UNIVERSIDAD NACIONAL DE INGENIERIA

Facultad de Ciencias

Juego de Damas

Alumnos:

Jhiens Angel Guerrero Ccompi	20210145J
Joel Benjamien Seminario Serna	20210056G
Jose Rodolfo Estacio Sánchez	20214027A
Zuñiga Naquiche Gerson Jairo	20152702B

Lima, Perú
1 de Julio de 2025

Índice

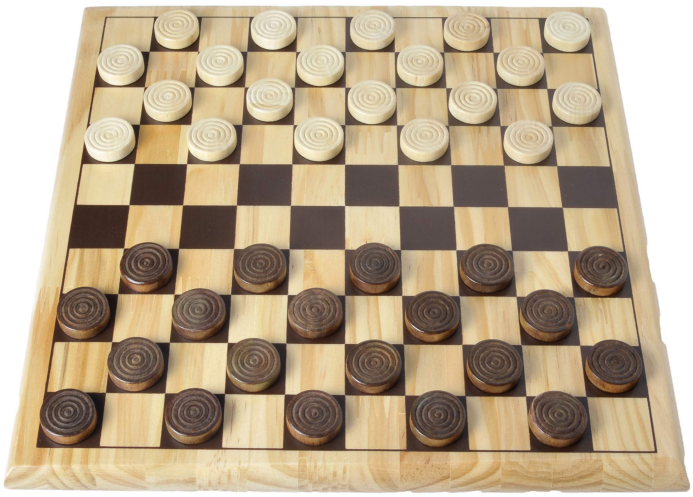
1. Objetivo del Proyecto

El objetivo de este proyecto fue desarrollar un juego de damas clasico jugable por consola en Python, donde el usuario compite contra una inteligencia artificial. Una parte fundamental del proyecto fue implementar estrategias de paralelizacion y secunsializacion , y a final comparar cual es mas eficiente.

2. Descripción del Juego

El juego implementa las reglas clásicas de las damas:

- Movimiento diagonal para piezas normales.
- Captura obligatoria si hay piezas para comer.
- Coronación(convertir a Reynas) de piezas al llegar al extremo del tablero.
- El jugador elige si juega con blancas o negras.



3. Codigo

```
# -*- coding: utf-8 -*-
"""Prueba.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1y_jnsffEnMxkI-
       s_WeKiQDKRuRsxwqbS
"""
```

```
import copy
import time
from functools import lru_cache
from concurrent.futures import ProcessPoolExecutor,
   ThreadPoolExecutor

#  Tablero y movimientos

def crear_tablero():
    tablero = [[' ' for _ in range(8)] for _ in range(8)]
    for fila in range(3):
        for col in range(8):
            if (fila + col) % 2 == 1:
                tablero[fila][col] = 'n'
    for fila in range(5, 8):
        for col in range(8):
            if (fila + col) % 2 == 1:
                tablero[fila][col] = 'b'
    return tablero


def imprimir_tablero(tablero):
    print("    A B C D E F G H")
    print("   +-----------------+")
    for idx, fila in enumerate(tablero):
        linea = f"{idx} |"
        for col in fila:
            linea += f" {col if col != ' ' else '.'}"
        linea += " |"
        print(linea)
    print("   +-----------------+")


def coronar(tablero, x, y):
    if tablero[x][y] == 'b' and x == 0:
        tablero[x][y] = 'B'
    elif tablero[x][y] == 'n' and x == 7:
        tablero[x][y] = 'N'


def mover_ficha(tablero, origen, destino, jugador):
    ox, oy = origen
    dx, dy = destino
    pieza = tablero[ox][oy]
    if pieza.lower() != jugador or tablero[dx][dy] != ' ':
        return False
    direccion = -1 if jugador == 'b' else 1
    if abs(dx-ox)==1 and abs(dy-oy)==1 and (dx-ox)==direccion:
        tablero[dx][dy] = pieza; tablero[ox][oy] = ' '
        coronar(tablero, dx, dy); return True
    if abs(dx-ox)==2 and abs(dy-oy)==2:
```

```
        mx, my = (ox+dx)//2, (oy+dy)//2
        enemigo = 'n' if jugador=='b' else 'b'
        if tablero[mx][my].lower()==enemigo:
            tablero[dx][dy] = pieza; tablero[ox][oy]='␣'; tablero
               [mx][my]='␣'
            coronar(tablero, dx, dy); return True
    return False


def serializar(tablero):
    return ''.join(''.join(fila) for fila in tablero)

def deserializar(tab_str):
    return [list(tab_str[i*8:(i+1)*8]) for i in range(8)]


def mover_y_serializar(tab_str, origen, destino, jugador):
    tablero = deserializar(tab_str)
    mover_ficha(tablero, origen, destino, jugador)
    return serializar(tablero)


def calcular_movimientos(tablero, origen):
    ox, oy = origen
    pieza = tablero[ox][oy]
    if pieza=='␣': return []
    jugador = pieza.lower()
    dirs = [(-1,-1),(-1,1)] if jugador=='b' else [(1,-1),(1,1)]
    if pieza in ('B','N'): dirs = [(-1,-1),(-1,1),(1,-1),(1,1)]
    movs = []
    for dx,dy in dirs:
        nx,ny = ox+dx, oy+dy
        if 0<=nx<8 and 0<=ny<8 and tablero[nx][ny]=='␣':
            movs.append((origen,(nx,ny)))
        cx,cy = ox+2*dx, oy+2*dy
        mx,my = ox+dx, oy+dy
        if (0<=cx<8 and 0<=cy<8 and tablero[cx][cy]=='␣' and
            tablero[mx][my].lower() in ('b','n') and tablero[mx][
               my].lower()!=jugador):
            movs.append((origen,(cx,cy)))
    return movs


def movimientos_jugador(tablero, jugador):
    allm=[]
    posiciones = [(x, y) for x in range(8) for y in range(8) if
       tablero[x][y].lower()==jugador]
    with ThreadPoolExecutor() as executor:
        resultados = executor.map(lambda pos:
           calcular_movimientos(tablero, pos), posiciones)
    for res in resultados:
```

```
        allm.extend(res)
    return allm

#  Riesgo de captura

def en_riesgo(tablero, pos, jugador):
    x, y = pos
    enemigo = 'n' if jugador == 'b' else 'b'
    dirs = [(-1, -1), (-1, 1), (1, -1), (1, 1)]
    for dx, dy in dirs:
        mx, my = x + dx, y + dy
        cx, cy = x - dx, y - dy
        if (0 <= mx < 8 and 0 <= my < 8 and
            0 <= cx < 8 and 0 <= cy < 8 and
            tablero[mx][my].lower() == enemigo and tablero[cx][cy
               ] == '␣'):
            return True
    return False

#  Evaluaci n y memoizaci n

@lru_cache(maxsize=None)
def evaluar_tablero_memo(tab_str, jugador):
    enemigo = 'n' if jugador=='b' else 'b'
    score=0
    for c in tab_str:
        if c.lower()==jugador: score += 3 if c.isupper() else 1
        elif c.lower()==enemigo: score -= 3 if c.isupper() else 1
    return score

#  Minimax secuencial

def minimax_secuencial_str(tab_str, jugador, prof, maximizando):
    if prof==0:
        return evaluar_tablero_memo(tab_str, jugador), None
    tablero = deserializar(tab_str)
    turno = jugador if maximizando else ('n' if jugador=='b' else
        'b')
    movs = movimientos_jugador(tablero, turno)
    best_val = float('-inf') if maximizando else float('inf')
    best_mov = None
    for ori,dest in movs:
        new_str = mover_y_serializar(tab_str, ori, dest, turno)
        val,_ = minimax_secuencial_str(new_str, jugador, prof-1,
           not maximizando)
        if (maximizando and val>best_val) or (not maximizando and
            val<best_val):
            best_val, best_mov = val, (ori,dest)
    return best_val, best_mov

#  Minimax paralelo
```

```
def _eval_branch_str(args):
    tab_str, jugador, prof, mov = args
    new_str = mover_y_serializar(tab_str, mov[0], mov[1], jugador
       )
    val,_ = minimax_secuencial_str(new_str, jugador, prof-1,
       False)
    return val, mov


def minimax_paralelo(tablero, jugador, prof):
    movs = movimientos_jugador(tablero, jugador)
    if not movs: return None
    tab_str = serializar(tablero)
    tasks = [(tab_str, jugador, prof, m) for m in movs]
    with ProcessPoolExecutor() as executor:
        results = executor.map(_eval_branch_str, tasks)
    return max(results, key=lambda x: x[0])[1]

#  Comparaci n detallada paso a paso

def mostrar_movimientos(movs):
    return [f"{chr(o[1]+65)}{o[0]}->{chr(d[1]+65)}{d[0]}" for o,d
        in movs]


def clasificar_movimientos(tablero, jugador):
    sin_captura, con_captura, en_peligro = [], [], []
    todos = movimientos_jugador(tablero, jugador)
    for ori, dest in todos:
        if abs(dest[0] - ori[0]) == 2:
            con_captura.append((ori, dest))
        else:
            sin_captura.append((ori, dest))
    for x in range(8):
        for y in range(8):
            if tablero[x][y].lower() == jugador and en_riesgo(
               tablero, (x, y), jugador):
                en_peligro.append((x, y))
    return sin_captura, con_captura, en_peligro


def comparar_jugadas(n_turnos, prof):
    tiempos = {}
    for modo in ['paralelo', 'secuencial']:
        print(f"\n--- MODO {modo.upper()} ---")
        tablero = crear_tablero()
        tiempo = 0
        turno = 'b'
        for i in range(n_turnos):
```

```
            print(f"\nTurno␣{i+1}:␣{'Blancas'␣if␣turno=='b'␣else␣
                'Negras'}")
            imprimir_tablero(tablero)

            sin_cap, con_cap, riesgo = clasificar_movimientos(
                tablero, turno)
            print("Movimientos␣con␣captura:", mostrar_movimientos
                (con_cap))
            print("Movimientos␣sin␣captura:", mostrar_movimientos
                (sin_cap))
            print("Fichas␣en␣riesgo:", [f"{chr(y+65)}{x}" for x,y
                in riesgo])

            t1 = time.perf_counter()
            if modo == 'paralelo':
                mejor = minimax_paralelo(tablero, turno, prof)
            else:
                _, mejor = minimax_secuencial_str(serializar(
                    tablero), turno, prof, True)
            t2 = time.perf_counter()
            tiempo += (t2 - t1)

            if mejor:
                print("Mejor␣jugada␣IA:", mostrar_movimientos([
                    mejor])[0])
                mover_ficha(tablero, mejor[0], mejor[1], turno)
            else:
                print("Sin␣jugadas␣posibles.␣Fin␣anticipado.")
                break

            turno = 'n' if turno == 'b' else 'b'

        tiempos[modo] = tiempo

    print("\n---␣RESUMEN␣DE␣TIEMPOS␣---")
    print(f"Total␣en␣paralelo:␣␣␣{tiempos['paralelo']:.4f}␣
        segundos")
    print(f"Total␣en␣secuencial:␣{tiempos['secuencial']:.4f}␣
        segundos")

if __name__ == '__main__':
    n = int(input("  Cuntos  ␣turnos␣deseas␣simular␣(IA␣vs␣IA)?␣"
        ))
    prof = int(input("  Profundidad  ␣del␣algoritmo␣Minimax?␣"))
    comparar_jugadas(n, prof)
```

4. Explicación del Paralelismo y Secuencialidad

Programación Paralela

```
def minimax_paralelo(tablero, jugador, prof):
    movs = movimientos_jugador(tablero, jugador)
    if not movs: return None
    tab_str = serializar(tablero)
    tasks = [(tab_str, jugador, prof, m) for m in movs]
    with ProcessPoolExecutor() as executor:
        results = executor.map(_eval_branch_str, tasks)
    return max(results, key=lambda x: x[0])[1]
```

Explicación: Esta función implementa una versión paralela del algoritmo Minimax, diseñada para analizar múltiples movimientos posibles de un jugador en un juego de damas de forma simultánea, aprovechando múltiples núcleos del procesador.

- **1.** `movimientos_jugador(...)`: Genera todos los movimientos posibles del jugador actual.
- **2.** Si no hay movimientos, retorna `None`.
- **3.** Convierte el tablero a cadena con `serializar`.
- **4.** Crea una lista de tareas con los movimientos posibles.
- **5.** Usa `ProcessPoolExecutor` para evaluar en paralelo.
- **6.** Devuelve el movimiento con mayor puntuación.

```
def movimientos_jugador(tablero, jugador):
    allm=[]
    posiciones = [(x, y) for x in range(8) for y in range(8) if
       tablero[x][y].lower()==jugador]
    with ThreadPoolExecutor() as executor:
        resultados = executor.map(lambda pos:
           calcular_movimientos(tablero, pos), posiciones)
    for res in resultados:
        allm.extend(res)
    return allm
```

Explicación: Calcula todos los movimientos posibles que un jugador puede hacer usando hilos.

- **1.** Identifica todas las fichas del jugador en el tablero.
- **2.** Usa `ThreadPoolExecutor` para distribuir el cálculo por ficha.
- **3.** Combina todos los movimientos en una sola lista.
- **4.** Retorna la lista completa de movimientos.

Programación Secuencial

```
def minimax_secuencial_str(tab_str, jugador, prof, maximizando):
    if prof==0:
        return evaluar_tablero_memo(tab_str, jugador), None
    tablero = deserializar(tab_str)
    turno = jugador if maximizando else ('n' if jugador=='b' else
        'b')
    movs = movimientos_jugador(tablero, turno)
    best_val = float('-inf') if maximizando else float('inf')
    best_mov = None
    for ori,dest in movs:
        new_str = mover_y_serializar(tab_str, ori, dest, turno)
        val,_ = minimax_secuencial_str(new_str, jugador, prof-1,
           not maximizando)
        if (maximizando and val>best_val) or (not maximizando and
            val<best_val):
            best_val, best_mov = val, (ori,dest)
    return best_val, best_mov
```

Explicación: Esta función implementa el algoritmo Minimax de forma secuencial y recursiva.

Parámetros:

- `tab_str`: estado del tablero como cadena.
- `jugador`: jugador actual ('b' o 'n').
- `prof`: profundidad de búsqueda.
- `maximizando`: si se busca maximizar o minimizar el resultado.

Pasos:

1. Si la profundidad es 0, evalúa el tablero y retorna.
2. Convierte `tab_str` a matriz.
3. Determina el turno según si está maximizando o no.
4. Genera todos los movimientos posibles.
5. Llama recursivamente a la función para cada movimiento.
6. Selecciona el mejor valor y lo retorna.

5. Resultados

Ejemplo de impresión en consola:

```
--- RESUMEN DE TIEMPOS ---
Total en paralelo:   1.4285 segundos
Total en secuencial: 4.9213 segundos
```

El modo paralelo demostró ser mucho más eficiente que el secuencial, especialmente en profundidades mayores del árbol Minimax.