UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

FINAL TASK DOCUMENTATION

Juan Manuel Porrero Almansa
José Carlos Gualo Cejudo
Ernesto Plata Fernández

Subject:	Intelligent Systems
Work Group:	A1 - 4
GitHub repository:	https://github.com/juanma1501/Grupo-A1_4
Grade:	Computer Engineering
Date:	24th November 2020

Index of contents

1. Goal of the project

The main goal of the project is to implement a program using the search algorithms that we have seen in the theoretical part of the subject to draw mazes and find their cheaper solution. The main execution has as entry of our program a JSON file with information about the initial cell, the objective cell and the maze that we want to use. From this entry, the program will print by console the sequence of nodes that form the solution and will generate a *.txt* with this sequence.

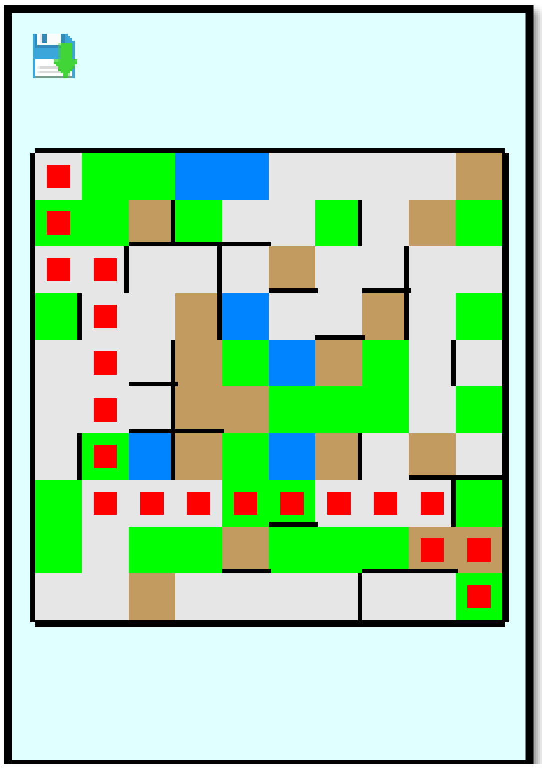
We began our project with a **first task** that had as objective build the necessary classes for the maze, draw it and export it to a JSON file. Once we had it, we implement the function of read mazes from other JSON files.

In the **second task**, we implemented the classes States Space, Problem, Frontier and Node (of the search tree).

Finally, in the **third task** the objective was to implement the basic search algorithm and the different search strategies (depth, breadth, uniform, greedy and A*). Each cell has different cost depending of the material formed by. This material can be **Grass** (in green), **Water** (in blue), **Asphalt** (in grey) and **Land** (in brown).

We have also try to control all the possible **exceptions** that can be generated by an error from the user using this program.

The program has different types of execution that will be explained later in this documentation.


1 Example of maze and solution generated by the program.

2. Tools used

2.1. Programming language

We have decided that for the realization of the Intelligent Systems practices we will use the Python 3 programming language. The main motivation for this decision is that we would like to learn how to program in this language, as we believe that it is quite versatile and can be used for the development of any kind of task. Therefore, we consider that it is highly recommended to learn how to program in Python and it may be useful to us in the future.

In addition, another reason that leads us to opt for Python as a programming language is that, as it is an open source language, we can find a large number of publications that will help us when developing our work.

2.2. Code repository

We have used a private code repository on **GitHub** that has allowed us to have version control of our practice, as well as offering the possibility of all members of the working group to work in parallel.

https://github.com/juanma1501/Grupo-A1_4

2.3. Modules and libraries

- pygame

While doing task number 1, we have used library Pygame for drawing the maze. Pygame is a set of modules of the Python language that allows the creation of two-dimensional videogames in a simple way. It is oriented to the handling of sprites. Thanks to the language, it can be prototyped and developed quickly. We have used it to draw lines (walls of the maze), colour the cells with the corresponding colours depending of their value and to print small squares to represent the solution. We have consulted the following documentation for this library:

https://www.pygame.org/docs/

The best way to install pygame is with the pip tool (which is what python uses to install packages). Note, this comes with python in recent versions. We use the --user flag to tell it to install into the home directory, rather than globally.

```
python3 -m pip install -U pygame --user
```

To see if it works, run one of the included examples:

```
python3 -m pygame.examples.aliens
```

In **Windows** the installation is done by:

```
py -m pip install -U pygame --user

py -m pygame.examples.aliens
```

- json

  In our program's JsonFile class, we use the json library to have access to the methods that we use to read JSON strings.

  https://docs.python.org/3.7/library/json.html

- random

  We use the random module in the class Cell, to assign a random value between 0 and 1 to the cells of the maze that we create. It is also used in the class Board to select a random cell from maze board and in Wilson class.

  https://docs.python.org/3/library/random.html

- sys

  We have used it to have access to sys.exit() in order to end the program when required. It is used in Draw class.

  https://docs.python.org/3/library/sys.html

- os

  It is used to hide a welcome message from the pygame library.

  https://docs.python.org/3/library/os.html

- argparse

  We have used this library that allows us to manage command line with efficiency and supplies help for the user in case of error. It is used in the main class Maze.py

  https://docs.python.org/3/library/argparse.html

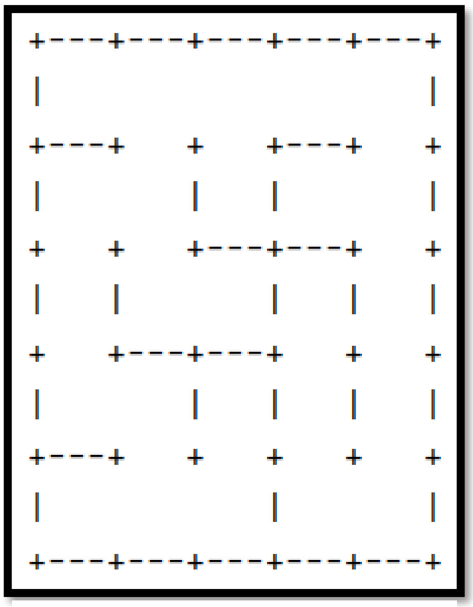
3. How to compile the program

Our program has four different methods of execution. To run the program we have to use the command console line.

3.1. Draw the maze in the console

It was not required, but we decided to include it as an extra functionality. To draw the maze in the console we have to write the next command:

```
python3 Maze.py console <rows> <cols>
```



```
+---+---+---+---+---+
|                   |
+---+   +   +---+   +
|       |   |       |
+   +   +---+---+   +
|   |       |   |   |
+   +---+---+   +   +
|       |   |   |   |
+---+   +   +   +   +
|           |       |
+---+---+---+---+---+
```

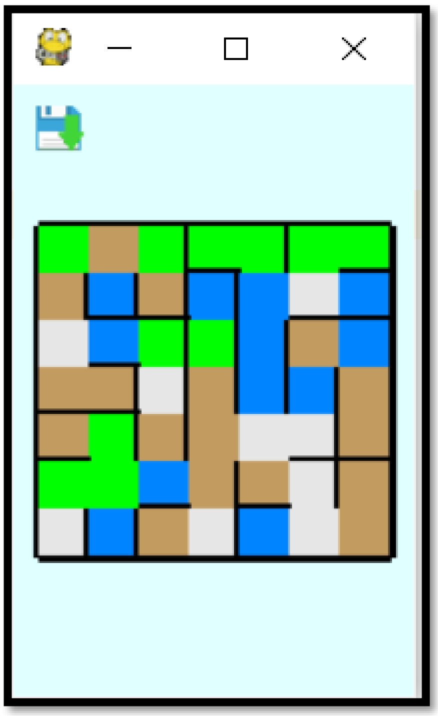
2 Output of the command console 5 5

3.2. Generate a maze and save its image and JSON file.

If we want to generate our maze from “zero” we have to introduce the following command:

```
python3 Maze.py image <rows> <cols>
```

Then, once the image is generated we can save the maze as a *.png* image and as a JSON file by clicking on the save icon (it is highly recommended to do several clicks).



Save icon

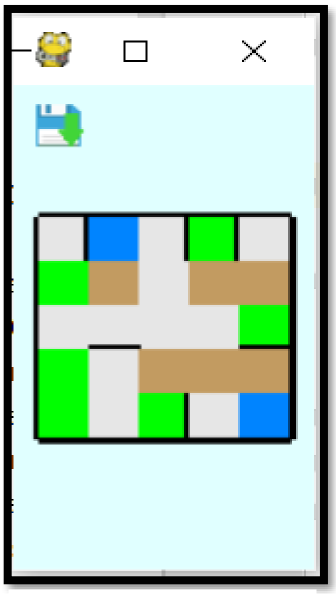
3 Example of execution of the command image 7 7

3.3. Draw a maze from a JSON file (without solution)

To draw a maze (without solution) from a JSON file we have to introduce the following command:

```
python3 Maze.py json <"path of the .json">
```

The difference now, is that is click on the save button, the program will only save the image, because this maze comes from a JSON file (in order to do not have duplicates).


4 Example of execution of the command json "problema_5x5_maze.json"

3.4. Generate a solution for a maze and export it into a *.txt* file

The main goal of the program, generate solution for a maze. The program will draw the solution path, will generate a *.txt* file with the solution and will print by console this path. We have to introduce the following command:

```
python3 Maze.py problem “path of the json problem” <strategy>
```

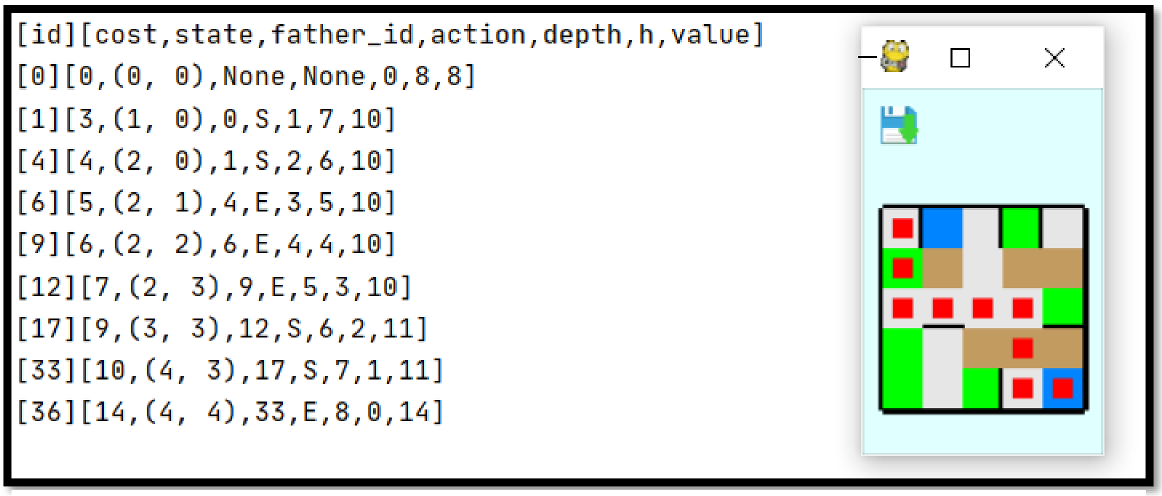
The **.json file** has the following structure:

```
{"INITIAL": "(0, 0)", "OBJECTIVE": "(4, 4)", "MAZE": "problema_5x5_maze.json"}
```

Strategy can be:

- A
- GREEDY
- UNIFORM
- DEPTH
- BREADTH

If we write *problem "problema_5x5.json" A*



```
[id][cost,state,father_id,action,depth,h,value]
[0][0,(0, 0),None,None,0,8,8]
[1][3,(1, 0),0,S,1,7,10]
[4][4,(2, 0),1,S,2,6,10]
[6][5,(2, 1),4,E,3,5,10]
[9][6,(2, 2),6,E,4,4,10]
[12][7,(2, 3),9,E,5,3,10]
[17][9,(3, 3),12,S,6,2,11]
[33][10,(4, 3),17,S,7,1,11]
[36][14,(4, 4),33,E,8,0,14]
```

5 Example of execution of the command problem

A *.txt* file called `solution_5x5_A.txt` is generated in the execution directory.

If we click on the save button, we can also save the image as a *png* file.

3.5. Exceptions controlled

- The user introduces a not consistent JSON file.
- The user introduces a JSON file that not correspond to a maze.
- The user misspelled the tags of the problem JSON.
- The user introduces an initial state that is not in the space states.
- The user introduces an objective state that is not in the space states.
- The user introduces wrong commands when executing the program (handle by *argparse* library).
- The user introduces wrong parameters when executing the program (also handled by *argparse* library).

4. Implementation

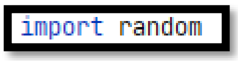
In this section, we are going to explain how the program has been implemented from the point of view of the three deliveries.

4.1. Task 1

The objectives for task 1 were to build a program to draw mazes, and then these mazes can be export as a JSON file and the image had to be saved. The last week until delivery 1, we had to implement a function for reading mazes from other JSON files and draw them. The classes to achieve these objectives are the following ones:

4.1.1. Cell class

Class to represent the cell of a maze. A maze is formed by *n x m* cells. We had to import the random library in order to assign a random value when creating a new maze.



```
import random
```

- Constructor and getters

To create a cell we only need to write its row and column.

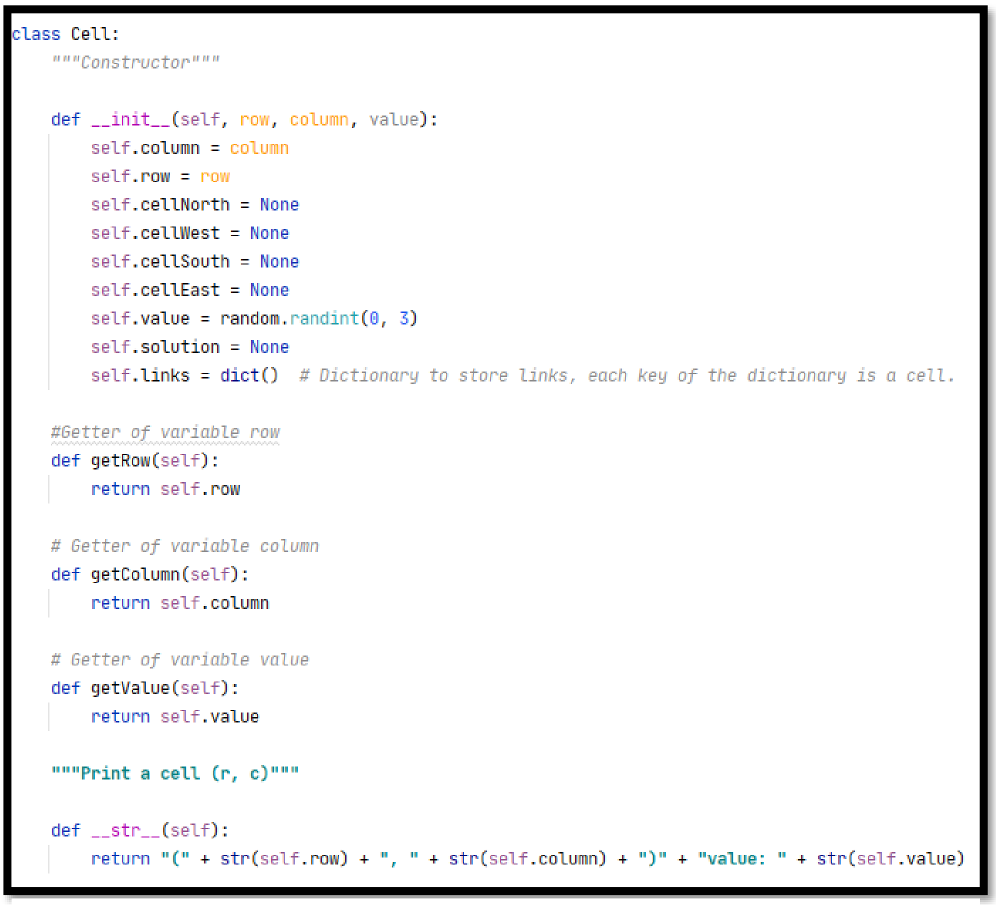
A cell has four **neighbours** as maximum, so the attributes cellNorth, cellWest, cellEast and cellSouth represent these neighbours and can be None or another cell.

Attribute **value** represents cell's material: 0 (asphalt), 1 (land), 2(grass) and 3 (water).

Attribute **solution** can be True or False depending on if this cell is part of the solution path.

Attribute **links** is a dictionary with the links of a cell, if a cell is not linked with another, there will be a wall between them.

The method **__str__** is used when we want to print an object of the class Cell with the format *(0, 0) value: 0*

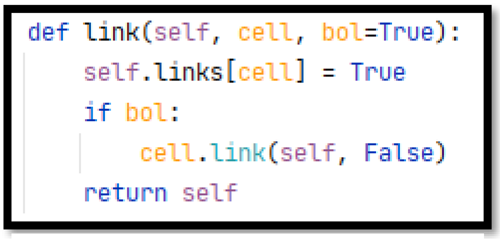


```python
class Cell:
    """Constructor"""

    def __init__(self, row, column, value):
        self.column = column
        self.row = row
        self.cellNorth = None
        self.cellWest = None
        self.cellSouth = None
        self.cellEast = None
        self.value = random.randint(0, 3)
        self.solution = None
        self.links = dict()  # Dictionary to store links, each key of the dictionary is a cell.

    #Getter of variable row
    def getRow(self):
        return self.row

    # Getter of variable column
    def getColumn(self):
        return self.column

    # Getter of variable value
    def getValue(self):
        return self.value

    """Print a cell (r, c)"""

    def __str__(self):
        return "(" + str(self.row) + ", " + str(self.column) + ")" + "value: " + str(self.value)
```

6 Cell class constructor and getters

- ## link method

This method link two cells, is very useful when reading the maze from JSON file, in order to link the cell with its neighbours.

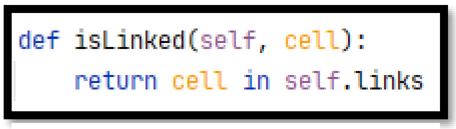


```python
def link(self, cell, bol=True):
    self.links[cell] = True
    if bol:
        cell.link(self, False)
    return self
```

7 link method

- ## isLinked method

To check if a cell is linked with another one.

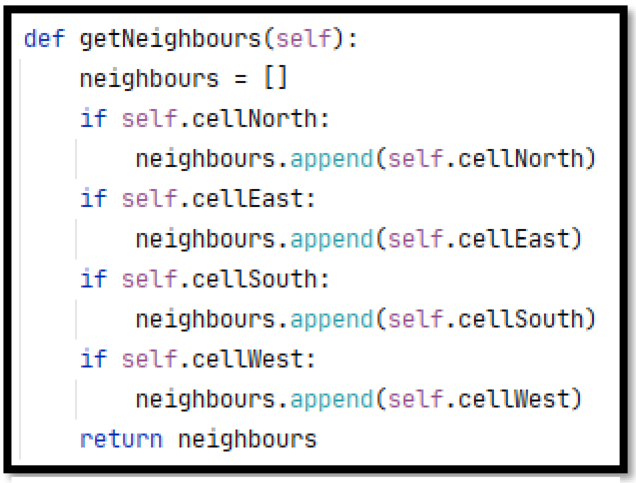


```python
def isLinked(self, cell):
    return cell in self.links
```

8 isLinked method

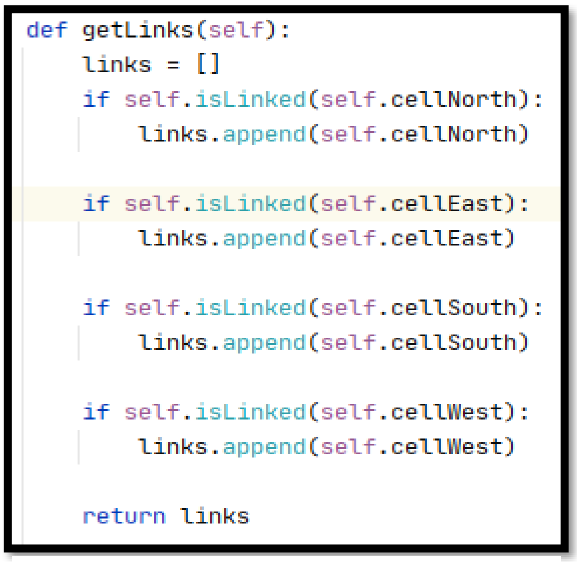
- **getNeighbours and getLinks methods**

These methods are to return the neighbours and the links of a cell. They are different concepts, neighbours are all the cells around a cell (N, S, O, E) and links are only cells that not have a wall between them.



```python
def getNeighbours(self):
    neighbours = []
    if self.cellNorth:
        neighbours.append(self.cellNorth)
    if self.cellEast:
        neighbours.append(self.cellEast)
    if self.cellSouth:
        neighbours.append(self.cellSouth)
    if self.cellWest:
        neighbours.append(self.cellWest)
    return neighbours
```

10 getNeighbours method



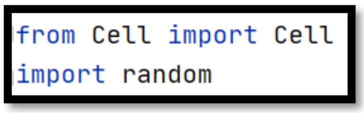
```python
def getLinks(self):
    links = []
    if self.isLinked(self.cellNorth):
        links.append(self.cellNorth)

    if self.isLinked(self.cellEast):
        links.append(self.cellEast)

    if self.isLinked(self.cellSouth):
        links.append(self.cellSouth)

    if self.isLinked(self.cellWest):
        links.append(self.cellWest)

    return links
```

9 getLinks method

4.1.2. Class Board

This class is used to represent a board of N rows and M columns, formed by cells (NxM cells), where the maze is going to be created following Wilson's algorithm.

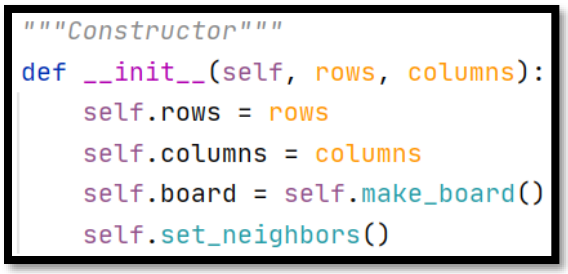
We have to import Cell and library random.



```python
from Cell import Cell
import random
```

- **Constructor**

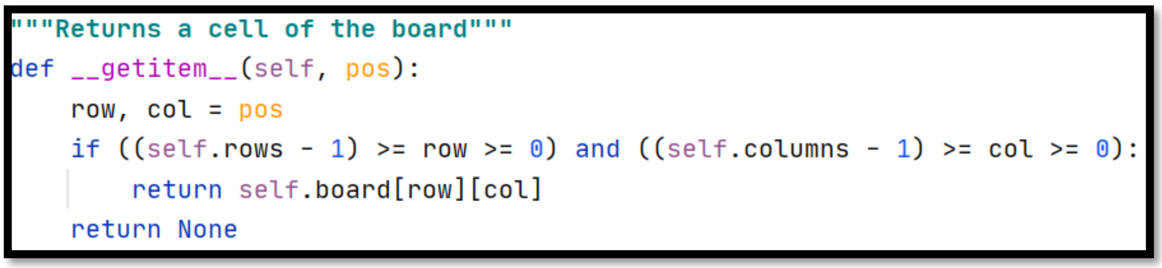
To construct a Maze object, we only have to write the nº of rows and columns that the board is going to have.



```python
"""Constructor"""
def __init__(self, rows, columns):
    self.rows = rows
    self.columns = columns
    self.board = self.make_board()
    self.set_neighbors()
```

- __getitem__ method

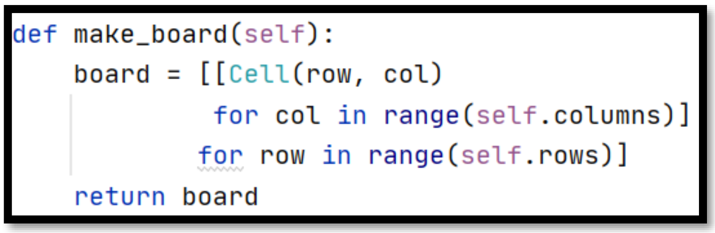
This method returns a cell of the board. An example will be: board[2,3]



```python
"""Returns a cell of the board"""
def __getitem__(self, pos):
    row, col = pos
    if ((self.rows - 1) >= row >= 0) and ((self.columns - 1) >= col >= 0):
        return self.board[row][col]
    return None
```

- make_board method

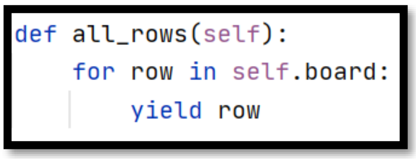
This method is to initialize the cells of the maze.



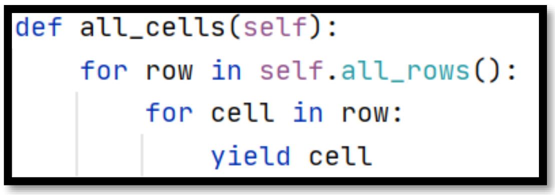
```python
def make_board(self):
    board = [[Cell(row, col)
              for col in range(self.columns)]
             for row in range(self.rows)]
    return board
```

- all_rows method and all_cells method

The functionality of these methods are very similar, they use yield instead of return to make it easier to use them in a for, in order to iterate all over the rows or the cells of the maze.



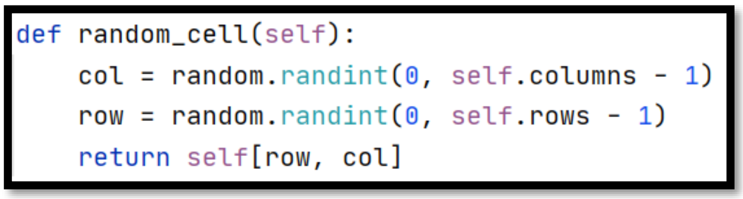
```python
def all_rows(self):
    for row in self.board:
        yield row
```



```python
def all_cells(self):
    for row in self.all_rows():
        for cell in row:
            yield cell
```

- random_cell method

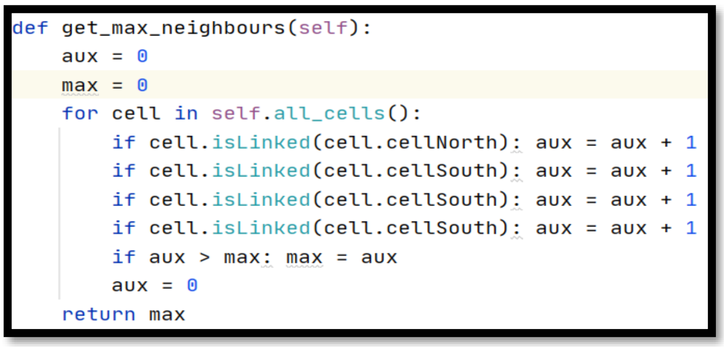
Method used to get a random cell from the board.



```python
def random_cell(self):
    col = random.randint(0, self.columns - 1)
    row = random.randint(0, self.rows - 1)
    return self[row, col]
```

- ## get_max_neighbours method

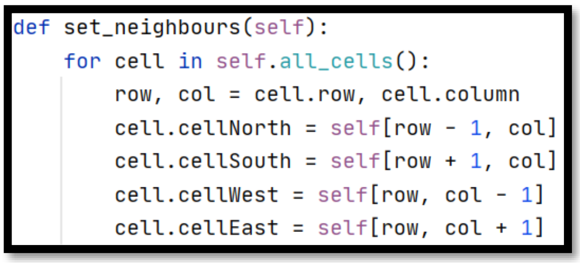
This method returns the maximum number of neighbours that a cell has.



```
def get_max_neighbours(self):
    aux = 0
    max = 0
    for cell in self.all_cells():
        if cell.isLinked(cell.cellNorth): aux = aux + 1
        if cell.isLinked(cell.cellSouth): aux = aux + 1
        if cell.isLinked(cell.cellSouth): aux = aux + 1
        if cell.isLinked(cell.cellSouth): aux = aux + 1
        if aux > max: max = aux
        aux = 0
    return max
```

- ## set_neighbours method

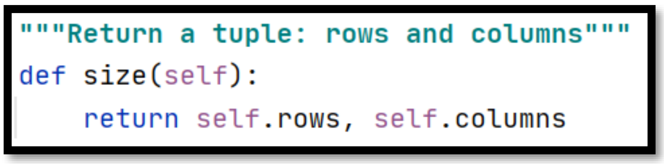
Method to set the neighbours cells of all the cells in the maze



```
def set_neighbours(self):
    for cell in self.all_cells():
        row, col = cell.row, cell.column
        cell.cellNorth = self[row - 1, col]
        cell.cellSouth = self[row + 1, col]
        cell.cellWest = self[row, col - 1]
        cell.cellEast = self[row, col + 1]
```

- ## size method

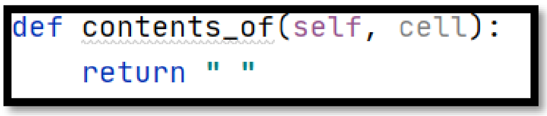
Returns the size of the board (NxM)



```
"""Return a tuple: rows and columns"""
def size(self):
    return self.rows, self.columns
```

- ## contents_of method

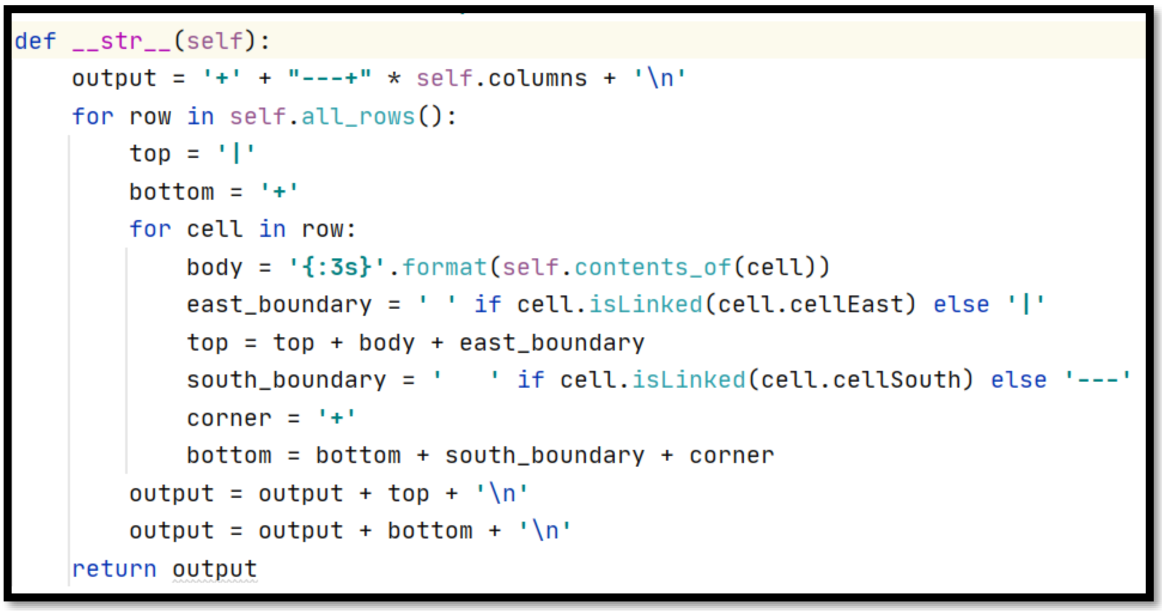
Return the ‘content’ of a cell



```
def contents_of(self, cell):
    return " "
```

- __str__ method

This method is used to print the maze by console, as we said, this functionality is not compulsory, but we found it very curious and maybe we can use it later to another job (who knows).

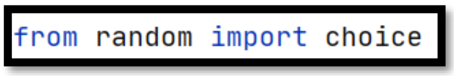


```python
def __str__(self):
    output = '+' + "---+" * self.columns + '\n'
    for row in self.all_rows():
        top = '|'
        bottom = '+'
        for cell in row:
            body = '{:3s}'.format(self.contents_of(cell))
            east_boundary = ' ' if cell.isLinked(cell.cellEast) else '|'
            top = top + body + east_boundary
            south_boundary = '   ' if cell.isLinked(cell.cellSouth) else '---'
            corner = '+'
            bottom = bottom + south_boundary + corner
        output = output + top + '\n'
        output = output + bottom + '\n'
    return output
```

4.1.3. Wilson Class

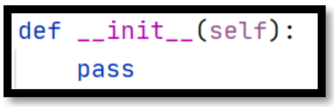
This class is to store the Wilson's algorithm, used to create a maze.

The only import of this class is:



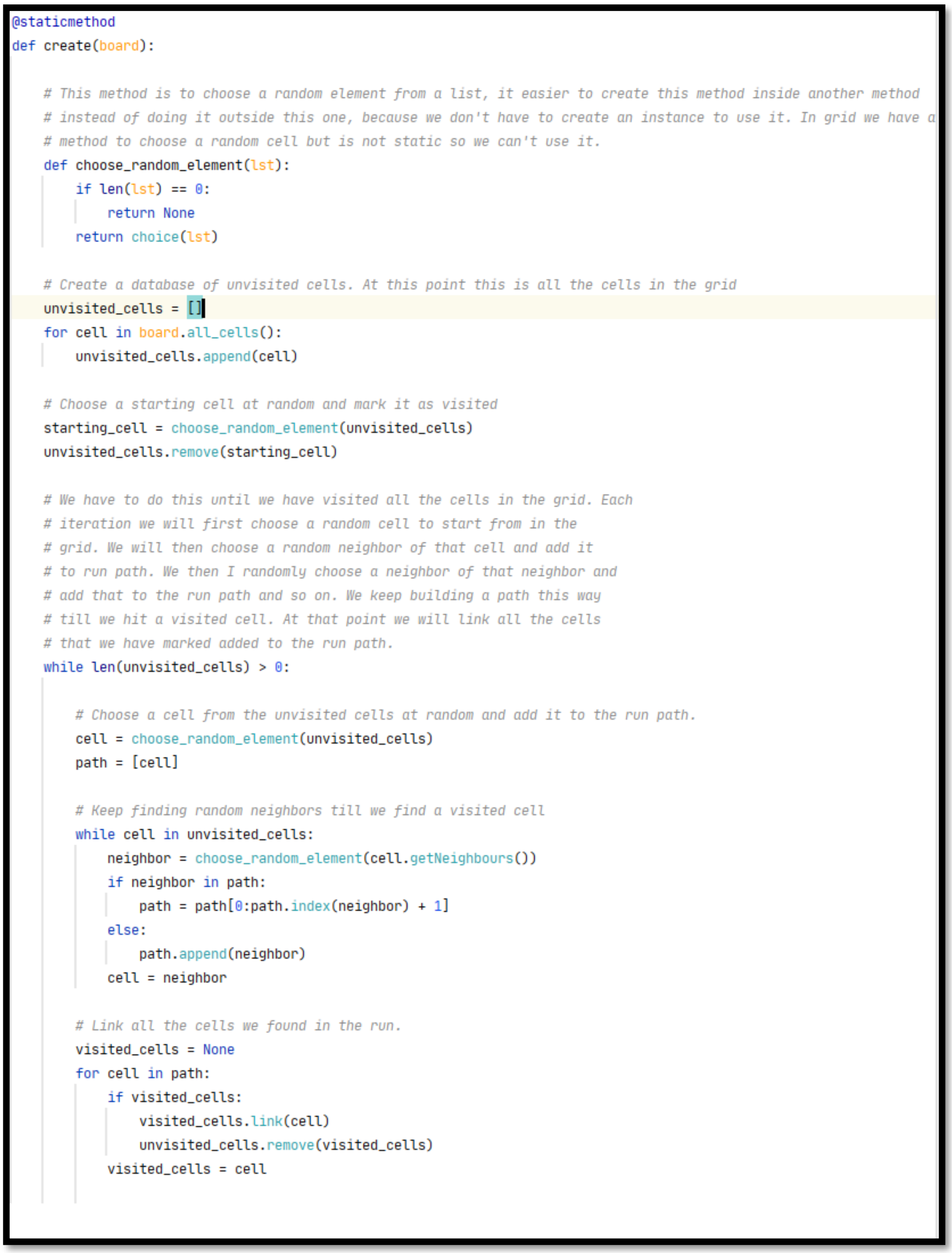
```python
from random import choice
```

- Constructor



```python
def __init__(self):
    pass
```

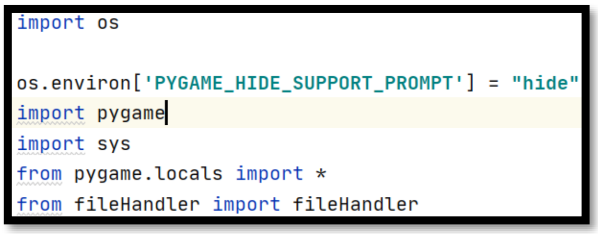
The algorithm performs a random walk on the grid to create passages. Since every iteration we are performing a random walk from a random starting cell, and also erasing any loops in the process.



```
@staticmethod
def create(board):

    # This method is to choose a random element from a list, it easier to create this method inside another method
    # instead of doing it outside this one, because we don't have to create an instance to use it. In grid we have a
    # method to choose a random cell but is not static so we can't use it.
    def choose_random_element(lst):
        if len(lst) == 0:
            return None
        return choice(lst)

    # Create a database of unvisited cells. At this point this is all the cells in the grid
    unvisited_cells = []
    for cell in board.all_cells():
        unvisited_cells.append(cell)

    # Choose a starting cell at random and mark it as visited
    starting_cell = choose_random_element(unvisited_cells)
    unvisited_cells.remove(starting_cell)

    # We have to do this until we have visited all the cells in the grid. Each
    # iteration we will first choose a random cell to start from in the
    # grid. We will then choose a random neighbor of that cell and add it
    # to run path. We then I randomly choose a neighbor of that neighbor and
    # add that to the run path and so on. We keep building a path this way
    # till we hit a visited cell. At that point we will link all the cells
    # that we have marked added to the run path.
    while len(unvisited_cells) > 0:

        # Choose a cell from the unvisited cells at random and add it to the run path.
        cell = choose_random_element(unvisited_cells)
        path = [cell]

        # Keep finding random neighbors till we find a visited cell
        while cell in unvisited_cells:
            neighbor = choose_random_element(cell.getNeighbours())
            if neighbor in path:
                path = path[0:path.index(neighbor) + 1]
            else:
                path.append(neighbor)
            cell = neighbor

        # Link all the cells we found in the run.
        visited_cells = None
        for cell in path:
            if visited_cells:
                visited_cells.link(cell)
                unvisited_cells.remove(visited_cells)
            visited_cells = cell
```

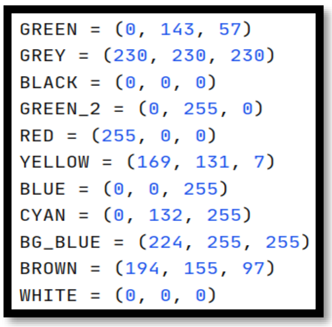
4.1.4. Draw class

This class is used to generate an image of the maze and to save it in the execution directory. The imports are the following ones:



```python
import os

os.environ['PYGAME_HIDE_SUPPORT_PROMPT'] = "hide"
import pygame
import sys
from pygame.locals import *
from fileHandler import fileHandler
```

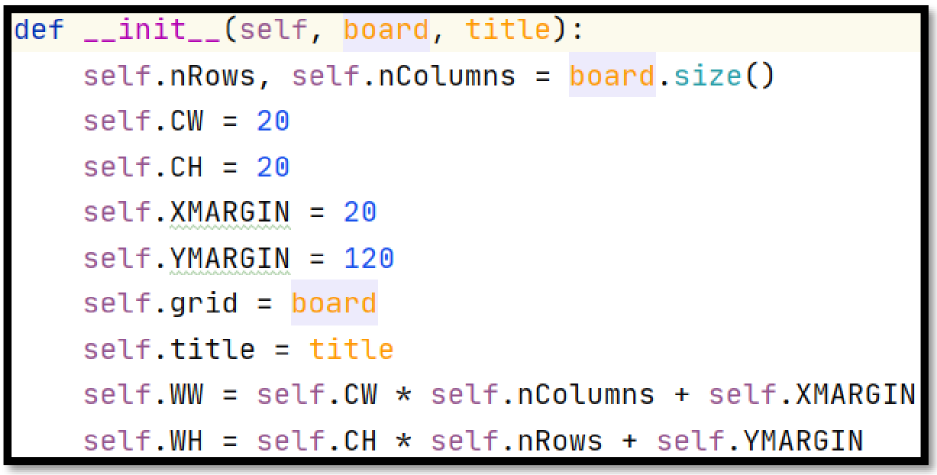
We also define some colors in RGB format that can be used while displaying maze image:



```python
GREEN = (0, 143, 57)
GREY = (230, 230, 230)
BLACK = (0, 0, 0)
GREEN_2 = (0, 255, 0)
RED = (255, 0, 0)
YELLOW = (169, 131, 7)
BLUE = (0, 0, 255)
CYAN = (0, 132, 255)
BG_BLUE = (224, 255, 255)
BROWN = (194, 155, 97)
WHITE = (0, 0, 0)
```

- Constructor

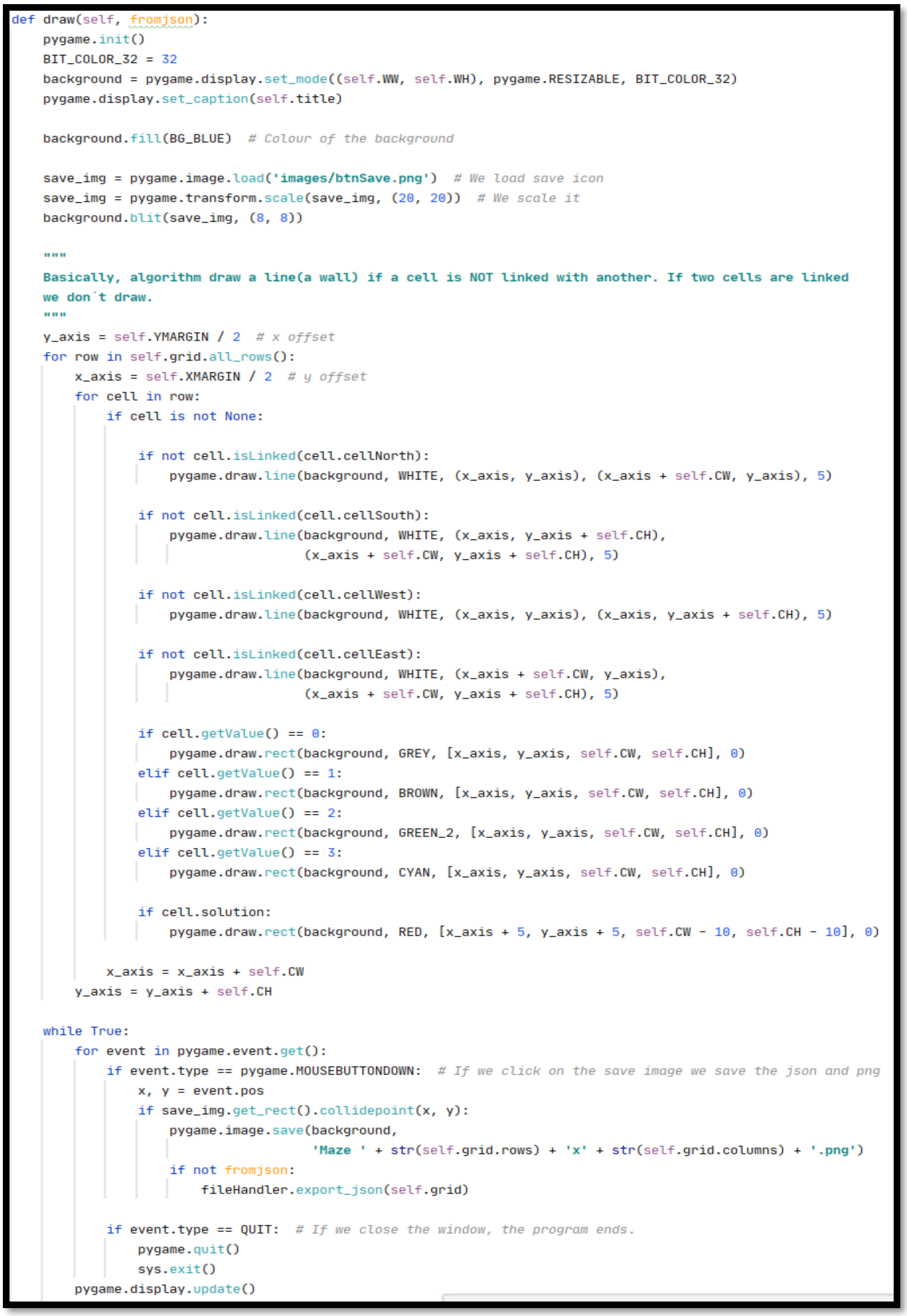
We receive as parameter a board object and a string representing the tittle that the window (and the maze) is going to save with. Then we define properties such us the margins and the height and weight.



```python
def __init__(self, board, title):
    self.nRows, self.nColumns = board.size()
    self.CW = 20
    self.CH = 20
    self.XMARGIN = 20
    self.YMARGIN = 120
    self.grid = board
    self.title = title
    self.WW = self.CW * self.nColumns + self.XMARGIN
    self.WH = self.CH * self.nRows + self.YMARGIN
```

- ## Draw method

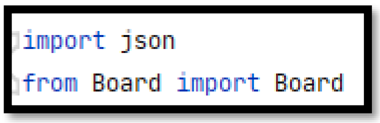
Method used to draw the image of the maze. Parameter fromjson indicates if the maze comes from a json file or if the maze has been generated from 0 by the user. True in the first case and false in the second one.



```python
def draw(self, fromjson):
    pygame.init()
    BIT_COLOR_32 = 32
    background = pygame.display.set_mode((self.WW, self.WH), pygame.RESIZABLE, BIT_COLOR_32)
    pygame.display.set_caption(self.title)

    background.fill(BG_BLUE)  # Colour of the background

    save_img = pygame.image.load('images/btnSave.png')  # We load save icon
    save_img = pygame.transform.scale(save_img, (20, 20))  # We scale it
    background.blit(save_img, (8, 8))

    """
    Basically, algorithm draw a line(a wall) if a cell is NOT linked with another. If two cells are linked
    we don´t draw.
    """
    y_axis = self.YMARGIN / 2  # x offset
    for row in self.grid.all_rows():
        x_axis = self.XMARGIN / 2  # y offset
        for cell in row:
            if cell is not None:

                if not cell.isLinked(cell.cellNorth):
                    pygame.draw.line(background, WHITE, (x_axis, y_axis), (x_axis + self.CW, y_axis), 5)

                if not cell.isLinked(cell.cellSouth):
                    pygame.draw.line(background, WHITE, (x_axis, y_axis + self.CH),
                                     (x_axis + self.CW, y_axis + self.CH), 5)

                if not cell.isLinked(cell.cellWest):
                    pygame.draw.line(background, WHITE, (x_axis, y_axis), (x_axis, y_axis + self.CH), 5)

                if not cell.isLinked(cell.cellEast):
                    pygame.draw.line(background, WHITE, (x_axis + self.CW, y_axis),
                                     (x_axis + self.CW, y_axis + self.CH), 5)

                if cell.getValue() == 0:
                    pygame.draw.rect(background, GREY, [x_axis, y_axis, self.CW, self.CH], 0)
                elif cell.getValue() == 1:
                    pygame.draw.rect(background, BROWN, [x_axis, y_axis, self.CW, self.CH], 0)
                elif cell.getValue() == 2:
                    pygame.draw.rect(background, GREEN_2, [x_axis, y_axis, self.CW, self.CH], 0)
                elif cell.getValue() == 3:
                    pygame.draw.rect(background, CYAN, [x_axis, y_axis, self.CW, self.CH], 0)

                if cell.solution:
                    pygame.draw.rect(background, RED, [x_axis + 5, y_axis + 5, self.CW - 10, self.CH - 10], 0)

            x_axis = x_axis + self.CW
        y_axis = y_axis + self.CH

    while True:
        for event in pygame.event.get():
            if event.type == pygame.MOUSEBUTTONDOWN:  # If we click on the save image we save the json and png
                x, y = event.pos
                if save_img.get_rect().collidepoint(x, y):
                    pygame.image.save(background,
                                      'Maze ' + str(self.grid.rows) + 'x' + str(self.grid.columns) + '.png')
                    if not fromjson:
                        fileHandler.export_json(self.grid)

            if event.type == QUIT:  # If we close the window, the program ends.
                pygame.quit()
                sys.exit()
        pygame.display.update()
```

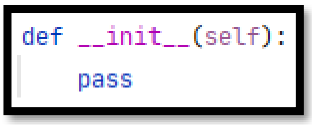
4.1.5. fileHandler Class

This class is used to handle all the methods related to txt files and json files. We import the followings:



```python
import json
from Board import Board
```

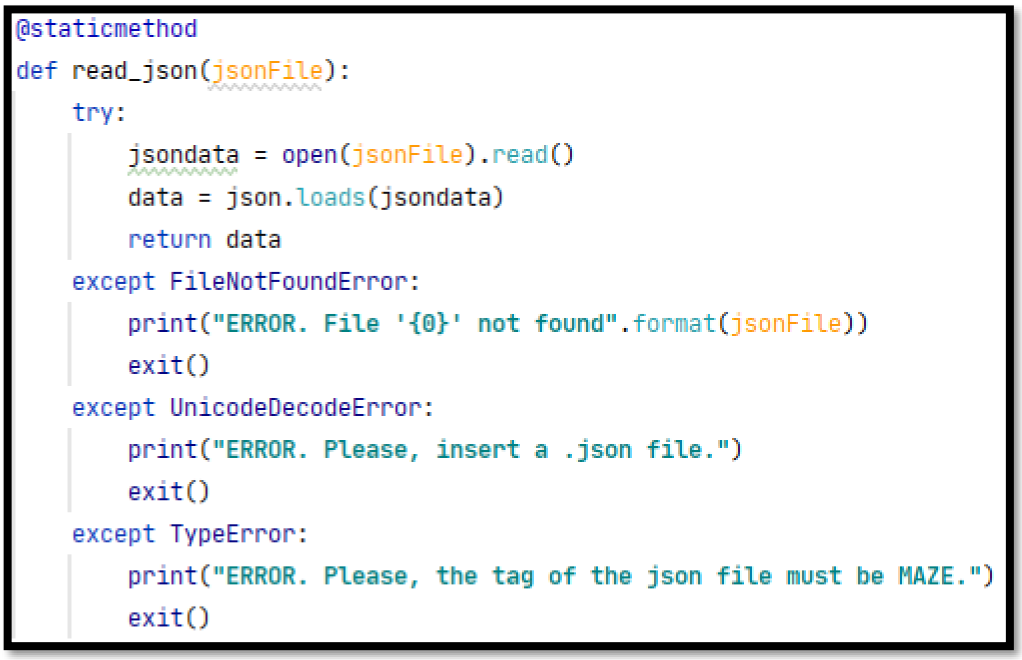
- Constructor



```python
def __init__(self):
    pass
```

- read_json method

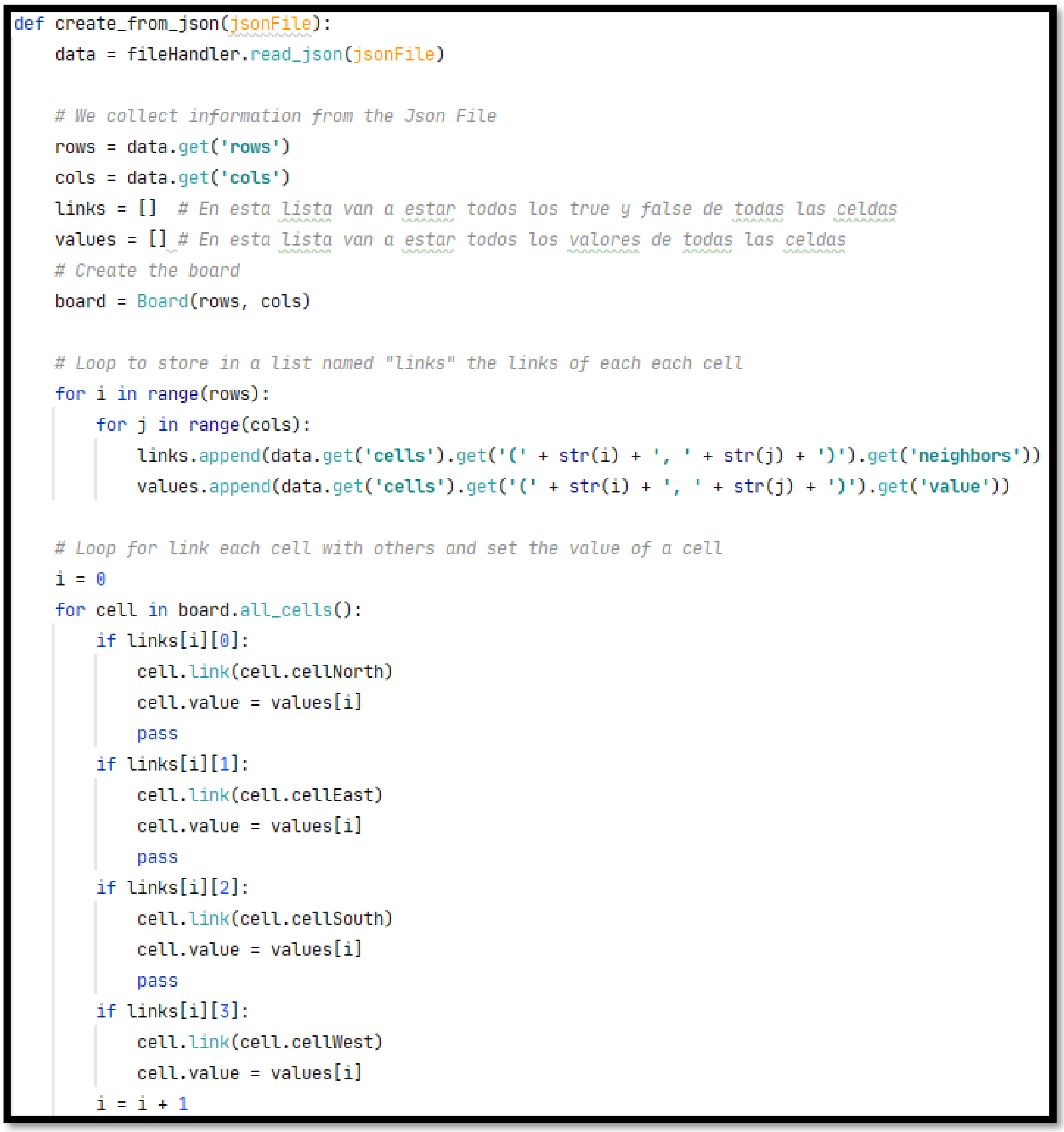
This method receive as parameter the path of the json file and returns “data” a dictionary with the information of the json file.



```python
@staticmethod
def read_json(jsonFile):
    try:
        jsondata = open(jsonFile).read()
        data = json.loads(jsondata)
        return data
    except FileNotFoundError:
        print("ERROR. File '{0}' not found".format(jsonFile))
        exit()
    except UnicodeDecodeError:
        print("ERROR. Please, insert a .json file.")
        exit()
    except TypeError:
        print("ERROR. Please, the tag of the json file must be MAZE.")
        exit()
```

- create_from_json method

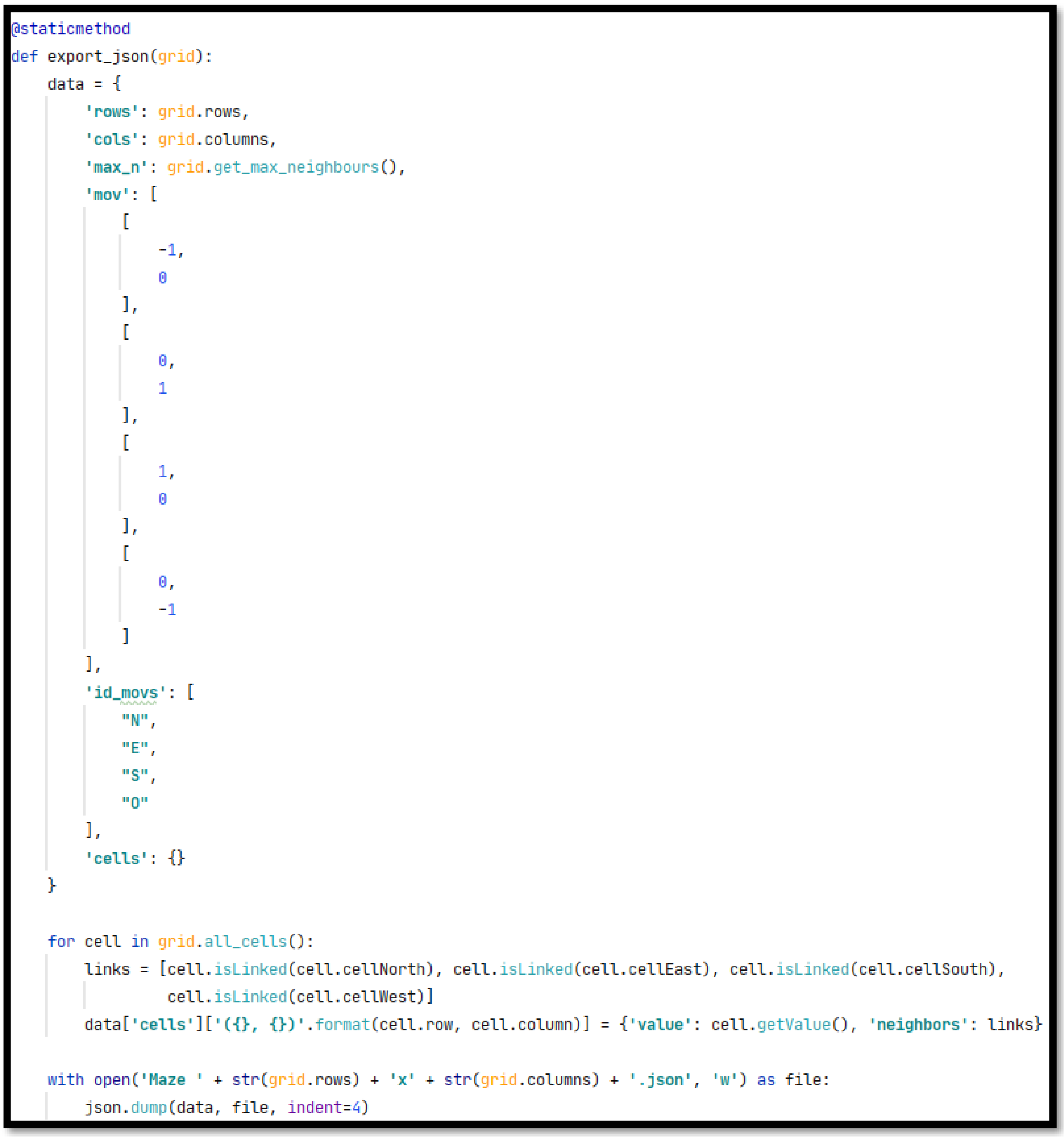
We receive the path of the json file and we call the method read_json. Then we read all the information from the dictionary and we set the links (walls) and the values of the cell of the board and finally we return this board object.



```python
def create_from_json(jsonFile):
    data = fileHandler.read_json(jsonFile)

    # We collect information from the Json File
    rows = data.get('rows')
    cols = data.get('cols')
    links = []  # En esta lista van a estar todos los true y false de todas las celdas
    values = [] # En esta lista van a estar todos los valores de todas las celdas
    # Create the board
    board = Board(rows, cols)

    # Loop to store in a list named "links" the links of each each cell
    for i in range(rows):
        for j in range(cols):
            links.append(data.get('cells').get('(' + str(i) + ', ' + str(j) + ')').get('neighbors'))
            values.append(data.get('cells').get('(' + str(i) + ', ' + str(j) + ')').get('value'))

    # Loop for link each cell with others and set the value of a cell
    i = 0
    for cell in board.all_cells():
        if links[i][0]:
            cell.link(cell.cellNorth)
            cell.value = values[i]
            pass
        if links[i][1]:
            cell.link(cell.cellEast)
            cell.value = values[i]
            pass
        if links[i][2]:
            cell.link(cell.cellSouth)
            cell.value = values[i]
            pass
        if links[i][3]:
            cell.link(cell.cellWest)
            cell.value = values[i]
        i = i + 1
```

- export_json method

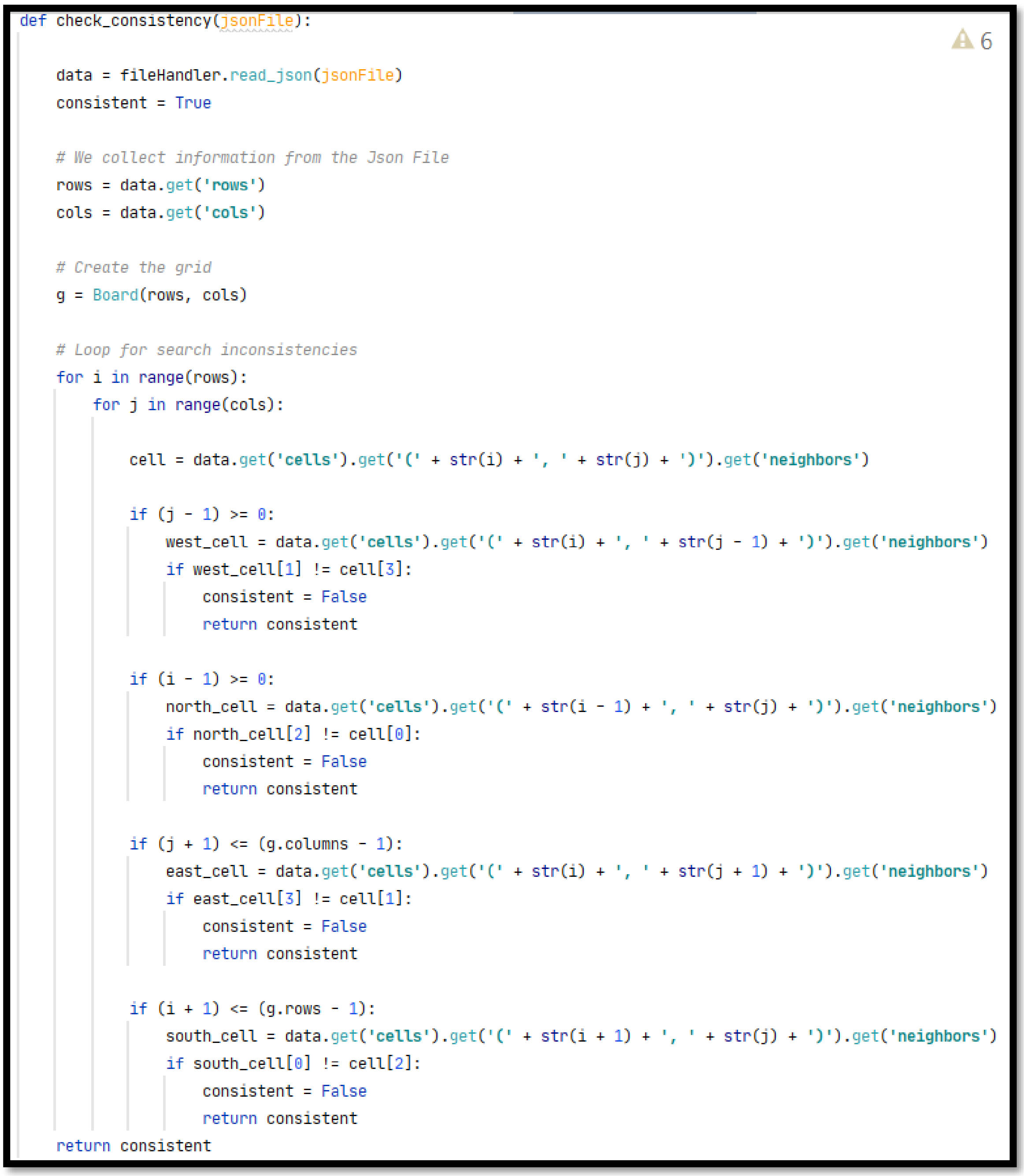
This method export the json file of a maze passed through parameter of the method making use of the methods of the library imported json.



```python
@staticmethod
def export_json(grid):
    data = {
        'rows': grid.rows,
        'cols': grid.columns,
        'max_n': grid.get_max_neighbours(),
        'mov': [
            [
                -1,
                0
            ],
            [
                0,
                1
            ],
            [
                1,
                0
            ],
            [
                0,
                -1
            ]
        ],
        'id_movs': [
            "N",
            "E",
            "S",
            "O"
        ],
        'cells': {}
    }

    for cell in grid.all_cells():
        links = [cell.isLinked(cell.cellNorth), cell.isLinked(cell.cellEast), cell.isLinked(cell.cellSouth),
                 cell.isLinked(cell.cellWest)]
        data['cells']['({}, {})'.format(cell.row, cell.column)] = {'value': cell.getValue(), 'neighbors': links}

    with open('Maze ' + str(grid.rows) + 'x' + str(grid.columns) + '.json', 'w') as file:
        json.dump(data, file, indent=4)
```

- ## check_consistency method

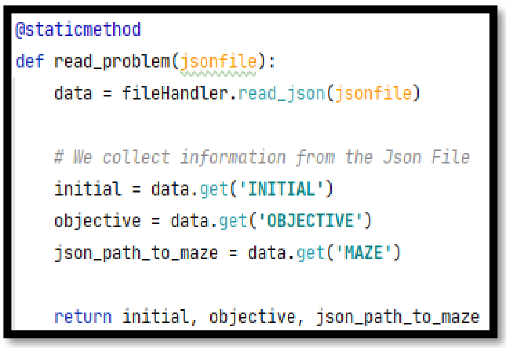
We use this method to check the consistency of a json file. This method works in the same way than the previous one, first, we read the json calling read_json method and the we iterate all over the cells of the maze checking for errors. We return True if the json is consistent and False if the json is not consistent.



```
def check_consistency(jsonFile):
                                                                        ⚠6
    data = fileHandler.read_json(jsonFile)
    consistent = True

    # We collect information from the Json File
    rows = data.get('rows')
    cols = data.get('cols')

    # Create the grid
    g = Board(rows, cols)

    # Loop for search inconsistencies
    for i in range(rows):
        for j in range(cols):

            cell = data.get('cells').get('(' + str(i) + ', ' + str(j) + ')').get('neighbors')

            if (j - 1) >= 0:
                west_cell = data.get('cells').get('(' + str(i) + ', ' + str(j - 1) + ')').get('neighbors')
                if west_cell[1] != cell[3]:
                    consistent = False
                    return consistent

            if (i - 1) >= 0:
                north_cell = data.get('cells').get('(' + str(i - 1) + ', ' + str(j) + ')').get('neighbors')
                if north_cell[2] != cell[0]:
                    consistent = False
                    return consistent

            if (j + 1) <= (g.columns - 1):
                east_cell = data.get('cells').get('(' + str(i) + ', ' + str(j + 1) + ')').get('neighbors')
                if east_cell[3] != cell[1]:
                    consistent = False
                    return consistent

            if (i + 1) <= (g.rows - 1):
                south_cell = data.get('cells').get('(' + str(i + 1) + ', ' + str(j) + ')').get('neighbors')
                if south_cell[0] != cell[2]:
                    consistent = False
                    return consistent
    return consistent
```

- read_problem method

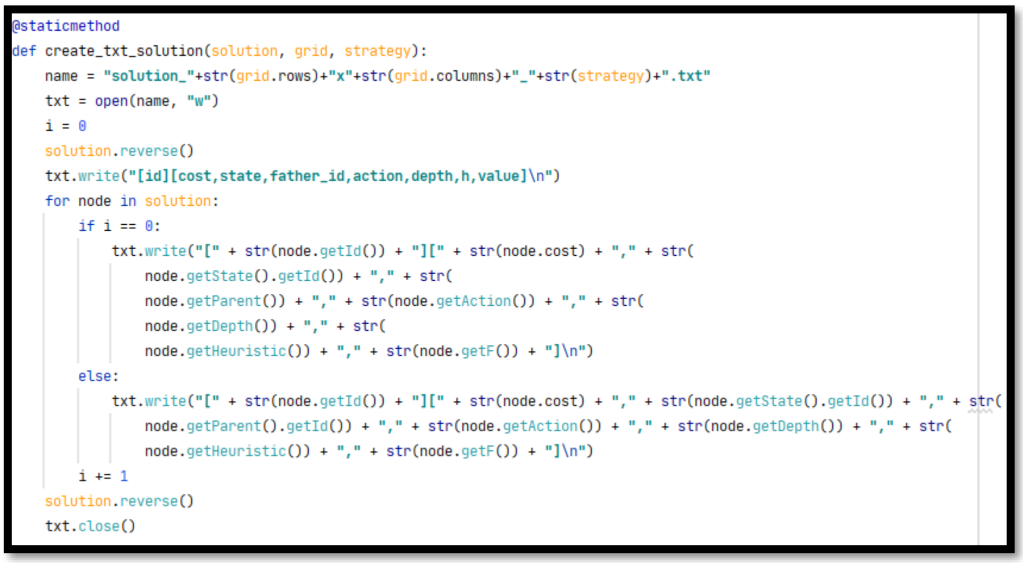
This method was added in task number 3, but we commented it here in order to make it easier to the user to know what is in each class. It read the problem json file and returns the initial node, the objective and the path of the json file that contains the maze.



```
@staticmethod
def read_problem(jsonfile):
    data = fileHandler.read_json(jsonfile)

    # We collect information from the Json File
    initial = data.get('INITIAL')
    objective = data.get('OBJECTIVE')
    json_path_to_maze = data.get('MAZE')

    return initial, objective, json_path_to_maze
```

- create_txt_solution method

This method was also added in task 3. It is used to create the txt file that contains the path solution and all its info about the nodes. The method receives as parameter the solution array, the grid and the strategy followed.



```python
@staticmethod
def create_txt_solution(solution, grid, strategy):
    name = "solution_"+str(grid.rows)+"x"+str(grid.columns)+"_"+str(strategy)+".txt"
    txt = open(name, "w")
    i = 0
    solution.reverse()
    txt.write("[id][cost,state,father_id,action,depth,h,value]\n")
    for node in solution:
        if i == 0:
            txt.write("[" + str(node.getId()) + "][" + str(node.cost) + "," + str(
                node.getState().getId()) + "," + str(
                node.getParent()) + "," + str(node.getAction()) + "," + str(
                node.getDepth()) + "," + str(
                node.getHeuristic()) + "," + str(node.getF()) + "]\n")
        else:
            txt.write("[" + str(node.getId()) + "][" + str(node.cost) + "," + str(node.getState().getId()) + "," + str(
                node.getParent().getId()) + "," + str(node.getAction()) + "," + str(node.getDepth()) + "," + str(
                node.getHeuristic()) + "," + str(node.getF()) + "]\n")
        i += 1
    solution.reverse()
    txt.close()
```

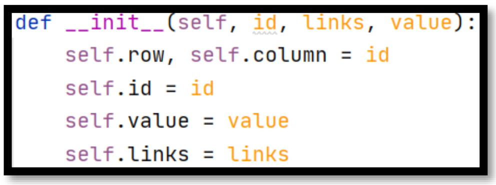
4.2. Task 2

In this task the goal was to define and implement a software artifact problem-exit maze and all the structures needed to create and manipulate the tree search. To achieve this objective we have created the following classes.

4.2.1. State Class

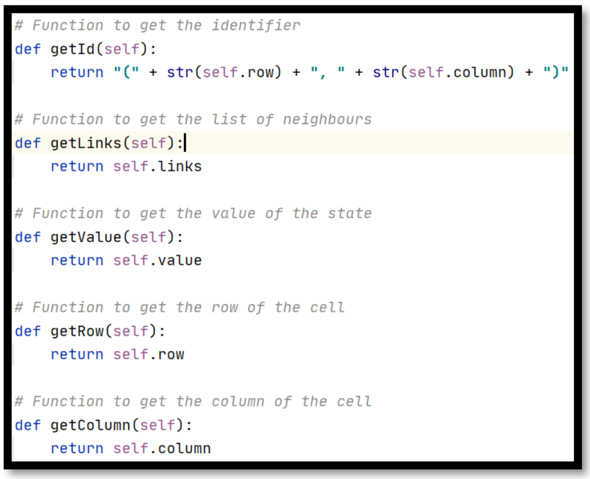
This class represents a state. A state is represented by the id (2, 3) for example, but also has links with the neighbours cells and a value (0, 1, 2 or 3). There are no imports in this class.

- Constructor



```python
def __init__(self, id, links, value):
    self.row, self.column = id
    self.id = id
    self.value = value
    self.links = links
```

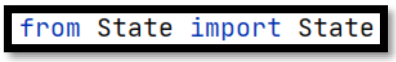
- Getters



```python
# Function to get the identifier
def getId(self):
    return "(" + str(self.row) + ", " + str(self.column) + ")"

# Function to get the list of neighbours
def getLinks(self):
    return self.links

# Function to get the value of the state
def getValue(self):
    return self.value

# Function to get the row of the cell
def getRow(self):
    return self.row

# Function to get the column of the cell
def getColumn(self):
    return self.column
```

4.2.2. StatesSpace class

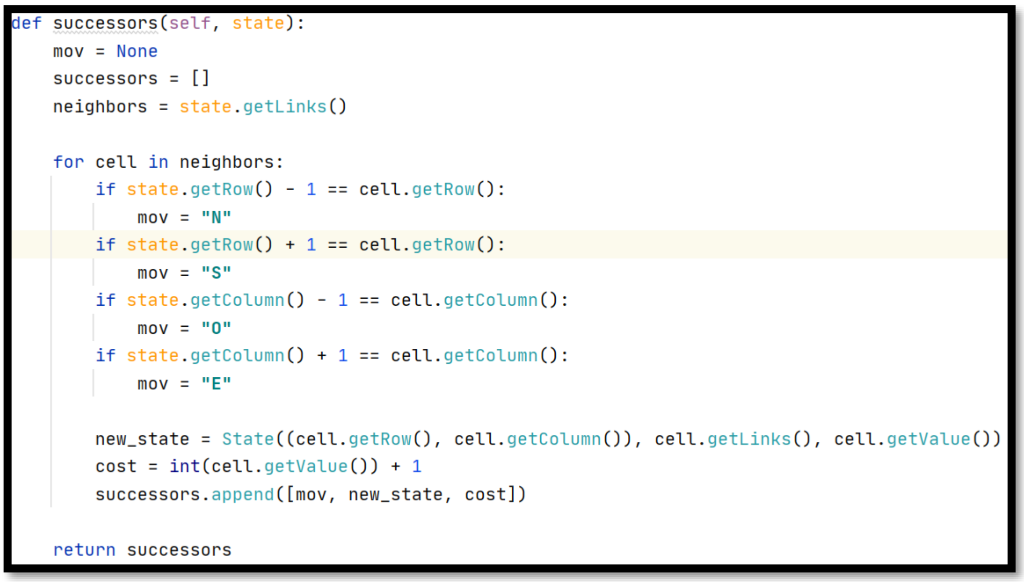
Class to store the successor function and heuristic calculation. The import is:



```python
from State import State
```

- Successor function

Function to set the successor states of the given one. We create a list of successor and the we receive the links of the cell (because if there is a wall between two cells this is not a possible successor) and then we iterate through the cells that are linked, evaluate the movement, construct the new state and set the cost. Finally we append this successor to the successors list. We have to pass the state as parameter to find its successors.

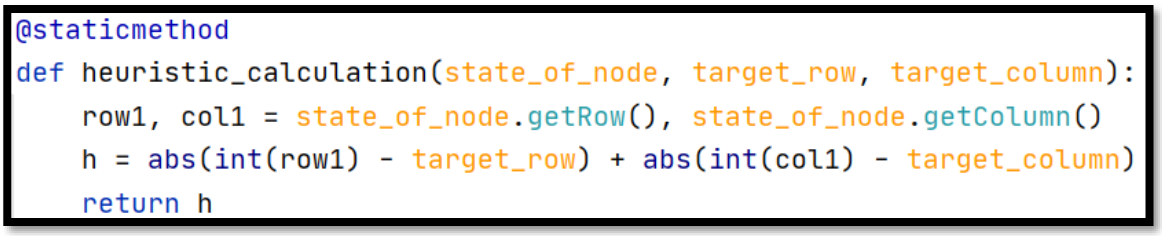


```python
def successors(self, state):
    mov = None
    successors = []
    neighbors = state.getLinks()

    for cell in neighbors:
        if state.getRow() - 1 == cell.getRow():
            mov = "N"
        if state.getRow() + 1 == cell.getRow():
            mov = "S"
        if state.getColumn() - 1 == cell.getColumn():
            mov = "O"
        if state.getColumn() + 1 == cell.getColumn():
            mov = "E"

        new_state = State((cell.getRow(), cell.getColumn()), cell.getLinks(), cell.getValue())
        cost = int(cell.getValue()) + 1
        successors.append([mov, new_state, cost])

    return successors
```

- heuristic_calculation function

This method simply calculate the heuristic according to the operation described in the guideline of the task 3. This method was added in task number 3.

H = |row – target_row| + |col – target_col|



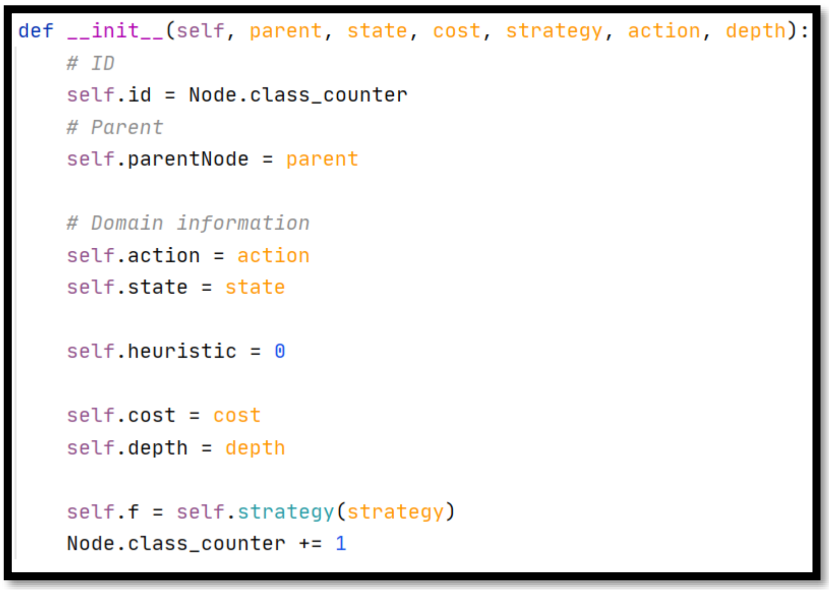
```python
@staticmethod
def heuristic_calculation(state_of_node, target_row, target_column):
    row1, col1 = state_of_node.getRow(), state_of_node.getColumn()
    h = abs(int(row1) - target_row) + abs(int(col1) - target_column)
    return h
```

4.2.3. Node Class

Class to represent a node of the search tree. A node has the following attributes: id, parent node, action, state, heuristic (in informed strategies), cost and depth.

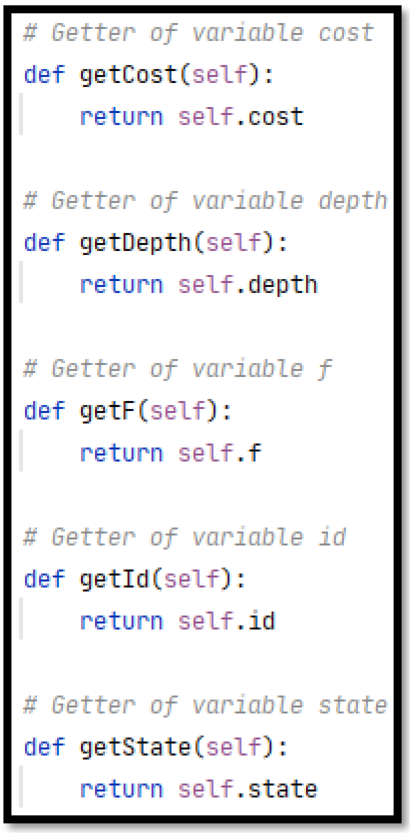
- Constructor

Class counter is a global variable that increments in 1 each time a Node object is created, and is used to assign id to the nodes.

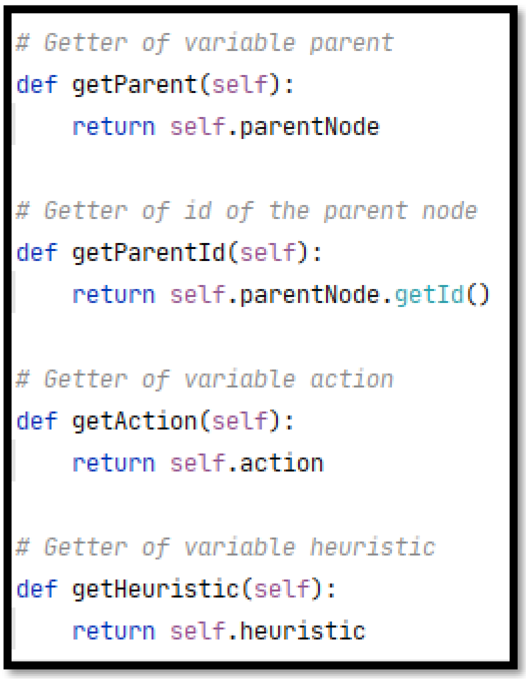


```python
def __init__(self, parent, state, cost, strategy, action, depth):
    # ID
    self.id = Node.class_counter
    # Parent
    self.parentNode = parent

    # Domain information
    self.action = action
    self.state = state

    self.heuristic = 0

    self.cost = cost
    self.depth = depth

    self.f = self.strategy(strategy)
    Node.class_counter += 1
```

- Getters



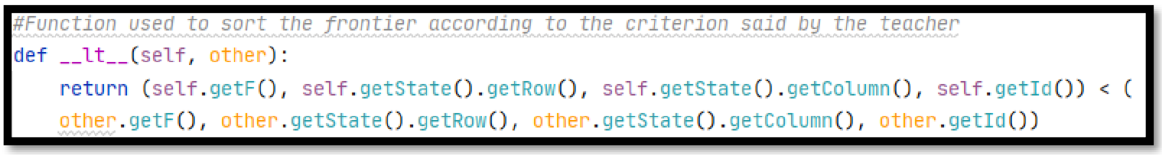
```python
# Getter of variable cost
def getCost(self):
    return self.cost

# Getter of variable depth
def getDepth(self):
    return self.depth

# Getter of variable f
def getF(self):
    return self.f

# Getter of variable id
def getId(self):
    return self.id

# Getter of variable state
def getState(self):
    return self.state
```



```python
# Getter of variable parent
def getParent(self):
    return self.parentNode

# Getter of id of the parent node
def getParentId(self):
    return self.parentNode.getId()

# Getter of variable action
def getAction(self):
    return self.action

# Getter of variable heuristic
def getHeuristic(self):
    return self.heuristic
```

- __lt__ method

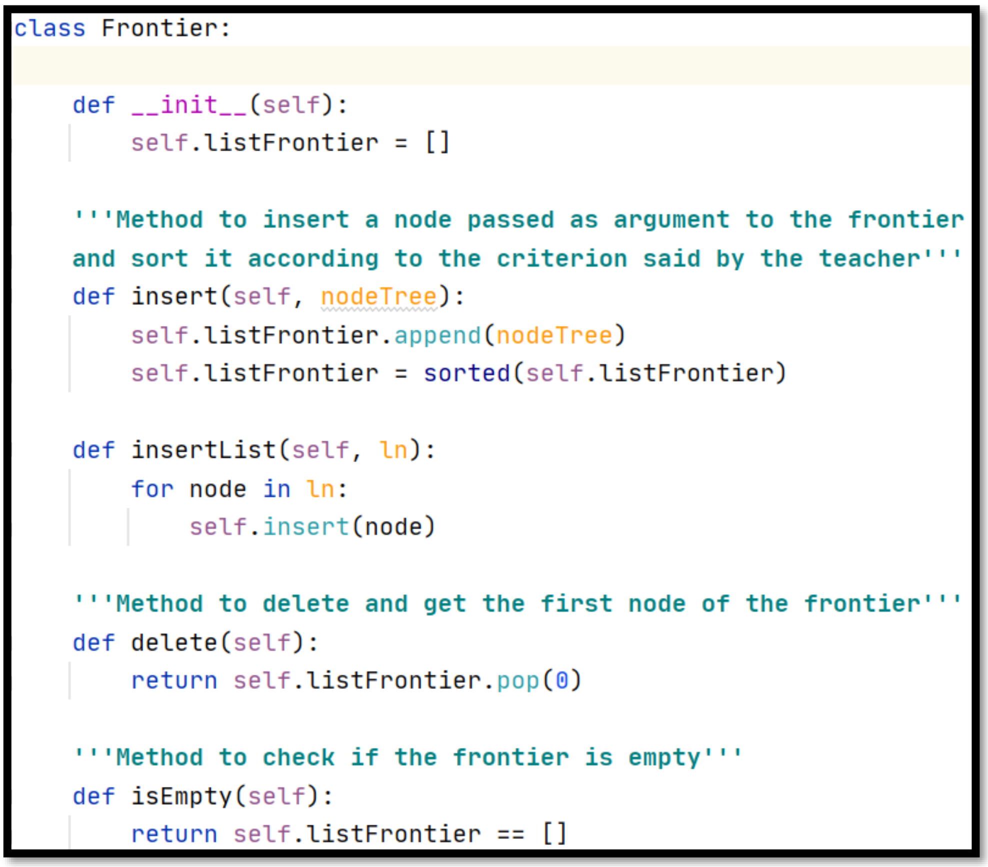
This method is to set the order of the frontier. With this method the Node objects that we insert in a list are automatically ordered following this criteria.



```python
#Function used to sort the frontier according to the criterion said by the teacher
def __lt__(self, other):
    return (self.getF(), self.getState().getRow(), self.getState().getColumn(), self.getId()) < (
    other.getF(), other.getState().getRow(), other.getState().getColumn(), other.getId())
```

4.2.4. Frontier Class

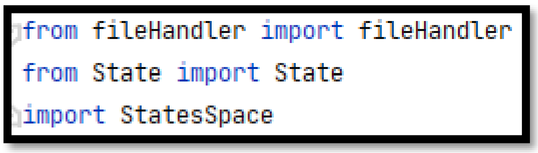
This class is used just to represent the frontier. It is a list and we have created the methods insertList, delete and isEmpty.



```python
class Frontier:

    def __init__(self):
        self.listFrontier = []

    '''Method to insert a node passed as argument to the frontier
    and sort it according to the criterion said by the teacher'''
    def insert(self, nodeTree):
        self.listFrontier.append(nodeTree)
        self.listFrontier = sorted(self.listFrontier)

    def insertList(self, ln):
        for node in ln:
            self.insert(node)

    '''Method to delete and get the first node of the frontier'''
    def delete(self):
        return self.listFrontier.pop(0)

    '''Method to check if the frontier is empty'''
    def isEmpty(self):
        return self.listFrontier == []
```

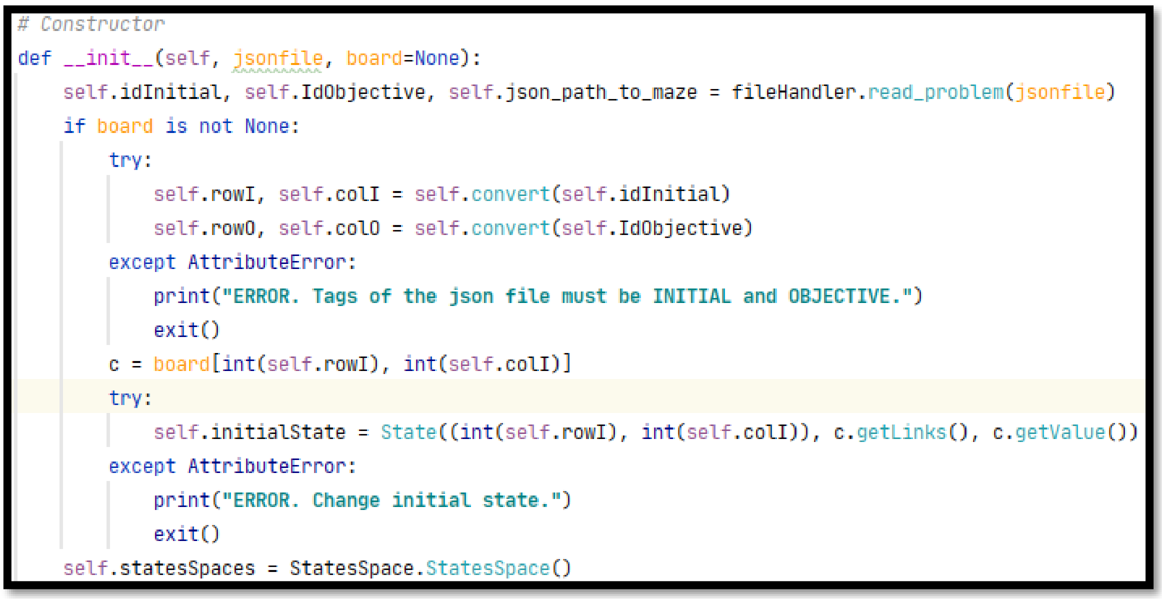
4.2.5. Problem Class

Class used to read all the information of the problem json, if board is none we only read initial state, final state and the path of the maze json. If we pass as parameter the board, we initialize the initial state. The imports are the following ones:



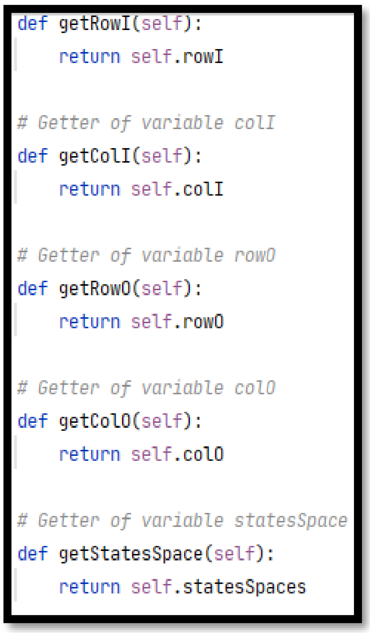
```python
from fileHandler import fileHandler
from State import State
import StatesSpace
```

- Constructor

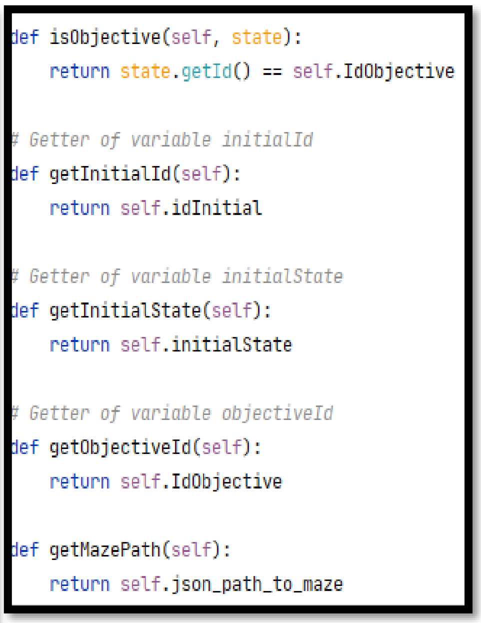


```python
# Constructor
def __init__(self, jsonfile, board=None):
    self.idInitial, self.IdObjective, self.json_path_to_maze = fileHandler.read_problem(jsonfile)
    if board is not None:
        try:
            self.rowI, self.colI = self.convert(self.idInitial)
            self.rowO, self.colO = self.convert(self.IdObjective)
        except AttributeError:
            print("ERROR. Tags of the json file must be INITIAL and OBJECTIVE.")
            exit()
        c = board[int(self.rowI), int(self.colI)]
        try:
            self.initialState = State((int(self.rowI), int(self.colI)), c.getLinks(), c.getValue())
        except AttributeError:
            print("ERROR. Change initial state.")
            exit()
    self.statesSpaces = StatesSpace.StatesSpace()
```

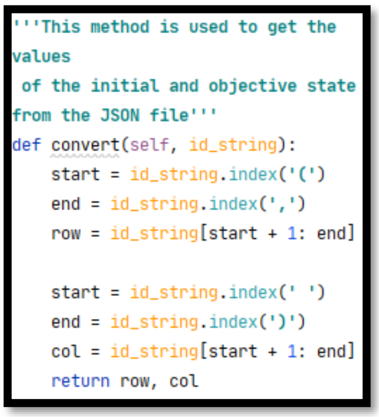
- Getters, isObjective and convert method



```python
def getRowI(self):
    return self.rowI

# Getter of variable colI
def getColI(self):
    return self.colI

# Getter of variable rowO
def getRowO(self):
    return self.rowO

# Getter of variable colO
def getColO(self):
    return self.colO

# Getter of variable statesSpace
def getStatesSpace(self):
    return self.statesSpaces
```



```python
def isObjective(self, state):
    return state.getId() == self.IdObjective

# Getter of variable initialId
def getInitialId(self):
    return self.idInitial

# Getter of variable initialState
def getInitialState(self):
    return self.initialState

# Getter of variable objectiveId
def getObjectiveId(self):
    return self.IdObjective

def getMazePath(self):
    return self.json_path_to_maze
```



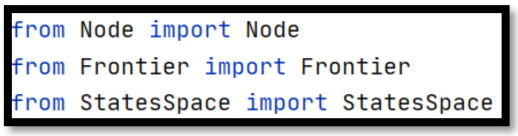
```python
'''This method is used to get the
values
 of the initial and objective state
from the JSON file'''
def convert(self, id_string):
    start = id_string.index('(')
    end = id_string.index(',')
    row = id_string[start + 1: end]

    start = id_string.index(' ')
    end = id_string.index(')')
    col = id_string[start + 1: end]
    return row, col
```

4.3. Task 3

The goals of this task are to introduce different costs in the implemented actions, too define an admissible heuristic function, to implement the search algorithm in the State Space and finally to solve the problem with the following strategies: Breadth, Uniform, Cost, Depth, Greedy and A*.

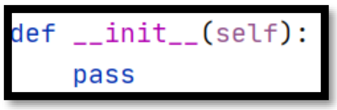
4.3.1. Search Class

Class to store the search algorithm and all the methods involved in the search. We needed to import:



```python
from Node import Node
from Frontier import Frontier
from StatesSpace import StatesSpace
```

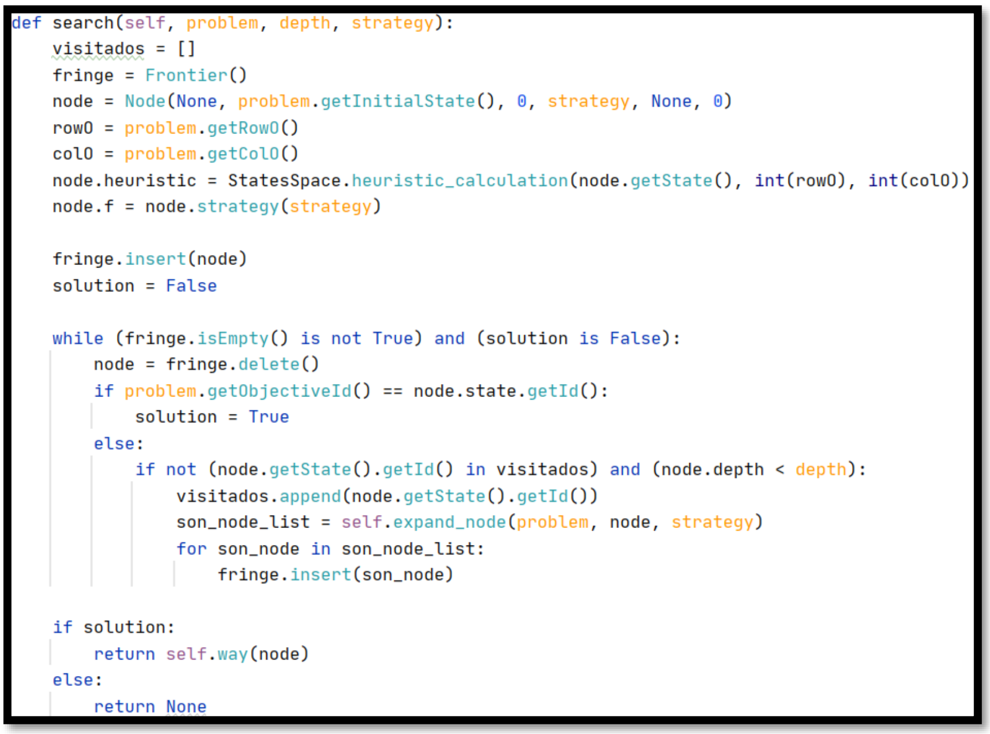
- Constructor



```python
def __init__(self):
    pass
```

- Search algorithm

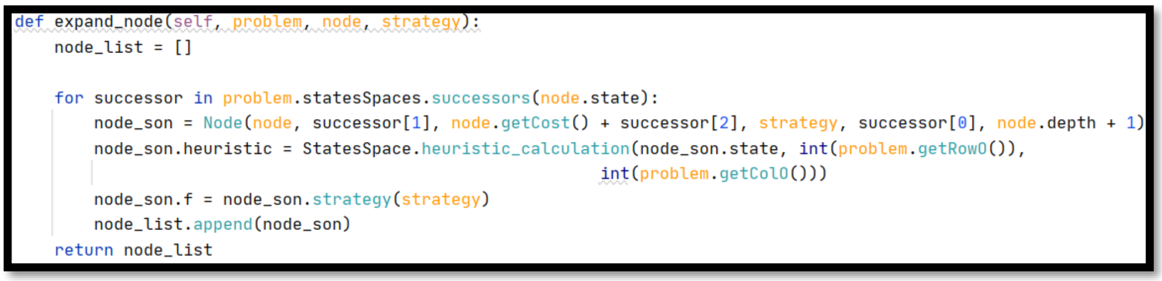
Search algorithm based on the one given us in moodle, in a visited nodes list and the method expand node.



```python
def search(self, problem, depth, strategy):
    visitados = []
    fringe = Frontier()
    node = Node(None, problem.getInitialState(), 0, strategy, None, 0)
    row0 = problem.getRow0()
    col0 = problem.getCol0()
    node.heuristic = StatesSpace.heuristic_calculation(node.getState(), int(row0), int(col0))
    node.f = node.strategy(strategy)

    fringe.insert(node)
    solution = False

    while (fringe.isEmpty() is not True) and (solution is False):
        node = fringe.delete()
        if problem.getObjectiveId() == node.state.getId():
            solution = True
        else:
            if not (node.getState().getId() in visitados) and (node.depth < depth):
                visitados.append(node.getState().getId())
                son_node_list = self.expand_node(problem, node, strategy)
                for son_node in son_node_list:
                    fringe.insert(son_node)

    if solution:
        return self.way(node)
    else:
        return None
```

- expand_node method

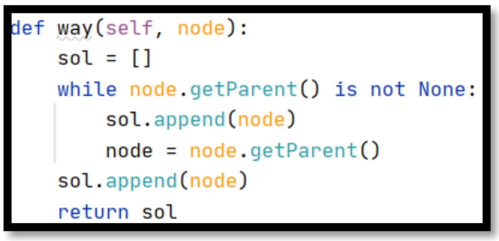
Method to expand a node based on the search algorithm given on moodle.



```
def expand_node(self, problem, node, strategy):
    node_list = []

    for successor in problem.statesSpaces.successors(node.state):
        node_son = Node(node, successor[1], node.getCost() + successor[2], strategy, successor[0], node.depth + 1)
        node_son.heuristic = StatesSpace.heuristic_calculation(node_son.state, int(problem.getRow0()),
                                                                int(problem.getCol0()))
        node_son.f = node_son.strategy(strategy)
        node_list.append(node_son)
    return node_list
```

- way

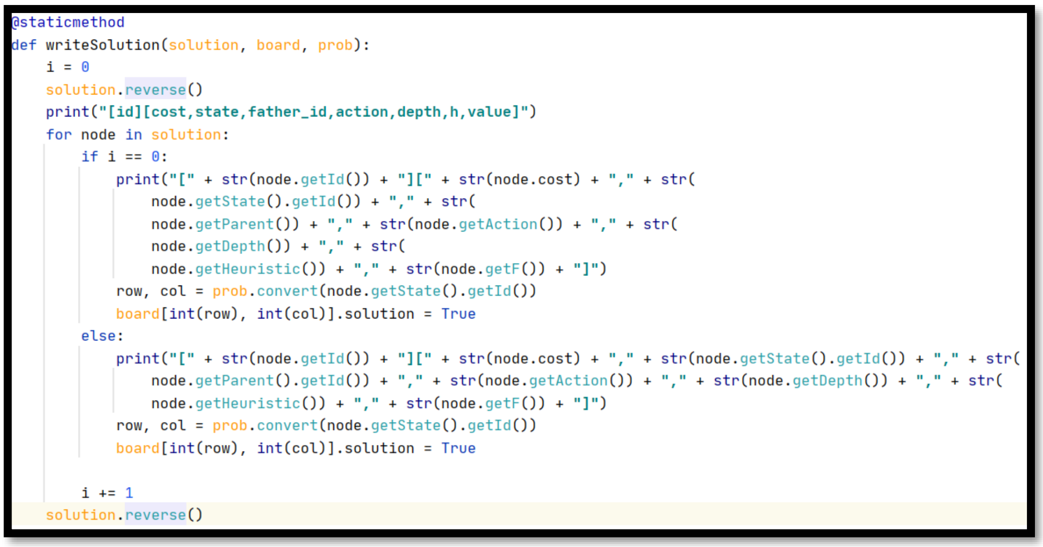
We use this method to return the solution path, we begin from final node and getting its parent until we arrive to the initial node that has no parent node.



```
def way(self, node):
    sol = []
    while node.getParent() is not None:
        sol.append(node)
        node = node.getParent()
    sol.append(node)
    return sol
```

- writeSolution method

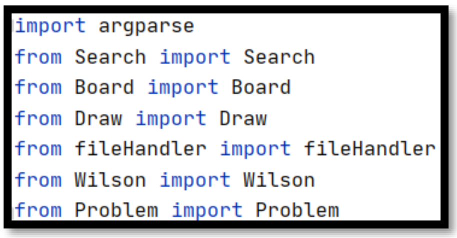
Just a simply method to print the path also in console. At the end we have to use reverse() in order to not alter the txt file when saving. In we reverse then we reverse another time.



```
@staticmethod
def writeSolution(solution, board, prob):
    i = 0
    solution.reverse()
    print("[id][cost,state,father_id,action,depth,h,value]")
    for node in solution:
        if i == 0:
            print("[" + str(node.getId()) + "][" + str(node.cost) + "," + str(
                node.getState().getId()) + "," + str(
                node.getParent()) + "," + str(node.getAction()) + "," + str(
                node.getDepth()) + "," + str(
                node.getHeuristic()) + "," + str(node.getF()) + "]")
            row, col = prob.convert(node.getState().getId())
            board[int(row), int(col)].solution = True
        else:
            print("[" + str(node.getId()) + "][" + str(node.cost) + "," + str(node.getState().getId()) + "," + str(
                node.getParent().getId()) + "," + str(node.getAction()) + "," + str(node.getDepth()) + "," + str(
                node.getHeuristic()) + "," + str(node.getF()) + "]")
            row, col = prob.convert(node.getState().getId())
            board[int(row), int(col)].solution = True

        i += 1
    solution.reverse()
```

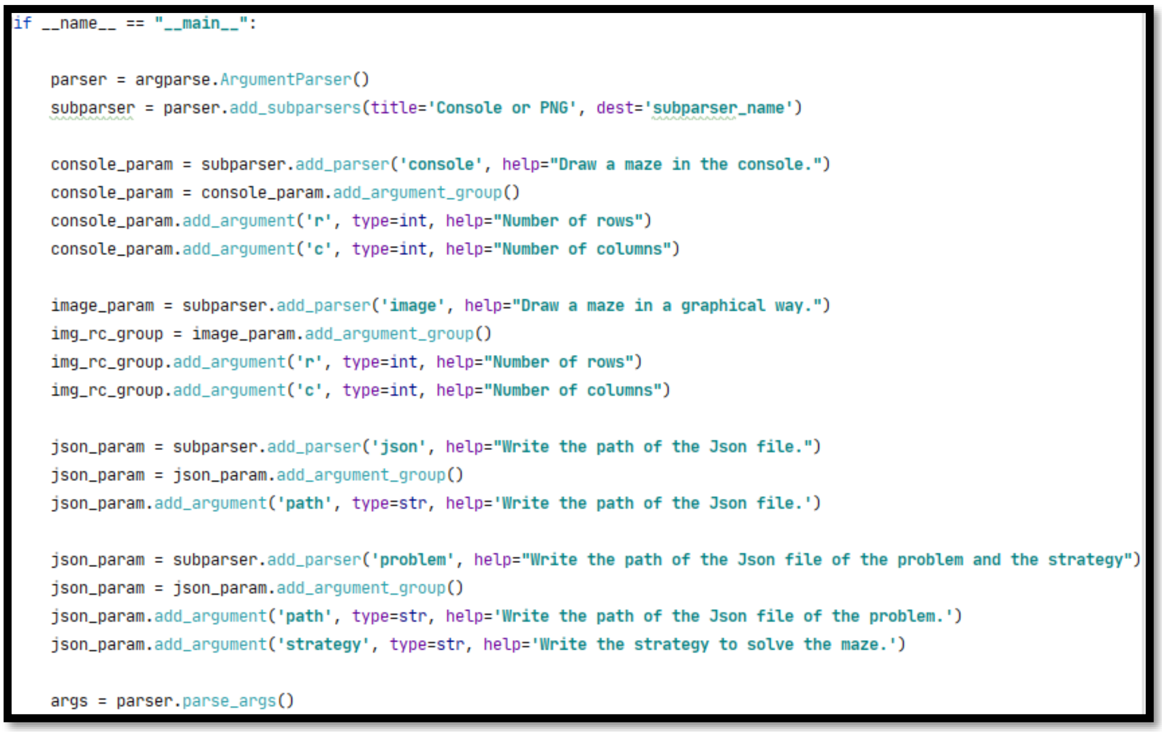
4.3.2. Main Class “Maze.py”

This is the main class of the project, the one that we execute when we want to run the program. We do several imports, and we have to highlight argparse library, used to manage the different options of execution and the arguments that each param receive.



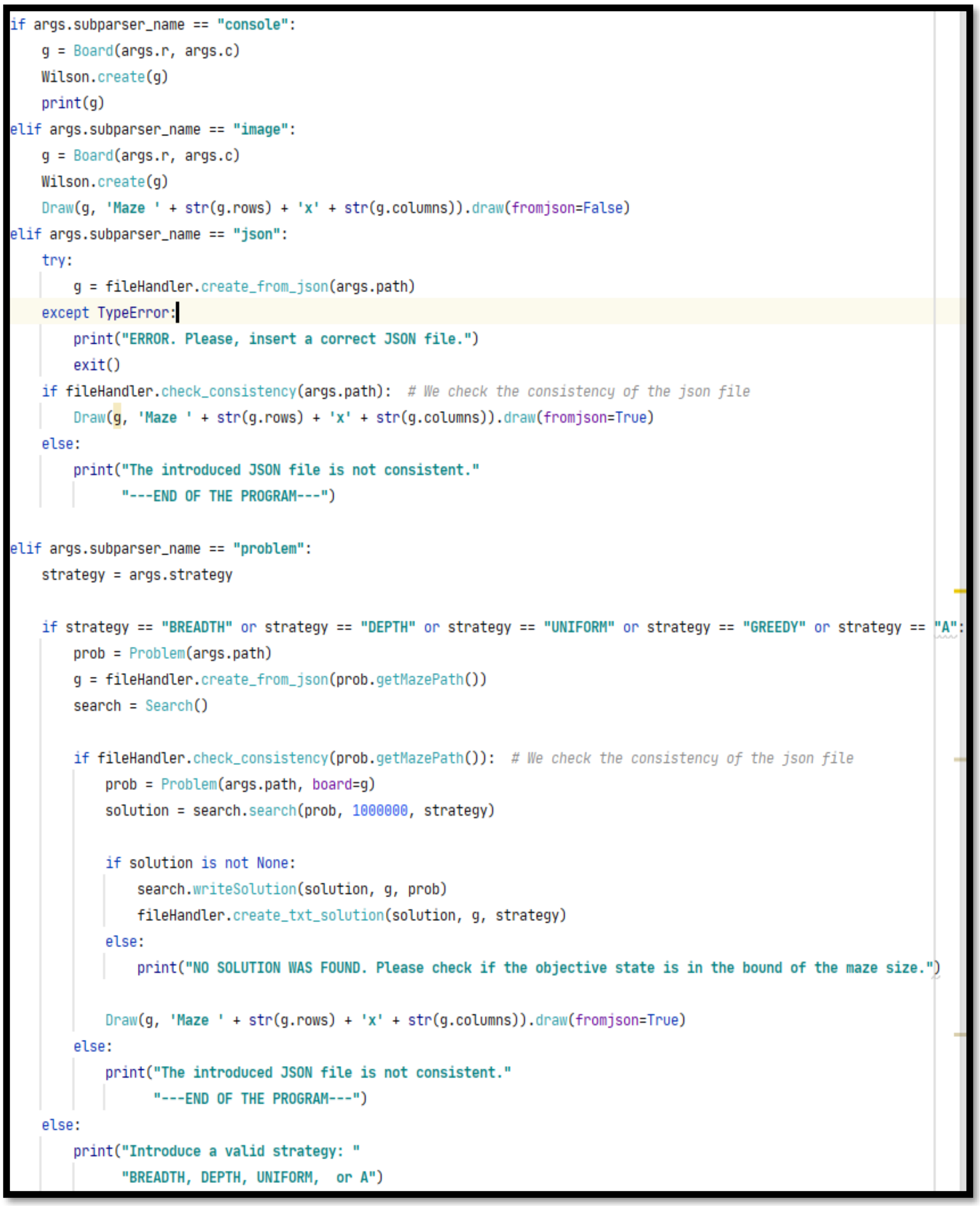
```python
import argparse
from Search import Search
from Board import Board
from Draw import Draw
from fileHandler import fileHandler
from Wilson import Wilson
from Problem import Problem
```

- Management of the command line



```python
if __name__ == "__main__":

    parser = argparse.ArgumentParser()
    subparser = parser.add_subparsers(title='Console or PNG', dest='subparser_name')

    console_param = subparser.add_parser('console', help="Draw a maze in the console.")
    console_param = console_param.add_argument_group()
    console_param.add_argument('r', type=int, help="Number of rows")
    console_param.add_argument('c', type=int, help="Number of columns")

    image_param = subparser.add_parser('image', help="Draw a maze in a graphical way.")
    img_rc_group = image_param.add_argument_group()
    img_rc_group.add_argument('r', type=int, help="Number of rows")
    img_rc_group.add_argument('c', type=int, help="Number of columns")

    json_param = subparser.add_parser('json', help="Write the path of the Json file.")
    json_param = json_param.add_argument_group()
    json_param.add_argument('path', type=str, help='Write the path of the Json file.')

    json_param = subparser.add_parser('problem', help="Write the path of the Json file of the problem and the strategy")
    json_param = json_param.add_argument_group()
    json_param.add_argument('path', type=str, help='Write the path of the Json file of the problem.')
    json_param.add_argument('strategy', type=str, help='Write the strategy to solve the maze.')

    args = parser.parse_args()
```

- Main execution



```python
if args.subparser_name == "console":
    g = Board(args.r, args.c)
    Wilson.create(g)
    print(g)
elif args.subparser_name == "image":
    g = Board(args.r, args.c)
    Wilson.create(g)
    Draw(g, 'Maze ' + str(g.rows) + 'x' + str(g.columns)).draw(fromjson=False)
elif args.subparser_name == "json":
    try:
        g = fileHandler.create_from_json(args.path)
    except TypeError:
        print("ERROR. Please, insert a correct JSON file.")
        exit()
    if fileHandler.check_consistency(args.path):  # We check the consistency of the json file
        Draw(g, 'Maze ' + str(g.rows) + 'x' + str(g.columns)).draw(fromjson=True)
    else:
        print("The introduced JSON file is not consistent."
              "---END OF THE PROGRAM---")

elif args.subparser_name == "problem":
    strategy = args.strategy

    if strategy == "BREADTH" or strategy == "DEPTH" or strategy == "UNIFORM" or strategy == "GREEDY" or strategy == "A":
        prob = Problem(args.path)
        g = fileHandler.create_from_json(prob.getMazePath())
        search = Search()

        if fileHandler.check_consistency(prob.getMazePath()):  # We check the consistency of the json file
            prob = Problem(args.path, board=g)
            solution = search.search(prob, 1000000, strategy)

            if solution is not None:
                search.writeSolution(solution, g, prob)
                fileHandler.create_txt_solution(solution, g, strategy)
            else:
                print("NO SOLUTION WAS FOUND. Please check if the objective state is in the bound of the maze size.")

            Draw(g, 'Maze ' + str(g.rows) + 'x' + str(g.columns)).draw(fromjson=True)
        else:
            print("The introduced JSON file is not consistent."
                  "---END OF THE PROGRAM---")
    else:
        print("Introduce a valid strategy: "
              "BREADTH, DEPTH, UNIFORM,  or A")
```

5. Personal assessment of the practice model

Ernesto Plata Fernández. The practice of the labyrinth has been a great challenge for me. A practice that has taken a lot of work and has meant a lot of learning when programming, such as knowing how to use a JSON file correctly or the use of different algorithms.

Juan Manuel Porrero Almansa. This practice is one of the most complicated we have done so far. It has been a total challenge for me, as subjects like the JSON archives I had never seen before and I found it a bit difficult to understand them and what they were for. The language was practically for me, and that's partly why we chose it, as it was a challenge and in the end it has become my favourite programming language, over and above the Java I was more used to. The concepts used by these intelligent search systems have been quite complex to implement and take a long time, but the final result is very satisfactory and I am very happy as the program works perfectly, as does the subject matter teacher. It is a great satisfaction to see how it works correctly. I qualify it as a challenge that has been overcome.

José Carlos Gualo Cejudo. It seems to me that this practice has been quite positive, since it has allowed us to acquire the expected knowledge in terms of search algorithms and both the use of a successor function and the borderline seen in the theory classes, in a way that is understandable when dealing with a case we know as the resolution of a maze.

6. Assessment of the team.

Juan Manuel Porrero Almansa. It seems to me that we have worked as a team from beginning to end, communicating in any case about everything we did, giving each other support, if one didn't know how to do one thing or found it more difficult, the other helped him or vice versa. Everyone's involvement was perfect and we respected all the video calls we were making. As far as the personal aspect is concerned, I'm also very happy with the work, as it has been a challenge and I think we've more than overcome it. Therefore, very happy with the result obtained, so it is a luxury to work as a team!

Ernesto Plata Fernández. From the first moment we started with the practice, we held multiple meetings to see how to carry it out. There were no problems in carrying it out and I especially wanted to highlight the great work of my colleagues: Juan Manuel Porrero and Jose Carlos Gualo, who have helped me at all times.

José Carlos Gualo Cejudo. I think the group's behaviour has been quite good, as we haven't had any internal problems or anything like that. It is worth noting the great perseverance in solving the various problems and difficulties that were put in our way.