Text adventure builder Documentation

Release 0.1

Erik Nyquist

Apr 06, 2020

Contents

Contents

text_game_maker is a framework for building simple text-based games. Supports Python 2 and 3 on Linux and Windows (tested on Debian and Windows 10).

CHAPTER 1

Features

- Builder API pattern that implements a 2D tile-based grid system for building maps and creating game objects with a few lines of Python.
- Flexible object model for defining game objects and NPCs. Many useful game objects and behaviours are pre-defined so you can start making a game straight away, but these classes are easy to extend if you need to implement your own custom game objects.
- NPCs that you can actually speak to, like a chatbot. `text_game_maker` NPCs allow you to define custom responses and contexts for discussions using regular expressions, so you can build an NPC which responds with some contextual awareness like a chatbot.
- Arbitrary saving/loading of game states. All game objects can be serialized to, or loaded from, JSON data.
- Flexible and extensible parser. The parser that handles game input already has a lot of useful commands and behaviours built-in. However, any of the built-in commands can be disabled if necessary, and the parser provides methods that allow new commands to be defined.
- Configurable input/output streams. `text_game_maker` allows custom handlers to be set for handling user input, and for displaying game output. By default, `text_game_maker` will use a `prompt-toolkit` session that interacts with `stdin` and `stdout` of the process it is running in. Defining custom input/output handlers allows you to run your game, for example, as a slack bot instead (see `scripts/text-game-slackbot-runner.py`).
- `text_game_maker` has a music system that allows polyphonic music (not fancy, just simple tones that sound like an old Nokia ringtone, but with full polyphony) to be written as simple text files that can be loaded and played during the game (see PTTTL). `pygame` is used for playback of the raw audio samples.
- Much more... check out the API documentation!

CHAPTER 2

Getting started

2.1 Install

```
pip install text_game_maker
```

2.2 Run the example map

```
python -m text_game_maker.example
```

CHAPTER 3

Writing maps

1. Write a class that extends the text_game_maker.utils.runner.MapRunner class and implements the `build_map` (and optionally `build_parser`) methods. See `example-map/example_map.py` and the API documentation for reference.
2. Run `text_game_maker.runner` with the name of the `.py` file containing your MapRunner class, e.g.:

   ```
   python -m text_game_maker.runner mymaprunner.py
   ```

   In this example, `text_game_maker.runner` will import the `mymaprunner.py` file and run the first instance it finds of a class which is a subclass of text_game_maker.utils.runner.MapRunner

To run a MapRunner class as a slackbot to play your game over slack, the procedure is the same, except use the `text_game_maker.slackbot_runner` script, e.g.:

```
python -m text_game_maker.slackbot_runner mymaprunner.py
```

Note that when using the slackbot runner you must have already created a bot user in your slack workspace, and the API token for the bot user must be available in an environment variable named `SLACK_BOT_TOKEN`.

CHAPTER 4

API Documentation

text-game-maker.readthedocs.io

4.1 text_game_maker

4.1.1 text_game_maker package

Subpackages

text_game_maker.audio package

Submodules

text_game_maker.audio.audio.**init**(*frequency=44100*, *samplewidth=-16*, *numchannels=1*, *buffersize=1024*)

Initialize game audio and load PTTTL files for default sounds (called automatically by text_game_maker.builder.map_builder.MapBuilder.run_game)

Parameters

- **frequency** (*int*) – frequency in HZ
- **samplewidth** (*int*) – sample size in bits
- **numchannels** (*int*) – number of audio channels
- **buffersize** (*int*) – size in bytes of the buffer to be used for playing audio samples

text_game_maker.audio.audio.**load_file**(*filename*, *sound_id=None*)

Read a PTTTL file, convert to PCM samples and save for playback

Parameters

- **filename** (*str*) – filename for PTTTL file to read

- **sound_id** – key used to retrieve sound for playback (if None, filename is used)

text_game_maker.audio.audio.**play_sound**(*sound_id*)
Play a loaded sound

Parameters **sound_id** – key for sound to play

text_game_maker.audio.audio.**quit**()
Disable audio after it has been initialized

text_game_maker.audio.audio.**wait**()
Wait until currently playing sound is finished playing (if any)

text_game_maker.builder package

Submodules

class text_game_maker.builder.map_builder.**MapBuilder**(*parser*)
Bases: object

Base class for building a tile-based map

__init__(*parser*)
Initialises a MapBuilder instance.

Parameters **text_game_maker.parser.parser.CommandParser** – command parser

add_door(*prefix*, *name*, *direction*, *doorclass=<class 'text_game_maker.tile.tile.LockedDoor'>*, *door_id=None*)
Add a locked door that blocks the player from exiting the current room

Parameters

- **prefix** (*str*) – prefix for door name, e.g. "a"
- **name** (*str*) – door name, e.g. "locked door"
- **direction** (*str*) – direction to locked door from current tile, e.g. "north"
- **doorclass** – class object to instantiate for door
- **door_id** – unique ID to represent door in save files

add_enter_event_handler(*handler*)
Add a handler to be invoked when player enters the current tile

Parameters **handler** – handler of the form `handler(player, src, dest)`, where `player` is the `text_game_maker.player.player.Player` instance, `src` is the `text_game_maker.tile.tile.Tile` instance that the player just exited, and `dest` is the `text_game_maker.tile.tile.Tile` instance the player has just entered

add_exit_event_handler(*handler*)
Add a handler to be invoked when player exits the current tile

Parameters **handler** – handler of the form `handler(player, src, dest)`, where `player` is the `text_game_maker.player.player.Player` instance, `src` is the `text_game_maker.tile.tile.Tile` instance that the player just exited, and `dest` is the `text_game_maker.tile.tile.Tile` instance the player has just entered

add_item(*item*)
Add item to current tile (see text_game_maker.tile.tile.Tile.add_item)

Parameters **item** (*text_game_maker.game_objects.base.Item*) – the item to add

add_items(*items*)
Add multiple items to current tile

Parameters **items** (*[text_game_maker.game_objects.item.Item]*) – list of items to add

add_keypad_door(*prefix*, *name*, *direction*, *code*, *doorclass=<class 'text_game_maker.tile.tile.LockedDoorWithKeypad'>*, *door_id=None*, *prompt=None*)
Add a locked door that blocks the player from exiting the current room, and requires a specific code to be entered on the keypad to unlock it.

Parameters

- **prefix** (*str*) – prefix for door name, e.g. "a"
- **name** (*str*) – door name, e.g. "locked door"
- **direction** (*str*) – direction to locked door from current tile, e.g. "north"
- **code** (*int*) – keypad code required to unlock door
- **doorclass** – class object to instantiate for door
- **door_id** – unique ID to represent door in save files

add_new_game_start_event_handler(*handler*)
Add a handler to be invoked when a new game is started

Parameters **handler** – handler to be invoked when a new game is started. Handler should be of the form `handler(player)` where `player` is the `text_game_maker.player.player.Player` instance

add_person(*person*)
Add person to current tile (see text_game_maker.tile.tile.Tile.add_person)

Parameters **person** (text_game_maker.game_objects.person.Person) – person to add

clear_enter_event_handler(*handler*)
Clear specific enter event handler attached to the current tile

Parameters **handler** – enter event handler that was previously added to the current tile

clear_enter_event_handlers()
Clear all enter event handler attached to the current tile

clear_exit_event_handler(*handler*)
Clear specific exit event handler attached to the current tile

Parameters **handler** – exit event handler that was previously added to the current tile

clear_exit_event_handlers()
Clear all exit event handler attached to the current tile

inject_input(*data*)
Inject data into the game's input stream (as if player had typed it)

Parameters **data** (*str*) – string of text to inject

load_map_data(*filename*)
Load a map file saved from the map editor GUI

Parameters **filename** (*str*) – name of map editor save file to load

move_east(*num=1*, *name=None*, *description=None*, *tileclass=<class 'text_game_maker.tile.tile.Tile'>*)
Move east by one or more tiles. On each move, if a tile does not already exist at the current position, a new tile will be created and set as the current tile to build on. If a tile already exists at the current position, it will be set to the current tile and no new tile will be created.

Parameters

- **name** (*str*) – short description of tile
- **description** (*str*) – long description of tile
- **tileclass** – class object to create tile from
- **num** (*int*) – distance ot move in tiles

move_north(*num=1*, *name=None*, *description=None*, *tileclass=<class 'text_game_maker.tile.tile.Tile'>*)
Move north by one or more tiles. On each move, if a tile does not already exist at the current position, a new tile will be created and set as the current tile to build on. If a tile already exists at the current position, it will be set to the current tile and no new tile will be created.

Parameters

- **name** (*str*) – short description of tile
- **description** (*str*) – long description of tile
- **tileclass** – class object to create tile from
- **num** (*int*) – distance ot move in tiles

move_south(*num=1*, *name=None*, *description=None*, *tileclass=<class 'text_game_maker.tile.tile.Tile'>*)
Move south by one or more tiles. On each move, if a tile does not already exist at the current position, a new tile will be created and set as the current tile to build on. If a tile already exists at the current position, it will be set to the current tile and no new tile will be created.

Parameters

- **name** (*str*) – short description of tile
- **description** (*str*) – long description of tile
- **tileclass** – class object to create tile from
- **num** (*int*) – distance ot move in tiles

move_west(*num=1*, *name=None*, *description=None*, *tileclass=<class 'text_game_maker.tile.tile.Tile'>*)
Move west by one or more tiles. On each move, if a tile does not already exist at the current position, a new tile will be created and set as the current tile to build on. If a tile already exists at the current position, it will be set to the current tile and no new tile will be created.

Parameters

- **name** (*str*) – short description of tile
- **description** (*str*) – long description of tile
- **tileclass** – class object to create tile from
- **num** (*int*) – distance ot move in tiles

run_game()
Start running the game

set_current_tile(*tile_id*)
Set the current tile to build on by tile ID

Parameters **tile_id** (*str*) – tile ID of tile to set as current tile

set_dark(*value*)
Set whether this tile is dark or not. Dark tiles require player to equip a light source

Parameters **value** (*bool*) – True for dark, False for not dark

set_description(*desc*)
Add long description for current tile (see text_game_maker.tile.tile.Tile.set_description)

Parameters **desc** (*str*) – description text

set_first_visit_message(*message*)
Add text to be printed only once, on the player's first visit to the current tile

Parameters **messsage** (*str*) – message to show on first player visit

set_first_visit_message_in_dark(*value*)
Defines whether the current tile shows a first visit message in the dark. if False, first visit message for current tile will be shown the first player is on the current tile and has a light source.

Parameters **value** (*bool*) – value to set

set_ground_material(*material*)
Set the material type of the ground on this tile :param Material material: material type

set_ground_smell(*text*)
Set the text that will be printed when player types 'smell ground' or equivalent on this tile

Parameters **text** (*str*) – text to be printed on smell ground command

set_input_prompt(*prompt*)
Set the message to print when prompting a player for game input

Parameters **prompt** (*str*) – message to print

set_name(*name*)
Add short description for current tile (see text_game_maker.tile.tile.Tile.set_tile_id)

Parameters **desc** (*str*) – description text

set_name_from_east(*name*)
Set the name that will be shown when player looks at the current tile from an adjacent tile to the east

Parameters **desc** (*str*) – description text

set_name_from_north(*name*)
Set the name that will be shown when player looks at the current tile from an adjacent tile to the north

Parameters **desc** (*str*) – description text

set_name_from_south(*name*)
Set the name that will be shown when player looks at the current tile from an adjacent tile to the south

Parameters **desc** (*str*) – description text

set_name_from_west(*name*)
Set the name that will be shown when player looks at the current tile from an adjacent tile to the west

Parameters **desc** (*str*) – description text

set_on_game_run(*callback*)
Set callback function to be invoked when player starts a new game (i.e. not from a save file). Callback function should accept one parameter:

def callback(player): pass

Callback parameters:

- *player* (text_game_maker.player.player.Player): player instance

Parameters **callback** – callback function

set_smell(*text*)
Set the text that will be printed when player types 'smell' or equivalent on this tile

Parameters **text** (*str*) – text to be printed on smell command

set_tile_id(*tile_id*)
Set tile ID for current tile (see text_game_maker.tile.tile.Tile.set_tile_id)

Parameters **tile_id** – tile ID

start_map(*name=u"*, *description=u"*)
Start building the map; create the first tile

Parameters

- **name** (*str*) – short name for starting Tile
- **description** (*str*) – short name for starting Tile

text_game_maker.builder.map_builder.**add_format_tokens**()

text_game_maker.builder.map_builder.**get_instance**()

text_game_maker.chatbot_utils package

Submodules

class text_game_maker.chatbot_utils.redict.**ReDict**(**args*, ***kwargs*)
Bases: dict

Special dictionary which expects values to be *set* with regular expressions (REs) as keys, and expects values to be retreived using input text for an RE as keys. The value corresponding to the regular expression which matches the input text will be returned. In the case where the input text matches multiple REs, one of the matching values will be returned, but precisely which one is undefined.

Example usage:

```
>>> d = ReDict()
>>> d['hello( world(!)*)?'] = 1
>>> d['regex|dict key'] = 2
>>> d['hello']
1
>>> d['hello world!!!!']
1
>>> d['regex']
2
>>> d['dict key']
2
```

__init__(**args*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

clear()
Clear all key/value pairs stored in this dict

compile()
Compile all regular expressions in the dictionary

copy()
Create a new ReDict instance and copy all items in this dict into the new instance

Returns new ReDict instance containing copied data

Return type *ReDict*

dump_to_dict()
Dump all pattern/value pairs to a regular dict, where the regular expressions are the keys

Returns dict of pattern/value pairs

Return type dict

groups()
Return tuple of all subgroups from the last regex match performed when fetching an item, as returned by re.MatchObject.groups()

Returns tuple of subgroups from last match

Return type tuple

items()
Return all values stored in this dict

Returns list of values

Return type list

iteritems()
Returns a generator to get all key/value pairs stored in this dict

Returns generator to get pattern/value pairs

keys()
Return all keys stored in this dict

Returns list of keys

Return type list

load_from_dict(*data*)
Load pattern/value pairs from a regular dict. This overwrites any existing pattern/value pairs

Parameters **data** (*dict*) – pattern/value pairs to load

pop(*text*)
Return and delete the first value associated with a pattern matching 'text'

Parameters **text** (*str*) – text to match against

Returns value associated with pattern matching 'text' (if any)

update(*other*)
Add items from 'other' into this dict

Parameters **other** (ReDict) – dict containing items to copy

values()
Return all values stored in this dict

Returns list of values

Return type list

class text_game_maker.chatbot_utils.responder.**Context**(*lists=None*)
Bases: object

Class representing a "discussion" context, allowing for a Responder that responds with contextual awareness

__init__(*lists=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

add_chained_phrases(**pattern_response_pairs*)
Add multiple chained pattern/response pairs. A chain defines a sequence of pattern/response pairs that are expected in the order that they occur in the passed arguments to this method. Whenever a Responder is inside a context and input matching the first pattern/response pair in a chain is seen, the Responder will continually expect the next pattern in the current chain until another chain or another context is entered. When the last pattern in the chain is reached, Responders will continue expecting this pattern until another chain or context is entered.

Parameters **pattern_response_pairs** – one or more pattern/response pairs, where a pattern/response pair is a tuple of the form `(regexs, value)`, where `regexs` is a regular expression or list of regular expressions and `value` is an arbitrary object

add_context(*context*)
Add context that can only be entered when already in this context

Parameters **context** (text_game_maker.chatbot_utils.responder.Context) – context instance to add

add_contexts(**contexts*)
Add one or more context instances to this context

Parameters **contexts** (text_game_maker.chatbot_utils.responder.Context) – context instances to add

add_entry_phrase(*patterns*, *response*)
Add a pattern/response pair to be used as an entry point for this context. If input matching matching one of the patterns passed here is seen, Responders will return the corresponding response object and enter the context.

Parameters

- **patterns** – regular expression or list of regular expressions. If the input passed to `get_response` matches one of these patterns, then the object passed here as `response` will be returned.
- **response** (*object*) – object to return from `get_response` if the passed input matches one of the regular expressions passed here as `response`.

add_entry_phrases(**pattern_response_pairs*)
Add one or more pattern/response pairs to be used as entry points for this context. If input matching matching one of the patterns passed here is seen, Responders will return the corresponding response object and enter the context.

Parameters **pattern_response_pairs** – one or more pattern/response pairs, where a pattern/response pair is a tuple of the form `(regexs, value)`, where `regexs` is a regular expression or list of regular expressions and `value` is an arbitrary object

add_response(*patterns*, *response*)
Add a pattern/response pair that will be only be recognized when a Responder is in this context

Parameters

- **patterns** – regular expression or list of regular expressions. If the input passed to `get_response` matches one of these patterns, then the object passed here as `response` will be returned.
- **response** (*object*) – object to return from `get_response` if the passed input matches one of the regular expressions passed here as `response`.

add_responses(**pattern_response_pairs*)
Add one more more pattern/response pairs that will be only be recognized when a Responder is in this context

Parameters **pattern_response_pairs** – one or more pattern/response pairs, where a pattern/response pair is a tuple of the form `(regexs, value)`, where `regexs` is a regular expression or list of regular expressions and `value` is an arbitrary object

compile()
Compile all regular expressions contained in this context so they are ready for immediate matching

get_response(*text*)
Find a response object associated with a pattern in this context that matches 'text', and return it (if any). If no matching patterns can be found, 'text' itself will be returned.

Parameters **text** (*str*) – input text to check for matching patterns against

Returns tuple of the form `(response, groups)`. `response` is the response object associated with the matching regular expression, if any, otherwise 'text'. `groups` is a tuple of subgroups from the regular expression match (as returned by re.MatchObject.groups), if any, otherwise None.

class text_game_maker.chatbot_utils.responder.**NoResponse**
Bases: object

class text_game_maker.chatbot_utils.responder.**Responder**
Bases: object

Represents a high-level responder object which can be used to register pattern/response pairs, and can accept input text to retrieve matching response objects

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

add_context(*context*)
Add context instance to this responder

Parameters **context** (text_game_maker.chatbot_utils.responder.Context) – context instance to add

add_contexts(**contexts*)
Add one or more context instances to this responder

Parameters **contexts** (text_game_maker.chatbot_utils.responder.Context) – context instances to add

add_default_response(*response*)
Set response to return when no other matching responses can be found

Parameters **response** – object to return as default response

add_response(*patterns*, *response*)
Add a pattern/response pair that will always be recognized by a Responder, regardless of context

Parameters

- **patterns** (*list*) – list of regular expressions. If the input passed to `get_response` matches one of these patterns, then the object passed here as `response` will be returned.
- **response** (*object*) – object to return from `get_response` if the passed input matches one of the regular expressions passed here as `response`.

add_responses(**pattern_response_pairs*)
Add one or moe pattern/response pairs that will always be recognized by a Responder, regardless of context

Parameters **pattern_response_pairs** – one or more pattern/response pairs, where a pattern/response pair is a tuple of the form `(regexs, value)`, where `regexs` is a regular expression or list of regular expressions and `value` is an arbitrary object

compile()
Compile all regular expressions contained in this responder (including contexts), so they are ready for matching immediately

get_response(*text*)
Find a response object associated with a pattern that matches 'text', and return it (if any). If no matching patterns can be found, 'text' itself will be returned.

Parameters **text** (*str*) – input text to check for matching patterns against

Returns tuple of the form `(response, groups)`. `response` is the response object associated with the matching regular expression, if any, otherwise 'text'. `groups` is a tuple of subgroups from the regular expression match (as returned by re.MatchObject.groups), if any, otherwise None.

text_game_maker.crafting package

Submodules

text_game_maker.crafting.crafting.**add**(*items*, *item*)
Add new craftable item

Parameters

- **items** (*[text_game_maker.game_objects.items.Item]*) – list of ingredients
- **item** (*text_game_maker.game_objects.items.Item*) – new item created by combining ingredients

text_game_maker.crafting.crafting.**can_craft**(*name*)
Check if player has the ability to craft an item by name. Note this function only checks if player has acquired the blueprint to craft an item, and does not care whether the player has ingredients required to craft the item.

Parameters **name** (*str*) – item name

Returns True if player can craft the item, False otherwise

Return type bool

text_game_maker.crafting.crafting.**craft**(*name*, *word*, *player*)
Craft an item by name. Deletes ingredients from player's inventory and places crafted item into player's inventory.

Parameters

- **name** (*str*) – name of the item to craft
- **word** (*str*) – command/action word used by player
- **player** (text_game_maker.player.player.Player) – player instance

Returns crafted item, or None if crafting fails

Return type text_game_maker.game_objects.items.Item

text_game_maker.crafting.crafting.**deserialize**(*data*, *version*)

text_game_maker.crafting.crafting.**help_text**()
Retreive human-readable description of all added craftable items

Returns description of all added craftable items

Return type str

text_game_maker.crafting.crafting.**serialize**()

text_game_maker.event package

Submodules

class text_game_maker.event.event.**Event**
Bases: object

Class to represent a generic event that handlers can be registered for

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

add_handler(*handler*)
Registers a handler to run when this event is generated.

Parameters **handler** – handler to add. Handler should be of the form: handler(*event_args) where event_args is all of the arguments for the event

Returns Event instance

Return type *text_game_maker.event.event.Event*

clear_handler(*handler*)
Unregisters a handler.

Parameters **handler** – the handler that was previously registered

Returns Event instance

Return type *text_game_maker.event.event.Event*

clear_handlers()
Clear any registered handlers for event.

Returns Event instance

Return type *text_game_maker.event.event.Event*

deserialize(*attrs*)

generate(**event_args*)
Generate an event. Runs all registered handlers.

Parameters **event_args** – arguments to pass to event handlers

Returns Event instance

Return type *text_game_maker.event.event.Event*

serialize()

text_game_maker.game_objects package

Submodules

class text_game_maker.game_objects.base.**GameEntity**

Bases: object

Base class for anything that the player can interact with in the game, like usable items, food, people, etc.

Variables

- **inanimate** (*bool*) – defines whether this item is an inanimate object
- **combustible** (*bool*) – defines whether this item can be destroyed by fire
- **scenery** (*bool*) – defines whether this item is scenery; an immovable prop in the scene that should be mentioned before smaller interactive items when describing a room, e.g. furniture, architectural features of the room
- **edible** (*bool*) – defines whether this item is edible
- **alive** (*bool*) – defines whether this item is currently alive
- **energy** (*int*) – defines how much energy player gains by consuming this item
- **damage** (*int*) – defines how much health player loses if damaged by this item
- ***value*** (*int*) – defines how much money player will make by selling this item
- **name** (*str*) – name that this item will be referred to as in the game (e.g. "metal key")
- **prefix** (*str*) – preceding word required when mentioning item, e.g. "a" for "a metal key", and "an" for "an apple"
- **home** (*list*) – list that this Item instance lives inside; required for the deleting/moving items within the game world
- **is_container** (*bool*) – defines whether this item can contain other items
- **is_electricity_source** (*bool*) – defines whether this item is an electricity source
- **requires_electricity** (*bool*) – defines whether this item requires an electricity source to operate
- **is_flame_source** (*bool*) – defines whether this item can be used as a flame source
- **is_light_source** (*bool*) – defines whether this item can be used as a light source
- **material** (Material) – material this item is made of
- **capacity** (*int*) – number of items this item can contain (if container)
- ***items*** (*list*) – items contained inside this item (if container)
- **size** (*int*) – size of this item; containers cannot contain items with a larger size than themselves

- **verb** (*str*) – singluar verb e.g. "the key is on the floor", or plural e.g. "the coins are on the floor"

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

add_item(*item*)
Put an item inside this item

Parameters **item** (text_game_maker.game_objects.base.GameEntity) – item to add

Returns the item that was added

add_items(*items*)
Put multiple items inside this item

Parameters **items** ([text_game_maker.game_objects.base.GameEntity]) – list of items to add

add_migration(*from_version*, *to_version*, *migration_function*)
Add function to migrate a serialized version of this object to a new object model version

Parameters

- **from_version** (*str*) – version to migrate from
- **to_version** (*str*) – version to migrate to
- **migration_function** – function to perform migration

add_to_player_inventory(*player*)
Put this item inside player's inventory. If add_to_player_inventory returns True, execution of the current command will continue normally. If False, execution of the current command will stop immediately.

Parameters **player** (text_game_maker.player.player.Player) – player object

Returns the object that was added

copy()

delete()
Delete the instance of this item from whatever location list it lives in (if any)

full_class_name = u'text_game_maker.game_objects.base.GameEntity'

get_attrs()
Recursively serialize all attributes of this item and any contained items

Returns serializable dict of item and contained items

Return type dict

get_special_attrs()
Serialize any attributes that you want to handle specially here. Any attributes present in the dict returned by this function will not be serialized by the main get_attrs method. Intended for subclasses to override

Returns serializable dict of special attributes

Return type dict

global_skip_attrs = ['home', '_migrations']

matches_name(*name*)
Tests if name matches this item's name. Will accept substrings of a matching name, e.g. Item(name="metal key").matches_name("key") will return True.

Parameters **name** (*str*) – name to compare against this item's name

Returns True if name matches this items name

Return type bool

migrate(*old_version*, *attrs*)
Migrate serialized version of this object to the current object model version

Parameters

- **old_version** (*str*) – object model version to migrate from
- **attrs** – object to migrate as a serialized dict

Returns migrated serialized dict

move(*location*)
Move this item to a different location list

Parameters **location** (*list*) – location list to move item to

Returns item that was moved

on_attack(*player*, *item*)
Called when player attacks this item

Parameters

- **player** (text_game_maker.player.player.Player) – player instance
- **item** (text_game_maker.game_objects.base.GameEntity) – item that player is using to attack this item

on_burn(*player*)
Called when player burns this item.

Parameters **player** (text_game_maker.player.player.Player) – player object

on_eat(*player*, *word*)
Called when player eats this item

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **word** (*str*) – command word used by player

on_equip(*player*)
Called when player equips this item from inventory

Parameters **player** (text_game_maker.player.player.Player) – player object

on_look(*player*)
Called when player looks at this item

Parameters **player** (text_game_maker.player.player.Player) – player object

on_look_under(*player*)
Called when player looks under this item

Parameters **player** (text_game_maker.player.player.Player) – player object

on_open(*player*)
Called when player opens this item

Parameters **player** (text_game_maker.player.player.Player) – player object

on_read (*player*)
Called when player reads this item

Parameters **player** (text_game_maker.player.player.Player) – player object

on_smell (*player*)
Called when player smells this item

Parameters **player** (text_game_maker.player.player.Player) – player object

on_speak (*player*)
Called when player speaks to this item

Parameters **player** (text_game_maker.player.player.Player) – player object

on_take (*player*)
Called when player attempts to take this item. If on_take returns True, this item will be added to player's inventory. If False, this item will not be added to player's inventory.

Parameters **player** (text_game_maker.player.player.Player) – player object

on_taste (*player*)
Called when player tastes this item

Parameters **player** (text_game_maker.player.player.Player) – player object

on_unequip (*player*)
Called when player unequips this item from inventory (by unequipping explicitly, by dropping or by equipping a different item)

Parameters **player** (text_game_maker.player.player.Player) – player object

prep

set_attrs (*attrs*, *version*)
Recursively deserialize all attributes of this item and any contained items

Parameters

- **attrs** (*dict*) – item attributes
- **version** (*str*) – object model version of item attributes

set_special_attrs (*attrs*, *version*)
Deserialize any attributes that you want to handle specially here. Make sure to delete any specially handled attributes from the return dict so that they are not deserialized by the main set_attrs method

Parameters

- **attrs** (*dict*) – all serialized attributes for this object
- **version** (*str*) – object model version of serialized attributes

Returns all attributes not serialized by this method

Return type dict

skip_attrs = []

class text_game_maker.game_objects.base.**ObjectModelMigration** (*from_version*, *to_version*, *migration_function*)

Bases: object

__init__ (*from_version*, *to_version*, *migration_function*)
x.__init__(...) initializes x; see help(type(x)) for signature

migrate(*attrs*)

text_game_maker.game_objects.base.**build_instance**(*type_key*)
Build an instance of a registered serializable class, by the registered class name

Parameters **type_key** – registered class name

Returns instance of class associated with class name

text_game_maker.game_objects.base.**deserialize**(*data*, *version*)
Recursively deserialize item and all contained items

Parameters

- **data** (*dict*) – serialized item data
- **version** (*str*) – object model version of serialized data

Returns deserialized object

text_game_maker.game_objects.base.**is_deserializable_type**(*obj*)

text_game_maker.game_objects.base.**serialize**(*attr*)
Recursively serialize item and return a serializable dict representing the item and all contained items

Parameters **attr** – object to serialize

Returns serialized object

Return type dict

class text_game_maker.game_objects.generic.**Container**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

Generic container with limited capacity and item size

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

add_item(*item*)
Put an item inside this item

Parameters **item** (text_game_maker.game_objects.base.GameEntity) – item to add

Returns the item that was added

full_class_name = u'text_game_maker.game_objects.generic.Container'

class text_game_maker.game_objects.generic.**ElectricLightSource**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.LightSource*

Base class for an item that can be used as a light source, and can be rejuvenated with batteries when dead

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

add_item(*item*)
Put an item inside this item

Parameters **item** (text_game_maker.game_objects.base.GameEntity) – item to add

Returns the item that was added

full_class_name = u'text_game_maker.game_objects.generic.ElectricLightSource'

get_fuel()

set_fuel(*value*)

class text_game_maker.game_objects.generic.**FlameSource**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.LightSource*

Base class for anything that can be used as a flame source

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.generic.FlameSource'

class text_game_maker.game_objects.generic.**FuelConsumer**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

decrement_fuel()

full_class_name = u'text_game_maker.game_objects.generic.FuelConsumer'

get_fuel()

make_spent(*print_output=True*)

on_fuel_empty()

on_refuel()

refuel (*fuel=None*)

set_fuel (*value*)

class text_game_maker.game_objects.generic.**InventoryBag** (**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Container*

Class to represent a small bag used to carry player items

__init__ (**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

add_to_player_inventory (*player*)
Put this item inside player's inventory. If add_to_player_inventory returns True, execution of the current command will continue normally. If False, execution of the current command will stop immediately.

Parameters **player** (text_game_maker.player.player.Player) – player object

Returns the object that was added

full_class_name = u'text_game_maker.game_objects.generic.InventoryBag'

class text_game_maker.game_objects.generic.**Item** (*prefix=''*, *name=''*, ***kwargs*)
Bases: *text_game_maker.game_objects.base.GameEntity*

Base class for collectable item

__init__ (*prefix=''*, *name=''*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.generic.Item'

on_eat (*player*, *word*)
Called when player eats this item

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **word** (*str*) – command word used by player

set_location (*desc*)
Set the location description of the item, e.g. "on the floor". Items with the same location description will automatically be grouped when described to the player, e.g."an apple, a key and a knife are on the floor"

Parameters **desc** (*str*) – item location description

set_name(*name*)
Set the name of this item

Parameters **name** (*str*) – object name

set_prefix(*prefix*)
Set item prefix word (usually 'an' or 'a')

set_value(*value*)
Set the value of this item in coins

Parameters **value** (*int*) – item value in coins

class text_game_maker.game_objects.generic.**ItemSize**
Bases: object

LARGE = 3

MEDIUM = 2

SMALL = 1

VERY_LARGE = 4

class text_game_maker.game_objects.generic.**LargeContainer**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Container*

Generic container with limited capacity and item size

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.generic.LargeContainer'

class text_game_maker.game_objects.generic.**LightSource**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.FuelConsumer*

Base class for any item that can act as a light source

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.generic.LightSource'

on_equip(*player*)
Called when player equips this item from inventory

Parameters **player** (text_game_maker.player.player.Player) – player object

on_fuel_empty(*print_output=True*)

on_refuel()

on_unequip(*player*)
Called when player unequips this item from inventory (by unequipping explicitly, by dropping or by equipping a different item)

Parameters **player** (text_game_maker.player.player.Player) – player object

class text_game_maker.game_objects.items.**AdvancedLockpick**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

A lockpick that can be used to unlock a finite number of locked doors.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.AdvancedLockpick'

class text_game_maker.game_objects.items.**Bag**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.InventoryBag*

A player inventory bag that can hold a medium number of items.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.Bag'

class text_game_maker.game_objects.items.**BaseballBat**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.items.Weapon*

A baseball bat

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.BaseballBat'

class text_game_maker.game_objects.items.**Battery**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

A single-use battery. Can be inserted into flashlights or other items requiring electric power.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.Battery'

set_fuel(*value*)

class text_game_maker.game_objects.items.**Blueprint**(*ingredients=[]*, *item=None*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

A blueprint for crafting one new item from a fixed set of other items.

__init__(*ingredients=[]*, *item=None*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

add_to_player_inventory(*player*)
Put this item inside player's inventory. If add_to_player_inventory returns True, execution of the current command will continue normally. If False, execution of the current command will stop immediately.

Parameters **player** (text_game_maker.player.player.Player) – player object

Returns the object that was added

full_class_name = u'text_game_maker.game_objects.items.Blueprint'

on_take(*player*)
Called when player attempts to take this item. If on_take returns True, this item will be added to player's inventory. If False, this item will not be added to player's inventory.

Parameters **player** (text_game_maker.player.player.Player) – player object

class text_game_maker.game_objects.items.**BoxOfMatches**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.FlameSource*

A box of matches. Can be used as a light source. Can also be used to burn things. When the fuel runs out, there are no more matches in the box.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.BoxOfMatches'

class text_game_maker.game_objects.items.**Coins**(***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

One or more coins

__init__(***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

add_to_player_inventory(*player*)
Put this item inside player's inventory. If add_to_player_inventory returns True, execution of the current command will continue normally. If False, execution of the current command will stop immediately.

Parameters **player** (text_game_maker.player.player.Player) – player object

Returns the object that was added

decrement(*value=1*)

full_class_name = u'text_game_maker.game_objects.items.Coins'

increment(*value=1*)

on_smell(*player*)
Called when player smells this item

Parameters **player** (text_game_maker.player.player.Player) – player object

on_taste(*player*)
Called when player tastes this item

Parameters **player** (text_game_maker.player.player.Player) – player object

value

class text_game_maker.game_objects.items.**Crowbar**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.items.Weapon*

A crowbar

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"

- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.Crowbar'

class text_game_maker.game_objects.items.**Drawing**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.Drawing'

class text_game_maker.game_objects.items.**Flashlight**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.ElectricLightSource*

An electric flashlight. Allows player to see in darkness. Requires a battery.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.Flashlight'

on_take(*player*)
Called when player attempts to take this item. If on_take returns True, this item will be added to player's inventory. If False, this item will not be added to player's inventory.

Parameters **player** (text_game_maker.player.player.Player) – player object

class text_game_maker.game_objects.items.**Food**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

Generic food item

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.Food'

class text_game_maker.game_objects.items.**Furniture**(*prefix=''*, *name=''*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

An immovable object that the player cannot take or destroy.

__init__(*prefix=''*, *name=''*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.Furniture'

class text_game_maker.game_objects.items.**HuntingKnife**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.items.Weapon*

A larger knife. Higher damage.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.HuntingKnife'

class text_game_maker.game_objects.items.**LargeBag**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.InventoryBag*

A player inventory bag that can hold a large number of items.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.LargeBag'

class text_game_maker.game_objects.items.**Lighter**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.FlameSource*

A disposable lighter. Can be used as a light source. Can also be used to burn things. When the fuel runs out, the lighter is useless.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.Lighter'

class text_game_maker.game_objects.items.**Lockpick**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

A lockpick that can be used to unlock a finite number of locked doors.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.Lockpick'

class text_game_maker.game_objects.items.**Machete**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.items.Weapon*

A machete

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.Machete'

class text_game_maker.game_objects.items.**Paper**(*prefix=''*, *name=''*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

A sheet of paper that can contain information for the player to read

__init__(*prefix=''*, *name=''*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

footer_text()

full_class_name = u'text_game_maker.game_objects.items.Paper'

header_text()

on_look(*player*)
Called when player looks at this item

Parameters **player** (text_game_maker.player.player.Player) – player object

on_read(*player*)
Called when player reads this item

Parameters **player** (text_game_maker.player.player.Player) – player object

paragraphs_text()

class text_game_maker.game_objects.items.**PaperBag**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Container*

A small paper bag that can hold a small number of items.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.PaperBag'

class text_game_maker.game_objects.items.**PocketKnife**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.items.Weapon*

A small knife. Low damage.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.PocketKnife'

class text_game_maker.game_objects.items.**SmallBag**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.InventoryBag*

A player inventory bag that can hold a small number of items.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"

- **name** (*str*) – Item name, e.g. “apple”
- **location** (*str*) – Item location, e.g. “on the floor”
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.SmallBag'

class text_game_maker.game_objects.items.**SmallTin**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Container*

A small tin that can contain a small number of items.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either “a” or “an”
- **name** (*str*) – Item name, e.g. “apple”
- **location** (*str*) – Item location, e.g. “on the floor”
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.SmallTin'

class text_game_maker.game_objects.items.**StrongLockpick**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

A lockpick that can be used to unlock a finite number of locked doors.

__init__(**args*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either “a” or “an”
- **name** (*str*) – Item name, e.g. “apple”
- **location** (*str*) – Item location, e.g. “on the floor”
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.StrongLockpick'

class text_game_maker.game_objects.items.**Weapon**(*prefix=”*, *name=”*, ***kwargs*)
Bases: *text_game_maker.game_objects.generic.Item*

Class to represent a weapon

__init__(*prefix=”*, *name=”*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either “a” or “an”
- **name** (*str*) – Item name, e.g. “apple”
- **location** (*str*) – Item location, e.g. “on the floor”
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.items.Weapon'

class text_game_maker.game_objects.living.**Living**
Bases: object

Base class to represent a living thing with a health metric

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

decrement_health(*val=1*)
Decrement health

Parameters **val** (*int*) – number to decrement health by

increment_health(*val=1*)
Increment health

Parameters **val** (*int*) – number to increment health by

is_dead()
Check if health has been depleted

Returns True if health has been depleted, False otherwise

Return type bool

class text_game_maker.game_objects.living.**LivingGameEntity**
Bases: *text_game_maker.game_objects.living.Living*, *text_game_maker.game_objects.base.GameEntity*

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

full_class_name = u'text_game_maker.game_objects.living.LivingGameEntity'

class text_game_maker.game_objects.living.**LivingItem**(*prefix=''*, *name=''*, ***kwargs*)
Bases: *text_game_maker.game_objects.living.Living*, *text_game_maker.game_objects.generic.Item*

__init__(*prefix=''*, *name=''*, ***kwargs*)
Initialises an Item instance

Parameters

- **prefix** (*str*) – Generally either "a" or "an"
- **name** (*str*) – Item name, e.g. "apple"
- **location** (*str*) – Item location, e.g. "on the floor"
- **value** (*int*) – Item value in coins

full_class_name = u'text_game_maker.game_objects.living.LivingItem'

class text_game_maker.game_objects.person.**Context**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.base.GameEntity*, *text_game_maker.chatbot_utils.responder.Context*

See text_game_maker.chatbot_utils.responder.Context

__init__(**args*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

full_class_name = u'text_game_maker.game_objects.person.Context'

get_special_attrs()
Serialize any attributes that you want to handle specially here. Any attributes present in the dict returned by this function will not be serialized by the main get_attrs method. Intended for subclasses to override

Returns serializable dict of special attributes

Return type dict

set_special_attrs(*attrs*, *version*)
Deserialize any attributes that you want to handle specially here. Make sure to delete any specially handled attributes from the return dict so that they are not deserialized by the main set_attrs method

Parameters

- **attrs** (*dict*) – all serialized attributes for this object
- **version** (*str*) – object model version of serialized attributes

Returns all attributes not serialized by this method

Return type dict

class text_game_maker.game_objects.person.**Person**(*prefix=''*, *name=''*, ***kwargs*)
Bases: *text_game_maker.game_objects.living.LivingItem*

Represents a person that the player can interact with

__init__(*prefix=''*, *name=''*, ***kwargs*)
Initialises a Person instance

Parameters

- **name** (*str*) – name of Person, e.g. "John"
- **description** (*str*) – location description of Person, e.g. "squatting in the corner"
- **items** (*list*) – List of Items held by this person

add_coins(*value=1*)
Give person some coins

Parameters **value** (*int*) – number of coins to add

add_context(*context*)
Add a discussion context to this person. See text_game_maker.chatbot_utils.responder.Context.add_context

add_contexts(**contexts*)
Add multiple discussion contexts to this person. See text_game_maker.chatbot_utils.responder.Context.add_contexts

add_default_responses(**responses*)
Set responses to reply with when the player types something does not match any of the added patterns when speaking to this person

Parameters **responses** – one or more responses to pick randomly from. Responses may be either a string of text to speak, or a callback of the form `callback(person, player)`, where `person` is the Person object the player is speaking to, and `player` is the Player object

add_response(*patterns*, *response*)
Set responses to reply with when player types a specific pattern when talking to this person

Parameters

- **patterns** (*list*) – list of regular expressions that will be used to check player input

- **responses** (*list*) – list of responses to pick randomly from if the player says something that matches one of the patterns in `patterns`. Responses may be either a string of text to speak, or a callback of the form `callback(person, player)`, where `person` is the Person object the player is speaking to, and `player` is the Player object

add_responses(**pattern_response_pairs*)
Set multiple pattern/response pairs at once

Parameters **responses** – one or more response pairs, where each pair is a tuple containing arguments for a single `add_response` call, e.g. `add_responses((['cat.*'], ['meow']), (['dog.*], ['woof']))`

add_shopping_list(**item_value_pairs*)
Add the names of items this person is willing to buy from the player if the player asks, and the price person is willing to pay for them

Parameters **item_value_pairs** – one or more tuples of the form `(item_name, value)`, where `item_name` is the item name as a string, and `value` is an integer representing the price in coins

clear_shopping_list()
Clear this persons current shopping list

die(*player*, *msg=None*)
Kill this person, and print a message to inform the player of this person's death.

Parameters

- **player** (`text_game_maker.player.player.Player`) – player instance
- **msg** (*str*) – message to print informing player of person's death

find_item_class(*classobj*)
Find an item held by this person which is an instance of a specific class

Parameters **classobj** – class to look for an instance of

Returns instance of classobj if found, otherwise None

full_class_name = u'text_game_maker.game_objects.person.Person'

get_response(*text*)
Get the response this person should speak back to the player, for a given input string typed by the player

Parameters **text** (*str*) – input string to respond to

Returns tuple of the form (response, groups) where `response` is the response text as a string, and `groups` is a tuple of subgroups from the regular expression match (as returned by re.MatchObject.groups), if any, otherwise None.

get_special_attrs()
Serialize any attributes that you want to handle specially here. Any attributes present in the dict returned by this function will not be serialized by the main get_attrs method. Intended for subclasses to override

Returns serializable dict of special attributes

Return type dict

on_attack(*player*, *item*)
Called when player attacks this item

Parameters

- **player** (`text_game_maker.player.player.Player`) – player instance

- **item** (text_game_maker.game_objects.base.GameEntity) – item that player is using to attack this item

on_eat(*player*, *word*)
Called when player eats this item

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **word** (*str*) – command word used by player

on_look(*player*)
Called when player looks at this item

Parameters **player** (text_game_maker.player.player.Player) – player object

on_speak(*player*)
Called when player speaks to this item

Parameters **player** (text_game_maker.player.player.Player) – player object

say(*msg*)
Speak to the player

Parameters **msg** (*str*) – message to speak

set_introduction(*msg*)
Set some text for the person to say unprompted, immediately, when the player initates a conversation

Parameters **msg** (*str*) – text to speak

set_special_attrs(*attrs*, *version*)
Deserialize any attributes that you want to handle specially here. Make sure to delete any specially handled attributes from the return dict so that they are not deserialized by the main set_attrs method

Parameters

- **attrs** (*dict*) – all serialized attributes for this object
- **version** (*str*) – object model version of serialized attributes

Returns all attributes not serialized by this method

Return type dict

class text_game_maker.game_objects.person.**Responder**(**args*, ***kwargs*)
Bases: *text_game_maker.game_objects.base.GameEntity*, *text_game_maker.chatbot_utils.responder.Responder*

See text_game_maker.chatbot_utils.responder.Responder

__init__(**args*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

full_class_name = u'text_game_maker.game_objects.person.Responder'

get_special_attrs()
Serialize any attributes that you want to handle specially here. Any attributes present in the dict returned by this function will not be serialized by the main get_attrs method. Intended for subclasses to override

Returns serializable dict of special attributes

Return type dict

set_special_attrs(*attrs*, *version*)
Deserialize any attributes that you want to handle specially here. Make sure to delete any specially handled attributes from the return dict so that they are not deserialized by the main set_attrs method

Parameters

- **attrs** (*dict*) – all serialized attributes for this object
- **version** (*str*) – object model version of serialized attributes

Returns all attributes not serialized by this method

Return type dict

text_game_maker.materials package

Submodules

class text_game_maker.materials.materials.**Material**
Bases: object

BREAD = 'bread'

CARDBOARD = 'cardboard'

CHEESE = 'cheese'

CONCRETE = 'concrete'

DIRT = 'dirt'

GLASS = 'glass'

LEATHER = 'leather'

MEAT = 'meat'

METAL = 'metal'

MUD = 'mud'

PAPER = 'paper'

PLASTIC = 'plastic'

SKIN = 'skin'

STONE = 'stone'

WATER = 'water'

WOOD = 'wood'

class text_game_maker.materials.materials.**MaterialProperties**(*material*, ***kwargs*)
Bases: object

Class representing all the properties of a specific material

__init__(*material*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

text_game_maker.materials.materials.**add_material**(*material*, ***kwargs*)
Define a new material type

Parameters

- **material** (*str*) – unique name to represent material type
- **kwargs** – all attributes of the MaterialProperties class can be set by name here

text_game_maker.materials.materials.**get_materials**()
Return a list of all material names

Returns list of material names

Return type [str]

text_game_maker.materials.materials.**get_properties**(*material*)
Get a MaterialProperties object by material name

Parameters **material** (*str*) – material name

Returns material properties

Return type text_game_maker.material.material.MaterialProperties

text_game_maker.messages package

Submodules

text_game_maker.messages.messages.**attack_corpse_message**(*target_name*, *item_name*)

text_game_maker.messages.messages.**attack_inanimate_object_message**(*target_name*, *weapon_name*)

text_game_maker.messages.messages.**attack_not_returned_message**(*target_name*)

text_game_maker.messages.messages.**attack_with_nonweapon_message**(*target_name*, *item_name*)

text_game_maker.messages.messages.**attack_with_weapon_message**(*target_name*, *item_name*)

text_game_maker.messages.messages.**attacked_with_nonweapon_message**(*target_name*, *item_name*)

text_game_maker.messages.messages.**attacked_with_weapon_message**(*target_name*, *item_name*)

text_game_maker.messages.messages.**bad_taste_message**()

text_game_maker.messages.messages.**badword_message**()

text_game_maker.messages.messages.**burn_combustible_message**(*item_name*)

text_game_maker.messages.messages.**burn_noncombustible_message**(*item_name*)

text_game_maker.messages.messages.**container_too_small_message**(*item_name*, *container_name*)

text_game_maker.messages.messages.**dark_search_message**()

text_game_maker.messages.messages.**dark_stumble_message**()

text_game_maker.messages.messages.**dontknow_message**(*ambiguous_action*)

text_game_maker.messages.messages.**eat_living_person_message**(*name*)

text_game_maker.messages.messages.**gross_action_message**(*gross_action*)

text_game_maker.messages.messages.**no_inventory_item_message**(*item_name*)

text_game_maker.messages.messages.**no_item_message**(*item_name*)

text_game_maker.messages.messages.**nonsensical_action_message**(*nonsensical_action*)

text_game_maker.messages.messages.**pointless_action_message**(*action*)

text_game_maker.messages.messages.**sleep_message**(*sleepword*)

text_game_maker.messages.messages.**strange_action_message**(*strange_action*)

text_game_maker.messages.messages.**suicide_message**()

text_game_maker.parser package

Submodules

text_game_maker.parser.commands.**add_commands**(*parser*)

class text_game_maker.parser.parser.**CharacterTrie**
Bases: object

Simple trie structure where each node is a single character. Used to hold all action words for quick retrieval of the command object for a particular action word.

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

add_token(*string*, *token*)
Add an action word to the parser

Parameters

- **string** (*str*) – action word
- **token** – object to set as token attribute of the last node of action word

dump_json()
Dump the whole trie as JSON-encoded text

Returns JSON-encoded trie structure

Return type str

get_children()
Return the text of all nodes below the node reached by the last `run` call. Used to generate action word suggestions when an action word is partially or incorrectly typed.

Returns list of action words from all children nodes

Return type [str]

iterate()
Iterate over all nodes in the trie that have non-None token attributes :return: iterator for all trie nodes with non-None tokens

run(*input_string*)

set_search_filter(*callback*)
Set function to filter token objects when iterating through the trie

Parameters **callback** – callback function of the form `callback(token)`, where `token` is the token attribute from a Node in the trie. If callback returns True, then this token object will be included in the objects returned by iteration. Otherwise, this token object will be exluded from the objects returned by iteration.

class text_game_maker.parser.parser.**CharacterTrieNode**(*char*, *token=None*, *text=None*)

Bases: object

A single node in a CharacterTrie

__init__(*char*, *token=None*, *text=None*)

Parameters

- **char** (*str*) – character for this node.
- **token** – optional arbitrary object to store at this node.
- **text** (*str*) – optional string to store at this node, currently used to hold the full matching text in the last node of a command. Allows for easy/quick iterating of all words in the trie.

class text_game_maker.parser.parser.**Command**(*word_list*, *callback*, *desc*, *usage_fmt*, *hidden*)

Bases: object

Container class for data needed to execute a particular game command

__init__(*word_list*, *callback*, *desc*, *usage_fmt*, *hidden*)

Parameters

- **word_list** (*list*) – list of action words that can trigger this parser command.
- **callback** – callback function to be invoked when the player types something beginning with one of the action words. Callback should be of the form `callback(player, word, remaining)`, where `player` is the `text_game_maker.player.player.Player` instance, `word` is the action word typed by the player as a string, and `remaining` is the remaining text following the action word as a string.
- **desc** (*str*) – text that will be shown when player types "help" followed by one of the action words.
- **usage_fmt** (*str*) – format string that will be used to show a usage example for each action word. The format string should contain a single string format character, which will be replaced with the action word e.g.‘“%s <item>”‘ where `%s` will be replaced with an action word.
- **hidden** (*bool*) – if True, command can still be triggered normally by player input but will be excluded from "help" queries and parser suggestions.

help_text()

Get the help text with usage examples

Returns generated help text and usage examples

Return type str

class text_game_maker.parser.parser.**CommandParser**

Bases: *text_game_maker.parser.parser.CharacterTrie*

Thin wrapper around a CharacterTrie that adds some default commands

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

add_command(*word_set*, *callback*, *help_text=None*, *usage_fmt=None*, *hidden=False*)

Add a command to the parser.

Parameters

- **word_set** (*[str]*) – list of action words that can trigger command
- **callback** – callback function to be invoked when the player types something beginning with one of the action words. Callback should be of the form `callback(player, word, remaining)`, where `player` is the `text_game_maker.player.player.Player` instance, `word` is the action word typed by the player as a string, and `remaining` is the remaining text following the action word as a string.
- **help_text** (*str*) – text that will be shown when player types "help" followed by one of the action words.
- **usage_fmt** (*str*) – format string that will be used to show a usage example for each action word. The format string should contain a single string format character, which will be replaced with the action word e.g.‘“”%s <item>”“‘ where `%s` will be replaced with an action word.
- **hidden** (*bool*) – if True, command can still be triggered normally by player input but will be excluded from "help" queries and parser suggestions.

add_event_handler(*word*, *callback*)

Add an event handler to run whenever a command is used, in addition to the regular handler for that command.

Parameters

- **word** (*str*) – action word associated with the command to add the event handler to.
- **callback** – callback function to be invoked when the player types something beginning with one of the action words. Callback should be of the form `callback(player, word, remaining)`, where `player` is the `text_game_maker.player.player.Player` instance, `word` is the action word typed by the player as a string, and `remaining` is the remaining text following the action word as a string.

clear_event_handler(*word*, *callback*)

Remove a previously added event handler for a command

Parameters

- **word** (*str*) – action word associated with the command to remove the event handler from.
- **callback** – callback function for event handler to remove

text_game_maker.player package

Submodules

class text_game_maker.player.player.**Player**(*start_tile=None*, *input_prompt=None*)

Bases: *text_game_maker.game_objects.living.LivingGameEntity*

Base class to hold player related methods & data

__init__(*start_tile=None*, *input_prompt=None*)

Parameters

- **start_tile** (*text_game_maker.game_objects.tile.Tile*) – Starting tile
- **input_prompt** (*str*) – Custom string to prompt player for game input

add_coins(*value=1*)

Give player some coins

Parameters **value** (*int*) – number of coins to add

buy_item_from(*person*)
Show player a listing of items this person has and allow them to select one to purchase (if they have enough coins)

Parameters **person** (text_game_maker.game_objects.person.Person) – person object

can_see()
Check if the player can see their surroundings in the current game location. Takes the following things into account;

- Is the current tile dark?
- Does the player have a light source equipped?
- Does the light source need fuel, and if so, does it have some?

Returns True if the player can see their surroundings, False otherwise

Return type bool

clear_task(*task_id*)
Clear a specific scheduled task by task ID

Parameters **task_id** (*int*) – task ID of the task to remove

Returns False if the provided task ID was not found in the pending scheduled tasks list, True otherwise

Return type bool

clear_tasks()
Clear all pending scheduled tasks (tasks which have been added but whose timers have not expired)

darkness_message()
Return the message to print when the player enters an area which is dark and they do not have a working light source

Returns message to print when player cannot see surrounding area

Return type str

death()
Called whenever the player dies

decrement_energy(*val=1*)
Decrement player energy

Parameters **val** (*int*) – number to decrement player energy by

describe_current_tile()
Returns the full descriptive text for the tile player is currently on, including everything contained in the tile and all adjacent tiles visible to the player

Returns text description of current game state

Return type str

describe_current_tile_contents(*capitalize=True*)
Return a string that describes the game state at the players current, location, including people, objects and scenery on the players current tile, and any adjacent tiles that connect to the players current tile.

Parameters **capitalize** (*bool*) – if True, the first letter of each sentence will be capitalized

Returns string describing the players current loca

full_class_name = u'text_game_maker.player.player.Player'

get_equipped()
Get player equipped item

Returns equipped item. None if no item is equipped.

Return type text_game_maker.game_objects.items.Item

get_special_attrs()
Serialize any attributes that you want to handle specially here. Any attributes present in the dict returned by this function will not be serialized by the main get_attrs method. Intended for subclasses to override

Returns serializable dict of special attributes

Return type dict

has_item(*item*)
Check if an item is in the player's inventory

Parameters **item** (text_game_maker.game_objects.generic.Item) – item to check

Returns True if item is in the player's inventory, False otherwise

Return type bool

increment_energy(*val=1*)
Increment player energy

Parameters **val** (*int*) – number to increment player energy by

injure(*health_points=1*)
Injure player by removing a specific number of health points. Player will die if resulting health is less than or equal to 0.

param int health_points: number of health points to remove from player

inventory_space()
Check number of remaining items the players inventory can fit. When players inventory is full, this method will return 0.

Returns number of remaining items player's inventory has space for

;rtype: int

on_smell()
Called when player smells themselves

on_taste()
Called when player tastes themselves

previous_tile()
Get the tile that the player was on before the current tile

Returns previous tile object

Return type *text_game_maker.tile.tile.Tile*

read_player_name_and_set()
Helper function to read a name from the user and set as the player's name

remove_coins(*value=1*)
Take some coins from player

Parameters **value** (*int*) – number of coins to remove

save_to_file (*filename*, *compression=True*)
Serialize entire map and player state and write to a file

Parameters

- **filename** (*str*) – name of file to write serialized state to
- **compression** (*bool*) – whether to compress string

save_to_string (*compression=True*)
Serialize entire map and player state and return as a string

Parameters **compression** (*bool*) – whether to compress string

Returns serialized game state

Return type str

schedule_task (*callback*, *turns=1*)
Add a function that will be invoked after the player has taken some number of moves (any valid input from the player counts as a move)

The function should accept one parameter, and return a bool:

def callback(player, turns): pass

Callback parameters:

- *player* (text_game_maker.player.player.Player): player instance
- *turns* (int): number of turns this callback was scheduled for
- *Return value* (bool): if True, this task will be scheduled again with the same number of turns

Parameters

- **callback** (*str*) – function that returns the message to print
- **turns** (*int*) – number of turns to pass before callback will be invoked

Returns task ID

Return type int

scheduler_tick ()
'tick' function for task scheduler. Called on each move the player makes (unparseable or otherwise invalid input does not incur a 'tick'), and executes any tasks scheduled for that move.

sell_item_to (*person*)
Show player a listing of this person's shopping list, and allow player to sell something to person if the player has it

Parameters **person** (text_game_maker.game_objects.person.Person) – person object

set_alternate_names (*tile*)

set_name (*name*)
Set player name

Parameters **name** (*str*) – new player name

set_special_attrs(*attrs*, *version*)
Deserialize any attributes that you want to handle specially here. Make sure to delete any specially handled attributes from the return dict so that they are not deserialized by the main set_attrs method

Parameters

- **attrs** (*dict*) – all serialized attributes for this object
- **version** (*str*) – object model version of serialized attributes

Returns all attributes not serialized by this method

Return type dict

skip_attrs = ['parser', 'new_game_event']

text_game_maker.player.player.**load_from_file**(*filename*, *compression=True*)
Load a serialized state from a file and create a new player instance

Parameters

- **filename** (*str*) – name of save file to read
- **compression** (*bool*) – whether data is compressed

Returns new Player instance

Return type *text_game_maker.player.player.Player*

text_game_maker.player.player.**load_from_string**(*strdata*, *compression=True*)
Load a serialized state from a string and create a new player instance

Parameters

- **strdata** (*str*) – string data to load
- **compression** (*bool*) – whether data is compressed

Returns new Player instance

Return type *text_game_maker.player.player.Player*

text_game_maker.ptttl package

Submodules

text_game_maker.ptttl.ptttl_audio_encoder.**ptttl_to_mp3**(*ptttl_data*, *mp3_filename*, *amplitude=0.5*, *wavetype=0*)

text_game_maker.ptttl.ptttl_audio_encoder.**ptttl_to_sample_data**(*ptttl_data*, *amplitude=0.5*, *wavetype=0*)

text_game_maker.ptttl.ptttl_audio_encoder.**ptttl_to_wav**(*ptttl_data*, *wav_filename*, *amplitude=0.5*, *wavetype=0*)

class text_game_maker.ptttl.ptttl_parser.**PTTTLParser**
Bases: object

parse(*ptttl_string*)

exception text_game_maker.ptttl.ptttl_parser.**PTTTLSyntaxError**
Bases: exceptions.Exception

exception text_game_maker.ptttl.ptttl_parser.**PTTTLValueError**
Bases: exceptions.Exception

text_game_maker.ptttl.ptttl_parser.**ignore_line**(*line*)

text_game_maker.ptttl.ptttl_parser.**int_setting**(*key*, *val*)

text_game_maker.ptttl.ptttl_parser.**invalid_note**(*note*)

text_game_maker.ptttl.ptttl_parser.**invalid_note_duration**(*note*)

text_game_maker.ptttl.ptttl_parser.**invalid_octave**(*note*)

text_game_maker.ptttl.ptttl_parser.**invalid_setting**(*key*)

text_game_maker.ptttl.ptttl_parser.**invalid_value**(*key*, *val*)

text_game_maker.ptttl.ptttl_parser.**missing_setting**(*key*)

text_game_maker.ptttl.ptttl_parser.**unrecognised_setting**(*key*)

text_game_maker.tile package

Submodules

class text_game_maker.tile.tile.**LockedDoor**(*prefix=''*, *name=''*, *src_tile=''*, *replacement_tile=''*)
Bases: *text_game_maker.tile.tile.Tile*

Locked door with a mechanical lock, requires a key or lockpick to unlock

__init__(*prefix=''*, *name=''*, *src_tile=''*, *replacement_tile=''*)
Initialise a Tile instance

Parameters

- **name** (*str*) – short description, e.g. "a dark cellar"
- **description** (*str*) – long description, printed when player enters the room e.g. "a dark, scary cellar with blah blah blah... "

full_class_name = u'text_game_maker.tile.tile.LockedDoor'

get_special_attrs()
Serialize any attributes that you want to handle specially here. Any attributes present in the dict returned by this function will not be serialized by the main get_attrs method. Intended for subclasses to override

Returns serializable dict of special attributes

Return type dict

is_door()

matches_name(*name*)
Check if a string fuzzily matches the name of this door

Parameters **name** (*str*) – string to check

Returns True if string fuzzily matches this doors name

Return type bool

on_enter(*player*, *src*)
Called when player tries to move into this tile. If on_enter returns True, the player will be moved to this tile. If False, the move will not be allowed.

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **src** (text_game_maker.tile.tile.Tile) – tile that player is currently on

Returns True if player move should be allowed

Return type bool

on_open(*player*)
Called when player attempts to open this door

Parameters **player** (text_game_maker.player.player.Player) – player object

prep

unlock()
Unlocks the door, revealing whichever tile is connected behind it

class text_game_maker.tile.tile.**LockedDoorWithKeypad**(*unlock_code=None*, ***kwargs*)
Bases: *text_game_maker.tile.tile.LockedDoor*

Locked door with an electronic lock, can be opened with a numeric code

__init__(*unlock_code=None*, ***kwargs*)
Initialise a Tile instance

Parameters

- **name** (*str*) – short description, e.g. "a dark cellar"
- **description** (*str*) – long description, printed when player enters the room e.g. "a dark, scary cellar with blah blah blah... "

full_class_name = u'text_game_maker.tile.tile.LockedDoorWithKeypad'

on_enter(*player*, *src*)
Called when player tries to move into this tile. If on_enter returns True, the player will be moved to this tile. If False, the move will not be allowed.

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **src** (text_game_maker.tile.tile.Tile) – tile that player is currently on

Returns True if player move should be allowed

Return type bool

on_open(*player*)
Called when playyer attempts to open this door.

Parameters **player** (text_game_maker.player.player.Player) – player object

set_prompt(*prompt*)
Set text to be display when prompting player to input keypad code

Parameters **prompt** (*str*) – text to prompt with

class text_game_maker.tile.tile.**Tile**(*name=None*, *description=None*)
Bases: *text_game_maker.game_objects.base.GameEntity*

Represents a single 'tile' or 'room' in the game

__init__(*name=None*, *description=None*)
Initialise a Tile instance

Parameters

- **name** (*str*) – short description, e.g. "a dark cellar"
- **description** (*str*) – long description, printed when player enters the room e.g. "a dark, scary cellar with blah blah blah... "

add_item(*item*)
Add item to this tile

Parameters **item** (GameEntity) – item to add

Returns the added item

add_person(*person*)
Add person to this tile

Parameters **text_game_maker.game_objects.person.Person** – person to add

copy()

default_locations = ['on the ground']

describe_items()
Return sentences describing all non-scenery items on this tile

Returns description of non-scenery on this tile

Return type str

describe_people()
Return sentences describing all people on this tile

Returns description of all people (NPCs) on this tile

Return type str

describe_scene()
Return sentences describing all scenery items on this tile

Returns description of scenery on this tile

Return type str

direction_from(*tile*)
Get the direction from the given tile to this tile

Parameters **tile** (Tile) – tile to check the direction from

Returns direction from the given tile to this tile

Return type str

direction_to(*tile*)
Get the direction from this tile to the given tile

Parameters **tile** (Tile) – tile to check the direction to

Returns direction to the given tile

Return type str

full_class_name = u'text_game_maker.tile.tile.Tile'

get_special_attrs()
Serialize any attributes that you want to handle specially here. Any attributes present in the dict returned by this function will not be serialized by the main get_attrs method. Intended for subclasses to override

Returns serializable dict of special attributes

Return type dict

is_connected_to(*tile*)
Checks if this tile is connected to the given tile

Parameters **tile** (Tile) – tile to check against this tile

Returns True if tile is connected to this tile, False otherwise

Return type bool

is_door()

iterate_directions()
Iterator for all tiles connected to this tile

Returns tile iterator

Return type Iterator[*text_game_maker.tile.tile.Tile*]

map_identifier

matches_name(*name*)
Check if a string fuzzily matches the name of this tile

Parameters **name** (*str*) – string to check

Returns True if string fuzzily matches this tiles name

Return type bool

on_enter(*player*, *src*)
Called when player tries to move into this tile. If on_enter returns True, the player will be moved to this tile. If False, the move will not be allowed.

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **src** (text_game_maker.tile.tile.Tile) – tile that player is currently on

Returns True if player move should be allowed

Return type bool

on_exit(*player*, *dest*)
Called when player tries to move out of this tile. If on_exit returns True, the player will be moved to dest. If False, the move will not be allowed.

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **dest** (text_game_maker.tile.tile.Tile) – tile that player is trying to move to

Returns True if player move should be allowed

Return type bool

on_smell()
Called when player types 'smell' or equivalent on this tile

on_smell_ground()
Called when player types 'smell ground' or equivalent on this tile

on_taste_ground()
Called when player types 'taste ground' or equivalent on this tile

set_name_from_east(*name*)
Set the name that will be shown when player looks at this tile from an adjacent tile east from this tile

Parameters **desc** (*str*) – description text

set_name_from_north(*name*)
Set the name that will be shown when player looks at this tile from an adjacent tile north from this tile

Parameters **desc** (*str*) – description text

set_name_from_south(*name*)
Set the name that will be shown when player looks at this tile from an adjacent tile south from this tile

Parameters **desc** (*str*) – description text

set_name_from_west(*name*)
Set the name that will be shown when player looks at this tile from an adjacent tile west from this tile

Parameters **desc** (*str*) – description text

set_special_attrs(*data*, *version*)
Deserialize any attributes that you want to handle specially here. Make sure to delete any specially handled attributes from the return dict so that they are not deserialized by the main set_attrs method

Parameters

- **attrs** (*dict*) – all serialized attributes for this object
- **version** (*str*) – object model version of serialized attributes

Returns all attributes not serialized by this method

Return type dict

set_tile_id(*tile_id*)
Sets the ID for this tile. This ID will be used to represent the tile in save files. Setting explicit tile names is recommended, as it ensures that the tile IDs will not change. If the ID of a tile changes, save files created with previous tile IDs will no longer work as expected.

Parameters **tile_id** – tile ID

summary()
Return a description of all available directions from this tile that the player can see

Returns description of all available directions from this tile

Return type str

tile_id = 0

text_game_maker.tile.tile.**builder**(*tiledata*, *start_tile_id*, *version*, *clear_old_tiles=True*)
Deserialize a list of serialized tiles, then re-link all the tiles to re-create the map described by the tile links

Parameters

- **tiledata** (*list*) – list of serialized tiles
- **start_tile_id** – tile ID of tile that should be used as the start tile
- **version** (*str*) – object model version of the tile data to be deserialized

Returns starting tile of built map

Return type *text_game_maker.tile.tile.Tile*

text_game_maker.tile.tile.**crawler**(*start*)
Crawl over a map and serialize all tiles and contained items

Parameters **start** (text_game_maker.tile.tile.Tile) – map starting tile

Returns serializable dict representing all tiles and contained items

Return type dict

text_game_maker.tile.tile.**get_tile_by_id**(*tile_id*)
Get Tile instance by tile ID

Parameters **tile_id** – ID of tile to fetch

Returns tile instance

Return type *text_game_maker.tile.tile.Tile*

text_game_maker.tile.tile.**reverse_direction**(*direction*)
Returns the opposite direction for a given direction, e.g. "north" becomes "south"

Parameters **direction** (*str*) – direction to reverse

Returns opposite direction. None if invalid direction is provided

Return type str

text_game_maker.tile.tile.**unregister_tile_id**(*tile_id*)
Unregister a tile ID so it can be used again. Should only be used if you are deleting all instances of a tile object.

Parameters **tile_id** – tile ID to unregister

text_game_maker.utils package

Submodules

class text_game_maker.utils.runner.**MapRunner**
Bases: object

Extend this class and implement the two methods below to run maps with the text-game-runner.py script

build_map(*builder*)
Implement this method to build a map using functions from text_game_maker.builder.map_builder

Parameters **builder** (text_game_maker.builder.map_builder.MapBuilder) – map builder instance

build_parser(*parser*)
Implement this method to add custom commands to the game parser

Parameters **parser** (text_game_maker.parser.parser.CommandParser) – command parser

exception text_game_maker.utils.runner.**MapRunnerError**
Bases: exceptions.Exception

text_game_maker.utils.runner.**get_runner_from_filename**(*filename*)
Import the given file, look for any classes that are subclasses of text_game_maker.utils.runner.MapRunner, and return the class object of the first one found (if any)

Parameters **filename** (*str*) – file to read

Returns MapRunner subclass object found (None if no MapRunner subclasses are found

text_game_maker.utils.runner.**run_map_from_class**(*classobj*)
Create an instance of the given map runner class, and run it

Parameters **classobj** – mapp runner class object

text_game_maker.utils.runner.**run_map_from_filename**(*filename*)
Import a file, look for any classes that are subclasses of text_game_maker.utils.runner.MapRunner, and run the first one found (if any)

Parameters **filename** (*str*) – file to read

class text_game_maker.utils.utils.**BorderType**
Bases: object

DOOR = 2

OPEN = 0

WALL = 1

class text_game_maker.utils.utils.**SubclassTrackerMetaClass**(*name*, *bases*, *clsdict*)
Bases: type

__init__(*name*, *bases*, *clsdict*)
x.__init__(...) initializes x; see help(type(x)) for signature

text_game_maker.utils.utils.**add_format_token**(*token*, *func*)
Add a format token

Parameters

- **token** (*str*) – token string to look for
- **func** – function to call to obtain replacement text for format token

text_game_maker.utils.utils.**add_serializable_callback**(*callback*)
Add a callback object to registry of callbacks that can be securely referenced in a save file. An ID will be assigned to represent this callback in save files. When loading a save file, whatever object you pass here will be used when the same ID is seen.

Parameters **callback** – callback object to register

text_game_maker.utils.utils.**ask_multiple_choice**(*choices*, *msg=None*, *cancel_word=u'cancel'*, *default=None*)
Ask the user a multiple-choice question, and return their selection

Parameters **choices** (*[str]*) – choices to present to the player

Returns list index of player's selection (-1 if user cancelled)

Return type int

text_game_maker.utils.utils.**ask_yes_no**(*msg*, *cancel_word=u'cancel'*)
Ask player a yes/no question, and repeat the prompt until player gives a valid answer

Parameters

- **msg** (*str*) – message to print inside prompt to player
- **cancel_word** (*str*) – player response that will cause this function to return -1

Returns 1 if player responded 'yes', 0 if they responded 'no', and -1 if they cancelled

Return type int

text_game_maker.utils.utils.**capitalize**(*text*)
Capitalize first non-whitespace character folling each period in string

Parameters **text** (*str*) – text to capitalize

Returns capitalized text

Return type str

text_game_maker.utils.utils.**centre_text**(*string*, *line_width=None*)
Centre a line of text within a specific number of columns

Parameters

- **string** (*str*) – text to be centred
- **line_width** (*int*) – line width for centreing text. If None, the current line width being used for game output will be used

Returns centred text

Return type str

text_game_maker.utils.utils.**container_listing**(*container*, *item_fmt=u' {0:33}{1:1}({2})'*, *width=50*, *bottom_border=False*, *name=None*)

text_game_maker.utils.utils.**del_from_lists**(*item*, **lists*)

text_game_maker.utils.utils.**deserialize_callback**(*callback_name*)

text_game_maker.utils.utils.**disable_command**(*command*)
Disable a parser command. Useful for situations where you want to disable certain capabilities, e.g. loading/saving game states

Parameters **command** (*str*) – word that makes to parser command to disable

text_game_maker.utils.utils.**disable_commands**(**commands*)
Disable multiple commands at once. Equivalent to multiple 'disable_command' calls

Parameters **commands** – one or more words mapping to parser commands to disable

text_game_maker.utils.utils.**draw_map_of_nearby_tiles**(*player*)
Draw a ASCII representation of tile surrounding the player's current position

Parameters **player** (text_game_maker.player.player.Player) – player object

Returns created map

Return type str

text_game_maker.utils.utils.**english_to_list**(*text*)
Convert a string of the form 'a, b, c and d' to a list of the form ['a','b','c','d']

Parameters **text** (*str*) – input string

Returns list of items in string, split on either ',' or 'and'

Return type list

text_game_maker.utils.utils.**find_any_item**(*player*, *name*)
Find an item by name in either the player's inventory or in the current tile

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **name** (*str*) – name of item to search for

Returns found item. If no matching item is found, None is returned.

Return type text_game_maker.game_objects.items.Item

text_game_maker.utils.utils.**find_inventory_item**(*player*, *name*)
Find an item by name in player's inventory

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **name** (*str*) – name of item to search for

Returns found item. If no matching item is found, None is returned.

Return type text_game_maker.game_objects.items.Item

text_game_maker.utils.utils.**find_inventory_item_class**(*player*, *classobj*)
Find first item in player's inventory which is an instance of a specific class

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **classobj** – class to check for instances of

Returns found item. If no matching item is found, None is returned.

Return type text_game_maker.game_objects.items.Item

text_game_maker.utils.utils.**find_inventory_wildcard**(*player*, *name*)
Find the first item in player's inventory whose name matches a wildcard pattern ('*').

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **name** (*str*) – wildcard pattern

Returns found item. If no matching item is found, None is returned.

Return type text_game_maker.game_objects.items.Item

text_game_maker.utils.utils.**find_item**(*player*, *name*, *locations=None*, *ignore_dark=False*)
Find an item by name in the provided locations

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **name** (*str*) – name of item to find
- **locations** (*[[text_game_maker.game_objects.items.Item]]*) – location lists to search. If None, the item list of the current tile is used

Returns found item (None if no matching item is found)

Return type text_game_maker.items.Item

text_game_maker.utils.utils.**find_item_class**(*player*, *classobj*, *locations=None*, *ignore_dark=False*)
Find the first item that is an instance of a specific class in the provided locations

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **name** (*str*) – name of item to find

- **locations** (*[[text_game_maker.game_objects.items.Item]]*) – location lists to search

Returns found item (None if no matching item is found)

Return type text_game_maker.items.Item

text_game_maker.utils.utils.**find_item_wildcard**(*player*, *name*, *locations=None*)
Find the first item whose name matches a wildcard pattern ('*') in specific locations.

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **name** (*str*) – wildcard pattern
- **locations** (*[[text_game_maker.game_objects.items.Item]]*) – location lists to search. If None, the item list of the current tile is used

Returns found item. If no matching item is found, None is returned.

Return type text_game_maker.game_objects.items.Item

text_game_maker.utils.utils.**find_person**(*player*, *name*)
Find a person by name in the current tile

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **name** (*str*) – name of person to search for

Returns found person. If no matching person is found, None is returned.

Return type *text_game_maker.game_objects.person.Person*

text_game_maker.utils.utils.**find_tile**(*player*, *name*)
Find an adjacent tile that is connected to the current tile by name

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **name** (*str*) – name of adjacent tile to search for

Returns adjacent matching tile. If no matching tiles are found, None is returned

Return type *text_game_maker.tile.tile.Tile*

text_game_maker.utils.utils.**flush_waiting_prints**()

text_game_maker.utils.utils.**game_print**(*msg*, *wait=False*)
Print one character at a time if player has set 'print slow', otherwise print normally

Parameters **msg** (*str*) – message to print

text_game_maker.utils.utils.**get_all_contained_items**(*item*, *stoptest=None*)
Recursively retrieve all items contained in another item

Parameters

- **item** (*text_game_maker.game_objects.items.Item*) – item to retrieve items from
- **stoptest** – callback to call on each sub-item to test whether recursion should continue. If stoptest() == True, recursion will continue

Returns list of retrieved items

Return type [text_game_maker.game_objects.items.Item]

text_game_maker.utils.utils.**get_all_items**(*player*, *locations=None*, *except_item=None*)
Retrieves all items from specified locations

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **locations** (*[[text_game_maker.game_objects.items.Item]]*) – location lists to search. If None, the item list of the current room/tile is used
- **except_item** (*object*) – do not retrive item from location if it is the same memory object as except_item. If None, no items are ignored.

Returns list of retreived items

Return type [text_game_maker.game_objects.items.Item]

text_game_maker.utils.utils.**get_basic_controls**()
Returns a basic overview of game command words

text_game_maker.utils.utils.**get_builder_instance**()

text_game_maker.utils.utils.**get_chardelay**()

text_game_maker.utils.utils.**get_full_controls**(*parser*)
Returns a comprehensive listing of of all game command words

text_game_maker.utils.utils.**get_full_import_name**(*classobj*)

text_game_maker.utils.utils.**get_last_command**()

text_game_maker.utils.utils.**get_local_tile_map**(*tileobj*, *crawltiles=2*, *mapsize=5*)
Builds a 2D list of tiles, representing an x/y grid of tiles sorrounding the players current position.

Parameters

- **tileobj** (text_game_maker.tile.tile.Tile) – starting tile
- **crawltiles** (*int*) – maximum tiles to travel from starting tile
- **mapsize** (*int*) – map height and width in tiles

Returns 2D list representing x/y grind of tiles around player

text_game_maker.utils.utils.**get_print_controls**()

text_game_maker.utils.utils.**get_random_name**()
Get a random first and second name from old US census data, as a string e.g. "John Smith"

Returns random name

Return type str

text_game_maker.utils.utils.**get_sequence_count**(*count*)

text_game_maker.utils.utils.**get_serializable_class**(*name*)

text_game_maker.utils.utils.**get_slow_printing**()

text_game_maker.utils.utils.**import_module_attribute**(*fullname*)

text_game_maker.utils.utils.**inputfunc**(*prompt*)
Block until user input is available

Parameters **prompt** (*str*) – string to prompt user for input

Returns user input

Return type str

text_game_maker.utils.utils.**is_disabled_command**(**commands*)
Checks if any of the provided words map to a disabled command

Parameters **commands** – one or more strings containing words mapping to parser commands

text_game_maker.utils.utils.**is_location**(*player*, *name*)
Checks if text matches the name of an adjacent tile that is connected to the current tile

Parameters

- **player** (text_game_maker.player.player.Player) – player object
- **name** (*str*) – text to check

Returns True if text matches adjacent tile name

Return type bool

text_game_maker.utils.utils.**last_saved_sound**()
Retrieve last sound ID saved with text_game_maker.save_sound

Returns last saved sound ID

text_game_maker.utils.utils.**line_banner**(*text*, *width=None*, *bannerchar=u'-'*, *spaces=1*)
Centre a line of text within a specific number of columns, and surround text with a repeated character on either side.

Example:

——— centred text ———

Parameters

- **text** (*str*) – text to be centred
- **width** (*int*) – line width for centreing text
- **bannerchar** (*str*) – character to use for banner around centred text
- **spaces** (*int*) – number of spaces seperating centred text from banner

Returns centred text with banner

Return type str

text_game_maker.utils.utils.**list_to_english**(*strlist*, *conj=u'and'*)
Convert a list of strings to description of the list in english. For example, [‘4 coins’, ‘an apple’, ‘a sausage’] would be converted to ‘4 coins, an apple and a sausage’

Parameters **strlist** (*str*) – list of strings to convert to english

Returns english description of the passed list

Return type str

text_game_maker.utils.utils.**multisplit**(*s*, **seps*)
Split a string into substrings by multiple tokens

Parameters

- **s** (*str*) – string to split
- **seps** (*[str]*) – list of strings to split on

Returns list of substrings

Return type [str]

text_game_maker.utils.utils.**pop_command**()
Pop oldest command from game command sequence list

Returns oldest command in game command sequence list

Return type str

text_game_maker.utils.utils.**pop_waiting_print**()

text_game_maker.utils.utils.**printfunc**(*text*)
Display game output

Parameters **text** (*str*) – text to display

Returns value returned by print function

text_game_maker.utils.utils.**queue_command_sequence**(*seq*)
Add to game command sequence list

Parameters **seq** (*[str]*) – list of command strings to add

text_game_maker.utils.utils.**read_line**(*msg*, *cancel_word=None*, *default=None*)

text_game_maker.utils.utils.**read_line_raw**(*msg=u"*, *cancel_word=None*, *default=None*)
Read a line of input from stdin

Parameters **msg** (*str*) – message to print before reading input

Returns a line ending with either a newline or carriage return character

Return type str

text_game_maker.utils.utils.**read_path_autocomplete**(*msg*)

text_game_maker.utils.utils.**register_serializable_class**(*classobj*, *name*)

text_game_maker.utils.utils.**replace_format_tokens**(*text*)
Replace format tokens in string (if any)

Parameters **text** (*str*) – text that may contain format tokens

Returns formatted text

Return type str

text_game_maker.utils.utils.**run_parser**(*parser*, *action*)

text_game_maker.utils.utils.**save_sound**(*sound*)
Save a sound to be played when parsing of the current command is completed. Overwrites any previously saved sound.

Parameters **sound** – sound ID key needed by text_game_maker.audio.audio.play_sound

text_game_maker.utils.utils.**serializable_callback**(*callback*)
Decorator version of add_serializable_callback. Example:

```
from text_game_maker.player.player import serializable_callback

@serializable_callback
def my_callback_function():
    pass
```

text_game_maker.utils.utils.**serialize_callback**(*callback*)

text_game_maker.utils.utils.**set_builder_instance**(*ins*)

text_game_maker.utils.utils.**set_chardelay**(*delay*)

text_game_maker.utils.utils.**set_inputfunc**(*func*)
Set function to be used for blocking on input from the user. The default if unset is to use a prompt session from prompt-toolkit reading from stdin of the process where text_game_maker is running. The provided function is responsible for blocking until user input is available, and must return the user input as a string

Parameters **func** – function to use for reading user input

text_game_maker.utils.utils.**set_last_command**(*cmd*)

text_game_maker.utils.utils.**set_printfunc**(*func*)
Set function to be used for displaying game output. The default if unset is to use standard python "print".

Parameters **func** – function to pass game output text to be displayed

text_game_maker.utils.utils.**set_sequence_count**(*count*)

text_game_maker.utils.utils.**set_slow_printing**(*val*)

text_game_maker.utils.utils.**set_wrap_width**(*width*)
Set the maximum line width (in characters) used by text_game_maker when wrapping long lines of text (default is 60)

Parameters **width** (*int*) – maximum line width in characters

- genindex
- modindex
- search

Python Module Index

t

Index

Symbols

A

B

C

D

E

F

G

Q

R

S

T

U

V

W