FOP_Assignment_22881119

Project Report : Untitled Bee Simulation

Name: Thomas Han

Student Number: 22881119

Date: 12/5/2025

Overview

The program aims to create a simplified simulation of creatures which mimic bee movement in a 2d top down environment. The project uses simple behaviour rules to create complex behaviours for bees.

User Guide

Required Packages:

- Python 3.x
- pygame # pygame is the primary library used for the simulation engine
- numpy # this is for np's useful array features and maths
- noise import pnoise2 # map generation
- argparse # parsing arguments
- random, math, sys, time

You can install the packages by running:

```
$ pip install -r requirements.txt
```

or

```
$ pip install pygame numpy noise
```

How to Run:

Basic Simulation:

```
$ python main.py
```

Interactive Mode (prompts for parameters):

```
$ python main.py -i
```

With a Specific Seed:

```
$ python main.py -s 12345
```

Batch Mode (using predefined map and parameter files):

```
$ python main.py -b -f <map_filename.csv> -p <parameter_filename.csv>
```

Example:

```
$ python main.py -b -f scenario_map.csv -p scenario_params.csv -s <seed>
```

Note: For batch mode, you need to provide CSV files for the map and parameters as described in the main project report.

Key Controls (During Simulation)

Pan Camera: Move mouse to screen edges.

Zoom In: `=` or `+` key.

Zoom Out: `-` key.

Select Entity: Left-click on a bee or hive to view its details.

Traceability Matrix

Feature	Code Reference	Test Reference	Status	Date Completed
1.0 Basic Setup	**Overall main.py structure, Pygame initialization**	**Program runs without immediate errors.**	P	18/4/25
1.1 Map	**Environment class, MAP_SIZE constant**			
1.1.1 Perlin Noise Generation	Environment.generate_background_texture(), Environment.get_tile_color(), pnoise2 import and usage	Visual inspection of map variety; Color mapping logic in get_tile_color().	P	18/4/25
1.1.2 Camera Zoom	Simulation.scaled, main() event loop (K_EQUALS, K_MINUS), gridpos2screen()	Tested +/- keys; observed map scaling and object re-rendering.	P	18/4/25
1.1.3 Camera Movement	Simulation.scaled, main() event loop (K_EQUALS, K_MINUS), gridpos2screen()	Mouse at screen edges; observed map panning.	P	18/4/25
1.2 Creatures (Bees)	**Creature class**			
1.2.1 Basic Properties	Creature.__init__() (attributes: role, pos, energy, honey, hive, velocity, acceleration etc.)	Inspected selected bee stats via UI; Debug prints during initialization.	P	19/4/25
1.2.2 Drawing Creatures on map	Creature.draw() method	Visual: Bees drawn on map at correct world positions.	P	19/4/25
~~1.2.3 Hovering mouse over bees~~	Legacy Feature now Removed. Remnants canbe found in is_hovered()	Hovering over bees draws a circle around them. **Now removed.**	P	19/4/25
1.3 Flowers (Food Sources)	**Flower class Flower.__init__() (pos, pollen), Flower.update() (pollen decrease, manager.rem_flo())**	**Visual: Flowers appear, bees interact, flowers removed when pollen empty.**	P	20/4/25
2.0 Advanced Bee Behaviours	**Creature.update(), Creature.calculateForces()**			
2.1. Boid Separation	Creature.separation()	Visual: Bees maintain distance from each other.	P	22/4/25
2.2. Boid Alignment	Creature.align()	Visual: Bees tend to match average velocity of neighbors.	P	22/4/25

2.3. Boid Cohesion	Creature.cohesion()	Visual: Bees move towards center of mass of neighbors.	P	22/4/25
2.4. Flower Detection	Creature.seekFlowers(), Creature.goFlower(), distance() checks	Bees identify and move towards closestflower.	P	22/4/25
2.5. Random Directional Movement	Creature.update() (pygame.math.Vector2.rotate(self.velocity, rotate))	Visual: Bee movement appears less deterministic, more natural.	P	22/4/25
2.6. Border Detection	Creature.avoidedge() (position clamping), Creature.calculate_avoid_edge_force()	Visual: Bees stay within map/hive; repel from edges.	P	23/4/25
2.7 Clicking on Bees	Simulation.handle_click() (detects bee click), Simulation.show_selected() (displays bee info)	Clicked bees; info panel appeared with correct stats.	P	23/4/25
3.0 Hives	**Hive class**			
3.1. Hive Spawns Bees	Bees Hive.__init__() (initial bees), Hive.update_eggs() calls manager.spawn_new_bee()	Observed initial bees; new bees appear after egg "hatching".	P	24/4/25
3.2. Hive Spawning Rules	Hive.update_eggs() (j <= -30 for hatch), Creature.dohoneythings() (queen egg laying), EGG_CELL_VALUE	Queen lays eggs in combs; eggs "develop" (j decrements); new bees spawn.	P	25/4/25
3.3. Hive View	Simulation.show_selected() (hive part), hivepos2screen(), Creature.draw_inside_hive(), comb drawing logic	Clicked hive; internal view with combs and bees rendered.	P	30/4/25
3.4. Bee Movement in Hive	Creature.update() (when is_outside is False), Creature.dohoneythings(), Creature.hive_pos	Bees inside hive move towards combs to deposit honey/lay eggs.	P	4/5/25
3.5. Honey Combs	Hive.combs, Hive.combs_honey, Creature.dohoneythings() (honey deposit logic), drawing in show_selected	Honey levels in combs change; visual color change in hive view.	P	5/2/25
4.0 Obstacles	**Obstacle class**			
Obstacle Spawning	Simulation.add_obstacles(), Obstacle.__init__()	Visual: Obstacles appear on map at random valid locations.	P	6/5/25

Obstacle Avoidance	Creature.avoidrock(), Obstacle.get_closest_point(), Simulation.obstaclemap (used in A*)	Visual: Bees navigate around obstacles.	P	6/5/25
Map Avoidance (Water)	Creature.avoidwater(), checks manager.background.maparr using B_WATER_THRESH	Visual: Bees change direction when approaching water tiles.	P	6/5/25
5.0 General Improvements & UI				
5.1 Hexagona Combs	Hive.__init__() (comb generation logic: radius_long, horizontal_diff), drawing in show_selected	Visual: Comb pattern in hive view appears hexagonal.	P	7/5/25
5.2 FPS Improvements	Environment.update_background() (cached_bg, cached_camera_offset, cached_scaled), Creature.update() (manager.frames % 5 == self.selectedframe for calculateForces)	FPS counter check; subjective smoothness with many entities.	P	8/5/25
5.3 Interactive Mode	argparse setup (bottom of code), sim.update_values(), if args.interactive: block	Ran with -i; prompted for inputs; simulation used those values.	P	9/5/25
5.4 Code Cleanup	Simulation instance (sim or manager) passed to methods (e.g., Creature.update(self, manager))	Code review: sim/manager passed, not direct global access for state.	P	9/5/25
6.0 Advanced Pathfinding	Creature.a_star_pathfind(), Node class.	(Currently commented out in Creature.update()) Manually called for testing.	P	9/5/25
7.0 Queen Bee	Creature (role == 'queen'), Creature.do_queen_things(), Creature.dohoneythings() (for egg laying) H_INITIAL_QUEENS, Queen specific size in Creature.__init__ / update	Queen bee lays eggs; self.eggs attribute tracked.	P	10/5/25

Discussion

The bee simulation creates a top down world where bees have to interact with objects in their environment. The goal of the simulation is to learn and make use of languages and algorithms to deliver a compelling simulation of 2D creatures which mimic bee-like movements and behaviours. For example, in-depth research into boid movement, physics, and pathfinding algorithms were explored in the creation of this project. Below is a comprehensive UML diagram of the interactions between the different classes:

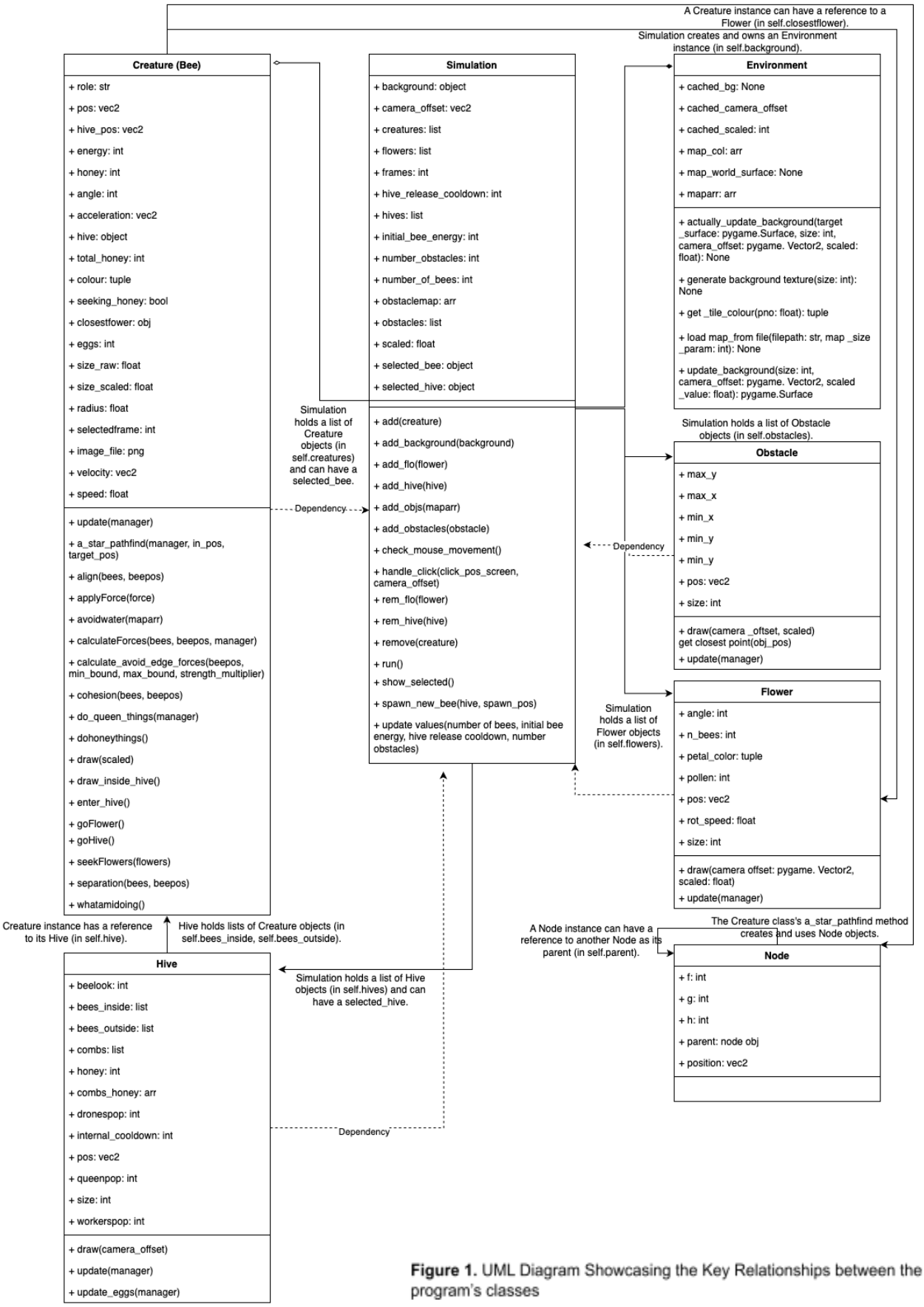


Figure 1. UML Diagram Showcasing the Key Relationships between the program's classes

1. Core Simulation Engine

The engine runs a main() loop that processes events (like key presses) and updates the game state every frame (at 60 FPS). Frames are similar to discrete time steps like step_change in the original Practical 3 assignment. An advantage of this FPS approach, which effectively increases the number of step_changes significantly, is that bees can be modeled under more realistic physics, such as velocity and acceleration, resulting in individual movement that differs from bee to bee. Bees are capable of moving in all directions, at a vector they choose to. This would be detailed later in the discussion.

1.1. Simulation Class (Central Coordinator):

The Simulation class acts as the central manager, holding game objects (bees, hives, etc.) and global settings (zoom, camera offset) to avoid global variables [TM 5.4]. The run() method orchestrates frame-by-frame updates for all active entities. Camera panning and zooming are handled by adjusting camera_offset and scaled attributes, used by gridpos2screen() for rendering." Zooming was initially basic, scaling the map directly, which worked best with powers of two. This results in the zoom being limited to double or halving the map size.

2. Bee Representation and Behaviours (Creature class) [TM 1.2, 2.0, 6.0, 7.0]

Bees are autonomous agents represented with Creature objects that each contain positional (pos, velocity), static (role, hive), and internal (energy, honey) attributes [TM 1.2.1]. Their initial state is set upon instantiation.

2.1. Movement Logic (in Creature.update())

Each bee contains a position, which is affected every frame by velocity. This velocity is clamped between the MIN and MAX speed constants, and is updated every frame as per the bees acceleration.

Movement logic in Creature.update() includes:

Boundary Clamping (avoidedge() [TM 2.6]): Confines bees to world/hive limits.

Edge Avoidance (calculate_avoid_edge_force() [TM 2.6]): Gentle repulsion from boundaries.

Boids Flocking (calculateForces() [TM 2.1-2.3]): Combines separation, alignment, and cohesion for group movement, with calculations staggered (e.g., manager.frames % 5) for performance. This is where the main advantage of pygame comes in. As the current iteration of boids requires each individual bee to access the location of every other bee, it results in a computational complexity of $O(n^2)$. This would not have been traditionally possible with matplotlib, as it is not designed for high frame by frame iteration, and would have caused significant performance decreases. The 128x128 map and high number of bees and hives could not be achievable with the traditional matplotlib library.

Steering forces are summed and applied via applyForce() to update acceleration, velocity (capped), and position (pos or hive_pos), with a random rotation for naturalism. This rotation is to reflect wind / other factors that may affect the path of a creature in aviation.

2.2. Lifecycle and Energy [TM 1.2.1]

Bees lose energy (CR_ENERGY_DECAY > 0), are removed if energy depletes, and size can be energy-linked [TM 1.2.1]. This is to ensure that the population is able to decrease.

2.3. Queen Bee Specifics [TM 7.0]

Queens (self.role == "queen") don't seek honey. They reproduce (do_queen_things()). Inside the hive, dohoneythings() helps them find an empty comb cell to lay an egg if they have self.eggs.

3. Environmental Elements [TM 1.1, 1.3, 4.0]

The Environment class generates the MAP_SIZE x MAP_SIZE terrain using Perlin noise (pnoise2). get_tile_colour() maps noise values to visual representations (water, sand, grass) [TM 1.1.1]. Background rendering is cached for performance. A now fixed old bug called the re-rendering of the background every frame, causing performance drops when zoomed out, which has now been updated to only occur when the tile is specifically updated (by zoom or otherwise).

3.1. Flower objects [TM 1.3] have position and pollen. Pollen decreases on collection, and flowers are removed when empty.

To manage crowding realistically, a maximum number of bees per flower was implemented rather than pollen recharge, reflecting difficulty in pollen extraction.

3.2. Obstacle objects [TM 4.0] are static rectangular barriers. get_closest_point() aids bees in calculating avoidance. Rectangular shape necessitated array-based avoidance over simpler radius checks, posing an implementation challenge.

4. Hive Mechanics (Hive class) [TM 3.0, 5.1]

Hive objects are central bases [TM 3.0], managing bees (inside/outside lists), population counts, and an internal honeycomb structure (self.combs for positions, self.combs_honey NumPy array for cell states like honey/eggs)

4.1. Bee Management (Hive.update())

Hive.update() periodically releases bees from bees_inside to bees_outside if seeking_honey, regulated by internal_cooldown (H_BEE_COOLDOWN) and a beelook index for leaving the hive. Hive size on map scales with manager.scaled*4.

4.2. Honey Deposition [TM 3.5]: Worker bees with honey use Creature.dohoneythings() inside the hive to find non-full/non-egg comb cells. They deposit honey (incrementing self.hive.combs_honey, decrementing self.honey) and resume seeking when empty.

4.3. Egg Laying & Hatching [TM 3.1, 3.2, 7.0]: Queens use Creature.dohoneythings() to find empty cells (value <= -1) for eggs if self.eggs > 0, marking cells with EGG_CELL_VALUE. Hive.update_eggs() periodically decrements egg cell values to simulate development. Upon reaching HATCH_THRESHOLD, the cell resets, and manager.spawn_new_bee() creates a new worker. The current hatching process is basic, with new bees immediately seeking honey.

4.4. Visualization [TM 3.3]: Selecting a hive (Simulation.show_selected()) renders its internal view. hivepos2screen() converts comb coordinates. Honey cells are colored by level (0-100, often HSL shades); egg cells have a distinct color. Bees inside (self.hive.bees_inside) are drawn via Creature.draw_inside_hive().

5. User Interface and Visualization [TM 1.1.2, 1.1.3, 1.2.3, 2.7, 5.0, 5.2, 5.3]

Pygame handles all rendering and interaction. This includes the main game engine, text rendering and graph rendering (population graph)

Interactive Mode Setup [TM 5.3]: Triggered by -i (parsed via argparse), prompts for key parameters (bee count, energy). User inputs update Simulation attributes via sim.update_values(). Batch mode can be triggered by -b, which takes a map and params csv file using -f and -p respectively.

5.2. Information Display [TM 1.2.3, 2.7, 3.3]

Simulation.handle_click() sets self.selected_bee or self.selected_hive. Simulation.show_selected() then renders an info panel: bee stats (pos, energy, etc.) or detailed hive internal view. This implementation was

chosen to allow for multiple hives to coexist in the simulation, to observe the variables that causes success in a hive.

5.3. Visual Style and Optimizations [TM 5.1, 5.2]

Visual Style & Optimizations [TM 5.1, 5.2]: Bees are colored circles (queens distinct). Perlin noise terrain and hexagonal combs enhance visual appeal. Performance benefits from cached background rendering (Environment.update_background()), crucial for larger maps [TM 5.2].

Pixel art textures were made for the bees and flowers (bee.png and flower.png), but they were not used as it did not fit to the rest of the simulation's visual style.

5.4. Code Structure [TM 5.4]

Logic is encapsulated in classes (Simulation, Creature, etc.). The Simulation instance (manager) is passed for context, reducing global variable reliance. Constants are grouped for configurability.

6.0 Advanced Features

Beyond the core requirements, several advanced features and techniques were implemented to improve the simulation's detail, useability, performance and quality of life. These are detailed below for bonus consideration.

6.1. Advanced A* Pathfinding Implementation [TM 6.0]

An A* algorithm (a_star_pathfind() [TM 6.0]) is implemented for optimal pathfinding around obstacles/water but is not used for regular bee movement due to a preference for reactive behaviors. While the default bee movement relies on reactive boid behaviors for naturalism and computational efficiency in a large-scale simulation, the A* algorithm provides a framework for more directed pathfinding. The algorithm finds the shortest path, whilst considering terrain.

Seen in **Appendix B**, each bee attempts to A* pathfind to a random location. This algorithm is both expensive and not faithful to how real life creatures path find (they dont always choose the most optimal path from X to Y), this feature was not used for creatures. Whilst not enabled, this demonstrates the possible use of search algorithms to make “smart” creatures with complex behaviour.

6.2. High-Quality Visualization: Detailed Hive View & Population Graph

The simulation incorporates several high-quality visualization techniques:

Interactive Hive Visualization [TM 3.3, TM 3.5, TM 5.1]: Selecting a hive provides a detailed internal view. This goes beyond a simple numerical display, rendering a hexagonal comb structure (Hive.__init__() comb generation logic, Simulation.show_selected() drawing). Trigonometry was used for calculating exact hexagonal specifications, but an easier method could have possibly existed.

Individual cells dynamically change color based on honey levels (using HSL for smooth gradients) and egg development stages, allowing users to visualise the progress of hive cells in their respective use cases. Bees are rendered moving within this internal hive view (Creature.draw_inside_hive()), adding to the immersive detail.

Real-Time Population Graph: A dynamic graph (Simulation.draw_population_graph()) tracks the total bee population over time. This provides immediate visual feedback on the colony's health, growth, or decline in response to environmental conditions and simulation variables.

6.3. Performance Optimization: Cached Background Rendering [TM 5.2]

To ensure smooth performance in larger maps and high entity counts, the background is cached in the rendering system in Environment.update_background(). The terrain is only redrawn if the camera offset or

zoom level (scaled value) changes. This significantly reduces the rendering load per frame and offered an up to 3x performance increase, demonstrating an understanding of Pygame optimization techniques crucial for complex simulations.

6.4. Reactive Behaviors for Realistic Movement

The combination of boid flocking [TM 2.1-2.3], edge avoidance [TM 2.6], obstacle avoidance [TM Obstacle Avoidance], water avoidance [TM Map Avoidance (Water)], and random directional movement [TM 2.5] creates emergent, complex, and natural-looking bee behaviors. This demonstrates how complex movement can emerge from a set of simple pre-programmed rules.

These extensions collectively provide an intricate simulation environment. As observed in the showcase.

Showcase

This showcase section aims to demonstrate the diverse feature set and results of the "Untitled Bee Simulation." It will explore three distinct scenarios, each configured to highlight different aspects of the simulation. Whilst the simulation is capable of handling millions of varying scenarios with its vast configurability, here are three basic ones to showcase how the bees are able to handle different environments.

Scenario 1: Normal Mode (with Default Settings)

The objective is to observe the default baseline bee population behaviour, resource collection, and hive sustenance under default or standard simulation parameters, providing a benchmark for understanding the core ecosystem balance.
To set up, run:

```
$ python3 main.py -s 202505
```

-s ensures that the seed is the same throughout the simulation. It does not affect constants, which remain default.

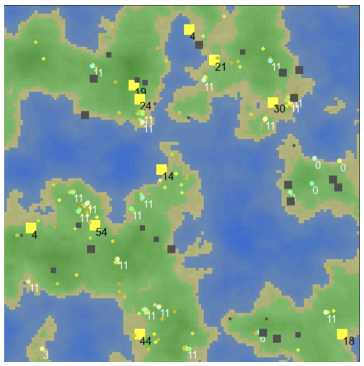
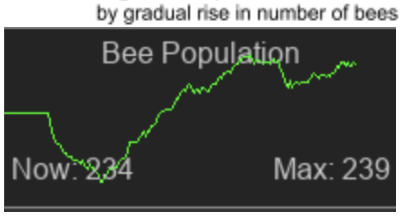


Figure 2. Scenario 1 Midway through the simulation

This scenario is the more stable ones, as variables are configured and optimised for performance and stable bee populations.

Natural selection takes within the first 60 seconds of the simulation, matching up with the default initial energy for the bees. We observe that weaker hives, (those who have more competition and/or is spawned further away from any flowers) struggle and most likely experience a population collapse. Straggler bees who did not have time to find a flower (and in turn replenish their energy) also collapsed. There was a general Initial dip in population that was observed, especially for hives distant from early flowers.

Figure 3. Population fall followed by gradual rise in number of bees



Population recovered in much more successful colonies, who managed to spawn near flowers. Successful workers supplied hives, enabled egg-laying and new bee hatches. Total bee count stabilized around 240 bees. It is important to note that the stagnation in population had a far larger variance than I anticipated; population was swinging up and down 20 and was far less stable than originally anticipated. In general, the success of the hive is dictated by the availability of flowers and the distance of the flowers. More

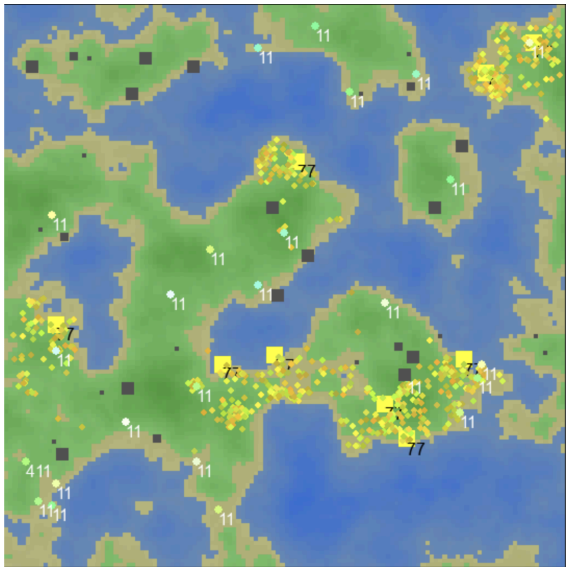
successful colonies reached a population plateau as the one queen bee was unable to produce enough new bees to sustain the supply of honey.

This scenario illustrates the core ecological feedback loop: resource availability dictates bee survival and honey production, which fuels hive growth. The equilibrium suggests an emergent carrying capacity based on default resource density and bee life cycle parameters.

Scenario 2: Interactive Mode

```
$ python3 main.py -i -s 101010
::::::::::::::::::::INPUTS::[Interactive Mode]::::::::::::::::::::
Warning: Using extreme values may yield unexpected, untested and unaccounted for
results. Performance could also be decreased.
Number of bees per hive (default=20) (1-100): 75
Bee Initial Energy (default=50) (10-1000) WARNING: BEE SIZE SCALES WITH ENERGY!: 50
Hive Bee Release Cooldown (default=3) (0-300)3
Number of obstacles (default=30) (0-100): 30
Number of hives (default=10) (1-50): 9
Initial queens per hive (default=1) (0-10): 2
Creature Max Velocity (default=0.1) (0.01-1.0): 0.1
Creature Min Velocity (default=0.02) (0.001-0.5): 0.02
Creature Detection Radius (default=2.5) (0.1-20.0): 3
Creature Separation Threshold (default=0.7) (0.1-10.0): 0.7
Flower Size (default=8): 8
::::::::::::::::::::[End Interactive Mode]::::::::::::::::::::
```

Key inputs: 75 bees/hive, 9 hives, 2 queens/hive, 50 initial energy.



Observations & Results: Massive initial bee population (693 bees) led to rapid depletion of all map flowers within seconds. Most bees are unable to find a flower due to the flower's max bee capacity.

Extreme congestion around the hive was observed externally.

A sharp population collapse followed, to <170 bees, due to resource exhaustion and high competition. FPS dropped to ~18 FPS during peak density.

Discussion: The simulation demonstrated an emergent systemic failure due to overpopulation exceeding environmental carrying capacity. High bee density stressed hive internal mechanics, resource access, and boid interactions, leading to a classic population crash.

Figure 3. Initial screenshot of the start of Scenario 2

Figure 4. Population Collapse

Scenario 3: Batch Mode

Objective: Test long-range foraging and resource exploitation on a custom map with concentrated resources.

```
$ python3 main.py -b -f map_oasis.csv -p params.csv
```

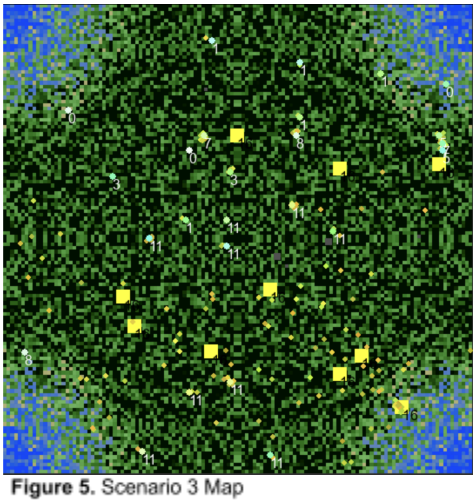


Figure 5. Scenario 3 Map

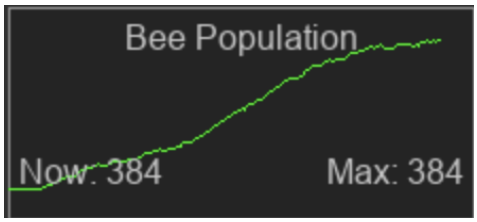


Figure 6. Scenario 3 Population

map_oasis.csv: Features a central fertile oasis surrounded by water/barren land. Hives start peripherally.
params.csv: Allows the change of any constant.

Observations & Results:

Bees from peripheral hives gradually discovered and navigated to the distant central oasis.
The oasis became a high-activity zone with significant competition for flowers.
Hive success correlated with proximity/speed in locating the oasis; some distant hives struggled.
FPS drops to ~30 as the population continues to increase. The population graph trended towards a sigmoid as time increased and continued to peak until ~420 bees before the simulation was stopped.

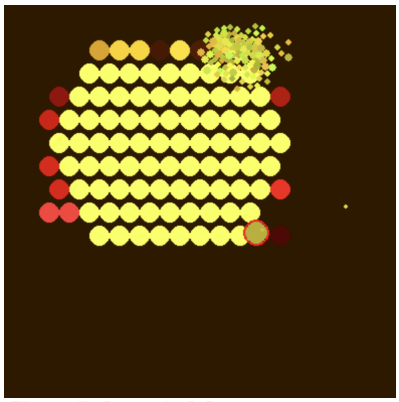


Figure 7. Scenario 3 Screenshots

Discussion: This scenario showcased the bees' capacity for long-range resource exploitation, driven by their search behaviors and ability to navigate terrain defined by the custom map. It highlighted how concentrated resources can become focal points for inter-hive competition.

Conclusion

This assignment provided a challenging yet rewarding opportunity to apply object-oriented programming principles to create a dynamic simulation. The goal was to simulate bee-like creatures with complex interactions, and overall, the project successfully implements core features like boid flocking, resource collection, hive management, and environmental interaction.

Key challenges included:

Balancing the various steering behaviors for natural-looking movement without excessive computational load.

Designing the hive's internal logic for honey storage and bee reproduction in a way that was both functional and visually representable.

Optimizing rendering, especially for the background and a large number of entities, leading to techniques like background caching.

Through this project, I gained significant experience with Pygame for visualization and event handling, NumPy for efficient data management (like combs_honey), and the practical application of algorithms like Boids and Perlin noise generation. The current simulation provides a solid foundation, and the implemented features (like interactive mode, detailed hive view, and distinct bee roles) met or exceeded the initial vision for a "kind of cool bee simulation." While A* Pathfinding was implemented, its integration

into real-time bee decision-making was deferred in favor of more reactive behaviors, which felt more aligned with simple agent simulation.

Future Work

While the current simulation is comprehensive, several avenues for future work could enhance its complexity and realism:

Camera: Whilst not a big deal, I would like to potentially overhaul the current camera zoom and scroll implementation. I find it far more intuitive for drag to move and linear scale factors (not powers of 2). However, due to limitation of key resources (time), the current implementation is “good enough”

Drifting Bees & Genetic Exchange: Implementing "drifting bees" that might join other hives could introduce a mechanism for genetic diversity or information spread between colonies. This could involve bees having a "home hive" loyalty that can wane, or accidental entry into foreign hives.

Drones: Adding male bees, who interact with the queen bee to produce larvae. Drones could share their genetics with the queen bee, and as a result the average genetic performance of the colony improves as the weaker drones are unable to reproduce.

Further FPS Optimisation: Currently each bee iterates through every other bee to calculate its boid forces. This results in $O(n^2)$ computational complexity, which is nonoptimal as the simulation struggles to handle high bee counts.

Genetics: Individualised factors that affect a bee’s performance. Resistance to disease, speed, energy efficiency and other factors that could impact getting honey. This would add another layer of complexity to the project.

Visual Overhaul: Update graphics and use more modernised textures. This was briefly attempted but I quickly realised that a complete overhaul of the style is required for the project to look coherent. (Also, the simple shapes have a lovely feel to it!)

Environmental Dynamics & Seasons: Introduce dynamic environmental changes, such as seasons affecting flower availability (e.g., fewer flowers in "winter," more in "spring/summer") or resource depletion in heavily foraged areas. This would add another layer of challenge for the bee colonies.

More Complex Bee Roles & Communication: Expand beyond workers and queens. Introduce drones with specific mating behaviors, or scout bees that more efficiently locate new food sources and perhaps communicate this information back to the hive (e.g., a simplified "waggle dance" mechanic influencing other bees' target locations).

Predators or Threats: Introduce predators (e.g., spiders, birds) or environmental threats (e.g., pesticides appearing in certain areas) that bees would need to avoid, adding a survival element.

Advanced Hive Economy: Implement more detailed hive resource management, such as different types of nectar leading to different honey qualities, or bees consuming stored honey during periods of scarcity.

Neural Networks (Sorry Harry!): Overhauling movements and replacing them with neural networks could create extremely complex creatures who produce varying strategies and thought processes in order to optimise to survive in the simulation. This was initially planned but the scale of the project would be far outside my scope.

References

Adams, van Hunter. n.d. “Boids Algorithm: Alignment, Cohesion, and Separation.” Accessed May 4, 2025. https://vanhunteradams.com/Pico/Animal_Movement/Boids-algorithm.html.

b001. 2022. *Python Classes in 1 Minute!* YouTube video. (Originally posted October 26, 2023). https://www.youtube.com/watch?v=yYALsys-P_w.

Clear Code. 2022. *Cameras in Pygame*. YouTube video. February 12, 2022. https://www.youtube.com/watch?v=u7LPRqrzry8.

Curtin University. 2024. “Chicago 17th Author-Date.” UniSkills. https://uniskills.library.curtin.edu.au/referencing/chicago17/introduction/.

DaFluffyPotato. 2025. *Pygame In 18 Minutes.* YouTube video. January 10, 2025. https://www.youtube.com/watch?v=blLLtdv4tvo.

EverythingScience. 2023. *How Do Bees Find Their Way Home?* YouTube video. (Originally posted August 21, 2020). https://www.youtube.com/watch?v=MJGpMJQrvms.

Martinez, Frank. 2023. “CSCE450 Project Submission.” Texas A&M University. December 11. https://people.engr.tamu.edu/sueda/courses/CSCE450/2023F/projects/Frank_Martinez/index.html.

Pygame. 2025. *Pygame Documentation.* http://pygame.org/docs/.

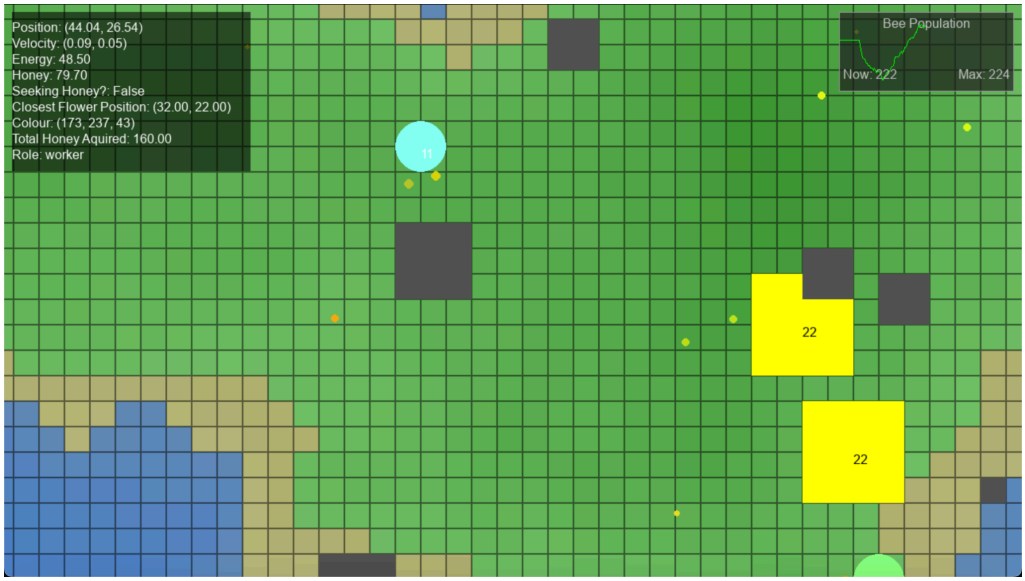
Python Software Foundation. 2025. “argparse—Parser for Command-Line Options, Arguments and Sub-Commands.” Python Documentation. https://docs.python.org/3/library/argparse.html.

Wikipedia. 2025. “Perlin Noise.” Last modified April 27, 2025. https://en.wikipedia.org/wiki/Perlin_noise.

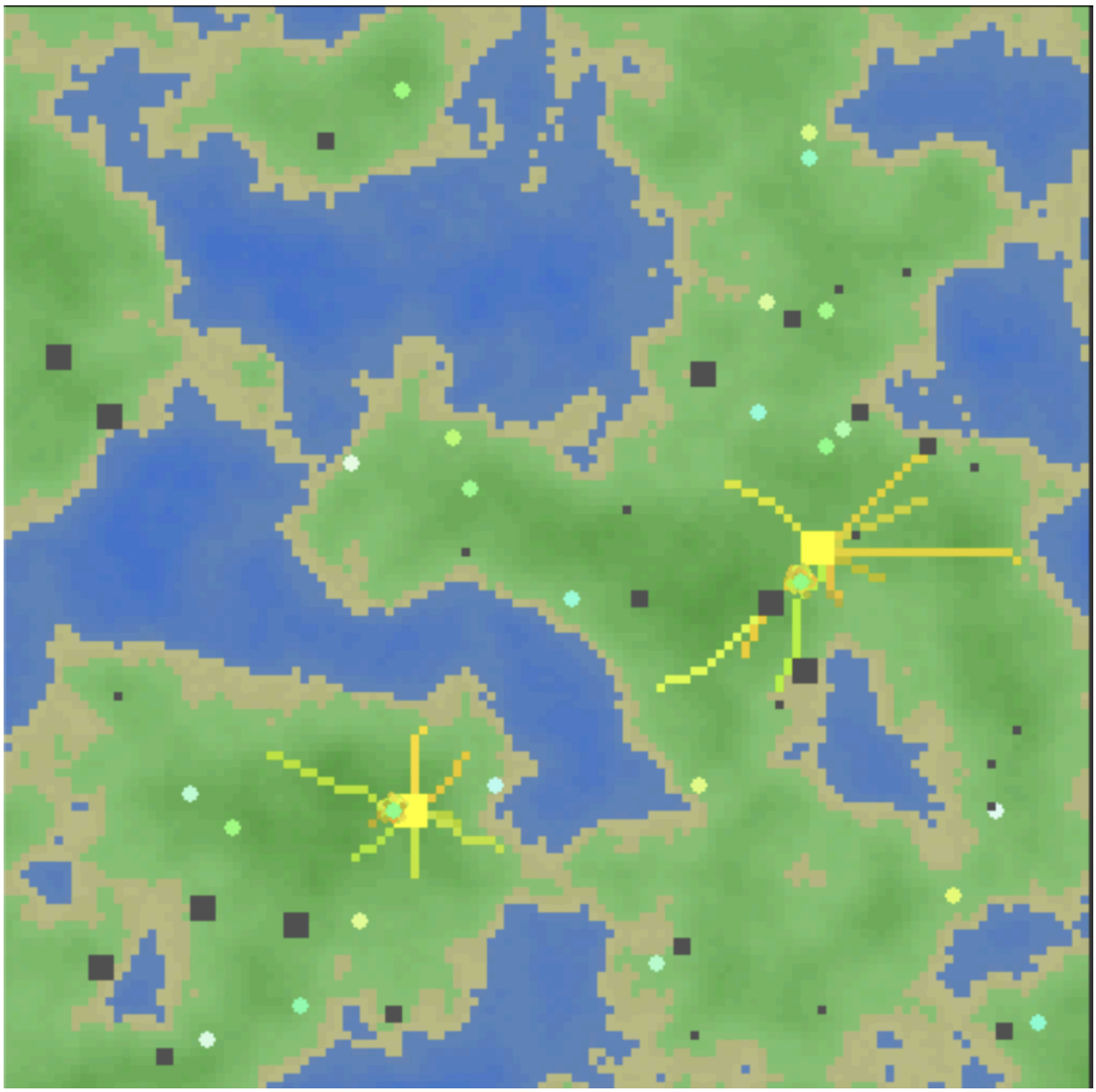
Zhang, Samson. 2020. *Building a Neural Network from Scratch (No TensorFlow/PyTorch, Just NumPy & Math)*. YouTube video. November 24, 2020. https://www.youtube.com/watch?v=w8yWXqWQYmU.

Appendix (Screenshots):

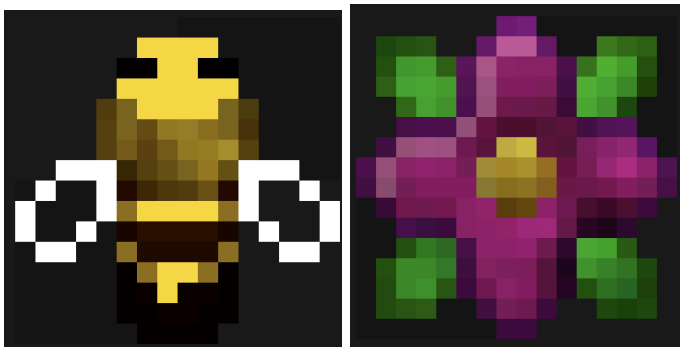
Appendix A. Clicking on bees



Appendix B. A* pathfinding visual demonstration



Appendix C. Unused Textures



Appendix D. Not that good of a UML generated using pyreverse