Document version: 1.1 (2015-11-15)

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:	Wickramasinghe Mudalige	Student ID:	20469160
Other name(s):	Naradha		
Unit name:	FOP - COMP 1005	Unit ID:	COMP1005
Lecturer / unit coordinator:		Tutor:	
Date of submission:	11th October 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:

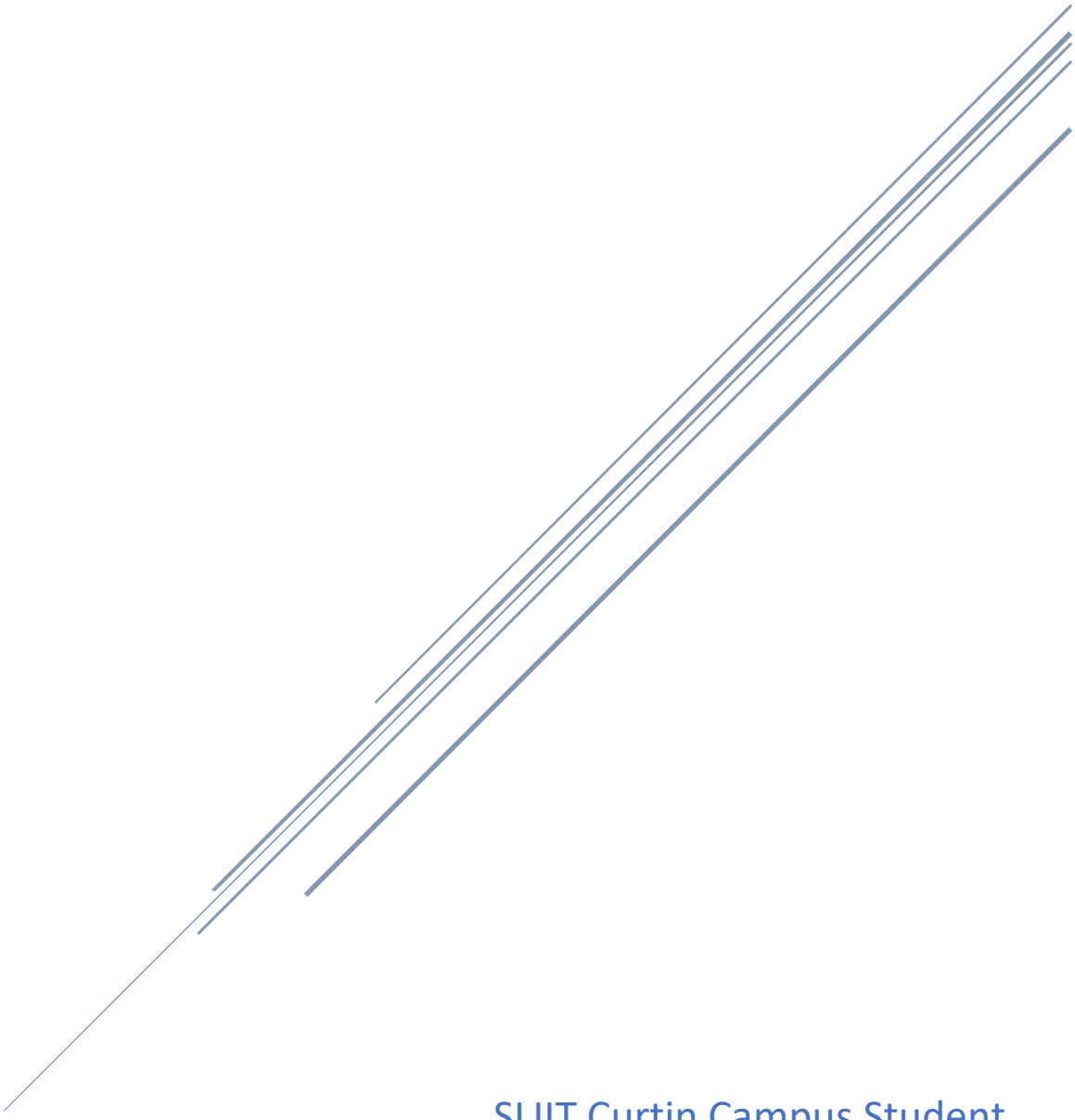
- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which 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.)

GAME OF CATS

By: W.M.Naradha STUDENT ID:20469160



SLIIT Curtin Campus Student
FOP COMP 1005

Contents

Overview: 1
User Guide: 3
Setting up the simulator: 3
Input commands for simulation: 4
Terrain map: 7
Steps to making a simulation: 7
How to do data sweeps to study the change of various in game object attributes? 8
Information provided in plots: 9
Show attributes: 9
Exceptions and how to work around them: 11
Traceability Matrix: 12
Showcase: 17
Introduction: 17
There are three main update functions 18
How the above functions are integrated to each other: 19
Methodology involved to run the simulations of the results section: 20
Results of extensive simulations: 21
Simulation for Cat Landmark Interactions and Cat Death Simulations. 21
Methods affiliated to above simulation: 22
Cat Stress how it randomly changes the requirement. 24
Cats Stress from the simulation CLI simulated for a week: 24
Cats Stress from the simulation CLI simulated for a day with attributes affecting it: 24
Modeling the panic and stress of the cat: 25
Cat – Cat Interactions (Mating, Parenting and Fighting) (Focus on 3 different simulations) 27
Simulation of Cats mating, conceiving and parent status: 30
Plotting Male cat Scent, Female Cat Scent, Female Cat Conceived day count and CPS Change 31
Multiple Cats with terrain: 33
Case 1: 33
Case 2: Cats Terrain and everything: 34
Overall Results: 36
Conclusion and Future work: 37

Overview:

The main aim of this game of cat's model is to first and foremost simulate the cat's behavior accurately for a defined set of rules within a certain timespan set by the user. Secondly allow the user to setup a scenario/simulation depending on his/her requirements which can be called and simulated accordingly. Thirdly make the program setup such that the user can simply be able to execute, visualize and extract simulation data effectively.

First it is important to identify the behavior of a cat the attributes that lead to such behavior of a cat, which can then be applied to develop the simulation program. In this model of game of cats, the focus is on domestic cats and their behavior. A cat if observed in an isolated environment would eat when hungry, drink when thirsty and sleep whenever it is tired, given that it is provided food, water, and some comfortable resting area a bed for example. When we bring multiple cats into the space of concern there would be fights over the food, water and sleeping spots. If it was further observed mating, conceiving, and parenting are also ways cats interact with other cats. It is quite easy to now identify the attributes of the cat to model interactions with non-cat entities, while cat-cat interactions are rather complex scenarios.

However, to put the simulation in simple terms, we can setup certain rules and conditions that can be seen in the simulation.

1. A cat will have different priorities based on its hunger, thirst, scent, and stress levels, which will drive the cat to move towards the priority of the moment, if there is no priority it will do nothing at all for as long as it can.
The model used for hunger, thirst, tiredness, scent, and stress are all simple linear or quadratic mathematical models based on the age of the cat to change the respective attribute based on time.
2. Cats when near landmarks of priority (water, food, bed, box) can interact with them in the case of food and water from a neighboring cell, in the case of a bed or a box can interact by getting into the bed or box cell.
3. In cat-cat interactions a cat with the need to go to a position of advantage/value can challenge a cat occupying such a position, this can lead to a linked fight with multiple cats fighting for the same position on the grid or for consecutive positions, it is important to note that the cat holding the position can ignore, give away, accept to win, or lose and even ghost/cram.
4. In the case of mating, cats have a scent which describes the readiness and intensity of sexual stress of the cat which would allow two cats of different genders meeting on the same cell to reproduce. However, it is important to note that in this simulation to allow mating and reproduction a box landmark is required, when time comes the kittens will be put into the box.
5. The kittens will come out of the box after a period of 1 year in the simulation time until which the female cat would nurse the kittens. Note that this component was setup with the idea of setting up some form of model to simulate parenting and conceiving kittens, while it does not completely depict the reality.

This provides an overview of the purpose and the respective components affiliated to that purpose in the simulation. It also provides how the rules that dictate the simulation are setup.

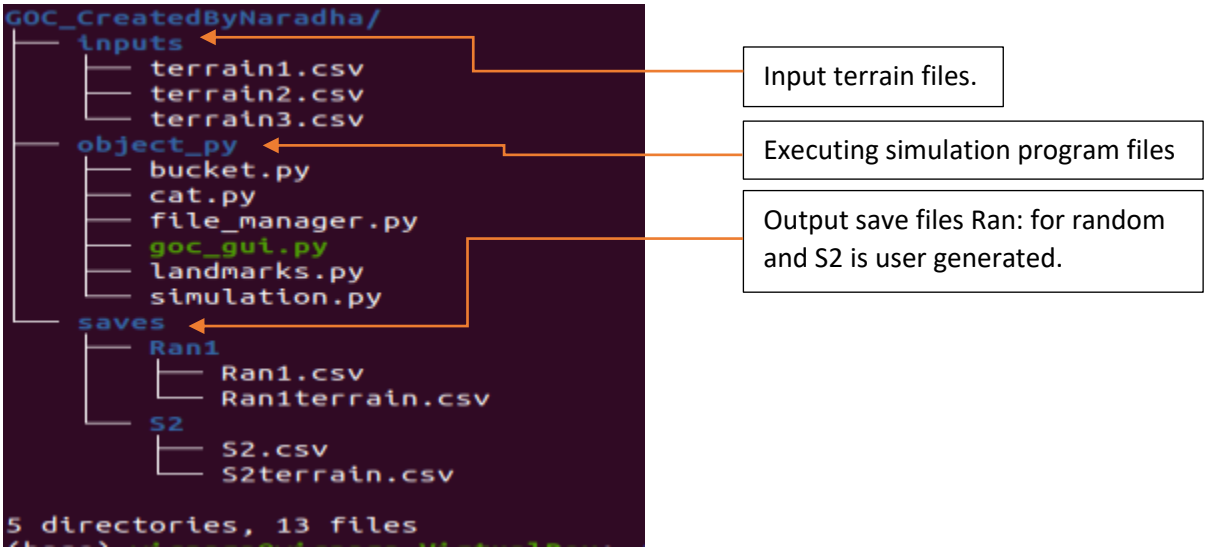
The main feature of this model would be the object-based simulations and buckets, which are the main code components that make the functionality of this model possible. In a summary the simulation object handles everything from a perspective of the entire simulation space, while the bucket handles everything from a perspective of a single cell. The code used to interact with these respective programs would be on the user end. This provides the user so many possibilities as he/she can create a simulation of their will add cats/landmarks and more. Some additional features that allow the user to visualize these specific interactions are directional landmarks, scent-based movement and the matplotlib canvas with Tkinter based plotting GUI to visualize changes between time frames for various components in the simulation. A simulation can be saved, run for a unit time, run and visualized Realtime or just provide a csv based simulated output without any visual output and quit the program. The code uses a tailor-made GUI interface with multithreading to allow the simulation to run at this level of functionality.

User Guide:

Setting up the simulator:

When setting up the program it is recommended to keep the file layout as provided on the default program setup. The user is only expected to copy the main file system into his system. Changing the file layout inside the main system folder would make the system generate exceptions and errors.

File Structure of the Game of Cats:



Before running the python file the user must make sure that the system has a stable running python 3 version on his/her computer. The software uses a handful of external libraries that the user might have to install for the program to work. An easy work around is to setup anacondas' libraries and python interpreter as the default python. As all the additional libraries used are available in the anaconda system. The user can also execute the setup.sh file to install anaconda directly from the web to setup the required libraries. It is provided with the README file in the main directory GOC_CreatedByNaradha. It is important to make sure that the user sets anaconda up in the default library given and initialize conda_init_ as well. Pip installing the libraries is not sufficient as tkinter cannot be installed using pip.

External Libraries Used (Available in Anaconda):

1. numpy
2. matplotlib
3. Tkinter (In this case we are using Tk but tkinter is compulsory)

Other libraries (Available in Python by default):

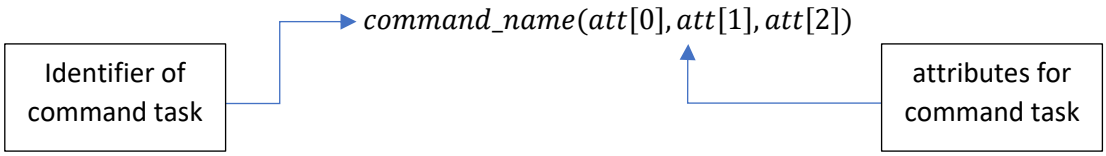
1. threading
2. random
3. math

The user can change the shebang line on the top of the code to keep the executable file working in the user's environment as well. Note it is important for the user to provide the anaconda-based interpreter or the interpreter which has the above libraries in its path.

Input commands for simulation:

The model is based on a command line-based input, there are a set of commands that allow the user to provide the necessary data to load up a previous simulation or run an entirely new simulation. Before going through how to make a simulation it would be key to understand the commands and their attributes.

Command format:



All the commands used in the simulation, are of a similar format. Hence, understanding the format better would allow the user to use the program at ease. The attributes must be provided in the correct order and of the correct type as it is just a string taken in to identify specific input.

Command List:

Command example	Attribute Meanings
create_sim: Used to create a simulation from scratch. *create_sim(name,type,spd,rows,cols,map)*	name: unique identifier for the simulation.
	type: Whether the simulation uses 'Moore' or 'Neuman' neighbor analysis.
	spd: stands for steps per day this defines the clarity of the simulation. The higher number of steps the more descriptive simulation can be generated.
	rows and cols: provide the maximum number of rows and columns for a flat world simulation if a map is provided just set them as zero.
	map: provide the name of the terrain file make sure it is in the input file of the simulation.
add_cat: Used to add cats to the simulation object list. *add_cat(name, gender, dob)*	name: The name of the cat to be added to the simulation object list. (Provides a unique identifier (ID) to the cat)
	gender: If the cat is Male or Female.
	dob: the date of birth of the cat provided in days relative to the start of the simulation. (e.g.: a cat born 10 days before the simulation would have a dob of -10). Note: providing positive dob for a cat at the start of the simulation would simulate a cat with negative age which leads to discrepancies.

add_landmark: Used to add landmarks to the simulation object list. *add_landmark(name,type,quantity,reset,direction)*	name: The name of the landmark to be added to the simulation object list. (Provides a unique identifier (ID) to the landmark)
	type: Whether the landmark is a 'food' type, 'water' type, 'bed' type or 'box' type.
	quantity: quantity of food/water in the landmark given in grams if the landmark is a bed or a box the quantity describes the elevation to enter or exit the box in meters.
	reset: If the landmark was water or food how many times a day should it refill. (When set as zero will never refill.)
	direction: provide the direction from which a cat can enter / interact with the specific landmark.
add_grid: Used to add objects in the simulation object list to cells/buckets on the simulation grid. *add_grid(row,col,name)*	name: Provide the unique name of the object to be added to the grid.
	row and col: Provide the exact position on the grid that the object is to be placed into.
save: Used to save the simulation currently used. *save()*	No attributes, still it is important to make sure that there is a simulation to be saved, which other wise would raise an exception.
load: Used to load a simulation saved beforehand. *load(name,fps)*	name: unique identifier of the pre-saved simulation in the save's directory of the program paths.
	fps: expected frames per second visual output for the user. Given in frames per second ex: 5.
set: Used to set a simulation attribute. *set(fps,30)* *set(speed,10)* *set(attribute,ALL)*	If the first attribute is fps, the user would set the visual outputs fps to a new value.
	Speed sets the number of iterations run before generating a visual output, for instance if set to two would show every second visual output.
	attribute sets the visual attribute the user wishes to see. There are a handful of attributes the default set to 'ALL'. **Show Attributes:** **ALL:** Show all the objects and their attributes in respective colors. (Markers will be explained below) **Terrain:** If a map is loaded will show the terrain in blues color map with markers for the objects. **Cat-Scent:** Shows cat scent propagation with a red for male and blue for female, and purple for a cell where they are mating. **Food-Scent, Water-Scent, Bed-Scent, Box-Scent:** Shows the scent propagation with a hot color map and setup of the simulation with markers for the objects.

play_sim: Used to play the simulation currently loaded/setup in realtime. *play_sim(attribute)*	This command has a provided attribute argument that can be provided if not set above. This is provided for ease in real time simulations. However, it is recommended not to fool around by pausing and loading different attributes as it would cause exceptions in the tkinter thread.
play_for: Used to run a simulation for a given number of steps and provide output as csv and visually on the canvas. *play_for(steps,attribute,visual,log_output, plot_save,loop,delay,close)*	steps: The number of steps the user needs the simulation to run for this must be given as a unit of steps per day, provided that if the simulations steps per day was 100 to simulate 10 days 1000 steps must be simulated.
	attributes are the visual output required by the user incase they want the visual set to on.
	visual, log_output, loop, close and plot_save: These are all requesting an input in the for of 1 or 0. 1 provides that the task must be executed if 1 for visual the simulation will be shown visually after completion. **visual**: should the output be shown through the canvas. **log_output**: should the output be provided as a csv log. **loop**: should the visual output loop and delay sets how long it should loop. **plot_save**: provided as 1 for the plots of the simulation to be saved.
	delay: provides the time for delay in seconds inbetween visual loops.
pause_sim: Used to pause a simulation currently played on the canvas in realtime or from play_for(next command). *pause_sim()*	It will only pause the simulation if real time, else if it were a play_for simulation the simulation is complete and the visual out put either on loop or not will be stopped.
random: Used to run a simulation for a given number of steps and provide output as csv and visually on the canvas. *random(type,spd,rows,cols,map,cats,landmarks)*	type: Whether the simulation uses 'Moore' or 'Neuman' neighbor analysis.
	spd: stands for steps per day this defines the clarity of the simulation. The higher number of steps the more descriptive simulation can be generated.
	rows and cols: provide the maximum number of rows and columns for a flat world simulation if a map is provided just set them as zero.
	map: provide the name of the terrain file make sure it is in the input file of the simulation.
	cats: number of unique cats to be placed on the grid, their attributes will be generated in random.
	landmarks: number of unique landmarks to be generated and placed on the grid, their attributes will be generated in random.

Terrain map:

The terrain map is a csv file which describes the nature of the surroundings specifically how the terrains heights change from one section to another. Even if the terrain is uneven, it is important to make sure that the shape is inside a regular rectangle or square. The terrain map can be generated using vim editor or even liber software in the ubuntu setup. It is important to maintain that a cell of the map represents a cat and its personal space.

In order to use a terrain map in the simulation, the terrain map must be added into the input file, which would allow the user to make a terrain map from it.

Steps to making a simulation:

It is recommended to go through the input commands for the respective step from above accurately as making sure the syntax and order of attributes to the command are accurate, would be challenging otherwise.

Step 1:

Open the goc_gui.py python file through the terminal using python3 or as an executable. You are free to give the input here as system arguments as well, but in string form and should be commands of correct syntax. The commands and the respective attributes are provided above.

Step 2:

After opening the user input interface, to create a sim the user can use the create_sim() command with the respective arguments to create a simulation with a terrain map added to the input folder or a flat grid by providing the grid dimensions. The user can also load a previously saved simulation from saves directory automatically provided the name of the simulation through the load() command with fps.

Step 3:

Adding Cats and landmarks to the created/loaded simulations object list follow the same process. When making a new cat/landmark the user is allowed to provide a name, dob for the cat and gender from the add_cat() command, while for the landmarks the user is allowed to provide a name, quantity, a reset time and direction if any using the add_landmark() command. It is important to note and provide unique names for the objects as they stand as a form of identification for that object.

Please note that when naming cats and landmarks do not name them as c0, c1 or l1, l0.
(Default in program names)

Step 4:

Adding items in the object list of a simulation to the grid to use the add_grid() command with its respective arguments and the object identifier(name of the object).

Step 5:

Running the simulation, to use the Realtime play feature the user can use the play_sim() command with the main attribute being only what the user wishes to see, weather it is the terrain, just the objects or individual scents. The user can use the play_for() commands with the respective arguments which would allow the user to run the simulation for a specific timespan and show it after the simulation is completed and write simulation data into a log file if necessary.

Step 6:

The user can choose to save the simulation using the save command, which would save the simulation in a directory under its name as a csv file which can be loaded up again. To quit the program the user can use the quit() command first and then use ctrl-c to stop the main thread in the terminal.

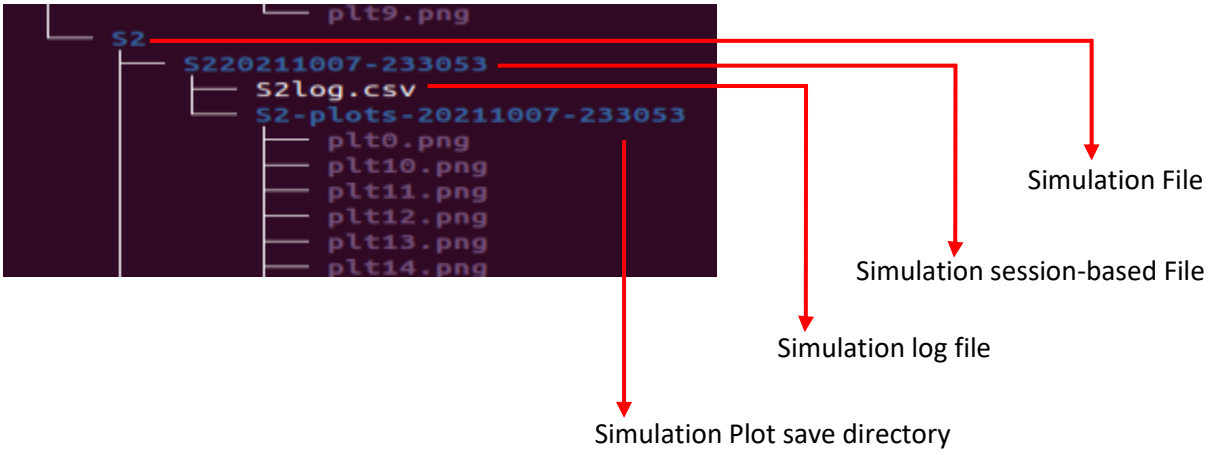
Please note that when saving not to use Ran as the simulation name.

In the case that a simulation is loaded or a random simulation is called the user can straight away use the play_sim() or play_for() commands to provide input attributes. Similarly, data sweeps can be handled by integrating a workflow into this system as the interface file does accept and process system arguments. It is also worth noting that in using the terminal if an exception is raised it will be printed on the terminal if handled, making providing terminal input difficult in such cases. Hence, it is either recommended to provide input through system arguments or user command interface of the app.

How to do data sweeps to study the change of various in game object attributes?

As commands to run a simulation can be provided as strings through system arguments the simulation program can be hooked up to an entirely automated workflow for data sweeps. The simulation program already does provide a csv log with the hunger, thirst, fatigue(tired), sexual stress and stress of cats with location and quantity of landmarks. The requirement for the user is a workflow system based on the specific attribute data that the user wishes to extract and visualize. The simulation also provides a file with the cat's motions and landmarks plotted directly into it providing the user additional information that can be used for data extraction and data sweeps as well.

Using the random command with play_for and the quit with proper attributes is an easy way to load the program and simulate for a given amount of time steps. The plots and csv logs are created for every simulation executed using the play_for session based on the attributes given.



Information provided in plots:

To read the plot data, it is important to understand how the simulation plots the information and what each icon on the screen depicts. To read the plots, first it is important to know the different types of plots the user can generate using the program.

Markers and what they represent:

Cat Markers:


-Cat marker


-Drinking Cat marker


-Sleeping Cat marker


-Eating Cat marker

Landmark Markers:


-Food marker


-Water marker


-Bed marker


-Box marker

There are more different markers, that are shown in the results section:

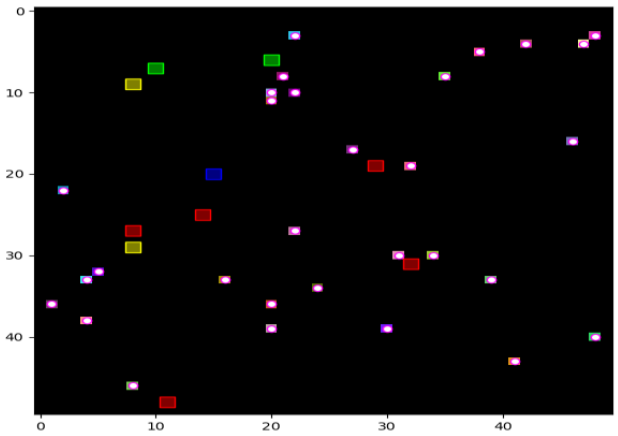
Show attributes:

- All:

  Provides a plot of the cats and landmarks, the marker defines whether it is a cat or a landmark. The color of the tile occupied by the object depicts its attributes.

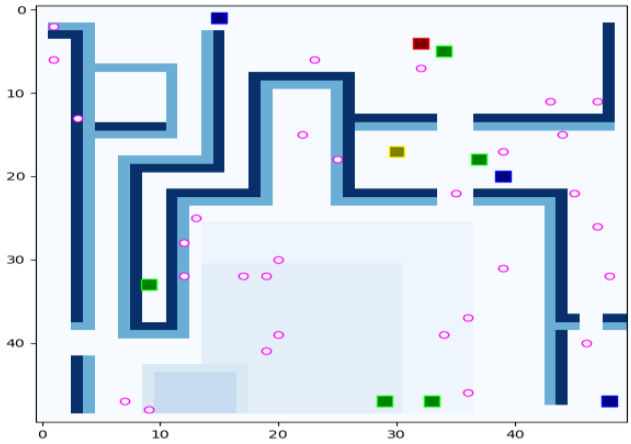
  The landmark color of the tile shows the type of landmark and intensity shows the remaining quantity. Blue landmarks are water type, green landmarks are food type, red landmarks are bed type and orange landmarks are box type.

  The cat color of the tile shows the hunger, thirst, and fatigue(tired) corresponding to red, green, and blue of the tile.



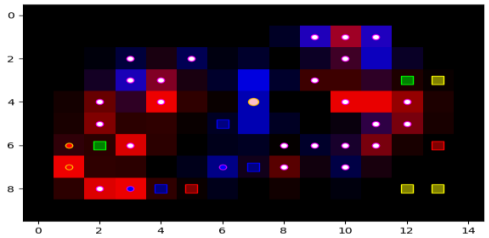
- Terrain:

  Provides a plot of the cats and landmarks, the marker defines whether it is a cat or a landmark. The color of the tile occupied by the object depicts the terrain cell height. The height is presented in blues color map, provided that the height is proportional to the intensity of blue and intensity is taken relative to the minimum and maximum height of the terrain.

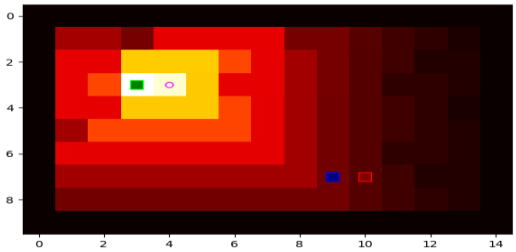


- Scent maps:

  Provides a plot of the scent propagation of the specific source specified by the user, the cats and landmarks will still be shown using markers as presented above. Hear the scent propagation of landmarks is presented using the color map hot. The cat scent is presented using red for male and blue for female cats, in an RGB scale.



Cat Scents | Blue: Female Cat Scent | Red: Male Scent



Food | Water | Beds | Boxes

Exceptions and how to work around them:

However foolproof a system is errors and exceptions do happen, how does the user work around them and make sure they are not raised. Most command syntax-based exceptions that are raised can be ignored if the exception is not from the thread. In the case that the exception is raised from the thread when running the simulation, it is recommended to restart the simulation again. Most of the time an error/ exception would be raised from the system if the commands were not provided in the correct sequence, for instance if an add_grid, add_cats or add_landmarks command was executed before loading a simulation would raise an exception as there is no simulation object loaded, while playing a simulation when not loaded would also raise an exception. In case an exception that leads to crashing the session, it is important for the user to restart the software and start work again. It is recommended to save the simulation sessions expecting a unexpected error from the system.

This brings the end of the user guide; it is recommended for the user not to change code in the four object code files and the main user interface file. However, the user can change the shebang line on the top of the code to keep the executable file working in the user's environment as well.

The use of three threads to handle different parts of the process allow the user to provide commands while a simulation is running, however note not to execute commands when a play_for command is executed.

Traceability Matrix:

The table below provides the Main features and sub features as a list and refers to the respective code components and sections used for implementation in code. The third column shows the amount of testing done and how the results were validated.

Main Feature: Object Behavior of Cat	Implementation in code:	Tests:
Sub Feature: ✓ Implemented model to simulate cat age relative to dob provided by	Functions defined in the Cat object to get age. Cats dob (date of birth): Given as an object instance variable.	Tested more than 10000 iterations is printed into the csv log.
✓ Implemented model to simulate cat hunger.	The hunger attribute assigns the hunger and the hunger_calc function in the cat object updates the hunger.	Tested more than 10000 iterations is printed into the csv log. Can also be visualized from the plots color. Separately tested as well.
✓ Implemented model to simulate cat thirst.	The thirst attribute assigns the thirst and the thirst_calc function in the cat object updates the hunger.	Tested more than 10000 iterations is printed into the csv log. Can also be visualized from the plots color. Separately tested as well.
✓ Implemented model to simulate cat fatigue.	The tired attribute assigns the fatigue and the tired_calc function in the cat object updates the hunger.	Tested more than 10000 iterations is printed into the csv log. Can also be visualized from the plots color. Separately tested as well.
✓ Implemented model to simulate cat stress. To make the cat act at random and show mood like modes.	The stress attribute assigns the stress and the stress_calc function in the cat object updates the hunger.	Tested more than 10000 iterations is printed into the csv log. Separately tested as well.
✓ Implemented model to simulate cat sexual stress.	The scent attribute assigns sexual tension and the scent_calc function in the cat object updates the hunger.	Tested more than 10000 iterations is printed into the csv log. Can also be visualized from the plots color. Separately tested as well.
✓ Implemented stress-based decision making of the cat. To make the cat act at random and show mood like modes.	The requirement attribute of the cat class sets the requirement of the moment. The CIBS functions resets/assigns a requirement based on the cats stress level.	Tested more than 10000 iterations is printed into the csv log. Separately tested as well.

Main Feature: Movement		
Sub Feature: ✓ Implemented movement based on the user preferred simulation type and stress-based requirement identified by the cat. User can make simulations to run in Moore or in Von Neuman.	The move_cat function in the cat object is the main function handling cat motion. It takes the scent information from the neighboring cells and identifies the next bucket to move to depending on the requirement. It does not move into obstructed/occupied buckets. The move_to function is what changes the cats cell.	Tested over 10000 iterations and the cat's motion can be visualized from the plots. Initially in debugging used print statements in the move_to function to double check movement.
✓ Implemented cat jump height based on age of cat.	The jump function of the cat returns the height it can jump based on the age of the cat.	Tested over 10000 iterations and this motion can be visualized from plots with terrain simulations.
✓ Implemented movement based on the terrain given by the user and cat jump height.	The jump function above is used in the move_cat function to check before considering a cell if the cat can move to the cell.	Tested over 10000 iterations and this motion can be visualized from plots with terrain simulations.
Main Feature: Terrain and Boundaries		
Sub Feature: ✓ Implemented reading user given terrain to the input file, given as a csv of height values given as integer or float in meters.	This is implemented in two places first it is in the file_manager python file as the read_csv function, which is called into the simulation class constructor where the csv is both read and assigned to generate the custom grid through the gen_grid function. The type is set to float and hence manages integers as well.	Tested over 10000 iterations and this motion can be visualized from plots with terrain simulations.
✓ Implemented terrain-based boundaries can be cat movement-based boundaries and scent propagation-based boundaries. (It is a boundary with sudden changes to restrict scent propagation.)	This is a feature due to the move_cat function restricting cat movement across such boundaries based on jump height. In scent propagation the boundaries can cause irregularities/abrupt changes that decrease the scent propagated to the other side of tall boundaries. This is handled by the update_scents function in the bucket object.	Tested over 10000 iterations and this motion can be visualized from plots with terrain simulations.

Main Feature: Interactions		
Cat-Landmark Interactions: ✓ Implemented cat can eat and drink water from a neighboring food source/ water source.	This is implemented from three functions primarily the CIncO function which stands for cat interaction with non-cat objects and the eat_drink_sim function of the cat object, while the eat_drin_sim function of the landmark handles this on the landmark side. The cat object calls the landmarks function before consuming the food/water.	These functions were setup at the initial stages of the program and have been tested ever since. Used print statements and the color changes in the plot to validate their functionality.
✓ Implemented landmark quantity and scent change based on remaining quantity when interacting with cats. (Applies mainly to food type and water type sources.)	The scent is updated based on the remaining quantity in the landmark this is setup through the remaining function in the cat object and the scent update function in the bucket object.	These functions were setup at the initial stages of the program and have been tested ever since. Used print statements and the color changes in the plot to validate their functionality.
✓ Implemented cat can occupy a bed to sleep or rest in it.	The fatigue level is updated from the tired_calc function unlike the hunger and thirst, while the bucket will be occupied from the buckets point of view.	These functions were setup at the initial stages of the program and have been tested ever since. Used print statements and the color changes in the plot to validate their functionality.
✓ Implemented cat can occupy a box to give birth to its litter.	Handled by the conc_update, cps_update functions of the cat object class, while the box only requires a cat to be able to jump out of it regulated by the jump function and move_to and move_cat functions of the cats in the box.	The function was setup in the last stages of development sufficient testing was done to identify functionality. Used print statements and the color changes in the plot to validate their functionality.
Cat-Cat Interactions: ✓ Implemented challenge and reply to challenge-based cat-cat fights, that depend on cat stress and requirement.	The Fight object handles the fight and simulates the fights, while the fight is started from the chall function and rec_chall function which allow the cat to send a challenge and if accepted a fight object is created, which will the be called by the non-challenger cat.	The function was setup in the last stages of development sufficient testing was done to identify functionality. Gave the most errors in the program that have changed the simulation rules as well originally to compromise.

✓ Implemented cat mating based on the challenge-based setup used in the fights above and map requirements.	The mating function uses the same method as that of the fights, but it has its own mchall and mrec_chall functions which are mainly for mating, and if the mating challenge was accepted the mate function will allow the female cats conc attribute to turn True.	Also, setup in the final stages of development. It also produced multiple errors which were fixed after debugging.
✓ Implemented cat parenting model where female cat will interact with kittens until they are old enough to jump out of the box.	This handled by the cps function that handles the cat parent status attributes both the cps, boxpop and cps_val to implement a sort of parent like behavior for the female cat.	Tested only a handful of times but did work in all those incidents. Can be visualized and referred to from the log. (Takes very long to run simulations that are in sim time 2 years approximately)
Main Feature: Food/Water/Bed/Boxes		
✓ Implemented all 4 landmark types around one main object defined as Landmarks.	There is an attribute in the landmark class called ltype which defines the landmark type and models the object accordingly.	These types were setup in the initial stages of the program and have been tested ever since. Used print statements and the color changes in the plot to validate their functionality.
✓ Implemented model to simulate scent propagation of the landmark in the bucket object which is affected by the terrain as well.	The bucket objects update_scent function handles all the scent related updates. The scents of the bucket are originally set when the objects are added through the add_object function in the bucket, but primarily everything related to scent propagation is handled by the update_scent function.	As of now the scent models are all simple linear models. This is also tested as it is one of the back bones that allows the simulation to run successfully. Can be visualized through plots.
✓ Directional Landmarks. This defines an instance variable on the landmark that sets the direction of the landmark.	The direction is set as an attribute in the landmark. However, the actual directional nature is implemented from the cats directional_checker called by the move_cat function and the directional_checker called by the buckets update_scent function for scent propagation.	It was setup initially for testing fighting of cats and hence is thoroughly tested. Can be visualized through plots.

Main Feature: Visualization/Results		
✓ Implemented different user requirement based visual plots that can be generated. The plots allow the user to visualize different properties of the cats, landmarks, and terrains.	This is handled by around 4 – 5 functions depending on the input command given in the user interface and the primary back bone is the show_step function of the simulation object. It provides a RGB array or a scent map for the user interfaces dynamic canvas to plot through the play_sim and refresh_canvas primarily.	Originally setup to test whether the outcomes are to expectations and for testing. Hence, is widely tested setup in this system.
✓ Implemented a CSV log-based output which can be integrated with data sweeps and workflows to extract respective information.	This log is generated into the file from the user interfaces save function, while the actual log is generated from the log-based functions in the cat and landmark objects combined with the gen log function in the simulation class. The final log is a list of such updates for every timestep in the log attribute of the simulation object.	Generated logs can be seen in the simulation save files.
✓ Implemented ability to load and save simulations that were executed in the system.	The save_sim and load_sim functions use the file_managers write and read functions to generate csv save files that can be re inputted through the load function, while calling the load or save function is handled by the comm_process function in the user interface which handles input commands.	This also tested thoroughly, although as fights are recent additions to the save and loading functions. Such scenarios might lead to errors.

Showcase:

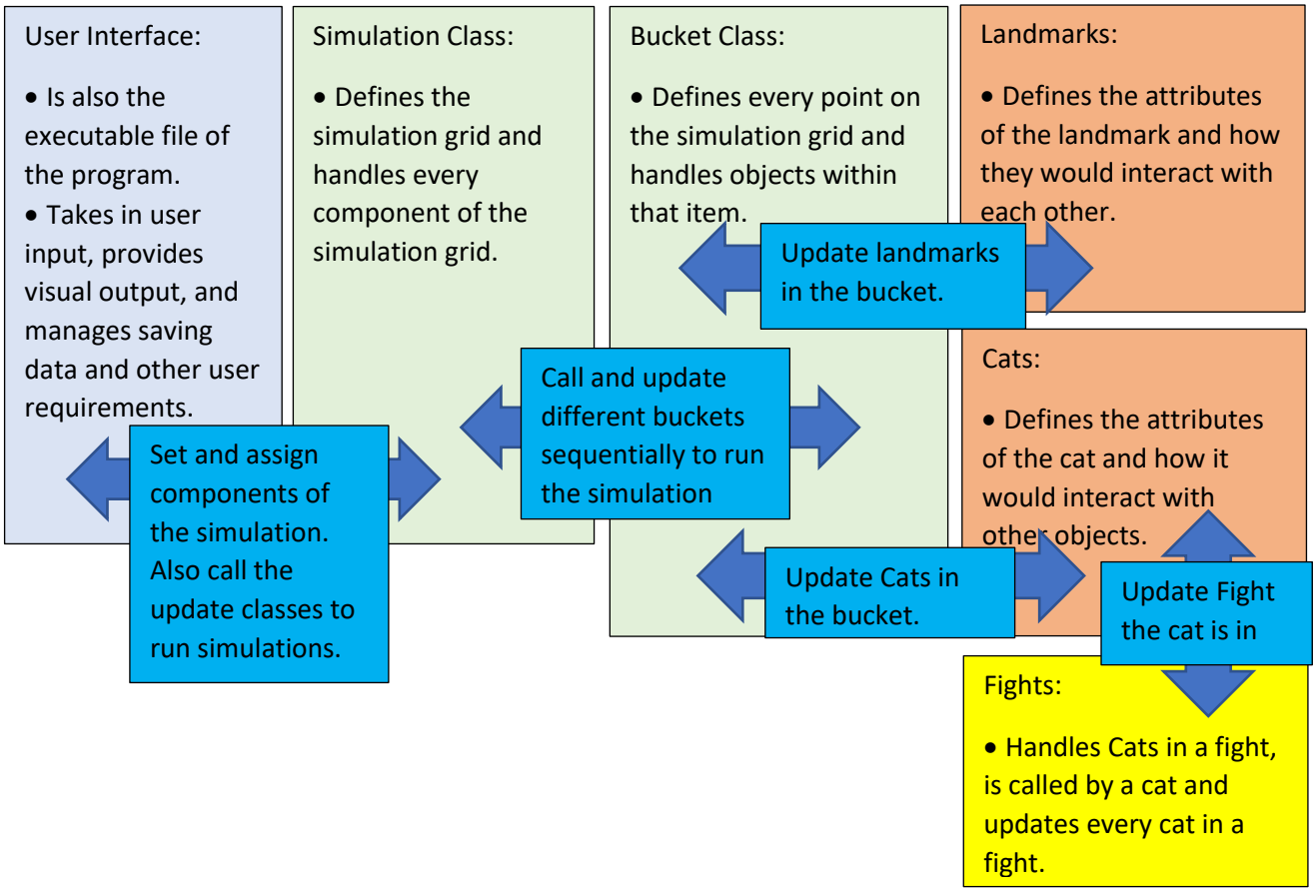
Introduction:

To make this model work in a manner as presented in the above sections, there are more than 1000 lines of code that are controlling and dictating the output step to step. There are 4 object classes that call respective components of each other to make a simulation work according to the given conditions. The simulation class handles all the structures and components of the entire grid including generating output, the bucket class describes a unit cell the objects and items in a unit cell, the landmarks object class describes and provides the attributes of all the 4 different landmark types while the cats object class handles how the cat is to manipulate itself to any given situation and how it would interact with the components on the grid.

The user interface is just the tip of the iceberg which the user can see and interact with. It also handles most of the exceptions in the code, but there is a lot of space for further development in coding structure, organization, and modeling to make this setup provide more complex and generate interesting simulations. The modeling methods and modeling setups require more fine tuning and minor changes which potentially could be implemented in a wide range of applications, which would be overlooked in the results section.

The show case compromises of the overall overlook of the program structure and specific simulations and tests done to validate whether and how those functions work. There will be a results section with regards to each simulation executed or group of simulations executed for the respective cases.

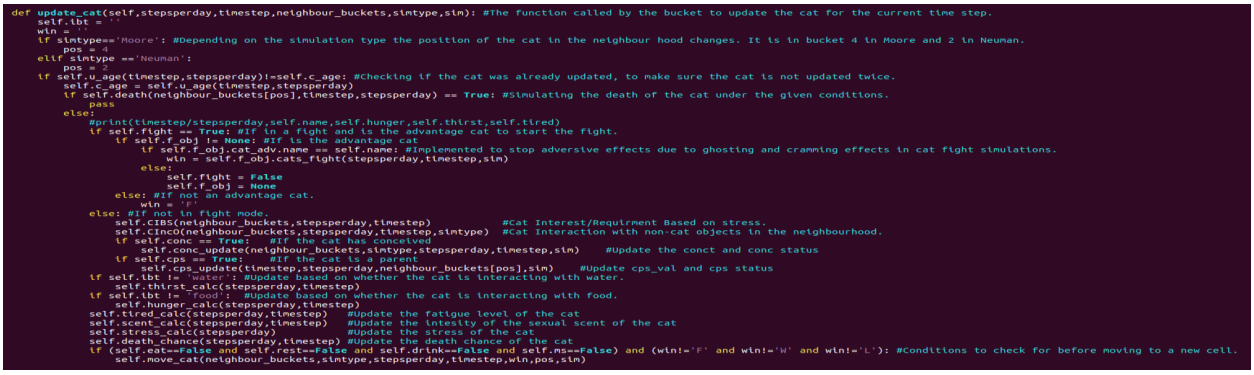
Understanding the structure of the program would be a good start to visualize how the independent structures call each other and maintain the grid to keep the simulation running.



There are three main update functions:

Every object excluding the simulation and Fight classes have an update function in it. This update function will update whatever the attributes based on the current conditions of the code.

Cat update function to update the cat for every timestep: *slightly changed in further debugging*



Landmark Update function to update the landmark for every timestep:

```
def update_landmark(self,stepsperday):  #update the landmark attributes
    self.scent()
    self.resetlandmark(stepsperday)
```

Bucket Update function to update the bucket for every timestep and the update_scents in the bucket class as mentioned in the traceability matrix:

```
def update_object(self,stepsperday,timestep,neighbour_buckets,simtype,sim):
    for c in self.cats:
        c.update_cat(stepsperday,timestep,neighbour_buckets,simtype,sim)
    for l in self.landmarks:
        l.update_landmark(stepsperday)
```

These functions are called by the simulate run_step function which will simulate a timestep of the simulation:

How the above functions are integrated to each other:

The simulation class handles the grid and for every timestep and also will update the scent attributes of the buckets as shown in the run_step function, which calls the update_grid_scents function in the simulation class. This function goes through the entire grid requesting the neighbor generator function to generate the neighborhood of the respective bucket depending on the simulation type to be updated and then call the buckets update_scent function which will update the scent based on the provided attributes. Similarly, the simulate_step will call the buckets update_objects function which will update the objects in the respective bucket and as a bucket definiens the unit cell of the grid when iterated through under certain conditions to not cause exceptions and errors. We would run through a timestep of the simulation.

The number of steps to iterate through or the simulation running on real time is dictated with a while loop in the play_sim function in the user interface. If the command is play_for it will run a thread with the play_sim function using a counter, on the other hand in real time simulations there is no counter. The plot saves and log save are also handled by the user interface to a certain extent.

This would be a summarized idea of the concept that executes the respective components of the code. It is rather complicated as there are multiple conditions and attributes within functions that are attached/affiliated to these functions.

Methodology involved to run the simulations of the results section:

The simulation files are provided by default even with the functionable plotter file to generate the respective plots within the respective output data files. If these simulations were to be simulated again the results would be different as there is a certain amount of randomness involved in the program structure. The plots with cats and landmarks that are referred to are automatically generated if the correct commands are given as shown in the respective simulation. The line plots to show the different models are generated using a separate plotter file using matplotlib from the log directory.

The input terrain files for the terrain testing are also provided in the user input directory as terrain1.csv, terrain2.csv, terrain3.csv and terrain4.csv. Feel free to use them for other simulations as well. The commands for the simulations that are given as system arguments are provided extensively in the user guide, while the commands executed for the respective simulation are also presented.

Almost all the simulations done below can be easily called using the given commands can be executed using sys.args as shown below in this image:

```
/GOC_CreatedByNaradha/object_py$ python3 goc_gui.py 'load(CtoC_extensive,5)' 'set(speed,100)' 'play_for(73000,ALL,1,1,1,1,5,0)'
```

A short description on the concept that allows the user to create the simulations quickly:

The custom simulations are not made using the create_sim command but by taking a saved random flat terrain simulation and editing that csv for the respective cases which is easier and a much faster method to make custom simulations. However, if the csv load file is not properly setup the simulation will return errors.

CSV Save file:

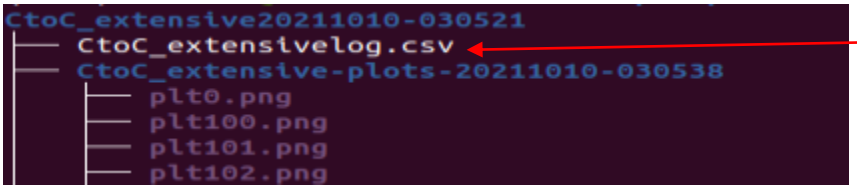
```
Sim,CtoC_extensive,Moore,100,12,12,0,,ALL
Cat,c0,Male,1967,3,3,0.0,0.0,0.0,0.0,0.0,0.0,0.002,False,False,False,False,False,0,,food,False,None,0,0,False,0.0,0
Cat,c1,Female,2000,1,5,0.0,0.0,0.0,0.0,0.0,0.0,0.002,False,False,False,False,False,0,,food,False,None,0,0,False,0.0,0
Landmark,water,water,1000,3,['all'],False,1.0,0.0,3,2
Landmark,food,food,500,3,['all'],False,1.0,0.0,8,8
Landmark,bed,bed,0,2,['all'],False,101.0,0.0,5,2
Landmark,box,box,1.2,2,['all'],False,101.0,0.0,3,5
```

Cats

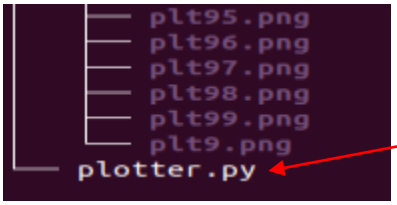
Landmarks

It is much easier to change the attributes from the save file and loading it up again from the simulator instead of making a simulation from scratch.

To generate the matplotlib plots in some simulations a plotter file was used, as they were heavily altered in the process of generating plots, they might need minor changes to regenerate the same plots again. The data for the plots were taken from the log csv file as shown below.



Log file for the CtoC_extensive test



Plotter used to generate the CtoC_extensive plot

Results of extensive simulations:

It is easier to go through each section or concept in the code and results by referring to a set of simulations. These simulations were also the test simulations conducted to test the respective components.

1. Simulation for Cat Landmark Interactions and Cat Death Simulations.
2. Cat Stress how it randomly changes the requirement.
3. Cat – Cat Interactions (Mating, Parenting and Fighting) (Focus on 3 different simulations)
4. Multiple Cats with terrain.

By going through these components of code we can understand the method and the results we would obtain for the above simulations. We would go through every simulation independently and identify the methods involved and results.

Simulation for Cat Landmark Interactions and Cat Death Simulations.

By executing the commands to run a landmark-cat interaction-based simulation for different spans of time with an isolated cat in different scenarios and cases that can be referred to understand the concepts behind the object behavior.

The decision making of the cat class is based on the CIBS function which stands for Cat Interest Based on Stress. This function sets a requirement for the moment, which the cat will pursue as long as his stress level does not induce a change.

The stress level in this case is modeled by hunger, thirst and fatigue. This will be gone through thoroughly in the next section. However, the interactions with the landmarks are modeled and made possible due to the CIBS, cats hunger level, cats thirst level and fatigue level.

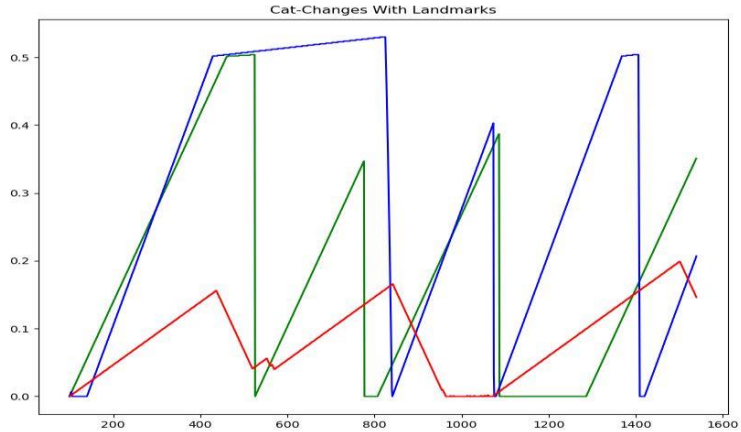
We will visualize how these attributes of the cat change in a simulation with landmarks to understand the method and identify the results that can be drawn from it.

1. Running a simulation of an isolated cat with all landmark types (for a timespan of a day in simulation time = 1440 timesteps):

   Commands executed:

```
> load(CLI,5)
> set(speed,144)
> play_for(1440,ALL,1,1,1,1,5,0)
> Compleleted Play For Excution.
```

   Changes to Cat Attributes with all three landmarks:



Legend:

Thirst

Hunger

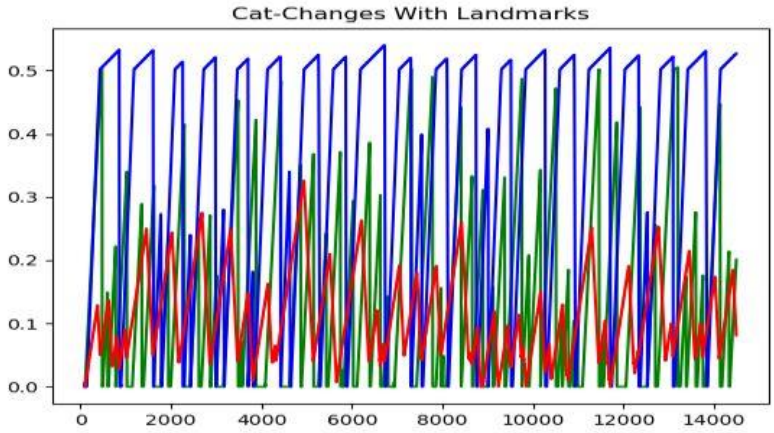
Fatigue

2. Running a simulation of an isolated cat with all landmark types (for a timespan of a week in simulation time = 14400 timesteps)

Commands executed:

```
> load(CLI,5)
> set(speed,5)
> play_for(14400,ALL,1,1,1,1,5,0)
> Compleleted Play For Excution.
```

Changes to Cat Attributes with all three landmarks:



Legend:

Thirst

Hunger

Fatigue

Methods affiliated to above simulation:

The Load command will load a csv based saved simulation in this case the simulation is the CLI simulation. This function will then create a simulation to the parameters of the previously saved simulation.

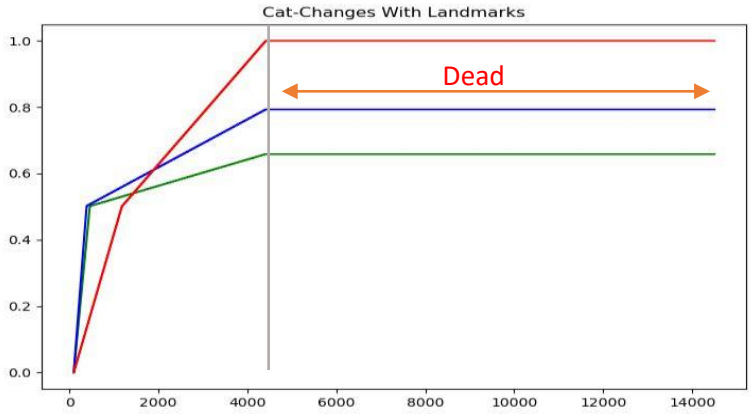
On the start of the simulation in the day simulations graph we can see the cat's hunger, thirst and fatigue models working as expected. Initially until 0.5 the change is of a higher magnitude presenting the region of change where the hunger, thirst and fatigue is less apparent, while after 0.5 the change is slow and if the cause of death was natural, it would most likely be from fatigue as the cat will search for food and water under the stress and die from fatigue. This is how the modeling for the three components are implemented in the respective components.

The hunger model is similar to that of thirst and fatigue, but they have different tolerances. The fatigue model has the lowest tolerance, which is why the cat dies from fatigue and is explained by the three death-based simulation plots shown below

(The tolerance is how long it takes for the model to become 1)

In the cat death simulations, the cat is dead after one of the values reaches one, and to establish that the cat was killed by the code we can see that there is no update to the attributes after one attribute reaches 1. These simulations present that the models of the respective components do provide expected results with randomness in the simulation. The simulations where the cats death was tested were CD_Sim simulation loaded similarly for 14400 timesteps (CD stands for Cat Death Simulations)

1. Cat Death simulation without landmarks:



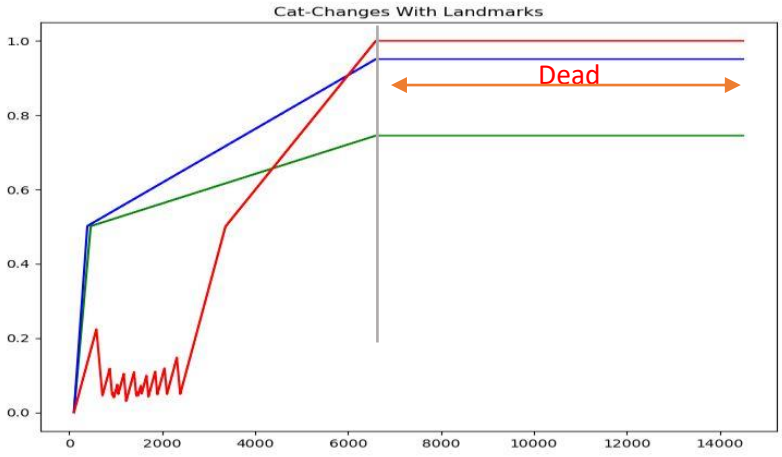
Legend:

Thirst

Hunger

Fatigue

2. Cat Death simulation with Bed:



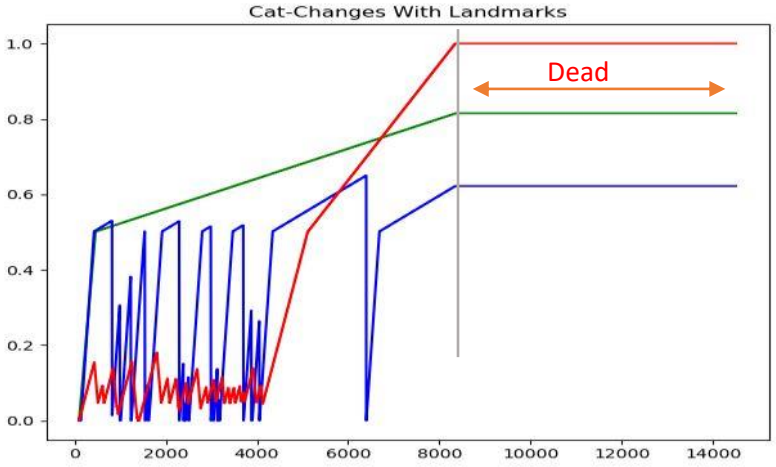
Legend:

Thirst

Hunger

Fatigue

3. Cat Death simulation with water and Bed:



Legend:

Thirst

Hunger

Fatigue

The data is extracted and plotted after the simulation log file is created by the play_for command execution which will generate saved plot files as well depending on the speed set to the simulation.

Cat Stress how it randomly changes the requirement.

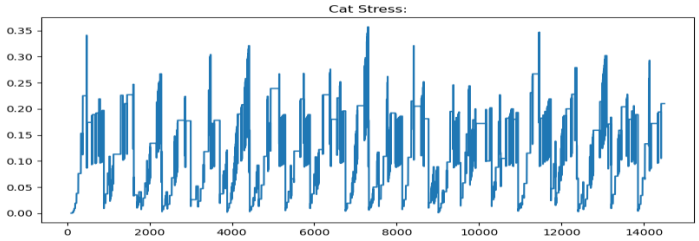
Commands Executed:

```
> load(CtoC_Stress,5)
> play_for(1440,ALL,1,1,1,1,5,0)
> Compleleted Play For Excution.
```

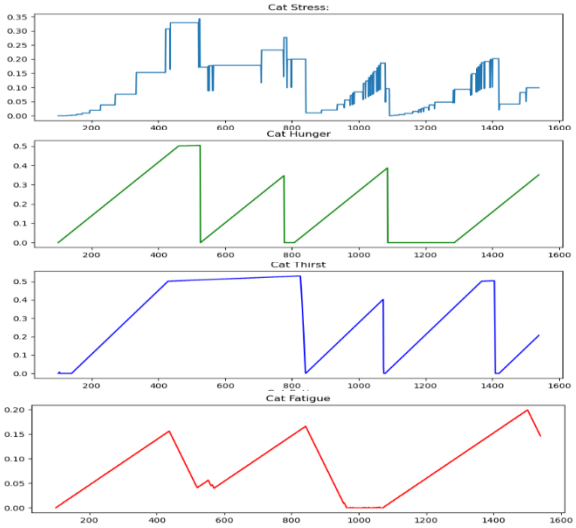
In this scenario the first test would be to visualize the stress of an isolated cat in a simulation. Using the CLI simulation above we can visualize the stress of an isolated cat provided with all the requirements the sub plots provided below describe present the stress and the cat attributes that dictate stress. This will allow us to understand how the stress changes in isolated environments.

The main reason of implementing the stress-based model was to bring in randomness in the cat's movement and to bring in another variable to allow simulating cat fights.

Cats Stress from the simulation CLI simulated for a week:

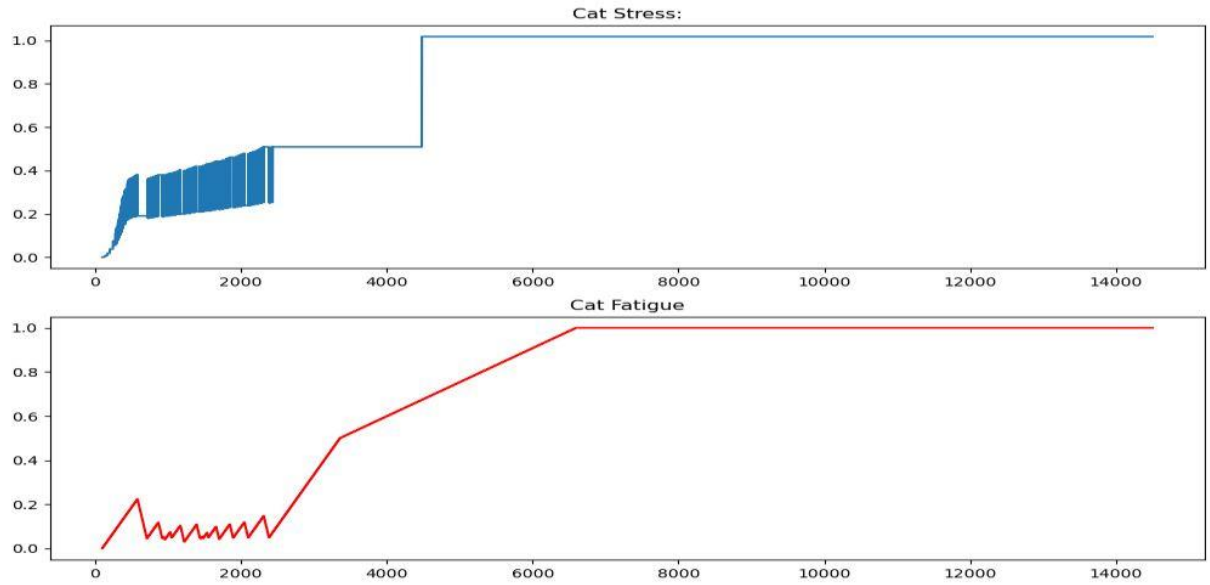


Cats Stress from the simulation CLI simulated for a day with attributes affecting it:



Out of the above plots if we were to consider the stress model, it is a model that takes the squared mean of the three values. It is called in multiple sections to instigate a random panic-based behavior of the cat when the given values increase. The sudden drops show that the cat's requirement of the moment changed, it can be seen in the day simulation the first requirement was rest hence, the cat takes some rest in the bed but later as the hunger was high the cats requirement changed to food, and it went on to eat food.

- In the Cat Death simulation only with a bed, where cause of death was fatigue:



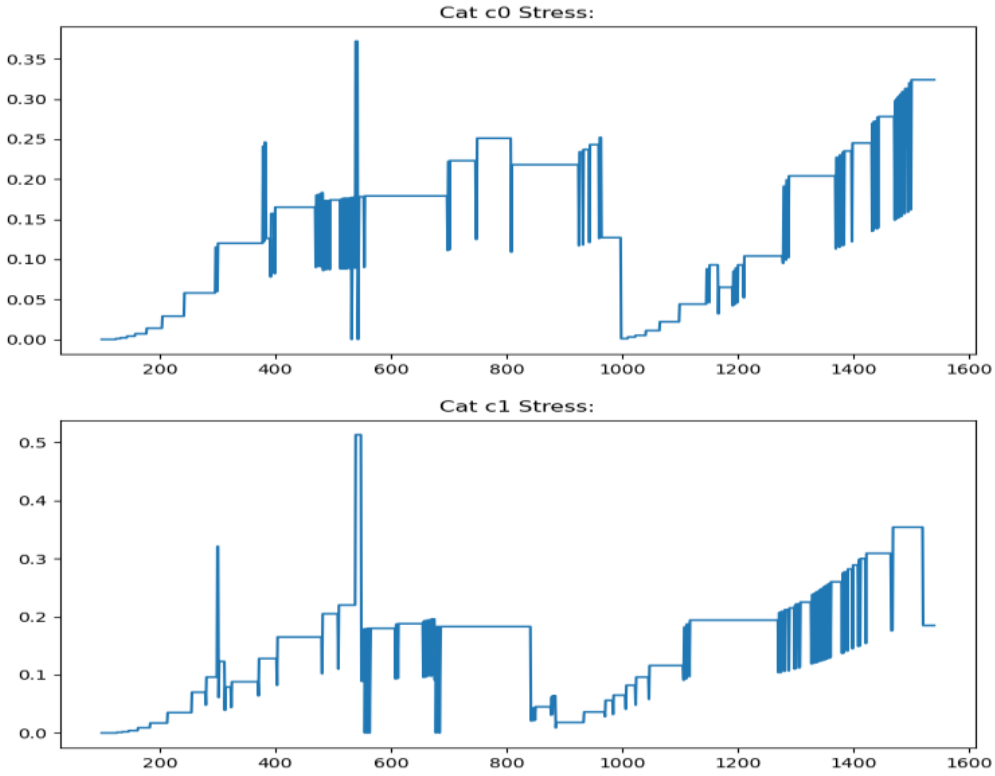
Modeling the panic and stress of the cat:

Even though the cause of death was fatiguing the stress value does not reside even though the cat takes multiple rests initially. As soon as the thirst and hunger pass critical levels as seen in the death simulations. The change in the stress level is drastic as the cat will continuously try to resolve either hunger or thirst, but as neither of those sources are available the cat cannot find them and hence will not take rest ending up dead due to fatigue.

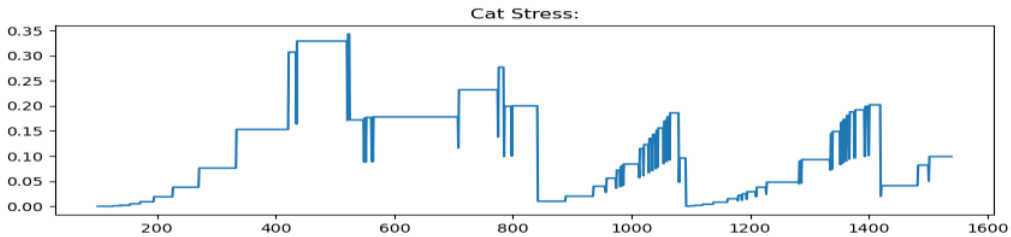
This behavior of the cat object is made by the CIBS cat interest based on stress function which is programmed to change the requirement and resolve some need when under stress. This function checks food and water as the main needs, while rest is the third priority. Hence, the function would flag either food or water alternatingly as the requirement of the moment. But the cat ends up passing away due to fatigue. This was implemented as it seems true to nature where when some entity searches for water and food, they most probably pass away due to the fatigue in the searching process by passing out.

This does not provide that there is no chance for the cat to die out of pure hunger or thirst. These cases are also possible but are nearly rare to non-occurring occurrences.

Cat Stress in simulation with multiple cats.



In this simulation we are having two cats in the same CLI environment in a new simulation called CtoC_Stress simulation file. It can be seen that there are significant changes in the cat's stress levels in this case with two cats. This will be more apparent when compared with the isolated curve.



The structure of the stress curve is also different the isolated cats stress peaks below 0.35 and in 4 peaks. The simulation with two cats is different with c1 having 3 peaks the highest being 0.5, c0 having 3 peaks as well but with a lower stress peak at around 0.35. It also should be noted that a fight was initiated at around the 500th timestep which explains the sudden change in stress of the two cats, as stress is amplified in the cases of a fighting, mating and parent cats.

The results that can be taken from this section is how the CIBS function though designed from scratch to bring about a certain level of randomness to the simulation is doing a quite good job. Still, it can be noted that the curve does not define a function of sorts meaning it is highly fluctuating. Even though this is the expectation of the developer this requires more testing to implement in a more stable form factor.

Cat – Cat Interactions (Mating, Parenting and Fighting) (Focus on 3 different simulations)

- Simulation of Cat fights:

If a simulation was setup with minimum resources and more cats, if their hunger, thirst, and fatigue were set to zero at the start. It would create a simulation where the cats would reach the same requirement in the same time intervals and by moving towards the same items, they would start to fight with each other depending on their stress levels and domination scores.

Commands executed to make random simulation:

```
> random(Moore, 1440, 12, 12, , 8, 3)
> play_for(100, ALL, 1, 1, 1, 1, 5, 0)
> Compleleted Play For Excution.
> save()
```

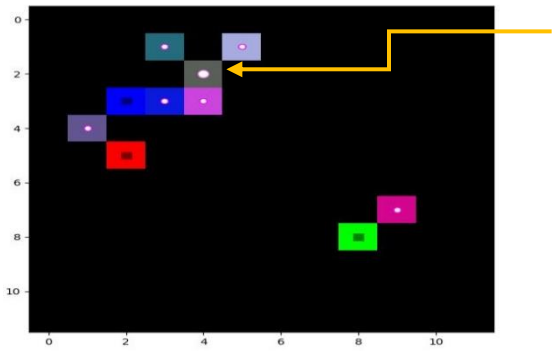
The initial test is generated by creating a random flat terrain simulation, which is then converted to a loadable edited simulation (copy the cats in the save csv to the edited csv).

In the loaded simulation the various fight-based instances can be seen below:

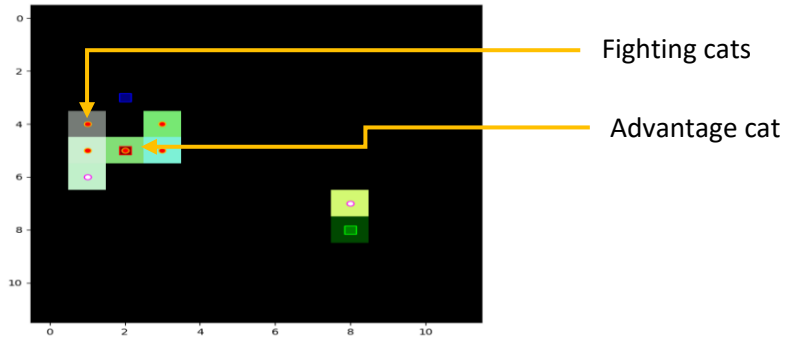
```
(base) wlcnara@wlcnara-VirtualBox:~/FOP/Assesment/Final_Model/GOC_CreatedByNaradha/object_py$ python3 goc_gui.py 'load(CtoC_Fights,5)' 'play_for(1440,ALL,1,1,1,1,5,0)'
0.3597222222222212 1 0.506872743290 79
c1   388   1967.2694444444444  died due to natural causes.
```

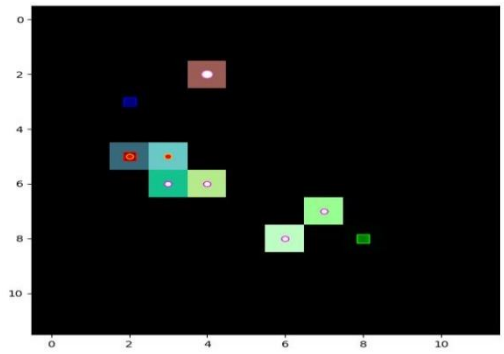
A cat in the simulation died out of fatigue. A similar scenario can be seen in one of the next simulations. This is as the cat did not challenge for the bed but just stayed and died from fatigue.

First simulation timestep:

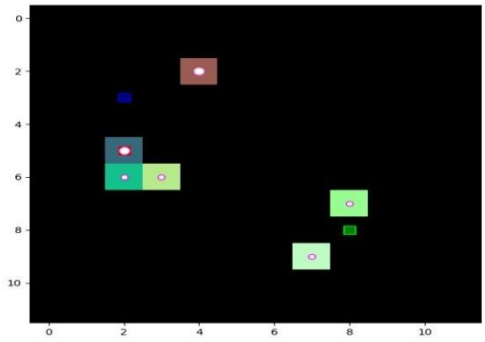


A Fight timestep: 893





Another Fight timestep
in a previous simulation



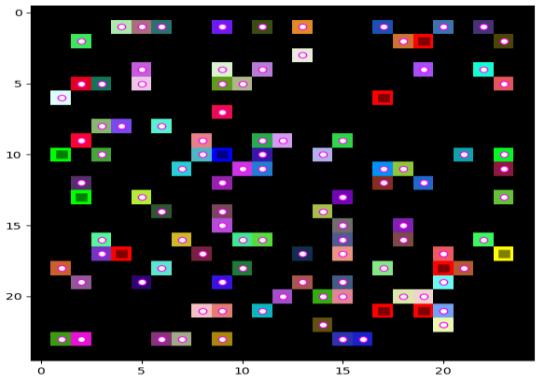
Another cramming situation
around the bed

The above plots show different instances where the cats went into two stages in a fight one being a fight and another being cramming. In the fight at timestep 893 the challenge was from a cat to the cat holding the advantage cell with the bed at 2,5 approximately. The cat was initially challenged by a cat using the challenge function then a fight object was created which handles the fight and as the fight is for this one cell it is only one fight object handling all the cats fighting for this position.

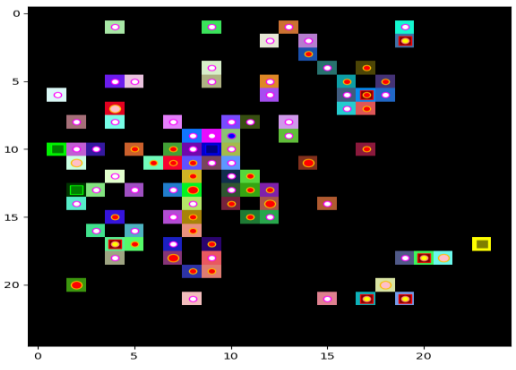
The cramming effect occurs when the challenge was ignored by the advantage cat or even when there is the small interval between which a bucket is set as occupied and unoccupied which is identified as the ghosting effect. Initially this was identified as an error due to the ghosting effect as the system cannot change a position of a cat which is not at that position. The error came up with the initial iteration of the fight object, hence as the exact cause of the error could not be pinpointed exactly and as it brings an interesting side to the simulation the errors were handled so that the ghosting was allowed. The main cause of such cases were from the fight object.

To observe a network of fights we need to run a simulation with a lot more cats, the easiest way being to make a simulation with around 100 cats and 10 landmarks, let's use the random command and run such a simulation. The random command to create such a simulation requires a considerably larger flat map, hence in this we would use a 25*25 grid for the simulation. In a simulation of this extent usually we could also see mating occur but to identify all three stages of mating, conceiving, and parenting requires a more extensive simulation.

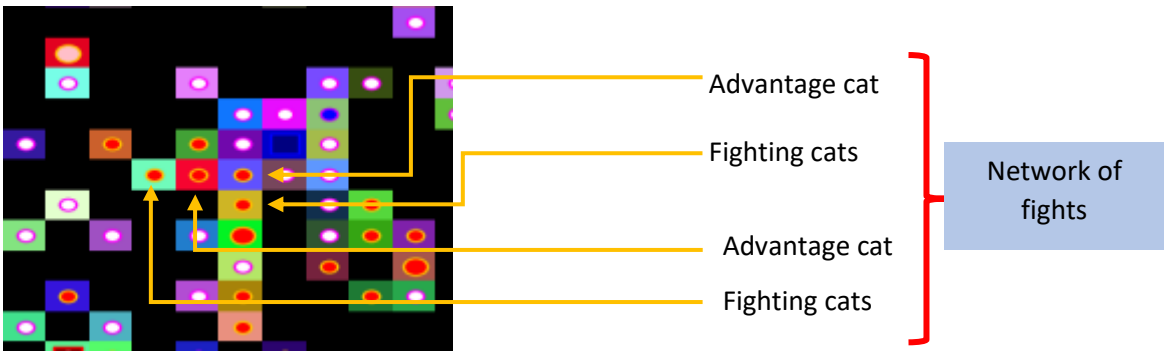
First simulation timestep:



A network of fights in the 13th Timestep:



The network of fights can be seen in a section and such a network would contain multiple fights where the cats are queuing to get their positions from another.



This section of the fight generated, was because of a network of fights. There are multiple fights that link around the same group of cats. This is as the fighting cat is in a position of advantage for another cat. Thus, another fight starts involving them hence creating a network of fights. This is as expected by the fight class. However, there are angry crammed cats which is also because of the ghosting effect explained above. However, this case is caused when they move in the middle of the fight.

Some cats died during the simulation:

```
c61  died due to natural causes.
c94  died due to natural causes.
c47  died due to natural causes.
c40  died due to natural causes.
c44  died due to natural causes.
c55  died due to natural causes.
c86  died due to natural causes.
c2  died due to natural causes.
c43  died due to natural causes.
c83  died due to natural causes.
c64  died due to natural causes.
c26  died due to natural causes.
c1  died due to natural causes.
c50  died due to natural causes.
c41  died due to natural causes.
c48  died due to natural causes.
c59  died due to natural causes.
c54  died due to natural causes.
c80  died due to natural causes.
c69  died due to natural causes.
c95  died due to natural causes.
c68  died due to natural causes.
c4  died due to natural causes.
c49  died due to natural causes.
c78  died due to natural causes.
```

This random simulation is not available in the save files, but the user can simply use the random and play_for commands to regenerate a similar simulation.

Simulation of Cats mating, conceiving and parent status:

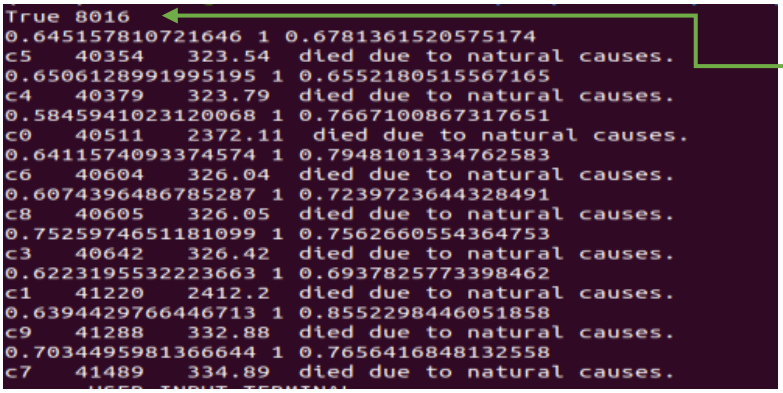
Simulations to show and test mating, conceiving and parent state of the cats in the simulation, require another tailor-made simulation which will run for a total of 2 years in simulation time. From the mating instance to the instance, where the kittens will leave the box of litter and the parent state of the female cat resets.

In this simulation, we can go through the sexual life of a cat, and the parent model implemented in the model. The commands executed for this tailor-made simulation:

```
> load(CtoC_extensive,5)
> set(speed,100)
> play_for(73000,ALL,1,1,1,1,5,0)
> Compleleted Play For Excution.
```

It is worth knowing that the CtoC_extensive is slightly different to that of the other simulations, the definition of the simulation is low set at 100 steps per day rather than the usual 1440 which is as this simulation must be run for a span of 2 years in simulation time. If the definition was set to 1440 it would take the developers laptop over a day to provide the simulation results. Initially the scent change of both the female and male cat must be studied to understand the model implemented for mating of cats.

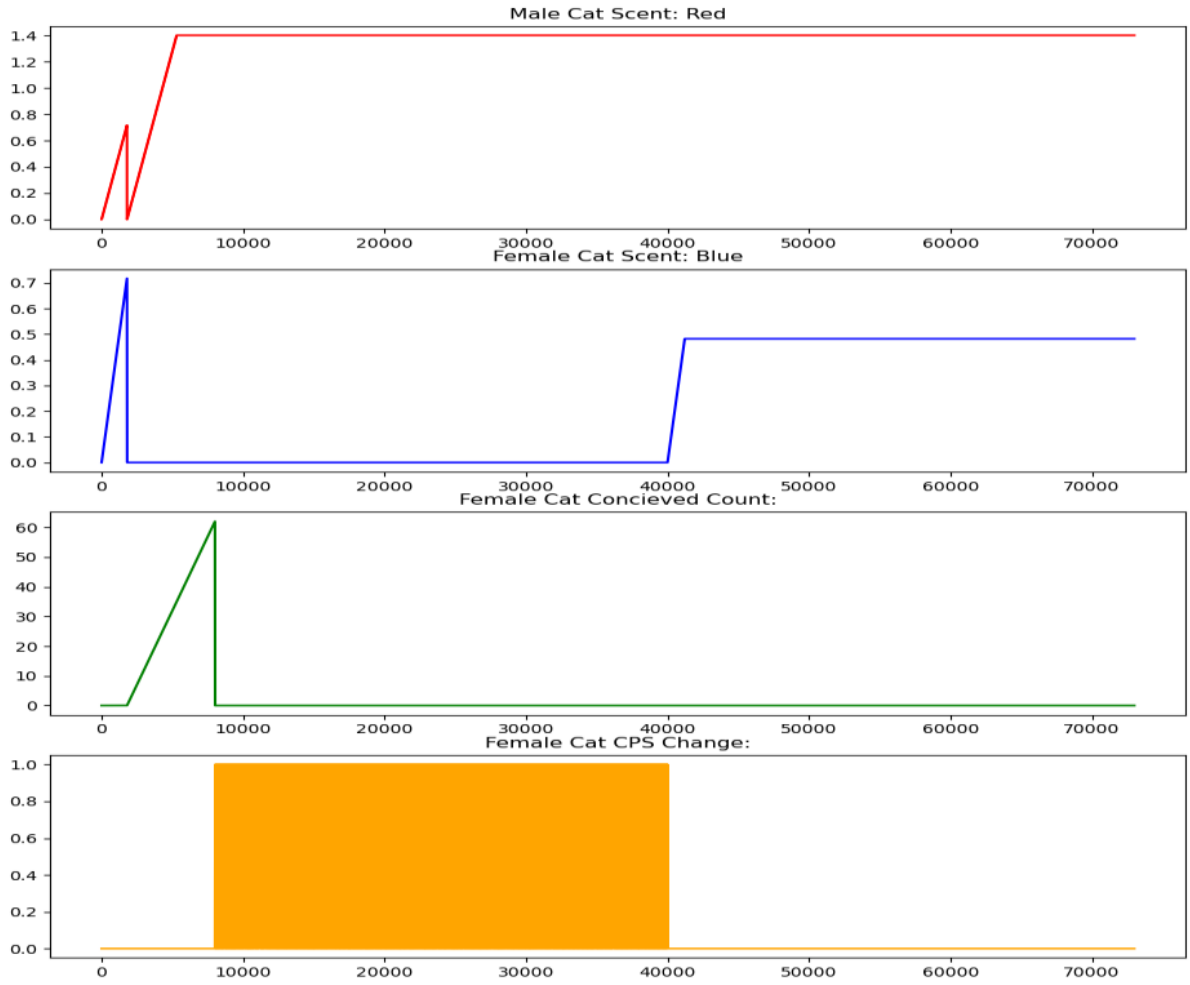
This was the terminal output that was taken, when testing the above cases that represent the final scenario of the above simulation:



Initially when looking at the output the deemed result would be that the code failed. However, what had happened was an overcrowding effect. This can be explained by going through the visual data, but before going through the plot data let us see how the cats multiplied in this scenario.

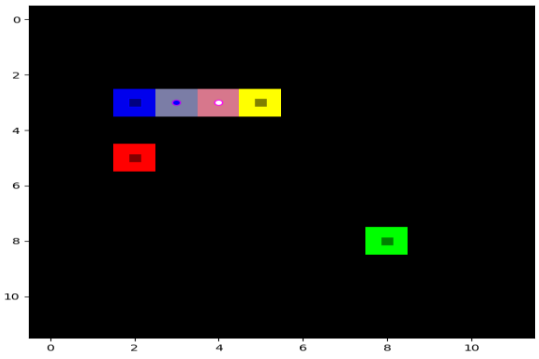
A certain level of debugging and changes were made when simulating for the 2 years in simulation time certain minor bugs were found. Hence, the fight section would alter how it handles related situations and such altered test results are produced for the first fight simulation, while other simulations have not been majorly affected.

Plotting Male cat Scent, Female Cat Scent, Female Cat Conceived day count and CPS Change:

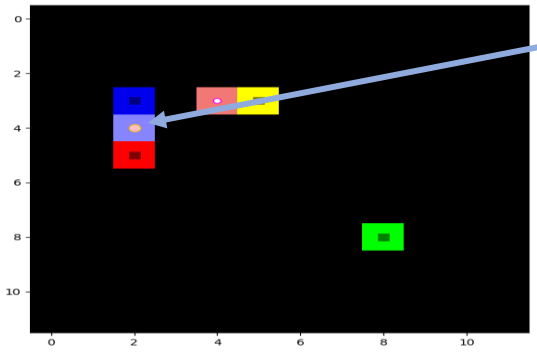


This plot shows how the cats scents manipulate mating, the scents maximum is 1.4 and the point at which the cat possibly searches for mates is from 0.7 onwards. This explains why the scents reset immediately after 0.7 as the two cats mated.

Cause they were successful in conceiving the female cat, the female cats conceive counter conct adds up and when it reaches 62 the female cat gives birth to the kittens. However as this is a very complex scenario to simulate, to simplify the concept, the box is used as an item to raise the kittens to which the female cat comes back to and nurses the kittens depicting some form of parent child relationship. This is where CPS comes into effect, CPS stands for Cat Parent Status and the CPS_val changes in a manner such that it makes the female parent cat check on the kittens a minimum of 4 times a day. If the female cat is in the box, the hunger and thirst of the kittens would reset as they are most probably in rest mode in the box.

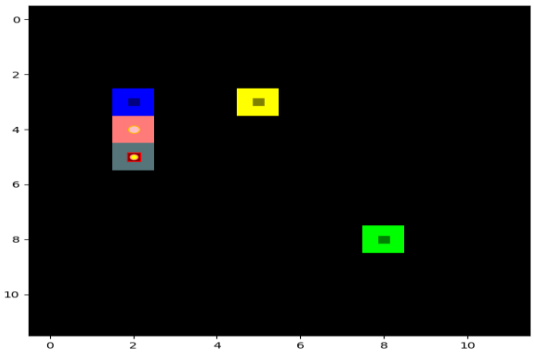


Before Mating

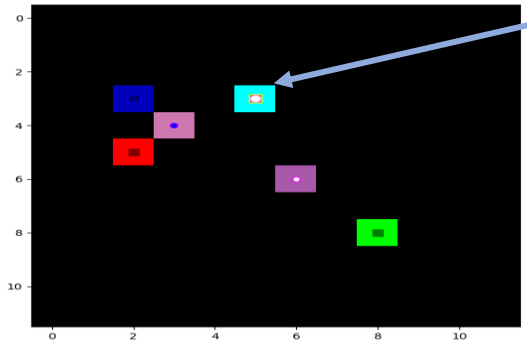


Conceived Female Cat

After Mating

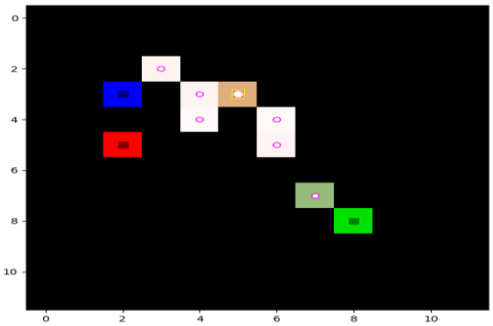


Before Giving birth



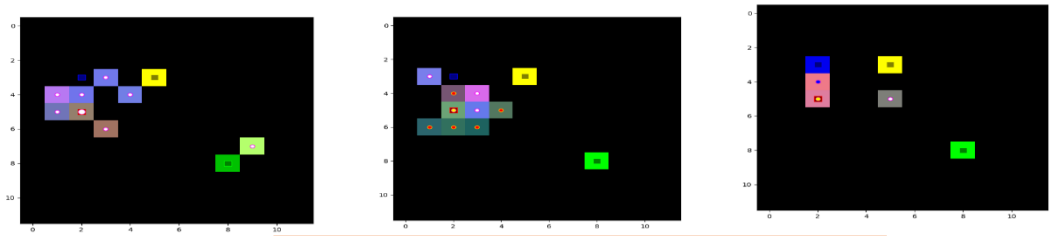
Kitten litter in Box : Box color change indicates the litter.

After Giving birth



After a year the kittens leave the litter box as they can jump out of it.

The incident at the end of the simulation was because of over congestion in the map:



Dreadlock Fight and no sufficient Food/Water

The idea of the bed is a cell in which the cat can rest and clearly in this simulation one bed is not sufficient as most of the cats are very young, they could die off fatigue quickly. Still the adult cats also died as a result, it was due to the quickly draining water and food resources by the end of the fight 4 – 5 cats had passed away.

Multiple Cats with terrain:

Case 1:

In this simulation, the scent propagation of an item surrounded by a terrain acting as a barrier will first be shown, through two simulations one involving no barrier and one landmark, the other with a barrier and a landmark. The barrier is due to the sudden change in terrain height for scent propagation.

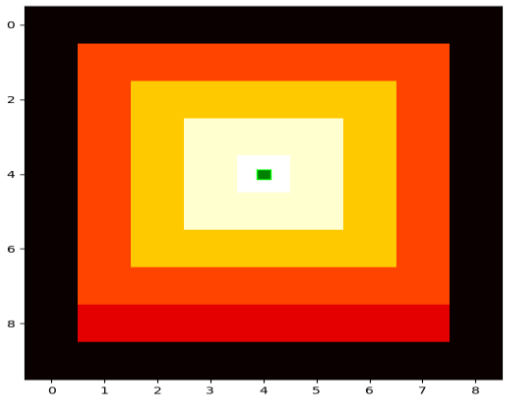
Commands executed:

```
> random(Moore, 1440, 0, 0, terrain0.csv, 0, 1)
> save()
```

This save file is then edited to the two conditions by editing the terrain files for two simulations. Terrain0.csv is a 10*10 grid of zeroes. The actual terrain grids are edited in the respective save folders.

No Boundary:

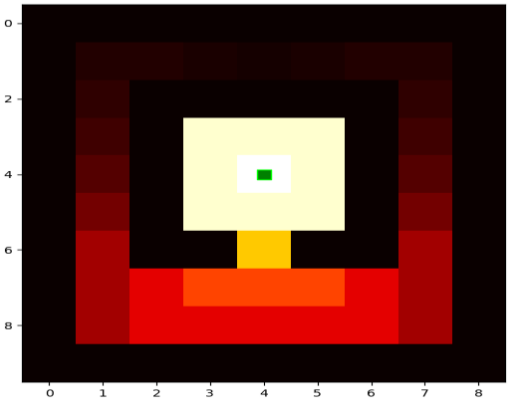
```
> load(Terrain_Testing, 5)
> play_for(100, Food-Scent, 1, 1, 1, 1, 5, 0)
> Compleleted Play For Excution.
```



Flat Terrain.

With Boundary:

```
> load(Terrain_Testing2, 5)
> play_for(100, Food-Scent, 1, 1, 1, 1, 5, 0)
> Compleleted Play For Excution.
```

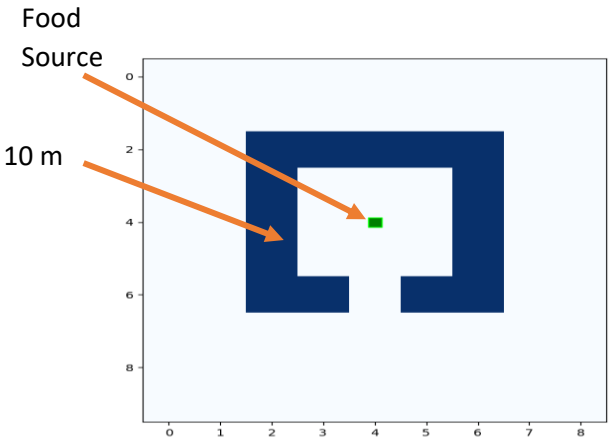


Terrain acting as Boundary:

```
> load(Terrain_Testing2, 5)
> play_for(100, Terrain, 1, 1, 1, 1, 5, 0)
> Compleleted Play For Excution.
```

The Terrain to restrict scent propagation is a wall as can be seen in the right.
This is an additional feature to get the cats moving in the proper orientation.

The same effect from landmarks can be taken using directional landmarks.



Case 2: Cats Terrain and everything:

We will be running a simulation in the terrain2.csv which consists of a small residential area road like terrain. Here, with food, water and cats in a random simulation. To visualize everything together and for a change this simulation will be of Von Neauman simulation Type. There will be plots showing the motion of the cats and a small explanation on what was visualized.

As the cat checks for whether it can move to a certain position before moving it never moves into a block with a higher height difference, this too can be established in this simulation.

Commands for execution:

```
> random(Neuman, 1440, 0, 0, terrain2.csv, 10
> play_for(1440, Terrain, 1, 1, 1, 1, 5, 0)
> Compleleted Play For Excution.
> save()
```

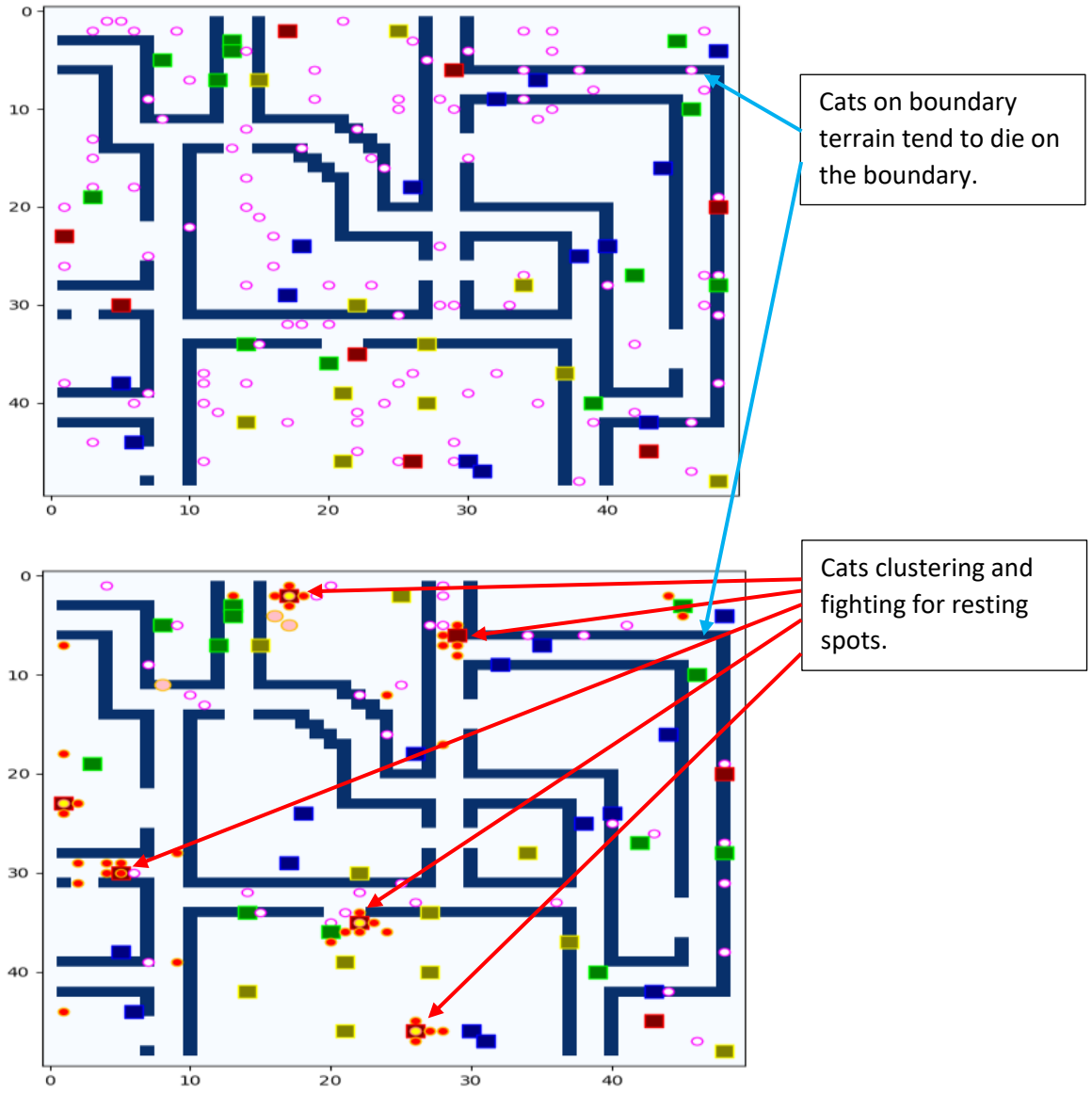
For future execution can load the saved Ran1 simulation to visualize this simulation.

As there are 100 cats, 45 landmarks and the cats on top of the boundary cells start to die as they cannot move easily:

```
0.4454479401067931 1 0.7757523319084015
c65   35   4591.024305555556  died due to natural causes.
1 0.5839176815134351 0.4990518524093563
c94   124   2409.086111111111  died due to natural causes.
0.7028813454961643 0.7744177590781132 1
c81   139   4843.096527777778  died due to natural causes.
0.2819444444444437 1 0.3958333333333322
c22   251   4198.174305555555  died due to natural causes.
0.45264197601605793 1 0.4842404625904654
c45   280   3634.1944444444443  died due to natural causes.
0.5072803198188023 1 0.5206488145256318
c76   308   2251.213888888889  died due to natural causes.
0.5039586362202616 1 0.44791666666666535
c50   311   5277.215972222222  died due to natural causes.
0.5181141704396063 1 0.5200570850143242
c88   363   1833.2520833333333  died due to natural causes.
0.5096342189339477 1 0.5297654125260934
c73   538   1273.3736111111111  died due to natural causes.
0.7342016542468297 0.2173504765806475 1
c78   583   2007.4048611111111  died due to natural causes.
0.5084138404491133 1 0.5218480515324937
c24   631   1541.4381944444444  died due to natural causes.
1 0.6599797307261411 0.9748747083658122
c55   722   2663.501388888889  died due to natural causes.
0.5283285172677603 1 0.5210511897629906
c46   998   4776.6930555555555  died due to natural causes.
0.9510164914734869 1 0.5642066491509531
c93   1045   3514.7256944444443  died due to natural causes.
0.5371164397444677 1 0.5736131757483784
c60   1144   3598.7944444444443  died due to natural causes.
0.5460935756408367 1 0.7980371146620899
c90   1204   3536.836111111111  died due to natural causes.
0.5499023182703439 0.9298536757257178 1
c80   1269   628.88125  died due to natural causes.
0.2569444444444438 1 0.5911535018315227
c57   1283   4198.890972222222  died due to natural causes.
0.5459956965932504 1 0.5768881064759451
c36   1319   2724.915972222222  died due to natural causes.
0.55127318569342 1 0.5935479565090914
c32   1363   2500.9465277777776  died due to natural causes.
0.552155383355118 1 0.5971264839551865
c54   1435   2989.996527777778  died due to natural causes.
```

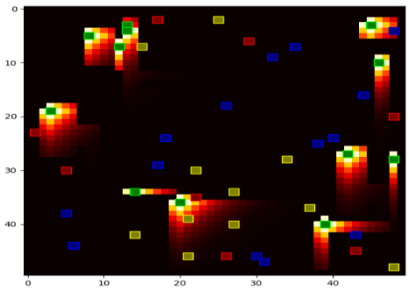
The first and last frame of the simulation give us a summary of the entire simulation. The cats on the boundaries only can move along the boundary due to the height restriction, while the scent propagation is also regulated accordingly.

The first and last frame of the simulation:

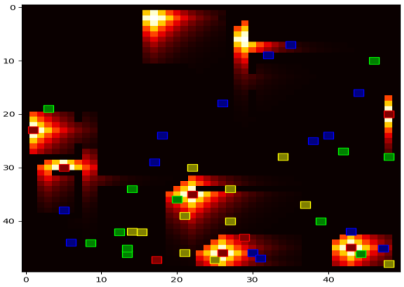


The cats on the boundary terrain either pass away or try to move towards randomly generated items on the map. Again, as this is a completely random simulation the continuity of the cats in the simulation is not likely, but it does provide a good image on the simulation. The clustering around beds is expected as the bed to cat ratio is low and they tend to get tired quickly. When making simulations they can be placed in grids on areas like a sofa or bed etc. so that it models a realistic simulation.

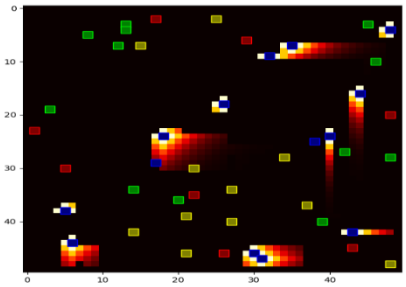
To get a better picture on the scent map, of the above simulation let us see the scent propagation of the landmarks in the map without the cats.



Food



Bed



Water

This provides that the scent propagation also works in the Neuman simulation type, even though most of the map is black, the scent of the source propagates in small values attracting the cats along the path. Based on how the boundaries restrict the scent propagation.

In the simulation, we can also see the cats mating, fighting and more these are all visible on multiple other frames of the simulation.

Overall Results:

The tests establish and present how the respective code components work and provide for the results. The program allows the user to generate different types of simulations, isolate specific data from the logs and visualize user specific views that can be used to further understand the specifics. The results above establish how the various models work and provide for any simulation to run.

Conclusion and Future work:

It can be concluded that most of the initial goals that were set by the developer were met, given that it was a tough and difficult piece of code to make. Still there are certain components that require further debugging and simplification, while also some other ideas that could have been implemented.

At the latter of testing, an error was identified with the age function which was used through out the code. A firm conclusion into how and what causes the error was not achieved, but it was worked around to provide a setup that does not cause the errors of the initial setup. Another instance being the move_cat and update_scent functions in the bucket and cat object, they are long and heavy with repetitive components which can be added into separate functions with more readability and simplicity, but as the conditionals are not easy to play around, they were left as they were to complete the rest of the model. There are such components here and there that need resolution, simplification, and rectification.

For Future Iterations:

1. Providing the user to be able to provide 3d maps, by making floors of maps and bucket objects that can connect to other buckets on the different levels.
2. Applying more accurate models instead of the linear and quadratic models applied.
3. Providing a more descriptive parent Cat and kitten interaction, rather than that provided in the system now.
4. Provide a more dynamic view.
5. providing maybe game modes where the user can play as a cat, developing the simulator as an entertaining game.
6. Simplifying code blocks and rectifying errors that might rise in the future.
7. Creating a multiplayer environment where people can possibly share each other's simulations.