PyJumble: Powered by PyGame

A Python Based Word Unscramble Game

Developed by **Gerard Ian M. Balaoro**
In Partial Fulfillment on the Requirements for the Subject
Computer Science 11
1st Semester, A.Y. 2018-2019
University of the Philippines Diliman

Code Documentation

File Structure

The source files of the application are arranged according to the diagram below.

```
.
├── ...
├── assets # Application Assets
| ├── audio # Sounds / Music Files (*.wav)
| ├── fonts # Font Files (*.ttf)
| ├── images # Images (*.png)
| └── source # Dictionary Files (.*txt)
├── config.py # Application Configuration
├── engine.py # Game Logic Engine Class
├── interface.py # PyGame Interface Class and Methods
├── main.py # Main / Entry Script
├── sprites.py # Sprites / Game Object Classes
└── ...
```

Interface

The application interface powered by the **PyGame Framework** are composed of two parts: the **Game** and **Sprite** classes.

Game Class

The **Game** class is located at the **interface.py** file and serves as the primary provider and manager of the application's interface.

```
game = Game()
```

When called, the class uses the values stored at the **config.py** to initialize the game window and the resources required by each of its components.

load

```
game.load()
```

Load Resources

Returns: `None`

new

```
game.new(engine, [lives = 3, time = 60])
```

Start a New Game

> From the engine instance passed, this method automatically retrieves 3 words from the dictionary using the `random.sample()` function and creates a new letter pool.
>
> It also sets the global variables `lives` and `time`.

- Arguments:
  - **engine** `Engine`: an instance of **game engine**
  - **lives** `int` : Starting lives, 0 = infinite
  - **time** `int` : Time limit in seconds, 0 = infinite
- Returns: `None`

run

```
game.new()
```

Run Game Loop. Update Components and Sprites. Check Game Mode Rules.

> Stop execution if `lives` or `timer` has reached zero

- Returns: `None`

update

```
game.update()
```

Refresh Game Window

- Returns: `None`

events

```
game.events()
```

Handles Game Events *[i.e. user input and key press]*

- Returns: `None`

draw

```
game.draw()
```

Draw Game Elements

- Returns: `None`

start_screen

```
game.start_screen()
```

Show Start Screen

- Returns:
  - `dict` : Game mode configuration, see **config.py**
  - `str` : Only if `'i'` is selected

credits

```
game.credits()
```

Show Credits Screen

- Returns: `None`

game_over

```
game.game_over()
```

Show Game Over Screen

- Returns: `None`

wait_input

```
game.wait_input()
```

Wait for User Input

> Used for Static Screens (Start Menu, Game Over, etc.)

- Arguments:
  - **accepted** `list`: List of accepted keys
- Returns:
  - `str`: Key name, none

Sprites

The **sprites.py** contains `pygame.sprite.Sprite` classes that are used throughout the interface

Letter

```
Letter(letter, [position = 1, length = 1, size = 80, margin = 55, button = False])
```

Produces a single letter tile

Image

```
Image(name, [scale = 1, x = 0, y = 0])
```

Produces an image block

Text

```
Text(letter, [text, size, color, x = 0, y = 0, font = 'GothamNarrow-Medium.ttf'])
```

Produces a text block

> Adapted from the work of Gareth Rees
>
> Copyright (c) 2013. Under CC-BY-SA 3.0 License

Button

```
Button(text, [x, y, size = 15, scale = 0.50, color = None, image = 'button.png', font =
'GothamNarrow-Medium.ttf', text_offset = (-1,-1)])
```

Produces a button

Media

All media used on the **Game** class are loaded through the `__init__()` and `load` functions. Sources are declared at the **config.py**

Engine

The `Engine` class located at the **engine.py** contains the core program logic of the application.

```
engine = Engine([path = ''])
```

- Arguments:
  - **path** `str` : Optional path to dictionary file

When called, the class automatically loads the default dictionary file from **config.py** if no path is passed.

seed

```
engine.seed(path)
```

Read Dictionary Text File

> The file contents are assigned to the global variable `dictionary` accessible in-class using `self.dictionary` and outside the class using `engine.dictionary`

- Arguments:
  - **path** `str` : Path to dictionary file
- Returns: `None`

pick

```
engine.pick(indices)
```

Get List of Words from Dictionary

> This method is not used in the implementation as the method `random.sample()` provides a quicker way of retrieving a select number of values from an iterable object

- Arguments:
  - **indices** `list` : List of `int` indices
- Returns:
  - `list` : List of words

search

```
engine.search(anagram)
```

Find Words from Dictionary Using an Anagram

- Arguments:
  - **anagram** `str`

- Returns:
  - `list` : List of words

combine

```
engine.combine(words)
```

Create a Letter Pool from Words

> The pool characters are assigned to the global variable `pool` accessible in-class using `self.pool` and outside the class using `engine.pool`

- Arguments:
  - **words** `str`
- Returns:
  - `list` : List of letters

check

```
engine.check(word)
```

Check if a Word from a `dictionary` can be formed using the characters from a `pool`

- Arguments:
  - **word** `str`
- Returns: `bool`

score

```
engine.score(word)
```

Calculates the Score of a Word using Scrabble Points

- Arguments:
  - **word** `str`
- Returns:
  - `int` : Word score

Configuration

All constants and other objects that are used throughout the application are defined at the **config.py**. This file is imported in all of the scripts that constitute the program.

Interface

```
#: ====================================
#: INTERFACE
#: ====================================
TITLE = 'PyJumble'                              # Window Title
WIDTH = 900                                     # Window Width
HEIGHT = 600                                    # Window Height
FPS = 60                                        # Window Refresh Rate
BACKGROUND = 'assets/images/background.png'     # Background Image
ICON = 'assets/images/icon.png'                 # Window Icon
```

Game Modes

```
#: ====================================
#: GAME MODES
#: ------------------------------------
#: Lives/Time: 0 = Infinite
#: Key: Selection Key on Start Screen
#: ====================================
MODES = [
    {
        'name': 'BASIC',
        'key': 'b',
        'lives': 3,
        'time': 0
    }
]
```

Audio Files

```
#: ====================================
#: AUDIO FILES
#: ====================================
AUDIO = {
    'enter': 'assets/audio/swap.wav',        # Enter/Backspace/Esc Key Press
    'click': 'assets/audio/swap.wav',        # Alpha Key Press
    'success': 'assets/audio/match.wav',     # Correct Answer
    'fail': 'assets/audio/error.wav',        # Wrong Answer
    'start': 'assets/audio/start.wav',       # Start of a New Game
    'end': 'assets/audio/over.wav',          # End of Game
    'menu': 'assets/audio/yippee.wav',       # Background Music on Start Screen
    'game': 'assets/audio/happytune.wav',    # Background Music on Game Screen
}
```

Running the Game

```
python main.py
```

Credits

- Interface Based on Graphic by Vecteezy
- YIPPEE by Snabisch
- Happy Tune by syncopica
- Other Sounds are Generated using Diforb
- Hearts Icon by Smashicons from www.flaticon.com
- Three quarters of an hour Icon by Freepik from www.flaticon.com
- Game Structure Based on https://github.com/kidscancode/pygame_tutorials