3
    j -= j % 3

    for fila in range(int(len(tablero) / 3)):
        for col in range(int(len(tablero) / 3)):
            excluidos.add(tablero[i + fila][j + col])

    restantes = set(range(1, 10)).difference(excluidos)
    return restantes

def encontrar_vacio(tablero):
    minv = 10
    minn = set()
    pos = ()
    flag = 0
    for i in range(len(tablero)):
        for j in range(len(tablero[0])):
            if tablero[i][j] == 0:
                flag = 1
                numeros = completo(tablero, i, j)
                if (minv > len(numeros) and len(numeros) > 0):
                    minv = len(numeros)
                    minn = numeros
                    pos = (i, j)

    if (minv == 10):
        return (None, None, flag)

    return (minn, pos, flag)

```

Imagen :Código de los métodos de la clase JuegoSudoku

```

def valido(bo, num, pos): #Analiza si el numero esta bien en la casilla
    for i in range(len(bo[0])):
        if bo[pos[0]][i] == num and pos[1] != i:
            return False

    for i in range(len(bo)):
        if bo[i][pos[1]] == num and pos[0] != i:
            return False

    caja_x = pos[1] // 3
    caja_y = pos[0] // 3

    for i in range(caja_y * 3, caja_y * 3 + 3):
        for j in range(caja_x * 3, caja_x * 3 + 3):
            if bo[i][j] == num and (i, j) != pos:
                return False

    return True

juego = JuegoSudoku()
juego.ejecutar()

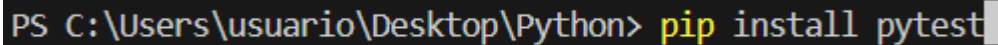
```

Imagen 18: Código de los métodos de la clase JuegoSudoku

PRUEBAS AUTOMATIZADAS

Las pruebas automatizadas en Python para un juego de Sudoku utilizando “pytest” permiten verificar el funcionamiento correcto del juego de manera eficiente y sistemática.

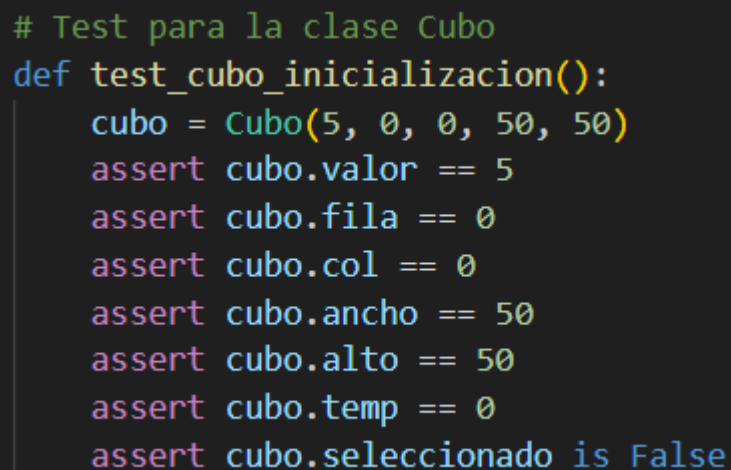
Para comenzar, se debe instalar “pytest”, un framework de pruebas que facilita la creación, ejecución y reporte de pruebas automatizadas en Python. Esto se realiza con el siguiente comando:



```
PS C:\Users\usuario\Desktop\Python> pip install pytest
```

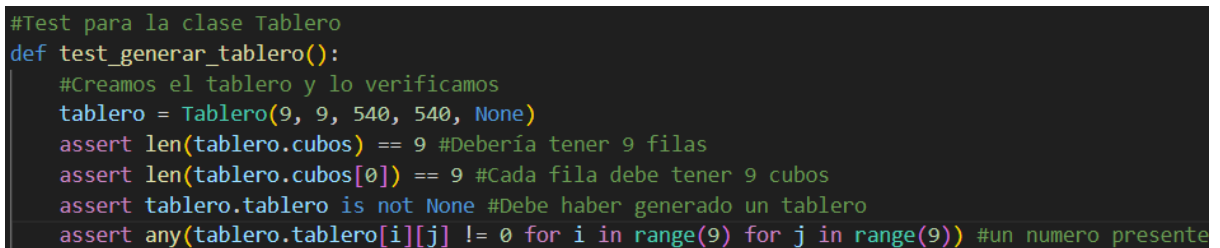
Imagen 19: Instalación de la librería pytest

Las pruebas automatizadas para el juego Sudoku se centran en verificar diferentes aspectos de la lógica del juego, compartiendo los métodos más importantes en el juego como presentaremos en las siguientes imágenes:



```
# Test para la clase Cubo
def test_cubo_inicializacion():
    cubo = Cubo(5, 0, 0, 50, 50)
    assert cubo.valor == 5
    assert cubo.fila == 0
    assert cubo.col == 0
    assert cubo.ancho == 50
    assert cubo.alto == 50
    assert cubo.temp == 0
    assert cubo.seleccionado is False
```

Imagen 20: Testeo de método y la clase correspondiente a cubo



```
#Test para la clase Tablero
def test_generar_tablero():
    #Creamos el tablero y lo verificamos
    tablero = Tablero(9, 9, 540, 540, None)
    assert len(tablero.cubos) == 9 #Debería tener 9 filas
    assert len(tablero.cubos[0]) == 9 #Cada fila debe tener 9 cubos
    assert tablero.tablero is not None #Debe haber generado un tablero
    assert any(tablero.tablero[i][j] != 0 for i in range(9) for j in range(9)) #un numero presente
```

Imagen 21: Testeo de método generar tablero

```
#Test para la función "valido"
def test_valido():
    # Tablero de prueba donde la casilla (0,0) contiene un valor 5
    tablero = [
        [5, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]
    ]
    assert valido(tablero, 5, (0, 1)) is False #No se puede colocar un 5 en la posición (0,1)
    assert valido(tablero, 3, (0, 1)) is True #Sí se puede colocar un 3 en la posición (0,1)
```

Imagen 22: Testeo de método valido

```
#Test para la función "completo"
def test_completo():
    tablero = [
        [5, 3, 0, 0, 7, 0, 0, 0, 0],
        [6, 0, 0, 1, 9, 5, 0, 0, 0],
        [0, 9, 8, 0, 0, 0, 0, 6, 0],
        [8, 0, 0, 0, 6, 0, 0, 0, 3],
        [4, 0, 0, 8, 0, 3, 0, 0, 1],
        [7, 0, 0, 0, 2, 0, 0, 0, 6],
        [0, 6, 0, 0, 0, 0, 2, 8, 0],
        [0, 0, 0, 4, 1, 9, 0, 0, 5],
        [0, 0, 0, 0, 8, 0, 0, 7, 9]
    ]
    posibles = completo(tablero, 0, 2) #Devuelve los numeros que se pueden colocar en la posición (0, 2)
    assert 1 in posibles #Debería permitir el 1
    assert 2 in posibles #Debería permitir el 2
    assert 4 in posibles #Debería permitir el 4
```

Imagen 23: Testeo de método completo

Para poder analizar y ver si los testeos estén funcionando y estén devolviendo los valores que se requieren, necesitamos abrir el terminal y poner lo siguiente:

```
PS C:\Users\usuario\Desktop\Python> pytest testeo2.py
```

Imagen 24: Comando para poder revisar las pruebas

Entonces se nos imprimirá el siguiente mensaje:

```
===== test session starts =====
platform win32 -- Python 3.11.2, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\usuario\Desktop\Python
collected 4 items

testeo2.py .... [100%]

===== 4 passed in 3.47s =====
```

Imagen 25: Resultados de las pruebas automatizadas

En la anterior imagen, podemos observar el inicio del test, donde pudo encontrar 4 testeos, que recordemos son los métodos mas importantes en la lógica del juego, y nos da como resultado un 100% de confianza y que si se devuelve los valores esperados por cada método llamado.

CONCLUSIONES

- El uso de bibliotecas como pygame, pytest o time facilita mucho la creación del código y la realización de pruebas. Sin embargo, pueden llegar a darse complicaciones a la hora de importar funciones de otros scripts. Esto sucedió cuando se intentó implementar modelos de Agentes que resolverían el sudoku con RL, pues la complejidad de relación entre las funciones y clases pueden generar conflictos cuando se llaman desde otros códigos.
- El desarrollo del proyecto de un juego de Sudoku en Python orientado a objetos permitió alcanzar múltiples objetivos tanto técnicos como personales. Desde una perspectiva de impacto personal, el juego logró estimular la agilidad mental y lógica, fomentando la paciencia y la perseverancia en la búsqueda de soluciones, lo cual también contribuyó a la reducción del estrés y la mejora de la salud cerebral mediante el uso de habilidades cognitivas como la planificación y toma de decisiones.
- Desde el punto de vista técnico, el proyecto facilitó el desarrollo de algoritmos eficientes, promoviendo el aprendizaje y la optimización de soluciones computacionales. Además, el enfoque en la programación orientada a objetos permitió diseñar un código modular y estructurado, facilitando la reutilización y el mantenimiento del programa. El uso de pruebas automatizadas, que abarcó desde pruebas unitarias hasta pruebas de rendimiento, garantizó la calidad y robustez del código, consolidando buenas prácticas en el desarrollo de software.
- En conjunto, este proyecto no solo cumplió con los objetivos planteados, sino que también brindó una experiencia enriquecedora para fomentar habilidades personales y técnicas, integrando aprendizaje, creatividad y crecimiento en el ámbito de la programación.

REFERENCIAS BIBLIOGRAFICAS

Arias, E. (2019). El juego sudoku y el desarrollo del pensamiento lógico matemático en la institución educativa integrada Pedro Sánchez Gavidia – Huánuco – 2017. Tesis para optar el grado académico de maestro en ciencias de la educación. Escuela de posgrado de la Universidad de Huánuco. Perú. p. 17.

Bermón Angarita, L., Prieto Taborda, A., Escobar Márquez, J. D., & Vergara Díaz, J. D. (2021). Videojuego para el aprendizaje de lógica de programación. *Revista Educación en Ingeniería*, 16(31), 46-56. <https://doi.org/10.26507/rei.v16n31.1141>

Pérez, L. (2008). El juego didáctico sudoku y su influencia en el desarrollo del razonamiento lógico – matemático en niños del sexto y séptimos año Educación Básica de la Escuela Fiscal Mixta Amemos al niño de la parroquia Eloy Alfaro de la Ciudad de Manta, en el periodo lectivo 2007 – 2008.