1. Overview

"Animal Kingdom" aims to simulate the wild environment with various changeable input parameters. With the correct parameters, it can be used to model population growth in a controlled environment.

1.1 Features

There are endless possibilities with twenty-five different input parameters that can be altered. Almost all input parameters can be changed ranging from the number of a species to its personal attributes that can alter behavioural changes. Animal kingdom features Pygame graphical interface with the ability to output CSV containing game data and output a graph figure which plots population against time.

2. User Guide

Running "Animal Kingdom" is quite simple. Initially we have a default map stored as a 'txt' file. The format will look something like this.

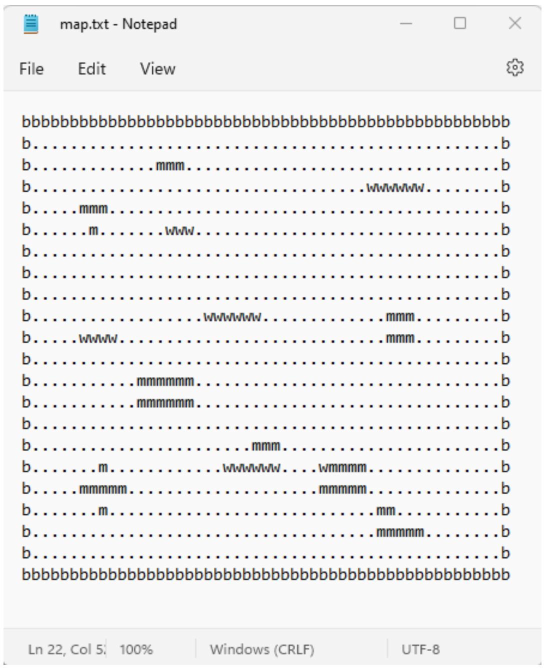
2.1 Legend

'b' – are the boundaries

'm' – are mountains

'w' – are water

'.' – are grass



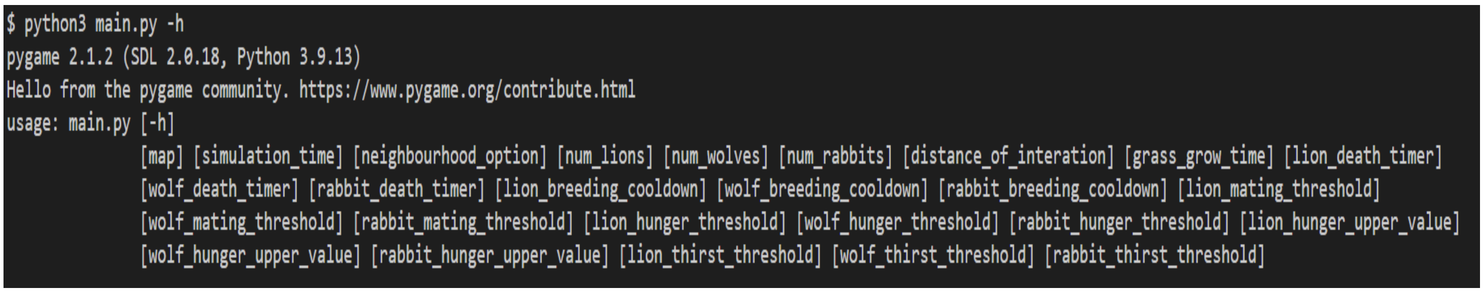
```
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b.................................................b
b.............mmm.................................b
b...................................wwwwww........b
b.....mmm.........................................b
b......m.......www................................b
b.................................................b
b.................................................b
b.................................................b
b..................wwwwww.............mmm.........b
b.....wwww............................mmm.........b
b.................................................b
b...........mmmmmm................................b
b...........mmmmmm................................b
b.................................................b
b.......................mmm.......................b
b.......m............wwwwww....wmmmm..............b
b.....mmmmm....................mmmmm..............b
b.......m............................mm...........b
b....................................mmmmm........b
b.................................................b
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
```

Users may amend the map however they like, from settings up mountain ranges and water terrains to enlarging or reducing the entire map. Or completely producing a new map.

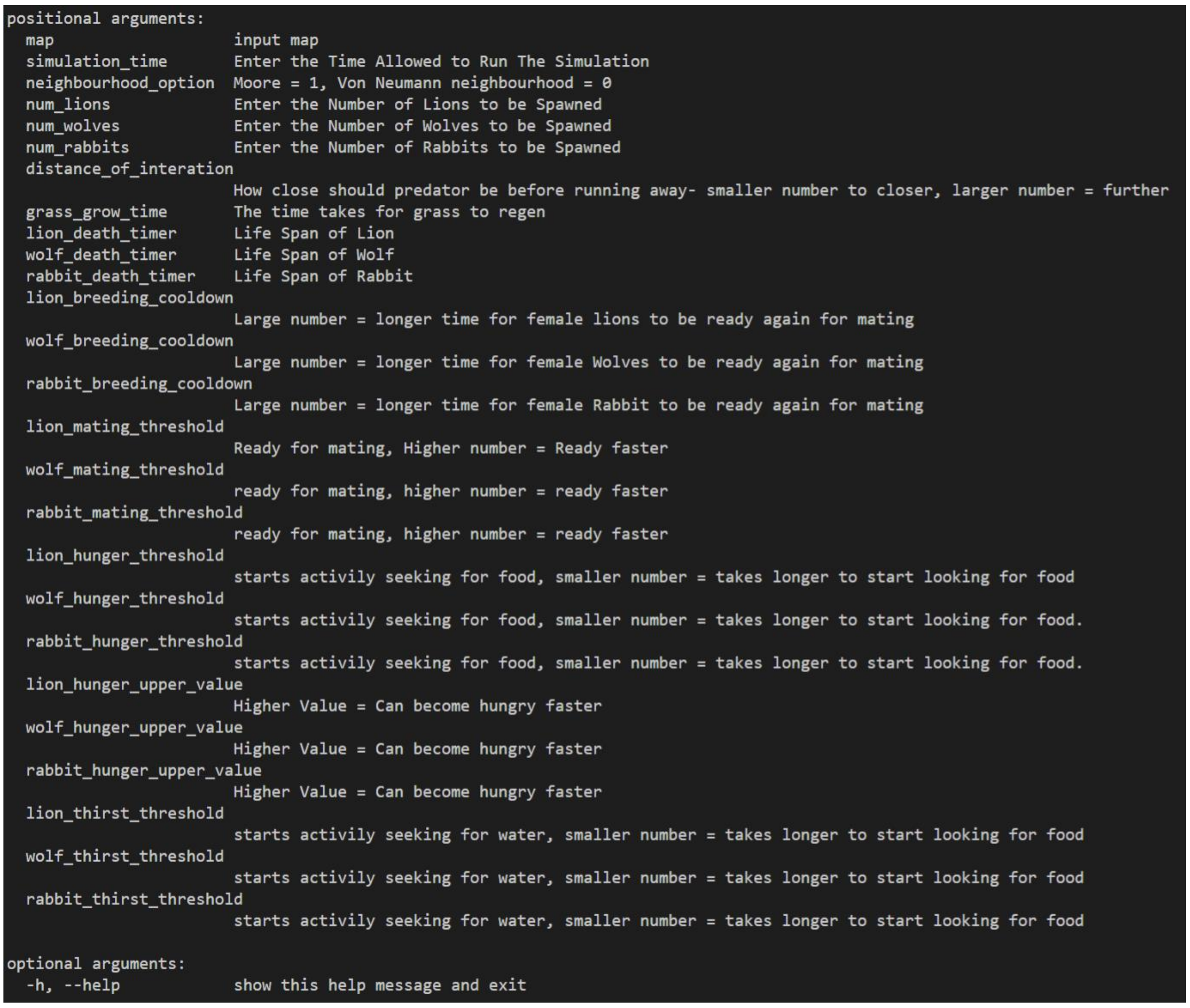
The program will intelligently read in the selected map and determine which block goes where. However you must make sure it is enclosed up with 'b's otherwise animals will run away!

2.2 Usage

Once that is finished, we can now execute the program. You may want to execute this first: python3 main.py -h , this will bring up the below usage information.



```
$ python3 main.py -h
pygame 2.1.2 (SDL 2.0.18, Python 3.9.13)
Hello from the pygame community. https://www.pygame.org/contribute.html
usage: main.py [-h]
               [map] [simulation_time] [neighbourhood_option] [num_lions] [num_wolves] [num_rabbits] [distance_of_interation] [grass_grow_time] [lion_death_timer]
               [wolf_death_timer] [rabbit_death_timer] [lion_breeding_cooldown] [wolf_breeding_cooldown] [rabbit_breeding_cooldown] [lion_mating_threshold]
               [wolf_mating_threshold] [rabbit_mating_threshold] [lion_hunger_threshold] [wolf_hunger_threshold] [rabbit_hunger_threshold] [lion_hunger_upper_value]
               [wolf_hunger_upper_value] [rabbit_hunger_upper_value] [lion_thirst_threshold] [wolf_thirst_threshold] [rabbit_thirst_threshold]
```



```
positional arguments:
  map                   input map
  simulation_time       Enter the Time Allowed to Run The Simulation
  neighbourhood_option  Moore = 1, Von Neumann neighbourhood = 0
  num_lions             Enter the Number of Lions to be Spawned
  num_wolves            Enter the Number of Wolves to be Spawned
  num_rabbits           Enter the Number of Rabbits to be Spawned
  distance_of_interation
                        How close should predator be before running away- smaller number to closer, larger number = further
  grass_grow_time       The time takes for grass to regen
  lion_death_timer      Life Span of Lion
  wolf_death_timer      Life Span of Wolf
  rabbit_death_timer    Life Span of Rabbit
  lion_breeding_cooldown
                        Large number = longer time for female lions to be ready again for mating
  wolf_breeding_cooldown
                        Large number = longer time for female Wolves to be ready again for mating
  rabbit_breeding_cooldown
                        Large number = longer time for female Rabbit to be ready again for mating
  lion_mating_threshold
                        Ready for mating, Higher number = Ready faster
  wolf_mating_threshold
                        ready for mating, higher number = ready faster
  rabbit_mating_threshold
                        ready for mating, higher number = ready faster
  lion_hunger_threshold
                        starts activily seeking for food, smaller number = takes longer to start looking for food
  wolf_hunger_threshold
                        starts activily seeking for food, smaller number = takes longer to start looking for food.
  rabbit_hunger_threshold
                        starts activily seeking for food, smaller number = takes longer to start looking for food.
  lion_hunger_upper_value
                        Higher Value = Can become hungry faster
  wolf_hunger_upper_value
                        Higher Value = Can become hungry faster
  rabbit_hunger_upper_value
                        Higher Value = Can become hungry faster
  lion_thirst_threshold
                        starts activily seeking for water, smaller number = takes longer to start looking for food
  wolf_thirst_threshold
                        starts activily seeking for water, smaller number = takes longer to start looking for food
  rabbit_thirst_threshold
                        starts activily seeking for water, smaller number = takes longer to start looking for food

optional arguments:
  -h, --help            show this help message and exit
```

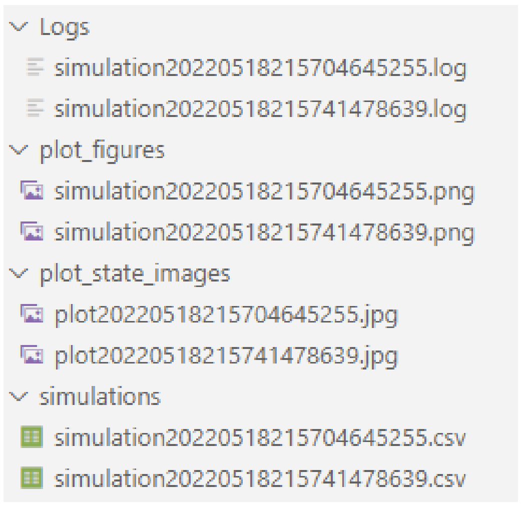
An example to run the simulation would be :

python3 main.py map.txt 150 0 4 4 15

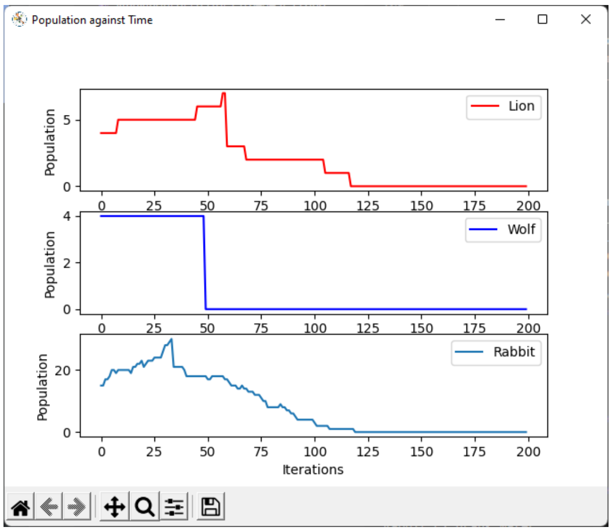
- main.py is the file we are executing,
- map.txt is the input map
- '150' is the "simulation time.
- '0' is the neighbourhood option, here it is using Von Neumann Neighbourhood.
- '4' '4' '15' equates to 4 lions, 4 wolves and 15 rabbits.

You are welcome to add more paraments to alter behavioural changes, otherwise leaving them blank will just incorporate their default values.

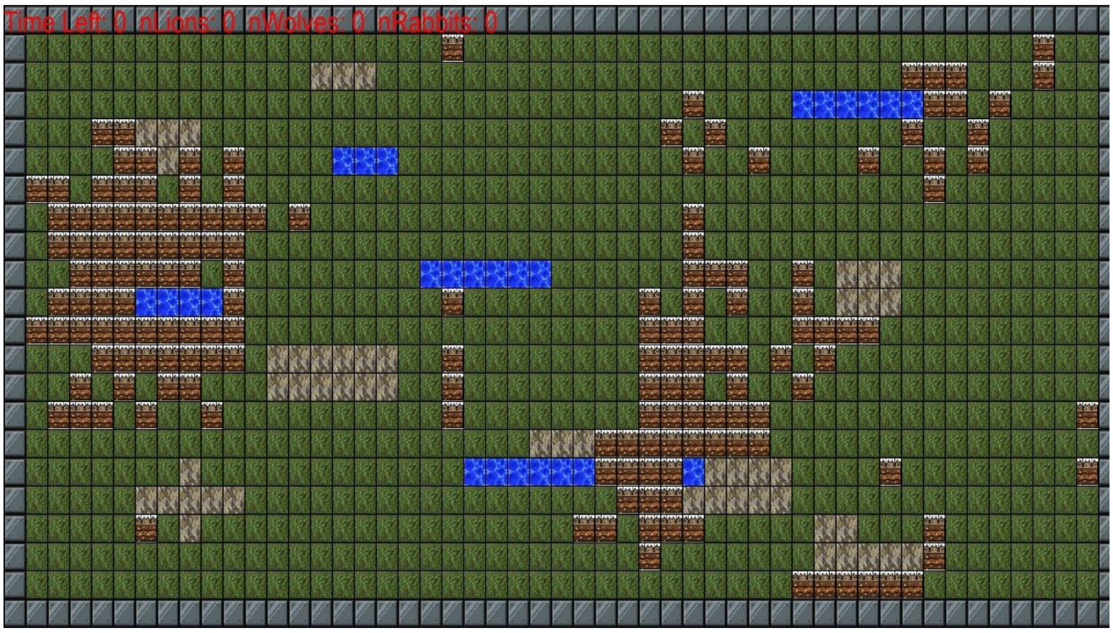
Once a simulation has finished running you will see a graph, a CSV file, a log file, and a game state image:



- Logs
  - simulation20220518215704645255.log
  - simulation20220518215741478639.log
- plot_figures
  - simulation20220518215704645255.png
  - simulation20220518215741478639.png
- plot_state_images
  - plot20220518215704645255.jpg
  - plot20220518215741478639.jpg
- simulations
  - simulation20220518215704645255.csv
  - simulation20220518215741478639.csv



Game State is recorded when the game ends



If you would like to access even more inputs, you can simply open up config.py and change up the values as you like.

2.3 Parameter sweep

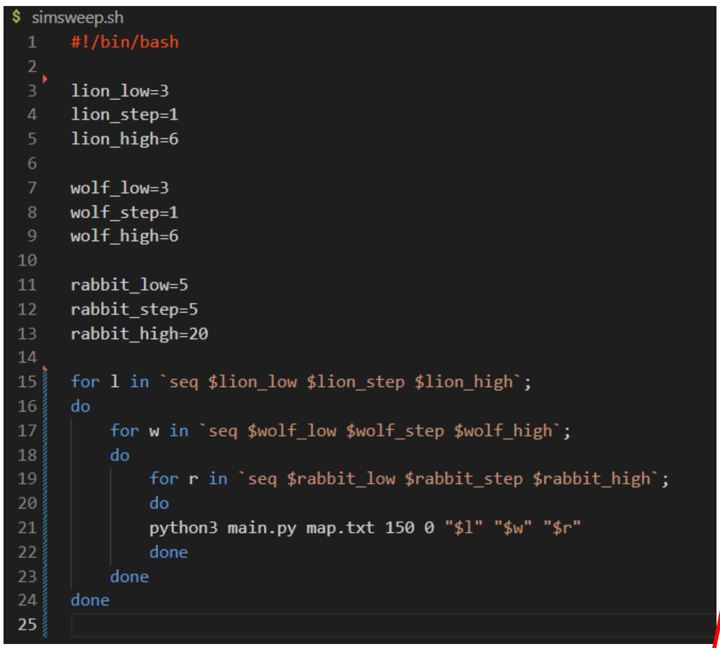
A parameter sweep bash file has been made available however I have only added number of animals at this stage. Considering I have twenty-five parameters, if I were to incorporate all of them into the parameter sweep, this will have a time complexity of O(n^m). Which can take an exceptionally long time to finish as it will need 25^{25} runs to complete the sweep.

However it is remarkably simple to amend the bash file to sweep through different parameters. You can do this by making some parameters a constant and others a variable.

2.4 Changing the Parameter Sweep file:

Initially it may look something like this. In order to sweep through different parameters, we will need to add in placement holders, for example:

Original **Amending it.**



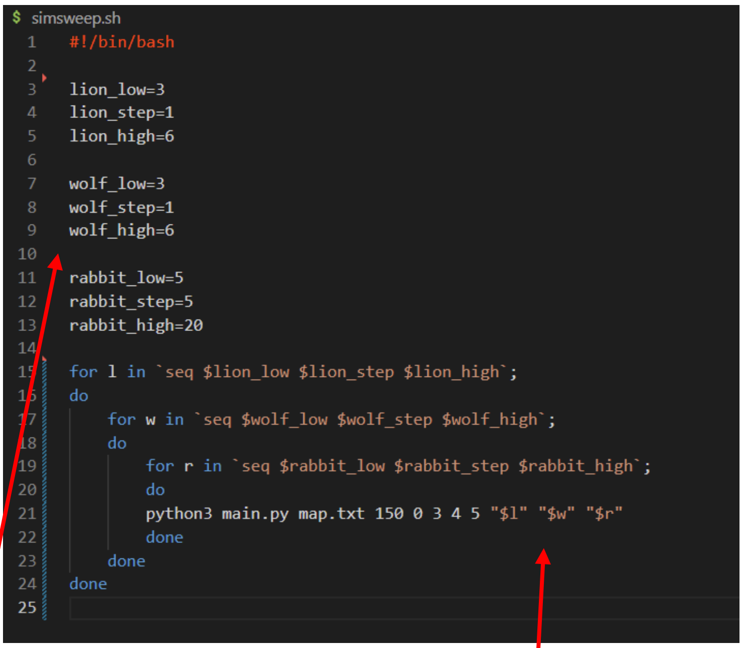
```
simsweep.sh
#!/bin/bash

lion_low=3
lion_step=1
lion_high=6

wolf_low=3
wolf_step=1
wolf_high=6

rabbit_low=5
rabbit_step=5
rabbit_high=20

for l in `seq $lion_low $lion_step $lion_high`;
do
    for w in `seq $wolf_low $wolf_step $wolf_high`;
    do
        for r in `seq $rabbit_low $rabbit_step $rabbit_high`;
        do
        python3 main.py map.txt 150 0 "$l" "$w" "$r"
        done
    done
done
```



```
simsweep.sh
#!/bin/bash

lion_low=3
lion_step=1
lion_high=6

wolf_low=3
wolf_step=1
wolf_high=6

rabbit_low=5
rabbit_step=5
rabbit_high=20

for l in `seq $lion_low $lion_step $lion_high`;
do
    for w in `seq $wolf_low $wolf_step $wolf_high`;
    do
        for r in `seq $rabbit_low $rabbit_step $rabbit_high`;
        do
        python3 main.py map.txt 150 0 3 4 5 "$l" "$w" "$r"
        done
    done
done
```

The variable names and value can also be changed

The $t $l $w $r can be changed for other variables

2.5 Using the parameter sweep

To use “simsweep.sh”, first enter “chmod u+x simsweep.sh” (no quotation marks) into your terminal. Then we can easily execute it by typing “./simsweep.sh” (no quotation marks) into the terminal.

3. Traceability matrix

Feature	**Code**	**Test**

1. Object behaviour position	def __init__(self,game,x,y):
	self.x = x
	self.y = y	

1.1. Lion object has position	def __init__(self,game,x,y):
	self.x = x
	self.y = y
		[Passed] Lion object has assigned position and display position

1.2 Wolf object has position	def __init__(self,game,x,y):
	self.x = x
	self.y = y
		[Passed] wolf object has assigned position and display position

1.3 Rabbit object has position	def __init__(self,game,x,y):
	self.x = x
	self.y = y
		[Passed] wolf object has assigned position and display position

2. Object attributes		
2.1 Lion object has unique attribute	Class Lion, self.name, self.gender, self.hunger, self.death , etc.	[Passed] Able to extract information from individual lion objects Lion will die after set amount of time
2.2 Wolf object has unique attribute	Class Wolf, self.name, self.gender, self.hunger, self.death etc.	[Passed] Able to extract information from individual wolf objects Wolf will die after set amount of time
2.3 Rabbit object unique attribute	Class Rabbit, self.name, self.gender, self.hunger, self.death etc.	[Passed] Able to extract information from individual wolf objects Rabbit will die after set amount of time
3. Object movement		
3.1 Lion Object has movement feature	Class Lion, used in move(self,dx,dy)	[Tested] Moves Lion object around the display screen
3.2 Wolf Object has movement feature	Class Wolf, used in move(self,dx,dy)	[Tested] Moves Wolf object around the display screen
3.3 Rabbit Object has movement feature	Class Wolf, used in move(self,dx,dy)	[Tested] Moves Wolf object around the display screen

4. Movement behaviour		

4.1 Lion object moves different based on scenario	Chase(Lion,prey),
	look_for_nearst_water(Lion)
	Chase(Lion,partner)
		[Passed]
		Lion will chase nearest prey.
		
		Lion will look for nearest water
		and drink when water is one of its neighbours
		
		Lion will chase partners if partner is available

4.2 Wolf object moves different based on scenario	Chase(Wolf, prey)
	look_for_nearst_water(Wolf)
	Chase(Wolf,partner)
	Runaway(Wolf,Predator)
		[Passed]
		Wolf will chase nearest prey.
		
		Wolf will look for nearest water and drink when water is one of its neighbours
		
		Wolf will chase partners if partner is available
		Wolf would run away if a predator were close by.
		
		Wolf would run away from predator is its one of its neighbours.

4.3 Rabbit object moves different based on scenario	look_for_nearst_water(Rabbit)
	Chase(Rabbit,partner)
	Runaway(Rabbit, Predator)
		[Passed]
		Rabbits chase after partner if partner is available
		
		Rabbit will seek for nearest water when thirsty and drink when water is one of its neighbours
		
		Rabbit would run away from predator if they were close by.

4.4 Movements are prohibited on invalid areas such as boundaries	def collide_with_entity(self, dx=0, dy=0):	[Passed] Objects cannot move into restricted areas

5. Object interaction		
5.1 Lion interaction	Inside Event loop for Lion Lion will call get_neighbors(option) which will alter interaction	[Passed] Lion will eat prey if prey is one of its neighbours Lion will drink water if water is one of its neighbours
5.2 Wolf interaction	Inside Event loop for Wolf Wolf will call get_neighbors(option) which will alter interaction	[Passed] wolf will eat prey if prey is one of its neighbours Wolf will drink water if water is one of its neighbours
5.3 Rabbit interaction	Inside Event loop for Rabbit Rabbit will call get_neighbors(option) which will alter interaction	[Passed] Rabbit will eat grass when hungry- usually the block under itself. Rabbit will drink water if water is one of its neighbours
6. Data input		
6.1 Map reading	load_data(map)	[Passed] Reads in map and generates sprites
6.2 CSV reading	graph_data(name)	[passed] Data is read from a CSV file
7. Output		
7.1 Log to file	graph_data(self,name)	[Passed] Saves all game logs to file
7.2 CSV file output	write_file(self,name,data)	[Passed] Writes game data to file
7.3 Plot image and plot graph	pygame.image.save(…), graph_data(self,name)	[Passed] Saves an image when the game ends Plot a graph using the data from the CSV file

4. Introduction

4.1 Framework

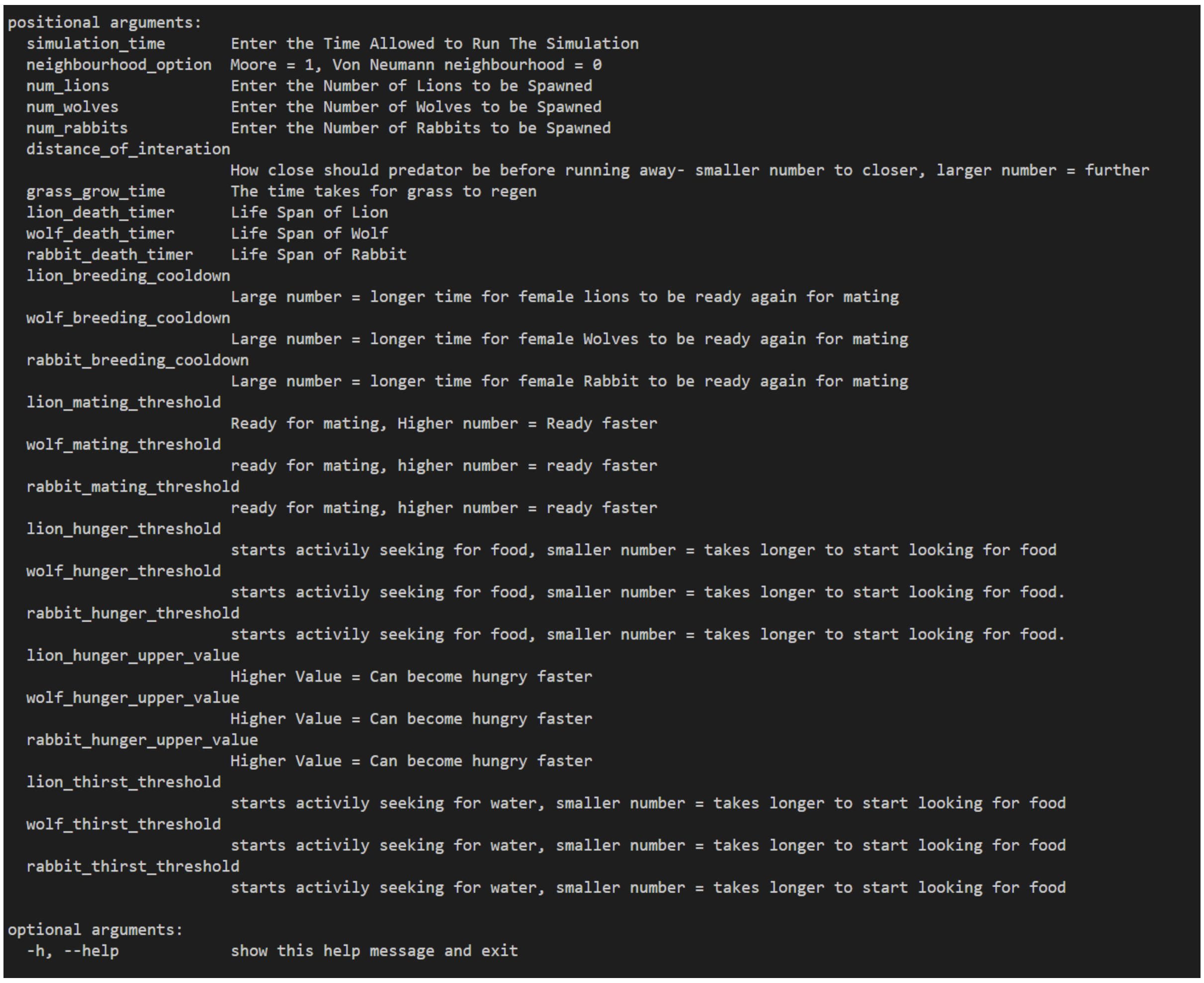
Animal kingdom uses "pygame" as the main framework, each Animal Class inherits the pygame's Sprite Class, this way I can easily add any new objects into sprite groups and manage them efficiently.

4.2 Display:

The sprites are represented by rectangles and have a x and y coordinate. This is how they are shown on the screen at various different locations.

They have been set up in layers, this way I can have a background and a foreground.

4.3 Inputs Parameters



```
positional arguments:
  simulation_time       Enter the Time Allowed to Run The Simulation
  neighbourhood_option  Moore = 1, Von Neumann neighbourhood = 0
  num_lions             Enter the Number of Lions to be Spawned
  num_wolves            Enter the Number of Wolves to be Spawned
  num_rabbits           Enter the Number of Rabbits to be Spawned
  distance_of_interation
                        How close should predator be before running away- smaller number to closer, larger number = further
  grass_grow_time       The time takes for grass to regen
  lion_death_timer      Life Span of Lion
  wolf_death_timer      Life Span of Wolf
  rabbit_death_timer    Life Span of Rabbit
  lion_breeding_cooldown
                        Large number = longer time for female lions to be ready again for mating
  wolf_breeding_cooldown
                        Large number = longer time for female Wolves to be ready again for mating
  rabbit_breeding_cooldown
                        Large number = longer time for female Rabbit to be ready again for mating
  lion_mating_threshold
                        Ready for mating, Higher number = Ready faster
  wolf_mating_threshold
                        ready for mating, higher number = ready faster
  rabbit_mating_threshold
                        ready for mating, higher number = ready faster
  lion_hunger_threshold
                        starts activily seeking for food, smaller number = takes longer to start looking for food
  wolf_hunger_threshold
                        starts activily seeking for food, smaller number = takes longer to start looking for food.
  rabbit_hunger_threshold
                        starts activily seeking for food, smaller number = takes longer to start looking for food.
  lion_hunger_upper_value
                        Higher Value = Can become hungry faster
  wolf_hunger_upper_value
                        Higher Value = Can become hungry faster
  rabbit_hunger_upper_value
                        Higher Value = Can become hungry faster
  lion_thirst_threshold
                        starts activily seeking for water, smaller number = takes longer to start looking for food
  wolf_thirst_threshold
                        starts activily seeking for water, smaller number = takes longer to start looking for food
  rabbit_thirst_threshold
                        starts activily seeking for water, smaller number = takes longer to start looking for food

optional arguments:
  -h, --help            show this help message and exit
```

4.4 Entities

The classes in sprites.py represents entities within the simulation. These classes include mountain, water, soil, boundary and the three animal classes. The classes have various different attributes which determines their behaviour in the simulation.

Each class instance has their unique value for their personal attributes such as self.hunger_limit, self.thrist_limit, self.mated, self.breedingCooldown, self.time etc.

self.time is especially important as it is used to implement cooldown effects such as having a cooldown after mating and prevents offspring's from instantly mating. This is obtained via pygame.time.get_ticks(), where it's called when an event happens, and it is then subtracted from the current time. If the result is greater than the set cooldown, it will allow the animal to mate again.

Entities also have a neighbour attribute which is in the form a list and is generated and appended based on the given input option of Von Neumann or Moore.

4.5 Movement:

Each animal class has a "move()" function which calls "collide_with_entity()". move() takes 2 integer value and adds to the objects current X and Y values respectively. An object will only move if it the future move does not collide with mountain, water or boundary blocks.

Each class also has an update function which updates the current rectangle location with the new x and y values and is called in the game loop which updates all the sprites' location on the screen.

There are 3 functions that allows movement towards or away from another target/entity. These are "chase()", "run_away()" and "look_for_nearest_water()".

"chase()" this function takes in 2 parameters, e.g., "chase(animal,prey)". It calls "animal.getlocation()", getlocation() is a function inside every animal class that obtains the animals current x and y location, and returns as a tuple. "chase()" will check its current x location with the prey's x location, similar, its y location to the prey's y location and move accordingly.

"run_away()" ,takes in two parameters, e.g. "run_away(animal,predator)" , it is the opposite of "chase()" where it tries to stay away from the predator. It uses similar logic to chase(), instead of moving towards the target location, if a target is nearby it tries to move away.

"look_for_nearest_water()", this function takes in one parameter, e.g. "look_for_nearest_water(animal)", it compares the animal's current location and nearby water sources and locates the closest one, moves towards it and drinks once water is one of the neighbours of the animal.

4.6 Data input and output

“load_data” will read in a map file and save the data in a 2d array. Which is then used in “new()”. In “new()”, it reads the map data obtained previously and add necessary terrain such as, water and mountains and boundaries.

Data is outputted in 4 ways, the population values for different animals are saved into a csv file. A graph is then plotted using the data from the csv file. There is a complete log file of all the actions of the animals and lastly an image of the end game state.

4.7 Behavioural logic

All the animal behavioural components are inside the event loop. Let’s use Lions as an example, each time “events()” are called which is always called in a while loop until a timer ends. A particular lion object will be selected, for the particular lion object it will have several depleting values such as hunger and thirst and reproduction level. Upon reaching a specific threshold, the lion object will perform certain actions.

For example when the lion is hungry, it will chase after food by calling, “chase(lion,prey)”, once it has caught up to the prey, in other words, the prey is a neighbour of the lion, the lion will then consume the prey and replenish its hunger level.

When the lion is thirsty, it will approach the nearest water source, this is done by calling “look_for_nearest_water()” and once a water source is a neighbour of the lion, it will consume the water and replenish its thirst level.

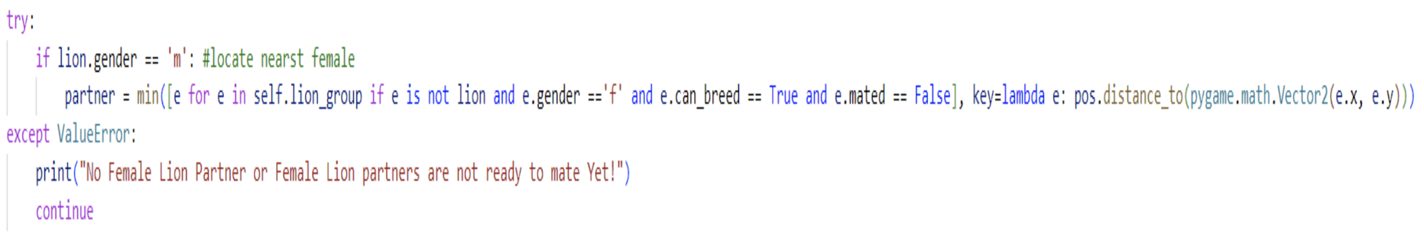
When the lion is ready for mating, the lion object will look for the nearest partner, and proceed to approach by calling chase(lion,partner) and if the partner is a neighbour of the lion object, it will create a new lion object called lion_offspring. Both the offspring and the female partner will have a cooldown set so they are restricted to mate for a certain time period, this timer can be changed in the input command line argument.

Otherwise, if it’s not hungry, not thirsty, and not wanting to mate, the lion will just move randomly.

4.8 Distance calculations

All distance calculations are done using vectors with a list comprehension.

For example,



```
try:
    if lion.gender == 'm': #locate nearst female
        partner = min([e for e in self.lion_group if e is not lion and e.gender =='f' and e.can_breed == True and e.mated == False], key=lambda e: pos.distance_to(pygame.math.Vector2(e.x, e.y)))
except ValueError:
    print("No Female Lion Partner or Female Lion partners are not ready to mate Yet!")
    continue
```

Essentially what it does is try to look for a partner, if there are no partners that suit the requirements then it will be ignored. Otherwise, first checks if the lion is a male, then it will look for other lions not itself and it has to be a female lion and that the female lion can breed and has not mated recently. If numerous lions fit those descriptions, it will calculate the distance to each the compatible lions and locates the nearest one, using min(). This base code was obtained from [1] , and I have amended it to suit my needs.

4.9 Extra Features

1) Graph figure output using matplotlib. Population of the animals are recorded against time and saved inside a CSV file and matplotlib.pyplot was used to plot the figures.

2)Parameter Sweep, explained in the user guide. It is implemented by using a bash file that repeatedly loops over parameters and running the python script with the different parameters.

3)Logging, all the print statements are logged and saved as a log file. [4]

5. Methodology

<u>Apparatus Used:</u>

Simulation was run using a Laptop with the following specs:

i7 12700H processor, 16GB RAM and RTX 3070ti Graphics Card

IDE: Visual studio code

Parameter sweep is ran using "git bash" terminal.

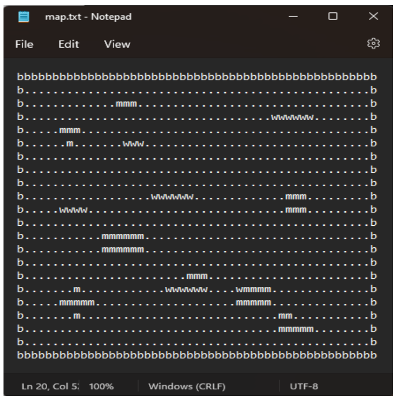
<u>Method:</u>

As I am trying to replicate realistic behaviours, therefore "hunger, thirst, and mating" have different values for different objects. This means animals objects will have different depleting hunger level and etc, even if they are from the same class of animals. Thus is not possible to reproduce the same exact results as I have implemented it with randomness.

Experiment 1: Control Group**,** simulations are run under default values for the input parameters.

To run the simulation in default mode → python3 main.py

Input map: map.txt (default)



```
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
b.................................................b
b.............mmmm................................b
b...................................wwwwww........b
b.....mmm.........................................b
b......m.......www................................b
b.................................................b
b.................................................b
b.................................................b
b..................wwwwww.............mmm.........b
b.....wwww............................mmm.........b
b.................................................b
b...........mmmmmm................................b
b...........mmmmmm................................b
b.................................................b
b.......................mmm.......................b
b.......m............wwwwww....wmmmm..............b
b.....mmmmm....................mmmmm..............b
b.......m............................mm...........b
b....................................mmmmm........b
b.................................................b
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
```

Experiment 2: Experimental Group, simulations are run under different values for the input parameters.

Experimental group with be when we change up the input parameters such as number of lions, number of wolves and number of rabbits. We can extend this by changing their values which alters they behaviours in a certain way, for example, life span of animals and how fast they get hungry/thirsty.

Run 1: Increased number of animals

→ python3 main.py map.txt 150 0 **5 8 20**

Input map: map.txt

Run 2: Same number of animals as run 1, different neighbourhood option.

→ python3 main.py map.txt 150 **1** 5 8 20

Input map: map.txt

Run 3: Same number of animals as run 1,2, increased life span of animals

→ python3 main.py map.txt 150 0 5 8 20 2.0 500000 **150 80 70**

Input map: map.txt

Run 4: Same number of animals as run 1,2,3. All animals have lower hunger threshold.

→ python3 main.py map.txt 150 0 5 8 20 2.0 500000 150 80 70 1000 600 100 80 85 95 **50 50 50**

Options to directly change the default values to 50 50 50 in config.py

Input map: map.txt

Run 5: Same number of animals as run 1,2,3. Lions and wolf breeding cooldown lowered, rabbit remain constant.

→ python3 main.py map.txt 150 0 5 8 20 2.0 500000 150 80 70 **600 300** 100 80 85 95 **50 50 50**

input map: map.txt

Run 6: Keep all changes made, use smaller map.

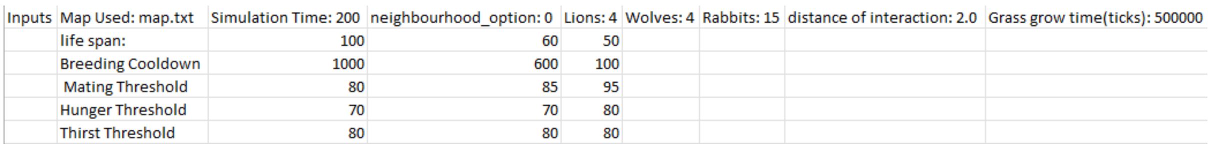
→ python3 main.py **map_2.txt** 150 0 5 8 20 2.0 500000 **150 80 70** **600 300** 100 80 85 95 **50 50 50**

input map: map_2.tx

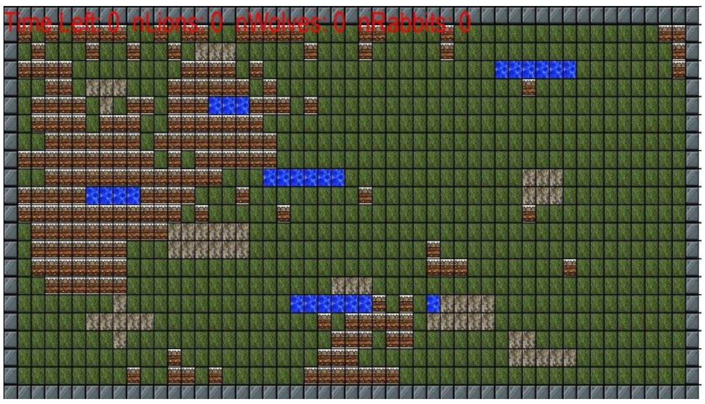
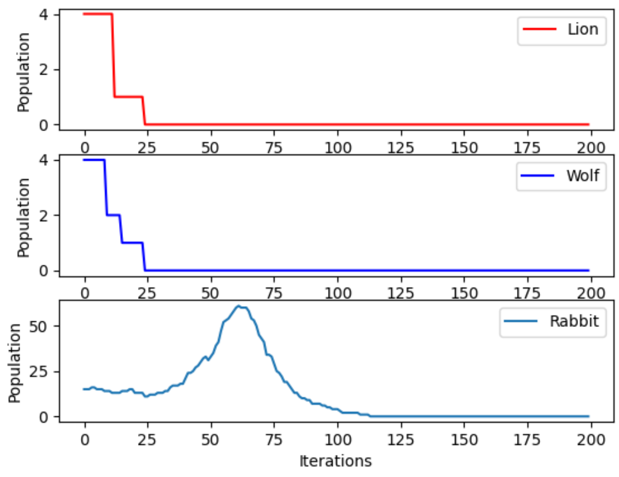
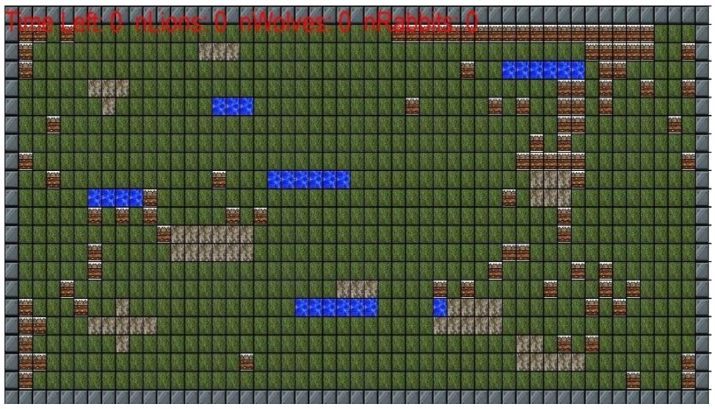
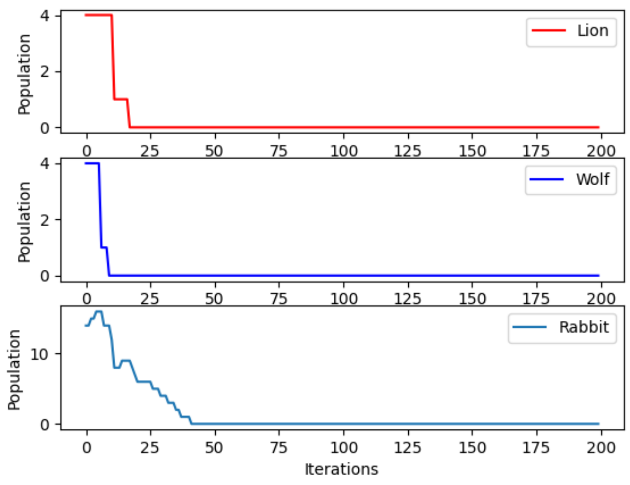
6. Results

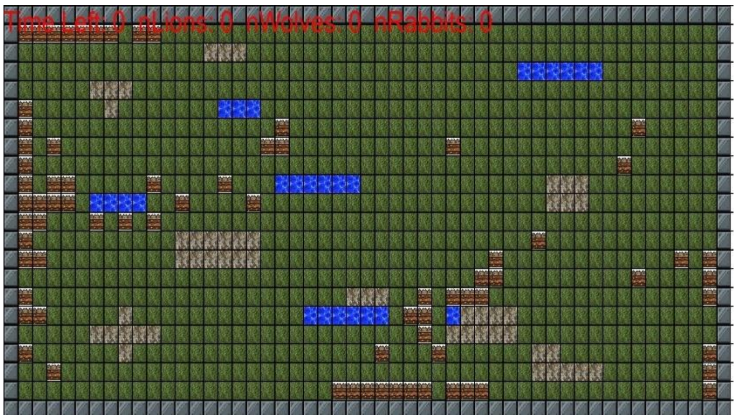
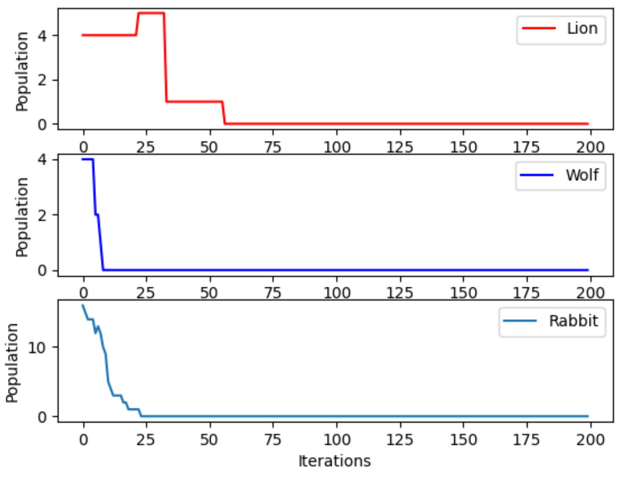
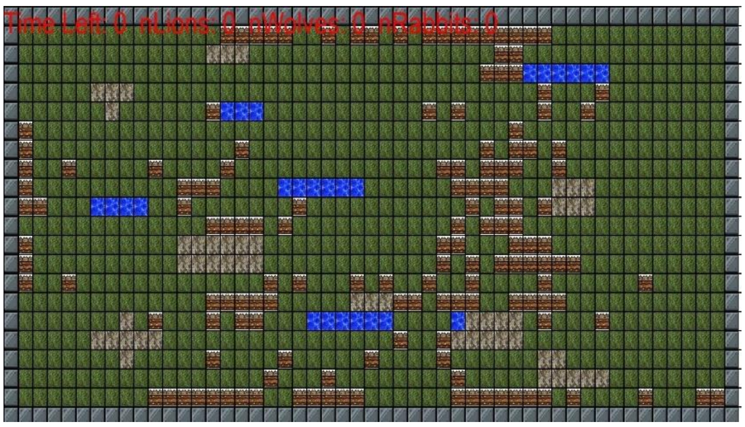
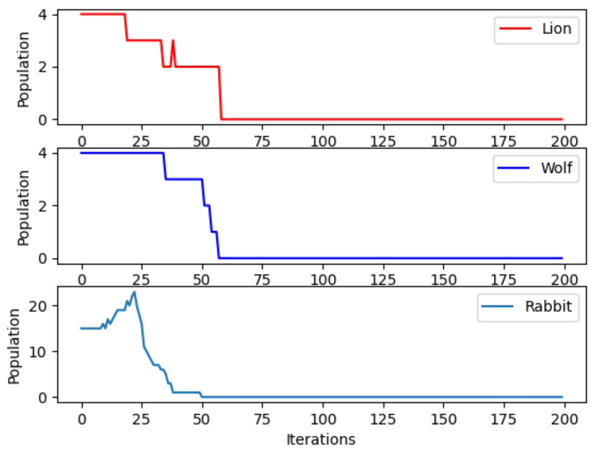
Experiment 1: 4 runs under default parameters

Default parameters: Source: simulation20220526215754984701.csv



Inputs	Map Used: map.txt	Simulation Time: 200	neighbourhood_option: 0	Lions: 4	Wolves: 4	Rabbits: 15	distance of interaction: 2.0	Grass grow time(ticks): 500000
	life span:	100	60	50				
	Breeding Cooldown	1000	600	100				
	Mating Threshold	80	85	95				
	Hunger Threshold	70	70	80				
	Thirst Threshold	80	80	80				

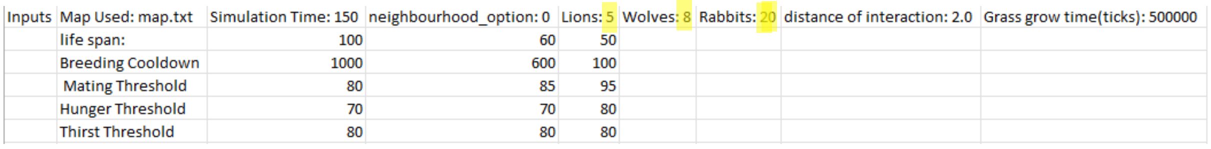
Run 1: simulation20220526214054309304.png	Run 2: simulation20220526214419170671.png
 	 
Summary: - Low beginning population for lion and wolves/ limited females lead to early extinction. - Rabbits on the other hand was able to breed and populate given no threat of apex predators. - Rabbit's population reached its peak at around 62 iterations.	Summary: - Likley due to limited females/ not eough food. - All animals extincted early.

Run 3: simulation20220526214509849491.png	Run 4: simulation20220526215754984701.png
 	 
Summary: - Very similar to run 2. - However a new lion was born but quickly died after due to no food avaible.	Summary: - Rabbit’s population grew early on, giving enough food for wolves and lions to sustain making them live longer than run 2 and 3.

Experiment 2: 6 runs under different scenarios

Run 1: More animals than default, default neighbourhood

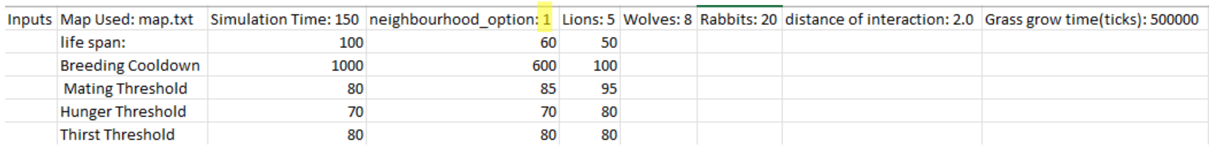
Source: simulation20220527192718391832.csv



Inputs	Map Used: map.txt	Simulation Time: 150	neighbourhood_option: 0	Lions: 5	Wolves: 8	Rabbits: 20	distance of interaction: 2.0	Grass grow time(ticks): 500000
	life span:	100	60	50				
	Breeding Cooldown	1000	600	100				
	Mating Threshold	80	85	95				
	Hunger Threshold	70	70	80				
	Thirst Threshold	80	80	80				

Run 2: More animals than default, different neighbourhood

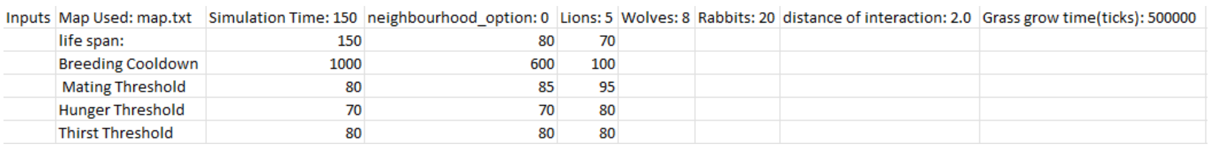
Source: simulation20220527193121138497.csv



Inputs	Map Used: map.txt	Simulation Time: 150	neighbourhood_option: 1	Lions: 5	Wolves: 8	Rabbits: 20	distance of interaction: 2.0	Grass grow time(ticks): 500000
	life span:	100	60	50				
	Breeding Cooldown	1000	600	100				
	Mating Threshold	80	85	95				
	Hunger Threshold	70	70	80				
	Thirst Threshold	80	80	80				

Run 3: Same number of animals as run 1 and 2, default neighbourhood however longer lifespan for each animal, everything else default

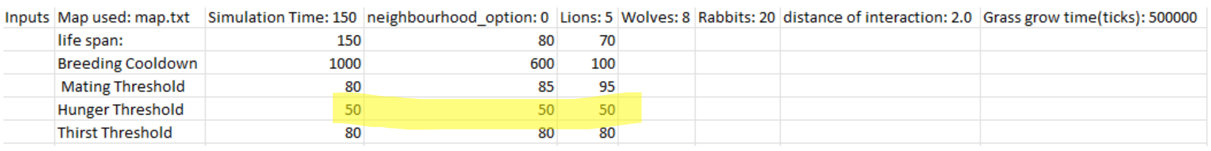
Source: simulation20220527201202157266.csv



Inputs	Map Used: map.txt	Simulation Time: 150	neighbourhood_option: 0	Lions: 5	Wolves: 8	Rabbits: 20	distance of interaction: 2.0	Grass grow time(ticks): 500000
	life span:	150	80	70				
	Breeding Cooldown	1000	600	100				
	Mating Threshold	80	85	95				
	Hunger Threshold	70	70	80				
	Thirst Threshold	80	80	80				

Run 4: Same number of animals as run 1,2,3. All animals have lower hunger threshold.

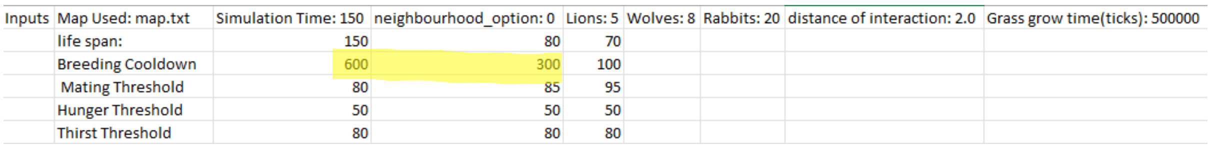
Source: simulation20220527211941580282.csv



Inputs	Map used: map.txt	Simulation Time: 150	neighbourhood_option: 0	Lions: 5	Wolves: 8	Rabbits: 20	distance of interaction: 2.0	Grass grow time(ticks): 500000
	life span:	150	80	70				
	Breeding Cooldown	1000	600	100				
	Mating Threshold	80	85	95				
	Hunger Threshold	50	50	50				
	Thirst Threshold	80	80	80				

Run 5: Same number of animals as run 1,2,3. Lions and wolf breeding cooldown lowered, rabbit remain constant.

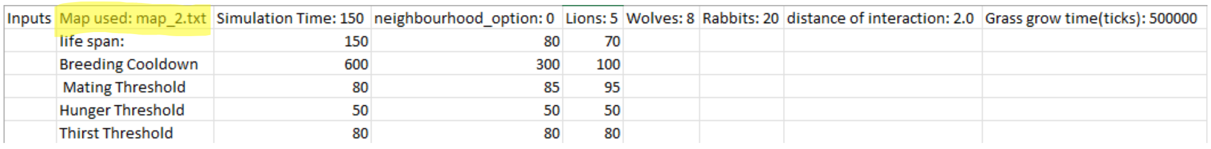
Source: simulation20220527213348675892.csv



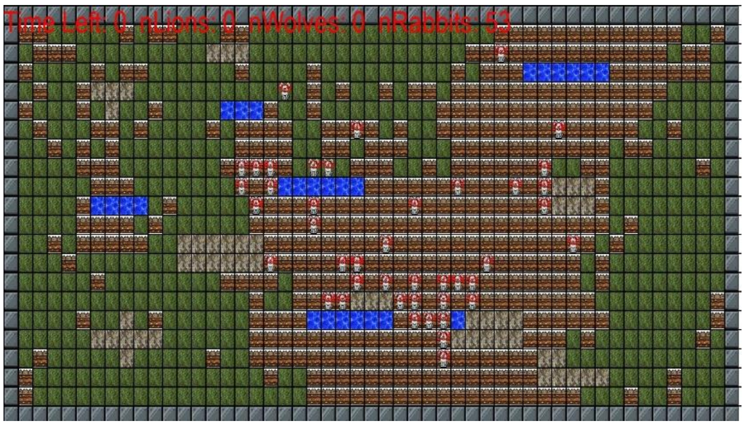
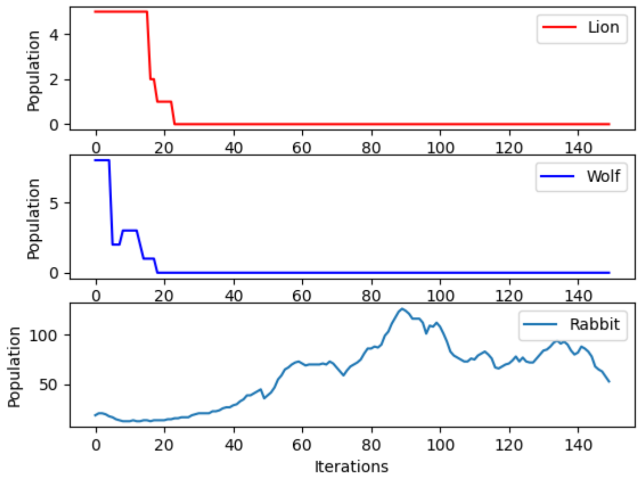
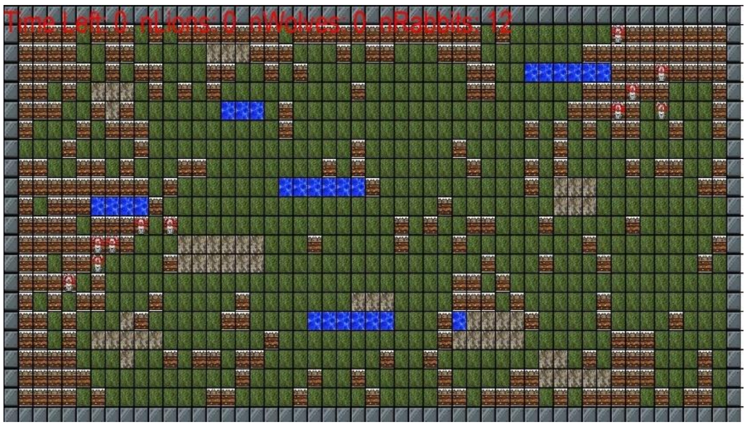
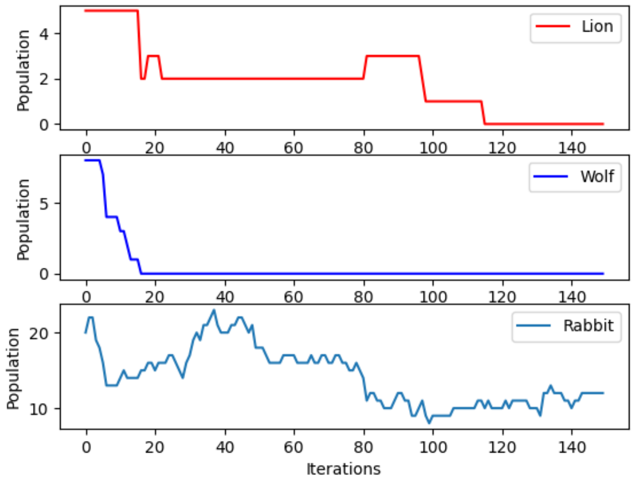
Inputs	Map Used: map.txt	Simulation Time: 150	neighbourhood_option: 0	Lions: 5	Wolves: 8	Rabbits: 20	distance of interaction: 2.0	Grass grow time(ticks): 500000
	life span:	150	80	70				
	Breeding Cooldown	600	300	100				
	Mating Threshold	80	85	95				
	Hunger Threshold	50	50	50				
	Thirst Threshold	80	80	80				

Run 6: Keep all changes made, use smaller map.

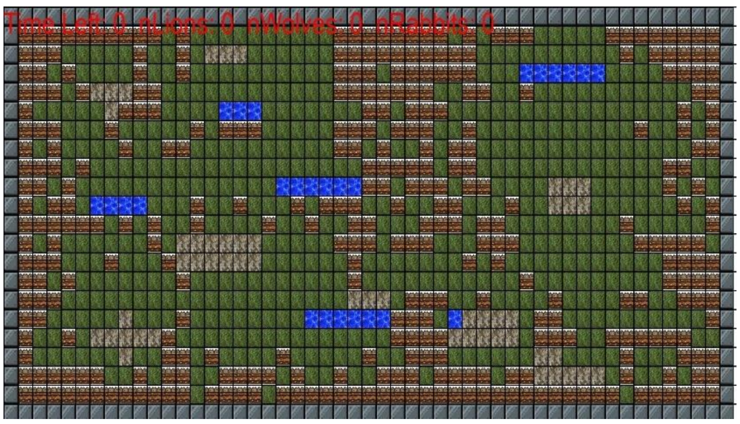
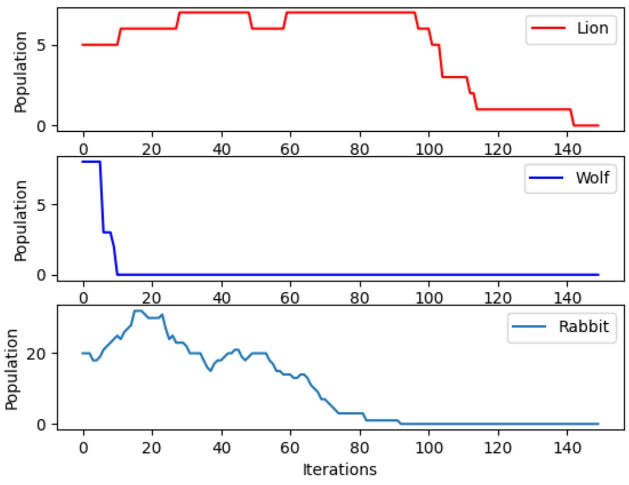
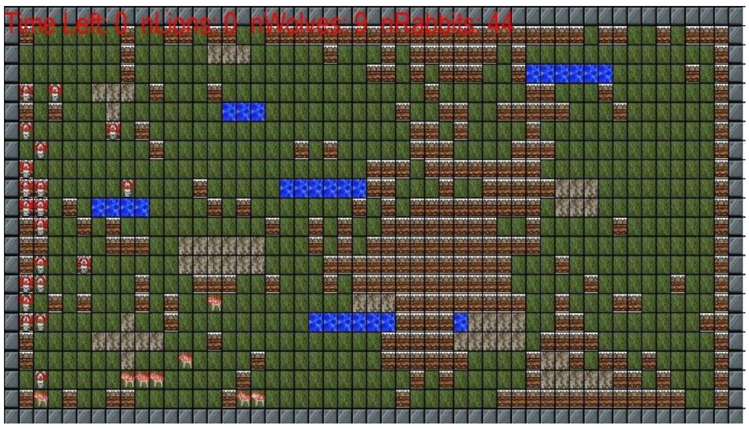
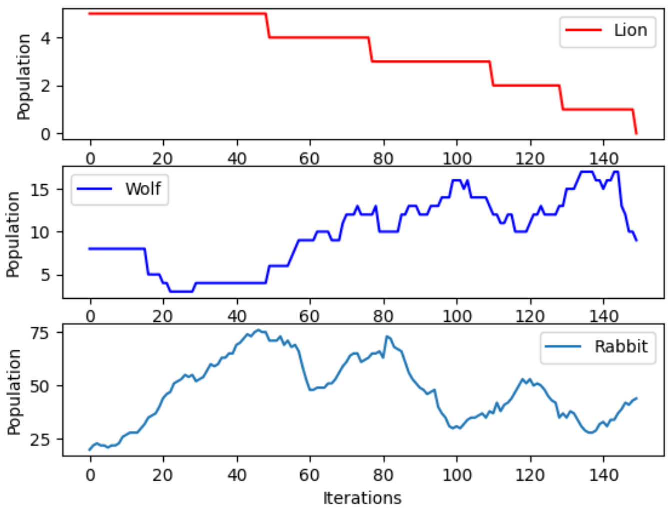
Source: simulation20220527214308298080.csv



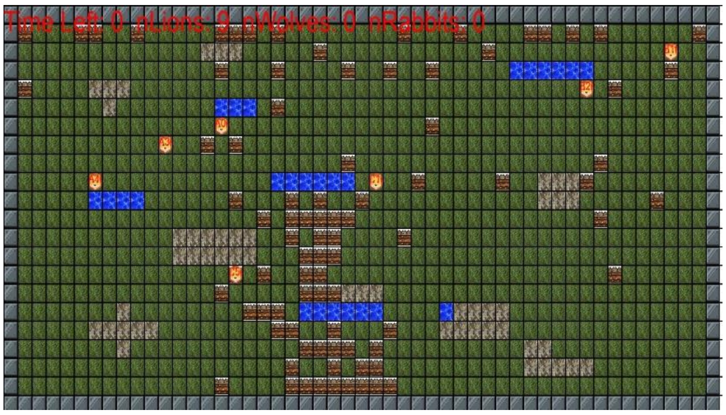
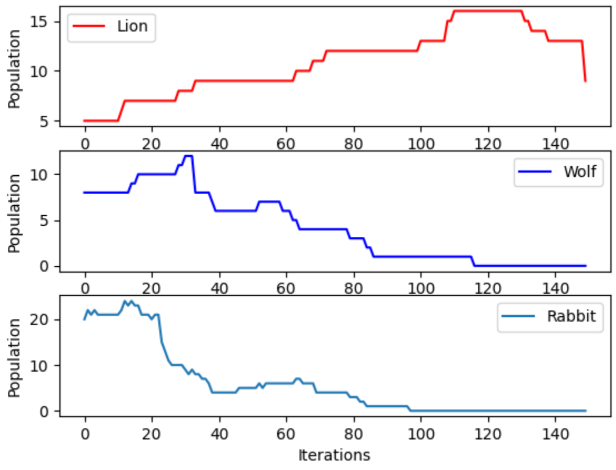
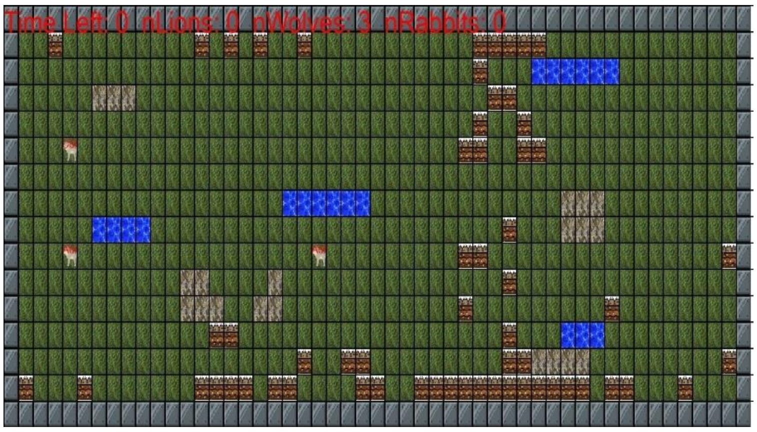
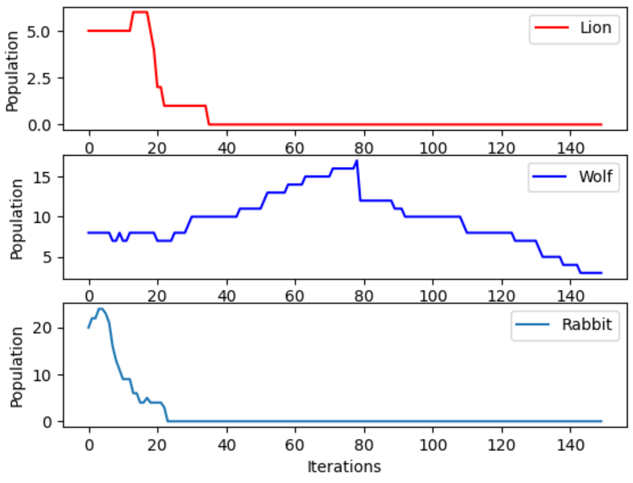
Inputs	Map used: map_2.txt	Simulation Time: 150	neighbourhood_option: 0	Lions: 5	Wolves: 8	Rabbits: 20	distance of interaction: 2.0	Grass grow time(ticks): 500000
	life span:	150	80	70				
	Breeding Cooldown	600	300	100				
	Mating Threshold	80	85	95				
	Hunger Threshold	50	50	50				
	Thirst Threshold	80	80	80				

Run 1: More animals, same neighbourhood, rest default	Run 2: More animals, different neighbourhood, rest default
Figure: plot20220527192718391832.jpg  	Figure plot20220527193121138497.jpg  
Summary: - Even with an increased beginning number of lions and wolves, they died out relatively quickly. Possibly due to limited females and was not able to breed. - With the lack of predators, rabbits were able to grow but didn't grow out of control due to limited food . - Indicated by the number of soil blocks. - Rabbit population peaked at around 90.	Summary: - Similar to run 1 which is predicable given, only neighbourhood option was changed. - However lion lived longer due to easier access to food, possibly due to Moore neighbourhood. - Rabbit population peaked earlier possibly due to easier access to neighbouring partners.

Ben Huang 21020526

Run 3: Same number of animals as run 1 and 2, however longer lifespan for each animal. Default neighbourhood.	Run 4: Same number of animals as run 1,2,3. All animals have lower hunger threshold.
 	 
Summary: - In this scenario I have given lions the most increase in life span. Which is reflected in the figure, they almost lived to the end of simulation given there are limited food. - Wolves died early due to not being around food and died before they caught up to the prey, this is reflected in the logs. - Modest increase in age of rabbits didn't help too much as predators lived longer as well.	Summary: - In this scenario, all animals have their hunger thresh hold decreased, meaning they will take longer before they get hungry enough to actively search for food. - Since this time is longer, animals have a longer safe period and therefore can breed more. This is reflected in the log and the figure.

Ben Huang 21020526

Run 5: Same number of animals as run 1,2,3. Lions and wolf breeding cooldown lowered, rabbit remain constant.	Run 6: Keep all changes made, use smaller map.
 	 
Summary: - Lowered breeding cooldown meaning animals can bread more often. - When lion and wolf breeding cooldown are lowered. Their population growth increased. - While rabbits lowered since they are more predators, and its breeding cooldown remained the same.	Summary: - On a smaller map, animals are more likely to meet each other. - In this simulation, lion died out very quickly since it did not have much female lions. - Once lions died, wolves roamed free and became the apex predator, which resulted in the rapid decay of rabbit population.

7. Conclusion

I believe that the extensions, “movements, boundaries and terrain, interaction, non- moving targets and visualisation/results” have been completed at a satisfactory level.

Due to complexity of the program and the time constraints, I wasn’t able to implement more animals and special terrain for the animals.

The idea of randomness in the animal’s attribute made runs vastly different to each other, I am not sure if this was a good idea or should I have just left it as a constant depleting value instead of a random one.

Other than that, the program achieved what I had ambitioned. In the results section, the runs reflected directly to changes that was made, and I believe that was a wonderful result to see.

7.1 Future Work

This program still lacks in some respects such as visualisation and scalability. Especially scalability, at the moment the game heavily depends on graphics as every time a new animal is spawned a new sprite is created. When the simulation has around seven hundred sprites it will lag and slow down. This makes this program incapable of simulating substantial amounts of animals. In the future, if given the opportunity, I wish to increase its running capacity and scalability to simulate thousands if not millions of animals. In terms of visualisation, animations such as drinking, and eating should be implemented to indict the interaction of the animals.

Reference

1. PyGameExamplesAndAnswers/pygame_math_vector_and_reflection.md at master · Rabbid76/PyGameExamplesAndAnswers [Internet]. GitHub. 2022 [cited 29 May 2022]. Available from: https://github.com/Rabbid76/PyGameExamplesAndAnswers/blob/master/documentation/pygame/pygame_math_vector_and_reflection.md

2. http://www.fantasticmaps.com/2015/01/killing-hex-bugs-and-the-importance-of-sprite-sheets/ [Images]
3. https://www.flaticon.com/ [images]
4. https://www.delftstack.com/howto/python/python-logging-stdout/ [logging]