Universidad Peruana de Ciencias Aplicadas

Facultad de Ingeniería

Ciclo 04

Nombre del curso:
Complejidad Algorítmica

Sección:
CC42

Nombre del profesor:
Luis Martin Carnaval Sanchez

"HITO 02"

Relación de integrantes:

Rojas Cuadros Fabian Marcelo - U202218498
Su Caletti Eddo - U20221A390
Alexander Fernandez Garfias - U202019498

Índice de contenido

1. Descripción del problema:

Este proyecto se centra en el desarrollo de un código para simular partidas del juego de damas, en el cual los movimientos de las piezas se realizan de manera aleatoria. Los movimientos se determinarán en función de la estructura del tablero, el tamaño del mismo y las piezas involucradas, así como el número de jugadores. El objetivo es evaluar cómo la disposición inicial y las condiciones del juego influyen en los movimientos posibles, utilizando técnicas para encontrar soluciones óptimas a partir de movimientos generados aleatoriamente.

Según Gupta et al. (2021), el juego de damas puede modelarse eficazmente utilizando un grafo, donde cada casilla del tablero representa un nodo y los movimientos legales entre las casillas se representan como arcos del grafo.

Este enfoque permite aplicar estrategias de búsqueda heurística para optimizar las decisiones del programa, explorando diferentes configuraciones del tablero y movimientos posibles de manera sistemática En este contexto, las técnicas de búsqueda heurística son fundamentales para resolver problemas complejos como el juego de damas, proporcionando métodos eficientes para explorar y evaluar opciones de juego (Allis, 1994; López, 2022). Estas estrategias permiten al programa adaptarse dinámicamente a las condiciones cambiantes del juego, mejorando la calidad de las decisiones tomadas durante la simulación de partidas

2. Descripción del Dataset

El conjunto de datos utilizado para la simulación se genera a partir de la estructura del tablero de damas, modelado como un grafo donde las casillas representan los nodos y los movimientos permitidos entre casillas adyacentes o a través de saltos representan los arcos. Para la gestión y procesamiento de estos datos se utilizan las siguientes herramientas y librerías:

- **Python:** Lenguaje de programación para desarrollar el simulador.
- **Librería networkx:** Permite crear y manipular el grafo que representa el tablero y las conexiones entre las casillas.
- **Librería numpy:** Utilizada para manejar las matrices de adyacencia que representan las conexiones entre los nodos
- **Librería matplotlib**: Empleada para la visualización gráfica del tablero y las posiciones de las piezas. 2.1 Estructura

2.1. Estructura del dataset:(grafos)

El dataset incluye las siguientes columnas:

- **Nodo Inicial**: Coordenadas de la posición inicial (x, y).
- **Nodo Final**: Coordenadas de la posición de destino (x, y).
- **Tipo de Movimiento:** "Simple" (movimiento adyacente) o "Salto" (movimiento sobre una ficha).
- **Jugador**: Identifica al jugador (1 o 2).
- **Estado del Nodo:** Indica si la posición está ocupada (1) o libre (0).

Nodo Inicial	Nodo Final	Tipo de Movimiento	Jugador	Estado del Nodo
(0, 0)	**(0, 1)**	**Simple**	**1**	**0**
(1, 1)	**(0, 2)**	**Salto**	**2**	**1**
(2, 0)	**(1, 0)**	**Simple**	**1**	**0**

Matriz de Adyacencia

La matriz de adyacencia representa los movimientos posibles en el tablero, donde cada celda indica si existe una conexión entre dos posiciones. Por ejemplo, si se verifica el movimiento de (1, 1) a (0, 1), un valor de 1 en la celda correspondiente indica que el movimiento es válido.

Ejemplo para Tablero 3x3

Uso de la Matriz

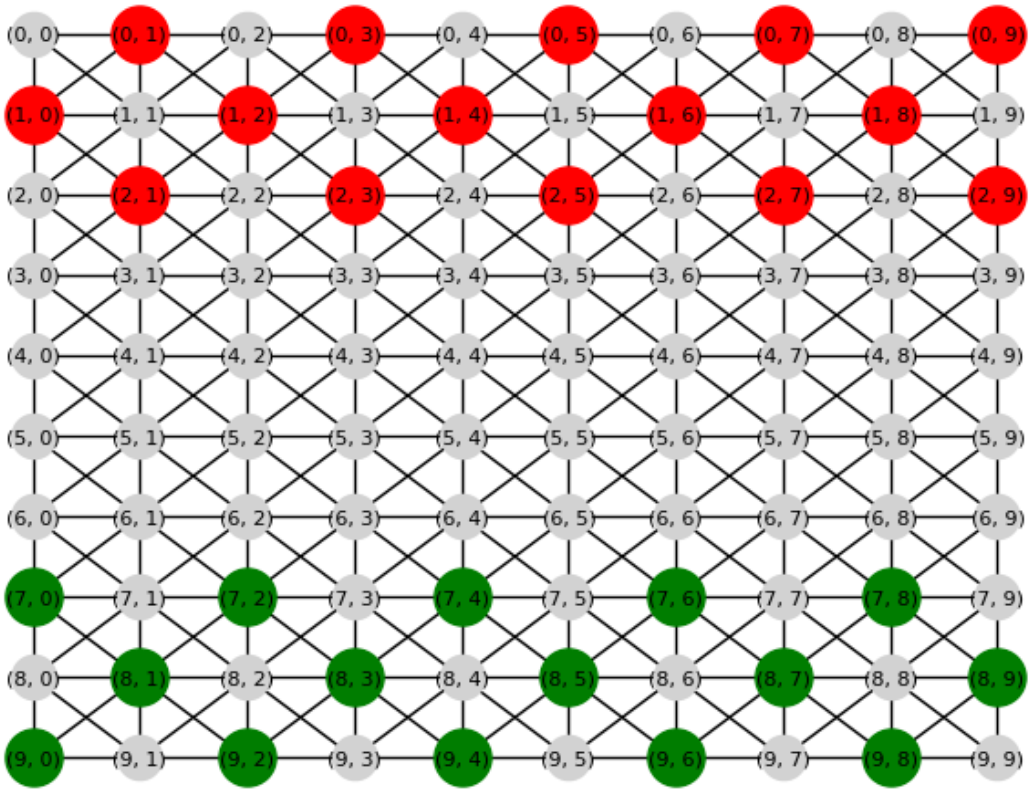
	(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	(2,0)	(2,1)	(2,2)
(0,0)	0	1	0	1	0	0	0	0	0
(0,1)	1	0	1	0	1	0	0	0	0
(0,2)	0	1	0	0	0	1	0	0	0
(1,0)	1	0	0	0	1	0	1	0	0
(1,1)	0	1	0	1	0	1	0	1	0
(1,2)	0	0	1	0	1	0	0	0	1
(2,0)	0	0	0	1	0	0	0	1	0
(2,1)	0	0	0	0	1	0	1	0	1
(2,2)	0	0	0	0	0	1	0	1	0

Si se quiere verificar si un movimiento de (1,1)(1, 1)(1,1) a (0,1)(0, 1)(0,1) es permitido, se consulta la celda (1,1),(0,1)(1, 1), (0, 1)(1,1),(0,1). Un valor de 1 indica que el movimiento es válido.

```
matriz_adyacencia = [

    [0, 1, 0, 1, 0, 0, 0, 0, 0],

    [1, 0, 1, 0, 1, 0, 0, 0, 0],

    [0, 1, 0, 0, 0, 1, 0, 0, 0],

    [1, 0, 0, 0, 1, 0, 1, 0, 0],

    [0, 1, 0, 1, 0, 1, 0, 1, 0],

    [0, 0, 1, 0, 1, 0, 0, 0, 1],

    [0, 0, 0, 1, 0, 0, 0, 1, 0],

    [0, 0, 0, 0, 1, 0, 1, 0, 1],

    [0, 0, 0, 0, 0, 1, 0, 1, 0]

]



def es_movimiento_legal(pos_inicial, pos_final):

    return matriz_adyacencia[pos_inicial][pos_final] == 1
```

```
print(es_movimiento_legal(0, 1))  # True
print(es_movimiento_legal(0, 2))  # False
```

Estas herramientas permiten simular el comportamiento del juego y registrar los movimientos generados para su análisis posterior.



La imagen muestra un modelo gráfico del tablero de 10x10 utilizado en el juego. Cada casilla está representada como un nodo conectado con sus adyacentes mediante aristas, simulando las posibles rutas de movimiento. Las fichas del jugador 1 aparecen en rojo y las del jugador 2 en verde. Esta visualización referencial permite entender cómo se organizan las posiciones iniciales de las fichas y sus conexiones en el tablero.

3. Propuesta

En esta sección se detallan los objetivos del proyecto, la técnica seleccionada y la metodología empleada para el desarrollo del juego de Damas Chinas.

3.1. Objetivos

El principal objetivo del proyecto es implementar una versión del juego de Damas Chinas con una interfaz gráfica amigable que permita a los usuarios competir contra algoritmos de toma de decisiones eficientes. Los objetivos específicos son:

- Crear un tablero interactivo que permita a los jugadores mover fichas y visualizar sus movimientos en tiempo real.
- Desarrollar algoritmos inteligentes, como Fuerza Bruta y Backtracking, para enfrentar al usuario en el juego, aumentando la dificultad.
- Implementar validaciones precisas para movimientos legales, utilizando una matriz de adyacencia para la representación del tablero.
- Brindar una experiencia fluida y entretenida, con opciones de personalización en el nivel de dificultad.

3.2. Técnica

Para la implementación del proyecto, se optó por modelar el tablero de Damas Chinas como un grafo, donde cada posición del tablero es un nodo y los movimientos permitidos son aristas que conectan estos nodos. Las técnicas utilizadas incluyen:

- **Matriz de Adyacencia**: Representa el tablero y facilita la validación rápida de movimientos.
- **Algoritmos de Fuerza Bruta, Backtracking y Mini-Max**: Fuerza Bruta explora todos los movimientos posibles, mientras que Backtracking optimiza la búsqueda

eliminando caminos no válidos, Mini-Max selecciona movimientos óptimos en juegos de competencia, optimizando mediante la evaluación de estados y, si es posible, con poda alfa-beta para reducir el número de evaluaciones.
- **Interfaz Gráfica con Tkinter**: Proporciona una interfaz visual interactiva para el usuario, permitiendo ver el tablero y mover las fichas con facilidad.

3.3. Metodología

La metodología utilizada en el desarrollo del proyecto es una combinación de **Desarrollo Iterativo** y **Metodología Ágil**, permitiendo realizar ajustes según el feedback obtenido y avanzar de manera incremental:

- **Análisis del Problema**: Se definieron los requisitos y se estudió el comportamiento del juego de Damas
- **Modelado del Tablero**: Se diseñó una representación del tablero como grafo, utilizando una matriz de adyacencia.
- **Desarrollo de Algoritmos**: Se implementaron los algoritmos de Fuerza Bruta y Backtracking para la toma de decisiones.
- **Pruebas y Validación**: Se realizaron pruebas con diferentes configuraciones para validar los movimientos y el comportamiento de los algoritmos.
- **Iteraciones**: Se realizaron mejoras continuas basadas en los resultados de las pruebas y feedback del usuario.

4. Diseño de la Aplicación

En esta sección se describe el proceso de diseño del aplicativo, considerando las etapas de la ingeniería de software y el análisis de algoritmos.

4.1. Análisis de Requisitos

Se identificaron los requisitos funcionales y no funcionales del aplicativo:

- **Requisitos Funcionales**:
  - El usuario debe poder seleccionar el nivel de dificultad (algoritmo contra el cual competir).
  - El tablero debe ser visualizado de manera interactiva.
  - El sistema debe validar los movimientos y actualizar el tablero en tiempo real.
- **Requisitos No Funcionales**:
  - El aplicativo debe ser rápido y responder de manera fluida.
  - La interfaz debe ser intuitiva y amigable para usuarios novatos.

4.2. Diseño del Sistema

El diseño del sistema se llevó a cabo utilizando un enfoque modular, dividiendo el aplicativo en varios componentes:

- **Módulo de Interfaz Gráfica**:
  - Utiliza la biblioteca tkinter para la visualización del tablero.
  - Permite al usuario iniciar el juego, mover fichas y ver los movimientos del oponente (algoritmo).
- **Módulo de Validación de Movimientos**:
  - Utiliza la matriz de adyacencia para validar si un movimiento es permitido.
  - Actualiza el estado del tablero después de cada movimiento.
- **Módulo de Algoritmos**:

    - Implementa los algoritmos de Fuerza Bruta y Backtracking para calcular los movimientos del oponente.
    - Integra los algoritmos con la lógica del juego para desafiar al usuario.

4.3. Proceso de Desarrollo

El proceso de desarrollo siguió las siguientes etapas:

- **Modelado del Tablero**:
    - Se creó una matriz de adyacencia para representar el grafo del tablero, facilitando la validación de movimientos.
- **Implementación de Algoritmos**:
    - Se implementaron y probaron los algoritmos de Fuerza Bruta y Backtracking.
    - Se optimizaron los algoritmos para mejorar el rendimiento y la experiencia de juego.
- **Desarrollo de la Interfaz Gráfica**:
    - Se construyó la interfaz utilizando tkinter, permitiendo la interacción del usuario con el tablero.
    - Se añadieron botones y mensajes para mejorar la experiencia del usuario.
- **Pruebas y Mejora Continua**:
    - Se realizaron pruebas con diferentes configuraciones del tablero y niveles de dificultad.
    - Se corrigieron errores y se realizaron ajustes basados en el feedback.

5. Validación de resultados y pruebas

```
import tkinter as tk
from tkinter import ttk, messagebox
from PIL import Image, ImageTk
import pygame
import random
from copy import deepcopy

ANCHO, ALTURA = 600, 600
FILAS, COLUMNAS = 8, 8
PERIMETRO_CUADRADO = ANCHO // COLUMNAS

MARRON_OSCURO = (139, 69, 19)
KHAKI = (240, 230, 140)
ROJO = (255, 0, 0)
NEGRO = (0, 0, 0)
GRIS = (128, 128, 128)
AZUL = (59, 131, 189)
CORONA = pygame.transform.scale(pygame.image.load("corona.jpeg"),
(45, 25))

class Piezas:
    RELLENO = 15
    BORDE = 2

    def __init__(self, color, fil, col):
        self.fil = fil
        self.col = col
        self.color = color
        self.king = False
        self.x = 0
```

```
        self.y = 0
        self.calc_pos()

    def calc_pos(self):
        self.x = PERIMETRO_CUADRADO * self.col + PERIMETRO_CUADRADO
// 2
        self.y = PERIMETRO_CUADRADO * self.fil + PERIMETRO_CUADRADO
// 2

    def make_king(self):
        self.king = True

    def draw(self, win):
        radio = PERIMETRO_CUADRADO // 2 - self.RELLENO
        pygame.draw.circle(win, GRIS, (self.x, self.y), radio +
self.BORDE)
        pygame.draw.circle(win, self.color, (self.x, self.y), radio)
        if self.king:
            win.blit(CORONA, (self.x - CORONA.get_width() // 2,
self.y - CORONA.get_height() // 2))

    def move(self, fil, col):
        self.fil = fil
        self.col = col
        self.calc_pos()

class Tablero:
    def __init__(self):
        self.tablero = []
        self.ROJO_left = self.NEGRO_left = 12
        self.ROJO_Kings = self.NEGRO_kings = 0
        self.crear_tablero()

    def draw_cuadrados(self, win):
        win.fill(KHAKI)
        for fil in range(FILAS):
            for col in range(fil % 2, COLUMNAS, 2):
                pygame.draw.rect(win, MARRON_OSCURO, (col *
PERIMETRO_CUADRADO, fil * PERIMETRO_CUADRADO, PERIMETRO_CUADRADO,
PERIMETRO_CUADRADO))
```

```
    def crear_tablero(self):
        for fil in range(FILAS):
            self.tablero.append([])
            for col in range(COLUMNAS):
                if col % 2 == ((fil + 1) % 2):
                    if fil < 3:
                        self.tablero[fil].append(Piezas(NEGRO, fil,
col))
                    elif fil > 4:
                        self.tablero[fil].append(Piezas(ROJO, fil,
col))
                    else:
                        self.tablero[fil].append(0)
                else:
                    self.tablero[fil].append(0)

    def draw(self, win):
        self.draw_cuadrados(win)
        for fil in range(FILAS):
            for col in range(COLUMNAS):
                pieza = self.tablero[fil][col]
                if pieza != 0:
                    pieza.draw(win)

    def move(self, pieza, fil, col):
        self.tablero[pieza.fil][pieza.col], self.tablero[fil][col] =
self.tablero[fil][col], self.tablero[pieza.fil][pieza.col]
        pieza.move(fil, col)
        if (pieza.color == ROJO and fil == 0) or (pieza.color ==
NEGRO and fil == FILAS - 1):
            pieza.make_king()

    def obtener_movimientos_validos(self, pieza):
        movimientos = {}
        fil, col = pieza.fil, pieza.col
        direcciones = [(-1, -1), (-1, 1), (1, -1), (1, 1)]
        movimientos.update(self._explorar(pieza, fil, col,
direcciones))
        return movimientos
```

```
    def _explorar(self, pieza, fil, col, direcciones):
        movimientos = {}

        if not pieza.king:
            if pieza.color == ROJO:
                direcciones = [(-1, -1), (-1, 1)]  # hacia arriba
            else:
                direcciones = [(1, -1), (1, 1)]  # hacia abajo

        for dfil, dcol in direcciones:
            f, c = fil + dfil, col + dcol
            if 0 <= f < FILAS and 0 <= c < COLUMNAS:
                casilla_destino = self.tablero[f][c]
                if casilla_destino == 0:
                    # Movimiento simple
                    movimientos[(f, c)] = []
                elif casilla_destino.color != pieza.color:
                    # Detectar salto
                    salto_f, salto_c = f + dfil, c + dcol
                    if (
                        0 <= salto_f < FILAS
                        and 0 <= salto_c < COLUMNAS
                        and self.tablero[salto_f][salto_c] == 0
                    ):
                        movimientos[(salto_f, salto_c)] = [(f, c)]
        return movimientos

    def evaluate(self):
        return self.ROJO_left - self.NEGRO_left

class Juego:
    def __init__(self, win, algoritmo):
        self.win = win
        self.algoritmo = algoritmo
        self._init()

    def _init(self):
        self.selected = None
        self.tablero = Tablero()
```

```
        self.turn = ROJO
        self.movimientos_validos = {}

    def update(self):
        self.tablero.draw(self.win)
        self.draw_movimientos_validos(self.movimientos_validos)
        pygame.display.update()

    def reset(self):
        self._init()

    def select(self, fil, col):
        if self.selected:
            result = self._move(fil, col)
            if not result:
                self.selected = None
                self.select(fil, col)
        pieza = self.tablero.tablero[fil][col]
        if pieza != 0 and pieza.color == self.turn:
            self.selected = pieza
            self.movimientos_validos =
self.tablero.obtener_movimientos_validos(pieza)
            return True
        return False

    def _move(self, fil, col):
        if (fil, col) in self.movimientos_validos:
            self.tablero.move(self.selected, fil, col)
            if self.movimientos_validos[(fil, col)]:
                for f, c in self.movimientos_validos[(fil, col)]:
                    self.tablero.tablero[f][c] = 0
                    if self.selected.color == ROJO:
                        self.tablero.NEGRO_left -= 1
                    else:
                        self.tablero.ROJO_left -= 1

            if (self.selected.color == ROJO and fil == 0) or
(self.selected.color == NEGRO and fil == FILAS - 1):
                self.selected.make_king()
            self.change_turn()
```

```
            return True
        return False

    def change_turn(self):
        self.selected = None
        self.movimientos_validos = {}
        self.turn = NEGRO if self.turn == ROJO else ROJO
        self.check_winner()

    def check_winner(self):
        if self.tablero.ROJO_left <= 0:
            self.show_winner_message("¡Negro ganó!")
            return True
        elif self.tablero.NEGRO_left <= 0:
            self.show_winner_message("¡Rojo ganó!")
            return True
        return False

    def show_winner_message(self, message):
        pygame.quit()
        root = tk.Tk()
        root.withdraw()
        messagebox.showinfo("Juego Terminado", message)
        root.destroy()

    def draw_movimientos_validos(self, movimientos):
        for mov in movimientos:
            fil, col = mov
            pygame.draw.circle(self.win, AZUL, (col *
PERIMETRO_CUADRADO + PERIMETRO_CUADRADO // 2, fil *
PERIMETRO_CUADRADO + PERIMETRO_CUADRADO // 2), 15)

def get_fil_col_from_mouse(pos):
    x, y = pos
    fil = y // PERIMETRO_CUADRADO
    col = x // PERIMETRO_CUADRADO
    return fil, col

def backtracking(juego):
    def minimax(tablero, depth, max_player):
```

```
        if depth == 0 or juego.check_winner():
            return tablero.evaluate(), tablero

        if max_player:
            max_eval = float('-inf')
            best_move = None
            for move in get_all_moves(tablero, NEGRO):
                evaluation = minimax(move, depth - 1, False)[0]
                max_eval = max(max_eval, evaluation)
                if max_eval == evaluation:
                    best_move = move
            return max_eval, best_move
        else:
            min_eval = float('inf')
            best_move = None
            for move in get_all_moves(tablero, ROJO):
                evaluation = minimax(move, depth - 1, True)[0]
                min_eval = min(min_eval, evaluation)
                if min_eval == evaluation:
                    best_move = move
            return min_eval, best_move

    def get_all_moves(tablero, color):
        moves = []
        for fil in range(FILAS):
            for col in range(COLUMNAS):
                pieza = tablero.tablero[fil][col]
                if pieza != 0 and pieza.color == color:
                    valid_moves =
tablero.obtener_movimientos_validos(pieza)
                    for move, skip in valid_moves.items():
                        temp_tablero = deepcopy(tablero)
                        temp_pieza = temp_tablero.tablero[fil][col]
                        new_tablero = simulate_move(temp_pieza, move,
temp_tablero, skip)
                        moves.append(new_tablero)
        return moves

    def simulate_move(pieza, move, tablero, skip):
        tablero.move(pieza, move[0], move[1])
```

```
        if skip:
            for f, c in skip:
                tablero.tablero[f][c] = 0
                if pieza.color == ROJO:
                    tablero.NEGRO_left -= 1
                else:
                    tablero.ROJO_left -= 1
        return tablero

    _, best_move = minimax(juego.tablero, 3, True)
    if best_move:
        juego.tablero = best_move
        juego.change_turn()

def programacion_dinamica(juego):
    # Crear una tabla para almacenar los resultados de los
subproblemas
    dp_table = {}

    def dp(tablero, depth, max_player):
        key = (str(tablero.tablero), depth, max_player)
        if key in dp_table:
            return dp_table[key]

        if depth == 0 or juego.check_winner():
            return tablero.evaluate(), tablero

        if max_player:
            max_eval = float('-inf')
            best_move = None
            for move in get_all_moves(tablero, NEGRO):
                evaluation = dp(move, depth - 1, False)[0]
                max_eval = max(max_eval, evaluation)
                if max_eval == evaluation:
                    best_move = move
            dp_table[key] = (max_eval, best_move)
            return max_eval, best_move
        else:
            min_eval = float('inf')
            best_move = None
```

```
            for move in get_all_moves(tablero, ROJO):
                evaluation = dp(move, depth - 1, True)[0]
                min_eval = min(min_eval, evaluation)
                if min_eval == evaluation:
                    best_move = move
            dp_table[key] = (min_eval, best_move)
            return min_eval, best_move

    def get_all_moves(tablero, color):
        moves = []
        for fil in range(FILAS):
            for col in range(COLUMNAS):
                pieza = tablero.tablero[fil][col]
                if pieza != 0 and pieza.color == color:
                    valid_moves =
tablero.obtener_movimientos_validos(pieza)
                    for move, skip in valid_moves.items():
                        temp_tablero = deepcopy(tablero)
                        temp_pieza = temp_tablero.tablero[fil][col]
                        new_tablero = simulate_move(temp_pieza, move,
temp_tablero, skip)
                        moves.append(new_tablero)
        return moves

    def simulate_move(pieza, move, tablero, skip):
        tablero.move(pieza, move[0], move[1])
        if skip:
            for f, c in skip:
                tablero.tablero[f][c] = 0
                if pieza.color == ROJO:
                    tablero.NEGRO_left -= 1
                else:
                    tablero.ROJO_left -= 1
        return tablero

    _, best_move = dp(juego.tablero, 3, True)
    if best_move:
        juego.tablero = best_move
        juego.change_turn()
```

```
def iniciar_juego():
    jugador1 = entrada_jugador1.get()
    algoritmo = combo_algoritmo.get()

    if not jugador1:
        print("Por favor, ingresa el nombre del Jugador 1.")
        return
    if algoritmo == "Seleccionar Algoritmo":
        print("Por favor, selecciona un algoritmo para el Jugador
2.")
        return

    print(f"Jugador 1: {jugador1}")
    print(f"Jugador 2 seleccionó el algoritmo: {algoritmo}")

    ventana.destroy()
    iniciar_juego_damas(algoritmo)

def iniciar_juego_damas(algoritmo):
    pygame.init()
    win = pygame.display.set_mode((ANCHO, ALTURA))
    pygame.display.set_caption("Damas")

    icono = pygame.image.load("UPC.JFIF")
    pygame.display.set_icon(icono)

    juego = Juego(win, algoritmo)
    running = True

    while running:
        juego.update()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            if event.type == pygame.MOUSEBUTTONDOWN:
                pos = pygame.mouse.get_pos()
                fil, col = get_fil_col_from_mouse(pos)
                juego.select(fil, col)
        if juego.turn == NEGRO:
            if algoritmo == "Backtracking":
```

```
                backtracking(juego)
            elif algoritmo == "Programación Dinámica":
                programacion_dinamica(juego)

    pygame.quit()

ventana = tk.Tk()
ventana.title("Hito 03")

icono_tk = Image.open("UPC.JFIF")
icono_tk = ImageTk.PhotoImage(icono_tk)
ventana.iconphoto(False, icono_tk)

ventana.geometry("250x250")

imagen_fondo = Image.open("U.jpeg")
imagen_fondo = imagen_fondo.resize((600, 400), Image.LANCZOS)
imagen_fondo_tk = ImageTk.PhotoImage(imagen_fondo)

label_fondo = tk.Label(ventana, image=imagen_fondo_tk)
label_fondo.place(x=0, y=0, relwidth=1, relheight=1)

titulo = tk.Label(ventana, text="HITO 03", font=("Anson", 18, "bold
italic"), bg="white", fg="black")
titulo.pack(pady=10)

label_jugador1 = tk.Label(ventana, text="Jugador 1:", bg="white",
fg="black")
label_jugador1.pack()

entrada_jugador1 = tk.Entry(ventana)
entrada_jugador1.pack(pady=10)

label_jugador2 = tk.Label(ventana, text="Algoritmo:", bg="white",
fg="black")
label_jugador2.pack()

combo_algoritmo = ttk.Combobox(ventana, values=["Backtracking",
"Programación Dinámica"])
combo_algoritmo.set("Seleccionar Algoritmo")
```

```
combo_algoritmo.pack(pady=10)

boton_iniciar = tk.Button(ventana, text="Iniciar Juego",
command=iniciar_juego)
boton_iniciar.pack(pady=10)

ventana.mainloop()
```

6. Conclusiones

El desarrollo de nuestro proyecto nosotros hemos utilizado backtracking, nodos y vértices con los cuales nos han permitido explorar cuáles serían los algoritmos más eficientes para poder resolver nuestros problemas que está relacionado con lo que sería juegos de tablero y estructuras de los grafos.Nuestro enfoque se destacó por la capacidad para generar soluciones óptimas a través de la búsqueda sistemática y controlada de todas las posibilidades.Nuestra implementación demostró ser flexible y adaptable a los diferentes escenarios dados , consolidando el valor del backtracking en los proyectos algorítmicos.

Recomendaciones: Para futuros trabajos de proyectos orientados a back tracking, nodos y vértices, recomendamos utilizar Python para el desarrollo del mismo.

7. Anexo

Github Rojas_Caletti_Fernandez:
https://github.com/Asalreon520/Complegidad-algoritmica-
Trello Rojas_Caletti_Fernandez:
https://trello.com/invite/b/66e5a2434d10def7c6a5033a/ATTIb8df9c5af47de1391b27cf76ec22f09e2A0FCBEC/trabajo-parcial

8. Bibliografías:

Gupta, P., Nagrath, V., & Preeti, N. (2021). Checkers-AI. En *Deep Learning in Gaming and Animations* (pp. 1-18). CRC Press.
https://www.taylorfrancis.com/chapters/edit/10.1201/9781003231530-1/checkers-ai-priyanshi-gupta-vividha-preeti-nagrath

Building a Checkers Gaming Agent Using Deep Q-Learning. (n.d.). https://blog.paperspace.com/building-a-checkers-gaming-agent-using-neural-networks-and-reinforcement-learning/

GeeksforGeeks. (s.f.). *Introducción a los algoritmos*. Recuperado el 22 de septiembre de 2024, de https://www.geeksforgeeks.org/introduction-to-algorithms/

Khan Academy. (s.f.). *Algoritmos*. Recuperado el 22 de septiembre de 2024, de https://www.khanacademy.org/computing/computer-science/algorithms

Codecademy. (s.f.). *Aprende algoritmos y estructuras de datos*. Recuperado el 22 de septiembre de 2024, de https://www.codecademy.com/learn/paths/algorithms