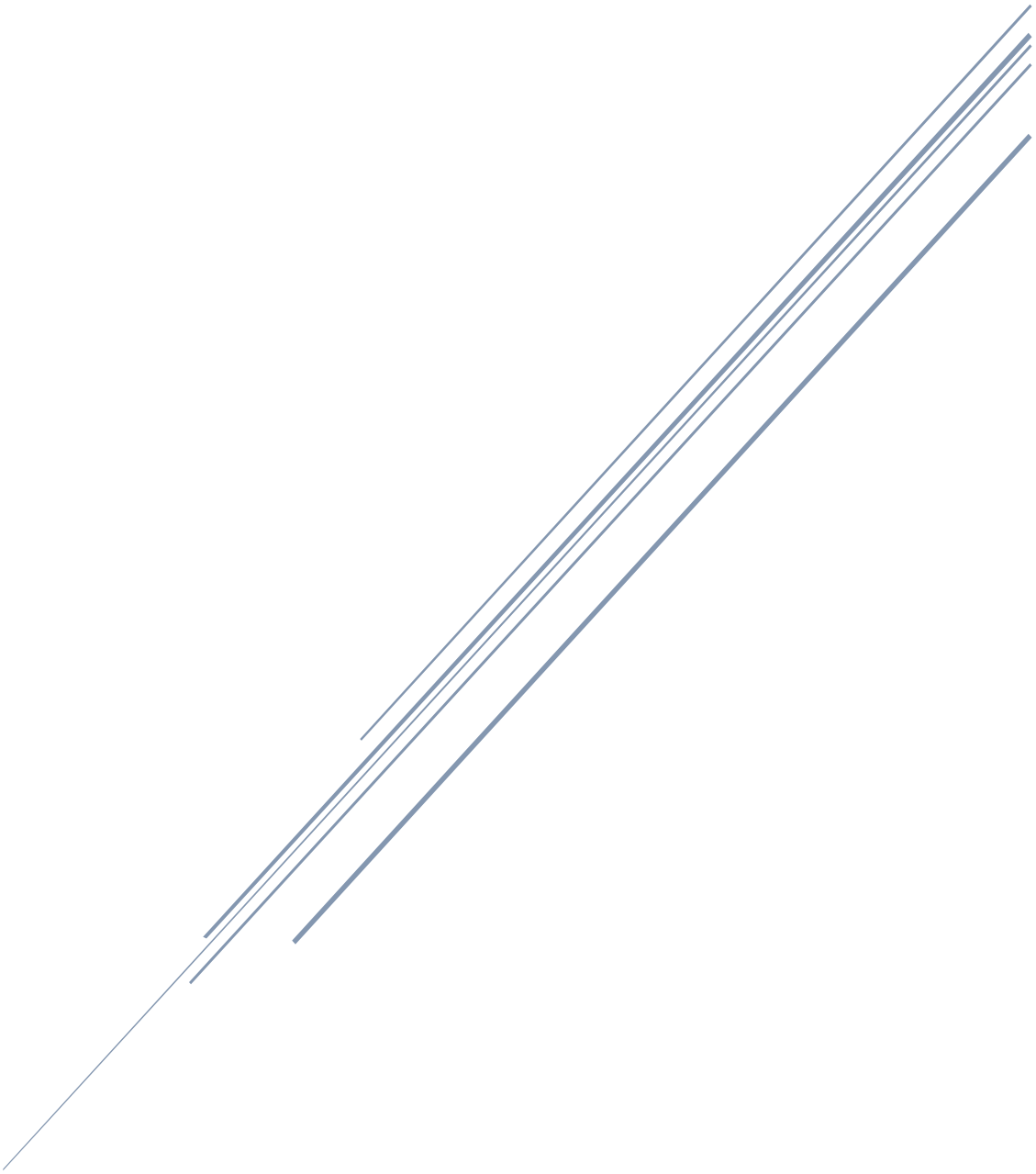
MEMORIA PRÁCTICA 2 IA

Elena Diego Bernabeu y Ángel Bernal García



1. Documentación de minimax
    a. Detalles de la implementación
        i. Para saber si nuestra implementación era la correcta primero hemos seleccionado una heurística trivial, que sería la que tomamos como referencia. Después ejecutamos el minimax sin poda ni alfa beta para saber qué decisiones tomaba. Posteriormente ejecutamos el minimax alfa beta con poda para ver si efectivamente, seguía el mismo camino, pero en menos tiempo.
        ii. Hemos seguido el pseudocódigo sugerido en las diapositivas sin mucho cambio, puesto que al "pasarlo" a código funcional, su resultado ya era satisfactorio. Cabe destacar que tenemos una función que es la llamada desde fuera (next_move), la cuál entre otras cosas, llama a min_value, y ésta a max_value, formando así un bucle.

Para la implementación hemos tomado como base el código del minimax normal proporcionado en la práctica y lo hemos modificado para que se adecuara a la poda alfa beta. Código:

```
class MinimaxAlphaBetaStrategy(Strategy):
    """Minimax alpha-beta strategy."""

    def __init__(
        self,
        heuristic: Heuristic,
        max_depth_minimax: int,
        verbose: int = 0,
    ) -> None:
        super().__init__(verbose)
        self.heuristic = heuristic
        self.max_depth_minimax = max_depth_minimax

    def next_move(
        self,
        state: TwoPlayerGameState,
        gui: bool = False,
    ) -> TwoPlayerGameState:
        """Compute next state in the game."""

        # NOTE <YOUR CODE HERE>
        successors = self.generate_successors(state)
```

```
        minimax_value = -np.inf

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

            successor_minimax_value = self._min_value(
                successor,
                self.max_depth_minimax,
                -np.inf,
                np.inf
            )
            if (successor_minimax_value > minimax_value):
                minimax_value = successor_minimax_value
                next_state = successor

        if self.verbose > 0:
            if self.verbose > 1:
                print('\nGame state before move:\n')
                print(state.board)
                print()
            print('Minimax value = {:.2g}'.format(minimax_value)
)
        return next_state

    def _min_value(
            self,
            state: TwoPlayerGameState,
            depth: int,
            alpha: int,
            beta: int,
        ) -> float:
            """Min step of the minimax algorithm."""
            if state.end_of_game or depth == 0:
                successor_minimax_value = self.heuristic.evaluat
e(state)

            else:
                successor_minimax_value = np.inf

                successors = self.generate_successors(state)
```

```
                for successor in successors:
                    if self.verbose > 1:
                        print('{}: {}'.format(state.board, succe
ssor_minimax_value))

                    successor_minimax_value = self._max_value(su
ccessor, depth - 1, alpha, beta)
                    if (successor_minimax_value <= alpha):
                        return successor_minimax_value

                    beta = max(alpha, successor_minimax_value)

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

            return successor_minimax_value

    def _max_value(
        self,
        state: TwoPlayerGameState,
        depth: int,
        alpha: int,
        beta: int,
    ) -> float:
        """Max step of the minimax algorithm."""
        if state.end_of_game or depth == 0:
            successor_minimax_value = self.heuristic.evaluate(st
ate)

        else:
            successor_minimax_value = -np.inf

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

                successor_minimax_value = self._min_value(succes
sor, depth - 1, alpha, beta)
                if (successor_minimax_value >= beta):
```

```
                    return successor_minimax_value

                alpha = max(alpha, successor_minimax_value)

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

        return successor_minimax_value
```

b. Eficiencia de la poda alfa beta
   i. Los siguientes tiempos han sido conseguidos en un ordenador con 16GB de RAM y un procesador Intel i7-10700K

	Heurística trivial	Heurística propia
Con poda	17.89 seg	46.03 seg
Sin poda	58.74 seg	150.85 seg

   ii. Mejorar nuestras heurísticas con otras más eficientes. Tras haber examinado el código no hemos encontrado demasiado margen de mejora, pues los tiempos conseguidos son bastante buenos.
   iii. Podemos ver que las heurísticas triviales obviamente consumen menos tiempo que nuestras propias heurísticas, debido al ínfimo coste computacional que tienen. También se observa una clara mejora en el tiempo de realización del minimax con poda con respecto al normal, siendo prácticamente un 300% más lento el sin poda.
   iv. No procede.

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.*
      Al principio estuvimos informándonos de cuáles son las mejores estrategias y técnicas para ganar en Reversi. Nos quedamos con dos heurísticas principales: controlar las esquinas y controlar los muros. Sin embargo, al empezar a programar nos dimos cuenta de que no éramos capaces de que dieran resultados satisfactorios y competitivos, por lo que optamos por heurísticas más sencillas, que acabaron resultando en mejores puntuaciones y posiciones más altas en las tablas de rankings.
   b. *Descripción del proceso de diseño*
      i. *¿Cómo se planificó y realizó el proceso de diseño?*
         Hemos ido probando distintas heurísticas usando tanto datos externos como datos propios del juego actual. En base a los resultados obtenidos en los distintos ránkings y las distintas entregas, acabamos eligiendo una heurística que usaba datos

internos y externos a la vez. Por simple que fuera, superaba de forma consistente al resto de heurísticas que teníamos, por lo que deducimos que no era suerte.

ii. *¿Se utilizó un procedimiento sistemático para la evaluación de las heurísticas diseñadas?*
Sí, nos basamos en el resultado de nuestras heurísticas en los diversos ránkings y viendo cuántas funciones de referencia superaban. Se ha escogido la que ha dado mayor puntuación en el último ránking.

iii. *¿Utilizaste en el diseño estrategias diseñadas por otros?*
Estuvimos investigando heurísticas recomendadas por otros programadores para saber cuál era una buena estrategia en reversi e implementarlo nosotros mismos, aunque no dieran los resultados que esperábamos.

c. *Descripción de la heurística final entregada*
Hemos optado por una estrategia muy simple que se basa en la suma entre un número aleatorio del 1 al 10 y se le suma la puntuación mínima del algoritmo minimax usado en la resolución del tablero. Estuvimos probando con la puntuación máxima, pero daba consistentemente peores resultados que con la heurística final.

d. *Otra información relevante*
No procede