Práctica 2: Reversi

Laboratorio de Inteligencia Artificial – Curso 2023-2024

Fecha de publicación: 2023-10-11
Fecha de entrega parcial en Moodle: todos los grupos el viernes 2023-11-03, 23:59
Fecha del torneo final: 2023-11-13
Fecha de entrega en Moodle: todos los grupos el viernes 2023-11-17, 23:59

---

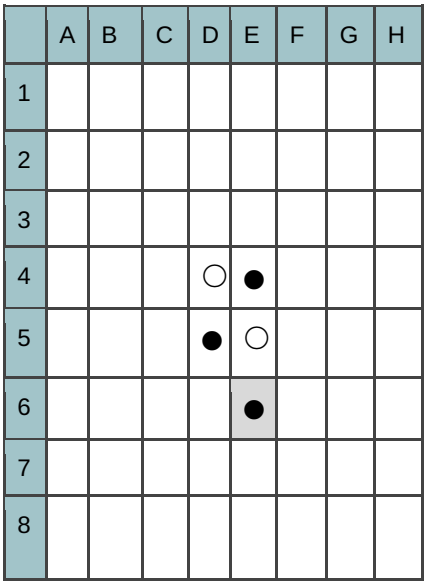
1. El juego del Reversi

Reversi (también conocido como **Othello** o **Yang)** es un juego de estrategia de dos jugadores. Se juega en un tablero de 8×8 con la siguiente configuración inicial:

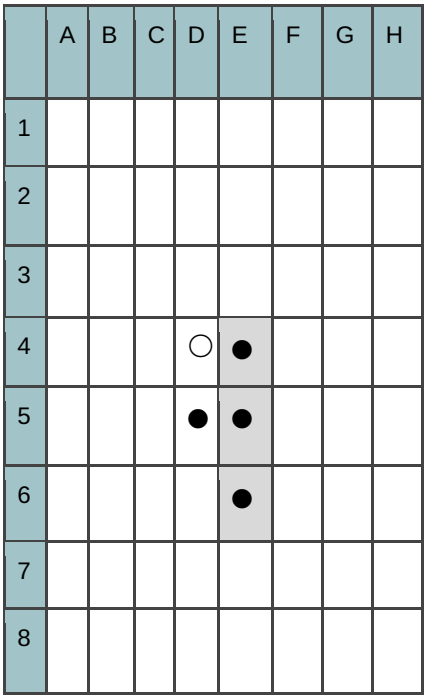
	A	B	C	D	E	F	G	H
1								
2								
3								
4				○	●			
5				●	○			
6								
7								
8								

Los dos jugadores, representados por las fichas negras y blancas, alternan turnos empezando las negras. En cada turno, el jugador coloca una pieza de su color sobre el tablero de forma que capture alguna de las fichas del contrario. Se realiza una captura cuando la nueva ficha, junto con alguna o algunas fichas del mismo color en el tablero, encierra hileras de fichas del contrario a lo largo de las filas, las columnas o las diagonales del tablero. Las fichas capturadas son sustituidas por fichas del color del jugador que realiza la captura. En caso de que el jugador que tiene el turno no pueda capturar ninguna ficha del contrario, el turno pasa al otro jugador. El juego termina cuando ninguno de los jugadores puede capturar fichas del contrario o el tablero está completamente ocupado. El ganador de la partida es el jugador que tiene más fichas en el tablero.

En el siguiente ejemplo el jugador coloca la ficha ● en E6:



Y captura la ficha ○ que está en E5, quedando el tablero de la siguiente manera:



Podéis jugar online en https://webgamesonline.com/reversi/ para familiarizados con sus reglas y dinámicas.

2. Implementación de Reversi y de las funciones de evaluación

El código proporcionado requiere la versión 3.7 de Python, aunque recomendamos usar la 3.10.

La infraestructura para un juego entre dos jugadores está en el módulo `game.py`. Las clases y funciones específicas para Reversi están implementadas en el fichero `reversi.py`. **Estos ficheros no deben ser modificados**.

Las estrategias para el juego (jugador aleatorio, jugador manual, jugador minimax) están implementadas en **`strategy.py`**. **Este es el fichero que se debe modificar para implementar un jugador que realice una búsqueda minimax con poda alfa-beta (ver parte 1 de la evaluación).**

En el módulo **`heuristic.py`** se ha implementado un ejemplo de función de evaluación de estados del juego, la cual es utilizada en estrategias de búsqueda. Esta función es general al no usar conocimiento del juego en cuestión, por lo que tiene bastante margen de mejora, sin embargo, por ello mismo se puede usar la misma heurística para varios juegos.

Podéis encontrar ejemplos de cómo jugar en el fichero `demo_reversi.py`.
En los distintos ejemplos que podéis ver en este fichero, se observa lo siguiente:

- La lógica del juego (en este caso, `Reversi`) se abstrae dentro de una clase de tipo `TwoPlayerGameState`, que almacena la información necesaria para juegos de dos jugadores (además de la lógica del juego, el tablero, el jugador inicial, si se ha terminado el juego, las puntuaciones, etc.)
- La lógica de una partida de dos jugadores se maneja a través de la clase `TwoPlayerMatch`, que recibe un estado de tipo `TwoPlayerGameState` y es el encargado de dar turno a cada jugador, imprimir más o menos información por pantalla (según el valor de la variable `verbose`), comprobar que ningún jugador invierta mucho tiempo en cada turno (ver variable `max_seconds_per_move`), así como de controlar si la interfaz será textual o gráfica (variable `gui`).

Para terminar de entender el código entregado, se explican a continuación las clases y funcionalidades más importantes:

- El parámetro `gui` de la clase `TwoPlayerMatch` sirve para decidir si el juego se mostrará como texto en la terminal o si se creará una ventana gráfica. En cualquiera de los dos casos se pueden usar cualquiera de los tipos de jugadores (manual o automático). Si el jugador es manual, el comportamiento es diferente: en el caso de que se haya elegido jugar con la interfaz gráfica (el valor de la variable `gui` es `True`), entonces el jugador tendrá que hacer clic en la ventana; en modo texto (`gui=False`) realizará su jugada por la terminal.
- La variable `verbose > 0` permite obtener información sobre la búsqueda minimax. En concreto,
    - `verbose = 1 # Se imprime el valor minimax de la jugada seleccionada`
    - `verbose = 2 # Se ejecuta el juego paso a paso y se imprimen los valores minimax de todos los estados explorados en la búsqueda.`
    - `verbose = 3 # Se ejecuta el juego de manera continua y se imprimen los valores minimax de todos los estados explorados en la búsqueda.`

  Adicionalmente, en los modos con `0 < verbose < 3` se ofrece la posibilidad de almacenar el estado intermedio correspondiente; es decir la información sobre el jugador que va a realizar el movimiento y el estado del tablero.
- En `demo_reversi.py` podéis encontrar distintos jugadores inicializados: manuales, aleatorios, que utilizan estrategias con búsqueda (aplicando el algoritmo minimax para la búsqueda y una función

para la evolución de estados del juego). Los jugadores se implementan como estrategias de juego (clase abstracta `Strategy` definida en el módulo `strategy.py`). También se puede ver cómo crear un jugador que utilice una heurística creada por vosotros.

- Podéis también explorar otros juegos más simples (puede ser útil para depurar el código): `demo_tictactoe.py` y `demo_simple_game_tree.py`.
- En el módulo `heuristic.py` se incluye una función heurística (`simple_evaluation_function`) que se puede pasar a jugadores de tipo minimax. En el siguiente apartado veréis que **uno de vuestros objetivos en esta práctica es diseñar una función heurística que sea competitiva**.
- El tipo de estrategia `MinimaxStrategy`, además de la función heurística, recibe como argumento la máxima profundidad a la que se hará la búsqueda (`max_depth_minimax`).
- Para permitir experimentar más rápido y probar distintas situaciones, la infraestructura entregada permite jugar a partir de un estado intermedio. Para ello es necesario inicializar las variables `initial_board` e `initial_player` con los valores correspondientes. Para jugar a partir de la configuración estándar se puede utilizar `board = None` en la instanciación de un objeto de tipo `TwoPlayerGameState`.
- Las funciones `from_dictionary_to_array_board`, `from_array_to_dictionary_board` que se encuentran en el fichero `reversi.py` permiten transformar el tablero en forma de diccionario (el usado en la implementación del juego) a uno en forma de array (más fácil de visualizar), y viceversa.
- Los ficheros `tictactoe.py` y `demo_tictactoe.py` permiten jugar al Tres en Raya, un juego más sencillo que Reversi y con menor factor de ramificación, que puede ser útil para vuestras pruebas.
- Para la implementación del algoritmo minimax con poda alfa-beta, puede ser útil implementar un juego más sencillo o analizar alguno de los que se dan en la práctica. Por ejemplo, uno definido por un árbol de juego (ver `simple_game_tree.py` y `demo_simple_game_tree.py`).

Además de probar a jugar las partidas que están configuradas en el módulo `demo_reversi.py`, también podéis llevar a cabo un torneo con varias estrategias y funciones de evaluación llamando a la función `run` del módulo `tournament.py` indicando:

1. Las estrategias.
2. El número de partidas con fichas negras (se jugará el mismo número con blancas).
3. Los nombres o alias de las estrategias.

Una posible salida del código que tenéis en `demo_tournament.py` es la siguiente:

```
             total:      opt1_dummy  opt2_random opt3_heuristic
opt1_dummy         2:    ---         2                  0
opt2_random 9:     8           ---               1
opt3_heuristic     19:   10          9                  ---
```

En esta configuración se han jugado 30 partidas en total (20 para cada estrategia): 30 = 5 × 2 × 3 = 5 repeticiones (parámetro `n_pairs`) × 2 jugadores (cada estrategia juega como blancas y negras) × número de estrategias (3 en este caso). Se puede ver que la estrategia `random` gana 9 de los 30 juegos, 8 contra `dummy` y 1 contra `heuristic`.

En el mismo fichero podéis ver (entre comentarios) cómo cargar estrategias desde un fichero que exista en una carpeta, tal y como se cargarán en el torneo. Puede ser útil probarlo de esta manera para confirmar que funciona sin problemas.

Observación: la función `run` del módulo `tournament.py` usa `MinimaxStrategy`, `sin embargo, cuando tengáis implementado` `MinimaxAlphaBetaStrategy`, es aconsejable que lo cambiéis para ahorrar tiempo en vuestras ejecuciones.

3. Implementación de funciones de evaluación

Para implementar vuestras funciones de evaluación heurística debéis crear un fichero *python* con:

1. Tantas clases que hereden de `StudentHeuristic` como funciones de evaluación queráis probar.
2. Dentro de cada clase, tenéis que implementar dos funciones: una de nombre `evaluation_function` (que recibe un potencial movimiento en la partida, encapsulado como un estado de tipo `TwoPlayerGameState`, y debe devolver una estimación de su interés para el jugador) y otra `get_name` donde indicaréis el nombre con el que esta heurística aparecerá en el torneo.

A continuación, tenéis un ejemplo con una heurística trivial (no hace nada con el estado), donde podéis ver cómo se pueden incluir funciones auxiliares:

```
from game import (
    TwoPlayerGameState,
)
from tournament import (
    StudentHeuristic,
)

class MySolution1(StudentHeuristic):
  def get_name(self) -> str:
    return "mysolution1"
  def evaluation_function(self, state: TwoPlayerGameState) -> float:
    # let's use an auxiliary function
    aux = self.dummy(123)
    return aux

  def dummy(self, n: int) -> int:
    return n + 4

class MySolution2(StudentHeuristic):
   ...
```

Importante: **el número de heurísticas por envío está limitado a 3**, por lo que, aunque incluyáis más clases, no se tendrán en cuenta en el torneo (no debéis asumir qué 3 heurísticas se usarán). A través de una interfaz web, podréis subir **el único fichero python** que se os pide para que podáis definir hasta un máximo de 3 estrategias, usando un nombre único que identifique a cada pareja: ***p2_[gggg]_[mm]_[apellido1]_[apellido2].py*** **utilizando el formato especificado en el último apartado de este enunciado**.

Ejemplo:

- p2_1311_01_delval_sanchez.py (estrategias de la pareja 01 en el grupo de prácticas 1311).

Observaciones:

- No se admitirán jugadores que hagan llamadas a funciones del sistema, usen funciones de control del juego o manipulen sus estructuras de datos.
- Es importante que la función de evaluación sea eficiente, ya que se controlará el tiempo invertido por cada jugador, de manera que se dará la partida por perdida a aquel jugador que sea muy lento. Este baremo de cálculo no podrá ser un tiempo absoluto (que depende, además de la máquina, de diversos factores, entre ellos del SO utilizado y de la profundidad de la búsqueda). No obstante, para dar una indicación que sea reproducible en cualquier máquina, habrá que tener en cuenta que **aquellos jugadores que tarden más que 5 veces lo que tarda la heurística trivial** (clase `Heuristic1` en `demo_tournament.py`) en un tablero 8x8 muy probablemente serán eliminados por *timeout*. No es posible asegurar que siempre serán eliminados, porque la duración de una evaluación depende también del nº de movimientos posibles, lo que introduce cierta variabilidad.

4. Evaluación de la práctica

Esta práctica está compuesta por tres partes.

Parte 1 (3 puntos)

Como parte del código del juego se proporciona al alumno una implementación del algoritmo minimax (ver clase `MinimaxStrategy` en el módulo `strategy.py`). Esta implementación está basada en el pseudocódigo que se ha visto en clase:

```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
  return v
```

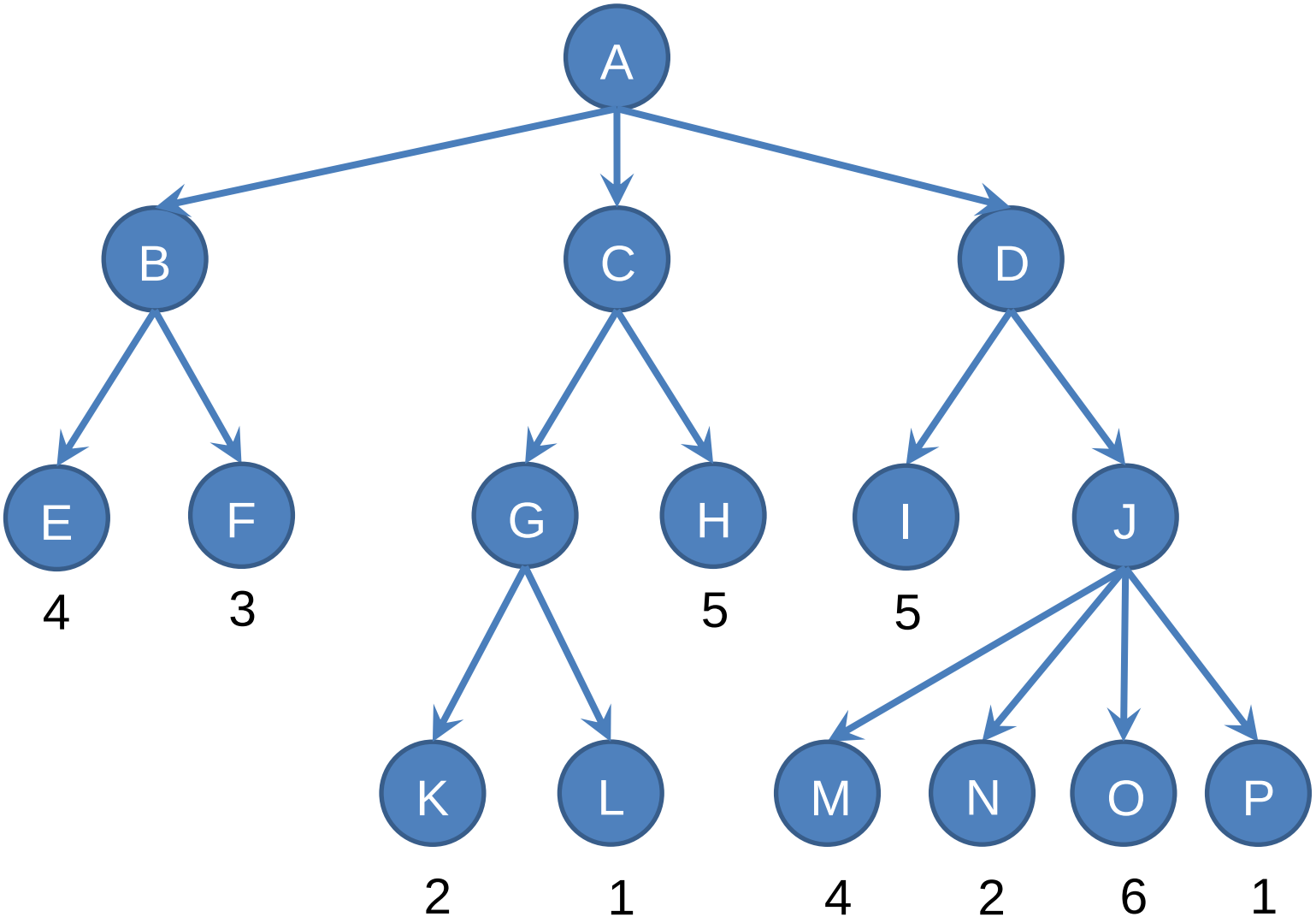
Codifique una función que implemente el algoritmo minimax con poda alfa-beta; para ello, define una clase de nombre `MinimaxAlphaBetaStrategy` que herede de `Strategy` (puedes utilizar la clase `MinimaxStrategy` como punto de partida). La implementación debe ser acorde con el pseudocódigo visto en clase:

```
function ALPHA-BETA-DECISION(state) returns an action
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  same as MAX-VALUE but with roles of α, β reversed
```

El código entregado se valorará tanto por funcionalidad como por estilo. Es importante documentar en la memoria de forma clara el algoritmo empleado en la función realizada, así como su funcionamiento y sus características más relevantes. Para justificar que la poda está funcionando, se debe entregar un informe de tiempos, para lo cual, se aconseja el uso de la librería `timeit`[1]. Para ser más específicos y evitar variaciones debidas a jugadores no deterministas, se realizará el análisis con un jugador que utilice la estrategia determinista mencionada en la sección anterior, es decir, `Heuristic1` en `demo_tournament.py` a profundidades 3 y 4. Además, deberías realizar el mismo análisis con una heurística tuya y comparar los resultados, incluyendo la discusión en la memoria.

Ten en cuenta que los tiempos pueden depender del ordenador en el que trabajas, por lo que, además de los tiempos absolutos, deberíais proporcionar alguna medida de la mejora que sea independiente de dicho ordenador.

Con el fin de determinar si las implementaciones de minimax y minimax con poda alfa-beta son correctas, se propone considerar el juego definido por el árbol



La especificación del juego se encuentra en el fichero `simple_game_tree.py`.

[1]Documentación de timeit (visitada por última vez en Febrero 2022): https://docs.python.org/3/library/timeit.html

Para este juego:

(i) Aplica manualmente el algoritmo minimax. Sobre una copia del árbol, identifica el número de iteración, el nodo que se está visitando y la estimación del valor minimax para dicho nodo en cada paso del algoritmo.

(ii) Comprueba que la ejecución manual coincide con la realizada por el jugador minimax en `demo_simple_game_tree.py` que inicia la partida. Para ello, indica en la memoria la información correspondiente al primer movimiento del juego con `verbose=3`.

(iii) Aplica manualmente el algoritmo minimax con poda alfa-beta. Sobre una copia del árbol, identifica el número de iteración, el nodo que se está visitando y la estimación del intervalo [α, β] para dicho nodo en cada paso del algoritmo. Finalmente, indica el valor minimax en el nodo raíz y la secuencia de movimientos hasta el final de la partida.

(iv) Comprueba que la ejecución manual coincide con la realizada por el jugador minimax que utiliza poda alfa-beta en `demo_simple_game_tree.py`. que inicia la partida. Para ello, **indica en la memoria la secuencia de valores del intervalo [α, β] asociados a los nodos visitados en cada iteración del algoritmo.**

Parte 2 (3 puntos)

Consiste en la implementación y presentación de funciones de evaluación heurística a un torneo entre todas las parejas de estudiantes de la asignatura. En el torneo se probarán **tableros 8x8** comenzando tanto con el estado inicial como con estados intermedios legales y a **profundidades de minimax no mayores de 4**.

Como resultado de los enfrentamientos entre funciones de evaluación se publicará una clasificación, accesible mediante un enlace que se mostrará en la parte del torneo del curso. En las fechas señaladas más abajo, se realizará un torneo entre las estrategias que estén subidas y se actualizará el ranking en la página de Moodle habilitada para ello. En cierto momento se introducirán de manera anónima funciones de evaluación suministradas por los profesores, cuyo objetivo es servir de indicadores de nivel de juego.

La nota de esta parte se calculará de la siguiente manera:

- Cada torneo tendrá un peso que se irá incrementando: 10, 15, 25 y 50%
- Es obligatoria la participación en el torneo con al menos una función. No presentar funciones, haber sido descalificados por errores de forma o ejecución, o por no reunir los mínimos requisitos de calidad o por cualquier otro motivo: 0 puntos.
- Los puntos obtenidos en cada torneo se calcularán según el ranking en el que aparezcan las funciones heurísticas: el 100% de la nota si la mejor de las heurísticas entregadas se encuentra en el percentil 50-90, 150% si está por encima del percentil 90, 75% de la nota si el percentil correspondiente está en el rango 10-50, y 50% en los demás casos. Nótese que el percentil se calcula con las mejores heurísticas entregadas por cada pareja, no con los rankings donde hay varias estrategias de la misma pareja.

Habrá torneos abiertos los siguientes días, antes de cuya fecha se tendrá que subir 3 funciones (al menos una distinta a las anteriores):

- Viernes 20 de octubre (subir antes de las 20:00)
- Viernes 27 de octubre (subir antes de las 20:00)
- Lunes 6 de noviembre (subir antes de las 20:00)

Estas tres entregas son obligatorias y **será penalizada la no entrega** (con un 0 en el ranking correspondiente por cada entrega que falte). La entrega final de funciones de evaluación será el día:

- Lunes 13 de noviembre (subir antes de las 20:00).

La participación a todos los torneos se hace separada de la entrega completa de la P2, que se realizará (por Moodle) en las fechas indicadas al comienzo de este enunciado. Para los torneos se usará la interfaz web disponible en
http://150.244.56.40:5000/
Para entrar a este sistema, deberás usar tu dirección de correo electrónico y una contraseña que podrás encontrar en la tarea de Moodle con nombre "*Acceso para usar sistema de envío de P2*".

Importante: los envíos están controlados por parejas de prácticas, por lo que es crítico que el sistema tenga esta información actualizada. Si el sistema muestra a los dos miembros de la pareja información diferente sobre su grupo de prácticas o número de pareja, informad al coordinador de prácticas.

En cualquier caso, tanto en el código como en la memoria entregados (ver parte 3) se volverán a incluir las estrategias seleccionadas y se analizarán en detalle las distintas soluciones presentadas, incluyendo referencias a artículos (siguiendo el formato APA[2]) que expliquen o justifiquen las heurísticas planteadas.

Resolución de puzles concretos. En la carpeta "`test/`" hay unos *puzles* de prueba que os permitirán saber cómo se comporta vuestra heurística en determinados escenarios. En estos *puzles*, dado un tablero inicial se evalúa con la función `heuristic_1` todos los posibles movimientos que se podrían realizar en ese tablero. Para evaluar el rendimiento de vuestra heurística en ese *puzle* modificad el código de la función `heuristic_1`.

A continuación, se muestra un *puzle* de ejemplo. Dado este tablero donde mueven **negras**:

	A	B	C	D	E	F	G	H
1								
2								
3								
4				○	●			
5				●	○	●		
6						○		
7								
8								

Los posibles movimientos válidos son: C4, D3, E6 y F7. En el fichero test_opening_1_puzzle.py se evalúan los 4 tableros resultantes de hacer esos movimientos. El mejor movimiento y único que beneficia a las negras

[2] APA Quick Citation Guide (visitada por última vez en febrero 2022):
https://guides.libraries.psu.edu/apaquickguide/intext.
Observación: se pueden obtener referencias en formato APA muy fácilmente buscando el artículo en cuestión en Google Scholar, y luego pulsando en el botón de "Cite". Por ejemplo, al hacer eso con la siguiente URL: https://scholar.google.com/scholar?q=the%20texbook, eligiendo el formato APA nos devuelve:
Knuth, D. E., & Bibby, D. (1984). *The texbook* (Vol. 15). Reading: Addison-Wesley.

es E6, el siguiente mejor movimiento es D3, el siguiente C4 y la peor opción es F7. Sin embargo, es importante recordar que una heurística puede obtener buenos resultados en el torneo, pero rendir mal en todos los *puzles* de test, aunque no sería común y significaría que todavía tiene margen de mejora.

Para ejecutar el primero de los *puzles* de test se debe escribir lo siguiente en la línea de comandos:
`python tests/test_opening_1_puzzle.py`

Ejemplo de salida:

```
This is the ranking of the possible movements: E6, D3, C4, F7
1- E6  -- your score for this movement: 3  <- This is the best board (so, your strategy should prefer it).
2- D3  -- your score for this movement: 3
3- C4  -- your score for this movement: 3
4- F7  -- your score for this movement: 3  <- This is the worst board (so, your strategy should not prefer it).
```

Observación: además de usar las heurísticas en el torneo y en los *puzles* de ejemplo, también se recomienda plantear (y entregar) nuevos *puzles* con diferentes situaciones concretas que os permitan identificar qué versiones de vuestras heurísticas funcionan mejor. Esto os ayudará a entender cómo deciden puntuar cada estado.

Parte 3 (4 puntos)

Consiste en una memoria con una descripción de las funciones de evaluación implementadas, así como la documentación asociada a la implementación del algoritmo alfa-beta, que se subirá a Moodle junto con el código de las otras partes.

¿Qué se debe incluir en la documentación?
El informe debe ser claro, completo y conciso. Por favor, incluye sólo información relevante.

1. Documentación de minimax y la poda alfa-beta
    a. Detalles de implementación
        i. ¿Qué tests se han diseñado y aplicado para determinar si la implementación es correcta?
        ii. Diseño: estructuras de datos seleccionadas, descomposición funcional, etc.
        iii. Implementación.
        iv. Otra información relevante.

    b. Eficiencia de la poda alfa-beta.
        i. Descripción completa del protocolo de evaluación.
        ii. Tablas donde se incluyan tiempos con y sin poda.
        iii. Medidas de mejora independientes del ordenador.
        iv. Análisis correcto, completo y claro de los resultados.
        v. Otra información relevante.

2. Documentación del diseño de la heurística.
    a. Revisión de trabajos previos sobre estrategias de Reversi, incluyendo referencias en el formato APA.

    b. Descripción del proceso de diseño:
        i. ¿Cómo se planeó y ejecutó el proceso de diseño?
        ii. ¿Seguiste algún procedimiento sistemático para evaluar las heurísticas diseñadas?
        iii. ¿Cuál fue el rendimiento de tus heurísticas en los puzles entregados? ¿Te apoyaste en ellos para plantear modificaciones?
        iv. ¿Utilizaste ideas desarrolladas por otros para mejorar las estrategias diseñadas? Si están disponibles públicamente, incluye referencias en formato APA; en otro caso, incluye el nombre de la persona que te dio la información y dale el crédito oportuno como “comunicación privada”.

    c. Descripción de la heurística enviada finalmente.

    d. Otra información relevante.

Para la entrega FINAL en formato electrónico, se creará un archivo ZIP que contenga todo el material de la entrega (código completo de funciones heurísticas de evaluación e implementación del algoritmo alfa-beta + memoria en pdf), cuyo nombre, todo él en minúsculas y sin acentos, tildes, o caracteres especiales, tendrá la siguiente estructura:

p2_[gggg]_[mm]_[apellido1]_[apellido2].zip

donde

[gggg] : Número de grupo: (1311, 1312, 1391, etc.)
[mm] : Número de orden de la pareja con dos dígitos (01, 02, 03, etc.)
[apellido1] : Primer apellido del miembro 1 de la pareja
[apellido2] : Primer apellido del miembro 2 de la pareja

Los miembros de la pareja deben aparecer en orden alfabético.

Ejemplos:

- p2_1311_01_delval_sanchez.zip (práct. 2 de la pareja 01 en el grupo de prácticas 1311)
- p2_1392_18_bellogin_suarez.zip (práct. 2 de la pareja 18 en el grupo de prácticas 1392)

Fechas de entrega

Debido al calendario, este curso se recomienda dividir la entrega de esta práctica en dos:

- En la entrega intermedia (3 de noviembre), debes entregar la parte 1 (poda alfa-beta) y la parte correspondiente de la memoria (parte 3.1).
- En la entrega final (17 de noviembre), debes entregar la práctica completa.