# ARTIFICIAL EVOLUTION OF ANT

BY: KRITAGYA AND PRIYANK

# PROJECT OVERVIEW

Nature is full of miracles, just a little genetic shift can promote the inception of a new species. In this new era of artificial intelligence, we are keen to replicate it on artificial species to thrive in their artificial environment.

Ant is a small yet complex creature on earth, different species have different traits and they possess some of the most complex behaviour due to natural selection and survival of the fittest.

In our artificial earth, we want to produce complex behaviour in ants with the help of artificial intelligence.

Algorithms used for artificial earth:

- Genetic Algorithm
- Artificial Neural Network

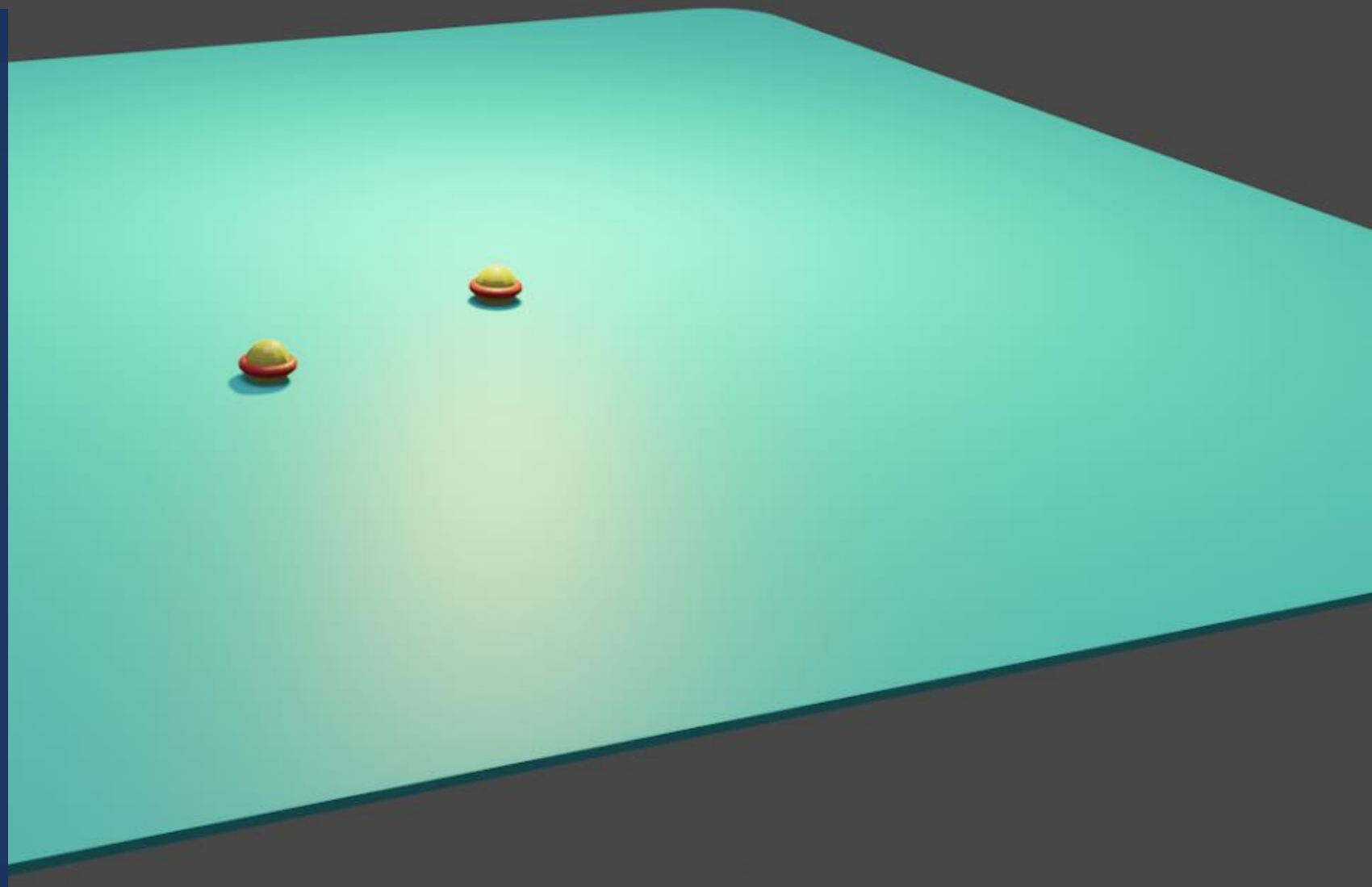
# PROJECT DESCRIPTION

**Single Ant Interaction and Evolution**

In our artificial environment, there is a single ant. The ant's motive is to eat food which is present in the environment.

# ENVIRONMENT

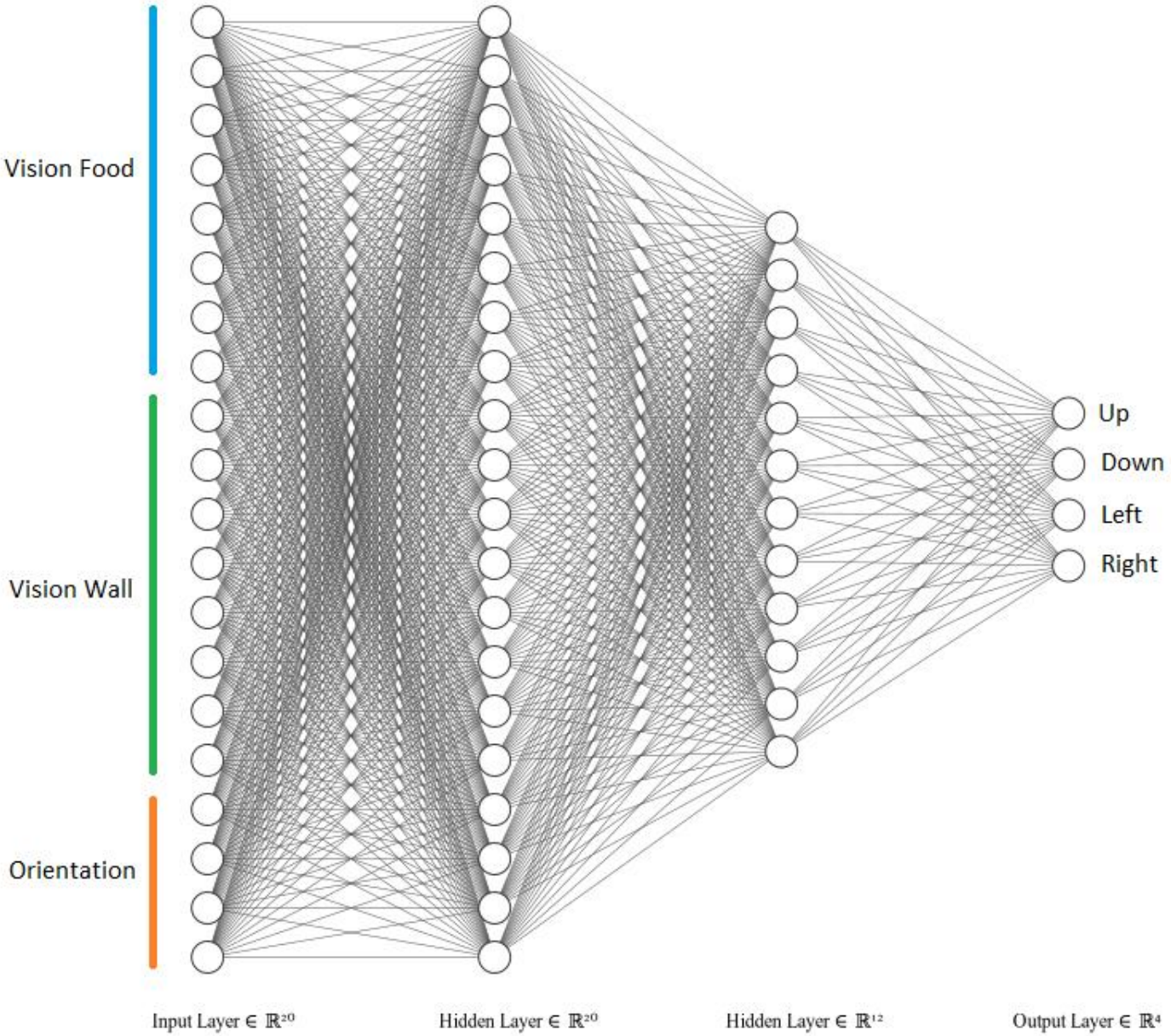
- The artificial environment consists of a grid of 20 x 20. The ant can't go out of the grid. The environment provides food on the grid, the ant has to eat the food to survive.

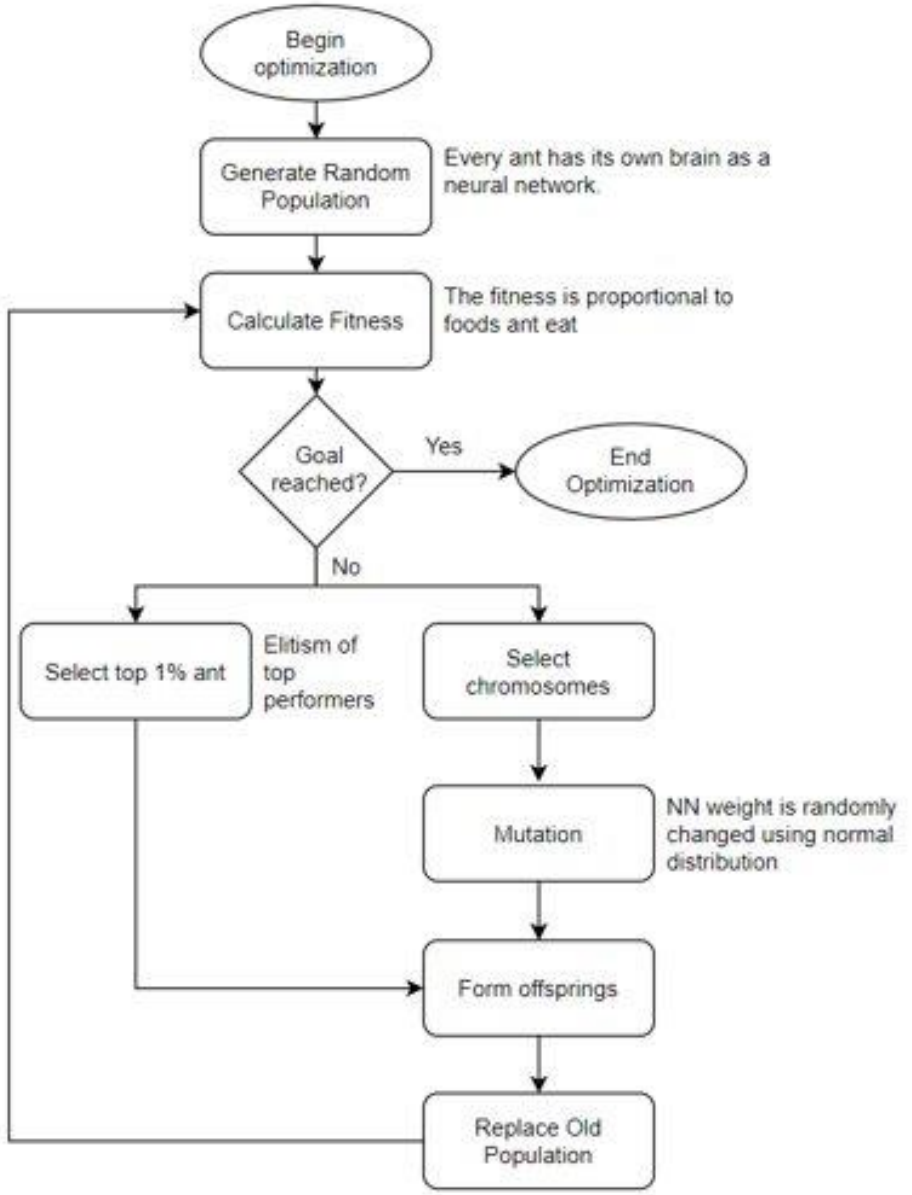


# ANT

- The ant resides in the grid in the environment. The ant can move in three directions - left, right and front. The ant can roam freely in the environment, the constraint for the ant is that if it goes out of board it will die.

# NEURAL NETWORK ARCHITECTURE




Begin optimization
Generate Random Population
Every ant has its own brain as a neural network.
Calculate Fitness
The fitness is proportional to foods ant eat
Goal reached?
Yes
End Optimization
No
Select top 1% ant
Elitism of top performers
Select chromosomes
Mutation
NN weight is randomly changed using normal distribution
Form offsprings
Replace Old Population

# FLOW CHART

# ALGORITHM

Initialize 100 ants having brain as Neural Network.(Each Neural Network has two hidden layer with 20 Neurons in first layer and 12 neurons in second layer.

Put the ant in a grid in the environment, let it move according to the decisions made by neural network(brai n)

Calculate fitness according to the food eaten by it.

If Goal reached then End the optimization Else goto 5.

Select top one per cent ant and add to new population array.

Select chromosomes randomly from top 50 per cent parent then mutate it using normal distribution.

Merge the array in 5 and 6 to make a new population array.

Repeat from 2 to 7.

Ant.py > Ant

```python
class Ant:
    def __init__  (self, x, y):
        self.x = x
        self.y = y
        self.foods = set() # generate food
        self.grid = []
        self.num_moves = 0
        self.allowed_moves = 100
        self.food_storage = copy.deepcopy(food_storage)
        self.mat = np.zeros((GRID_HEIGHT, GRID_WIDTH))
        self.num_food = 0
        self.direction = 'u'
        self.prev_min = 40
        for i in range(GRID_HEIGHT):
            tmp = []
            for j in range(GRID_WIDTH):
                tmp.append(Tile(i, j))
            self.grid.append(tmp)

        for i in range(1):
            self.addFood()

    def canEat(self, x,y):···

    def eat(self, x, y):···

    def addFood(self):···

    def isBackMove(self, prev_direction):···

    def move(self, output):···
```

# ANT CLASS

Ant.py > ...

```python
class Food:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def draw(self, WIN):
        food = pygame.Surface((TILE_SIZE, TILE_SIZE))
        food.fill((255, 0, 0))
        WIN.blit(food, (self.x*TILE_SIZE, self.y*TILE_SIZE))

class Grid:
    def __init__(self):
        self.grid = np.zeros((GRID_WIDTH, GRID_HEIGHT))

class Tile:
    def __init__(self, x, y, ant = None, food = None):
        self.x = x
        self.y = y
        self.food = food
        self.ant = ant

```

# ENVIRONMENT COMPONENTS CLASSES

```python
from typing import List, Callable, NewType, Optional
import numpy as np


ActivationFunction = NewType('ActivationFunction', Callable[[np.ndarray], np.ndarray])

# cell 1
class NeuralNetwork:
    def __init__(self, X, Y, n_h, params=None):
        self.input_nodes = layer_sizes(X, Y)[0]
        self.ouput_nodes = layer_sizes(X, Y)[2]
        self.hidden_nodes = n_h
        self.layer_nodes = [X.shape[0]]
        self.layer_nodes.extend(n_h)
        self.layer_nodes.append(Y.shape[0])
        self.params = {}
        self.initialize_parameters(self.input_nodes, self.hidden_nodes, self.ouput_nodes)

    def forward_propagation(self, X):···

    def initialize_parameters(self, n_x, n_h, n_y):···

def layer_sizes(X, Y):···

relu = ActivationFunction(lambda X: np.maximum(0, X))
def softmax(z):···

def sigmoid(x):···

def MinMaxScaler(x, min, max):···
```

# NEURAL NETWORK IMPLEMENTATION

GA.py > ...

```python
# GENETIC ALGORTITHM IMPLEMENTATION
population = []
generationCount = 0
popRanked = {}
def GA(X, Y, n_h, main, generations=10, popSize=100, eliteSize=10, mutationRate=0.05):

    def initial_population(popSize):···

    def mutation(child, mutationRate):···

    def rankPopulation():···

    def random_pick():···

    def next_generation(eliteSize, mutationRate):···

    def genetic_algorithm(popSize, eliteSize, mutationRate, generations):
        global population, generationCount, popRanked
        generationCount = 0
        population = initial_population(popSize)
        best_fitness = -1e9
        best_pop = []

        for i in range(generations):
            generationCount += 1
            rankPopulation()
            fitness = popRanked[0][1]
            if best_fitness < fitness:
                best_fitness = fitness
                best_pop = copy.deepcopy(population[popRanked[0][0]])
            print("Generation : {}\t Fitness: {}".format(str(i+1), str(fitness)))

            population = next_generation(eliteSize, mutationRate)
            if (i+1)%1==0:
                with open('weights.pickle', 'wb') as handle:
                    pickle.dump(best_pop, handle, protocol=pickle.HIGHEST_PROTOCOL)
        return best_pop
    return genetic_algorithm(popSize, eliteSize, mutationRate, generations)

```

# GENETIC ALGORITHM IMPLEMENTATION

# DEMONSTRATION

01 Generation 0

02 Generation 10

03 Generation 50

04 Generation 100

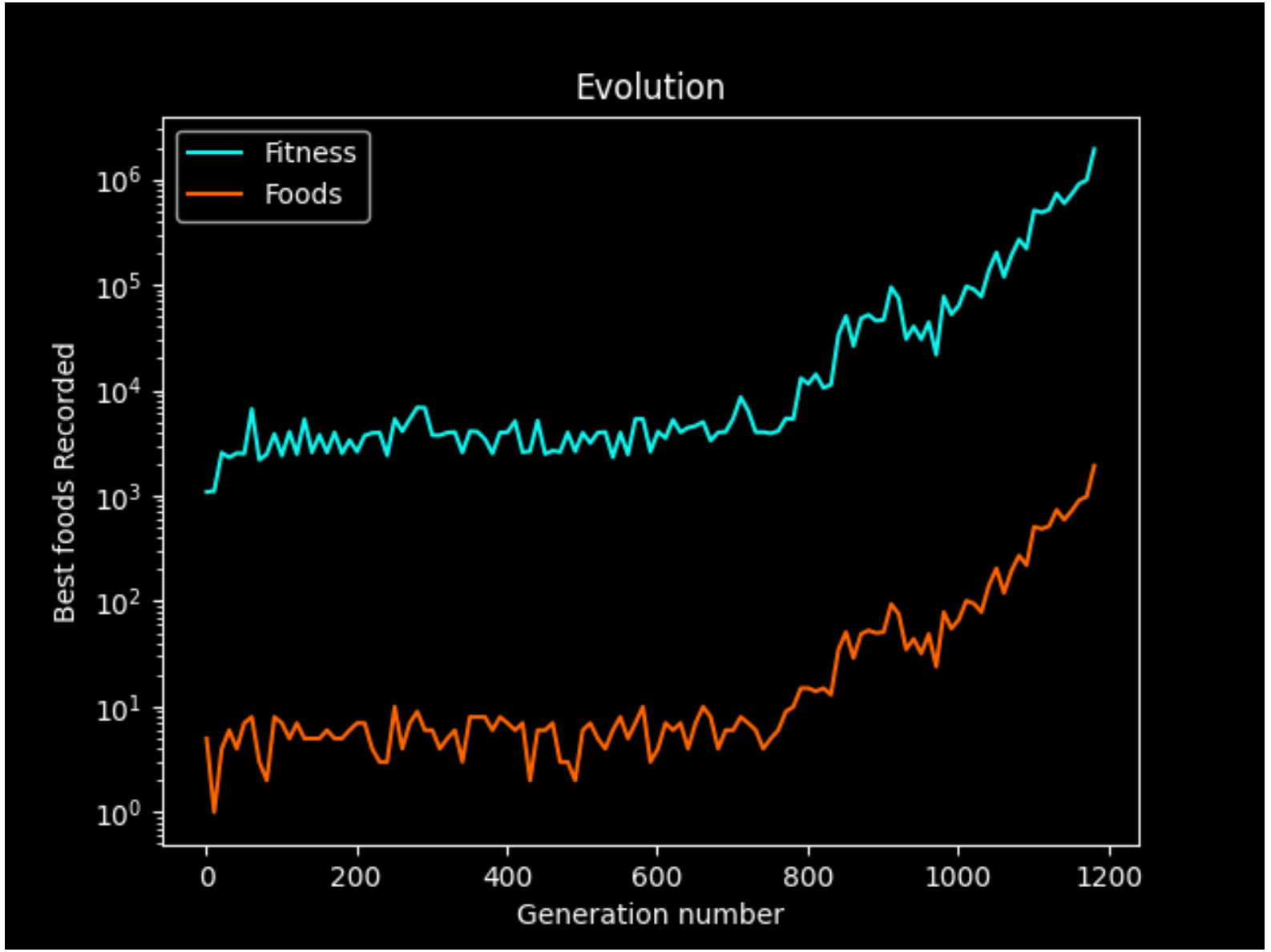
05 Generation 500

06 Generation 800

07 Generation 1190

# RESULT AND ANALYSIS

- In our experimental investigations, we have successfully trained the brain (ANN) of ant to get the highest fitness value as 45592083. The maximum food eaten by the ant is 45596.
- Our trial is based on running the genetic algorithm to 1190 generations after which we have got the best fitness value.



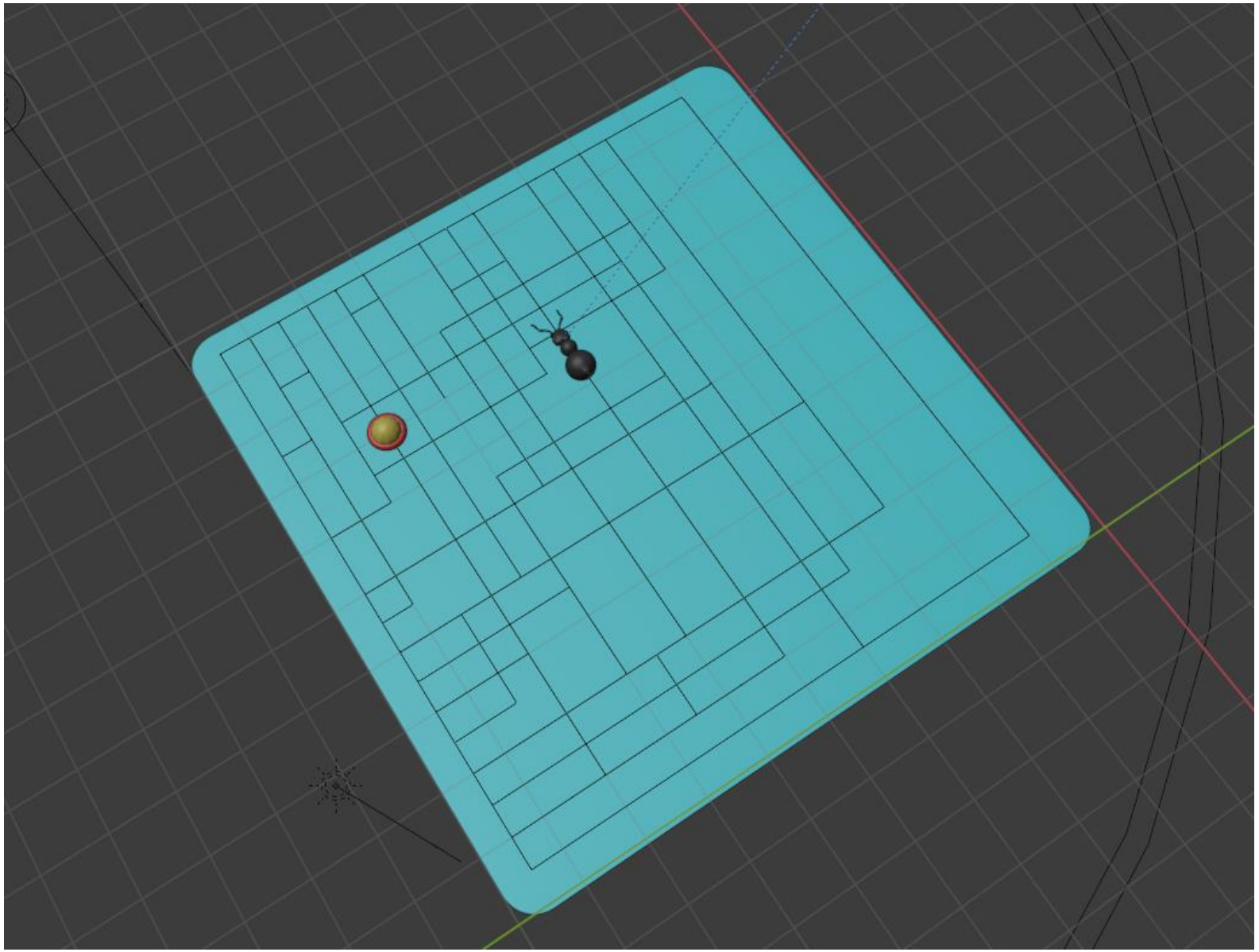
# LIMITATIONS

Geometry

More complex Environment

Ants complex behavior

Slow training rate of Genetic Algorithm



# LIMITATION

# FUTURE WORK

In our model we have taken a single ant to learn the direction it needs to go to get the food, the next goal is to make an artificial environment of multiple ants to interact with each other and the environment to learn some complex behaviours like teamwork, self-organizing, ant colony building, dead reckoning. For training, we have used a genetic algorithm and it has slow learning due to its brute force nature, so for learning reinforcement learning can be used to optimize.

# THANK YOU