Lab5

September 22, 2021

106119029
Lab 5 AI/ML

Write a program in Python to implement Tic-tac-toe game simulation using Minmax and reinforcement algorithms.

Collab link

This is board class. This is inheriting from object class.

The `winner()` method checks if we can find a winner in board. It check if there's a winner horizontally(by calling `horizontal_winner()`) or vertically(by calling `vertical_winner()`) or diagonally (by calling `diagonal_winner()`).

There's `play` method which will place X or O in given position.

[21]:
```
class Board(object):
        def __init__(self, inner=None):
                self.inner = inner or [None for x in range(9)]

        def play(self, position, player):
                assert(0 <= position < len(self.inner))
                if self.inner[position]:
                        return False
                self.inner[position] = player
                return True

        def get(self, position):
                assert(0 <= position < len(self.inner))
                return self.inner[position]

        def winner(self):
                return self.horizontal_winner() or self.vertical_winner() or␣
 ↪self.diagonal_winner()

        def horizontal_winner(self):
                I = self.inner
                for i in range(0,9,3):
                        if I[i] and I[i] == I[i+1] and I[i] == I[i+2]:
                                return I[i]
        def vertical_winner(self):
                I = self.inner
                for i in range(3):
```

```
                if I[i] and I[i] == I[i+3] and I[i] == I[i+6]:
                        return I[i]
        def diagonal_winner(self):
                I = self.inner
                if I[0] and I[0] == I[4] and I[0] == I[8]:
                        return I[0]
                if I[2] and I[2] == I[4] and I[2] == I[6]:
                        return I[2]

        def full(self):
                return all(self.inner)

        def __str__(self):
                acc = []
                for i, n in enumerate(self.inner):
                        if acc:
                                acc.append(i%3 and ' ' or '\n')
                        acc.append(str(n or i+1))
                return ''.join(acc)
        def copy(self):
                return Board(self.inner[:])
```

[22]:
```
class Player(object):
        def __init__(self, color, name):
                self.color = color
                self.name = name
        def __str__(self):
                return self.color
        def play(self, board):
                raise NotImplementedError
```

[23]:
```
class Human(Player):
        def play(self, board):
                print(board)
                i = 0
                while i < 1 or i > 9 or board.get(i - 1):
                        i = input(self.name + ": ")
                        i = i.isdigit() and int(i) or 0
                board.play(i - 1, self)
```

[24]:
```
class AI(Player):
        other = Player(None, None)
        def play(self, board):
                if not self.other.name:
                        for i in range(9):
                                x = board.get(i)
                                if x and x != self:
                                        self.other = x
                                break
                for I in range(1, -3, -1):
                        for i in range(9):
                                if not board.get(i):
                                        b = board.copy()
                                        b.play(i, self)
                                        if self.minimax(b, min) > I:
                                                board.play(i, self)
                                                return
        def minimax(self, board, f):
                if board.winner():
                        return board.winner() == self and 1 or -1
                if board.full():
                        return 0
                m = f == min and 1 or -1
                for p in range(9):
                        if not board.get(p):
                                b = board.copy()
                                b.play(p, f == min and self.other or self)
                                m = f(m, self.minimax(b, f == min and max or␣
 ↪min))
                                if m == (f == min and -1 or 1):
                                        return m
                return m
```

[25]:
```
class Heuristic(AI):
        def play(self, board):
                if not self.other.name:
                        for i in range(9):
                                x = board.get(i)
                                if x and x != self:
                                        self.other = x
                                        break
                board.play(self.findmove(board), self)
        def findmove(self, board):
                if not any(board.inner): #empty board
                        return 0

                for I in range(1, -3, -1): #default to minimax
                        for i in range(9):
                                if not board.get(i):
                                        b = board.copy()
                                        b.play(i, self)
                                        if self.minimax(b, min) > I:
                                                return i
```

[26]:
```
class Game(object):
        def __init__(self, *players):
                self.players = list(players)
                self.board = Board()
        def play(self):
                while not self.board.winner() and not self.board.full():
                        p = self.players.pop(0)
                        p.play(self.board)
                        self.players.append(p)
                if self.board.winner():
                        print(self.board.winner().name, "wins!")
                else:
                        print("Draw.")
                print(self.board)
```

[27]:
```
Game(Human('X', "Human"), Heuristic('O', "AI")).play()
```

```
1 2 3
4 5 6
7 8 9
Human: 3
1 2 X
4 O 6
7 8 9
Human: 6
1 2 X
4 O X
7 8 O
Human: 1
X O X
4 O X
7 8 O
Human: 8
X O X
O O X
7 X O
Human: 7
Draw.
X O X
O O X
X X O
```

[28]:
```
!cp drive/My\ Drive/Colab\ Notebooks/Lab5.ipynb ./
```

[29]:
```
!jupyter nbconvert --to PDF "Lab5.ipynb"
```

```
[NbConvertApp] Converting notebook Lab5.ipynb to PDF
[NbConvertApp] Writing 58324 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: [u'xelatex', u'./notebook.tex',
'-quiet']
[NbConvertApp] Running bibtex 1 time: [u'bibtex', u'./notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 43757 bytes to Lab5.pdf
```

[ ]:

```
```