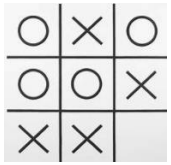
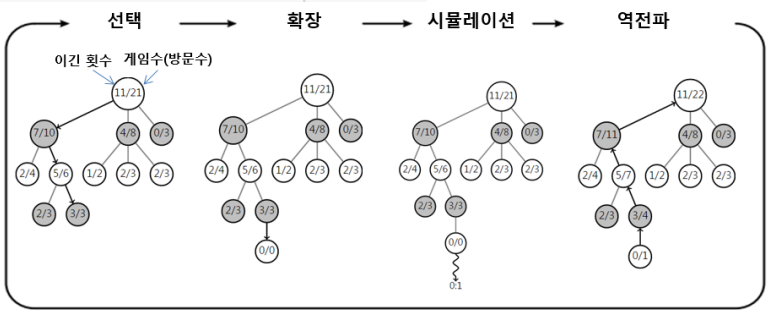
프로그래밍 실습 : 틱-택-토 MCTS

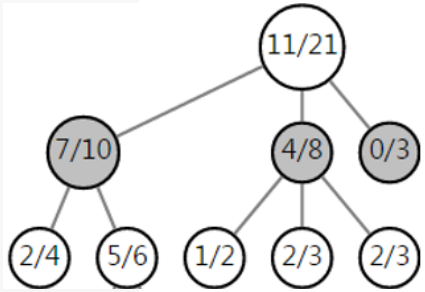
```python
from abc import ABC, abstractmethod    # abstract base class
from collections import defaultdict
import math

class MCTS:
    "Monte Carlo tree searcher. 먼저 rollout한 다음, 위치(move) 선택 "
    def __init__(self, c=1):
        self.Q = defaultdict(int)  # 노드별 이긴 횟수(reward) 값을 0으로 초기화
        self.N = defaultdict(int)  # 노드별 방문횟수(visit count)를 0으로 초기화
        self.children = dict()     # 노드의 자식노드
        self.c = c                 # UCT 계산에 사용되는 계수

    def choose(self, node):
        "node의 최선인 자식 노드 선택"
        if node.is_terminal():   # node가 단말인 경우 오류
            raise RuntimeError(f"choose called on terminal node {node}")
        if node not in self.children:  # node가 children에 포함되지 않으면 무작위 선택
            return node.find_random_child()

        def score(n):   # 점수 계산
            if self.N[n] == 0:
                return float("-inf")  # 한번도 방문하지 않은 노드인 경우 - 선택 배제
            return self.Q[n] / self.N[n]  # 평균 점수

        return max(self.children[node], key=score)

    def do_rollout(self, node):
        "게임 트리에서 한 층만 더 보기"
        path = self._select(node)
        leaf = path[-1]
        self._expand(leaf)
        reward = self._simulate(leaf)
        self._backpropagate(path, reward)
```





Original developer: Luke Harold Miles, 2019

```
35      def _select(self, node):  # 선택 단계
36          "node의 아직 시도해보지 않은 자식 노드 찾기"
37          path = []
38          while True:
39              path.append(node)
40              if node not in self.children or not self.children[node]:
41                  # node의 child나 grandchild가 아닌 경우: 아직 시도해보지 않은 것 또는 단말 노드
42                  return path
43              unexplored = self.children[node] - self.children.keys() # 차집합
44              if unexplored:
45                  n = unexplored.pop()
46                  path.append(n)
47                  return path
48              node = self._uct_select(node)  # 한 단계 내려가기
49
50      def _expand(self, node):  # 확장 단계
51          "children에 node의 자식노드 추가"
52          if node in self.children:
53              return  # 이미 확장된 노드
54          self.children[node] = node.find_children()  # 선택가능 move들을 node의 children에 추가
55
56      def _simulate(self, node): # 시뮬레이션 단계
57          "node의 무작위 시뮬레이션에 대한 결과(reward) 반환"
58          invert_reward = True
59          while True:
60              if node.is_terminal():  # 단말에 도달하면 승패여부 결정
61                  reward = node.reward()
62                  return 1 - reward if invert_reward else reward
63              node = node.find_random_child()    # 선택할 수 있는 것 중에서 무작위로 선택
64              invert_reward = not invert_reward
65
```



```
66      def _backpropagate(self, path, reward): # 역전파 단계
67          "단말 노드의 조상 노드들에게 보상(reward) 전달"
68          for node in reversed(path):  # 역순으로 가면서 Monte Carlo 시뮬레이션 결과 반영
69              self.N[node] += 1
70              self.Q[node] += reward
71              reward = 1 - reward  # 자신에게는 1 상대에게는 0, 또는 그 반대
72
73      def _uct_select(self, node): # UCB 정책 적용을 통한 노드 확장 대상 노드 선택
74          "탐험(exploration)과 이용(exploitation)의 균형을 맞춰 node의 자식 노드 선택"
75          # node의 모든 자신 노드가 이미 확장되었는지 확인
76          assert all(n in self.children for n in self.children[node])
77          log_N_vertex = math.log(self.N[node])
78
79          def uct(n):
80              "UCB(Upper confidence bound) 점수 계산 "
81              return self.Q[n] / self.N[n] + self.c * math.sqrt(2*log_N_vertex / self.N[n])
82
83          return max(self.children[node], key=uct)
84
```

$$\frac{Q(v')}{N(v')} + c\sqrt{\frac{2\ln N(v)}{N(v')}}$$

```
84
85 class Node(ABC):
86     " 게임 트리의 노드로서 보드판의 상태 표현 "
87     @abstractmethod
88     def find_children(self):  # 해당 보드판 상태의 가능한 모든 가능한 후속 상태
89         return set()
90
91     @abstractmethod
92     def find_random_child(self): # 현 보드에 대한 자식 노드 무작위 선택
93         return None
94
95     @abstractmethod
96     def is_terminal(self):   # 자식 노드인지 판단
97         return True
98
99     @abstractmethod
100    def reward(self):    # 점수 계산
101        return 0
102
103    @abstractmethod
104    def __hash__(self):  # 노드에 해시적용 가능하도록(hashable) 함
105        return 123456789
106
107    @abstractmethod
108    def __eq__(node1, node2):  # 노드는 서로 비교 가능해야 함
109        return True
```

```python
from collections import namedtuple
from random import choice
#from monte_carlo_tree_search import MCTS, Node

TTTB = namedtuple("TicTacToeBoard", "tup turn winner terminal")

class TicTacToeBoard(TTTB, Node):  # TTTB의 속성들도 상속
    def find_children(board): # 전체 가능한 move들 집합으로 반환
        if board.terminal:  # 게임이 끝나면 아무것도 하지 않음
            return set()
        return {   # 그렇지 않으면, 비어있는 곳에 각각 시도
            board.make_move(i) for i, value in enumerate(board.tup) if value is None
        }

    def find_random_child(board): # 무작위로 move 선택
        if board.terminal:
          return None  # 게임이 끝나면 아무것도 하지 않음
        empty_spots = [i for i, value in enumerate(board.tup) if value is None]
        return board.make_move(choice(empty_spots))

    def reward(board):  # 점수 계산
        if not board.terminal:
            raise RuntimeError(f"reward called on nonterminal board {board}")
        if board.winner is board.turn:
            # 자기 차례이면서 자기가 이긴 상황은 불가능
            raise RuntimeError(f"reward called on unreachable board {board}")
        if board.turn is (not board.winner):
            return 0  # 상대가 이긴 상황
        if board.winner is None:
            return 0.5  # 비긴 상황
        # 일어날 수 없는 상황
        raise RuntimeError(f"board has unknown winner type {board.winner}")

```

```
34      def is_terminal(board):  # 게임 종료 여부
35          return board.terminal
36
37      def make_move(board, index):   # index 위치에 board.turn 표시하기 하기
38          tup = board.tup[:index] + (board.turn,) + board.tup[index + 1 :]
39          turn = not board.turn      # 순서 바꾸기
40          winner = find_winner(tup)  # 승자 또는 미종료 판단
41          is_terminal = (winner is not None) or not any(v is None for v in tup)
42          return TicTacToeBoard(tup, turn, winner, is_terminal)  # 보드 상태 반환
43
44      def display_board(board):  # 보드 상태 출력
45          to_char = lambda v: ("X" if v is True else ("O" if v is False else " "))
46          rows = [
47              [to_char(board.tup[3 * row + col]) for col in range(3)] for row in range(3)
48          ]
49          return ("₩n  1 2 3₩n"
50              + "₩n".join(str(i + 1) + " " + " ".join(row) for i, row in enumerate(rows)) + "₩n")
51
```

```python
def play_game():  # 게임하기
    tree = MCTS()
    board = new_Board()
    print(board.display_board())
    while True:
        row_col = input("위치 row,col: ")
        row, col = map(int, row_col.split(","))
        index = 3 * (row - 1) + (col - 1)
        if board.tup[index] is not None:  # 비어있는 위치가 아닌 경우
            raise RuntimeError("Invalid move")
        board = board.make_move(index)    # index 위치의 보드 상태 변경
        print(board.display_board())
        if board.terminal:  # 게임 종료
            break

        for _ in range(50):   # 매번 50번의 rollout을 수행
            tree.do_rollout(board)
        board = tree.choose(board)  # 최선의 값을 갖는 move 선택하여 보드에 반영
        print(board.display_board())
        if board.terminal:
            print('게임 종료')
            break

```

```
  1 2 3
1
2
3
위치 row,col: 1,1

  1 2 3
1 X
2
3
위치 row,col: 1,2

  1 2 3
1 X
2
3     0
위치 row,col: 1,3

  1 2 3
1 X X
2
3     0
위치 row,col: 2,2

  1 2 3
1 X X 0
2
3     0
위치 row,col: 3,1

  1 2 3
1 X X 0
2
3 X   0
위치 row,col: 2,3

  1 2 3
1 X X 0
2     0
3 X   0
게임 종료
```

```
75 def winning_combos():  # 이기는 배치 조합
76     for start in range(0, 9, 3):  # 행에 3개 연속
77         yield (start, start + 1, start + 2)
78     for start in range(3):  # 열에 3개 연속
79         yield (start, start + 3, start + 6)
80     yield (0, 4, 8)  # 오른쪽 아래로 가는 대각선 3개
81     yield (2, 4, 6)  # 왼쪽 아래로 가는 대각선 3개
82 
83 def find_winner(tup): # X가 이기면 True, O가 이기면 False, 미종료 상태이면 None 반환
84     for i1, i2, i3 in winning_combos():
85         v1, v2, v3 = tup[i1], tup[i2], tup[i3]
86         if False is v1 is v2 is v3:
87             return False
88         if True is v1 is v2 is v3:
89             return True
90     return None
91 
92 def new_Board(): # 비어있는 보드판 생성
93     return TicTacToeBoard(tup=(None,) * 9, turn=True, winner=None, terminal=False)
94 
95 if __name__ == "__main__":
96     play_game()
```