Learning Chemotactic Strategies:

Observation and study of agent strategies in simulators

Xiang Mao
20251952

Final Year Project – 2020 – 2021
B.Sc. Single Honours in
Computer Science and Software Engineering

Department of Computer Science

Maynooth University

Maynooth, Co. Kildare

Ireland

A thesis submitted in partial fulfilment of the requirements for the B.Sc.

Single Honours in Computer Science and Software Engineering.

Supervisor: Barak Pearlmutter.

Contents

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study as part of **Bachelor of Science** qualification, is *entirely* my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

I hereby acknowledge and accept that this thesis may be distributed to future final year students, as an example of the standard expected of final year projects.

Signed: Date:

Acknowledgements

I would like to extend my sincere gratitude to my supervisor, Barak Pearlmutter, who have instructed and helped me a lot in the past. Without Professor Pearlmutter's illuminating instruction, this thesis could not have reached its present form.

I also owe a special debt of gratitude to all those who helped me during the writing of this thesis.

Abstract

Chemotaxis is a common phenomenon in natural word and is the movement of an organism in response to a chemical stimulus. For example, male moths need to trace odorant particles spread by female moths to finish its courtship. In this paper, we are about to simulate the chemotaxis of a virtual agent who utilizes different strategies to chase certain sources. Moreover, reinforcement learning will be included in the strategies. Reinforcement learning is a learning method based on rewards in which agent explores different high-reward actions while avoiding low-reward actions. In addition, reinforcement learning gets optimal policy through trial-and-error and interaction with dynamic environment. Its properties of self-improving and online learning make reinforcement learning become one of most important machine learning methods. Different from supervised learning, agent can only utilize information feedback after each action without referring to any prior knowledge, which is appropriate for the learning of chemotactic strategy in unknown environment. Proximal policy optimization is one of the most significantly important reinforcement learning algorithm, which proposes to train agent in an off-policy setting. This paper will mainly explore the reinforcement learning based chemotactic strategy to show how well reinforcement learning will perform under this scenario.

Key words: reinforcement learning, chemotaxis, proximal policy optimization

List of Figures

List of Tables

Chapter One: Introduction

Summary

This project implements a particle simulator with interactive features to observe and study the motion of agents with reinforcement learning in different situations in a simple and intuitive way.

1.1 Topic addressed in this project

1.1.1 Chemotaxis

Chemotaxis is widespread in the biological environment, and this phenomenon seems to possess significance for many organisms, and for some, it seems to directly govern their mode of locomotion.

1.1.2 Reinforcement learning

Reinforcement learning (Hu, Niu, Carrasco, Lennox, & Arvin, 2020) is a part of machine learning, but unlike supervised learning in machine learning, although reinforcement learning also requires training to obtain a model, reinforcement learning does not require a large number of labels or data with a large number of labels to be trained to obtain a model, as in supervised learning. (e.g. PPO).

1.1.3 Archimedean spiral

The Archimedean spiral is a type of spiral, and the polar equation of the Archimedean spiral in the polar coordinate system (Weisstein, 2021):

$$r = a + b\theta$$

In the Cartesian coordinate system, the parametric equation of the Archimedean spiral:

$$\begin{cases} x = (a + b\theta)\cos(\theta) \\ y = (a + b\theta)\sin(\theta) \end{cases}$$

1.1.4 Simulator Development

We have implemented a simulator with interactive features. In the particle simulator described in this paper, the particles moving in the direction of the Archimedean spiral are given an angle based on the time of their creation, which plays an important role in the coordinates (x, y) of the particles, and another factor determining the particles moving in the direction of the Archimedean spiral is the random The presence of this offset gives the particles in this helix a dispersion law similar to Brownian motion. (Feynman, 1964)

1.2 Motivation

Chemotaxis is the movement of an organism in response to a chemical stimulus. (Chisholm, 1911) Somatic cells, bacteria, and other single-cell or multicellular organisms direct their movements according to certain chemicals in their environment. This is important for bacteria to find food (e.g., glucose) by swimming toward the highest concentration of food molecules, or to flee from poisons (e.g., phenol). (Oliveira, Rosowski, & Huttenlocher, 2016) However, the chemotaxis process is not always rapid. If chemotaxis experiments are to be created in a biological laboratory, the participation of laboratory personnel is required and to perform such biological or physical experiments requires the consumption of experimental materials.

When using computers for simulation calculations, the expenses include the wear and tear of the computer hardware, as well as the cost of electricity needed to run the computer. However, both computer losses and electricity expenses are easier to quantify and control.

If a computer can be used to simulate the movement of cells, bacteria, or single-celled organisms in the environment based on certain chemicals, it is possible to try to obtain a large amount of data on chemotaxis processes and phenomena at a low cost. The whole process can be easily quantified, controlled and observed because it is carried out in a computer simulation environment. This is relevant to the study of cells as well as to the study of chemical substances and drugs.

Therefore, the above reasons constitute the main motivation for implementing a simulator that can observe the agent's motion in an environment with moving particles.

1.3 Problem statement

In order to be able to simulate the chemotaxis of organisms in a simulated environment, we need to create a simulated environment and then somehow simulate chemotaxis and somehow directly observe the process of chemotaxis in progress.

The first thing we need to address is to make the emulator work and update the latest position of the agent and particles in the emulator in real time.

In order to see more particle effects, a formula for particle motion effects is needed, which is then used to design algorithms related to particle positions. The particles that move in the direction of the Archimedean spiral in the simulator are calculated using the parametric equation of the Archimedean spiral, and then the algorithm is implemented using the parametric equation of the Archimedean spiral. Once the simulator is able to display the agent and particles properly, it is necessary to use some reinforcement learning algorithms so that the agent learning a strategy, using reinforcement learning. finally, the data generated during the running of the program also needs to be saved.

1.4 Approach

According to the definition of chemotaxis, if chemotaxis is to occur at least three conditions are needed, the first is an environment where chemotaxis occurs, the second is at least one object with the ability to move, and the third is a chemical substance.

Therefore, the simulator first needs an environment to accommodate and observe the entire process of learning chemotaxis strategies, second needs at least one agent to simulate the organism, and third needs at least one source of particle production to simulate the production and release of chemicals. On top of this, the agent needs to be equipped with a strategy to simulate the movement pattern of the organism when chemotaxis occurs.

Pygame is a Python module designed for video games. (Pygame, 2021) So it is very good to choose Pygame for implementing emulators.

PPO is one of the reinforcement learning algorithms that usually has good performance and is not particularly difficult to implement, so it is a good choice to choose PPO as a learning strategy for agent.

In order to make the simulator include other effects, so Archimedean spiral is chosen as one of the models of particle motion, and Archimedean spiral is widely available in both biological and industrial worlds. And Archimedean spiral has clear polar coordinate equations and parametric coordinate equations. The relationship between the distance r of the particle to the source and the angle of particle motion θ can be understood by the polar equation, and the relationship between the particle coordinates (x, y) and the angle of particle motion θ can be understood by the parametric equation of Archimedes' spiral, so that the coordinates of the particle can be calculated according to the time t when the particle is produced.

In order to observe Learning Chemotactic Strategies, the simulator needs to be run many times with different particle uptake rates and save important data for each run, so a save function is needed to save the data to a .CSV file.

In order to observe the difference in motion between the agent with reinforcement learning strategies and the agent without reinforcement learning, it is necessary to obtain at least data on the running conditions of the agent without reinforcement strategies, for different absorption rates. Then run and observe the agent with the reinforcement learning strategy in the simulator.

1.5 Metrics

This work requires the use of computers to simulate not only living life forms, but also non-living physical environments, and to observe and look for patterns that may be contained therein.

By comparing the data of agents without reinforcement learning strategies with the observed operation of agents with reinforcement learning strategies, it is possible to visualize whether there is a significant change in the operation of the agents with reinforcement learning strategies.

1.6 Project

1.6.1 Particle simulator with interactive features

We can interact with the simulator by clicking on the actions. We can switch between different functions by simple and easy operation. Since the way Pygame implements interaction is very different from WEB applications, the prerequisite for this feature is to figure out how Pygame draws the interaction interface in the simulator, so as long as at least one interaction function can be implemented, it means that many interaction functions can be implemented, however, in the initial stage of the simulator, such interaction operations are not available at all, so this is a significant achievement.

1.6.2 Archimedean spiral

One way of simulating molecular motion using Archimedean spirals, the most characteristic feature of this direction of motion embodied in the plane is that the difference between the maximum and minimum angles of the direction of the moving particles is very large, up to 180 degrees, which provides a less intrusive but contrasting situation for observing the specific and complex motion of agents and particles. However, in the initial stage of the simulator, the direction of particle motion was more homogeneous, and the difference between the maximum and minimum angles of particle motion direction could not greater than 90 degrees in the limited simulator display frame, and the angle was significantly smaller than that of a particle moving in an Archimedean spiral. Therefore the simulator with Archimedean spiral was used to make the particle motion pattern more abundant and was one of the results.

1.6.3 Reinforcement of learning strategies

In this simulator, we added reinforcement learning to enable agents to learn strategies through reinforcement learning. However, in the initial version of the simulator, the agent's motor rules were not equipped with any learning nature, so giving the agent access to strategies equipped with learning nature makes an essential change in the agent's motor rules and is therefore one of the results.

1.6.4 Data Storage

In this simulator, we implemented a loop to run the program multiple times and save the results of the runs. However, in the initial version of the simulator, the agent's motor rules were not equipped with any learning nature, so giving the agent access to strategies equipped with learning nature makes an essential change in the agent's motor rules and is therefore one of the results.

Chapter Two: Technical Background

Summary

We will discuss some basic knowledges about reinforcement learning. basics of reinforcement learning will be presented first along with Markov decision process and some important mathematical basis. This chapter will mainly focus on the introduction of several classic algorithms in reinforcement learning.

2.1 Topic material

2.1.1 Basics of reinforcement learning

Reinforcement learning simulates the learning process of animals which encompasses two typical features: trial-and-error exploration and delayed reward. During the learning process, previous knowledge and experiment are not provided for agent, thus, agent has to decide which actions to take to get a high reward and optimize its decision policy to maximize the reward it gets. Most of the problem in reinforcement learning are based on Markov Decision Processes (MDP).

2.1.2 Markov decision process

The Markov decision process is the main research field of sequential decision making. Markov property refers to the ineffectiveness of the state transition probability, that is, the probability of the observed environmental state at the next moment is irrelevant to the previous state before the observation, but only related to current status.

2.1.3 Reinforcement learning model and its basic elements

Reinforcement learning is based on Markov decision process in that the problem to be solved requires Markov property. In other words, during the learning process of agent, the probability of taking positive and high-reward actions will be improved constantly and contrarily, the probability of taking positive and high-reward actions will be reduced. (Watkins, 1989) The process of reinforcement learning has the following two characteristics:

(1) Feedback and reward to exploration.

(2) Agent acquires knowledge during the interaction with environment and optimizes policy to adapt to the environment.

Reinforcement learning system composes of agent and environment, which can be illustrated by figure 2-1.

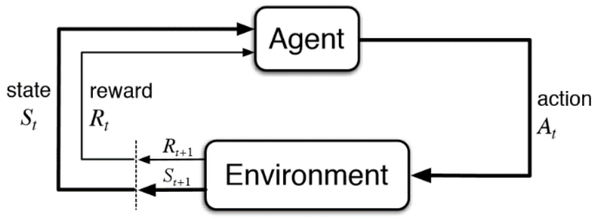


Figure 2-1: Diagram of reinforcement learning system

2.1.4 Classic algorithms in reinforcement learning

Reinforcement learning can be categorized into two main algorithms, one is called value function estimation method (such as Monte Carlo method, Q-Learning (Watkins, & Dayan, 1992), Temporal Difference (Sutton & Barto, 1998), etc), the other is called policy optimization method (Proximal Policy Optimization. (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) Moreover, we can classify reinforcement learning by whether it is based on MDP model and we will obtain two new categories called model-based and model-free respectively. Compared to model-based reinforcement learning algorithm, model-free method has a low computational cost and can better adapt to dynamic environment. Algorithms like Monte Carlo algorithm, Q-Learning, Temporal Difference(Sutton & Barto, 1998) are model-free methods.

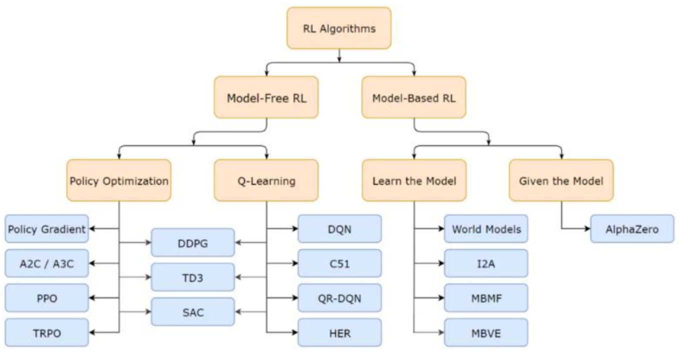


Figure 2-2 Diagram of categorization of reinforcement learning.

2.1.5 Monte Carlo method

Monte Carlo method is a model-free algorithm which solves the problem in reinforcement learning by averaging the sample reward without the need of state transfer function and reward function. By directly interacting with environment, Monte Carlo method obtains sequences of pairs structured like (State, Action, Reward) to achieve an optimal policy.

2.1.6 Temporal Difference

Temporal Difference (Sutton & Barto, 1998) was proposed to solve the time reliability allocation problem and laid an important foundation for the research of reinforcement learning algorithms. TD is a model-free algorithm. It combines MC and dynamic programming to estimate values based on different state. Different TD algorithms can be inferred from different forms of value function and amid TD algorithms, TD(0) is the simplest algorithm.

2.1.7 Q-Learning

Q-Learning (Watkins, & Dayan, 1992) is one of the most important milestones in reinforcement learning.

2.1.8 Policy gradient

Before introducing the concept of proximal policy optimization, we need to first understand policy gradient. Policy gradient is a policy-based reinforcement learning method.

Policy gradient is an on-policy reinforcement learning which indicates that the training agent and the agent interact with environment are the same. Contrarily, off-policy stresses that training an agent via data sequences sampled by another agent.

2.2 Technical material

In this paper, we need to simulate the common chemotaxis phenomena in nature and train the agent under reinforcement learning scenario. I choose Pycharm as my IDE for development for its light weight property and flexibility. In addition, to ensure the convenience of programming development, python will be the programming language in this paper.

It's worth noting that instead of building the entire reinforcement learning system from scratch, I utilize the currently most popular python toolkit for developing and comparing reinforcement learning algorithms – gym, which supports teaching agents everything from walking to playing games like Pong or Pinball. (OpenAI, 2020) Meanwhile, another important module for the experiment is called "stable_baselines", which provides set of improved implementations of reinforcement learning algorithms based on OpenAI Baselines. (OpenAI, 2020)

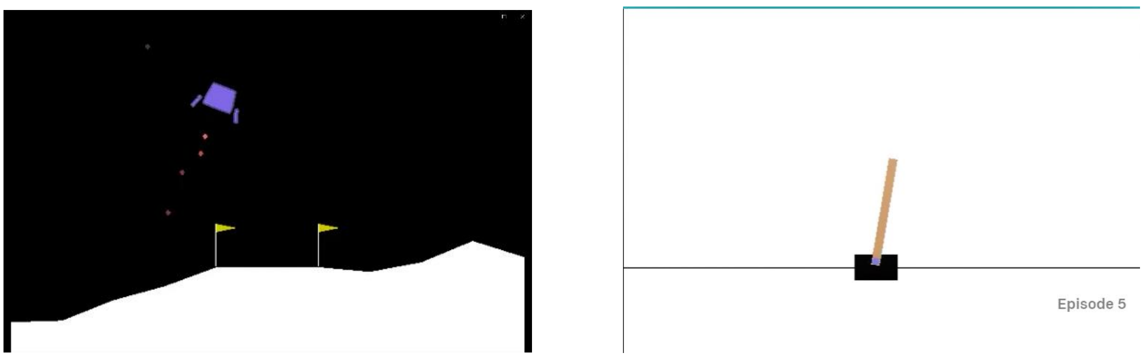


Figure 2-3: Diagram of games supported in module "gym". For example, in lunar lander (left), the agent(lunar lander) learns how to land on moon safely with its three engines on its bottom, left and right respectively. Besides, in artPole (right), agent (cart) learns to balance the pole standing on it, preventing it from falling over.

I consider two different scenarios in total: the particle source can emit odorant particles either in linear mode or Archimedean spiral mode. The linear mode can be illustrated by figure 4-2 and the Archimedean spiral mode is shown below.

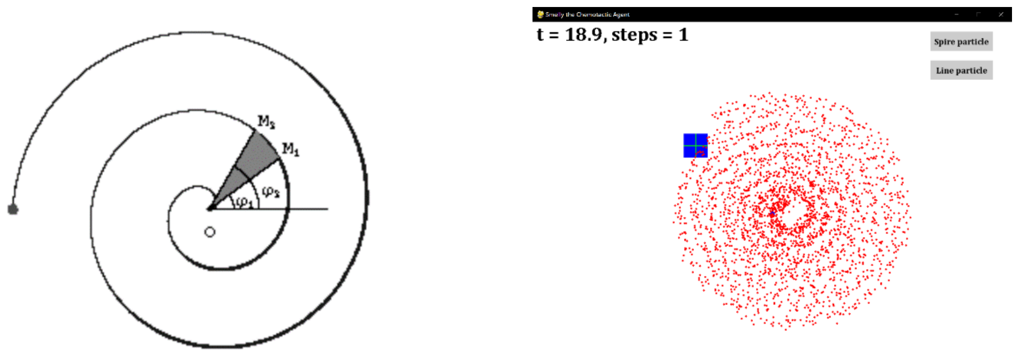


Figure 2-4: Diagram of Archimedean spiral (left) and source that emitting particles in a way of Archimedean spiral mode (right).

Following what has been mentioned above, the agent will also take two different kinds of strategies to track odorant source, the first one is a basic but simple method which considers the number of odorant particles in the vicinity of agent, the other is proximal policy optimization in the setting of reinforcement learning.

Although basic strategy is convenient and easy to implement, it is expected that PPO algorithm will outperform basic chemotactic strategy in the aspect of stability and robustness.

Table 2-1 and table 2-2 respectively show major modules and important API used in this experiment.

Table 2-1 Major modules used in the experiment

Modules used in the experiment	Brief introduction
pygame	Cross platform python module designed for game design.

numpy	Python module supports efficient matrix operation and mathematical calculation
gym	a toolkit for developing and comparing reinforcement learning algorithms.
stable_lines	Python module contains a set of improved reinforcement learning algorithms based on OpenAI baselines.
matplotlib.pyplot	A state-based interface to matplotlib. It provides a MATLAB-like way of plotting.

Table 2-2 Important API used in the experiment

API used in the experiment	Brief introduction
pygame. init()	Initialize all imported pygame modules and runtime environment
pygame.display.set_caption(str)	Set the caption of running window. It takes one str type input which is the expected text
pygame.draw.circle(surface, color, center, radius)	Draw a rectangle on the given surface. Parameter “surface” indicates surface to draw on, “color” denotes color to draw with, center and radius represent the “center” and “radius” of the circle to be drawn.
pygame.draw.rect(surface, color, rect)	Draw a rectangle on the given surface. Parameter “surface” indicates surface to draw on, “color” denotes “color” to draw with, “rect” is the rectangle to draw.
pygame.mouse.get_pos()	get the mouse cursor position.

pygame.mouse.get_pressed()	get the state of the mouse buttons.
stable_baselines.PPO2(policy, env, gamma=0.99, n_steps=128, ent_coef=0.01, learning_rate=0.00025, vf_coef=0.5, max_grad_norm=0.5,)	Create a PPO2 model. Parameter "policy" is the policy model to use, "env" is the environment to learn from (gym in this project), "gamma" it the discount factor, "n_steps" is the number of steps to run for each environment per update, "ent_coef" is the entropy coefficient for the loss calculation, "learning_rate" is just as its name implies, "vf_coef" denotes value function coefficient for the loss calculation and "max_grad_norm" represents the maximum value for the gradient clipping.
gym.spaces.box.Box(low, high, shape)	a Box represents the Cartesian product of n closed intervals. Parameters "low" and "high" represent the bound for each dimension. "shape" can be determined by "low" and "high"

Chapter Three: The Problem

Summary

From a code perspective, the simulator is composed of 2 classes and several global variables and many global methods needed to implement the simulation of learning convergence strategies and the simulation of particle motion.

3.1 Project UML documentation

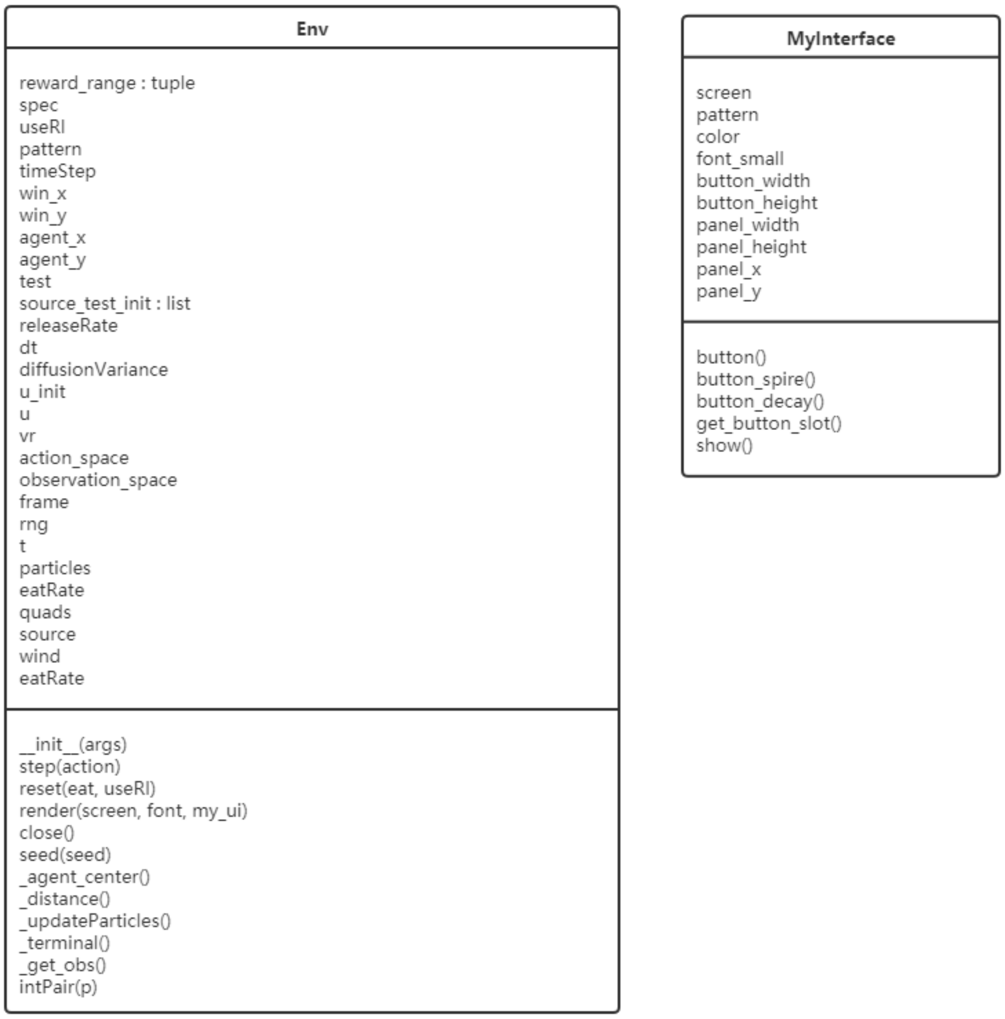


Figure 3-1: Diagram of Project UML.

3.2 Problem analysis

3.2.1 Demand-side problem analysis

This simulator needs to simulate the process of learning Chemotaxis strategies, and all the outputs obtained by computer simulation of this process are in the storage medium of the computer, so the content generated by the simulation process kind needs to be saved and visualized. Since it is possible to compare the motion process of the agent by comparing different scenes, the simulation environment needs to be controlled and modified during the observation process, so the simulator needs to be interactive and we can change the simulator through interactive actions like clicking on the simulator.

3.2.2 Technical-side problem analysis

Pygame is a way to implement an emulator. Pygame has been around for many years and has extensive technical documentation, but it is very different from a WEB application in terms of implementing interactive effects, so the need to implement interactive effects requires a complete understanding of how to make the images in the Pygame emulator screen receive interactive information. One possible way to do this is to determine the coordinates of the mouse and mouse click events in each frame that the mouse moves through the emulator, thus calculating whether an area of the screen has been clicked or not.

Saving data through files is very important. The way the save file function is implemented can affect the efficiency of running the program, because the data in the computer's hard disk and the computer's memory are accessed at different rates, and it would consume a lot of time to save every piece of data to the computer's hard disk while running the program in a loop. If the computer do not store the data on the computer's hard disk during the process of running the program in a loop but wait until the program is finished and then store all the data on the hard disk, then the program does not have to consume this amount of time during the process of running.

Reinforcement learning is a way to equip an agent with the ability to learning chemotactic strategies. If the reinforcement learning we use is not complex, then we only need to complete a small program with the most basic reinforcement learning capability first, and then gradually overlay more functions on this small program to make it gradually become a simulator.

Python programs do not run as efficiently as C programs, but if we develop in C, we will encounter different problems from two different programming languages, which will greatly affect the development schedule. If we only use C to complete part of the core code, the number of C-related problems is much lower, although the program may still encounter problems from both languages as it is being developed. However, even this does not necessarily mean that the program runs fast enough, so using C does not necessarily save this amount of time; Python has access to a very large number of libraries that C does not have, so Python can be developed more efficiently than C. So the advantage of choosing Python over C is that the total time can be reduced by using Python's various libraries, but not necessarily increased by not using C.

Chapter Four: The Solution

Summary

Chapter four will discuss the existing problem of basic strategy in that it may incurs a situation where the agent wanders around or move far from the particle source. Moreover, some non-negligible details of the implementation are presented

4.1 Analytical Work

In the beginning of the experiment, the agent adopts the basic strategy that counts the number of odorant particles in its four directions: upper left, upper right, lower left, lower right and chooses the direction that contains most of the particles. This can be illustrated by figure 4-1.

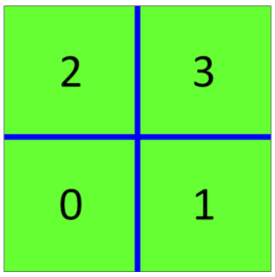
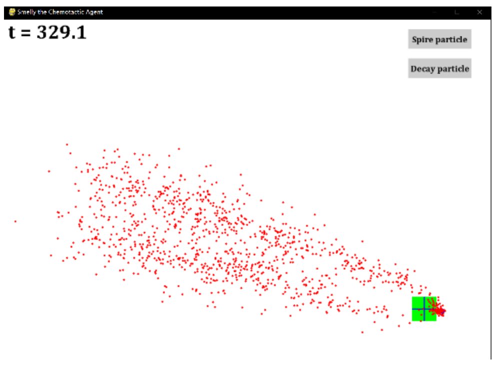


Figure 4-1: Diagram of Agent Structure.

However during the experiment, I observed that in some circustance the agent cannot move towards source but rather being pushed away from it. Analytically, it roots in the tracking strategy always pick the direction that contains more ordorant particles. Therefore, agent wanders around when the particles are not absorbed in time and particles in four directions are nearly of the same number.



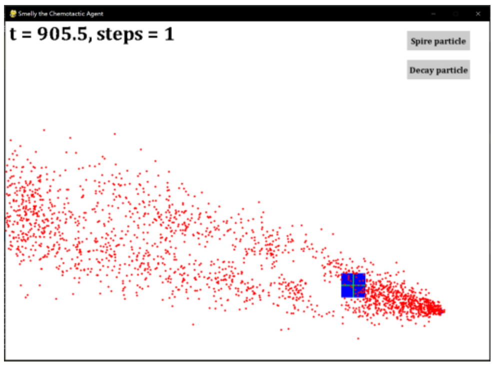


Figure 4-2: Diagram of agent's tracking strategy (left). the green box represents the neighbour of agent and the black arrow indicates the expected direction that the agent should move towards. Agent wanders around(right).

To solve this problem, we may adjust the absorb rate or use a mechanism like momentum that cons-trains the direction changes at each step. However, these modifications still cannot essentially tackle the problem where we should consider another chemotaxis strategy. Hence, I choose to implement PPO algo-rithm to make the procedure more stable and robust, although it adds to some affordably computational expense.

4.2 Code details

We now discuss some important code details of experiment in section 4.3. To begin with, some environment setting will be introduced following the explanation of agent and strategy or model, in other words.

4.2.1 Environment [Archimedean spiral] and agent

The simulating environment and agent are implemented based on the popular module "gym". First we need to create our environment class inherit from "gym.Env" which will contains all the parameters to control the entire training procedure. (e.g. time step, window size, agent size and action space, etc) Most importantly, we need to reimplement member methods which includes "step", "reset", "render", "close". For example, method "reset" indicates how will the environment be reset.

```
def reset(self, eat=0, useRl=True):
      self.useRl = useRl     # whether adopt reinforcement strategy or not
      self.particles = []    # clear all the particles
      self.t = 0.0            # reset time
      self.eatRate = eat     # reset absorbing rate
      # randomly generate the position of odorant particle source
      self.source = [ np.random.rand()*self.win_x,
      np.random.rand()*self.win_y]
```

4.2.2 Model

Due to the convenience provided by python, we only need to create a model based on PPO with the methods encapsulated in module "stable_baselines" and simply call its member function "learn" to start training.

```
model=PPO2('MlpPolicy', env=env, seed=0, verbose=2, gamma=0.9,
learning_rate=3e-4, nminibatches=4, n_steps=1024*2, cliprange_vf=-1,
max_grad_norm=0.5, tensorboard_log='log/ppo2')
model.learn(total_timesteps=30_0000, reset_num_timesteps=False)
```

Chapter Five: Chapter five: Evaluation

Summary

In this chapter, experiment results including figures will be presented to illustrate the effectiveness of reinforcement learning strategy. We will first present the diagrams of the training procedure and afterwards we will compare the two different strategies in time expense and success rate respectively.

5.1 Results

5.1.1 Explanation of Results

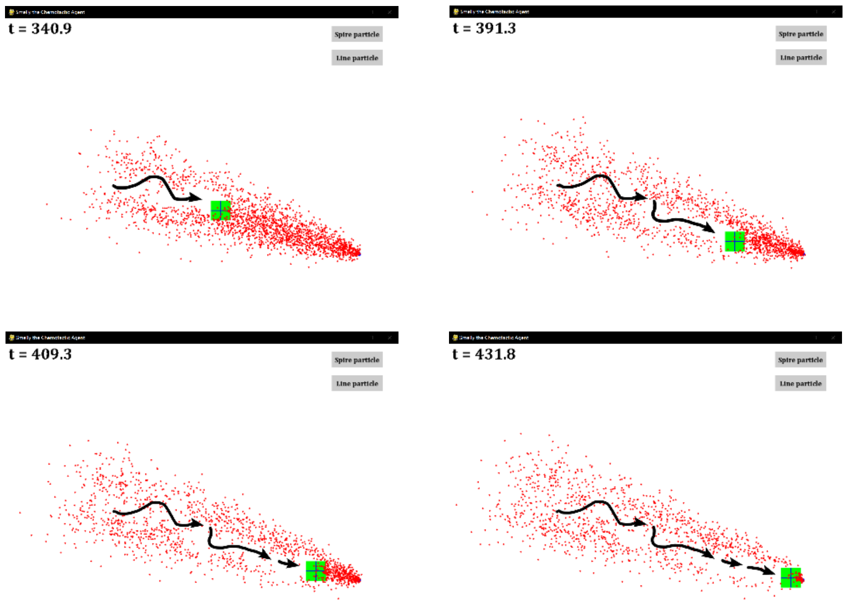


Figure 5-1: Diagram of agent stabilized close to the particle source

5.1.2 **Analysis of Results**

The result above shows reinforcement learning is highly effective and stable in multiple situations, compared with basic strategy in source tracking. It is noticeable that, by finetuning the parameters, we can achieve an even better result.

5.2 Comparison

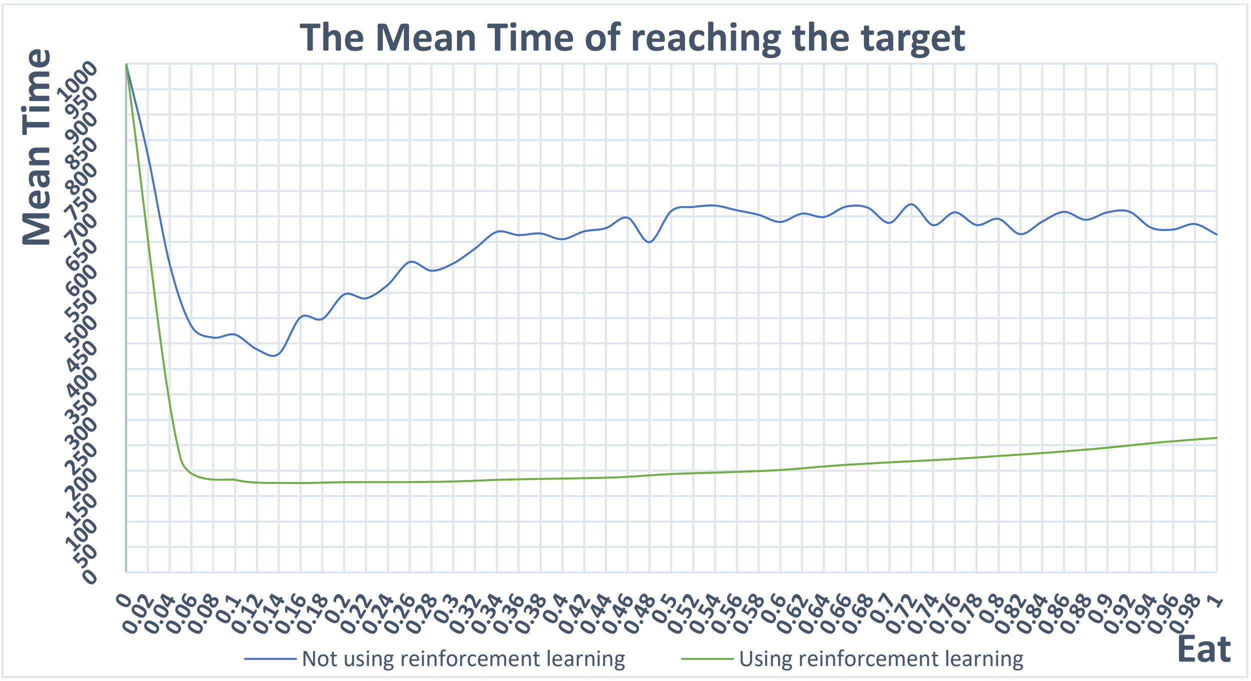


Figure 5-2: Diagram of Average time taken to reach the source without and with reinforcement learning strategies.

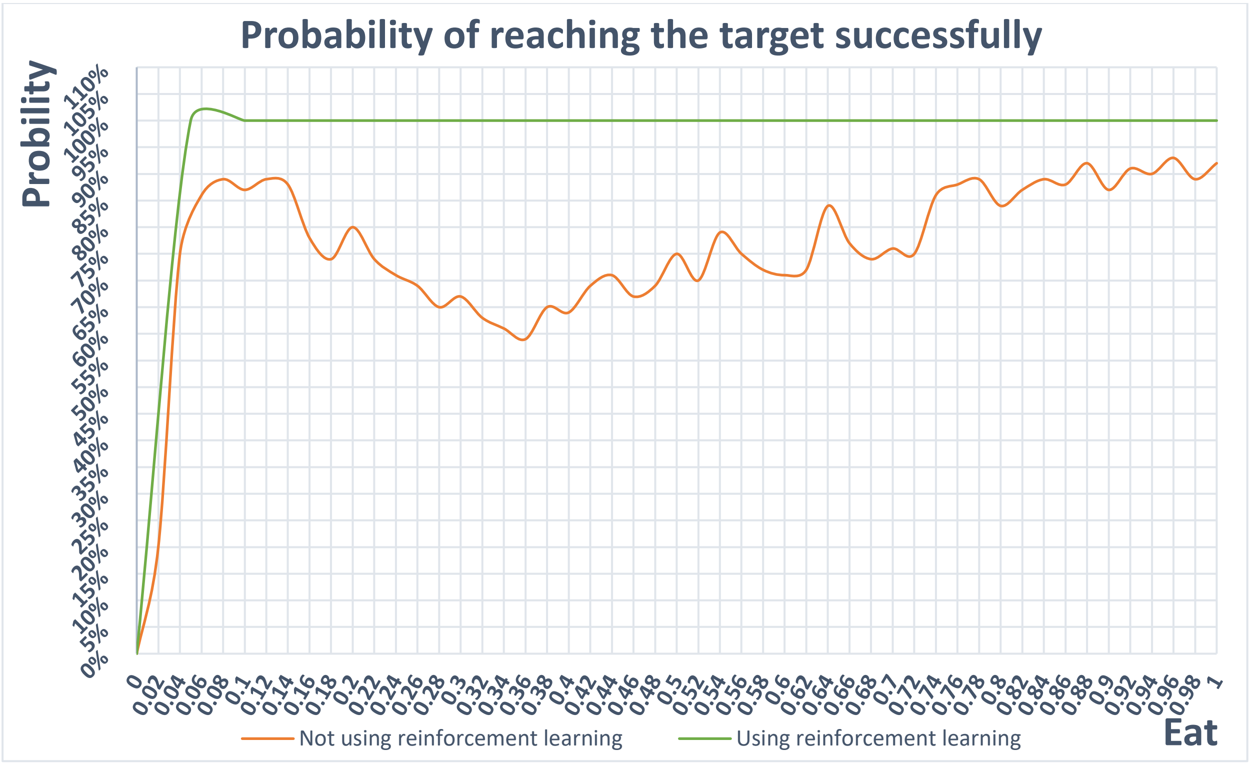


Figure 5-3: Diagram of Success rate of reaching the source without and with reinforcement learning strategies.

It is obvious that reinforcement learning strategy greatly outperforms basic method, considering time expense and success rate.

Chapter Six: Conclusion

Summary

A summarization will be given in this chapter to conclude some pros and cons of reinforcement learning according to the experiment. Besides, some interesting directions based on this work are proposed for future work.

6.1 Results discussion

In the simulator as an important strategy for observational learning of Chemotaxis strategies in agents, reinforcement learning has a clear performance. So we can learn from the experiment results shown in chapter 4 that the strong practicability of reinforcement learning. So the learning Chemotaxis strategy of using reinforcement learning to simulate living organisms on a computer has effect.

However, there are several drawbacks that exhibited during the training procedure. Hard-to-train is the most obvious phenomena that I observed in experiments.

In addition, It is also very difficult to obtain statistical analysis of the operation of a large number of simulations when the agent takes different eats or other parameters, because it requires not only that the program is not faulty, but also that the hardware running the program is good enough.

Maybe, the curse of dimensionality(Bellman, 1957) limits reinforcement learning heavily for real physical system. In this paper, agent only needs to choose from four directions to move towards.

6.2 Future Work

Until now, we have implemented reinforcement learning strategy to help agent tracking the odorant particle source. But we train our agent merely with PPO algorithm, thus we may endow agent with more strategies to learn and compare in-between various reinforcement learning methods. Meanwhile, the experiment setting is relatively simple, so we can make things more complicated. For example, the agent will be a virtual moth that it needs to control its flying attitude while searching for the odorant particle source in a more complicated environment in which there are obstacles and predators.

References

Bellman, R. E. (1957). Dynamic programming. Princeton University Press. p. ix. ISBN 978-0-691-07951-6. Retrieved March 29, 2021 from https://books.google.com/books?id=wdtoPwAACAAJ

Chisholm, H. (1911). Chemotaxis. In H. Chisholm (Ed.), *British Encyclopaedia: Volume 06.djvu/90* (p. 77). Cambridge University Press. Retrieved March 28, 2021, from https://en.wikisource.org/wiki/Page%3AEB1911_-_Volume_06.djvu/90

Hu, J., Niu, H., Carrasco, J., Lennox, B., & Arvin F., (2020). Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning, *IEEE Transactions on Vehicular Technology: Volume 69: Issue* (pp. 14413 - 14423). IEEE. Retrieved March 28, 2021, from https://ieeexplore.ieee.org/abstract/document/9244647

Oliveira, S., Rosowski, E. E., & Huttenlocher, A. (2016). Neutrophil migration in infection and wound repair: going forward in reverse. *Nature Reviews. Immunology*. 16 (6): 378–91. doi:10.1038/nri.2016.49. PMC 5367630. PMID 27231052. Retrieved March 28, 2021, from https://www.nature.com/articles/nri.2016.49.pdf?origin=ppub

OpenAI, (2021). *Gym: A toolkit for developing and comparing reinforcement learning algorithms*. Retrieved March 28, 2020, from https://gym.openai.com/

OpenAI, (2021). *Openai/baselines*. Retrieved March 28, 2021, from https://github.com/openai/baselines

Sutton, RS. & Barto, AG. (1998). *Reinforcement Learning*. MIT Press. ISBN 978-0-585-02445-5. Archived from the original on 30-03-2017. Retrieved March 28, 2021, from https://login.cs.utexas.edu/sites/default/files/legacy_files/research/documents/1%20intro%20up%20to%20RL%3ATD.pdf

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*, Cornell University. Retrieved March 28, 2021, from https://arxiv.org/abs/1707.06347

Watkins, C., & Dayan, P. (1992). *Practical Issues in Temporal Difference Learning*. Boston: Kluwer Academic Publishers. Retrieved March 28, 2021, from https://link.springer.com/content/pdf/10.1007/BF00992698.pdf

Weisstein, E. (2021), Wolfram Research. Wolfram Research. Retrieved March 29, 2021 from https://mathworld.wolfram.com/ArchimedesSpiral.html

Feynman, R. (1964). The Brownian Movement. *The Feynman Lectures of Physics, Volume I.* (pp. 41–1). Retrieved March 29, 2021 from https://www.feynmanlectures.caltech.edu/I_41.html

Pygame (2021), About – pygame wiki. Retrieved March 29, 2021 from https://www.pygame.org/wiki/about

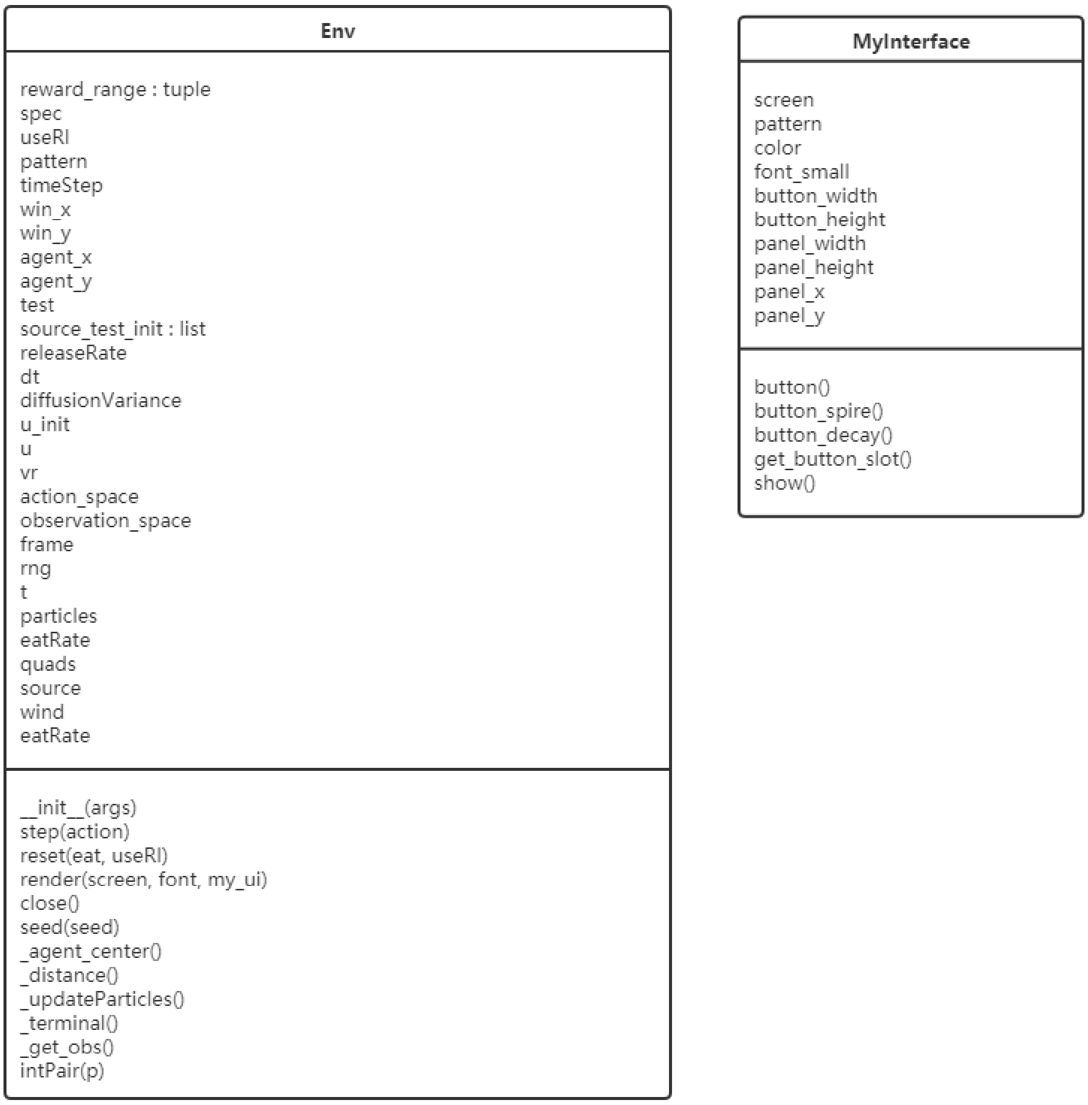
Appendices

6.3 Appendix 1 Code developed for this project.

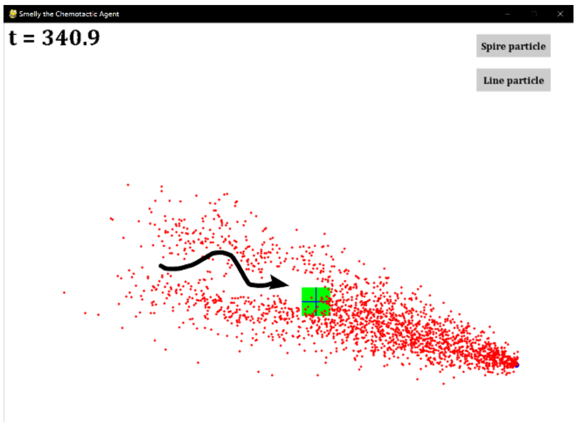
```
def reset(self, eat=0, useRl=True):
      self.useRl = useRl     # whether adopt reinforcement strategy or not
      self.particles = []    # clear all the particles
      self.t = 0.0            # reset time
      self.eatRate = eat     # reset absorbing rate
      # randomly generate the position of odorant particle source
      self.source = [ np.random.rand()*self.win_x,
      np.random.rand()*self.win_y]
```

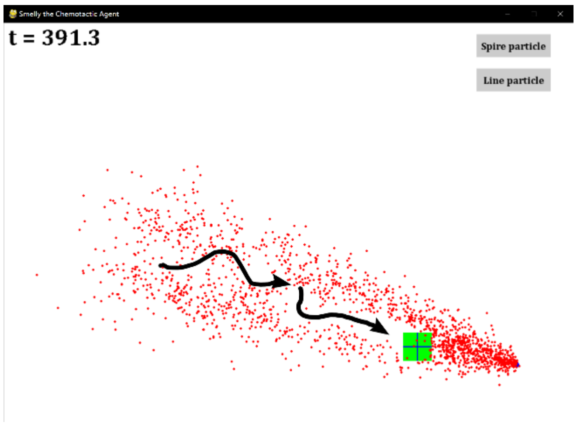
```
model=PPO2('MlpPolicy', env=env, seed=0, verbose=2, gamma=0.9,
learning_rate=3e-4, nminibatches=4, n_steps=1024*2, cliprange_vf=-1,
max_grad_norm=0.5, tensorboard_log='log/ppo2')
model.learn(total_timesteps=30_0000, reset_num_timesteps=False)
```

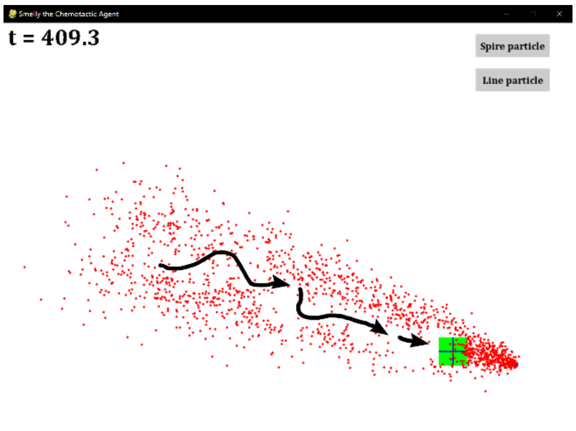
6.4 Appendix 2 UML Class for this project.

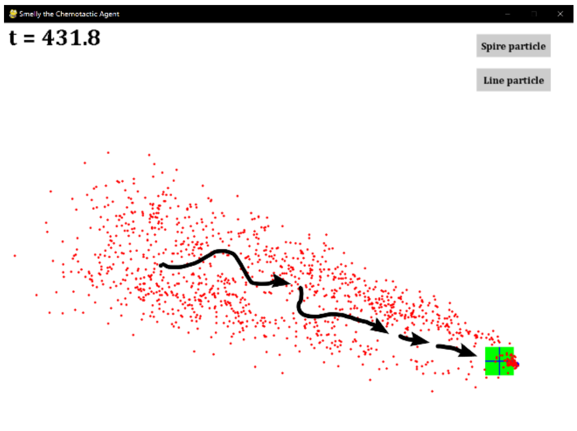


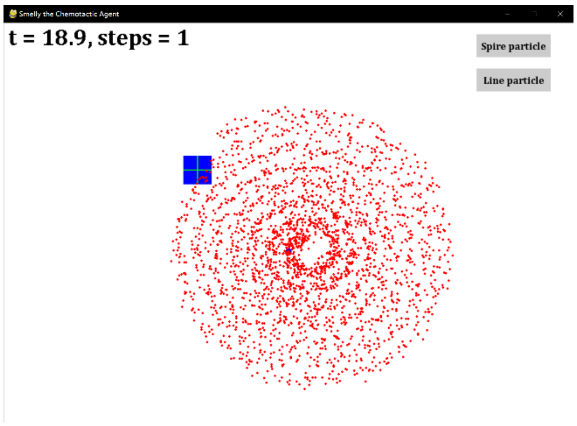
6.5 Appendix 3 Screen shots of the project implementation


Smelly the Chemotactic Agent
t = 340.9
Spire particle
Line particle


Smelly the Chemotactic Agent
t = 391.3
Spire particle
Line particle


Smelly the Chemotactic Agent
t = 409.3
Spire particle
Line particle


Smelly the Chemotactic Agent
t = 431.8
Spire particle
Line particle


Smelly the Chemotactic Agent
t = 18.9, steps = 1
Spire particle
Line particle