Gregorio Blázquez Martínez
Diego Vicente Miguel

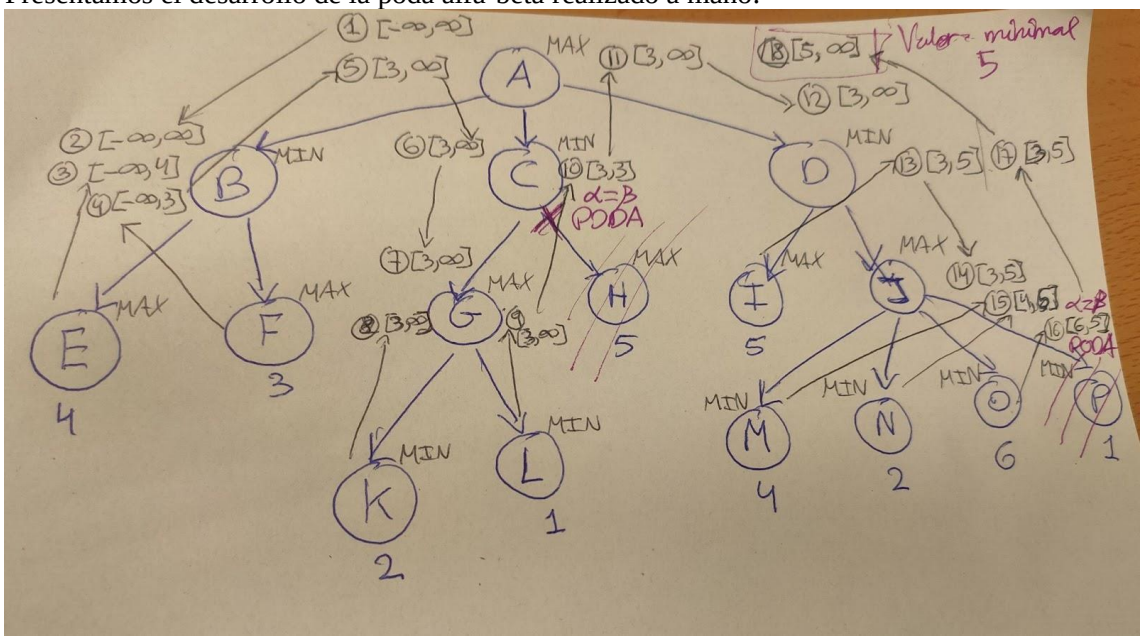
MEMORIA P2:

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?*

Hemos probado demo_simple_game_tree.py comparándola con la poda alfa-beta que hemos hecho a mano, obteniendo el mismo resultado en cada paso en cuanto a sucesores y nodos expandidos y comprobando así su correcto funcionamiento. También comprobamos experimentalmente que el tiempo de ejecución de las partidas se reducía considerablemente con el nuevo algoritmo con poda alpha-beta.
Presentamos el desarrollo de la poda alfa-beta realizado a mano:



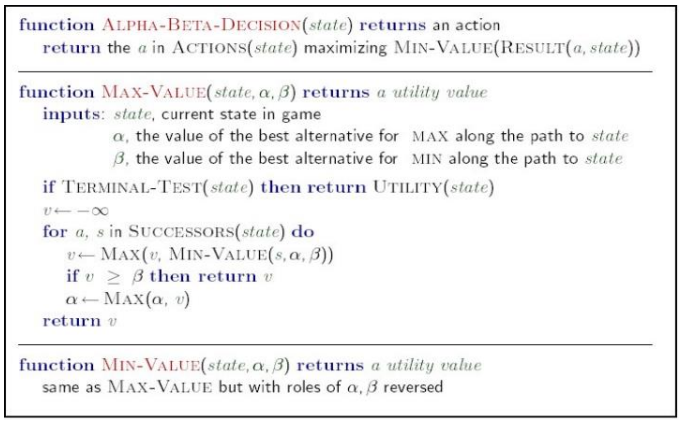
ii. *Diseño: estructuras de datos seleccionadas, descomposición funcional, etc.*

Para implementar la poda alfa beta hemos partido, como se nos pedía, de las funciones de minimax dadas y hemos añadido dos enteros (alpha y beta) a los argumentos de las llamadas a max_value y min_value para poder transferir la información necesaria para implementar la poda alfa beta.
La implementación funciona exactamente como el algoritmo minimax en los casos base (las hojas), en el resto de caso al igual que en el algoritmo minimax separamos en casos MAX o MIN propagando ahora los alphas y betas para poder podar.

iii. *Implementación.*

Hemos seguido las recomendaciones en cuanto al pseudocódigo proporcionado en la descripción de la práctica. Hemos adaptado este código a python teniendo especial cuidado con un fallo que tuvimos en cuanto a la actualización de sucesores minimax.
Pseudocódigo seguido:

Gregorio Blázquez Martínez
Diego Vicente Miguel



```
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
```

Código de la poda alfa-beta implementado:

```python
def _min_value(
    self,
    state: TwoPlayerGameState,
    depth: int,
    alpha: int,
    beta: int,
  ) -> float:
    """Min step of the minimax algorithm."""
    minimax_successor = None
    if state.end_of_game or depth == 0:
        minimax_value = self.heuristic.evaluate(state)
    else:
        minimax_value = np.inf

        for successor in self.generate_successors(state):
            if self.verbose > 1:
                print('{}: {}'.format(state.board, minimax_value))

            x, _ = self._max_value(
                successor,
                depth - 1,
                alpha,
                beta,
            )
            successor_minimax_value = min(minimax_value, x)
            if (successor_minimax_value <= alpha):
                return successor_minimax_value, minimax_successor
            beta = min(beta, successor_minimax_value)
            if(x<minimax_value):
                minimax_successor = successor
            minimax_value = beta

    if self.verbose > 1:
        print('{}: {}'.format(state.board, minimax_value))

    return minimax_value, minimax_successor



    if self.verbose > 1:
        print('{}: {}'.format(state.board, minimax_value))
```

```
    return minimax_value, minimax_successor

def _max_value(
    self,
    state: TwoPlayerGameState,
    depth: int,
    alpha: int,
    beta: int,
) -> float:
    """Max step of the minimax algorithm."""
    minimax_successor = None
    if state.end_of_game or depth == 0:
        minimax_value = self.heuristic.evaluate(state)
    else:
        minimax_value = -np.inf

        for successor in self.generate_successors(state):
            if self.verbose > 1:
                print('{}: {}'.format(state.board, minimax_value))

            x, _ = self._min_value(
                successor,
                depth - 1,
                alpha,
                beta,
            )
            successor_minimax_value = max(minimax_value, x)
            if (successor_minimax_value >= beta):
                return successor_minimax_value, minimax_successor
            alpha = max(alpha, successor_minimax_value)
            if(x>minimax_value):
                minimax_successor = successor
            minimax_value = alpha

    if self.verbose > 1:
        print('{}: {}'.format(state.board, minimax_value))

    return minimax_value, minimax_successor
```

iv. Otra información relevante.

Hemos comprobado una clara mejoría en el tiempo de ejecución que nos ha permitido realizar numerosos torneos para mejorar la eficiencia y ejecución de nuestras heurísticas realizadas, ya que el algoritmo minimax presenta una exploración en profundidad de todos los nodos y supone una ejecución muy lenta y pesada de los diferentes algoritmos probados.

b. Eficiencia de la poda alfa-beta.

i. Descripción completa del protocolo de evaluación.

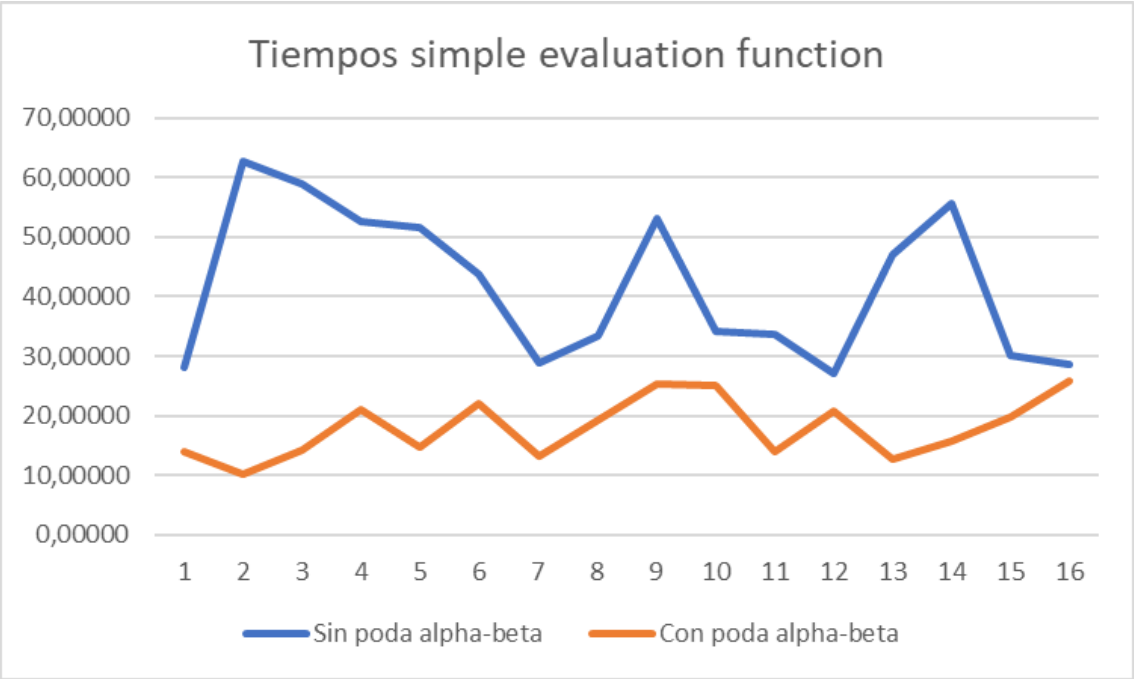
Para evaluar los resultados de la poda alfa-beta, hemos realizado varias comprobaciones. En primer lugar, nos hemos asegurado manualmente (apartados anteriores) que la ejecución de

simple_game_tree.py se corresponde con el resultado real de la poda alfa-beta, tanto en el valor minimax, como en la generación de sucesores.
Además, hemos establecido un análisis de tiempos por partida en un torneo de una heurística contra sí misma, de forma que comparemos tiempo entre algoritmos con estrategias minimax y otros con poda alfa beta. (Comentar como hemos medido con time.time())

ii. Tablas donde se incluyan tiempos con y sin poda.

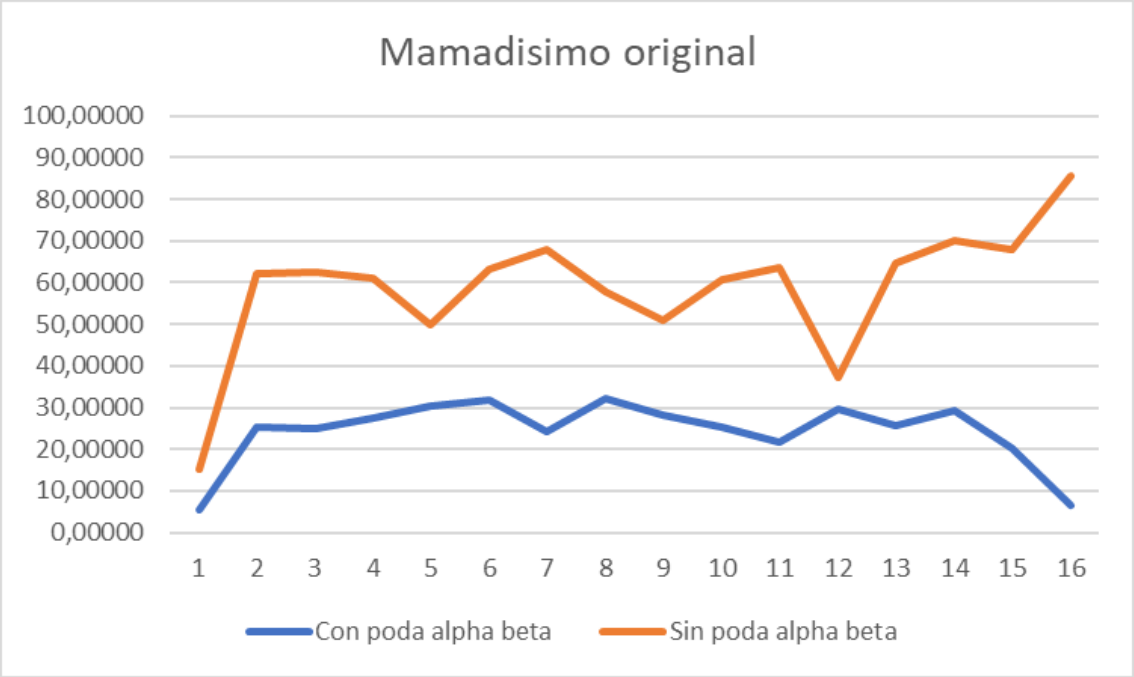
Simple_evaluation_funtion	
Sin poda Alpha-beta	Con poda Alpha-beta
28,04831	13,9198823
62,65666	10,1520128
58,97309	14,1918921
52,71399	20,9505565
51,59214	14,7395225
43,86367	21,9455082
28,81134	13,1909676
33,33422	19,1297123
53,14337	25,2894282
34,19200	25,059166
33,74930	14,0275097
27,15149	20,8882003
47,03356	12,5665429
55,66103	15,8353412
30,12916	19,8172946

Gregorio Blázquez Martínez
Diego Vicente Miguel



Mamadísima original	
Sin poda Alpha-beta	Con poda Alpha-beta
15,1169009	5,33637
62,2699258	25,29727
62,5774162	24,80735
61,1510739	27,39107
49,8219099	30,38569
63,2929938	31,82372
67,8502836	24,28893
57,8182402	32,05238
50,9372985	28,15888
60,8966241	25,15031
63,6668856	21,60301
37,1514924	29,62221
64,7052169	25,51300

70,1270151	29,42851
67,8318088	20,31008



Podemos observar que en ambas heurísticas la poda alpha-beta presenta un menor tiempo de ejecución en todos los casos probados, demostrando que es un método eficaz y muy efectivo.

iii. Medidas de mejora independientes del ordenador.

Para tener una medida de la mejora independiente del ordenador hemos tomado medidas de cuantos nodos expandidos por el algoritmo minimax con y sin la poda alpha-beta. Para esto hemos añadido un contador en el algoritmo y hemos obtenido resultados de numerosas partidas para hacer un promedio con un número significativo de elementos. También para que sea realmente significativa la comparación hemos comparado la misma partida con y sin poda alpha-beta, de esta manera estamos comparando las mismas situaciones con y sin poda alpha beta. En particular hemos comparado para la situación inicia del juego con los algoritmos *simple_evaluation_funtion* y para nuestra heurística *mamadisima_original* obteniendo los siguientes resultados:

Número de nodos expandidos	***simple_evaluation_funtion***	***mamadisima_original***
Sin poda alpha-beta	782,758333	1159,05
Con poda alpha beta	331,133333	302,15

Esta tabla muestra los promedios de los datos, guardados de las partidas descritas arriba, en operaciones.txt que incorporamos en la entrega para poder ser consultado.

iv. Análisis correcto, completo y claro de los resultados.

Queda totalmente claro que hay una mejoría notable en el rendimiento los programas ejecutados con poda alpha-beta respecto de los que utilizan el algoritmo minimax; esto se ve reflejado tanto en el número de operaciones que realizan, siendo inferior en la poda alpha-beta y en el tiempo de ejecución que también se nota una gran disminución (en torno a ⅓ del tiempo de ejecución con algoritmo minimax).

v. Otra información relevante.

El ahorro de tiempo y operaciones que supone la poda alpha-beta ha sido fundamental para poder desarrollar esta práctica ahorrando tiempo a la hora de probar nuestras heurísticas.

2. Documentación del diseño de la heurística.

a. Revisión de trabajos previos sobre estrategias de Reversi strategies, incluyendo referencias en el formato APA.

Referencias consultadas:

Heuristic Function for Reversi (Othello) (n.d.) https://github.com/kartikkukreja/blog-codes/blob/master/src/Heuristic%20Function%20for%20Reversi%20(Othello).cpp
Reversi (n.d.) https://github.com/arminkz/Reversi
Morillo, D. y Busquiel C. Juego de Inteligencia Artificial. Recuperado de https://www.it.uc3m.es/jvillena/irc/practicas/07-08/Othello.pdf el 3 de marzo del 2022
Sannidhanam V. y Annamalai M. An Analisis of Heuristics in Othello. Recuperado de https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/RUSSIA/Final_Paper.pdf el 8 de marzo del 2022
Jacopo Festa, Davino S. "IAgo Vs Othello": An artificial intelligence agent playing Reversi http://ceur-ws.org/Vol-1107/paper2.pdf el 12 de marzo del 2022

Tras consultar varias referencias de trabajos previamente realizados y estudios deterministas de las situaciones y estrategias óptimas para conseguir ganar en la partida de Reversi, hemos dado con algunas heurísticas que combinaban las ideas principales de las encontradas.

b. Descripción del proceso de diseño:

i. ¿Cómo se planeó y ejecutó el proceso de diseño?

En primer lugar, hicimos una recopilación de información en cuanto a las estrategias óptimas y a los puntos a tener en cuenta para consolidar una heurística aceptable. Llegamos a la conclusión de que los puntos más relevantes para crear una heurística decente son los siguientes: diferencias de fichas, fichas frontera y cuadrados que forman el borde de las piezas; ocupación y cercanía a las esquinas; número de movimientos posibles en relación a tu oponente; valores y pesos prefijados a diferentes posiciones del tablero, etc.
Tras realizar un código con los valores dentro de cada ámbito normalizados, pasamos a buscar una forma de ponderar adecuadamente cada valor en función del momento de la partida en el que nos encontramos. En una primera aproximación, establecimos una ponderación estática, obteniendo resultados precisos pero mejorables. A continuación, tratamos de establecer funciones con variaciones significativas en función del momento de la partida en el que nos encontremos. Por ejemplo, darle una mayor prioridad al número de fichas del usuario a medida que avanza la partida, teniendo una gran importancia y peso en los últimos movimientos.
.

ii. ¿Seguiste algún procedimiento sistemático para evaluar las heurísticas diseñadas?

En nuestro caso realizamos diferentes torneos, primero contra las heurísticas ya anteriormente creadas como dummy, simple y random. Una vez conseguimos vencerlas en distintos tipos de tableros y realizando numerosas partidas, pasamos a ejecutar torneos entre nuestras heurísticas incluyendo varios tableros diferentes y una generación pseudo-aleatoria de entre ellos, de forma que se consiguieron unos resultados más o menos fiables para poder comprobar si íbamos por buen camino. Por último, los resultados en los torneos nos han permitido valorar en qué momento de la partida nuestra heurística no era tan buena respecto a las realizadas por nuestros compañeros, y así comparar soluciones y metodología para seguir evolucionando y mejorando nuestras estrategias.

iii. ¿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".

Las ideas y referencias empleadas ya están anotadas en el apartado a, también compartimos ideas entre compañeros para conseguir mejorar nuestra estrategia y heurística y obtener unos resultados mejores y eficientes.

c. Descripción de la heurística enviada finalmente.

En nuestro caso hemos enviado tres heurísticas.

La primera, y más simple de todas, de pesos estáticos asignados a cada posición del tablero de forma que en función del movimiento se de mayor o menor puntuación a la hora de conseguir determinadas localizaciones del tablero. Con una única variación, en las últimas jugadas también tenemos en cuenta el número de fichas que comemos. De esta manera tratamos de evitar que podamos desaprovechar jugadas finales en las que podamos comer un gran número de fichas para ganar la partida ya que en las últimas jugadas la posición táctica va perdiendo importancia en comparación con el número de fichas que tenemos al acabar la partida determina el ganador.

A continuación, realizamos una heurística que tenía en cuenta varios aspectos del juego (mencionados anteriormente) como la movilidad, número de fichas del usuario respecto del rival, ocupación de esquinas, etc. Para establecer el valor de la heurística realizamos una ponderación estática (con valores prefijados), dando mayor o menor importancia a los distintos aspectos de juego.

Por último, hemos implementado una heurística dinámica, que tenía en cuenta los mismos aspectos del juego que la heurística anterior pero esta vez ponderaba cada elemento en función del momento de la partida en el que nos encontráramos. De esa forma, el control del centro presenta una gran importancia a mediados de partida y el número de fichas respecto de las del rival, al final de la partida.

d. Otra información relevante.

En un primer momento nos resultó muy difícil entender el funcionamiento del código que se nos proporcionaba y para la primera entrega teníamos muchas dudas de cómo tenía que tener en cuenta nuestra función de evaluación si el jugador era MAX o MIN o si el jugador era blancas o negras. Hasta tal punto que en la primera entrega subimos el código mal y no sabía reconocer si tenía que devolver el valor positivo o negativo de la función de evaluación. Más tarde entendimos que había que utilizar el valor de next_player de el estado del juego que se nos pasa a la función de evaluación, y por tanto añadimos que tuviera en cuenta si el siguiente jugador en MAX o MIN y que cambiara los labels con los que se calculan la diferencia de número de fichas y resto de ponderaciones.

Aunque después de hacer esto obtuvimos mejores resultados, siguen siendo peores de lo esperado.