Module `main`

Functions

Function `main`

```
def main()
```

The function that will start the game

Name : main()

Import needed :

`-argparse : - pip install argparse -`

Arguments :

`-f --file : take a maze file and print its solution. Required.`

`-g --gui : print the solution in a nice GUI. Optional.`

`-p --play : launch the game. Optional.`

Module `MazeGUI`

Classes

Class `MazeGUI`

```
class MazeGUI(
    win_title='PyMaze - AROF',
    win_height=900,
    win_width=900
)
```

Class that displays the game

Constructor of the class

Args ——= **`win_title`** : str, optional : title of the window. Defaults to “PyMaze - AROF”.

`win_height` : int, optional height of the window. Defaults to 900.
`win_width` : int, optional width of the windows. Defaults to 900.
`matrix` : str, optional matrix of the game.

Instance variables

Variable `clock` Getter of self.___clock

Variable `matrix` Getter method of self.___matrix

Variable `ressources` Getter method of self.___ressources

Variable `screen` Getter of self.___screen

Variable `win_height` Getter method of self.___win_height

Variable `win_title` Getter method of self.___win_title

Variable `win_width` Getter method of self.___win_width

Methods

Method `load_ressources`

```
def load_ressources(
    self
)
```

Method to load all the ressources of the game

Method `new_GUI`

```
def new_GUI(
    self
)
```

Create a new window

Method `screen_update`

```
def screen_update(
    self,
    matrix
)
```

Method that updates the current screen/window with the matrix in argument

Args —= **`matrix`** : list : 2D array of the game

Method `show_solution`

```
def show_solution(
    self,
    matrix,
    solution_path,
    refresh_rate=3
)
```

Method that will update the screen and show the path

Args —= **`matrix`** : list : matrix of the game

`solution_path` : list list of tuple position of the solutions.

Module `MazeSolver`

Classes

Class `MazeSolver`

```
class MazeSolver(
    maze
)
```

Instance variables

Variable `doors_available`

Variable `keys_available`

Variable `maze`

Variable `plus_courte_distance`

Variable `solution_finale`

Variable `solutions`

Methods

Method `get_co_of_the_key`

```
def get_co_of_the_key(
    self,
    key
)
```

Get the coordonates of the key given in paramater

Method `key_finding`

```
def key_finding(
    self,
    i,
    j,
    sol,
    path,
    distance=0
)
```

AVOID TO USE THIS FUNCTION DIRECTLY

Start from i and j and search for a key, then give the path to this key

The keys are checked from key_available

Method `path_finding`

```
def path_finding(
    self,
    i,
    j,
    sol,
    path,
    distance=0
```

```
)
```

AVOID TO USE THIS FUNCTION DIRECTLY

start from i and j and search for the end

Method `print_solution`

```
def print_solution(
    self
)
```

Method `solving`

```
def solving(
    self
)
```

Use this function to solve the maze

Module `MazeLoader`

Classes

Class `MazeLoader`

```
class MazeLoader
```

Class to load and use the map.

Constructor of the class MazeLoader.

Instance variables

Variable `matrix` Getter of self.__matrix

Returns ——= list : list: the matrix in the file in a 2D array (list).

Methods

Method `load_maze`

```
def load_maze(
    self,
    filepath
)
```

Method to load the maze and set the attribute matrix.

Args ——= **`filepath`** : str : the path to the file that contains the matrix.

Returns ——= list : the matrix in the file in a 2D array (list).

Module `MazeExamples`

Module `GameEngine`

Classes

Class `GameEngine`

```
class GameEngine
```

Class that represents the GameEngine. It will 'connect' everything together.

Instance variables

Variable `Maze` Getter method of Maze attribute.

Returns —= Maze : Instance of Maze

Variable `MazeGUI` Getter method of MazeGUI attribute.

Returns —= MazeGUI : Instance of MazeGUI

Variable `MazeLoader` Getter method of MazeLoader attribute.

Returns —= MazeLoader : Instance of MazeLoader

Variable `MazeSolver` Getter method of MazeSolver attribute.

Returns —= MazeSolver : Instance of MazeSolver

Variable `player_coords` Getter method of player_coords attribute.

Returns —= list : Instance of player_coords

Variable `player_key` Getter of player_key

Returns —= str : key possesed by the player

Variable `playermaze` Getter method of playermaze attribute.

Returns —= Maze : Instance of Maze

Methods

Method `check_win`

```
def check_win(
    self
)
```

Method `door_open`

```
def door_open(
    self,
    x,
    y
)
```

Method that returns true if the position enterred is a door

Method `init_game`

```
def init_game(
    self,
    maze_instance,
    mazeloader_instance,
    mazesolver_instance
)
```

Method that will initialize the game by initializing all the needed classes.

with the parser in main.py init_game will be called with the argument matrix_path as the path of the file

Method `is_door`

```
def is_door(
    self,
    x,
    y
)
```

Method that returns true if the position enterred is a door

Method `is_ghost`

```
def is_ghost(
    self,
    x,
    y
)
```

Method that returns true if the position enterred is a ghost block

Method `is_key`

```
def is_key(
    self,
    x,
    y
)
```

Method that will return if the current position is at a key.

Args ——= **`x`** : int : position in x

`y : int` position in y

Returns ——= int : 1 if is key 0 if is not key

Method `is_wall`

```
def is_wall(
    self,
    x,
    y
)
```

Method that returns true if the position enterred is a wall

Method `move_down`

```
def move_down(
    self
)
```

Method to move the player down

Returns —–= int : If the player moved

Method `move_left`

```
def move_left(
    self
)
```

Method to move the player left

Returns —–= int : If the player moved

Method `move_right`

```
def move_right(
    self
)
```

Method to move the player right

Returns —–= int : If the player moved

Method `move_up`

```
def move_up(
    self
)
```

Method to move the player up

Returns —–= int : If the player moved

Method `play`

```
def play(
    self
)
```

Method `sound_player`

```
def sound_player(
    self,
    sound
)
```

Sound player

Method `take_key`

```
def take_key(
    self,
    x,
    y
)
```

Module `Maze`

Classes

Class `Maze`

```
class Maze
```

What does this class do? please give it a description !!!

Instance variables

Variable `co_end`

Variable `co_start`

Variable `doors`

Variable `empty_maze`

Variable `ghost_coords`

Variable `ghosts`

Variable `keys`

Variable `length`

Variable `maze`

Methods

Method `affichage`

```
def affichage(
    self,
    maze
)
```

Temporary function to print a maze to the console

Method `find_end_start`

```
def find_end_start(
    self,
    maze: list
) -> list
```

Take a maze and returns the coordonates of the start and the end

Examples ——=

OUTPUT:

[(0,0),(9,9)]

Method `generateur_maze_vide`

```
def generateur_maze_vide(
    self,
    maze
)
```

Generate an empty maze, with the size of the maze

Method `get_doors`

```
def get_doors(
    self,
    maze
) -> list
```

Return a list of the doors available in the maze

Method `get_keys`

```
def get_keys(
    self,
    maze
) -> list
```

Return a list of the keys available in the maze

Method `ghost_block`

```
def ghost_block(
    self
)
```

This function is made to take the position of the of ghosts and place -1 on it range.

Args ——= **`maze`** : maze : take any maze of maze

Returns ——= maze : Return the maze update with the range of ghosts

Method `initializer`

```
def initializer(
    self,
    matrice
)
```

Initialize the Maze class with a matrix, get all the ghost, find the start the end, put the range of the ghost,etc.

Method `solve_ghost_coords`

```
def solve_ghost_coords(
    self,
    matrice
)
```

Return the list of coords of the ghosts

Module `test_class`

Classes

Class `TestClass`

```
class TestClass
```

Methods

Method `test_co_end`

```
def test_co_end(
    self
)
```

Method `test_co_start`

```
def test_co_start(
    self
)
```

Method `test_length`

```
def test_length(
    self
)
```

Method `test_solution`

```
def test_solution(
    self
)
```
