Práctica 2: Reversi

Laboratorio de Inteligencia Artificial – Curso 2020-2021

Fecha de publicación: 10-03-2021
Fecha del torneo final: 06-04-2021
Fecha de entrega en Moodle:

- Grupos de los miércoles: miércoles 07-04-2020, 08:59
- Grupos de los jueves: jueves 08-04-2020, 08:59
- Grupos de los viernes: viernes 09-04-2020, 08:59

Criterios de evaluación generales:

- La implementación es correcta.
- El código está bien estructurado. En concreto se ha realizado una descomposición funcional apropiada.
- Se usan estructuras y un estilo de programación propios de python (por ejemplo, se evitan los bucles haciendo uso de operaciones vectorizadas, o *list comprehensions*; se usan las estructuras de datos adecuadas: tuplas, listas, diccionarios, arrays de numpy, iterables, etc.).
- El código sigue las especificaciones contenidas en PEP 8.
- El código es claro, legible y está bien documentado (por ejemplo, las funciones son cortas y llevan a cabo una única operación de alto nivel; los nombres de las variables y las funciones son significativos, no hay constantes mágicas, etc.)

1. El juego del Reversi

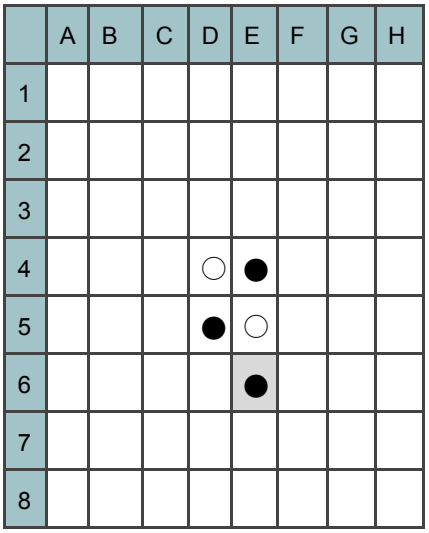
Reversi (también conocido como **Othello** o **Yang)** es un juego de estrategia de dos jugadores. Se juega en un tablero de 8×8 con la siguiente configuración inicial:

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

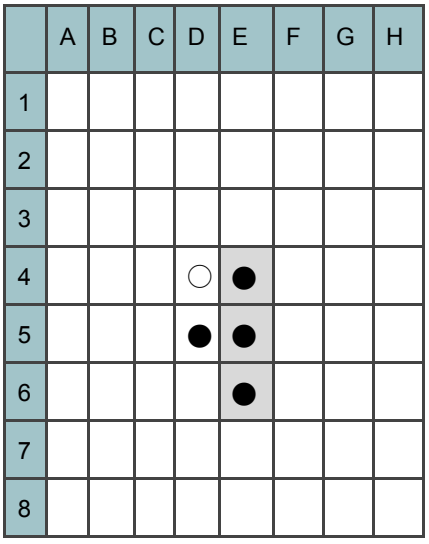
Los dos jugadores, representados por las fichas negras y blancas, alternan turnos empezando las negras. En cada turno, el jugador coloca una pieza de su color sobre el tablero de forma que capture alguna de las

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 pueden 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:



2. Implementación de Reversi y de las funciones de evaluación

El código proporcionado para esta práctica requiere la versión 3.7 de Python o superior.

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.**

En el módulo **`heuristic.py`** se han implementado algunos ejemplos de funciones de evaluación de estados del juego, las cuales son utilizadas en estrategias de búsqueda.

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_sec_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 click 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 imprimen los valores minimax de todos los estados explorados en la búsqueda.`

  Adicionalmente, en los modos con `verbose > 0` 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`).

- 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 una 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. Por ejemplo, uno definido por un árbol de juego. La implementación consiste en
    - Crear una clase derivada de `TwoPlayerGame`.
      Por ejemplo: `class MyGame(TwoPlayerGame):`
    - Implementar las funciones:
        - `__init__`
        - `initialize_board`
        - `display`
        - `generate_successors`
        - `score`

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`.

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 él 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. A través de una tarea en Moodle, 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: ***[gggg]_p2_[mm]_[apellido1]_[apellido2].py*** **utilizando el formato especificado en el último apartado de este enunciado**.

Ejemplos:

- 2311_p2_01_delval_sanchez.py (estrategias de la pareja 01 en el grupo de prácticas 2311)
- 2362_p2_18_bellogin_suarez.py (estrategias de la pareja 18 en el grupo de prácticas 2362)

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 hemos 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 que hemos 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, realiza el mismo análisis con una heurística tuya y compara 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.

Parte 2 (4 puntos)

Consiste en la implementación y presentación de funciones de evaluación heurística a un torneo entre todas las parejas de la asignatura. En el torneo se probarán **tableros 8x8** comenzando 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 (como se puede observar, es posible obtener *más de los 4 puntos* asignados a este apartado):

- Participación anticipada en el torneo: 2 puntos
- En función del resultado de la mejor de las 3 funciones de evaluación heurística presentadas por cada pareja en el momento de cierre del torneo, en relación con las funciones de evaluación de referencia, se puntuará:
    - función que supera a 5 de las funciones de referencia: 3 puntos
    - función que supera a 4 de las funciones de referencia: 2 puntos
    - función que supera a 3 de las funciones de referencia: 1 puntos
    - función que supera a 2 de las funciones de referencia: 0.5 puntos
    - función que supera a 1 de las funciones de referencia: 0.25 puntos
- Se concederá un premio adicional de 1 punto a las 5 funciones que ocupen las primeras posiciones de la clasificación final.
- 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.

Para obtener los 2 puntos por participación anticipada habrá que subir:
- Tres funciones el 15 de marzo de 2021 (subir antes de las 23:59)
- Tres funciones el 17 de marzo de 2021 (subir antes de las 23:59)

Habrá torneos abiertos los siguientes días, en los que se tendrá que subir 3 funciones:
- 23 de marzo (subir antes de las 20:00)
- 26 de marzo (subir antes de las 20:00)
- 5 de abril (subir antes de las 20:00)

[1]Documentación de timeit (visitada por última vez en Febrero 2021): https://docs.python.org/3/library/timeit.html

Estas tres entregas son obligatorias y **será penalizada la no entrega** (1 punto de penalización por cada entrega que falte). La entrega final de funciones de evaluación será el día:

- 6 de abril de 2021 (subir antes de las 23:59).

Esta entrega final se hace separada de la entrega completa de la P2, que se realizará en las fechas indicadas al comienzo de este enunciado. Esto es así para poder lanzar el torneo con esta última versión de las funciones de evaluación.

No obstante, 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.

Parte 3 (3 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 memoria?

La memoria debe ser clara, completa, y concisa. Por favor, incluid únicamente información relevante.

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

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

2. Documentación del proceso de diseño de la heurística.
    a. Descripción de trabajo anterior sobre estrategias para el juego Reversi, incluyendo estrategias en formato APA.
    b. Descripción del proceso de diseño:
        i. ¿Cómo se planificó y realizó el proceso de diseño?

[2] APA Quick Citation Guide (visitada por última vez en Febrero 2021): 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.

ii. ¿Se utilizó un procedimiento sistemático para la evaluación de las heurísticas diseñadas?
iii. ¿Utilizaste en el diseño estrategias diseñadas por otros? Si estas están disponibles de manera pública, proporciona referencias en formato APA; en caso contrario, indica el nombre de la persona que proporcionó la información y reconoce dicha contribución como “comunicación privada”.

c. Descripción de la heurística final entregada.
d. Otra información relevante.

Para la entrega en formato electrónico, se creará un archivo ZIP que contenga todo el material de la entrega (código completo + memoria en pdf), cuyo nombre, todo él en minúsculas y sin acentos, tildes, o caracteres especiales, tendrá la siguiente estructura:

[gggg]_p2_[mm]_[apellido1]_[apellido2].zip

donde

[gggg] : Número de grupo: (2301, 2311, 2312, 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 aparecen en orden alfabético.

Ejemplos:

- 2311_p2_01_delval_sanchez.zip (práct. 2 de la pareja 01 en el grupo de prácticas 2311)
- 2362_p2_18_bellogin_suarez.zip (práct. 2 de la pareja 18 en el grupo de prácticas 2362)