Importamos las librerias

```python
import os
import gymnasium as gym
from gymnasium import spaces
import numpy as np
from stable_baselines3 import DQN
from stable_baselines3.common.env_checker import check_env
from stable_baselines3.common.callbacks import EvalCallback, BaseCallback
from stable_baselines3.common.monitor import Monitor
import matplotlib.pyplot as plt
```

Esta clase es un callback personalizado que guarda imágenes del progreso del entrenamiento cada eval_freq pasos. BaseCallback es una clase base de stable_baselines3 para callbacks personalizados

```python
class SaveImageCallback(BaseCallback):
    """
    Callback para guardar una imagen PNG del progreso del entrenamiento.
    """
    def __init__(self, log_dir, eval_freq=1000):
        super().__init__()
        self.log_dir = log_dir
        self.eval_freq = eval_freq
        self.rewards = []

    def _on_step(self) -> bool:
        if self.n_calls % self.eval_freq == 0:
            mean_reward = np.mean(self.rewards)
            self.save_png(mean_reward)
        return True

    def _on_rollout_end(self) -> None:
        rewards = self.locals['rewards']
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS COMMENTS

Esta clase define el entorno del rompecabezas. El agente se mueve en una cuadrícula de 4x5 con las acciones: arriba, abajo, izquierda y derecha. La observación es la matriz 4x5 que representa el estado actual del rompecabezas.

```python
        plt.close()

class PuzzleEnv(gym.Env):
    def __init__(self):
        super(PuzzleEnv, self).__init__()
        self.action_space = spaces.Discrete(4)  # 4 acciones: arriba, abajo, izquierda, derecha
        self.observation_space = spaces.Box(low=0, high=1, shape=(4, 5), dtype=float)  # espacio de observación 2D
        self.state = np.zeros((4, 5), dtype=float)
        self.agent_pos = [0, 0]  # posición inicial del agente
        self.goal_pos = [3, 4]   # posición objetivo

    def reset(self, seed=None, options=None):
        super().reset(seed=seed)
        self.state = np.zeros((4, 5), dtype=float)
        self.agent_pos = [0, 0]
        self.state[self.agent_pos[0], self.agent_pos[1]] = 0.5
        self.state[self.goal_pos[0], self.goal_pos[1]] = 1
        return self.state, {}

    def step(self, action):
        x, y = self.agent_pos
        if action == 0 and x > 0:  # mover arriba
            x -= 1
```

Exploracion :

1. Se crea el directorio para guardar los registros y modelos.
2. Se envuelve el entorno en Monitor para registrar las estadísticas de rendimiento.
3. Se verifica el entorno utilizando check_env.
4. Se crea el modelo DQN utilizando la política MlpPolicy.
5. Se definen los callbacks para evaluar el modelo y guardar imágenes del progreso.
6. Se entrena el modelo con model.learn.
7. Se guarda el modelo entrenado.

Explotacion

- Se carga el modelo entrenado.
- Se evalúa el modelo prediciendo acciones y ejecutándolas en el entorno, mostrando el estado en cada paso.

Sobre las tecnicas utilizadas

DQN utiliza una red neuronal para aproximar la función Q, que estima la utilidad de realizar una acción en un estado dado.

código entrena un agente usando DQN para resolver un rompecabezas en una cuadrícula 4x5. El agente aprende a moverse de su posición inicial a la posición objetivo a través de pasos de entrenamiento, utilizando una combinación de exploración y explotación