donjuan

Release 12-20-2020

Tom McClintock, Jamie McClintock, James Devore

Jan 12, 2021

CONTENTS:

This package is a rebuild of the donjon dungeon generator, with all the benefits that come from using modern tools. In *donjuan*, all parts of the dungeon are objects that can be subclassed and serialized (saved to disk) on their own. This includes things like walls, doors, and rooms, as well as algorithms for actual dungeon generation, renderers for different image formats, and exporters for saving map files for different VTT applications.

CHAPTER

ONE

DONJUAN

A translation/rebuild of the original donjon dungeon generator. This package aims to deconstruct the original script into extendable parts, and provide all pieces for customization for different purposes and not just the map image. For example, when complete this package will automatically generate walls, doors, and light sources for use in Foundry Virtual Tabletop.

You can find the documentation here.

1.1 Installation

Install `donjuan` with `pip`:

```
pip install donjuan
```

You can find the package details here on PyPI.

You can also install `donjuan` using the `setup.py` file. To do so, you must first clone or download this repository and install the requirements.

Assuming you have git, you can do:

```
git clone https://github.com/tmcclintock/donjuan
cd donjuan
pip install -r requirements.txt
```

Then you can install with:

```
python setup.py install
```

If you have conda you can install the requirements using the *environment.yml* file before installing:

```
conda env create -f environment.yml
conda activate donjuan
python setup.py install
```

To run the test suite, you must have pytest installed. You can run the tests with:

```
pytest
```

which can be done from the root of the repository. To run all tests, including those with graphical outputs, run with the `runslow` flag:

```
pytest --runslow
```

Please report any issues you encounter on our issue page. Doing so will help make donjuan even better!

1.2 Contributing

To contribute to donjuan please see the developing page.

CHAPTER

TWO

DEVELOPING

Developing *donjuan* should follow best practices. This includes documentation, testing, and using composability.

Composability is the design concept that complex objects use more simple objects that are responsible for very specfic tasks. For example, a *Dungeon* is composed of rooms and passages that are fundamentally made up of *Cell* objects that individually have properties, such as *Door* objects and *Faces*.

If you want to implement a new piece of logic and you are not sure where it goes, create a new class for it and use that class as appropriate.

2.1 Contributing

To contribute, please start by forking the repository and creating a branch for the feature you would like to work on. Once you clone your feature branch, install the developer environment, activate it and install *donjuan* in editable mode:

```
conda env create -f environment_dev.yml
conda activate djdev
pip install -e .
```

In this setup, any changes you make to the source code of *donjuan* will be seen immediately by your *djdev* environment. Furthermore, you will not have to reinstall *donjuan* if you switch branches.

Once your development environment is activated, please install pre-commit before committing any changes:

```
pre-commit install
```

This way, any time you perform a *git commit* command, the *black*, *flake8*, and *isort* packages will run to clean up your code. If you see a "failure" message, then simply *git reset* and then fix any issues the *pre-commit* packages are giving you. If *pre-commit* is being too annoying, then you can always recreate your conda environment without installing it.

Once your code is ready, commit and push it to your branch and issue a Pull Request on GitHub.

2.2 Testing data and a sample dungeon

Tests should be written in the *tests/* directory, so that running *pytest* at the root finds and runds all tests found there. Tests should be functions attached to classes that subclass *unittest.TestCase*. This allows you to write *setUp* and *tearDown* functions that run before and after all tests. A sample dungeon is found in *tests/fixtures/dummy_dungeon.json*, where the *filled* attribute of a five-by-five dungeon have been saved in a 2D array. To load in this dungeon in memory simply run:

```
import json
fp ="/path/tp/dummy_dungeon.json"
with open(fp, "r") as file:
    dungeon_json = json.load(file)
dungeon_array = dungeon_json["dungeon"]  # the actual array
```

Then, to turn this into an actual *Dungeon* object, instantiate a *Dungeon* that is five by five and assign the *filled* values according to the values in the read array.

```
dungeon = Dungeon(n_rows=5, n_cols=5)

for i in range(5):
    for j in range(5):
        dungeon.grid.cells[i][j].filled = bool(dungeon_array[i][j])
```

2.3 Docstrings

Docstrings are automatically rendered in the online documentation using Napolean sphinx. Docstrings follow the Google style. All classes and non-trivial functions and properties should be documented. Function signatures should be typed. Module-level docstrings at the top of *.py* files are not required.

CHAPTER

THREE

API REFERENCE

This page contains auto-generated API reference documentation[1].

3.1 donjuan

3.1.1 Submodules

donjuan.cell

Module Contents

Classes

Cell	The cell represents a single 'unit' in the map. The attributes on the cell
SquareCell	A cell for a square grid.
HexCell	A cell for a hexagonal grid.

class donjuan.cell.**Cell**(*filled: bool = False*, *door_space: Optional[DoorSpace] = None*, *contents: Optional[List[Any]] = None*, *coordinates: Optional[Tuple[int, int]] = None*, *space: Optional['Space'] = None*, *edges: Optional[List['Edge']] = None*)

Bases: abc.ABC

The cell represents a single 'unit' in the map. The attributes on the cell define what exists in that unit. This includes the terrain, doors, walls, lights, other room details etc.

set_coordinates(*self*, *y: int*, *x: int*) → None

set_edges(*self*, *edges: List['Edge']*) → None

set_space(*self*, *space: Type['Space']*) → None

set_x(*self*, *x: int*) → None

set_y(*self*, *y: int*) → None

property coordinates(*self*) → Tuple[int, int]

property edges(*self*) → List['Edge']

[1] Created with sphinx-autoapi

property space(*self*) → Optional[Type['Space']]
`Space` this cell is a part of.

property x(*self*) → int

property y(*self*) → int

property n_sides(*self*) → int

class donjuan.cell.**SquareCell**(*filled: bool = False, door_space: Optional[DoorSpace] = None, contents: Optional[List[Any]] = None, coordinates: Optional[Tuple[int, int]] = None*)

Bases: *donjuan.cell.Cell*

A cell for a square grid.

Parameters

- **filled** (*bool, optional*) – flag indicating whether the cell is filled (default `False`)
- **door_space** (*Optional[DoorSpace]*) – kind of doorway in this cell
- **contents** (*Optional[List[Any]]*) – things in this cell

_n_sides = 4

class donjuan.cell.**HexCell**(*filled: bool = False, door_space: Optional[DoorSpace] = None, contents: Optional[List[Any]] = None, coordinates: Optional[Tuple[int, int]] = None*)

Bases: *donjuan.cell.Cell*

A cell for a hexagonal grid.

Parameters

- **filled** (*bool, optional*) – flag indicating whether the cell is filled (default `False`)
- **door_space** (*Optional[DoorSpace]*) – kind of doorway in this cell
- **contents** (*Optional[List[Any]]*) – things in this cell

_n_sides = 6

donjuan.door_space

Door ways that connect rooms to hallways and rooms to rooms.

Module Contents

Classes

DoorSpace	Abstract base class for different kinds of doors. Door spaces can have many
Archway	An archway to walk through.
Door	A generic door.
Portcullis	You can look but can't touch!

class donjuan.door_space.**DoorSpace**(*secret: bool, locked: bool, closed: bool, jammed: bool, blocked: bool, broken: bool, material: str, name: str*)

Bases: `abc.ABC`

Abstract base class for different kinds of doors. Door spaces can have many properties, like if they are locked or blocked etc. To facilitate this logic in the generative process, these are encompassed in the attributes of a `DoorSpace`.

__slots__ = ['locked', 'closed', 'jammed', 'blocked', 'secret', 'broken', 'material',

__str__(*self*)
Return str(self).

class donjuan.door_space.**Archway**(*material: str = 'stone'*, *blocked: bool = False*, *broken: bool = False*, *secret: bool = False*)
Bases: *donjuan.door_space.DoorSpace*

An archway to walk through.

class donjuan.door_space.**Door**(*secret: bool = False*, *locked: bool = False*, *closed: bool = True*, *jammed: bool = False*, *blocked: bool = False*, *broken: bool = False*, *material: str = 'wood'*)
Bases: *donjuan.door_space.DoorSpace*

A generic door.

class donjuan.door_space.**Portcullis**(*locked: bool = False*, *closed: bool = True*, *jammed: bool = False*, *broken: bool = False*, *material: str = 'metal'*)
Bases: *donjuan.door_space.DoorSpace*

You can look but can't touch!

donjuan.dungeon

Module Contents

Classes

Dungeon

class donjuan.dungeon.**Dungeon**(*n_rows: Optional[int] = 5*, *n_cols: Optional[int] = 5*, *grid: Optional[Grid] = None*, *rooms: Optional[Dict[str, Room]] = None*, *hallways: Optional[Dict[str, Hallway]] = None*, *randomizers: Optional[List['Randomizer']] = None*)

add_room(*self*, *room: Room*) → None

add_hallway(*self*, *hallway: Hallway*) → None

property grid(*self*) → Grid

property hallways(*self*) → Dict[str, Hallway]

property n_cols(*self*) → int

property n_rows(*self*) → int

property randomizers(*self*) → List['Randomizer']

property rooms(*self*) → Dict[str, Room]

randomize(*self*) → None
For each item in *randomizers*, run the `Randomizer.randomize_dungeon()` method on this dungeon.

emplace_rooms(*self*) → None
Replace the cells in the *grid* with the cells of the *rooms*.

emplace_space(*self*, *space: Space*) → None
Replace the cells in the *grid* with the cells of the *Space*, and automatically link them with the *Edge`s in the `Grid*.

Parameters **space** (Space) – room to emplace in the *grid*

donjuan.dungeon_randomizer

Module Contents

Classes

DungeonRandomizer	Randomize a dungeon by first creating rooms and then applying

class donjuan.dungeon_randomizer.**DungeonRandomizer**(*room_entrance_randomizer: Optional[Randomizer] = None*, *room_size_randomizer: Optional[Randomizer] = None*, *room_name_randomizer: Optional[Randomizer] = None*, *room_position_randomizer: Optional[Randomizer] = None*, *max_num_rooms: Optional[int] = None*, *max_room_attempts: int = 100*)

Bases: *donjuan.randomizer.Randomizer*

Randomize a dungeon by first creating rooms and then applying room size, name, and position randomizers to sequentially generated rooms.

Parameters

- **room_entrance_randomizer** (*Optional[Randomizer]*) – randomizer for the entrances of a room. If None then default to a RoomEntrancesRandomizer.
- **room_size_randomizer** (*Optional[Randomizer]*) – randomizer for the room size. It must have a 'max_size' attribute. If None then default to a RoomSizeRandomizer.
- **room_name_randomizer** (*Optional[Randomizer]*) – randomizer for the room name. If None default to a AlphaNumRoomName.
- **room_position_randomizer** (*Optional[Randomizer]*) – randomizer for the room position. If None default to a RoomPositionRandomizer.
- **max_num_rooms** (*Optional[int]*) – maximum number of rooms to draw, if ``None` then default to the max_room_attempts. See DungeonRoomRandomizer.get_number_of_rooms() for details.
- **max_room_attempts** (*int, optional*) – default is 100. Maximum number of attempts to generate rooms.

get_number_of_rooms (*self*, *dungeon_n_rows: int*, *dungeon_n_cols: int*) → int
Randomly determine the number of rooms based on the size of the incoming grid or the `max_num_rooms` attribute, whichever is less.

Parameters

- **dungeon_n_rows** (*int*) – number of rows
- **dungeon_n_cols** (*int*) – number of columns

randomize_dungeon (*self*, *dungeon: Dungeon*) → None
Randomly put rooms in the dungeon.

Parameters **dungeon** (Dungeon) – dungeon to randomize the rooms of

donjuan.edge

Module Contents

Classes

Edge	An edge sits between two `Cell`s.

class donjuan.edge.**Edge**(*cell1: Optional[Cell] = None*, *cell2: Optional[Cell] = None*, *has_door: bool = False*)
An edge sits between two `Cell`s.

Parameters

- **cell1** (*Optional[Cell]*) – cell on one side of the edge
- **cell2** (*Optional[Cell]*) – cell on the other side of the edge
- **has_door** (*bool, optional*) – default `False`, indicates whether this object has a door

property cell1 (*self*) → Optional[Cell]

property cell2 (*self*) → Optional[Cell]

set_cell1 (*self*, *cell: Cell*) → None

set_cell2 (*self*, *cell: Cell*) → None

property is_wall (*self*) → bool

donjuan.face

Module Contents

Classes

Face	Abstract base class for the geometric face of a
BareFace	A face with nothing on it.
DoorFace	A face with a door on it.

continues on next page

Table 6 – continued from previous page

Faces	A collection of faces of a cell.
SquareFaces	Four faces surrounding a square cell.
HexFaces	Six faces surrounding a hexagonal cell.

class donjuan.face.**Face**(*direction: int = 0*)
Bases: abc.ABC

Abstract base class for the geometric face of a *Cell*.

Parameters **direction** (*int, optional*) – represents the outer direction of the face (default 0)

class donjuan.face.**BareFace**(*direction: int = 0*)
Bases: *donjuan.face.Face*

A face with nothing on it.

class donjuan.face.**DoorFace**(*door_space: DoorSpace*, *direction: int = 0*)
Bases: *donjuan.face.Face*

A face with a door on it.

class donjuan.face.**Faces**(*faces: List[*Face*]*)
A collection of faces of a cell.

_init_faces(*self*) → None

property faces(*self*) → List[*Face*]

__len__(*self*) → int

__getitem__(*self*, *key*)

class donjuan.face.**SquareFaces**(*faces: Optional[List[*Face*]] = None*)
Bases: *donjuan.face.Faces*

Four faces surrounding a square cell.

class donjuan.face.**HexFaces**(*faces: Optional[List[*Face*]] = None*)
Bases: *donjuan.face.Faces*

Six faces surrounding a hexagonal cell.

donjuan.grid

Module Contents

Classes

Grid	Abstract base class for a grid of cells. The underlying grid can either
SquareGrid	Rectangular grid of square cells. In a square grid, the cell positions are
HexGrid	Rectangular grid of hexagonal cells. In a hex grid, the cell positions

class donjuan.grid.**Grid**(*n_rows: int*, *n_cols: int*, *cells: Optional[List[List[Cell]]] = None*, *edges: Optional[List[List[List[Edge]]]] = None*)

Bases: abc.ABC

Abstract base class for a grid of cells. The underlying grid can either be square or hexagonal.

get_filled_grid(*self*) → List[List[bool]]
Obtain a 2D array of boolean values representing the filled state of the cells attached to the grid.

property n_rows(*self*) → int

property n_cols(*self*) → int

property cells(*self*) → List[List[Cell]]

property edges(*self*) → List[List[List[Edge]]]

reset_cell_coordinates(*self*) → None
Helper function that sets the coordinates of the cells in the grid to their index values.

check_edges(*self*, *edges: Optional[List[List[List[Edge]]]]*) → None
Check the dimensions of the *edges*.

init_edges(*self*) → List[List[List[Edge]]]

link_edges_to_cells(*self*) → None
For an *Edge*, the Edge.cell1 always points to either the left or upper *Cell*. The Edge.cell2 always points to the right or the bottom.

abstract link_cells_to_edges(*self*) → None

class donjuan.grid.**SquareGrid**(*n_rows: int*, *n_cols: int*, *cells: Optional[List[List[SquareCell]]] = None*)
Bases: *donjuan.grid.Grid*

Rectangular grid of square cells. In a square grid, the cell positions are integers.

cell_type

link_cells_to_edges(*self*) → None

class donjuan.grid.**HexGrid**(*n_rows: int*, *n_cols: int*, *cells: Optional[List[List[HexCell]]] = None*)
Bases: *donjuan.grid.Grid*

Rectangular grid of hexagonal cells. In a hex grid, the cell positions are integers, with odd rows being "offset" by half a cell size when rendered.

cell_type

link_cells_to_edges(*self*) → None

donjuan.hallway

Module Contents

Classes

Hallway	A hallway in a dungeon. It has a start and end cell.

class donjuan.hallway.**Hallway**(*ordered_cells: Optional[List[Cell]] = None*, *name: Union[int, str] = ''*)
Bases: *donjuan.space.Space*

A hallway in a dungeon. It has a start and end cell.

Parameters

- **ordered_cells** (*Optional[Sequence[Cell]]*) – ordered list of cells, where the order defines the path of the hallway. If `None` defaults to an empty list.
- **name** (*Union[int, str], optional*) – defaults to '', the name of the hallway

property ordered_cells (*self*) → List[Cell]
Cells that make up the path of the hallway. Does not contain extra cells that may be associated with this object (i.e. those off of the "path"). For the set of all cells, use `cells`.

property end_cell (*self*) → Cell

property start_cell (*self*) → Cell

append_ordered_cell_list (*self*, *cells: List[Cell]*) → None
Append cells in order to the hallway. To add a cell to the hallway without including it in the hallways path, use `add_cells()`.

Parameters **cells** – (List[Cell]): cells to append to the hallway

get_coordinate_path (*self*) → List[Tuple[int, int]]
Get the coordinates of the cells that make up this path this hallway takes. Does not contain coordinates of extra cells on this object.

Returns coordinates of the hallway path

donjuan.randomizer

Module Contents

Classes

Randomizer	Class for randomizing features of a dungeon.
RandomFilled	Randomly set the `filled` attribute of cells.

class donjuan.randomizer.**Randomizer**
Class for randomizing features of a dungeon.

randomize_cell (*self*, *cell: Cell*) → None
Randomize properties of the *Cell*

randomize_dungeon (*self*, *dungeon: Dungeon*) → None
Randomize properties of the *Dungeon*

randomize_grid (*self*, *grid: Grid*) → None
Randomize properties of the *Grid*

randomize_hallway (*self*, *hallway: Hallway*) → None
Randomize properties of the *Hallway*

randomize_room_entrances (*self*, *room: Room*, **args*) → None
Randomize the entrances of the *Room*

randomize_room_size (*self*, *room: Room*, **args*) → None
Randomize the size of the *Room*

randomize_room_name (*self*, *room: Room*, **args*) → None
Randomize the name of a *Room*

randomize_room_position(*self*, *room: Room*, **args*) → None
Randomize the position of a *Room*

classmethod seed(*cls*, *seed: Optional[int] = None*) → None

Parameters **seed** (*Optional[int]*) – seed passed to `random.seed()`

class donjuan.randomizer.**RandomFilled**
Bases: *donjuan.randomizer.Randomizer*

Randomly set the `filled` attribute of cells.

randomize_cell(*self*, *cell: Cell*) → None
Randomly fill the cell with probability 50%

randomize_grid(*self*, *grid: Grid*) → None
Randomly fill all cells of the grid individually

donjuan.renderer

Module Contents

Classes

BaseRenderer	Base class for rendering dungeons into images.
Renderer	Default renderer for rendering dungeons into PNG files using *matplotlib*.

class donjuan.renderer.**BaseRenderer**(*scale: float*)
Bases: `abc.ABC`

Base class for rendering dungeons into images.

property scale(*self*) → float
The scale size of a single *Cell*. The meaning differs depending on the subclass.

abstract render(*self*, *dungeon: Dungeon*) → None

class donjuan.renderer.**Renderer**(*scale: float = 1.0*)
Bases: *donjuan.renderer.BaseRenderer*

Default renderer for rendering dungeons into PNG files using *matplotlib*.

Parameters **scale** (*float, optional*) – size of a single cell in inches (default is 1 inch).

render(*self*, *dungeon: Dungeon*, *file_path: str = 'rendered_dungeon.png'*, *dpi: int = 200*, *save: bool = True*) → Tuple
Render the dungeon.

Parameters

- **dungeon** (Dungeon) – dungeon to render
- **file_path** (*str, optional*) – path to save the dungeon at (default is *rendered_dungeon.png*)
- **dpi** (*int, optional*) – dots per inch used to save (default is 200)
- **save** (*bool, optional*) – flag indicating whether to save the dungeon with `matplotlib.pyplot.savefig()`

donjuan.room

Module Contents

Classes

Room	A room in a dungeon. A room has is a named `Space`.

class donjuan.room.**Room**(*cells: Optional[Set[Cell]] = None*, *name: Union[int, str] = ''*)

Bases: *donjuan.space.Space*

A room in a dungeon. A room has is a named `Space`.

donjuan.room_randomizer

Module Contents

Classes

RoomSizeRandomizer	Randomize the size of a *Room*.
AlphaNumRoomName	Simple room name randomizer that names rooms as alphabetical letters.
RoomPositionRandomizer	Randomly shift a room, assuming its left edge is at column 0 and it's top
RoomEntrancesRandomizer	Randomizes the number of entrances on a room. The number is picked to be

class donjuan.room_randomizer.**RoomSizeRandomizer**(*min_size: int = 2*, *max_size: int = 4*, *cell_type: Type[Cell] = SquareCell*)

Bases: *donjuan.randomizer.Randomizer*

Randomize the size of a *Room*.

randomize_room_size(*self*, *room: Room*) → None

Randomly determine the size of the room, and set the cells of the room to a 2D array of unfilled cells of that size.

class donjuan.room_randomizer.**AlphaNumRoomName**

Bases: *donjuan.randomizer.Randomizer*

Simple room name randomizer that names rooms as alphabetical letters. followed by a number. Rooms are sequentially named 'A0', 'B0', ... 'Z0', 'A1', 'B1', ...

next_name(*self*) → str

randomize_room_name(*self*, *room: Room*, **args*) → None

Randomize the name of a *Room*

class donjuan.room_randomizer.**RoomPositionRandomizer**

Bases: *donjuan.randomizer.Randomizer*

Randomly shift a room, assuming its left edge is at column 0 and it's top edge is at row 0.

randomize_room_position(*self*, *room: Room*, *dungeon: Dungeon*) → None

Parameters

- **room** (Room) – room to move around
- **dungeon** (Dungeon) – dungeon to move the room around in

class donjuan.room_randomizer.**RoomEntrancesRandomizer**(*max_attempts: int = 100*)
Bases: *donjuan.randomizer.Randomizer*

Randomizes the number of entrances on a room. The number is picked to be the square root of the number of cells in the room divided by 2 plus 1 (N) plus a uniform random integer from 0 to N.

gen_num_entrances(*self*, *cells: Set[Cell]*) → int

randomize_room_entrances(*self*, *room: Room*, **args*) → None
Randomly open edges of cells in a *Room*. The cells in the room must already be linked to edges in a *Grid*. See emplace_rooms().

Note: This algorithm does not allow for a cell in a room to have two entrances.

Parameters **room** (Room) – room to try to create entrances for

donjuan.space

Module Contents

Classes

Space	A space is a section of a dungeon composed of `Cell`s. This object

class donjuan.space.**Space**(*cells: Optional[Set[Cell]] = None*, *name: Union[int, str] = ''*)
Bases: abc.ABC

A space is a section of a dungeon composed of *Cell`s. This object contains these cells in a ``set`* under the property *cells*. It also has a *name* and knows about any entrances to the room (a list of *Edge* objects) via the entrances property.

Parameters

- **cells** (*Optional[Set[Cell]]*) – cells that make up this space
- **name** (*Union[int, str], optional*) – defaults to '', the room name

assign_space_to_cells(*self*) → None
Set the space attribute for each *Cell* to self.

reset_cell_coordinates(*self*) → None

property cells(*self*) → Set[Cell]

property cell_coordinates(*self*) → Set[Tuple[int, int]]

property name(*self*) → Union[int, str]

add_cells(*self*, *cells: Sequence[Cell]*) → None
Add cells to the set of cells in this space. Cells are added to both the *cells* set and the cell coordinates to the *cell_coordinates* set.

Parameters **cells** (*Sequence[Cell]*) – any iterable collection of cells

overlaps (*self*, *other:* Space) → bool
Compare the cells of this space to the other space to determine whether they overlap or not. Note, this algorithm is `O(N)` where `N` is the number of cells in this space, since set lookup is `O(1)`.

Parameters **other** (Space) – other space to check against

Returns `True` if they overlap, `False` if not

set_name (*self*, *name: Union[int, str]*) → None

shift_vertical (*self*, *n: int*) → None
Change the `y` coordinates of all *cells* by `n`.

Parameters **n** (*int*) – number to increment vertical position of cells

shift_horizontal (*self*, *n: int*) → None
Change the `x` coordinates of all *cells* by `n`.

Parameters **n** (*int*) – number to increment horizontal position of cells

3.1.2 Package Contents

Classes

Cell	The cell represents a single 'unit' in the map. The attributes on the cell
HexCell	A cell for a hexagonal grid.
SquareCell	A cell for a square grid.
Archway	An archway to walk through.
Door	A generic door.
DoorSpace	Abstract base class for different kinds of doors. Door spaces can have many
Portcullis	You can look but can't touch!
Dungeon	
DungeonRandomizer	Randomize a dungeon by first creating rooms and then applying
Edge	An edge sits between two `Cell`s.
Grid	Abstract base class for a grid of cells. The underlying grid can either
HexGrid	Rectangular grid of hexagonal cells. In a hex grid, the cell positions
SquareGrid	Rectangular grid of square cells. In a square grid, the cell positions are
Hallway	A hallway in a dungeon. It has a start and end cell.
RandomFilled	Randomly set the `filled` attribute of cells.
Randomizer	Class for randomizing features of a dungeon.
BaseRenderer	Base class for rendering dungeons into images.
Renderer	Default renderer for rendering dungeons into PNG files using *matplotlib*.
Room	A room in a dungeon. A room has is a named `Space`.
AlphaNumRoomName	Simple room name randomizer that names rooms as alphabetical letters.

continues on next page

Table 14 – continued from previous page

RoomEntrancesRandomizer	Randomizes the number of entrances on a room. The number is picked to be
RoomPositionRandomizer	Randomly shift a room, assuming its left edge is at column 0 and it's top
RoomSizeRandomizer	Randomize the size of a *Room*.
Space	A space is a section of a dungeon composed of `Cell`s. This object

donjuan.**__version__ = 0.0.3**

donjuan.**__docs__ = Package for generating dungeons.**

class donjuan.**Cell**(*filled: bool = False*, *door_space: Optional[DoorSpace] = None*, *contents: Optional[List[Any]] = None*, *coordinates: Optional[Tuple[int, int]] = None*, *space: Optional['Space'] = None*, *edges: Optional[List['Edge']] = None*)

Bases: abc.ABC

The cell represents a single 'unit' in the map. The attributes on the cell define what exists in that unit. This includes the terrain, doors, walls, lights, other room details etc.

set_coordinates(*self*, *y: int*, *x: int*) → None

set_edges(*self*, *edges: List['Edge']*) → None

set_space(*self*, *space: Type['Space']*) → None

set_x(*self*, *x: int*) → None

set_y(*self*, *y: int*) → None

property coordinates(*self*) → Tuple[int, int]

property edges(*self*) → List['Edge']

property space(*self*) → Optional[Type['Space']]

Space this cell is a part of.

property x(*self*) → int

property y(*self*) → int

property n_sides(*self*) → int

class donjuan.**HexCell**(*filled: bool = False*, *door_space: Optional[DoorSpace] = None*, *contents: Optional[List[Any]] = None*, *coordinates: Optional[Tuple[int, int]] = None*)

Bases: *donjuan.cell.Cell*

A cell for a hexagonal grid.

Parameters

- **filled** (*bool, optional*) – flag indicating whether the cell is filled (default False)
- **door_space** (*Optional[DoorSpace]*) – kind of doorway in this cell
- **contents** (*Optional[List[Any]]*) – things in this cell

_n_sides = 6

class donjuan.**SquareCell**(*filled: bool = False*, *door_space: Optional[DoorSpace] = None*, *contents: Optional[List[Any]] = None*, *coordinates: Optional[Tuple[int, int]] = None*)

Bases: *donjuan.cell.Cell*

A cell for a square grid.

Parameters

- **filled** (*bool, optional*) – flag indicating whether the cell is filled (default `False`)
- **door_space** (*Optional[DoorSpace]*) – kind of doorway in this cell
- **contents** (*Optional[List[Any]]*) – things in this cell

_n_sides = 4

class donjuan.**Archway**(*material: str = 'stone'*, *blocked: bool = False*, *broken: bool = False*, *secret: bool = False*)

Bases: *donjuan.door_space.DoorSpace*

An archway to walk through.

class donjuan.**Door**(*secret: bool = False*, *locked: bool = False*, *closed: bool = True*, *jammed: bool = False*, *blocked: bool = False*, *broken: bool = False*, *material: str = 'wood'*)

Bases: *donjuan.door_space.DoorSpace*

A generic door.

class donjuan.**DoorSpace**(*secret: bool*, *locked: bool*, *closed: bool*, *jammed: bool*, *blocked: bool*, *broken: bool*, *material: str*, *name: str*)

Bases: abc.ABC

Abstract base class for different kinds of doors. Door spaces can have many properties, like if they are locked or blocked etc. To facilitate this logic in the generative process, these are encompassed in the attributes of a `DoorSpace`.

__slots__ = ['locked', 'closed', 'jammed', 'blocked', 'secret', 'broken', 'material',

__str__(*self*)

Return str(self).

class donjuan.**Portcullis**(*locked: bool = False*, *closed: bool = True*, *jammed: bool = False*, *broken: bool = False*, *material: str = 'metal'*)

Bases: *donjuan.door_space.DoorSpace*

You can look but can't touch!

class donjuan.**Dungeon**(*n_rows: Optional[int] = 5*, *n_cols: Optional[int] = 5*, *grid: Optional[Grid] = None*, *rooms: Optional[Dict[str, Room]] = None*, *hallways: Optional[Dict[str, Hallway]] = None*, *randomizers: Optional[List['Randomizer']] = None*)

add_room(*self*, *room: Room*) → None

add_hallway(*self*, *hallway: Hallway*) → None

property grid(*self*) → *Grid*

property hallways(*self*) → Dict[str, *Hallway*]

property n_cols(*self*) → int

property n_rows(*self*) → int

property randomizers(*self*) → List['Randomizer']

property rooms(*self*) → Dict[str, *Room*]

randomize(*self*) → None

For each item in *randomizers*, run the *Randomizer.randomize_dungeon()* method on this dungeon.

emplace_rooms(*self*) → None
Replace the cells in the *grid* with the cells of the *rooms*.

emplace_space(*self*, *space:* Space) → None
Replace the cells in the *grid* with the cells of the *Space*, and automatically link them with the *Edge`s in the `Grid*.

Parameters **space** (Space) – room to emplace in the *grid*

class donjuan.**DungeonRandomizer**(*room_entrance_randomizer: Optional[Randomizer] = None*, *room_size_randomizer: Optional[Randomizer] = None*, *room_name_randomizer: Optional[Randomizer] = None*, *room_position_randomizer: Optional[Randomizer] = None*, *max_num_rooms: Optional[int] = None*, *max_room_attempts: int = 100*)
Bases: *donjuan.randomizer.Randomizer*

Randomize a dungeon by first creating rooms and then applying room size, name, and position randomizers to sequentially generated rooms.

Parameters

- **room_entrance_randomizer** (*Optional[Randomizer]*) – randomizer for the entrances of a room. If None then default to a RoomEntrancesRandomizer.
- **room_size_randomizer** (*Optional[Randomizer]*) – randomizer for the room size. It must have a 'max_size' attribute. If None then default to a RoomSizeRandomizer.
- **room_name_randomizer** (*Optional[Randomizer]*) – randomizer for the room name. If None default to a AlphaNumRoomName.
- **room_position_randomizer** (*Optional[Randomizer]*) – randomizer for the room position. If None default to a RoomPositionRandomizer.
- **max_num_rooms** (*Optional[int]*) – maximum number of rooms to draw, if ``None` then default to the max_room_attempts. See DungeonRoomRandomizer.get_number_of_rooms() for details.
- **max_room_attempts** (*int, optional*) – default is 100. Maximum number of attempts to generate rooms.

get_number_of_rooms(*self*, *dungeon_n_rows: int*, *dungeon_n_cols: int*) → int
Randomly determine the number of rooms based on the size of the incoming grid or the max_num_rooms attribute, whichever is less.

Parameters

- **dungeon_n_rows** (*int*) – number of rows
- **dungeon_n_cols** (*int*) – number of columns

randomize_dungeon(*self*, *dungeon:* Dungeon) → None
Randomly put rooms in the dungeon.

Parameters **dungeon** (Dungeon) – dungeon to randomize the rooms of

class donjuan.**Edge**(*cell1: Optional[Cell] = None*, *cell2: Optional[Cell] = None*, *has_door: bool = False*)
An edge sits between two `Cell`s.

Parameters

- **cell1** (*Optional[Cell]*) – cell on one side of the edge

- **cell2** (*Optional[Cell]*) – cell on the other side of the edge
- **has_door** (*bool, optional*) – default `False`, indicates whether this object has a door

property cell1(*self*) → Optional[*Cell*]

property cell2(*self*) → Optional[*Cell*]

set_cell1(*self*, *cell:* Cell) → None

set_cell2(*self*, *cell:* Cell) → None

property is_wall(*self*) → bool

class donjuan.**Grid**(*n_rows: int*, *n_cols: int*, *cells: Optional[List[List[*Cell*]]] = None*, *edges: Optional[List[List[List[*Edge*]]]] = None*)

Bases: `abc.ABC`

Abstract base class for a grid of cells. The underlying grid can either be square or hexagonal.

get_filled_grid(*self*) → List[List[bool]]

Obtain a 2D array of boolean values representing the `filled` state of the cells attached to the grid.

property n_rows(*self*) → int

property n_cols(*self*) → int

property cells(*self*) → List[List[*Cell*]]

property edges(*self*) → List[List[List[*Edge*]]]

reset_cell_coordinates(*self*) → None

Helper function that sets the coordinates of the cells in the grid to their index values.

check_edges(*self*, *edges: Optional[List[List[List[*Edge*]]]]*) → None

Check the dimensions of the *edges*.

init_edges(*self*) → List[List[List[*Edge*]]]

link_edges_to_cells(*self*) → None

For an *Edge*, the *Edge.cell1* always points to either the left or upper *Cell*. The *Edge.cell2* always points to the right or the bottom.

abstract link_cells_to_edges(*self*) → None

class donjuan.**HexGrid**(*n_rows: int*, *n_cols: int*, *cells: Optional[List[List[*HexCell*]]] = None*)

Bases: *donjuan.grid.Grid*

Rectangular grid of hexagonal cells. In a hex grid, the cell positions are integers, with odd rows being "offset" by half a cell size when rendered.

cell_type

link_cells_to_edges(*self*) → None

class donjuan.**SquareGrid**(*n_rows: int*, *n_cols: int*, *cells: Optional[List[List[*SquareCell*]]] = None*)

Bases: *donjuan.grid.Grid*

Rectangular grid of square cells. In a square grid, the cell positions are integers.

cell_type

link_cells_to_edges(*self*) → None

class donjuan.**Hallway**(*ordered_cells: Optional[List[*Cell*]] = None*, *name: Union[int, str] = ''*)
Bases: *donjuan.space.Space*

A hallway in a dungeon. It has a start and end cell.

Parameters

- **ordered_cells** (*Optional[Sequence[Cell]]*) – ordered list of cells, where the order defines the path of the hallway. If `None` defaults to an empty list.
- **name** (*Union[int, str], optional*) – defaults to '', the name of the hallway

property ordered_cells(*self*) → List[*Cell*]
Cells that make up the path of the hallway. Does not contain extra cells that may be associated with this object (i.e. those off of the "path"). For the set of all cells, use `cells`.

property end_cell(*self*) → *Cell*

property start_cell(*self*) → *Cell*

append_ordered_cell_list(*self*, *cells: List[*Cell*]*) → None
Append cells in order to the hallway. To add a cell to the hallway without including it in the hallways path, use `add_cells()`.

Parameters **cells** – (List[Cell]): cells to append to the hallway

get_coordinate_path(*self*) → List[Tuple[int, int]]
Get the coordinates of the cells that make up this path this hallway takes. Does not contain coordinates of extra cells on this object.

Returns coordinates of the hallway path

class donjuan.**RandomFilled**
Bases: *donjuan.randomizer.Randomizer*

Randomly set the `filled` attribute of cells.

randomize_cell(*self*, *cell:* Cell) → None
Randomly fill the cell with probability 50%

randomize_grid(*self*, *grid:* Grid) → None
Randomly fill all cells of the grid individually

class donjuan.**Randomizer**
Class for randomizing features of a dungeon.

randomize_cell(*self*, *cell:* Cell) → None
Randomize properties of the *Cell*

randomize_dungeon(*self*, *dungeon:* Dungeon) → None
Randomize properties of the *Dungeon*

randomize_grid(*self*, *grid:* Grid) → None
Randomize properties of the *Grid*

randomize_hallway(*self*, *hallway:* Hallway) → None
Randomize properties of the *Hallway*

randomize_room_entrances(*self*, *room:* Room, **args*) → None
Randomize the entrances of the *Room*

randomize_room_size(*self*, *room:* Room, **args*) → None
Randomize the size of the *Room*

randomize_room_name(*self*, *room:* Room, **args*) → None
Randomize the name of a *Room*

randomize_room_position(*self*, *room:* Room, **args*) → None
Randomize the position of a *Room*

classmethod seed(*cls*, *seed: Optional[int] = None*) → None

Parameters **seed** (*Optional[int]*) – seed passed to `random.seed()`

class donjuan.**BaseRenderer**(*scale: float*)
Bases: `abc.ABC`

Base class for rendering dungeons into images.

property scale(*self*) → float
The scale size of a single *Cell*. The meaning differs depending on the subclass.

abstract render(*self*, *dungeon:* Dungeon) → None

class donjuan.**Renderer**(*scale: float = 1.0*)
Bases: *donjuan.renderer.BaseRenderer*

Default renderer for rendering dungeons into PNG files using *matplotlib*.

Parameters **scale** (*float, optional*) – size of a single cell in inches (default is 1 inch).

render(*self*, *dungeon:* Dungeon, *file_path: str = 'rendered_dungeon.png'*, *dpi: int = 200*, *save: bool = True*) → Tuple
Render the dungeon.

Parameters

- **dungeon** (Dungeon) – dungeon to render
- **file_path** (*str, optional*) – path to save the dungeon at (default is *rendered_dungeon.png*)
- **dpi** (*int, optional*) – dots per inch used to save (default is 200)
- **save** (*bool, optional*) – flag indicating whether to save the dungeon with `matplotlib.pyplot.savefig()`

class donjuan.**Room**(*cells: Optional[Set[*Cell*]] = None*, *name: Union[int, str] = ''*)
Bases: *donjuan.space.Space*

A room in a dungeon. A room has is a named `Space`.

class donjuan.**AlphaNumRoomName**
Bases: *donjuan.randomizer.Randomizer*

Simple room name randomizer that names rooms as alphabetical letters. followed by a number. Rooms are sequentially named 'A0', 'B0', … 'Z0', 'A1', 'B1', …

next_name(*self*) → str

randomize_room_name(*self*, *room:* Room, **args*) → None
Randomize the name of a *Room*

class donjuan.**RoomEntrancesRandomizer**(*max_attempts: int = 100*)
Bases: *donjuan.randomizer.Randomizer*

Randomizes the number of entrances on a room. The number is picked to be the square root of the number of cells in the room divided by 2 plus 1 (`N`) plus a uniform random integer from 0 to `N`.

gen_num_entrances(*self*, *cells: Set[*Cell*]*) → int

randomize_room_entrances(*self*, *room:* Room, **args*) → None
Randomly open edges of cells in a *Room*. The cells in the room must already be linked to edges in a *Grid*. See `emplace_rooms()`.

Note: This algorithm does not allow for a cell in a room to have two entrances.

Parameters **room** (Room) – room to try to create entrances for

class donjuan.**RoomPositionRandomizer**
Bases: *donjuan.randomizer.Randomizer*

Randomly shift a room, assuming its left edge is at column 0 and it's top edge is at row 0.

randomize_room_position(*self*, *room:* Room, *dungeon:* Dungeon) → None

Parameters

- **room** (Room) – room to move around
- **dungeon** (Dungeon) – dungeon to move the room around in

class donjuan.**RoomSizeRandomizer**(*min_size: int = 2*, *max_size: int = 4*, *cell_type: Type[*Cell*] = SquareCell*)
Bases: *donjuan.randomizer.Randomizer*

Randomize the size of a *Room*.

randomize_room_size(*self*, *room:* Room) → None
Randomly determine the size of the room, and set the cells of the room to a 2D array of unfilled cells of that size.

class donjuan.**Space**(*cells: Optional[Set[*Cell*]] = None*, *name: Union[int, str] = ''*)
Bases: abc.ABC

A space is a section of a dungeon composed of *Cell`s. This object contains these cells in a ``set`* under the property *cells*. It also has a *name* and knows about any entrances to the room (a list of *Edge* objects) via the `entrances` property.

Parameters

- **cells** (*Optional[Set[Cell]]*) – cells that make up this space
- **name** (*Union[int, str], optional*) – defaults to '', the room name

assign_space_to_cells(*self*) → None
Set the *space* attribute for each *Cell* to `self`.

reset_cell_coordinates(*self*) → None

property cells(*self*) → Set[*Cell*]

property cell_coordinates(*self*) → Set[Tuple[int, int]]

property name(*self*) → Union[int, str]

add_cells(*self*, *cells: Sequence[*Cell*]*) → None
Add cells to the set of cells in this space. Cells are added to both the *cells* set and the cell coordinates to the *cell_coordinates* set.

Parameters **cells** (*Sequence[Cell]*) – any iterable collection of cells

overlaps (*self*, *other:* Space) → bool
Compare the cells of this space to the other space to determine whether they overlap or not. Note, this algorithm is `O(N)` where `N` is the number of cells in this space, since set lookup is `O(1)`.

Parameters **other** (Space) – other space to check against

Returns `True` if they overlap, `False` if not

set_name (*self*, *name: Union[int, str]*) → None

shift_vertical (*self*, *n: int*) → None
Change the `y` coordinates of all *cells* by `n`.

Parameters **n** (*int*) – number to increment vertical position of cells

shift_horizontal (*self*, *n: int*) → None
Change the `x` coordinates of all *cells* by `n`.

Parameters **n** (*int*) – number to increment horizontal position of cells

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

X

Y