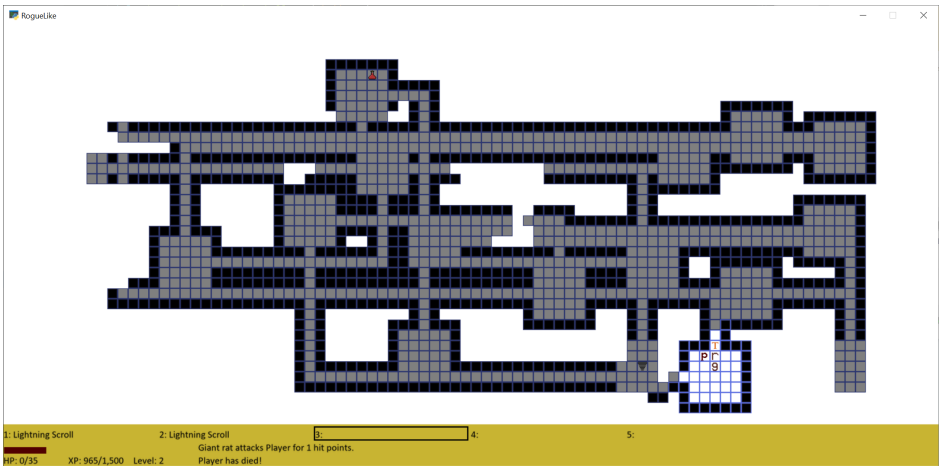
PyPi Package Example Documentation

Release 1.0.0

Paul Vincent Craven

Dec 05, 2020

CONTENTS



This is a rogue-like adventure written in the Python computer language using the Arcade Library.

Use this code to hack away at creating your own adventure game!

CHAPTER

ONE

HOW TO INSTALL

- Download or clone the code from GitHub: https://github.com/pythonarcade/roguelike
- Switch to the directory, or open in an IDE like PyCharm
- Create a new virtual environment
- Install the depended packages through your IDE or via `pip install -r requirements.txt`
- Run with `python source`

CHAPTER

TWO

HOW TO PLAY

2.1 Key Bindings

- Move with the number pad in 8 directions (num lock off)
- Pick up an item with `G` or `Num 5`
- Select an item with the numbers `1 - 9`
- Use the selected item with `U`
- Drop the selected item with `D`
- Save game with `S`
- Load game with `L`
- Bring up the character screen with `C`
  - If you have ability points, click on the + to increase that stat
- Cancel the grid selection, character screen, etc. with `Esc`

2.2 Combat

- Move 'into' a monster to attack it
- Fireball is an area of effect weapon, and can damage the player.
- Lightning attacks the closest monster

CHAPTER

THREE

FEATURES

- Procedural dungeons
- Character leveling system
- Ranged lighting spell
- Area of effect spell
- Field of vision
- Monster table
- Inventory management system
- A-star path-finding for monsters
- Can save/restore dungeon via JSON formatted data
- Message/event system

CHAPTER

FOUR

CODE DOCUMENTATION

At the top level, the `__main__.py` file will create an instance of the `GameWindow` class. This class manages the GUI and responds to events.

The next layer down is the `GameEngine`. This is more of a logic layer, although it isn't completely divorced from the display.

4.1 Game Window

Main Window Manager.

class source.game_window.**MyGame**(*width*, *height*, *title*)

Main application class. Manage the GUI

__init__(*width*, *height*, *title*)

Parameters

- **width** (int) –
- **height** (int) –
- **title** (str) –

check_for_player_movement()

Figure out if we should move the player or not based on keys currently held down.

draw_character_screen()

draw_hp_and_status_bar()

draw_in_normal_state()

draw_in_select_location_state()

draw_inventory()

draw_messages()

draw_mouse_over_text()

draw_sprites_and_status_panel()

handle_character_screen_click(*x*, *y*)

handle_messages()

load()

Load the game from disk.

on_draw()
Render the screen.

on_key_press(*key*, *modifiers*)
Manage key-down events

Parameters

- **key** (int) –
- **modifiers** (int) –

on_key_release(*key*, *modifiers*)
Called when the user releases a key.

Parameters

- **key** (int) –
- **modifiers** (int) –

on_mouse_motion(*x*, *y*, *dx*, *dy*)
Handle mouse motion, mostly just used for mouse-over text.

on_mouse_press(*x*, *y*, *button*, *modifiers*)
Handle mouse-down events

Parameters

- **x** (float) –
- **y** (float) –
- **button** (int) –
- **modifiers** (int) –

on_update(*delta_time*)
Manage regular updates for the game

Parameters **delta_time** (float) –

save()
Save the current game to disk.

setup()
Set up the game here. Call this function to restart the game.

source.game_window.**main**()
Main method for starting the rogue-like game

4.2 Game Engine

Define the game engine

class source.game_engine.**GameEngine**
This is the main game engine class, that manages the game and its actions.

__init__()
Set the game engine's attributes

check_experience_level()
See if the player should level up

dying(*target*)
Handle event of an entity dying

Parameters **target** (Entity) –

Return type list

get_dict()
Get a dictionary object for the entire game. Used in serializing the game state for saving to disk or sending over the network.

grid_click(*grid_x*, *grid_y*)
Handle a click on the grid

move_enemies()
Process enemy movement.

move_player(*cx*, *cy*)
Process player movement

Parameters

- **cx** (int) –
- **cy** (int) –

pick_up()
Handle a pick-up item entity request.

process_action_queue(*delta_time*)
Process the action queue, kind of a dispatch-center for the game.

Parameters **delta_time** (float) –

restore_from_dict(*data*)
Restore this object from a dictionary object. Used in recreating a game from a saved state, or from over the network.

Parameters **data** (dict) –

setup()
Set up the game here. Call this function to restart the game.

setup_level(*level_number*)

Parameters **level_number** (int) –

Return type *GameLevel*

use_stairs()

class source.game_engine.**GameLevel**

__init__()
Initialize level instance.

Other links:

- Source on GitHub
- Arcade Library
- License (MIT)

PYTHON MODULE INDEX

S

INDEX

Symbols

C

D

G

H

L

M

O