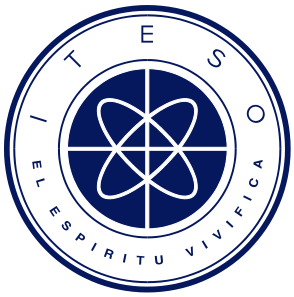
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

ALGORITMOS Y PROGRAMACIÓN



LOS CHESS E-NANOS

Presentan

Diego Yael Islas Santoyo 742477

Natalia Jocelyn Peña Diaz 734896

Aline Quetzalli Rockenzahn Gallegos 745986

Profesor: Jesús Alejandro Rizo Domínguez

Fecha :10/05 /2024

Contenido

Nombre del proyecto

Los Chess e-nanos

Introducción

Jugar ajedrez nunca había sido tan sencillo y emocionante. Gracias a la tecnología, ahora puedes disfrutar de este juego clásico sin necesidad de cargar con un tablero físico. Simplemente abres el programa y estás listo para jugar.

Los juegos de mesa tradicionales están enfrentando desafíos debido al auge de la tecnología digital. Por eso, es fundamental que evolucionen para mantenerse relevantes y atractivos. Una de las mejores maneras de adaptarse es llevar el ajedrez al entorno digital. Con una versión virtual, ya no tendrás que preocuparte por perder piezas o dañar el tablero. Todo se muestra en la pantalla y está siempre disponible para jugar.

El ajedrez virtual que hemos creado en Python permite que dos personas jueguen en el mismo monitor, facilitando la experiencia para ambos jugadores. El programa tiene características útiles, como destacar la pieza que seleccionas y mostrar las piezas que han sido capturadas a lo largo del juego, tanto para las piezas negras como para las blancas. Esta funcionalidad no solo facilita el juego, sino que también lo hace más accesible y entretenido para todos, desde principiantes hasta expertos.

Alcance del Proyecto

Funcionalidades Clave

- **Juego de Ajedrez para Dos Jugadores**:
    - El juego permitirá a dos personas jugar ajedrez en el mismo monitor, turnándose para hacer movimientos.
- **Interfaz Gráfica Intuitiva**:
    - El juego contará con un tablero gráfico y piezas representadas visualmente, proporcionando una experiencia intuitiva y atractiva.
- **Validación de Movimientos**:
    - El juego validará los movimientos para asegurarse de que sean conformes a las reglas del ajedrez, mostrando movimientos permitidos y manejando piezas capturadas.
- **Gestión de Turnos**:
    - El juego controlará el cambio de turno entre jugadores y mostrará visualmente a quién le toca jugar.
- **Detección de Fin del Juego**:
    - El juego será capaz de detectar situaciones de jaque mate y otras condiciones que indiquen el fin del juego.

Entorno y Requisitos Técnicos

- **Plataforma de Pygame**:
    - El juego será desarrollado usando Pygame, por lo que requerirá Python y la biblioteca de Pygame para ejecutarse.
- **Compatibilidad con Sistemas Operativos**:
    - El alcance incluye la capacidad de ejecución en diferentes sistemas operativos, como Windows, macOS, y Linux, siempre que tengan Python y Pygame instalados.

Limitaciones y Exclusiones

- **Modo Multijugador Remoto**:
    - El alcance no incluye el desarrollo de funcionalidades para jugar a través de internet o en red. El juego está diseñado para dos jugadores en el mismo monitor.
- **Funciones de Inteligencia Artificial**:
    - El juego no incluirá jugadores controlados por IA ni mecanismos avanzados para asistencia en el juego.
- **Características de Personalización**:
    - El alcance no incluye opciones para personalizar el aspecto de las piezas, tableros u otros elementos visuales.

Entregables del Proyecto

- **Código Fuente del Juego**:
    - El proyecto entregará el código fuente completo para ejecutar el juego de ajedrez.
- **Documentación del Usuario**:
    - El alcance incluye la creación de documentación para usuarios, con instrucciones para instalar y jugar el juego.
- **Documentación Técnica**:
    - El alcance también considera documentación para desarrolladores, detallando la estructura del código y las funciones principales.

Requerimientos

<Especificaciones del sistema>

El sistema deberá tener instalada la aplicación VisualStudioCode. Se deberá instalar en esta aplicación el lenguaje de Python más reciente que se tenga a disposición para su computadora. El sistema deberá importar al Visual la librería Pygame versión 2.5.2. Finalmente, en el sistema deberá estar instalada la carpeta Assets dentro de los archivos que pueda leer el código dentro del visual. Esta carpeta contiene los diseños de las piezas del juego.

Se puede acceder a la carpeta en cuestión con el link:

https://iteso01-my.sharepoint.com/:f:/g/personal/diego_islas_iteso_mx/EuowX1KQbZZLsFNO-RRRTaYB-gLvqMhKmUlIMtzoQ_hURg?e=49FJ4H

Si el sistema no tiene instalados los requerimientos anteriores, no será posible que ejecute el código.

Diseño de la solución

<Incluye el diagrama de descomposición modular de la solución>

Dibujar el tablero de ajedrez y algunos elementos adicionales en la pantalla:

La función draw_board utiliza pygame.draw.rect() para crear el tablero a cuadros de ajedrez, alternando colores entre gris claro y oscuro según la fila, además de dibujar la base y el borde dorado del tablero. Se emplea big_font.render() y screen.blit() para mostrar el estado del turno en la parte inferior izquierda de la pantalla. Finalmente, pygame.draw.line() se usa para dibujar las líneas de la cuadrícula del tablero, completando así la representación visual del juego de ajedrez.

Esta función no genera un nuevo tipo de dato en sí mismo, sino que se centra en la manipulación de tipos de datos existentes para crear la representación visual del tablero.

Dibujar piezas en el tablero:

La funcion def_pieces permite dibujar las piezas de ajedrez blancas y negras en el tablero del juego, utilizando las bibliotecas pygame y numpy para la manipulación de gráficos.

Funciona recorriendo las listas white_pieces y black_pieces para identificar los tipos de piezas. Luego, se obtienen las imágenes de las piezas de las listas piece_list, white_images y black_images, y se calculan sus posiciones en el tablero utilizando las listas white_locations y black_locations. Finalmente, las piezas se dibujan en las posiciones correctas con la función pygame.blit().

La pieza seleccionada para mover se destaca con un rectángulo rojo o azul, dependiendo del turno del jugador.

El código no genera un nuevo tipo de dato, sino que interactúa con los datos existentes para lograr su función principal de dibujar las piezas de ajedrez en la pantalla.

Verificar las opciones validas de las piezas

La funcion check_options crea dos listas vacías: moves_list para almacenar los movimientos posibles de la pieza actual y all_moves_list para los movimientos de todas las piezas del jugador en turno. Mediante un bucle for, se itera sobre cada pieza del jugador, obteniendo su posición y tipo de la lista correspondiente. Dependiendo del tipo de pieza, se llama a una función específica (como check_pawn, check_rook, etc.) que determina y almacena los movimientos posibles en moves_list. Posteriormente, los movimientos de cada pieza se agregan a all_moves_list. Al final, la función check_options devuelve all_moves_list con todos los movimientos válidos disponibles para ese turno.

Comprobar si hay movimientos válidos para la pieza seleccionada:

La función check_valid_moves verifica los movimientos válidos para una pieza según el turno del juego. Utiliza la condición turn_step < 2 para determinar si el turno está en sus primeras fases y asigna las opciones de movimiento adecuadas: white_options para las blancas en los primeros pasos y black_options para las negras en los siguientes. Dependiendo de la pieza seleccionada, identificada por la variable selection, accede a options_list para obtener las casillas válidas específicas para esa pieza. Finalmente, la función retorna valid_options, que son las casillas legales a las que la pieza seleccionada puede moverse en el turno actual.

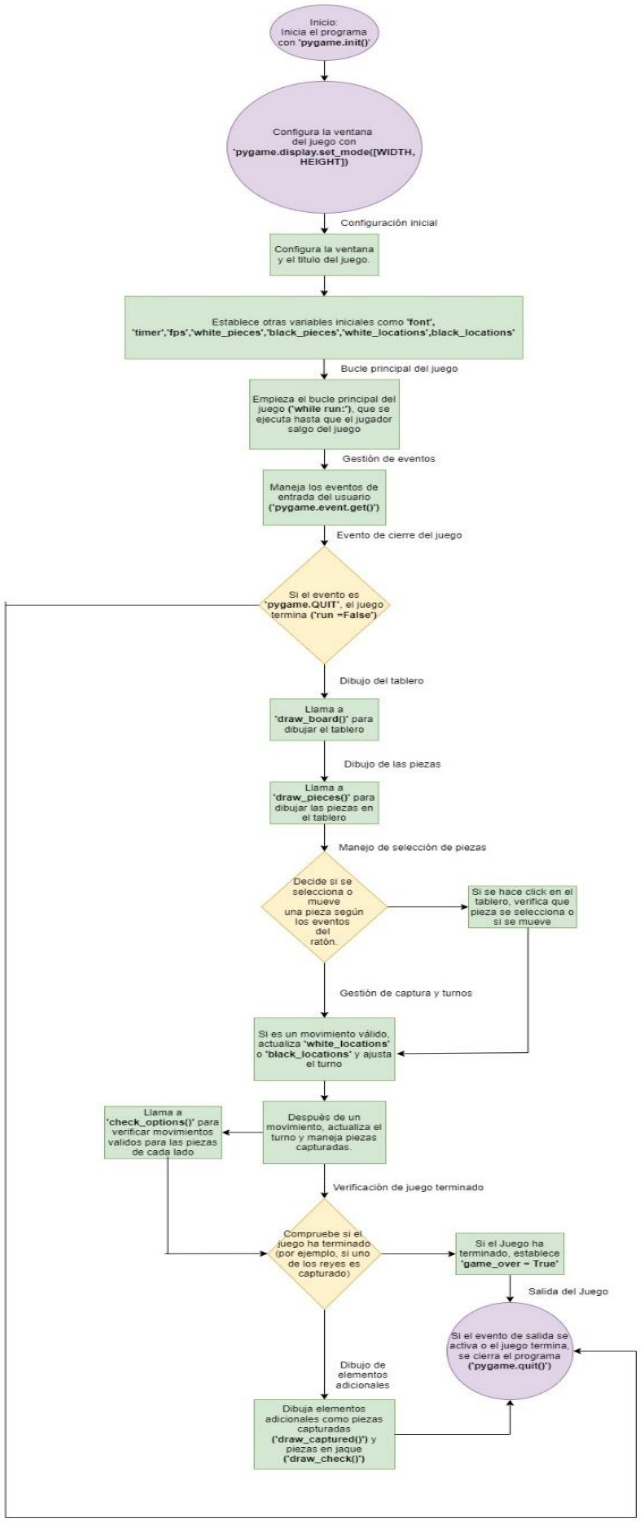
dibujar movimientos válidos en la pantalla:

La función draw_valid muestra los movimientos válidos de una pieza. Primero, determina el color basado en el paso del turno: 'red' si turn_step < 2, lo que indica los primeros dos pasos del turno (probablemente seleccionando una pieza blanca), y 'blue' si turn_step >= 2, indicando pasos posteriores (seleccionando una pieza negra o un destino). Luego, itera a través de la lista de casillas válidas moves, dibujando círculos en cada casilla con pygame.draw.circle(). Cada círculo representa una casilla válida a la que la pieza puede moverse, con su posición calculada a partir de las coordenadas de fila y columna de la casilla.

Fin del juego:

La función draw_game_over en Python se encarga de proporcionar una señal visual de "Juego Terminado" en la pantalla del juego. En primer lugar, crea un rectángulo negro en la pantalla con la función pygame.draw.rect(), definiendo su posición y tamaño. Luego, mediante la función screen.blit(), muestra un mensaje indicando quién ha ganado el juego, interpolando el nombre del ganador. Este mensaje se visualiza en color blanco, con los bordes suavizados. Por último, proporciona instrucciones claras para reiniciar el juego, también en blanco y posicionadas debajo del mensaje anterior. Estas instrucciones indican al jugador que presione la tecla ENTER para reiniciar el juego y comenzar de nuevo.

Diagrama de flujo



Implementación

Código Fuente

```
import pygame

pygame.init()
WIDTH = 1000
HEIGHT = 900
screen = pygame.display.set_mode([WIDTH, HEIGHT])
pygame.display.set_caption('Two-Player Pygame Chess!')
font = pygame.font.Font('freesansbold.ttf', 20)
medium_font = pygame.font.Font('freesansbold.ttf', 40)
big_font = pygame.font.Font('freesansbold.ttf', 50)
timer = pygame.time.Clock()
fps = 60
# Juego de variable e imagenes
white_pieces = ['rook', 'knight', 'bishop', 'king', 'queen', 'bishop',
'knight', 'rook',
                'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn',
'pawn']
white_locations = [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0),
(7, 0),
                   (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1),
(7, 1)]
black_pieces = ['rook', 'knight', 'bishop', 'king', 'queen', 'bishop',
'knight', 'rook',
                'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn',
'pawn']
black_locations = [(0, 7), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7),
(7, 7),
                   (0, 6), (1, 6), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6),
(7, 6)]
captured_pieces_white = []
captured_pieces_black = []
#0 - los blancos giran sin selección: 1 - los blancos giran la pieza
seleccionada: 2 - los negros giran sin selección, 3 - los negros giran la
pieza seleccionada
turn_step = 0
selection = 100
valid_moves = []
# cargar imágenes de piezas del juego (reina, rey, torre, alfil, caballo,
peón) x 2
black_queen = pygame.image.load('assets/images/black queen.png')
black_queen = pygame.transform.scale(black_queen, (80, 80))
black_queen_small = pygame.transform.scale(black_queen, (45, 45))
black_king = pygame.image.load('assets/images/black king.png')
```

```
black_king = pygame.transform.scale(black_king, (80, 80))
black_king_small = pygame.transform.scale(black_king, (45, 45))
black_rook = pygame.image.load('assets/images/black rook.png')
black_rook = pygame.transform.scale(black_rook, (80, 80))
black_rook_small = pygame.transform.scale(black_rook, (45, 45))
black_bishop = pygame.image.load('assets/images/black bishop.png')
black_bishop = pygame.transform.scale(black_bishop, (80, 80))
black_bishop_small = pygame.transform.scale(black_bishop, (45, 45))
black_knight = pygame.image.load('assets/images/black knight.png')
black_knight = pygame.transform.scale(black_knight, (80, 80))
black_knight_small = pygame.transform.scale(black_knight, (45, 45))
black_pawn = pygame.image.load('assets/images/black pawn.png')
black_pawn = pygame.transform.scale(black_pawn, (65, 65))
black_pawn_small = pygame.transform.scale(black_pawn, (45, 45))
white_queen = pygame.image.load('assets/images/white queen.png')
white_queen = pygame.transform.scale(white_queen, (80, 80))
white_queen_small = pygame.transform.scale(white_queen, (45, 45))
white_king = pygame.image.load('assets/images/white king.png')
white_king = pygame.transform.scale(white_king, (80, 80))
white_king_small = pygame.transform.scale(white_king, (45, 45))
white_rook = pygame.image.load('assets/images/white rook.png')
white_rook = pygame.transform.scale(white_rook, (80, 80))
white_rook_small = pygame.transform.scale(white_rook, (45, 45))
white_bishop = pygame.image.load('assets/images/white bishop.png')
white_bishop = pygame.transform.scale(white_bishop, (80, 80))
white_bishop_small = pygame.transform.scale(white_bishop, (45, 45))
white_knight = pygame.image.load('assets/images/white knight.png')
white_knight = pygame.transform.scale(white_knight, (80, 80))
white_knight_small = pygame.transform.scale(white_knight, (45, 45))
white_pawn = pygame.image.load('assets/images/white pawn.png')
white_pawn = pygame.transform.scale(white_pawn, (65, 65))
white_pawn_small = pygame.transform.scale(white_pawn, (45, 45))
white_images = [white_pawn, white_queen, white_king, white_knight,
white_rook, white_bishop]
small_white_images = [white_pawn_small, white_queen_small, white_king_small,
white_knight_small,
                      white_rook_small, white_bishop_small]
black_images = [black_pawn, black_queen, black_king, black_knight,
black_rook, black_bishop]
small_black_images = [black_pawn_small, black_queen_small, black_king_small,
black_knight_small,
                      black_rook_small, black_bishop_small]
piece_list = ['pawn', 'queen', 'king', 'knight', 'rook', 'bishop']
# comprobar variables/contador parpadeante
counter = 0
```

```
winner = ''
game_over = False


# dibujar el tablero de juego principal
def draw_board():
    for i in range(32):
        column = i % 4
        row = i // 4
        if row % 2 == 0:
            pygame.draw.rect(screen, 'light gray', [600 - (column * 200),
row * 100, 100, 100])
        else:
            pygame.draw.rect(screen, 'light gray', [700 - (column * 200),
row * 100, 100, 100])
        pygame.draw.rect(screen, 'gray', [0, 800, WIDTH, 100])
        pygame.draw.rect(screen, 'gold', [0, 800, WIDTH, 100], 5)
        pygame.draw.rect(screen, 'gold', [800, 0, 200, HEIGHT], 5)
        status_text = ['White: Select a Piece to Move!', 'White: Select a
Destination!',
                       'Black: Select a Piece to Move!', 'Black: Select a
Destination!']
        screen.blit(big_font.render(status_text[turn_step], True, 'black'),
(20, 820))
        for i in range(9):
            pygame.draw.line(screen, 'black', (0, 100 * i), (800, 100 * i),
2)
            pygame.draw.line(screen, 'black', (100 * i, 0), (100 * i, 800),
2)
        screen.blit(medium_font.render('FORFEIT', True, 'black'), (810,
830))


# dibujar piezas en el tablero
def draw_pieces():
    for i in range(len(white_pieces)):
        index = piece_list.index(white_pieces[i])
        if white_pieces[i] == 'pawn':
            screen.blit(white_pawn, (white_locations[i][0] * 100 + 22,
white_locations[i][1] * 100 + 30))
        else:
            screen.blit(white_images[index], (white_locations[i][0] * 100 +
10, white_locations[i][1] * 100 + 10))
        if turn_step < 2:
            if selection == i:
```

```
                pygame.draw.rect(screen, 'red', [white_locations[i][0] * 100
+ 1, white_locations[i][1] * 100 + 1,
                                                 100, 100], 2)

    for i in range(len(black_pieces)):
        index = piece_list.index(black_pieces[i])
        if black_pieces[i] == 'pawn':
            screen.blit(black_pawn, (black_locations[i][0] * 100 + 22,
black_locations[i][1] * 100 + 30))
        else:
            screen.blit(black_images[index], (black_locations[i][0] * 100 +
10, black_locations[i][1] * 100 + 10))
        if turn_step >= 2:
            if selection == i:
                pygame.draw.rect(screen, 'blue', [black_locations[i][0] *
100 + 1, black_locations[i][1] * 100 + 1,
                                                  100, 100], 2)


# Función para verificar todas las opciones válidas de piezas a bordo.
def check_options(pieces, locations, turn):
    moves_list = []
    all_moves_list = []
    for i in range((len(pieces))):
        location = locations[i]
        piece = pieces[i]
        if piece == 'pawn':
            moves_list = check_pawn(location, turn)
        elif piece == 'rook':
            moves_list = check_rook(location, turn)
        elif piece == 'knight':
            moves_list = check_knight(location, turn)
        elif piece == 'bishop':
            moves_list = check_bishop(location, turn)
        elif piece == 'queen':
            moves_list = check_queen(location, turn)
        elif piece == 'king':
            moves_list = check_king(location, turn)
        all_moves_list.append(moves_list)
    return all_moves_list


# comprobar movimientos válidos del rey
def check_king(position, color):
    moves_list = []
```

```
    if color == 'white':
        enemies_list = black_locations
        friends_list = white_locations
    else:
        friends_list = black_locations
        enemies_list = white_locations
    # 8 casillas para buscar reyes, pueden recorrer una casilla en cualquier
dirección
    targets = [(1, 0), (1, 1), (1, -1), (-1, 0), (-1, 1), (-1, -1), (0, 1),
(0, -1)]
    for i in range(8):
        target = (position[0] + targets[i][0], position[1] + targets[i][1])
        if target not in friends_list and 0 <= target[0] <= 7 and 0 <=
target[1] <= 7:
            moves_list.append(target)
    return moves_list


# comprobar movimientos válidos de la reina
def check_queen(position, color):
    moves_list = check_bishop(position, color)
    second_list = check_rook(position, color)
    for i in range(len(second_list)):
        moves_list.append(second_list[i])
    return moves_list


# comprobar los movimientos del alfil
def check_bishop(position, color):
    moves_list = []
    if color == 'white':
        enemies_list = black_locations
        friends_list = white_locations
    else:
        friends_list = black_locations
        enemies_list = white_locations
    for i in range(4):  # arriba derecha, arriba izquierda, abajo derecha,
abajo izquierda
        path = True
        chain = 1
        if i == 0:
            x = 1
            y = -1
        elif i == 1:
            x = -1
```

```
            y = -1
        elif i == 2:
            x = 1
            y = 1
        else:
            x = -1
            y = 1
        while path:
            if (position[0] + (chain * x), position[1] + (chain * y)) not in
friends_list and \
                    0 <= position[0] + (chain * x) <= 7 and 0 <= position[1]
+ (chain * y) <= 7:
                moves_list.append((position[0] + (chain * x), position[1] +
(chain * y)))
                if (position[0] + (chain * x), position[1] + (chain * y)) in
enemies_list:
                    path = False
                chain += 1
            else:
                path = False
    return moves_list


# comprobar los movimientos de la torre
def check_rook(position, color):
    moves_list = []
    if color == 'white':
        enemies_list = black_locations
        friends_list = white_locations
    else:
        friends_list = black_locations
        enemies_list = white_locations
    for i in range(4):  # abajo, arriba, derecha, izquierda
        path = True
        chain = 1
        if i == 0:
            x = 0
            y = 1
        elif i == 1:
            x = 0
            y = -1
        elif i == 2:
            x = 1
            y = 0
        else:
```

```
            x = -1
            y = 0
        while path:
            if (position[0] + (chain * x), position[1] + (chain * y)) not in
friends_list and \
                    0 <= position[0] + (chain * x) <= 7 and 0 <= position[1]
+ (chain * y) <= 7:
                moves_list.append((position[0] + (chain * x), position[1] +
(chain * y)))
                if (position[0] + (chain * x), position[1] + (chain * y)) in
enemies_list:
                    path = False
                chain += 1
            else:
                path = False
    return moves_list


# comprobar movimientos de peón válidos
def check_pawn(position, color):
    moves_list = []
    if color == 'white':
        if (position[0], position[1] + 1) not in white_locations and \
                (position[0], position[1] + 1) not in black_locations and
position[1] < 7:
            moves_list.append((position[0], position[1] + 1))
        if (position[0], position[1] + 2) not in white_locations and \
                (position[0], position[1] + 2) not in black_locations and
position[1] == 1:
            moves_list.append((position[0], position[1] + 2))
        if (position[0] + 1, position[1] + 1) in black_locations:
            moves_list.append((position[0] + 1, position[1] + 1))
        if (position[0] - 1, position[1] + 1) in black_locations:
            moves_list.append((position[0] - 1, position[1] + 1))
    else:
        if (position[0], position[1] - 1) not in white_locations and \
                (position[0], position[1] - 1) not in black_locations and
position[1] > 0:
            moves_list.append((position[0], position[1] - 1))
        if (position[0], position[1] - 2) not in white_locations and \
                (position[0], position[1] - 2) not in black_locations and
position[1] == 6:
            moves_list.append((position[0], position[1] - 2))
        if (position[0] + 1, position[1] - 1) in white_locations:
            moves_list.append((position[0] + 1, position[1] - 1))
```

```
        if (position[0] - 1, position[1] - 1) in white_locations:
            moves_list.append((position[0] - 1, position[1] - 1))

def draw_captured():
    for i in range(len(captured_pieces_white)):
        captured_piece = captured_pieces_white[i]
        index = piece_list.index(captured_piece)
        screen.blit(small_black_images[index], (825, 5 + 50 * i))
    for i in range(len(captured_pieces_black)):
        captured_piece = captured_pieces_black[i]
        index = piece_list.index(captured_piece)
        screen.blit(small_white_images[index], (925, 5 + 50 * i))


# dibuja un cuadrado parpadeante alrededor del rey si está en jaque
def draw_check():
    if turn_step < 2:
        if 'king' in white_pieces:
            king_index = white_pieces.index('king')
            king_location = white_locations[king_index]
            for i in range(len(black_options)):
                if king_location in black_options[i]:
                    if counter < 15:
                        pygame.draw.rect(screen, 'dark red',
[white_locations[king_index][0] * 100 + 1,
                                                                white_location
s[king_index][1] * 100 + 1, 100, 100], 5)
    else:
        if 'king' in black_pieces:
            king_index = black_pieces.index('king')
            king_location = black_locations[king_index]
            for i in range(len(white_options)):
                if king_location in white_options[i]:
                    if counter < 15:
                        pygame.draw.rect(screen, 'dark blue',
[black_locations[king_index][0] * 100 + 1,
                                                                black_locatio
ns[king_index][1] * 100 + 1, 100, 100], 5)


def draw_game_over():
    pygame.draw.rect(screen, 'black', [200, 200, 400, 70])
    screen.blit(font.render(f'{winner} won the game!', True, 'white'), (210,
210))
```

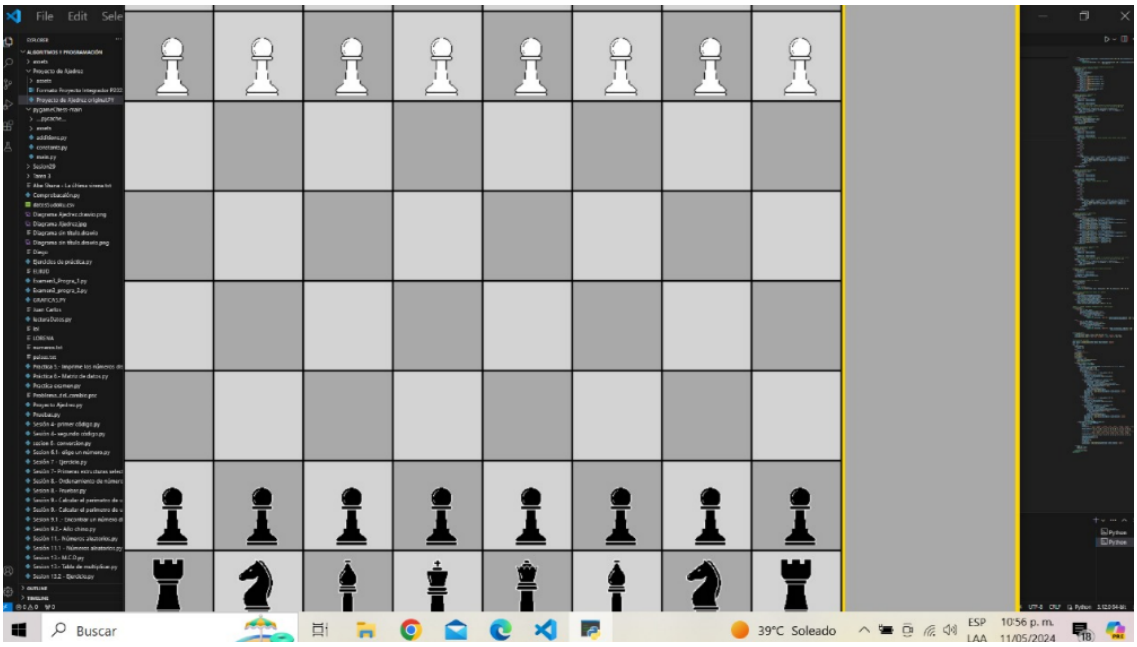
```
    screen.blit(font.render(f'Press ENTER to Restart!', True, 'white'),
(210, 240))


# bucle principal del juego
black_options = check_options(black_pieces, black_locations, 'black')
white_options = check_options(white_pieces, white_locations, 'white')
run = True
while run:
    timer.tick(fps)
    if counter < 30:
        counter += 1
    else:
        counter = 0
    screen.fill('dark gray')
    draw_board()
    draw_pieces()
    draw_captured()
    draw_check()
    if selection != 100:
        valid_moves = check_valid_moves()
        draw_valid(valid_moves)
    # manejo de eventos
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1 and
not game_over:
            x_coord = event.pos[0] // 100
            y_coord = event.pos[1] // 100
            click_coords = (x_coord, y_coord)
            if turn_step <= 1:
                if click_coords == (8, 8) or click_coords == (9, 8):
                    winner = 'black'
                if click_coords in white_locations:
                    selection = white_locations.index(click_coords)
                    if turn_step == 0:
                        turn_step = 1
                if click_coords in valid_moves and selection != 100:
                    white_locations[selection] = click_coords
                    if click_coords in black_locations:
                        black_piece = black_locations.index(click_coords)
                        captured_pieces_white.append(black_pieces[black_piec
e])
                        if black_pieces[black_piece] == 'king':
```

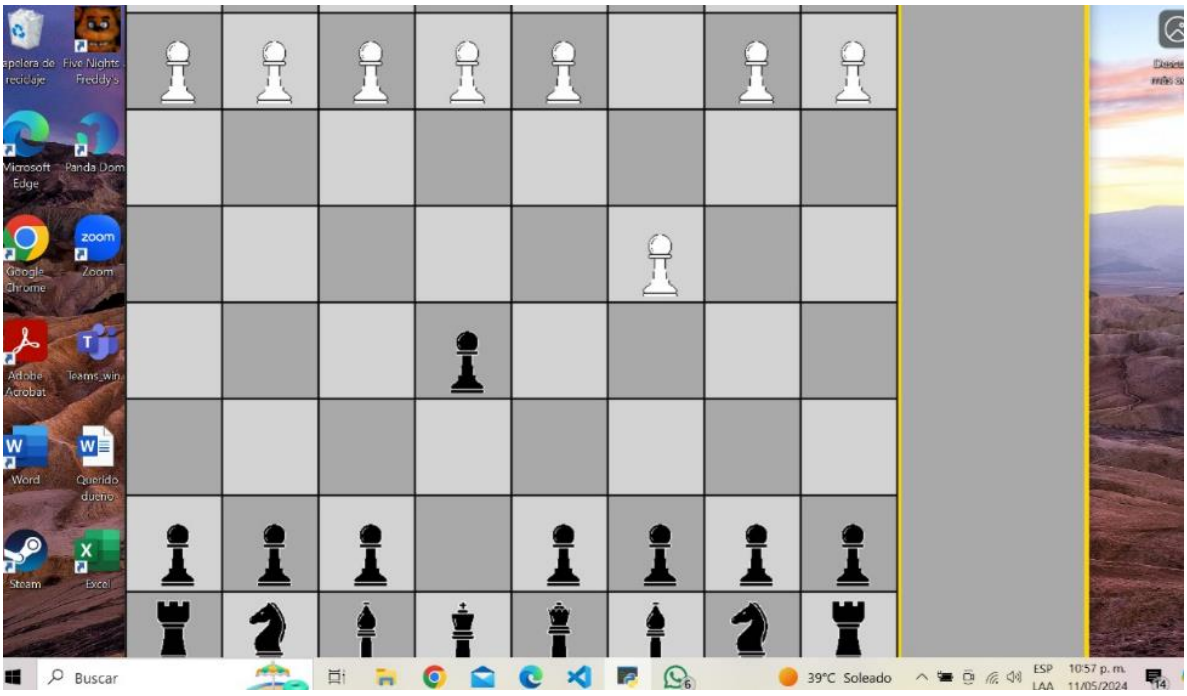
```
                        winner = 'white'
                    black_pieces.pop(black_piece)
                    black_locations.pop(black_piece)
                black_options = check_options(black_pieces,
black_locations, 'black')
                white_options = check_options(white_pieces,
white_locations, 'white')
                turn_step = 2
                selection = 100
                valid_moves = []
            if turn_step > 1:
                if click_coords == (8, 8) or click_coords == (9, 8):
                    winner = 'white'
                if click_coords in black_locations:
                    selection = black_locations.index(click_coords)
                    if turn_step == 2:
                        turn_step = 3
                if click_coords in valid_moves and selection != 100:
                    black_locations[selection] = click_coords
                    if click_coords in white_locations:
                        white_piece = white_locations.index(click_coords)
                        captured_pieces_black.append(white_pieces[white_piec
e])
                        if white_pieces[white_piece] == 'king':
                            winner = 'black'
                        white_pieces.pop(white_piece)
                        white_locations.pop(white_piece)
                    black_options = check_options(black_pieces,
black_locations, 'black')
                    white_options = check_options(white_pieces,
white_locations, 'white')
                    turn_step = 0
                    selection = 100
                    valid_moves = []
        if event.type == pygame.KEYDOWN and game_over:
            if event.key == pygame.K_RETURN:
                game_over = False
                winner = ''
                white_pieces = ['rook', 'knight', 'bishop', 'king', 'queen',
'bishop', 'knight', 'rook',
                                'pawn', 'pawn', 'pawn', 'pawn', 'pawn',
'pawn', 'pawn', 'pawn']
                white_locations = [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0),
(5, 0), (6, 0), (7, 0),
```

```
                                (0, 1), (1, 1), (2, 1), (3, 1), (4, 1),
(5, 1), (6, 1), (7, 1)]
                black_pieces = ['rook', 'knight', 'bishop', 'king', 'queen',
'bishop', 'knight', 'rook',
                                'pawn', 'pawn', 'pawn', 'pawn', 'pawn',
'pawn', 'pawn', 'pawn']
                black_locations = [(0, 7), (1, 7), (2, 7), (3, 7), (4, 7),
(5, 7), (6, 7), (7, 7),
                                (0, 6), (1, 6), (2, 6), (3, 6), (4, 6),
(5, 6), (6, 6), (7, 6)]
                captured_pieces_white = []
                captured_pieces_black = []
                turn_step = 0
                selection = 100
                valid_moves = []
                black_options = check_options(black_pieces, black_locations,
'black')
                white_options = check_options(white_pieces, white_locations,
'white')

    if winner != '':
        game_over = True
        draw_game_over()

    pygame.display.flip()
pygame.quit()
```

Pruebas

<Pantallas de entrada y salida de datos y/o resultados>





Conclusiones

Desarrollar este juego de ajedrez en Python utilizando Pygame fue una experiencia que implicó varios retos, pero también muchas oportunidades para aprender y mejorar nuestras habilidades. A lo largo del proyecto, nos enfrentamos a desafíos técnicos y de diseño, y descubrimos nuevas formas de abordar problemas relacionados con el desarrollo de juegos. Aquí, queremos compartir nuestros aprendizajes y reflexiones sobre el proceso.

Desafíos y Problemas Enfrentados

Uno de los mayores retos fue crear una interfaz gráfica que no solo fuera visualmente atractiva, sino también funcional. Fue necesario diseñar y ubicar las piezas del ajedrez en el tablero correctamente y asegurarse de que se movieran según las reglas del juego. Esto implicó trabajar con coordenadas y gráficos, además de lidiar con posibles errores y excepciones.

Otro desafío clave fue garantizar que el flujo del juego fuera intuitivo y que los dos jugadores pudieran participar de manera fluida. Esto requirió un enfoque cuidadoso para la gestión de eventos, como clics de mouse y teclas presionadas, así como para la validación de movimientos.

Descubrimientos y Aprendizajes

Durante el desarrollo, aprendimos mucho sobre cómo Pygame maneja gráficos y eventos, permitiéndonos crear un tablero de ajedrez interactivo. Descubrimos que es fundamental tener una estructura de código organizada, con módulos y funciones separados para diferentes partes del juego, como dibujar el tablero, mover piezas y manejar capturas.

También adquirimos habilidades en el manejo de errores y en la implementación de bucles y controles de flujo para mantener el juego funcionando sin problemas. Estas habilidades no solo nos ayudaron a resolver problemas técnicos, sino que también nos permitieron diseñar un juego más robusto.

Reflexiones y Conocimientos Nuevos

A medida que trabajábamos en el proyecto, nos dimos cuenta de la importancia del trabajo en equipo y la colaboración. El desarrollo de software es un esfuerzo colectivo, y cada miembro del equipo aportó sus habilidades y experiencia para lograr el resultado final. Además, nos dimos cuenta de que el proceso de desarrollo es iterativo; a medida que avanzábamos, encontramos formas de mejorar el juego y hacerlo más accesible y divertido para los jugadores.

Otra reflexión importante fue sobre la experiencia del usuario. Un juego de ajedrez puede parecer simple, pero garantizar que sea agradable y fácil de usar requiere atención a los detalles y un enfoque centrado en el usuario. Esto nos llevó a ajustar la interfaz y la lógica del juego para crear una experiencia más amigable.

Fuentes de información

- *LeMaster Tech. (2023, 3 abril). How to Make Chess in Python! [Vídeo]. YouTube.* *https://www.youtube.com/watch?v=X-e0jk4I938*
- *Jeff Aporta. (2021, 25 abril). PYGAME & PYTHON ► INSTALACIÓN WINDOWS 10 ► CONFIGURACIÓN EN VISUAL STUDIO CODE [Vídeo]. YouTube.* *https://www.youtube.com/watch?v=Txb5PPDX_70*
- *pygame news*. (s. f.). https://www.pygame.org/news

Anexos

<Información adicional que no clasifica en ninguno de los puntos anteriores>