# Sliding Puzzle Game

Assignment 1
Semester 1, 2021
CSSE1001/CSSE7030

Due date: 20:00 (AEST), 26 March, 2021

## 1 Introduction

In this assignment, you will implement a text-based version of the popular sliding puzzle game.[1] It is recommended that you try an online version of the game to familiarise yourself with the concepts and gameplay.[2]

## 2 Getting Started

To start, download `a1.zip` from Blackboard and extract the contents. The `a1.zip` archive contains all the necessary files to start this assignment. Some support code has been included to assist with implementing the program.

Within the `a1.zip` archive you will find the following:

- `a1.py`: You are required to implement your assignment solution entirely within this file. It includes some initial code to help you get started on the assignment. This is the only file that you will submit for this assignment, **do not** make changes to any other files.
- `a1_support.py`: This file contains code that you must use to assist you in implementing your assignment. The content of this file is described in Section 6.1.
- `words.txt`: This file contains a number of common English words.[3] The file is used to generate new sliding puzzles, however, you are **not** to generate puzzles from this file yourself, instead, you will use functions available in `a1_support.py` to read the file contents.

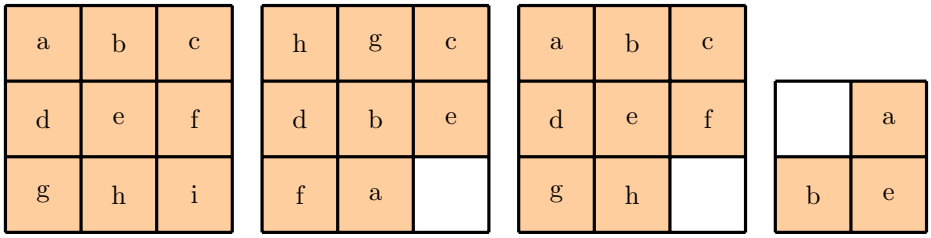
## 3 Terminology

In the context of this assignment:

- *Sliding puzzle* is a puzzle consisting of a square-shaped grid.
- The *grid* is evenly divided into tiles, with the same number of tiles in each row and column.
- *Tile* is a single cell in the grid containing a single letter, or is a *blank* tile.
- *Blank tile* represents an empty space in the physical puzzle on which this assignment is based.
- *Puzzle* is a grid in which every tile contains a single letter, except for a single tile that is *blank*.
- *Solution* is a grid which is filled so that every tile contains a single letter.
- The player's *goal* is to solve the puzzle by sliding the non-blank tiles so that the grid corresponds to the solution, with the blank tile at the bottom right corner.

[1] https://en.wikipedia.org/wiki/Sliding_puzzle
[2] https://www.helpfulgames.com/subjects/brain-training/sliding-puzzle.html
[3] https://github.com/first20hours/google-10000-english

An example of the sliding puzzle games is shown in the image below.




# 4 Game Stages

## 4.1 Setup

At the beginning of the game, the player is asked to choose a *difficulty*. Difficulty is an integer which is the number of rows and columns of the puzzle. This determines the size of words used in the puzzle, where each word fits into one row. You may assume the difficulty (size of the puzzle) given by the player is an integer between two and fourteen.

After the difficulty is specified, a solution is generated from the words provided in a word file. A shuffled puzzle is then generated by shuffling the tiles in the solution and replacing the tile at the bottom right corner with an empty tile.

## 4.2 Solving the Puzzle

The player tries to solve the puzzle by sliding the tiles to reach their goal. At each turn, the solution and current state of the game are displayed. The player is then prompted to enter an action. The action can be one of the following:

| **Input** | **Description** |
|---|---|
| `"H"` | Display a help message. |
| `"GU"` | Stop playing the current game. |
| `"U"` or `"D"` or `"L"` or `"R"` | Move the empty tile in the up/down/left/right direction, respectively. |

The prompt will repeat until the player either wins the game, or gives up by entering the `"GU"` action.

## 4.3 End of Game

At the end of a game, the player is prompted to choose if they want to start a new game or to stop the program.

| **Input** | **Description** |
|---|---|
| `"Y"` or `"y"` or `""` | Start a new game. |
| Anything else | Close the program |

If they choose to start a new game, they need to set a new difficulty. A new solution and puzzle are generated, and the new game is started.

# 5 Conventions

This assignment follows the conventions specified below:

1. `solution`: A solution is stored in a string. This string contains all letters in the grid from left to right and then top to bottom, without any special characters and/or new lines. For example, the `solution`, `"dogcatpig"`, represents the grid below:

| | | |
|---|---|---|
| d | o | g |
| c | a | t |
| p | i | g |

2. `puzzle`: The puzzle is a string that is a copy of the solution, except for the last character. All the characters in the puzzle are shuffled at the start of the game. A space character (i.e. `" "`) is added to the end of the puzzle to represent the blank tile. For example, after several moves the `puzzle`, `"fhg bcade"`, is represented by the grid below:

| | | |
|---|---|---|
| f | h | g |
| | b | c |
| a | d | e |

   To implement the logic of sliding a tile, you need to swap the position of the character in the string that is being slid with the position of the space character. Your logic needs to ensure that you only perform the swap if the character and the space are adjacent to each other in the grid.

3. `direction`: A direction can either be `"U"` or `"D"` or `"L"` or `"R"`, which correspond to up/down/left/right.

# 6 Implementation

The following functions **must** be implemented in `a1.py`. You may implement additional functions, if you think they will help with your logic or make your code easier to understand. These functions have been listed in order of increasing difficulty.

`check_win(puzzle: str, solution: str) -> bool`
Returns `True` if the game is won, given the puzzle and the solution, and `False` otherwise. Examples of calling `check_win`:

```
>>> check_win("abcdefgh ", "abcdefghi")
True
>>> check_win("dabecghf ", "abcdefghi")
False
```

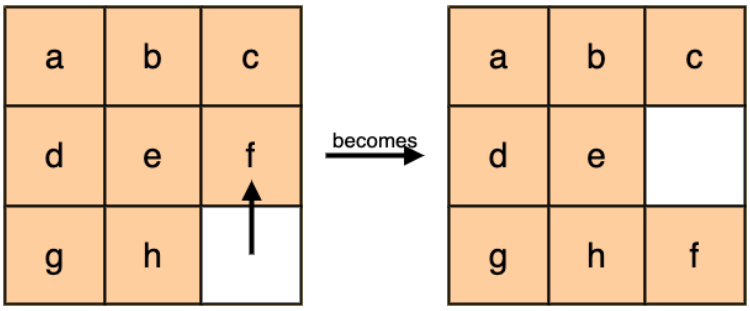
`swap_position(puzzle: str, from_index: int, to_index: int) -> str`
Swaps the positions of the characters at `from_index` and `to_index` in the puzzle and returns the updated puzzle. Examples of calling `swap_position`:

```
>>> swap_position("care", 0, 2)
'race'
>>> swap_position("does", 3, 2)
'dose'
```

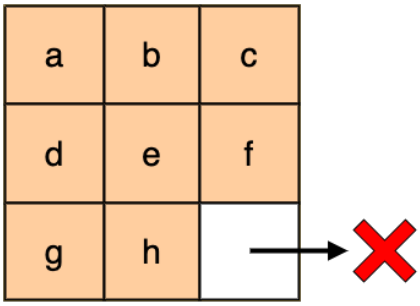
`move(puzzle: str, direction: str) -> Optional[str]`
Moves the empty tile in the given `direction` and returns the updated puzzle. If the move cannot be made (i.e. moving in the given direction results in the empty tile being outside the grid), it returns `None`. Examples of calling `move`:

```
>>> move("abcdefgh ", "U")
'abcde ghf'
>>> move("abcdefgh ", "R")
>>>
```

The first line of code in the examples above moves the empty tile from the bottom corner up by one tile. It corresponds to the following graphical demonstration.




The second line of code returns nothing. This is because the empty tile is in the bottom right corner and cannot be moved to the right.




`print_grid(puzzle: str) -> None`
Displays the puzzle in a user-friendly format. Examples of calling `print_grid`:

```
>>> print_grid("nevagonagiveu up")
+---+---+---+---+
| n | e | v | a |
+---+---+---+---+
| g | o | n | a |
+---+---+---+---+
| g | i | v | e |
+---+---+---+---+
| u |   | u | p |
+---+---+---+---+
>>> print_grid("nevergonnalet udooooowwwn")
+---+---+---+---+---+
| n | e | v | e | r |
+---+---+---+---+---+
| g | o | n | n | a |
+---+---+---+---+---+
| l | e | t |   | u |
+---+---+---+---+---+
| d | o | o | o | o |
+---+---+---+---+---+
| o | w | w | w | n |
+---+---+---+---+---+
```

`main() -> None`
Handles the main user interaction in the game, as specified in Section 4.2. You need to replace the `print` statement with the main logic of your program.

### 6.1 Support Code

The file `a1.py` includes an implementation of the `shuffle_puzzle` function. You should **not** modify this function. You must call this function **once and only once** per game to create a shuffled puzzle.

`shuffle_puzzle(solution: str) -> str`
Generates a solvable sliding game given the solution.
This function calls `swap_position`, which is specified above. You **must** implement `swap_position` before you can successfully call `shuffle_puzzle`.

You **must** use the code provided in `a1_support.py` to implement your assignment. Do **not** make changes to `a1_support.py` as it could cause unexpected errors. In this file you will find some predefined named constants and functions. You are expected to use the provided named constants in your implementation of the game. Of the three functions, you only need to call:

`get_game_solution(file_name: str, grid_size: int) -> str`
This function returns a string, which is the solution to the puzzle. The first parameter is the name of the file from which words are to be loaded. The second parameter is the size of the grid.

## 7 Example Gameplay

```
Welcome to the big brain sliding puzzle game.
This game is based on the real life sliding puzzle: https://en.wikipedia.org/wiki/Sliding_puzzle
Try the game online here: https://www.helpfulgames.com/subjects/brain-training/sliding-puzzle.html


Choose your difficulty. The larger the puzzle, the harder it is to solve.
How big do you want the puzzle to be? 2
Solution:
+---+---+
| a | c |
+---+---+
| f | c |
+---+---+

Current position:
+---+---+
| c | f |
+---+---+
| a |   |
+---+---+

Please input a direction (enter "H" for instructions): R
'R' is not a possible move here. Please try again.

Solution:
+---+---+
| a | c |
+---+---+
| f | c |
+---+---+

Current position:
+---+---+
| c | f |
+---+---+
| a |   |
+---+---+

Please input a direction (enter "H" for instructions): C
That's not a valid input, try again.
```

```
Solution:
+---+---+
| a | c |
+---+---+
| f | c |
+---+---+

Current position:
+---+---+
| c | f |
+---+---+
| a |   |
+---+---+

Please input a direction (enter "H" for instructions): H

Options:
    H:          Display this help message
    GU:         Stop the current game
    [U|D|L|R]:  Move the empty cell in the up/down/left/right direction,
                respectively.


Solution:
+---+---+
| a | c |
+---+---+
| f | c |
+---+---+

Current position:
+---+---+
| c | f |
+---+---+
| a |   |
+---+---+

Please input a direction (enter "H" for instructions): U
Solution:
+---+---+
| a | c |
+---+---+
| f | c |
+---+---+

Current position:
+---+---+
| c |   |
+---+---+
| a | f |
+---+---+
```

```
Please input a direction (enter "H" for instructions): L
Solution:
+---+---+
| a | c |
+---+---+
| f | c |
+---+---+

Current position:
+---+---+
|   | c |
+---+---+
| a | f |
+---+---+

Please input a direction (enter "H" for instructions): D
Solution:
+---+---+
| a | c |
+---+---+
| f | c |
+---+---+

Current position:
+---+---+
| a | c |
+---+---+
|   | f |
+---+---+

Please input a direction (enter "H" for instructions): R
Solution:
+---+---+
| a | c |
+---+---+
| f | c |
+---+---+

Current position:
+---+---+
| a | c |
+---+---+
| f |   |
+---+---+

Congratulations, you've won the game. Here's your candy:
               ___         .-""-.       ___
               \  "-.  /       \  .-"  /
                > -=.\/         \/.=- <
                  > -='\        /\'=- <
               /__.-'  \       /  '-.__\
                        '-..-'


Do you want to play again? [Y/n] y
```

```
Choose your difficulty. The larger the puzzle, the harder it is to solve.
How big do you want the puzzle to be? 3
Solution:
+---+---+---+
| r | o | d |
+---+---+---+
| s | e | o |
+---+---+---+
| p | a | r |
+---+---+---+

Current position:
+---+---+---+
| a | r | s |
+---+---+---+
| p | o | d |
+---+---+---+
| o | e |   |
+---+---+---+

Please input a direction (enter "H" for instructions): U
Solution:
+---+---+---+
| r | o | d |
+---+---+---+
| s | e | o |
+---+---+---+
| p | a | r |
+---+---+---+

Current position:
+---+---+---+
| a | r | s |
+---+---+---+
| p | o |   |
+---+---+---+
| o | e | d |
+---+---+---+

Please input a direction (enter "H" for instructions): D
Solution:
+---+---+---+
| r | o | d |
+---+---+---+
| s | e | o |
+---+---+---+
| p | a | r |
+---+---+---+

Current position:
+---+---+---+
| a | r | s |
+---+---+---+
| p | o | d |
+---+---+---+
| o | e |   |
+---+---+---+

Please input a direction (enter "H" for instructions): GU
```

```
Aww, too bad. Better luck next time.

Do you want to play again? [Y/n] n
Bye.
```

# 8 Assessment and Marking Criteria

This assignment assesses course learning objectives:

1. apply program constructs such as variables, selection, iteration and sub-routines,
3. read and analyse code written by others,
5. read and analyse a design and be able to translate the design into a working program, and
6. apply techniques for testing and debugging.

## 8.1 Functionality

Your program's functionality will be marked out of a total of 6 marks. Your assignment will be put through a series of tests and your functionality mark will be proportional to the number of tests you pass. If, say, there are 50 functionality tests and you pass 40 of them, then your functionality mark will be $\frac{40}{50} \times 6$. You will be given a *subset* of the functionality tests before the due date for the assignment.

You need to perform your **own** testing of your program to make sure that it meets *all* specifications given in the assignment. Only relying on the provided tests is likely to result in your program failing in some cases and you losing some functionality marks. Note: Functionality tests are automated, so string outputs need to match **exactly** what is expected.

Your program must run in the Python interpreter (the IDLE environment). Partial solutions will be marked but if there are errors in your code that cause the interpreter to fail to execute your program, you will get zero for functionality marks. If there is a part of your code that causes the interpreter to fail, comment out the code so that the remainder can run. Your program must run using the Python 3.9 interpreter. If it runs in another environment (e.g. Python 3.8 or PyCharm) but not in the Python 3.9 interpreter, you will get zero for the functionality mark.

## 8.2 Code Style

The style of your assignment will be assessed by a tutor. Style will be marked according to the style rubric provided with the assignment. The style mark will be out of 4.

The key consideration in marking your code style is whether the code is easy to understand. There are several aspects of code style that contribute to how easy it is to understand code. In this assignment, your code style will be assessed against the following criteria.

- Readability
  - Program Structure: Layout of code makes it easier to read and follow its logic. This includes using whitespace to highlight blocks of logic.
  - Descriptive Identifier Names: Variable, constant, and function names clearly describe what they represent in the program's logic. Do **not** use what is called the *Hungarian Notation* for identifiers. In short, this means do not include the identifier's type in its name (e.g. `employee_number`), rather make the name meaningful (e.g. `employee_identifier`). The main reason for this restriction is that most people who follow the *Hungarian Notation* convention, use it poorly (including Microsoft).
  - Named Constants: Any non-trivial fixed value (literal constant) in the code is represented by a descriptive named constant (identifier).
- Algorithmic Logic
  - Single Instance of Logic: Blocks of code should not be duplicated in your program. Any code that needs to be used multiple times should be implemented as a function.

  - Variable Scope: Variables should be declared locally in the function in which they are needed. Global variables should not be used.

  - Control Structures: Logic is structured simply and clearly through good use of control structures (e.g. loops and conditional statements).

- Documentation

  - Comment Clarity: Comments provide meaningful descriptions of the code. They should not repeat what is already obvious by reading the code (e.g. `# Setting variable to 0.`). Comments should not be verbose or excessive, as this can make it difficult to follow the code.

  - Informative Docstrings: Every function should have a docstring that summarises its purpose. This includes describing parameters and return values so that others can understand how to use the function correctly.

  - Description of Logic: All significant blocks of code should have a comment to explain how the logic works. For a small function, this would usually be the docstring. For long or complex functions, there may be different blocks of code in the function. Each of these should have an in-line comment describing the logic.

## 8.3 Assignment Submission

You must submit your assignment electronically via Gradescope (https://gradescope.com/). You will need to sign up using your UQ email address which is based on your student number (e.g. s4123456@student.uq.edu.au). When you sign up to Gradescope, you will need to provide a course entry code. The course entry code for CSSE1001 and CSSE7030 is **6PKYV5**. You will be able to submit after March 5 but you may sign up before then.

When you login to Gradescope you may be presented with a list of courses. Select CSSE1001/CSSE7030. You will see a list of assignments. Choose **Assignment 1**. You will be prompted to choose a file to upload. The prompt may say that you can upload any files, including zip files. You must submit your assignment as a single Python file called `a1.py` (use this name – all lower case), and *nothing* else. Your submission will be automatically run to determine the functionality mark. If you submit a file with a **different name**, the tests will **fail** and you will get **zero** for functionality. Do **not** submit the `a1_support.py` or `words.txt` files. Do **not** submit **any** sort of archive file (e.g. zip, rar, 7z, etc.).

Upload an initial version of your assignment *at least* a week before the due date. Do this even if it is just the initial code provided with the assignment. Ensure that you are able to login to Gradescope and upload files for the assignment. If you are unable access to the course on Gradescope, contact course staff **immediately** to ensure that your email address is registered. Excuses, such as you were not able to login or were unable to upload a file will not be accepted as reasons for granting an extension.

When you upload your assignment it will run a **subset** of the functionality autograder tests on your submission. It will show you the results of these tests. It is your responsibility to ensure that your uploaded assignment file runs and that it passes the tests you expect it to pass.

Late submissions of the assignment will **not** be marked. Do not wait until the last minute to submit your assignment, as the time to upload it may make it late. Multiple submissions are allowed, so ensure that you have submitted an almost complete version of the assignment *well* before the submission deadline of 20:00. Your latest, on time, submission will be marked. Ensure that you submit the correct version of your assignment.

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. See the course profile for details of how to apply for an extension.

Requests for extensions must be made **before** the submission deadline. The application and supporting documentation (e.g. medical certificate) must be submitted via my.UQ. You must retain the original documentation for a minimum period of six months to provide as verification, should you be requested to do so.

## 8.4 Plagiarism

This assignment must be your own individual work. By submitting the assignment you are claiming it is entirely your own work. You **may** discuss general ideas about the solution approach with other students. Describing details of how you implement a function or sharing part of your code with another student is considered to be **collusion** and will be counted as plagiarism. You may **not** copy fragments of code that you find on the Internet to use in your assignment.

Please read the section in the course profile about plagiarism. You are encouraged to complete *both* parts A and B of the academic integrity modules *before* starting this assignment. Submitted assignments will be electronically checked for potential cases of plagiarism.