Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

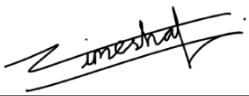
Last name:	Jayatunge	**Student ID:**	20464631
Other name(s):	Nimesha Thatsara		
Unit name:	Fundamentals of Programming	**Unit ID:**	COMP 1005
Lecturer / unit coordinator:	Dr Valerie Maxville	**Tutor:**	Dr. Charika Weerasiriwardhane
Date of submission:	11/Oct/2021	**Which assignment?**	**(Leave blank if the unit has only one assignment.)**

I declare that:

- **The above information is complete and accurate.**
- **The work I am submitting is** *entirely my own***, except where clearly indicated otherwise and correctly referenced.**
- **I have taken (and will continue to take) all reasonable steps to ensure my work is** *not accessible* **to any other students who may gain unfair advantage from it.**
- **I have** *not previously submitted* **this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.**

I understand that:

- **Plagiarism and collusion are dishonest, and unfair to all other students.**
- **Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).**
- **If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.**
- **Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.**
- **It is my responsibility to ensure that my submission is complete, correct and not corrupted.**

Signature:  **Date of signature:** 11/Oct/2021

(By submitting this form, you indicate that you agree with all the above text.)

CONTENTS

1 – OVERVIEW

Cats.py is a simulation of cats in a grid-based environment with variable parameters. The cats move around the grid, interacting with various environmental features such as food and water as well as with each other. The outcomes of these interactions depend on several factors, including the cats' personalities, sex, age, and height of the terrain. In this simulation, one timestep represents one hour.

Terrain height and landmark (food and water) distribution data are stored in csv files that can be read into the program to generate the environment. Additionally, variables such as the number of cats, number of hours to be simulated, and the type of cellular neighbourhood (Moore or Von Neumann) can be specified by the user.

The rules of cat behaviour in the simulation are outlined below:

Movement

- A cat can only move to cells within its immediate neighbourhood (Moore or Von Neumann) and within the grid boundaries
- A cat cannot jump to a position that is more than a certain number of units (jump_height) higher or lower than its current position
- A cat cannot move to a cell that is occupied by another cat, a food cell, or a water cell
- A cat will get hungrier and thirstier over time, and will move in the direction of food or water depending on whether it is more hungry or more thirsty
- A cat leaves behind a scent when moving through a cell. A cat will tend to follow the scent left by a cat of the opposite sex, and avoid the scent left by a cat of the same sex

Cat-Environment Interaction

- A cat can detect food cells and water cells inside its immediate neighbourhood, unless they are across a height difference of more than the jump height
- A cat will eat from a neighbouring food cell or drink from a water cell if its hunger or thirst is over a certain threshold. Eating and drinking reduces the cat's hunger and thirst respectively
- A cat can die of hunger or thirst
- Food and water cells get depleted if a cat consumes them, and they do not get replenished

Cat-Cat Interaction

- A cat can be either "aggressive", "friendly", or "meek"
- A cat can detect other cats inside its immediate neighbourhood, unless they are across a height difference of more than the jump height
- Cats on higher ground deal more damage
- Older cats deal more damage
- Aggressive cats attack all other cats of the same sex

- Friendly cats only fight if they are attacked first
- Meek cats run away from all other cats of the same sex
- A cat will not fight or run away from a cat of the opposite sex
- Two cats of opposite sex can reproduce
- Once a cat reproduces, it must wait 24 hours (or user-specified number) before it can do so again

Sleeping

- A cat has a random chance to go to sleep at any timestep
- Once a cat goes to sleep, it will sleep continuously for 8 hours (or user-specified number) until attacked by another cat
- While sleeping, cats regain health over time
- Cats are more likely to go to sleep during night-time (9pm to 5am)
- The lower the cat's health, the more likely it is to go to sleep

The contents of the submitted zipped folder are as follows:

ProjectReport.pdf - This report
Cats.py - Main source code
SweepBase.py - Base code for parameter sweep
ParameterSweep.sh - Bash script for parameter sweep
terrain.csv - csv containing terrain height data
terrain2.csv - csv containing terrain height data for testing
landmarks.csv - csv containing food and water data
landmarks2.csv - csv containing food and water data for testing
landmarks3.csv - empty csv for testing
heart.png - Sprite of a cartoon heart, used as a visual in the simulation
README.txt - README file
Tests - Directory containing outputs of the tests discussed in the Results section

2 – USER GUIDE

Main Simulation

The simulation requires two external python packages: **numpy** for array manipulation, and **pygame** for enhanced visuals and animation. Hence, if the user does not have these packages installed on their machine, they must do so using the following commands:

```
pip install pygame==2.0.1
pip install numpy==1.16.6
```

Once the dependencies are installed, the user can run the program using the following command:

```
python3 Cats.py terrain.csv landmarks.csv
```

`terrain.csv` may be replaced with the path to any csv file containing terrain height data, with one integer in each cell, 0 being the minimum height and 10 being the maximum.

`landmarks.csv` may be replaced with the path to any csv file containing food and water data. Any cell in the csv must either be empty, contain only the character "F" for food, or contain only the character "W" for water.

The user may provide three additional integer command line arguments after the landmark filename: **framerate** (timesteps per second), **mating_cooldown_time** (hours a cat must wait after reproducing) and **sleep_hours** (maximum uninterrupted hours of sleep), in that order. If these arguments are not given, the program will still run using default values of 2, 24 and 8, respectively.

Once the program starts, the user will be prompted to enter the desired cellular neighbourhood that the simulation will employ. The user must enter "M" for Moore neighbourhood or "V" for Von Neumann neighbourhood.

Next, the user is prompted to enter the number of hours they wish the program to simulate (1 hour = 1 timestep). If "0" is entered, the simulation will run indefinitely until the window is closed.

Next, the user is prompted to enter the number of cats they wish the program to simulate. The user must enter an integer.

The **pygame** window will now open and the simulation will run until closed or until the desired number of hours is reached. As events occur in the simulation, they will get printed to the

terminal as a running log. With the **pygame** window active, the user can interact with the simulation by pressing the following keys:

's' – toggles the visualisation of cat scents (blue for male and pink for female)

'f' – toggles the visualisation of food scents (green)

'w' – toggles the visualisation of water scents (blue)

Once the simulation is over, some additional statistics and data will be displayed for the user's analysis.

Finally, the user will be asked if they want to save the final grid state and/or event log. The user must enter "Y" or "N" to each of these prompts. If the user answers "Y" to any of these prompts, a new directory will be created with the name format "Simulation_<date>_<time>".

If the user chooses to save the grid state, the following files will be saved to this new directory:

final_cats.csv – csv file containing the positions of the cats in the final frame of the simulation. "A" indicates a live cat at that cell and "D" indicates a dead one.

final_landmarks.csv – csv file containing the food and water cells in the final frame of the simulation. A field in the csv begins with the letter "F" (for food) or "W" (for water) followed by a space and the number of units of food or water that were present at that cell.

terrain_used.csv – csv file containing the terrain data used for that particular simulation.

simulation.png – An image of the final frame of the simulation.

If the user chooses to save the event log, the following file will be saved to the new directory:

log.txt – Text file with the event log and statistical data for the simulation.

Parameter Sweep

The user can choose to perform a parameter sweep as well, which will sweep through the user-provided value ranges for the variables **mating_cooldown_time** each and **sleep_hours**.

The sweep is initiated by typing the following single command in the command line:

```
sh ParameterSweep.sh terrain.csv landmarks.csv <neighbourhood>
<max_hours> <cat_number> <low_cooldown> <hi_cooldown>
<step_cooldown> <low_sleep> <hi_sleep> <step_sleep>
```

`<neighbourhood>` – "M" or "V"

`<max_hours>` – Simulation length (in hours)

`<cat_number>` – Initial number of cats

`<low_cooldown>` – Minimum mating cooldown time (hours)

`<hi_cooldown>` – Maximum mating cooldown time (hours)

`<step_cooldown>` – Step size for the mating cooldown time range (hours)

`<low_sleep>` – Minimum number of uninterrupted sleep hours

`<hi_sleep>` – Maximum number of uninterrupted sleep hours

`<step_sleep>` – Step size for sleep hours range

A new directory will be created with the name format "Sweep_<date>_<time>". Inside this directory will be one folder for each simulation performed in the sweep. The name format for each of these folders is "Simulation_M< m >_S< s >", where < m > and < s > are the **mating_cooldown_time** and **sleep_hours** values, respectively, for that particular simulation.

3 – TRACEABILITY MATRIX

All line numbers, functions, and methods referenced in the table below belong to **Cats.py**

Feature	Sub Feature	Implementation	Testing
Object Behaviour	Cats are represented as objects, with different personalities, genders, and ages	**line 59, class Cat()**	[PASSED] Called the display_self() method for each cat and made sure all attributes were implemented correctly.
Terrain and Boundaries	Can read in terrain and landmark csv files	**line 210, read_terrain()** **line 223, read_landmarks()**	[PASSED] Created terrain.csv and lansmarks.csv and made sure the program read them in correctly and converted them to arrays.
	Terrain and landmark csv files do not need to match the dimensions of the environment grid	**line 216 in read_terrain() and** **line 233 in read_landmarks():** Exception handling ensures that program continues to run even if dimensions do not match	[PASSED] Created terrain2.csv and landmarks2.csv which did not match the dimensions of the environment grid. Read these files into the program and confirmed that it executed without crashing.
	Cats cannot leave the grid boundaries	**line 254 in get_valid_moves()**	[PASSED] Number of cats was reduced to one. Possible moves list was printed at each timestep. Made sure that cells outside the grid did not appear in the list.
Movement	User can choose between Moore or Von Neumann neighbourhoods	**line 696:** User is asked for desired neighbourhood **line 246 in get_valid_moves():** If statement makes the cat's possible moves different based on neighbourhood choice **line 279 in check_surroundings():** If statement ensures that surrounding cells of a cat depend on neighbourhood choice **line 383 in diffuse():** If statement ensures that diffusion of scents depends on neighbourhood choice	[PASSED] Number of cats was reduced to one. Possible moves list and surrounding cells list were printed at each timestep, first with Moore and then with Von Neumann. List contents were as expected.
	Cats cannot move across steep slopes	**line 256 in get_valid_moves()**	[PASSED] Created terrain2.csv which has sections of abrupt elevation change. Ran the simulation with this file and made sure that cats do not cross the "cliff" edges.

	Cats cannot move to a cell that is occupied by another cat, a food cell, or a water cell	**line 258 to line 263 in get_valid_moves()**	[PASSED] Made grid size very small and filled many cells with food and water. Cat did not overlap with food or water, as expected. Made grid size very small, reduced number of cats to two and made sure they did not overlap.
	Cats leave behind a scent when moving through a cell	**line 364, update_cat_scents():** Function creates cat scents **line 449 to line 452 in draw_screen():** Draws scents to screen	[PASSED] Ran the simulation with scents turned on and made sure they were drawn properly.
Food/ Water	Cats search for food and water	**line 642 to line 661 in main_loop()**	[PASSED] Reduced the number of cats to one, and observed to make sure the cat selectively moved towards food or water.
	Cats will eat from a neighbouring food cell or drink from a water cell, reducing their hunger and thirst	**line 302, eat_or_drink()** **line 134, method eat() in class Cat()** **line 142, method drink() in class Cat()**	[PASSED] Reduced the number of cats to one, printed the cat's hunger and thirst at each timestep, and made sure that eating and drinking reduced hunger and thirst.
	Cats can die of hunger or thirst	**line 130 in method hunger_and_thirst() in class Cat()**	[PASSED] Created landmarks3.csv which is empty, reduced the number of cats to one, then ran the simulation for an indefinite number of hours. With no food or water to eat, the cat died after 200 hours, as expected.
Interaction	Cats follow opposite sex scent, and avoid same sex scent	**line 266 to line 271 in get_valid_moves():** Avoid same sex **line 629 to line 638 in main_loop():** Follow opposite sex	[PASSED] Made grid size very small and reduced number of cats to two. Observed male and male cats. Observed female and female cats. Observed male and female cats. Confirmed that pathfinding was correct.
	"Aggressive", "Friendly", and "Meek" cats interact in different ways (with cats of the same sex)	**line 151, method interact() in class Cat()**	[PASSED] Made the grid size very small, reduced number of cats to two. Observed the interactions between all personality combinations of cats (of the same sex) and made sure they were correct.
	Older cats deal more damage	**line 69 in method __init__() in class Cat()**	[PASSED] Reduced number of cats to two, set different ages, placed them in each other's neighbourhood. Printed each cat's health at each timestep and made sure the younger cat lost health at a faster rate.

	Cats on higher ground deal more damage	**line 160 and line 171 in method interact() in class Cat()**	[PASSED] Reduced number of cats to two, placed them in each other's neighbourhood with one cat on a higher elevation. Printed each cat's health at each timestep and made sure the cat on lower ground lost health at a faster rate.
	Two cats of opposite sex can reproduce	**line 597 to line 607 in main_loop():** Checking conditions for reproduction **line 314, reproduce(): E**xecuting reproduction	[PASSED] Made the grid size very small, reduced number of cats to two, set one cat to male and one to female. Observed the interaction and made sure they reproduced successfully.
Sleeping	Cats sleep for sleep_hours number of hours at a time unless attacked, and regain health over time while sleeping	**line 197, method sleep() in class Cat()**	[PASSED] Reduced number of cats to one and printed health at each timestep. Made sure that the cat slept for correct number of hours and regained health over time. Reduced number of cats to two, and observed the interaction while one was asleep. Made sure the cat wakes up if attacked.
	Cats are more likely to go to sleep at night-time and when health is low	**line 610 to line 618 in main_loop()**	[PASSED] Reduced number of cats to one, printed its sleep_chance (probability that it will go to sleep) at each timestep. Made sure the sleep_chance was correctly affected by changing time of day and health.
Visualisation/ Results	User can toggle visualisation of cat scents, food scents, and water scents	**line 736 to line 742:** Detect key presses **line 449 to line 460 in draw_screen():** Draw scents if toggles are on	[PASSED] Ran the simulation and pressed the toggle keys ("s", "f", and "w") to make sure that they function as intended.
	Prints a running event log and statistical data.	**line 407 to line 414 in kill_cats():** Updating event log for deaths **line 327 to line 329 in reproduce():** Updating event log for births **line 509, show_stats():** Statistical data	[PASSED] Ran the simulation and made sure the log and statistical data got printed.
	User can save the grid state and event log to a new directory	**line 767 to line 819**	[PASSED] Ran the simulation and answered "Y" to the save prompts. Inspected the generated directory and confirmed that contents were as intended.
	User can perform a parameter sweep of mating_cooldown_time and sleep_hours	**SweepBase.py** and **ParameterSweep.sh** files	[PASSED] Executed ParameterSweep.sh and confirmed that the intended output was produced.

4 – SHOWCASE

Introduction

All line numbers, functions, and methods referenced below belong to **Cats.py**

The simulation models cats as objects that have certain attributes and methods. The **Cat()** class (defined on line 59) contains instance variables that are immutable characteristics of each cat – such as temperament, age, sex, and a unique ID – as well as properties that can vary moment to moment, such as its health, position, and whether it is interacting with something. These variables are used throughout the code to alter or even restrict certain behaviours based on them.

hunger_and_thirst() in line 118 of the **Cat()** class is a method that makes the cat hungrier and thirstier after each timestep, giving the cats a motivation to eat and drink.

The **eat()** method (line 134 in **Cat()**) takes in the position of a food cell and removes half of a unit from that cell, and 15 points from the cat's hunger level, to simulate the cat eating from that food cell. The **drink()** method (line 142) performs the same actions but with a water cell and the cat's thirst level instead.

The **interact()** method (line 151 in **Cat()**) is responsible for enforcing the rules of same-sex cat to cat interaction described in Section 1 of this report. The method takes in all the cats within the immediate neighbourhood (either Moore or Von Neumann) of the cat in question along with other variables required to perform the required interaction. An if-elif-else block is used to check whether the current cat is friendly, aggressive, or meek, depending on which the response is different. If it is aggressive, the cat takes away health points from (attacks) all of its neighbours, unless it is sleeping. If the aggressive cat is sleeping and its neighbour is also aggressive, however, it will attack because the neighbour would have attacked first and interrupted its sleep. If the current cat is friendly, it only attacks aggressive neighbours (i.e., only the neighbours that would have attacked first). If it is meek, it will find a new position by calling **get_valid_moves(),** and check whether the new position has another cat next to it using **check_surroundings().** If there isn't another cat there it will flee to that position.

The **sleep()** method (line 197) simply makes sure that if a cat goes to sleep, it sleeps for a specified number of hours (**sleep_hours**) at a stretch unless it is interrupted by an interaction. It also makes it so that the cat recovers some health for every hour that it sleeps.

read_terrain() in line 210 reads in a terrain csv file and converts it to an array of heights. Similarly, **read_landmarks()** in line 223reads in a landmarks csv file and converts it into two arrays: one for food cells and one for water cells. Both functions utilise exception handling when transcribing values from the csv files into the arrays so that even if the csv files do not match the dimensions of the environment grid (defined by **num_rows** and **num_cols**), the grid will be filled in by whatever values can fit, and the program will keep running.

Both **get_valid_moves()** in line 243 and **check_surroundings()** in line 276 perform similar tasks. Given a specific cat and its position, both functions create a list of neighbouring cells according to Moore or Von Neumann neighbourhood (**get_valid_moves()** includes the original position as well). This list is then pruned so that only cells that satisfy certain conditions are

considered. In **get_valid_moves()** all cells that are outside the grid environment, occupied by food, water, or other cats, or have too large a height difference from the original cat's position are removed. Additionally, cells which contain the scent of a cat that is the same sex as the original cat have a chance of being removed, proportional to the intensity of the scent. The output is thus a list of valid moves that the cat can make, adhering to the movement rules. **check_surroundings()** returns the cats, food cells, and water cells within the original cat's neighbourhood, therefore only cells that have too large a height difference are removed from the list of neighbouring cells.

The **eat_or_drink()** function (line 302) simply decides for a cat whether to eat from a neighbouring food cell or drink from a neighbouring water cell. This decision is made based on the cat's current hunger and thirst levels.

The function **reproduce()** in line 314 takes in two cats of opposite sex and creates a new cat object of age 1, spawning it in a cell neighbouring one of the parent cats. When two cats reproduce, a clipart image of a heart is added to the list **hearts** so that it can be drawn to the screen. The function also prints and stores a log update for the event.

assign_terrain_colour() in line 333 is a custom colourmap function. It takes in the height value of a cell (0 to 10) and uses a linear scale to return an RGB value corresponding to that height.

cat_scent_colour() in line 340 and **landmark_scent_colour()** in line 352 return coloured **pygame** images that correspond to the type and intensity of a scent. These images can then be drawn to the screen at the positions of the scents. The opacity of the image is proportional to the intensity of the scent, so that scents appear to fade out over time.

update_cat_scents() in line 364 updates the array of cat scents so that a new scent is created at each cat's position and every existing scent's intensity is reduced each timestep, to simulate evaporation. A cat's scent is stored as a list of [cat, cat's sex, intensity], hence every cat has a unique scent. If a scent's intensity falls below 0.01, the function simply removes it by setting its list contents to None, None, and 0.

In order to make cats search for food and water, food and water cells were given a scent that diffuses into the environment (as unrealistic as that may be in the case of water) and can be followed by a cat. The function **diffuse()** in line 377 handles this diffusion by using the model created by Dr Valerie Maxville in "heat.py" (Maxville, 2021a).

The function **kill_cats()** in line 401 removes all the dead cats (health below zero) from the **alive_cats** list and places them in the **dead_cats** list. It also prints and saves a log update for the time and cause of death.

The initial population of cats is created in the **create_cats()** function (line 419). This is done using a for loop, and also a while loop to ensure that cats do not spawn on top of food, water, or other cats.

The function **draw_screen()** in line 439 uses **pygame** methods to draw the current state of all cats, food, water, and terrain to the **pygame** screen, along with the images of hearts whenever cats reproduce. The scents will also be drawn if their respective toggles have been pressed by the user. Food and water are drawn as green and blue (respectively) circles with radius proportional to the units of food/water, cats are drawn as circles with radius proportional to the cat's age, and

dead cats are drawn as crosses. The function **display_time()** in line 476 also uses **pygame** methods to display the current timestep (in days and hours) of the simulation on the screen.

get_stats() in line 485 returns statistics for a given list of cats: number of aggressive, friendly, and meek cats, average age of the cats, and average health of the cats. **get_stats()** is called by the function **show_stats()** in line 509, which prints these statistics to the terminal in a readable format.

ask_choice() in line 553 and **ask_number()** in line 564 are functions that ask repeatedly ask the user for an input until one of valid format is given. The user is asked for inputs frequently in the program so having reusable functions increases efficiency.

The core movement and behavioural decisions are made in the **main_loop()** function (line 576), which is divided into four sections.

In the first section of **main_loop()** (line 580 to line 594), every cat observes its surroundings by calling the **check_surroundings()** function, and decides how to proceed accordingly. If another cat is in its neighbourhood, the **interact()** method is called to handle the interaction. Then if it is not fighting or fleeing, it will decide whether to eat or drink any surrounding food or water cells by calling the **eat_or_drink()** function.

The second section of **main_loop()** (line 597 to line 607) handles the reproduction rules. Every cat checks its neighbourhood for any cats of the opposite sex. Provided the cats are not already interacting with another cat, sleeping, too hungry, or too thirsty, they will reproduce by calling the **reproduce()** function. The births that occur in a given timestep are stored in the list **births**.

In the third section (line 610 to line 618), every cat's probability of going to sleep is calculated, taking into account the time of day and the cat's health. This random chance is then used to put the cat to sleep if they are not too hungry or already interacting with something else.

The final section of **main_loop()** handles movement of the cats. First, each cat's **sleep()** is called so that if the cat is sleeping, it will behave as intended (gain health). The rest of this section only executes if the cat is not already interacting with something and not sleeping. In that case, the possible moves for the cat are calculated using **get_valid_moves()**. If the cat is not too hungry and not too thirsty and able to reproduce (i.e., its mating cooldown is 0), it will try to follow the scent of cats of the opposite sex (line 627 to line 638). This is done by iterating through the list of possible moves and finding the cell with the highest intensity of opposite sex scent. This cell then becomes the only possible move choice for the cat. However, if the cat is too hungry, too thirsty, or not able to reproduce yet, it will try to follow the scent of food or water (line 640 to line 661), depending on whether it is more hungry than thirsty or vice versa. In order to add an element of randomness and prevent all cats from essentially following the same path between food and water sources, cells containing scents area added to the cat's list of move choices based on a random chance that is proportional to the intensity of the scent. Finally, the cat chooses a random move from the list of choices, and moves to that position.

At the end of **main_loop()**, the births of the timestep are added to the list of cats, the cats' colours are set by calling the **set_colour()** method on each of them (males are blue, females are pink, cats are red while fighting, and yellow while fleeing), and their hunger and thirst levels are updated by calling the **hunger_and_thirst()** method.

The **"__main__"** code (line 672 onwards) begins with the required initializations – reading in terrain and landmark data, creating arrays and variables, and initializing **pygame**. Exception handling is also used here (line 680 to line 691) so that the user can supply additional command line arguments if they wish to, with the program still running if they don't. The while loop on line 727 functions as the timestep loop, inside which the **main_loop()** function is called, the arrays and lists are updated, and the screen is drawn.

After the while loop, (i.e., after the simulation is complete), the statistics are printed by calling **show_stats()**, following which the scent toggles are all set to False, and the screen is redrawn. This is so that if an image of the simulation is saved, it will not be cluttered with scent visuals. In lines 767 to 770, a unique new directory name is created using the current date and time, to which the output will be saved. If the user chooses to save the grid state, lines 774 to 804 will execute, making arrays out of the final positions of cats, food, and water, and then saving them as csv files, along with an image of the final frame of the simulation, to the new directory. If the user chooses to save the event log, lines 810 to 819 will run, converting the list **event_log** into a text file and saving it in the new directory.

SweepBase.py is a copy of **Cats.py**, with some changes made to make the parameter sweep run smoothly (such as removing visuals and print statements). It is the base file that will be executed by **ParameterSweep.sh**, the bash script for the sweep. **ParameterSweep.sh** was created using **dosage_sweep.sh** (Maxville, 2021b) from Practical 8 as a reference.

Methodology

There are a number of parameters that the user can change from the command line. The mating cooldown time and maximum uninterrupted sleep hours can be provided as command line arguments, while the type of neighbourhood, number of simulation hours, and initial number of cats are provided as inputs for prompts.

In order to investigate how these parameters affect the simulation, three tests were designed.

Test 1

Number of hours = 100
Initial number of cats = 20
Mating cooldown time (hours) = 24
Sleep hours = 8
Keeping the above parameters constant, one simulation was run using Moore neighbourhood and one simulation was run using Von Neumann.

As the type of neighbourhood changes the number of food or water cells a cat can detect around it, the expected outcome was that the average number of food and water units consumed would be less for Von Neumann than Moore. Additionally, since there would be fewer cat to cat interactions in the Von Neumann simulation, it was expected that the Von Neumann simulation would have fewer cats killed.

The test was performed in the following manner:

1. Type `python3 Cats.py terrain.csv landmarks.csv 20` into the command line (framerate of 20 for efficiency; mating cooldown time and sleep hours default to 24 and 8 if not specified)
2. Enter "M" when prompted for the neighbourhood
3. Enter "100" when prompted for the number of hours to simulate
4. Enter "20" when prompted for the number of cats, then allow the simulation to run
5. Enter "Y" to save the grid state
6. Enter "Y" to save the event log
7. Repeat steps 1 to 6, but enter "V" instead of "M" in step 2

The event logs were then analysed, and the average food/water units consumed and number of killed cats were compared between the two simulations.

Test 2

Neighbourhood = Moore
Number of hours = 100
Initial number of cats = 20
Sleep hours = 8
Keeping the above parameters constant, one simulation was run with a mating cooldown time of 24 hours, and one simulation was run with a mating cooldown time of 48 hours.

With a longer interval between reproduction, it is expected that there would be fewer births in the 48-hour cooldown simulation, and consequently a higher average age.

The test was performed in the following manner:

1. Type `python3 Cats.py terrain.csv landmarks.csv 20 24` into the command line (sleep hours defaults to 8 if not specified)
2. Enter "M" when prompted for the neighbourhood
3. Enter "100" when prompted for the number of hours to simulate
4. Enter "20" when prompted for the number of cats, then allow the simulation to run
5. Enter "Y" to save the grid state
6. Enter "Y" to save the event log
7. Repeat steps 1 to 6, but type "48" instead of "24" in step 1

The event logs were then analysed, and the number of births were compared between the two simulations.

Test 3

Neighbourhood = Moore
Number of hours = 100
Initial number of cats = 20
Mating cooldown time (hours) = 24
Keeping the above parameters constant, one simulation was run with 8 maximum sleep hours, and one simulation was run with 16 maximum sleep hours

As cats recover health while sleeping, the expected result is that the average cat health will be greater in the simulation run with 16 sleep hours.

The test was performed in the following manner:

1. Type `python3 Cats.py terrain.csv landmarks.csv 20 24 8` into the command line
2. Enter "M" when prompted for the neighbourhood
3. Enter "100" when prompted for the number of hours to simulate
4. Enter "20" when prompted for the number of cats, then allow the simulation to run
5. Enter "Y" to save the grid state
6. Enter "Y" to save the event log
7. Repeat steps 1 to 6, but type "16" instead of "8" in step 1

The event logs were then analysed, and the average health of the cats were compared between the two simulations.

Results

The full outputs of all three tests are saved in the Tests directory, for further reference

Test 1

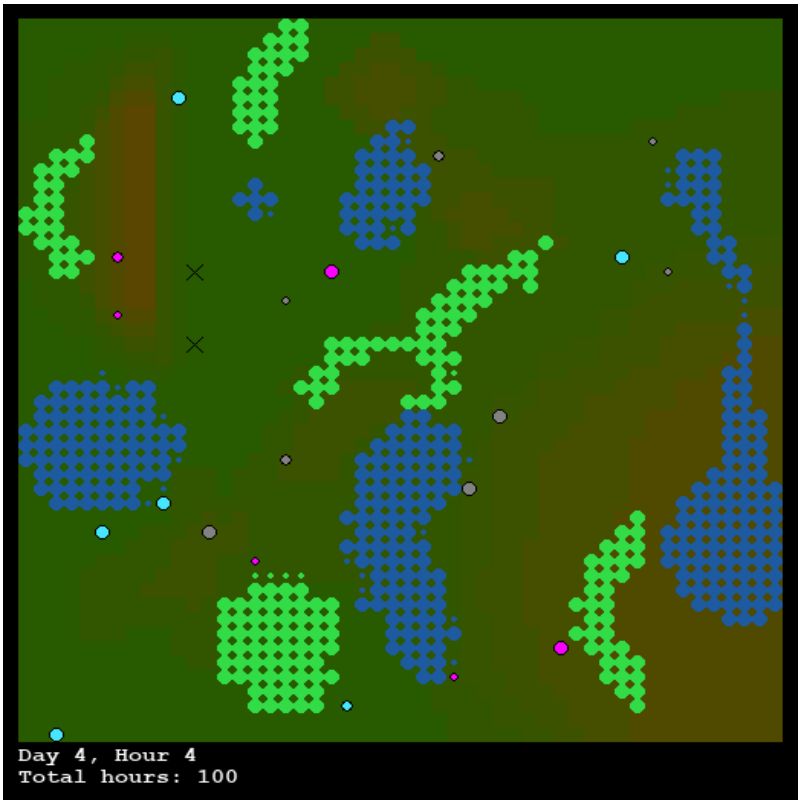


Figure 1: Moore neighbourhood simulation after 100 hours

Figure 1 above shows an image of the final frame of the first simulation of Test 1, which used the Moore neighbourhood. It can be seen that two cats have died, and a number of food and water cells have been consumed. Below are the statistics for average number of food/water units consumed in this simulation, taken from the event log produced.

```
Average units of food eaten by a single cat: 0.39
Average units of water drunk by a single cat: 1.57
```

The number of log updates that showed a cat had been killed was counted, and recorded to be 2.

Below are the image and statistics for the simulation that used Von Neumann neighbourhoods.

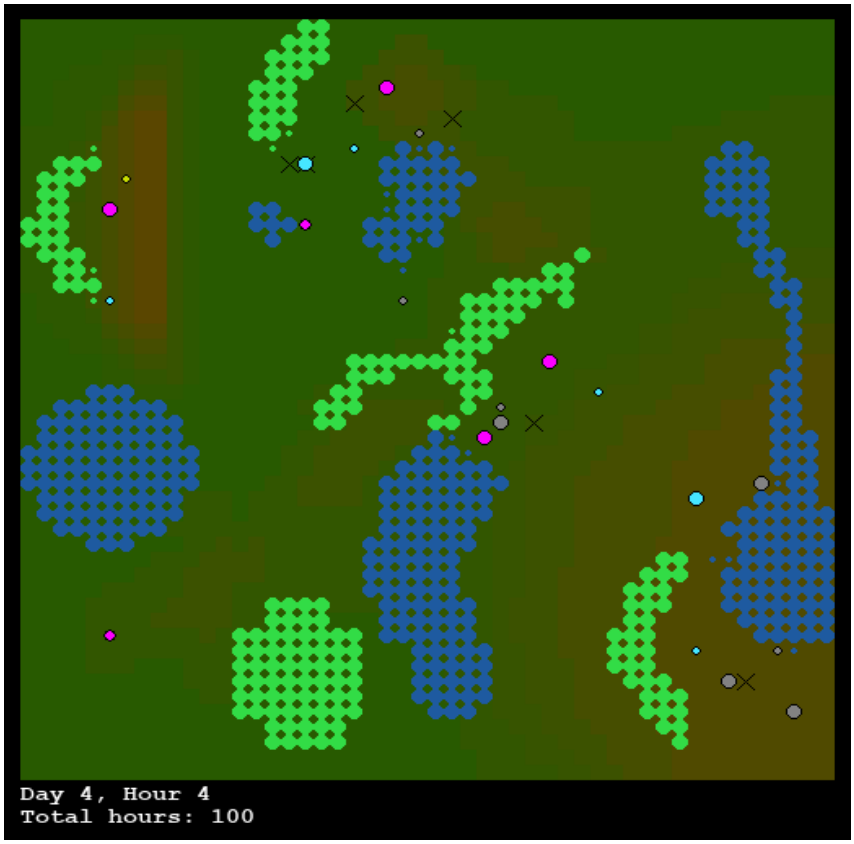


Figure 2: Von Neumann neighbourhood simulation after 100 hours

```
Average units of food eaten by a single cat: 0.26
Average units of water drunk by a single cat: 0.81
```

6 cats were killed during the Von Neumann simulation.

Upon comparison of the two simulations, it can be seen that the number of food/water units consumed is lower in the Von Neumann simulation - 0.39 food and 1.57 water for Moore, and 0.26 food and 0.81 water for Von Neumann. This was in fact the expected result, as cats can detect fewer surrounding cells under the Von Neumann neighbourhood.

Interestingly, however, the number of cats that were killed is greater under the Von Neumann neighbourhood. This is contrary to the expected result, indicating that there may be little or negative correlation between the number of detectable neighbouring cells and number of fights that occur.

Test 2

The first simulation in Test 2 was with a mating cooldown time of 24 hours, the results of which are presented below.

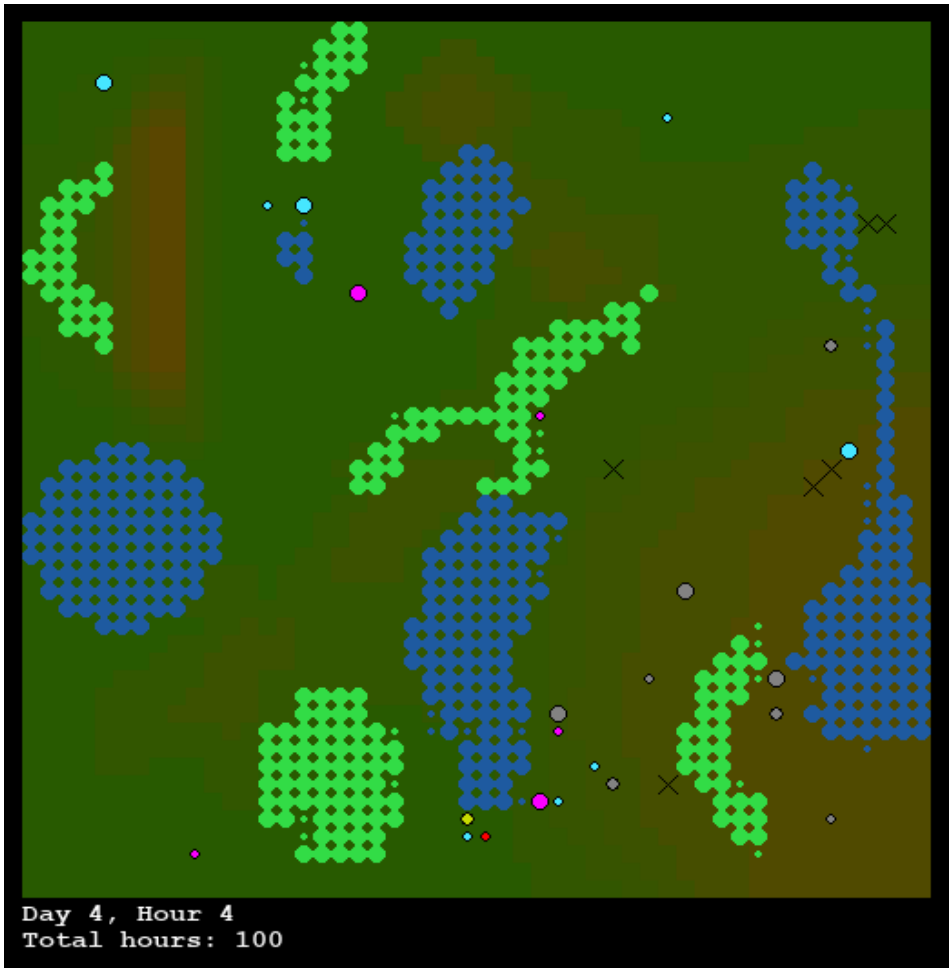


Figure 3: 24-hour mating cooldown simulation after 100 hours

An interesting note about Figure 3 above: In the southern region of the environment a cat can be seen fleeing (in yellow colour) from an attacking cat (in red), which is evidence of the cat interactions functioning as intended.

The event log was inspected to find the number of births that occurred during this simulation, shown below.

```
Births: 9
```

The simulation was then run again using a 48-hour cooldown time, the results of which are presented below.

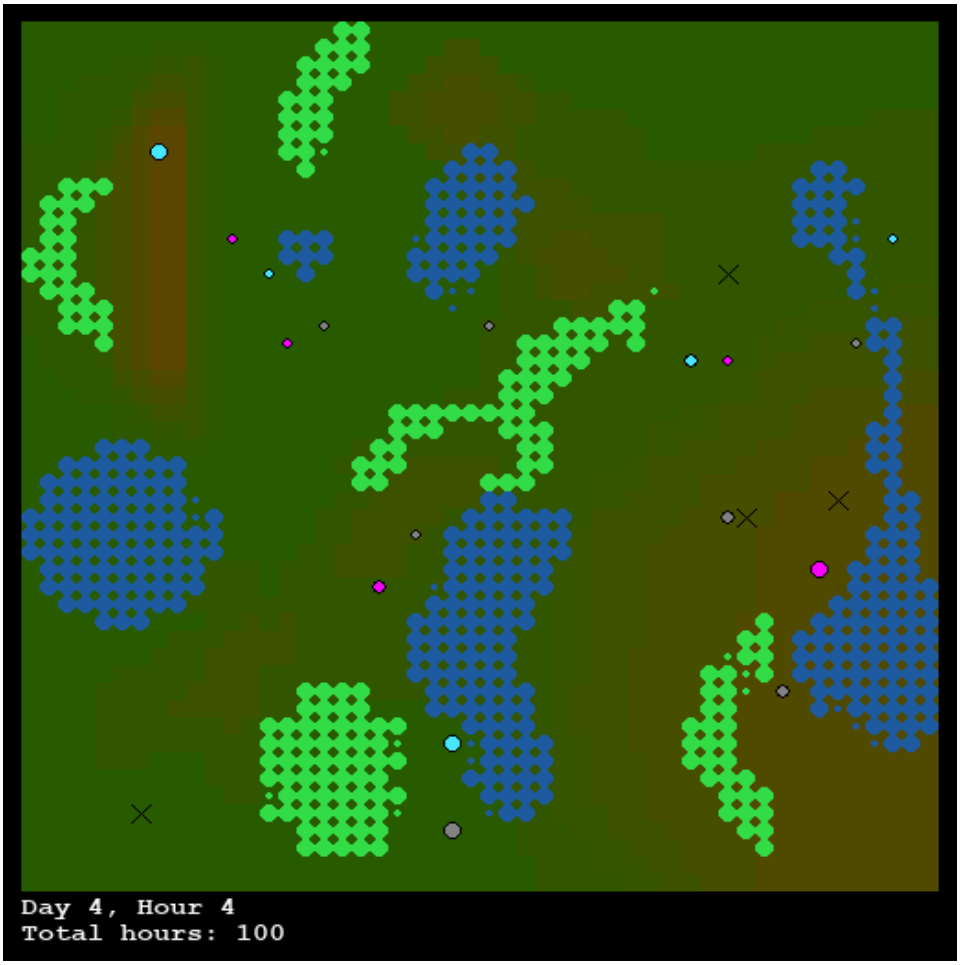


Figure 4: 48-hour mating cooldown simulation after 100 hours

```
Births: 1
```

The outcome of this test agreed completely with the expectation. Nine births occurred with a mating cooldown time of 24 hours, whereas only one birth occurred with a cooldown of 48 hours. This shows that there is indeed an inverse proportionality between the two variables, and that the functionality of the code is as intended.

Test 3

In Test 3, a simulation was first run using 8 maximum uninterrupted sleep hours for each cat.

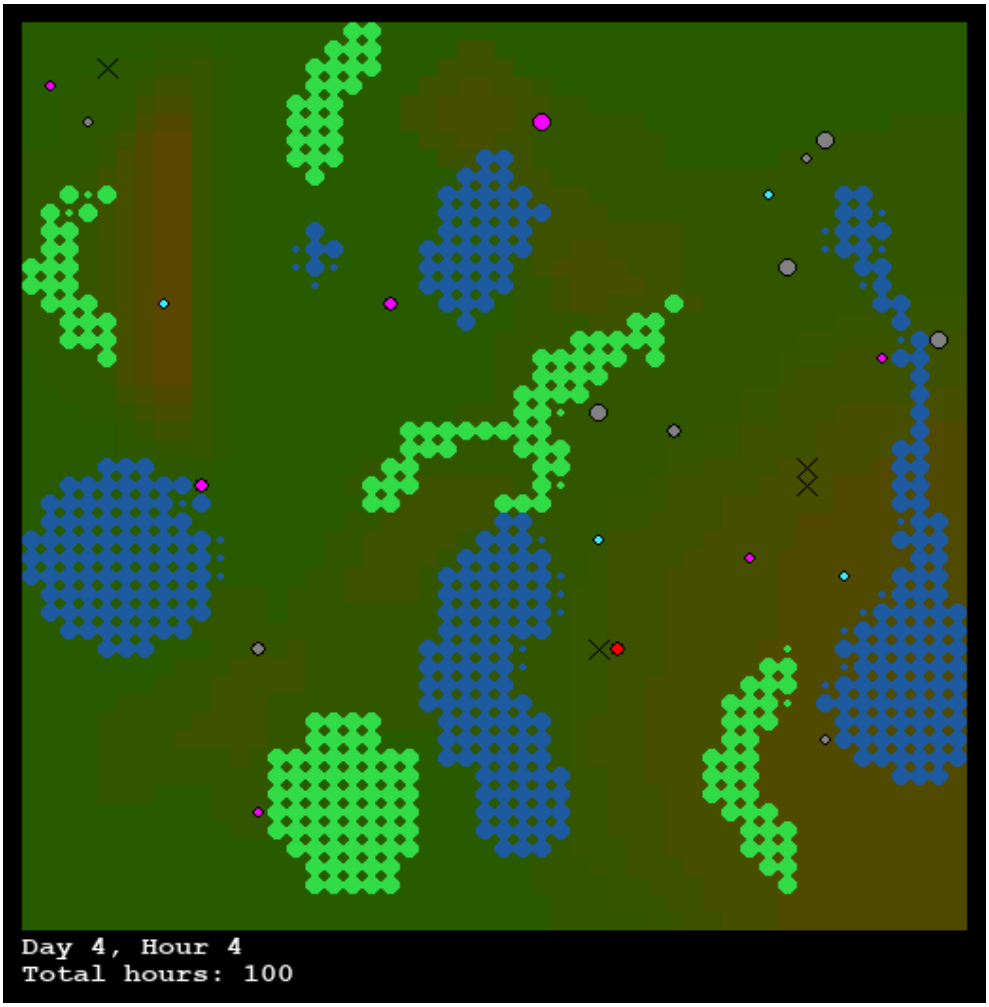


Figure 5: 8 sleep hours simulation after 100 hours

The average health of the cats at the end of the simulation was extracted from the generated event log, and is shown below.

```
Current average health of cats: 98.05
```

The simulation was then run with 16 maximum hours of uninterrupted sleep for each cat, the results of which are shown in the following page.

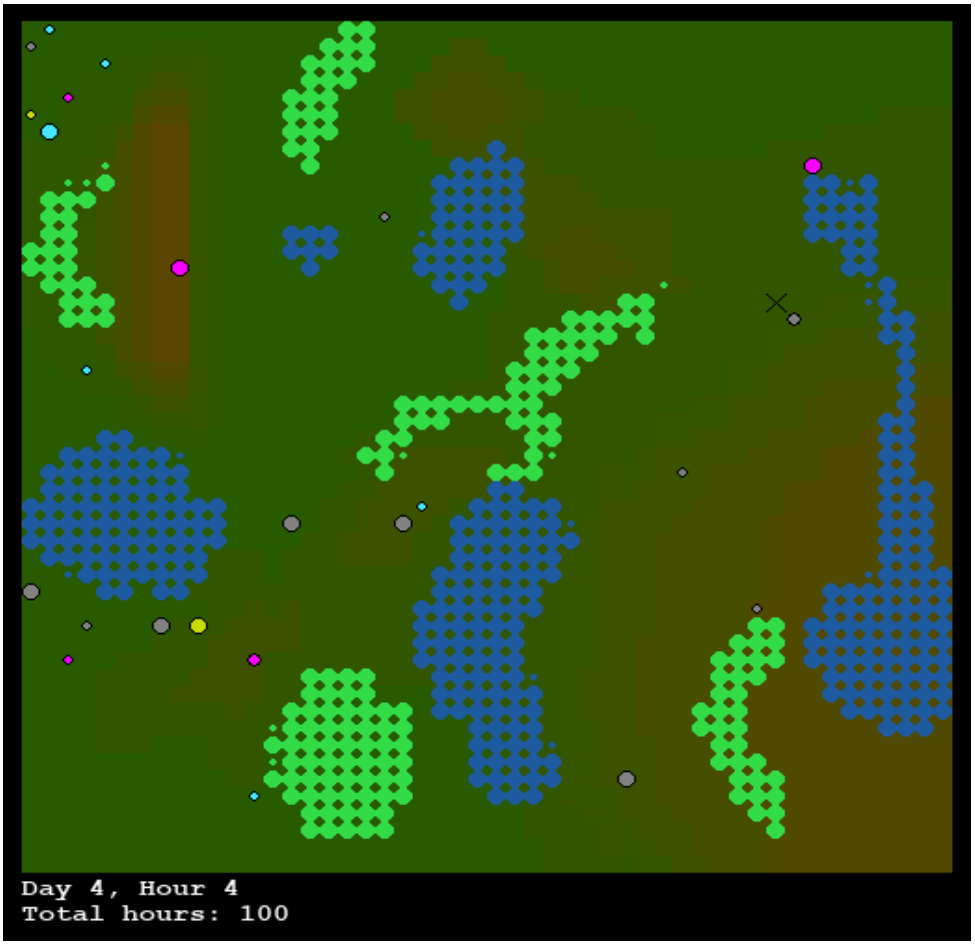


Figure 6: 16 sleep hours simulation after 100 hours

```
Current average health of cats: 99.58
```

The average cat health in the 16-hour simulation is greater than that of the 8-hour simulation, which agrees with the expected result. This confirms that the number of sleep hours is proportional to the average cat health at the end of the simulation, all other factors being equal. This conclusion holds up to a surface level analysis as well, because if the cats sleep for longer, it follows that they have more time to recover lost health.

Conclusion and Recommendations

Cats.py is a simplistic approach to simulating the behaviour of cats in the real world. While it incorporates a host of rules and decision-making algorithms, it lacks the complexity of interactions in nature. This is due in part to time-based restrictions and technical capability.

Nevertheless, the program succeeds in its core endeavour: applying a rules system to objects in a discrete environment. All of the rules that were planned out initially were implemented and tested successfully, such as the pathfinding algorithm and cat to cat interactions. The variable parameters in the program also allowed for an exploration of the effect that certain factors have on the population of cats. The Methodology and Results sections of this report detail how fair tests can be devised in order to investigate these effects, by keeping the parameters constant and varying one at a time. In performing these tests, it was established that cats consume more food and water using a Moore neighbourhood, the number of births is inversely proportional to the mating cooldown time, and the number of maximum sleep hours is proportional to the average health of cats. The tests served an additional purpose of reaffirming the functionality of the code, and with further sophistication, the model could prove to be a versatile tool in animal behavioural analysis.

There are certainly many features of the program that can be expanded upon in future iterations The following are few such improvements:

- Making cat scents diffuse into the environment for increased realism
- 3D visualisation of the terrain
- More user interactivity, such as adding cats or placing landmarks using the mouse
- Improving the visuals by using sprites or 3D models for the cats
- Extending the model to include different animal classes
- Implementing a predator/prey mechanic

During the development of the program, significant research was required to effectively manipulate python objects and numpy arrays. The knowledge and experience gained through implementing all the features of the application will no doubt be valuable, and form a strong foundation for future programming projects.

REFERENCES

Maxville, Valerie. 2021a. “heat.py” Practical 5, COMP1005 Fundamentals of Programming, Semester 2, 2021

Maxville, Valerie. 2021b. “dosage_sweep.sh” Practical 8, COMP1005 Fundamentals of Programming, Semester 2, 2021