TP FINAL PROGRAMACION 2

SANTIAGO PRATO

Introducción

El objetivo del trabajo fue implementar el juego Othello utilizando C y Python, asegurando una correcta validación de jugadas, manejo del estado del juego y lectura de archivos, además de aplicar buenas prácticas de diseño, modularidad y testeo. Durante el desarrollo se buscó separar claramente las responsabilidades de cada parte del programa y mantener coherencia entre ambas implementaciones.

---

1. Proceso de resolución del trabajo

El desarrollo del trabajo se realizó de manera incremental. En primer lugar, se definió cómo debía interpretarse el archivo de entrada, que contiene los nombres de los jugadores, sus colores, el color que inicia la partida y la secuencia de jugadas. A partir de esta definición se implementaron funciones específicas para cada responsabilidad:

- `leerJugador` para leer nombres y colores.
- `empezo` para determinar qué color comienza.
- `procesarJugadasDesdeArchivo` para recorrer y aplicar las jugadas.

Separar la lectura del archivo de la lógica del juego permitió simplificar el razonamiento, evitar errores en el manejo del tablero y facilitar el testeo de cada componente de forma independiente.

Una vez resuelta la lectura de datos, se avanzó sobre la representación del tablero y la implementación de las reglas del juego, priorizando la reutilización de lógica tanto en C como en Python.

---

2. Representación del tablero

En C, el tablero se representó mediante una matriz de estructuras `Celda`, mientras que en Python se utilizó una matriz de caracteres. A pesar de la diferencia de lenguajes, se mantuvo exactamente la misma convención en ambos casos:

- `'X'` para casillas vacías
- `'B'` para fichas blancas
- `'N'` para fichas negras

Esta decisión permitió que la lógica del juego fuera prácticamente idéntica en ambos lenguajes, reduciendo la posibilidad de inconsistencias y facilitando la verificación de resultados entre C y Python.

---

3. Validación y aplicación de jugadas

La validación de jugadas fue uno de los puntos centrales del trabajo. Para resolverla, se utilizó un enfoque basado en recorrer las ocho direcciones posibles desde una casilla (horizontal, vertical y diagonales). En cada dirección se buscó la secuencia característica del juego: fichas del rival seguidas por una ficha propia.

Este enfoque fue clave porque:

- Evitó duplicación de código.
- Permitió una implementación clara y compacta.
- Facilitó la detección y corrección de errores.
- Mantuvo la lógica coherente entre C y Python.

Una vez validada la jugada, la función `aplicarJugada` reutiliza las mismas direcciones para voltear las fichas correspondientes, asegurando consistencia entre validación y aplicación.

---

4. Manejo del estado del juego en C

La función `estadoJuego` se encargó de determinar la situación final del juego una vez procesadas todas las jugadas del archivo. Para ello, sigue un flujo claro y ordenado:

1. Mostrar el tablero final.
2. Verificar si el jugador siguiente tiene jugadas disponibles.
3. Detectar si corresponde saltear turno.
4. Determinar si la partida terminó.
5. Contar las fichas de cada color y definir el ganador.

Este orden evita ambigüedades y permite entender fácilmente cómo evoluciona el estado del juego en cada situación posible.

---

5. Interacción y niveles de juego en Python

La versión en Python se utilizó para implementar la parte interactiva del juego. En esta versión, el jugador humano ingresa las jugadas manualmente y la computadora puede jugar en dos niveles de dificultad:

- **Nivel 0:** la computadora elige una jugada válida al azar.
- **Nivel 1:** la computadora analiza todas las jugadas posibles y selecciona la que maximiza la cantidad de fichas volteadas.

Para implementar estos niveles se reutilizaron funciones ya probadas como `jugadaValida`, `jugadasPosibles` y `contarFichasVueltas`, lo que aseguró coherencia con la versión en C y redujo errores.

Decisiones más importantes

Las decisiones que más influyeron en el diseño final del trabajo fueron:

- **Uso de vectores de direcciones** para validar y aplicar jugadas.
  Esto simplificó notablemente el código y evitó casos especiales innecesarios.
- **Separación clara de responsabilidades** entre lectura de archivos, lógica del juego y presentación.
  Esta decisión facilitó el mantenimiento del código y el testeo de cada módulo por separado.
- **Uso de memoria dinámica en C para los nombres de jugadores**, reservando únicamente la cantidad necesaria de bytes.
  Esto evitó desperdicio de memoria y posibles errores.
- **Creación de tests desde las primeras etapas del desarrollo**, tanto en C como en Python.
  Esto permitió detectar errores tempranamente y evitar que se propagaran a partes más complejas del programa.

Conclusión

El trabajo permitió aplicar conceptos fundamentales de Programación 2 como modularización, manejo de memoria, validación de datos y diseño de funciones claras. Mantener una lógica común entre C y Python fue clave para asegurar consistencia y facilitar el desarrollo. Las decisiones tomadas durante el proceso contribuyeron a obtener un programa robusto, claro y fácil de probar.