SWEN90004 Modelling Complex Software Systems Assignment2 Report

SaiEr Ding(1011802)
Lin Fan (945510)
Jinxin Hu (963171)

Word Count:

1 Overview

The model of Rebellion is a classic social science problem adapted from Joshua Epstein's civil violence model proposed by 2002. The main idea is analysing the behaviours of a group of subjugated populations against the central authority and generating patterns based on the experiments.

There are two types of people in this model, cops, the police offers and agents, the rebels. The trigger for agents to openly rebel depends on the value of grievance and the cost in jail, when the grievance is large enough and the cost of being jailed is small enough, the rebellion happens. Simultaneously, the cops represent the central government who works on quelling the rebellion effectively, they wander around randomly and arrest people who are actively rebelling.

2 Design of the Netlogo Model

In this model, there are two separate threads respectively representing 'cops' and 'agents'. At the set-up stage, they are both assigned to each patch of the map according to the 'initial-density' randomly.

- **Agents**
  - 'Move' action for agents limits 'agents' can only enter an empty patch where the candidate is empty or contains only jailed agents in vision.
  - 'Grievance' is influenced by 'perceived-hardship' (whether 'agents' accept the current government) and 'government-legitimacy'.
  - 'The cost of being jailed' is influenced by 'risk-aversion' (the degree that 'agent' are reluctant to rebel because fearing of being arrested) and 'estimated-arrest-probability' (determined by the number of 'cops' and the number of active 'agents' in vision).
  - 'Active' action means the 'Agents' are rebelling against the government and the prerequisite for rebellion is the difference between 'grievance' and 'the cost of being jailed' less than a minimum threshold.
- **Cops**
  - 'Move' action for cops is almost similar to 'agents'. While when the global variable 'movement' is set to 'off', 'agents' cannot move but 'cops' still can do it.
  - 'Enforce' action for cops means when there exists an active agent, 'cops' will arrest it and send it into jail with a random jail-term which is less than max-jail-term.

For the move states, the agent and cop are assigned with the random location initially and move in the next state to the other available patch within the visible range. In each move, the field of the patch is limited to one, and only unoccupied patches are valid for moving.

Apart from move state, the cops can also change to arrest state, once the active agents appear in the vision, the arrest state for cops will be triggered and one active agent will be arrested randomly to the jail.

The states of the agent's behaviours can be divided into three variables:

- **active:** The "active" agents in the group of active rebels which has not been arrested yet.
  - The state will be changed to "jailed" once s/he appears on the cops' vision.
  - The state will be changed to 'quiet" if the agents start to calm down and the balance between grievance and cost of being jailed becomes stable.

$$grievance - riskAversion \times estimatedArrestProbability \leq threshold \tag{1}$$

- **quiet:** The "quiet" agents represent the stable people who remain quiet without "active" on the rebellion. The "quiet" state changes to "active" once the balance between grievance and cost of being jailed has been broken and exceeds a certain threshold.

$$grievance - riskAversion \times estimatedArrestProbability > threshold \tag{2}$$

$$(grievance = perceivedHardship \times (1 - governmentLegitimacy)) \tag{3}$$

- **jailed:** The "jailed" state grouped the agents who had been already arrested, and the "jailed" state will be finally changed to "quiet" after passing through the maximum jailed period.

3 Design of the Python Model

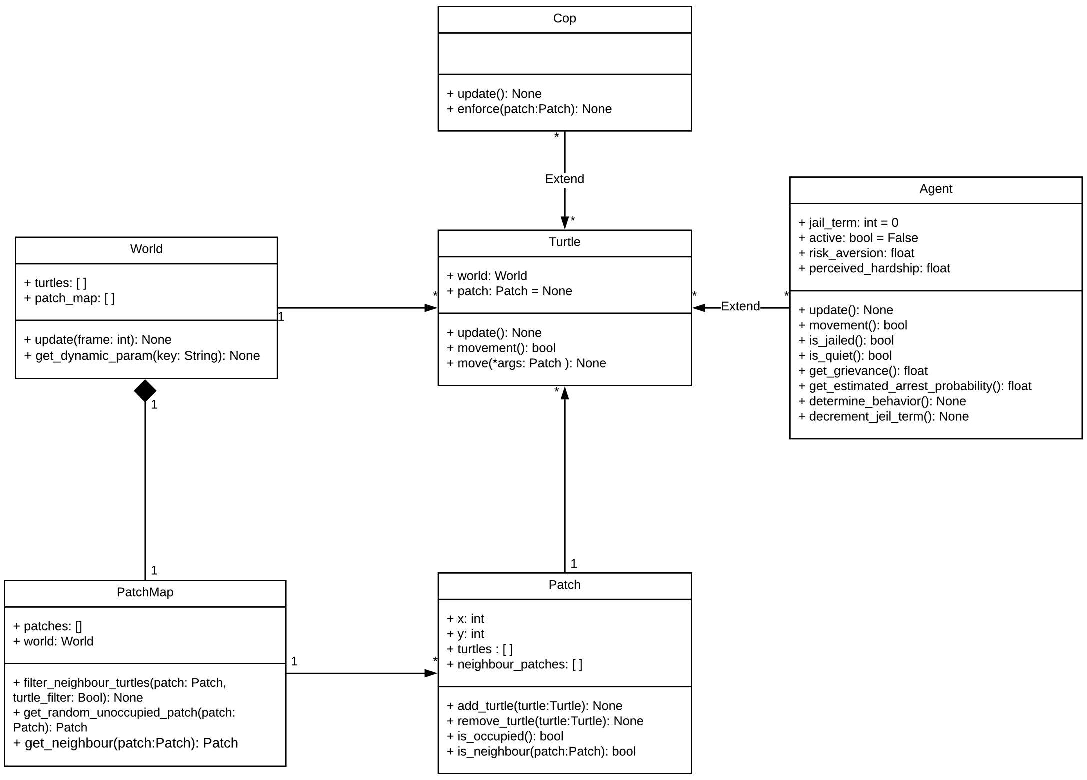
Cop
+ update(): None
+ enforce(patch:Patch): None
Extend
Agent
+ jail_term: int = 0
+ active: bool = False
+ risk_aversion: float
+ perceived_hardship: float
+ update(): None
+ movement(): bool
+ is_jailed(): bool
+ is_quiet(): bool
+ get_grievance(): float
+ get_estimated_arrest_probability(): float
+ determine_behavior(): None
+ decrement_jeil_term(): None
World
+ turtles: [ ]
+ patch_map: [ ]
+ update(frame: int): None
+ get_dynamic_param(key: String): None
Turtle
+ world: World
+ patch: Patch = None
+ update(): None
+ movement(): bool
+ move(*args: Patch): None
Extend
PatchMap
+ patches: []
+ world: World
+ filter_neighbour_turtles(patch: Patch, turtle_filter: Bool): None
+ get_random_unoccupied_patch(patch: Patch): Patch
+ get_neighbour(patch:Patch): Patch
Patch
+ x: int
+ y: int
+ turtles : [ ]
+ neighbour_patches: [ ]
+ add_turtle(turtle:Turtle): None
+ remove_turtle(turtle:Turtle): None
+ is_occupied(): bool
+ is_neighbour(patch:Patch): bool

Figure 1: UML Diagram

- **World:** The world-class is the world of the whole map. For each time tick, the world updates the action of turtles.
- **Patch:** Patch class corresponding to the single patch in the map, a turtle can move a patch and signal as an occupied patch by modifying the method in the patch class. Turtles can also detect the neighbour target by accessing through the turtles' list nearby stored in the Patch class.
- **PatchMap:** The PatchMap class represent the whole patch map in the world, by accessing through it, the neighbour patch can be found and the random unoccupied patch can also be detected.
- **Turtle:** Turtle is a super-class for all movable objects in the map, all turtles on the map can move randomly and they can also be restricted the movement if necessary. The subclass for Turtle includes Agent and Cop.

- **Agent:** Agent is one of the child classes for Turtle, the “jail_term” has been initialised as 0 and the status of the agent begins with “quiet”. The agents can only change actions when the “jail_term” is zero.
- **Cop:** The other child class Cop only contains two methods, “move” and “enforce”, the cop can randomly arrest one active agent after moving to the surrounding area of agents.

4 Experiments

4.1 Default Parameters

initial cop density	initial agent density	vision	government legitimacy	max jail term
4%	70%	7	0.82	30

Table 1: Default Parameters Table

The default parameters on Netlogo have been listed on Table 1, to compare the model behaviours between Netlogo and Python, the team has set all the parametera same initially in the Python Design and following all strategies when implementing the model. From Figure 2 and Figure 3, one can see that after 200 ticks run, the Python model present a similar pattern and behave the same trends as the NetLogo model, because of the randomly setting for ’risk_aversion’ and ’perceived_hardship’, the minor different for the patterns are acceptable. Therefore, the team would assume that the replication is successful and complete.

To examine the models in detail and investigate the effects of parameters tuning on model behaviour, the team has designed two experiments in the following sections.

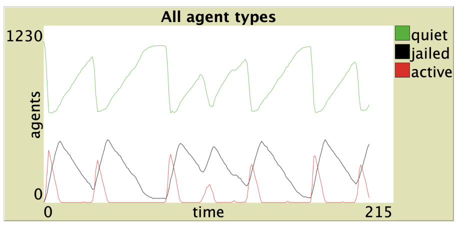

Figure 2: Default Parameters for Netlogo Model

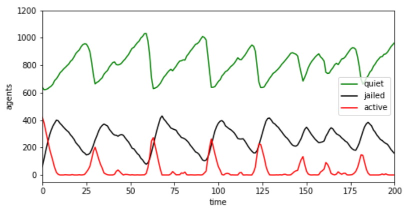

Figure 3: Default Parameters for Python Model

4.2 Hypothesis 1

If the ‘government-legitimacy’ is low enough, the number of jailed agents will be increased and the number of quiet agent will be increased.

- **Parameter setting:** Turn the value of ‘government-legitimacy’ from 0.82 to 0.5 than 0.2.
- **Result:** The number of ‘agents’ in each type. (‘quiet’, ‘jailed’, ‘active’)

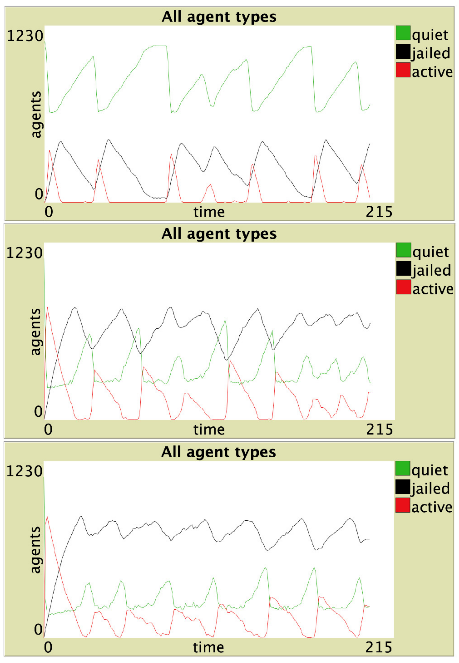

Figure 4: Result from Netlogo Pattern

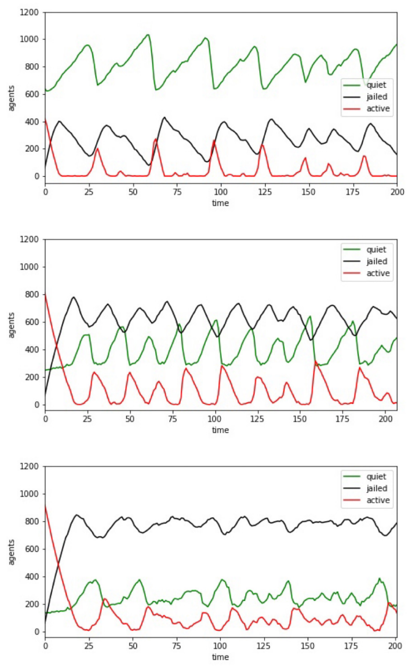

Figure 5: Result from Python Pattern

- **Analysis:** From Figure 4 and Figure 5, one can summarize that the number of quiet agents stay at a high level at the beginning and start to decrease rapidly. In contrast, the number of jailed agents experience quick growth and finally stay steadily.

  From Equation (2) and (3) on section 2, the trigger for becoming active agent from quiet mode depends on the level of grievance and the cost of being jailed, and the level of grievance’ is clearly affected by the value of ’perceived_hardship’ and the ’government_legitimacy’. As the random ’perceived_hardship’ is generated uniformly between 0 to 1, the decrease of ’government_legitimacy’ will cause the raising of grievance, which means the quiet threshold will be exceeded easily with more quiet agents turning to active mode.

However, the cop is busy with arresting active agents simultaneously, once more quiet agents become active, more active nearby the cops and are arrested immediately. Therefore, the number of jailed agents increase and the active agent is always staying in a stable trend.

4.3 Hypothesis 2

Is the increase of cop density affect the number of active agents.

- **Parameter setting:** Turn the value of 'initial-cop-density' from 2% to 4% than 8%.
- **Result:** The number of 'agents' in each type. ('quiet', 'jailed', 'active')

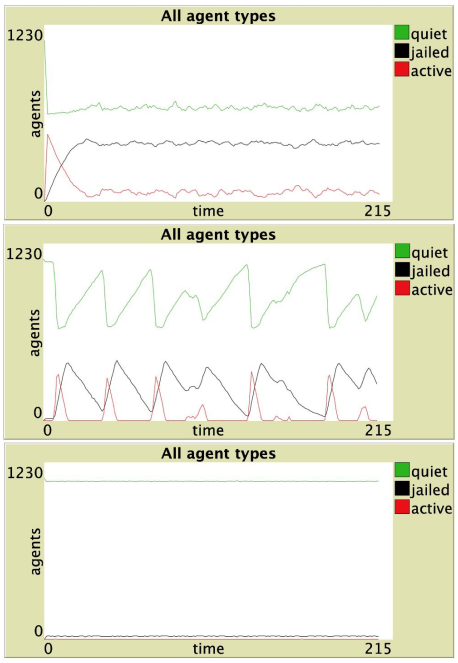

Figure 6: Result from Netlogo Pattern

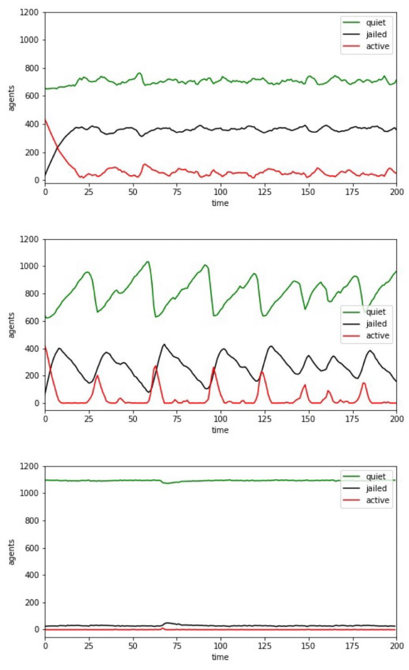

Figure 7: Result from Python Pattern

- **Analysis:** From the Figure 6 and Figure 7, the team observe that both NetLogo and Python model behaves similar by tuning the parameter 'initial-cop-density', the manner difference is caused by the randomly selection of 'risk_aversion' and 'perceived_hardship'.

As the growth of the cop density, the number of active agents experiences a fluctuation and eventually reduce to low levels. This is because the increase of cops raises the total vision area on the map, and the active agents nearby become more likely to be arrested. Since the transformation from quiet agents to active agents depends on the amount of cops and active agents nearby, once the density of cop has been increased, the chances for transfer to active agents from quiet will be decreased.

5 Extension

5.1 Extension Design

By considering how the rebellion may affect the behaviours of other residences and what would be the worst result, the team has designed an approach for assuming that the perceived hardship of a person may be influenced by others nearby. Once a movable agent (active/quiet) is surrounded by any active agent, the value of 'perceived_hardship' for her/him will be increased. When the 'perceived_hardship' exceeds a certain threshold, (0.8 in the implementation) this agent will become a killing machine and randomly kill any innocent agent around her/him. However, no criminal lives with impunity, after arrest by the cop, the killer will be given no chance to be acquitted and will stay in jail for the rest of her/his life.

5.2 Experiment

- **Parameter setting:** Turn the value of 'dangerous_threshold' from 0.8 to 0.7 and 0.6.
- **Result:** The number of 'agents' in each type. ('quiet', 'quiet_alive', 'killed', 'jailed', 'active')

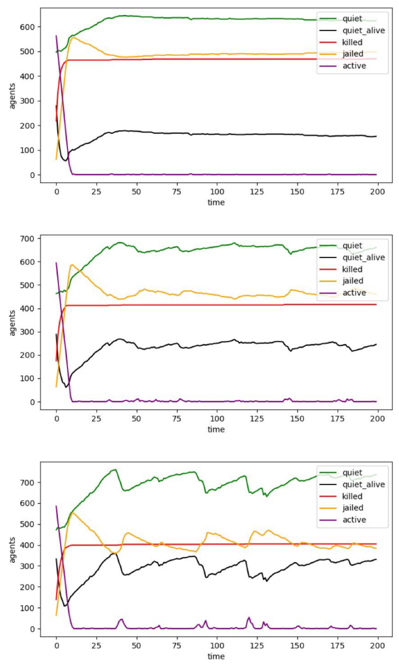

Figure 8: Result from Extension Pattern

- **Analysis:** From Figure 7 we can find that the number of active agents drops rapidly with the number of killed agents and jailed agents sharply increase at the beginning stage. At the same time, the number of quiet and alive agents declines at first and then increases at the same pace with quiet agents. In terms of the death ratio, we can infer from the results of these experiments that higher dangerous threshold will lead to a lower death ratio of quiet agents, which is because the higher the threshold is, the less active agents are over-dangerous. As for the mutual impact of perceived hardship, we can see the fluctuations in the Figure 7 that the most obvious one is that with the lowest dangerous threshold. With more quiet agents alive, they are more likely to affect their perceived hardship with each other, which leads to the fluctuation as the result.

Appendices

A Team Successes

In conclusion, all team members have complete the assigned tasks with quality and quantity, thank you for everyone's contribution. By gathering the ideas of experiments and extensions, the team held three meetings via Zoom and some excellent discussion has been made. To well collaborate with others, some collaboration tools has been successfully used included Lucidtchart for drawing the UML diagram, Github for controlling the version of code, Google Doc and Overleaf for sharing the report writing.

B Team Challenges

The time for replicate the original model with NetLogo has taken longer time than expect, the starting date listed on the table for experiments has to been shifted from 22nd of May to 24th of May. However, by the hard work of all team members, all the implementation work has been finally finished on time (26th May) without anymore delay.

11st May	Proposal Submission.
18th May	Model Implementation.
22nd May	Experiments.
24th May	Model Extension.
27th May	Report Writing.
29th May	Report Submission.

Table 2: Original Timeline

References

[1] Wilensky, U. (2004). NetLogo Rebellion model. http://ccl.northwestern.edu/netlogo/models/Rebellion. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[2] Wilensky, U. (1999). NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.