Project Report : Snoo Simulation

Name: Ben Niu

Student Number: 21678145

Date:01/05/2024

# Overview

The Snoo simulation program simulates the behaviour of different animals and people in a house. The owner wakes up and sleeps, opens and closes doors, feeds food, and plays with animals at the correct time of the day. The animals find food by smelling, water and the owner by hearing, and toys by sight. They can eat food, drink water, play swings, take balls away, communicate with people, and sleep at night.

User Guide

This program requires basic packages, including: numpy, matplotlib, random, time, unittest, configparser, argparse and os. Please install these packages before starting this program.

To run this program, use a command like: “python3 snoo.py”. Program support configuration file and arguments, run “python3 snoo.py -h” to get the help information as below:

|  |
| --- |
| usage: snoo.py [-h] [-f CONFIG] [-s SPEED]  Base simulation for the FOP Assignment, Sem 1 2024  optional arguments:  -h, --help show this help message and exit  -f CONFIG, --config CONFIG  specify config file to the program: config\_def.cfg  -s SPEED, --speed SPEED  specify simulation speed: 1 for 1s, 5 for 0.2s |

Please use “-s SPEED” or “--speed SPEED” to control simulated speed.

Please specify a customised configure file using “-f config\_def.cfg” or “--config config\_demo.cfg”. There are some preset configuration files named \*.cfg in the same folder.

There are many configuration items in the \*.cfg file. Please see the details below:

|  |
| --- |
| [DEFAULT]  # Define the figure default size(width, height) in inches.  fig\_width = 10  fig\_height = 15  # Define the simulation speed.  speed = 1  [MAP]  # Define the map items by the array size.  size\_width = 120  size\_height = 90  house\_width = 50  hourse\_height = 40  garden\_width = 20  garden\_height = 15  road\_width = 5  wall\_width = 1  fence\_width = 1  road\_fence\_distance = 10  [SIMULATION]  # Define the simulation numbers by second.  simu\_num = 100  # Define the simulation start time in a day. 22\*60\*60-30 means 10 pm - 30 s.  simu\_start = 22\*60\*60-30  # Define the animal's number and the human’s number.  simu\_dog = 1  simu\_puppy = 1  simu\_dog\_colours = ["gold/black", "white/black", "black"]  simu\_squirrel = 1  simu\_owner = 1  simu\_friend = 1  simu\_stranger = 1  simu\_intruder = 1  simu\_water = 1  simu\_swing = 1  simu\_ball = 1  simu\_ball\_colours = ["red", "pink", "blue"] |

After the program launched, there was a screenshot shown below:

A screenshot of a computer screen

Description automatically generated

In the upper part of the picture, there are 5 figures listed. The first figure, titled with the simulation time, shows a house with grass, house, a front/back yard, a garden in the backyard, fences, and roads. There are many animals, humans, and things placed on the map. There is an annotation that describes the item’s name and status. The next 4 figures are designed to monitor the behaviour for different purposes. The “Position” figure shows all item's positions. The “History” figure shows the movement of the item's history. The “Sense” figure shows a smell/hear/sight heat map. The “Mask” figure shows the scope of the movement item steps.

In the lower part of the picture, there are 4 buttons. The “1 hour passed” button adds 1 hour for time simulation. The “1 min passed” button adds 1 minute for time simulation. The “Pause/Resume” button pauses or resumes the current simulation, which is useful for checking the detailed simulation status. The “Exit” button is used to exit the program before the simulation ends.

Traceability Matrix

The table gives an overview of the features and the implementation and testing of the code.

* **Feature** - numbered for easy referencing
* **Code reference** – reference to files/classes/methods or snippets of code only, do not put the whole program in the report OR “Not Implemented”
* **Test reference** – test code or describe how you tested your feature, N/A if not implemented
* **Status** – P = passed tests, S = skipped, F = failed, or N/A
* **Date Completed** – date or “Ongoing” or N/A if not implemented

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | Code Reference | Test Reference | Status | Date Completed |
| 1.0 Animals |  |  |  |  |
| 1.1 Animal | Class Animal in animals.py |  |  |  |
| 1.1.1 Animal has hungry, thirsty, lonely, bored attribute | hungry, thirsty, lonely, bored attributes and is\_hungry(), is\_thirsty(), is\_bored(), is\_lonely() methods in Animal class | testAnimals.py: TestAnimals.test\_dog | P | 01/05/24 |
| 1.1.2 Animal colour parsing | parse\_colour() in Animal class | testAnimals.py: TestAnimals.test\_animal | P | 01/05/24 |
| 1.1.3 Animal age and energy degree | age, counter attributes and energy\_degree(), check\_energy() methods in Animal class | testAnimals.py: TestAnimals.test\_animal | P | 01/05/24 |
| 1.1.4 Animal senses | smell, sight, hear in Animal class | N/A | N/A | 02/05/24 |
| 1.2 Dog | Class Dog in animals.py |  |  |  |
| 1.2.1 Dog is plotted as specified colours | draw() method in Dog class | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 1.2.2 Dog actions: eat, drink, play, communicate | eat(), drink(), play(), commu() methods in Dog class | N/A | N/A | 01/05/24 |
| 1.2.3 Dog max step and sense distance | MAX\_STEP, SMELL\_DISTANCE, SIGHT\_DISTANCE, HEAR\_DISTANCE defined in Dog class | Plotting in “Sense” figure | P | 01/05/24 |
| 1.2.4 Dog actions: take a toy | take\_toy(), release\_toy() methods in Dog class | N/A | N/A | 03/05/24 |
| 1.3 Puppy | Class Puppy in animals.py |  |  |  |
| 1.3.1 Puppy is plotted as specified colours and shape | draw() method in Puppy class | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 1.3.2 Puppy max step and sense distance | MAX\_STEP, SMELL\_DISTANCE, SIGHT\_DISTANCE, HEAR\_DISTANCE defined in Puppy class | Plotting in “Sense” and “Mask” figure | P | 02/05/24 |
| 1.4 Squirrel | Class Squirrel in animals.py |  |  |  |
| 1.3.1 Squirrel is plotted as specified colours and shape | draw() method in Squirrel class | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 1.5 ButterFly | Not Implemented | N/A | N/A | N/A |
| 1.6 Ant | Not Implemented | N/A | N/A | N/A |
| 2.0 Things |  |  |  |  |
| 2.1 Thing | Class Thing in things.py |  |  |  |
| 2.2 Door | Class Door in things.py |  |  |  |
| 2.2.1 Door has state, type attributes | state, type attributes and open(), close() methods in Door class | testThings.py: TestThings.test\_door | P | 01/05/24 |
| 2.2.2 Door is plotted as specified state | draw() methods in Door class | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 2.3 Water | Class Water in things.py |  |  |  |
| 2.3.1 Water is plotted as specified state | draw() methods in Water class | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 2.3.2 Drink water | drink\_per\_sec() methods in Water class | testThings.py: TestThings.test\_water | P | 01/05/24 |
| 2.4 Food | Class Food in things.py |  |  |  |
| 2.4.1 Food has amount, interest and need dig attributes | amount, dig, INTEREST attributes and is\_eat\_up() method in Food class | testThings.py: TestThings.test\_food | P | 01/05/24 |
| 2.4.1 Eat food | eat\_per\_sec() methods in Water class | testThings.py: TestThings.test\_food | P | 01/05/24 |
| 2.5 Bone | Class Bone in things.py |  |  |  |
| 2.5.1 Bone is plotted as specified state | draw() methods in Bone class | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 2.5.2 Bone INTEREST override | INTEREST in Bone class | testThings.py: TestThings.test\_food | P | 01/05/24 |
| 2.6 Meat | Class Meat in things.py |  |  |  |
| 2.6.1 Meat is plotted as specified state | draw() methods in Bone class | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 2.7 DogFood | Class DogFood in things.py |  |  |  |
| 2.7.1 DogFood is plotted as specified state | draw() methods in DogFood class | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 2.7.2 DogFood INTEREST override | INTEREST in DogFood class | testThings.py: TestThings.test\_food | P | 01/05/24 |
| 2.8 Toy | Class Toy in things.py |  |  |  |
| 2.8.1 Play toy | play\_per\_sec() methods in Toy class | testThings.py: TestThings.test\_toy | P | 01/05/24 |
| 2.9 Swing | Class Swing in things.py. Swing can’t be taken by the dog |  |  |  |
| 2.9.1 Swing is plotted as specified | draw() methods in Swing class | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 2.10 Ball | Class Ball in things.py. Ball can be taken by the dog |  |  |  |
| 2.10.1 Ball is plotted as specified colour | draw() methods in Ball class | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 2.10.2 Ball taken by human | taken() methods in Ball class | N/A | N/A | 03/05/24 |
| 3.0 Humans |  |  |  |  |
| 3.1 Human | Class Human in humans.py |  |  |  |
| 3.1.1 Human is plotted as specified | draw() methods in Human class | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 3.1.2 Communicate with animals | commu\_per\_sec() methods in Human class | testHumans.py: TestHumans.test\_human | P | 01/05/24 |
| 3.2 Owner | Class Owner in humans.py |  |  |  |
| 3.2.1 Owner control door | maintain\_door() methods in Owner class | N/A | N/A | 02/05/24 |
| 3.2.2 Place food | place\_food(), feed() methods in Owner class | testHumans.py: TestHumans.test\_feed | P | 01/05/24 |
| 4.0 Common |  |  |  |  |
| 4.1 Param | Class Param in common.py, defined common parameters |  |  |  |
| 4.2 Position | Class Position in common.py |  |  |  |
| 4.2.1 Flip coordinators | flip\_coords() method in Position class (COMP5005a, 2024) | N/A | N/A | 28/04/24 |
| 4.2.2 Common attributes and abstract methods | position, priority, target, posi\_history attributes and draw(), decide\_next\_step(), time\_changed() methods in Position class | testCommon.py: TestCommon.test\_position | P | 01/05/24 |
| 4.2.3 Generate mask common method | gen\_mask(), gen\_step\_mask(), get\_place\_mask() methods in Position class | testCommon.py: TestCommon.test\_gen\_step\_mask | P | 01/05/24 |
| 4.2.4 Select next step common method | gen\_random\_pos(), gen\_next\_positions(), select\_near\_pos(), get\_near\_pos() methods in Position class | testCommon.py: TestCommon.test\_near | P | 01/05/24 |
| 4.3 Map | Class Map in common.py |  |  |  |
| 4.3.1 Build map array | \_\_init\_\_() method in Map class (COMP5005b, 2024) | testAnimals.py: TestAnimals.test\_plot | P | 01/05/24 |
| 4.4 Config | Class Config in config.py, support different cfg files (Python Software Foundation, 2024b) | testConfig.py: TestConfig | P | 01/05/24 |
| 4.5 Container | Class Config in config.py | testContainer.py: TestContainer | P | 01/05/24 |
| 4.4 ArgumentParser | \_\_main\_\_ in snoo.py (Python Software Foundation, 2024a) |  | P | 01/05/24 |
| 4.4 Pause and time change | pause\_fun(),time\_passed\_hour() methods in snoo.py (The Matplotlib development team, 2023) | testContainer.py: TestContainer.test\_pause | P | 03/05/24 |
| 4.5 Custom exception | ConfigException, PositionException in exceptions.py |  |  | 01/05/24 |
| 4.6 Timetable | init\_timetable() in container.py |  |  | 01/05/24 |
| 5.0 Behaviours |  |  |  |  |
| 5.1 Owner opens doors after wakeup and closes doors before sleep | maintain\_door() in humans.py | N/A | N/A | 02/05/24 |
| 5.2 Owner feeds animals | place\_food()/feed() in humans.py | N/A | N/A | 02/05/24 |
| 5.3 Navigate the next step, targeting the destination | select\_near\_pos() in common.py | N/A | N/A | 01/05/24 |
| 5.4 Eating/drinking | Eat: eat()/Drink: drink() in animals.py | N/A | N/A | 02/05/24 |
| 5.5 Play with toy/human | Toy: play()/Human: commu() in animals.py | N/A | N/A | 03/05/24 |
| 5.6 Collision handling | Input position mask to gen\_next\_positions() in command.py | N/A | N/A | 01/05/24 |
| 5.7 Sense map setup | update\_sense\_map() in senses.py | N/A | N/A | 01/05/24 |
| 5.8 Navigate across terrain and obstacles | gen\_next\_positions() in common.py | N/A | N/A | 01/05/24 |
| 5.9 Change time | time\_fly() in container.py | N/A | N/A | 03/05/24 |
| 5.10 Interact with squirrel/butterfly | Not Implemented | N/A | N/A | N/A |
| 5.11 Protect home: prevent strange/intruder | Not Implemented | N/A | N/A | N/A |
| 5.12 Squirrel/ant stealing food | Not Implemented | N/A | N/A | N/A |
| 5.13 Dog across fence by jumping | Not Implemented | N/A | N/A | N/A |

Discussion

Some valuable designs and solutions are listed below.

1. **How to decide the next step with collision handling?**

All the items’ positions are recorded in a 2D array named “posi\_record” and shown in the “Position” figure.

Dogs, puppies, and humans have different priorities. The item with higher priority selects the next position first and updates the new position in the “posi\_record” array. Then, the item with lower priority uses the latest “posi\_record” array to decide its next position. This prevents collision.

How to calculate all possible positions for the next step？ There are some arrays used as masks to get all available positions. The first one is the area mask from the map, like the dog only positive at grass, house, and garden. The second one is related to the max step per second. If the item is at (1, 1) and the max step per second is 1 step, all available next steps are: (0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2). The last one is “posi\_record” for collision prevention.

After getting all possible positions, the next step is to find the position nearest to the target position.

1. **How to get across obstacles?**

The dogs, puppies, and even humans can’t jump across the fence. If the target position is on the other side of the fence, how to navigate it? The ideal path is to return to the house, through the house, and then navigate to the target.

From 1, the last step is getting the nearest position. This request calculates the distance from position A to position B. If positions A and B are in the same area, it’s easy to calculate the distance. However, if they are in different areas, for example, position A in the back yard and B in the front yard, The distance should be calculated as the sum of the distance from A to the back door, the distance from the back door to the front door, and the distance from the front door to position B.

1. **How to take the toy?**

When a dog plays with a toy, it can’t be taken to another position if it is a swing. But if it is a ball, the dog and the ball keep each other’s reference. The dog moves to a new position, and then the toy updates its position to the same as the dog. When the dog doesn’t take the ball, release the ball, and clear the link between the dog and the ball.

1. **How to handle the time change?**

There is a global counter to record seconds as the current time. The animal has the age attribute and overrides the parent class’s method. Time changes by 1 second, which leads to an increase in age 1. When increasing 10 minutes, reduce the energy by 1.

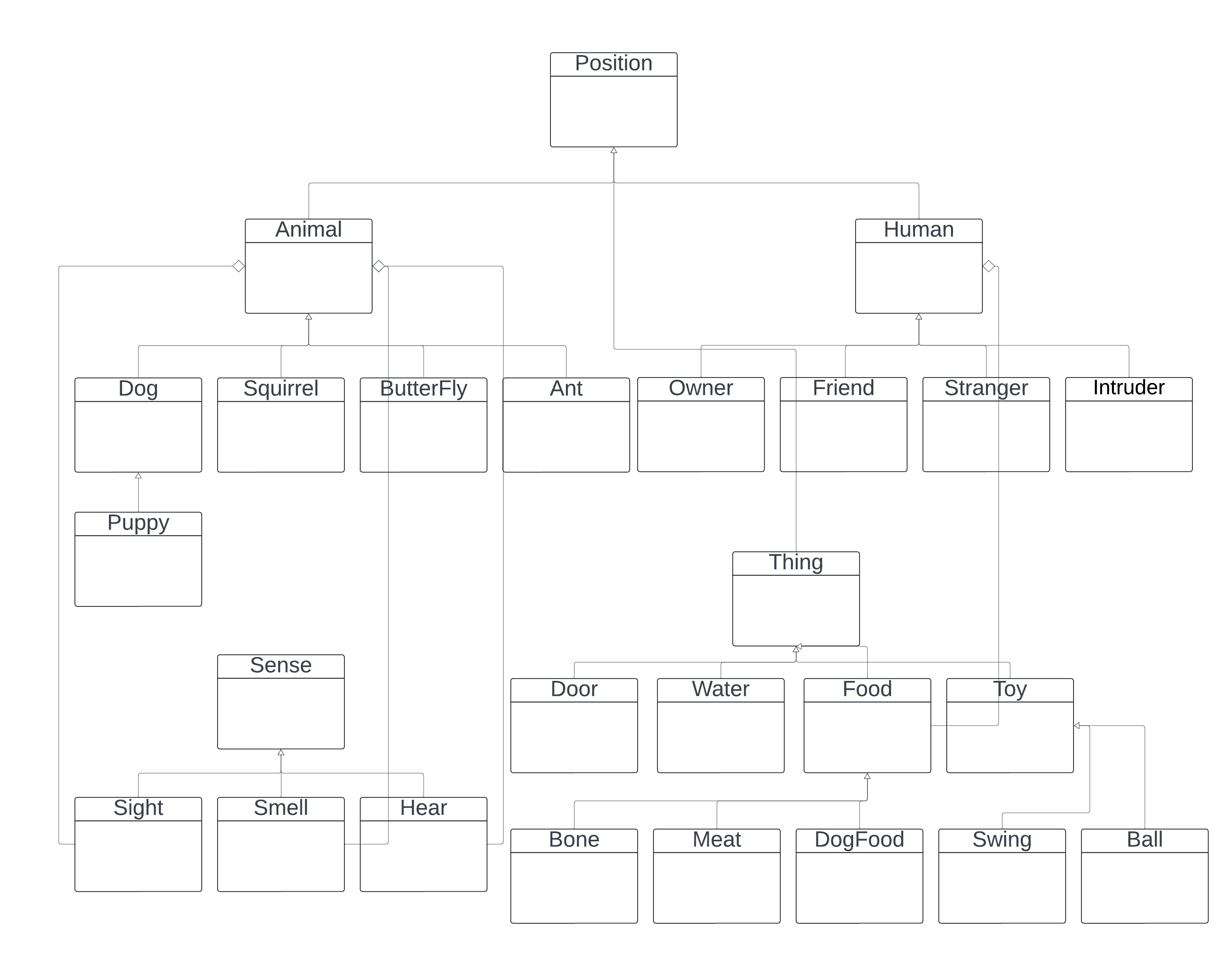
2 buttons can be used to change time by 1 hour or 1 minute. When time increases by 1 hour or 1 minute, a “time\_changed()” callback method should be called. The animal class implements this method and re-calculates the age and energy.

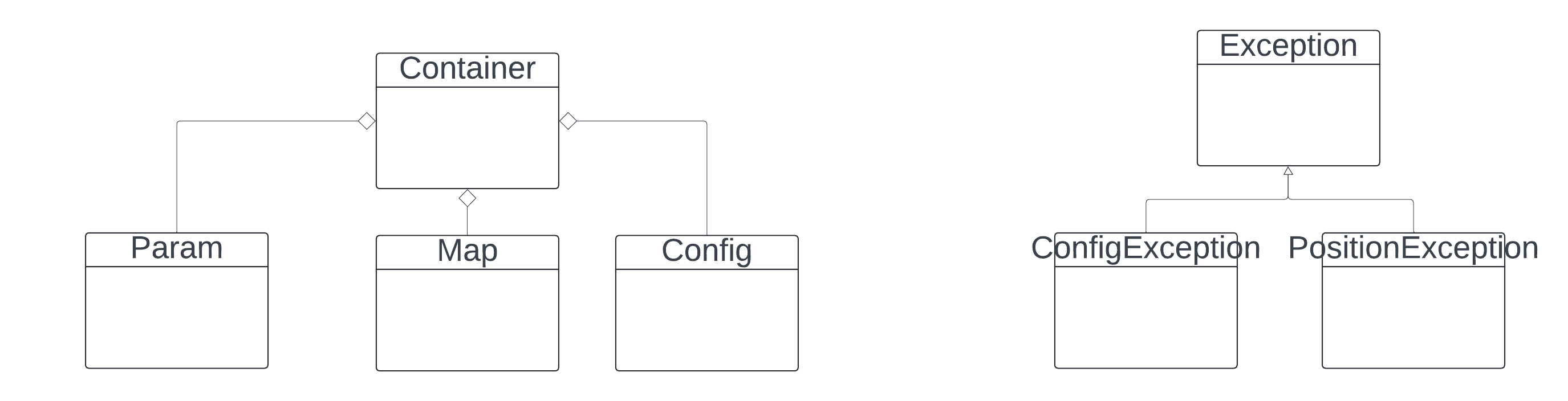
When time changes, the activities of humans and dogs change with the latest time section.

1. **How does the configuration file work?**

This program uses an INI file as a configuration file. The configuration file defines many parameters used to define current time, map size, and the numbers of animals, humans, and things. These parameters are used to build the instances. The program has preset some different configuration files to simulate different scenarios. The program defines a default configuration file and uses a command line argument to select a different one.

1. **All the objects and their relationships are listed below:**





# Showcase

The simulation is a day/night cycle. 1 day has 24 hours. These hours are divided into sections, as shown in the table below. This cycle is repeated day by day.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sleep** | **Feed** | **Play** | **Feed** | **Play** | **Sleep** |
| 0 ~ 07:59:59 | 8 ~ 09:59:59 | 10 ~ 14:59:59 | 15 ~ 16:59:59 | 17 ~ 21:59:59 | 22 ~ 23:59:59 |

During the sleep section, dogs, puppies, and the owner sleep in the house. During the feed section, the owner opens 2 doors (front and back), then goes to the grassland and places food. After the doors open, dogs and puppies smell food outside the house. After eating full, hear where the water is and drink full. During the play section, dogs and puppies “sight” where the toys are and play with the swing or take the ball together. After playing completely, hear where the owner and the friend are and play/communicate with them. After 22 o’clock, dogs and puppies return to the house and sleep. The owner also goes to sleep after closing 2 doors.

There are 4 attributes of dogs. They are shown as annotations above the picture. “H” means hungry, “T” means thirsty, “B” means bored, and “L” means lonely. The attributes’ value decreases by 10 per hour (1 per 6 minutes). When time changes, the values decrease.

## Scenario 1

**The owner feeds a dog and a puppy.**

The dogs and owner wake up at 8 o'clock. After waking up, the owner opens 2 doors, goes to the grassland and places food in the back and front yards. At the same time, the dog and the puppy wake up and go outside the house, using their sense of “smell” to find food. The food the owner places has 2 attributes: “amount” (mass) and “dig”. “Dig” means be buried, and the smell is weaker than un-dug. When a dog finds 1 dug food, it must dig it up and then eat it. The owner randomly places food in 3 types: Meat, Bone and DogFood. They have different attractions for dogs. The dog will select the most interesting food and the nearest food (not considering the amount of food now). Besides, the dog is older than the puppy, so the dog runs faster than the puppy and smells more far away than the puppy. This can be observed in figure “History”. After 1 food is eaten up, it disappears from the map. They will find more food if the “hungry” attribute does not get 100. They find water using the “hear” sense after the “hungry” attribute reaches 100. After drinking water (the “thirsty” attribute to 100), they will do the free activities. Monitor the animals’ annotation and the figure “Sense” map to check the sense of “smell” or “hear”.

How to start this scenario: run the command “python3 snoo.py -f config\_demo\_eat.cfg”.

Result: The owner places some food in the grass. The dog and puppy seek and eat food from “H: 0” to “H: 100”. Then go to the garden and drink water from “T: 0” to “T: 100”. After that, free activity.

## Scenario 2

**The dog plays with a ball and interacts with the owner.**

The play time starts at 10 o’clock. The owner opens 2 doors, goes to the grassland and places food in the back and front yards if there is no food outside. At the same time, the dog goes outside the house, using its sense of “sight” to find toys. In this scenario, only 1 ball is present outside the house. If the dog uses “sight” to find the ball, it goes to the ball and takes it to a rare position, and this action lets the “bored” attribute increase from 0 to 100. After that, the dog finds humans using the “hear” sense to find the humans and play/communicate with the owner or the friend. The owner has higher priority and could add 10 for the “lonely” attribute, but the owner is always moving. The friend could add 5 for the “lonely” attribute. After the “lonely” attribute increases to 100, the dog will do the free activities. Monitor the dog’s annotation and the figure “Sense” map to check the sense of “sight” or “hear”.

How to start this scenario: run the command “python3 snoo.py -f config\_demo\_play.cfg”.

Result: The dog seeks and plays the ball from “B: 0” to “B: 100”. Then go to find humans and play/communicate with them from “L: 0” to “L: 100”. After that, the dog will do free activities.

## Scenario 3

**Simulate the whole day for a dog and its owner.**

The simulation starts at 0 o’clock, and the dog and its owner sleep in the house. Change the time to 8 o’clock, and the owner feed the dog. After the dog’s “hungry” and “thirsty” change to 100 (eat and drink full), change the time to 10 o’clock. The dog finds the swing or the ball to play with and communicates with the owner or the friend. After the dog’s “bored” and “lonely” change to 100, change the time to 15 o’clock. The dog will eat and drink again. Then change the time to 17 o’clock, and the dog plays again. Change the time to 22 o'clock, the dog returns to the house, and the owner closes the doors. They sleep now, back to the beginning status.

How to start this scenario: run the command “python3 snoo.py -f config\_demo\_time.cfg”. Firstly, click the button “1 hour passed” continually to change the time to 8 o’clock. Secondly, click the same button continually to change the time to 10 o'clock. Thirdly, click the same button continually to change the time to 15 o'clock. Then change the time to 17 o'clock. At last, change the time to 22 o'clock.

Result: The attributes’ value decreases by 10 per hour (1 per 6 minutes). When time changes, the values decrease. The dog has 2 feeding times and playing times in a day. After 22 o’clock, the dog returns to the house and sleeps, backing to the initial status.

# Conclusion

This program implements most of the basic requirements from the specification. It involves several animals, such as the owner and the friend, and different types of food and toys. The animals can eat food, drink water, play toys and communicate with humans. The program also implements time simulation by daily cycles, like sleep, feed, and play time. However, some more complex interactions have not been implemented due to the lack of time.

# Future Work

Future work includes these items.

1. **Bug fix and optimisation.**

There are some bugs when running the program, and the navigation algorithm should be optimised.

1. **Code refactoring.**

The different priorities of activity handling.

1. **Complex interaction implementation.**

Not implement part in “Traceability Matrix”.

# References

References

COMP5005a. “Fundamentals of Programming: Prac Test 3.” Accessed April 23rd, 2024 via Blackboard

COMP5005b. “Fundamentals of Programming: Prac Test 3.” Accessed April 23rd, 2024 via Blackboard

The Matplotlib development team. 2023. Buttons - Matplotlib 3.7.0 documentation. https://matplotlib.org/3.7.0/gallery/widgets/buttons.html#sphx-glr-gallery-widgets-buttons-py

Python Software Foundation. 2024a. argparse — Parser for command-line options, arguments and sub-commands. https://docs.python.org/3/library/argparse.html

Python Software Foundation. 2024b. configparser — Configuration file parser. https://docs.python.org/3/library/configparser.html