Shuffle

SCHOOL **MINDARIE SENIOR COLLAGE** STUDENT **LAITH STRIEGHER**

LICENSE **GPLV3**

A classic puzzle game by Laith Striegher

About The Project

This simple shuffle game was created for Computer Science ATAR Unit 1, Task 1 Year 11

Getting Started

Prerequisites

To run the program you require;

One of:

OS **WINDOWS 10/11** OS **MACOSX** OS **DEBIAN**

With a desktop environment and

All of:

LANGUAGE **PYTHON 3** BUILTIN **RANDOM** BUILTIN **TKINTER** PYPI **PILLOW**

Installation

Windows 10/11

This will show you how to install Python3, Pip3, and Pillow on Windows 10/11.

Step 1: Download Python3

- Go to the Python download page and click on the "Download Python 3.x.x" button (x.x.x represents the latest version).
- Scroll down and select the appropriate version for your operating system (32-bit or 64-bit).

- Click on the "Download" button to start the download.

Step 2: Install Python3

- Double-click the downloaded file to start the installation process.
- Select "Add Python X.X to PATH" (X.X represents the version number) and click on the "Customize installation" button.
- Choose the components you want to install and click on the "Next" button.
- Select the installation directory and click on the "Install" button.
- Wait for the installation to complete.

Step 3: Download Pip3

- Go to the Pip download page and download the "get-pip.py" file.
- Save the file to your computer.

Step 4: Install Pip3

- Open the command prompt by pressing the "Windows" key and typing "cmd".
- Right-click on the "Command Prompt" option and select "Run as administrator".
- Navigate to the directory where the "get-pip.py" file is saved using the "cd" command (e.g. `cd C:\Users\Username\Downloads`).
- Type `python3 get-pip.py` and press "Enter".
- Wait for the installation to complete.

Official installation guides here

Step 5: Download Pillow

- Open the command prompt by pressing the "Windows" key and typing "cmd".
- Right-click on the "Command Prompt" option and select "Run as administrator".
- Type `pip install pillow` and press "Enter".
- Wait for the installation to complete.

Step 6: Verify the Installation

- Open the command prompt by pressing the "Windows" key and typing "cmd".
- Type `python3` and press "Enter" to open the Python interpreter.
- Type `import pillow` and press "Enter".
- If no error message appears, Pillow has been installed successfully.

You have successfully installed Python3, Pip3, and Pillow on your Windows 10/11 system.

Debian

Installing Python3 and pip3 on Debian

Step 1. First, update the package list using the following command:

```
sudo apt update
```

Step 2. Once the update process completes, install Python3 and pip3 by running the following command:

```
sudo apt install python3 python3-pip
```

Step 3. After the installation process completes, verify the installation of Python3 and pip3 using the following commands:

```
python3 --version
pip3 --version
```

Installing Pillow on Debian

Step 1. Install the required dependencies for Pillow using the following command:

```
sudo apt install libjpeg-dev zlib1g-dev libtiff-dev libfreetype6-dev liblcms2-dev libwebp-dev tcl8.6-d
```

Step 2. Once the dependencies are installed, install Pillow using pip3:

```
sudo -H pip3 install Pillow
```

Step 3. After the installation process completes, verify the installation of Pillow using the following command:

```
python3 -c "import PIL; print(PIL.__version__)"
```

Pillow is now successfully installed on your Debian system.

MacOSX

Step 1. Check if Python3 is already installed on your system by typing the following command in Terminal:

```
python3 --version
```

Step 2. If Python3 is not installed, download the latest version from the official Python website

Step 3. Once the download is complete, double-click on the downloaded file to launch the Python installer and follow the instructions to complete the installation process.

Step 4. Check if pip3 is installed by typing the following command in Terminal:

```
pip3 --version
```

Step 5. If pip3 is not installed, download the get-pip.py script by typing the following command in Terminal:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
```

Step 6. Once the download is complete, navigate to the directory where the get-pip.py script is saved and run the following command in Terminal:

```
sudo python3 get-pip.py
```

Step 7. After pip3 is installed, you can use it to install Pillow by typing the following command in Terminal:

```
sudo pip3 install pillow
```

Step 8. Pillow should now be installed on your system. You can check its version by typing the following command in Terminal:

```
python3 -c "import PIL; print(PIL.version)"
```

That's it! You now have Python3, pip3, and Pillow installed on your MacOSX system.

Usage

logic.py

This code implements the logic of the Shuffle game, which is a tile puzzle game where a player moves numbered tiles on a board to form a particular pattern or sequence. This file can be used by importing it into other Python scripts.

The `logic` class defines the game's logic and contains several methods to manipulate the game board. The class has the following methods:

__init__(board_size, self_start_enabled)

The `__init__(board_size=3, self_start_enabled=True)` method is a constructor method in Python classes. It is called when an object of the class is created. In this specific code, `__init__` initializes the `logic` class by setting its `board_size` attribute to the value passed in the `board_size` parameter (which defaults to 3 if no value is passed), and its `game_board` attribute to `None` .

If the optional `self_start_enabled` parameter is `True` (which is also the default value), `initialize_game_board()` , `shuffle_game_board()` , and `start_game()` methods are called to create a complete board, shuffle it using the `shuffle` method, and then `start_game` the game. If `self_start_enabled` is `False` , these methods are not called, and it is up to the user to call them manually.

Overall, `__init__` initializes the attributes of the `logic` class, and if specified, sets up the game board and shuffles it, ready to start_game playing the game.

initialize_game_board()

The `initialize_game_board()` method in the logic class of the given program creates a new game board by calling the `get_complete_game_board()` method, which returns a complete, ordered board. The ordered board is then assigned to the `game_board` instance variable of the `logic` object, thus creating a new game board.

In summary, `initialize_game_board()` is responsible for initializing a new game board with an ordered state.

shuffle_game_board(moves)

The `shuffle_game_board(moves)` method in this Python program is responsible for shuffling the board game's pieces randomly based on the number of moves specified by the `moves` parameter.

It uses a while loop to repeat the shuffling process until the specified number of moves have been made. Within each iteration of the while loop, the method chooses a random direction (0, 1, 2, or 3) and determines if the empty cell (represented by -1) can be moved in that direction. If the move is possible, it swaps the empty cell with the adjacent piece, effectively shuffling the board.

This process is repeated for the specified number of moves until the board is shuffled to the desired degree. The method returns the shuffled board.

move_cell_by_id(cell_id)

The `move_cell_by_id(cell_id)` method is a method of the `logic` class that is responsible for moving a given cell on the board. The method takes in a cell value as its input and uses this value to find the position of the cell on the board using the `find_cell_location` method. The method then checks if the cell can be moved in any of the four directions (up, down, left, or right) and if so, it swaps the position of the cell with the empty cell (-1) on the board. The method returns a value of 0 if the move was successful and a value of 1 if the cell cannot be moved.

find_cell_location(cell_id)

The method `find_cell_location(cell_id)` is a method of the `logic` class in the `logic.py` program. This method is used to find the location of a cell in the `game_board` 2D list.

The input `cell_id` is the value that needs to be found in the `game_board`. The method iterates through each element of the 2D list by scanning each column and row using nested `for` loops. If the current element being scanned matches the input `cell_id`, the method returns the location of the element as a list of `[y, x]`, where `y` is the row index and `x` is the column index.

If the input `cell_id` is not found in the `game_board`, the method returns `1` as an error code.

Import the 'logic' class from 'logic.py' into your script using the following command:

get_complete_game_board()

The `get_complete_game_board()` method is a method defined within the `logic` class of the `logic.py`, which returns a two-dimensional list that represents the completed game board, in the form of a list of lists.

Each sub-list within the list contains integers that represent the values of the cells in the respective row of the completed game board. The method creates a new 2D list with `self.board_size` rows and `self.board_size` columns, where each cell in the game board is numbered sequentially from 0 to (board_size * board_size - 1), except for the last cell, which is set to -1.

This represents an empty cell that can be used to move adjacent cells around to solve the puzzle.

The method returns the generated 2D list as the completed game board that can be used to verify if the current game board matches the completed board, which is used to determine if the game has been completed successfully.

get_game_board_string()

The `get_game_board_string()` method in the `logic` class returns a string representation of the current game board. It creates a string that consists of a grid of cells that represent the game board. Each cell is either a number or an empty space, and each cell is separated by a vertical bar `|`. The method also adds a horizontal bar `-` between each row and at the top and bottom of the grid to make the grid look like a game board. The method uses the `self.board` attribute of the class to create the string, which represents the current state of the game board. The returned string can be printed to display the current state of the game board.

is_game_board_complete()

The method `is_game_board_complete()` in `logic.py` is a method of the `logic` class that checks if the current board state is complete by comparing it with the complete board state. It returns a boolean value `True` if the current board state is complete, i.e., if it is in the same state as the complete board, and `False` otherwise.

The `get_complete_game_board()` method creates a complete board, which is a 2D incremental list representing a completed board state of the game. This method is called by the `is_game_board_complete()` method to obtain the complete board state, which is then compared with the current board state.

The `is_game_board_complete()` method returns `True` if the current board state is equal to the complete board state, and `False` otherwise. If the current board is in the same state as the complete board, it means that the game has been completed successfully.

get_cell_value(row, col)

The `get_cell_value(row, col)` method in the given code is used to retrieve the value of a particular cell on the board based on its row and column indices. It takes two arguments: `row` and `col`, which represent the indices of the cell whose value is being retrieved. The method returns the value of the cell located at the given row and column indices on the current game board.

For example, if you want to retrieve the value of the cell located at row 2 and column 3, you would call `get_cell_value(2, 3)` and the method would return the value of the cell at that location on the board.

move_cell(row, column)

The `move_cell(row, column)` method is used to move a tile on the game board to an empty cell, if the move is valid. The method takes two arguments, `row` and `column`, which represent the coordinates of the tile to be moved.

The method first checks if the tile is a valid move, by checking if it is adjacent to the empty cell. If the move is valid, it calls the `move_cell_by_id(cell_id)` method to move the tile into the empty cell. The `move_cell_by_id` method returns 0 if the move is successful.

If the move is not valid, the `move_cell` method returns 1.

start_game()

The method `start_game()` is a method of the `logic` class that represents the game interface. This method allows the user to play the game by displaying the board and asking the user for input on which cell to move.

The method starts with an infinite loop that only exits when the game is complete. Within the loop, the method first prints the current state of the board using `print(self.get_game_board_string())`. The user is then prompted to input the number of the cell they want to move using `int(input("What to move: "))`. This value is passed to the `move_cell_by_id(cell_id)` method to move the selected cell.

If the `move_cell_by_id(cell_id)` method returns `0`, meaning the move was successful, the `start_game()` method prints `Moved`. If the move was unsuccessful, the method prints `Can't move that`. The loop then checks if the game is complete by calling the `is_game_board_complete()` method. If the game is complete, the method prints `Complete!` and displays the final board state using `print(self.get_game_board_string())`. The loop is then broken and the method exits.

main.py

This is a program used to create a graphical user interface for the game. The program is implemented in Python using the `tkinter` library for creating the GUI and the logic module `logic.py` for implementing the core logic of the game.

The `shuffle` class is defined, and an object of this class is created when the program is run. The object has many attributes, including board size, window size, game title, shuffle rule, image file, background color, and more. The `shuffle_rule` attribute is used to set the number of shuffling moves for the puzzle.

The program initializes an empty list for the buttons and cell photos and a `logic` object for the game. The main `tkinter` window is created when the program is run.

The program uses the `logic` module to initialize and shuffle the game board. It then opens the image specified by the `image_location` attribute, crops and resizes it to fit the window when necessary. The class has the following methods:

__init__(board_size, window_size, game_title, shuffle_rule, image_file, default_bg_color, complete_bg_color, exit_on_complete, override, pos_esp, highlight_thickness, photo_options)

The method `__init__([args])` is the constructor method of the `shuffle` class. It is called when an object of the class is created. It initializes the attributes of the class with the values passed as arguments or with default values.

The arguments that can be passed are:

- `board_size` : an integer value to set the size of the board. The default value is 3.
- `window_size` : an integer value to set the size of the window. The default value is 1000.
- `game_title` : a string value to set the title of the game window. The default value is "Shuffle Game".
- `shuffle_rule` : an integer value to set the number of shuffles the game board will undergo. The default value is None.
- `image_file` : a string value to specify the location of the image file used in the game. The default value is None.
- `default_bg_color` : a string value to set the default background color of the buttons in the game. The default value is "#f0f0f0".
- `complete_bg_color` : a string value to set the background color of the buttons when they are correctly placed. The default value is "#00ff00".
- `exit_on_complete` : a boolean value to specify whether the game window should close automatically when the game is completed. The default value is False.
- `override` : a boolean value to specify whether the game window should be decorated or not. The default value is False.
- `pos_esp` : a boolean value to specify whether the game board should contain an empty cell. The default value is False.
- `highlight_thickness` : an integer value to specify the thickness of the border around the buttons. The default value is 1.
- `photo_options` : an array of strings to specify the list of image files that can be used in the game (a random one will be selected from the list). The default value is an array of image files.

The `__init__([args])` function also initializes several other attributes of the `shuffle` class such as the button size, number of moves, list of buttons, list of cell photos, and a logic object for the game. It also creates the main tkinter window.

The `reset` function is also defined in the class to reset the attributes of the `shuffle` object.

reset([args])

The `reset([args])` method is part of the shuffle class and is used to reset the game by reinitializing the attributes of the class with new values.

This method takes several arguments, which are used to initialize the attributes of the class in the same way as the `__init__([args])` method.

start()

The `start()` method is responsible for starting the shuffle game by initializing and shuffling the game board, opening the image specified by the `image_file` , adjusting the size of the image to fit the window, creating the buttons and cell photos, and starting the main loop of the tkinter window.

In summary, this method is the main driver of the game, and it orchestrates the various components of the game to create a playable interface for the user.

handle_click(clicked_button)

The method `handle_click(clicked_button)` is responsible for handling the event of clicking on a button.

When a button is clicked, this function extracts the row and column of the clicked button, and then calls the `move_cell(col, row)` method of the `game_logic` object (which is an instance of the `logic` class from the `logic` module) to move the clicked button to the empty cell. If the move is valid, it increments the `num_moves` counter.

After updating the state of the game, the function then calls the `update_buttons()` method to update the text and image of the buttons on the GUI.

update_buttons()

The method `update_buttons()` updates the text and images of the buttons on the game board based on the current state of the game. It is called every time a player makes a move by clicking on a button.

The function iterates through every button on the game board and gets the value of the cell at that button's position from the `game_logic` object. If the cell is not blank, the button's text is set to the value of the cell and its image is set to the corresponding image from the list of photos. If the cell is blank, the button's text is set to a blank string and its image is set to a blank image.

If the `pos_esp` attribute is set to `True` , the function checks if the current cell's location matches its location on the complete game board, and removes any highlighting if they match.

The function also checks if the game board is complete and sets the background color of the button to the complete color if it is. If the `exit_on_complete` attribute is set to `True` , the function terminates the program.

run.py

The purpose of this program is to run the game and implement classes. It imports the ▶ `main` module and creates a game instance with specific configurations such as board size, window size, title, shuffle rule, colors, photos, and more. It then starts the game and runs it in an infinite loop, resetting the game after each completion.

Usage

The following details how you can run the game without the `run.py` template

Usage with user interface (main.py)

Here are some usage implementation examples for the "shuffle" class:

- Create a new instance of the "shuffle" class with default settings and start the game:

```
from main import shuffle
game = shuffle()
game.start()
```

- Create a new instance of the "shuffle" class with custom settings and start the game:

```
from main import shuffle
game = shuffle(board_size=4, window_size=1200, game_title="My Shuffle Game", shuffle_rule=100, image_f

               default_bg_color="#000000", complete_bg_color="#ffffff", exit_on_complete=True)
game.start()
```

- Reset the settings of an existing game instance and start a new game:

```
game.reset(board_size=5, window_size=1500, game_title="My New Shuffle Game", shuffle_rule=50, image_fi
           default_color="#ffffff", complete_color="#000000", exit_on_completion=True)
game.start()
```

Usage without graphical user interface (standalone logic.py)

Here are some usage examples for the `logic` class:

- Creating a new instance of the game logic class:

```
from logic import logic
game = logic()
```

This creates a new game with a board size of 3 (the default) and enables automatic shuffling and starting of the game.

- Creating a new instance of the game logic class with a larger board:

```
from logic import logic
game = logic(board_size=4)
```

This creates a new game with a board size of 4 and enables automatic shuffling and starting of the game.

- Initializing a game board:

```
game.initialize_game_board()
```

This creates a new game board based on a complete board (a board with the cells in numerical order from 1 to N^2-1, with the last cell empty).

- Shuffling a game board:

```
game.shuffle_game_board(100)
```

This shuffles the game board by making 100 random moves.

- Moving a cell on the game board:

```
game.move_cell_by_id(5)
```

This moves the cell with value 5 by swapping it with the empty space.

License