AI Capstone HW3

110550088 李杰穎

May 7, 2023

In this homework, we're asked to write a minesweeper game and use logical inference to build an AI that automatically play minesweeper game.

I wrote this homework using Python, in this report, I will first introduce the modules I create for this homework including module for literal, clause, knowledge base, game and player. And then talk about how to use logical inference technique, especially in this homework, resolution, to develop a AI for this game.

1 Game Module

The code of Game Module can be referred to the appendix, Code 1. In this module, I implement some useful functions that will be later used in the development.

The `__init__` function initialize

2 Player Module

Appendix A Code of Game Module

Code 1: Game Module

```python
class Game:
    def __init__(self, difficulty=0):
        board_configurations = [
            (9, 9, 10),
            (16, 16, 25),
            (16, 30, 99)
        ]
        self.h, self.w, self.num_of_mines = board_configurations[difficulty]
        self.board = [[0 for _ in range(self.w)] for _ in range(self.h)]
        self.shown_cell = [[False for _ in range(self.w)] for _ in range(self.h)]
        self.mine_pos = set()
        self.found_mines = set()

        while len(self.mine_pos) < self.num_of_mines:
            i = random.randrange(self.h)
            j = random.randrange(self.w)
            if (i, j) not in self.mine_pos:
                self.mine_pos.add((i, j))
                self.board[i][j] = -1

    def open_cell(self, cell, safe):
        if ((cell in self.mine_pos) ^ (not safe)) or self.shown_cell[cell[0]][cell[1]]:
            return -1
        if cell not in self.mine_pos:
            self.board[cell[0]][cell[1]] = self.get_surround_mines(cell)
        else:
            self.board[cell[0]][cell[1]] = ”X”

        self.shown_cell[cell[0]][cell[1]] = True

        return self.board[cell[0]][cell[1]]

    def cell_status(self, cell):
        return self.board[cell[0]][cell[1]]

    def get_hint(self, cell):
        cnt = 0
        res = []
        for i in range(cell[0]-1, cell[0]+2):
            for j in range(cell[1]-1, cell[1]+2):
                if i < 0 or i >= self.h or j < 0 or j >= self.w:
                    continue
                if self.shown_cell[i][j]:
                    continue
                if (i, j) != cell:
                    if (i, j) in self.mine_pos:
                        cnt += 1
                    res.append((i, j))
        return res, cnt

    def get_surround_mines(self, cell):
        cnt = 0
        for i in range(cell[0]-1, cell[0]+2):
            for j in range(cell[1]-1, cell[1]+2):
                if (i, j) in self.mine_pos:
                    cnt += 1
        return cnt

    def is_mine(self, cell):
        return cell in self.mine_pos

    def check_win(self):
        return self.found_mines == self.mine_pos

    def get_init_safe_cells(self):
        num = round(math.sqrt(self.h * self.w))
        # num = 10
        init_cells = set()
        while len(init_cells) < num:
            i = random.randrange(self.h)
            j = random.randrange(self.w)
            if (i, j) not in self.mine_pos and (i, j) not in init_cells:
                init_cells.add((i, j))

        return init_cells

    def print_board(self):
        os.system('cls')
        for i in range(self.h):
            for j in range(self.w):
                if self.shown_cell[i][j]:
                    print(self.board[i][j], end=' ')
                else:
                    print('?', end=' ')
            print()
```