jericho

Release 1.3.1

Sep 03, 2021

Tutorials

CHAPTER 1

Jericho Quick-start

1.1 Install

Jericho requires Linux, Python 3, Spacy, and basic build tools like gcc.

There are two ways to install Jericho:

1.1.1 From PyPi

```
pip3 install jericho
python3 -m spacy download en_core_web_sm
```

1.1.2 From Github

1. Clone Jericho Repository:

```
git clone https://github.com/microsoft/jericho.git
```

2. Install Jericho:

```
cd jericho; pip3 install .
python3 -m spacy download en_core_web_sm
```

1.2 Acquire Games

An archive of games can be downloaded as follows:

```
wget https://github.com/BYU-PCCL/z-machine-games/archive/master.zip
unzip master.zip
```

Jericho-supported games may be found in the `jericho-game-suite` subdirectory.

1.3 Basic Usage

Jericho implements a reinforcement learning interface in which the agent provides string-based actions, and the environment responds with string-based observations, rewards, and termination status:

```
from jericho import *
# Create the environment, optionally specifying a random seed
env = FrotzEnv("z-machine-games-master/jericho-game-suite/zork1.z5")
initial_observation, info = env.reset()
done = False
while not done:
    # Take an action in the environment using the step fuction.
    # The resulting text-observation, reward, and game-over indicator is returned.
    observation, reward, done, info = env.step('open mailbox')
    # Total score and move-count are returned in the info dictionary
    print('Total Score', info['score'], 'Moves', info['moves'])
print('Scored', info['score'], 'out of', env.get_max_score())
```

1.4 Game Dictionary

jericho.FrotzEnv.get_dictionary() returns the list of vocabulary words recognized by a game:

```
print('Recognized Vocabulary Words', list(env.get_dictionary()))
```

Note that most games recognize words up to a fixed length of 6 or 9 characters, so truncated words like 'northe' are functionally equivalent to 'northeast.' For more information see *Game Dictionary*.

1.5 Loading and Saving

It's possible to save and load the game state using *jericho.FrotzEnv.get_state()* and *jericho.FrotzEnv.set_state()*:

```
from jericho import *
env = FrotzEnv(rom_path)
state = env.get_state() # Save the game to state
env.step('attack troll') # Oops!
'You swing and miss. The troll neatly removes your head.'
env.set_state(state) # Restore to saved state
```

1.6 Object Tree

The object tree is an internal representation of game state. Jericho provides functions to access all or parts of this tree. For more information see *Object Tree*.

```
all_objects = env.get_world_objects()
print('Me:', env.get_player_object())
print('My Inventory:', env.get_inventory())
print('My Current Location:', env.get_player_location())
```

1.7 Getting Valid Actions

One of the most common difficulties with parser-based text games is identifying which actions are recognized by the parser and applicable in the current location. Jericho's *jericho.FrotzEnv.get_valid_actions()* provides a best-guess list of *valid-actions* that will have an effect on the current game state:

```
>>> from jericho import *
>>> env = FrotzEnv("z-machine-games-master/jericho-game-suite/zork1.z5")
>>> env.reset()
'You are standing in an open field west of a white house, with a boarded front door.␣
↪There is a small mailbox here.'
>>> valid_actions = env.get_valid_actions()
['south', 'open mailbox', 'west', 'north']
```

1.8 Walkthroughs

Jericho provides walkthroughs for supported games using *jericho.FrotzEnv.get_walkthrough()*. To use the walkthrough, it is necessary to reset the environment with the desired seed:

```
>>> from jericho import *
>>> env = FrotzEnv("z-machine-games-master/jericho-game-suite/zork1.z5")
>>> walkthrough = env.get_walkthrough()
>>> for act in walkthrough:
>>>     env.step(act)
```

CHAPTER 2

Frotz Environment

The Frotz Environment is the interface between a learning agent and an interactive fiction game.

class jericho.**FrotzEnv**(*story_file*, *seed=None*)

The Frotz Environment is a fast interface to Z-Machine games.

Parameters

- **story_file** (*path*) – Path to a Z-machine rom file (.z3/.z5/.z6/.z8).
- **seed** (*int*) – Seed the random number generator used by the emulator. Default: use walkthrough's seed if it exists,

  otherwise use value of -1 which changes with time.

bindings

Gets information pertaining to the currently loaded game. Returns a dictionary with the following keys:

- *name*: Name of the game. E.g. *zork1*
- *rom*: Name of the file used to load the game. E.g. *zork1.z5*
- *walkthrough*: A walkthrough for the game.
- *seed*: Seed necessary to replicate the walkthrough.
- *grammar*: List of action templates for the game.
- *max_word_length*: Maximum number of characters per word recognized by the parser.

Returns Dictionary containing game-specific bindings if it exists. Otherwise, an empty dictionary.

Example

```
>>> import jericho
>>> env = jericho.FrotzEnv('zork1')
>>> env.bindings
{
```

(continues on next page)

(continued from previous page)

```
'name': 'zork1',
'rom': 'zork1.z5',
'seed': 12,
'max_word_length': 6,
'minimal_actions': 'Ulysses/wait/pray/inventory/go down/...',
'grammar': 'again/g;answer/reply;back;barf/chomp/...',
'walkthrough': 'N/N/U/Get egg/D/S/E/Open window/...'
}
```

Note: Walkthroughs are defined for only a few games.

close()
Cleans up the FrotzEnv, freeing any allocated memory.

copy()
Forks this FrotzEnv instance.

game_over()
Returns *True* if the game is over and the player has lost.

get_dictionary()
Returns a list of *jericho.DictionaryWord* words recognized by the game's parser. See *Game Dictionary*.

get_inventory()
Returns a list of *jericho.ZObject* in the player's posession.

get_max_score()
Returns the integer maximum possible score for the game.

get_moves()
Returns the integer number of moves taken by the player in the current episode.

get_object(*obj_num*)
Returns a *jericho.ZObject* with the corresponding number or *None* if the object doesn't exist in the *Object Tree*.

Parameters **obj_num** (*int*) – Object number between 0 and len(get_world_objects()).

get_player_location()
Returns the *jericho.ZObject* corresponding to the location of the player in the world.

get_player_object()
Returns the *jericho.ZObject* corresponding to the player.

get_score()
Returns the integer current game score.

get_state()
Returns the internal game state. This state can be subsequently restored using *jericho.FrotzEnv.set_state()*.

Returns Tuple of (ram, stack, pc, sp, fp, frame_count, rng).

```
>>> from jericho import *
>>> env = FrotzEnv(rom_path)
>>> state = env.get_state()
>>> env.step('attack troll') # Oops!
```

(continues on next page)

(continued from previous page)

```
'You swing and miss. The troll neatly removes your head.'
>>> env.set_state(state) # Whew, let's try something else
```

get_valid_actions(*use_object_tree=True*, *use_ctypes=True*, *use_parallel=True*)
Attempts to generate a set of unique valid actions from the current game state.

Parameters

- **use_object_tree** (*boolean*) – Query the *Object Tree* for names of surrounding objects.
- **use_ctypes** (*boolean*) – Uses the optimized ctypes implementation of valid action filtering.
- **use_parallel** (*boolean*) – Uses the parallized implementation of valid action filtering.

Returns A list of valid actions.

get_walkthrough()
Returns the walkthrough for the game.

Returns A list containing walkthrough action strings needed to complete the game.

Note: To reproduce the walkthrough it's also necessary to reset the environment with use_walkthrough_seed=True.

---

get_world_objects(*clean=False*)
Returns an array containing all the *jericho.ZObject* in the game.

Parameters **clean** (*Boolean*) – If True, disconnects noisy objects like Zork1's thief from the Object Tree. This is mainly useful if using world objects as an indication of unique game state.

Returns Array of *jericho.ZObject*.

Example

```
>>> from jericho import *
>>> env = FrotzEnv('zork1.z5')
>>> env.get_world_objects()
 Obj0:  Parent0 Sibling0 Child0 Attributes [] Properties [],
 Obj1: pair hands Parent247 Sibling2 Child0 Attributes [14, 28] Properties␣
↪[18, 16],
 Obj2: zorkmid Parent247 Sibling3 Child0 Attributes [] Properties [18, 17],
 Obj3: way Parent247 Sibling5 Child0 Attributes [14] Properties [18, 17, 16],
 Obj4: cretin Parent180 Sibling181 Child0 Attributes [7, 9, 14, 30]␣
↪Properties [18, 17, 7],
 Obj5: you Parent247 Sibling6 Child0 Attributes [30] Properties [18, 17],
 ...
 Obj250: board Parent249 Sibling73 Child0 Attributes [14] Properties [18, 17]
```

get_world_state_hash()
Returns a MD5 hash of the clean world-object-tree. Such a hash may be useful for identifying when the agent has reached new states or returned to existing ones.

Example

```
>>> env = FrotzEnv('zork1.z5')
>>> env.reset()
# Start at West of the House with the following hash
>>> env.get_world_state_hash()
'79c750fff4368efef349b02ff50ffc23'
>>> env.step('n')
# Moving to North of House changes the hash
>>> get_world_state_hash(env)
'8a3a8538c0019a69128f755e4b719fbd'
>>> env.step('w')
# Moving back to West of House we recover the original hash
>>> env.get_world_state_hash()
'79c750fff4368efef349b02ff50ffc23'
```

load(*story_file*, *seed=None*)
Loads a Z-Machine game.

Parameters

- **story_file** (*path*) – Path to a Z-machine rom file (.z3/.z5/.z6/.z8).
- **seed** (*int*) – Seed the random number generator used by the emulator. Default: use walkthrough's seed if it exists,

  otherwise use value of -1 which changes with time.

reset()
Resets the game.

Parameters **use_walkthrough_seed** – Seed the emulator to reproduce the walkthrough.

Returns A tuple containing the initial observation, and a dictionary of info.

Return type string, dictionary

seed(*seed=None*)
Changes seed used for the emulator's random number generator.

Parameters **seed** – Seed the random number generator used by the emulator. Default: use walkthrough's seed if it exists,

otherwise use value of -1 which changes with time.

Returns The value of the seed.

Note: *jericho.FrotzEnv.reset()* must be called before the seed takes effect.

---

set_state(*state*)
Sets the game's internal state.

Parameters **state** (*tuple*) – Tuple of (ram, stack, pc, sp, fp, frame_count, rng) as obtained by *jericho.FrotzEnv.get_state()*.

```
>>> from jericho import *
>>> env = FrotzEnv(rom_path)
>>> state = env.get_state()
>>> env.step('attack troll') # Oops!
'You swing and miss. The troll neatly removes your head.'
>>> env.set_state(state) # Whew, let's try something else
```

step(*action*)
Takes an action and returns the next state, reward, termination

Parameters **action** (*string*) – Text command to send to the interpreter.

Returns A tuple containing the game's textual response to the action, the immediate reward, a boolean indicating whether the game is over, and a dictionary of info.

Return type string, float, boolean, dictionary

victory()
Returns *True* if the game is over and the player has won.

CHAPTER 3

Object Tree

The Object Tree is an internal representation of all locations and objects present in a Z-machine game, including the player and their inventory.

Each object has *parent*, *child*, and *sibling* object. Parent objects typically contain the child objects. For example if we have a parent object *Treasure Chest*, it may contain a child object *Cutlass* that has a sibling object *Eye Patch*. And the treasure chest may have a parent object *Sandy Beach* corresponding to the location of the chest. If and when the player reaches the Sandy Beach, the object tree will be rearranged so that the player-object's sibling will become the Treasure Chest.

Objects additionally have attributes which are set and unset to keep track of the state of an object. For example a *window* can either be in the open or closed position; this is tracked through its attributes. Finally, the *properties* of an object are numerical values that determine how the object may be interacted with.

For a great tutorial on the Object Tree, see Z-Machine Standards.

In Jericho, the full object tree can be accessed with *jericho.FrotzEnv.get_world_objects()*. Jericho also provides shortcuts to particular objects of interest such as the player *jericho.FrotzEnv.get_player_object()*, the player's location *jericho.FrotzEnv.get_player_location()*, and the player's inventory *jericho.FrotzEnv.get_inventory()*.

3.1 ZObject

Jericho's ZObject is a ctypes structure that providing access to the name, parent, sibling, child, attributes, and properties of an object.

class jericho.**ZObject**

A Z-Machine Object contains the following fields:

Parameters

- **num** (*int*) – Object number
- **name** (*string*) – Short object name
- **parent** (*int*) – Object number of parent

- **sibling** (*int*) – Object number of sibling
- **child** (*int*) – Object number of child
- **attr** (*array*) – 32-bit array of object attributes
- **properties** (*list*) – object properties

Example

```
>>> from jericho import *
>>> env = FrotzEnv('zork1.z5')
>>> env.get_player_object()
Obj4: cretin Parent180 Sibling181 Child0 Attributes [7, 9, 14, 30] Properties [18,
 ↪ 17, 7]
```

CHAPTER 4

Game Dictionary

Jericho allows access to the dictionary of a game through *jericho.FrotzEnv.get_dictionary()*. The dictionary contains a list of *jericho.DictionaryWord* that are recognized by the game's parser.

class jericho.**DictionaryWord**(*word*, *is_noun=False*, *is_verb=False*, *is_prep=False*, *is_meta=False*, *is_plural=False*, *is_dir=False*, *is_adj=False*, *is_special=False*)

A word recognized by a game's parser.

Parameters

- **word** (*string*) – The dictionary word itself.
- **is_noun** (*boolean*) – Is the word a noun.
- **is_verb** (*boolean*) – Is the word a verb.
- **is_prep** (*boolean*) – Is the word a preposition.
- **is_meta** (*boolean*) – Used for meta commands like 'verbose' and 'script'
- **is_plural** (*boolean*) – Used for pluralized nouns (e.g. bookcases, rooms)
- **is_dir** (*boolean*) – Used for direction words (e.g. north, aft)
- **is_special** (*boolean*) – Used for special commands like 'again' which repeats last action.

Example

```
>>> from jericho import *
>>> env = FrotzEnv('zork1.z5')
>>> vocab = env.get_dictionary()
[$ve, ., ,, ", a, across, activa, advent, ... zork, zorkmi, zzmgck]
>>> [w for w in vocab if w.is_noun]
[advent, advert, air, air-p, altar, art, aviato, ax, axe, bag, ... ]
>>> [w for w in vocab if w.is_adj]
[ancien, antiqu, bare, beauti, birds, black, bloody, blue, ... ]
```

Note: Many parsers will recognize only the first six or nine characters of each word. For example *northwest* is functionally equivalent to *northw*.

Warning: Not all games specify the parts of speech for dictionary words.

CHAPTER 5

Template Action Generator

The TemplateActionGenerator is designed to facilitate action generation using a game-specific template-action space.

class jericho.template_action_generator.**TemplateActionGenerator**(*rom_bindings*)

Generates actions using the template-action-space.

Parameters **rom_bindings** (*Dictionary*) – Game-specific bindings from *jericho.FrotzEnv.bindings()*.

generate_actions(*objs*)

Given a list of objects present at the current location, returns a list of possible actions. This list represents all combinations of templates filled with the provided objects.

Parameters **objs** (*List of strings*) – Candidate interactive objects present at the current location.

Returns List of action-strings.

Example

```
>>> import jericho
>>> env = jericho.FrotzEnv(rom_path)
>>> interactive_objs = ['phone', 'keys', 'wallet']
>>> env.act_gen.generate_actions(interactive_objs)
['wake', 'wake up', 'wash', ..., 'examine wallet', 'remove phone', 'taste keys
↪']
```

generate_template_actions(*objs*, *obj_ids*)

Given a list of objects and their corresponding vocab_ids, returns a list of possible TemplateActions. This list represents all combinations of templates filled with the provided objects.

Parameters

- **objs** (*List of strings*) – Candidate interactive objects present at the current location.
- **obj_ids** (*List of int*) – List of ids corresponding to the tokens of each object.

Returns List of *jericho.defines.TemplateAction*.

Example

```
>>> import jericho
>>> env = jericho.FrotzEnv(rom_path)
>>> interactive_objs = ['phone', 'keys', 'wallet']
>>> interactive_obj_ids = [718, 325, 64]
>>> env.act_gen.generate_template_actions(interactive_objs, interactive_obj_
↪ids)
[
  TemplateAction(action='wake', template_id=0, obj_ids=[]),
  TemplateAction(action='wake up', template_id=1, obj_ids=[]),
  ...
  TemplateAction(action='turn phone on', template_id=55, obj_ids=[718]),
  TemplateAction(action='put wallet on keys', template_id=65, obj_ids=[64,␣
↪325])
 ]
```

CHAPTER 6

Utilities

jericho.util.**chunk**(*items: List[T], n: int*) → List[List[T]]
Split a list of items into *n* chunks.

The first *len(items) % n* chunks will contain one more item.

Args: items (List): list of items to be split. n (int): number of chunks to return.

Returns: List[List]: Chunks containing the items.

References: https://stackoverflow.com/a/2135920

jericho.util.**clean**(*s*)
Cleans a string for compact output.

Example

```
>>> clean('*This string\nneeds a good clean.--\n\n*')
'This string needs a good clean.'
```

jericho.util.**extract_objs**(*text*)
Uses Spacy to extract a set of tokens corresponding to possible objects in the provided text.

Example

```
>>> extract_objs('The quick brown fox jumped over the lazy dog.')
{'brown', 'dog', 'fox', 'lazy', 'quick'}
```

Note: Returns adjectives as well as nouns since many games allow players to refer to objects using adjectives.

jericho.util.**get_subtree**(*obj_nb*, *full_object_tree*)
Traverses the object tree, returning a list containing full subtree at obj_nb.

Parameters

- **obj_nb** (*int*) – jericho.ZObject.num corresponding to the root of the subtree to collect.

- **full_object_tree** (*list*) – Full list of objects as given by Jericho's get_world_objects().

Returns A list *jericho.ZObject* of all descendents and siblings of obj_nb.

Example

```
>>> env = FrotzEnv('zork1.z5')
>>> world_objs = env.get_world_objects()
>>> env.get_player_object()
Obj4: cretin Parent180 Sibling181 Child0 Attributes [7, 9, 14, 30] Properties [18,
↪ 17, 7]
>>> get_subtree(4, world_objs)
[
  Obj4: cretin Parent180 Sibling181 Child0 Attributes [7, 9, 14, 30] Properties␣
↪[18, 17, 7],
  Obj181: door Parent180 Sibling160 Child0 Attributes [14, 23] Properties [18, 17,
↪ 16],
  Obj160: mailbox Parent180 Sibling0 Child161 Attributes [13, 19] Properties [18,␣
↪17, 16, 10],
  Obj161: leaflet Parent160 Sibling0 Child0 Attributes [16, 17, 26] Properties␣
↪[18, 16, 15, 11, 8]
]
```

jericho.util.**recognized**(*response*)

Returns *True* if the game's response indicates the last action was successfully parsed.

Parameters **response** (*string*) – The game's textual response to the last action.

Example

```
>>> recognized('I dont know the word "Azerbijan".')
False
>>> recognized('The vines block your way.')
True
```

Note: Invalid actions may often be recognzied. Only ungrammatical actions are detected by this method.

jericho.util.**tokenize**(*str*)

Returns a list of tokens in a string (using Spacy).

Example

```
>>> tokenize('This is an example sentence.')
['this', 'is', 'an', 'example', 'sentence', '.']
```

jericho.util.**unabbreviate**(*action*)

Returns an unabbreviated version of a text action.

Example

```
>>> unabbreviate('nw')
'northwest'
>>> unabbreviate('x sewer')
'examine sewer'
```

jericho.util.**verb_usage_count**(*verb*, *max_word_length=None*)

Returns the number of times that a particular verb appears across all clubfloyd transcripts.

Params verb Retrieve a usage count for this verb.

Example

```
>>> verb_usage_count('take')
8909
>>> verb_usage_count('jump')
786
```

CHAPTER 7

Defines

jericho.defines.**ILLEGAL_ACTIONS**
List of illegal actions that manipulate game state.

jericho.defines.**BASIC_ACTIONS = ['north', 'south', 'west', 'east', 'northwest', 'southwest**
List of basic actions applicable to almost any game.

jericho.defines.**NO_EFFECT_ACTIONS = ['examine', 'x', 'look', 'l', 'i', 'inventory', 'gaze']**
List of actions that usually don't change the world state.

jericho.defines.**UNRECOGNIZED_REGEXPS**
List of regular expressions meant to capture game responses that indicate the last action was unrecognized. This list covers most common paterns. However, some games like *loose.z5* and *lostpig.z8* write custom responses that may note be captured.

jericho.defines.**ABBRV_DICT = {'d': 'down', 'e': 'east', 'g': 'again', 'i': 'inventory',**
The action abbreviation dictionary contains abbreviations of common actions.

class jericho.defines.**TemplateAction**(*action*, *template_id*, *obj_ids*)
A TemplateAction provides a class to bundle a textual action with the *template_id* and *object_ids* that generated the action.

Parameters

- **action** (*string*) – The action text.
- **template_id** (*int*) – The numerical index of the template used to generate the action.
- **obj_ids** (*list*) – List of vocabulary ids used to fill in the template.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

j

Index

U

V

Z