KING MONGKUT'S UNIVERSITY OF TECHNOLOGY LATKRABANG

ARTIFICIAL INTELLIGENCE MINOR PROGRAM

<u>01416305 - ARTIFICIAL INTELLIGENCE TECHNOLOGY</u>

NATURAL SELECTION SIMULATION

INSTRUCTED BY: DR. RAPEEPAN PROMYOO, DR. POOH EAMCHAROENYING

NAME : JATUPONG OBOUN

STUDENT ID : 62010096

DEPARTMENT : COMPUTER ENGINEERING

Natural Selection Simulation

Natural Selection is something many people have learned since high school in biology class but even though it might seem to make sense to everyone but usually you don't see it in real life in our human life span, right? With that being said, it brings a lot of curiosity to me, and someday I would find a good clarification to it.

A couple of years have gone and now I'm 3rd year at university. There is a channel name Primer that mainly did simulation content which I found interesting especially the one where he explains natural selection, It is the best natural selection theory explanation that I have ever heard.

At the same time, I also start learning about the Genetic Algorithm in Artificial Intelligence Technologies class which I also found very interesting and really related to both natural selection and simulation content from Primer that I found very interesting and can be done by myself.

Therefore, I decided to make this project, by making a natural selection simulation using Python and matplotlib to visualize the result and find the proves for existing theories that I've learned since high school.

Objective

- To make a simulation in python by creating my own environmental rules.
- Proves the real-world theories. (That was already proved.)
- Gain experience using Python and matplotlib in general.

Methodology

1. Algorithm Formulation

Environment rules

1. Map will be size sizei * sizej sq.unit
2. There will be food randomly generated in 64% of the area from the center of the map for each generation

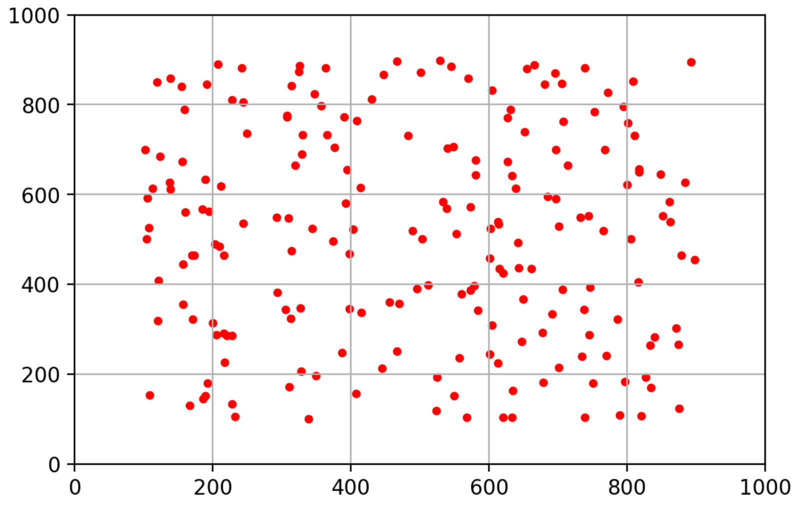


Figure 1 Show map size 1000 * 1000 sq.unit with food only generate in between [100,800] both x and y axis (64% area in the map)

3. Creatures will be generate n_creature in first generation which will randomly spawn along the green line in Figure 2

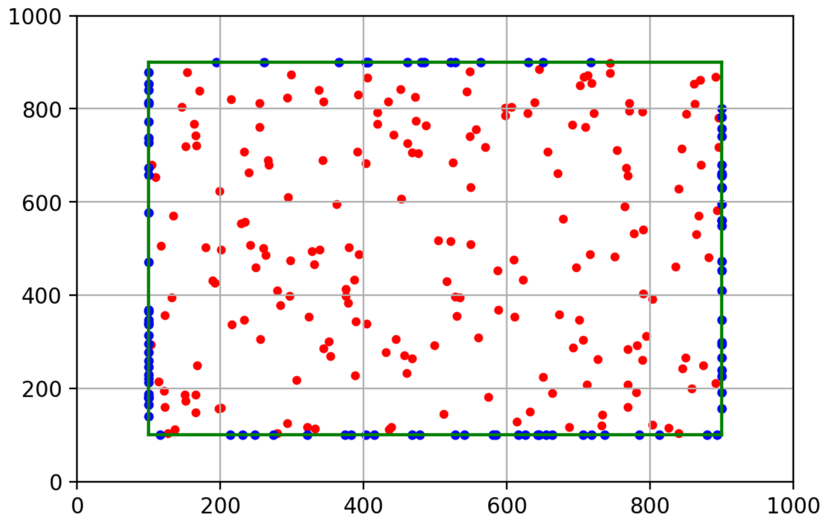


Figure 2 Show 100 creatures(blue) that get random spawn on the green line

4. In each time (second), every creature will randomly move with crea.speed unit and If they find any food in their crea.sight they will eat one food per single time unit.

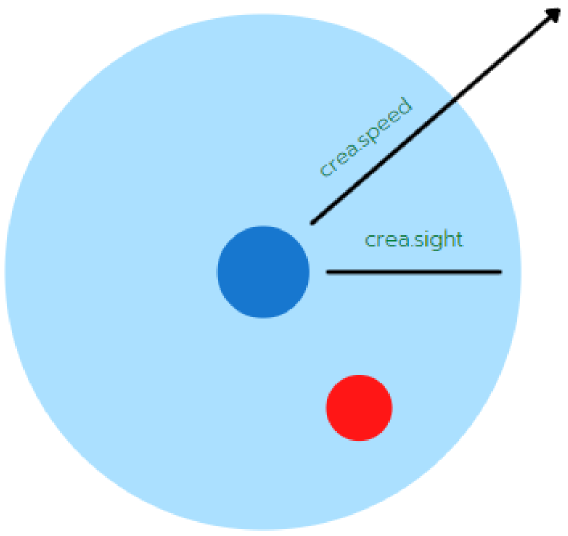


Figure 3 Shows how each creature if they find a food in crea.sight radius they will be able to eat the food (only one per time unit) and will be move in a random direction for crea.speed long.

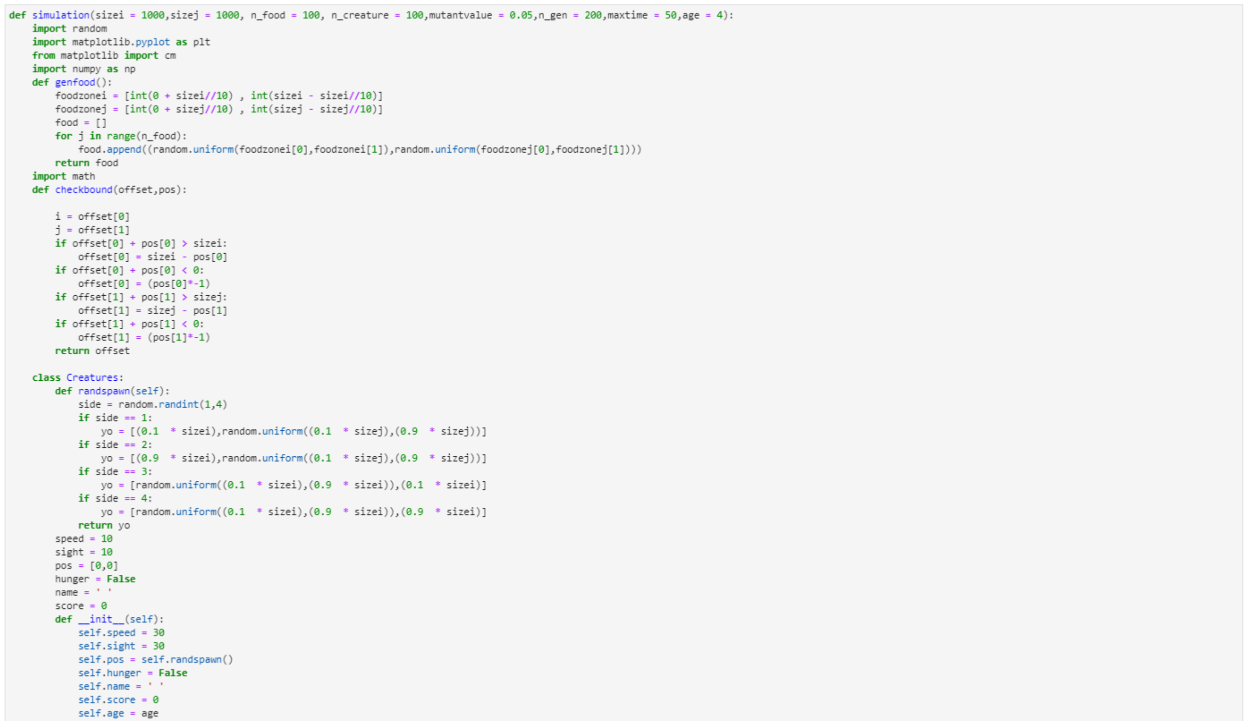
5. At the end of every generation, It will make one child per 2 food that they already ate. With every of their child will have a chance to mutant as
   - newborn.speed = parent.speed * random.uniform(1-mutantvalue,1+mutantvalue)
   - newborn.sight = parent.sight * random.uniform(1-mutantvalue,1+mutantvalue)
   and each of them will have a life span for age generations and creatures will die if their age is equal to their life span or they aren't able to eat at least 1 food in that generation.

2. Code

Library

- Matplotlib & Matplotlib animation
- numpy
- random

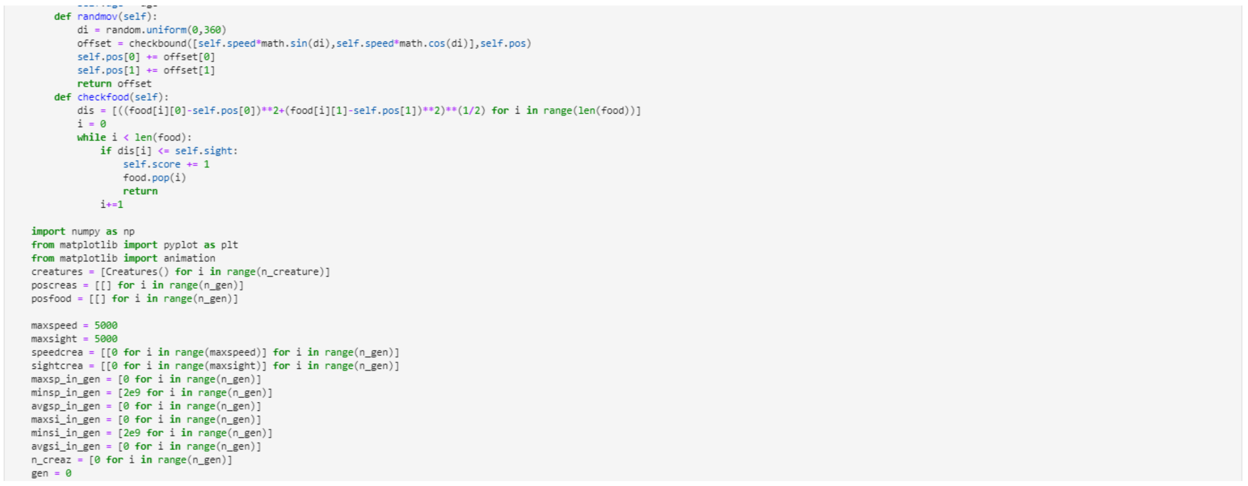
Code



```python
def simulation(sizei = 1000,sizej = 1000, n_food = 100, n_creature = 100,mutantvalue = 0.05,n_gen = 200,maxtime = 50,age = 4):
    import random
    import matplotlib.pyplot as plt
    from matplotlib import cm
    import numpy as np
    def genfood():
        foodzonei = [int(0 + sizei//10) , int(sizei - sizei//10)]
        foodzonej = [int(0 + sizej//10) , int(sizej - sizej//10)]
        food = []
        for j in range(n_food):
            food.append((random.uniform(foodzonei[0],foodzonei[1]),random.uniform(foodzonej[0],foodzonej[1])))
        return food
    import math
    def checkbound(offset,pos):

        i = offset[0]
        j = offset[1]
        if offset[0] + pos[0] > sizei:
            offset[0] = sizei - pos[0]
        if offset[0] + pos[0] < 0:
            offset[0] = (pos[0]*-1)
        if offset[1] + pos[1] > sizej:
            offset[1] = sizej - pos[1]
        if offset[1] + pos[1] < 0:
            offset[1] = (pos[1]*-1)
        return offset

    class Creatures:
        def randspawn(self):
            side = random.randint(1,4)
            if side == 1:
                yo = [(0.1  * sizei),random.uniform((0.1  * sizej),(0.9  * sizej))]
            if side == 2:
                yo = [(0.9  * sizei),random.uniform((0.1  * sizej),(0.9  * sizej))]
            if side == 3:
                yo = [random.uniform((0.1  * sizei),(0.9  * sizei)),(0.1  * sizei)]
            if side == 4:
                yo = [random.uniform((0.1  * sizei),(0.9  * sizei)),(0.9  * sizei)]
            return yo
        speed = 10
        sight = 10
        pos = [0,0]
        hunger = False
        name = ' '
        score = 0
        def __init__(self):
            self.speed = 30
            self.sight = 30
            self.pos = self.randspawn()
            self.hunger = False
            self.name = ' '
            self.score = 0
            self.age = age
```



```python
        def randmov(self):
            di = random.uniform(0,360)
            offset = checkbound([self.speed*math.sin(di),self.speed*math.cos(di)],self.pos)
            self.pos[0] += offset[0]
            self.pos[1] += offset[1]
            return offset
        def checkfood(self):
            dis = [((food[i][0]-self.pos[0])**2+(food[i][1]-self.pos[1])**2)**(1/2) for i in range(len(food))]
            i = 0
            while i < len(food):
                if dis[i] <= self.sight:
                    self.score += 1
                    food.pop(i)
                    return
                i+=1

    import numpy as np
    from matplotlib import pyplot as plt
    from matplotlib import animation
    creatures = [Creatures() for i in range(n_creature)]
    poscreas = [[] for i in range(n_gen)]
    posfood = [[] for i in range(n_gen)]

    maxspeed = 5000
    maxsight = 5000
    speedcrea = [[0 for i in range(maxspeed)] for i in range(n_gen)]
    sightcrea = [[0 for i in range(maxsight)] for i in range(n_gen)]
    maxsp_in_gen = [0 for i in range(n_gen)]
    minsp_in_gen = [2e9 for i in range(n_gen)]
    avgsp_in_gen = [0 for i in range(n_gen)]
    maxsi_in_gen = [0 for i in range(n_gen)]
    minsi_in_gen = [2e9 for i in range(n_gen)]
    avgsi_in_gen = [0 for i in range(n_gen)]
    n_creaz = [0 for i in range(n_gen)]
    gen = 0
```

```python
for gen in range(n_gen):
    food = genfood()
    for crea in creatures:
        crea.pos = crea.randspawn()
    time = 0
    poscreas_in_this_gen = list([])
    posfood_in_this_gen = list([])
    while time < maxtime:
        poscreas_in_this_gen.append([])
        posfood_in_this_gen.append([])
        for crea in creatures:
            crea.randmov()
            crea.checkfood()
            cache = [crea.pos[0],crea.pos[1]]
            poscreas_in_this_gen[-1].append(cache)
        for f in food:
            cache = [f[0],f[1]]
            posfood_in_this_gen[-1].append(cache)
        time+=1
```

```python
    newcreas = []
    for crea in creatures:
        maxsp_in_gen[gen] = max(maxsp_in_gen[gen],int(crea.speed))
        maxsi_in_gen[gen] = max(maxsi_in_gen[gen],int(crea.sight))
        avgsp_in_gen[gen] += int(crea.speed)
        avgsi_in_gen[gen] += int(crea.sight)
        minsp_in_gen[gen] = min(minsp_in_gen[gen],int(crea.speed))
        minsi_in_gen[gen] = min(minsi_in_gen[gen],int(crea.sight))
        speedcrea[gen][int(crea.speed)]+=1
        sightcrea[gen][int(crea.sight)]+=1
    avgsp_in_gen[gen] /= len(creatures)
    avgsi_in_gen[gen] /= len(creatures)
    n_creaz[gen] = len(creatures)
    maxsp = -1
    maxsi = -1
    for i in range(len(creatures)):
        crea = creatures[i]
        if crea.score >= 2:
            parent = Creatures()
            parent.speed = crea.speed
            parent.sight = crea.sight
            parent.age = crea.age-1
            for i in range(crea.score//2):
                newborn = Creatures()
                newborn.speed = parent.speed * random.uniform(1-mutantvalue,1+mutantvalue)
                newborn.sight = parent.sight * random.uniform(1-mutantvalue,1+mutantvalue)
                maxsp = max(maxsp,newborn.speed)
                maxsi = max(maxsi,newborn.sight)
                newcreas.append(newborn)
            if parent.age >=1:
                newcreas.append(parent)
        elif crea.score == 1:
            parent = Creatures()
            parent.speed = crea.speed
            parent.sight = crea.sight
            parent.age = crea.age-1
            if parent.age >=1:
                newcreas.append(parent)
    creatures = newcreas[:]
    random.shuffle(creatures)
    poscreas[gen]= list(poscreas_in_this_gen[:])
    posfood[gen]= list(posfood_in_this_gen[:])
```

```python
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(16,6), dpi=80)
line1, = ax1.plot(np.arange(0, n_gen, 1),avgsp_in_gen[:n_gen],color='b',label ='Speed')
line2, = ax1.plot(np.arange(0, n_gen, 1),avgsi_in_gen[:n_gen],color='r',label ='Sight')
ax1.fill_between(np.arange(0, n_gen, 1), maxsp_in_gen[:n_gen],minsp_in_gen[:n_gen], alpha=0.2,color='b')
ax1.fill_between(np.arange(0, n_gen, 1), maxsi_in_gen[:n_gen],minsi_in_gen[:n_gen], alpha=0.2,color='r')
ax2.set_ylim([0,100])
ax1.set_xlim([0,n_gen])
ax1.grid()
ax2.plot(np.arange(0, n_gen, 1),n_creaz[:n_gen],color='g')
ax2.grid()
ax1.legend((line1, line2), ('Speed', 'Sight'), loc='upper left', shadow=True)
st = str('mutantrate = '+str(mutantvalue))
ax1.set_title(st)
ax2.set_title('number of creature(s)')
print('Food :',n_food, ' n_creature :',n_creature,' n_gen : ',n_gen,' maxtime : ',maxtime)
```

```python
def simulation_gene_comparison_nsim(sizei = 1000,sizej = 1000,
                                    n_food = 100, n_creatureA = 50,AGE = {'A':4,'B':4},
                                    n_creatureB = 50,mutantvalueA = 0.04,mutantvalueB = 0.1,
                                    n_gen = 200,maxtime = 50,number_of_simulations = 10):
    import random
    import matplotlib.pyplot as plt
    from matplotlib import cm
    import numpy as np
    def genfood():
        foodzonei = [int(0 + sizei//10) , int(sizei - sizei//10)]
        foodzonej = [int(0 + sizej//10) , int(sizej - sizej//10)]
        food = []
        for j in range(n_food):
            food.append((random.uniform(foodzonei[0],foodzonei[1]),random.uniform(foodzonej[0],foodzonej[1])))
        return food
    import math
    def checkbound(offset,pos):
        i = offset[0]
        j = offset[1]
        if offset[0] + pos[0] > sizei:
            offset[0] = sizei - pos[0]
        if offset[0] + pos[0] < 0:
            offset[0] = (pos[0]*-1)
        if offset[1] + pos[1] > sizej:
            offset[1] = sizej - pos[1]
        if offset[1] + pos[1] < 0:
            offset[1] = (pos[1]*-1)
        return offset
```

```python
    class Creatures:
        def randspawn(self):
            side = random.randint(1,4)
            if side == 1:
                yo = [(0.1  * sizei),random.uniform((0.1  * sizej),(0.9  * sizej))]
            if side == 2:
                yo = [(0.9  * sizei),random.uniform((0.1  * sizej),(0.9  * sizej))]
            if side == 3:
                yo = [random.uniform((0.1  * sizei),(0.9  * sizei)),(0.1  * sizei)]
            if side == 4:
                yo = [random.uniform((0.1  * sizei),(0.9  * sizei)),(0.9  * sizei)]
            return yo
        speed = 10
        sight = 10
        pos = [0,0]
        hunger = False
        name = ' '
        score = 0
```

```python
        def __init__(self, mutantrate,gene,age):
            self.speed = 30
            self.sight = 30
            self.pos = self.randspawn()
            self.hunger = False
            self.name = ' '
            self.score = 0
            self.age = age
            self.mutantrate = mutantrate
            self.gene = gene

        def randmov(self):
            di = random.uniform(0,360)
            offset = checkbound([self.speed*math.sin(di),self.speed*math.cos(di)],self.pos)
            self.pos[0] += offset[0]
            self.pos[1] += offset[1]
            return offset
        def checkfood(self):
            dis = [((food[i][0]-self.pos[0])**2+(food[i][1]-self.pos[1])**2)**(1/2) for i in range(len(food))]
            i = 0
            while i < len(food):
                if dis[i] <= self.sight:
                    self.score += 1
                    food.pop(i)
                    return
                i+=1
    maxspeed = 200
    maxsight = 200
    winner = {'A':[],'B':[]}
    winny = {'A':[0 for i in range(n_gen)],'B':[0 for i in range(n_gen)]}
    n_creaz_all = {'A':[0 for i in range(n_gen)],'B':[0 for i in range(n_gen)]}
    lastgen = {'A':{'speed':[0 for i in range(maxspeed)],'sight':[0 for i in range(maxspeed)]},'B' : {'speed':[0 for i in range(maxspeed)],'sight':[0 for i in range(maxspeed)]}}
    for SIM in range(number_of_simulations):
        creatures = []
        for i in range(n_creatureA):
            creatures.append(Creatures(mutantrate = mutantvalueA,gene = 'A',age = AGE['A']))
        for i in range(n_creatureB):
            creatures.append(Creatures(mutantrate = mutantvalueB,gene = 'B',age = AGE['B']))
        n_creaz = {'A':[0 for i in range(n_gen)],'B':[0 for i in range(n_gen)]}
        gen = 0
        hey = False
        while gen < n_gen:
            food = genfood()
            for crea in creatures:
                crea.pos = crea.randspawn()
            time = 0
            while time < maxtime:
                for crea in creatures:
                    crea.randmov()
                    crea.checkfood()
                time+=1
            newcreas = []
        for i in range(len(creatures)):
            crea = creatures[i]
            n_creaz[crea.gene][gen] += 1
            if crea.score >= 1:
                parent = Creatures(mutantrate = crea.mutantrate,gene = crea.gene,age = crea.age-1)
                parent.speed = crea.speed
                parent.sight = crea.sight
                for j in range(crea.score//2):
                    newborn = Creatures(mutantrate = crea.mutantrate,gene = crea.gene,age = AGE[crea.gene])
                    newborn.speed = parent.speed * random.uniform(1-newborn.mutantrate,1+parent.mutantrate)
                    newborn.sight = parent.sight * random.uniform(1-newborn.mutantrate,1+parent.mutantrate)
                    newcreas.append(newborn)
                if parent.age >=1:
                    newcreas.append(parent)
        n_creaz_all['A'][gen] += n_creaz['A'][gen]
        n_creaz_all['B'][gen] += n_creaz['B'][gen]
        if n_creaz['A'][gen] == 0:
            if hey == False:
                winner['B'].append(gen)
                winny['B'][gen] += 1
                hey = True
        elif n_creaz['B'][gen] == 0:
            if hey == False:
                winner['A'].append(gen)
                winny['A'][gen] += 1
                hey = True
        creatures = newcreas[:]
        random.shuffle(creatures)
        gen += 1
    for crea in creatures:
        lastgen[crea.gene]['speed'][int(crea.speed)] += 1
        lastgen[crea.gene]['sight'][int(crea.sight)] += 1
n_creaz_all['A'] = [n_creaz_all['A'][g]/number_of_simulations for g in range(n_gen)]
n_creaz_all['B'] = [n_creaz_all['B'][g]/number_of_simulations for g in range(n_gen)]

for i in range(n_gen):
    if n_creaz_all['A'][i] == 0 and n_creaz_all['B'][i] == 0:
        n_gen = i
        break
```

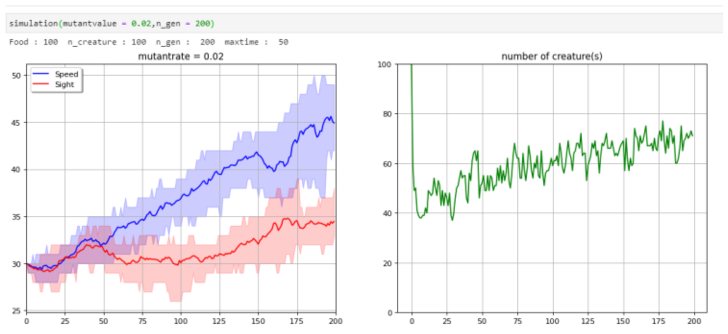
```
fig, ((ax1,ax2),(ax3,ax4)) = plt.subplots(2,2,figsize=(19,6), dpi=80)
ax1.set_xlim([0,n_gen-2])
ax1.set_ylim([0,max([winny['A'][i] + winny['B'][i] for i in range(n_gen-1)])+1])
ax2.set_xlim([0,n_gen-2])
ax2.set_ylim([0,max([n_creaz_all['A'][i] + n_creaz_all['B'][i] for i in range(n_gen-1)])])
#line1, = ax1.bar(np.arange(0, n_gen-1, 1),[winny['A'][i] + winny['B'][i] for i in range(n_gen-1)],color='#FF6E5A')
line1 = ax1.bar(np.arange(0, n_gen-1, 1), [winny['A'][i] + winny['B'][i] for i in range(n_gen-1)], 0.6, color='#FF6E5A')
line2 = ax1.bar(np.arange(0, n_gen-1, 1),winny['B'][:n_gen-1], 0.6,color='#90DCFF')
line5, = ax2.plot(np.arange(0, n_gen-1, 1),[n_creaz_all['A'][i] + n_creaz_all['B'][i] for i in range(n_gen-1)],color='#FF6E5A')
line6, = ax2.plot(np.arange(0, n_gen-1, 1),n_creaz_all['B'][:n_gen-1],color='#90DCFF')
ax2.fill_between(np.arange(0, n_gen-1, 1), [n_creaz_all['A'][i] + n_creaz_all['B'][i] for i in range(n_gen-1)],n_creaz_all['B'][:n_gen-1], alpha=0.2,color='#FF6E5A')
ax2.fill_between(np.arange(0, n_gen-1, 1), n_creaz_all['B'][:n_gen-1],0, alpha=0.2,color='#90DCFF')
ga = 'Gene ('+ str(mutantvalueA) +' : '+str(AGE['A'])+' days )'
gb = 'Gene ('+ str(mutantvalueB) +' : '+str(AGE['B'])+' days )'
ax1.set_title('Generation where each Gene type will dominate')
ax1.legend((line1,line2),(ga,gb),loc='upper left',shadow = True)
ax2.set_title('Average number of creatures across the simulations')
ax2.legend((line5, line6), (ga,gb), loc='upper left', shadow=True)
#print([lastgen['A']['speed'][i] + lastgen['B']['speed'][i] for i in range(maxspeed)])ax1.set_title('Generation where each Gene type will dominate')
ax3.set_title('Speed Distribution')
ax4.set_title('Sight Distribution')

line7 = ax3.bar(np.arange(0, maxspeed, 1),[lastgen['A']['speed'][i] + lastgen['B']['speed'][i] for i in range(maxspeed)] , 0.5, color='#FF6E5A')
line8 = ax4.bar(np.arange(0, maxsight, 1), [lastgen['A']['sight'][i] + lastgen['B']['sight'][i] for i in range(maxspeed)], 0.5, color='#FF6E5A')
line9 = ax3.bar(np.arange(0, maxspeed, 1), [lastgen['B']['speed'][i] for i in range(maxspeed)], 0.5, color='#90DCFF')
line10 = ax4.bar(np.arange(0, maxspeed, 1), [lastgen['B']['sight'][i] for i in range(maxspeed)], 0.5, color='#90DCFF')
ax3.legend((line7, line9), (ga,gb), loc='upper left', shadow=True)
ax4.legend((line8, line10), (ga,gb), loc='upper left', shadow=True)
fig.tight_layout()
```

Results and Discussion

1st Experiment:

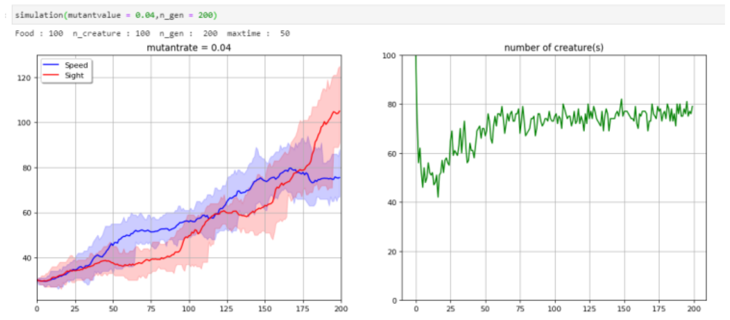
Find how much the difference mutant rate will affect in duration to reach the equilibrium point of the environment.



```
simulation(mutantvalue = 0.02,n_gen = 200)
```

```
Food : 100  n_creature : 100  n_gen :  200  maxtime :  50
```

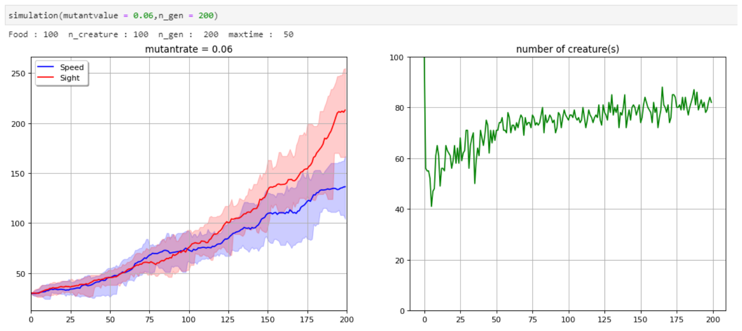
Figure 4 Mutant Rate = 0.02



```
simulation(mutantvalue = 0.04,n_gen = 200)
```

```
Food : 100  n_creature : 100  n_gen :  200  maxtime :  50
```

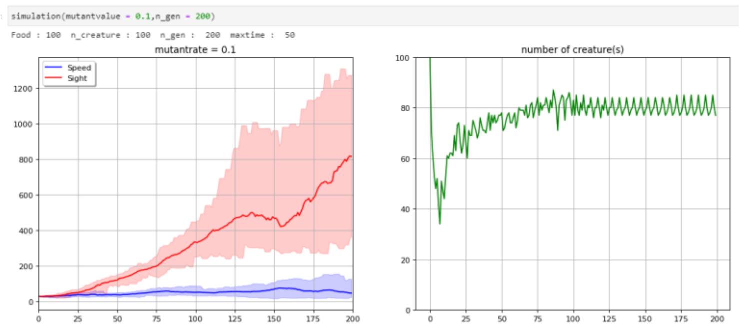
Figure 5 Mutant Rate = 0.04



```
simulation(mutantvalue = 0.06,n_gen = 200)
```

```
Food : 100  n_creature : 100  n_gen :  200  maxtime :  50
```

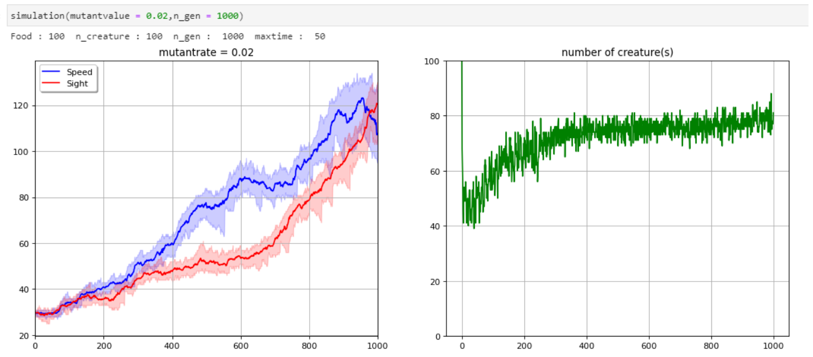
Figure 6 Mutant Rate = 0.06



```
simulation(mutantvalue = 0.1,n_gen = 200)
Food : 100  n_creature : 100  n_gen :  200  maxtime :  50
```

Figure 7 Mutant Rate = 0.1

It seems like 80 will be an equilibrium point for this environment but because of the low mutation rate in Figure 4 so what if we give it enough time will they come to an equilibrium point?



```
simulation(mutantvalue = 0.02,n_gen = 1000)
Food : 100  n_creature : 100  n_gen :  1000  maxtime :  50
```

Figure 8, After 1000 generations of 0.02 mutation rate, It eventually reaches an equilibrium of this environment.

The result is true If there is only one type of creature that lives in the environment. They will eventually reach the equilibrium point of the environment, a number of generations will depend on the mutant rate.

Another main point in this experiment is the environment can reach its equilibrium even though the creatures' performance in speed and sight are different across the mutation rate. It can be assumed that when they only need to compete with other creatures that have the same mutation rate and can be survived in the environment because there are no other creatures that can survive better to compete with them.

2nd Experiment :

This experiment will focus on whether two types of creatures that have different mutation rates will have a better chance of surviving in the environment.

```
simulation_gene_comparison_nsim(mutantvalueA = 0.04,mutantvalueB = 0.06,AGE = {'A':5,'B':5},number_of_simulations = 50,n_gen = 40)
```

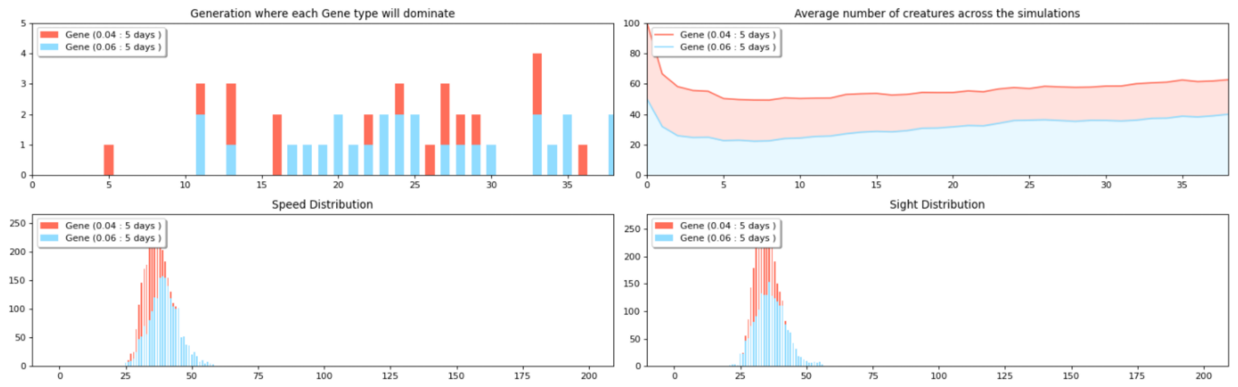


Figure 9, With creatures that have mutation rates of 0.04 and 0.06

```
simulation_gene_comparison_nsim(mutantvalueA = 0.04,mutantvalueB = 0.08,AGE = {'A':5,'B':5},number_of_simulations = 50,n_gen = 40)
```

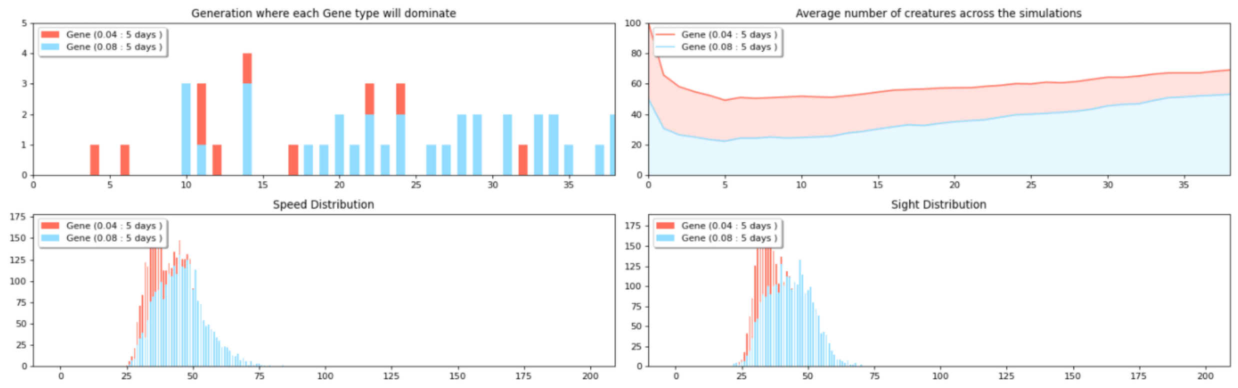


Figure 10, With creatures that have mutation rates of 0.04 and 0.08

```
simulation_gene_comparison_nsim(mutantvalueA = 0.04,mutantvalueB = 0.1,AGE = {'A':5,'B':5},number_of_simulations = 50,n_gen = 40)
```

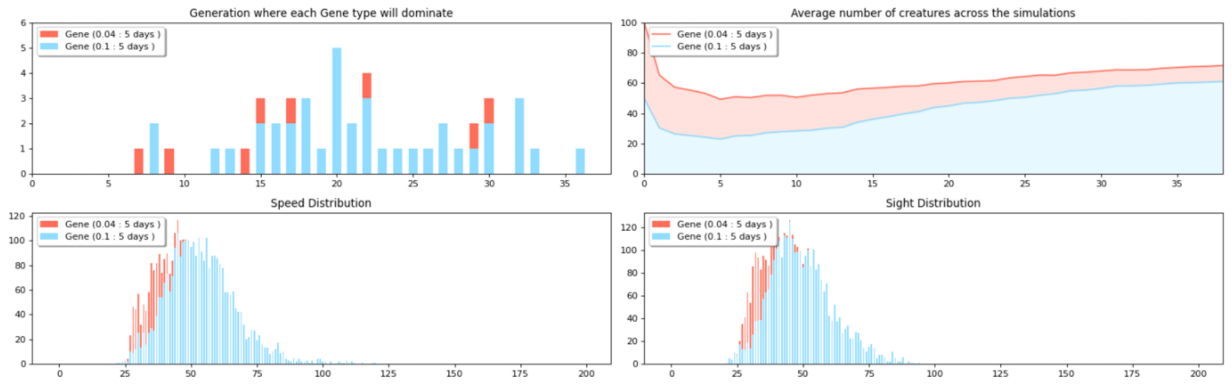
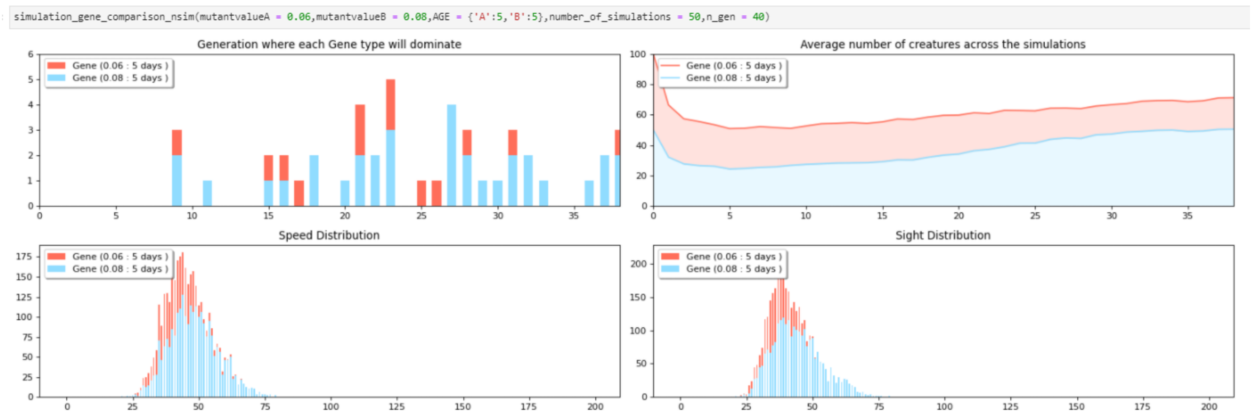
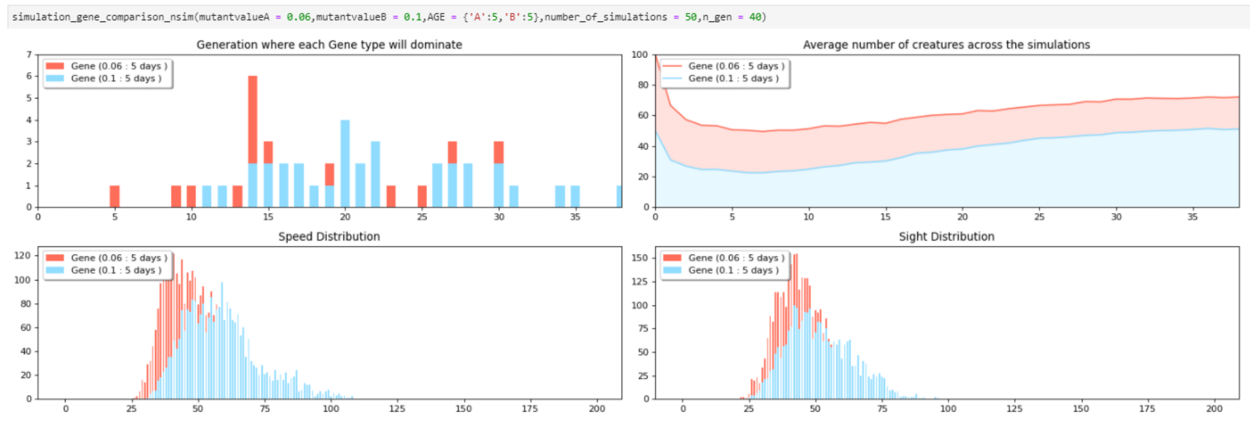


Figure 11, With creatures that have mutation rates of 0.04 and 0.1



```
simulation_gene_comparison_nsim(mutantvalueA = 0.06,mutantvalueB = 0.08,AGE = {'A':5,'B':5},number_of_simulations = 50,n_gen = 40)
```

Figure 12, With creatures that have mutation rates of 0.06 and 0.08



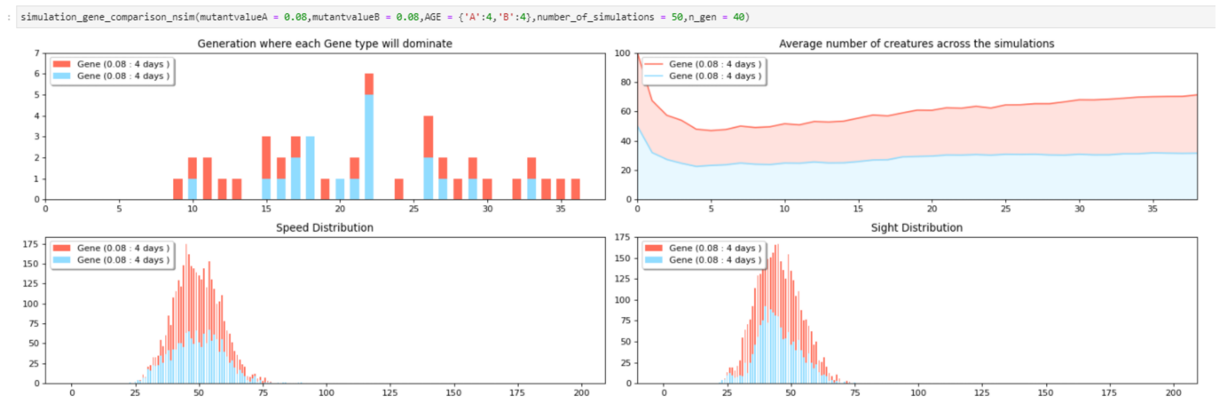
```
simulation_gene_comparison_nsim(mutantvalueA = 0.06,mutantvalueB = 0.1,AGE = {'A':5,'B':5},number_of_simulations = 50,n_gen = 40)
```

Figure 13, With creatures that have mutation rates of 0.06 and 0.1

The result shows that creatures with a higher mutation rate will have a higher chance of surviving in this environment. The chance of dominating the environment will depend on the gap between creatures' mutation rate which also means that to survive in this environment, every creature needs to produce a child that has better traits to be able to survive, and even if the lower mutation rate creatures can survive, they also have relatively high phenotypic traits because they need to outperform higher mutation creatures in order to dominate an environment.

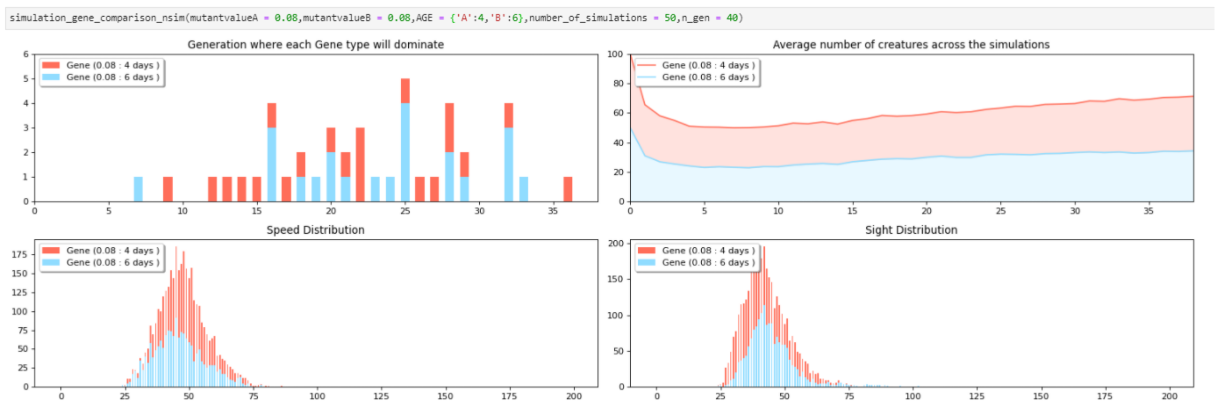
3rd Experiment:

This experiment will focus on 2 types of creatures that have different life span and find out the relation between life span and how its lives in the environment.



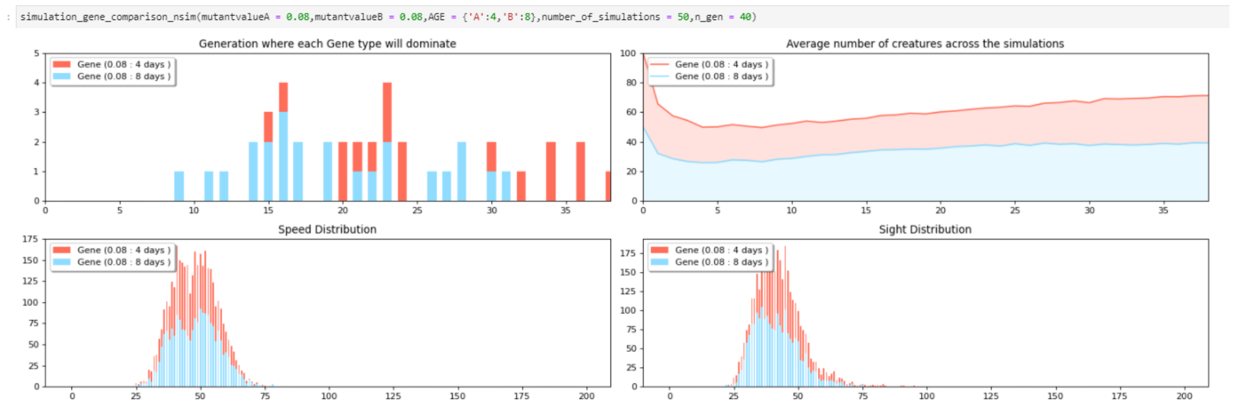
```
simulation_gene_comparison_nsim(mutantvalueA = 0.08,mutantvalueB = 0.08,AGE = {'A':4,'B':4},number_of_simulations = 50,n_gen = 40)
```

Figure 14, With creatures that have 4 generations of life-span and 4 generations of life-span.



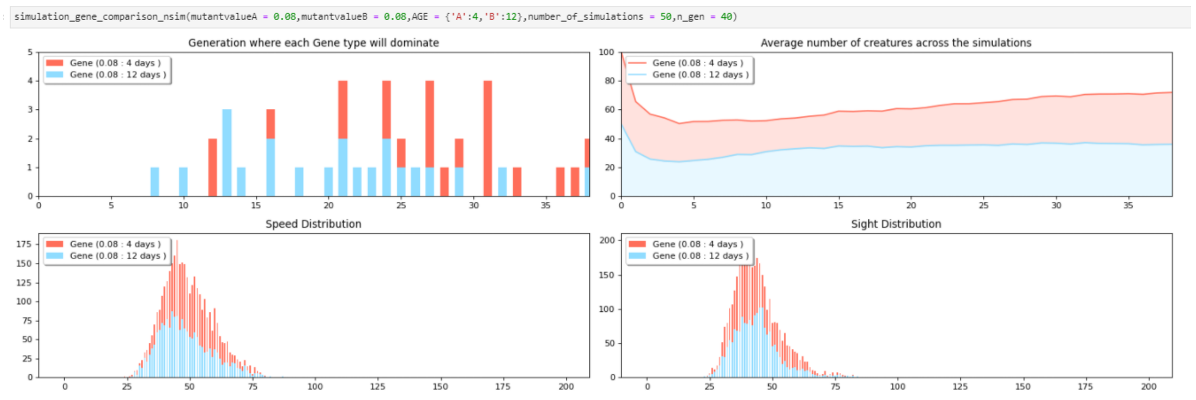
```
simulation_gene_comparison_nsim(mutantvalueA = 0.08,mutantvalueB = 0.08,AGE = {'A':4,'B':6},number_of_simulations = 50,n_gen = 40)
```

Figure 15, With creatures that have 4 generations of life-span and 6 generations of life-span.



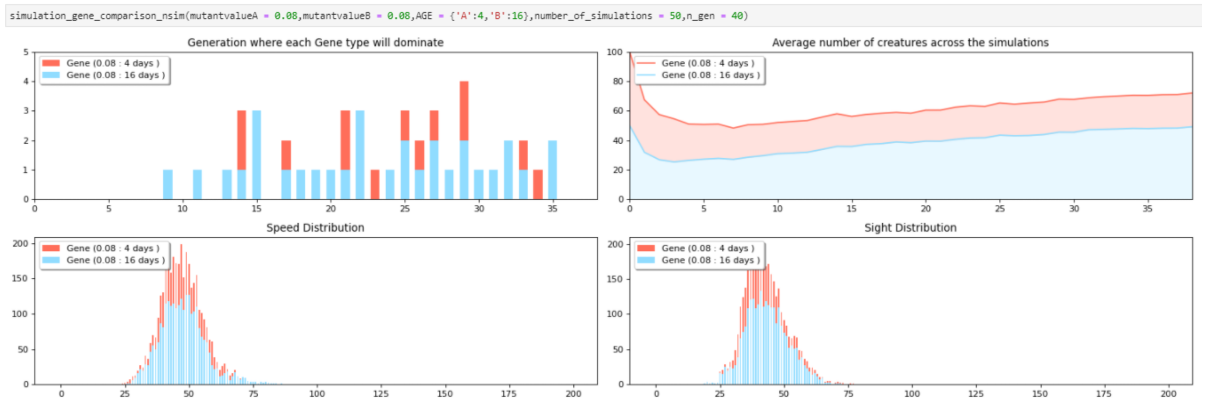
```
simulation_gene_comparison_nsim(mutantvalueA = 0.08,mutantvalueB = 0.08,AGE = {'A':4,'B':8},number_of_simulations = 50,n_gen = 40)
```

Figure 16, With creatures that have 4 generations of life-span and 8 generations of life-span.



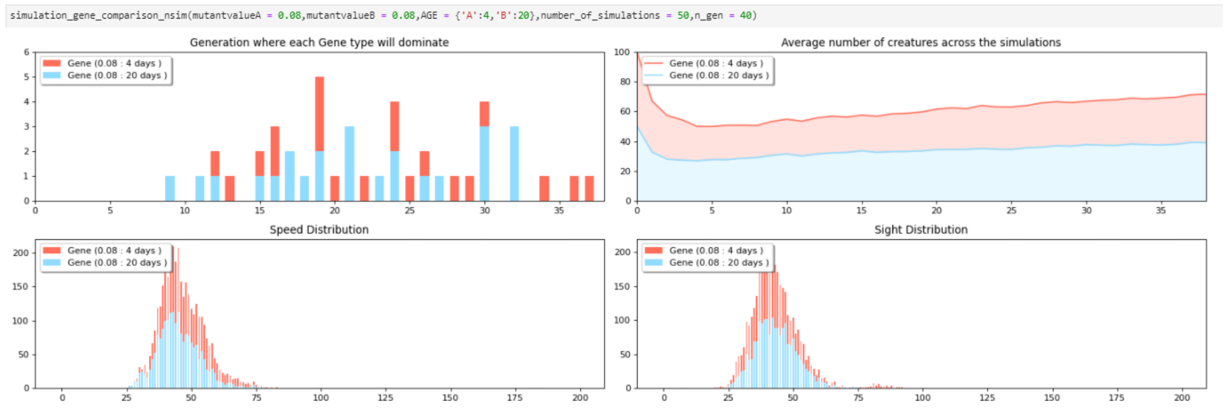
```
simulation_gene_comparison_nsim(mutantvalueA = 0.08,mutantvalueB = 0.08,AGE = {'A':4,'B':12},number_of_simulations = 50,n_gen = 40)
```

Figure 17, With creatures that have 4 generations of life-span and 12 generations of life-span.



```
simulation_gene_comparison_nsim(mutantvalueA = 0.08,mutantvalueB = 0.08,AGE = {'A':4,'B':16},number_of_simulations = 50,n_gen = 40)
```

Figure 18, With creatures that have 4 generations of life-span and 16 generations of life-span.



```
simulation_gene_comparison_nsim(mutantvalueA = 0.08,mutantvalueB = 0.08,AGE = {'A':4,'B':20},number_of_simulations = 50,n_gen = 40)
```

Figure 19, With creatures that have 4 generations of life-span and 20 generations of life-span.

Results of the experiments show that life-span has a co-relation with how creatures will perform when they have to compete with another type of creature. Longer lifespans help the creature to survive better because they only need to get one food in order to survive and have a longer chance of booking their spot in this environment until they are defunct. But it isn’t the only parameter that affects the environment because if other creatures make a new generation creature that performs better, other creatures that only have a longer lifespan but worse traits in comparison will eventually die.

Conclusion

In order to survive in this environment for creatures that have speed, sight, mutation rate, and lifespan as their phenotypic traits. There are several rules that can be assumed after the experiments.

1. In an environment that only has one type of creature, it will reach equilibrium after time unless if the first generation can survive. Period of taken by stabilizing state depends on mutation rate which will be faster if the mutation rate is higher.

2. Creatures with a higher mutation rate will have a better chance of dominating the environment because when creatures have to compete with other creatures, They need better phenotypic traits (speed and sight) in order to survive, so having a higher mutation rate means that they will have a better chance to get better in phenotypic traits and be able to survive.

3. Longer lifespan can contribute to a creature having a higher chance to survive in an environment but is not linearly dependent because a longer lifespan can help the creature to survive by only eating one piece of food but after time where other creatures have developed, they will eventually lose the competition and die.

With that being said, All of the phenotypic traits are affect how creatures survive in the environment as Charles Darwin said in Natural Selection rules.

Reference

Youtube.com. 2018. *Simulating Natural Selection*. [online] Available at: https://www.youtube.com/watch?v=0ZGbIKd0XrM

Youtube.com. 2018. *Simulating Competition and Logistic Growth*. [online] Available at: https://www.youtube.com/watch?v=uRTtlpD_U54

Youtube.com. 2018. *Mutations and the First Replicators*. [online] Available at: https://www.youtube.com/watch?v=UhSStR-FpQc

Youtube.com. 2018. *How life grows exponentially*. [online] Available at: https://www.youtube.com/watch?v=mMc_z1r96gs

Johnson, Clifford (1976). Introduction to Natural Selection. Baltimore, MD: University Park Press. ISBN 978-0-8391-0936-5. LCCN 76008175. OCLC 2091640.

Gould, Stephen Jay (2002). The Structure of Evolutionary Theory. Cambridge, MA: Belknap Press of Harvard University Press. ISBN 978-0-674-00613-3. LCCN 2001043556. OCLC 47869352.

Sammut-Bonnici, Tanya; Wensley, Robin (September 2002). "Darwinism, probability and complexity: Market-based organizational transformation and change explained through the theories of evolution" (PDF). International Journal of Management Reviews. 4 (3): 291–315. doi:10.1111/1468-2370.00088. ISSN 1460-8545.