

Project report: Bee simulation

Name: Frank Zhao



May 17, 2025

student number: 22225924

Curtin University: COMP1005

# Overview

This Project implemented a time simulation of a honeybee colony in 2 modes: interactive mode and Batch mode.

Interactive mode ( beeworld\_interactive.py) launches a command window where the user can input values and choosing different seasons to dictate the bee’s behaviours and its success in gathering nectars. Secondly, it launches a Matplotlib window where the user is able to control one wasp with the arrow keys and watches bees forage, deposit honey, fill up the combs and avoid water, houses, and trees.

In Batch mode ( beeworld\_batchmode.py) runs the same world headless, driven by two CSV files ( world.csv terrain, params.csv parameters). It logs nectars collected and live bee count each step and can then be export results to stats.csv for analysis.

Key features of this coding assignment are worker bees, Queen, Flowers, Comb, Terrain, Wasp. They all carry different responsibilities and behaviours. Worker bees wander around, detect flowers, collect nectar (muj), return to hive, deposit into the nearest empty comb from the entrance, then they experience a life cycle in the simulation like age and die. Queen( only in interactive mode) spawns a new worker every 30 time steps in the simulation. Flowers have finite nectars and can only respawn during the run on grass only, a bonus feature or an easter egg is that there is a 15% chance to be golden which carries double the nectar. There are 3 hexagonal shaped cells, which are combs that have a capacity of 5 units each comb ( total maximum nectar is 15). Terrain in the simulation has a code of 10,3,0,15 and grass, tree, water, house respectively, only 0,3,15 (water,tree,house) are treated as obstacles(genes). Wasp is user-controlled in interactive mode; disabled in batch mode to avoid wiping the colony instantly and restore game balance for the user.

User Guide

Minimum version of required packages include python 3.9+ , NumPy 1.20 and matplotlib 3.4.

1. sudo apt-get install python3 python3-pip

2. pip3 install NumPy

3. pip3 install matplotlib are the codes required to install the respective packages

these packages are required to install on Linux(Curtin) or alternative clients like PyCharm or visual studio code. No external libraries are required, matplotlib handles all the drawings.

Here’s a folder layout on how to navigate the files

* Assignment\_22225924
* beeworld\_interactive.py - GUI/user-controlled wasp version
* beeworld\_batchmode.py - headless, csv driven version
* worker.py – worker bee class
* Queenbee.py – Queen bee class
* Flowers.py – Flower class
* Comb.py – comb/honey-cell class
* Wasp.py – wasp class
* Plot.py – all drawing helpers
* World.csv – 40x30 terrain grid ( 10 = grass, 0 = water, 3= tree, 15 = house)
* Params.csv – default batch parameters( steps, bees, flowers, etc)
* Stats.csv – generated by batch runs ( this can be deleted)
* Project\_report\_22225924.docx-final report

To navigate beeworld\_interactive.py you will be required to enter python3 beeworld\_interactive.py and then you will be prompted for “ timesteps” indicating the maximum frames to run the simulations, ‘number of worker bees’ indicates initial bees spawned in the hive, ‘summer or winter’, summer means more flowers and faster respawn, and winter means less flowers and slower respawn rate. Each are given valid ranges, where timesteps can only range from 50 to 500, number of workers range from 1 to 20, whereas for seasons there’s only two options, summer and winter. If given out of range values, it will pop up errors like "Error: Enter an integer between 1 and 20." For seasons it will be "Error: Type 'summer' or 'winter'.". Secondly, during the run the user can control the wasps ( marked with red X) moving one square at a time controlled by the arrow keys (up, down, left, right); however, the wasp cannot enter water, trees or the house which are considered obstacles in the map. Any bee within the radius of 0.5 on the grid is eliminated. The window automatically closes when all three combs reach 5 units of nectars ( 15 in total), timestep limit is reached or all the bees are killed by the wasps.

To navigate the batch mode, the user needs to use two CSV files in the assignment\_22225924 directory, or optionally ( not recommended if the user doesn’t know the code that well) create their own two CSV files to run the Batch mode. World.csv consists of 40 rows x 30 columns of integers. 10 = grass ( walkable), 3 = tree, 0 = water, 15 = house, this is already supplied in the directory. Params.csv has a pair ( one per line) consists of a variable and a value. For example, steps = 500, number of bees = 19, season = summer, spawn flower probability = 0.07. Finally, run the Batch mode by inserting this line: python3 beeworld\_batchmode.py\--field world.csv \ --params params.csv\--csv stat.csv\. the line –csv stats.csv is optional unless the user wants to see the values of each variables during each time step in the total of 500 timesteps.

You will be given the following output: “Batch finished – 500 steps; nectar collected: 15 ; bees alive:12 stats saved to stats.csv”. if requested the stats.csv has three columns consists of steps, nectar and bees alive that goes from 0 time steps to 500 steps, and showing the values of the number of bees and nectars collected during this 500 timesteps.

In Linux, open the terminal, and type “ cd ~/bee-world” or wherever the user unzipped the files. Verify the relevant packages are there like, python, NumPy and matplotlib. Then run the batch driver with the two CSV files, “ python3 beeworld\_batchmode.p\ - - field world.csv \ - - params params.csv \ - - csv stats.csv “. - - field ( or -f) points to the terrain grid ( 40x30), - - params ( or -p) points to the parameter file ( steps, bees, etc). after the run, the user will see something like “ Batch finished – 500 steps; nectar collected: 15 ; bees alive: 12 stats saved to stats.csv”.

Alternatively, in PyCharm, the user run it by clicking the edit configurations ( in the three circles drop downs) and then select script. Script consists of beeworld\_interactive.py or beeworlld\_batchmode.py. script parameter(batch only); however, need to enter “ - -field world.csv - -params params.csv - -csv stats.csv”. then run the program clicking the green triangle.

In terms of troubleshooting, if “FileNotFoundError: world.csv” pops up, put world.csv in the same working directory or give full path with - -field. Secondly, if window opens but bees never move, as in not collecting honey, then the problem is likely to be terrain grid is all non-grass; so, ensure to make grass= 10 and obstacles as 0/3/15 only. Thirdly, if the error “ PermissionError saving stats.csv” pops up then close the file in Excel or choose a new output name, generally it is common that users will accidentally open the excel while running the batch mode closing it is recommended. This guide is for users who operate on Curtin’s Linux lab machines and in PyCharm on Window or Mac.

Traceability Matrix

Table giving overview of features, and the implementation and testing of your code. Example below.

* **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 bee is finite | Worker.step\_change()  Lines 40-150 | Live run: bees collect nectars, age, rest, nectar history increases with every deposit | p | 10/5/2025 |
| 1.1 Worker ageing and death | Timesclicking() in same worker class | Batch mode runs 500 steps bee count in alive\_log bees die after reaching max\_age | p | 10/5/2025 |
| 1.2 Queens spawns a new bee every 30 timesteps ( interactive only) | QueenBee.step\_change() | In interactive mode: bee list length jumps by +1 in every 30 frames | P | 12/5/2024 |
| 1.3 Flower objects: golden=double nectar and capacity | Flower.\_\_init\_\_, collect\_nectar() | Unit-check: golden flower returns 2 nectars per call; primemuj = 6vs3 ( max capacity of the nectars in golden flower is 6, normal is 3) | P | 12/5/2025 |
| 1.4 Flower regrowth after some time | Flower.step\_change() | Batch mode: nectar level of a depleted flower rises again after 5-10 ticks | p | 12/5/2025 |
| 1.5 three combs, capacity 5 each | Comb list and comb.addrawhoney() | Batch runs reaches nectar\_log 15, prints victory message, then stops the simulation | P | 16/5/2025 |
| 2.0 wasp kills bees within 0.5 radius | Wasp.eliminate\_bees() | When bees in the range the bee count drops indicating killing the bee | P | 16/5/2025 |
| 2.1 Obstacle set same in both Batch and interactive modes | Genes = { r, c) for r in range(rows)  for c in range(cols)  if humanity[r, c] in (0, 3, 15) } in both modes | Flowers, trees, bees, wasps will never be on the obstacles | p | 17/5/2025 |
| 2.2 Plot alignment ( for example no white bands appearing ) | Plot\_world() extent and limit | Image and scatter plot perfectly align and overlap on one and another | p | 17/5/2025 |
| 2.3 Interactive input validations for ranges and strings | Three while loops at top of file so that the user enter the correct or valid values until its correct and break the loop | Entered invalid values= program re-prompt with error messages | p | 18/5/2025 |
| 2.4 user- controlled wasps ( arrow keys) with obstacle checks | On\_key() handler | Press arrows so that the red X moves but cannot cross the obstacles | P | 18/5/2025 |
| 2.5 graph: total number of bees spawning + killed | Total\_bees\_log.append(len(beetlehuice)), and axes[2].plot(total\_bees\_log,label=”Total bees”) | Line rises overtime in interactive mode. | p | 18/5/2025 |

Discussion

Discussion of implemented features (referring to the Traceability Matrix), explaining how they work and how you implemented them. A UML Class Diagram should be included for objects and their relationships.

This section reflects on the implemented features and how they were designed, structured, and verified across both interactive and batch modes. Each feature corresponds with the number listed in the Traceability matrix.

worker is finite (1.0) – bees switch between moving, collecting nectar, returning to the hive, depositing honey and resting. These behaviours are managed by flags like hasmuj, inhoneyhold, and depositing inside the step\_change() method. The design makes each state clear and easy to test. It is confirmed by observing and watching the bee collecting nectar and checking if the nectar log increased.

Worker aging and death(1.1) – each bee lives for 400-600 ticks, it tracked with max\_age. The \_tick\_age() method increases age and kills the bee once it reaches the limit. When batch mode runs, bee count dropping slowly as bee aged, this is the same as interactive, but more obvious in the batch mode. This is evident by stats.csv.

Queen spawns new bees(1.2) – the queen creates new bee every 30 frames using a cool down timer. This only happens in interactive mode, specifically, inside the QueenBee.step\_change() function. This is tested by printing the bee list length after every 30 frames.

Golden flower ( 1.3) -Golden flowers have more nectars than regular ones which gives 2 nectars per collections instead of 1 by the bees. A flag called golden is used in the Flower class to enable this. This is tested by spawning flowers and checking the return value from collect\_nectar().

Flower regrowth (1.4) – After a flower is empty, it waits 5-10 frames before regrowing 1 unit of nectar. This is handled in step\_changes() in flower.py file. It is confirmed by logging flower nectar values over time empty flowers slowly refilled as expected.

Three combs, 15 capacity in total (1.5) – the stimulation uses three comb objects, each holding 5 units of nectars. When a bee deposits, addrawhoney() is called if the comb is full , it is marked with fullrawhoneyy = true. When all combs are full, the game ends. This was tested by running batch and watching it stop when nectar reached exactly 15

Wasp eliminates bee within range (2.0) – in interactive mode, the wasp is collected with arrows keys. When it moves , it checks if any bee is within 0.5 squares. If it is then the bee is marked dead. This was tested by moving the red wasp icon towards a group of bees and watching their numbers drop.

Same obstacles in both modes (2.1) – both batch and interactive modes use the same rule for obstacles: if a terrain square is 0( water ), 3(tree), or 15(house), its blocked . these valued go into genes set. Bees, flowers and wasps all check genes before moving or spawning. This keeps behavious consistent between both programs.

Plot alignment ( 2.2) – the program uses extent = [-0.5,col-0.5,-0.5, row-0.5] to make sure images and scatter points line up in the world view ( second graph). This removed earlier issues where bees looked like they were on water even though they weren’t. the fix was confirmed visually.

Input validation( interactive) (2.3) – the program uses while loops to make sure user inputs are valid. If the user types an invalid number or an incorrect season, the program shows an error and asks again. This was tested by typing wrong inputs like “ abc” or “autumn” and confirming that the correct message appeared.

Wasp movement Is blocked by the obstacles (2.4) – the wasps movement verifies against the gene set. It only moves if the next square is grass. This was tested by trying to move into water or the house; the red X stopped at the edge and didn’t enter.

Total bees created in the graph ( 2.5) – a third graph was added in interactive mode that tracks the total number of bees that have existed. This is useful in runs with an active queen, where new bees are created in every 30 frames. The count increases until users control the wasps to kill the bees. This is confirmed and tested visually with experimenting if the line goes down or not after the wasp kills and the bees.

In summary, all features were tested through live runs and logs. The code was written to make each feature testable and consistent across both batch and interactive modes. Obstacles rule, nectar collection, ageing, and termination all work the same way in both of the versions. This made the simulation easier to debug and more predictable.

A diagram of a work flow

AI-generated content may be incorrect. UML diagram

# Showcase

The following three scenarios demonstrate how the bee simulation behaves under different conditions. Each one modifies key inputs such as flower count, spawn rate, season, or user interaction to highlight different outcome. The terrain grid world.csv is kept constant, and each run is executed either in batch mode ( with params.csv) or interactively with prompts. Outputs such as nectar collected, number of bees alive, and whether the hive is filled are recorded using live graph s or stat.csv. These results are compared to evaluate how the system responds to resource availability and user strategy.

## Scenario 1: summer, balanced set up ( default), wasp idle

User input: timestep: 400, number of worker bees: 15, season: summer

With 40 flowers at the start and a high respawn rate ( 0.07), the colony collected nectar very efficiently. The queen spawned 2 extra bees before filling up all the combs. Nectars reached 15 by 66th frame, trigger mission complete. All 15 original bees survived, plus 2 more.

Key takeaway: this shows the baseline system behaviour with enough resources and no user interference, the bees complete their task smoothly. The graphs clearly showed nectar rising, bee count stable, and total bee count increasing

## Scenario 2: winter, wasp steered aggressively

User input: timestep: 400, number of worker bees: 15, season: winter

Player actively steered the red wasp icon into the flower-rich zones. Any bee within radius 0.5 was eliminated on contact. Winter starts with only 20 flowers at the start and a low spawn rate of 0.02, making nectar harder to find. 7 bees were eliminated by the wasp, and then the queen spawned 4 replacements before dying. The hive was filled very late around 387th frame, and only 6 bees were alive at the end.

Key takeaway: this scenario shows how user interference can hurt a colonies survival. Although the wasp didn’t prevent success, but it delayed the progress and lowered survivability of the bees. It also highlights the importance of a Queen bee to keep supplying new bees to prevent a total failure.

## Scenario 3: small colony challenge, wasp idle

User input: timestep: 500, number of worker bees: 5, season: summer

The simulation began with only 5 bees, but the queen immediately started spawning new bees every 30 frames. By frame 112, the hive reached full capacity with 15 nectar units, and the simulation ended in victory. By that point, a total of 9 bees had been created. The original bees were still alive, and the system never stalled despite the small starting colony.

Key takeaways: this scenario shows that even a small colony can still complete its goal quickly with abundance of resources. Additionally, due to its efficient foraging and the queen’s ability to grow the colony. It confirms that the system works reliably even when starting with a low population, and that performance isn’t limited by the number of initial bees.

# Conclusion

Reflection on your assignment with respect to the specification

This assignment provided an opportunity to design and implement a simulation that models the behaviour of aa bee colony in both interactive and batch mode. All required features from the specification were implemented successfully, including finite state logic for worker bees, obstacle- aware movement, flower spawning and regrowth, comb filling, and automatic simulation termination once the hive is full.

The simulation was designed to be consistent and testable across both modes. Through sharing a common logic such as obstacle checking system and comb structure, the program avoided duplication and ensure that behaviour remains predictable. Bonus features like user-controlled wasp, golden flowers and a statistical graph that tracks nectars and bees were introduced in a modular way that preserves the integrity of the base system.

Interactive mode allowed for real time observation and strategy, while batch mode supported controlled testing using CSV input and output. Scenario testing confirmed that the simulation responds to different inputs, including resource shortage, reduced initial bee populations and ultimately user interference.

In reflection to this project, a key strength of the project was its flexible structure, which allows new features to be added without compromising its functionality. Overall, this assignment helped reinforce key programming concepts such as object- orientation design, simulation loops, input validations, and real- world modelling using codes.

# Future Work

Further investigations and/or extensions that could follow.

Possible extension may be to include Dijkstra’s pathfinding, as this would allow the bees to navigate more efficiently and allow the wasps to catch the bees more easily. Second extension may be adding a survival mechanism for the bees, as they prioritize its life compared to getting nectars. An example of this may be, if a wasp Is in a bee’s range, then the bee will run away from the wasp until it is out of range and giving up on collecting flowers. Third extension can be creating multiple beehive that compete in terms of resources. Lastly, a cool extension to add is multiplayer where two players control rival colonies that compete with the same resources.

# References

*Use Chicago 17th Author-Date, refs in alphabetical order*

Beekman, Madeleine, and Francis L. W. Ratnieks. 2000. “Long-Range Foraging by the Honey-Bee, *Apis mellifera* L.” *Functional Ecology* 14 (4): 490–496.  
<https://doi.org/10.1046/j.1365-2435.2000.00443.x>

Dyer, Fred C. 2002. “The Biology of the Dance Language.” *Annual Review of Entomology* 47: 917–949.  
<https://doi.org/10.1146/annurev.ento.47.091201.145306>

Matplotlib Developers. 2024. *Matplotlib: Python Plotting Library*.  
<https://matplotlib.org/>

Cappa, Federica, and Alessandro Cini. 2022. “Predatory Wasps and Their Impact on Honey Bees (*Apis mellifera*).” *Insects* 13 (2): 117.  
<https://doi.org/10.3390/insects13020117>

Seeley, Thomas D. 1985. *Honeybee Ecology: A Study of Adaptation in Social Life*. Princeton, NJ: Princeton University Press.  
<https://press.princeton.edu/books/hardcover/9780691083873/honeybee-ecology>

PlantUML. 2024. *PlantUML: Open-Source UML Tool*.  
<https://plantuml.com/>.