산업인공지능 개론

Home Work #1

Monte Carlo Tree Search
Python 코드, 실행결과

학과 : 산업인공지능학과
학번 : 2024254022
이름 : 정현일

2024.03.14.

monte_carlo_tree_search.py (1/2)

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
산업인지능 개론

HW1
tic-tac-toe에 Monte Carlo Tree Search를 적용한 프로그램을 직접 실행해 보고 결과 화면을 올리기

학번 : 2024254022
이름 : 정현일


Created on Tue Mar 12 09:16:59 2024

@author: chohi
"""

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():      # 노드가 단말인 경우 오류
            raise RuntimeError(f"choose called on terminal node {node}")

        if node not in self.children:   # 노드가 childre에 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)
        self._backprogagae(path, reward)

    def _select(self, node):    # 서너택 단계
        "node의 아직 시도해보지 않은 자식 노드 찾기"
        path = []
        while True:
            path.append(node)
            if node not in self.children or not self.children[node]:
                # node 의 child나 grandchild가 아닌 경우: 아직 시도해보지 않은 것 또는 단말 노드
                return path
            unexplored = self.children[node] - self.children.keys()     # 차집합
            if unexplored:
                n = unexplored.pop()
                path.append(n)
                return path
            node = self._uct_select(node)   # 한 단계 내려가기

    def _expand(self, node):    # 확장 단계
        "children에 node의 자식노드 추가"
        if node in self.children:
            return  # 이미 확장된 노드
        self.children[node] = node.find_children()  # 선택가능 move들을 node의 children에 추가

    def _simulate(self, node):  # 시뮬레이션 단계
        "node의 무작위 시뮬레이션에 대한 결과(reward) 반환"
        invert_reward = True
        while True:
            if node.is_terminal():
                reward = node.reward()
                return 1 - reward if invert_reward else reward
            node = node.find_random_child()     # 선택할 수 있는 것 중에서 무작위로 선택
            invert_reward = not invert_reward

    def _backprogagae(self, path, reward):  # 역전파 단계
        "단말 노드의 조상 노드들에게 보상(reward) 전달"
        for node in reversed(path):     # 역순으로 가면서 Monte Carilo 시뮬레이션 결과 반영
            self.N[node] += 1
            self.Q[node] += reward
            reward = 1 - reward     # 자신에게는 1 상대에게는 0, 또는 그 반대

    def _uct_select(self, node):    # UCB 정책 적용을 통한 노드 확장 대상 노드 선택
        "탐험(exploration)과 이용(exploitation)의 균형을 맞춰 node의 자식 노드 선택"
        # node의 모든 자신 노드가 이미 확장되었는지 확인
        assert all(n in self.children for n in self.children[node])
        log_N_vertex = math.log(self.N[node])

        def uct(n):
            "UCB(Upper confidence bound) 점수 계산 "
            return self.Q[n] / self.N[n] + self.c * math.sqrt(2*log_N_vertex / self.N[n])
```

monte_carlo_tree_search.py (2/2)

```
101
102             return max(self.children[node], key=uct)
103
104
105
106     class Node(ABC):
107         "게임 트리의 노드로서 보드판의 상태 표현"
108
109         @abstractmethod
110         def find_children(self):          # 해당 보드판 상태의 가능한 모
111             return set()
112
113         @abstractmethod
114         def find_random_child(self):      # 현 보드에 대한 자식 노드 무작
115             return Node
116
117         @abstractmethod
118         def is_terminal(self):            # 자식 노드인지 판단
119             return True
120
121         @abstractmethod                   # 점수 계산
122         def reward(self):
```

tic_tac_toe.py (1/2)

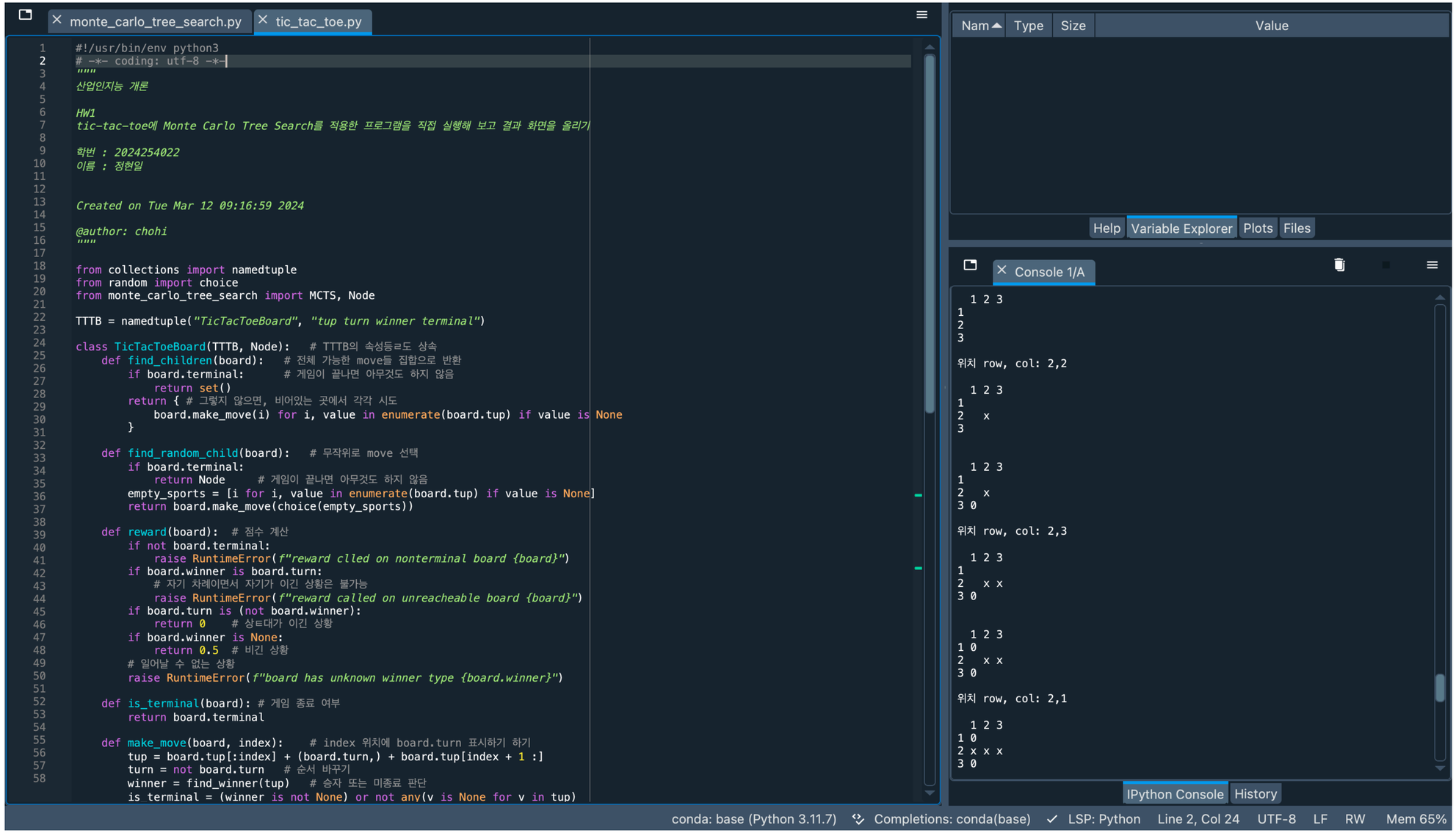
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
산업인지능 개론

HW1
tic-tac-toe에 Monte Carlo Tree Search를 적용한 프로그램을 직접 실행해 보고 결과 화면을 올리기

학번 : 2024254022
이름 : 정현일


Created on Tue Mar 12 09:16:59 2024

@author: chohi
"""

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 속성등ㄹ도 상속
    def find_children(board):   # 전체 가능한 move들 집합으로 반환
        if board.terminal:      # 게임이 끝나면 아무것도 하지 않음
            return set()
        return { # 그렇지 않으면, 비어있는 곳에서 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 Node     # 게임이 끝나면 아무것도 하지 않음
        empty_sports = [i for i, value in enumerate(board.tup) if value is None]
        return board.make_move(choice(empty_sports))

    def reward(board):  # 점수 계산
        if not board.terminal:
            raise RuntimeError(f"reward clled on nonterminal board {board}")
        if board.winner is board.turn:
            # 자기 차례이면서 자기가 이긴 상황은 불가능
            raise RuntimeError(f"reward called on unreacheable board {board}")
        if board.turn is (not board.winner):
            return 0    # 상ㅌ대가 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}")
```

```python
    def is_terminal(board): # 게임 종료 여부
        return board.terminal

    def make_move(board, index):    # index 위치에 board.turn 표시하기 하기
        tup = board.tup[:index] + (board.turn,) + board.tup[index + 1 :]
        turn = not board.turn   # 순서 바꾸기
        winner = find_winner(tup)   # 승자 또는 미종료 판단
        is_terminal = (winner is not None) or not any(v is None for v in tup)
        return TicTacToeBoard(tup, turn, winner, is_terminal) # 보드 상태 반환

    def display_board(board):   # 보드 상태 출력
        to_char = lambda v: ("x" if v is True else ("0" if v is False else " "))
        rows = [
            [to_char(board.tup[3 * row + col]) for col in range(3)] for row in range(3)
            ]
        return ("\n  1 2 3\n" + "\n".join(str(i+1) + " " + " ".join(row)
                                          for i, row in enumerate(rows)) + "\n")

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

def winning_combos():   # 이기는 배치 조합
    for start in range(0, 9, 3):    # 행에 3개 연속
        yield(start, start + 1, start + 2)
    for start in range(3):  # 열에 3개 연속
        yield(start, start + 3, start + 6)
    yield(0, 4, 8) # 오른쪽 아래로 가는 대각선 3개
    yield(2, 4, 6) # 왼쪽 아래로 가는 대각선 3개
```

tic_tac_toe.py (2/2)

```
101
102
103 def find_winner(tup):   # X가 이기면 True, 0가 이기면 False, 미종료 상태이면 None 반환
104     for i1, i2, i3 in winning_combos():
105         v1, v2, v3 = tup[i1], tup[i2], tup[i3]
106         if False is v1 is v2 is v3:
107             return False
108         if True is v1 is v2 is v3:
109             return True
110     return None
111
112
113 def new_Board(): # 비어있는 보다판 생성
114     return TicTacToeBoard(tup=(None,) * 9, turn=True, winner=None, terminal=False)
115
116 if __name__ == "__main__":
    play_game()
```

실행 결과


monte_carlo_tree_search.py
tic_tac_toe.py
#!/usr/bin/env python3
-*- coding: utf-8 -*-
"""
산업인지능 개론
HW1
tic-tac-toe에 Monte Carlo Tree Search를 적용한 프로그램을 직접 실행해 보고 결과 화면을 올리기
학번 : 2024254022
이름 : 정현일
Created on Tue Mar 12 09:16:59 2024
@author: chohi
"""
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 속성들ㄹ도 상속
def find_children(board): # 전체 가능한 move들 집합으로 반환
if board.terminal: # 게임이 끝나면 아무것도 하지 않음
return set()
return { # 그렇지 않으면, 비어있는 곳에서 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 Node # 게임이 끝나면 아무것도 하지 않음
empty_sports = [i for i, value in enumerate(board.tup) if value is None]
return board.make_move(choice(empty_sports))
def reward(board): # 점수 계산
if not board.terminal:
raise RuntimeError(f"reward clled on nonterminal board {board}")
if board.winner is board.turn:
# 자기 차례이면서 자기가 이긴 상황은 불가능
raise RuntimeError(f"reward called on unreacheable board {board}")
if board.turn is (not board.winner):
return 0 # 상ㅌ대가 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}")
def is_terminal(board): # 게임 종료 여부
return board.terminal
def make_move(board, index): # index 위치에 board.turn 표시하기 하기
tup = board.tup[:index] + (board.turn,) + board.tup[index + 1 :]
turn = not board.turn # 순서 바꾸기
winner = find_winner(tup) # 승자 또는 미종료 판단
is_terminal = (winner is not None) or not any(v is None for v in tup)
Nam
Type
Size
Value
Help
Variable Explorer
Plots
Files
Console 1/A
1 2 3
1
2
3
위치 row, col: 2,2
1 2 3
1
2 x
3
1 2 3
1
2 x
3 0
위치 row, col: 2,3
1 2 3
1
2 x x
3 0
1 2 3
1 0
2 x x
3 0
위치 row, col: 2,1
1 2 3
1 0
2 x x x
3 0
IPython Console
History
conda: base (Python 3.11.7)
Completions: conda(base)
LSP: Python
Line 2, Col 24
UTF-8
LF
RW
Mem 65%