```
''' Implementation of a non-static version of the game Word Search Puzzle.

'''

from random import shuffle, randint, sample
from string import ascii_uppercase
import xml.etree.ElementTree as ET
import glob, os, time, re


__author__ = "Oscar Guillen, Patricia Reinoso"
__date__ = "17/02/2017"
__version__ = "1"

commands = ["help","exit","rotate","find"]
bool_answers = ["yes","no"]
directions = ["right","left","up","down"]
messages = {
                                "already_found_word":" was already found.",
                                "choose_subject":"Please, choose a subject...\n",
                                "error": "\nAn error occurred. Leaving the game.\n",
                                "error_instruction": "\nAn error occurred. No instructions. "\
                                                                                "Leaving the game.\n",
                                "enter_to_continue":"\nEnter to Continue.",
                                "exit_game": "\nLeaving the game.\n",
                                "final_cell": "\nPlease, insert the coordinates of the final cell:",
                                "good_word":"\nCongratulations! You just found ",
                                "highscore": "NEW HIGHSCORE",
                                "incorret_format":"\nERROR: Incorrect format. It was not added.",
                                "incorret_format_clue":"\nERROR: Incorrect format. It was not "\
                                                                                                "selected.",
                                "incorret_word":" is not on the list.",
                                "initial_cell": "\nPlease, insert the coordinates of the initial"\
                                                                                        " cell:",
                                "insert_column_number": "Column number ?  ",
                                "insert_command": "\nAction [rotate|find|help|exit] ? ",
                                "insert_direction": "Direction [up|down|right|left] ? ",
                                "insert_menu_action": "\nAction [p|h|e] ? ",
                                "insert_number_spaces": "Spaces ? ",
                                "insert_row_number": "Row number ?  ",
                                "invalid_cell": "\nERROR: That cell is not inside the board.",
                                "invalid_column": "\nERROR: That column is not inside the board.",
                                "invalid_command": "\nERROR: That is an invalid command. Only "\
                                                                                        "'rotate', 'find', 'help', 'exit' are possible.\n",
                                "invalid_direction": "\nERROR: That is an invalid direction. Only"\
                                                                                        " 'right', 'left', 'up', 'down' are
possible.",
                                "invalid_menu_option":"\nERROR: Invalid option. Only 'p' for play,"\
                                                                                                " 'h' for help and 'e' for exit.
\n",
                                "invalid_number_spaces": "\nERROR: Invalid number of spaces.",
                                "invalid_play_again_option": "\nERROR: Invalid option. Only 'yes'"\
                                                                                                        " or 'no'.",
                                "invalid_row": "\nERROR: That row is not inside the board.",
                                "invalid_subject": "\nERROR: It is not a valid subject. Try "\
                                                                                        "again.\n",
                                "it_is_not_a_line": "\nERROR: The cells must form a vertical or a"\
                                                                                        " horizontal line of at least 3 letters.",
                                "it_is_not_an_integer": "\nERROR: It must be an integer.",
                                "menu": "MENU\n\n[p] Play\n[h] Help\n[e] Exit\n",
                                "play_again": "\n\nPlay again [yes|no] ? ",
                                "separator": "\n----------------------------------------------"\
                                                                        "----------------\n",
                                "setting_up": "Setting up...",
                                "single_player_modes":"MODE\n\nPractice mode\n",
                                "starting":"Starting...",
                                "get_subject":"Type here your subject: ",
                                "welcome":"WORD SEARCH PUZZLE\n\nLet's play!\n",
                                "win": "\n\n\nCongratulations! You found all the words. You won. ",
                                "winner":" is the winner.",
                                "words_to_find":"Words to find: "
                                }

class Instruction:
        ''' Manage all the information corresponding to the rules of
                the game.
        '''

        def __init__ (self):
                self.instruction = None

        def import_instruction(self, file):
                ''' Import the set of instructions of the game from a file.

                Parameters
                ----------
                file : str
                        Path of the file where the instructions are found.

                '''

                try:
                        f = open(file,'r')
                        self.instruction = f.read()
                except:
                        print(messages["error_instruction"])
                        exit()

        def display(self):
                ''''Display on the standard output the rules of the game.

                '''
                print(self.instruction)
```

```
class Board:
        ''' Manage the board where the words to be found are placed.

        '''

        def __init__(self):
                self.original_grid = [[]]
                self.current_grid = [[]]
                self.rows = 10
                self.columns = 10
                self.min_word_length = 3
                self.max_word_length = 8
                self.max_num_words = 10

        def build (self, words):
                ''' Add to the board all the words given and fill the blanks spaces left
                        with random letters .

                Parameters
                ----------
                        words : [str]
                                An array with the words to be placed on the board.

                '''
                self.current_grid = [['' for x in range(10)] for y in range(10)]
                num_words = len(words)
                words.sort(key=len)
                words.reverse()
                cols = [0,1,2,3,4,5,6,7,8,9]
                rows = [0,1,2,3,4,5,6,7,8,9]
                for word in words:
                        added = False
                        while not added:
                                position = randint(0,1) # 0 is row, 1 is column.
                                if position and len(cols) > 0:
                                        num = sample(cols,1)[0]
                                        added = self.add_word_into_row(num,word)
                                        if added:
                                                cols.remove(num)
                                elif not position and len(rows) > 0:
                                        num = sample(rows,1)[0]
                                        added = self.add_word_into_column(num,word)
                                        if added:
                                                rows.remove(num)
                self.fill()

                # Copy the built list.
                self.original_grid = []
                for row in self.current_grid:
                        self.original_grid.append(list(row))

        def is_valid_row(self,row):
                """ Check if the row number is inside the grid.

                Paramenters
                -----------
                row : int
                        row number.

                Returns
                -------
                bool
                        True for success, False otherwise.

                """
                return (row < self.rows and row >= 0)

        def is_valid_column(self,column):
                """ Check if the column number is inside the grid.

                Paramenters
                -----------
                column : int
                        column number.

                Returns
                -------
                bool
                        True for success, False otherwise.

                """
                return (column < self.columns and column >= 0)

        def is_valid_number_spaces(self,direction,number):
                """ Check if the number is to rotate is correct.

                Paramenters
                -----------
                direction : str
                        direction to rotate.
                number : int
                        number of spaces to rotate.

                Returns
                -------
                bool
                        True for success, False otherwise.

                """
                if number > 0 and \
                        (((direction == "up" or direction == "down") and \
                                number < self.columns) or\
                        ((direction == "left" or direction == "right") and\
                                number < self.rows)):
```

```
                                        return True
                    return False

          def rotate_horizontally(self, row, number, direction):
                    """ Rotates a row of the current grid a 'number' amount of spaces.

                    Paramenters
                    -----------
                    row : int
                              row number to rotate.
                    number : int
                              number of space to rotate.
                    direction : str
                              direction to rotate the row ("left" or "right").

                    Returns
                    -------
                    bool
                              True if the board was successfully rotate, False otherwise.

                    """
                    if self.is_valid_row(row):
                              number = number % self.rows
                              if direction == "right":
                                        self.current_grid[row] = self.current_grid[row][-number:] +\

self.current_grid[row][:-number]
                              elif direction == "left":
                                        self.current_grid[row] = self.current_grid[row][number:] +\

self.current_grid[row][:number]
                              return True
                    return False

          def rotate_vertically(self, column, number, direction):
                    """ Rotates a column of the current grid a 'number' amount of spaces.

                    Paramenters
                    -----------
                    column : int
                              column number to rotate.
                    number : int
                              number of space to rotate.
                    direction : str
                              direction to rotate the column ("up" or "down").

                    Returns
                    -------
                    bool
                              True if the board was successfully rotate, False otherwise.

                    """
                    if self.is_valid_column(column):
                              number = number % self.columns
                              temp_col=[]
                              for i in range(self.columns):
                                        temp_col.append(self.current_grid[i][column])

                              if direction == "up":
                                        temp_col = temp_col[number:] + temp_col[:number]
                              elif direction == "down":
                                        temp_col = temp_col[-number:] + temp_col[:-number]

                              for i in range(self.columns):
                                        self.current_grid[i][column]=temp_col[i]
                              return True
                    return False

          def get_letter(self,row,column):
                    ''' Return the letter that inside a cell on the board.

                    Paramenters
                    -----------
                    row : int
                              row of the cell of the letter wanted.
                    column : int
                              column of the cell of the letter wanted.

                    Returns
                    -------
                    str
                              Letter extratect from the cell of the board.

                    '''
                    if self.is_valid_cell(row,column):
                              return self.current_grid[row][column]
                    return None

          def display(self):
                    """ Display the grid on the standard output. Row and columns numbers
                    are included.

                    """
                    print ("\n   ", end='')
                    for i in range (self.columns):
                              print (str(i) + "   ", end='')
                    print("\n")
                    for i in range (self.rows):
                              for j in range (self.columns):
                                        if j == 0:
                                                  print (str(i) + "  ", end='')
                                        print (self.get_letter(i,j) + "   ", end='')
                                        if j == self.rows - 1:
                                                  print (str(i), end='')
```

```
                        print("\n")
                print ("   ", end='')
                for i in range (self.columns):
                        print (str(i) + "   ", end='')
                print("\n")


        def is_valid_cell(self,row,column):
                """ Check if the cell is inside the grid.

                Paramenters
                -----------
                row : int
                        row number of the cell.
                column : int
                        column number of the cell.

                Returns
                -------
                bool
                        True for success, False otherwise.

                """
                if self.is_valid_row(row) and self.is_valid_column(column):
                        return True
                return False


        def can_generate_a_word(self,row1,column1,row2,column2):
                ''' Receive 2 cells and check they are valid, if they form a vertical
                        or an horizontal line, and if they form a word long enough.

                Paramenters
                -----------
                row1 : int
                        row number of the starting cell.
                column1 : int
                        column number of the starting cell.
                row2 : int
                        row number of the final cell.
                column2 : int
                        column number of the final cell.

                Returns
                -------
                bool
                        True for success, False otherwise.

                '''
                if  (self.is_valid_cell(row1,column1) and
                         self.is_valid_cell(row2,column2) and \

                        (row1 == row2 and column1 != column2 and
                         abs(column1 - column2) + 1 >= self.min_word_length) or \

                        (column1 == column2 and row1 != row2 and \
                        abs(row1 - row2) + 1 >= self.min_word_length)):
                        return True
                return False


        def get_word(self,row1,column1,row2,column2):
                """ Get the word from the grid. The word is specified by a starting cell
                        and a final cell.

                Paramenters
                -----------
                        row1 : int
                                row number of the starting cell.
                        column1 : int
                                column number of the starting cell.
                        row2 : int
                                row number of the final cell.
                        column2 : int
                                column number of the final cell.

                Returns
                -------
                str
                        Word obtained from the grid. None if the cells are not valid.

                """

                if self.can_generate_a_word(row1,column1,row2,column2):
                        word = ""
                        # The word is placed horizontally
                        if (row1 == row2):
                                start = min(column1,column2)
                                end = max(column1,column2)
                                for i in range (start,end+1):
                                        word = word + self.get_letter(row1,i)
                                return word
                        # The word is placed vertically
                        elif (column1 == column2):
                                start = min(row1,row2)
                                end = max(row1,row2)
                                for i in range (start,end+1):
                                        word = word + self.get_letter(i,column1)
                                return word
                else:
                        return None

```

```
333         def add_word_into_row(self, row, word):
334                 ''' Insert a word into a row of the board.
335
336                 Parameters
337                 ----------
338                 row: int
339                         The number of the row.
340                 word: str
341                         The word to be inserted.
342
343                 Returns
344                 -------
345                 bool
346                         True for success, False otherwise.
347
348                 '''
349                 column = randint(0,9)
350                 aux_column = column
351                 # Checking the posibility to add the word
352                 for letter in word:
353                         if aux_column > 9:
354                                 aux_column = 0
355                         if self.current_grid[row][aux_column] != '':
356                                 if self.current_grid[row][aux_column] != letter:
357                                         return False
358                         aux_column += 1
359                 # Adding the word.
360                 for letter in word:
361                         if column > 9:
362                                 column = 0
363                         if self.current_grid[row][column] == '':
364                                 self.current_grid[row][column] = letter
365                         column += 1
366                 return True
367
368         def add_word_into_column(self, column, word):
369                 ''' Insert a word into a column of the board.
370
371                 Parameters
372                 ----------
373                 column: integer
374                         The number of the column.
375                 word: string
376                         The word to be inserted.
377
378                 Returns
379                 -------
380                 bool
381                         True for success, False otherwise.
382
383                 '''
384                 row = randint(0,9)
385                 aux_row = row
386                 # Checking the posibility to add the word
387                 for letter in word:
388                         if aux_row > 9:
389                                 aux_row = 0
390                         if self.current_grid[aux_row][column] != '':
391                                 if self.current_grid[aux_row][column] != letter:
392                                         return False
393                         aux_row += 1
394                 # Adding the word.
395                 for letter in word:
396                         if row > 9:
397                                 row = 0
398                         if self.current_grid[row][column] == '':
399                                 self.current_grid[row][column] = letter
400                         row += 1
401                 return True
402
403         def fill(self):
404                 ''' Fill the blank spaces of the board with random letters.
405
406                 '''
407
408                 alfabet = ascii_uppercase
409                 for i in range(10):
410                         for j in range(10):
411                                 if self.current_grid[i][j] == '':
412                                         self.current_grid[i][j] = sample(alfabet,1)[0]
413
414         def get_max_num_words(self):
415                 ''' Return the maximun number of words allowed.

416
417                 Returns
418                 -------
419                 int
420                         The maximun number of words.
421
422                 '''
423                 return self.max_num_words
424
425         def get_min_length_word(self):
426                 ''' Return the Minimun length of a word allowed.
427
428                 Returns
429                 -------
430                 int
431                         The minimun length.
432                 '''
433                 return self.min_word_length
434
435         def get_max_length_word(self):
436                 ''' Return the Maximun length of a word allowed.
```

```
                    Returns
                    -------
                    int
                              The maximun length

                    '''
                    return self.max_word_length
class Subject:
          ''' Manage the subjects all the words on a board are related to. '''

          def __init__(self):
                    self.name = None
                    self.content = []

          def add_word(self, word):
                    ''' Add a word into the subject.

                    Parameters
                    ----------
                    word : str
                              Word to be added.

                    '''

                    if word is not None:
                              self.content.append(word)


          def import_subject(self, file):
                    ''' Import the content of a well-formed xml into a subject.

                    Parameters
                    ----------
                    file : str
                              The path of the file.

                    '''

                    # Parsing XML
                    # xml.etree.ElementTree from Python The ElementTree XML API
                    # Source code: https://docs.python.org/3.4/library/xml.etree.elementtree.html
                    tree = ET.parse(file)
                    root = tree.getroot()
                    words = 0
                    if len(root) > 0:
                              self.name = root[0].attrib['name']
                              root = root[0]
                              for child in root:
                                        if child. tag == "word":
                                                  new_word = child.text
                                                  # Parsing a string using Regular Expression
                                                  # re from Python Regular expression operations
                                                  # Source code: https://docs.python.org/3.4/library/re.html
                                                  if re.match('^[a-zA-Z]+$', new_word) is not None:
                                                            self.add_word(new_word)
                                                            words += 1
                    else:
                              return False
                    return words >= 12


          def get_name(self):
                    ''' Return the name of the subject.

                    Returns
                    -------
                    str
                              Name of the subject.

                    '''
                    return self.name

          def get_words(self):
                    ''' Return all the words of the subject.

                    Returns
                    -------
                    [str]
                              All the words of the subject. Can be empty.

                    '''
                    return self.content


class Dictionary:
          ''' Manage all the subjects available on the game.

          '''

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

          def add_subject(self, subject):
                    ''' Add a subject object into the dictionary.

                    Parameters
                    ----------
                    subject: Subject
                              Subject to be added.

                    '''
                    self.subjects.append(subject)
```

```
499
500             def display_subjects(self):
501                     ''' Display on the standard output the list of the names of the
502                     all the subjects.
503
504                     '''
505                     print('Subjects: ')
506                     for subject in self.subjects:
507                             print('- ' + subject.get_name())
508                     print('')
509
510             def load(self):
511                     ''' Load the dictionary from xml files into subjects directory.
512
513                     Returns
514                     -------
515                     bool
516                             True for success, False otherwise.
517
518                     '''
519
520                     # Manage files and directories.
521                     # os.walk from Python Miscellaneous operating system interfaces.
522                     # Source code: https://docs.python.org/3.4/library/os.html#os.walk
523                     try:
524                             num_of_correct_subjects = 0
525                             for root, dirs, files in os.walk("subjects"):
526                                     for file in files:
527                                             if file.endswith(".xml"):
528                                                     new_subject = Subject()
529                                                     correct = new_subject.import_subject(os.path.join(root,\
530                                                                     file))
531                                                     if correct:
532                                                             self.add_subject(new_subject)
533                                                             num_of_correct_subjects += 1
534                             return num_of_correct_subjects > 0
535                     except:
536                             return False
537
538
539             def get_subject_by_name(self, name):
540                     ''' Return a subject using its name element.
541
542                     Parameters
543                     ----------
544                     name : str
545                             The name of the subject as a string.
546
547                     Returns
548                     -------
549                     Subject
550                             Subject corresponding to `name`.
551
552                     '''
553                     for subject in self.subjects:
554                             if subject.get_name().lower() == name.lower():
555                                     return subject
556                     return None
557
558     class Clue:
559             ''' Manage the list of words that the player needs to find on the game.
560
561             '''
562
563             def __init__(self):
564                     self.subject_name = None
565                     self.words_not_found = []
566                     self.words_found = []
567
568             def build(self, subject, num_words, min_length, max_length):
569                     ''' Build the clue, using the words of a subject, adding randomly
570                             the words into the not found list. There will be 10 clues per board
571                             by default.
572
573                     Parameters
574                     ----------
575                     subject : Subject
576                             Subject object that contain the words to be used.
577                     num_words : int
578                             Number of words to be selected.
579                     min_length : int
580                             Minimun length of a word.
581                     max_length: in
582                             Maximun length of a word.
583
584                     '''
585                     self.subject_name = subject.get_name()
586                     words = list(subject.get_words())
587                     # Functions from Python Random Lib.
588                     shuffle(words)
589                     if len(words) < num_words:
590                             lim = len(words)
591                     elif num_words <= 0:
592                             lim = 10
593                     else:
594                             lim = num_words
595
596                     i = 0
597                     while lim > 0 and i < len(words):
598                             if len(words[i]) <= max_length and len(words[i]) >= min_length:
599                                     self.add_word_to_not_found(words[i].upper())
600                                     lim -= 1
601                             i += 1
```

```
                    # Inconsistency
                    if lim > 0:
                              print(messages["error"])
                              exit()

          def word_already_found(self,word):
                    """ Checks if a word is on the list of words already found.

                    Parameters
                    ----------
                    word : str
                              Word to check.

                    Returns
                    -------
                    bool
                              True for success, False otherwise.

                    """
                    return (word in self.words_found)

          def word_not_found(self,word):
                    """ Cheks if a word is on the list of words that have not been found.

                    Parameters
                    ----------
                    word : str
                              word to be checked.

                    Returns
                    -------
                    bool
                              True for success, False otherwise.

                    """
                    return (word in self.words_not_found)

          def word_in_clue(self, word):
                    """ Cheks if a word belongs to a clue. Checks if a word is included on
                              the list of words already found or on the list of words that have
                              not been found.

                    Parameters
                    ----------
                    word : str
                              word to be checked.

                    Returns
                    -------
                    bool
                              True for success, False otherwise.

                    """
                    return (self.word_already_found(word) or self.word_not_found(word))

          def add_word_to_not_found(self, word):
                    ''' Add a word to the clue as not found.

                    Parameters
                    ----------
                    word : string
                              Word that will be added.

                    '''
                    if word == "" or word == None:
                              print(messages["incorret_format"])
                    else:
                              self.words_not_found.append(word)

          def add_word_to_found(self, word):
                    """ Add a word to the list of words that have been found.

                    Parameters
                    ----------
                              word (str): word to be added.

                    """
                    self.words_found.append(word)

          def remove_word_from_not_found(self, word):
                    """ Check if the word is on the list of words that have not been found
                              and removed it.

                    Parameters
                    -----------
                    word : str
                              Word to be removed.

                    Returns
                    -------
                    bool
                              True if the word was successfully removed. False otherwise.

                    """
                    if (self.word_not_found(word)):
                              self.words_not_found.remove(word)
                              return True
                    return False

          def found_all_the_words(self):
                    """ Check if the list of words that have not been found is empty and
                              the list of words that have been found is not empty.

                    Returns
```

```
                        -------
                        bool
                                True for success, False otherwise.

                        """
                        return ((len(self.words_not_found) == 0) and\
                                        (len(self.words_found) != 0))

        def get_words_not_found(self):
                        ''' Return the list of words not found.

                        Returns
                        -------
                        [str]
                                List of words that has not been found by the player.

                        '''
                        return self.words_not_found
class BySolution(Clue):
        ''' Subclass that displays all the words in the Clue.

        '''

        def display (self):
                        ''' Display on the standard output the name of the subject and the
                                complete`list of words that the player needs to find.

                        '''
                        print(self.subject_name + ": ")
                        print(self.words_not_found)

class Game:
        ''' Superclass. Manage information of a game on its different modes.

        '''

        def __init__(self):
                        self.board = None
                        self.clue = None
                        self.start_again = False

        def add_board(self, board):
                        ''' Associate a built board to the game.

                        Parameters
                        ----------
                        board : Board
                                Board to be added to the game.

                        '''
                        self.board = board


        def select_subject(self):
                        ''' Return a selected subject from the dictionary.

                        Returns
                        -------
                        Subject
                                Subject selected by the player.

                        '''
                        sname = ""
                        subject = None
                        while subject == None:
                                print(messages["choose_subject"])
                                dictionary.display_subjects()
                                sname = input(messages["get_subject"])
                                subject = dictionary.get_subject_by_name(sname)
                                if subject == None:
                                        print(messages["separator"])
                                        print(messages["invalid_subject"])
                        return subject

        def select_clue(self, subject, num_words, min_length, max_length):
                        ''' Select a specific type of clue. For this version, it will be
                                BySolution.

                        Parameters
                        ----------
                        subject : Subject
                                subject that will be used into the building clue process.
                        num_words : int
                                Number of words to be selected
                        min_length : int
                                Minimun length of a word.
                        max_length: int
                                Maximun length of a word.

                        '''
                        clue = BySolution()
                        if subject == None:
                                print(messages["incorret_format_clue"])
                        else:
                                clue.build(subject, num_words,min_length, max_length)
                                self.clue = clue


        def get_row_number(self):
                        """ Read from standard input an integer.

                        Returns
                        -------
```

```
                                int : row number read.

                """
                is_valid = False
                while not is_valid:
                        try:
                                n = int(input(messages["insert_row_number"]))
                                if self.board.is_valid_row(n):
                                        return n
                                print(messages["invalid_row"])
                        except:
                                print(messages["it_is_not_an_integer"])

        def get_column_number(self):
                """ Read from standard input an integer.

                Returns
                -------
                        int : column number read.

                """
                is_valid = False
                while not is_valid:
                        try:
                                n = int(input(messages["insert_column_number"]))
                                if self.board.is_valid_column(n):
                                        return n
                                print(messages["invalid_column"])
                        except:
                                print(messages["it_is_not_an_integer"])

        def find_word(self):
                """ Find a word specified by the user. Get the row and the column
                        numbers corresponding to initial and the final cells of the word.
                        Get the word from the board, and check if the word is valid.
                        Display appropiate messages to the user.

                """
                print (messages["initial_cell"])
                row1 = self.get_row_number()
                column1 = self.get_column_number()
                print (messages["final_cell"])
                row2 = self.get_row_number()
                column2 = self.get_column_number()
                word = self.board.get_word(row1,column1,row2,column2)
                if word != None:
                        if not self.clue.word_in_clue(word):
                                print("\n'" + word + "'"+ messages["incorret_word"])
                                return
                        if self.clue.word_already_found(word):
                                print("'" + word + "'" + messages["already_found_word"])
                        else:
                                self.clue.add_word_to_found(word)
                                self.clue.remove_word_from_not_found(word)
                                print(messages["good_word"] + "'" + word + "'.")
                else:
                        print(messages["it_is_not_a_line"])

        def setup(self):
                ''' Setup all the configuration of the game before starting.'''

                new_board = Board()
                selected_subject = self.select_subject()
                self.select_clue(selected_subject,
                                                        new_board.get_max_num_words(),
                                                        new_board.get_min_length_word(),
                                                        new_board.get_max_length_word())
                words = list(self.clue.get_words_not_found())
                new_board.build(words)
                self.add_board(new_board)

        def get_direction(self):
                """ Read from standard input a string corresponding to a direction.
                        Only 'right', 'left', 'up', 'down' are possible.

                Returns
                -------
                str
                        direction read.

                """

                is_valid = False
                while not is_valid:
                        direction = (input(messages["insert_direction"])).lower()
                        if direction in directions:
                                return direction
                        print(messages["invalid_direction"])

        def get_number_spaces(self,direction):
                """ Read from standard input an integer.

                Parameters
                ----------
                direction : str
                        Direction the board is going to rotate to.

                Returns
                -------
                int
                        Number of spaces read.

                """
                is_valid = False
```

```
                        while not is_valid:
                                try:
                                        n = int(input(messages["insert_number_spaces"]))
                                        if self.board.is_valid_number_spaces(direction,n):
                                                return n
                                        print(messages["invalid_number_spaces"])
                                except:
                                        print(messages["it_is_not_an_integer"])

        def get_rotation_attr(self):
                """ Get the direction to rotate the board, the number of row or column
                that is going to rotate and the amount of spaces to rotate. Then
                rotates the board accordint to these parameters.

                """
                direction = self.get_direction()
                if direction == "up" or direction == "down":
                        column = self.get_column_number()
                        number = self.get_number_spaces(direction)
                        self.board.rotate_vertically(column,number,direction)
                elif direction == "left" or direction == "right":
                        row = self.get_row_number()
                        number = self.get_number_spaces(direction)
                        self.board.rotate_horizontally(row,number,direction)

        def get_play_again(self):
                ''' Ask the user if he-she wants to start a new game.

                Returns
                -------
                bool
                        True if success, False otherwise.

                '''
                while True:
                        answer = (input(messages["play_again"])).lower()
                        if answer in bool_answers:
                                if answer == "yes":
                                        return True
                                elif answer == "no":
                                        return False
                        print(messages["invalid_play_again_option"])

        def end_current_game(self):
                ''' If the player wants to end the game, sets the variable start_again
                        to True. Otherwise exit the game.

                '''
                if self.get_play_again():
                        self.start_again = True
                else:
                        exit_game()

        def read_command(self):
                ''' Reads a command from standard input. Only 'rotate', 'word', 'help',
                        'exit' are valid. Calls the corresponding function to execute each
                        command.

                '''
                is_valid = False
                while not is_valid:
                        command = (input(messages["insert_command"])).lower()
                        if command in commands:
                                if command == "help":
                                        inst.display()
                                elif command == "exit":
                                        self.end_current_game()
                                elif command == "rotate":
                                        self.get_rotation_attr()
                                elif command == "find":
                                        self.find_word()
                                return None
                        else:
                                print(messages["invalid_command"])

        def turn(self):
                ''' Manages the turns of the player. Call the function to read the
                        commands, and display the current state of the game.

                '''
                print (messages["separator"])
                self.board.display()
                self.clue.display()
                self.read_command()

        def play_again(self):
                ''' Return if the player wants to play again.

                Returns
                -------
                bool
                        True if the player wants to play again. False otherwise.

                '''
                return self.start_again


class PracticeMode(Game):
        ''' Single player mode of game implemented on version 1.
                Subclass managing specific behaviours of the game.

        '''
```

```
            def is_win (self):
                        ''' Check the winning condition.

                        Returns
                        -------
                        bool
                                    True if success, False otherwhise.

                        '''
                        return (self.clue.found_all_the_words())

            def play(self):
                        '''
                                    Manage the gameplay. Handle the turns.
                        '''
                        self.start_again = False
                        while not self.start_again:
                                    self.turn()
                                    if self.is_win():
                                                print(messages["win"])
                                                self.end_current_game()

# Dictionary of words.
dictionary = Dictionary()

# Class game.
game = PracticeMode()

# Instructions of the game
inst = Instruction()

def configure_instructions():
            ''' Import the instructions into the program.

            '''

            inst.import_instruction("./rules/instructions.txt")

def display_initial_message():
            ''' Display on the standard output a welcome message.

            '''
            print (messages["welcome"])

def display_menu():
            ''' Display on the standard output a welcome message.

            '''
            print (messages["menu"])

def start_game():
            ''' Start the game. '''

            os.system('clear')
            display_initial_message()
            input(messages["enter_to_continue"])
            os.system('clear')

            option = ''
            while(option != 'p'):
                        display_menu()
                        option = input(messages["insert_menu_action"])
                        option = option.lower()
                        os.system('clear')
                        if option == 'h':
                                    inst.display()
                                    print(messages["separator"])
                        elif option == 'e':
                                    messages["exit_game"]
                                    exit()
                        elif option != 'p':
                                    print(messages["invalid_menu_option"])

            print(messages["setting_up"])
            time.sleep(2)
            os.system('clear')
            game.setup()
            os.system('clear')
            print(messages["starting"])
            time.sleep(2)
            os.system('clear')
            game.play()

def exit_game():
            ''' Display an exit message and exit the program.

            '''
            print (messages["exit_game"])
            exit()

def main():
            ''' Main function of the program

            '''

            configure_instructions()
            correct = dictionary.load()
            if not correct:
                        print(messages["error"])
                        exit()
            play_again = True
            while play_again:
                        start_game()
                        play_again = game.play_again()
```

```
997
998 if __name__ == '__main__':
999   main()
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
```