National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2022/2023

Mission 14
Hungry Games Training, Part III

Release date: 19th October 2022
Due: 1st November 2022, 23:59

Required Files

- mission14.pdf
- mission14.zip

Background

It is now time to equip your Tribute with the skills necessary to survive in the Hungry Games.

The goal of the Games is survival. A number of Tributes will be sent into an arena and the winner is the last one standing. Tributes can die from hunger, perish from attacks by roaming wild animals, or be killed in combat by other Tributes.

What should your Tribute do to survive in the Games? Should she always attack Animals for food, or should she only do so when her hunger value has become extremely low? Should she always be on the offensive and attack other Tributes, or should she try to be defensive and run away to outlast the other Tributes? You decide!

Introduction

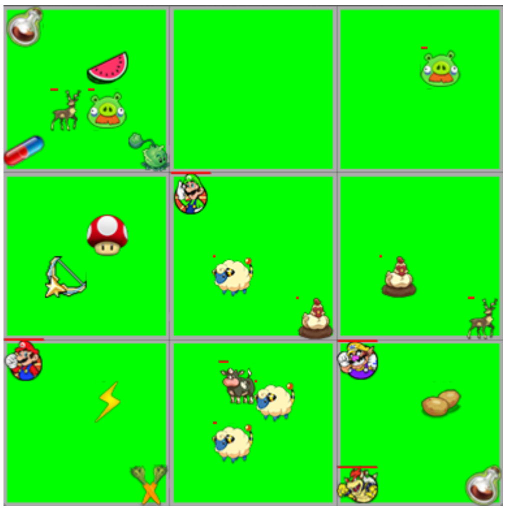
In this mission, you will create an "Artificial Intelligence" (AI) agent. Create a new class, `XX_AI` (where `XX` is your name) that extends the `Tribute` class, and give your Tribute intelligence. You will then run a simulation with `XX_AI` to ensure that your AI can perform some basic tasks.

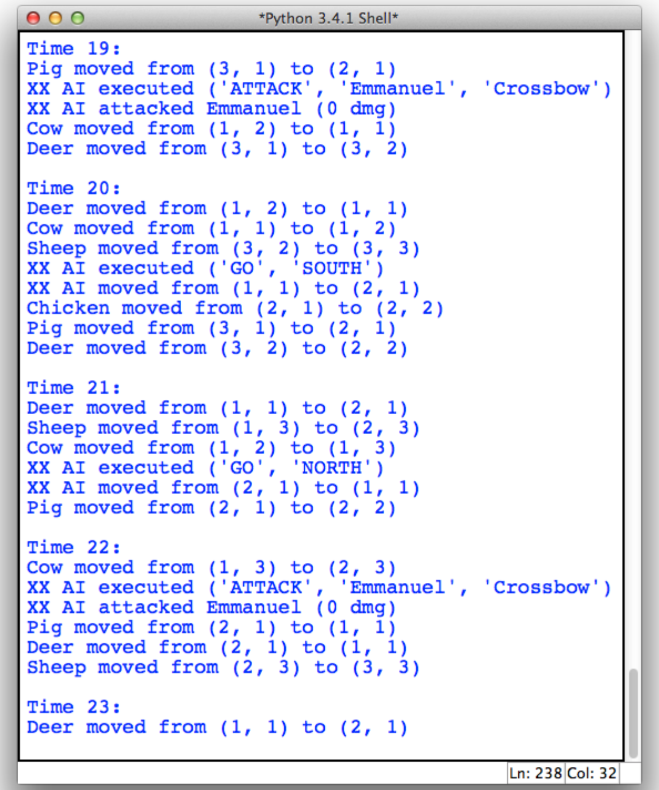
Simulation

For this mission, we have provided you with a simulation engine to test the behaviour of your AI agent. Please go to **Coursemology** and **download** the archived file `mission14.zip` and **extract it** somewhere on your computer. The template file `mission14-template.py` is included in the zip file.

Similiar to previous missions, you should implement your AI class in `mission14-template.py`. To run the simulations, simply uncomment the task you wish to simulate at the bottom of this file, and run this file in IDLE.

However, due to limitations of the graphical interface, please do **NOT** simulate more than one task at any time. Do it one at a time, and comment the tasks you do not wish to simulate.


(a) Graphics User Interface



```
Time 19:
Pig moved from (3, 1) to (2, 1)
XX AI executed ('ATTACK', 'Emmanuel', 'Crossbow')
XX AI attacked Emmanuel (0 dmg)
Cow moved from (1, 2) to (1, 1)
Deer moved from (3, 1) to (3, 2)

Time 20:
Deer moved from (1, 2) to (1, 1)
Cow moved from (1, 1) to (1, 2)
Sheep moved from (3, 2) to (3, 3)
XX AI executed ('GO', 'SOUTH')
XX AI moved from (1, 1) to (2, 1)
Chicken moved from (2, 1) to (2, 2)
Pig moved from (3, 1) to (2, 1)
Deer moved from (3, 2) to (2, 2)

Time 21:
Deer moved from (1, 1) to (2, 1)
Sheep moved from (1, 3) to (2, 3)
Cow moved from (1, 2) to (1, 3)
XX AI executed ('GO', 'NORTH')
XX AI moved from (2, 1) to (1, 1)
Pig moved from (2, 1) to (2, 2)

Time 22:
Cow moved from (1, 3) to (2, 3)
XX AI executed ('ATTACK', 'Emmanuel', 'Crossbow')
XX AI attacked Emmanuel (0 dmg)
Pig moved from (2, 1) to (1, 1)
Deer moved from (2, 1) to (1, 1)
Sheep moved from (2, 3) to (3, 3)

Time 23:
Deer moved from (1, 1) to (2, 1)
```

(b) Console/Shell

Figure 1: Hungry Games Simulation

We have provided both a graphical output (Figure 1) and a textual output (Figure 2) for the simulation. The graphical output will execute each event one at a time, and might take very long to complete if your solution takes a long time to run. If you do not wish to see the graphical output, simply set `gui=False` in the simulation task code.

```
# Run task 1 with gui
simulation.task1(XX_AI("XX AI", 100), gui=True)

# Without gui
simulation.task1(XX_AI("XX AI", 100), gui=False)
```

Please note that in order for the simulation to work correctly, you should **NOT** be modifying any file other than `mission14-template.py`.

NOTE: To ensure consistency across all the AI classes, we have provided you with the reference classes for the Hungry Games. These classes are correct implementations of what you did in Missions 12 and 13. As such, there is no need for you to include your code from Mission 12 and 13.

Tribute AI

In order to stand a fighting chance at the Hungry Games, your Tribute must be able to, based on the prevailing circumstances, perform one of the following actions during each turn:

a. Successfully attack another `LivingThing` at the current location
b. Pick up a `Thing` object from the ground, consisting of any `Weapon`, `RangedWeapon`, `Food`, `Medicine` or `Ammo` object
c. Eat a `Food` or `Medicine` object from inventory
d. Move to a different location that is accessible from the current location
e. Load a `RangedWeapon` object with a compatible `Ammo` object when both are in inventory
f. Do nothing - skip the current turn

The appropriate action to perform at each turn is to be determined by your AI implementation. In order to achieve this, your AI class must implement a compulsory method, `next_action()`, and this method should return a tuple indicating what your Tribute will do in the current turn.

In the `next_action()` method, you are only allowed the use the following methods of Tribute:

a. `get_health()`: Returns your Tribute's current health
b. `get_weapons()`: Returns the tuple of weapons currently in your inventory.
c. `get_food()`: Returns the tuple of food currently in your inventory.
d. `get_medicine()`: Returns the tuple of medicines currently in your inventory.
e. `objects_around()`: Returns the objects at the current location, including other Tributes, but excluding yourself
f. `get_exits()`: Returns the list of possible exits at your current location.
g. `get_hunger()`: Returns your Tribute's current hunger
h. `get_inventory()`: Returns a list of your Tribute's current inventory

You are also allowed to access the getters (methods names that start with a `get_`) on the objects you see around you, i.e. you can call `get_weapons()` for the other Tributes, or `get_food_value()` for the food you see, or `get_health()` for the animals you see.

In addition, you are allowed to query certain objects using the following methods.

a. `min_damage()` of weapons
b. `max_damage()` of weapons
c. `weapon_type()` of ammos
d. `shots_left()` of ranged weapons

However, you are **NOT** allowed call the setters, or access the properties of any objects directly! That means that you should not be making any changes to the state of the world in your code. In addition, you should not allow your Tribute to see beyond your current location (i.e. you are **not** allowed to interact with any `Place` object directly). Doing so would result in significant deduction in your marks for this Mission!

`next_action()` can have the following valid return values:

a. `("ATTACK", LivingThing object, Weapon object)` : Attacks `LivingThing` with `Weapon`
b. `("TAKE", Thing object)` : Takes `Thing` and puts it in your inventory. Note that your Tribute can only take `Thing` if it is in the same Place as the Tribute.

c. `("EAT", Food object)` : Eats the `Food` in your inventory

d. `("GO", Direction)` : Goes to `Direction` (one of the values in `self.get_exits()`)

e. `("LOAD", RangedWeapon object, Ammo object)` : Loads `Ammo` for `RangedWeapon`

f. `None`: Do nothing

Remember: You are **not** allowed to access the state and properties of any objects directly, and you are only allowed to use getters for the objects. You may, however, store states in your AI class if necessary.

IMPORTANT WARNING: Because we provide you with the flexibility in choosing the approach by which you solve the problems in this mission, **we require that you submit well commented/annotated code.** Describe your approach to the programming questions, and then annotate the blocks of relevant code that implements your idea. **If you fail to do so, you risk having marks deducted by your tutor.** Do not assume that your tutor can read your mind.

We have created a couple of special training grounds, to help you verify your `next_action` method. Your method should be able to produce a sequence of actions that satisfies the objective of each training ground.

A training ground of size N contains $N \times N$ `Place` instances arranged in a square grid. In this mission, a `LivingThing` who stays at a place in the training ground can go north, south, east, and west. The `LivingThing` cannot move out of the boundary of the training ground.

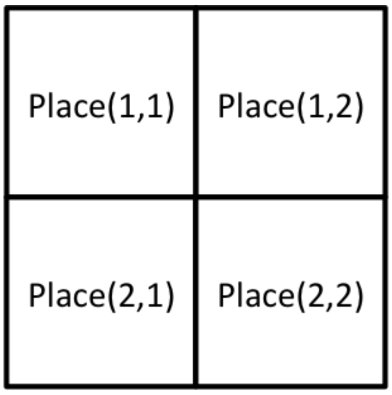
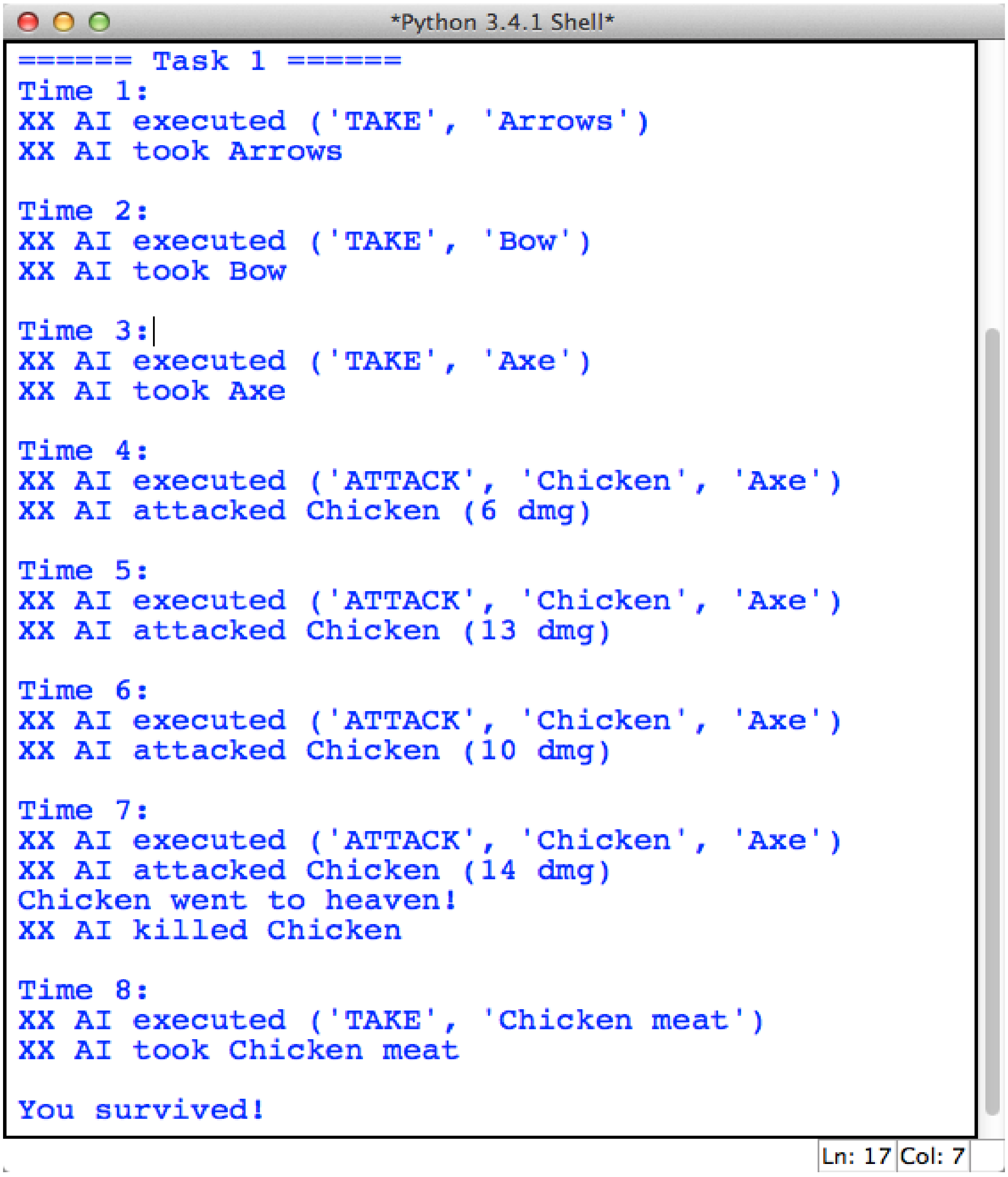


Figure 2: 2x2 training area

Task 1: Eating and Hunting (5 marks)

In this 1×1 training ground, there is only one area, and it contains a `Chicken`, an `Axe`, and a `Bow`. Your objective is to make sure that your `next_action` method is capable of producing a sequence of actions that will allow your AI to kill the `Chicken` and take the `Chicken` meat.



Python 3.4.1 Shell

```
====== Task 1 ======
Time 1:
XX AI executed ('TAKE', 'Arrows')
XX AI took Arrows

Time 2:
XX AI executed ('TAKE', 'Bow')
XX AI took Bow

Time 3:
XX AI executed ('TAKE', 'Axe')
XX AI took Axe

Time 4:
XX AI executed ('ATTACK', 'Chicken', 'Axe')
XX AI attacked Chicken (6 dmg)

Time 5:
XX AI executed ('ATTACK', 'Chicken', 'Axe')
XX AI attacked Chicken (13 dmg)

Time 6:
XX AI executed ('ATTACK', 'Chicken', 'Axe')
XX AI attacked Chicken (10 dmg)

Time 7:
XX AI executed ('ATTACK', 'Chicken', 'Axe')
XX AI attacked Chicken (14 dmg)
Chicken went to heaven!
XX AI killed Chicken

Time 8:
XX AI executed ('TAKE', 'Chicken meat')
XX AI took Chicken meat

You survived!
```

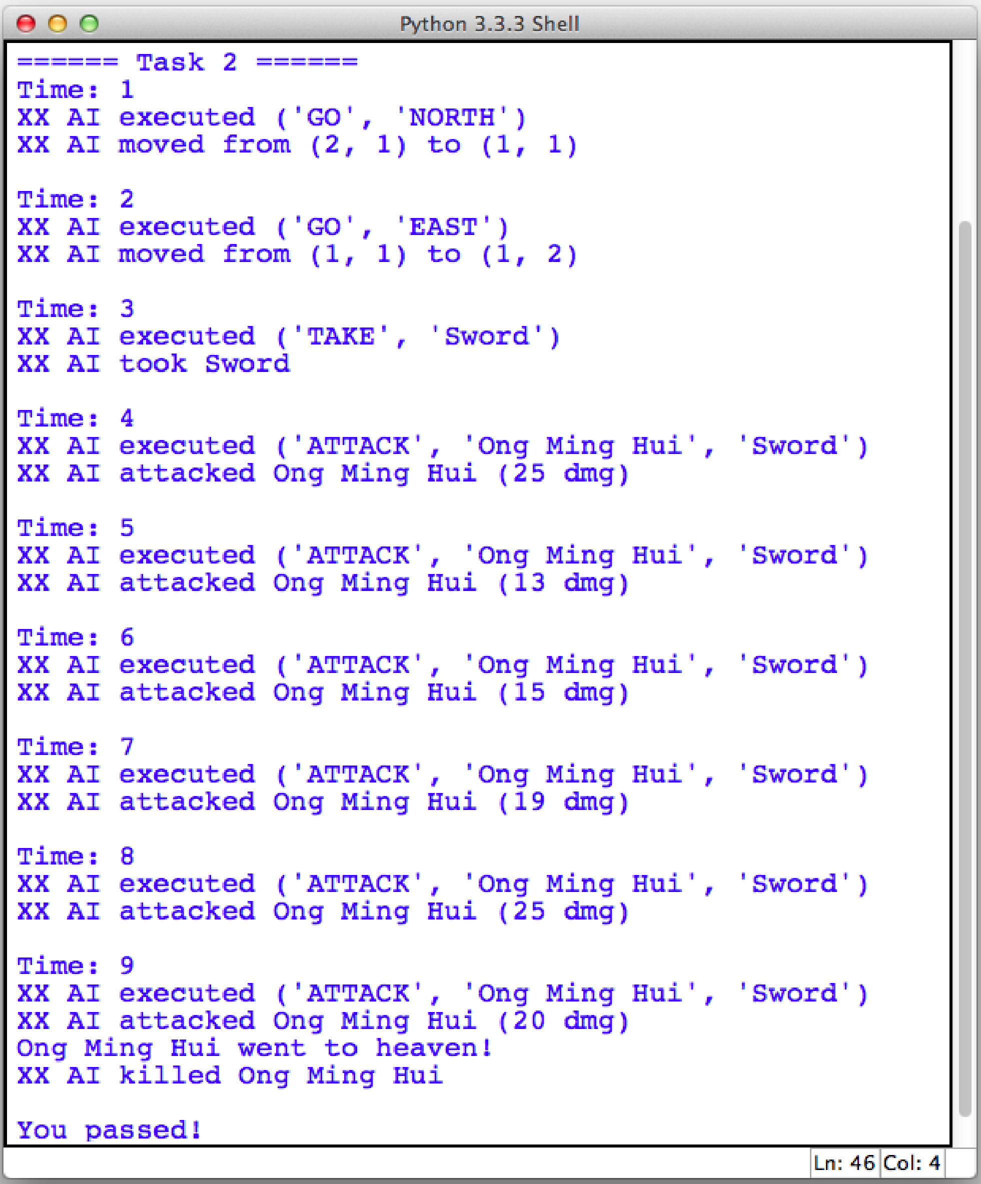
Ln: 17 Col: 7

Sample run for Task 1

Task 2: Attack! (5 marks)

In this 2 × 2 training ground, there are weapons scattered in the map, as well as a stationary docile Tribute. Your objective is to make sure that your `next_action` method is capable of producing a sequence of actions that will allow your AI to pick up weapons, find the docile Tribute, and kill him before starving to death.

Note: your Tribute cannot see beyond the place it is staying (i.e., it cannot take items or attack enemies at other places without first moving there).



```
====== Task 2 ======
Time: 1
XX AI executed ('GO', 'NORTH')
XX AI moved from (2, 1) to (1, 1)

Time: 2
XX AI executed ('GO', 'EAST')
XX AI moved from (1, 1) to (1, 2)

Time: 3
XX AI executed ('TAKE', 'Sword')
XX AI took Sword

Time: 4
XX AI executed ('ATTACK', 'Ong Ming Hui', 'Sword')
XX AI attacked Ong Ming Hui (25 dmg)

Time: 5
XX AI executed ('ATTACK', 'Ong Ming Hui', 'Sword')
XX AI attacked Ong Ming Hui (13 dmg)

Time: 6
XX AI executed ('ATTACK', 'Ong Ming Hui', 'Sword')
XX AI attacked Ong Ming Hui (15 dmg)

Time: 7
XX AI executed ('ATTACK', 'Ong Ming Hui', 'Sword')
XX AI attacked Ong Ming Hui (19 dmg)

Time: 8
XX AI executed ('ATTACK', 'Ong Ming Hui', 'Sword')
XX AI attacked Ong Ming Hui (25 dmg)

Time: 9
XX AI executed ('ATTACK', 'Ong Ming Hui', 'Sword')
XX AI attacked Ong Ming Hui (20 dmg)
Ong Ming Hui went to heaven!
XX AI killed Ong Ming Hui

You passed!
```

Sample run for Task 2

Optional Task: Final training before the Battle

Before you proceed to our **Hungry Games Contest** , here is a final training ground for you to get used to the enviroment.

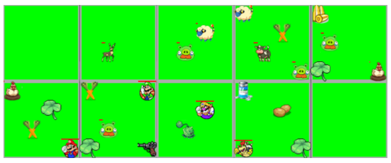
In this 3×3 training ground, there are various objects scattered in the map, and same docile Tributes. Ensure that your `next_action` method can use Weapons, Ranged-Weapons, Food and Medicine.

The map for the optional task is generated with the `config` function. Feel free to modify the code inside the `config` function and see how your AI behaves!

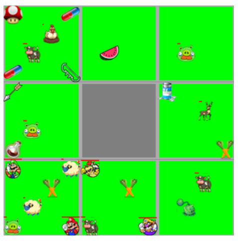
Various Map Designs

Additionally, we have also provided you various map designs to try out for fun. Instead of using `SquareMap(3)` in the `config` function, you can replace it with different kinds of maps listed below:

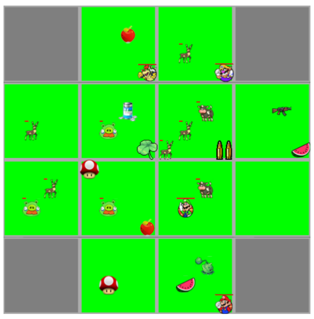
- `RectangleMap(width, height)`, e.g. `RectangleMap(5, 2)`.



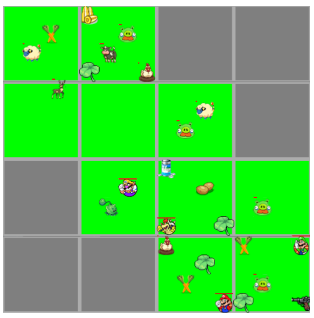
- `SquareWithHoleMap(size)`, e.g. `SquareWithHoleMap(3)`.



- `CrossMap(size)`, e.g. `CrossMap(4)`.



- `TridiagonalMap(size)`, e.g. `TridiagonalMap(4)`.



- `CircleMap(size)`, a circle-shaped map.
- `InputMap(layout)`, takes in a 2D-array of 0s and 1s. For example, `InputMap([[1, 1, 1], [1, 0, 1], [1, 1, 1]])` is equivalent to `SquareWithHoleMap(3)`.

NOTE: This is an optional task and will not be graded.