Flap(AI) Bird

Kirk Harlow
Keiland Cooper
Chris East

The Project Idea

Overview

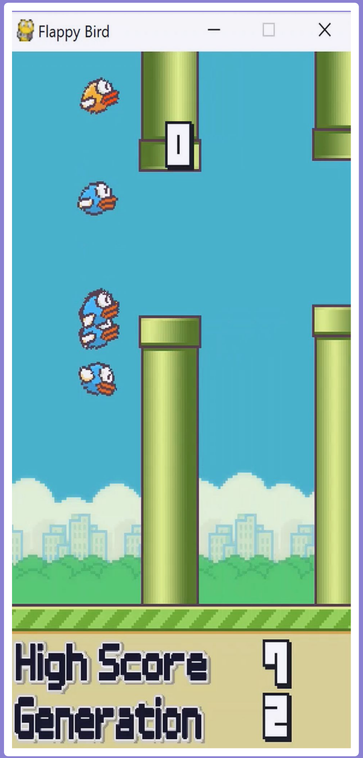
Convert the game Flappy Bird to use both Neural Networks and Genetic Algorithms to teach the Birds' AI to play.

Why Flappy Bird?

- The game is infamously known for it's difficulty for human players to achieve high scores
- The game has only two inputs: Horizontal Distance and Vertical Distance from the center of the pipe opening.
- The game only has one output: Jump or don't Jump
- The Pipes are randomly generated so there's no deterministic solution

The environment

AN OPEN-SOURCE* FLAPPY BIRD™ RENDITION IN PYTHON



System Environment

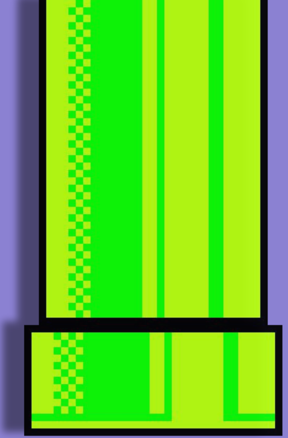
- Pygame (*SDL2* libraries).
- Sprite-sheet graphics modeling.

Runtime Environment

- Autonomous non-terminating simulations (*enabled*)
- User-friendly interface for real-time visualization of progress in successive generations.

User Environment

- User-input control is still available within the game environment.



*https://github.com/sourabhv/FlapPyBird

The Birds

```python
# The main bird class
class Bird:
    def __init__(self,initx, inity, index, key, initVelY, initAccY, initRot,net=0):
        n_inputs = 2
        n_outputs = 1

        self.x              = initx
        self.y              = inity
        self.key            = key
        self.index          = index
        self.width          = 0
        self.height         = 0
        self.moving         = True
        self.velY           = initVelY
        self.maxVelY        = 10   # max vel along Y, max descend speed
        self.minVelY        = -8   # min vel along Y, max ascend speed
        self.accY           = initAccY
        self.rot            = initRot
        self.velRot         = 3   # angular speed
        self.rotThr         = 20   # rotation threshold
        self.flapAcc        = -9   # players speed on flapping
        self.flapped        = True
        self.score          = 0
        self.distTraveled   = 0
        self.distFromOpen   = 0
        if net == 0:
            self.network    = Net(n_inputs, n_outputs)
        else:
            self.network    = net

    def calculate_fitness(self):
        return self.score + self.distTraveled - self.distFromOpen

    # Determines if the birds flaps, calling on it's neural net
    def flaps(self, inputX, inputY):
        flaps = self.network.propagate((inputX, inputY))[0]
        return flaps > .50
```

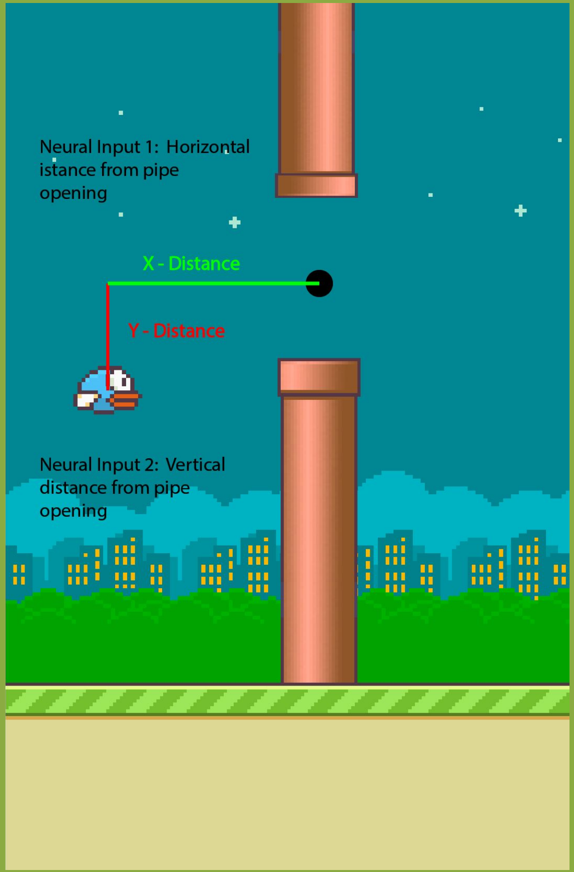
Bird( ) - each individual Bird is a self-contained instance that allows for easy data-collection + manipulation.

- A *Generation-of-Birds* consists of **10 birds** running concurrently - ending only when all birds have failed.
- Each Bird is assigned its *own* **neural network** for determining when to jump (*neural.py*).
- Each Bird is responsible for maintaining its progress, a value determined by the **fitness score** algorithm.

The Inputs

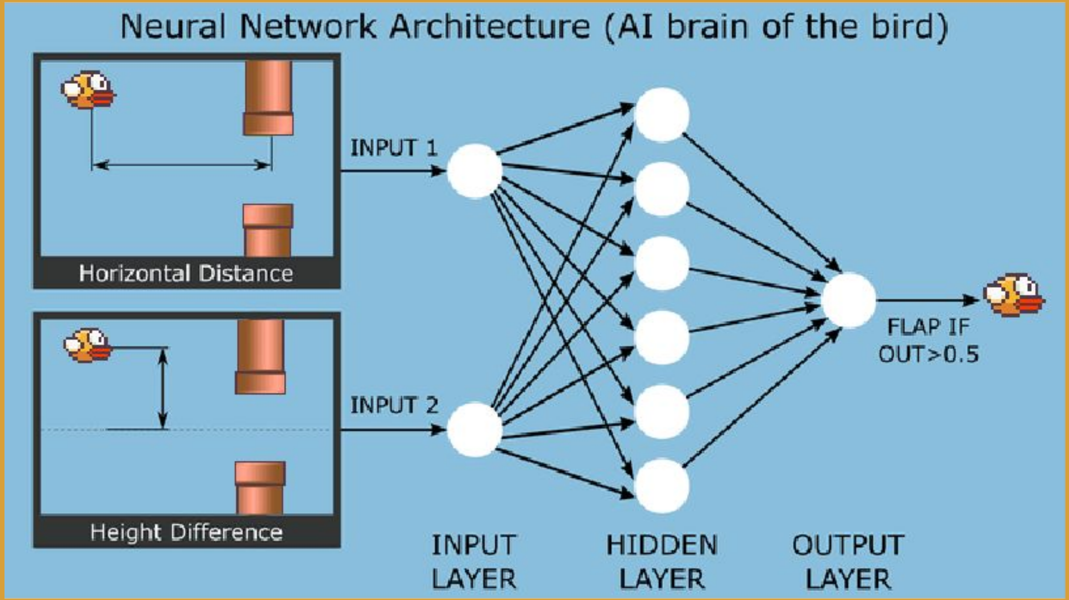
<u>Our Neural Net uses two inputs:</u>

- The X *(Horizontal)* Distance from the center of the pipe opening
- The Y *(Vertical)* Distance from the center of the pipe opening
- Calculated each tick/frame of the game
- These values are passed to the individual Bird’s neural network to decide whether or not to jump



How does the Neural Net Decide when to jump?

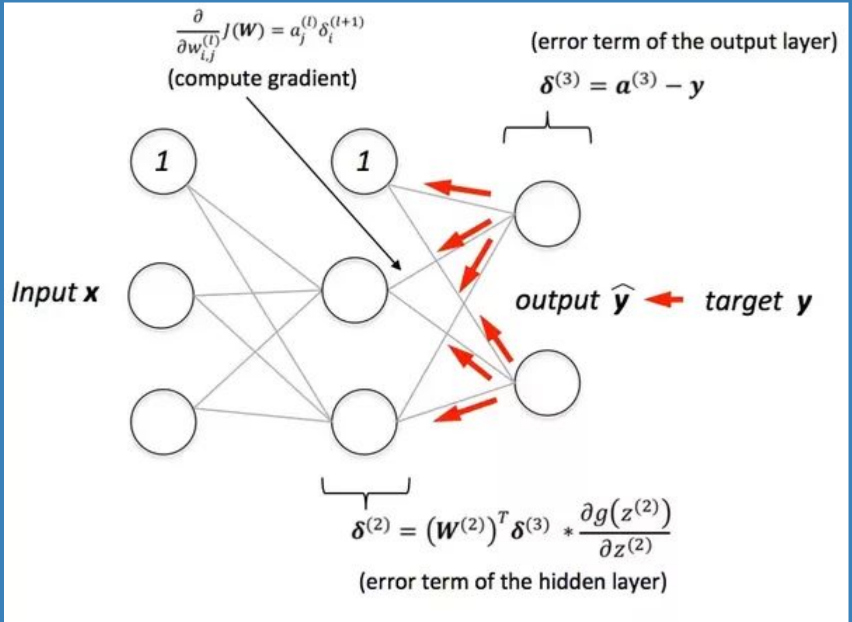
- The inputs are fed to a *Feed Forward Neural Network*
- Which propagates the inputs along the network, using Sigmoid Activation Function to reduce to one output
- If the output > 0.5 the bird "flaps" or jumps once



How to get a machine to learn?

Traditional Method: *Back Propagation*

- Neural Networks use a concept known as Back Propagation to propagate an Error function backwards updating the weights
- Issues with this approach is the Error Function. How do we determine how "off" the neural net was in its decision to jump

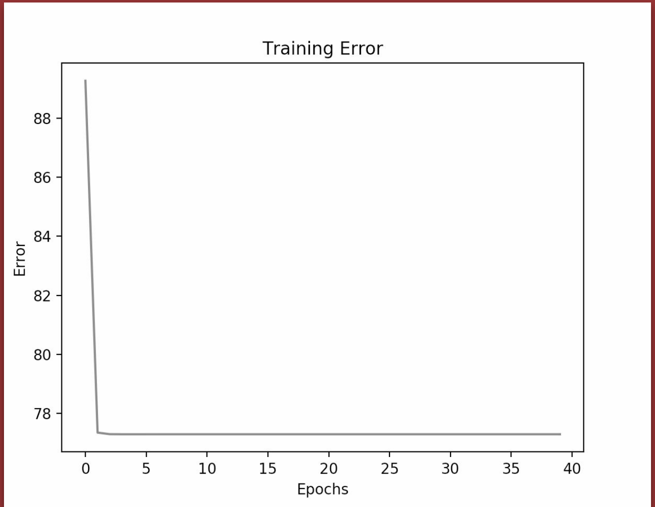


How to get a machine to learn?

Traditional Method: *Back Propagation - Implemented*

SUPERVISED NEURAL NETWORK ALGORITHM

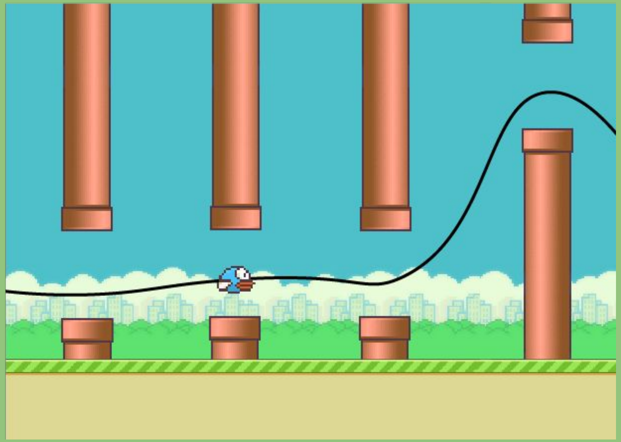
- Built neural network from scratch (only numpy used)
- Trained on data captured from the game, over 25k dp
- Had high error rate (70%) , even with uniform pipes
- Proof that it was not the most viable option



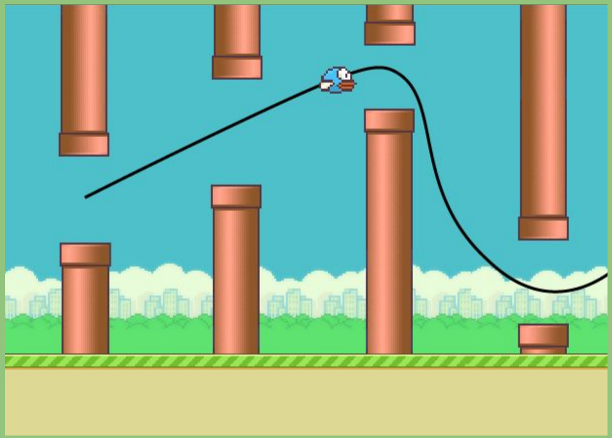
Pipe Patterns Difficult for AI

- Due to the random nature of the pipes several different patterns emerge. These are examples of pipe patterns that often cause Birds to crash.
- The goal is to evolve a neural network that can handle ANY pattern

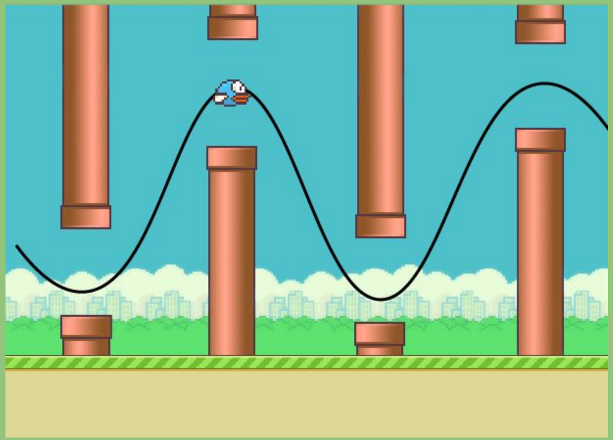
Flat Pop



Ascending Drop



Jig - Saw



How to get a machine to learn?

Our Method: *Genetic Algorithm*

Genetic Algorithms:

They have two main parts:

- Fitness Function
- Generating new Genomes

The algorithm selects the best birds and passes their "genetic" material onto future generations. Thus, the neural networks are modified and passed on, NOT trained!

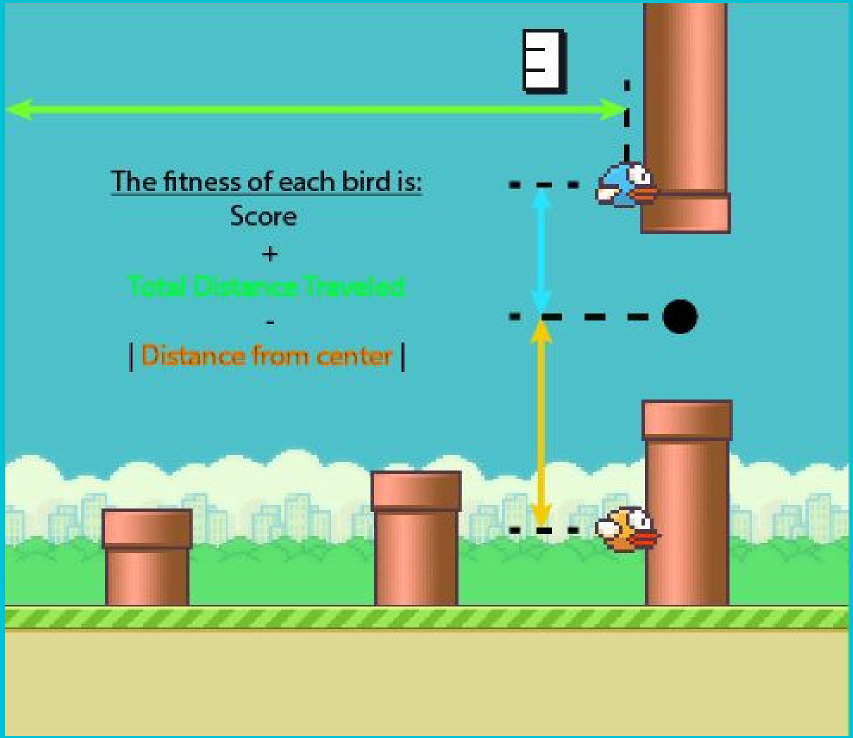
Initially, the birds are given random weights and bias vectors. This is the "Genome"

```python
def generateBirds(birds, fitness,  FIRST, initx, inity, birdIndex, initVelY,initAccY,initRot):
    if FIRST:
        for i in range(NUMBERBIRDS):
            yplus = 0 # random.randint(5,15)
            birds["Bird " + str(i)] = Bird(initx, inity + yplus, birdIndex, "Bird " + str(i), initVelY, initAccY,
                                           initRot, 0)
    else:
        newGeneration = {}
        winners = fitness[-4:]
        winners = winners[::-1]
        first = birds[winners[0][0]]
        second = birds[winners[1][0]]
        third = birds[winners[2][0]]
        fourth = birds[winners[3][0]]
        selection = [first, second, third, fourth]
        numInputs = first.network.inputs
        numOutputs = first.network.outputs
        newGeneration['Bird 0'] = Bird(initx, inity, first.index, "Bird " + str(0), initVelY, initAccY,
                                       initRot, Net(numInputs, numOutputs, deepcopy(first.network.network),
                                                    deepcopy(first.network.bias)))
        newGeneration['Bird 1'] = Bird(initx, inity, second.index, "Bird " + str(1), initVelY, initAccY,
                                       initRot, mutation(Net(numInputs, numOutputs, deepcopy(first.network.network),
                                                    deepcopy(first.network.bias),.05)))
        newGeneration['Bird 2'] = Bird(initx, inity, third.index, "Bird " + str(2), initVelY, initAccY,
                                       initRot, mutation(Net(numInputs, numOutputs, deepcopy(first.network.network),
                                                    deepcopy(first.network.bias)), .10))
        newGeneration['Bird 3'] = Bird(initx, inity, fourth.index, "Bird " + str(3), initVelY, initAccY,
                                       initRot, mutation(Net(numInputs, numOutputs, deepcopy(first.network.network),
                                                    deepcopy(second.network.bias)), .15))
        newGeneration['Bird 4'] = Bird(initx, inity, fourth.index, "Bird " + str(4), initVelY, initAccY,
                                       initRot, mutation(Net(numInputs, numOutputs, deepcopy(first.network.network),
                                                    deepcopy(second.network.bias)), .15))
        newGeneration['Bird 5'] = Bird(initx, inity, fourth.index, "Bird " + str(5), initVelY, initAccY,
                                       initRot, mutation(Net(numInputs, numOutputs, deepcopy(first.network.network),
                                                    deepcopy(third.network.bias)), .1))
        newGeneration['Bird 6'] = Bird(initx, inity, first.index, "Bird " + str(6), initVelY, initAccY,
                                       initRot, crossOver(first, second))
        newGeneration['Bird 7'] = Bird(initx, inity, second.index, "Bird " + str(7), initVelY, initAccY,
                                       initRot, crossOver(second, third))
        birdOne = random.choice(selection)
```

Genetic Algorithm:
Fitness Function

<u>The Fitness Function:</u>

- Responsible for assigning a value to the bird for its “Fitness”. This allows for the birds to be sorted by fitness.
- The fitness formula we went with was:
  Score + Total Distance - Vertical Distance from the Center of the Pipe Opening
- In the example the Blue Bird is “fitter” than the Orange Bird despite traveling the same distance & score, because the Blue Bird was closer to the opening

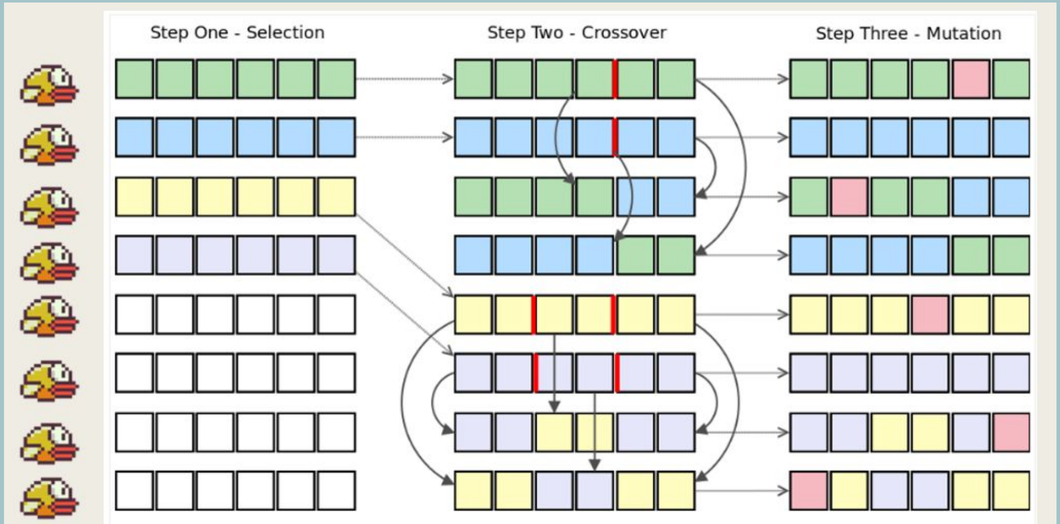


Genetic Algorithm:
Genome

The "Genome" of the birds is 2 dimensional array of weight vectors and bias vectors

<ins>The Genetic Algorithm:</ins>

- The first generation of birds have entirely random weights assigned to their individual neural nets
- Successive generations work by taking the weights from the neural networks of the top four fittest birds
- 1st and 2nd place are passed directly on to the new generation
- Birds 3-4 are modified versions of 1st and 2nd
- The rest randomly select 2 birds as parents and use a biological idea known as *Crossover* to create new genomes

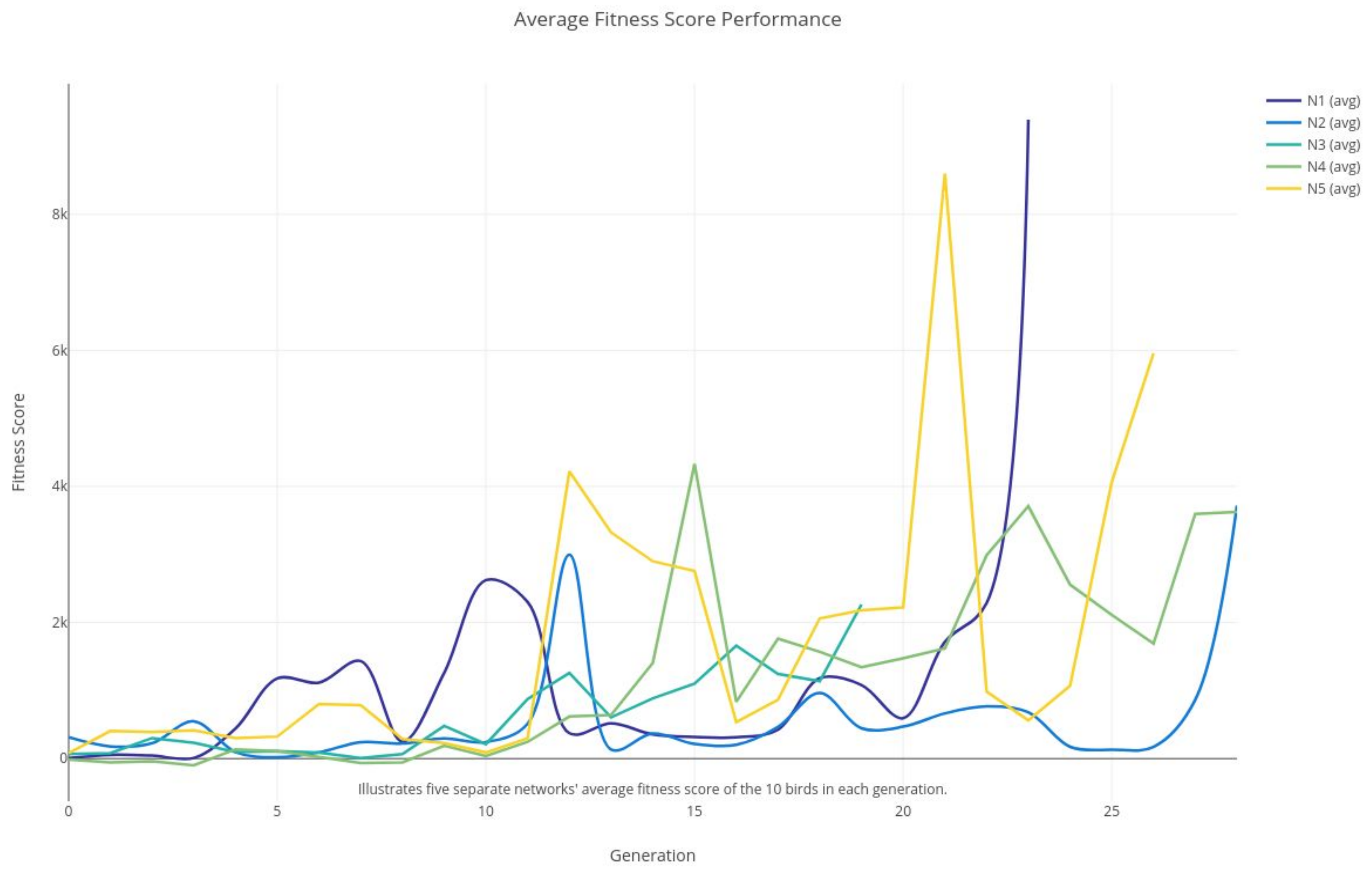


Genetic Algorithm:
Extinction

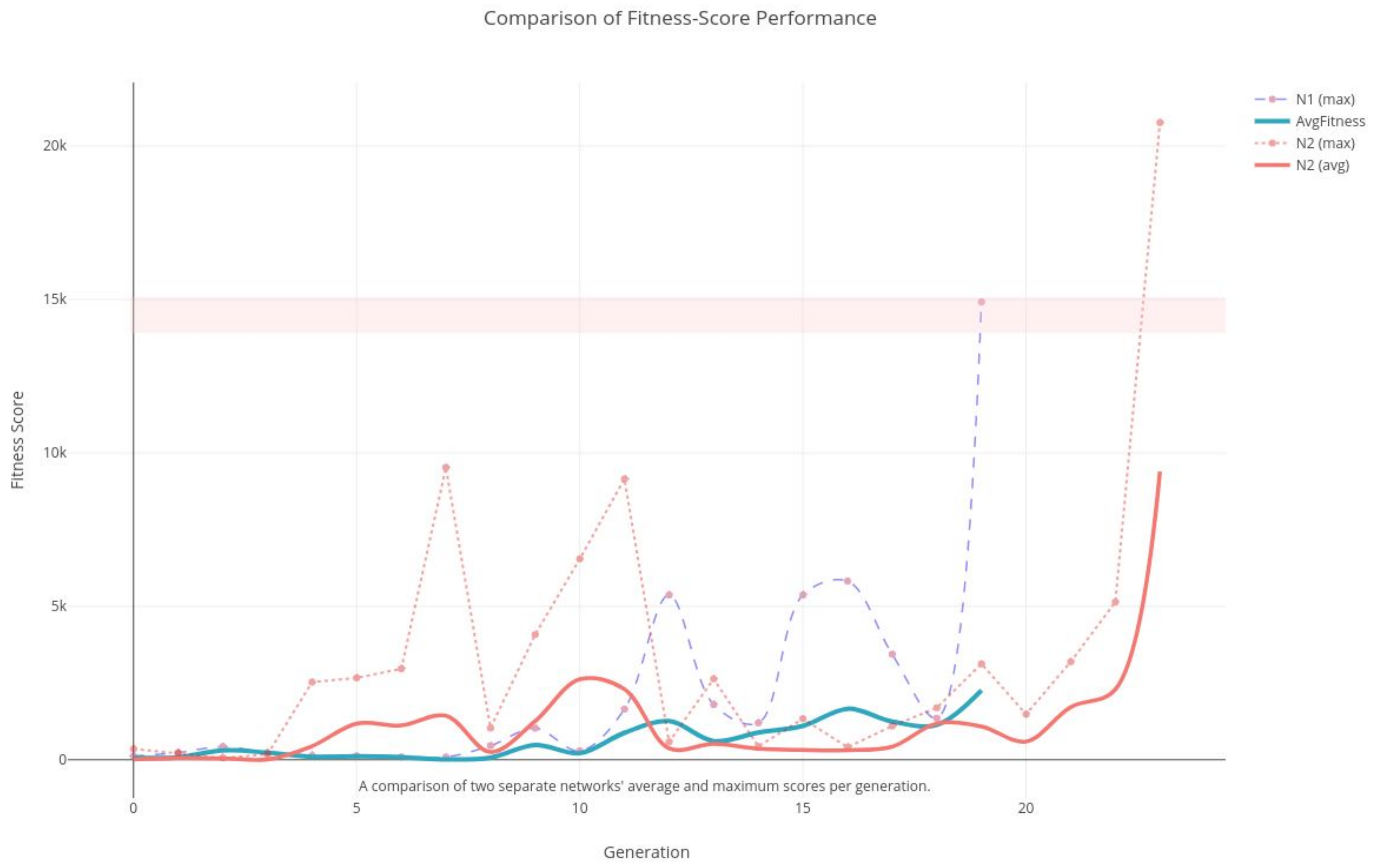
The "Extinction" of the birds is when none of the genetic information from the last generation is passed on to the next.

Extinction:

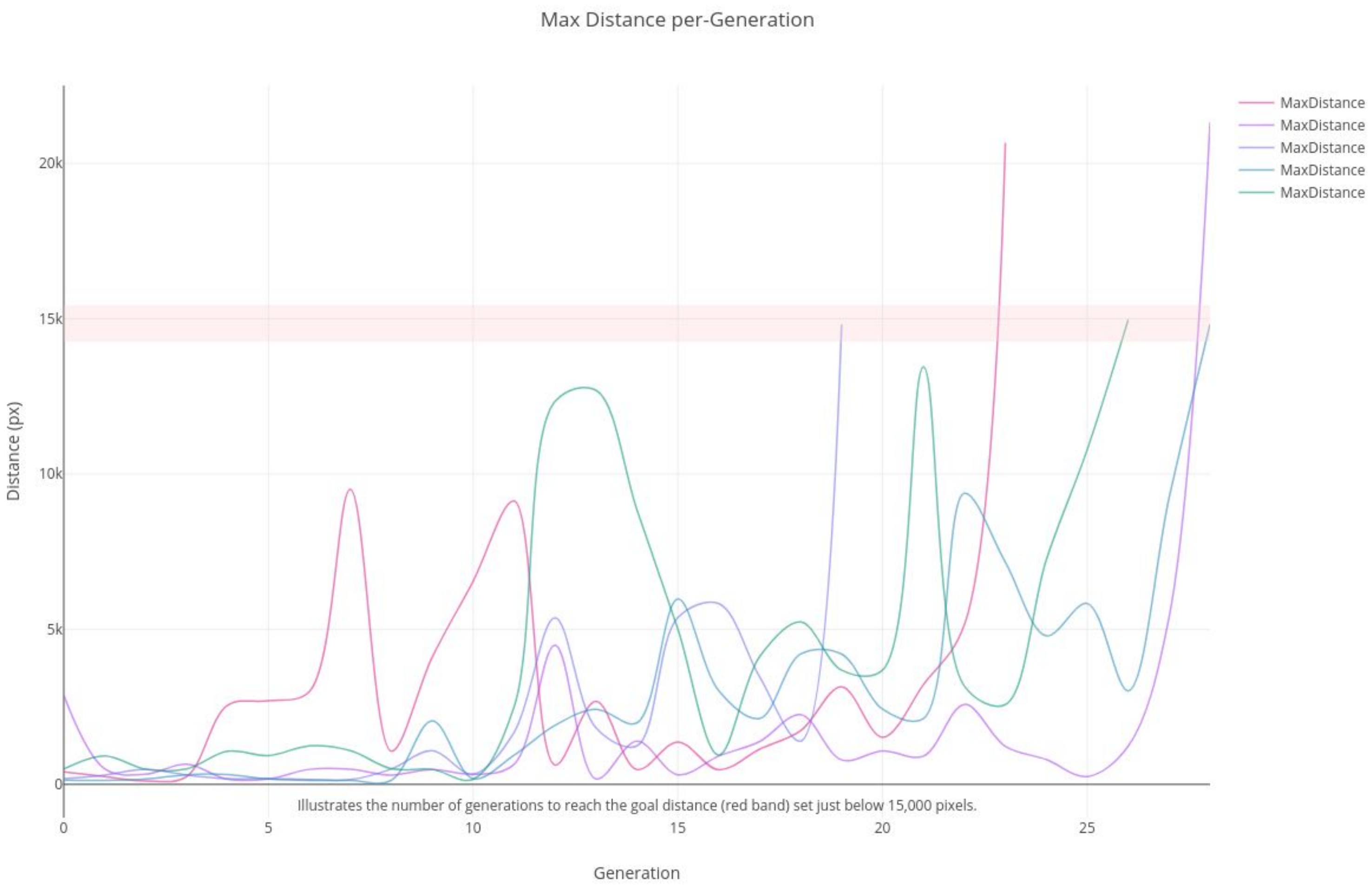
- Due to the random assignment of weights and bias values. The best birds can often be still far off from optimal solution
- To speed of the process if by generation 40 the high score isn't greater than 50. Then extinction occurs, and an entirely new set of birds is created



Illustrates five separate networks' average fitness score of the 10 birds in each generation.



A comparison of two separate networks' average and maximum scores per generation.

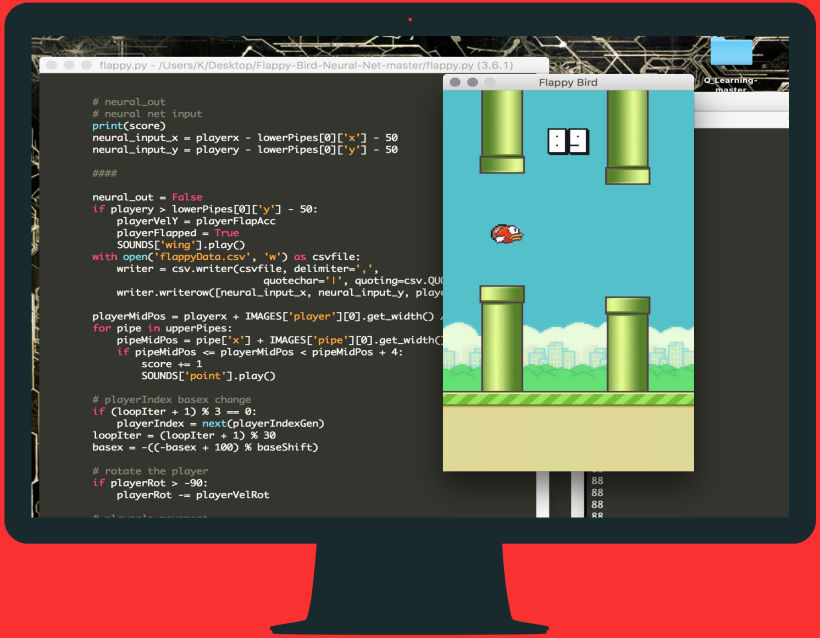

Max Distance per-Generation
MaxDistance
MaxDistance
MaxDistance
MaxDistance
MaxDistance
Distance (px)
20k
15k
10k
5k
0
0
5
10
15
20
25
Illustrates the number of generations to reach the goal distance (red band) set just below 15,000 pixels.
Generation

Beginning Stages:

0 Generations

Late Stages:

175 Generations


flappy.py - /Users/K/Desktop/Flappy-Bird-Neural-Net-master/flappy.py (3.6.1)
Flappy Bird
# neural_out
# neural net input
print(score)
neural_input_x = playerx - lowerPipes[0]['x'] - 50
neural_input_y = playery - lowerPipes[0]['y'] - 50
####
neural_out = False
if playery > lowerPipes[0]['y'] - 50:
playerVelY = playerFlapAcc
playerFlapped = True
SOUNDS['wing'].play()
with open('flappyData.csv', 'w') as csvfile:
writer = csv.writer(csvfile, delimiter=',',
playerMidPos = playerx + IMAGES['player'][0].get_width()
for pipe in upperPipes:
if pipeMidPos <= playerMidPos < pipeMidPos + 4:
score += 1
SOUNDS['point'].play()
# playerIndex basex change
if (loopIter + 1) % 3 == 0:
playerIndex = next(playerIndexGen)
loopIter = (loopIter + 1) % 30
basex = -((-basex + 100) % baseShift)
# rotate the player
if playerRot > -90:
playerRot -= playerVelRot

questions?