MEMORIA PRACTICA 2

Título: Practica 2 Reversi.

Autores: Guillermo Hoyo Bravo y Saul Almazan del Pie.

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?

Las pruebas se han realizado con las heuristicas simple_evaluation_function y con complex_heuristic_evaluation. La poda Alfa-Beta es más rápida.

ii. Diseño: Estructuras de datos utilizadas, descomposición funcional, etc.

Estructuras de Datos: ninguna.

Descomposición funcional:

Alfa-Beta para inicializar las variables de valores máximos y mínimo según el algoritmo.

- alfa = -np.inf
- beta = np.inf

Funciones usadas:

```
- self.heuristic.evaluate(state)
- self._max_value(sucesor, self.max_depth_minimax, alfa, beta)
- self._max_value(sucesor, self.max_depth_minimax, alfa, beta)
```

Actualizamos los minímos y máximos

iii. Implementación.

<u>PODA ALFA BETA</u>

```
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."""
        alfa = -np.inf
        beta = np.inf
```

```
        minmax = -np.inf
        next_state = None
        sucesores = self.generate_successors(state)

        for sucesor in sucesores:
            minimax_value_successor, alfa_s, beta_s =
self._min_value(sucesor, self.max_depth_minimax, alfa, beta)
            if minimax_value_successor > minmax:
                minmax = minimax_value_successor
                next_state = sucesor

            if beta_s > alfa:
                alfa = beta_s

            if alfa >= beta:
                break
        return next_state

    def min_value(
        self,
        state: TwoPlayerGameState,
        depht: int,
        alfa=int,
        beta=int,
    ) -> (float, float, float):
        if state.end_of_game or depht == 0:
            minmax = self.heuristic.evaluate(state)
            alfa = minmax
            beta = minmax
        else :
            minmax = np.inf
            sucesores = self.generate_successors(state)
            for sucesor in sucesores:
                minmax_sucesores, alfa_s, beta_s =
self._max_value(sucesor, depht - 1, alfa, beta)

                if alfa_s < beta:
                    beta = alfa_s

                if minmax_sucesores <= minmax:
                    minmax = minmax_sucesores

                if alfa >= beta:
                    break
        return minmax, alfa, beta

    def max_value(
            self,
            state: TwoPlayerGameState,
            depht: int,
            alfa=int,
            beta=int,
    ) -> (float, float, float):
        if state.end_of_game or depht == 0:
            minmax = self.heuristic.evaluate(state)
            alfa = minmax
```

```
            beta = minmax
        else:
            minmax = -  np.inf
            sucesores = self.generate_successors(state)
            for sucesor in sucesores:
                minmax_sucesores, alfa_s, beta_s =
self._min_value(sucesor,  depht - 1, alfa, beta)

                if alfa_s > beta:
                    beta = alfa_s

                if minmax_sucesores > minmax:
                    minmax = minmax_sucesores

                if alfa >= beta:
                    break
        return minmax, alfa, beta
```

b. Eficiencia de la poda alfa-beta. Descripción completa del protocolo de evaluación.

Para la eficiencia como bien se nos sugería en el enunciado hemos utilizado la biblioteca timeit añadiendo en demo_reversi

t = timeit.timeit("match.play_match()", number=5, globals=globals())

Y luego solamente imprimirlo con la sentencia print (t) para obtener el tiempo medio

i. Tablas con información sobre los tiempos empleados con y sin poda.

Profundidad	Sin Poda	Con Poda
3	17,8	7,8
4	58,1	10,61

Para realizar los cálculos hemos jugado 3 partidas y hecho la media de las 3 para obtener un resultado más preciso que jugando solo una partida

ii. Medidas de mejora independientes del ordenador.

Se podrían analizar otras condiciones de poda más específicas y especializadas para el juego o procedimiento en cuestión, en este caso el reversi.

iii. Un análisis correcto, claro y completo de los resultados.

Vemos claramente que con la poda Alfa-Beta el tiempo de realización del torneo es menor como cabía de esperar, y cuando mas profundidad tenga más grande será la diferencia entre ambos algoritmos

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?

Primero estudiamos el propio juego, sus reglas y sus estrategias más fuertes para luego reproducirlas. Empezamos creando una estrategia que se guiase a coger solo las esquinas, pero se nos complicó el proceso.

Esto nos llevó a necesitar resolver un par de heurísticas rápidamente antes del primer torneo, para lo que entregamos 2 heurísticas que actuaban por el número de sucesores más grande, y el número de sucesores de cada sucesor.

Sabíamos de antemano que estas estrategias eran las peores para el juego reversi, pero era un comienzo. Después continuamos por realizar una heurística que prioriza las paredes del tablero, y otra que priorizaba las esquinas, para juntarlas y priorizar todos los bordes en una heurística. Continuamos añadimos que la heurística evitase las casillas de borde adyacentes a las esquinas.

ii. ¿Se utilizó un procedimiento sistemático para la evaluación de las heurísticas diseñadas?

No

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".

Comunicación Privada: Nos hemos guiado con las estrategias proporcionadas por el profesor y el conocimiento obtenido del juego. Este en las clases dijo que la mejor estrategia consistía en ir a por las esquinas y a por las paredes del tablero. De igual modo, dijo que las peores estrategias eran las de maximizar el número de sucesores y coger las casillas que son borde y están pegadas a una esquina.

c. Descripción de la heurística final entregada.

La heurística final que queríamos obtener era la mezcla de las dos que vamos a explicar, aunque no lo hemos conseguido, por eso paso a explicar las 2 por orden de mayor rendimiento y casos de victoria.

Heurísticas finales:
Solución 4

Esta heurística prioriza las esquinas, para ello guarda en un array las esquinas y genera los sucesores. Para cada sucesor y cada esquina transforma su posición a una lista de dos ints, resta las coordenadas x e y para realizar el teorema de Pitágoras, sacar la diagonal a la esquina y devolver el valor más pequeño.

Solución3:

Esta heurística comprueba cuando distancia hay hasta la pared más cercana según la posición de la ficha, además, comprueba que pasa si hay varios valores iguales y coge la mínima distancia existente.
Estas heurísticas hacen uso de diccionarios y funciones que ayudan a transformar las posiciones del tablero a listas de ints

d. Otra información relevante.

Finalmente intentamos implementar una heurística conjunta de estas otras dos heurísticas sin llegar a logarlo.

```
alpha_to_num = {
  'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8
}

str_num = {
  '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8
}

def letra_a_num(str) -> int:

  for letra in alpha_to_num:
    if letra == str:
      return alpha_to_num[letra]

  return -1

def str_to_num(str) -> int:
  for letra in str_num:
    if letra == str:
      return str_num[letra]

  return -1

class Solution2(StudentHeuristic):
  def get_name(self) -> str:
    return "2301_15_sol2"

  def evaluation_function(self, state: TwoPlayerGameState) -> float:
    if not isinstance(state.game, Reversi):
      return 0
    max = 1
    moves = state.generate_successor()
    for move in moves:
      if len(move.generate_successor()) > max:
        max = len(move.generate_successor())

    return max

def not_corner(L) -> bool:
  vecino_esquina = [[1, 2], [2, 1], [1, 7], [2, 8], [7, 1], [8, 2], [7,
8], [8, 7]]

  if (L in vecino_esquina):
    return True
  return False
class Solution3(StudentHeuristic):
  def get_name(self) -> str:
    return "2301_15_sol3"
  def evaluation_function(self, state: TwoPlayerGameState) -> float:
    if not isinstance(state.game, Reversi):
```

```
      return 0
    game = state.game
    moves = game.generate_successors(state)

    cuatro = 10
    min_DB = 10
    min_DA = 10
    min_IB = 10
    min_IA = 10
    while moves:
      move = moves.pop().move_code

      if (move):
        y = letra_a_num(move[4])
        x = str_to_num(move[1])

        L = [x, y]

        if (not_corner(L) == True):
          return 20
        if (y == 1 and x == 1) or (y == 8 and x == 8) or (y == 1 and x ==
8) or (y == 8 and x == 1):
          return 1
        if y > 4 and x > 4:
          D_B = min(8 - x, 8 - y)
          if min_DB > D_B:
            min_DB = D_B

        if y < 4 and x > 4:
          D_A = min(8 - x, y)
          if min_DA > D_A:
            min_DA = D_A

        if y > 4 and x < 4:
          I_B = min(x, 8 - y)
          if min_IB > I_B:
            min_IB = I_B

        if y < 4 and x < 4:
          I_A = min(x, y)
          if min_IA > I_A:
            min_IA = I_A

        if y == 4 or x == 4:
          cuatro = min(x, y)

    return min(min_DA, min_IA, min_IB, min_DB, cuatro)

class Solution4(StudentHeuristic):
  def get_name(self) -> str:
    return "2301_15_sol4"

  def evaluation_function(self, state: TwoPlayerGameState) -> float:
    if not isinstance(state.game, Reversi):
      return 0
    corners = [[1, 'a'], [1, 'h'], [8, 'a'], [8, 'h']]
```

```
    minDist = 10
    game = state.game
    moves = game.generate_successors(state)

    while moves:
      move = moves.pop().move_code

      if(move):
        y = letra_a_num(move[4])
        x = str_to_num(move[1])

        L = [x, y]

        if (not_corner(L) == True):
          return 20
        for corner in corners:
          distX = abs(corner[0] - x)
          distY = abs(letra_a_num(corner[1]) - y)
          dist = math.sqrt(distX * distX + distY * distY)
          if dist < minDist:
            minDist = dist

    return minDist
```

Sección 7

Comentarios personales de la realización de esta práctica

Nota de la memoria (40% de la práctica)

Total de puntos (X/31.5)