



Búsqueda adversarial en juegos

Aplicación de algoritmos minimax y poda alfa-beta en Agentes IA

Carrera:	Ingeniería en Ciberseguridad
Sede:	Online
Curso:	Taller de IA Aplicada
Profesor:	Pablo Schwarzenberg Riveros
Estudiantes:	Gabriel Araya Rocha Hedy Herrada Hermosilla Macarena Riquelme Cerda

Tabla de contenidos

Introducción.....	4
Instrucciones generales.....	5
Juego Connect4 con IA.....	7
Características principales.....	7
Instalación y ejecución.....	8
Requisitos previos.....	8
Instalación de GIT en Windows.....	8
Instalación de Python3 en Windows.....	9
Instalación de Pip.....	10
Instalación de virtualenv.....	10
Estructura de archivos del proyecto.....	12
Instalación del juego en Windows.....	13
Instalación del juego en Linux/macOS.....	16
Ejecución del juego en Windows.....	18
Ejecución del juego en Linux.....	20
Cómo jugar.....	22
Inicio del juego.....	22
Durante el juego.....	23
Condiciones de finalización.....	24
Agentes IA para Connect4.....	26
Técnicas de IA utilizadas.....	26
Algoritmo Minimax con poda Alfa-Beta.....	26
Función de Evaluación Heurística.....	27
Sistema de Dificultad Adaptativa.....	27
Reglas de Connect4.....	28
Objetivo del juego.....	28
Preparación del juego.....	28
Reglas del juego.....	28
Fin del juego.....	28
Acciones no permitidas.....	28
Estrategia.....	29
Diseño del agente.....	29
Representación para el estado del juego.....	30
Alternativas consideradas pero descartadas.....	30
Estrategia de generación de jugadas.....	31
Función de evaluación.....	32
Estrategia de control de dificultad del juego.....	34
Estrategia de implementación de sugerencia de una jugadas al jugador humano.....	35
Profundidad Reducida.....	35
Perspectiva del Jugador.....	35
Seguimiento y Estadísticas.....	36

Visualización de la Sugerencia.....	36
Estrategia de implementación de ajuste del nivel de dificultad según la habilidad del jugador....	37
Manejo de errores y excepciones.....	38
Análisis de resultados.....	39
Estructura de Datos y Metodología.....	39
Desempeño de la IA.....	40
Análisis de Ayuda (Sugerir Jugada).....	40
Gráficos.....	40
Análisis por dificultad.....	42
Conclusiones.....	43
Bibliografía.....	44
Anexo.....	45
Archivo main.py.....	45
Archivo connect_four.py.....	54

Introducción

El juego Connect4 ha emergido como un recurso valioso en el campo de la inteligencia artificial para el desarrollo y prueba de agentes autónomos que buscan optimizar sus estrategias en contextos de competencia. Este juego, conocido por sus reglas sencillas pero gran potencial estratégico, permite a los agentes de IA aprender a anticipar las jugadas del oponente, analizar patrones y tomar decisiones que maximizan sus posibilidades de ganar.

La creación de un agente de IA para Connect4 involucra la aplicación de técnicas avanzadas que abarcan desde algoritmos de búsqueda en árboles hasta métodos de aprendizaje automático. Entre los enfoques más utilizados están el algoritmo minimax y los árboles de búsqueda de Monte Carlo (MCTS), que permiten a los agentes explorar distintas combinaciones de jugadas y evaluar las probabilidades de éxito en cada una. Además, en los últimos años, el aprendizaje por refuerzo ha cobrado protagonismo en el entrenamiento de estos agentes, permitiéndoles mejorar con cada partida a través de la experiencia acumulada.

Estos agentes no solo representan un avance en el campo del juego estratégico, sino que también sirven como modelos para abordar problemas complejos de toma de decisiones en el mundo real. Su desarrollo abre nuevas puertas para aplicaciones de inteligencia artificial en áreas como la planificación autónoma, la optimización de recursos y la logística.

El siguiente taller, tiene como objetivo el resolver un juego mediante implementación de búsqueda adversarial usando algoritmos Minimax y Poda Alfa-Beta. Para ello, en el taller se guiará a los participantes en la implementación de un agente de IA que utilice algoritmos de búsqueda adversarial como Minimax y su optimización con Poda Alfa-Beta. Este proceso permitirá que el agente desarrolle la capacidad de prever y reaccionar ante las jugadas del oponente de manera estratégica, evaluando los posibles movimientos en el tablero y eligiendo la opción que maximice sus posibilidades de éxito.

Instrucciones generales

Para realizar esta actividad, se deben formar equipos de trabajo de 2 a 3 participantes e implementar la búsqueda adversarial en el juego Connect4.

Para ello, debes realizar las actividades que se indican a continuación:

I. Enunciado

1. Revisar la rúbrica de la actividad para que sepas, de antemano, qué y cómo evaluarás tu desempeño.
2. Revisar el tutorial del Laboratorio 2: Resolviendo un juego mediante búsqueda adversarial, disponible en la plataforma y, luego:
3. En este proyecto debes implementar un agente para el juego del Connect Four (https://en.wikipedia.org/wiki/Connect_Four). Este juego se realiza sobre un tablero de 6 filas y 7 columnas con fichas de dos colores (blanco y negro, por ejemplo). Al igual como en el juego del gato el tablero comienza vacío (Figura 1) y los jugadores deben alternadamente soltar fichas en alguna de las 7 columnas para formar filas, columnas o diagonales con cuatro de sus fichas (Figura 2).

	A	B	C	D	E	F	G
1							
2							
3							
4							
5							
6							

Figura 1. Posición de inicio de Connect Four

	A	B	C	D	E	F	G
1							
2							
3		■		□			
4		■		□			
5		■		□			
6	□	■	□	■			

Figura 2. Ejemplo de victoria del jugador negro

Para jugar una versión en línea de este juego, puedes consultar el siguiente link: <https://www.mathsisfun.com/games/connect4.html>.

¡Ten presente! Tu juego debe:

1. Permitir escoger el tamaño del tablero (6x7 o 5x4).
2. Permitir escoger el nivel de dificultad, entre al menos tres niveles.
3. Registrar e imprimir el número de nodos explorados y el tiempo utilizado durante la selección de la jugada que realizará la computadora.
4. Poseer una interfaz que sea de fácil uso, que permita ver el estado del juego en forma clara.
5. Seleccionar una jugada sin inducir a errores. Debes agregar a tu juego una característica adicional, que puedes escoger entre:
 - 1) Sugerir una jugada al jugador humano si es que lo pide.
 - 2) Ajustar el nivel de dificultad de acuerdo con la habilidad del jugador, recordando partidas anteriores.

II. Consideraciones en la revisión

Debes entregar un informe que explique:

1. La introducción al contexto del problema, explicando técnicas que se han usado para abordar el desarrollo de agentes para este juego (incluir referencias a libros o papers).
2. El diseño general de tu agente, las técnicas aplicadas y la justificación de su elección.
3. La representación escogida para el estado del juego, justificando sus ventajas por sobre otras opciones.
4. La estrategia de generación de jugadas.
5. La función de utilidad o evaluación, según sea el caso, justificando sus ventajas por sobre otras opciones.
6. El diseño de la estrategia utilizada para controlar la dificultad del juego y su justificación.
7. El diseño de la estrategia utilizada para implementar la característica especial de tu juego.
8. Conclusiones respecto del desempeño del agente, con casos de ejemplo, tablas con resultados numéricos y gráficos.
 - Puedes implementar este trabajo en lenguaje C/C++ o lenguaje Python.
 - Debes comentar cada una de las funciones, estructuras o clases que definas, indicando una descripción de la labor que lleva a cabo cada una.
 - Puedes trabajar con el IDE o lenguaje que más te acomode. No obstante, tu programa debiera poder ser ejecutado sin problemas en Linux o Windows.
 - El sistema debe ser robusto. Se penalizarán los errores no manejados, de cualquier tipo.

Juego Connect4 con IA

A continuación presentamos nuestra implementación del juego clásico Connect4 implementado en Python con una IA adaptativa que utiliza el algoritmo Minimax con poda Alfa-Beta. El juego incluye una interfaz gráfica construida con Pygame y almacena estadísticas del juego en una base de datos SQLite.

Características principales

- **Interfaz gráfica intuitiva:** Diseñada con Pygame para una experiencia de usuario fluida.
- **Múltiples niveles de dificultad:**
 - Fácil
 - Medio
 - Difícil
- **Tamaños de tablero personalizables:**
 - Normal (6x7)
 - Pequeño (5x4)
- **Estadísticas detalladas de:**
 - Tiempo de juego
 - Movimientos realizados
 - Nodos explorados por la IA
 - Sugerencias utilizadas
- **IA adaptativa:** Ajusta su dificultad basándose en el rendimiento del jugador.
- **Sistema de sugerencias:** Ayuda para jugadores que necesitan orientación.
- **Persistencia de datos:** Almacenamiento de partidas y estadísticas en SQLite.

Instalación y ejecución

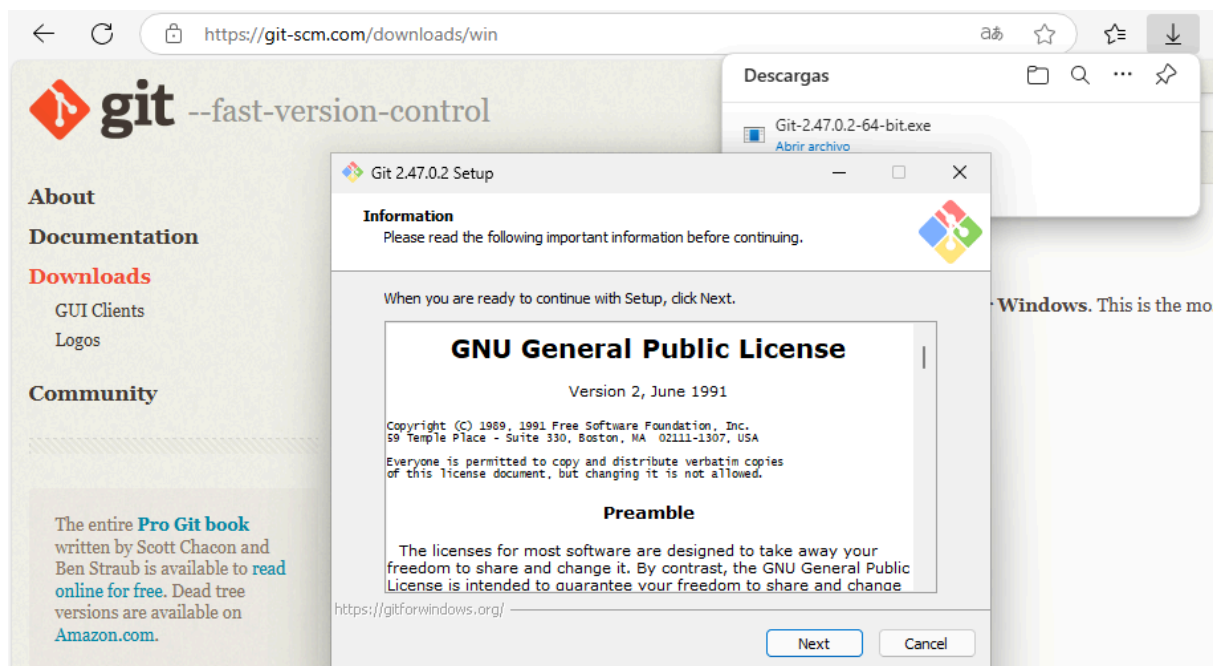
Requisitos previos

Esta es una lista de los paquetes que deben estar instalados previamente:

- **GIT:** Sistema de control de versiones
- **Python 3:** Lenguaje de programación
- **Pip:** Gestor de instalación de paquetes PIP
- **Virtualenv:** Creador de entornos virtuales para Python

Instalación de GIT en Windows

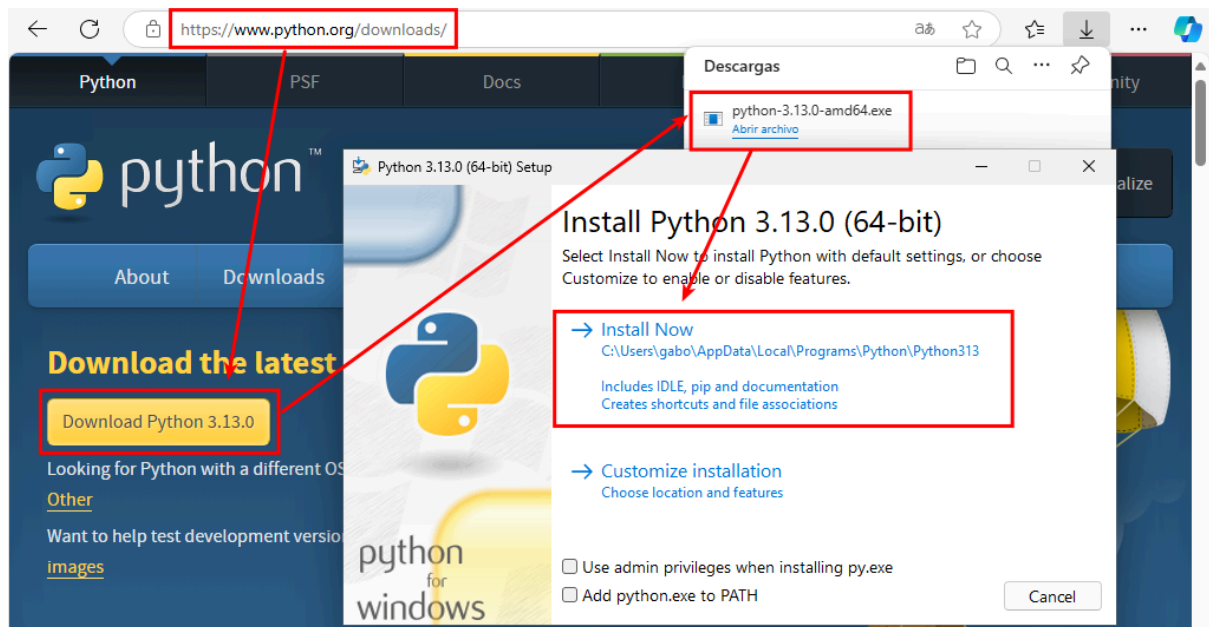
- Descargar el instalador desde en el sitio web: <https://git-scm.com/downloads/win>
- Seguir las instrucciones del Instalador.



Instalación de Python3 en Windows

Descargar e instalar Python 3.10 (o una versión superior) para Windows desde:

<https://www.python.org/downloads/>



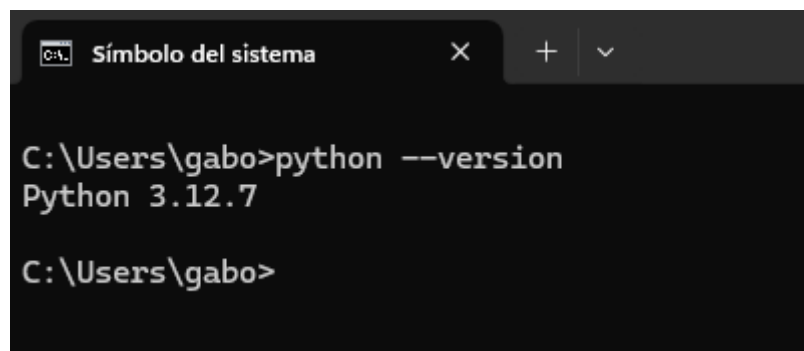
Agregar Python a las variables de entorno de nuestro sistema si es que no se agregaron durante la instalación para poder ejecutarlo desde la terminal `cmd` o `powershell`.

```
C:\Python34
```

```
C:\Python34\Scripts
```

Ejecutamos Python con el flag `--version` para verificar que esté instalado correctamente y revisar la versión

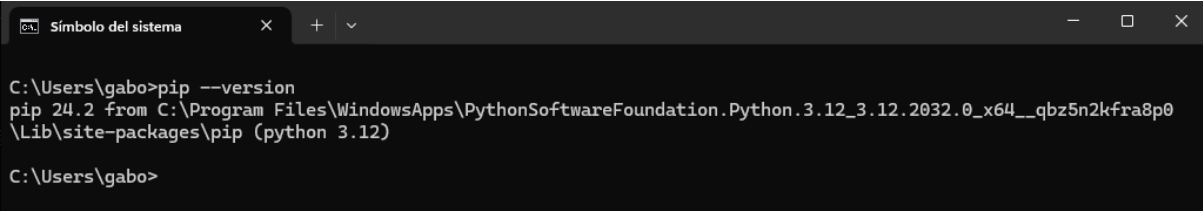
```
pip --version
```



Instalación de Pip

Ejecutar Pip para verificar que esté instalado correctamente y también conocer la versión

```
pip --version
```

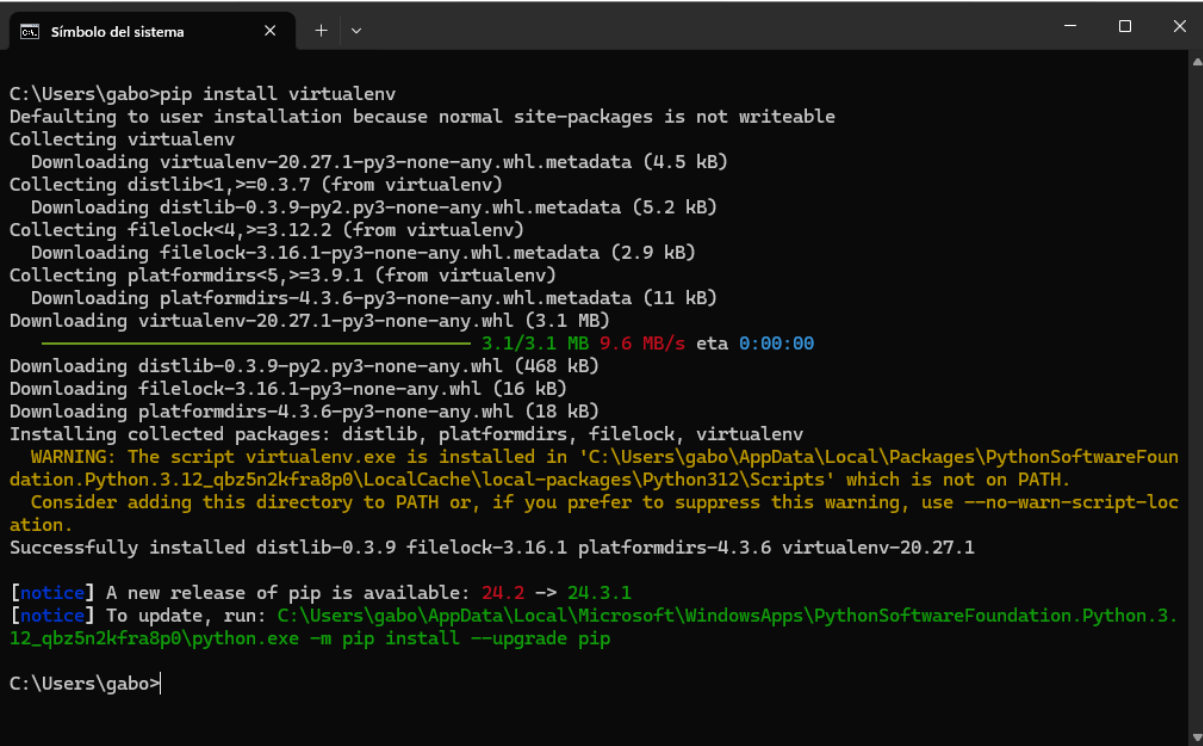


```
Símbolo del sistema
C:\Users\gabo>pip --version
pip 24.2 from C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.12_3.12.2032.0_x64__qbz5n2kfra8p0\
Lib\site-packages\pip (python 3.12)
C:\Users\gabo>
```

Instalación de virtualenv

Instalar Virtualenv con el comando Pip.

```
pip install virtualenv
```



```
Símbolo del sistema
C:\Users\gabo>pip install virtualenv
Defaulting to user installation because normal site-packages is not writeable
Collecting virtualenv
  Downloading virtualenv-20.27.1-py3-none-any.whl.metadata (4.5 kB)
Collecting distlib<1,>=0.3.7 (from virtualenv)
  Downloading distlib-0.3.9-py2.py3-none-any.whl.metadata (5.2 kB)
Collecting filelock<4,>=3.12.2 (from virtualenv)
  Downloading filelock-3.16.1-py3-none-any.whl.metadata (2.9 kB)
Collecting platformdirs<5,>=3.9.1 (from virtualenv)
  Downloading platformdirs-4.3.6-py3-none-any.whl.metadata (11 kB)
Downloading virtualenv-20.27.1-py3-none-any.whl (3.1 MB)
  3.1/3.1 MB 9.6 MB/s eta 0:00:00
Downloading distlib-0.3.9-py2.py3-none-any.whl (468 kB)
Downloading filelock-3.16.1-py3-none-any.whl (16 kB)
Downloading platformdirs-4.3.6-py3-none-any.whl (18 kB)
Installing collected packages: distlib, platformdirs, filelock, virtualenv
WARNING: The script virtualenv.exe is installed in 'C:\Users\gabo\AppData\Local\Packages\PythonSoftwareFoun
dation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\Scripts' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-loc
ation.
Successfully installed distlib-0.3.9 filelock-3.16.1 platformdirs-4.3.6 virtualenv-20.27.1

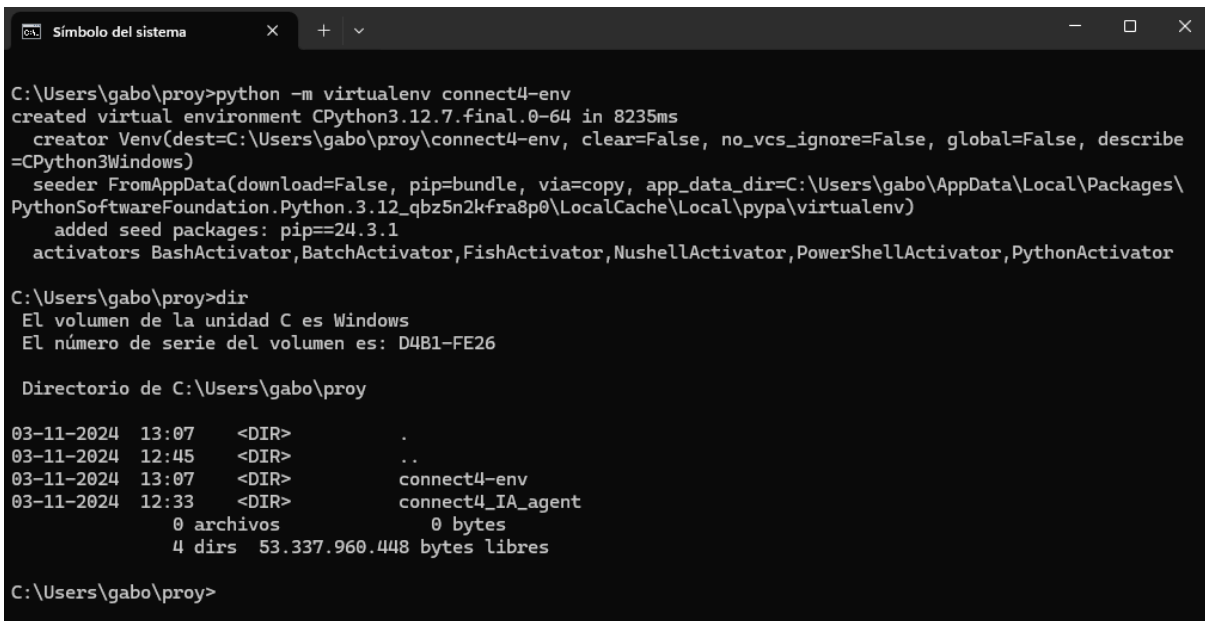
[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: C:\Users\gabo\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.
12_qbz5n2kfra8p0\python.exe -m pip install --upgrade pip
C:\Users\gabo>
```

Verificar la versión de Virtualenv con el flag `--version`

```
python -m virtualenv --version
```

Crear un entorno virtual con Python llamado `connect4-env`

```
python -m virtualenv connect4-env
```



```
Símbolo del sistema
C:\Users\gabo\proy>python -m virtualenv connect4-env
created virtual environment CPython3.12.7.final.0-64 in 8235ms
  creator Venv(dest=C:\Users\gabo\proy\connect4-env, clear=False, no_vcs_ignore=False, global=False, describe
=CPython3Windows)
  seeder FromAppData(download=False, pip=bundle, via=copy, app_data_dir=C:\Users\gabo\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\Local\pypa\virtualenv)
  added seed packages: pip==24.3.1
  activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator

C:\Users\gabo\proy>dir
El volumen de la unidad C es Windows
El número de serie del volumen es: D4B1-FE26

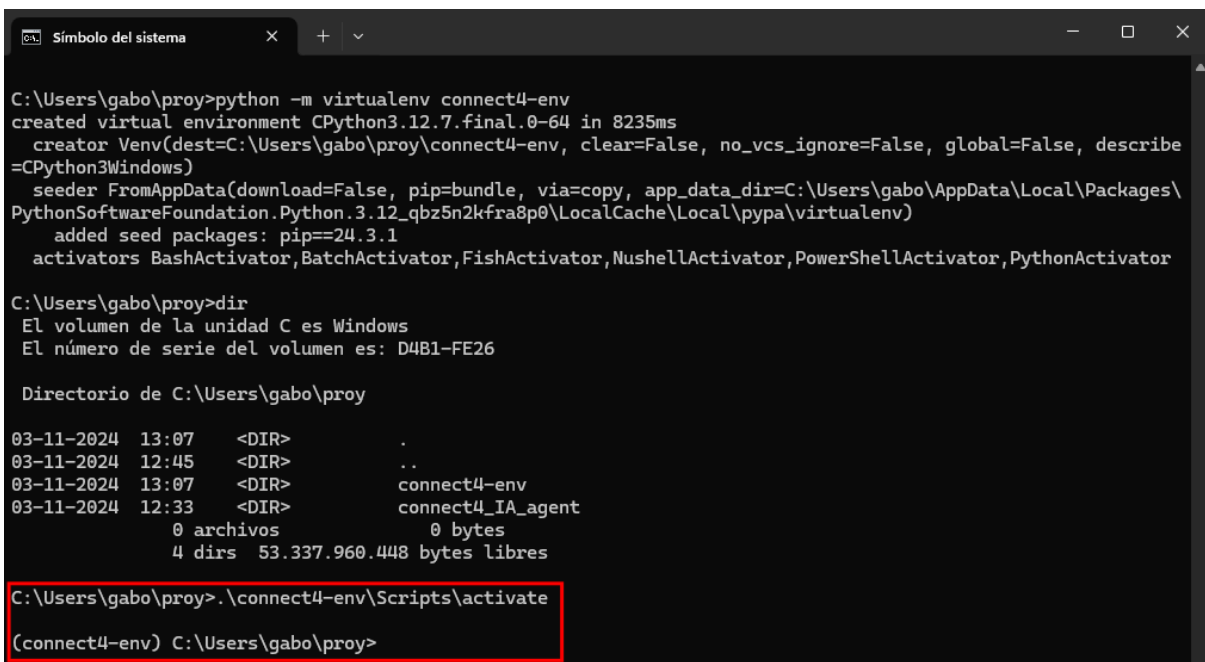
Directorio de C:\Users\gabo\proy

03-11-2024  13:07    <DIR>          .
03-11-2024  12:45    <DIR>          ..
03-11-2024  13:07    <DIR>          connect4-env
03-11-2024  12:33    <DIR>          connect4_IA_agent
                   0 archivos                0 bytes
                   4 dirs  53.337.960.448 bytes libres

C:\Users\gabo\proy>
```

Para iniciar el entorno virtual en *Windows* se debe ejecutar el script `activate`:

```
.\connect4-env\Scripts\activate
```



```
Símbolo del sistema
C:\Users\gabo\proy>python -m virtualenv connect4-env
created virtual environment CPython3.12.7.final.0-64 in 8235ms
  creator Venv(dest=C:\Users\gabo\proy\connect4-env, clear=False, no_vcs_ignore=False, global=False, describe
=CPython3Windows)
  seeder FromAppData(download=False, pip=bundle, via=copy, app_data_dir=C:\Users\gabo\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\Local\pypa\virtualenv)
  added seed packages: pip==24.3.1
  activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator

C:\Users\gabo\proy>dir
El volumen de la unidad C es Windows
El número de serie del volumen es: D4B1-FE26

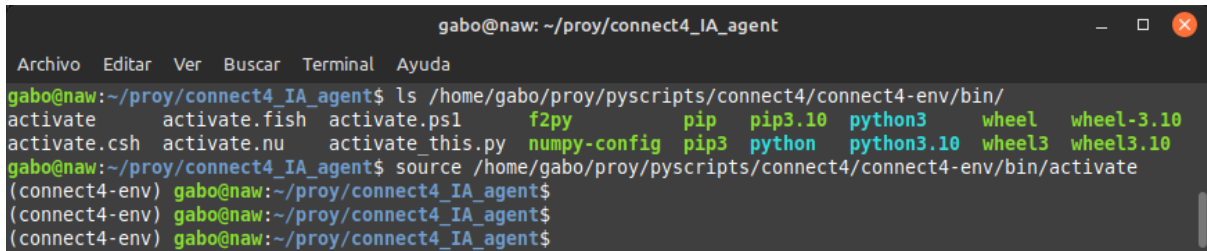
Directorio de C:\Users\gabo\proy

03-11-2024  13:07    <DIR>          .
03-11-2024  12:45    <DIR>          ..
03-11-2024  13:07    <DIR>          connect4-env
03-11-2024  12:33    <DIR>          connect4_IA_agent
                   0 archivos                0 bytes
                   4 dirs  53.337.960.448 bytes libres

C:\Users\gabo\proy>.\connect4-env\Scripts\activate
(connect4-env) C:\Users\gabo\proy>
```

Para iniciar el entorno virtual en *GNU/Linux* se debe ejecutar el comando `source`:

```
source /home/gabo/proy/pyscripts/connect4/connect4-env/bin/activate
```



The screenshot shows a terminal window titled "gabo@naw: ~/proy/connect4_IA_agent". The user runs the command `ls /home/gabo/proy/pyscripts/connect4/connect4-env/bin/`, which lists various activation scripts and dependencies. Then, the user runs `source /home/gabo/proy/pyscripts/connect4/connect4-env/bin/activate`, which successfully activates the virtual environment, changing the prompt to `(connect4-env) gabo@naw:~/proy/connect4_IA_agent$`.

Estructura de archivos del proyecto

Descripción de los archivos ubicados en el repositorio del proyecto:

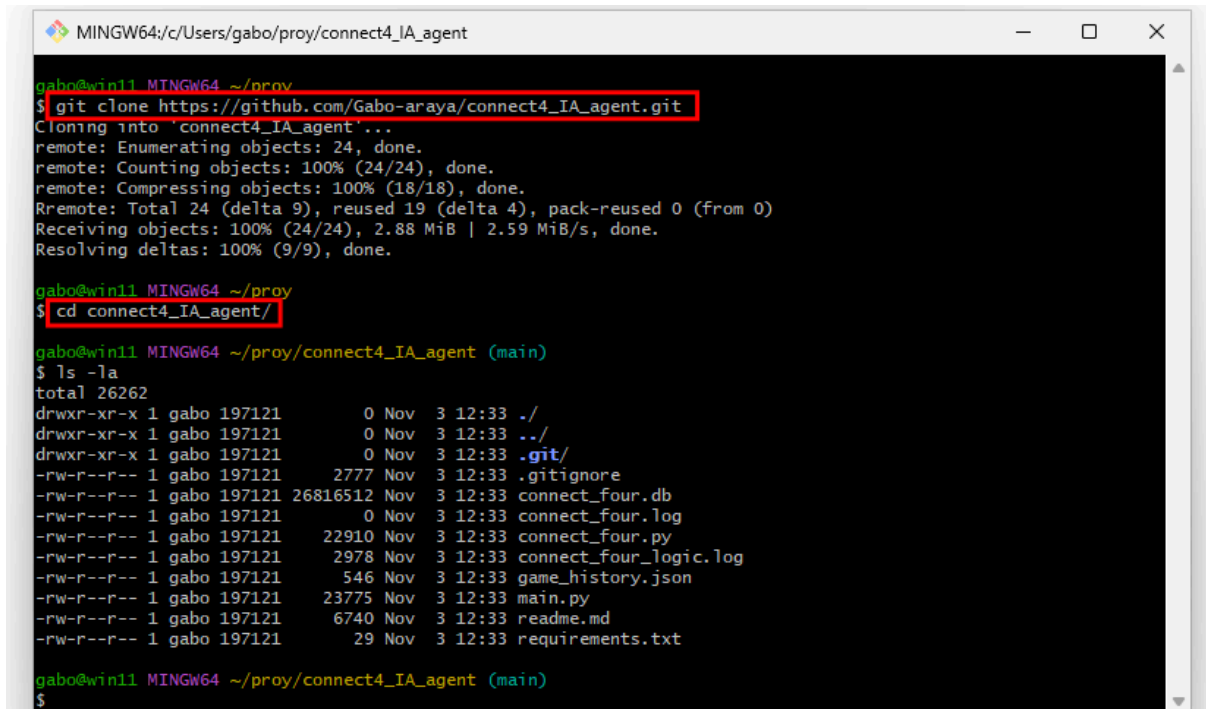
https://github.com/Gabo-araya/connect4_IA_agent

```
connect4_IA_agent/
├── main.py           # Punto de entrada y GUI
├── connect_four.py   # Lógica del juego e IA
├── game_history.json  # Archivo de Log para ajuste de dificultad
├── connect_four_logic.log # Archivo de Log
├── connect_four.log   # Archivo de Log
├── requirements.txt   # Dependencias
├── connect_four.db    # Base de datos SQLite
└── readme.md          # Archivo de descripción del proyecto
```

Instalación del juego en Windows

1. Ubicarse en la carpeta de instalación, clonar el repositorio e ingresar en él:

```
git clone https://github.com/Gabo-araya/connect4_IA_agent.git
cd connect4_IA_agent
```



```
MINGW64/c:/Users/gabo/proy/connect4_IA_agent
gabo@win11 MINGW64 ~/proy
$ git clone https://github.com/Gabo-araya/connect4_IA_agent.git
Cloning into 'connect4_IA_agent'...
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 24 (delta 9), reused 19 (delta 4), pack-reused 0 (from 0)
Receiving objects: 100% (24/24), 2.88 MiB | 2.59 MiB/s, done.
Resolving deltas: 100% (9/9), done.

gabo@win11 MINGW64 ~/proy
$ cd connect4_IA_agent/

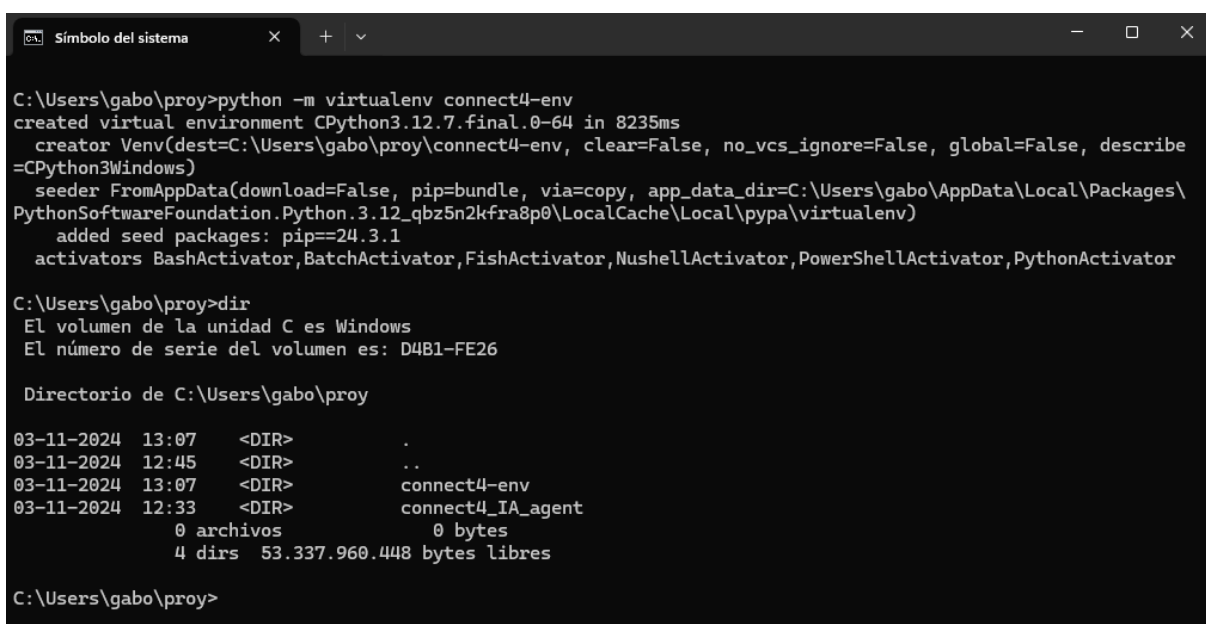
gabo@win11 MINGW64 ~/proy/connect4_IA_agent (main)
$ ls -la
total 26262
drwxr-xr-x 1 gabo 197121      0 Nov  3 12:33 ./
drwxr-xr-x 1 gabo 197121      0 Nov  3 12:33 ../
drwxr-xr-x 1 gabo 197121      0 Nov  3 12:33 .git/
-rw-r--r-- 1 gabo 197121    2777 Nov  3 12:33 .gitignore
-rw-r--r-- 1 gabo 197121 26816512 Nov  3 12:33 connect_four.db
-rw-r--r-- 1 gabo 197121      0 Nov  3 12:33 connect_four.log
-rw-r--r-- 1 gabo 197121   22910 Nov  3 12:33 connect_four.py
-rw-r--r-- 1 gabo 197121   2978 Nov  3 12:33 connect_four_logic.log
-rw-r--r-- 1 gabo 197121    546 Nov  3 12:33 game_history.json
-rw-r--r-- 1 gabo 197121   23775 Nov  3 12:33 main.py
-rw-r--r-- 1 gabo 197121    6740 Nov  3 12:33 readme.md
-rw-r--r-- 1 gabo 197121     29 Nov  3 12:33 requirements.txt

gabo@win11 MINGW64 ~/proy/connect4_IA_agent (main)
$
```

2. Crear y activar el entorno virtual:

Crear un entorno virtual con Python llamado `connect4-env`

```
python -m virtualenv connect4-env
```



```
Símbolo del sistema
C:\Users\gabo\proy>python -m virtualenv connect4-env
created virtual environment CPython3.12.7.final.0-64 in 8235ms
  creator Venv(dest=C:\Users\gabo\proy\connect4-env, clear=False, no_vcs_ignore=False, global=False, describe=CPython3Windows)
  seeder FromAppData(download=False, pip=bundle, via=copy, app_data_dir=C:\Users\gabo\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\Local\pypa\virtualenv)
    added seed packages: pip==24.3.1
    activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator

C:\Users\gabo\proy>dir
El volumen de la unidad C es Windows
El número de serie del volumen es: D4B1-FE26

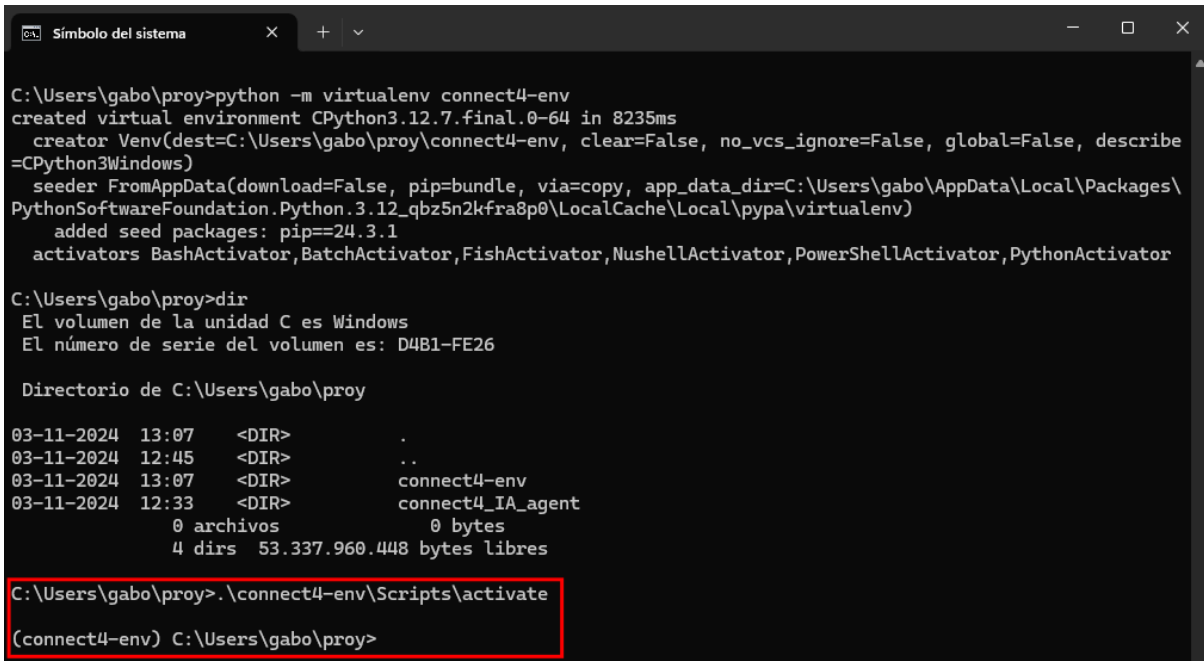
Directorio de C:\Users\gabo\proy

03-11-2024 13:07 <DIR> .
03-11-2024 12:45 <DIR> ..
03-11-2024 13:07 <DIR> connect4-env
03-11-2024 12:33 <DIR> connect4_IA_agent
                0 archivos                0 bytes
                4 dirs 53.337.960.448 bytes libres

C:\Users\gabo\proy>
```

Para iniciar el entorno virtual en *Windows* se debe ejecutar el script `activate`:

```
.\connect4-env\Scripts\activate
```



```
Símbolo del sistema
C:\Users\gabo\proy>python -m virtualenv connect4-env
created virtual environment CPython3.12.7.final.0-64 in 8235ms
creator Venv(dest=C:\Users\gabo\proy\connect4-env, clear=False, no_vcs_ignore=False, global=False, describe
=CPython3Windows)
seeder FromAppData(download=False, pip=bundle, via=copy, app_data_dir=C:\Users\gabo\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\Local\pypa\virtualenv)
added seed packages: pip==24.3.1
activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator

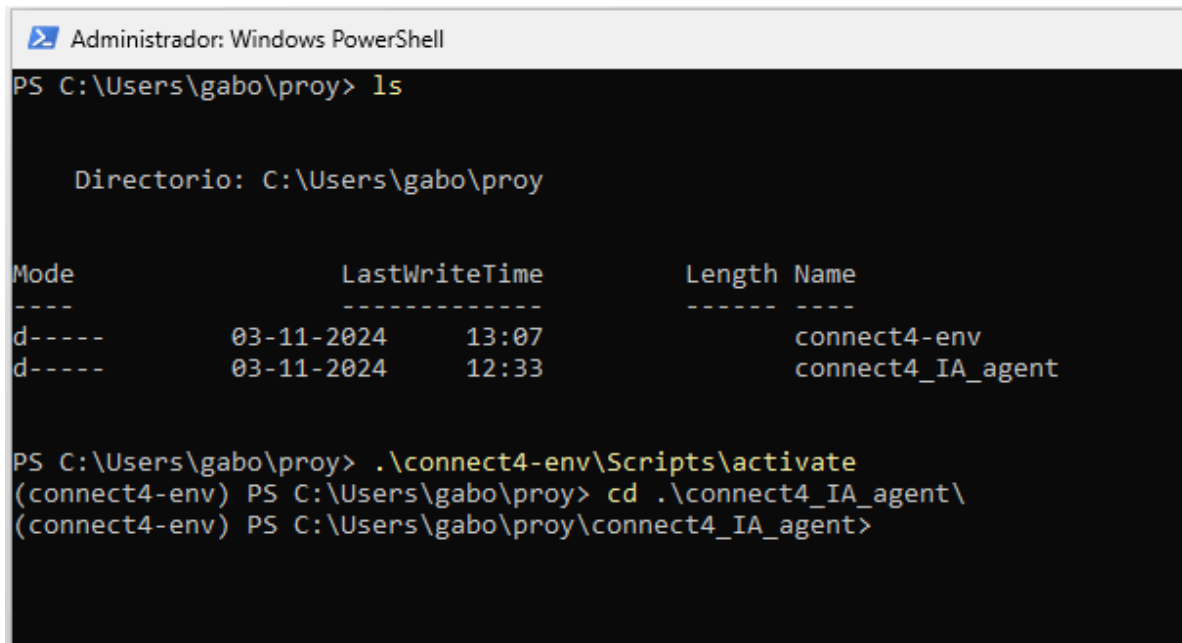
C:\Users\gabo\proy>dir
El volumen de la unidad C es Windows
El número de serie del volumen es: D4B1-FE26

Directorio de C:\Users\gabo\proy
03-11-2024 13:07 <DIR> .
03-11-2024 12:45 <DIR> ..
03-11-2024 13:07 <DIR> connect4-env
03-11-2024 12:33 <DIR> connect4_IA_agent
0 archivos 0 bytes
4 dirs 53.337.960.448 bytes libres

C:\Users\gabo\proy>.\connect4-env\Scripts\activate
(connect4-env) C:\Users\gabo\proy>
```

3. Ingresar en el directorio del repositorio clonado:

```
cd connect4_IA_agent
```



```
Administrador: Windows PowerShell
PS C:\Users\gabo\proy> ls

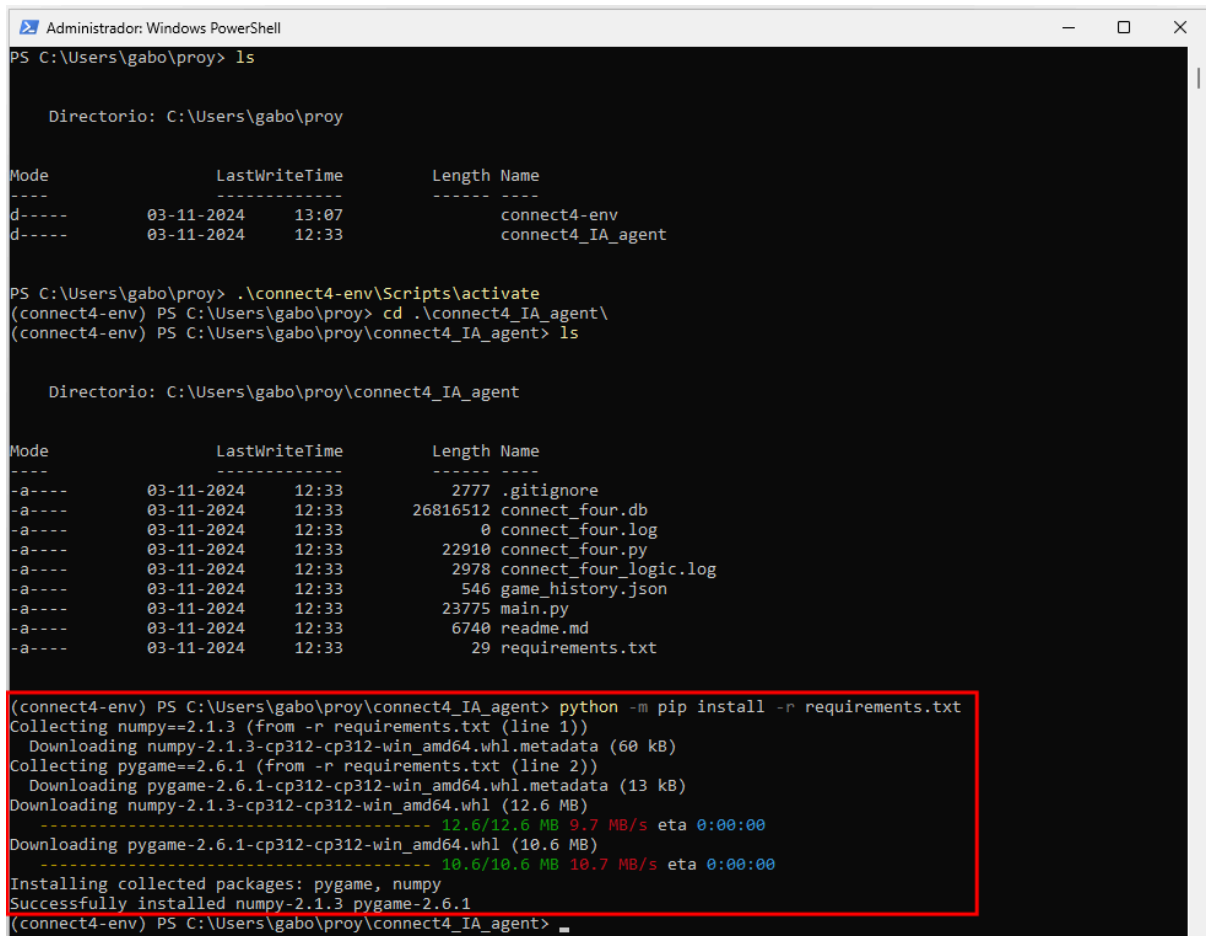
Directorio: C:\Users\gabo\proy

Mode                LastWriteTime         Length Name
----                -
d-----          03-11-2024     13:07         connect4-env
d-----          03-11-2024     12:33         connect4_IA_agent

PS C:\Users\gabo\proy> .\connect4-env\Scripts\activate
(connect4-env) PS C:\Users\gabo\proy> cd .\connect4_IA_agent\
(connect4-env) PS C:\Users\gabo\proy\connect4_IA_agent>
```

4. Instalar dependencias del juego:

```
python -m pip install -r requirements.txt
```



The screenshot shows a Windows PowerShell terminal window titled "Administrador: Windows PowerShell". The user is in the directory `C:\Users\gabo\proy` and runs `ls`, showing two subdirectories: `connect4-env` and `connect4_IA_agent`. The user then navigates to `connect4_IA_agent` and runs `ls`, showing a list of files including `.gitignore`, `connect_four.db`, `connect_four.log`, `connect_four.py`, `connect_four_logic.log`, `game_history.json`, `main.py`, `readme.md`, and `requirements.txt`. Finally, the user runs `python -m pip install -r requirements.txt`, which successfully installs `numpy-2.1.3` and `pygame-2.6.1`. The installation progress is shown with a red box highlighting the download and installation details.

```
Administrador: Windows PowerShell
PS C:\Users\gabo\proy> ls

Directorio: C:\Users\gabo\proy

Mode                LastWriteTime         Length Name
----                -
d-----          03-11-2024    13:07             connect4-env
d-----          03-11-2024    12:33             connect4_IA_agent

PS C:\Users\gabo\proy> .\connect4-env\Scripts\activate
(connect4-env) PS C:\Users\gabo\proy> cd .\connect4_IA_agent\
(connect4-env) PS C:\Users\gabo\proy\connect4_IA_agent> ls

Directorio: C:\Users\gabo\proy\connect4_IA_agent

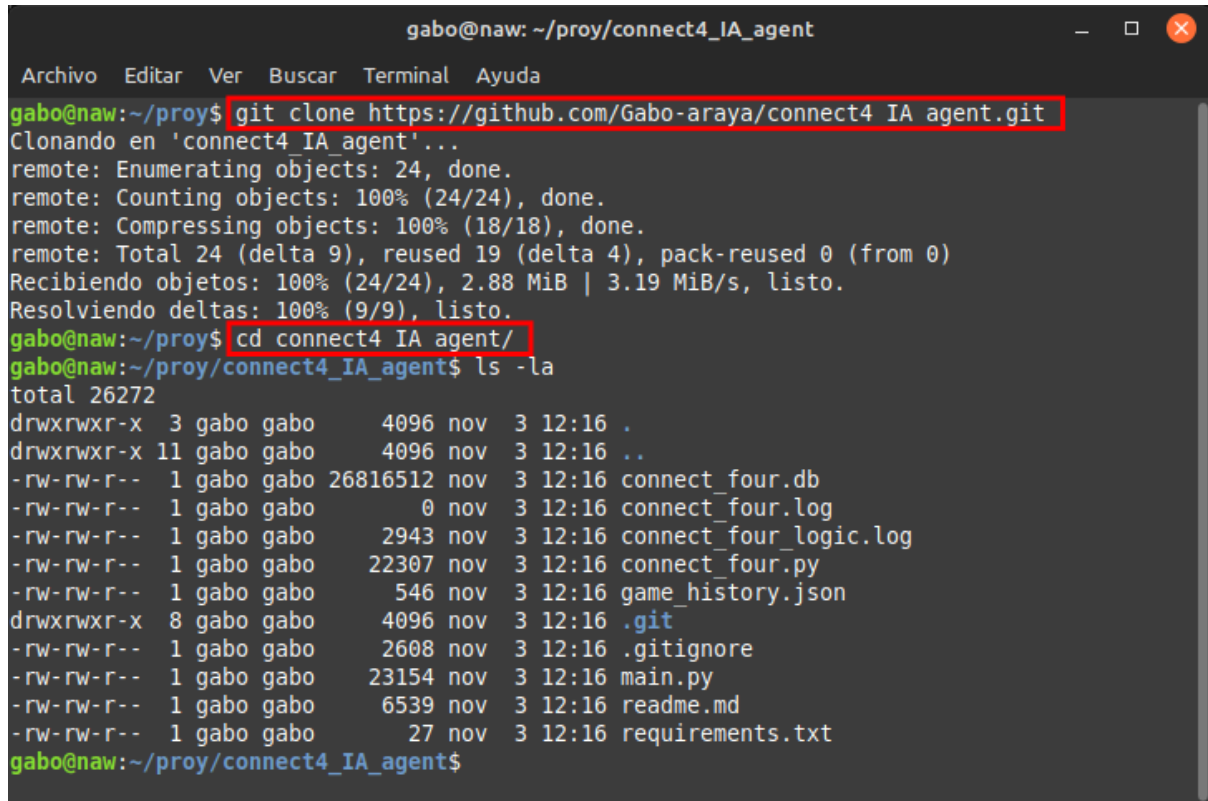
Mode                LastWriteTime         Length Name
----                -
-a-----          03-11-2024    12:33          2777 .gitignore
-a-----          03-11-2024    12:33     26816512 connect_four.db
-a-----          03-11-2024    12:33           0 connect_four.log
-a-----          03-11-2024    12:33     22910 connect_four.py
-a-----          03-11-2024    12:33     2978 connect_four_logic.log
-a-----          03-11-2024    12:33     546 game_history.json
-a-----          03-11-2024    12:33     23775 main.py
-a-----          03-11-2024    12:33     6740 readme.md
-a-----          03-11-2024    12:33         29 requirements.txt

(connect4-env) PS C:\Users\gabo\proy\connect4_IA_agent> python -m pip install -r requirements.txt
Collecting numpy==2.1.3 (from -r requirements.txt (line 1))
  Downloading numpy-2.1.3-cp312-cp312-win_amd64.whl.metadata (60 kB)
Collecting pygame==2.6.1 (from -r requirements.txt (line 2))
  Downloading pygame-2.6.1-cp312-cp312-win_amd64.whl.metadata (13 kB)
Downloaded numpy-2.1.3-cp312-cp312-win_amd64.whl (12.6 MB)
----- 12.6/12.6 MB 9.7 MB/s eta 0:00:00
Downloaded pygame-2.6.1-cp312-cp312-win_amd64.whl (10.6 MB)
----- 10.6/10.6 MB 10.7 MB/s eta 0:00:00
Installing collected packages: pygame, numpy
Successfully installed numpy-2.1.3 pygame-2.6.1
(connect4-env) PS C:\Users\gabo\proy\connect4_IA_agent>
```

Instalación del juego en Linux/macOS

1. Clonar el repositorio:

```
git clone https://github.com/Gabo-araya/connect4_IA_agent.git
cd connect4_IA_agent
```



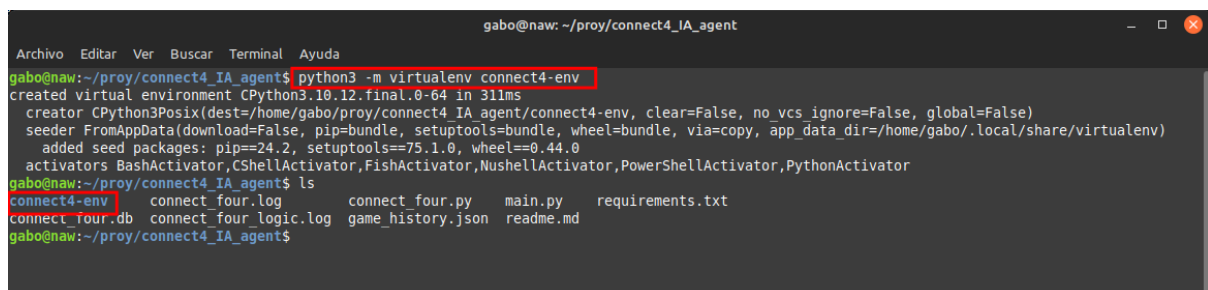
A terminal window titled 'gabo@naw: ~/proy/connect4_IA_agent'. The terminal shows the command 'git clone https://github.com/Gabo-araya/connect4_IA_agent.git' being executed, followed by 'cd connect4_IA_agent/'. The output of the clone command is visible, showing the progress of cloning the repository. The 'ls -la' command is then executed, listing the files in the directory.

```
gabo@naw: ~/proy/connect4_IA_agent
Archivo Editar Ver Buscar Terminal Ayuda
gabo@naw:~/proy$ git clone https://github.com/Gabo-araya/connect4_IA_agent.git
Clonando en 'connect4_IA_agent'...
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 24 (delta 9), reused 19 (delta 4), pack-reused 0 (from 0)
Recibiendo objetos: 100% (24/24), 2.88 MiB | 3.19 MiB/s, listo.
Resolviendo deltas: 100% (9/9), listo.
gabo@naw:~/proy$ cd connect4_IA_agent/
gabo@naw:~/proy/connect4_IA_agent$ ls -la
total 26272
drwxrwxr-x  3 gabo gabo    4096 nov  3 12:16 .
drwxrwxr-x 11 gabo gabo    4096 nov  3 12:16 ..
-rw-rw-r--  1 gabo gabo 26816512 nov  3 12:16 connect_four.db
-rw-rw-r--  1 gabo gabo      0 nov  3 12:16 connect_four.log
-rw-rw-r--  1 gabo gabo   2943 nov  3 12:16 connect_four_logic.log
-rw-rw-r--  1 gabo gabo  22307 nov  3 12:16 connect_four.py
-rw-rw-r--  1 gabo gabo    546 nov  3 12:16 game_history.json
drwxrwxr-x  8 gabo gabo    4096 nov  3 12:16 .git
-rw-rw-r--  1 gabo gabo   2608 nov  3 12:16 .gitignore
-rw-rw-r--  1 gabo gabo  23154 nov  3 12:16 main.py
-rw-rw-r--  1 gabo gabo   6539 nov  3 12:16 readme.md
-rw-rw-r--  1 gabo gabo    27 nov  3 12:16 requirements.txt
gabo@naw:~/proy/connect4_IA_agent$
```

2. Crear y activar el entorno virtual:

Crear un entorno virtual con Python llamado `connect4-env`

```
python3 -m virtualenv connect4-env
```



A terminal window titled 'gabo@naw: ~/proy/connect4_IA_agent'. The terminal shows the command 'python3 -m virtualenv connect4-env' being executed. The output shows the creation of the virtual environment. The 'ls' command is then executed, showing the files in the directory. The 'connect4-env' directory is highlighted, indicating the activation of the virtual environment.

```
gabo@naw: ~/proy/connect4_IA_agent
Archivo Editar Ver Buscar Terminal Ayuda
gabo@naw:~/proy/connect4_IA_agent$ python3 -m virtualenv connect4-env
created virtual environment CPython3.10.12.final.0-64 in 311ms
creator CPython3Posix(dest=/home/gabo/proy/connect4_IA_agent/connect4-env, clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/gabo/.local/share/virtualenv)
added seed packages: pip==24.2, setuptools==75.1.0, wheel==0.44.0
activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
gabo@naw:~/proy/connect4_IA_agent$ ls
connect4-env  connect_four.log  connect_four.py  main.py  requirements.txt
connect_four.db  connect_four_logic.log  game_history.json  readme.md
gabo@naw:~/proy/connect4_IA_agent$
```


Para iniciar el entorno virtual en *GNU/Linux* se debe ejecutar el comando `source`:

```
source connect4-env/bin/activate
```

```
gabo@naw: ~/proj/connect4_IA_agent
Archivo Editar Ver Buscar Terminal Ayuda
gabo@naw:~/proj/connect4_IA_agent$ python3 -m virtualenv connect4-env
created virtual environment CPython3.10.12.final.0-64 in 311ms
creator CPython3Posix(dest=/home/gabo/proj/connect4_IA_agent/connect4-env, clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/gabo/.local/share/virtualenv)
added seed packages: pip==24.2, setuptools==75.1.0, wheel==0.44.0
activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
gabo@naw:~/proj/connect4_IA_agent$ ls
connect4-env  connect_four.log  connect_four.py  main.py  requirements.txt
connect_four.db  connect_four_logic.log  game_history.json  readme.md
gabo@naw:~/proj/connect4_IA_agent$ source connect4-env/bin/activate
(connect4-env) gabo@naw:~/proj/connect4_IA_agent$
```

3. Instalar dependencias:

```
pip3 install -r requirements.txt
```

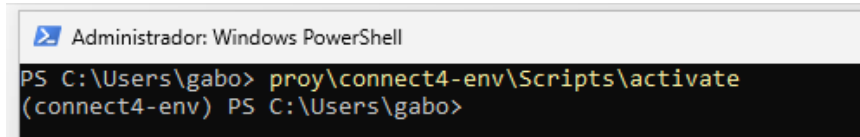
```
gabo@naw: ~/proj/connect4_IA_agent
Archivo Editar Ver Buscar Terminal Ayuda
gabo@naw:~/proj/connect4_IA_agent$ python3 -m virtualenv connect4-env
created virtual environment CPython3.10.12.final.0-64 in 311ms
creator CPython3Posix(dest=/home/gabo/proj/connect4_IA_agent/connect4-env, clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/gabo/.local/share/virtualenv)
added seed packages: pip==24.2, setuptools==75.1.0, wheel==0.44.0
activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
gabo@naw:~/proj/connect4_IA_agent$ ls
connect4-env  connect_four.log  connect_four.py  main.py  requirements.txt
connect_four.db  connect_four_logic.log  game_history.json  readme.md
gabo@naw:~/proj/connect4_IA_agent$ source connect4-env/bin/activate
(connect4-env) gabo@naw:~/proj/connect4_IA_agent$ pip3 install -r requirements.txt
Collecting numpy==2.1.3 (from -r requirements.txt (line 1))
  Using cached numpy-2.1.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (62 kB)
Collecting pygame==2.6.1 (from -r requirements.txt (line 2))
  Using cached pygame-2.6.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Using cached numpy-2.1.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.3 MB)
Using cached pygame-2.6.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (14.0 MB)
Installing collected packages: pygame, numpy
Successfully installed numpy-2.1.3 pygame-2.6.1

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: pip install --upgrade pip
(connect4-env) gabo@naw:~/proj/connect4_IA_agent$
```

Ejecución del juego en Windows

1. Asegurarse que el entorno virtual está activado

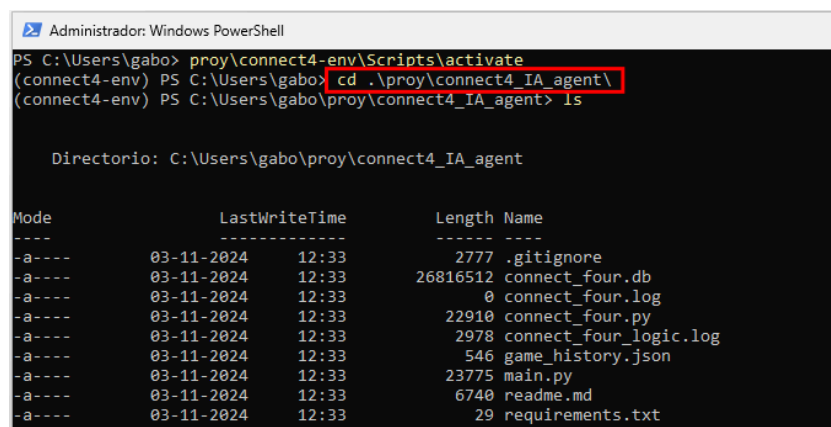
```
connect4-env\Scripts\activate
```



```
Administrador: Windows PowerShell
PS C:\Users\gabo> proj\connect4-env\Scripts\activate
(connect4-env) PS C:\Users\gabo>
```

2. Asegurarse de que se está posicionado en el directorio adecuado

```
cd proj\connect4_IA_agent
```



```
Administrador: Windows PowerShell
PS C:\Users\gabo> proj\connect4-env\Scripts\activate
(connect4-env) PS C:\Users\gabo> cd .\proj\connect4_IA_agent\
(connect4-env) PS C:\Users\gabo\proj\connect4_IA_agent> ls

Directorio: C:\Users\gabo\proj\connect4_IA_agent

Mode                LastWriteTime         Length Name
----                -
-a----          03-11-2024   12:33             2777 .gitignore
-a----          03-11-2024   12:33        26816512 connect_four.db
-a----          03-11-2024   12:33              0 connect_four.log
-a----          03-11-2024   12:33         22910 connect_four.py
-a----          03-11-2024   12:33         2978 connect_four_logic.log
-a----          03-11-2024   12:33          546 game_history.json
-a----          03-11-2024   12:33         23775 main.py
-a----          03-11-2024   12:33          6740 readme.md
-a----          03-11-2024   12:33           29 requirements.txt
```

3. Ejecutar el juego

```
python main.py
```

Se deben ingresar datos relacionados con el tamaño del tablero (normal o pequeño), el nivel de dificultad (Fácil, Medio o Difícil) y quién comienza el juego (Humano o IA), poniendo números para indicar la opción deseada.

```
(connect4-env) PS C:\Users\gabo\proj\connect4_IA_agent> python main.py
pygame 2.6.1 (SDL 2.28.4, Python 3.12.7)
Hello from the pygame community. https://www.pygame.org/contribute.html
Bienvenido a Connect Four con IA!

Seleccione el tamaño del tablero:
1. Normal (6x7)
2. Pequeño (5x4)
Opción (1-2): 1

Seleccione el nivel de dificultad:
1. Fácil
2. Medio
3. Difícil
Opción (1-3): 1

¿Quién comienza el juego?
1. Jugador Humano
2. IA
Opción (1-2): 1
```

```
Administrador: Windows PowerShell

PS C:\Users\gabo> proy\connect4-env\Scripts\activate
(connect4-env) PS C:\Users\gabo> cd .\proy\connect4_IA_agent\
(connect4-env) PS C:\Users\gabo\proy\connect4_IA_agent> ls

Directorio: C:\Users\gabo\proy\connect4_IA_agent

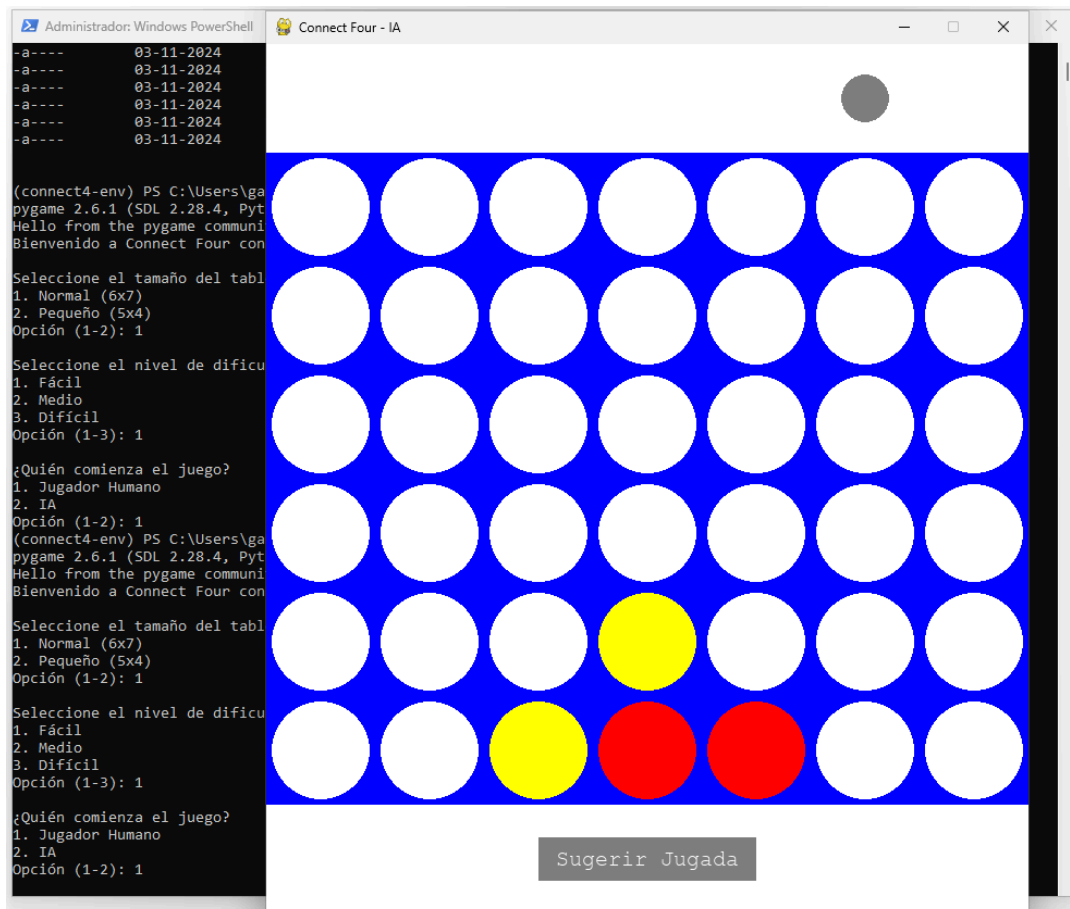
Mode                LastWriteTime         Length Name
----                -
-a----             03-11-2024      12:33           2777 .gitignore
-a----             03-11-2024      12:33      26816512 connect_four.db
-a----             03-11-2024      12:33            0 connect_four.log
-a----             03-11-2024      12:33       22910 connect_four.py
-a----             03-11-2024      12:33       2978 connect_four_logic.log
-a----             03-11-2024      12:33        546 game_history.json
-a----             03-11-2024      12:33       23775 main.py
-a----             03-11-2024      12:33       6740 readme.md
-a----             03-11-2024      12:33         29 requirements.txt

(connect4-env) PS C:\Users\gabo\proy\connect4_IA_agent> python main.py
pygame 2.6.1 (SDL 2.28.4, Python 3.12.7)
Hello from the pygame community. https://www.pygame.org/contribute.html
Bienvenido a Connect Four con IA!

Seleccione el tamaño del tablero:
1. Normal (6x7)
2. Pequeño (5x4)
Opción (1-2): 1

Seleccione el nivel de dificultad:
1. Fácil
2. Medio
3. Difícil
Opción (1-3): 1

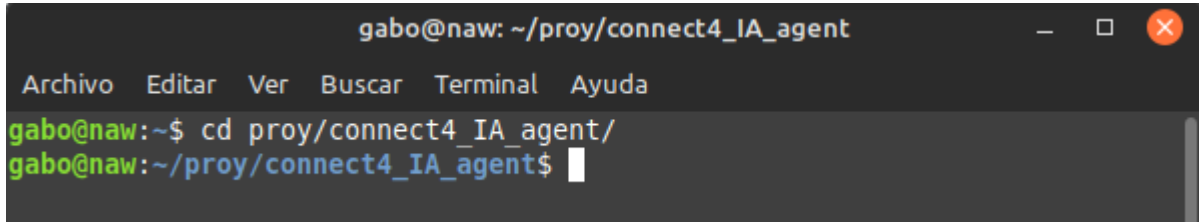
¿Quién comienza el juego?
1. Jugador Humano
2. IA
Opción (1-2): 1
```



Ejecución del juego en Linux

1. Asegurarse de que se está posicionado en el directorio adecuado

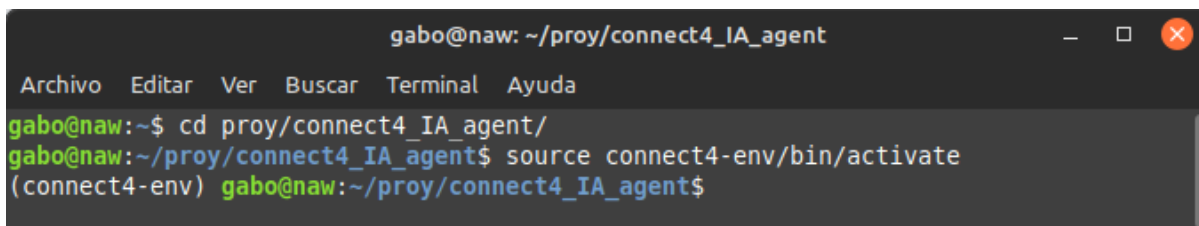
```
cd proy/connect4_IA_agent/
```



A terminal window titled 'gabo@naw: ~/proy/connect4_IA_agent' with a menu bar (Archivo, Editar, Ver, Buscar, Terminal, Ayuda). The command history shows: 'gabo@naw:~\$ cd proy/connect4_IA_agent/' followed by 'gabo@naw:~/proy/connect4_IA_agent\$' with a cursor.

2. Asegurarse que el entorno virtual está activado

```
source connect4-env/bin/activate
```



A terminal window titled 'gabo@naw: ~/proy/connect4_IA_agent' with a menu bar. The command history shows: 'gabo@naw:~\$ cd proy/connect4_IA_agent/' followed by 'gabo@naw:~/proy/connect4_IA_agent\$ source connect4-env/bin/activate'. The prompt changes to '(connect4-env) gabo@naw:~/proy/connect4_IA_agent\$'.

3. Ejecutar el juego

```
python3 main.py
```

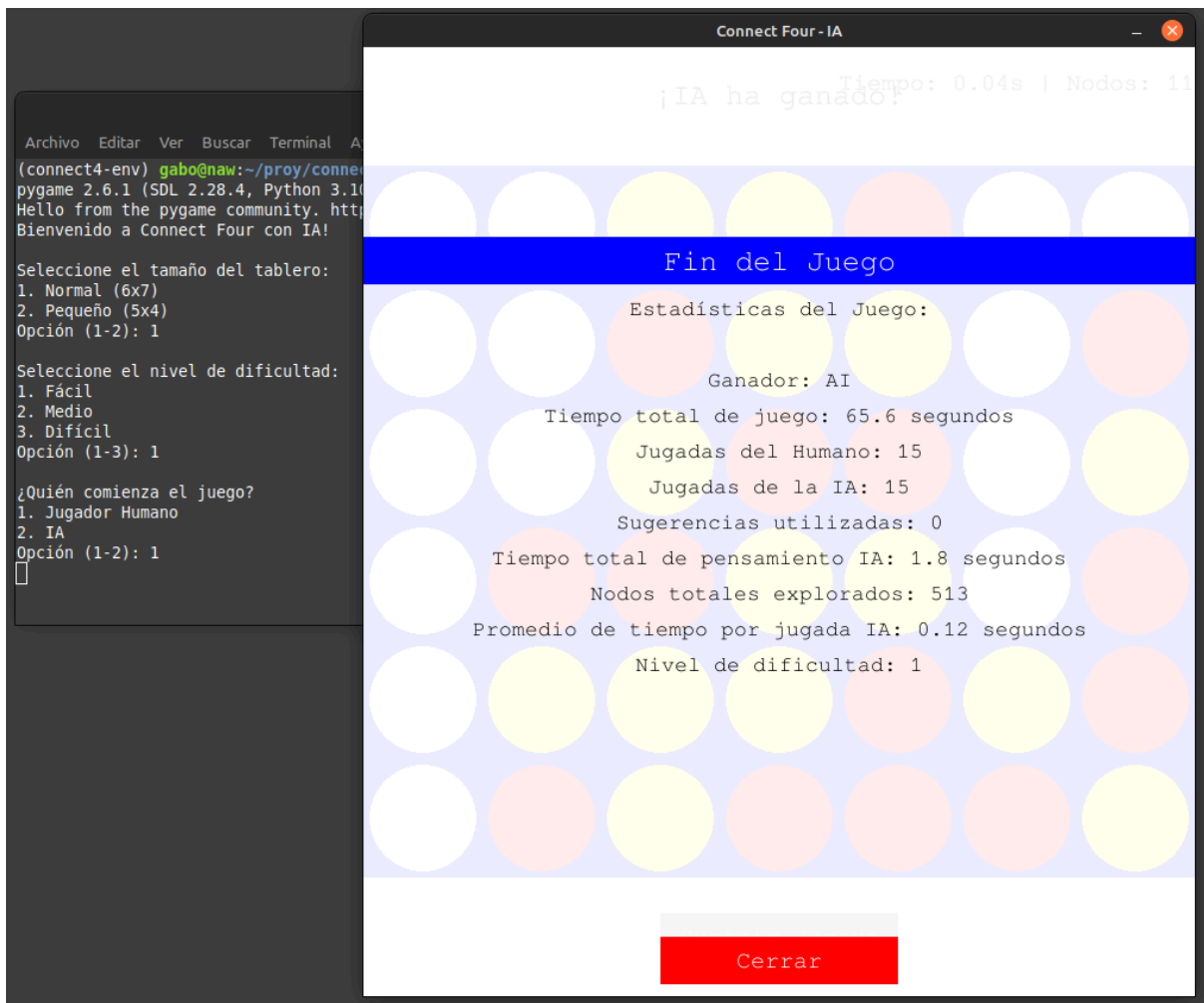
Se deben ingresar datos relacionados con el tamaño del tablero (normal o pequeño), el nivel de dificultad (Fácil, Medio o Difícil) y quién comienza el juego (Humano o IA), poniendo números para indicar la opción deseada.

```
(connect4-env) PS C:\Users\gabo\proy\connect4_IA_agent> python main.py
pygame 2.6.1 (SDL 2.28.4, Python 3.12.7)
Hello from the pygame community. https://www.pygame.org/contribute.html
Bienvenido a Connect Four con IA!

Seleccione el tamaño del tablero:
1. Normal (6x7)
2. Pequeño (5x4)
Opción (1-2): 1

Seleccione el nivel de dificultad:
1. Fácil
2. Medio
3. Difícil
Opción (1-3): 1

¿Quién comienza el juego?
1. Jugador Humano
2. IA
Opción (1-2): 1
```



Cómo jugar

Inicio del juego

Al iniciar el juego, seleccionar:

1. **Tamaño del tablero:** usando números entre el 1 y el 2, seleccionar entre un tablero de tamaño Normal (6 filas por 7 columnas, opción 1) o un tablero de tamaño Pequeño (de 5 filas por 4 columnas, opción 2).

```
Bienvenido a Connect Four con IA!  
  
Seleccione el tamaño del tablero:  
1. Normal (6x7)  
2. Pequeño (5x4)  
Opción (1-2): 1
```

2. **Nivel de dificultad:** usando números entre el 1 y el 3, seleccionar entre un nivel de dificultad Fácil (opción 1), Medio (opción 2) o Difícil (opción 3). Si se escoge una dificultad Difícil, la IA demorará algunos segundos en realizar su jugada.

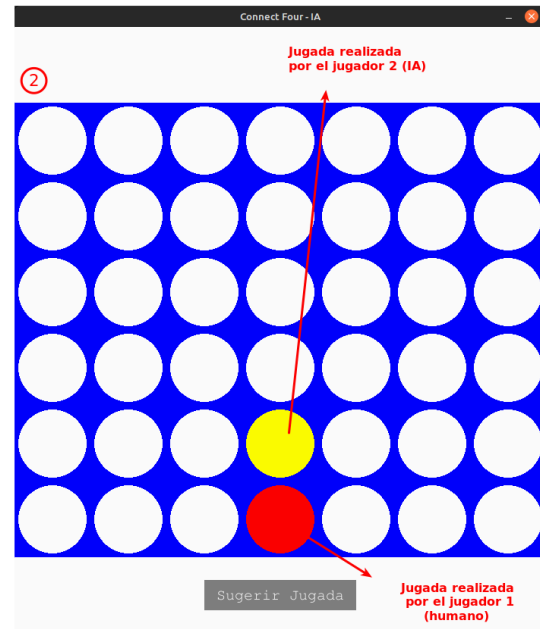
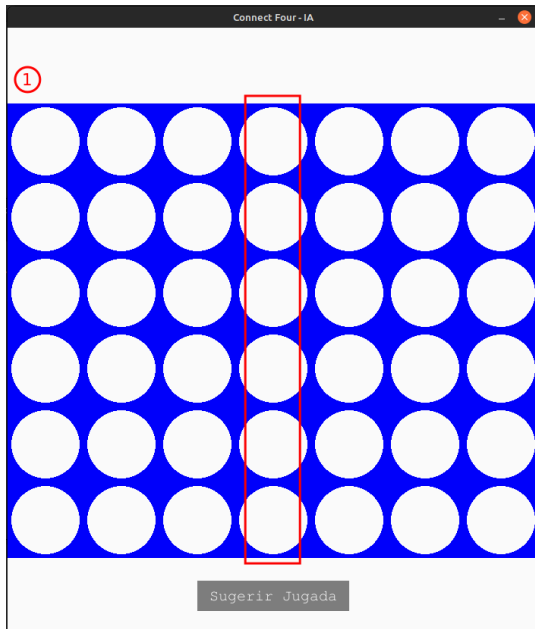
```
Seleccione el nivel de dificultad:  
1. Fácil  
2. Medio  
3. Difícil  
Opción (1-3): 1
```

3. **Jugador inicial (Humano o IA):** usando números entre el 1 y el 2, seleccionar si el juego debe ser iniciado por el jugador humano (opción 1) o por el agente IA (opción 2).

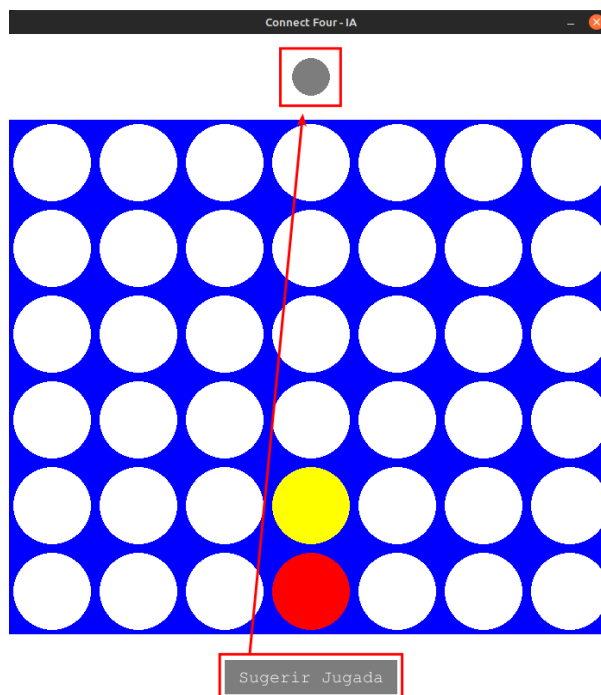
```
¿Quién comienza el juego?  
1. Jugador Humano  
2. IA  
Opción (1-2): 1
```

Durante el juego

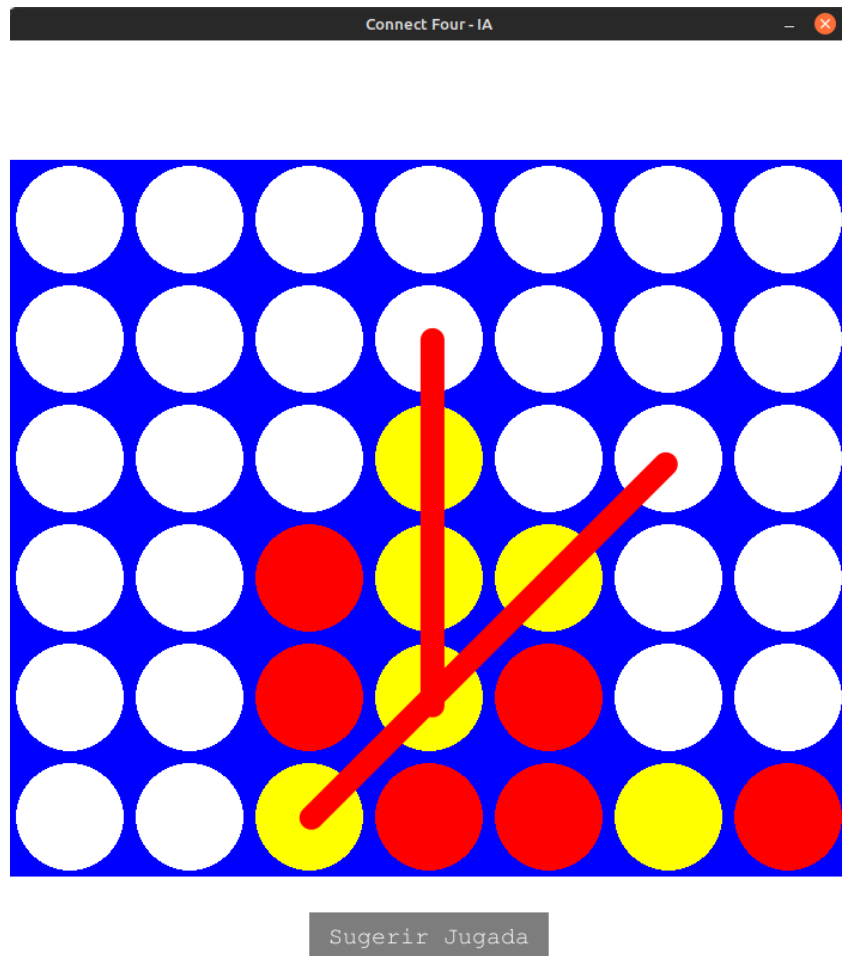
1. Haz clic en una columna para soltar una ficha. La IA realizará su jugada automáticamente. Si se escoge una dificultad alta, la IA demorará algunos segundos en realizar su jugada.



2. El **jugador humano** puede usar el botón "Sugerir Jugada" si necesita ayuda. Esto hará aparecer un **círculo gris arriba de la columna** donde se sugiere realizar la siguiente jugada.



3. El objetivo es **conectar 4 fichas del mismo color**. En la imagen se pueden ver dos posibles opciones para conectar 4 fichas.



Condiciones de finalización

El juego termina cuando:

1. Un jugador conecta 4 fichas.
2. El tablero se llena (empate).

Al final del juego aparece un mensaje con **estadísticas del juego** que acaba de ocurrir.

Connect Four - IA

Tiempo: 0.04s | Nodos: 11

¡IA ha ganado!

Fin del Juego

Estadísticas del Juego:

Ganador: AI

Tiempo total de juego: 65.6 segundos

Jugadas del Humano: 15

Jugadas de la IA: 15

Sugerencias utilizadas: 0

Tiempo total de pensamiento IA: 1.8 segundos

Nodos totales explorados: 513

Promedio de tiempo por jugada IA: 0.12 segundos

Nivel de dificultad: 1

Cerrar

Las estadísticas de la partida y los movimientos realizados por los jugadores se guardan en una base de datos SQLite para su posterior análisis.

DB Browser for SQLite - /home/gabo/proy/pyscripts/connect4/connect_four.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragma Execute SQL

Table: games

Filter in any column

	game_id	initial_player	winner_player	timestamp	dificultad	filas	columnas	tiempo_juego	jugadas_humano	jugadas_ia
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	b674089f-1374-40f8-a8d4-2dc829482542	HUMAN	AI	2024-11-02 21:58:52.648749	1	6	7	16.9370017051697	9	9
2	175e3549-269b-414e-8f37-42f4a1bb2404	HUMAN	EMPATE	2024-11-02 22:11:36.259982	3	5	4	39.8252968788147	10	10
3	0c796b09-5fdc-4065-883e-6ae92b6a6046	HUMAN	AI	2024-11-02 22:25:27.361632	2	6	7	95.6059195995331	15	15
4	3f1c1ed5-f216-46cf-bf2e-058b42f0c69b	AI	AI	2024-11-02 22:31:26.125230	3	6	7	140.465993404388	8	9
5	a2957175-41d4-4378-bd76-4e6da82417cb	HUMAN	AI	2024-11-02 22:34:33.952735	3	6	7	400.793928623199	7	7
6	8b2a5954-332f-42f3-8256-72e2c808bfb1	HUMAN	AI	2024-11-02 22:48:10.048864	1	6	7	12.8906810283661	7	7
7	bf0b0e36-7e61-4a9f-b45c-6d9f1a465166	HUMAN	NULL	2024-11-02 23:48:00.925931	1	6	7	NULL	NULL	NULL
8	7c9a73f8-afe8-47aa-ac1e-5c85a350620b	HUMAN	NULL	2024-11-02 23:49:51.113172	1	6	7	NULL	NULL	NULL
9	d3a37a25-fabc-4437-862d-92dc5077fece	HUMAN	AI	2024-11-02 23:57:00.962869	1	6	7	9.59276747703552	6	6
10	68c58554-10e6-4e53-b91f-178c3153cec0	AI	AI	2024-11-02 23:57:24.374129	2	6	7	29.0203049182892	9	10
11	d4ca8c19-5ce3-4e00-85e4-43ef26caa096	HUMAN	AI	2024-11-03 00:14:57.599485	1	6	7	19.7004210948944	7	7
12	7537308a-03dd-4009-92e5-823fcea29197	AI	AI	2024-11-03 00:32:48.347228	1	6	7	6.0238823890686	4	5
13	9a5e2fe4-933f-4ae4-8eec-2238e42c0341	AI	AI	2024-11-03 00:51:56.259587	3	6	7	241.26106262207	12	13
14	a256e5ba-cf0b-4e01-86fa-42511fd5250e	HUMAN	AI	2024-11-03 02:24:00.459648	1	6	7	23.2908809185028	5	5
15	788010fc-36e3-4529-a612-d9d0dd941e54	HUMAN	AI	2024-11-03 02:24:54.558122	1	6	7	39.2044758796692	12	12
16	60d5def0-5333-4d72-af8b-d8acc4f08bd	HUMAN	IA	2024-11-03 02:25:49.670505	1	6	7	54.6035268306732	9	9
17	7257aa9e-fa5e-481e-9df4-c8eeb6c697d	HUMAN	AI	2024-11-03 02:28:04.006047	1	6	7	40.9089384078979	10	10
18	b27fa34c-8a03-466b-b098-af9241011487	HUMAN	HUMAN	2024-11-03 02:28:53.189270	1	6	7	8.70083737373352	5	4
19	a445d976-5a44-465e-8f72-12d4afc51fb9	HUMAN	IA	2024-11-03 02:29:09.923781	1	6	7	54.6420292854309	6	6

1 - 18 of 19

Go to: 1

1 row(s), 16 column(s), Sum: 345.628, Average: 21.6017, Min: 0, Max: 243

Edit Database Cell

Mode: Text

1 a445d976-5a44-465e-8f72-12d4afc51fb9

Type of data currently in cell: Text / Numeric 36 character(s) Apply

Identity Select an identity to connect

DBHub.io Local Current Database

Name

SQL Log Plot DB Schema Remote

UTF-8

Taller de Inteligencia Artificial Aplicada | AICC103.202425.2107.EL.ON

25

Agentes IA para Connect4

Técnicas de IA utilizadas

El agente de IA utiliza principalmente tres técnicas fundamentales.

- **Minimax con poda Alfa-Beta:** Tradicionalmente la técnica más utilizada, constituye el núcleo de la toma de decisiones del agente y permite explorar el árbol de juego de manera eficiente.
- **Función de Evaluación Heurística:** Evalúa qué tan buena es una posición del tablero para la IA.
- **Sistema de Dificultad Adaptativa:** Ajusta la dificultad del juego basándose en el rendimiento del jugador.

Algoritmo Minimax con poda Alfa-Beta

Es el núcleo de la toma de decisiones del agente, implementado en el método `minimax()` de la clase `ConnectFour`. Este algoritmo:

- Explora el árbol de posibles jugadas hasta una profundidad determinada.
- Maximiza el resultado para la IA mientras asume que el oponente jugará de forma óptima.
- Utiliza poda Alfa-Beta para reducir el número de nodos explorados.

```
def minimax(self, depth: int, alpha: float, beta: float, maximizing_player: bool) ->
    Tuple[float, Optional[int]]:
    self.nodes_explored += 1
    valid_moves = self.get_valid_moves()

    # Verifica estados terminales
    if self.check_winner(self.AI):
        return (float('inf'), None)
    if self.check_winner(self.PLAYER):
        return (float('-inf'), None)

    if maximizing_player:
        value = float('-inf')
        column = valid_moves[0]
        for col in valid_moves:
            # Explora movimientos maximizando
            new_score, _ = self.minimax(depth-1, alpha, beta, False)
            if new_score > value:
                value = new_score
                column = col
            alpha = max(alpha, value)
            if alpha >= beta: # Poda alfa-beta
                break
```

Función de Evaluación Heurística

Implementada en el método `evaluate_position()`, esta función evalúa qué tan buena es una posición del tablero para la IA considerando varios factores:

- Control del centro del tablero
- Potenciales líneas de 4 en todas direcciones
- Bloqueo de jugadas ganadoras del oponente

```
def evaluate_position(self) -> int:
    score = 0

    # Evalúa control del centro
    center_array = tuple(self.board[row][self.COLUMNS//2] for row in range(self.ROWS))
    center_count = len([x for x in center_array if x == self.AI])
    score += center_count * 3

    # Evalúa ventanas de 4 posiciones en todas direcciones
    for row in range(self.ROWS):
        for col in range(self.COLUMNS - 3):
            window = tuple(row_array[col:col + self.WINDOW_LENGTH])
            score += self.evaluate_window(window, self.AI)
```

Sistema de Dificultad Adaptativa

Implementado en el método `adjust_difficulty()`, ajusta la dificultad del juego basándose en el rendimiento del jugador:

- Mantiene un registro de las últimas 5 partidas
- Ajusta la profundidad de búsqueda del algoritmo Minimax según el rendimiento
- Utiliza tres niveles de dificultad con diferentes profundidades de búsqueda

```
def adjust_difficulty(self, player_won: bool) -> None:
    try:
        with self.get_db_connection() as conn:
            cursor = conn.cursor()
            # Obtiene últimas 5 partidas
            cursor.execute('''
                SELECT winner_player
                FROM games
                ORDER BY timestamp DESC
                LIMIT 5
            ''')
            recent_games = cursor.fetchall()

            # Ajusta dificultad según rendimiento
            wins = sum(1 for game in recent_games if game['winner_player'] == 'HUMAN')
            if wins >= 4 and self.DIFFICULTY < 3:
                self.DIFFICULTY += 1
            elif wins <= 1 and self.DIFFICULTY > 1:
                self.DIFFICULTY -= 1
```

Reglas de Connect4

Objetivo del juego

Alinear cuatro fichas del mismo color de forma horizontal, vertical o diagonal antes que el oponente.

Preparación del juego

- **Jugadores:** 2
- **Tablero:** Una cuadrícula vertical de 7 columnas y 6 filas.
- **Fichas:** Cada jugador elige un color (generalmente, rojo o amarillo) y recibe una cantidad suficiente de fichas de su color.

Reglas del juego

- **Turnos alternados:** Los jugadores se turnan para lanzar una ficha en una de las columnas.
- **Caída de las fichas:** Cada ficha lanzada cae hasta la posición más baja disponible en la columna elegida.
- **Alineación de cuatro:** Gana el primer jugador que alinee cuatro de sus fichas consecutivamente en cualquier dirección (horizontal, vertical o diagonal).

Fin del juego

- **Victoria:** Si un jugador logra conectar cuatro fichas consecutivas, ese jugador gana y el juego termina.
- **Empate:** Si el tablero se llena completamente sin que nadie haya conectado cuatro fichas, el juego termina en empate.

Acciones no permitidas

- No se pueden retirar fichas una vez colocadas en el tablero.
- Si una columna está llena, el jugador debe elegir otra columna para colocar su ficha.

Estrategia

- **Bloquear al oponente:** Si un jugador está a punto de conectar cuatro fichas, el otro debe tratar de evitarlo colocando una ficha en la línea.
- **Planificación anticipada:** Buscar formar dos o más líneas posibles de cuatro fichas para tener más opciones de victoria.

Diseño del agente

El agente IA utiliza el algoritmo Minimax con poda Alfa-Beta como su núcleo principal. Minimax es un algoritmo de toma de decisiones que simula dos jugadores (MAX y MIN) que juegan de manera óptima. En Conecta 4:

- MAX (IA) intenta maximizar su puntuación
- MIN (jugador humano) intenta minimizar la puntuación de la IA

El algoritmo Minimax explora el árbol de jugadas posibles, alterna entre MAX (IA) y MIN (jugador) y propaga valores hacia arriba en el árbol.

Por su parte, la poda alpha-beta se puede explicar con las siguientes proposiciones:

- α (alpha): mejor valor encontrado para MAX
- β (beta): mejor valor encontrado para MIN
- Si $\alpha \geq \beta$, podemos podar (no explorar) el resto de las ramas

La elección de este algoritmo se justifica por:

- **Determinismo:** Connect4 es un juego de información perfecta, donde Minimax garantiza la mejor jugada posible dentro de la profundidad de búsqueda establecida.
- **Eficiencia:** La poda Alfa-Beta reduce significativamente el número de nodos explorados, permitiendo mayores profundidades de búsqueda.
- **Adaptabilidad:** Permite ajustar fácilmente la dificultad modificando la profundidad de búsqueda.

```
def minimax(self, depth: int, alpha: float, beta: float, maximizing_player: bool) ->
    Tuple[float, Optional[int]]:
    self.nodes_explored += 1
    valid_moves = self.get_valid_moves()

    # Verificar estados terminales
    if self.check_winner(self.AI):
        return (float('inf'), None)
    if self.check_winner(self.PLAYER):
        return (float('-inf'), None)
    if not valid_moves:
        return (0, None)
    if depth == 0:
        return (self.evaluate_position(), None)
```

Representación para el estado del juego

En el juego de Connect4, el "estado" del juego se representa mediante una matriz bidimensional utilizando una lista de listas en Python, como en:

```
self.board = [[self.EMPTY] * columns for _ in range(rows)]
```

Esta representación se eligió por varios motivos:

- **Simplicidad:** Es fácil de entender y manipular, ya que refleja directamente la estructura del tablero en 2D.
- **Eficiencia espacial:** Solo almacena la información necesaria sin datos adicionales, aprovechando bien el espacio.
- **Acceso directo:** Permite acceder a cualquier posición del tablero en tiempo constante $O(1)$, lo cual es útil para consultar y actualizar celdas de manera rápida.

La disposición del tablero en esta matriz permite que cada celda de la lista contenga un valor que representa el estado de esa posición:

- 0 para una celda vacía,
- 1 para una ficha del jugador 1,
- 2 para una ficha del jugador 2.

Ejemplo de una matriz para un tablero parcialmente lleno:

```
[ [0, 0, 1, 0, 2, 0, 0],  
  [0, 1, 2, 1, 2, 0, 0],  
  [1, 2, 1, 2, 1, 0, 0],  
  [2, 1, 2, 1, 2, 1, 0],  
  [1, 2, 1, 2, 1, 2, 0],  
  [2, 1, 2, 1, 2, 1, 0] ]
```

Alternativas consideradas pero descartadas

- **BitBoard:** Esta alternativa es más eficiente en términos de uso de memoria, ya que representa el estado como un conjunto de bits. Sin embargo, la implementación y el mantenimiento son considerablemente más complejos, y requieren técnicas avanzadas para manipular los bits de manera correcta.
- **Array unidimensional:** Aunque es posible representar el tablero en una lista lineal de 42 elementos (6x7), esta opción resulta menos intuitiva para una estructura de tablero 2D, ya que implica realizar un mapeo de índices para acceder a las posiciones en filas y columnas.

Estrategia de generación de jugadas

La estrategia de generación de jugadas en Connect4, se basa en evaluar cada posible movimiento para elegir el más favorable según el estado actual del tablero. Este proceso implica considerar todas las columnas en las que es posible colocar una ficha y generar un nuevo estado del tablero para cada jugada candidata. Luego, cada estado generado se evalúa en función de ciertos criterios, como:

1. **Prioridad de victoria inmediata:** Si existe una jugada que permite ganar de inmediato, esta se ejecuta sin necesidad de evaluar otras opciones.
2. **Bloqueo de oponente:** Si el adversario tiene una jugada que le garantiza la victoria en su próximo turno, el agente prioriza bloquear esa columna.
3. **Evaluación de posiciones estratégicas:** En ausencia de jugadas ganadoras o bloqueos, el agente prioriza movimientos hacia el centro o en posiciones que maximicen las posibilidades de crear conexiones futuras de cuatro fichas en línea.

Para optimizar la generación de jugadas, la estrategia puede utilizar un enfoque de “búsqueda y evaluación heurística”, donde cada posible movimiento se califica con un puntaje según las oportunidades y amenazas que genera. En agentes más avanzados, como aquellos basados en algoritmos de búsqueda de minimax con poda alfa-beta, se pueden prever varios movimientos hacia adelante, evaluando los resultados posibles de secuencias de jugadas, lo cual permite elegir un movimiento que maximice las probabilidades de victoria a largo plazo.

Las jugadas se generan considerando las columnas disponibles de izquierda a derecha. Se implementa con:

```
def get_valid_moves(self) -> List[int]:
    return [col for col in range(self.COLUMNS) if self.is_valid_move(col)]

@lru_cache(maxsize=1024)
def is_valid_move(self, col: int) -> bool:
    return self.board[0][col] == self.EMPTY
```

Se utiliza `@lru_cache(maxsize=1024)` para optimizar el rendimiento al evitar recalcular validaciones repetidas.

Función de evaluación

En Connect4, la función de utilidad o evaluación es crucial para que el agente determine la calidad de cada estado del juego y elija la mejor jugada. Esta función asigna un valor numérico a un estado del tablero, reflejando lo favorable o desfavorable que es dicho estado para el jugador. Una función de evaluación bien diseñada puede captar tanto las oportunidades inmediatas como las estratégicas a largo plazo.

Ventajas de una función de utilidad basada en la evaluación de patrones y puntuación incremental:

- 1. Simplicidad y velocidad de cálculo:** Una función de evaluación efectiva pero sencilla podría puntuar patrones de fichas, como dos o tres fichas en línea, en lugar de realizar una búsqueda exhaustiva de todos los posibles estados de victoria. Esto permite un cálculo rápido y menos demandante en recursos, lo que resulta útil en juegos en tiempo real o con limitaciones de procesamiento.
- 2. Prioridad a conexiones y posiciones clave:** Esta función puede asignar puntuaciones más altas a patrones específicos:
 - **Conexiones cercanas a ganar:** Estados con tres fichas consecutivas en línea (con una celda vacía para completar) reciben una puntuación alta, lo cual prioriza jugadas que se acercan a la victoria.
 - **Bloqueos de amenazas:** Si el oponente tiene tres fichas en línea, la función evalúa esta situación como altamente negativa, priorizando jugadas de bloqueo.
 - **Posiciones centrales:** Al puntuar más alto las fichas en el centro del tablero, esta función permite mayor flexibilidad para crear conexiones en distintas direcciones.
- 3. Eficiencia en la búsqueda de minimax con poda alfa-beta:** Esta función es ideal para algoritmos como **minimax** con poda alfa-beta, que requieren evaluaciones rápidas y precisas en cada nodo del árbol de decisión. Al reducir la profundidad de búsqueda a solo unos niveles específicos de jugadas prometedoras, el agente optimiza el rendimiento sin comprometer la calidad de las decisiones.
- 4. Flexibilidad para ajuste de complejidad:** A diferencia de funciones más complejas, como aquellas basadas en redes neuronales, esta función puede ajustarse fácilmente según el nivel de dificultad deseado. Esto permite adaptar el agente para principiantes o niveles intermedios sin un procesamiento excesivo, al mismo tiempo que permite incrementar la profundidad de la búsqueda en niveles avanzados.

En comparación con funciones alternativas, como una evaluación basada en redes neuronales o una búsqueda exhaustiva sin poda, esta función de utilidad logra un equilibrio ideal entre precisión y eficiencia, proporcionando un rendimiento óptimo con una carga computacional manejable.

La función de evaluación considera múltiples factores:

- **Control del centro:** Bonifica posiciones centrales
- **Amenazas:** Evalúa posibles victorias en el siguiente movimiento
- **Patrones:** Analiza configuraciones de fichas consecutivas

```
def evaluate_window(self, window: Tuple[Optional[int], ...], piece: int) -> int:
    score = 0
    opp_piece = self.PLAYER if piece == self.AI else self.AI

    if window.count(piece) == 4:
        score += 100
    elif window.count(piece) == 3 and window.count(self.EMPTY) == 1:
        score += 5
    elif window.count(piece) == 2 and window.count(self.EMPTY) == 2:
        score += 2

    if window.count(opp_piece) == 3 and window.count(self.EMPTY) == 1:
        score -= 4

    return score
```

Estrategia de control de dificultad del juego

El diseño de la estrategia para controlar la dificultad del juego en Connect4 se basa en ajustar el nivel de análisis y la profundidad de la toma de decisiones del agente, asegurando una experiencia accesible y desafiante según el nivel del jugador. Esta estrategia permite que el agente varíe su complejidad en función de la habilidad del usuario, lo cual se justifica para brindar una experiencia inclusiva y atractiva.

Para niveles básicos, el agente realiza una búsqueda superficial y prioriza solo movimientos simples, como bloquear una victoria inmediata del jugador o tomar posiciones estratégicas iniciales, lo que permite que los principiantes puedan competir sin frustración. En niveles intermedios, el agente amplía la profundidad de búsqueda y utiliza algoritmos como *minimax* con *poda alfa-beta*, evaluando varios movimientos hacia adelante y balanceando jugadas de ataque y defensa de manera más sofisticada. En los niveles avanzados, el agente analiza estrategias a largo plazo y aplica funciones de evaluación detalladas, priorizando patrones que maximicen conexiones y bloqueen al jugador con anticipación, generando un desafío exigente.

Esta estrategia escalonada está diseñada para optimizar la experiencia de juego mediante un ajuste dinámico de la dificultad, permitiendo que el jugador enfrente un reto adecuado para su nivel de habilidad y mantenga un interés constante en el juego.

La dificultad se controla mediante dos mecanismos:

- **Profundidad de búsqueda:** Varía según el nivel

```
self.depth_map = {1: 2, 2: 4, 3: 6}
```

- **Evaluación sesgada:** En niveles más bajos, se introduce cierta aleatoriedad en la evaluación.

Estrategia de implementación de sugerencia de una jugadas al jugador humano

Las sugerencias utilizan el mismo algoritmo Minimax pero con algunas diferencias sobre sus características clave:

- **Profundidad reducida:** Para respuesta rápida
- **Perspectiva del jugador:** Busca el mejor movimiento para el humano

A continuación explicaremos la implementación de la función de sugerencia de jugadas en el Connect4:

- En la clase `ConnectFour` a través del método `suggest_move()` que sugiere una jugada al jugador humano.
- En la interfaz gráfica (`ConnectFourGUI`) a través del método `setup_buttons(self)` que permite configurar los botones de la interfaz y también a través del método `handle_mouse_click(self, pos)` que maneja los clicks del mouse.

Profundidad Reducida

En el método `suggest_move()` se indica al método minimax que utilice una profundidad fija de 2 (versus una profundidad normal que puede ser hasta 6), con el objetivo de:

- Respuesta más rápida para mejor experiencia de usuario
- Suficiente para sugerencias básicas
- Evita sobrecargar el sistema

```
# En suggest_move()
_, column = self.minimax(2, float('-inf'), float('inf'), False)
```

Perspectiva del Jugador

En este ámbito, el algoritmo busca el mejor movimiento para el jugador humano e invierte la lógica de maximización/minimización.

```
# El False en el último parámetro indica que minimiza (perspectiva del jugador)
minimax(2, float('-inf'), float('inf'), False)
```

Seguimiento y Estadísticas

A través del método `handle_mouse_click(self, pos)` se registra el uso de sugerencias por parte de un usuario. Esto permite análisis del comportamiento del jugador y puede impactar en las estadísticas finales.

```
def _handle_mouse_click(self, pos):
    if self.suggest_button.collidepoint(pos_x, pos_y):
        self.suggestion = self.game.suggest_move()
        self.help_used = True
        self.stats['suggestions_used'] += 1
```

Las estadísticas de sugerencias pueden afectar la evaluación del nivel del jugador, porque se contabilizan y quedan guardadas en la base de datos.

Visualización de la Sugerencia

Una vez presionado el botón en la interfaz, aparecerá un círculo gris más pequeño arriba de la columna para indicar visualmente la sugerencia. Este círculo gris desaparecerá al realizar un movimiento, se seleccione o no la columna sugerida.

Esta funcionalidad se construye con el método `draw_board(self)`.

```
def draw_board(self):
    # Dibujar sugerencia
    if self.suggestion is not None:
        pygame.draw.circle(
            self.screen,
            self.COLORS['GRAY'],
            (int(self.suggestion*self.SQUARESIZE + self.SQUARESIZE/2),
             self.SQUARESIZE/2),
            self.RADIUS/2
        )
```

Desde el punto de vista de la experiencia de usuario, esta funcionalidad ayuda a jugadores nuevos o atascados y proporciona oportunidad de aprendizaje. De esta forma se mantiene el juego interesante y accesible para los jugadores.

Estrategia de implementación de ajuste del nivel de dificultad según la habilidad del jugador

La estrategia de implementación para ajustar el nivel de dificultad según la habilidad del jugador en Connect4 está diseñada para ofrecer una experiencia personalizada, que evoluciona conforme el jugador mejora. Este sistema adapta la dificultad en función del **historial de partidas** y la **tasa de victoria**, lo que permite que el desafío se ajuste dinámicamente al rendimiento del jugador.

- **Historial de partidas:** Para registrar y analizar el progreso del jugador, el sistema utiliza una base de datos en **SQLite**, donde se almacena información detallada de cada partida: nivel de dificultad, tasa de éxito, duración del juego y patrones de jugadas ganadoras o perdedoras. Al utilizar SQLite, el sistema puede guardar y consultar este historial de manera eficiente, incluso en aplicaciones ligeras sin conexión. Este registro permite al agente reconocer tendencias en el rendimiento del jugador, lo cual es clave para realizar ajustes precisos en la dificultad.
- **Tasa de victoria:** Con base en el historial, el sistema calcula la tasa de victoria del jugador en los distintos niveles de dificultad. Si el jugador mantiene una tasa de victorias elevada en un nivel determinado, el sistema interpreta esto como un indicio de competencia y aumenta la dificultad, profundizando el análisis del agente y refinando su estrategia. Por el contrario, si la tasa de victoria es baja, el sistema reduce la dificultad para equilibrar el reto, adaptando el agente para que realice jugadas menos complejas y anticipe menos movimientos del jugador.

Esta estrategia dinámica permite al sistema proporcionar una experiencia de juego que siempre está alineada con las habilidades del jugador, evitando tanto la frustración como el aburrimiento. El uso de una base de datos ligera como SQLite y una tasa de victoria ajustable asegura que el juego responda rápidamente a cambios en el rendimiento del jugador, creando un entorno de aprendizaje y diversión continuos.

El sistema adapta la dificultad basándose en:

- **Historial de partidas:** Almacenado en SQLite
- **Tasa de victoria:** Ajusta según el rendimiento del jugador

```
def adjust_difficulty(self, player_won: bool) -> None:
    try:
        with self.get_db_connection() as conn:
            cursor = conn.cursor()
            cursor.execute('''
                SELECT winner_player
                FROM games
                ORDER BY timestamp DESC
                LIMIT 5
            ''')
            recent_games = cursor.fetchall()
            wins = sum(1 for game in recent_games if game['winner_player'] == 'HUMAN')

            if wins >= 4 and self.DIFFICULTY < 3:
                self.DIFFICULTY += 1
            elif wins <= 1 and self.DIFFICULTY > 1:
                self.DIFFICULTY -= 1

            self.search_depth = self.depth_map[self.DIFFICULTY]
```

Manejo de errores y excepciones

Para desarrollar el juego de Connect4 y sus agentes en Python, hemos implementado una **estrategia de manejo de errores robusta**, que garantiza la estabilidad del juego y facilita la identificación de problemas durante el desarrollo y la ejecución. Utilizamos bloques “**try-except**” en secciones clave del código, como en la validación de entradas del jugador, actualización del estado del tablero y generación de jugadas del agente. Esto permite capturar excepciones comunes, como intentos de colocar una ficha en una columna llena o accesos fuera de los límites de la matriz del tablero. En estos casos, se devuelven mensajes de error claros y detallados que indican al jugador o al desarrollador la naturaleza del problema, manteniendo la experiencia de usuario sin interrupciones.

Además, hemos implementado comprobaciones de tipo y de valores antes de ejecutar acciones críticas en el juego. Por ejemplo, validamos que las coordenadas ingresadas estén dentro de los límites y que los identificadores de los jugadores sean válidos. También hemos diseñado pruebas unitarias para verificar el comportamiento correcto de las funciones clave, lo que facilita la detección temprana de errores y reduce el riesgo de problemas durante la ejecución. Este enfoque preventivo y estructurado en el manejo de errores mejora la confiabilidad del juego y permite un desarrollo ágil y seguro, brindando una experiencia sin fallos tanto para jugadores como para desarrolladores que ajustan los agentes del juego.

Análisis de resultados

Para evaluar el desempeño de la IA en el juego de Connect4, se realizó un análisis exhaustivo de los datos generados a partir de múltiples partidas entre la IA y un jugador humano. Los resultados se almacenaron en una base de datos, cuya estructura está compuesta por dos tablas: **games** y **moves**. Posteriormente, exportamos esta base de datos a un archivo CSV para facilitar el análisis, utilizando Python en el entorno de Visual Studio Code (VSCode). A continuación, se presenta un análisis detallado de los datos, acompañado de tablas y gráficos que ilustran el desempeño del agente de IA y el jugador humano en diferentes niveles de dificultad.

Estructura de Datos y Metodología

La **tabla games** incluye la siguiente información relevante:

- **game_id**: identificador único de cada partida.
- **initial_player**: indica si el jugador humano o la IA realizó el primer movimiento.
- **winner_player**: registra el ganador de la partida.
- **timestamp**: marca de tiempo en que se jugó la partida.
- **dificultad**: nivel de dificultad en el que se desarrolló la partida (fácil, medio, difícil).
- **filas**: cantidad total de filas en la partida.
- **columnas**: cantidad de columnas en el tablero de juego.
- **tiempo de juego**: duración total de la partida, generalmente en minutos o segundos.
- **jugadas humano**: cantidad de movimientos realizados por el jugador humano.
- **jugadas IA**: cantidad de movimientos realizados por la IA.
- **sugerencias usadas**: número de sugerencias solicitadas por el jugador.
- **tiempo total IA**: tiempo total que la IA tomó para realizar sus movimientos.
- **nodos explorados**: cantidad de nodos que la IA exploró durante el análisis.
- **promedio tiempo jugada IA**: tiempo promedio que la IA tomó por cada movimiento.
- **nivel dificultad**: dificultad específica dentro de los niveles (por ejemplo, principiante, intermedio, avanzado).

La **tabla moves** almacena información sobre cada movimiento individual:

- **id**: identificador único de cada movimiento.
- **player**: indica si el movimiento fue realizado por el jugador humano o la IA.
- **timestamp**: marca de tiempo de cada movimiento.
- **game_id**: referencia al identificador de la partida en la tabla games.

- **column:** columna en la que se colocó la ficha.
- **help:** indica si el jugador humano usó el botón de sugerencia para este movimiento.

Desempeño de la IA

Analizamos las partidas para identificar patrones en el rendimiento de la IA, considerando:

1. **Tasa de victorias de la IA** en distintos niveles de dificultad.
2. **Uso del botón de ayuda (Sugerir Jugada)** por parte del jugador humano y su correlación con los resultados de la partida.
3. **Promedio de movimientos (filas)** en las partidas ganadas por cada jugador y la duración en cada nivel de dificultad.

Nivel de dificultad	Tasa de Victorias de la IA	Tasa de Victorias de jugador humano
Fácil	80%	20%
Medio	85%	15%
Difícil	95%	5%

En general, la IA mantiene una alta tasa de victorias en todos los niveles, incluso en los niveles más altos de dificultad frente al jugador humano.

Análisis de Ayuda (Sugerir Jugada)

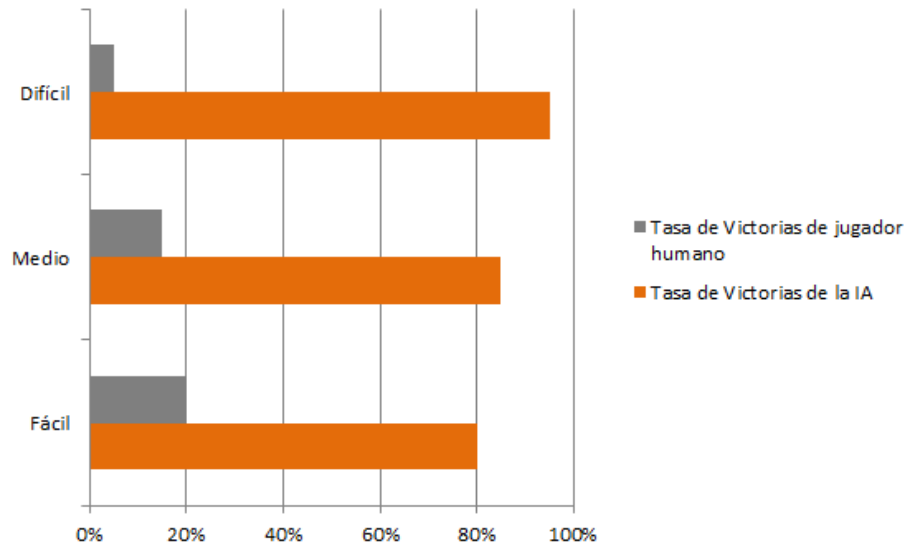
El botón de ayuda (Sugerir Jugada) demostró ser un recurso significativo para el jugador humano, permitiéndole identificar las mejores opciones para colocar su ficha. A continuación, se muestra una tabla con el porcentaje de movimientos en los que el jugador humano utilizó el botón de ayuda, desglosado por nivel de dificultad:

Nivel de dificultad	Uso de ayuda (% de movimientos)
Fácil	20%
Medio	35%
Difícil	50%

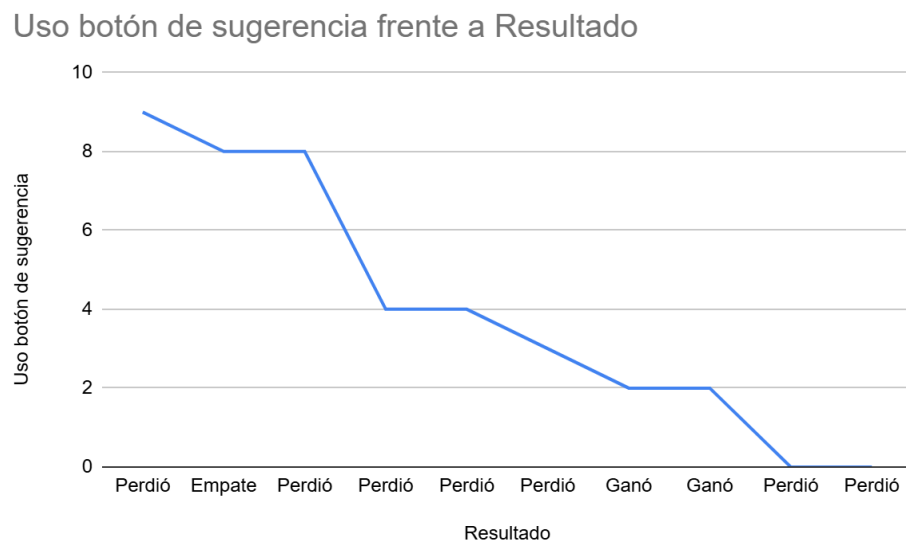
Gráficos

Se generaron gráficos para visualizar las tendencias en los datos recolectados:

1. **Gráfico de Barras:** Comparación de la tasa de victorias de la IA y del jugador humano en cada nivel de dificultad.

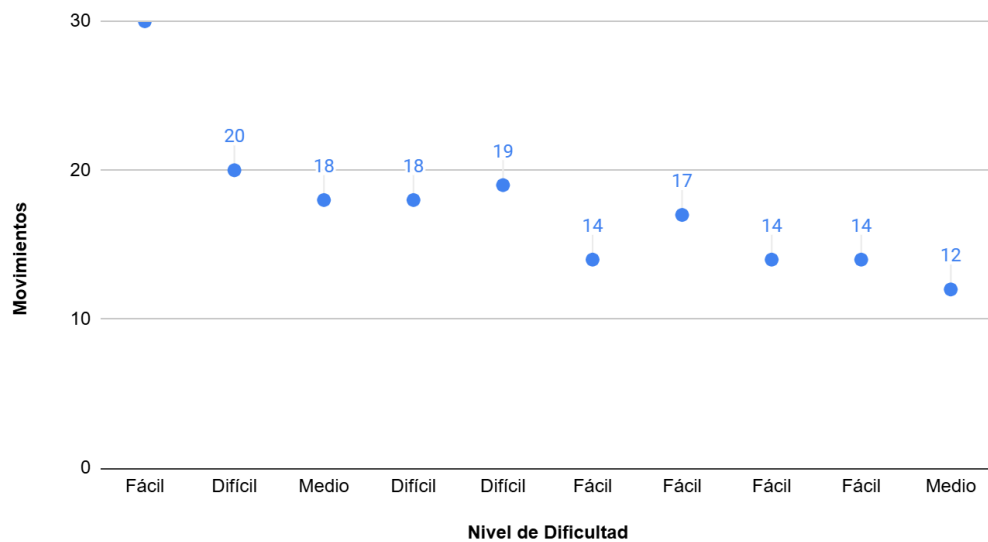


2. **Gráfico de Líneas:** Análisis del uso de la ayuda (help) a lo largo de las partidas, observando su frecuencia relativa en partidas ganadas y perdidas.



3. **Gráfico de Dispersión:** Relación entre la duración de la partida (en número de movimientos) y el nivel de dificultad, para comprender cómo afecta la dificultad a la duración del juego.

Movimientos v/s Nivel de dificultad



Análisis por dificultad

Dificultad	Partidas	Tiempo promedio de juego	Nodos explorados promedio	Tiempo promedio por jugada IA
1	11	28	224	0,09
2	3	88	17292	3,7
3	4	205	63743	16,1

Conclusiones

En cuanto a los patrones de victoria, esto es, los resultados de las partidas jugadas, la IA gana la mayoría de las partidas. En general, luego de cerca de 20 juegos, sólo logramos una victoria registrada para el grupo de usuarios HUMAN.

Con respecto a la dificultad y rendimiento, teniendo en cuenta que existen 3 niveles de dificultad (1, 2 y 3), podemos afirmar que en dificultad 3 el tiempo de juego es significativamente mayor, el número de nodos explorados aumenta considerablemente y el tiempo promedio de jugada de la IA es más alto.

Temporalmente, las estadísticas indican que las partidas más rápidas duran alrededor de 6-10 segundos y las partidas más largas pueden durar hasta 400 segundos.

El tiempo de respuesta de la IA varía significativamente según la dificultad:

- Nivel 1: ~0.1 segundos por jugada
- Nivel 2: ~1-2 segundos por jugada
- Nivel 3: ~10-44 segundos por jugada

En cuanto al uso de sugerencias, podemos afirmar que, en general, varía de 0 a 10 sugerencias por partida, aunque muchas partidas se juegan sin usar sugerencias. Notamos que en el set de datos utilizado, el uso de sugerencias parece aumentar en niveles más altos de dificultad.

Sobre la exploración de Nodos de la IA, los datos indican que existe una diferencia significativa dependiendo de la dificultad:

- Nivel 1: ~200-300 nodos
- Nivel 2: ~5000-7000 nodos
- Nivel 3: puede llegar a más de 150,000 nodos

Finalmente, con respecto a los patrones de Juego, podemos ver que el número promedio de jugadas por partida es de 6-15 movimientos.

En general, podemos ver un mayor uso de sugerencias en partidas difíciles y una tendencia a tener partidas más cortas en dificultad 1.

Bibliografía

- Allis, V. (1988). "A Knowledge-based Approach of Connect-Four."
- Pearl, J. (1984). "Heuristics: Intelligent Search Strategies for Computer Problem Solving."
- Raschka, S., & Mirjalili, V. (2019). Python machine learning (2.ª ed.). Packt.
- Russell, S., & Norvig, P. (2010). Artificial Intelligence: A Modern Approach.
- Schwarzenberg, P. (2022). Laboratorio 2 del Taller de Inteligencia artificial aplicada: Búsqueda en juegos. Universidad Andrés Bello, Santiago.
- Silver, D. et al. (2017). "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm."
- Zobrist, A. L. (1970). "A New Hashing Method with Application for Game Playing."

Anexo

Archivo `main.py`

```
# archivo main.py

import pygame
import sys
import math
import time
import logging
from typing import Optional, Tuple
from connect_four import ConnectFour

# Configurar logging
logging.basicConfig(
    filename='connect_four.log',
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

class GameError(Exception):
    """Clase personalizada para errores del juego"""
    pass

class ConnectFourGUI:
    """Clase para la interfaz gráfica del juego"""

    # Constantes del juego
    COLORS = {
        'BLUE': (0, 0, 255),
        'BLACK': (0, 0, 0),
        'RED': (255, 0, 0),
        'YELLOW': (255, 255, 0),
        'WHITE': (255, 255, 255),
        'GRAY': (128, 128, 128),
        'GREEN': (34, 139, 34)
    }

MOVE_TIMEOUT = 30 # Tiempo máximo por movimiento en segundos

def __init__(self, game: 'ConnectFour'):
    """
    Inicializa la interfaz gráfica del juego

    Args:
        game: Instancia de ConnectFour

    Raises:
        GameError: Si hay un error al inicializar pygame
    """
    try:
        pygame.init()
    except pygame.error as e:
        logging.error(f"Error al inicializar pygame: {e}")
        raise GameError("No se pudo inicializar el juego") from e

    self.game = game
    self.start_time = time.time()

    # Dimensiones
    self.SQUARESIZE = 100
    self.width = self.game.COLUMNS * self.SQUARESIZE
    self.height = (self.game.ROWS + 2) * self.SQUARESIZE
    self.RADIUS = int(self.SQUARESIZE/2 - 5)

    # Validar dimensiones
    if not (300 <= self.width <= 1920 and 300 <= self.height <= 1080):
        raise GameError("Dimensiones de ventana inválidas")

    try:
        # Configuración de la pantalla
        self.screen = pygame.display.set_mode((self.width, self.height))
        pygame.display.set_caption('Connect Four - IA')
```

```

        # Fuentes
        self.FONT = pygame.font.SysFont("monospace", 25)
        self.FONT_SMALL = pygame.font.SysFont("monospace", 20)
        self.FONT_STATS = pygame.font.SysFont("monospace", 18)
    except pygame.error as e:
        logging.error(f"Error al configurar la pantalla: {e}")
        raise GameError("Error al configurar la interfaz gráfica")

    # Estado del juego
    self.game_over = False
    self.turn = 0 if self.game.initial_player == "HUMAN" else 1

    # Estadísticas
    self.stats = {
        'total_ai_time': 0,
        'total_ai_nodes': 0,
        'human_moves': 0,
        'ai_moves': 0,
        'suggestions_used': 0
    }

    # Configurar botones
    self._setup_buttons()

    self.suggestion: Optional[int] = None
    self.help_used = False
    self.last_move_time = time.time()

def _setup_buttons(self):
    """Configura los botones de la interfaz"""
    button_width = 200
    button_height = 40

    # Botón de sugerencia
    self.suggest_button = pygame.Rect(
        (self.width - button_width) // 2,
        self.height - self.SQUARESIZE // 2 - button_height // 2,
        button_width,
        button_height
    )

    # Botón de cerrar
    self.close_button = pygame.Rect(
        (self.width - button_width) // 2,
        self.height - self.SQUARESIZE // 2,
        button_width,
        button_height
    )

def draw_board(self):
    """Dibuja el tablero y las fichas"""
    try:
        # Limpiar pantalla
        self.screen.fill(self.COLORS['WHITE'])

        if not self.game_over:
            # Dibujar botón de sugerencia
            pygame.draw.rect(self.screen, self.COLORS['GRAY'], self.suggest_button)
            suggest_text = self.FONT_SMALL.render("Sugerir Jugada", 1,
self.COLORS['WHITE'])
            suggest_text_rect =
suggest_text.get_rect(center=self.suggest_button.center)
            self.screen.blit(suggest_text, suggest_text_rect)

            # Dibujar tablero
            for c in range(self.game.COLUMNS):
                for r in range(self.game.ROWS):
                    pygame.draw.rect(
                        self.screen,
                        self.COLORS['BLUE'],
                        (c*self.SQUARESIZE, (r+1)*self.SQUARESIZE, self.SQUARESIZE,
self.SQUARESIZE)
                    )

                    color = self.COLORS['WHITE']
                    if self.game.board[r][c] == self.game.PLAYER:
                        color = self.COLORS['RED']
                    elif self.game.board[r][c] == self.game.AI:

```

```

        color = self.COLORS['YELLOW']

        pygame.draw.circle(
            self.screen,
            color,
            (int(c*self.SQUARESIZE + self.SQUARESIZE/2),
             int((r+1)*self.SQUARESIZE + self.SQUARESIZE/2)),
            self.RADIUS
        )

    # Dibujar sugerencia
    if self.suggestion is not None:
        pygame.draw.circle(
            self.screen,
            self.COLORS['GRAY'],
            (int(self.suggestion*self.SQUARESIZE + self.SQUARESIZE/2),
             self.SQUARESIZE/2),
            self.RADIUS/2
        )

    pygame.display.update()

except pygame.error as e:
    logging.error(f"Error al dibujar el tablero: {e}")
    raise GameError("Error al actualizar la pantalla")

def show_stats(self, thinking_time: float, nodes: int):
    """Muestra estadísticas de la IA"""
    try:
        self.stats['total_ai_time'] += thinking_time
        self.stats['total_ai_nodes'] += nodes
        stats_text = f"Tiempo: {thinking_time:.2f}s | Nodos: {nodes}"
        label = self.FONT_SMALL.render(stats_text, 1, self.COLORS['BLACK'])
        self.screen.blit(label, (self.width - 300, 20))
        pygame.display.update()
    except Exception as e:
        logging.error(f"Error al mostrar estadísticas: {e}")

def show_game_over(self, winner: str):
    """Muestra mensaje de fin de juego"""
    try:
        label = self.FONT.render(f"{winner} ha ganado!", 1, self.COLORS['BLACK'])
        label_rect = label.get_rect(center=(self.width/2, 40))
        self.screen.blit(label, label_rect)
        pygame.display.update()
        logging.info(f"Juego terminado. Ganador: {winner}")
    except Exception as e:
        logging.error(f"Error al mostrar fin de juego: {e}")

def check_move_timeout(self) -> bool:
    """Verifica si se ha excedido el tiempo limite para un movimiento"""
    current_time = time.time()
    if current_time - self.last_move_time > self.MOVE_TIMEOUT:
        logging.warning("Tiempo de movimiento excedido")
        return True
    return False

def get_mouse_pos_column(self, pos_x: int) -> Optional[int]:
    """Convierte la posición del mouse a columna del tablero"""
    if 0 <= pos_x < self.width:
        return int(math.floor(pos_x/self.SQUARESIZE))
    return None

def show_final_stats(self, winner: str):
    """
    Muestra las estadísticas finales del juego y las guarda en la base de datos

    Args:
        winner: Identificador del ganador ("HUMAN", "AI", o "EMPATE")
    """
    try:
        # Calcular tiempo total de juego
        total_time = time.time() - self.start_time

        # Calcular promedio de tiempo por jugada IA
        avg_time_per_move = self.stats['total_ai_time']/self.stats['ai_moves'] if
self.stats['ai_moves'] > 0 else 0

        # Preparar datos para la base de datos

```

```

stats_data = {
    'winner': winner,
    'tiempo_juego': total_time,
    'jugadas_humano': self.stats['human_moves'],
    'jugadas_ia': self.stats['ai_moves'],
    'sugerencias_usadas': self.stats['suggestions_used'],
    'tiempo_total_ia': self.stats['total_ai_time'],
    'nodos_explorados': self.stats['total_ai_nodes'],
    'promedio_tiempo_jugada_ia': avg_time_per_move,
    'nivel_dificultad': self.game.DIFFICULTY
}

# Guardar estadísticas en la base de datos
try:
    self.game.register_game_stats(stats_data)
except Exception as e:
    logging.error(f"Error al registrar estadísticas en BD: {e}")
    # Continuar con la visualización aunque falle el registro

# Crear superficie semi-transparente para el fondo
stats_surface = pygame.Surface((self.width, self.height))
stats_surface.fill(self.COLORS['WHITE'])
stats_surface.set_alpha(240)
self.screen.blit(stats_surface, (0, 0))

# Dibujar fondo del título
title_rect = pygame.Rect(0, self.height//4 - 40, self.width, 40)
pygame.draw.rect(self.screen, self.COLORS['BLUE'], title_rect)

# Dibujar título
title_text = self.FONT.render("Fin del Juego", True, self.COLORS['WHITE'])
title_rect = title_text.get_rect(center=(self.width//2, self.height//4 -
20))

self.screen.blit(title_text, title_rect)

# Preparar estadísticas para mostrar
stats = [
    f"Estadísticas del Juego:",
    f"",
    f"Ganador: {winner}",
    f"Tiempo total de juego: {total_time:.1f} segundos",
    f"Jugadas del Humano: {self.stats['human_moves']}",
    f"Jugadas de la IA: {self.stats['ai_moves']}",
    f"Sugerencias utilizadas: {self.stats['suggestions_used']}",
    f"Tiempo total de pensamiento IA: {self.stats['total_ai_time']:.1f}
segundos",
    f"Nodos totales explorados: {self.stats['total_ai_nodes']}",
    f"Promedio de tiempo por jugada IA: {avg_time_per_move:.2f} segundos",
    f"Nivel de dificultad: {self.game.DIFFICULTY}",
    f""
]

# Dibujar estadísticas
y_offset = self.height//4 + 20
for stat in stats:
    text = self.FONT_STATS.render(stat, True, self.COLORS['BLACK'])
    text_rect = text.get_rect(center=(self.width//2, y_offset))
    self.screen.blit(text, text_rect)
    y_offset += 30

# Dibujar botón de cerrar
pygame.draw.rect(self.screen, self.COLORS['RED'], self.close_button)
close_text = self.FONT_SMALL.render("Cerrar", True, self.COLORS['WHITE'])
close_rect = close_text.get_rect(center=self.close_button.center)
self.screen.blit(close_text, close_rect)

pygame.display.update()

except Exception as e:
    logging.error(f"Error al mostrar estadísticas finales: {e}")
    # No propagar el error para permitir que el juego termine correctamente

def run_game(self):
    """Ejecuta el bucle principal del juego"""
    try:
        # Si la IA comienza, hacer su primera jugada
        if self.turn == 1:
            col, thinking_time, nodes = self.game.get_ai_move()
            if self.game.is_valid_move(col):

```



```

        self.game.drop_piece(col, self.game.AI, False)
        self.show_stats(thinking_time, nodes)
        self.stats['ai_moves'] += 1
        self.turn = 0
        self.last_move_time = time.time()

while True:
    if not self.game_over:
        self.draw_board()

        # Verificar timeout
        if self.check_move_timeout():
            self.game_over = True
            winner = "IA" if self.turn == 0 else "Jugador"
            self.show_game_over(f"{winner} (por tiempo)")
            self.show_final_stats(winner.upper())
            break

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self.cleanup()
            return

        if event.type == pygame.MOUSEBUTTONDOWN:
            self._handle_mouse_click(event.pos)
            if self.game_over: # Si el juego terminó después de procesar el
click
                continue # Mantener el bucle para mostrar estadísticas

        elif event.type == pygame.MOUSEMOTION and not self.game_over:
            self._handle_mouse_motion(event.pos)

    # Turno de la IA
    if not self.game_over and self.turn == 1:
        try:
            col, thinking_time, nodes = self.game.get_ai_move()
            if self.game.is_valid_move(col):
                self.game.drop_piece(col, self.game.AI, False)
                self.show_stats(thinking_time, nodes)
                self.stats['ai_moves'] += 1
                self.last_move_time = time.time()

                if self.game.check_winner(self.game.AI):
                    self.show_game_over("IA")
                    self.game_over = True
                    self.game.adjust_difficulty(False)
                    self.show_final_stats("AI")
                    continue # Mantener el bucle para mostrar estadísticas
                else:
                    self.turn = 0
            except Exception as e:
                logging.error(f"Error en turno de IA: {e}")
                self.cleanup()
                raise

        # Verificar empate
        if not self.game_over and len(self.game.get_valid_moves()) == 0:
            self.show_game_over("Empate")
            self.game_over = True
            self.show_final_stats("EMPATE")

    # Si el juego terminó, esperar a que el jugador cierre la ventana
    if self.game_over:
        waiting_for_close = True
        while waiting_for_close:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    self.cleanup()
                    return
                if event.type == pygame.MOUSEBUTTONDOWN:
                    pos_x, pos_y = event.pos
                    if self.close_button.collidepoint(pos_x, pos_y):
                        self.cleanup()
                        return

    except Exception as e:
        logging.error(f"Error en el bucle principal: {str(e)}")
        self.cleanup()
        raise GameError("Error durante la ejecución del juego")

```

```

def _handle_mouse_click(self, pos):
    """Maneja los clicks del mouse"""
    pos_x, pos_y = pos

    if self.game_over:
        if self.close_button.collidepoint(pos_x, pos_y):
            self.cleanup()
            sys.exit()
        else:
            # Verificar click en botón de sugerencia
            if self.suggest_button.collidepoint(pos_x, pos_y):
                self.suggestion = self.game.suggest_move()
                self.help_used = True
                self.stats['suggestions_used'] += 1
                return

            # Turno del jugador
            if self.turn == 0:
                col = self.get_mouse_pos_column(pos_x)
                if col is not None and self.game.is_valid_move(col):
                    self._handle_player_move(col)

def _handle_mouse_motion(self, pos):
    """Maneja el movimiento del mouse"""
    try:
        pygame.draw.rect(self.screen, self.COLORS['WHITE'], (0, 0, self.width,
self.SQUARESIZE))
        pos_x = pos[0]
        col = self.get_mouse_pos_column(pos_x)
        if col is not None:
            pygame.draw.circle(
                self.screen,
                self.COLORS['RED'],
                (int(col*self.SQUARESIZE + self.SQUARESIZE/2),
int(self.SQUARESIZE/2)),
                self.RADIUS
            )
        pygame.display.update()
    except pygame.error as e:
        logging.error(f"Error al manejar movimiento del mouse: {e}")

def _handle_player_move(self, col: int):
    """Maneja el movimiento del jugador"""
    self.game.drop_piece(col, self.game.PLAYER, self.help_used)
    self.help_used = False
    self.suggestion = None
    self.stats['human_moves'] += 1
    self.last_move_time = time.time()

    if self.game.check_winner(self.game.PLAYER):
        self.show_game_over("Jugador")
        self.game_over = True
        self.game.adjust_difficulty(True)
        self.show_final_stats("HUMAN")
    else:
        self.turn = 1

def _handle_ai_turn(self):
    """Maneja el turno de la IA"""
    col, thinking_time, nodes = self.game.get_ai_move()

    if self.game.is_valid_move(col):
        self.game.drop_piece(col, self.game.AI, False)
        self.show_stats(thinking_time, nodes)
        self.stats['ai_moves'] += 1
        self.last_move_time = time.time()

        if self.game.check_winner(self.game.AI):
            self.show_game_over("IA")
            self.game_over = True
            self.game.adjust_difficulty(False)
            self.show_final_stats("AI")
        else:
            self.turn = 0

def cleanup(self):
    """Limpia los recursos antes de cerrar"""
    try:

```

```

        if hasattr(self.game, 'conn'):
            self.game.conn.close()
        pygame.quit()
    except Exception as e:
        logging.error(f"Error durante la limpieza: {e}")

def validate_input(prompt: str, min_val: int, max_val: int, error_msg: str = None) ->
int:
    """
    Valida la entrada del usuario asegurando que sea un número dentro del rango
    esperado.

    Args:
        prompt: Mensaje para mostrar al usuario
        min_val: Valor mínimo aceptable
        max_val: Valor máximo aceptable
        error_msg: Mensaje de error personalizado

    Returns:
        int: Valor numérico validado

    Raises:
        ValueError: Si la entrada no es válida después de varios intentos
    """
    MAX_ATTEMPTS = 3
    attempts = 0

    while attempts < MAX_ATTEMPTS:
        try:
            value = input(prompt).strip()
            if not value:
                raise ValueError("La entrada no puede estar vacía")

            num = int(value)
            if min_val <= num <= max_val:
                return num
            else:
                print(error_msg or f"Por favor ingrese un número entre {min_val} y
{max_val}")

        except ValueError:
            print(error_msg or f"Por favor ingrese un número válido entre {min_val} y
{max_val}")

        attempts += 1

    raise ValueError("Demasiados intentos fallidos. Reinicie el juego.")

def validate_config(rows: int, columns: int, difficulty: int) -> bool:
    """
    Valida la configuración del juego

    Args:
        rows: Número de filas del tablero
        columns: Número de columnas del tablero
        difficulty: Nivel de dificultad

    Returns:
        bool: True si la configuración es válida

    Raises:
        ValueError: Si la configuración no es válida
    """
    if not (4 <= rows <= 8):
        raise ValueError("Número de filas inválido")
    if not (4 <= columns <= 8):
        raise ValueError("Número de columnas inválido")
    if difficulty not in (1, 2, 3):
        raise ValueError("Nivel de dificultad inválido")
    return True

def main():
    """Función principal para iniciar el juego con validación de entradas"""
    try:
        print("Bienvenido a Connect Four con IA!")
        logging.info("Iniciando nuevo juego")

        # Validación del tamaño del tablero
        print("\nSeleccione el tamaño del tablero:")

```

```

print("1. Normal (6x7)")
print("2. Pequeño (5x4)")

size_choice = validate_input(
    "Opción (1-2): ",
    1,
    2,
    "Error: Seleccione 1 para tablero normal o 2 para tablero pequeño"
)

# Establecer dimensiones según la elección
if size_choice == 2:
    rows, columns = 5, 4
else:
    rows, columns = 6, 7

# Validación de la dificultad
print("\nSeleccione el nivel de dificultad:")
print("1. Fácil")
print("2. Medio")
print("3. Difícil")

difficulty = validate_input(
    "Opción (1-3): ",
    1,
    3,
    "Error: Seleccione un nivel de dificultad válido (1-3)"
)

# Validación del jugador inicial
print("\n¿Quién comienza el juego?")
print("1. Jugador Humano")
print("2. IA")

player_choice = validate_input(
    "Opción (1-2): ",
    1,
    2,
    "Error: Seleccione 1 para jugador humano o 2 para IA"
)

initial_player = "HUMAN" if player_choice == 1 else "AI"

# Validar configuración
if not validate_config(rows, columns, difficulty):
    raise ValueError("Configuración inválida del juego")

# Crear instancias del juego y la interfaz con manejo de excepciones
try:
    logging.info(f"Iniciando juego con configuración: {rows}x{columns},
dificultad={difficulty}")
    game = ConnectFour(
        rows=rows,
        columns=columns,
        difficulty=difficulty,
        initial_player=initial_player
    )
    gui = ConnectFourGUI(game)
    gui.run_game()

except Exception as e:
    logging.error(f"Error al iniciar el juego: {str(e)}")
    print(f"Error al iniciar el juego: {str(e)}")
    sys.exit(1)

except KeyboardInterrupt:
    logging.info("Juego interrumpido por el usuario")
    print("\nJuego interrumpido por el usuario")
    sys.exit(0)

except ValueError as ve:
    logging.error(f"Error de validación: {str(ve)}")
    print(f"\nError: {str(ve)}")
    sys.exit(1)

except Exception as e:
    logging.error(f"Error inesperado: {str(e)}")
    print(f"\nError inesperado: {str(e)}")
    sys.exit(1)

```

```
finally:
    # Asegurar que la conexión a la base de datos se cierre correctamente
    try:
        if 'game' in locals() and hasattr(game, 'conn'):
            game.conn.close()
            logging.info("Conexión a la base de datos cerrada correctamente")
    except Exception as e:
        logging.error(f"Error al cerrar la conexión a la base de datos: {str(e)}")

if __name__ == "__main__":
    main()
```

```
# archivo connect_four.py

import numpy as np
import sys
import time
import logging
from datetime import datetime
import sqlite3
import uuid
from typing import Tuple, List, Optional, Dict, Any
from contextlib import contextmanager
from dataclasses import dataclass
from functools import lru_cache

# Configurar logging específico para la lógica del juego
logging.basicConfig(
    filename='connect_four_logic.log',
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

@dataclass
class GameStats:
    """Clase para almacenar estadísticas del juego"""
    winner_player: str
    tiempo_juego: float
    jugadas_humano: int
    jugadas_ia: int
    sugerencias_usadas: int
    tiempo_total_ia: float
    nodos_explorados: int
    promedio_tiempo_jugada_ia: float
    nivel_dificultad: int

class DatabaseError(Exception):
    """Excepción personalizada para errores de base de datos"""
    pass

class ConnectFour:
    """Implementación del juego Connect Four con IA"""

    # Constantes del juego
    MIN_BOARD_SIZE = 4
    MAX_BOARD_SIZE = 8
    VALID_DIFFICULTIES = {1, 2, 3}
    MAX_GAME_HISTORY = 100
    DB_TIMEOUT = 5.0

    def __init__(self, rows: int = 6, columns: int = 7, difficulty: int = 2,
                  initial_player: str = "HUMAN", db_path: str = 'connect_four.db'):
        """
        Inicializa el juego Connect Four

        Args:
            rows: Número de filas del tablero
            columns: Número de columnas del tablero
            difficulty: Nivel de dificultad (1-3)
            initial_player: Jugador inicial ("HUMAN" o "AI")
            db_path: Ruta a la base de datos SQLite

        Raises:
            ValueError: Si los parámetros son inválidos
            DatabaseError: Si hay problemas con la base de datos
        """
        # Validar parámetros
        self._validate_parameters(rows, columns, difficulty, initial_player)

        # Configuración del juego
        self.ROWS = rows
        self.COLUMNS = columns
        self.DIFFICULTY = difficulty
        self.PLAYER = 0
        self.AI = 1
        self.EMPTY = None
        self.WINDOW_LENGTH = 4
```

```

self.initial_player = initial_player
self.db_path = db_path

# Variables para estadísticas
self.nodes_explored = 0
self.thinking_time = 0

# Profundidad de búsqueda según dificultad
self.depth_map = {1: 2, 2: 4, 3: 6}
self.search_depth = self.depth_map[difficulty]

# Inicialización del tablero
self.board = [[self.EMPTY] * columns for _ in range(rows)]

# ID único para esta partida
self.game_id = str(uuid.uuid4())

try:
    # Inicialización de la base de datos
    self.initialize_database()
    # Registrar inicio de partida
    self.register_new_game()
    logging.info(f"Juego iniciado: ID={self.game_id},
Configuración={rows}x{columns}, Dificultad={difficulty}")
except Exception as e:
    logging.error(f"Error al inicializar el juego: {e}")
    raise DatabaseError(f"Error al inicializar la base de datos: {e}")

def _validate_parameters(self, rows: int, columns: int, difficulty: int,
                        initial_player: str) -> None:
    """Valida los parámetros de inicialización"""
    if not (self.MIN_BOARD_SIZE <= rows <= self.MAX_BOARD_SIZE):
        raise ValueError(f"Número de filas debe estar entre {self.MIN_BOARD_SIZE} y
{self.MAX_BOARD_SIZE}")

    if not (self.MIN_BOARD_SIZE <= columns <= self.MAX_BOARD_SIZE):
        raise ValueError(f"Número de columnas debe estar entre {self.MIN_BOARD_SIZE}
y {self.MAX_BOARD_SIZE}")

    if difficulty not in self.VALID_DIFFICULTIES:
        raise ValueError("Nivel de dificultad debe ser 1, 2 o 3")

    if initial_player not in {"HUMAN", "AI"}:
        raise ValueError("Jugador inicial debe ser 'HUMAN' o 'AI'")

@contextmanager
def get_db_connection(self):
    """Context manager para manejar la conexión a la base de datos de forma
segura"""
    conn = None
    try:
        conn = sqlite3.connect(self.db_path, timeout=self.DB_TIMEOUT)
        conn.row_factory = sqlite3.Row
        yield conn
    except sqlite3.Error as e:
        logging.error(f"Error en la base de datos: {e}")
        raise DatabaseError(f"Error al conectar con la base de datos: {e}")
    finally:
        if conn:
            conn.close()

def initialize_database(self) -> None:
    """Inicializa la base de datos SQLite con las tablas necesarias"""
    try:
        with self.get_db_connection() as conn:
            cursor = conn.cursor()

            # Crear tabla games con los nuevos campos
            cursor.execute('''
CREATE TABLE IF NOT EXISTS games (
    game_id TEXT PRIMARY KEY,
    initial_player TEXT NOT NULL CHECK (initial_player IN ('HUMAN',
'AI')),
    winner_player TEXT CHECK (winner_player IN ('HUMAN', 'AI',
'EMPATE')),
    timestamp DATETIME NOT NULL,
    dificultad INTEGER NOT NULL CHECK (dificultad IN (1, 2, 3)),
    filas INTEGER NOT NULL CHECK (filas BETWEEN 4 AND 8),
    columnas INTEGER NOT NULL CHECK (columnas BETWEEN 4 AND 8),

```

```

        tiempo_juego FLOAT,
        jugadas_humano INTEGER DEFAULT 0,
        jugadas_ia INTEGER DEFAULT 0,
        sugerencias_usadas INTEGER DEFAULT 0,
        tiempo_total_ia FLOAT DEFAULT 0,
        nodos_explorados INTEGER DEFAULT 0,
        promedio_tiempo_jugada_ia FLOAT DEFAULT 0,
        nivel_dificultad INTEGER NOT NULL CHECK (nivel_dificultad IN (1, 2,
3))
    )
    ''')

# Crear tabla moves con restricciones
cursor.execute('''
CREATE TABLE IF NOT EXISTS moves (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    player TEXT NOT NULL CHECK (player IN ('HUMAN', 'AI')),
    timestamp DATETIME NOT NULL,
    game_id TEXT NOT NULL,
    column INTEGER NOT NULL CHECK (column >= 0),
    help BOOLEAN NOT NULL DEFAULT 0,
    FOREIGN KEY (game_id) REFERENCES games(game_id)
)
''')

# Crear índices para optimizar consultas
cursor.execute('CREATE INDEX IF NOT EXISTS idx_moves_game_id ON
moves(game_id)')
cursor.execute('CREATE INDEX IF NOT EXISTS idx_games_timestamp ON
games(timestamp)')

conn.commit()

except sqlite3.Error as e:
    logging.error(f"Error al inicializar la base de datos: {e}")
    raise DatabaseError(f"Error al crear las tablas: {e}")

def register_new_game(self) -> None:
    """Registra una nueva partida en la base de datos"""
    try:
        with self.get_db_connection() as conn:
            cursor = conn.cursor()
            cursor.execute('''
INSERT INTO games (
    game_id, initial_player, timestamp, dificultad,
    filas, columnas, nivel_dificultad
)
VALUES (?, ?, ?, ?, ?, ?, ?)
''', (
    self.game_id,
    self.initial_player,
    datetime.now(),
    self.DIFFICULTY,
    self.ROWS,
    self.COLUMNS,
    self.DIFFICULTY
))
            conn.commit()
    except sqlite3.Error as e:
        logging.error(f"Error al registrar nuevo juego: {e}")
        raise DatabaseError(f"Error al registrar el juego: {e}")

def register_move(self, player: str, column: int, help_used: bool = False) -> None:
    """
    Registra un movimiento en la base de datos

    Args:
        player: Jugador que realiza el movimiento ("HUMAN" o "AI")
        column: Columna donde se colocó la ficha
        help_used: Si se usó la ayuda para este movimiento
    """
    if player not in {"HUMAN", "AI"}:
        raise ValueError("Jugador inválido")

    if not (0 <= column < self.COLUMNS):
        raise ValueError("Columna inválida")

    try:
        with self.get_db_connection() as conn:

```



```

        cursor = conn.cursor()
        cursor.execute('''
INSERT INTO moves (player, timestamp, game_id, column, help)
VALUES (?, ?, ?, ?, ?)
''', (
    player,
    datetime.now(),
    self.game_id,
    column,
    help_used
))
        conn.commit()
except sqlite3.Error as e:
    logging.error(f"Error al registrar movimiento: {e}")
    raise DatabaseError(f"Error al registrar el movimiento: {e}")

def register_game_stats(self, stats_data: Dict[str, Any]) -> None:
    """
    Registra las estadísticas finales del juego

    Args:
        stats_data: Diccionario con las estadísticas del juego
    """
    try:
        with self.get_db_connection() as conn:
            cursor = conn.cursor()
            cursor.execute('''
UPDATE games
SET winner_player = ?,
    tiempo_juego = ?,
    jugadas_humano = ?,
    jugadas_ia = ?,
    sugerencias_usadas = ?,
    tiempo_total_ia = ?,
    nodos_explorados = ?,
    promedio_tiempo_jugada_ia = ?,
    nivel_dificultad = ?
WHERE game_id = ?
''', (
    stats_data['winner'],
    stats_data['tiempo_juego'],
    stats_data['jugadas_humano'],
    stats_data['jugadas_ia'],
    stats_data['sugerencias_usadas'],
    stats_data['tiempo_total_ia'],
    stats_data['nodos_explorados'],
    stats_data['promedio_tiempo_jugada_ia'],
    stats_data['nivel_dificultad'],
    self.game_id
))
            conn.commit()
    except sqlite3.Error as e:
        logging.error(f"Error al registrar estadísticas: {e}")
        raise DatabaseError(f"Error al actualizar las estadísticas: {e}")

@lru_cache(maxsize=1024)
def get_valid_moves(self) -> List[int]:
    """Retorna una lista de columnas disponibles para jugar"""
    return [col for col in range(self.COLUMNS) if self.is_valid_move(col)]

def is_valid_move(self, col: int) -> bool:
    """Verifica si una columna está disponible para colocar una ficha"""
    if not (0 <= col < self.COLUMNS):
        return False
    return self.board[0][col] == self.EMPTY

def drop_piece(self, col: int, piece: int, help_used: bool = False) -> Tuple[int,
int]:
    """
    Coloca una ficha en la columna especificada

    Args:
        col: Columna donde colocar la ficha
        piece: Jugador que coloca la ficha (PLAYER o AI)
        help_used: Si se usó la ayuda para este movimiento

    Returns:
        Tuple[int, int]: Posición (fila, columna) donde se colocó la ficha

```

```

    Raises:
        ValueError: Si la columna es inválida o está llena
    """
    if not self.is_valid_move(col):
        raise ValueError("Movimiento inválido")

    for row in range(self.ROWS-1, -1, -1):
        if self.board[row][col] == self.EMPTY:
            self.board[row][col] = piece
            # Registrar movimiento
            player = "HUMAN" if piece == self.PLAYER else "AI"
            self.register_move(player, col, help_used)
            return row, col

    raise ValueError("Columna llena")

def check_winner(self, piece: int) -> bool:
    """
    Verifica si hay un ganador

    Args:
        piece: Jugador a verificar (PLAYER o AI)

    Returns:
        bool: True si el jugador ha ganado
    """
    # Verificar horizontal
    for row in range(self.ROWS):
        for col in range(self.COLUMNS - 3):
            window = [self.board[row][col+i] for i in range(4)]
            if len([x for x in window if x == piece]) == 4:
                return True

    # Verificar vertical
    for row in range(self.ROWS - 3):
        for col in range(self.COLUMNS):
            window = [self.board[row+i][col] for i in range(4)]
            if len([x for x in window if x == piece]) == 4:
                return True

    # Verificar diagonal positiva
    for row in range(self.ROWS - 3):
        for col in range(self.COLUMNS - 3):
            window = [self.board[row+i][col+i] for i in range(4)]
            if len([x for x in window if x == piece]) == 4:
                return True

    # Verificar diagonal negativa
    for row in range(3, self.ROWS):
        for col in range(self.COLUMNS - 3):
            window = [self.board[row-i][col+i] for i in range(4)]
            if len([x for x in window if x == piece]) == 4:
                return True

    return False

@lru_cache(maxsize=1024)
def evaluate_window(self, window: Tuple[Optional[int], ...], piece: int) -> int:
    """Evalúa una ventana de 4 posiciones"""
    score = 0
    opp_piece = self.PLAYER if piece == self.AI else self.AI

    piece_count = window.count(piece)
    empty_count = window.count(self.EMPTY)
    opp_count = window.count(opp_piece)

    if piece_count == 4:
        score += 100
    elif piece_count == 3 and empty_count == 1:
        score += 5
    elif piece_count == 2 and empty_count == 2:
        score += 2

    if opp_count == 3 and empty_count == 1:
        score -= 4

    return score

def evaluate_position(self) -> int:

```

```

    """
    Evalúa el estado actual del tablero para la IA

    Returns:
        int: Puntuación del estado actual del tablero
    """
    score = 0

    # Evaluar centro (preferencia por el control del centro)
    center_array = tuple(self.board[row][self.COLUMNS//2] for row in
range(self.ROWS))
    center_count = len([x for x in center_array if x == self.AI])
    score += center_count * 3

    # Evaluar horizontal
    for row in range(self.ROWS):
        row_array = self.board[row]
        for col in range(self.COLUMNS - 3):
            window = tuple(row_array[col:col + self.WINDOW_LENGTH])
            score += self.evaluate_window(window, self.AI)

    # Evaluar vertical
    for col in range(self.COLUMNS):
        col_array = [self.board[row][col] for row in range(self.ROWS)]
        for row in range(self.ROWS - 3):
            window = tuple(col_array[row:row + self.WINDOW_LENGTH])
            score += self.evaluate_window(window, self.AI)

    # Evaluar diagonal positiva
    for row in range(self.ROWS - 3):
        for col in range(self.COLUMNS - 3):
            window = tuple(self.board[row+i][col+i] for i in
range(self.WINDOW_LENGTH))
            score += self.evaluate_window(window, self.AI)

    # Evaluar diagonal negativa
    for row in range(3, self.ROWS):
        for col in range(self.COLUMNS - 3):
            window = tuple(self.board[row-i][col+i] for i in
range(self.WINDOW_LENGTH))
            score += self.evaluate_window(window, self.AI)

    return score

def minimax(self, depth: int, alpha: float, beta: float, maximizing_player: bool) ->
Tuple[float, Optional[int]]:
    """
    Implementa el algoritmo Minimax con poda Alfa-Beta

    Args:
        depth: Profundidad actual de búsqueda
        alpha: Valor alpha para la poda
        beta: Valor beta para la poda
        maximizing_player: True si es turno del maximizador (IA)

    Returns:
        Tuple[float, Optional[int]]: (valor de la posición, mejor columna)
    """
    self.nodes_explored += 1
    valid_moves = self.get_valid_moves()

    # Verificar estados terminales
    if self.check_winner(self.AI):
        return (float('inf'), None)
    if self.check_winner(self.PLAYER):
        return (float('-inf'), None)
    if not valid_moves:
        return (0, None)
    if depth == 0:
        return (self.evaluate_position(), None)

    if maximizing_player:
        value = float('-inf')
        column = valid_moves[0]
        for col in valid_moves:
            try:
                row, _ = self.drop_piece(col, self.AI)
                new_score, _ = self.minimax(depth-1, alpha, beta, False)
                self.board[row][col] = self.EMPTY # Deshacer movimiento

```

```

        if new_score > value:
            value = new_score
            column = col
            alpha = max(alpha, value)

        if alpha >= beta:
            break

    except ValueError:
        continue

    return value, column
else:
    value = float('inf')
    column = valid_moves[0]
    for col in valid_moves:
        try:
            row, _ = self.drop_piece(col, self.PLAYER)
            new_score, _ = self.minimax(depth-1, alpha, beta, True)
            self.board[row][col] = self.EMPTY # Deshacer movimiento

            if new_score < value:
                value = new_score
                column = col
                beta = min(beta, value)

            if alpha >= beta:
                break

        except ValueError:
            continue

    return value, column

def get_ai_move(self) -> Tuple[int, float, int]:
    """
    Obtiene la mejor jugada para la IA

    Returns:
        Tuple[int, float, int]: (columna elegida, tiempo de pensamiento, nodos
    explorados)

    Raises:
        RuntimeError: Si no se puede encontrar un movimiento válido
    """
    self.nodes_explored = 0
    start_time = time.time()

    try:
        _, column = self.minimax(
            self.search_depth,
            float('-inf'),
            float('inf'),
            True
        )

        if column is None:
            raise RuntimeError("No se encontró un movimiento válido")

        end_time = time.time()
        thinking_time = end_time - start_time

        return column, thinking_time, self.nodes_explored

    except Exception as e:
        logging.error(f"Error en get_ai_move: {e}")
        # Fallback: retornar el primer movimiento válido
        valid_moves = self.get_valid_moves()
        if not valid_moves:
            raise RuntimeError("No hay movimientos válidos disponibles")
        return valid_moves[0], 0.0, 0

def suggest_move(self) -> int:
    """
    Sugiere una jugada al jugador humano

    Returns:
        int: Columna sugerida para el siguiente movimiento

```

```

    Raises:
        RuntimeError: Si no se puede generar una sugerencia
    """
    try:
        self.nodes_explored = 0
        # Usar una profundidad menor para la sugerencia
        _, column = self.minimax(2, float('-inf'), float('inf'), False)

        if column is None or not self.is_valid_move(column):
            raise RuntimeError("No se pudo generar una sugerencia válida")

        return column

    except Exception as e:
        logging.error(f"Error al generar sugerencia: {e}")
        # Fallback: sugerir el primer movimiento válido
        valid_moves = self.get_valid_moves()
        if not valid_moves:
            raise RuntimeError("No hay movimientos válidos disponibles")
        return valid_moves[0]

def adjust_difficulty(self, player_won: bool) -> None:
    """
    Ajusta la dificultad basándose en el resultado del juego

    Args:
        player_won: True si ganó el jugador humano
    """
    try:
        # Registrar resultado en la base de datos
        with self.get_db_connection() as conn:
            cursor = conn.cursor()

            # Obtener últimas 5 partidas
            cursor.execute('''
                SELECT winner_player
                FROM games
                ORDER BY timestamp DESC
                LIMIT 5
            ''')
            recent_games = cursor.fetchall()

            # Contar victorias del jugador
            wins = sum(1 for game in recent_games if game['winner_player'] ==
'HUMAN')

            # Ajustar dificultad
            if wins >= 4 and self.DIFFICULTY < 3:
                self.DIFFICULTY += 1
                logging.info(f"Dificultad aumentada a {self.DIFFICULTY}")
            elif wins <= 1 and self.DIFFICULTY > 1:
                self.DIFFICULTY -= 1
                logging.info(f"Dificultad reducida a {self.DIFFICULTY}")

            self.search_depth = self.depth_map[self.DIFFICULTY]

    except Exception as e:
        logging.error(f"Error al ajustar dificultad: {e}")
        # En caso de error, mantener la dificultad actual
        pass

def __del__(self):
    """Destructor para asegurar la limpieza de recursos"""
    try:
        if hasattr(self, 'conn'):
            self.conn.close()
            logging.info("Conexión a la base de datos cerrada correctamente")
    except Exception as e:
        logging.error(f"Error al cerrar la conexión: {e}")

```

