1. Documentación de minimax
   1. Detalles de la implementación
      1. 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.
      2. 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.evaluate(state)

            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.\_max\_value(successor, 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\_minimax\_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(state)

        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(successor, 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

* 1. Eficiencia de la poda alfa beta
     1. 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 |

* + 1. 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.
    2. 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.
    3. No procede.

1. Documentación del proceso de diseño de la heurística
   1. 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.