MiniHack

MiniHack Team @ Facebook AI Research, UCL, and Oxford

Feb 15, 2024

GETTING STARTED

1 MiniHack Level Editor 3

2 Language Wrapper 5

3 Papers using MiniHack 7

4 Installation 9

5 Submitting New Environments 11

6 Trying out MiniHack 13

7 Baseline Agents 15

8 Citation 17

9 Contributions and Maintenance 19

Python Module Index 115

Index 117

MiniHack is a sandbox framework for easily designing rich and diverse environments for Reinforcement Learning (RL). Based on the game of NetHack, MiniHack uses the NetHack Learning Environment (NLE) to communicate with the game and to provide a convenient interface for customly created RL training and test environments of varying complexity. Check out our NeurIPS 2021 paper and recent blogpost.

MiniHack comes with a large list of challenging environments. However, it is primarily built for easily designing new ones. The motivation behind MiniHack is to be able to perform RL experiments in a controlled setting while being able to increasingly scale the complexity of the tasks.

To do this, MiniHack leverages the so-called description files written using a human-readable probabilistic-programming-like domain-specific language. With just a few lines of code, people can generate a large variety of Gym environments, controlling every little detail, from the location and types of monsters, to the traps, objects, and terrain of the level, all while introducing randomness that challenges generalization capabilities of RL agents. For further details, we refer users to our brief overview, detailed tutorial, or interactive notebook.

Our documentation will walk you through everything you need to know about MiniHack, step-by-step, including information on how to get started, configure environments or design new ones, train baseline agents, and much more.

CHAPTER

ONE

MINIHACK LEVEL EDITOR

The MiniHack Level Editor allows to easily define MiniHack environments inside a browser using a convenient drag-and-drop functionality. The source code is available here.

CHAPTER

TWO

LANGUAGE WRAPPER

We thank ngoodger for implementing the NLE Language Wrapper that translates the non-language observations from Net/MiniHack tasks into similar language representations. Actions can also be optionally provided in text form which are converted to the Discrete actions of the NLE.

CHAPTER

THREE

PAPERS USING MINIHACK

- Raparthy et al. Learning to Solve New sequential decision-making Tasks with In-Context Learning (Meta AI, UCL, FMDM 2023)
- Nottingham et al. Selective Perception: Learning Concise State Descriptions for Language Model Actors (UC Irvine, FMDM 2023)
- Prakash et al. LLM Augmented Hierarchical Agents (Maryland, JHU, LangRob 2023)
- Castanyer et al. Improving Intrinsic Exploration by Creating Stationary Objectives (Mila, Ubisoft, Nov 2023)
- Henaff et al. A Study of Global and Episodic Bonuses for Exploration in Contextual MDPs (Meta AI, UCL, ICML 2023)
- Bagaria et al. Scaling Goal-based Exploration via Pruning Proto-goals (Brown, DeepMind, Feb 2023)
- Carvalho et al. Composing Task Knowledge with Modular Successor Feature Approximators (UMich, Oxford, LGAI, ICLR 2023)
- Kessler et al. The Surprising Effectiveness of Latent World Models for Continual Reinforcement Learning (Oxford, Polish Academy of Sciences, DeepRL Workshop 2022)
- Wagner et al. Cyclophobic Reinforcement Learning (HHU Düsseldorf, TU Dortmund, DeepRL Workshop 2022)
- Henaff et al. Integrating Episodic and Global Bonuses for Efficient Exploration (Meta AI, UCL, DeepRL Workshop 2022)
- Jiang et al. Grounding Aleatoric Uncertainty in Unsupervised Environment Design (FAIR, UCL, Berkeley, Oxford, NeurIPS 2022)
- Henaff et al. Exploration via Elliptical Episodic Bonuses (Meta AI, UCL, NeurIPS 2022)
- Mu et al. Improving Intrinsic Exploration with Language Abstractions (Stanford, UW, Meta AI, UCL, NeurIPS 2022)
- Chester et al. Oracle-SAGE: Planning Ahead in Graph-Based Deep Reinforcement Learning (RMIT University, Sept 2022)
- Walker et al. Unsupervised representational learning with recognition-parametrised probabilistic models (UCL, Sept 2022)
- Matthews et al. Hierarchical Kickstarting for Skill Transfer in Reinforcement Learning (UCL, Meta AI, Oxford, CoLLAs 2022)
- Powers et al. CORA: Benchmarks, Baselines, and a Platform for Continual Reinforcement Learning Agents (CMU, Georgia Tech, AI2, CoLLAs 2022)
- Nottingham et al. Learning to Query Internet Text for Informing Reinforcement Learning Agents (UC Irvine, May 2022)

- Matthews et al. SkillHack: A Benchmark for Skill Transfer in Open-Ended Reinforcement Learning (UCL, Meta AI, Oxford, April 2022)
- Parker-Holder et al. Evolving Curricula with Regret-Based Environment Design (Oxford, Meta AI, UCL, Berkeley, ICML 2022)
- Parker-Holder et al. That Escalated Quickly: Compounding Complexity by Editing Levels at the Frontier of Agent Capabilities (Oxford, FAIR, UCL, Berkeley, DeepRL Workshop 2021)
- Samvelyan et al. MiniHack the Planet: A Sandbox for Open-Ended Reinforcement Learning Research (FAIR, UCL, Oxford, NeurIPS 2021)

Open a pull request to add papers.

CHAPTER

FOUR

INSTALLATION

The simplest way to install MiniHack is through pypi:

```
pip install minihack
```

4.1 Extending MiniHack

If you wish to extend MiniHack, please install the package as follows:

```
git clone https://github.com/facebookresearch/minihack
cd minihack
pip install -e ".[dev]"
pre-commit install
```

See the full installation guide for further information on installing and extending MiniHack on different platforms, as well as pre-installed Dockerfiles.

CHAPTER

FIVE

SUBMITTING NEW ENVIRONMENTS

For submitting your own MiniHack-based environment to our zoo of public environments, please follow the instructions here.

CHAPTER

SIX

TRYING OUT MINIHACK

MiniHack uses the popular Gym interface for the interactions between the agent and the environment. A pre-registered MiniHack environment can be used as follows:

```
import gym
import minihack
env = gym.make("MiniHack-River-v0")
env.reset() # each reset generates a new environment instance
env.step(1)  # move agent '@' north
env.render()
```

To see the list of all MiniHack environments, run:

```
python -m minihack.scripts.env_list
```

The following scripts allow to play MiniHack environments with a keyboard:

```
# Play the MiniHack in the Terminal as a human
python -m minihack.scripts.play --env MiniHack-River-v0

# Use a random agent
python -m minihack.scripts.play --env MiniHack-River-v0  --mode random

# Play the MiniHack with graphical user interface (gui)
python -m minihack.scripts.play_gui --env MiniHack-River-v0
```

NOTE: If the package has been properly installed one could run the scripts above with `mh-envs`, `mh-play`, and `mh-guiplay` commands.

CHAPTER

SEVEN

BASELINE AGENTS

In order to get started with MiniHack environments, we provide a variety of baselines agent integrations.

7.1 TorchBeast

A TorchBeast agent is bundled in `minihack.agent.polybeast` together with a simple model to provide a starting point for experiments. To install and train this agent, first install torchbeast by following the instructions here, then use the following commands:

```
pip install -e ".[polybeast]"
python -m minihack.agent.polybeast.polyhydra env=MiniHack-Room-5x5-v0 total_steps=100000
```

More information on running our TorchBeast agents, and instructions on how to reproduce the results of the paper, can be found here. The learning curves for all of our polybeast experiments can be accessed in our Weights&Biases repository.

7.2 RLlib

An RLlib agent is provided in `minihack.agent.rllib`, with a similar model to the torchbeast agent. This can be used to try out a variety of different RL algorithms. To install and train an RLlib agent, use the following commands:

```
pip install -e ".[rllib]"
python -m minihack.agent.rllib.train algo=dqn env=MiniHack-Room-5x5-v0 total_
↪steps=1000000
```

More information on running RLlib agents can be found here.

7.3 Unsupervised Environment Design

MiniHack also enables research in *Unsupervised Environment Design*, whereby an adaptive task distribution is learned during training by dynamically adjusting free parameters of the task MDP. Check out the `ucl-dark/paired` repository for replicating the examples from the paper using the PAIRED.

CHAPTER

EIGHT

CITATION

If you use MiniHack in your work, please cite:

```
@inproceedings{samvelyan2021minihack,
  title={MiniHack the Planet: A Sandbox for Open-Ended Reinforcement Learning Research},
  author={Mikayel Samvelyan and Robert Kirk and Vitaly Kurin and Jack Parker-Holder and␣
↪Minqi Jiang and Eric Hambro and Fabio Petroni and Heinrich Kuttler and Edward␣
↪Grefenstette and Tim Rockt{\"a}schel},
  booktitle={Thirty-fifth Conference on Neural Information Processing Systems Datasets␣
↪and Benchmarks Track (Round 1)},
  year={2021},
  url={https://openreview.net/forum?id=skFwlyefkWJ}
}
```

If you use our example ported environments, please cite the original papers: MiniGrid (see license, bib), Boxoban (see license, bib).

CHAPTER NINE

CONTRIBUTIONS AND MAINTENANCE

MiniHack was built and is maintained by Meta AI (FAIR), UCL DARK and University of Oxford. We welcome contributions to MiniHack. If you are interested in contributing, please see this document. Our maintenance plan can be found here.

9.1 Installation

MiniHack is available on pypi and can be installed as follows:

```
pip install minihack
```

We advise using a conda environment for this:

```
conda create -n minihack python=3.8
conda activate minihack
pip install minihack
```

Note: Please refer to NLE installation instructions when having NLE-related dependency issues on **MacOS** and **Ubuntu 18.04**. NLE requires `cmake>=3.15` to be installed when building the package.

Note: **Windows** users should use *Docker*.

Note: Baseline agents have separate installation instructions. See here for more details.

9.1.1 Extending MiniHack

If you wish to extend MiniHack, please install the package as follows:

```
git clone https://github.com/facebookresearch/minihack
cd minihack
pip install -e ".[dev]"
pre-commit install
```

9.1.2 Docker

We have provided several Dockerfiles for building images with pre-installed MiniHack. Please follow the instructions described here.

9.2 Trying out MiniHack

MiniHack uses the popular Gym interface for the interactions between the agent and the environment.

A pre-registered MiniHack environment can be used as follows:

```
import gym
import minihack
env = gym.make("MiniHack-River-v0")
env.reset() # each reset generates a new environment instance
env.step(1)  # move agent '@' north
env.render()
```

The `gym.make` command can also be used to override specific environment parameters:

```
env = gym.make(
   "MiniHack-River-v0",
   observation_keys=("pixel", "glyphs", "colors", "chars"),
   max_episode_steps=100,
)
```

To see the list of all MiniHack environments, run:

```
python -m minihack.scripts.env_list
```

9.2.1 Playing as a human

The following scripts allow to play MiniHack environments with a keyboard:

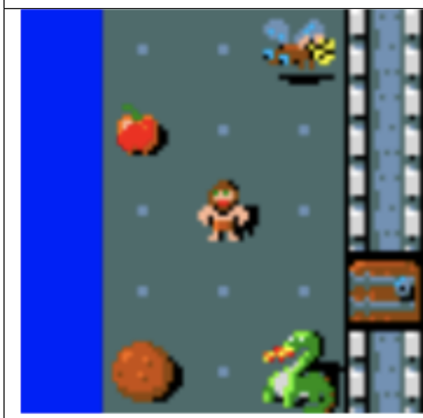
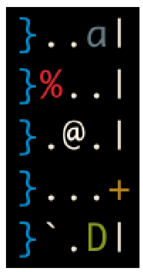
```
# Play the MiniHack in the Terminal as a human
$ python -m minihack.scripts.play --env MiniHack-River-v0

# Use a random agent
$ python -m minihack.scripts.play --env MiniHack-River-v0  --mode random

# Play the MiniHack with graphical user interface (gui)
$ python -m minihack.scripts.play_gui --env MiniHack-River-v0
```

Note: If the package has been properly installed, one could run the scripts above with `mh-envs`, `mh-play`, and `mh-guiplay` commands.

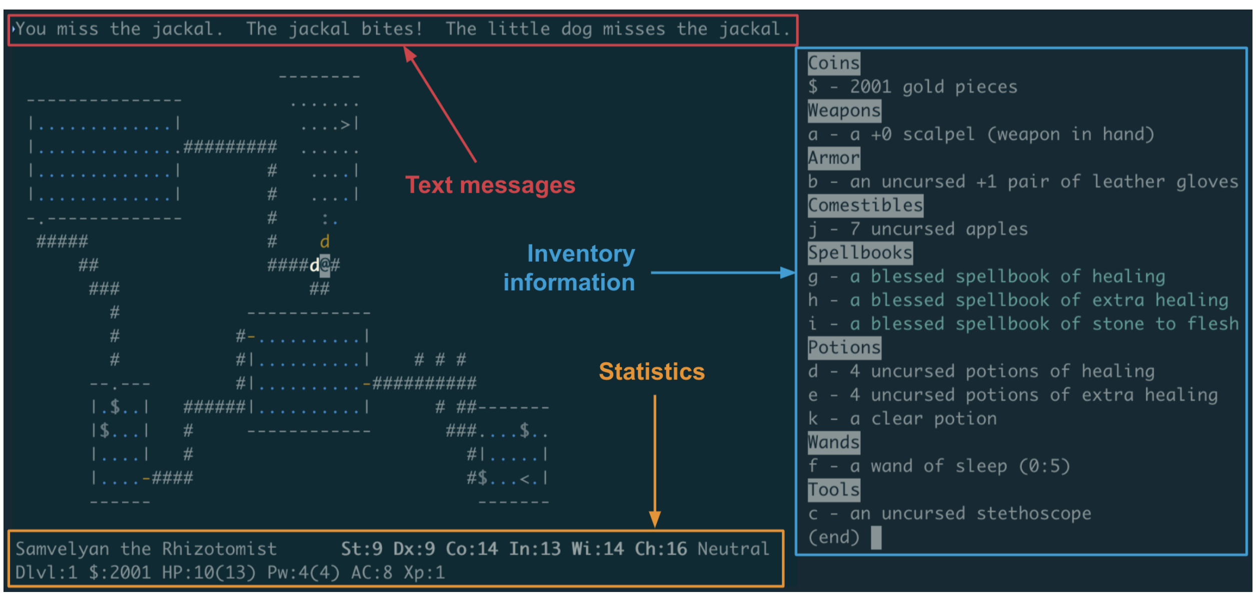
9.3 Observation Spaces

9.3.1 Overview

MiniHack supports several forms of observations, including global or agent-centred viewpoints (or both) of the grid. The table bellow illustrateds three forms of agent-centred observations of the grid of the map in MiniHack.

<table>
<tr><th>Pixel Observations</th><th>Symbolic Observations</th><th>Textual Observations</th></tr>
<tr><td></td><td>
</td><td>
<table>
<tr><td>water</td><td>floor</td><td>floor</td><td>killer bee</td><td>wall</td></tr>
<tr><td>water</td><td>an apple</td><td>floor</td><td>floor</td><td>wall</td></tr>
<tr><td>water</td><td>floor</td><td>agent</td><td>floor</td><td>wall</td></tr>
<tr><td>water</td><td>floor</td><td>floor</td><td>floor</td><td>closed door</td></tr>
<tr><td>water</td><td>a boulder</td><td>floor</td><td>green dragon</td><td>wall</td></tr>
</table>
</td></tr>
</table>

In addition, observations can include player statistics, in-game text messages, and inventory information.



9.3.2 Specifying the Observation Space

MiniHack has a dictionary-structured observation space. Most keys are inherited from NLE, while some are added in MiniHack. To make sure that the desired observations are returned by the environment, the corresponding options should be passed during the initialisation. The `observation_keys` parameter can be used to specify the observation space in any MiniHack environment:

```
env = gym.make(
   "MiniHack-River-v0",
   observation_keys=("glyphs", "chars", "colors", "pixel"),
)
```

Note that using different observation keys can make environments significantly easier or harder.

9.3.3 Options

Name	Description
`glyphs`	a 21×79 matrix of glyphs (ids of entities) on the map. Each glyph represents an entirely unique entity, so these are integers between 0 and 5991. In the standard terminal-based view of NetHack, these glyphs are represented by characters, with colours and other possible visual features.
`chars`	a 21×79 matrix of the characters representing the map.
`colors`	a 21×79 matrix of the colours of each of the characters on the map (some characters represent different objects or monsters depending on their colour).
`specials`	a 21×79 matrix of special extra information about the view of that cell on the map, for example, if the foreground and background colour should be reversed.
`screen_descriptions`	a $21 \times 79 \times 80$ tensor of utf-8 encodings of textual descriptions of each cell present in the map. NetHack provides these textual descriptions (which can be accessed by the user by using the describe action on a specific tile).
`pixel`	a representation of the current screen in image form, where each cell is represented by a $16 \times 16 \times 3$ image, meaning the entire observation is so $336 \times 1264 \times 3$ (with 3 channels for RGB).
`blstats`	a representation of the status line at the bottom of the screen, containing information about the player character's position, health, attributes and other statuses. It comes in the form of a dimension 25 vector.
`message`	the utf-8 encoding of the on-screen message displayed at the top of the screen. It's a 256-dimensional vector.
`inv_glyphs`	a 55-dimensional vector representing the glyphs present in the current inventory view.
`inv_letters`	a 55-dimensional vector representing the letters present in the current inventory view.
`inv_oclasses`	a 55-dimensional vector representing the class of objects present in the current inventory view.
`inv_strs`	a 55×80 matrix containing utf-8 encodings of textual descriptions of objects present in the current inventory view.
`tty_chars`	the character representation of the entire screen, including the message and map, of size 24×80.
`tty_colors`	the color representation of the entire screen, including the message and map, of size 24×80.
`tty_cursor`	the location of the cursor on the screen, a 2-dimensional vector of (x,y) coordinates.

Note: For `glyphs`, `chars`, `colors`, `specials`, `pixel`, `screen_descriptions`, `tty_chars`, and `tty_colors` a cropped observation centered the agent can be used by passing the observation name suffixed with `_crop` (e.g. `chars_crop`). This is a NxN matrix centered on the agent's current location containing the information normally present in the full view. The size of the crop can easily be configured using the `obs_crop_h` and `obs_crop_w` parameters of the environment (9 by default). Cropped observations can facilitate the learning, as the egocentric input makes representation learning easier.

9.4 Action Spaces

9.4.1 Overview

MiniHack has a large, structured and context-sensitive action space. We give practitioners an easy way to restrict the action space in order to promote targeted skill discovery. For example, navigation tasks mostly require movement commands, and occasionally, kicking doors, searching or eating. Skill acquisition tasks, on the other hand, require interactions with objects, e.g. managing the inventory, casting spells, zapping wands, reading scrolls, eating comestibles, quaffing potions, etc. In these tasks 75 actions are used.

The actual game of NetHack uses ASCII inputs, i.e., individual keyboard presses including modifiers like Ctrl and Meta. NLE pre-defines 98 actions, 16 of which are compass directions and 82 of which are command actions.

9.4.2 Specifying the Action Space

The actions used in MiniHack are defined here. The following example shows how to set the action space of the environment to movements towards 8 compass directions with `open`, `kick`, and `search` actions.

```
from nle import nethack
MOVE_ACTIONS = tuple(nethack.CompassDirection)
NAVIGATE_ACTIONS = MOVE_ACTIONS + (
    nethack.Command.OPEN,
    nethack.Command.KICK,
    nethack.Command.SEARCH,
)
env = gym.make(
    "MiniHack-Corridor-R3-v0",
    actions=NAVIGATE_ACTIONS,
)
```

Note that using different observation keys can make environments significantly easier or harder.

9.4.3 Possible Actions

Name	Value	Key	Description
EXTCMD	35	#	perform an extended command
EXTLIST	191	M-?	list all extended commands
ADJUST	225	M-a	adjust inventory letters
ANNOTATE	193	M-A	name current level
APPLY	97	a	apply (use) a tool (pick-axe, key, lamp…)
ATTRIBUTES	24	C-x	show your attributes
AUTOPICKUP	64	@	toggle the pickup option on/off
CALL	67	C	call (name) something
CAST	90	Z	zap (cast) a spell
CHAT	227	M-c	talk to someone
CLOSE	99	c	close a door
CONDUCT	195	M-C	list voluntary challenges you have maintained
DIP	228	M-d	dip an object into something
DOWN	62	>	go down (e.g., a staircase)
DROP	100	d	drop an item

continues on next page

Table 1 – continued from previous page

Name	Value	Key	Description
DROPTYPE	68	D	drop specific item types
EAT	101	e	eat something
ESC	27	C-[	escape from the current query/action
ENGRAVE	69	E	engrave writing on the floor
ENHANCE	229	M-e	advance or check weapon and spell skills
FIRE	102	f	fire ammunition from quiver
FIGHT	70	F	Prefix: force fight even if you don't see a monster
FORCE	230	M-f	force a lock
GLANCE	59	;	show what type of thing a map symbol corresponds to
HELP	63	?	give a help message
HISTORY	86	V	show long version and game history
INVENTORY	105	i	show your inventory
INVENTTYPE	73	I	inventory specific item types
INVOKE	233	M-i	invoke an object's special powers
JUMP	234	M-j	jump to another location
KICK	4	C-d	kick something
KNOWN	92	\	show what object types have been discovered
KNOWNCLASS	96	`	show discovered types for one class of objects
LOOK	58	:	look at what is here
LOOT	236	M-l	loot a box on the floor
MONSTER	237	M-m	use monster's special ability
MORE	13	C-m	read the next message
MOVE	109	m	Prefix: move without picking up objects/fighting
MOVEFAR	77	M	Prefix: run without picking up objects/fighting
OFFER	239	M-o	offer a sacrifice to the gods
OPEN	111	o	open a door
OPTIONS	79	O	show option settings, possibly change them
OVERVIEW	15	C-o	show a summary of the explored dungeon
PAY	112	p	pay your shopping bill
PICKUP	44	,	pick up things at the current location
PRAY	240	M-p	pray to the gods for help
PREVMSG	16	C-p	view recent game messages
PUTON	80	P	put on an accessory (ring, amulet, etc)
QUAFF	113	q	quaff (drink) something
QUIT	241	M-q	exit without saving current game
QUIVER	81	Q	select ammunition for quiver
READ	114	r	read a scroll or spellbook
REDRAW	18	C-r	redraw screen
REMOVE	82	R	remove an accessory (ring, amulet, etc)
RIDE	210	M-R	mount or dismount a saddled steed
RUB	242	M-r	rub a lamp or a stone
RUSH	103	g	Prefix: rush until something interesting is seen
SAVE	83	S	save the game and exit
SEARCH	115	s	search for traps and secret doors
SEEALL	42	*	show all equipment in use
SEETRAP	94	^	show the type of adjacent trap
SIT	243	M-s	sit down
SWAP	120	x	swap wielded and secondary weapons
TAKEOFF	84	T	take off one piece of armor

continues on next page

Table 1 – continued from previous page

Name	Value	Key	Description
TAKEOFFALL	65	A	remove all armor
TELEPORT	20	C-t	teleport around the level
THROW	116	t	throw something
TIP	212	M-T	empty a container
TRAVEL	95	_	travel to a specific location on the map
TURN	244	M-t	turn undead away
TWOWEAPON	88	X	toggle two-weapon combat
UNTRAP	245	M-u	untrap something
UP	60	<	go up (e.g., a staircase)
VERSION	246	M-v	list compile time options
VERSIONSHORT	118	v	show version
WAIT / SELF	46	.	rest one move while doing nothing / apply to self
WEAR	87	W	wear a piece of armor
WHATDOES	38	&	tell what a command does
WHATIS	47	/	show what type of thing a symbol corresponds to
WIELD	119	w	wield (put in use) a weapon
WIPE	247	M-w	wipe off your face
ZAP	112	z	zap a wand

The descriptions are mostly taken from the cmd.c file in the NetHack source code. For a detailed description of these actions, as well as other NetHack commands, we refer the reader to the NetHack guide book.

9.5 Description files

Note: This is a brief overview of the des-file format. An in-depth, visually-aided tutorial can be found *here*.

9.5.1 Overview

MiniHack leverages the description files of NetHack to provide a means to easily design rich and diverse environments. The description files (or des-files) are human-readable specifications of levels: distributions of grid layouts together with monsters, objects on the floor, environment features (e.g. walls, water, lava), etc. The developers of NetHack created a special domain-specific language for describing the levels of the game, called *des-file format*. The des-files can be compiled into binary using the NetHack level compiler, and MiniHack maps them to Gym environments.

Levels defined via des-file format can be fairly rich, as the underlying programming language has support for variables, loops, conditional statements, as well as probability distributions. Crucially, it supports underspecified statements, such as generating a random monster or an object at a random location on the map. Furthermore, it features commands that procedurally generate diverse grid layouts in a single line.

Below we present a brief overview of the different kinds of des-files, how to add entities to levels, and the main sources of randomness that can be used to create a distribution of levels on which to train RL agents. **Please check out our in-depth, visually-aided tutorial here**. We also refer users to the des-file tutorial in NetHack wiki.

Note: MiniHack addionally provides a convenient interface to describe the entire environment directly in Python. Check out our `LevelGenerator` here.

9.5.2 Types of des-files

There are two types of levels that can be created using des-file format, namely *MAZE-type* and *ROOM-type*:

- MAZE-type levels are composed of maps of the level (specified with the `MAP` command) which are drawn using ASCII characters, followed by descriptions of the contents of the level, described in detail below. In MAZE-type environments, the layout of the map is fixed, but random terrain can be created around (or within) that map using the `MAZEWALK` command, which creates a random maze from a given location and filling all available space of a certain terrain type.
- ROOM-type levels are composed of descriptions of rooms (specified by the `ROOM` command), each of which can have its contents specified by the commands described below. Generally, the `RANDOM_CORRIDORS` command is then used to create random corridors between all the rooms so that they are accessible. On creation, the file specifies (or leaves random) the room's type, lighting and approximate location. It is also possible to create sub-rooms (using the `SUBROOM` command) which are rooms guaranteed to be within the outer room and are otherwise specified as normal rooms (but with a location relative to the outer room).

9.5.3 Adding Entities to des-files

As we have seen above, there are multiple ways to define the layout of a level using the des-file format. Once the layout is defined, it is useful to be able to add entities to the level. These could be monsters, objects, traps or other specific terrain features (such as sinks, fountains or altars). In general, the syntax for adding one of these objects is:

```
ENTITY: specification, location, extra-details
```

For example:

```
MONSTER: ('F',"lichen"), (1,1)
OBJECT: ('%',"apple"), (10,10)
TRAP: 'teleportation', (1,1)
SINK: (1,1)
FOUNTAIN: (0,0)
```

Note that many of the details here can instead be set to `random`. In this case, the game engine chooses a suitable value for that argument randomly each time the level is generated. For monsters and objects, this randomness can be controlled by just specifying the class of the monster or object and letting the specific object or monster be chosen randomly. For example:

```
MONSTER: 'F', (1,1)
OBJECT: '%', (10,10)
```

This code would choose a random monster from the Fungus class, and a random object from the Comestible class.

9.5.4 Sources of Randomness in des-files

We have seen how to create either very fixed (MAZE-type) or very random (ROOM-type) levels, and how to add entities with some degree of randomness. The des-file format has many other ways of adding randomness, which can be used to control the level generation process, including where to add terrain and in what way. Many of these methods are used in \texttt{IF} statements, which can be in one of two forms:

```
IF[50%] {
    MONSTER: 'F', (1,1)
} ELSE {
```

(continues on next page)

(continued from previous page)

```
    # ELSE is not always necessary
    OBJECT: '%', (1,1)
}

IF[$variable_name < 15] {
    MONSTER: 'F', (1,1)
}
```

In the first form, a simple percentage is used for the random choice, whereas in the second, a variable (which could have been randomly determined earlier in the file) is used. A natural way to specify this variable is either in other conditional statements (perhaps you randomly add some number of monsters, and want to count the number of monsters you add such that if there are many monsters, you also add some extra weapons for the agent), or through dice notation. Dice notation is used to specify random expressions which resolve to integers (and hence can be used in any place an integer would be). They are of the form \texttt{NdM}, which means to roll N M-sided dice and sum the result. For example:

```
$roll = 2d6
IF[$roll < 7] {
    MONSTER: random, random
}
```

Dice rolls can also be used for accessing arrays, another feature of the des-file format. Arrays are initialised with one or more objects of the same type, and can be indexed with integers (starting at 0), for example:

```
# An array of monster classes
$mon_letters = { 'A', 'L', 'V', 'H' }
# An array of monster names from each monster class respectively
$mon_names = { "Archon", "arch-lich", "vampire lord", "minotaur" }
# The monster to choose
$mon_index = 1d4 - 1
MONSTER:($mon_letters[$mon_index],$mon_names[$mon_index]),(10,18)
```

Another way to perform random operations with arrays is using the `SHUFFLE` command. This command takes an array and randomly shuffles it. This would not work with the above example, as the monster name needs to match the monster class (i.e. we could not use `('A', "minotaur")`). For example:

```
$object = object: { '[',')','*','%' }
SHUFFLE: $object
```

Now the `$object` array will be randomly shuffled. Often, something achievable with shuffling can also be achieved with dice-rolls, but it is simpler to use shuffled arrays rather than dice-rolls (for example, if you wanted to guarantee each of the elements of the array was used exactly once, but randomise the order, it is much easier to just shuffle the array and select them in order rather than try and generate exclusive dice rolls).

9.5.5 Random Terrain Placement

When creating a level, we may want to specify the structure or layout of the level (using a MAZE-type level), but then randomly create the terrain within the level, which will determine accessibility and observability for the agent and monsters in the level. As an example, consider the following example. In this level, we start with an empty 11x9 `MAP`. We first replace 33% of the squares with clouds `'C'`, and then 25% with trees `'T'`. To ensure that any corner is accessible from any other, we create two random-walk lines using `randline` from opposite corners and make all squares on those lines floor `.`. To give the agent a helping hand, we choose a random square in the centre of the room with `rndcoord` (which picks a random coordinate from a selection of coordinates) and place an apple there.

```
MAZE: "mylevel", ' '
GEOMETRY:center,center
MAP
...........
...........
...........
...........
...........
...........
...........
...........
...........
ENDMAP
REGION:(0,0,11,9),lit,"ordinary"
REPLACE_TERRAIN:(0,0,11,9), '.', 'C', 33%
REPLACE_TERRAIN:(0,0,11,9), '.', 'T', 25%
TERRAIN:randline (0,9),(11,0), 5, '.'
TERRAIN:randline (0,0),(11,9), 5, '.'
$center = selection: fillrect (5,5,8,8)
$apple_location = rndcoord $center
OBJECT: ('%', "apple"), $apple_location

$monster = monster: { 'L','N','H','O','D','T' }
SHUFFLE: $monster
$place = { (10,8),(0,8),(10,0) }
SHUFFLE: $place
MONSTER: $monster[0], $place[0], hostile
STAIR:$place[2],down
BRANCH:(0,0,0,0),(1,1,1,1)
```

Several other methods of randomly creating selections such as `filter` (randomly remove points from a selection) and `gradient` (create a selection based on a probability gradient across an area) are described in the NetHack wiki.

9.5.6 Further Information

For more information on the des-file format, be sure to check out our in-depth, visually-aided tutorial here.

9.6 Creating New Environments

9.6.1 Overview

Creating new environments using MiniHack is very simple. There are two main MiniHack base classes to chose from.

MiniHackNavigation ba can be used to design mazes and navigation tasks that only require a small action space. All MiniHack navigation tasks make use of the MiniHackNavigation interface. The in-game multiple-choice question prompts, which are used for interacting with objects and using the inventory, are turned off by default here.

MiniHackSkill provides a convenient mean for designing diverse skill acquisition tasks that require a large action space, interactions with objects and more complex goals. All skill acquisition tasks in MiniHack use this base class. The in-game multiple-choice question prompts is turned on by default.

The quickest way for creating a new environment is to use `gym.make` and pass the description file to the environment:

```
import gym
import minihack

des_file = """
MAZE: "mylevel",' '
GEOMETRY:center,center
MAP
-------------
|.....|.....|
|.....|.....|
|.....+.....|
|.....|.....|
|.....|.....|
-------------
ENDMAP
REGION:(0,0,12,6),lit,"ordinary"
BRANCH:(1,1,6,6),(0,0,0,0)
DOOR:locked,(6,3)
STAIR:(8,3),down
"""
env = gym.make(
    "MiniHack-Navigation-Custom-v0",
    des_file=des_file,
    max_episode_steps=50,
)
```

Additional parameters for the environment can also be passed to `gym.make`, such as `observation_keys`, etc. By default, the goal of the agent is to reach the stair down. However, reward functions in MiniHack can easily be configured. See here for more information.

Alternatively, the users can subclass either MiniHackNavigation or MiniHackSkill classes.

```
from minihack import MiniHackNavigation
from minihack.envs import register
```

(continues on next page)

(continued from previous page)

```
class MiniHackNewTask(MiniHackNavigation):
    def __init__(self, *args, des_file, **kwargs):
        kwargs["max_episode_steps"] = kwargs.pop("max_episode_steps", 1000)
        super().__init__(*args, des_file=des_file, **kwargs)

register(
    id="MiniHack-NewTask-v0",
    entry_point="path.to.file:MiniHackNewTask", # use the actual the path
)
```

For information about the description files, check out our *brief overview*, detailed tutorial or community wiki.

9.6.2 Level Generator

When creating a new MiniHack environment, a description file must be provided. One way of providing this des-file is writing it entirely from scratch. However, this requires learning the des-file format and is more difficult to do programmatically, so as part of MiniHack we provide the LevelGenerator class which provides a convenient wrapper around writing a des-file. The `LevelGenerator` class can be used to create MAZE-type levels with specified heights and widths, and can then fill those levels with objects, monsters and terrain, and specify the start point of the level. Combined with the RewardManager which handles rewards, this enables flexible creation of a wide variety of environments.

The level generator can start with either an empty maze (in which case only height and width are specified, see *Example 1*) or with a pre-drawn map (see *Example 2*). After initialisation, the level generator can be used to add objects, traps, monsters and other terrain features. Terrain can also be added (\cref{code:python} line 9). Once the level is complete, the `get_des()` function returns the des-file which can then be passed to the environment creation.

Example 1 shows how to create a simple skill acquisition task that challenges the agent to eat an apple and wield a dagger that is randomly placed in a 10x10 room surrounded by lava, alongside a goblin and a teleportation trap. Here, a RewardManager is used to specify the tasks that need to be completed.

Example 2 illustrates how to create a labyrinth task. Here, the agent starts near the entrance of a maze and needs to reach its centre. A Minotaur is placed deep inside the maze, which is a powerful monster capable of instantly killing the agent in melee combat. There is a wand of death placed in a random location in the maze. The agent needs to pick it up, and upon seeing the Minotaur, zap it in the direction of the monster. Once the Minotaur is killed, the agent needs to navigate itself towards the staircase (this is the default goal when `RewardManager` is not used). Tools such as Monodraw can help draw the map layout.

9.6.3 Examples

Example 1

Creating a skill task using the LevelGenerator and RewardManager.

```
from minihack import LevelGenerator
from minihack import RewardManager

# Define a 10x10 room and populate it with
# different objects, monster and features
lvl_gen = LevelGenerator(w=10, h=10)
lvl_gen.add_object("apple", "%")
```

(continues on next page)

(continued from previous page)

```
lvl_gen.add_object("dagger", ")")
lvl_gen.add_trap(name="teleport")
lvl_gen.add_sink()
lvl_gen.add_monster("goblin")
lvl_gen.fill_terrain("rect", "L",
    0, 0, 9, 9)

# Define a reward manager
reward_manager = RewardManager()
# +1 reward and termination for eating
# an apple or wielding a dagger
reward_manager.add_eat_event("apple")
reward_manager.add_wield_event("dagger")
# -1 reward for standing on a sink
# but isn't required for terminating
# the episode
reward_manager.add_location_event("sink",
    reward=-1, terminal_required=False)

env = gym.make(
    "MiniHack-Skill-Custom-v0",
    des_file=lvl_gen.get_des(),
    reward_manager=reward_manager,
)
```

Example 2

Creating a MiniHack skill task using LevelGenerator with a pre-defined map layout.

```
from minihack import LevelGenerator

# Define the maze as a string
maze = """
--------------------
|.......|.|........|
|.-----.|.|.-----|.|
|.|...|.|.|......|.|
|.|.|.|.|.|-----.|.|
|.|.|...|....|.|.|.|
|.|.--------.|.|.|.|
|.|..........|...|.|
|.|--------------|.|
|..................|
--------------------
"""
# Set a start and goal positions
lvl_gen = LevelGenerator(map=maze)
lvl_gen.set_start_pos((9, 1))
lvl_gen.add_goal_pos((14, 5))
# Add a Minotaur at fixed position
lvl_gen.add_monster(name="minotaur",
```

(continues on next page)

(continued from previous page)

```
    place=(19, 9))
# Add wand of death
lvl_gen.add_object("death", "/")

env = gym.make(
    "MiniHack-Skill-Custom-v0",
    des_file = lvl_gen.get_des(),
)
```

9.7 Reward Function

9.7.1 Default Configuration

Reward functions in MiniHack can easily be configured. The default reward function of custom MiniHack environments is a sparse reward of +1 for reaching the staircase down (which terminates the episode), and 0 otherwise, with episodes terminating after a configurable number of timesteps. In addition, the agent receives a negative reward of -0.01 if the game timer doesn't progress during a step (e.g. the agent moves towards a wall).

These defaults can be easily adjusted using the following environment flags:

Parameter	Default Value	Description
`reward_win`	1	the reward received upon successfully completing an episode.
`reward_lose`	0	the reward received upon death or aborting.
`penalty_mode`	"constant"	name of the mode for calculating the time step penalty. Can be `constant`, `exp`, `square`, `linear`, or `always`.
`penalty_step`	-0.01	constant applied to amount of frozen steps.
`penalty_time`	0	constant applied to amount of non-frozen steps.

9.7.2 Reward Manager

We also provide an interface for designing custom reward functions. By using the RewardManager, users can control what events give the agent reward, whether those events can be repeated, and what combinations of events are sufficient or required to terminate the episode.

In order to use the reward managers, users need create an instance of the class and pass it to a MiniHack environment. In the example below, the agent receives +1 reward for eating an apple or +2 reward for wielding a dagger (both of which also terminate the episode). In addition, the agent receives -1 reward for standing on a sink, but the episode isn't termianted in this case.

```
from minihack import RewardManager

reward_manager = RewardManager()
reward_manager.add_eat_event("apple", reward=1)
reward_manager.add_wield_event("dagger", reward=2)
reward_manager.add_location_event("sink", reward=-1, terminal_required=False)

env = gym.make("MiniHackSkill",
```

(continues on next page)

(continued from previous page)

```
    des_file=des_file,
    reward_manager=reward_manager)
```

While the basic reward manager supports many events by default, users may want to extend this interface to define their own events. This can be done easily by inheriting from the Event class and implementing the `check` and `reset` methods. Beyond that, custom reward functions can be added to the reward manager through `add_custom_reward_fn` method. These functions take the environment instance, the previous observation, action taken and current observation, and should return a float.

We also provide two ways to combine events in a more structured way. The SequentialRewardManager works similarly to the normal reward manager but requires the events to be completed in the sequence they were added, terminating the episode once the last event is complete. The GroupedRewardManager combines other reward managers, with termination conditions defined across the reward managers (rather than individual events). This allows complex conjunctions and disjunctions of groups of events to specify termination. For example, one could specify a reward function that terminates if a sequence of events (a,b,c) was completed, or all events {d,e,f} were completed in any order and the sequence (g,h) was completed.

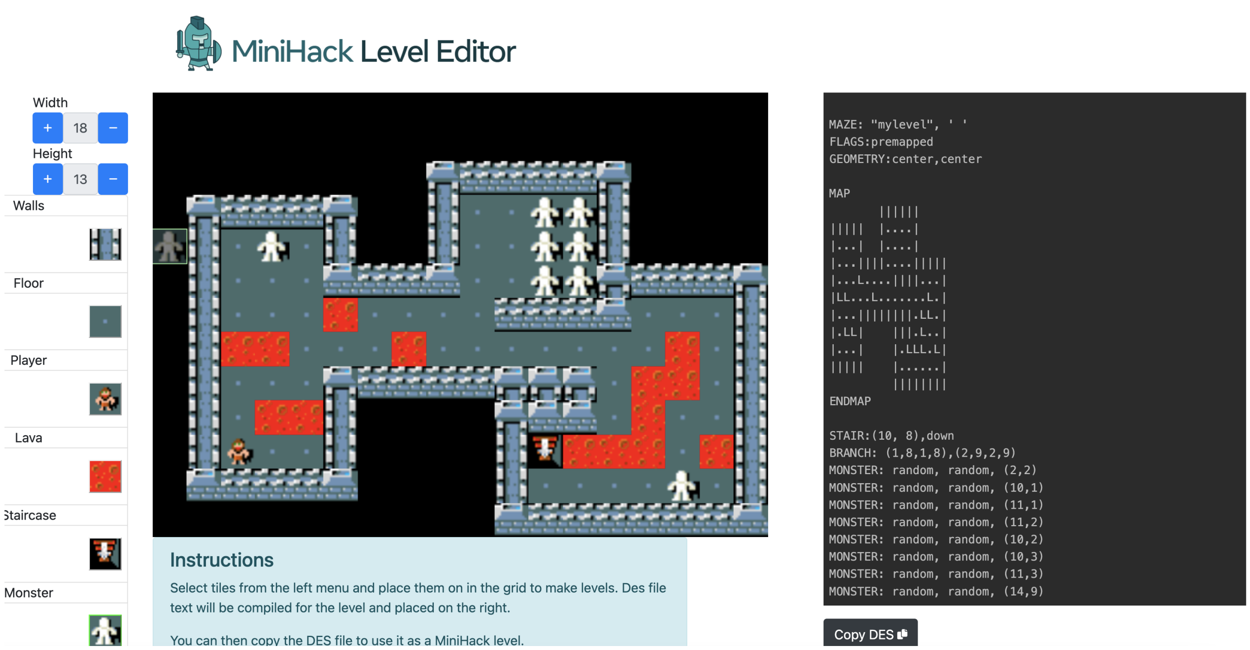
9.8 Level Editor

9.8.1 Overview

The MiniHack Level Editor allows to easily define MiniHack environments inside a browser using a convenient drag-and-drop functionality. Once the level has been designed using the user interface, its description file can be easily copied and applied to the downstream codebase.

9.8.2 Accessing the Level Editor

The MiniHack level editor can be accessed at minihack-editor.github.io. Please follow the instructions on the website for generating description files of the designed levels.



9.9 MiniHack Environment Zoo

This page describes all open-source MiniHack environments developed by its authors as well as members of the community. For adding your own MiniHack environment, please follow the instructions *here*.

9.9.1 Navigation Tasks

MiniHack navigation tasks challenge the agent to reach the goal position by overcoming various difficulties on their way, such as fighting monsters in corridors, crossing rivers by pushing boulders into it, navigating through complex or procedurally generated mazes. These tasks feature a relatively small action space, i.e., movement towards 8 compass directions, and based on the environment, search, kick, open, and eat actions.

Room

These tasks are set in a single square room, where the goal is to reach the staircase down. There are multiple variants of this level.

There are two sizes of the room (`5x5, 15x15`). In the simplest variants, (`MiniHack-Room-5x5-v0` and `MiniHack-Room-15x15-v0`), the start and goal position are fixed. In the `MiniHack-Room-Random-5x5-v0` and `MiniHack-Room-Random-15x15-v0` tasks, the start and goal position are randomised. The rest of the variants add additional complexity to the randomised version of the environment by introducing monsters (`MiniHack-Room-Monster-5x5-v0` and `MiniHack-Room-Monster-15x15-v0`), teleportation traps (`MiniHack-Room-Trap-5x5-v0` and `MiniHack-Room-Trap-15x15-v0`), darkness (`MiniHack-Room-Dark-5x5-v0` and `MiniHack-Room-Dark-15x15-v0`), or all three combined (`MiniHack-Room-Ultimate-5x5-v0` and `MiniHack-Room-Ultimate-15x15-v0`). The agent can attack monsters by moving towards them when located in an adjacent grid cell. Stepping on a lava tile instantly kills the agent. When the room is dark, the agent can only observe adjacent grid cells.

Examples of the `MiniHack-Room-Ultimate-15x15-v0` task:

Reward

The agent receives a reward of +1 for reaching the goal.

Source

Source

All Environments

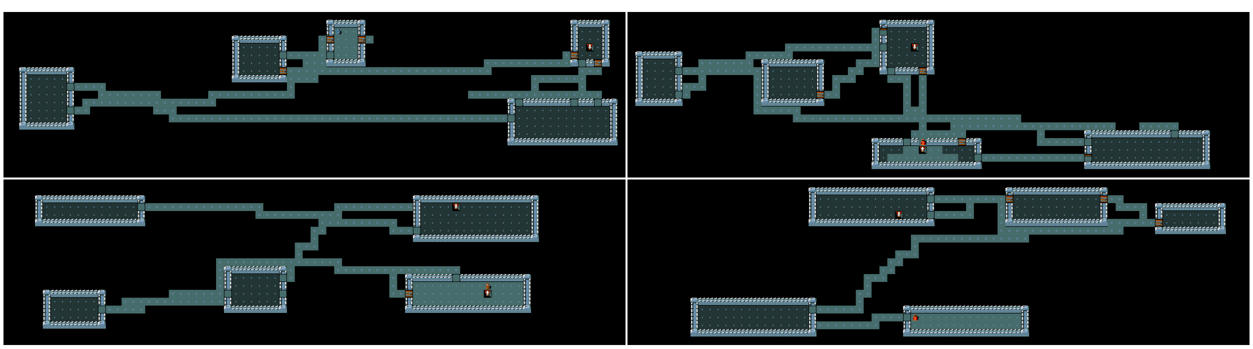
Name	Capability
MiniHack-Room-5x5-v0	Basic Learning
MiniHack-Room-15x15-v0	Basic Learning
MiniHack-Room-Random-5x5-v0	Basic Learning
MiniHack-Room-Random-15x15-v0	Basic Learning
MiniHack-Room-Dark-5x5-v0	Basic Learning
MiniHack-Room-Dark-15x15-v0	Basic Learning
MiniHack-Room-Monster-5x5-v0	Basic Learning
MiniHack-Room-Monster-15x15-v0	Basic Learning
MiniHack-Room-Trap-5x5-v0	Basic Learning
MiniHack-Room-Trap-15x15-v0	Basic Learning
MiniHack-Room-Ultimate-5x5-v0	Basic Learning
MiniHack-Room-Ultimate-15x15-v0	Basic Learning

Corridor

These tasks make use of the `RANDOM_CORRIDORS` command in the `des-file`. The objective is to reach the staircase located in one of the randomly generated rooms. The rooms have randomised positions and sizes. The corridors between the rooms are procedurally generated and are different for every episodes.

Different variants of this environment have different numbers of rooms, making the exploration challenge more difficult (`MiniHack-Corridor-R2-v0`, `MiniHack-Corridor-R3-v0`, and `MiniHack-Corridor-R5-v0` environments are composed of 2, 3, and 5 rooms, respectively).

Examples of the `MiniHack-Corridor-R5-v0` task:



Reward

The agent receives a reward of +1 for reaching the goal.

Source

Source

All Environments

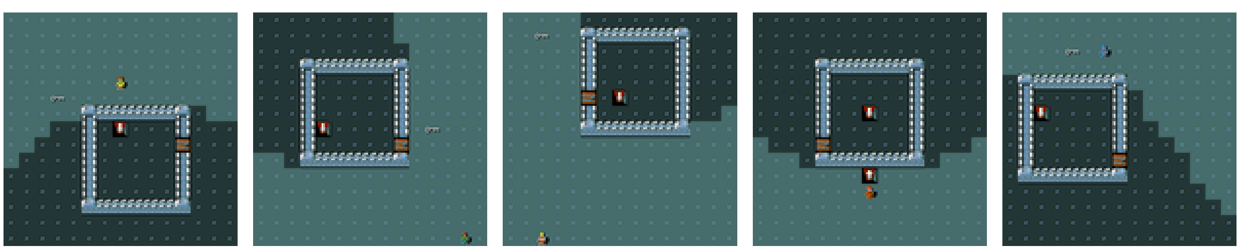
Name	Capability
`MiniHack-Corridor-R2-v0`	Exploration
`MiniHack-Corridor-R3-v0`	Exploration
`MiniHack-Corridor-R5-v0`	Exploration

KeyRoom

These tasks require the agent to pickup a key, navigate to a door, and use the key to unlock the door, reaching the staircase down within the locked room. The action space is the standard movement actions plus the pickup and apply action.

In the simplest variant of this task, (`MiniHack-KeyRoom-Fixed-S5-v0`), the location of the key, door and staircase are fixed. In the rest of the variants these locations randomised. The size the outer room is 5x5 for `MiniHack-KeyRoom-S5-v0` and 15x15 for `MiniHack-KeyRoom-S15-v0`. To increase the difficulty of the tasks, dark versions of the tasks are introduced (`MiniHack-KeyRoom-Dark-S5-v0` and `MiniHack-KeyRoom-Dark-S15-v0`), where the key cannot be seen if it is not in any of the agent's adjacent grid cells.

Examples of the `MiniHack-KeyRoom-S15-v0` task:



Reward

The agent receives a reward of +1 for reaching the goal located in the locked room.

Source

Source.

All Environments

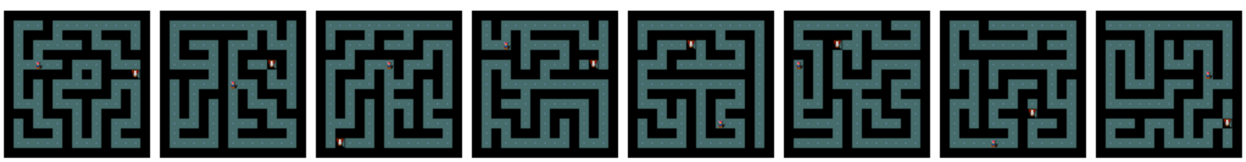
Name	Capability
MiniHack-KeyRoom-Fixed-S5-v0	Exploration
MiniHack-KeyRoom-S5-v0	Exploration
MiniHack-KeyRoom-Dark-S5-v0	Exploration
MiniHack-KeyRoom-S15-v0	Exploration
MiniHack-KeyRoom-Dark-S15-v0	Exploration

MazeWalk

These navigation tasks make use of the `MAZEWALK` command in the `des-file`, which procedurally generates diverse mazes.

We provide three maze sizes of 9x9, 15x15 and 45x19 grids corresponding to `MiniHack-MazeWalk-9x9-v0`, `MiniHack-MazeWalk-15x15-v0`, and `MiniHack-MazeWalk-45x19-v0` environments. In the mapped versions of these tasks (`MiniHack-MazeWalk-Mapped-9x9-v0`, `MiniHack-MazeWalk-Mapped-15x15-v0`, and `MiniHack-MazeWalk-Mapped-45x19-v0`), the map of the maze and the goal's locations are visible to the agent.

Examples of the `MiniHack-MazeWalk-15x15-v0` task:



Reward

The agent receives a reward of +1 for reaching the goal.

Source

Source

All environments

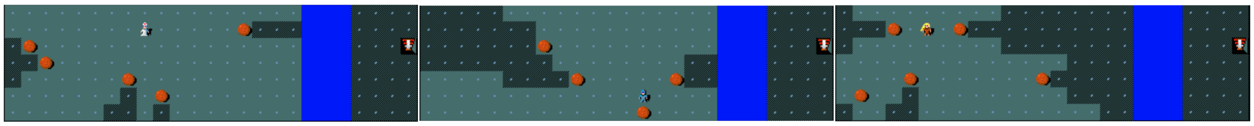
Name	Capability
MiniHack-MazeWalk-9x9-v0	Exploration & Memory
MiniHack-MazeWalk-Mapped-9x9-v0	Exploration & Memory
MiniHack-MazeWalk-15x15-v0	Exploration & Memory
MiniHack-MazeWalk-Mapped-15x15-v0	Exploration & Memory
MiniHack-MazeWalk-45x19-v0	Exploration & Memory
MiniHack-MazeWalk-Mapped-45x19-v0	Exploration & Memory

River

This group of tasks requires the agent to cross a river using boulders and reach the goal located on the other side. Boulders, when pushed into water, create a dry land to walk on, allowing the agent to cross it.

While the `MiniHack-River-Narrow-v0` task can be solved by pushing one boulder into the water, other `MiniHack-River-v0` require the agent plan a sequence of at least two boulder pushes into the river next to each other. In the more challenging tasks of the group, the agent needs to additionally fight monsters (`MiniHack-River-Monster-v0`), avoid pushing boulders into lava rather than water (`MiniHack-River-Lava-v0`), or both (`MiniHack-River-MonsterLava-v0`).

Examples of the `MiniHack-River-v0` task:



Reward

The agent receives a reward of +1 for reaching the goal.

Source

Source

All Environments

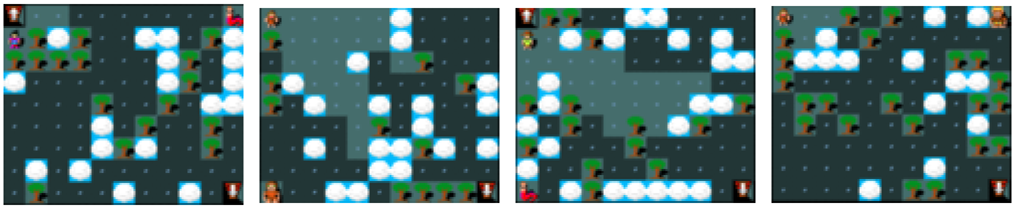
Name	Capability
`MiniHack-River-Narrow-v0`	Planning
`MiniHack-River-v0`	Planning
`MiniHack-River-Monster-v0`	Planning
`MiniHack-River-Lava-v0`	Planning
`MiniHack-River-MonsterLava-v0`	Planning

HideNSeek

In these tasks, the agent is spawned in a big room full of trees and clouds. The trees and clouds block the line of sight of the player and a random monster (chosen to be more powerful than the agent). The agent, monsters and spells can pass through clouds unobstructed. The agent and monster cannot pass through trees. The goals is to make use of the environment features, avoid being seen by the monster and quickly run towards the goal. The layout of the map is procedurally generated, hence requires systematic generalisation.

`MiniHack-HideNSeek-v0` is the standard version of the environment. Alternative versions of this environment additionally include lava tiles that need to be avoided `MiniHack-HideNSeek-Lava-v0`, have bigger size `MiniHack-HideNSeek-Big-v0`, or provide the locations of all environment features but not the powerful monster `MiniHack-HideNSeek-Mapped-v0`.

Examples of the `MiniHack-HideNSeek-v0` task:



Reward

The agent receives a reward of +1 for reaching the goal.

Source

Source

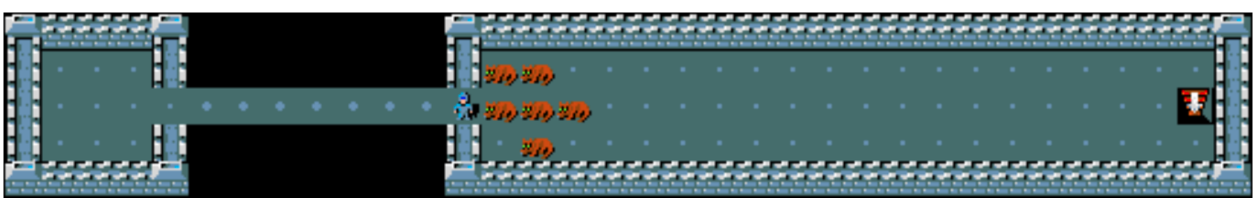
All Environments

Name	Capability
`MiniHack-HideNSeek-v0`	Planning
`MiniHack-HideNSeek-Mapped-v0`	Planning
`MiniHack-HideNSeek-Lava-v0`	Planning
`MiniHack-HideNSeek-Big-v0`	Planning

CorridorBattle

The `MiniHack-CorridorBattle-v0` task challenges the agent to make best use of the dungeon features to effectively defeat a horde of hostile monsters. Here, if the agent lures the rats into the narrow corridor, it can defeat them one at a time. Fighting in rooms, on the other hand, would result the agent simultaneously incurring damage from several directions and a quick death. The task also is offered in dark mode (`MiniHack-CorridorBattle-Dark-v0`), challenging the agent to remember the number of rats killed in order to plan subsequent actions.

An example of the `MiniHack-CorridorBattle-v0` task:



Reward

The agent receives a reward of +1 for reaching the goal.

Source

Source

All Environments

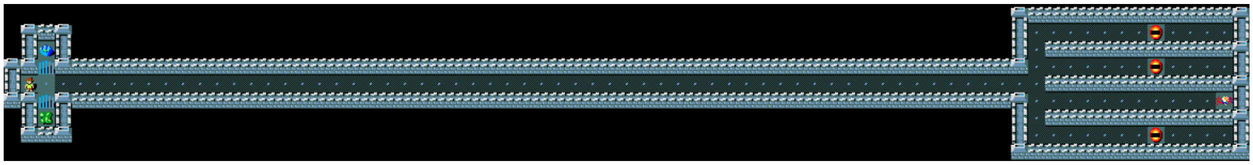
Name	Capability
`MiniHack-CorridorBattle-v0`	Planning & Memory
`MiniHack-CorridorBattle-Dark-v0`	Planning & Memory

Memento

This group of tasks test the agent's ability to use memory (within an episode) to pick the correct path. The agent is presented with a prompt (in the form of a sleeping monster of a specific type), and then navigates along a corridor. At the end of the corridor the agent reaches a fork, and must choose a direction. One direction leads to a grid bug, which if killed terminates the episode with +1 reward. All other directions lead to failure through a invisible trap that terminates the episode when activated. The correct path is determined by the cue seen at the beginning of the episode.

We provide three versions of this environment: one with a short corridor before a fork with two paths to choose from (`MiniHack-Memento-Short-F2-v0`), one with a long corridor with a two-path fork (`MiniHack-Memento-F2-v0`), and one with a long corridor and a four-fork path (`MiniHack-Memento-F4-v0`).

An example of the `MiniHack-Memento-F4-v0` task:



Reward

The agent receives a reward of 1 for killing the grid bug (navigating along the correct corridor) and -1 for stepping on the trap (naving along the incorrect corridor). Both events terminate the episode.

Source

Source

All Environments

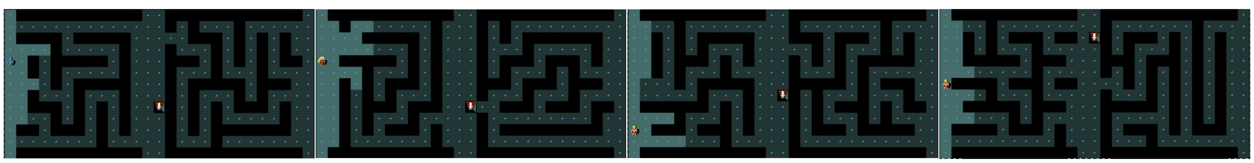
Name	Capability
MiniHack-Memento-Short-F2-v0	Memory
MiniHack-Memento-F2-v0	Memory
MiniHack-Memento-F4-v0	Memory

MazeExplore

These tasks test the agent's ability to perform deep exploration. It's inspired by the Apple-Gold domain from , where a small reward can be achieved easily, but to learn the optimal policy deeper exploration is required. The agent must first explore a simple randomised maze to reach the staircase down, which they can take for +1 reward. However, if they navigate through a further randomised maze, they reach a room with apples. Eating the apples gives +0.5 reward, and once the apples are eaten the agent should then return to the staircase down.

We provide an easy and a hard version of this task (`MazeExplore-Easy-v0` and `MiniHack-MazeExplore-Hard-v0`), with the harder version having a larger maze both before and after the staircase down. Variants can also be mapped (`MiniHack-MazeExplore-Easy-Mapped-v0` and `MiniHack-MazeExplore-Hard-Mapped-v0`), where the agent can observe the layout of the entire grid, making it easier to navigate the maze. Even in the mapped setting the apples aren't visible until the agent reaches the final room.

Examples of the `MiniHack-MazeExplore-Hard-v0` task. The apples are located near the right vertical wall (unobservable in the figure). The goal is located in the middle area of the grid.



Reward

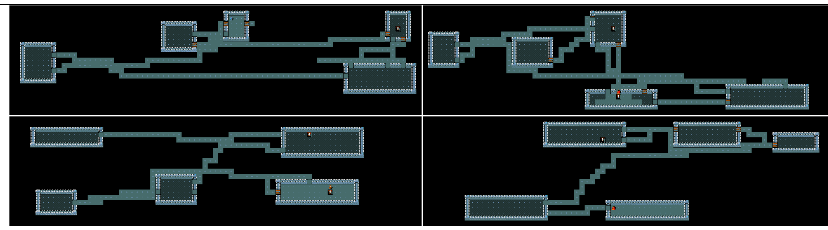
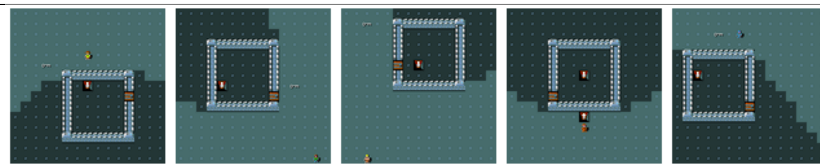
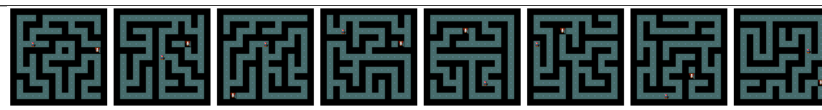
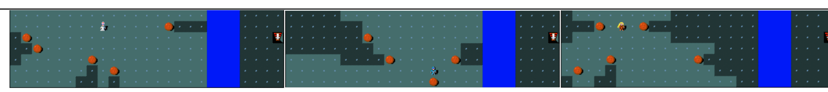
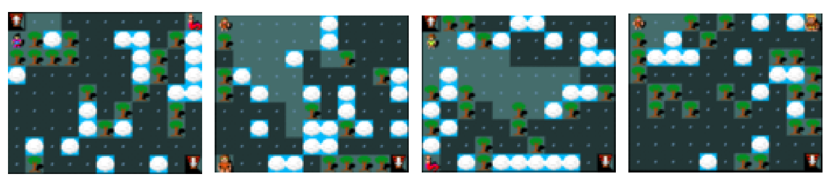
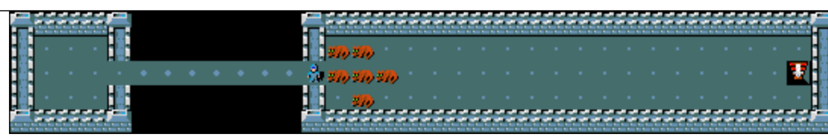
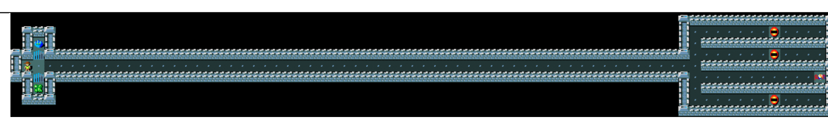
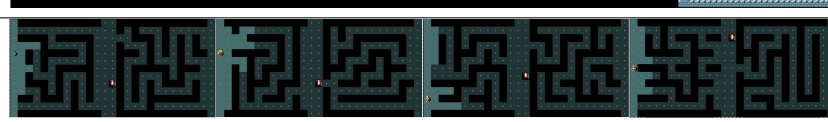
The agent receives a reward of +1 for reaching the goal and +0.5 for eating an apple.

Source

Source

All Environments

Name	Capability
MiniHack-MazeExplore-Easy-v0	Deep Exploration
MiniHack-MazeExplore-Hard-v0	Deep Exploration
MiniHack-MazeExplore-Easy-Mapped-v0	Deep Exploration
MiniHack-MazeExplore-Hard-Mapped-v0	Deep Exploration

Environment Family	Capability	Screenshots
Room	Basic Learning	
Corridor	Exploration	
KeyRoom	Exploration	
MazeWalk	Exploration & Memory	
River	Planning	
HideNSeek	Planning	
CorridorBattle	Planning & Memory	
Memento	Memory	
MazeExplore	Deep Exploration	

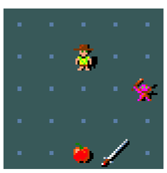
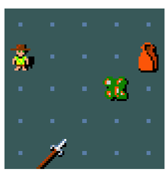
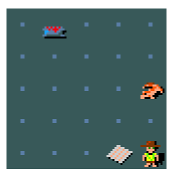
9.9.2 Skill Acquisition Tasks

MiniHack's skill acquisition tasks enable utilising the rich diversity of NetHack objects, monsters and dungeon features, and the interactions between them. The skill acquisition tasks feature a large action space (75 actions), where the actions are instantiated differently depending on which object they are acting on. Note that certain actions in skill acquisition tasks are factorised autoregresively, i.e., require performing a sequence of follow-up actions for the initial action to have an effect. For example, to put on a ring, the agent needs to select the `PUTON` action, choose the ring from the inventory and select which hand to put it on.

Simple Tasks

The simple skill acquisition tasks require discovering interaction between one object and the actions of the agent. These include: eating comestibles (`MiniHack-Eat-v0`), praying on an altar (`MiniHack-Pray-v0`), wearing armour (`MiniHack-Wear-v0`), and kicking locked doors (`LockedDoors`). In the regular versions of these tasks, the starting location of the objects and the agent is randomised, whereas in the fixed versions of these tasks (`MiniHack-Eat-Fixed-v0`, `MiniHack-Pray-Fixed-v0`, `MiniHack-Wear-Fixed-v0` and `MiniHack-LockedDoors-Fixed-v0`) both are fixed. To add a slight complexity to the randomised version of these tasks, distractions in the form of a random object and a random monster are added to the third version of these tasks (`MiniHack-Eat-Distract-v0`, `MiniHack-Pray-Distract-v0` and `MiniHack-Wear-Distract-v0`). These tasks can be used as building blocks for more advanced skill acquisition tasks.

Examples of `MiniHack-Eat-Distract-v0`, `MiniHack-Wear-Distract-v0` and `MiniHack-Pray-Distract-v0` tasks:

Reward

The agent receives a reward of +1 for using the required skill in the given environment, such as eating comestibles in `MiniHack-Eat-v0` or wearing armour in `MiniHack-Wear-v0`.

Source

Source

All Environments

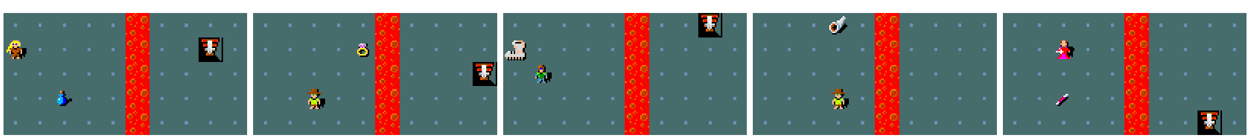
Name	Skill
MiniHack-Eat-v0	Confirmation or PickUp + Inventory
MiniHack-Eat-Fixed-v0	Confirmation or PickUp + Inventory
MiniHack-Eat-Distract-v0	Confirmation or PickUp + Inventory
MiniHack-Pray-v0	Confirmation
MiniHack-Pray-Fixed-v0	Confirmation
MiniHack-Pray-Distract-v0	Confirmation
MiniHack-Wear-v0	PickUp + Inventory
MiniHack-Wear-Fixed-v0	PickUp + Inventory
MiniHack-Wear-Distract-v0	PickUp + Inventory
MiniHack-LockedDoor-v0	Direction
MiniHack-LockedDoor-Random-v0	Direction

Lava Crossing

This family of skill acquisition tasks requires crossing a river of lava. The agent can accomplish this by either levitating over it (via a potion of levitation or levitation boots) or freezing it (by zapping the wand of cold or playing the frost horn).

In the simplest version of the task (MiniHack-LavaCross-Levitate-Potion-Inv-v0 and MiniHack-LavaCross-Levitate-Ring-Inv-v0), the agent starts with one of the necessary objects in the inventory. Requiring the agent to pickup the corresponding object first makes the tasks more challenging (MiniHack-LavaCross-Levitate-Potion-PickUp-v0 and MiniHack-LavaCross-Levitate-Ring-PickUp-v0). Most difficult variants of this task group require the agent to cross the lava river using one of the appropriate objects randomly sampled and placed at the random location. In LMiniHack-avaCross-Levitate-v0, one of the objects of levitation is placed on the map, while in the MiniHack-LavaCross-v0 task these include all of the objects for levitation as well as freezing.

Five random instances of the MiniHack-LavaCross-v0 task, where the agent needs to cross the lava using (i) potion of levitation, (ii) ring of levitation, (iii) levitation boots, (iv) frost horn, or (v) wand of cold.



Reward

The agent receives a reward of +1 for reaching the goal on the other side of the lava river.

Source

Source

All Environments

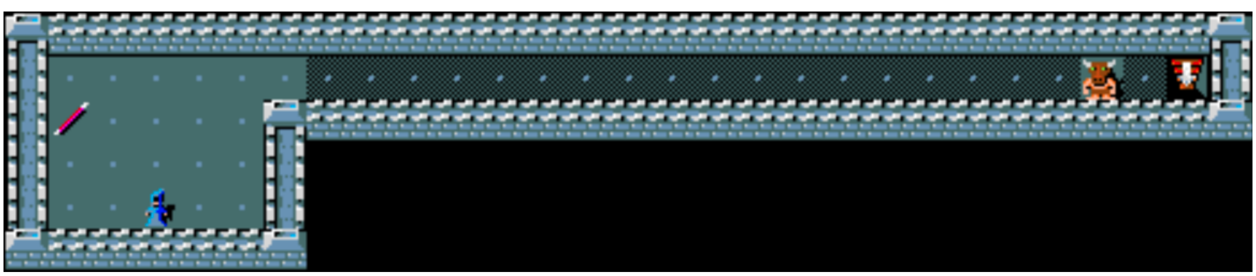
Name	Skill
MiniHack-LavaCross-Levitate-Ring-Inv-v0	Inventory
MiniHack-LavaCross-Levitate-Potion-Inv-v0	Inventory
MiniHack-LavaCross-Levitate-Ring-Pickup-v0	PickUp + Inventory
MiniHack-LavaCross-Levitate-Potion-PickUp-v0	PickUp + Inventory
MiniHack-LavaCross-Levitate-v0	PickUp + Inventory
MiniHack-LavaCross-v0	PickUp + Inventory

Wand of Death

These environments require mastering the usage of the wand of death (WoD). Zapping a WoD it in any direction fires a death ray which instantly kills almost any monster it hits.

In `MiniHack-WoD-Easy-v0` environment, the agent starts with a WoD in its inventory and needs to zap it towards a sleeping monster. `MiniHack-WoD-Medium-v0` requires the agent pick it up, approach the sleeping monster, kill it, and go to the staircase. In `MiniHack-WoD-Hard-v0` the WoD needs to be found first, only then the agent should enter the corridor with a monster (who is awake and hostile this time), kill it, and go to the staircase. In the most difficult task of the sequence, the `MiniHack-WoD-Pro-v0`, the agent starts inside a big labyrinth. It needs to find the WoD inside the maze and reach its centre, which is guarded by a deadly Minotaur.

An example of the `MiniHack-WoD-Hard-v0` task:



Reward

The agent receives a reward of +1 for killing the minotaur `MiniHack-WoD-Easy-v0`. For the other versions, a reward of +1 is received upon reaching the goal.

Source

Source

All Environments

Name	Skill
`MiniHack-WoD-Easy-v0`	Inventory + Direction
`MiniHack-WoD-Medium-v0`	PickUp + Inventory + Direction
`MiniHack-WoD-Hard-v0`	PickUp + Inventory + Direction
`MiniHack-WoD-Pro-v0`	Navigation + PickUp + Inventory + Direction

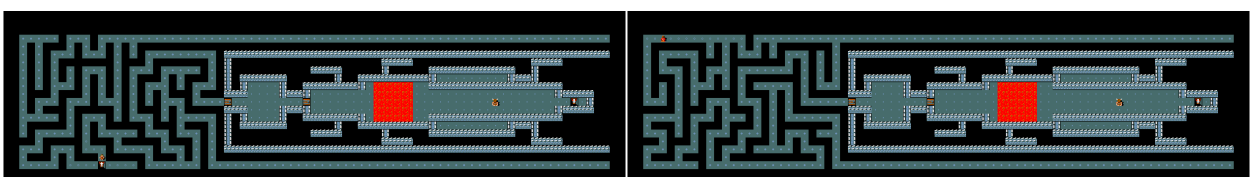
Quest

This family of environments features a mini-quest for the agent to complete.

The agents needs to make use of an object is laying around for crossing a lava rivver (this can be any object allowing levitation or freezing), while fighting monsters and navigating rooms or mazes. Towards the end of the quests, the agent needs to utlise a wand of death to kill a deadly monster guarding the goal location.

In `MiniHack-Quest-Easy-v0`, the map layout is relatively simple and fixed. The `MiniHack-Quest-Medium-v0` task features a narrow corridor. The agents needs to lure monsters into a narrow corridor and defeat them one at a time, before progressing further. In the most challenging version of the task, `MiniHack-Quest-Hard-v0`, features large procedurally generated maze which needs to solved first before embarking on the next steps of the quest.

Examples of the `MiniHack-Quest-Hard-v0` task:



Reward

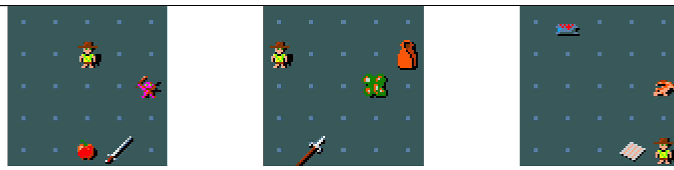
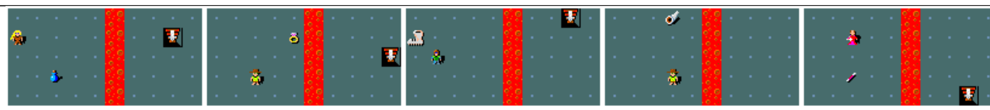
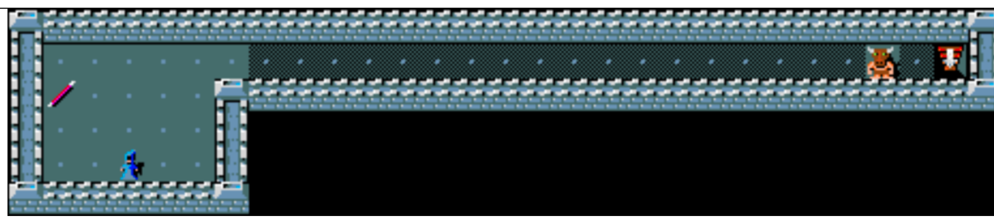
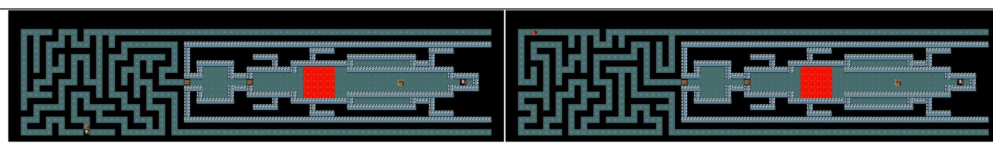
The agent receives a reward of +1 for reaching the goal.

Source

Source

All Environments

Name	Skill
`MiniHack-Quest-Easy-v0`	Inventory
`MiniHack-Quest-Medium-v0`	Navigation + Inventory
`MiniHack-Quest-Hard-v0`	Navigation + PickUp + Inventory + Direction

Environment Family	Screenshots
Simple Skills	
Lava Crossing	
Wand Of Death	
Quest	

9.9.3 Ported tasks

These tasks are ported to MiniHack from other existing benchmarks. Note that there might be substantial differences in MiniHack versions of these tasks. Despite these, the core capabilities of the tasks should be the same. Furthermore, some of the tasks have been extended to become more challenging using the entities and environment dynamics of MiniHack.

MiniGrid

This family of environments is ported to MiniHack from MiniGrid, a popular suite of procedurally generated grid-based environments that assess various capabilities of RL agents, such as exploration, memory, and generalisation. For more information, check out MiniGrid's documentation. After porting environments to MiniHack, one can make them substantially harder by adding additional environment dynamics to the task, such as monsters, dungeon features and objects.

The MultiRoom environments have a series of connected rooms. The final room has the goal location the agent needs to get to. We have ported the `MultiRoom` in three different room numbers, namely 2, 4 and 6 rooms. Moreover, we added additional complexity to them by adding monsters (e.g. `MiniHack-MultiRoom-N4-Monster-v0`), locked doors (e.g. `MiniHack-MultiRoom-N4-Locked-v0`), lava tiles instead of walls (e.g. `MiniHack-MultiRoom-N4-Lava-v0`), or all at one (e.g. `MiniHack-MultiRoom-N4-Extreme-v0`).

The LavaCrossing and SimpleCrossing environments require the agent to reach the goal on the other corner in the same room. In the case of LavaCrossing, there are several lava stream running across the room either horizontally or vertically, and only have a single crossing point which can be safely used. The agent needs to avoid the lava which would otherwise it will die immediately. In SimpleCrossing, on the other hand the lava is replaced by the walls. Several versions of the environments are ported from MiniGrid and provided in the table below.

Note: More tasks could have similarly been ported from MiniGrid. However, our goal was to showcase MiniHack's ability to port existing grid-world environments and easily enrich them, rather then porting all possible tasks.

The following figure presents examples of `MiniHack-MultiRoom-N4-v0` and `MiniHack-MultiRoom-N4-Extreme-v0` environments rendered using MiniHack's tiles:

envs/ported/imgs/multiroom.png

Reward

The agent receives a reward of +1 for reaching the goal.

Source

Source

All Environments

Name	Capability
MiniHack-MultiRoom-N2-v0	Exploration
MiniHack-MultiRoom-N4-v0	Exploration
MiniHack-MultiRoom-N6-v0	Exploration
MiniHack-MultiRoom-N2-Monster-v0	Exploration
MiniHack-MultiRoom-N4-Monster-v0	Exploration
MiniHack-MultiRoom-N6-Monster-v0	Exploration
MiniHack-MultiRoom-N2-Locked-v0	Exploration
MiniHack-MultiRoom-N4-Locked-v0	Exploration
MiniHack-MultiRoom-N6-Locked-v0	Exploration
MiniHack-MultiRoom-N2-Lava-v0	Exploration
MiniHack-MultiRoom-N4-Lava-v0	Exploration
MiniHack-MultiRoom-N6-Lava-v0	Exploration
MiniHack-MultiRoom-N2-Extreme-v0	Exploration
MiniHack-MultiRoom-N4-Extreme-v0	Exploration
MiniHack-MultiRoom-N6-Extreme-v0	Exploration
MiniHack-LavaCrossingS9N1-v0	Exploration
MiniHack-LavaCrossingS9N2-v0	Exploration
MiniHack-LavaCrossingS9N3-v0	Exploration
MiniHack-LavaCrossingS11N5-v0	Exploration
MiniHack-SimpleCrossingS9N1-v0	Exploration
MiniHack-SimpleCrossingS9N2-v0	Exploration
MiniHack-SimpleCrossingS9N3-v0	Exploration
MiniHack-SimpleCrossingS11N5-v0	Exploration

Boxoban

This family of environments is ported to MiniHack from Boxoban, a box-pushing puzzle game inspired by Sokoban. The goal is to push four boxes (or boulder's in MiniHack's version) to four goal locations (fountains).

The procedurally generated levels are divided into three difficulties - Ulfiltered (`MiniHack-Boxoban-Unfiltered-v0`), Medium (`MiniHack-Boxoban-Medium-v0`), and Hard (`MiniHack-Boxoban-Hard-v0`).

An example of Boxoban level ported into MiniHack.

envs/ported/imgs/boxoban.png

Reward

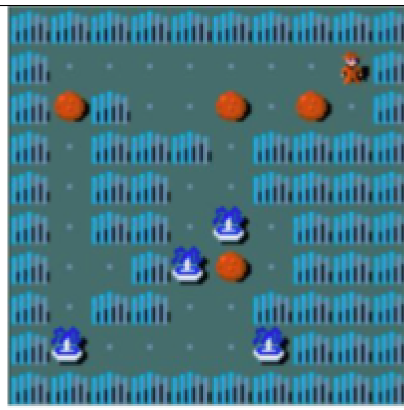
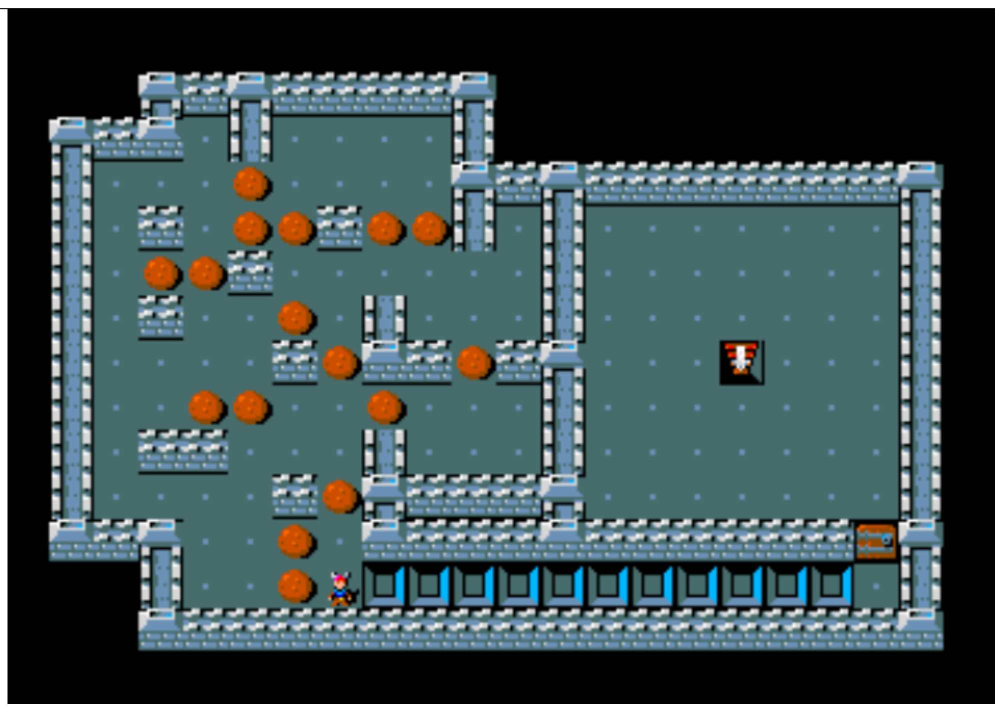
The agent receives a reward of +1 for pushing all boulders to different goal positions. The agent additional receives a shaped reward of +0.1 for each pushing each boulder to a goal position and -0.1 penalty for moving it out of there.

Source

Source

All Environments

Name	Capability
`MiniHack-Boxoban-Unfiltered-v0`	Planning
`MiniHack-Boxoban-Medium-v0`	Planning
`MiniHack-Boxoban-Hard-v0`	Planning

Environment Family	Screenshots
MiniGrid	 
Boxoban	
NetHack Sokoban	

9.10 Submitting New Environments

For submitting a new environment to MiniHack Environment Zoo, open a Pull Request on GitHub that includes the following:

- The .py file implementing the environment should be put into `minihack/envs` directory (with appropriate registration).
- If the environment includes a .des file, please put it into the `minihack/dat` directory.
- The description of the environment should reside in the `docs/envs` directory
  - Create a separate markdown file in the corresponding directory (`navigation`, `skills` or `ported`) describing the environment (and its possible variations), its objective, capabilities it assesses, reward, action space used, as well as the link to the source code.
  - Include a screenshot of the environment in `docs/envs/imgs` directory.
  - Update the tables (both Markdown table and {toctree} block) in `docs/envs/index.md` to reference your new environment (or family of environments).

We look forward to accepting diverse environment contributions from the community.

9.11 *des-file* format: A tutorial

Note: This tutorial assumes a small amount of knowledge of NetHack. Skimming through the community tutorial on des-file format would also be helpful.

Note: In case you are viewing the static version of this tutorial, the interactive version can be found here.

9.11.1 What is a des-file?

A *des-file* is a file which describes a level in the game of NetHack. In MiniHack, we use des-files to easily define new environments. The des-files are specified in a human-readable domain-specific language (DSL), which has some powerful probabilistic features which enable easy specification of interesting distributions of levels in only a few lines. While there are many features of these files and the DSL used to specify them, in this tutorial we will highlight the most relevant ones, which are most useful for designing environments for RL research.

9.11.2 The Two Types of des-files: MAZE and ROOM

There are two types of des-file, which have different properties in terms of how they define the layout of the level: * The ROOM-type des-file works by specifying a collection of ROOMs, possibly with SUBROOMs. Any of these rooms and subrooms can have their contents specified, as well as their approximate location and exact size. These rooms are then randomly places (loosely obeying the approximate location) on the map. Generally, the RANDOM_CORRIDORS command is then used to create random corridors enabling access to all the rooms on the map. * The MAZE-type des-file offers more control over defining the layout of the level. The layout can be "drawn" with ascii characters, and then contents can be specified to occupy certain locations on the level. The layout specified could only occupy part of the map, with the other parts being randomly generated.

[1]:
```
# Importing helper visualisation functions
from minihack.tiles.rendering import get_des_file_rendering

import IPython.display
```

(continues on next page)

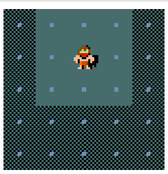
(continued from previous page)

```
def render_des_file(des_file, **kwargs):
    image = get_des_file_rendering(des_file, **kwargs)
    IPython.display.display(image)
```

MAZE-type levels

Let's look at a simple MAZE-type des-file:

[2]:
```
des_file = """
MAZE: "mylevel", ' '
FLAGS:premapped
GEOMETRY:center,center
MAP
.....
.....
.....
.....
.....
ENDMAP
"""
render_des_file(des_file, n_images=1, full_screen=False)
```



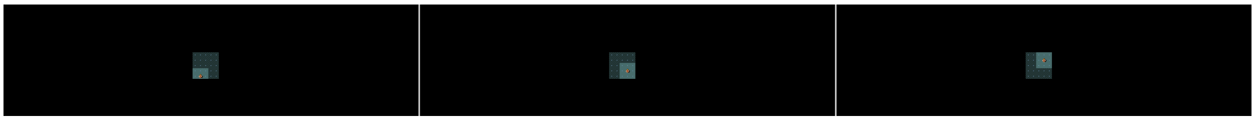
Here we can already see some basic structure of the des-file. The first line specifies the type (either MAZE: for maze-type or LEVEL: for room-type). The second line specifies a possible list of flags to apply to the level. Here we're using `premapped` so we can easily visualise the entire level. `GEOMETRY` determines where the following `MAP` definition will be on the whole screen, and `MAP` defines the map itself. `.` is a floor tile, and everything not specified becomes impassable rock.

Let's experiment with GEOMETRY to see what it does:

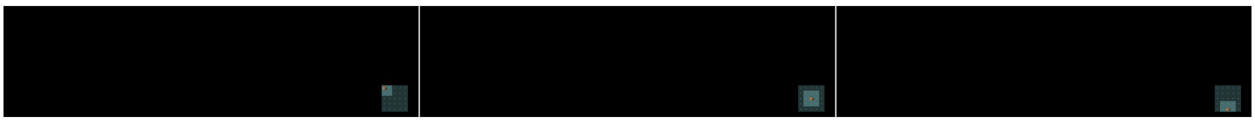
[3]:
```
des_file = """
MAZE: "mylevel", ' '
FLAGS:premapped
GEOMETRY:{},{}
MAP
.....
.....
.....
.....
.....
ENDMAP
"""
for geom_x, geom_y in [("left", "top"), ("center", "center"), ("right", "bottom")]:
    print("Levels with geometry: GEOMETRY:{},{}".format(geom_x, geom_y))
    render_des_file(des_file.format(geom_x, geom_y), n_images=3, full_screen=True)
```

```
Levels with geometry: GEOMETRY:left,top
```

```
Levels with geometry: GEOMETRY:center,center
```

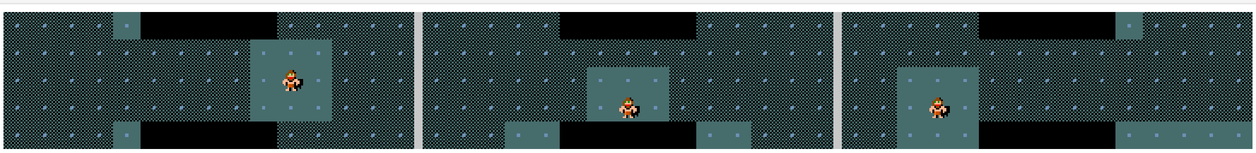


```
Levels with geometry: GEOMETRY:right,bottom
```



Lets try making the map a bit more interesting. Feel free to play around and make an interesting design. Some constraints: all lines must be the same length (you can use spaces to fill up the ends of lines if you want), and the whole maze must be within 79 x 21 size

[4]:
```
des_file = """
MAZE: "mylevel", ' '
FLAGS:premapped
GEOMETRY:center,center
MAP
.....     .....
...............
...............
...............
.....     .....
ENDMAP
"""
render_des_file(des_file, n_images=3)
```



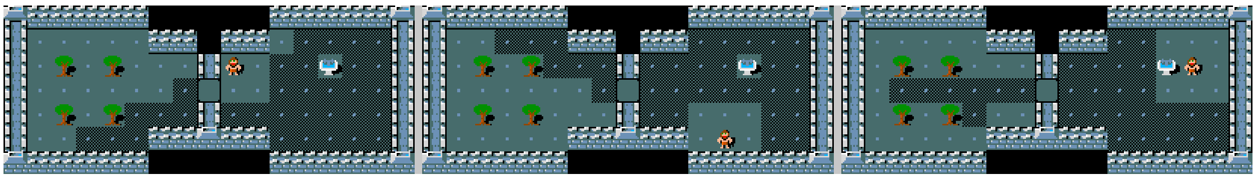
So far we've just changed the shape of the map, but what about making it out of something other than floor and walls? You can use any of the map characters defined in the des-file format; see here for the full list. For example, we could add a sink, some trees, a door, and some walls

[5]:
```
des_file = """
MAZE: "mylevel", ' '
FLAGS:premapped
GEOMETRY:center,center
MAP
|-----     ------
|.....-- --.....|
|.T.T...-....K..|
|.......+.......|
```

(continues on next page)

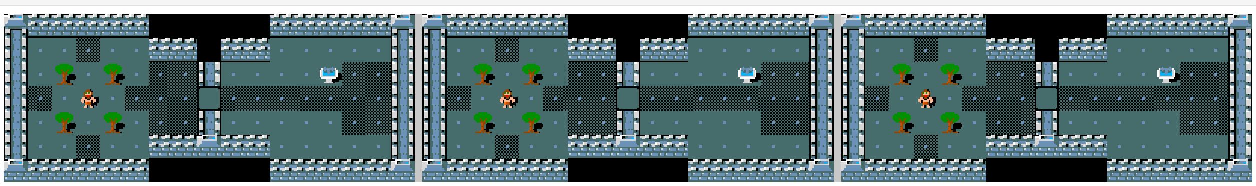
(continued from previous page)

```
|.T.T...-.......|
|.....-----.....|
|-----     ------
ENDMAP
"""
render_des_file(des_file, n_images=3, full_screen=False)
```



A good thing about MAZE levels is you can specify the starting point of the agent explicitly, using the BRANCH command. This command takes in two regions as input, and spawns the agent somewhere in the first region, as long as it's not in the second region. If you want the agent to spawn in a single spot, you can just pass the same coordinates twice. In the previous maps the agent has been spawning in a random location - lets try and spawn them in the middle of the four trees.

[6]:
```
des_file = """
MAZE: "mylevel", ' '
FLAGS:premapped
GEOMETRY:center,center
MAP
|-----     ------
|.....-- --.....|
|.T.T...-....K..|
|.......+.......|
|.T.T...-.......|
|.....-----.....|
|-----     ------
ENDMAP
BRANCH: (3,3,3,3),(4,4,4,4)
"""
render_des_file(des_file, n_images=3, full_screen=False)
```



ROOM-type levels

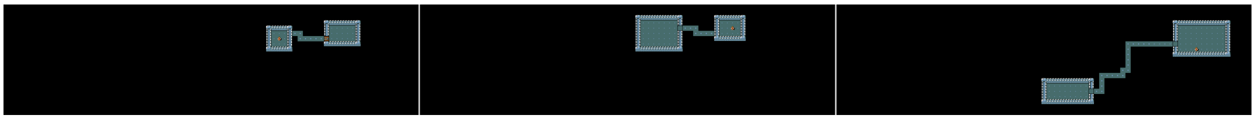
Let's look at a simple ROOM-type des-file:

[7]:
```
des_file = """
LEVEL: "mylevel"
FLAGS: premapped

ROOM: "ordinary" , lit, random, random, random {
}
```

(continues on next page)

(continued from previous page)

```
ROOM: "ordinary" , lit, random, random, random {
}

RANDOM_CORRIDORS
"""
render_des_file(des_file, n_images=3, full_screen=True)
```



This has a similar structure in the beginning of the file to the MAZE-type levels, using flags to specify some options. After that, several ROOMs are defined. The arguments to the ROOM command are:

- type, which specifies the type of room (generally we'll just want ordinary)
- lit or unlit - this controls how far the agent can see in the room
- position, a tuple with values between 1 and 5 which roughly specifies the room location
- align, a tuple of xalign and yalign. This specifies the alignment of the room within the position defined above
  - xalign is one of left, half-left, center, half-right, right or random
  - yalign is one of top, center, bottom or random.
- size, a tuple of (height, width).

Any of these can be left as `random`, which means they're randomly chosen.

Let's experiments with each of these arguments in turn, to get an idea of what they do to just one room:

[8]:
```
des_file = """
LEVEL: "mylevel"
FLAGS: premapped
ROOM: "ordinary" , lit, random, random, random {
}
RANDOM_CORRIDORS
"""
render_des_file(des_file, n_images=3, full_screen=True)
```

[9]:
```
des_file = """
LEVEL: "mylevel"
FLAGS: premapped

ROOM: "ordinary" , lit, (XX,YY), random, random {
}

RANDOM_CORRIDORS
"""
for x,y in [(1,1), (1,5), (5,1), (5,5)]:
```

(continues on next page)

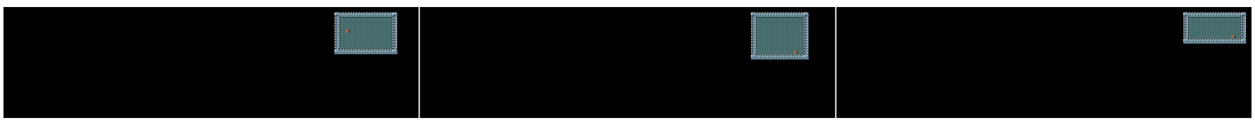
(continued from previous page)

```
    print("Room placed with position: ", x, ",", y)
    render_des_file(des_file.replace("XX", str(x)).replace("YY", str(y)), n_images=3,␣
↪full_screen=True)
```

```
Room placed with position:  1 , 1
```

```
Room placed with position:  1 , 5
```

```
Room placed with position:  5 , 1
```



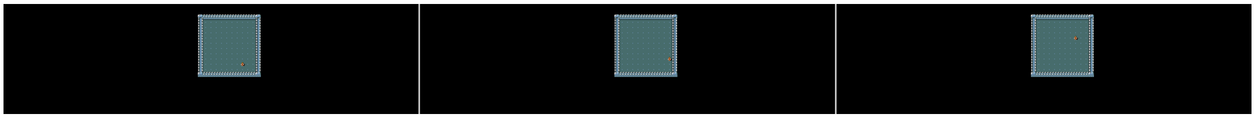
```
Room placed with position:  5 , 5
```

[10]:
```
des_file = """
LEVEL: "mylevel"
FLAGS: premapped

ROOM: "ordinary" , lit, (3,3), (xalign,yalign), (10,10) {
}

RANDOM_CORRIDORS
"""
for xalign,yalign in [("left", "top"), ("right", "top"), ("right", "bottom"), ("center",
↪"center")]:
    print("Room placed with align: ", xalign, ",", yalign)
    render_des_file(des_file.replace("xalign", xalign).replace("yalign", yalign), n_
↪images=3, full_screen=True)
```

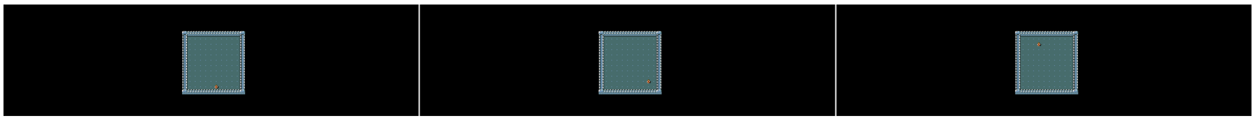
```
Room placed with align:  left , top
```

```
Room placed with align:  right , top
```

```
Room placed with align:  right , bottom
```

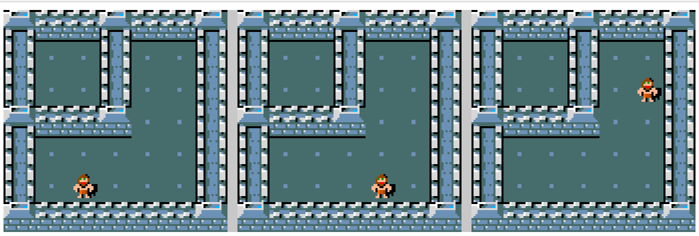


```
Room placed with align:  center , center
```



When defining a ROOM-type level, we can also place subrooms within rooms. These subrooms are initialised in a similar way to ROOMs

[11]:
```
des_file = """
LEVEL: "mylevel"
FLAGS: premapped

ROOM: "ordinary" , lit, (3,3), (center,center), (5,5) {
    SUBROOM: "ordinary", lit, (0,0), (2,2) {
    }
}

RANDOM_CORRIDORS
"""
render_des_file(des_file, n_images=3, full_screen=False)
```



Finally, we should probably explain the RANDOM_CORRIDORS command. This command creates random corridors between all the rooms. Look what happens if we don't include it (we won't be able to get between the rooms).

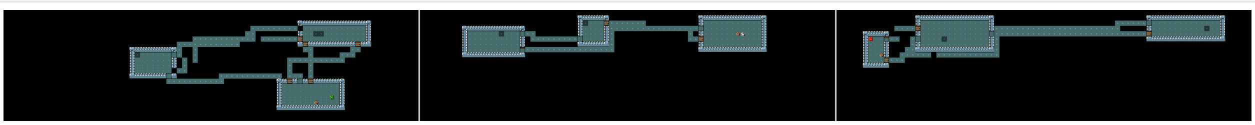
[12]:
```
des_file = """
LEVEL: "mylevel"
FLAGS: premapped

ROOM: "ordinary" , lit, random, random, random {
}
ROOM: "ordinary" , lit, random, random, random {
}
"""
render_des_file(des_file, n_images=3, full_screen=True)
```

9.11.3 Adding complexity: Monsters, Objects & Traps

While making nice mazes or rooms is all well and good, we want our agents to tackle more interesting challenges. This is where we start to draw on the richness of the game of NetHack; specifically, we'll see how to add monsters, objects and traps to the levels we've defined. These commands work the same for both ROOM and MAZE type levels, with one nuance when defining the location of these entities: in MAZE-type levels, the coordinates given are relative to the most recent MAP definintion; for ROOM-type levels, they're relative to the ROOM containing the entity being added.

Let's see how to add a monster to our 2-room environment from earlier (we've added a lit region covering the whole map, so that we can see what's happening everywhere):

[13]:
```
des_file = """
LEVEL: "mylevel"
FLAGS: premapped
REGION: (0,0,20,80), lit, "ordinary"

ROOM: "ordinary" , lit, random, random, random {
    MONSTER: random, random
}
ROOM: "ordinary" , lit, random, random, random {
    MONSTER: ('F', "lichen"), random
}
ROOM: "ordinary" , lit, random, random, random {
    MONSTER: ('F', "red mold"), (0,0)
}

RANDOM_CORRIDORS
"""
render_des_file(des_file, n_images=3, full_screen=True)
```



We can specify the monster type with a tuple of class and name (e.g. `('F', "lichen")`), or leave it random. Similarly with the location, we can leave it random or specify it, as we did in the last room.

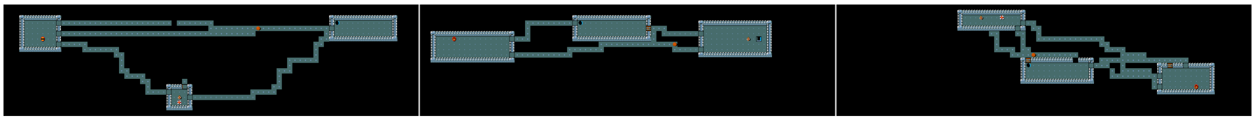
Adding traps is very similar to adding monsters, apart from we just need the trap name rather than the tuple of name and class:

[14]:
```
des_file = """
LEVEL: "mylevel"
FLAGS: premapped
REGION: (0,0,20,80), lit, "ordinary"

ROOM: "ordinary" , lit, random, random, random {
    TRAP: random, random
}
ROOM: "ordinary" , lit, random, random, random {
    TRAP:"fire", random
}
ROOM: "ordinary" , lit, random, random, random {
    TRAP:"hole",(0,0)
}
```

(continues on next page)

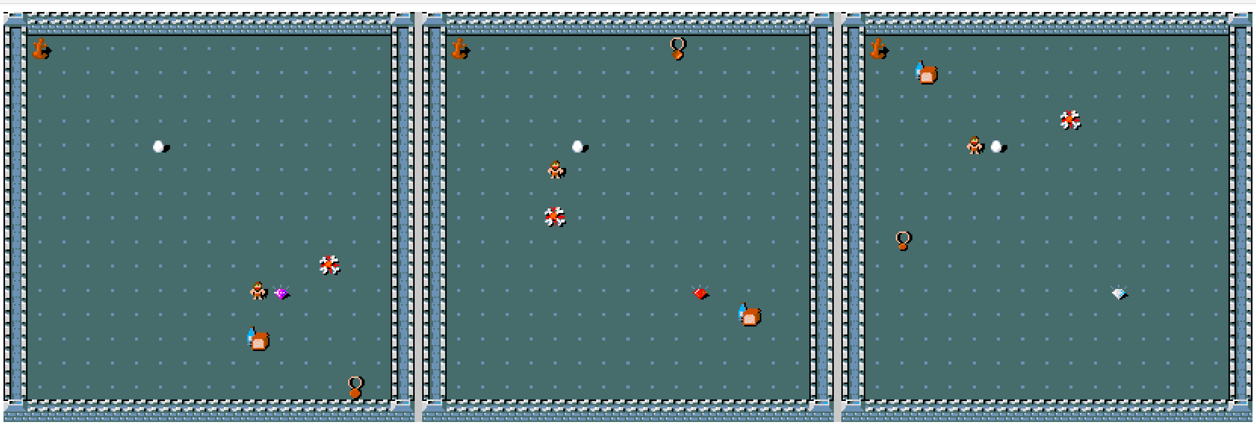
(continued from previous page)

```
RANDOM_CORRIDORS
"""
render_des_file(des_file, n_images=3, full_screen=True)
```



As well as monsters and traps, there are many objects in the game of NetHack that we can add to our level. We can specify the type, the location, and for certain objects extra arguments can be passed specific to that object (e.g. the status or the monster egg).

[15]:
```
des_file = """
LEVEL: "mylevel"
FLAGS: premapped
REGION: (0,0,20,80), lit, "ordinary"

ROOM: "ordinary", lit, random, (center,center), (15,15) {
    OBJECT:('%', "food ration"), random
    OBJECT:'*', (10,10)
    OBJECT :('"', "amulet of life saving"), random
    OBJECT:('%', "corpse"), random
    OBJECT:('`', "statue"), (0,0), montype:"forest centaur", 1
    OBJECT:('(', "crystal ball"), (17,08), blessed, 5,name:"The Orb of Fate"
    OBJECT:('%',"egg"), (05,04), montype:"yellow dragon"
}

RANDOM_CORRIDORS
"""
render_des_file(des_file, n_images=3)
```



As well as objects placed with `OBJECT`, there are several terrain features it's worth knowing about: `SINK`, `FOUNTAIN`, `ALTAR`, and `STAIR`. Stairs down are normally the objective for the level, as they lead deeper into the dungeon (unless you specify your own).

[16]:
```
des_file = """
LEVEL: "mylevel"
FLAGS: premapped
REGION: (0,0,20,80), lit, "ordinary"
```

(continues on next page)

(continued from previous page)

```
ROOM: "ordinary" , lit, random, random, random {
    SINK: random
    FOUNTAIN: random
    ALTAR: random, random, random
    STAIR: random, down
}

RANDOM_CORRIDORS
"""
render_des_file(des_file, n_images=3, full_screen=True)
```

Landscaping Terrain: selections and coordinates

So far we've seen how to create rooms and mazes, and add objects, traps and monsters to those rooms. Now we come to another way to add complexity to your environments: terrain. This takes the form of static objects or structures in the environment (such as trees, clouds, water and lava), all of which have different properties. When placing these pieces of terrain, we control their location exactly or leave it entirely to chance (in a similar way as for monsters, objects and traps). There's also a third option, which enables us to specify complex selections (sets of coordinates) and randomly place terrain within these selections.

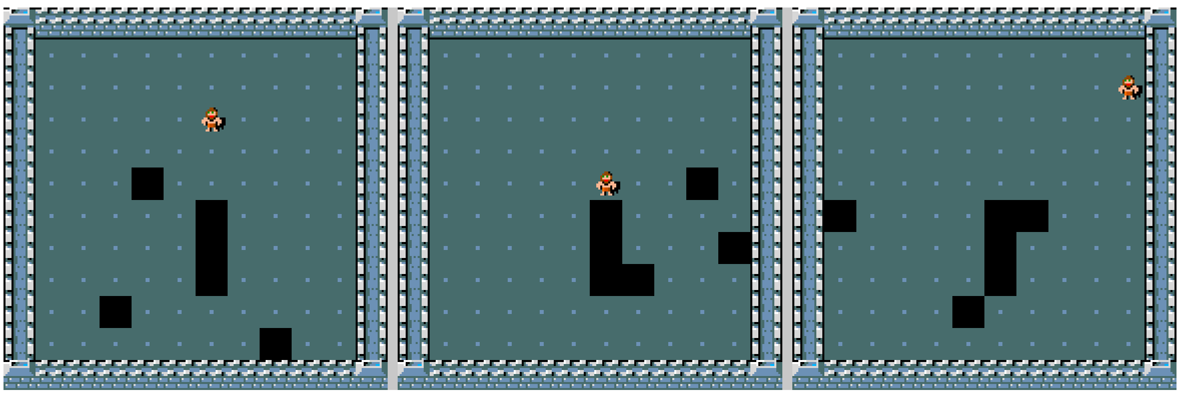
First, lets look at the basic types of terrain:

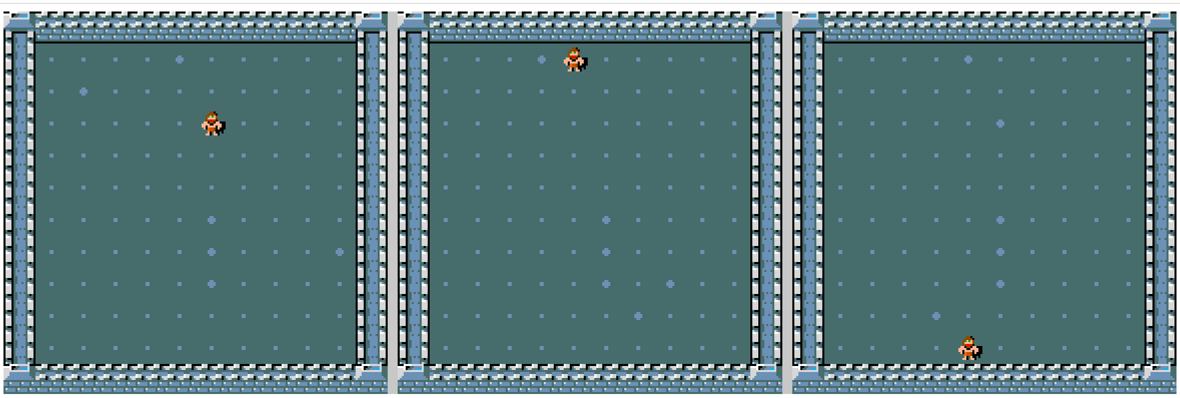
[17]:
```
des_file = """
LEVEL: "mylevel"
FLAGS: premapped
REGION: (0,0,20,80), lit, "ordinary"

ROOM: "ordinary" , lit, (3,3), random, (10,10) {
    TERRAIN: (5,5), '%terrain%'
    TERRAIN: (5,6), '%terrain%'
    TERRAIN: (5,7), '%terrain%'
    TERRAIN: random, '%terrain%'
    TERRAIN: random, '%terrain%'
    TERRAIN: random, '%terrain%'
}

RANDOM_CORRIDORS
"""

Terrains = [
    " ",  # solid wall
    "#",  # corridor
    ".",  # room floor (Unlit, unless lit with REGION-command)
    "-",  # horizontal wall
    "|",  # vertical wall
    "+",  # door (State is defined with DOOR -command)
```

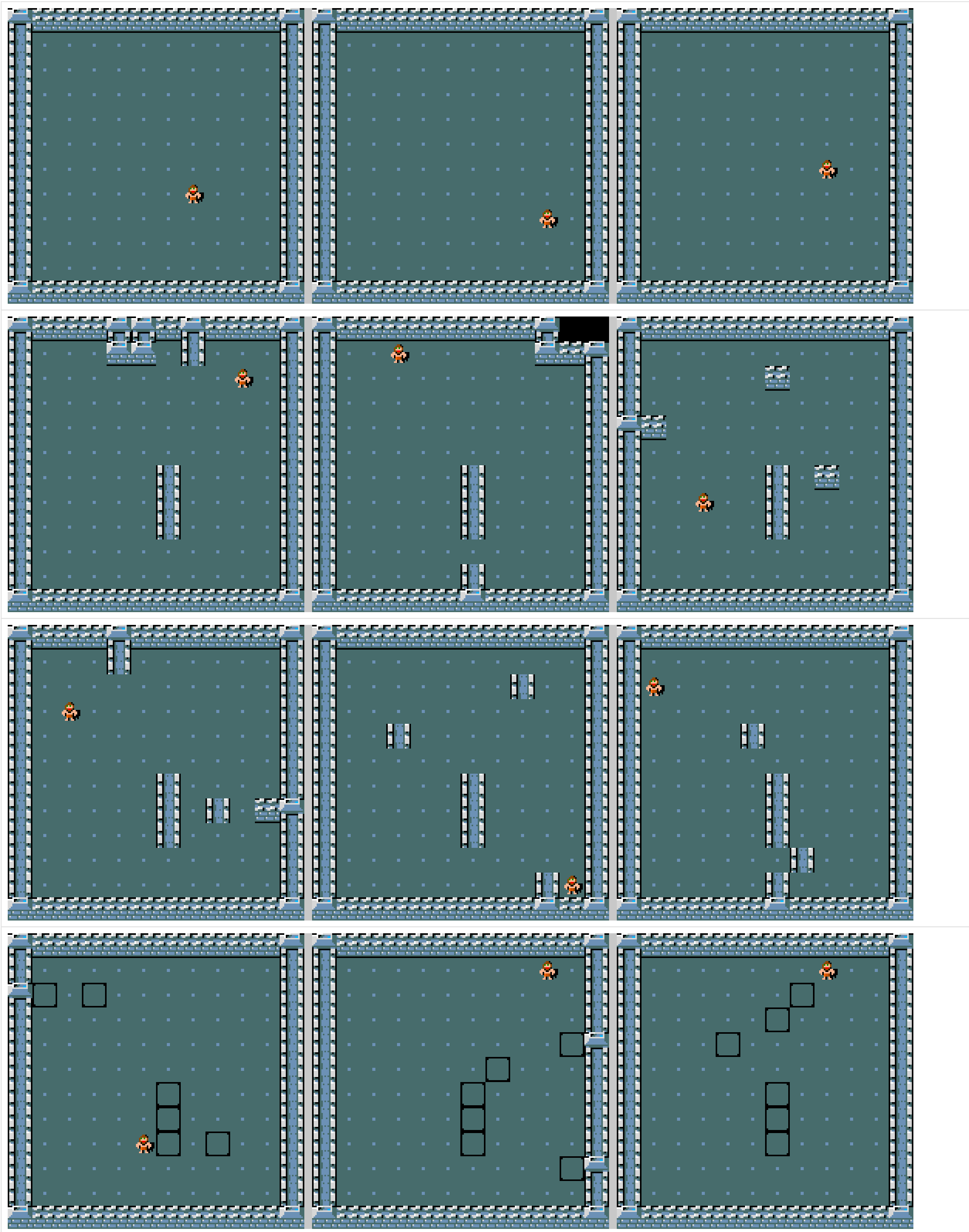
(continues on next page)

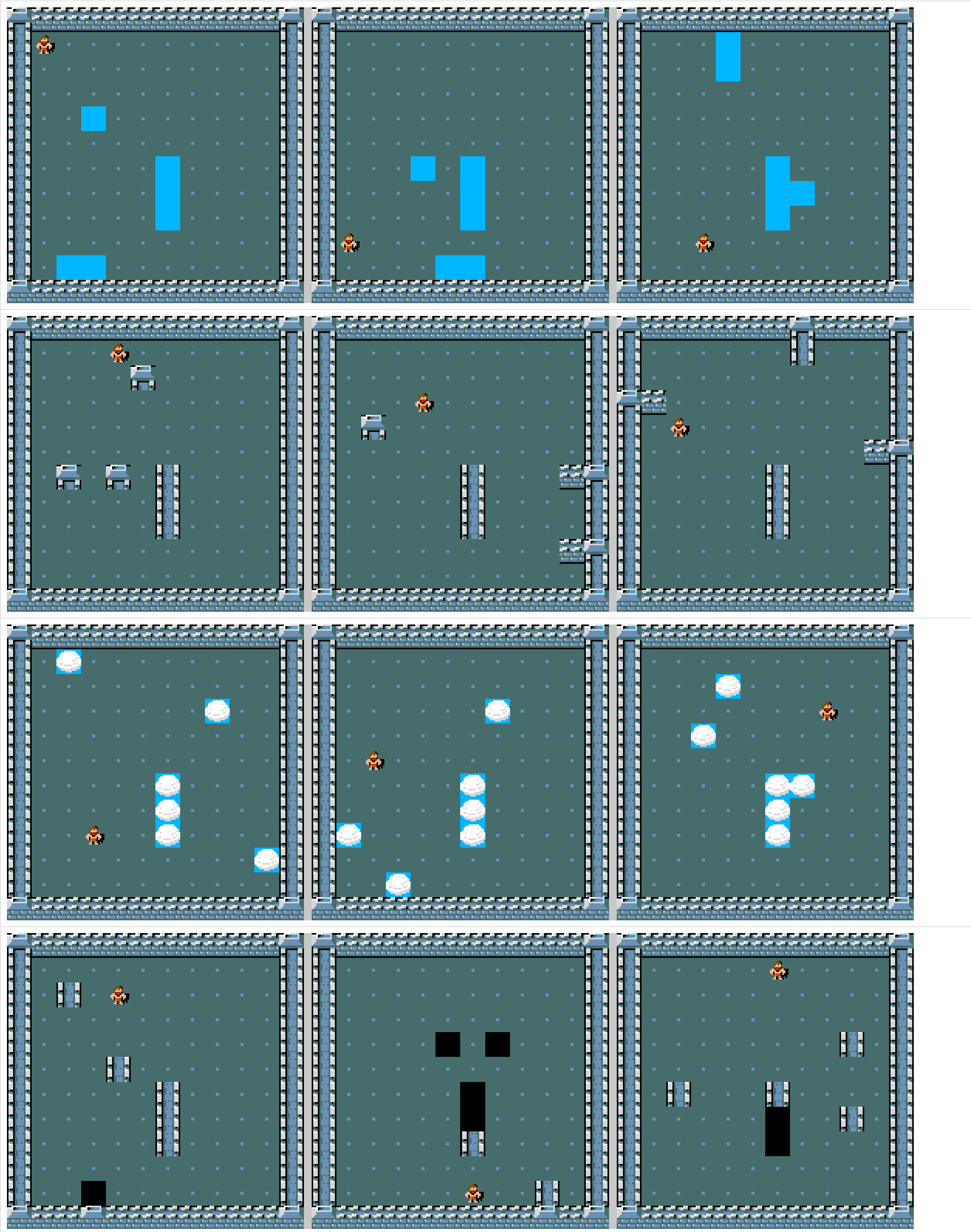
(continued from previous page)

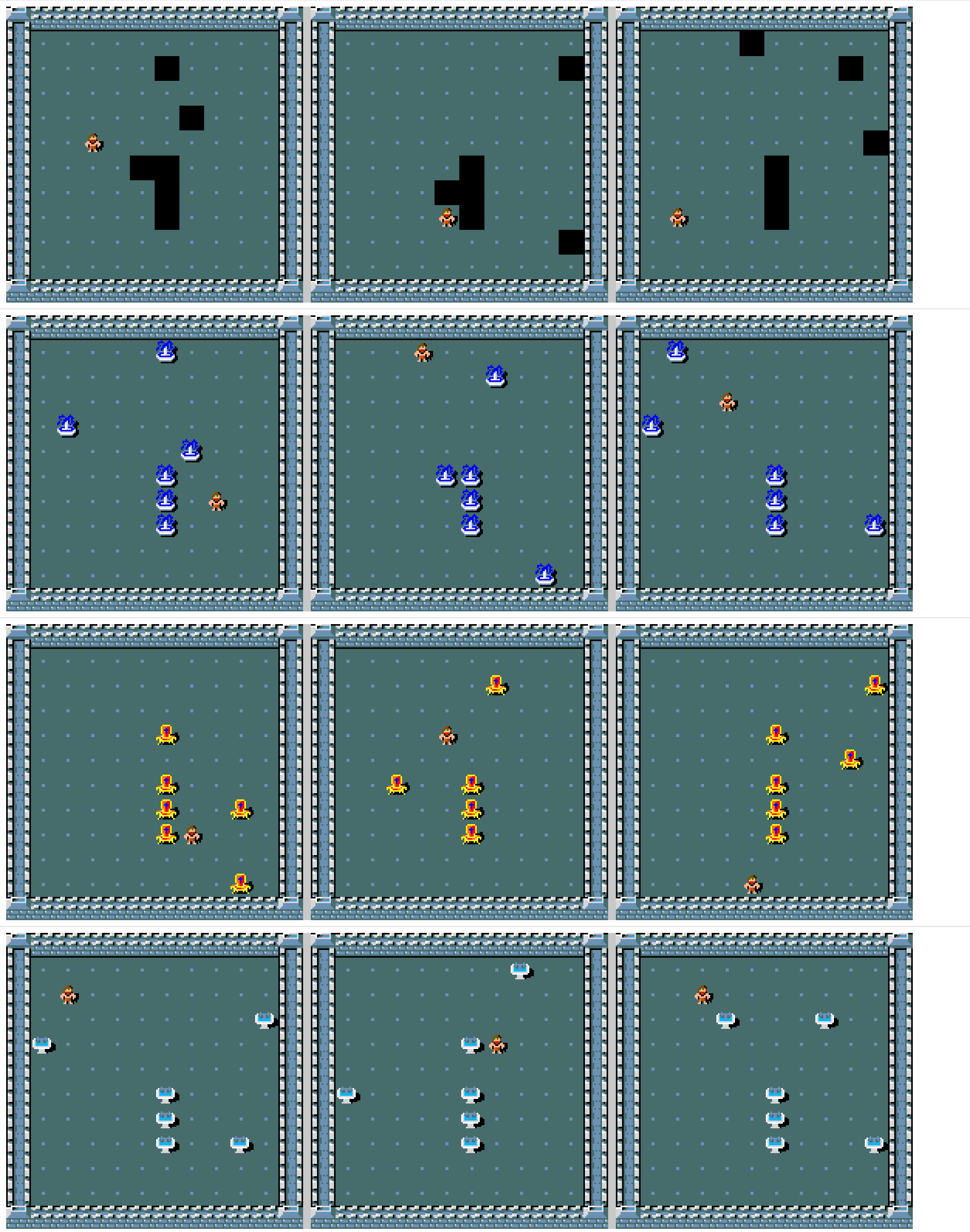
```
    "A",  # air
    "B",  # crosswall / boundary symbol hack (See REGION)
    "C",  # cloud
    "S",  # secret door
    "H",  # secret corridor
    "{",  # fountain
    "\\", # throne
    "K",  # sink
    "}",  # moat
    "P",  # pool of water
    "L",  # lava pool
    "I",  # ice
    "W",  # water
    "T",  # tree
    "F",  # iron bars
]

for terrain in Terrains:
    render_des_file(des_file.replace("%terrain%", terrain), n_images=3, full_
↪screen=False)
```

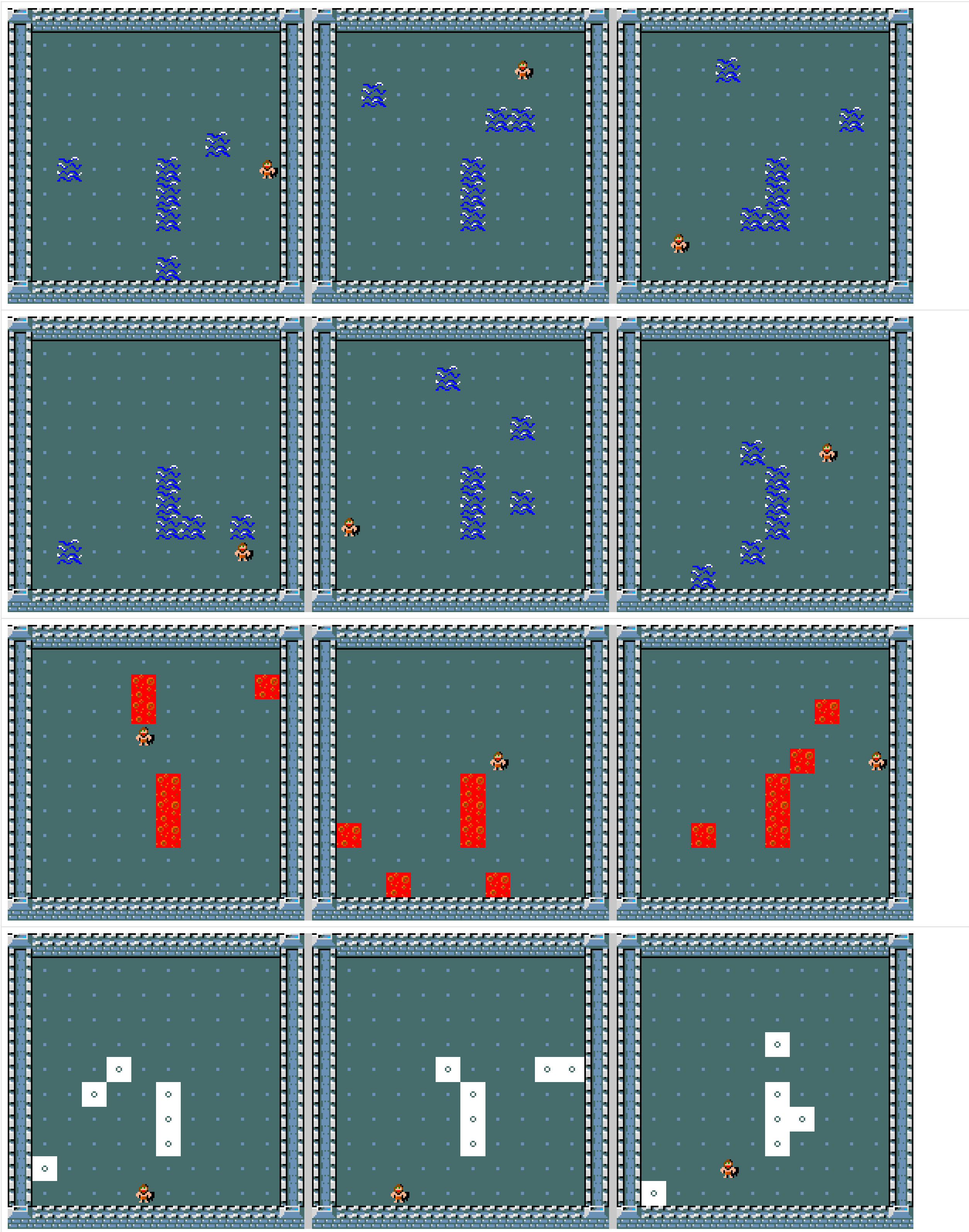


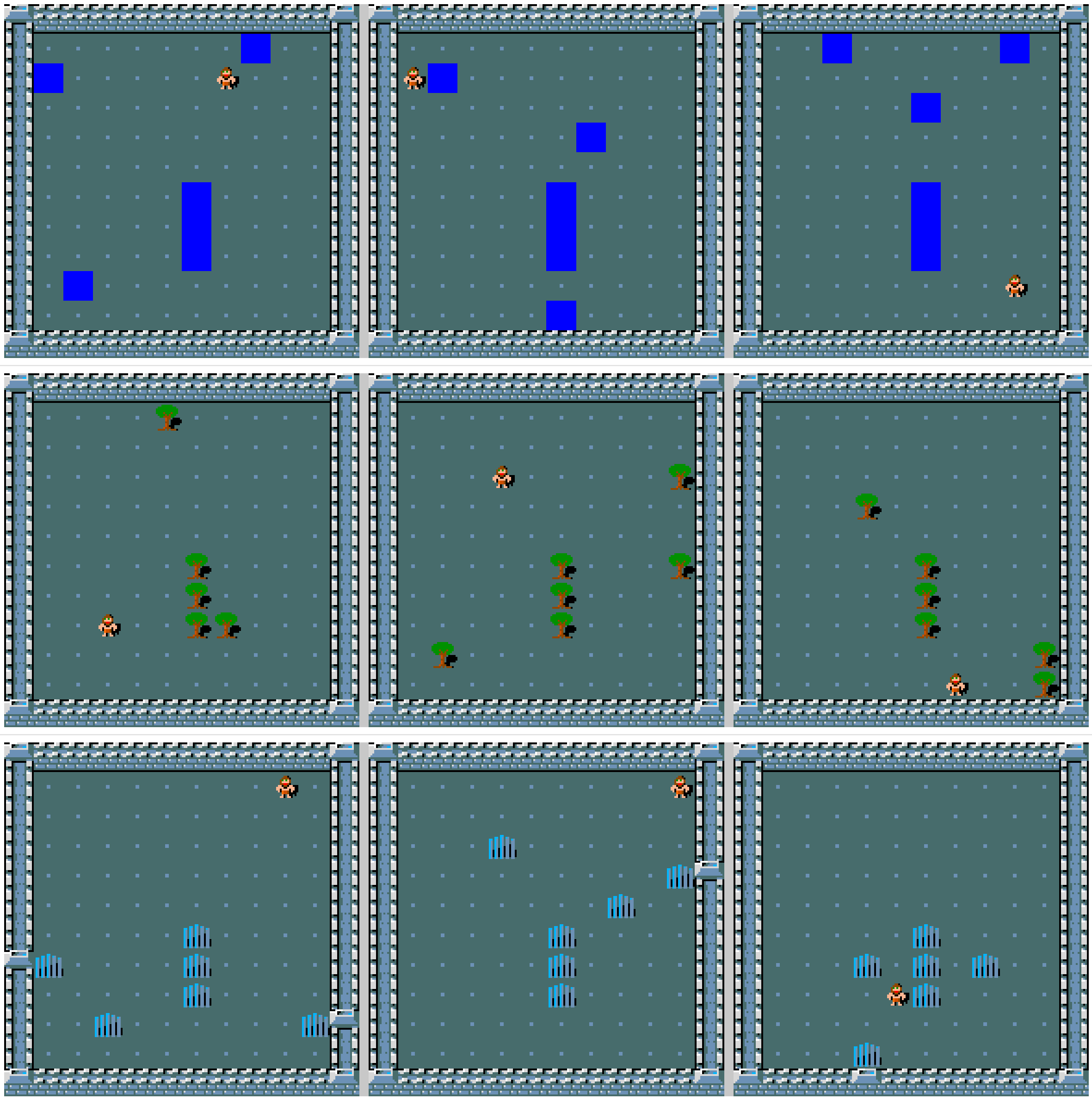












Adding terrain in fixed or entirely random places is great, but what if we want to control where the terrain is placed? For these we can use *selections*. These are collections of coordinates that can be combined, and then randomly selected from to place terrain. We can also use the selections without randomness to put terrain in an entire location. There are several ways of creating these selections. `rect` takes a region (specified by `(x_left, y_left, x_right, y_right)` specifying the top left and bottom right corners) and returns selection of coordinates on the edge of the rectangle. `fillrect` returns the entire filled rectangle coordinates

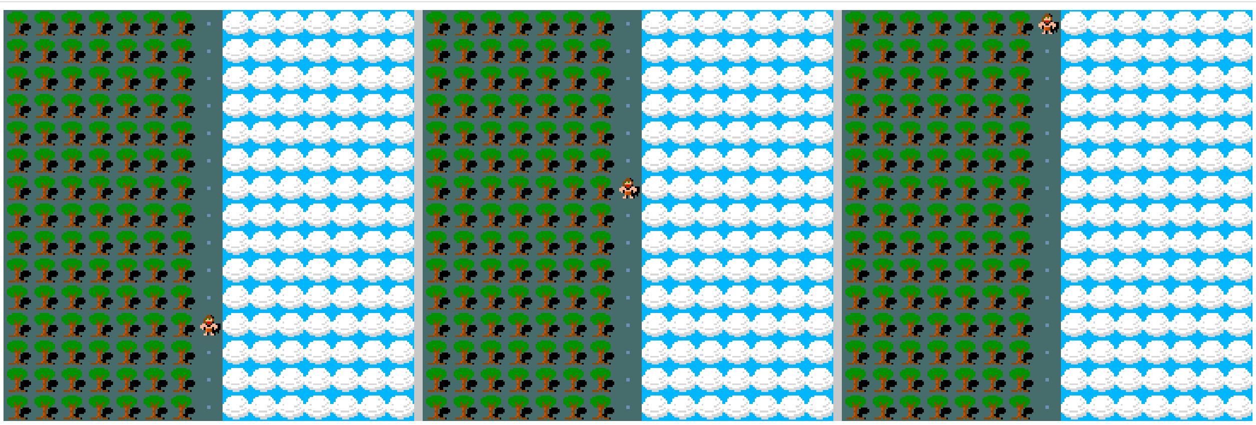
[18]:
```
des_file = """
MAZE: "mylevel", ' '
FLAGS:premapped
GEOMETRY:center,center
MAP
...............
...............
...............
```

(continues on next page)

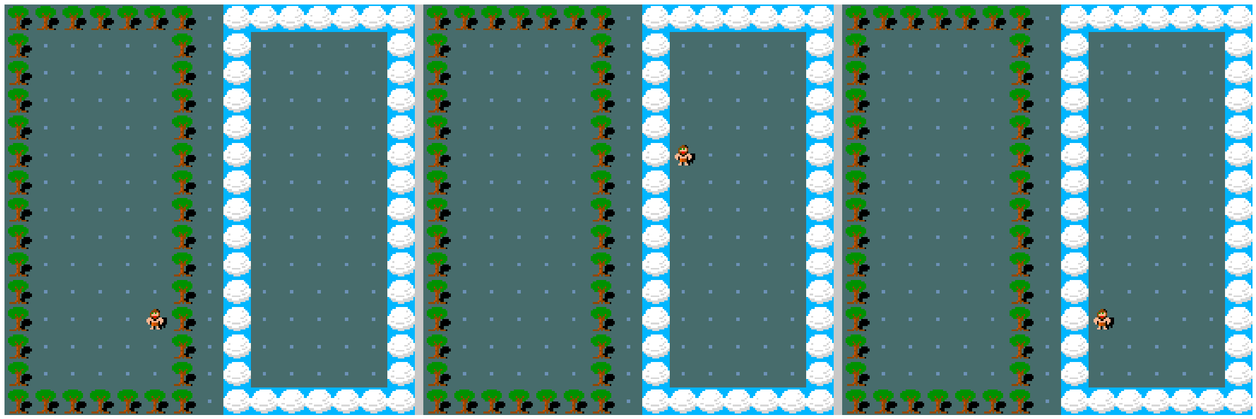
(continued from previous page)

```
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
ENDMAP
REGION: (0,0,20,20), lit, "ordinary"

$left = selection: %rect% (0,0,6,14)
$right = selection: %rect% (8,0,14,14)

TERRAIN: $left, 'T'
TERRAIN: $right, 'C'

"""
for rect in ("fillrect", "rect"):
    print(f"Using {rect}")
    render_des_file(des_file.replace("%rect%", rect), n_images=3)
```

```
Using fillrect
```

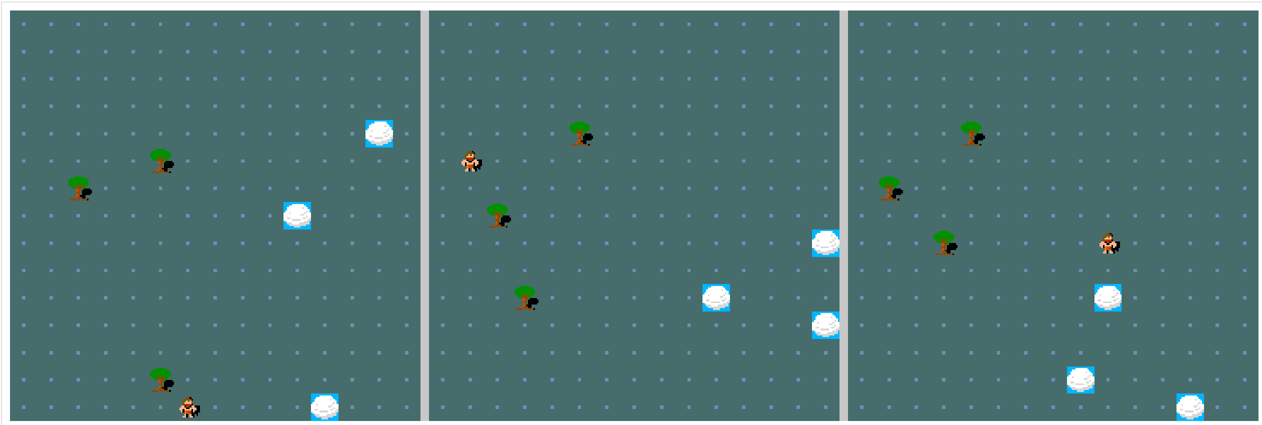


```
Using rect
```



We can select a single coordinate out of a selection using `rndcoord` as follows:

[19]:
```
des_file = """
MAZE: "mylevel", ' '
FLAGS:premapped
GEOMETRY:center,center
MAP
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
ENDMAP
REGION: (0,0,20,20), lit, "ordinary"

$left = selection: fillrect (0,0,6,14)
$right = selection: fillrect (8,0,14,14)

TERRAIN: rndcoord($left), 'T'
TERRAIN: rndcoord($left), 'T'
TERRAIN: rndcoord($left), 'T'
TERRAIN: rndcoord($right), 'C'
TERRAIN: rndcoord($right), 'C'
TERRAIN: rndcoord($right), 'C'

"""
render_des_file(des_file.replace("%rect%", rect), n_images=3)
```



We can also use the REPLACE_TERRAIN command, rather than just the TERRAIN command. This command takes a region, a terrain to replace, a terrain to replace with, and a probability (specified in a percentage), and then replaces each tile of the terrain to replace in the region with the terrain to replace with with the probability specified. Below, we randomly place some trees in the top left and clouds in the top right (by replacing floor), fill the bottom with lava and then randomly replace some of the laval with ice.

[20]:
```
des_file = """
MAZE: "mylevel", ' '
FLAGS:premapped
GEOMETRY:center,center
MAP
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
ENDMAP
REGION: (0,0,20,20), lit, "ordinary"

$top_left_region = (0,0,6,6)
$top_right_region = (8,0,14,6)
$bottom_region = (0,8,14,14)

REPLACE_TERRAIN: $top_left_region, '.', 'T', 20%
REPLACE_TERRAIN: $top_right_region, '.', 'C', 20%

TERRAIN: fillrect $bottom_region, 'L'
REPLACE_TERRAIN: $bottom_region, 'L','I', 50%
```

(continues on next page)

(continued from previous page)

```
"""
render_des_file(des_file, n_images=3)
```

Another way of generating selections (instead of `rect` or `fillrect`) is using `line` or `randline`. These both take a start and end position, and `randline` also takes a roughness parameter controlling how random the line is between the two coordinates. Here we generate a straight line of trees using `line`, and several different random lines of clouds with different roughnesses using `randline`

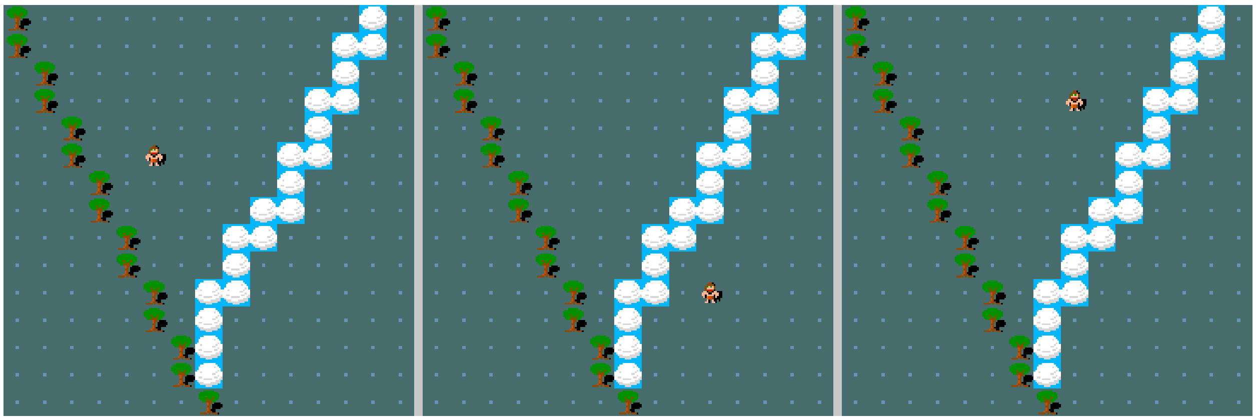
[21]:
```
des_file = """
MAZE: "mylevel", ' '
FLAGS:premapped
GEOMETRY:center,center
MAP
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
...............
ENDMAP
REGION: (0,0,20,20), lit, "ordinary"

$top_left = (0,0)
$top_right = (14,0)
$bottom_middle = (7,14)

$tree_line = selection: line $top_left,$bottom_middle
$cloud_line = selection: randline $top_right,$bottom_middle,%roughness%

TERRAIN: $tree_line, 'T'
```

(continues on next page)

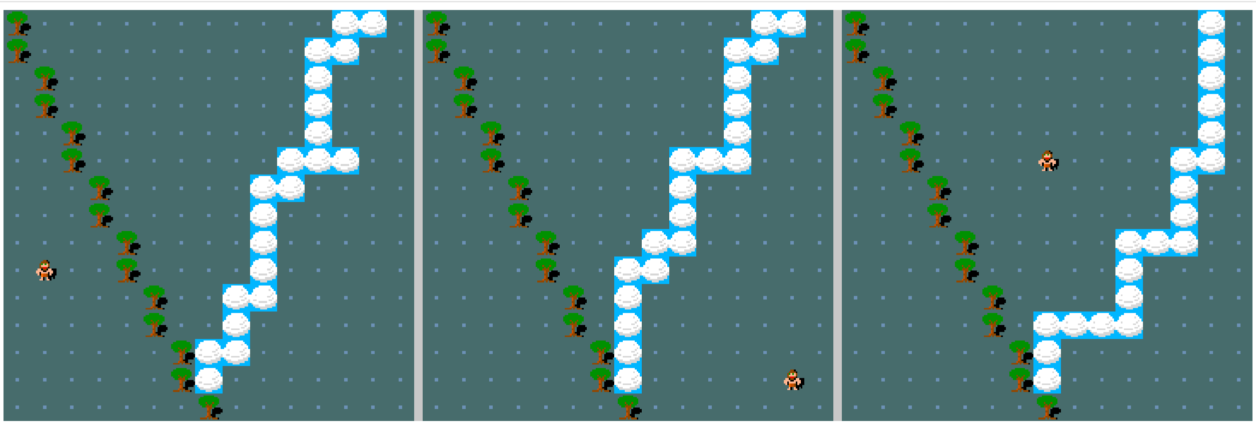
(continued from previous page)

```
TERRAIN: $cloud_line, 'C'

"""
for roughness in (0,5,15,30):
    print(f"Using roughness {roughness}")
    render_des_file(des_file.replace("%roughness%", str(roughness)), n_images=3)
```

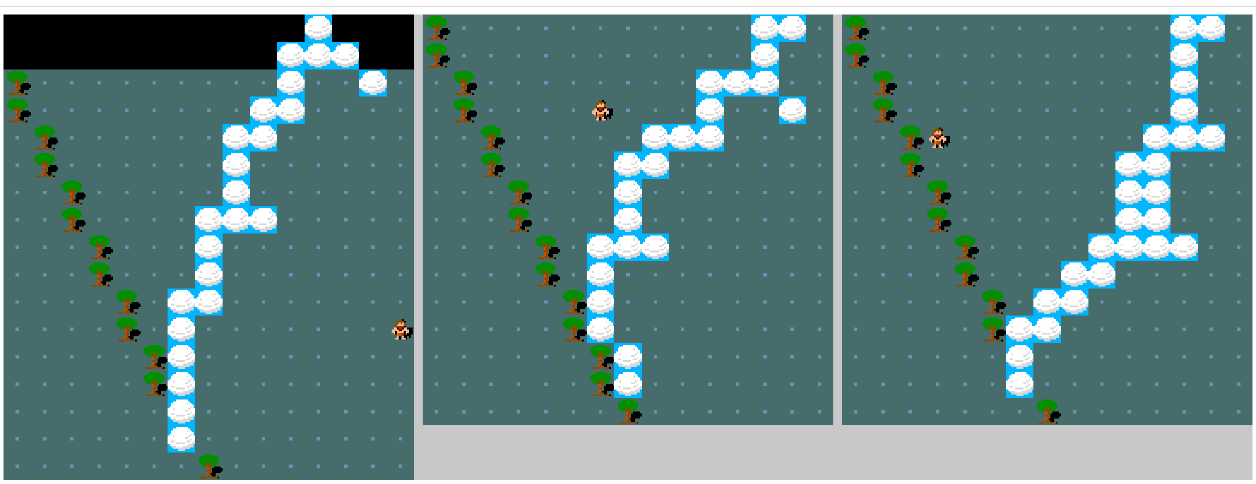
```
Using roughness 0
```



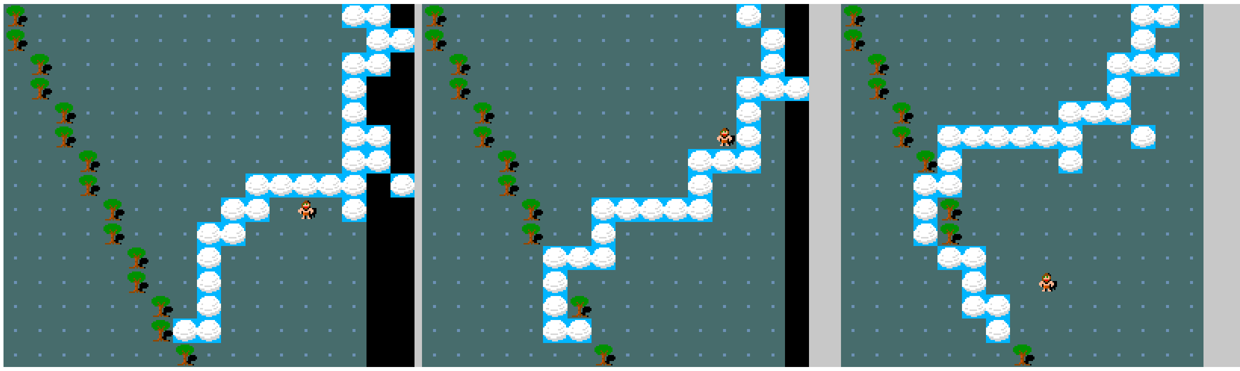
```
Using roughness 5
```



```
Using roughness 15
```



```
Using roughness 30
```



More information on different types of selections can be found in the nethack wiki here

Controlling Randomness

As we've seen so far, there are many different ways of controlling randomness in des-files. We'll cover them all in this section:

- `randline` and `rndcoord` we've already seen. They're ways of creating random selections or coordinates given some input
- The des-file format supports conditional statements using the `IF[...]` command. Inside the square brackets can either be a percentage (e.g. `IF[50%]`) or a comparison (e.g. `IF[4 < $variable]`)
- dice-rolls can be used to generate random integers. They take the form `MdN`, which means to roll `M` `N`-sided die and sum the result (e.g. `2d4`). These can be used in `IF` statements, like `IF[2d4 < 6]`, or any other place an integer is used.
- Arrays can be created in des-files, supporting lists of any object of the same type. The `SHUFFLE` command can be used to randomise the order of a list after it's been created, and then the list can be accessed by index to get a random element.

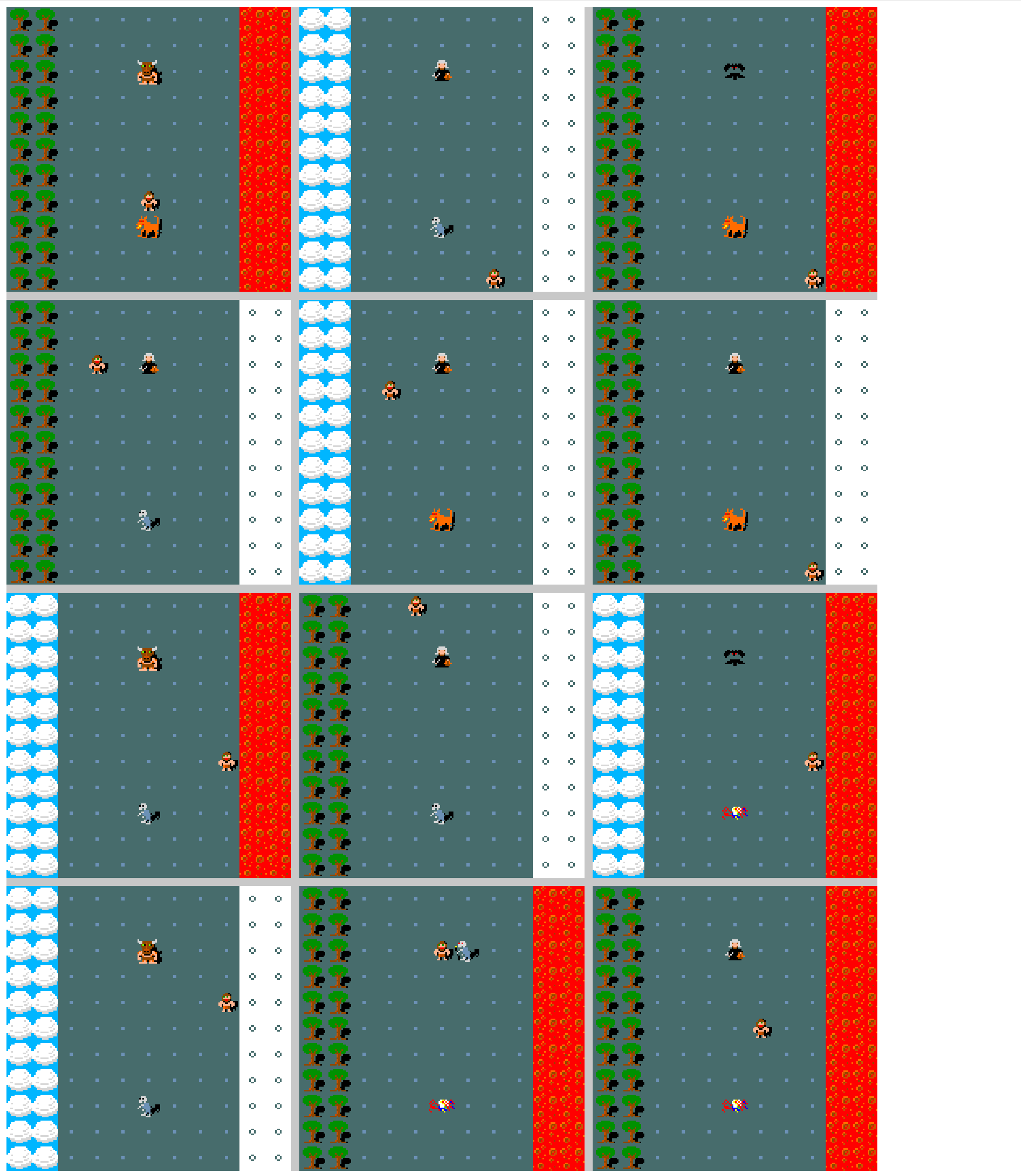
Here we use `IF[%50]` conditionals to place either trees or clouds on the left of the map, and dice-rolls to randomly place lava or ice on the right. We use shuffle to randomly pick one monster, and a dice-roll to randomly pick another

[22]:
```
des_file = """
MAZE: "mylevel", ' '
FLAGS:premapped
GEOMETRY:center,center
MAP
...........
...........
...........
...........
...........
...........
...........
...........
...........
...........
...........
ENDMAP
```

(continues on next page)

(continued from previous page)

```
REGION: (0,0,20,20), lit, "ordinary"

$left = selection: fillrect (0,0,1,10)
$right = selection: fillrect (9,0,10,10)

IF [50%] {
    TERRAIN: $left, 'T'
} ELSE {
    TERRAIN: $left, 'C'
}

$roll = 2d6
IF [$roll < 7] {
    TERRAIN: $right, 'I'
} ELSE {
    TERRAIN: $right, 'L'
}

$mon_names = monster: { "Archon", "arch-lich", "vampire lord", "minotaur"}
SHUFFLE: $mon_names
MONSTER: $mon_names[0], (5,2), hostile

$mon_names_new = monster: { "Lich", "grid bug", "hell hound", "red mold"}
$mon_index = 1d4
MONSTER: $mon_names_new[$mon_index], (5,8), hostile
"""
render_des_file(des_file, n_images=12)
```



9.11.4 Wrapping up

That brings us to the end of our tutorial. There are more features of NetHack to explore, and we recommend using the NetHack wiki as a useful source of information. Be sure to use MiniHack and the des-file format to create lots of interesting reinforcement learning environments.

9.12 TorchBeast

To get started with MiniHack environments, we provide baseline agents using the TorchBeast framework. TorchBeast provides a PyTorch implementation of IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures.

TorchBeast comes in two variants: MonoBeast and PolyBeast. PolyBeast is the more powerful version of the framework and allows training agents across multiple machines. For further details, see the TorchBeast paper.

For MiniHack, we use the PolyBeast implementation of TorchBeast and additionally provide an implementation of the following exploration methods:

- RND: Exploration by Random Network Distillation
- RIDE: Rewarding Impact-Driven Exploration for Procedurally-Generated Environments

9.12.1 Installation

To install and train a polybeast agent in MiniHack, first install polybeast by following the instructions here, then use the following commands:

```
pip install ".[polybeast]"
# Test IMPALA run
python3 -m minihack.agent.polybeast.polyhydra env=MiniHack-Room-5x5-v0 total_steps=100000
```

9.12.2 Running Experiments

We use the hydra framework for configuring our experiments. All environment and training parameters can be specified using command line arguments (or edited directly in `config.yaml`). See `config.yaml` file in `minihack.agent.polybeast` for more information. Be sure to set up appropriate parameters for logging with wandb (disabled by default).

```
# Single IMPALA run
python3 -m minihack.agent.polybeast.polyhydra model=baseline env=MiniHack-Room-5x5-v0␣
↪total_steps=1000000

# Single RND run
python3 -m minihack.agent.polybeast.polyhydra model=rnd env=MiniHack-Room-5x5-v0 total_
↪steps=1000000

# Single RND run
python3 -m minihack.agent.polybeast.polyhydra model=ride state_counter=coordinates␣
↪env=MiniHack-Room-5x5-v0 total_steps=1000000
```

(continues on next page)

(continued from previous page)

```
# To perform a sweep on the cluster: add another --multirun command and comma-separate␣
↪values
python3 -m minihack.agent.polybeast.polyhydra --multirun model=baseline,rnd env=MiniHack-
↪Room-Random-15x15-v0,MiniHack-Room-Monster-15x15-v0 total_steps=10000000
```

9.12.3 Replicating the Results of the Paper

To replicate results of the paper performed using polybeast, simply run a sweep of 5 runs with IMPALA, RND or RIDE agents on the desired environments as follows:

```
python3 -m minihack.agent.polybeast.polyhydra --multirun model=baseline name=1,2,3,4,5␣
↪env=MiniHack-Room-Random-15x15-v0,MiniHack-Room-Monster-15x15-v0 total_steps=10000000
```

For navigation tasks, the default parameters are already set. For skill acquisition tasks, additionally set `learning_rate=0.00005 msg.model=lt_cnn`.

The learning curves for all of our polybeast experiments can be accessed in our Weights&Biases repository.

9.12.4 Evaluate and Watch

The following script allows to evaluate the performance of a model pre-trained with polybeast:

```
# Watch the learned behaviour step-by-step in the terminal
python3 -m minihack.agent.polybeast.evaluate --env MiniHack-Room-5x5-v0 -c /path/to/
↪checkpoint/directory --watch

# Evaluate the pre-trained model for 1 episode and save the replay as a GIF file
python3 -m minihack.agent.polybeast.evaluate --env MiniHack-Room-5x5-v0 -c /path/to/
↪checkpoint/directory -n 1 --no-watch --save_gif --gif_path replay.gif

# Print all options of the evaluation script
python3 -m minihack.agent.polybeast.evaluate --help
```

9.13 RLlib

MiniHack additionally provides support for agents using the RLlib library. RLlib is an open-source library for reinforcement learning that offers high scalability and a unified API for a variety of applications. RLlib natively supports TensorFlow, TensorFlow Eager, and PyTorch. RLlib includes implementations of many popular algorithms, including IMPALA, PPO, Rainbow DQN, A3C, and many more. The full list of algorithms is available here.

For further details, checkout out RLlib paper and blog post.

9.13.1 Installation

To install and train an RLlib agent use the following commands:

```
pip install -e ".[rllib]"
# Test DQN run
python3 -m minihack.agent.rllib.train algo=dqn env=MiniHack-Room-5x5-v0 total_
↪steps=1000000 lr=0.000001
```

9.13.2 Running Experiments

We use the hydra framework for configuring our experiments. All environment and training parameters can be specified using command line arguments (or edited directly in `config.yaml`). See `config.yaml` file in `minihack.agent.rllib` for more information. Be sure to set up appropriate parameters for logging with wandb (disabled by default).

```
# Single A2C run
python3 -m minihack.agent.rllib.train algo=a2c env=MiniHack-Room-15x15-v0 total_
↪steps=1000000 a2c.entropy_coeff=0.001 lr=0.00001

# Single PPO run
python3 -m minihack.agent.rllib.train algo=ppo env=MiniHack-Room-15x15-v0 total_
↪steps=1000000 ppo.entropy_coeff=0.0001 lr=0.00001

# Single DQN run
python3 -m minihack.agent.rllib.train algo=dqn env=MiniHack-Room-15x15-v0 total_
↪steps=1000000 dqn.buffer_size=100000 lr=0.000001

# To perform a sweep on the cluster: add another --multirun command and comma-separate␣
↪values
python3 -m minihack.agent.rllib.train --multirun algo=a2c env=MiniHack-Room-15x15-v0␣
↪lr=0.00001 seed=0,1,2,3,4 total_steps=5000000
```

9.14 Unsupervised Environment Design

MiniHack also enables research in *Unsupervised Environment Design*, whereby an adaptive task distribution is learned during training by dynamically adjusting free parameters of the task MDP.

Check out the ucl-dark/paired repository for replicating the examples from the paper using the PAIRED.

9.15 minihack package

class minihack.**LevelGenerator**(*map=None*, *w=8*, *h=8*, *fill='.'*, *lit=True*, *flags=('hardfloor')*, *solidfill=' '*)

Bases: `object`

LevelGenerator provides a convenient Python interface for quickly writing description files for MiniHack. The LevelGenerator class can be used to create MAZE-type levels with specified heights and widths, and can then fill those levels with objects, monsters and terrain, and specify the start point of the level.

Parameters

- **map** (*str or None*) – The description of the map block of the environment. If None, the map will have a rectangle layout with the given height and width. Defaults to None.
- **w** (*int*) – The width of map. Only used when *map=None*. Defaults to 8.
- **h** (*int*) – The height of map. Only used when *map=None*. Defaults to 8.
- **fill** (*str*) – A character describing the environment feature that fills the map. Only used when `map=None`. Defaults to ".", which corresponds to floor.
- **lit** (*bool*) – Whether the layout is lit or not. This affects the observations the agent will receive. If an area is not lit, the agent can only see directly adjacent grids. Defaults to True.
- **flags** (*tuple*) – Flags of the environment. For the full list, see https://nethackwiki.com/wiki/Des-file_format#FLAGS. Defaults to ("hardfloor",).
- **solidfill** (*str*) – A character describing the environment feature used for filling solid / unspecified parts of the map. Defaults to " ", which corresponds to solid wall.

__init__(*map=None*, *w=8*, *h=8*, *fill='.'*, *lit=True*, *flags=('hardfloor')*, *solidfill=' '*)
Initialize self. See help(type(self)) for accurate signature.

add_altar(*place=None*, *align='random'*, *type='random'*)
Add an altar.

Parameters

- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.
- **align** (*str*) – The alignment. Possible values are "noalign", "law", "neutral", "chaos", "coaligned", "noncoaligned", and "random". Defaults to "random".
- **type** (*str*) – The type of the altar. Possible values are "sanctum", "shrine", "altar", and "random". Defaults to random.

add_boulder(*place=None*)
Add a boulder to the floor.

Parameters

- **amount** (*int*) – The amount of gold.
- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

add_door(*state*, *place=None*)
Add a door.

Parameters

- **state** (*str*) – The state of the door. Possible values are "locked", "closed", "open", "nodoor", and "random". Defaults to "random".
- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

add_fountain(*place=None*)
Add a fountain.

Parameters **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

add_goal_pos(*place=None*)
Add a goal at the given place. Same as *add_stair_down*.

add_gold(*amount*, *place=None*)
Add gold on the floor.

Parameters

- **amount** (*int*) – The amount of gold.
- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

add_line(*str*)
Add a custom string to the buttom of the description file.

Parameters **str** (*str*) – The string to be concatenated to the des-file.

add_mazewalk(*coord=None*, *dir='east'*)
Creates a random maze, starting from the given coordinate.

Mazewalk turns map grids with solid stone into floor. From the starting position, it checks the mapgrid in the direction given, and if it's solid stone, it will move there, and turn that place into floor. Then it will choose a random direction, jump over the nearest mapgrid in that direction, and check the next mapgrid for solid stone. If there is solid stone, mazewalk will move that direction, changing that place and the intervening mapgrid to floor. Normally the generated maze will not have any loops.

Pointing mazewalk at that will create a small maze of trees, but unless the map (at the place where it's put into the level) is surrounded by something else than solid stone, mazewalk will get out of that MAP. Substituting floor characters for some of the trees "in the maze" will make loops in the maze, which are not otherwise possible. Substituting floor characters for some of the trees at the edges of the map will make maze entrances and exits at those places.

For more details see https://nethackwiki.com/wiki/Des-file_format#MAZEWALK.

Parameters **coord** – A tuple with length two representing the (x, y) coordinates. If None is passed, the middle point of the map is selected. Defaults to None.

add_monster(*name='random'*, *symbol=None*, *place=None*, *args=()*)
Add a monster to the map.

Parameters

- **name** (*str*) – The name of the monster. Defaults to random.
- **symbol** (*str or None*) – The symbol of the monster. The symbol should correspond to the family of the specified mosnter. For example, "d" symbol corresponds to canine monsters, so the name of the object should also correspond to canines (e.g. jackal). Not used when name is "random". Defaults to None.
- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.
- **args** (*tuple*) – Additional monster arguments, e.g. "hostile" or "peaceful", "asleep" or "awake", etc. For more details, see https://nethackwiki.com/wiki/Des-file_format#MONSTER.

`add_object`(*name='random'*, *symbol='%'*, *place=None*, *cursestate=None*)
Add an object to the map.

Parameters

- **name** (*str*) – The name of the object. Defaults to random.
- **symbol** (*str*) – The symbol of the object. The symbol should correspond to the given object name. For example, "%" symbol corresponds to comestibles, so the name of the object should also correspond to commestibles (e.g. apple). Not used when name is "random". Defaults to "%".
- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.
- **cursetstate** (*str or None*) – The cursed state of the object. Can be "blessed", "uncursed", "cursed" or "random". Defaults to None (not used).

`add_object_area`(*area_name*, *name='random'*, *symbol='%'*, *cursestate=None*)
Add an object in an area of the map defined by *area_name* variable. See `add_object` for more details.

`add_sink`(*place=None*)
Add a sink.

Parameters **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

`add_stair_down`(*place=None*)
Add a stair down at the given place.

Parameters **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

`add_terrain`(*coord*, *flag*, *in_footer=False*)
Add terrain features to the map.

Parameters

- **coord** (*tuple*) – A tuple with length two representing the (x, y) coordinates.
- **flag** (*str*) – The flag corresponding to the desired terrain feature. Should belong to minihack.level_generator.MAP_CHARS. For more details, see https://nethackwiki.com/wiki/Des-file_format#Map_characters
- **in_footer** (*bool*) – Whether to define the terrain feature as an additional line in the description file (True) or directly modify the map block with the given flag (False). Defaults to False.

`add_trap`(*name='teleport'*, *place=None*)
Add a trap.

Parameters

- **name** (*str*) – The name of the trap. For possible values, see *minihack.level_generator.TRAP_NAMES*. Defaults to "teleport".
- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

fill_terrain(*type*, *flag*, *x1*, *y1*, *x2*, *y2*)
Fill the areas between (x1, y1) and (x2, y2) with the given dungeon feature:

Parameters

- **type** (*str*) – The type of filling. "rect" indicates an unfilled rectangle, containing just the edges and none of the interior points. "fillrect" denotes filled rectangle containing the edges and all of the interior points. "line" is used for a straight line drawn from one pair of coordinates to the other using Bresenham's line algorithm.
- **flag** (*str*) – The flag corresponding to the desired terrain feature. Should belong to minihack.level_generator.MAP_CHARS. For more details, see https://nethackwiki.com/wiki/Des-file_format#Map_characters
- **x1** (*int*) – x coordinate of point 1.
- **y1** (*int*) – y coordinate of point 1.
- **x2** (*int*) – x coordinate of point 2.
- **y2** (*int*) – y coordinate of point 2.

get_des()
Returns the description file.

Returns the description file as a string.

Return type str

get_map_array()
Returns the map as a two-dimensional numpy array.

get_map_str()
Returns the map as a string.

init_map(*map=None*, *x=8*, *y=8*, *fill='.'*)
Initialise the map block of the des-file.

set_area_variable(*var_name*, *type*, *x1*, *y1*, *x2*, *y2*)
Set a variable representing an area on the map.

Parameters

- **var_name** (*str*) – The name of the variable.
- **type** (*str*) – The type of filling. "rect" indicates an unfilled rectangle, containing just the edges and none of the interior points. "fillrect" denotes filled rectangle containing the edges and all of the interior points. "line" is used for a straight line drawn from one pair of coordinates to the other using Bresenham's line algorithm.
- **x1** (*int*) – x coordinate of point 1.
- **y1** (*int*) – y coordinate of point 1.
- **x2** (*int*) – x coordinate of point 2.
- **y2** (*int*) – y coordinate of point 2.

set_start_pos(*coord*)
Set the starting position of the agent.

Parameters **coord** (*tuple*) – A tuple with length two representing the (x, y) coordinates.

set_start_rect(*p1*, *p2*)
Set the starting position of the agent.

Parameters **coord** (*tuple*) – A tuple with length two representing the (x, y) coordinates.

wallify()

Wallify the map. Turns walls completely surrounded by other walls into solid stone ' '.

class minihack.**MiniHack**(**args: Any*, ***kwargs: Any*)

Bases: `nle.env.tasks.`

MiniHack base class.

All MiniHack environments are derived from this class, which itself is derived from NLE base class.

Note that this class itself is not used for creating new environment instances. Instead, `MiniHackNavigation` and `MiniHackSkill` provide a more convenient interface for doing this, both of which are directly derived from MiniHack for specific types of environments.

__init__(**args*, *des_file: str*, *reward_win=1*, *reward_lose=0*, *obs_crop_h=9*, *obs_crop_w=9*, *obs_crop_pad=0*, *reward_manager=None*, *use_wiki=False*, *autopickup=True*, *pet=False*, *observation_keys=['glyphs', 'chars', 'colors', 'specials', 'glyphs_crop', 'chars_crop', 'colors_crop', 'specials_crop', 'blstats', 'message']*, *seeds=None*, *include_see_actions=True*, *include_alignment_blstats=True*, ***kwargs*)

Constructs a new MiniHack environment.

Parameters

- **des_file** (*str*) – The description file for the environment.
- **reward_win** (*float*) – The reward received upon successfully completing an episode. Defaults to 1.
- **reward_lose** (*float*) – The reward received upon death or aborting. Defaults to 1.
- **obs_crop_h** (*int*) – The height of agent-centred cropped observation. Defaults to 9.
- **obs_crop_w** (*int*) – The width of agent-centred cropped observation. Defaults to 9.
- **obs_crop_pad** (*int*) – The padding for agent-centred cropped observation. Defaults to 0.
- **reward_manager** (*RewardManager or None*) – The reward manager that describes the custom reward function of the agent. If None, the goal of the agent is to reach the stair down. Defaults to None.
- **use_wiki** (*bool*) – Whether to use the NetHack wiki. Defaults to False.
- **autopickup** (*bool*) – Turning autopickup on or off. Defaults to True.
- **pet** (*bool*) – Whether to include the pet. Defaults to False.
- **observation_keys** (*list*) – The keys of observations returned after every timestep by the environment as a dictionary. Defaults to `minihack.base.MH_DEFAULT_OBS_KEYS`.
- **seeds** (*list or None*) – A list of integers used as level seeds for sampling episodes. The reset()` function samples a seed from this list uniformly at random and uses it for setting the level. When the `sample_seed` argument of the reset function is set to False, a random level will not be sampled from this list during environment resetting. If None, the entire level distribution is used. Defaults to None.
- **penalty_mode** (*str*) – The name of the mode for calculating the time step penalty. Can be `constant`, `exp`, `square`, `linear`, or `always`. Defaults to `constant`. Inherited from *NetHackScore*.
- **penalty_step** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.01. Inherited from *NetHackScore*.

- **penalty_time** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.0. Inherited from *NetHackScore*.
- **save_ttyrec_every** (*int*) – Integer, if 0, no ttyrecs (game recordings) will be saved. Otherwise, save a ttyrec every Nth episode. Defaults to 0. Inherited from *NLE*.
- **savedir** (*str or None*) – Path to save ttyrecs (game recordings) into, if save_ttyrec_every is nonzero. If nonempty string, interpreted as a path to a new or existing directory. If "" (empty string) or None, NLE choses a unique directory name. Defaults to None. Inherited from *NLE*.
- **character** (*str*) – Name of character. Defaults to "mon-hum-neu-mal". Interited from *NLE*.
- **max_episode_steps** (*int*) – maximum amount of steps allowed before the game is forcefully quit. In such cases, `info["end_status"]` ill be equal to `StepStatus.ABORTED`. Defaults to 200. Inherited from *NLE*.
- **actions** (*list*) – list of actions. If None, the full action space will be used, i.e. `nle.nethack.ACTIONS`. Defaults to *MH_FULL_ACTIONS*. Inherited from *NLE*.
- **wizard** (*bool*) – activate wizard mode. Defaults to False. Inherited from *NLE*.
- **allow_all_yn_questions** (*bool*) – If set to True, no y/n questions in step() are declined. If set to False, only elements of SKIP_EXCEPTIONS are not declined. Defaults to True. Inherited from *NLE*.
- **allow_all_modes** (*bool*) – If set to True, do not decline menus, text input or auto 'MORE'. If set to False, only skip click through 'MORE' on death. Defaults to False. Inherited from *NLE*.
- **spawn_monsters** (*bool*) – If False, disables normal NetHack behavior to randomly create monsters. Defaults to False. Inherited from *NLE*.
- **include_see_actions** (*bool*) – If True, the agent's action space includes the additional *NLE* actions introduced in the 0.8.1 release. Has no effect when the *actions* parameter is specified. Defaults to True.
- **include_alignment_blstats** (*bool*) – If True, the agent's observation space includes the alignment information in the blstats. This is introduced in *NLE* 0.9.0 release. Defaults to True.

get_neighbor_descriptions(*observation=None*)
Returns the descriptions of nine neighboring grids around the agent.

get_neighbor_wiki_pages(*observation=None*)
Returns the page contents of the neighboring objects from NetHack wiki.

get_object_direction(*name*, *observation=None*)
Find the game direction of the (first) object in the neighboring nine tiles that contains the given name in its description.

Parameters

- **name** (*str*) – Name of the object.
- **observation** (*dict*) – Agent observation.

Returns The index of the direction. None if not found.

Return type int

get_screen_description(*x*, *y*, *observation=None*)
Returns the description of the screen on (x,y) coordinates.

get_screen_wiki_page(*x*, *y*, *observation=None*)
Returns the wiki page matching the object on (x,y) coordinates.

key_in_inventory(*name*)
Returns key of the given object in the inventory.

Parameters **name** (*str*) – Name of the object.

Returns the key of the first item in the inventory that includes the argument name as a substring. Returns None if not found.

Return type str

reset(**args*, *sample_seed=True*, ***kwargs*)

screen_contains(*name*, *observation=None*)
Whether an object with the given name is visible on the screen, i.e. included in the screen descriptions of the observation dictionary.

Parameters

- **name** (*str*) – Name of the object or monster.
- **observation** (*dict*) – Agent observation.

Returns True if the name is contained on the screen, False otherwise.

Return type bool

step(*action: int*)

update(*des_file*)
Update the current environment by replacing its description file.

class minihack.**MiniHackNavigation**(**args: Any*, ***kwargs: Any*)
Bases: `nle.env.tasks.`

The base class for MiniHack Navigation tasks.

Navigation tasks have the following characteristics:

- Restricted action space: By default, the agent can only move towards eight compass directions.
- Yes/No questions, as well as menu-selection actions are disabled by default.
- The character is set to chaotic human male rogue.
- Auto-pick is enabled by default.
- Maximum episode limit defaults to 100 (can be overriden via the *max_episode_steps* argument)
- The default goal is to reach the stair down. This can be changed using a reward manager.

__init__(**args*, *des_file: Optional[str] = None*, ***kwargs*)
Constructs a new MiniHack environment.

Parameters

- **des_file** (*str*) – The description file for the environment.
- **reward_win** (*float*) – The reward received upon successfully completing an episode. Defaults to 1.
- **reward_lose** (*float*) – The reward received upon death or aborting. Defaults to 1.
- **obs_crop_h** (*int*) – The height of agent-centred cropped observation. Defaults to 9.
- **obs_crop_w** (*int*) – The width of agent-centred cropped observation. Defaults to 9.

- **obs_crop_pad** (*int*) – The padding for agent-centred cropped observation. Defaults to 0.
- **reward_manager** (RewardManager *or None*) – The reward manager that describes the custom reward function of the agent. If None, the goal of the agent is to reach the stair down. Defaults to None.
- **use_wiki** (*bool*) – Whether to use the NetHack wiki. Defaults to False.
- **autopickup** (*bool*) – Turning autopickup on or off. Defaults to True.
- **pet** (*bool*) – Whether to include the pet. Defaults to False.
- **observation_keys** (*list*) – The keys of observations returned after every timestep by the environment as a dictionary. Defaults to `minihack.base.MH_DEFAULT_OBS_KEYS`.
- **seeds** (*list or None*) – A list of integers used as level seeds for sampling episodes. The reset()` function samples a seed from this list uniformly at random and uses it for setting the level. When the `sample_seed` argument of the reset function is set to False, a random level will not be sampled from this list during environment resetting. If None, the entire level distribution is used. Defaults to None.
- **penalty_mode** (*str*) – The name of the mode for calculating the time step penalty. Can be `constant`, `exp`, `square`, `linear`, or `always`. Defaults to `constant`. Inherited from *NetHackScore*.
- **penalty_step** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.01. Inherited from *NetHackScore*.
- **penalty_time** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.0. Inherited from *NetHackScore*.
- **save_ttyrec_every** (*int*) – Integer, if 0, no ttyrecs (game recordings) will be saved. Otherwise, save a ttyrec every Nth episode. Defaults to 0. Inherited from *NLE*.
- **savedir** (*str or None*) – Path to save ttyrecs (game recordings) into, if save_ttyrec_every is nonzero. If nonempty string, interpreted as a path to a new or existing directory. If "" (empty string) or None, NLE choses a unique directory name. Defaults to None. Inherited from *NLE*.
- **character** (*str*) – Name of character. Defaults to "mon-hum-neu-mal". Interited from *NLE*.
- **max_episode_steps** (*int*) – maximum amount of steps allowed before the game is forcefully quit. In such cases, `info["end_status"]` ill be equal to `StepStatus.ABORTED`. Defaults to 200. Inherited from *NLE*.
- **actions** (*list*) – list of actions. If None, the full action space will be used, i.e. `nle.nethack.ACTIONS`. Defaults to *MH_FULL_ACTIONS*. Inherited from *NLE*.
- **wizard** (*bool*) – activate wizard mode. Defaults to False. Inherited from *NLE*.
- **allow_all_yn_questions** (*bool*) – If set to True, no y/n questions in step() are declined. If set to False, only elements of SKIP_EXCEPTIONS are not declined. Defaults to True. Inherited from *NLE*.
- **allow_all_modes** (*bool*) – If set to True, do not decline menus, text input or auto 'MORE'. If set to False, only skip click through 'MORE' on death. Defaults to False. Inherited from *NLE*.
- **spawn_monsters** (*bool*) – If False, disables normal NetHack behavior to randomly create monsters. Defaults to False. Inherited from *NLE*.

- **include_see_actions** (*bool*) – If True, the agent's action space includes the additional *NLE* actions introduced in the 0.8.1 release. Has no effect when the *actions* parameter is specified. Defaults to True.
- **include_alignment_blstats** (*bool*) – If True, the agent's observation space includes the alignment information in the blstats. This is introduced in *NLE* 0.9.0 release. Defaults to True.

class minihack.**MiniHackSkill**(**args: Any*, ***kwargs: Any*)

Bases: `nle.env.tasks.`

The base class for MiniHack Skill Acquisition tasks.

Navigation tasks have the following characteristics:

- The full action space is used.
- Yes/No questions are enabled, but the menu-selection actions are disabled by default.
- The character is set to a neutral human male caveman.
- Maximum episode limit defaults to 250 (can be overriden via the *max_episode_steps* argument)
- The default goal is to reach the stair down. This can be changed using a reward manager.
- Auto-pick is disabled by default.
- Inventory strings and corresponding letter are also included as part of the agent observations.

__init__(**args*, *des_file*, ***kwargs*)

Constructs a new MiniHack environment.

Parameters

- **des_file** (*str*) – The description file for the environment.
- **reward_win** (*float*) – The reward received upon successfully completing an episode. Defaults to 1.
- **reward_lose** (*float*) – The reward received upon death or aborting. Defaults to 1.
- **obs_crop_h** (*int*) – The height of agent-centred cropped observation. Defaults to 9.
- **obs_crop_w** (*int*) – The width of agent-centred cropped observation. Defaults to 9.
- **obs_crop_pad** (*int*) – The padding for agent-centred cropped observation. Defaults to 0.
- **reward_manager** (*RewardManager or None*) – The reward manager that describes the custom reward function of the agent. If None, the goal of the agent is to reach the stair down. Defaults to None.
- **use_wiki** (*bool*) – Whether to use the NetHack wiki. Defaults to False.
- **autopickup** (*bool*) – Turning autopickup on or off. Defaults to True.
- **pet** (*bool*) – Whether to include the pet. Defaults to False.
- **observation_keys** (*list*) – The keys of observations returned after every timestep by the environment as a dictionary. Defaults to `minihack.base.MH_DEFAULT_OBS_KEYS`.
- **seeds** (*list or None*) – A list of integers used as level seeds for sampling episodes. The reset()` function samples a seed from this list uniformly at random and uses it for setting the level. When the `sample_seed` argument of the reset function is set to False, a random level will not be sampled from this list during environment resetting. If None, the entire level distribution is used. Defaults to None.

- **penalty_mode** (*str*) – The name of the mode for calculating the time step penalty. Can be `constant`, `exp`, `square`, `linear`, or `always`. Defaults to `constant`. Inherited from *NetHackScore*.
- **penalty_step** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.01. Inherited from *NetHackScore*.
- **penalty_time** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.0. Inherited from *NetHackScore*.
- **save_ttyrec_every** (*int*) – Integer, if 0, no ttyrecs (game recordings) will be saved. Otherwise, save a ttyrec every Nth episode. Defaults to 0. Inherited from *NLE*.
- **savedir** (*str or None*) – Path to save ttyrecs (game recordings) into, if save_ttyrec_every is nonzero. If nonempty string, interpreted as a path to a new or existing directory. If "" (empty string) or None, NLE choses a unique directory name. Defaults to None. Inherited from *NLE*.
- **character** (*str*) – Name of character. Defaults to "mon-hum-neu-mal". Interited from *NLE*.
- **max_episode_steps** (*int*) – maximum amount of steps allowed before the game is forcefully quit. In such cases, `info["end_status"]` ill be equal to `StepStatus.ABORTED`. Defaults to 200. Inherited from *NLE*.
- **actions** (*list*) – list of actions. If None, the full action space will be used, i.e. `nle.nethack.ACTIONS`. Defaults to *MH_FULL_ACTIONS*. Inherited from *NLE*.
- **wizard** (*bool*) – activate wizard mode. Defaults to False. Inherited from *NLE*.
- **allow_all_yn_questions** (*bool*) – If set to True, no y/n questions in step() are declined. If set to False, only elements of SKIP_EXCEPTIONS are not declined. Defaults to True. Inherited from *NLE*.
- **allow_all_modes** (*bool*) – If set to True, do not decline menus, text input or auto 'MORE'. If set to False, only skip click through 'MORE' on death. Defaults to False. Inherited from *NLE*.
- **spawn_monsters** (*bool*) – If False, disables normal NetHack behavior to randomly create monsters. Defaults to False. Inherited from *NLE*.
- **include_see_actions** (*bool*) – If True, the agent's action space includes the additional *NLE* actions introduced in the 0.8.1 release. Has no effect when the *actions* parameter is specified. Defaults to True.
- **include_alignment_blstats** (*bool*) – If True, the agent's observation space includes the alignment information in the blstats. This is introduced in *NLE* 0.9.0 release. Defaults to True.

class minihack.**NetHackWiki**(*raw_wiki_file_name: str*, *processed_wiki_file_name: str*, *save_processed_json: bool = True*, *ignore_inpage_anchors: bool = True*, *preprocess_input: bool = True*, *exceptions: Optional[tuple] = None*)

Bases: `object`

A class representing Nethack Wiki Data - pages and links between them.

Parameters

- **raw_wiki_file_name** (*str*) – The path to the raw file of NetHack wiki. The raw file can be downloaded using the *get_nhwiki_data.sh* script located in *minihack/scripts*.
- **processed_wiki_file_name** (*str*) – The path to the processed file of NetHack wiki. The processing is performed in the *__init__* function of this classed.

- **save_processed_json** (*bool*) – Whether to save the processed json file of the wiki. Only considered when a raw wiki file is passed. Defaults to True.
- **ignore_inpage_anchors** (*bool*) – Whether to ingnore in-page anchors. Defaults to True.
- **preprocess_input** (*bool*) – Whether to perform a preprocessing on wiki data. Defaults to True.
- **exceptions** (*Tuple[str] or None*) – Name of entities in screen descriptions that are ingored. If None, there are no exceptions. Defaults to None.

__init__(*raw_wiki_file_name: str*, *processed_wiki_file_name: str*, *save_processed_json: bool = True*, *ignore_inpage_anchors: bool = True*, *preprocess_input: bool = True*, *exceptions: Optional[tuple] = None*) → None

Initialize self. See help(type(self)) for accurate signature.

get_page_data(*page: str*) → dict

Get the data of a page.

Parameters **page** (*str*) – The page name.

Returns The page data as a dict.

Return type dict

get_page_text(*page: str*) → str

Get the text of a page.

Parameters **page** (*str*) – The page name.

Returns The text of the page.

Return type str

class minihack.**RewardManager**

Bases: *minihack.reward_manager.AbstractRewardManager*

This class is used for managing rewards, events and termination for MiniHack tasks.

Some notes on the ordering or calls in the MiniHack/NetHack base class:

- `step(action)` is called on the environment
- Within `step`, first a copy of the last observation is made, and then the underlying NetHack game is stepped
- Then `_is_episode_end(observation)` is called to check whether this the episode has ended (and this is overridden if we've gone over our max_steps, or the underlying NetHack game says we're done (i.e. we died)
- Then `_reward_fn(last_observation, observation)` is called to calculate the reward at this time-step
- if `end_status` tells us the game is done, we quit the game
- then `step` returns the observation, calculated reward, done, and some

statistics.

All this means that we need to check whether an observation is terminal in `_is_episode_end` before we're calculating the reward function.

The call of `_is_episode_end` in `MiniHack` will call `check_episode_end_call` in this class, which checks for termination and accumulates any reward, which is returned and zeroed in `collect_reward`.

__init__()

Initialize self. See help(type(self)) for accurate signature.

add_amulet_event(*reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)

Add event which is triggered when an amulet is worn.

Parameters

- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_coordinate_event(*coordinates: Tuple[int, int]*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)

Add event which is triggered on when reaching the specified coordinates.

Parameters

- **coordinates** (*Tuple[int, int]*) – The coordinates to be reached (tuple of ints).
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_custom_reward_fn(*reward_fn: Callable[[MiniHack, Any, int, Any], float]*) → None

Add a custom reward function which is called every after step to calculate reward.

The function should be a callable which takes the environment, previous observation, action and current observation and returns a float reward.

Parameters **reward_fn** (*Callable[[MiniHack, Any, int, Any], float]*) – A reward function which takes an environment, previous observation, action, next observation and returns a reward.

add_eat_event(*name: str*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)

Add an event which is triggered when *name* is eaten.

Parameters

- **name** (*str*) – The name of the object being eaten.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_event(*event: minihack.reward_manager.Event*)
Add an event to be managed by the reward manager.

Parameters **event** (Event) – The event to be added.

add_kill_event(*name: str, reward=1, repeatable=False, terminal_required=True, terminal_sufficient=False*)
Add event which is triggered when a specified monster is killed.

Parameters

- **name** (*str*) – The name of the monster to be killed.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_location_event(*location: str, reward=1, repeatable=False, terminal_required=True, terminal_sufficient=False*)
Add event which is triggered on reaching a specified location.

Parameters

- **name** (*str*) – The name of the location to be reached.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_message_event(*msgs: List[str], reward=1, repeatable=False, terminal_required=True, terminal_sufficient=False*)
Add event which is triggered when any of the given messages are seen.

Parameters

- **msgs** (*List[str]*) – The name of the monster to be killed.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_positional_event(*place_name: str, action_name: str, reward=1, repeatable=False, terminal_required=True, terminal_sufficient=False*)
Add event which is triggered on taking a given action at a given place.

Parameters

- **place_name** (*str*) – The name of the place to trigger the event.
- **action_name** (*int*) – The name of the action to trigger the event.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_wear_event(*name: str*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)
Add event which is triggered when a specific armor is worn.

Parameters

- **name** (*str*) – The name of the armor to be worn.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_wield_event(*name: str*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)
Add event which is triggered when a specific weapon is wielded.

Parameters

- **name** (*str*) – The name of the weapon to be wielded.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

check_episode_end_call(*env*, *previous_observation*, *action*, *observation*) → bool
Check if the task has ended, and accumulate any reward from the transition in `self._reward`.

Parameters

- **env** (MiniHack) – The MiniHack environment in question.
- **previous_observation** (*tuple*) – The previous state observation.
- **action** (*int*) – The action taken.
- **observation** (*tuple*) – The current observation.

Returns Boolean whether the episode has ended.

Return type bool

collect_reward() → float

Return reward calculated and accumulated in check_episode_end_call, and then reset it.

Returns The reward.

Return type flaot

reset()

Reset all events, to be called when a new episode occurs.

9.15.1 Submodules

minihack.base module

class minihack.base.**MiniHack**(**args: Any*, ***kwargs: Any*)

Bases: `nle.env.tasks.`

MiniHack base class.

All MiniHack environments are derived from this class, which itself is derived from NLE base class.

Note that this class itself is not used for creating new environment instances. Instead, `MiniHackNavigation` and `MiniHackSkill` provide a more convenient interface for doing this, both of which are directly derived from MiniHack for specific types of environments.

__init__(**args*, *des_file: str*, *reward_win=1*, *reward_lose=0*, *obs_crop_h=9*, *obs_crop_w=9*, *obs_crop_pad=0*, *reward_manager=None*, *use_wiki=False*, *autopickup=True*, *pet=False*, *observation_keys=['glyphs', 'chars', 'colors', 'specials', 'glyphs_crop', 'chars_crop', 'colors_crop', 'specials_crop', 'blstats', 'message']*, *seeds=None*, *include_see_actions=True*, *include_alignment_blstats=True*, ***kwargs*)

Constructs a new MiniHack environment.

Parameters

- **des_file** (*str*) – The description file for the environment.
- **reward_win** (*float*) – The reward received upon successfully completing an episode. Defaults to 1.
- **reward_lose** (*float*) – The reward received upon death or aborting. Defaults to 1.
- **obs_crop_h** (*int*) – The height of agent-centred cropped observation. Defaults to 9.
- **obs_crop_w** (*int*) – The width of agent-centred cropped observation. Defaults to 9.
- **obs_crop_pad** (*int*) – The padding for agent-centred cropped observation. Defaults to 0.
- **reward_manager** (*RewardManager or None*) – The reward manager that describes the custom reward function of the agent. If None, the goal of the agent is to reach the stair down. Defaults to None.
- **use_wiki** (*bool*) – Whether to use the NetHack wiki. Defaults to False.
- **autopickup** (*bool*) – Turning autopickup on or off. Defaults to True.
- **pet** (*bool*) – Whether to include the pet. Defaults to False.

- **observation_keys** (*list*) – The keys of observations returned after every timestep by the environment as a dictionary. Defaults to `minihack.base.MH_DEFAULT_OBS_KEYS`.
- **seeds** (*list or None*) – A list of integers used as level seeds for sampling episodes. The reset()` function samples a seed from this list uniformly at random and uses it for setting the level. When the `sample_seed` argument of the reset function is set to False, a random level will not be sampled from this list during environment resetting. If None, the entire level distribution is used. Defaults to None.
- **penalty_mode** (*str*) – The name of the mode for calculating the time step penalty. Can be `constant`, `exp`, `square`, `linear`, or `always`. Defaults to `constant`. Inherited from *NetHackScore*.
- **penalty_step** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.01. Inherited from *NetHackScore*.
- **penalty_time** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.0. Inherited from *NetHackScore*.
- **save_ttyrec_every** (*int*) – Integer, if 0, no ttyrecs (game recordings) will be saved. Otherwise, save a ttyrec every Nth episode. Defaults to 0. Inherited from *NLE*.
- **savedir** (*str or None*) – Path to save ttyrecs (game recordings) into, if save_ttyrec_every is nonzero. If nonempty string, interpreted as a path to a new or existing directory. If "" (empty string) or None, NLE choses a unique directory name. Defaults to None. Inherited from *NLE*.
- **character** (*str*) – Name of character. Defaults to "mon-hum-neu-mal". Interited from *NLE*.
- **max_episode_steps** (*int*) – maximum amount of steps allowed before the game is forcefully quit. In such cases, `info["end_status"]` ill be equal to `StepStatus.ABORTED`. Defaults to 200. Inherited from *NLE*.
- **actions** (*list*) – list of actions. If None, the full action space will be used, i.e. `nle.nethack.ACTIONS`. Defaults to *MH_FULL_ACTIONS*. Inherited from *NLE*.
- **wizard** (*bool*) – activate wizard mode. Defaults to False. Inherited from *NLE*.
- **allow_all_yn_questions** (*bool*) – If set to True, no y/n questions in step() are declined. If set to False, only elements of SKIP_EXCEPTIONS are not declined. Defaults to True. Inherited from *NLE*.
- **allow_all_modes** (*bool*) – If set to True, do not decline menus, text input or auto 'MORE'. If set to False, only skip click through 'MORE' on death. Defaults to False. Inherited from *NLE*.
- **spawn_monsters** (*bool*) – If False, disables normal NetHack behavior to randomly create monsters. Defaults to False. Inherited from *NLE*.
- **include_see_actions** (*bool*) – If True, the agent's action space includes the additional *NLE* actions introduced in the 0.8.1 release. Has no effect when the *actions* parameter is specified. Defaults to True.
- **include_alignment_blstats** (*bool*) – If True, the agent's observation space includes the alignment information in the blstats. This is introduced in *NLE* 0.9.0 release. Defaults to True.

`get_neighbor_descriptions`(*observation=None*)

Returns the descriptions of nine neighboring grids around the agent.

`get_neighbor_wiki_pages`(*observation=None*)
Returns the page contents of the neighboring objects from NetHack wiki.

`get_object_direction`(*name*, *observation=None*)
Find the game direction of the (first) object in the neighboring nine tiles that contains the given name in its description.

Parameters

- **name** (*str*) – Name of the object.
- **observation** (*dict*) – Agent observation.

Returns The index of the direction. None if not found.

Return type int

`get_screen_description`(*x*, *y*, *observation=None*)
Returns the description of the screen on (x,y) coordinates.

`get_screen_wiki_page`(*x*, *y*, *observation=None*)
Returns the wiki page matching the object on (x,y) coordinates.

`key_in_inventory`(*name*)
Returns key of the given object in the inventory.

Parameters **name** (*str*) – Name of the object.

Returns the key of the first item in the inventory that includes the argument name as a substring. Returns None if not found.

Return type str

`reset`(**args*, *sample_seed=True*, ***kwargs*)

`screen_contains`(*name*, *observation=None*)
Whether an object with the given name is visible on the screen, i.e. included in the screen descriptions of the observation dictionary.

Parameters

- **name** (*str*) – Name of the object or monster.
- **observation** (*dict*) – Agent observation.

Returns True if the name is contained on the screen, False otherwise.

Return type bool

`step`(*action: int*)

`update`(*des_file*)
Update the current environment by replacing its description file.

minihack.level_generator module

class minihack.level_generator.**LevelGenerator**(*map=None*, *w=8*, *h=8*, *fill='.'*, *lit=True*, *flags=('hardfloor')*, *solidfill=' '*)

Bases: object

LevelGenerator provides a convenient Python interface for quickly writing description files for MiniHack. The LevelGenerator class can be used to create MAZE-type levels with specified heights and widths, and can then fill those levels with objects, monsters and terrain, and specify the start point of the level.

Parameters

- **map** (*str or None*) – The description of the map block of the environment. If None, the map will have a rectangle layout with the given height and width. Defaults to None.
- **w** (*int*) – The width of map. Only used when *map=None*. Defaults to 8.
- **h** (*int*) – The height of map. Only used when *map=None*. Defaults to 8.
- **fill** (*str*) – A character describing the environment feature that fills the map. Only used when map=None. Defaults to ".", which corresponds to floor.
- **lit** (*bool*) – Whether the layout is lit or not. This affects the observations the agent will receive. If an area is not lit, the agent can only see directly adjacent grids. Defaults to True.
- **flags** (*tuple*) – Flags of the environment. For the full list, see https://nethackwiki.com/wiki/Des-file_format#FLAGS. Defaults to ("hardfloor",).
- **solidfill** (*str*) – A character describing the environment feature used for filling solid / unspecified parts of the map. Defaults to " ", which corresponds to solid wall.

__init__(*map=None*, *w=8*, *h=8*, *fill='.'*, *lit=True*, *flags=('hardfloor')*, *solidfill=' '*)

Initialize self. See help(type(self)) for accurate signature.

add_altar(*place=None*, *align='random'*, *type='random'*)

Add an altar.

Parameters

- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.
- **align** (*str*) – The alignment. Possible values are "noalign", "law", "neutral", "chaos", "coaligned", "noncoaligned", and "random". Defaults to "random".
- **type** (*str*) – The type of the altar. Possible values are "sanctum", "shrine", "altar", and "random". Defaults to random.

add_boulder(*place=None*)

Add a boulder to the floor.

Parameters

- **amount** (*int*) – The amount of gold.
- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

add_door(*state*, *place=None*)

Add a door.

Parameters

- **state** (*str*) – The state of the door. Possible values are "locked", "closed", "open", "nodoor", and "random". Defaults to "random".
- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

add_fountain(*place=None*)

Add a fountain.

Parameters **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

add_goal_pos(*place=None*)

Add a goal at the given place. Same as *add_stair_down*.

add_gold(*amount*, *place=None*)

Add gold on the floor.

Parameters

- **amount** (*int*) – The amount of gold.
- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

add_line(*str*)

Add a custom string to the buttom of the description file.

Parameters **str** (*str*) – The string to be concatenated to the des-file.

add_mazewalk(*coord=None*, *dir='east'*)

Creates a random maze, starting from the given coordinate.

Mazewalk turns map grids with solid stone into floor. From the starting position, it checks the mapgrid in the direction given, and if it's solid stone, it will move there, and turn that place into floor. Then it will choose a random direction, jump over the nearest mapgrid in that direction, and check the next mapgrid for solid stone. If there is solid stone, mazewalk will move that direction, changing that place and the intervening mapgrid to floor. Normally the generated maze will not have any loops.

Pointing mazewalk at that will create a small maze of trees, but unless the map (at the place where it's put into the level) is surrounded by something else than solid stone, mazewalk will get out of that MAP. Substituting floor characters for some of the trees "in the maze" will make loops in the maze, which are not otherwise possible. Substituting floor characters for some of the trees at the edges of the map will make maze entrances and exits at those places.

For more details see https://nethackwiki.com/wiki/Des-file_format#MAZEWALK.

Parameters **coord** – A tuple with length two representing the (x, y) coordinates. If None is passed, the middle point of the map is selected. Defaults to None.

add_monster(*name='random'*, *symbol=None*, *place=None*, *args=()*)

Add a monster to the map.

Parameters

- **name** (*str*) – The name of the monster. Defaults to random.
- **symbol** (*str or None*) – The symbol of the monster. The symbol should correspond to the family of the specified mosnter. For example, "d" symbol corresponds to canine

monsters, so the name of the object should also correspond to canines (e.g. jackal). Not used when name is "random". Defaults to None.

- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.
- **args** (*tuple*) – Additional monster arguments, e.g. "hostile" or "peaceful", "asleep" or "awake", etc. For more details, see https://nethackwiki.com/wiki/Des-file_format#MONSTER.

add_object(*name='random'*, *symbol='%'*, *place=None*, *cursestate=None*)
Add an object to the map.

Parameters

- **name** (*str*) – The name of the object. Defaults to random.
- **symbol** (*str*) – The symbol of the object. The symbol should correspond to the given object name. For example, "%" symbol corresponds to comestibles, so the name of the object should also correspond to commestibles (e.g. apple). Not used when name is "random". Defaults to "%".
- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.
- **cursetstate** (*str or None*) – The cursed state of the object. Can be "blessed", "uncursed", "cursed" or "random". Defaults to None (not used).

add_object_area(*area_name*, *name='random'*, *symbol='%'*, *cursestate=None*)
Add an object in an area of the map defined by *area_name* variable. See `add_object` for more details.

add_sink(*place=None*)
Add a sink.

Parameters **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

add_stair_down(*place=None*)
Add a stair down at the given place.

Parameters **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

add_terrain(*coord*, *flag*, *in_footer=False*)
Add terrain features to the map.

Parameters

- **coord** (*tuple*) – A tuple with length two representing the (x, y) coordinates.
- **flag** (*str*) – The flag corresponding to the desired terrain feature. Should belong to minihack.level_generator.MAP_CHARS. For more details, see https://nethackwiki.com/wiki/Des-file_format#Map_characters
- **in_footer** (*bool*) – Whether to define the terrain feature as an additional line in the description file (True) or directly modify the map block with the given flag (False). Defaults to False.

`add_trap`(*name='teleport'*, *place=None*)
Add a trap.

Parameters

- **name** (*str*) – The name of the trap. For possible values, see *minihack.level_generator.TRAP_NAMES*. Defaults to "teleport".
- **place** (*None, tuple or str*) – The place of the added object. If None, the location is selected randomly. Tuple values are used for providing exact (x, y) coordinates. String values are copied to des-file as is. Defaults to None.

`fill_terrain`(*type*, *flag*, *x1*, *y1*, *x2*, *y2*)
Fill the areas between (x1, y1) and (x2, y2) with the given dungeon feature:

Parameters

- **type** (*str*) – The type of filling. "rect" indicates an unfilled rectangle, containing just the edges and none of the interior points. "fillrect" denotes filled rectangle containing the edges and all of the interior points. "line" is used for a straight line drawn from one pair of coordinates to the other using Bresenham's line algorithm.
- **flag** (*str*) – The flag corresponding to the desired terrain feature. Should belong to minihack.level_generator.MAP_CHARS. For more details, see https://nethackwiki.com/wiki/Des-file_format#Map_characters
- **x1** (*int*) – x coordinate of point 1.
- **y1** (*int*) – y coordinate of point 1.
- **x2** (*int*) – x coordinate of point 2.
- **y2** (*int*) – y coordinate of point 2.

`get_des`()
Returns the description file.

Returns the description file as a string.

Return type str

`get_map_array`()
Returns the map as a two-dimensional numpy array.

`get_map_str`()
Returns the map as a string.

`init_map`(*map=None*, *x=8*, *y=8*, *fill='.'*)
Initialise the map block of the des-file.

`set_area_variable`(*var_name*, *type*, *x1*, *y1*, *x2*, *y2*)
Set a variable representing an area on the map.

Parameters

- **var_name** (*str*) – The name of the variable.
- **type** (*str*) – The type of filling. "rect" indicates an unfilled rectangle, containing just the edges and none of the interior points. "fillrect" denotes filled rectangle containing the edges and all of the interior points. "line" is used for a straight line drawn from one pair of coordinates to the other using Bresenham's line algorithm.
- **x1** (*int*) – x coordinate of point 1.
- **y1** (*int*) – y coordinate of point 1.

- **x2** (*int*) – x coordinate of point 2.
- **y2** (*int*) – y coordinate of point 2.

set_start_pos(*coord*)
Set the starting position of the agent.

Parameters **coord** (*tuple*) – A tuple with length two representing the (x, y) coordinates.

set_start_rect(*p1*, *p2*)
Set the starting position of the agent.

Parameters **coord** (*tuple*) – A tuple with length two representing the (x, y) coordinates.

wallify()
Wallify the map. Turns walls completely surrounded by other walls into solid stone ' '.

minihack.navigation module

class minihack.navigation.**MiniHackNavigation**(**args: Any*, ***kwargs: Any*)
Bases: nle.env.tasks.

The base class for MiniHack Navigation tasks.

Navigation tasks have the following characteristics:

- Restricted action space: By default, the agent can only move towards eight compass directions.
- Yes/No questions, as well as menu-selection actions are disabled by default.
- The character is set to chaotic human male rogue.
- Auto-pick is enabled by default.
- Maximum episode limit defaults to 100 (can be overriden via the *max_episode_steps* argument)
- The default goal is to reach the stair down. This can be changed using a reward manager.

__init__(**args*, *des_file: Optional[str] = None*, ***kwargs*)
Constructs a new MiniHack environment.

Parameters

- **des_file** (*str*) – The description file for the environment.
- **reward_win** (*float*) – The reward received upon successfully completing an episode. Defaults to 1.
- **reward_lose** (*float*) – The reward received upon death or aborting. Defaults to 1.
- **obs_crop_h** (*int*) – The height of agent-centred cropped observation. Defaults to 9.
- **obs_crop_w** (*int*) – The width of agent-centred cropped observation. Defaults to 9.
- **obs_crop_pad** (*int*) – The padding for agent-centred cropped observation. Defaults to 0.
- **reward_manager** (RewardManager *or None*) – The reward manager that describes the custom reward function of the agent. If None, the goal of the agent is to reach the stair down. Defaults to None.
- **use_wiki** (*bool*) – Whether to use the NetHack wiki. Defaults to False.
- **autopickup** (*bool*) – Turning autopickup on or off. Defaults to True.
- **pet** (*bool*) – Whether to include the pet. Defaults to False.

- **observation_keys** (*list*) – The keys of observations returned after every timestep by the environment as a dictionary. Defaults to `minihack.base.MH_DEFAULT_OBS_KEYS`.
- **seeds** (*list or None*) – A list of integers used as level seeds for sampling episodes. The reset()` function samples a seed from this list uniformly at random and uses it for setting the level. When the `sample_seed` argument of the reset function is set to False, a random level will not be sampled from this list during environment resetting. If None, the entire level distribution is used. Defaults to None.
- **penalty_mode** (*str*) – The name of the mode for calculating the time step penalty. Can be `constant`, `exp`, `square`, `linear`, or `always`. Defaults to `constant`. Inherited from *NetHackScore*.
- **penalty_step** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.01. Inherited from *NetHackScore*.
- **penalty_time** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.0. Inherited from *NetHackScore*.
- **save_ttyrec_every** (*int*) – Integer, if 0, no ttyrecs (game recordings) will be saved. Otherwise, save a ttyrec every Nth episode. Defaults to 0. Inherited from *NLE*.
- **savedir** (*str or None*) – Path to save ttyrecs (game recordings) into, if save_ttyrec_every is nonzero. If nonempty string, interpreted as a path to a new or existing directory. If "" (empty string) or None, NLE choses a unique directory name. Defaults to None. Inherited from *NLE*.
- **character** (*str*) – Name of character. Defaults to "mon-hum-neu-mal". Interited from *NLE*.
- **max_episode_steps** (*int*) – maximum amount of steps allowed before the game is forcefully quit. In such cases, `info["end_status"]` ill be equal to `StepStatus.ABORTED`. Defaults to 200. Inherited from *NLE*.
- **actions** (*list*) – list of actions. If None, the full action space will be used, i.e. `nle.nethack.ACTIONS`. Defaults to *MH_FULL_ACTIONS*. Inherited from *NLE*.
- **wizard** (*bool*) – activate wizard mode. Defaults to False. Inherited from *NLE*.
- **allow_all_yn_questions** (*bool*) – If set to True, no y/n questions in step() are declined. If set to False, only elements of SKIP_EXCEPTIONS are not declined. Defaults to True. Inherited from *NLE*.
- **allow_all_modes** (*bool*) – If set to True, do not decline menus, text input or auto 'MORE'. If set to False, only skip click through 'MORE' on death. Defaults to False. Inherited from *NLE*.
- **spawn_monsters** (*bool*) – If False, disables normal NetHack behavior to randomly create monsters. Defaults to False. Inherited from *NLE*.
- **include_see_actions** (*bool*) – If True, the agent's action space includes the additional *NLE* actions introduced in the 0.8.1 release. Has no effect when the *actions* parameter is specified. Defaults to True.
- **include_alignment_blstats** (*bool*) – If True, the agent's observation space includes the alignment information in the blstats. This is introduced in *NLE* 0.9.0 release. Defaults to True.

minihack.reward_manager module

class minihack.reward_manager.**AbstractRewardManager**

Bases: abc.ABC

This is the abstract base class for the RewardManager that is used for defining custom reward functions.

__init__()

Initialize self. See help(type(self)) for accurate signature.

abstract check_episode_end_call(*env*, *previous_observation*, *action*, *observation*) → bool

Check if the task has ended, and accumulate any reward from the transition in self._reward.

Parameters

- **env** (MiniHack) – The MiniHack environment in question.
- **previous_observation** (*tuple*) – The previous state observation.
- **action** (*int*) – The action taken.
- **observation** (*tuple*) – The current observation.

Returns Boolean whether the episode has ended.

Return type bool

abstract collect_reward() → float

Return reward calculated and accumulated in check_episode_end_call, and then reset it.

Returns The reward.

Return type flaot

abstract reset() → None

Reset all events, to be called when a new episode occurs.

class minihack.reward_manager.**CoordEvent**(**args*, *coordinates: Tuple[int, int]*)

Bases: *minihack.reward_manager.Event*

An event which occurs when reaching certain coordinates.

__init__(**args*, *coordinates: Tuple[int, int]*)

Initialise the Event.

Parameters

- **coordinates** (*tuple*) – The coordinates to reach for the event.
- **reward** (*float*) – The reward for the event occuring
- **repeatable** (*bool*) – Whether the event can occur repeated (i.e. if the reward can be collected repeatedly
- **terminal_required** (*bool*) – Whether this event is required for the episode to terminate.
- **terminal_sufficient** (*bool*) – Whether this event causes the episode to terminate on its own.

check(*env*, *previous_observation*, *action*, *observation*) → float

Check whether the environment is in the state such that this event has occured.

Parameters

- **env** (MiniHack) – The MiniHack environment in question.
- **previous_observation** (*tuple*) – The previous state observation.

- **action** (*int*) – The action taken.
- **observation** (*tuple*) – The current observation.

Returns The reward.

Return type float

class minihack.reward_manager.**Event**(*reward: float*, *repeatable: bool*, *terminal_required: bool*, *terminal_sufficient: bool*)

Bases: abc.ABC

An event which can occur in a MiniHack episode.

This is the base class of all other events.

__init__(*reward: float*, *repeatable: bool*, *terminal_required: bool*, *terminal_sufficient: bool*)

Initialise the Event.

Parameters

- **reward** (*float*) – The reward for the event occuring
- **repeatable** (*bool*) – Whether the event can occur repeated (i.e. if the reward can be collected repeatedly
- **terminal_required** (*bool*) – Whether this event is required for the episode to terminate.
- **terminal_sufficient** (*bool*) – Whether this event causes the episode to terminate on its own.

abstract check(*env*, *previous_observation*, *action*, *observation*) → float

Check whether the environment is in the state such that this event has occured.

Parameters

- **env** (MiniHack) – The MiniHack environment in question.
- **previous_observation** (*tuple*) – The previous state observation.
- **action** (*int*) – The action taken.
- **observation** (*tuple*) – The current observation.

Returns The reward.

Return type float

reset()

Reset the event, if there is any state necessary.

class minihack.reward_manager.**EventType**(*value*)

Bases: enum.IntEnum

An enumeration.

COORD = 2

LOC = 3

LOC_ACTION = 1

MESSAGE = 0

class minihack.reward_manager.**GroupedRewardManager**

Bases: *minihack.reward_manager.AbstractRewardManager*

Operates as a collection of reward managers.

The rewards from each reward manager are summed, and termination can be specified by `terminal_sufficient` and `terminal_required` on each reward manager.

Given this can be nested arbitrarily deeply (as each reward manager could itself be a GroupedRewardManager), this enables complex specification of groups of rewards.

`__init__`()
Initialize self. See help(type(self)) for accurate signature.

`add_reward_manager`(*reward_manager: minihack.reward_manager.AbstractRewardManager*, *terminal_required: bool*, *terminal_sufficient: bool*) → None
Add a new reward manager, with `terminal_sufficient` and `terminal_required` acting as for individual events.

Parameters

- **`reward_manager`** (`RewardManager`) – The reward manager to be added.
- **`terminal_required`** (*`bool`*) – Whether this reward manager terminating is required for the episode to terminate.
- **`terminal_sufficient`** – Whether this reward manager terminating is sufficient for the episode to terminate.

`check_episode_end_call`(*env*, *previous_observation*, *action*, *observation*) → bool
Check if the task has ended, and accumulate any reward from the transition in `self._reward`.

Parameters

- **`env`** (`MiniHack`) – The MiniHack environment in question.
- **`previous_observation`** (*`tuple`*) – The previous state observation.
- **`action`** (*`int`*) – The action taken.
- **`observation`** (*`tuple`*) – The current observation.

Returns Boolean whether the episode has ended.

Return type bool

`collect_reward`()
Return reward calculated and accumulated in check_episode_end_call, and then reset it.

Returns The reward.

Return type flaot

`reset`()
Reset all events, to be called when a new episode occurs.

class `minihack.reward_manager.`**`LocActionEvent`**(**args*, *loc: str*, *action: nle.nethack.Command*)
Bases: *`minihack.reward_manager.Event`*

An event which checks whether an action is performed at a specified location.

`__init__`(**args*, *loc: str*, *action: nle.nethack.Command*)
Initialise the Event.

Parameters

- **`loc`** (*`str`*) – The name of the location to reach.
- **`action`** (*`int`*) – The action to perform.
- **`reward`** (*`float`*) – The reward for the event occuring

- **repeatable** (*bool*) – Whether the event can occur repeated (i.e. if the reward can be collected repeatedly
- **terminal_required** (*bool*) – Whether this event is required for the episode to terminate.
- **terminal_sufficient** (*bool*) – Whether this event causes the episode to terminate on its own.

check(*env*, *previous_observation*, *action*, *observation*) → float
Check whether the environment is in the state such that this event has occured.

Parameters

- **env** (MiniHack) – The MiniHack environment in question.
- **previous_observation** (*tuple*) – The previous state observation.
- **action** (*int*) – The action taken.
- **observation** (*tuple*) – The current observation.

Returns The reward.

Return type float

reset()
Reset the event, if there is any state necessary.

class minihack.reward_manager.**LocEvent**(**args*, *loc: str*)
Bases: *minihack.reward_manager.Event*

An event which checks whether a specified location is reached.

__init__(**args*, *loc: str*)
Initialise the Event.

Parameters

- **reward** (*float*) – The reward for the event occuring
- **repeatable** (*bool*) – Whether the event can occur repeated (i.e. if the reward can be collected repeatedly
- **terminal_required** (*bool*) – Whether this event is required for the episode to terminate.
- **terminal_sufficient** (*bool*) – Whether this event causes the episode to terminate on its own.

check(*env*, *previous_observation*, *action*, *observation*) → float
Check whether the environment is in the state such that this event has occured.

Parameters

- **env** (MiniHack) – The MiniHack environment in question.
- **previous_observation** (*tuple*) – The previous state observation.
- **action** (*int*) – The action taken.
- **observation** (*tuple*) – The current observation.

Returns The reward.

Return type float

class minihack.reward_manager.**MessageEvent**(**args*, *messages: List[str]*)
Bases: *minihack.reward_manager.Event*

An event which occurs when any of the *messages* appear.

`__init__`(**args*, *messages: List[str]*)
Initialise the Event.

Parameters

- **messages** (*list*) – The messages to be seen to trigger the event.
- **reward** (*float*) – The reward for the event occuring
- **repeatable** (*bool*) – Whether the event can occur repeated (i.e. if the reward can be collected repeatedly
- **terminal_required** (*bool*) – Whether this event is required for the episode to terminate.
- **terminal_sufficient** (*bool*) – Whether this event causes the episode to terminate on its own.

`check`(*env*, *previous_observation*, *action*, *observation*) → float
Check whether the environment is in the state such that this event has occured.

Parameters

- **env** (MiniHack) – The MiniHack environment in question.
- **previous_observation** (*tuple*) – The previous state observation.
- **action** (*int*) – The action taken.
- **observation** (*tuple*) – The current observation.

Returns The reward.

Return type float

class `minihack.reward_manager.`**`RewardManager`**
Bases: *minihack.reward_manager.AbstractRewardManager*

This class is used for managing rewards, events and termination for MiniHack tasks.

Some notes on the ordering or calls in the MiniHack/NetHack base class:

- `step(action)` is called on the environment
- Within `step`, first a copy of the last observation is made, and then the underlying NetHack game is stepped
- Then `_is_episode_end(observation)` is called to check whether this the episode has ended (and this is overridden if we've gone over our max_steps, or the underlying NetHack game says we're done (i.e. we died)
- Then `_reward_fn(last_observation, observation)` is called to calculate the reward at this time-step
- if `end_status` tells us the game is done, we quit the game
- then `step` returns the observation, calculated reward, done, and some

statistics.

All this means that we need to check whether an observation is terminal in `_is_episode_end` before we're calculating the reward function.

The call of `_is_episode_end` in `MiniHack` will call `check_episode_end_call` in this class, which checks for termination and accumulates any reward, which is returned and zeroed in `collect_reward`.

`__init__`()
Initialize self. See help(type(self)) for accurate signature.

add_amulet_event(*reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)
Add event which is triggered when an amulet is worn.

Parameters

- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_coordinate_event(*coordinates: Tuple[int, int]*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)
Add event which is triggered on when reaching the specified coordinates.

Parameters

- **coordinates** (*Tuple[int, int]*) – The coordinates to be reached (tuple of ints).
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_custom_reward_fn(*reward_fn: Callable[[MiniHack, Any, int, Any], float]*) → None
Add a custom reward function which is called every after step to calculate reward.

The function should be a callable which takes the environment, previous observation, action and current observation and returns a float reward.

Parameters **reward_fn** (*Callable[[MiniHack, Any, int, Any], float]*) – A reward function which takes an environment, previous observation, action, next observation and returns a reward.

add_eat_event(*name: str*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)
Add an event which is triggered when *name* is eaten.

Parameters

- **name** (*str*) – The name of the object being eaten.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_event(*event: minihack.reward_manager.Event*)
Add an event to be managed by the reward manager.

Parameters **event** (Event) – The event to be added.

add_kill_event(*name: str*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)
Add event which is triggered when a specified monster is killed.

Parameters

- **name** (*str*) – The name of the monster to be killed.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_location_event(*location: str*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)
Add event which is triggered on reaching a specified location.

Parameters

- **name** (*str*) – The name of the location to be reached.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_message_event(*msgs: List[str]*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)
Add event which is triggered when any of the given messages are seen.

Parameters

- **msgs** (*List[str]*) – The name of the monster to be killed.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_positional_event(*place_name: str*, *action_name: str*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)
Add event which is triggered on taking a given action at a given place.

Parameters

- **place_name** (*str*) – The name of the place to trigger the event.
- **action_name** (*int*) – The name of the action to trigger the event.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_wear_event(*name: str*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)

Add event which is triggered when a specific armor is worn.

Parameters

- **name** (*str*) – The name of the armor to be worn.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

add_wield_event(*name: str*, *reward=1*, *repeatable=False*, *terminal_required=True*, *terminal_sufficient=False*)

Add event which is triggered when a specific weapon is wielded.

Parameters

- **name** (*str*) – The name of the weapon to be wielded.
- **reward** (*float*) – The reward for this event. Defaults to 1.
- **repeatable** (*bool*) – Whether this event can be triggered multiple times. Defaults to False.
- **terminal_required** (*bool*) – Whether this event is required for termination. Defaults to True.
- **terminal_sufficient** (*bool*) – Whether this event is sufficient for termination. Defaults to False.

check_episode_end_call(*env*, *previous_observation*, *action*, *observation*) → bool

Check if the task has ended, and accumulate any reward from the transition in `self._reward`.

Parameters

- **env** (MiniHack) – The MiniHack environment in question.
- **previous_observation** (*tuple*) – The previous state observation.
- **action** (*int*) – The action taken.
- **observation** (*tuple*) – The current observation.

Returns Boolean whether the episode has ended.

Return type bool

collect_reward() → float

Return reward calculated and accumulated in check_episode_end_call, and then reset it.

Returns The reward.

Return type flaot

reset()

Reset all events, to be called when a new episode occurs.

class minihack.reward_manager.**SequentialRewardManager**

Bases: *minihack.reward_manager.RewardManager*

A reward manager that ignores `terminal_required` and `terminal_sufficient`, and just require every event is completed in the order it is added to the reward manager.

__init__()

Initialize self. See help(type(self)) for accurate signature.

check_episode_end_call(*env*, *previous_observation*, *action*, *observation*)

Check if the task has ended, and accumulate any reward from the transition in `self._reward`.

Parameters

- **env** (MiniHack) – The MiniHack environment in question.
- **previous_observation** (*tuple*) – The previous state observation.
- **action** (*int*) – The action taken.
- **observation** (*tuple*) – The current observation.

Returns Boolean whether the episode has ended.

Return type bool

minihack.skills module

class minihack.skills.**MiniHackSkill**(**args: Any*, ***kwargs: Any*)

Bases: `nle.env.tasks.`

The base class for MiniHack Skill Acquisition tasks.

Navigation tasks have the following characteristics:

- The full action space is used.
- Yes/No questions are enabled, but the menu-selection actions are disabled by default.
- The character is set to a neutral human male caveman.
- Maximum episode limit defaults to 250 (can be overriden via the *max_episode_steps* argument)
- The default goal is to reach the stair down. This can be changed using a reward manager.
- Auto-pick is disabled by default.
- Inventory strings and corresponding letter are also included as part of the agent observations.

__init__(**args*, *des_file*, ***kwargs*)

Constructs a new MiniHack environment.

Parameters

- **des_file** (*str*) – The description file for the environment.
- **reward_win** (*float*) – The reward received upon successfully completing an episode. Defaults to 1.
- **reward_lose** (*float*) – The reward received upon death or aborting. Defaults to 1.
- **obs_crop_h** (*int*) – The height of agent-centred cropped observation. Defaults to 9.
- **obs_crop_w** (*int*) – The width of agent-centred cropped observation. Defaults to 9.
- **obs_crop_pad** (*int*) – The padding for agent-centred cropped observation. Defaults to 0.
- **reward_manager** (*RewardManager or None*) – The reward manager that describes the custom reward function of the agent. If None, the goal of the agent is to reach the stair down. Defaults to None.
- **use_wiki** (*bool*) – Whether to use the NetHack wiki. Defaults to False.
- **autopickup** (*bool*) – Turning autopickup on or off. Defaults to True.
- **pet** (*bool*) – Whether to include the pet. Defaults to False.
- **observation_keys** (*list*) – The keys of observations returned after every timestep by the environment as a dictionary. Defaults to `minihack.base.MH_DEFAULT_OBS_KEYS`.
- **seeds** (*list or None*) – A list of integers used as level seeds for sampling episodes. The reset()` function samples a seed from this list uniformly at random and uses it for setting the level. When the `sample_seed` argument of the reset function is set to False, a random level will not be sampled from this list during environment resetting. If None, the entire level distribution is used. Defaults to None.
- **penalty_mode** (*str*) – The name of the mode for calculating the time step penalty. Can be `constant`, `exp`, `square`, `linear`, or `always`. Defaults to `constant`. Inherited from *NetHackScore*.
- **penalty_step** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.01. Inherited from *NetHackScore*.
- **penalty_time** (*float*) – A constant applied to amount of frozen steps. Defaults to -0.0. Inherited from *NetHackScore*.
- **save_ttyrec_every** (*int*) – Integer, if 0, no ttyrecs (game recordings) will be saved. Otherwise, save a ttyrec every Nth episode. Defaults to 0. Inherited from *NLE*.
- **savedir** (*str or None*) – Path to save ttyrecs (game recordings) into, if save_ttyrec_every is nonzero. If nonempty string, interpreted as a path to a new or existing directory. If "" (empty string) or None, NLE choses a unique directory name. Defaults to None. Inherited from *NLE*.
- **character** (*str*) – Name of character. Defaults to "mon-hum-neu-mal". Interited from *NLE*.
- **max_episode_steps** (*int*) – maximum amount of steps allowed before the game is forcefully quit. In such cases, `info["end_status"]` ill be equal to `StepStatus.ABORTED`. Defaults to 200. Inherited from *NLE*.
- **actions** (*list*) – list of actions. If None, the full action space will be used, i.e. `nle.nethack.ACTIONS`. Defaults to *MH_FULL_ACTIONS*. Inherited from *NLE*.
- **wizard** (*bool*) – activate wizard mode. Defaults to False. Inherited from *NLE*.

- **allow_all_yn_questions** (*bool*) – If set to True, no y/n questions in step() are declined. If set to False, only elements of SKIP_EXCEPTIONS are not declined. Defaults to True. Inherited from *NLE*.
- **allow_all_modes** (*bool*) – If set to True, do not decline menus, text input or auto 'MORE'. If set to False, only skip click through 'MORE' on death. Defaults to False. Inherited from *NLE*.
- **spawn_monsters** (*bool*) – If False, disables normal NetHack behavior to randomly create monsters. Defaults to False. Inherited from *NLE*.
- **include_see_actions** (*bool*) – If True, the agent's action space includes the additional *NLE* actions introduced in the 0.8.1 release. Has no effect when the *actions* parameter is specified. Defaults to True.
- **include_alignment_blstats** (*bool*) – If True, the agent's observation space includes the alignment information in the blstats. This is introduced in *NLE* 0.9.0 release. Defaults to True.

minihack.wiki module

class minihack.wiki.**NetHackWiki**(*raw_wiki_file_name: str*, *processed_wiki_file_name: str*, *save_processed_json: bool = True*, *ignore_inpage_anchors: bool = True*, *preprocess_input: bool = True*, *exceptions: Optional[tuple] = None*)

Bases: `object`

A class representing Nethack Wiki Data - pages and links between them.

Parameters

- **raw_wiki_file_name** (*str*) – The path to the raw file of NetHack wiki. The raw file can be downloaded using the *get_nhwiki_data.sh* script located in *minihack/scripts*.
- **processed_wiki_file_name** (*str*) – The path to the processed file of NetHack wiki. The processing is performed in the *__init__* function of this classed.
- **save_processed_json** (*bool*) – Whether to save the processed json file of the wiki. Only considered when a raw wiki file is passed. Defaults to True.
- **ignore_inpage_anchors** (*bool*) – Whether to ingnore in-page anchors. Defaults to True.
- **preprocess_input** (*bool*) – Whether to perform a preprocessing on wiki data. Defaults to True.
- **exceptions** (*Tuple[str] or None*) – Name of entities in screen descriptions that are ingored. If None, there are no exceptions. Defaults to None.

__init__(*raw_wiki_file_name: str*, *processed_wiki_file_name: str*, *save_processed_json: bool = True*, *ignore_inpage_anchors: bool = True*, *preprocess_input: bool = True*, *exceptions: Optional[tuple] = None*) → None

Initialize self. See help(type(self)) for accurate signature.

get_page_data(*page: str*) → dict

Get the data of a page.

Parameters **page** (*str*) – The page name.

Returns The page data as a dict.

Return type dict

get_page_text(*page: str*) → str

Get the text of a page.

Parameters **page** (*str*) – The page name.

Returns The text of the page.

Return type str

class minihack.wiki.**TextProcessor**

Bases: `object`

Base class for modeling relations between an object and subject.

__init__()

Initialize self. See help(type(self)) for accurate signature.

preprocess(*input_str: str*) → str

process(*input_str: str*) → str

minihack.wiki.**clean_page_text**(*text: List[str]*) → str

Clean Markdown text to make it more passable into an NLP model.

This is currently very basic, and more advanced parsing could be employed if necessary.

minihack.wiki.**load_json**(*file_name: str*) → list

Load a file containing a json object per line into a list of dicts.

minihack.wiki.**process_json**(*wiki_json: List[dict]*, *ignore_inpage_anchors*) → dict

Process a list of json pages of the wiki into one dict of all pages.

9.16 References

- MiniHack is open-source and available on GitHub.
- Read our recent Facebook AI Research blogpost.
- Check out the MiniHack NeurIPS 2021 paper.

9.17 NetHack

NetHack is one of the oldest and arguably most impactful videogames in history, as well as being one of the hardest roguelikes currently being played by humans. It is procedurally generated, rich in entities and dynamics, and overall an extremely challenging environment for current state-of-the-art RL agents, while being much cheaper to run compared to other challenging testbeds.

For more information about NetHack, check out its wikipedia article, nethack.org, and NetHack wiki.

9.17.1 NetHack Learning Environment

The NetHack Learning Environment (NLE) is a Reinforcement Learning environment based on NetHack 3.6.6 and designed to provide a standard RL interface to the game, and comes You can read more about NLE in the NeurIPS 2020 paper.

PYTHON MODULE INDEX

m

minihack, 78
minihack.base, 93
minihack.level_generator, 96
minihack.navigation, 100
minihack.reward_manager, 102
minihack.skills, 110
minihack.wiki, 112

INDEX

Symbols

__init__() (*minihack.LevelGenerator method*), 79
__init__() (*minihack.MiniHack method*), 83
__init__() (*minihack.MiniHackNavigation method*), 85
__init__() (*minihack.MiniHackSkill method*), 87
__init__() (*minihack.NetHackWiki method*), 89
__init__() (*minihack.RewardManager method*), 89
__init__() (*minihack.base.MiniHack method*), 93
__init__() (*minihack.level_generator.LevelGenerator method*), 96
__init__() (*minihack.navigation.MiniHackNavigation method*), 100
__init__() (*minihack.reward_manager.AbstractRewardManager method*), 102
__init__() (*minihack.reward_manager.CoordEvent method*), 102
__init__() (*minihack.reward_manager.Event method*), 103
__init__() (*minihack.reward_manager.GroupedRewardManager method*), 104
__init__() (*minihack.reward_manager.LocActionEvent method*), 104
__init__() (*minihack.reward_manager.LocEvent method*), 105
__init__() (*minihack.reward_manager.MessageEvent method*), 106
__init__() (*minihack.reward_manager.RewardManager method*), 106
__init__() (*minihack.reward_manager.SequentialRewardManager method*), 110
__init__() (*minihack.skills.MiniHackSkill method*), 110
__init__() (*minihack.wiki.NetHackWiki method*), 112
__init__() (*minihack.wiki.TextProcessor method*), 113

A

AbstractRewardManager (*class in minihack.reward_manager*), 102
add_altar() (*minihack.level_generator.LevelGenerator method*), 96
add_altar() (*minihack.LevelGenerator method*), 79
add_amulet_event() (*minihack.reward_manager.RewardManager method*), 106
add_amulet_event() (*minihack.RewardManager method*), 89
add_boulder() (*minihack.level_generator.LevelGenerator method*), 96
add_boulder() (*minihack.LevelGenerator method*), 79
add_coordinate_event() (*minihack.reward_manager.RewardManager method*), 107
add_coordinate_event() (*minihack.RewardManager method*), 90
add_custom_reward_fn() (*minihack.reward_manager.RewardManager method*), 107
add_custom_reward_fn() (*minihack.RewardManager method*), 90
add_door() (*minihack.level_generator.LevelGenerator method*), 96
add_door() (*minihack.LevelGenerator method*), 79
add_eat_event() (*minihack.reward_manager.RewardManager method*), 107
add_eat_event() (*minihack.RewardManager method*), 90
add_event() (*minihack.reward_manager.RewardManager method*), 107
add_event() (*minihack.RewardManager method*), 90
add_fountain() (*minihack.level_generator.LevelGenerator method*), 97
add_fountain() (*minihack.LevelGenerator method*), 79
add_goal_pos() (*minihack.level_generator.LevelGenerator method*), 97
add_goal_pos() (*minihack.LevelGenerator method*), 80
add_gold() (*minihack.level_generator.LevelGenerator method*), 97
add_gold() (*minihack.LevelGenerator method*), 80
add_kill_event() (*mini-

hack.reward_manager.RewardManager *method*), 108
add_kill_event() (*minihack.RewardManager method*), 91
add_line() (*minihack.level_generator.LevelGenerator method*), 97
add_line() (*minihack.LevelGenerator method*), 80
add_location_event() (*minihack.reward_manager.RewardManager method*), 108
add_location_event() (*minihack.RewardManager method*), 91
add_mazewalk() (*minihack.level_generator.LevelGenerator method*), 97
add_mazewalk() (*minihack.LevelGenerator method*), 80
add_message_event() (*minihack.reward_manager.RewardManager method*), 108
add_message_event() (*minihack.RewardManager method*), 91
add_monster() (*minihack.level_generator.LevelGenerator method*), 97
add_monster() (*minihack.LevelGenerator method*), 80
add_object() (*minihack.level_generator.LevelGenerator method*), 98
add_object() (*minihack.LevelGenerator method*), 80
add_object_area() (*minihack.level_generator.LevelGenerator method*), 98
add_object_area() (*minihack.LevelGenerator method*), 81
add_positional_event() (*minihack.reward_manager.RewardManager method*), 108
add_positional_event() (*minihack.RewardManager method*), 91
add_reward_manager() (*minihack.reward_manager.GroupedRewardManager method*), 104
add_sink() (*minihack.level_generator.LevelGenerator method*), 98
add_sink() (*minihack.LevelGenerator method*), 81
add_stair_down() (*minihack.level_generator.LevelGenerator method*), 98
add_stair_down() (*minihack.LevelGenerator method*), 81
add_terrain() (*minihack.level_generator.LevelGenerator method*), 98
add_terrain() (*minihack.LevelGenerator method*), 81
add_trap() (*minihack.level_generator.LevelGenerator method*), 98
add_trap() (*minihack.LevelGenerator method*), 81
add_wear_event() (*minihack.reward_manager.RewardManager method*), 109
add_wear_event() (*minihack.RewardManager method*), 92
add_wield_event() (*minihack.reward_manager.RewardManager method*), 109
add_wield_event() (*minihack.RewardManager method*), 92

C

check() (*minihack.reward_manager.CoordEvent method*), 102
check() (*minihack.reward_manager.Event method*), 103
check() (*minihack.reward_manager.LocActionEvent method*), 105
check() (*minihack.reward_manager.LocEvent method*), 105
check() (*minihack.reward_manager.MessageEvent method*), 106
check_episode_end_call() (*minihack.reward_manager.AbstractRewardManager method*), 102
check_episode_end_call() (*minihack.reward_manager.GroupedRewardManager method*), 104
check_episode_end_call() (*minihack.reward_manager.RewardManager method*), 109
check_episode_end_call() (*minihack.reward_manager.SequentialRewardManager method*), 110
check_episode_end_call() (*minihack.RewardManager method*), 92
clean_page_text() (*in module minihack.wiki*), 113
collect_reward() (*minihack.reward_manager.AbstractRewardManager method*), 102
collect_reward() (*minihack.reward_manager.GroupedRewardManager method*), 104
collect_reward() (*minihack.reward_manager.RewardManager method*), 110
collect_reward() (*minihack.RewardManager method*), 93
COORD (*minihack.reward_manager.EventType attribute*), 103
CoordEvent (*class in minihack.reward_manager*), 102

E

Event (*class in minihack.reward_manager*), 103
EventType (*class in minihack.reward_manager*), 103

F

fill_terrain() (*minihack.level_generator.LevelGenerator method*), 99
fill_terrain() (*minihack.LevelGenerator method*), 81

G

get_des() (*minihack.level_generator.LevelGenerator method*), 99
get_des() (*minihack.LevelGenerator method*), 82
get_map_array() (*minihack.level_generator.LevelGenerator method*), 99
get_map_array() (*minihack.LevelGenerator method*), 82
get_map_str() (*minihack.level_generator.LevelGenerator method*), 99
get_map_str() (*minihack.LevelGenerator method*), 82
get_neighbor_descriptions() (*minihack.base.MiniHack method*), 94
get_neighbor_descriptions() (*minihack.MiniHack method*), 84
get_neighbor_wiki_pages() (*minihack.base.MiniHack method*), 94
get_neighbor_wiki_pages() (*minihack.MiniHack method*), 84
get_object_direction() (*minihack.base.MiniHack method*), 95
get_object_direction() (*minihack.MiniHack method*), 84
get_page_data() (*minihack.NetHackWiki method*), 89
get_page_data() (*minihack.wiki.NetHackWiki method*), 112
get_page_text() (*minihack.NetHackWiki method*), 89
get_page_text() (*minihack.wiki.NetHackWiki method*), 112
get_screen_description() (*minihack.base.MiniHack method*), 95
get_screen_description() (*minihack.MiniHack method*), 84
get_screen_wiki_page() (*minihack.base.MiniHack method*), 95
get_screen_wiki_page() (*minihack.MiniHack method*), 84
GroupedRewardManager (*class in minihack.reward_manager*), 103

I

init_map() (*minihack.level_generator.LevelGenerator method*), 99
init_map() (*minihack.LevelGenerator method*), 82

K

key_in_inventory() (*minihack.base.MiniHack method*), 95
key_in_inventory() (*minihack.MiniHack method*), 85

L

LevelGenerator (*class in minihack*), 78
LevelGenerator (*class in minihack.level_generator*), 96
load_json() (*in module minihack.wiki*), 113
LOC (*minihack.reward_manager.EventType attribute*), 103
LOC_ACTION (*minihack.reward_manager.EventType attribute*), 103
LocActionEvent (*class in minihack.reward_manager*), 104
LocEvent (*class in minihack.reward_manager*), 105

M

MESSAGE (*minihack.reward_manager.EventType attribute*), 103
MessageEvent (*class in minihack.reward_manager*), 105
minihack
    module, 78
MiniHack (*class in minihack*), 83
MiniHack (*class in minihack.base*), 93
minihack.base
    module, 93
minihack.level_generator
    module, 96
minihack.navigation
    module, 100
minihack.reward_manager
    module, 102
minihack.skills
    module, 110
minihack.wiki
    module, 112
MiniHackNavigation (*class in minihack*), 85
MiniHackNavigation (*class in minihack.navigation*), 100
MiniHackSkill (*class in minihack*), 87
MiniHackSkill (*class in minihack.skills*), 110
module
    minihack, 78
    minihack.base, 93
    minihack.level_generator, 96
    minihack.navigation, 100
    minihack.reward_manager, 102
    minihack.skills, 110
    minihack.wiki, 112

N

NetHackWiki (*class in minihack*), 88
NetHackWiki (*class in minihack.wiki*), 112

P

preprocess() (*minihack.wiki.TextProcessor method*), 113
process() (*minihack.wiki.TextProcessor method*), 113
process_json() (*in module minihack.wiki*), 113

R

reset() (*minihack.base.MiniHack method*), 95
reset() (*minihack.MiniHack method*), 85
reset() (*minihack.reward_manager.AbstractRewardManager method*), 102
reset() (*minihack.reward_manager.Event method*), 103
reset() (*minihack.reward_manager.GroupedRewardManager method*), 104
reset() (*minihack.reward_manager.LocActionEvent method*), 105
reset() (*minihack.reward_manager.RewardManager method*), 110
reset() (*minihack.RewardManager method*), 93
RewardManager (*class in minihack*), 89
RewardManager (*class in minihack.reward_manager*), 106

S

screen_contains() (*minihack.base.MiniHack method*), 95
screen_contains() (*minihack.MiniHack method*), 85
SequentialRewardManager (*class in minihack.reward_manager*), 110
set_area_variable() (*minihack.level_generator.LevelGenerator method*), 99
set_area_variable() (*minihack.LevelGenerator method*), 82
set_start_pos() (*minihack.level_generator.LevelGenerator method*), 100
set_start_pos() (*minihack.LevelGenerator method*), 82
set_start_rect() (*minihack.level_generator.LevelGenerator method*), 100
set_start_rect() (*minihack.LevelGenerator method*), 82
step() (*minihack.base.MiniHack method*), 95
step() (*minihack.MiniHack method*), 85

T

TextProcessor (*class in minihack.wiki*), 113

U

update() (*minihack.base.MiniHack method*), 95
update() (*minihack.MiniHack method*), 85

W

wallify() (*minihack.level_generator.LevelGenerator method*), 100
wallify() (*minihack.LevelGenerator method*), 83