Analisi di un programma del Gioco della vita in Python

Il Gioco della vita è un automa cellulare ideato dal matematico John Conway nel 1969 ispirato dai lavori di John Von Neumann e Stanislaw Ulam. Segue quattro semplici regole per emulare la vita reale seguendo principi di sovrappopolazione, sottopopolazione e riproduzione. Le quattro leggi sono le seguenti:

1. Qualsiasi cella viva con meno di due celle vive adiacenti muore
2. Qualsiasi cella viva con due o tre celle vive adiacenti sopravvive
3. Qualsiasi cella viva con più di tre celle vive adiacenti muore
4. Qualsiasi cella morta con esattamente tre celle vive adiacenti diventa una cella viva

Per rendere più leggibile il programma lo divideremo in tre file: **cell.py**, **board.py** e **main.py**. Il primo si occuperà di definire tutte le caratteristiche della cellula, il secondo della griglia e l'ultimo metterà tutto insieme per fornire l'output

cell.py

```
class Cell:                         #creiamo la classe cellula con la quale determiniamo
    def __init__(self):             #tutti gli stati possibili e funzioni legati a essi
        self._status = 'Dead'

    def set_dead(self):             #imposta lo stato della cellula a morto
        self._status = 'Dead'

    def set_alive(self):            #imposta lo stato della cellula a vivo
        self._status = 'Alive'

    def is_alive(self):             #controlla se la cellula è viva. se lo è ritorna vero, se
        if self._status == 'Alive': #non lo è ritorna falso
            return True
        return False

    def get_print_character(self):  #facciamo ritornare, secondo lo stato della nostra
        if self.is_alive():         #cellula, un carattere di nostra scelta
            return 'O'
        return '.'
```

board.py

```
from cell import Cell
from random import randint

class Board:
    def __init__(self , rows , columns):  #queste variabili saranno poi scelte dall'utente
        self._rows = rows
        self._columns = columns
        self._grid = [[Cell() for column_cells in range(self._columns)] for row_cells in
range(self._rows)]

        self._generate_board()    #è posizionata nel costruttore così quando verrà
                                  #eventualmente creata l'istanza board in main.py le
                                  #cellule saranno subito decise


def draw_board(self):             #funzione per stampare la griglia
        print('\n'*10)            #pulisce il terminale
        print('printing board')
        for row in self._grid:                                  #usando due cicli stampa
            for column in row:                                  #prima ogni riga poi ogni
                                                                 #colonna

            print (column.get_print_character(),end='')
            print ()

    def _generate_board(self):            #funzione per determinare lo stato di ogni
        for row in self._grid:            #cellula
            for column in row:
                chance number = randint(0,2)   #c'è una possibilità su tre che la cellula
                if chance_number == 1:         #nasca viva
                    column.set_alive()

 def check_neighbour(self, check_row , check_column):   #funzione per ottenere le
        search_min = -1                                  #coordinate delle cellule vive
        search_max = 2
```

Per capire appieno questa funzione bisogna fare un esempio. Ogni cellula in questo programma ha due coordinate: x e y. Se prendiamo A in questa tabella ha come coordinate 0,0. Quello che i due cicli successivi faranno è creare le coordinate delle cellule vicine e controllare se sono validi o no. Quindi essenzialmente se stanno dentro la griglia.
Ora con search min e search max troveremo queste coordinate aggiungendoli alle coordinate x e y della cellula scelta. Così sarà ovvio vedere se abbiamo preso una cellula non esistente fuori dalla griglia in quanto x o y saranno negative. Sempre prendendo per esempio A con coordinate (0,0) quando aggiungendo search min otteniamo la cellula (-1,0) che ovviamente è fuori dalla griglia. Invece se a B(1,0) aggiungiamo search min otteniamo (0,0) che sono le coordinate di un vicino esistente e quindi viene aggiunto alla lista dei vicini validi.
Va tenuto a mente che questa funzione non controlla se sono vivi o no ma solo le coordinate dei vicini esistenti

A(0,0)	**B(1,0)**	**C(2,0)**
D(0,1)	**E(1,1)**	**F(2,1)**
G(0,2)	**H(1,2)**	**I(2,2)**

```
        neighbour_list = []                                  #lista vuota a cui aggiungeremo le
                                                             #coordinate dei vicini
        for row in range(search_min,search_max):
            for column in range(search_min,search_max):
                neighbour_row = check_row + row       #troviamo le coordinate come ho
                neighbour_column = check_column + column    #spiegato prima

                valid_neighbour = True   #valid_neighbour è sempre vero così se questi if
                                         #passano quello finale ritorna sempre e aggiunge
                                         #le coordinate alla lista
                if (neighbour_row) == check_row and (neighbour_column) ==
check_column:
                    valid_neighbour = False  #questo if controlla se abbiamo preso la cellula
                                             #di cui stiamo controllando i vicini

                if (neighbour_row) < 0 or (neighbour_row) >= self._rows:
                    valid_neighbour = False

                if (neighbour_column) < 0 or (neighbour_column) >= self._columns:
                    valid_neighbour = False  #questo if e quello precedente controllano se
                                             #abbiamo preso cellule fuori dalla griglia o
                                             #se abbiamo preso a cellula stessa
```

```
            if valid_neighbour:
                neighbour_list.append(self._grid[neighbour_row][neighbour_column])
            return neighbour_list


    def update_board(self):            #funzione per aggiornare la griglia ogni
                                       #generazione

        goes_alive = []                #creiamo due liste che contengono le coordinate
        gets_killed = []               #delle cellule vive e di quelle morte

        for row in range(len(self._grid)):              #con questi cicli percorriamo tutte le
            for column in range(len(self._grid[row])): #cellule e i loro vicini, vivi o morti
                check_neighbour = self.check_neighbour(row , column)

                living_neighbours_count = []      #creiamo una lista nella quale metteremo
                                                  #le coordinate delle cellule vicine vive

                for neighbour_cell in check_neighbour:   #controlliamo quali cellule sono
                    if neighbour_cell.is_alive():        #vive e le aggiungiamo alla lista
                        living_neighbours_count.append(neighbour_cell)

                cell_object = self._grid[row][column]     #prendiamo in considerazione una
                status_main_cell = cell_object.is_alive() #cellula e controlliamo se è viva
.
                if status_main_cell == True:      #applichiamo le leggi del Gioco della vita
                    if len(living_neighbours_count) < 2 or len(living_neighbours_count) > 3:
                        gets_killed.append(cell_object)

                    if len(living_neighbours_count) == 3 or len(living_neighbours_count) ==
2:
                        goes_alive.append(cell_object)

                else:
                    if len(living_neighbours_count) == 3:
                        goes_alive.append(cell_object)

        for cell_items in goes_alive:  #tutte le cellule che sono state poste in goes_alive
            cell_items.set_alive()     #diventano vive

        for cell_items in gets_killed: #tutte le cellule che sono state poste in gets_killed
            cell_items.set_dead()      #diventano morte
```

Main.py

```
from board import Board

def main():
    user_rows = int(input('how many rows? '))           #l'utente sceglie le grandezze
    user_columns = int(input('how many columns? ')) #della griglia

    game_of_life_board = Board(user_rows,user_columns) #creiamo un'istanza della
                                                       #classe board con i dati
                                                       #appena inseriti

    game_of_life_board.draw_board()   #facciamo partire la prima generazione
    game_of_life_board.update_board()

    user_action = ''
    while user_action != 'q':          #diamo all'utente l'opzione di uscire con q o di fare
                                       #una nuova generazione con invio
        user_action = input('premi invio per continuare la simulazione oppure q per
uscire')

        if user_action == '':
            game_of_life_board.update_board()
            game_of_life_board.draw_board()

main()   #chiamiamo la funzione main facendo partire il programma
```