Minesweeper

Have you ever played a game of minesweeper? It is provided in an operating system the name of which escapes us. Oh well, the intention of the game is to discover where all mines are hidden in a rectangular $m\times n$ field with m rows and n columns. To make things a little more easy, the numbers are given in the boxes of the field. These indicate the number of mines that are hidden in adjacent boxes. Here, both horizontally, vertically and diagonally adjacent boxes count. Suppose, for example, that two mines are hidden in a 4×5 field (their location is indicated with an asterisk):

```
*....
..*..
.....
.....
```

If we were to indicate the amount of mines in adjacent boxes in the empty boxes (indicated with a full stop), we would obtain the following representation of the field:

```
*2110
12*10
01110
00000
```

As you can see, every box has a maximum of 8 adjacent boxes.

Assignment

Define a class Minesweeper which can be used to represent a field of a game of minesweeper. This class must support the following methods:

- An initializing method __init__ that sees to an initial filling-in of the field as a rectangular $m \times n$ grid with m rows and n columns. The standard field is a rectangular grid that consists of 8 rows and 8 columns and contains no mines. However, three optional parameters can be given to the initializing method: a parameter rows that indicates the number of rows of the field (standard value 8), a parameter columns that indicates the numbers of columns in the field (standard value 8) and a parameter mines: a list of tuples that indicates where the mines of the field are hidden. Here, every tuple (x, y) indicates that a mine is hidden in the grid on row x and column y. Rows and columns are always numbered from 0. The given field must always consist of at least 2 rows and 2 columns. Look at the example below to verify how the initializing method must react if one of these conditions is not met. Also, the action the initializing method should undertake if a given position for the mine is not within the field, is given in the example.
- Methods rows and columns that respectively print the number of rows and the number of columns in the field.
- A method isMine that prints a Boolean value as a result. This value indicates whether or not a mine was placed on a given position in the field. The co-ordinates of the given position must be given to the method as an argument.
- Methods addMine and eraseMine that respectively add or erase a mine on a given position in the field. The co-ordinates of the given position must be given to the method as arguments.
- A method nearbyMines that indicates the number of mines on nearby positions for a given

position of the field. The co-ordinates of the given position must be given to the method as arguments.

- A method __repr__ that prints a string representation of the field in the format of a valid Python expression that can be used to construct a new field for the game minesweeper that has the same state as the current object. This string representation has the format Minesweeper(r, c, list), where r and c represent the rows and columns of the field. The list of tuples indicates the positions of the mines on the field, listed from left to right and from top to bottom.
- A method __str__ that prints a string representation of the field, where every row of the grid is represented as an independent line of text. Positions of the field on which mines are situated are indicated with an asterisk (*) and positions where no mine is situated are indicated with a digit that indicates the number of adjacent boxes that have a mine.

Methods to which a position on the field must be given (isMine, addMine, eraseMine and nearbyMines), must always verify whether this position is situated within the boundaries of the field. If this should not be the case, an AssertionError must be raised with an appropriate message as indicated in the example below.

Example

```
>>> field = Minesweeper(4, 5, [(0, 0), (1, 2)])
>>> print(field)
*2110
12*10
01110
00000
>>> field.rows()
4
>>> field.columns()
5
>>> field.isMine(0, 0)
True
>>> field.nearbyMines(1, 0)
1
>>> field.nearbyMines(1, 1)
2
>>> field.addMine(3, 3)
>>> print(field)
*2110
12*10
01221
001*1
>>> field.eraseMine(0, 0)
>>> print(field.isMine(0, 0))
False
>>> print(field)
01110
01*10
01221
001*1
>>> field
Minesweeper(4, 5, [(1, 2), (3, 3)])

>>> field.isMine(4, 4)
Traceback (most recent call last):
```

```
AssertionError: invalid position (4, 4)
>>> speelveld = Minesweeper(1, 1)
Traceback (most recent call last):
AssertionError: field must contain at least two rows
>>> speelveld = Minesweeper(2, 1)
Traceback (most recent call last):
AssertionError: field must contain at least two columns
```

Heb je ooit het spelletje mijnenveger gespeeld? Het wordt meegeleverd met een besturingssysteem waarvan de naam ons nu niet direct te binnen schiet. Nu ja, de bedoeling van het spel is te ontdekken waar alle mijnen verborgen liggen in een rechthoekig $m\times n$ speelveld met m rijen en n kolommen. Om je te helpen, worden getallen in de hokjes van het speelveld weergegeven. Deze geven aan hoeveel mijnen er in aangrenzende hokjes liggen. Hierbij worden zowel horizontaal, verticaal als diagonaal aangrenzende hokjes in rekening gebracht. Veronderstel bijvoorbeeld dat er in een 4×5 speelveld twee mijnen verborgen liggen (hun locatie wordt aangegeven met een sterretje):

```
*....
..*..
.....
.....
```

Indien we nu in de lege hokjes (aangegeven met een puntje) aangeven hoeveel mijnen er in aangrenzende hokjes liggen, dan krijgen we de volgende voorstelling van het speelveld:

```
*2110
12*10
01110
00000
```

Zoals je al zou kunnen opgemerkt hebben, heeft elk hokje ten hoogste 8 aangrenzende hokjes.

Opgave

Definieer een klasse `Mijnenveger` waarmee het speelveld van een spelletje mijnenveger kan voorgesteld worden. Deze klasse moet ondersteuning bieden aan de volgende methoden:

- Een initialisatiemethode `__init__` die zorgt voor een initiële invulling van het speelveld als een rechthoekig $m \times n$ rooster met m rijen en n kolommen. Het standaard speelveld is een rechthoekig rooster dat bestaat uit 8 rijen en 8 kolommen en dat geen mijnen bevat. Er kunnen aan de initialisatiemethode echter nog drie optionele parameters doorgegeven worden: een parameter `rijen` die het aantal rijen van het speelveld aangeeft (standaardwaarde 8), een parameter `kolommen` die het aantal kolommen van het speelveld aangeeft (standaardwaarde 8) en een parameter `mijnen`: een lijst van tuples die aangeven waar er mijnen in het speelveld verborgen zitten. Elk tuple (x, y) geeft hierbij aan dat er een mijn verborgen zit in het rooster op rij x en kolom y. Rijen en kolommen worden steeds genummerd vanaf 0. Het opgegeven spelbord moet steeds minimaal 2 rijen en 2 kolommen bevatten. Bekijk onderstaand voorbeeld om na te gaan hoe de initialisatiemethode moet reageren indien niet aan deze voorwaarde voldaan is. In het voorbeeld wordt ook aangegeven welke actie de initialisatiemethode moet ondernemen indien een opgegeven positie voor een mijn niet binnen het spelbord gelegen is.
- Methoden `rijen` en `kolommen` die respectievelijk het aantal rijen en het aantal kolommen van het spelbord teruggeven.

- Een methode isMijn die een Booleaanse waarde als resultaat teruggeeft. Deze waarde geeft aan of op een gegeven positie van het speelveld een mijn geplaatst werd of niet. De coördinaten van de opgegeven positie moeten als argumenten aan de methode doorgegeven worden.
- Methoden mijnToevoegen en mijnVerwijderen die respectievelijk een mijn toevoegen of verwijderen op een gegeven positie van het speelveld. De coördinaten van de opgegeven positie moeten als argumenten aan de methode doorgegeven worden.
- Een methode naburigeMijnen die voor een gegeven postie van het speelveld aangeeft hoeveel mijnen er op naburige posities gelegen zijn. De coördinaten van de opgegeven positie moeten als argumenten aan de methode doorgegeven worden.
- Een methode __repr__ die een stringvoorstelling van het speelveld teruggeeft onder de vorm van een geldige Python expressie die kan gebruikt worden om een nieuw speelveld voor het spelletje mijnenveger aan te maken dat dezelfde toestand heeft als de toestand van het huidige object. Deze stringvoorstelling heeft de vorm Mijnenveger(r, k, lijst), waarbij r en k staan voor het aantal rijen en kolommen van het speelveld. De lijst van tuples geeft aan waar er mijnen op het speelveld moeten geplaatst worden, opgelijst van links naar rechts en van boven naar onder.
- Een methode __str__ die een stringvoorstelling van het speelveld teruggeeft, waarbij elke rij van het rooster wordt voorgesteld als een afzonderlijke regel tekst. Posities van het speelveld waarop een mijn staat worden aangegeven met een sterretje (*) en posities waarop geen mijn staat worden aangegeven met een cijfer dat zegt op hoeveel naburige cellen er een mijn staat.

Methoden waaraan een postie op het speelveld moet doorgegeven worden (isMijn, mijnToevoegen, mijnVerwijderen en naburigeMijnen), moeten telkens nagaan of deze positie binnen de grenzen van het speelveld valt. Indien dit niet het geval is, moet hiervoor een AssertionError opgeworpen worden met een passende foutboodschap zoals aangegeven in onderstaand voorbeeld.

Voorbeeld

```
>>> speelveld = Mijnenveger(4, 5, [(0, 0), (1, 2)])
>>> print(speelveld)
*2110
12*10
01110
00000
>>> speelveld.rijen()
4
>>> speelveld.kolommen()
5
>>> speelveld.isMijn(0, 0)
True
>>> speelveld.naburigeMijnen(1, 0)
1
>>> speelveld.naburigeMijnen(1, 1)
2
>>> speelveld.mijnToevoegen(3, 3)
>>> print(speelveld)
*2110
12*10
01221
001*1
>>> speelveld.mijnVerwijderen(0, 0)
```

```
>>> print(speelveld.isMijn(0, 0))
False
>>> print(speelveld)
01110
01*10
01221
001*1
>>> speelveld
Mijnenveger(4, 5, [(1, 2), (3, 3)])

>>> speelveld.isMijn(4, 4)
Traceback (most recent call last):
AssertionError: ongeldige positie (4, 4)
>>> speelveld = Mijnenveger(1, 1)
Traceback (most recent call last):
AssertionError: speelveld moet minstens twee rijen hebben
>>> speelveld = Mijnenveger(2, 1)
Traceback (most recent call last):
AssertionError: speelveld moet minstens twee kolommen hebben
```