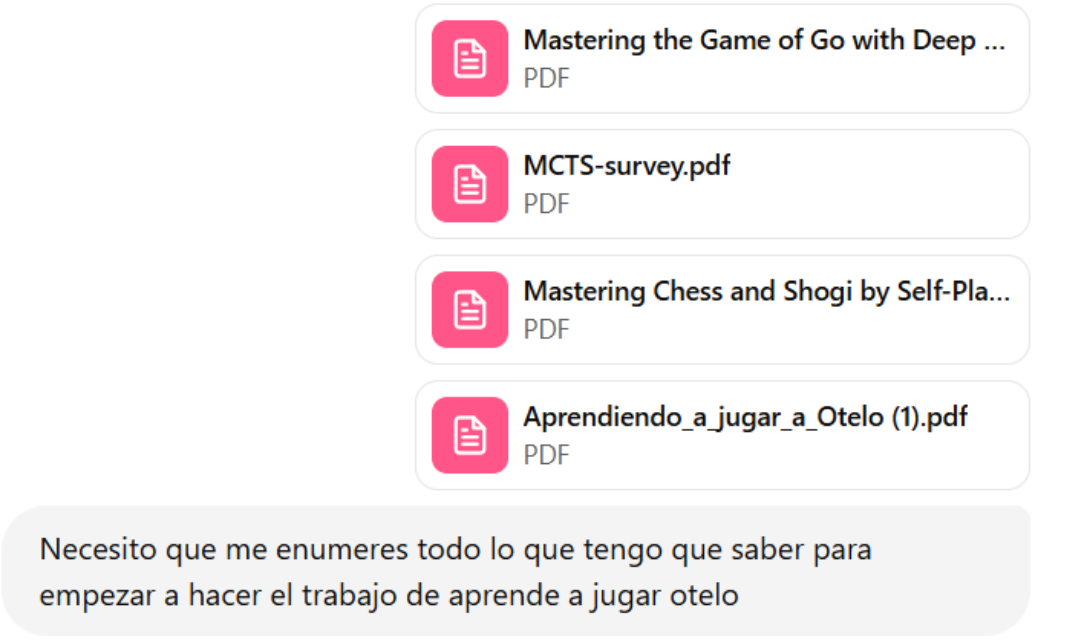
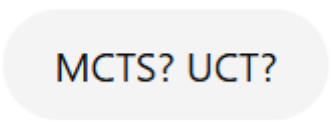
Uso de IA generativa

Iván Fernández Limárquez, solo se ha usado chatgpt

Para que la IA diera mejores respuestas y para tratar de entender inicialmente mejor el trabajo se pidió a esta que explicara los conceptos necesarios para empezar a abordar el proyecto



Se pidieron además explicaciones de los conceptos en la asignatura en conjunto con lecturas de los materiales teóricos para poder entender mejor como realizarlo:



Se hicieron preguntas alrededor de estos conceptos:

> "Desde la raíz (estado actual), se desciende el árbol eligiendo en cada nodo el hijo con la mejor balance entre explotación (jugadas con buenos resultados) y exploración (jugadas poco visitadas)." Entonces para ver si entiendo el concepto la idea es ir seleccionado jugadas posibles (a veces la mejor que sabemos con nuestras iteraciones, a veces una nueva) y jugar partidas de esa jugada, muchas, muchisimas, para determinar cual es la mas probable que nos de una victoria

Una vez proporcionadas las reglas y habiendo recibido y explicado varios de los contenidos teóricos basándose en el PDF se pidió una base inicial sobre la red neuronal para empezar a trabajar:

> Bueno venga po hazme una red neuronal con UCT y MTCS aplicado a esto:
>
> ```
> # Imports
> import pygame as pg
> from utiles.tablero import *
> from utiles.fichas import *
>
> # Setup de pygame
> pg.init()
>
> # Constantes
> LONG_TABLERO = 640
> LONG_CASILLA = 80
> COLOR_FONDO = (0, 128, 0)
> COLOR_LINEAS = (0, 0, 0)
> ```

(...)

Y explicaciones sobre ese código:

Vale pues necesito un desglose bastante detallado del codigo de la red neuronal, que significan los parametros dentro de la funcion que hacen y tambien ¿Has usado python orientado a objetos? necesito una breve explicacion de eso que lo hice en java pero enpython nunca:

```
import pygame as pg
import torch
import torch.nn as nn
import torch.nn.functional as F
import math
import random
from utiles.tablero import dibujar_tablero, obtener_fichas_a_voltear

# ================== Neural Network Definition
==================
class OthelloNet(nn.Module):
    def __init__(self):
        super(OthelloNet, self).__init__()
```

(...)

Pero, la capa de entrada de 64 no era porque le pasabas el tablero de 8*8 de otelo?

La confusión es comprensible: esos **64** en la primera capa *no* vienen del tamaño del tablero (8 × 8 = 64 casillas). Son dos números distintos:

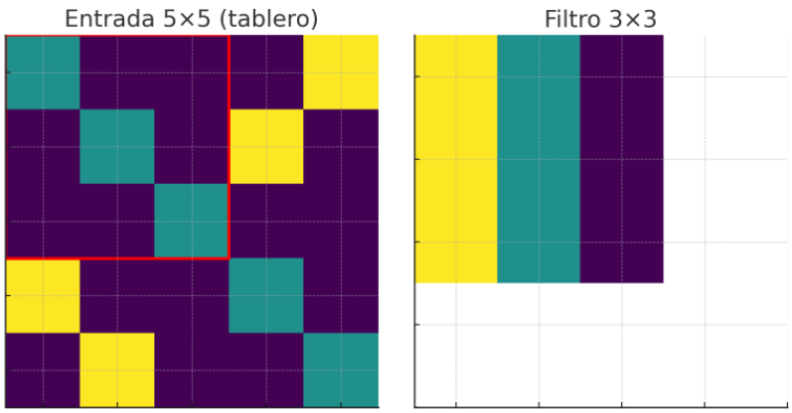
Dónde aparece el 64	Qué significa
`nn.Conv2d(2, 64, 3, padding=1)` → **argumento "64"**	**Número de filtros** (o *feature maps*) que producirá la primera convolución. Cada filtro barre todo el tablero y aprende a detectar un patrón diferente (líneas, esquinas, cruces...). Tras la operación obtenemos un tensor 64 × 8 × 8.

Pero para que vas a tener 2 canales binarios si en el tablero se asignan asi:

2. Guardar los estados: Cada vez que un jugador realice una jugada, se debe guardar
el estado resultante del tablero. Este estado puede representarse como una matriz
de números:
0:casilla vacía.
1:casilla ocupada por una ficha blanca.
2:casilla ocupada por una ficha negra

"Cada capa aplica docenas de filtros 3×3 con padding=1, de modo que la salida sigue siendo 8×8. Después de cada convolución va un ReLU (no-linealidad)." Que es una convolucion, filtro 3x3? padding?

No podras hacer um dibujito para que lo entienda no?



Arriba tienes un pequeño **esquema visual**:

Y Vale voy a asumir, amarillo blancas, azul negras, morado no hay nada, ahora para que es el filtro?

Ahh entiendo, detecta el patron de blancas arriba negras abajo o viceversa, se aplican varios filtros (varias capas ocultas) para reconocer mas patrones, ¿no?

Claro pero en este flujo:

3 · Flujo completo (simplificado)
text
Copiar
Editar
tablero 0/1/2 → state_to_tensor → Tensor 1×2×8×8
Tensor 1×2×8×8 → conv1 (64 filtros) → 1×64×8×8
1×64×8×8 → conv2 (128 filt.) → 1×128×8×8
1×128×8×8 → conv3 (128 filt.) → 1×128×8×8
1×128×8×8 → flatten → 1×8192
1×8192 → fc (256) → 1×256
1×256 → head_p (64) → softmax → π(a)
→ head_v (1) → tanh → v(s)

No se crea un tableró y "nada mas" 64 filtros no? Se crean 2 de 64, uno para blancas otro para negras no?

No: **no hay "64 para blancas" + "64 para negras"**.
Hay **solo 64 filtros en total** y **cada filtro "mira" simultáneamente los 2 canales** de entrada.

Como podemos ver, el código base proporcionado funcionó principalmente como guía para entender como se iba a hacer esa red neuronal, porque se iba a hacer así y que significaba cada concepto o capa de esta misma red.

No obstante este venía implementado en lenguaje de objetos y con librerías como torch no vistas en la asignatura, por lo que se decidió traspasar el código de la red neuronal a Keras con ayuda de la inteligencia artificial se preguntó como había implementado ciertos aspectos.

Que es ese (x) al final?
x = Conv2D(128, (3, 3), padding='same', activation='relu', name='conv2')(x)

"Poner (x) a continuación equivale a "llamar a esa capa con x como entrada". Es decir, toma el tensor que estaba en la variable x y pasa sus valores por la convolución, devolviendo un nuevo tensor de salida." Que

Porque al entrenar la red hay que establecer N? N = X_flat.shape[0]
X_raw = X_flat.reshape((N, 8, 8))

como funciona el dropout, cada cuanto desactivva el 50% de las neuronas, todas las epocas?

Tras esto, se preguntó por la lectura de ficheros en una red neuronal y como interpretar esos datos para poder empezar a entrenarla

Vale tenemos esta red neuronal a entrenar ok, no hace falta que respondas nada solo que lo tengas en cuenta

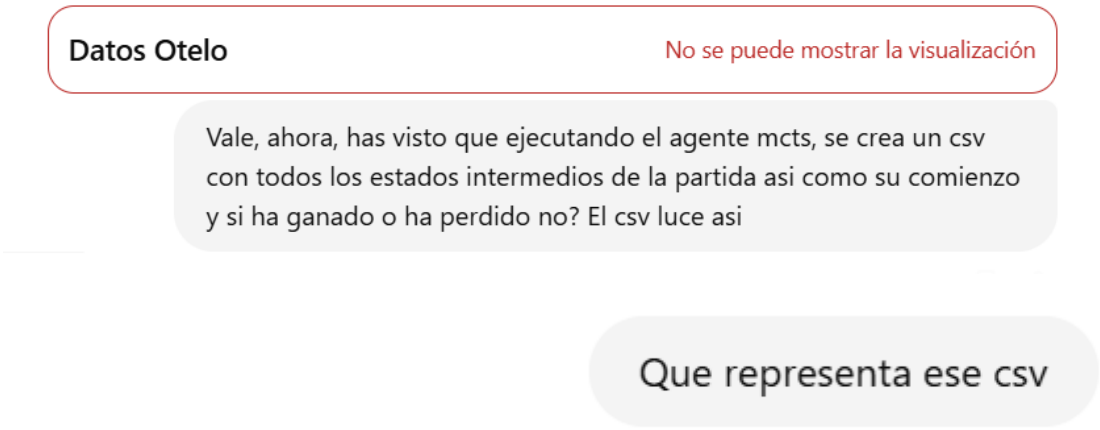
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
from keras._tf_keras.keras.utils import set_random_seed
import pandas as pd
import numpy as np
from keras import Sequential, Input, Model
from keras._tf_keras.keras.layers import Dense
from keras._tf_keras.keras.layers import Normalization
from keras._tf_keras.keras.layers import Conv2D
from keras._tf_keras.keras.layers import Activation
from keras._tf_keras.keras.optimizers import SGD
from sklearn.preprocessing import label_binarize
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.utils.class_weight import compute_class_weight
```

(...)

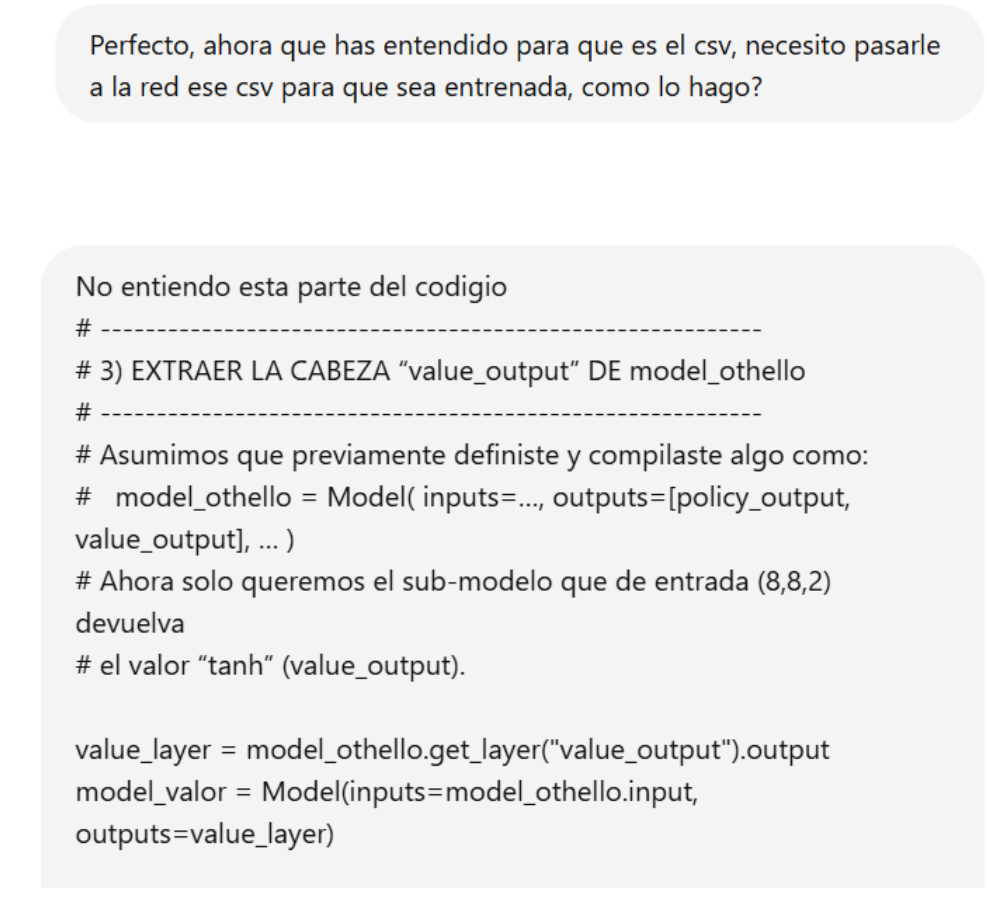
a su vez tenemos este agente montecarlo:

```
import sys
import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))

# Imports
import csv
import copy
import random
```

(...)



Pregunta para ver si la IA había entendido que es el CSV



(...)

Además se preguntó como integrar el código de la red neuronal entrenada en el propio algoritmo de Montecarlo:

Vale, necesito que una vez entrenada la red esta sustituya la default policy en montecarlo, como integro ambas cosas?

```
import sys
import os

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))

# Imports
from utiles.fichas import *
import copy
from math import sqrt, log
import random
```

(...)

Que significa el 1 en estado a tensor?

```
# ----------------------------------------------------------------
# 1) estado_a_tensor: convierte un tablero (8×8) a (1,8,8,2)
# ----------------------------------------------------------------
def estado_a_tensor(estado):
    """
    Convierte un tablero de Othello (matriz 8×8 de ints {0,1,2})
    en un numpy array de shape (1,8,8,2) con dos canales binarios:

      canal 0 = 1.0 si había ficha negra (==2)
      canal 1 = 1.0 si había ficha blanca (==1)

    Devolvemos un array dtype float32 listo para model.predict().
    """
    # "estado" puede ser lista de listas o np.array de shape (8,8)
    tablero = np.array(estado, dtype=np.int32)      # garantiza (8,8)

    tensor = np.zeros((1, 8, 8, 2), dtype=np.float32)
    tensor[0, :, :, 0] = (tablero == 2).astype(np.float32)
    tensor[0, :, :, 1] = (tablero == 1).astype(np.float32)
    return tensor
```

Y con todo entendido se comenzaron a hacer los comentarios, la IA y búsquedas en internet ayudaron a contrastar la veracidad o entendimiento de estas

explicaciones:

La explicacion esta bien?

```
def estado_a_tensor(estado):
    """
    Puesto que en este caso solo pasaremos un tablero, en lugar de un conjunto de datos con varios tableros,
    convertimos el tablero 8x8 directamente a un tensor de forma (1, 8, 8, 2).

    Relizamos una oprecion muy similar a "convertir_a_canales" de la red neuronal. Pero con un tablero nada mas

    Para ello, creamos el tensor (o tablero de salida) de ceros y de la misma forma, recorremos el tablero de entrada 8x8, y sustiuimos
    los valores transformados a canal en el tensor (o tablero de salida) de ceros.
    """

    tablero = np.array(estado, dtype=np.int32)
```

(...)

Finalmente para la documentación se hicieron preguntas acerca de como usar LaTeX y MiKTeX, además de para reescribir conceptos en un tono mas formal:

how to see latex dynamically in vscode

Reescribelo mas formal en forma paper cientifico

\item \textbf{Capa de aplanado}: Capa encargada de transformar los tableros logrados en la ultima capa convolucional a un solo vector con todos los pesos asociados a estos tableros
\item \textbf{Capa de desactivación o dropout}: Encargada de desactivar el 50\% de las neuronas de la siguiente capa, en este caso de la capa densa. Esto se hace para limitar la velocidad por la cual los pesos cambian, al estar apagadas estas no dan valores y por ende, no se tienen en cuenta para el ajuste de los pesos de la red
\item \textbf{Capa densa}: Toma los 8192 valores de la tercera capa convolucional y los transforma en 256 valores, estos representan la "cantidad" de patrones que hay en la posicion.