# CSSE1001/7030

**Semester 2, 2016**
**Assignment 1**
**Version 1.0.0**
**10 marks**

**Due Friday 26 August, 2016, 21:30**

# 1 Introduction

Ash has lost Pikachu! This assignment is centred on helping Ash find Pikachu in a maze, while avoiding other pesky pokémon. You will write a simple text-based interactive program that loads a maze from a text file and provides commands for moving about the maze.

The files `maze1.txt`, `maze2.txt`, `maze3.txt and maze4.txt` are text files containing representations of simple mazes. The contents of `maze1.txt` is given below.

```
#####
#  Z#
# ###
#  P#
#####
```

`#` represents a wall square. Spaces represent open squares. The single `P`, for Pikachu, represents the goal square. `D` and `Z`, for Doduo and Zubat respectively, represent the anti-goal squares (these squares must be avoided).

The start square is always in row 1, column 1 (counting from 0), which will always contain a space. Each maze file also contains a single goal square (Pikachu) and there should be at least one path from the start square to the goal square that uses only open squares. The first and last rows and columns are all `#`'s. A maze file is **valid** if and only if it meets all of these criteria.

The player can move north (up), south (down), east (right), or west (left).

Positions of squares are represented as row, column pairs. A square is **legal** if and only if the square does not contain a wall (`#`). A move from one square to an adjacent square is legal if and only if the adjacent square is legal. Similarly, a direction is legal if and only if a move in that direction is legal.

## 1.1 Examples

Here are three examples of what is expected from your program. **You should not begin implementing this top-level functionality until you reach section 2.2.6.** The input is everything after `Command:` and `Maze File:` on a line. Everything else is output. Your output should be **exactly the same** as below for the given input (including capitals, spaces, empty lines, etc.).

```
Maze File: maze1.txt

#####
```

```
#A Z#
# ###
#  P#
#####

Command: a
Invalid command: a

#####
#A Z#
# ###
#  P#
#####

Command: b
You cannot go back from the beginning.

#####
#A Z#
# ###
#  P#
#####

Command: ?
? - Help.
n - Move North one square.
s - Move South one square.
e - Move East one square.
w - Move West one square.
r - Reset to the beginning.
b - Back up a move.
p - List all legal directions from the current position.
q - Quit.

#####
#A Z#
# ###
#  P#
#####

Command: e

#####
# AZ#
# ###
#  P#
#####
```

```
Command: e
Oh no! A wild Zubat appeared - you lose :(
```

Another example.

```
Maze File: maze1.txt

#####
#A Z#
# ###
#  P#
#####

Command: s

#####
#  Z#
#A###
#  P#
#####

Command: w
You can't go in that direction.

#####
#  Z#
#A###
#  P#
#####

Command: s

#####
#  Z#
# ###
#A P#
#####

Command: b

#####
#  Z#
#A###
#  P#
#####

Command: s
```

```
#####
#  Z#
# ###
#A P#
#####

Command: r

#####
#A Z#
# ###
#  P#
#####

Command: s

#####
#  Z#
#A###
#  P#
#####

Command: s

#####
#  Z#
# ###
#A P#
#####

Command: e

#####
#  Z#
# ###
# AP#
#####

Command: q
Are you sure you want to quit? [y] or n: n

#####
#  Z#
# ###
# AP#
#####

Command: p
```

```
Possible directions: e, w

#####
#  Z#
# ###
# AP#
#####

Command: e
Congratulations - you found Pikachu!
```

A final example.

```
Maze File: maze1.txt
#####
#A Z#
# ###
#  P#
#####

Command: q
Are you sure you want to quit? [y] or n: asdasd
```

# 2 Assignment Tasks

## 2.1 Download files

The first task is to download `a1.py` and `a1_files.zip`.

The file `a1.py` is for your assignment. **Do not modify the assignment file outside the area provided for you to write your code**. When you have completed your assignment you will submit the file `a1.py` containing your solution to the assignment (see section 4 for submission details).

## 2.2 Write the code

There are several functions you need to write and these are described below. It is highly recommended that you review the support file, `a1_support.py`, before writing your code, as this contains many useful constants and functions. **Do not use global variables in your code.** You are not expected to deal with exceptions for this assignment and so your assignment will not be tested with incorrectly formatted maze files or invalid mazes.

A maze will be represented in Python by a single string of row strings separated by the newline character. Each row will be represented by a string of characters for each square in the row, from left to right. For example, the maze contained in `maze1.txt` would be represented as follows:

```
>>> maze = load_maze('maze1.txt')
```

```
>>> maze
'#####\n#  Z#\n# ###\n#  P#\n#####'
```

### 2.2.1 Commenting - 1 mark

Each function that you write must have a suitable docstring comment, as specified in the course notes. See http://csse1001.uqcloud.net/notes/commenting

### 2.2.2 get_position_in_direction - 1 mark

`get_position_in_direction(position, direction)` takes a row, column pair representing a position, and a direction character (one of `n, s, e, w`), and returns the position of the adjacent square in the given direction. It does not matter whether or not the direction is legal.

```
>>> get_position_in_direction((2, 3), 'e')
(2, 4)
>>> get_position_in_direction((2, 3), 's')
(3, 3)
>>>
```

### 2.2.3 print_maze - 1 mark

`print_maze(maze, position)` takes a maze string and the position of the player, and prints the maze with the player shown as an `'A'`.

```
>>> maze = load_maze('maze1.txt')
>>> print_maze(maze, (1, 1))
#####
#A Z#
# ###
#  P#
#####
```

### 2.2.4 move - 2 marks

`move(maze, position, direction)` takes a maze string, a position of a square and a direction and returns a pair of the form `(position, square)` where `position` is the position after the move and `square` is the resulting square after the move. When the move is invalid, the new position returned is the same as the old position.

```
>>> maze = load_maze('maze1.txt')
>>> move(maze, (1, 1), 's')
((2, 1), ' ')
>>> move(maze, (1, 1), 'n')
((1, 1), '#')
>>> move(maze, (3, 2), 'e')
```

```
((3, 3), 'P')
>>> move(maze, (1, 2), 'e')
((1, 3), 'Z')
```

### 2.2.5 get_legal_directions - 1 mark

`get_legal_directions(maze, position)` takes a maze string and a position, and returns a list of legal directions for that square. You may assume the given position is legal.

```
>>> maze = load_maze('maze1.txt')
>>> get_legal_directions(maze, (1, 1))
['s', 'e']
>>> get_legal_directions(maze, (1, 3))
['w']
>>> get_legal_directions(maze, (3, 1))
['n', 'e']
```

### 2.2.6 interact - 4 marks

`interact()` is the top-level function that defines the text-based user interface as described in the introduction (it takes no arguments and returns `None`). Your program must work according to the example output in section 1.1.

The program should perform the following actions, in order:

1. Prompt the user for a maze file. You can assume the file supplied is in the correct format and that it contains a valid maze.
2. Repeatedly prompt the user for a command, and immediately process that command. This should be repeated until the game ends or the user quits. Commands are listed below.

**Commands**

**?** is the help command. It lists all the commands.

**n, s, e, w** each attempt to move the player one square in the direction given (north for n, etc.). These commands will output an error message if that direction is illegal. If the user moves to a square with a pokémon (P, D, or Z), then the game ends with a suitable message.

**r** resets the player back to the beginning (the player cannot go back from this point).

**b** backs up one move. If the command is used, for example, 3 times in a row then the last 3 moves will be undone. If this command is used before any valid moves have been made, then this command outputs an error message (this includes after a reset).

**p** lists all the possible legal directions from the current position. They must appear sorted according to this order: `n, s, e, w`.

**q** is used to quit before finding Pikachu. If the command is used, the user should be asked to confirm. If the user types `n`, the game will continue; otherwise it will immediately quit. **Do not use `quit()` or `exit()`.**

Any other command is treated as invalid and an error message is printed. Whitespace at the beginning and end of a command should be ignored (i.e. `'  q  '` should be considered the same as `'q'`).

### 2.2.7 Hints

Handling whitespace: `strip`

User interaction: `input`

Converting maze positions to indices: `a1_support.py`

To deal with backtracking, consider using a list to record positions - the new position is appended and upon backtracking the last position is popped. The last state in the list is the "current position".

# 3 Assessment and Marking Criteria

In addition to providing a working solution to the assignment problem, the assessment will involve discussing your code submission with a tutor. This discussion will take place in the practical session you have signed up to in week 6. You **must** attend that session in order to obtain marks for the assignment.

In preparation for your discussion with a tutor you may wish to consider:

- any parts of the assignment that you found particularly difficult, and how you overcame them to arrive at a solution;
- whether you considered any alternative ways of implementing a given function;
- where you have known errors in your code, their cause and possible solutions (if known).

It is also important that you can explain to the tutor how each of the functions that you have written operates (for example, if you have used a for loop or a while loop in a function, why this was the right choice).

Marks will be awarded based on a combination of the correctness of your code and on your understanding of the code that you have written. **A technically correct solution will not elicit a pass mark unless you can demonstrate that you understand its operation.**

A partial solution will be marked. If your partial solution causes problems in the Python interpreter please comment out that code and we will mark that.

Please read the section in the course profile about plagiarism.

# 4 Assignment Submission

You must submit your completed assignment electronically through Blackboard.

For information on submitting through Blackboard, please read
`http://www.library.uq.edu.au/ask-it/blackboard-assessment`

You should electronically submit your copy of the file `a1.py` (use this name - all lower case).

You may submit your assignment multiple times before the deadline - only the last submission will be marked.

Late submission of the assignment will not be accepted. In the event of exceptional personal or medical circumstances that prevent you from handing in the assignment on-time, you may submit a request for an extension.

Requests for extensions should be made as soon as possible, and preferably before the assignment due date. The request should be made via the `Help` system on the course Blackboard page.

All requests for extension must be submitted on the UQ Application for Extension of Progressive Assessment form:
`http://www.uq.edu.au/myadvisor/forms/exams/progressive-assessment-extension.pdf`
no later than 48 hours prior to the submission deadline. The application and supporting documentation (e.g. medical certificate) must be submitted to the ITEE Coursework Studies office (78-425) or by email to `enquiries@itee.uq.edu.au`. If submitted electronically, you must retain the original documentation for a minimum period of six months to provide as verification should you be requested to do so.

# Changelog

Any changes to this document will be listed here.