به نام خدا

علی عطاءاللهی

پروژه سوم هوش مصنوعی

پاسخ سوالات تشریحی :

سوال اول ۱: یک heuristic خوب چه ویژگی‌هایی دارد؟ علت انتخاب heuristic شما و دلیل برتری آن نسبت به تعدادی از روش‌های دیگر را بیان کنید.

باید بین استیت های خوب و استیت های با ارزش کمتر تفاوت ایجاد کند و محاسبه آن هزینه بالایی نداشته باشد. موردی که بنده انتخاب کردم تعداد مثلث های آبی احتمالی – تعداد مثلث های قرمز احتمالی بود که دلیل آن این بود که هر چقد تعداد اولی بیشتر باشد به جواب مناسب نزدیک تر هستیم و هر چقدر دومی بیشتر باشد از جواب دور هستیم. برتری آن هم هزینه کم و تفاوتی بود که با استیت های دیگر ایجاد می‌کرد.

سوال ۲: آیا میان عمق الگوریتم و پارامترهای حساب شده روابطی می بینید؟ به طور کامل بررسی کنید که عمق الگوریتم چه تاثیراتی بر روی شانس پیروزی، زمان و گره‌های دیده شده می گذارد.

با توجه به خروجی‌های مشاهده شده، هر چقدر عمق بیشتر باشد، شانس پیروزی کاهش می‌یابد. دلیل این است که حریف ما شانسی عمل می‌کند ولی ما او را هوشمند در نظر می‌گیریم که باعث می‌شود بسیاری از حالت های مناسب و چند عدد از استیت‌های خوب را در نظر نگیریم.

هر چقدر عمق را بیشتر کنیم تعداد استیت های بیشتری را خواهیم دید و با توجه به توضیحات داده‌شده احتمال پیروزی ما کاهش می‌یابد. و با توجه به اینکه تعداد استیت های بیشتری می‌بینیم، زمان و حافظه مصرفی افزایش پیدا خواهد کرد.

سوال ۳: وقتی از روش هرس کردن استفاده می کنید، برای هر گره درخت، فرزندانش به چه ترتیبی اضافه می شوند؟ آیا این ترتیب اهمیت دارد؟ چرا این ترتیب را انتخاب کردید؟

فرزندانش به همان ترتیبی که نام‌گذاری شده‌اند اضافه می‌شوند ولیکن بهتر است که ترتیب بهتری داشته باشند. چون اگر ترتیب مناسبی داشته باشند به ترتیب بهتری هرس می‌شوند (در واقع اگر خطوطی را انتخاب کنیم که به راه حل اصلی نزدیک تر باشند، زود تر راه حل‌های دیگر را هرس می‌کنیم.)

گزارش کد :

minimax :

```
143     def minimax(self, depth, player_turn, state):
144         if depth == self.minimax_depth:
145             return self._evaluate(state)
146         value = 0
147         final_move = 0
148         ##########################################################################################################
149         if player_turn == 'red': # max
150             value = - math.inf
151             state.red.sort()
152 
153             for i in range (len(state.available_moves)):
154                 self.nodes_meeted += 1
155                 move = state.available_moves[i]
156                 if self.is_not_triangle(state.red, move[0], move[1]) == 0:
157                     if value < -END_GAME_VAL:
158                         value = -END_GAME_VAL
159                         final_move = move
160                     continue
161 
162                 available_moves_copy = copy.deepcopy(state.available_moves)
163                 available_moves_copy.pop(i)
164                 state.red.append(move)
165                 result_value = self.minimax(depth + 1, 'blue', State(state.red, state.blue, available_moves_copy))
166                 state.red.remove(move)
167                 if result_value >= value:
168                     value = result_value
169                     final_move = move
```

```
170         ##########################################################################################################
171         elif player_turn == 'blue' :  # min
172             value = math.inf
173             state.blue.sort()
174 
175             for i in range(len(state.available_moves)):
176                 self.nodes_meeted += 1
177                 move = state.available_moves[i]
178                 if self.is_not_triangle(state.blue, move[0], move[1]) == 0:
179                     if value > END_GAME_VAL:
180                         value = END_GAME_VAL
181                         final_move = move
182                     continue
183 
184                 available_moves_copy = copy.deepcopy(state.available_moves)
185                 available_moves_copy.pop(i)
186                 state.blue.append(move)
187                 result_value = self.minimax(depth + 1, 'red', State(state.red, state.blue, available_moves_copy))
188                 state.blue.remove(move)
189                 if result_value < value:
190                     value = result_value
191                     final_move = move
192         ##########################################################################################################
193         if  depth == 0:
194           return final_move
195         return value
```

```
    ########################################################################################################
    elif player_turn == 'blue' :  # min
        value = math.inf
        state.blue.sort()

        for i in range(len(state.available_moves)):
            self.nodes_meeted += 1
            move = state.available_moves[i]
            if self.is_not_triangle(state.blue, move[0], move[1]) == 0:
                if value > END_GAME_VAL:
                    value = END_GAME_VAL
                    final_move = move
                    if value <= alpha :
                        break
                    beta = min(beta, value)
                continue

            available_moves_copy = copy.deepcopy(state.available_moves)
            available_moves_copy.pop(i)
            state.blue.append(move)
            result_value = self.minimax_with_prune(depth + 1, 'red', State(state.red, state.blue, available_moves_copy), alpha, beta)
            state.blue.remove(move)
            if result_value < value:
                value = result_value
                final_move = move
                if value <= alpha :
                    break
                beta = min(beta, value)
    ########################################################################################################
    if  depth == 0:
      return final_move
    return value
```

: minimax_with_prune

```
def minimax_with_prune(self, depth, player_turn, state, alpha=-math.inf, beta=math.inf):
    if depth == self.minimax_depth:
        return self._evaluate(state)
    value = 0
    final_move = 0
    ########################################################################################################
    if player_turn == 'red':  # max
        value = - math.inf
        state.red.sort()

        for i in range (len(state.available_moves)):
            self.nodes_meeted += 1
            move = state.available_moves[i]
            if self.is_not_triangle(state.red, move[0], move[1]) == 0:
                if value < -END_GAME_VAL:
                    value = -END_GAME_VAL
                    final_move = move
                    if value >= beta :
                        break
                    alpha = max(alpha, value)
                continue

            available_moves_copy = copy.deepcopy(state.available_moves)
            available_moves_copy.pop(i)
            state.red.append(move)
            result_value = self.minimax_with_prune(depth + 1, 'blue', State(state.red, state.blue, available_moves_copy), alpha, beta)
            state.red.remove(move)
            if result_value >= value:
                value = result_value
                final_move = move
                if value >= beta :
                    break
                alpha = max(alpha, result_value)
```

تابع heuristic :

```python
    def is_not_triangle(self, nodes, i, j):
        for p in range(len(nodes)):
            for q in range(p+1,len(nodes)):
                if nodes[p][1] == nodes[q][1] or nodes[p][1] == nodes[q][0] or nodes[p][0] == nodes[q][0] :
                    points = set([nodes[p][0], nodes[p][1], nodes[q][0], nodes[q][1]])
                    if i in points and j in points:
                        return False
        return True

    def _evaluate(self, state):
        red_triangle = 0
        blue_triangle = 0
        state.red.sort()
        state.blue.sort()
        for (i,j) in state.available_moves:
            red_triangle += self.is_not_triangle(state.red, i, j)
            blue_triangle += self.is_not_triangle(state.blue, i, j)

        return -END_GAME_VAL if red_triangle == 0 else \
                END_GAME_VAL if blue_triangle == 0 else \
                red_triangle - blue_triangle
```

گزارش خروجی :

بدون prune :

```
C:\Users\ali18\Desktop\5\AI\CAs_and_HWs\CA3>python main.py 1 0
{'red': 100, 'blue': 0}
avg nodes meeted : 50.18
executed in 0.163 sec
```

```
C:\Users\ali18\Desktop\5\AI\CAs_and_HWs\CA3>python main.py 3 0
{'red': 100, 'blue': 0}
avg nodes meeted : 5393.14
executed in 16.400 sec
```

با prune :

```
C:\Users\ali18\Desktop\5\AI\CAs_and_HWs\CA3>python main.py 1 0
{'red': 100, 'blue': 0}
avg nodes meeted : 48.25
executed in 0.152 sec
```

```
C:\Users\ali18\Desktop\5\AI\CAs_and_HWs\CA3>python main.py 3 0
{'red': 95, 'blue': 5}
avg nodes meeted : 975.95
executed in 2.692 sec
```

```
C:\Users\ali18\Desktop\5\AI\CAs_and_HWs\CA3>python main.py 5 0
{'red': 92, 'blue': 8}
avg nodes meeted : 15734.89
executed in 46.521 sec
```