Mesh_world User Guide

This article provides a brief overview of the framework and basic logic of mesh_world. Due to the complexity of mesh_world and our time constraints, we won't be able to cover it in full details but only introduce the core content.

.

Table of Contents

1. Description of “World”

1.1 Attributes:

Spatial attributes:

- In mesh_world, all spatial attributes are represented as two-dimensional arrays.
- There are three spatial properties in mesh_world:

landform_map:

Types: int

The value of a grid cell represents the “height” of that grid cell, and this property describes the rise and fall of the terrain, that is the shape of the ground.

water_map:

Type: float

The value of a grid cell represents the "water height" of that grid cell. Most cells have a water height of zero. Water flows downwards (“mesh_world” has algorithm of water flowing), so water accumulates in low-lying areas.

terrain_map:

Types: int

The terrain of a cell grid defines the "terrain" of that grid cell.

There are seven landforms specified in “mesh_world:

The numbers represent the different types of landforms is from 0 to 6 and from dry to wet are:

“6” is underwater, “5” is marsh land,
“4” is mud land, “3” is common land,
“2” is sandy land, “1” is cobblestone land, “0” is big rock land.

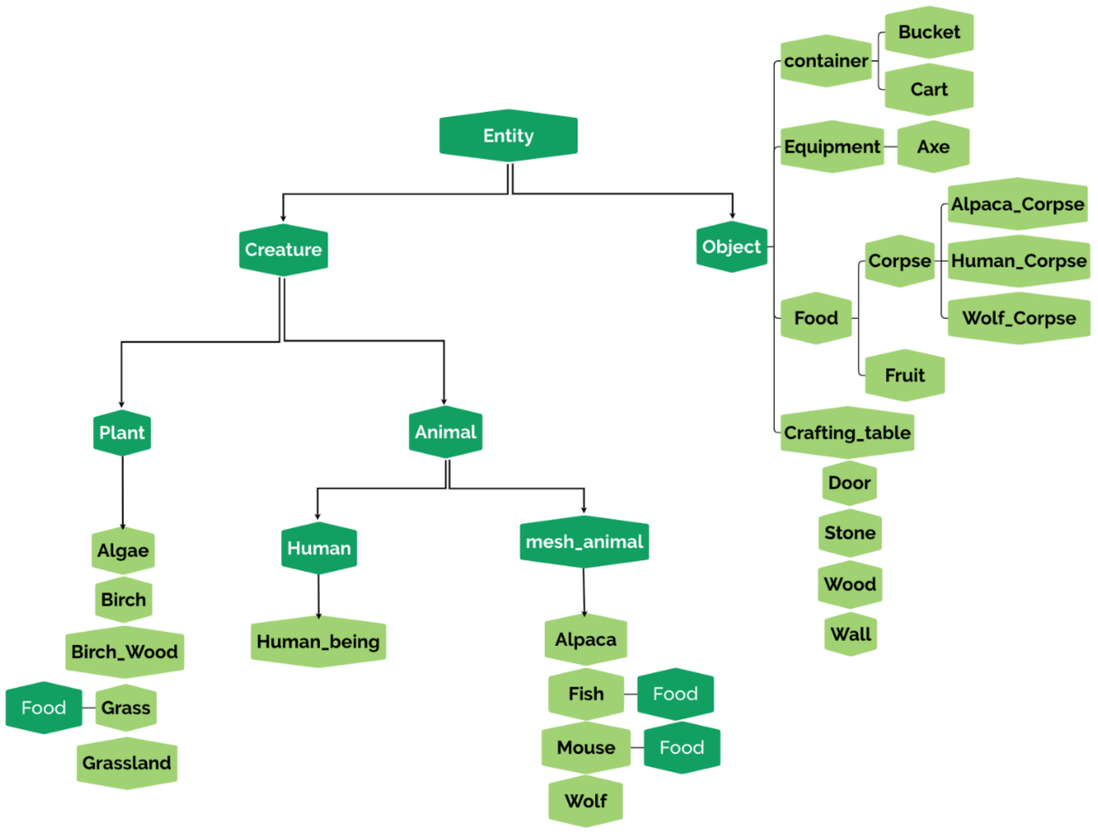
<u>Environment variables:</u>

In mesh_world, we do not set environment variables. In round_the_clock_world, there are environment variables that represent time and day and night.

<u>Entity Architecture</u>:

The following entities are defined in mesh_world:

"Human_being", "Alpaca", "Fish", "Mouse", "Wolf", "Algae", "Birch", "Birch_wood", "Grass", "Grassland", "Alpaca_corpse", "Axe", "Bucket", "Cart", "Crafting_table", "Door", "Fruit", "Human_corpse", "Stone", "Wall", "Wolf_corpse", "Wood".

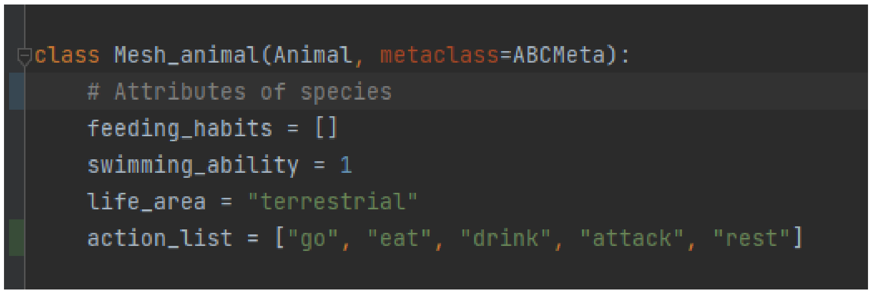


In mesh_world, entities are also divided into large entities and small entities. All entities that inherit from “Big obj” are large entities. Two instances of large entities cannot be on the same grid, and there is no limit to the overlap of small entities.

In mesh_world, our set Wolf, Human_being etc as big entity, mouse, fish etc as small entity.

Animals:

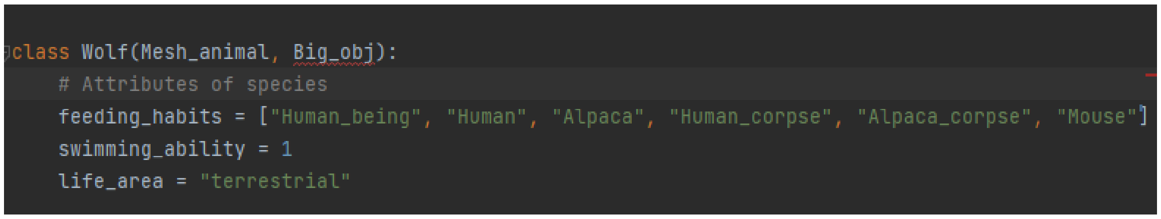
- In “mesh_world”, all animals inherit from the “mesh_animal” superclass. All mesh_animal‘s subclass have four attributes, which are: “feeding_habits”, “swimming_ability”, “life_area”, and “action_list”.



```python
class Mesh_animal(Animal, metaclass=ABCMeta):
    # Attributes of species
    feeding_habits = []
    swimming_ability = 1
    life_area = "terrestrial"
    action_list = ["go", "eat", "drink", "attack", "rest"]
```

- The difference in the value of these four attributes determines the difference between species.

For example, animal “Wolf”:



```python
class Wolf(Mesh_animal, Big_obj):
    # Attributes of species
    feeding_habits = ["Human_being", "Human", "Alpaca", "Human_corpse", "Alpaca_corpse", "Mouse"]
    swimming_ability = 1
    life_area = "terrestrial"
```

1. “Feeding_habits”: dictate what animals eat.

   The picture above illustrates the kinds of entities wolves can eat.

2. “Swimming_ability”: specifies the ability to swim.

   “Swimming_ability” of 1 means wolves cannot reach water depths greater than one. He will be drowned by the water if it does (this logic is not implemented yet).

3. “Life_area” defines the animal's living area.

Currently available values are “Terrestrial”, “Aquatic”, and “Amphibian”. Land animals can swim only as deep as their swimming ability permits and they will be flooded if they arrive deeper waters.

Aquatic animals can swim wherever there is water, but can't go ashore. They will be stranded if they go ashore (this logic has not been perfectly realized).

Amphibians can go ashore or swim into the water. (There are no amphibians yet).

4. “Action_list” specifies the behaviors that animals can perform. Since wolves can behave like most of the animals, therefore it inherited the “action_list” from the “mesh_animal” directly.

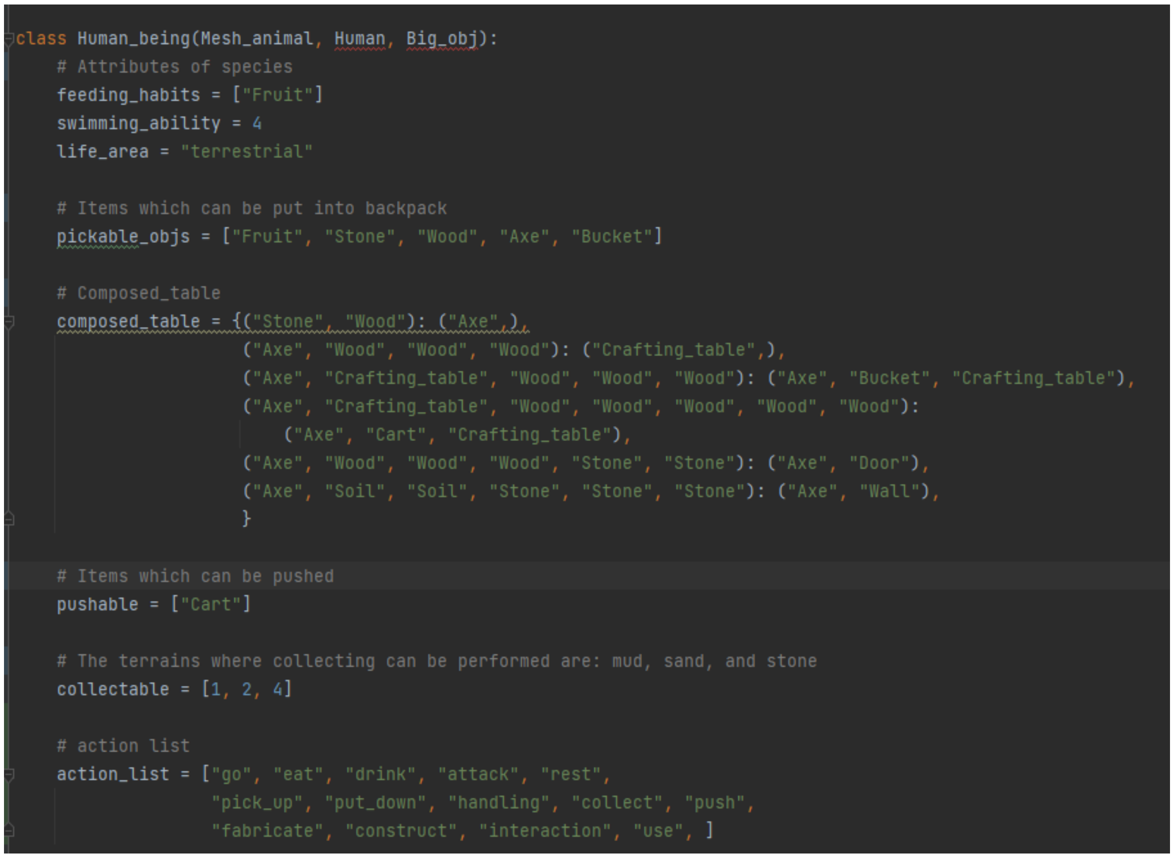
In “mesh_world”, only the human action_list differs from the mesh_animal default.

<u>Human:</u>

In “mesh_world”, humans have many more abilities than animals and there are several fundamental differences between humans and animals:

1. Humans have backpacks that allow them to pick up, carry or drop things.
2. Humans can collect items from specific areas.
3. Humans can push specific objects. (Not implements perfectly, there are still some bugs.)
4. Humans can also synthesize and build things.

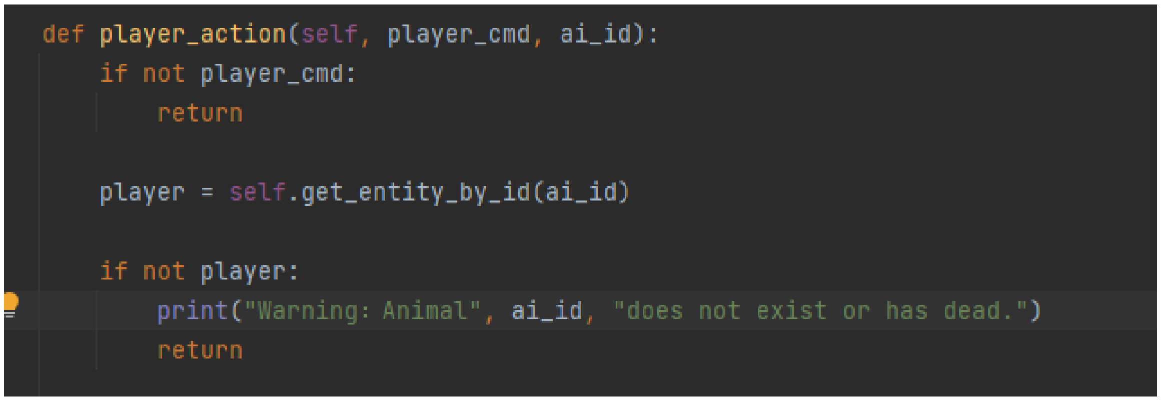
5. Humans can equip, use, and interact with objects. (This is not quite finish yet



```python
class Human_being(Mesh_animal, Human, Big_obj):
    # Attributes of species
    feeding_habits = ["Fruit"]
    swimming_ability = 4
    life_area = "terrestrial"

    # Items which can be put into backpack
    pickable_objs = ["Fruit", "Stone", "Wood", "Axe", "Bucket"]

    # Composed_table
    composed_table = {("Stone", "Wood"): ("Axe",),
                      ("Axe", "Wood", "Wood", "Wood"): ("Crafting_table",),
                      ("Axe", "Crafting_table", "Wood", "Wood", "Wood"): ("Axe", "Bucket", "Crafting_table"),
                      ("Axe", "Crafting_table", "Wood", "Wood", "Wood", "Wood", "Wood"):
                          ("Axe", "Cart", "Crafting_table"),
                      ("Axe", "Wood", "Wood", "Wood", "Stone", "Stone"): ("Axe", "Door"),
                      ("Axe", "Soil", "Soil", "Stone", "Stone", "Stone"): ("Axe", "Wall"),
                      }

    # Items which can be pushed
    pushable = ["Cart"]

    # The terrains where collecting can be performed are: mud, sand, and stone
    collectable = [1, 2, 4]

    # action list
    action_list = ["go", "eat", "drink", "attack", "rest",
                   "pick_up", "put_down", "handling", "collect", "push",
                   "fabricate", "construct", "interaction", "use", ]
```

Animal behavior:

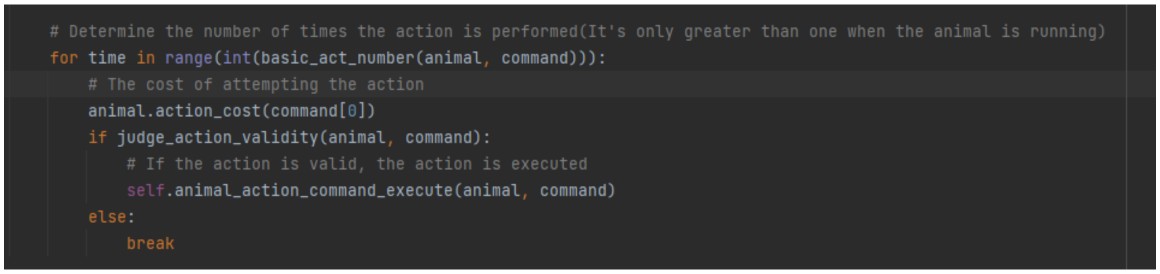
When “mesh_world” receives a command for an entity with an id to perform an action. “mesh_world” determines whether the entity with that ID exists



```python
def player_action(self, player_cmd, ai_id):
    if not player_cmd:
        return

    player = self.get_entity_by_id(ai_id)

    if not player:
        print("Warning: Animal", ai_id, "does not exist or has dead.")
        return
```

and it is an animal or not first. If the entity of the ID is not an animal, the action is invalid.

If the animal exists, make the animal bear the cost of attempting the behavior first. Then determine whether the action can be carried out. If yes, perform the action.



```python
# Determine the number of times the action is performed(It's only greater than one when the animal is running)
for time in range(int(basic_act_number(animal, command))):
    # The cost of attempting the action
    animal.action_cost(command[0])
    if judge_action_validity(animal, command):
        # If the action is valid, the action is executed
        self.animal_action_command_execute(animal, command)
    else:
        break
```

<u>Death of plants and animals:</u>

Creatures drop items when they die. For example, when human die, they will turn into human corpses.

A creature dies when their HP reaches zero. There are many kinds of things that cause the HP of plant and animal to fall. Such as hunger, injury and so on.

1.2 Logic:

- ■ In “mesh_world”, state changes take the following forms:

- ◆ Animal behavior
- ◆ Factors that change over time:

1. The changes that occur to all entities at the end of a turn.
   - a) The animals became hungrier and thirstier.
   - b) Fruits formed
   - c) Animal and plant death
   - d) Spoilage and decay of food(Not implemented yet)
2. The flow of water

Each turn “mesh_world” traverses each animal, gets an action command from the “Brain” instance of the animal (or from openAI), and determines whether the order can be performed. If the condition for executing the command is sufficient, the action will be executed.

After all the animals execute they command, “mesh_world” traverses each creature and determined the natural change of all creatures. For example, human getting hungrier, whether they die, plants bearing fruit, corpses rotting (that's not done yet), etc.

After all entities are updated, “mesh_world” will determine changes to the map, such as water flow, etc.

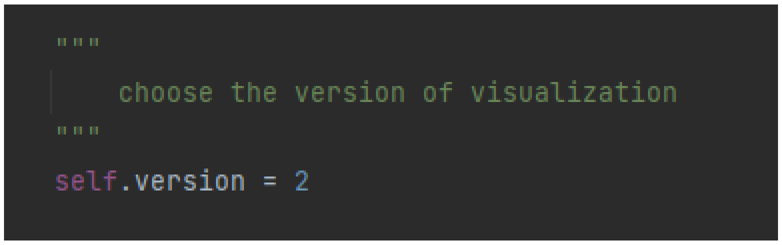
We planned to implement the weather system, but we didn't get it done because of time.

2.Visualization Description

2.1.1 Visualization scheme selection

We made two visualization versions for “mesh_world”. One represented the world with simple squares and circles, the other with more intuitive material.

User can decide which version to use by modifying the “version” attribute in Exhibitor:

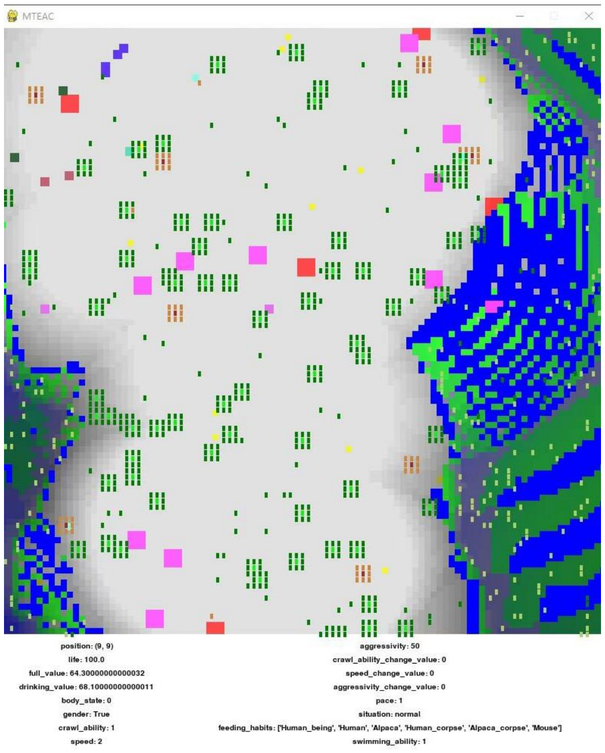


```
"""
    choose the version of visualization
"""
self.version = 2
```

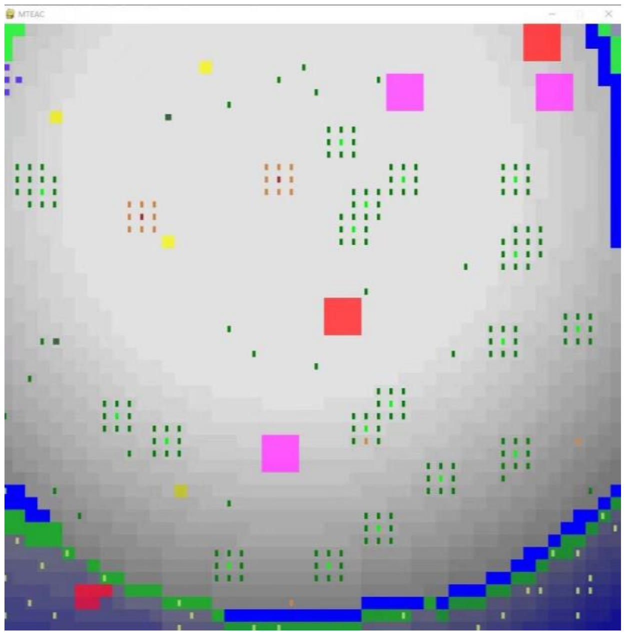
When “version” is 1, the older, cruder version is used. When the version value is 2, use the new, more intuitive version.

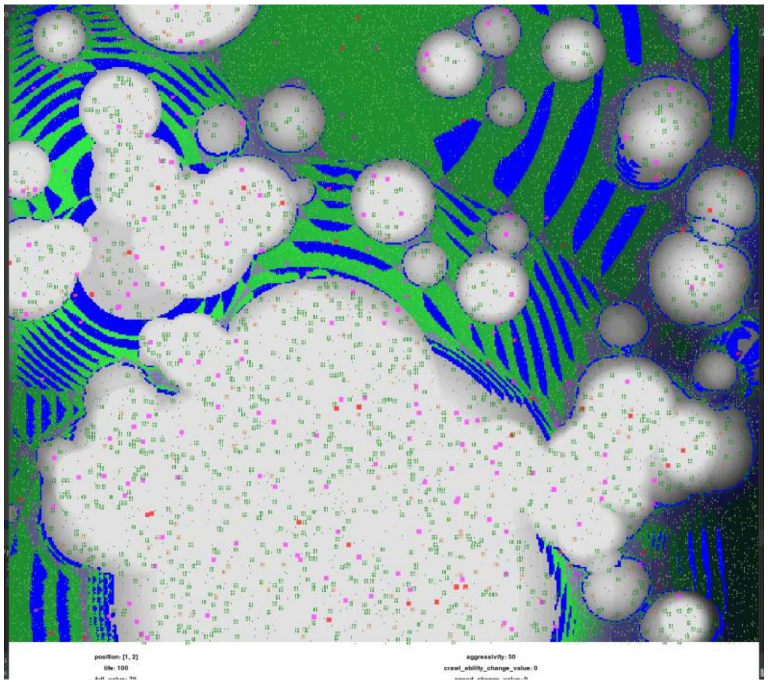
- This is the old version of the visual interface.

The small map we created as shown:



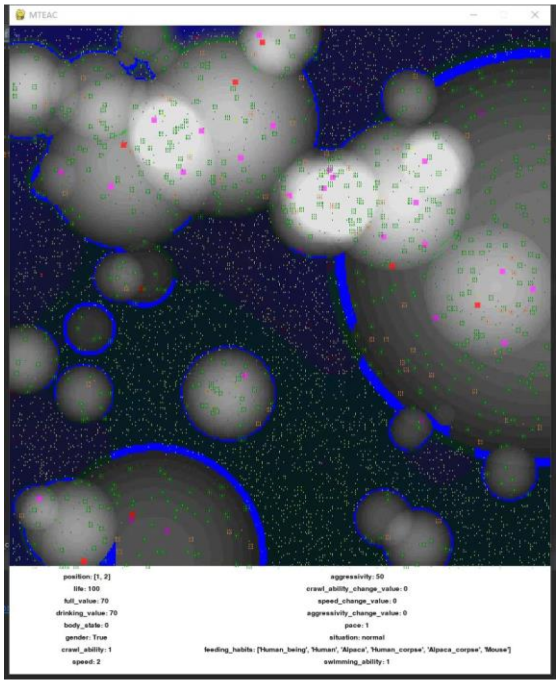
```
position: (9, 9)                                   aggressivity: 50
life: 100.0                                        crawl_ability_change_value: 0
full_value: 64.30000000000032                      speed_change_value: 0
drinking_value: 68.10000000000011                  aggressivity_change_value: 0
body_state: 0                                      pace: 1
gender: True                                       situation: normal
crawl_ability: 1                                   feeding_habits: ['Human_being', 'Human', 'Alpaca', 'Human_corpse', 'Alpaca_corpse', 'Mouse']
speed: 2                                           swimming_ability: 1
```



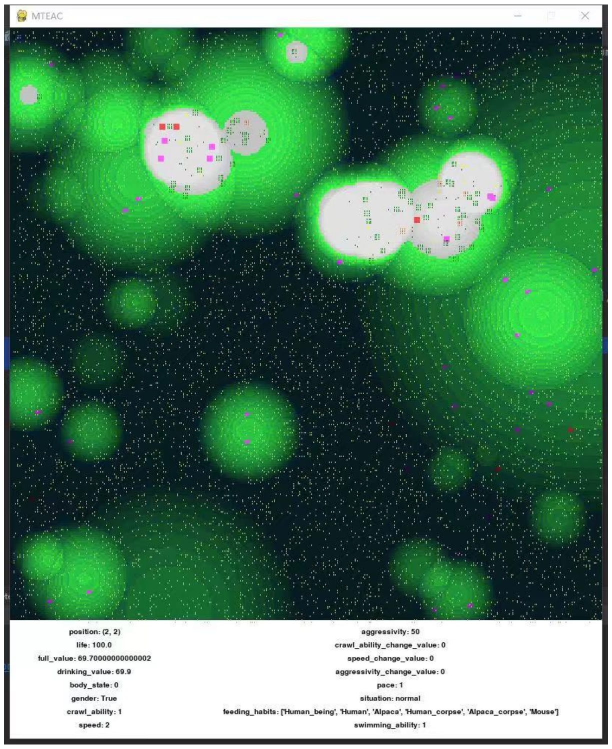


```
position: [1, 2]                                   aggressivity: 50
life: 100                                          crawl_ability_change_value: 0
```

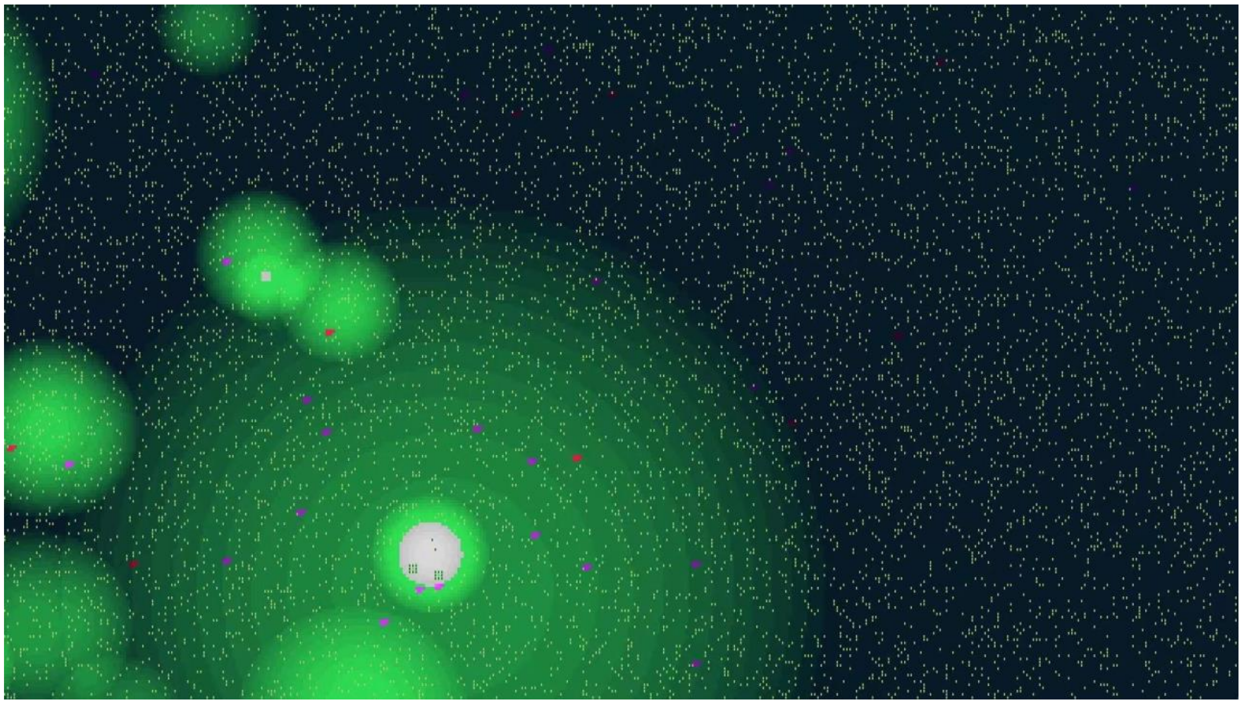
Super big map:



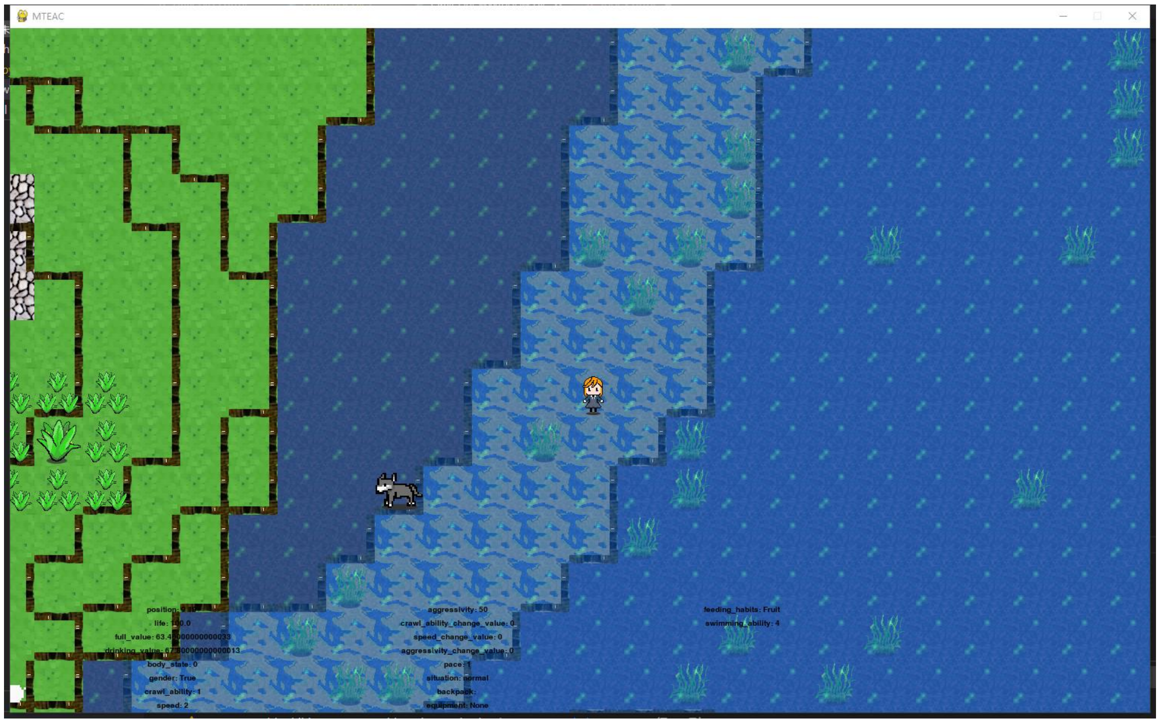
```
position: [1, 2]                aggressivity: 50
life: 100                       crawl_ability_change_value: 0
full_value: 70                  speed_change_value: 0
drinking_value: 70              aggressivity_change_value: 0
body_state: 0                   pace: 1
gender: True                    situation: normal
crawl_ability: 1                feeding_habits: ['Human_being', 'Human', 'Alpaca', 'Human_corpse', 'Alpaca_corpse', 'Mouse']
speed: 2                        swimming_ability: 1
```



```
position: (2, 2)                     aggressivity: 50
life: 100.0                          crawl_ability_change_value: 0
full_value: 69.70000000000002        speed_change_value: 0
drinking_value: 69.9                 aggressivity_change_value: 0
body_state: 0                        pace: 1
gender: True                         situation: normal
crawl_ability: 1                     feeding_habits: ['Human_being', 'Human', 'Alpaca', 'Human_corpse', 'Alpaca_corpse', 'Mouse']
speed: 2                             swimming_ability: 1
```

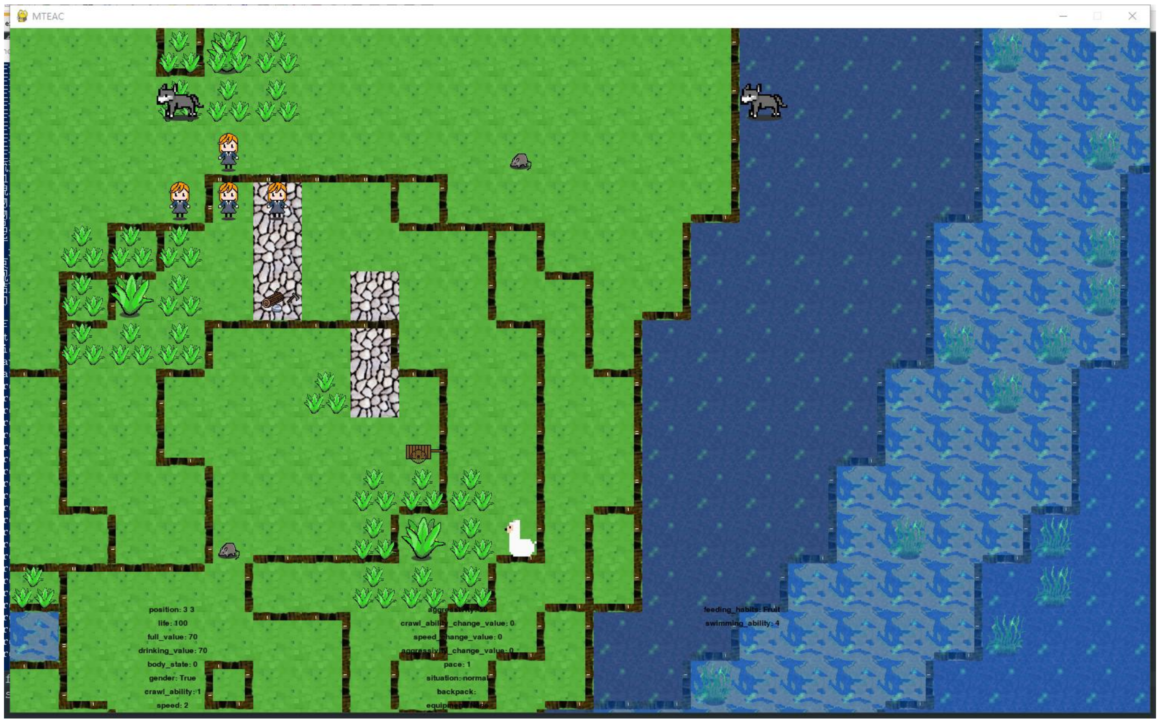


- New version display:



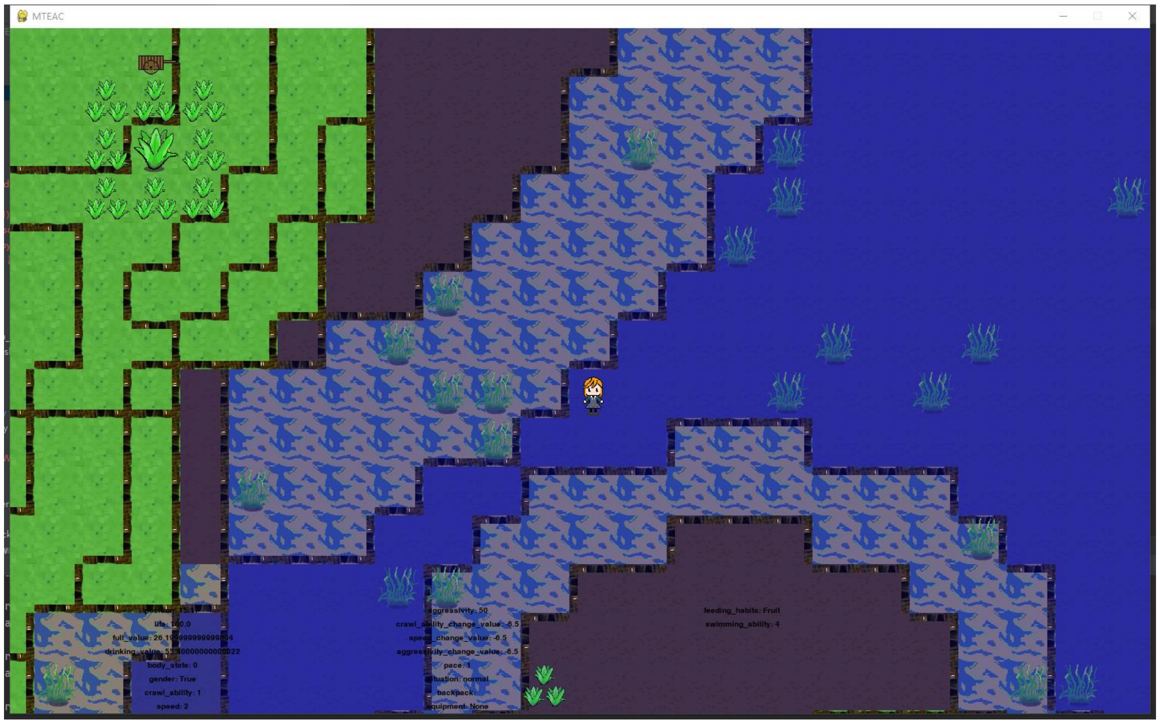
MTEAC

```
position: 9 1[illegible]                 aggressivity: 50                   feeding_habits: Fruit
life: 100.0                              crawl_ability_change_value: 0      swimming_ability: 4
full_value: 63.40000000000033            speed_change_value: 0
drinking_value: 67.80000000000013        aggressivity_change_value: 0
body_state: 0                            pace: 1
gender: True                             situation: normal
crawl_ability: 1                         backpack:
speed: 2                                 equipment: None
```

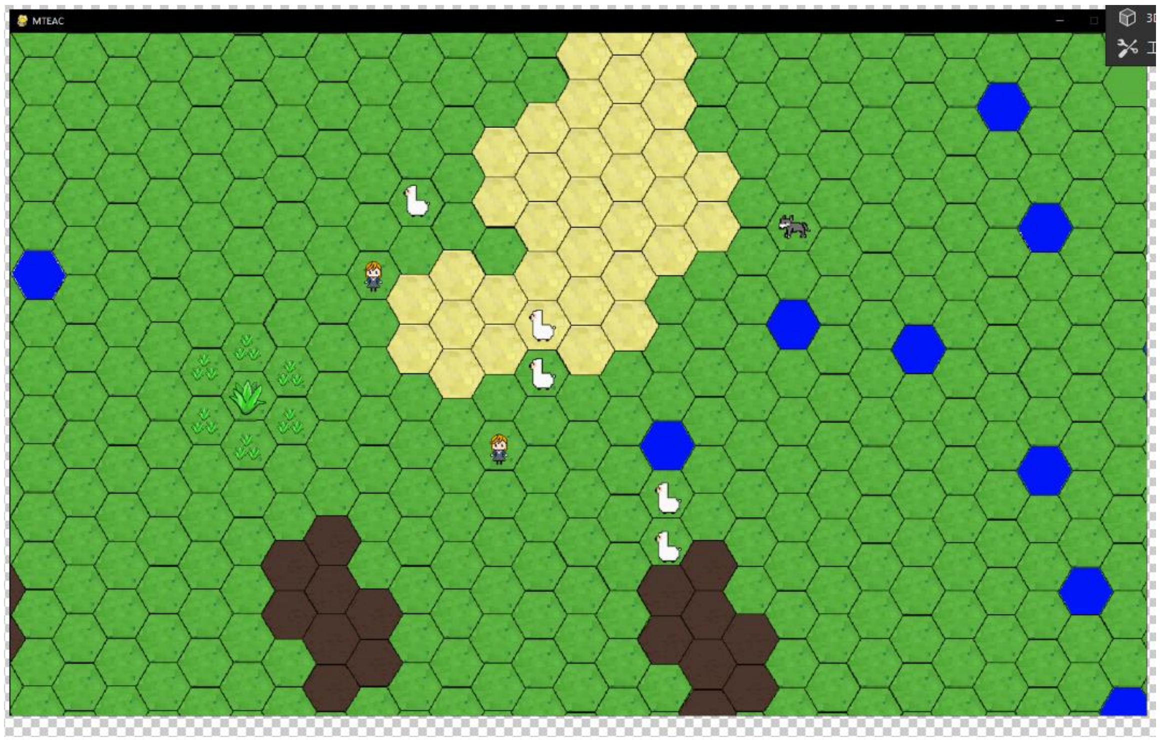


MTEAC

```
position: 3 3                            aggressivity: 50                   feeding_habits: Fruit
life: 100                                crawl_ability_change_value: 0      swimming_ability: 4
full_value: 70                           speed_change_value: 0
drinking_value: 70                       aggressivity_change_value: 0
body_state: 0                            pace: 1
gender: True                             situation: normal
crawl_ability: 1                         backpack:
speed: 2                                 equipment: None
```



```
position: 15 17
life: 100.0
full_value: 26.199999999999804
drinking_value: 55.40000000000022
body_state: 0
gender: True
crawl_ability: 1
speed: 2

aggressivity: 50
crawl_ability_change_value: -6.5
speed_change_value: -6.5
aggressivity_change_value: -6.5
pace: 1
situation: normal
backpack:
equipment: None

feeding_habits: Fruit
swimming_ability: 4
```



Old version visualization:

In the old version, animal entities were represented by squares. Purple squares represent Human_being, green squares represent wolves and pink squares represent fish. As the second picture shows, there are four humans in the upper left corner; Plant entities are represented by elongated rectangles; Object entities are represented by small circles.

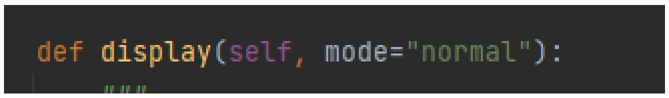
Different terrains correspond to different colors of the ground. For example, green is swamp, blue is mud, red is stone land and so on.

The height of the terrain is indicated by light and shade. The brighter the ground, the higher it is, and the darker the ground, the lower it is. Water depth is shown in translucent blue, with darker blue indicating deeper water depth. As the second picture shows, the north is high and the south is low. The water pooled in the southern lowlands.

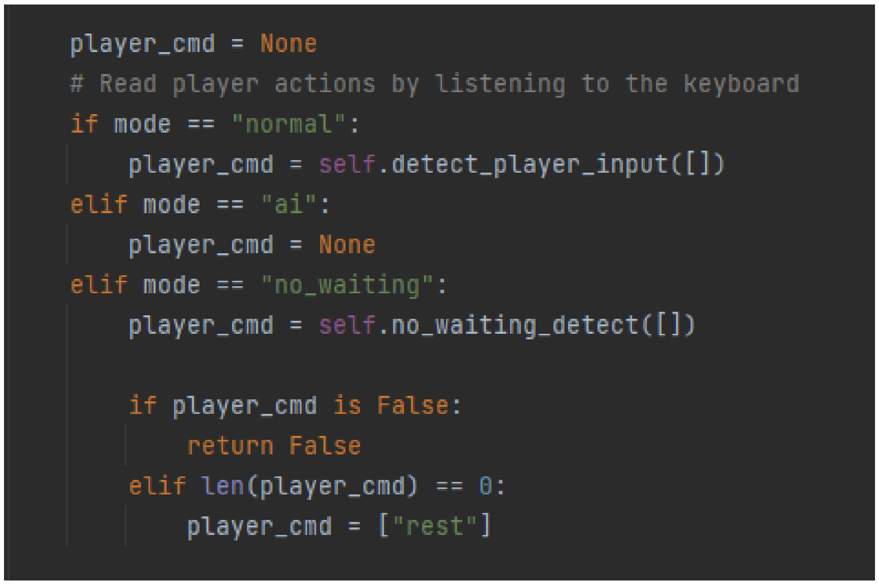
In new version, display of water takes the same logic as the old version.

2.1.2 Visualization mode selection

- The visualization selection under game mode:



```
def display(self, mode="normal"):
    """
```



```
player_cmd = None
# Read player actions by listening to the keyboard
if mode == "normal":
    player_cmd = self.detect_player_input([])
elif mode == "ai":
    player_cmd = None
elif mode == "no_waiting":
    player_cmd = self.no_waiting_detect([])

    if player_cmd is False:
        return False
    elif len(player_cmd) == 0:
        player_cmd = ["rest"]
```

- In “mesh_world”, the display function’s parameter “mode” can take on three different values, representing three different modes.

  - “ai”: In openAI mode, the user's keyboard is independent of the application process.
  - “normal”: The world is blocked in a listening process on the keyboard. Each time the player presses the keyboard, the world takes one turn run.
  - “no_waiting”: The world is not blocked by the player's keyboard.

3.Game mode

<u>mesh world manipulation:</u>

- Correspondence between keys and movements:

  Eat : “Z”, Attack : “X” , Drink : “C” , Rest :“V”,

  Synthesize :“F”, Build : “R”,

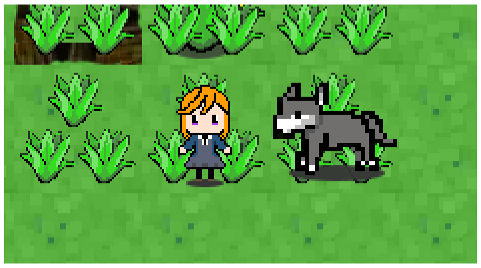
  Pick Up : ”A”, Put Down : “S”, Equip : ”E”, Collect : ”G”,

  Push and Pull : “T”

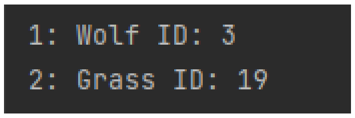
When the action does not require an object to be specified, the keystroke action that presses the action is executed. When an action needs to specify an object, user should press the arrow key of the action to select the object after they press the action’s key.

For example, the action of resting. Just press “v” then it will be executed, because the action “rest” do not need to specify objects.

If you are attacking an action with an action object, press “X” and then you need to press “↑”, “↓”, “←”, “→”, or “V” (In its original position) to select which object to attack. If there are multiple entities in that place, the terminal lists all entities in that place and sorts them by number, and then the player should presses the number key to select which numbered entity to attack.



For example, in the scene above, we as the human on the left attack the Wolf on the right. We need to press “X”, and then press the “right“ key →. The terminal will pop up:

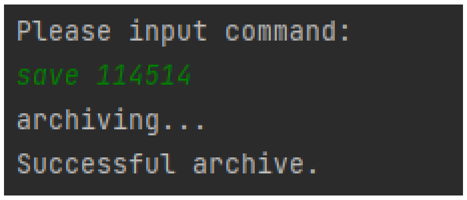


```
1: Wolf ID: 3
2: Grass ID: 19
```

At this point we press the number key “1” to attack the Wolf (press the number “2” will attack the grass).

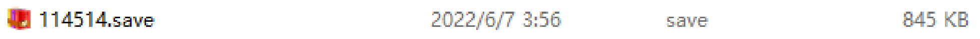
Load and Archive:

“mesh_world” has background functionality. We can use the background to save during the game, for example:



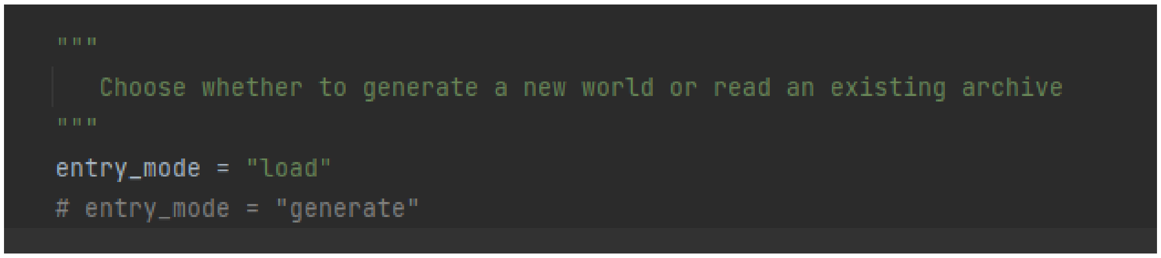
```
Please input command:
save 114514
archiving...
Successful archive.
```

Now you can see the “114514.save” file is added under the save folder.



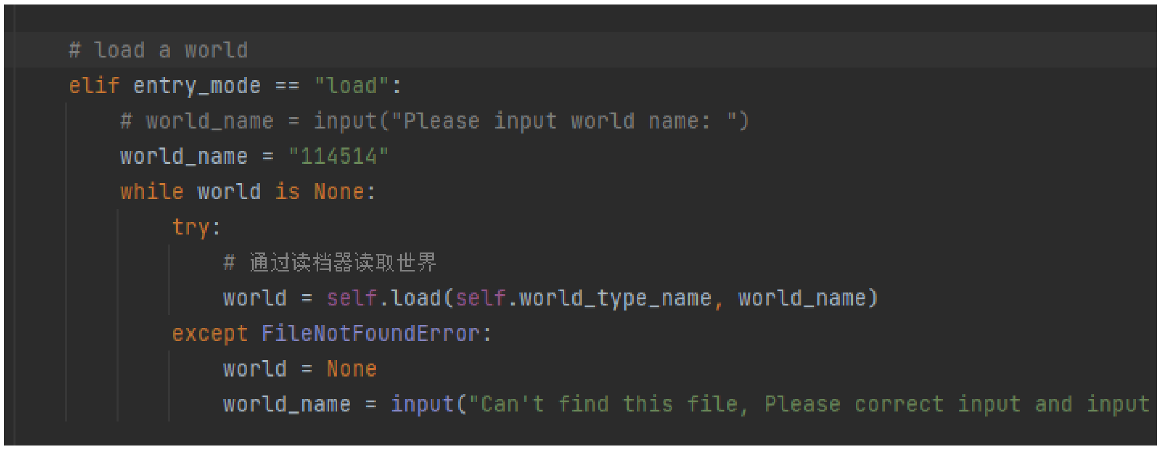
```
114514.save          2022/6/7 3:56       save        845 KB
```

If you want to read, find “entry_mode” in “world_env_interface.py” and assign “entry_mode” to "load".



```
"""
    Choose whether to generate a new world or read an existing archive
"""
entry_mode = "load"
# entry_mode = "generate"
```

Then find the following “world_name” variable and assign it to the file name (no suffixes needed).



```
# load a world
elif entry_mode == "load":
    # world_name = input("Please input world name: ")
    world_name = "114514"
    while world is None:
        try:
            # 通过读档器读取世界
            world = self.load(self.world_type_name, world_name)
        except FileNotFoundError:
            world = None
            world_name = input("Can't find this file, Please correct input and input
```

After running MTEAC, the archive will be loaded.