FLAPPY BIRD USING NEAT ALGORITHM

PROJECT REPORT

ARTIFICIAL INTELLIGENCE

(SWE4010)

CAL Course

in

M.Tech. – Software Engineering

by

RUTHVIK BHUPATHI: 16MIS0239

V. SAI NIKHIL: 16MIS0257

UNDER THE GUIDANCE OF

Prof. Chiranji Lal Chowdhary

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

Fall Semester 2019-20

INTRODUCTION:

In video games, artificial intelligence (AI) is used to generate responsive, adaptive or intelligent behaviours primarily in non-player characters (NPCs) similar to human-like intelligence. Artificial intelligence has been an integral part of video games since their inception in the 1950s. The role of AI in video games has expanded greatly since its introduction. Modern games often implement existing techniques from the field of artificial intelligence such as pathfinding and decision trees to guide the actions of NPCs. Additionally, AI is often used in mechanisms which are not immediately visible to the user, such as data mining and procedural-content generation.

The term "game AI" is used to refer to a broad set of algorithms that also include techniques from control theory, robotics, computer graphics and computer science in general, and so video game AI may often not constitute "true AI" in that such techniques do not necessarily facilitate computer learning or other standard criteria, only constituting "automated computation" or a predetermined and limited set of responses to a predetermined and limited set of inputs.

Flappy Bird is a mobile game developed by Vietnamese video game artist and programmer Dong Nguyen, under his game development company dotGears. The game is a side-scroller where the player controls a bird, attempting to fly between columns of green pipes without hitting them.

The game was released in May 2013 but Flappy Bird was removed from both the App Store and Google Play by its creator on February 10, 2014.

ABSTRACT:

NeuroEvolution of Augmenting Topologies (NEAT) is a genetic algorithm (GA) for the generation of evolving artificial neural networks. NeuroEvolution of Augmenting Topologies (NEAT) is a genetic algorithm (GA) for the generation of evolving artificial neural networks.

NEAT is an example of a topology and weight evolving artificial neural network (TWEANN) which attempts to simultaneously learn weight values and an appropriate topology for a neural network. The NEAT approach begins with a perceptron-like feed-forward network of only input neurons and output neurons. As evolution progresses through discrete steps, the complexity of the network's topology may grow, either by inserting a new neuron into a connection path, or by creating a new connection between (formerly unconnected) neurons.

Flappy Bird is a game that is very addictive it is played by tapping on the screen to fly through between the gap of the pipe, because the game is very addictive and the user tend to score low on their few first baby attempts they try again and again thus they become addictive. We are using NEAT (**NeuroEvolution of Augmenting Topologies**) algorithm for the AI gameplay of Flappy Bird. That would increase the chance of winning the game by a couple of cycles of learning and adapting its weights with every cycles in order to easily play the game and this data can be used for analysis that would be used to increase the gameplay of other related game as the gaming industry is a huge market with various number of games deployed every day.

LITERATRE REVIEW:

Paper-1

Problem Analysis:

Modern computer games are at the same time an attractive application domain and an interesting testbed for the evolutionary computation techniques. This paper is used to enhance the competitive non-player characters for a racing game. A competitive controller should have two basic skills: it should be able to drive fast and reliably on a wide range of tracks and it should be able to effectively overtake the opponents avoiding the collisions.

Proposed Algorithm

NeuroEvolution of Augmenting Topologies (NEAT)

Evaluation of proposed algorithm.

NEAT is specifically designed to evolve neural networks without assuming any a priori knowledge on the optimal topology nor on the type of connections (e.g., simple or recurrent connections). NEAT is based on three main ideas. First, in NEAT the evolutionary search starts from a network topology as simple as possible. Second, NEAT deals with the problem of recombining networks with different structures through an historical marking mechanism Third, NEAT protects the structural innovations through the mechanism of speciation. The competition to survive does not happen in the whole population but it is restricted to niches where all the networks have a similar topology. Therefore, when a new topology emerges, NEAT has enough time to optimize it.

In this section we briefly introduce the controllers submitted they are angle, curLapTime, damage, distFromStartLine, distracted, fuel, gear, lastLapTime, opponents, racePos, racePos, speedX, speedY, track, trackPos, wheelSpinVel.

Track: Vector of 19 range finder sensors: each sensors represents the distance between the track edge and the car. Sensors are oriented every 10 degrees from $-\pi/2$ and $+\pi/2$ in front of the car. Distances are in meters and sensors are limited to 100 meters. When the car is outside of the track (i.e., pos is less than -1 or greater than 1), these values are not reliable!

TrackPos: Distance between the car and the track axis. The value is normalized w.r.t the track width: it is 0 when car is on the axis, -1 when the car is on the left edge of the track and +1 when it is on the right edge of the car. Values greater than 1 or smaller than -1 means that the car is outside of the track.

wheelSpinVel: Vector of 4 sensors representing the rotation speed of the wheels.

speedX: Speed of the car along the longitudinal axis of the car.

speedY: Speed of the car along the transverse axis of the car.

Some of the actuators are, accel, brake, gear, steering, meta

Accel: Virtual gas pedal (0 is no gas, 1 is full gas).

Brake: Virtual brake pedal (0 is no brake, 1 is full brake).

Gear: Gear value defined in {-1,0,1,2,3,4,5,6} where -1 is reverse and 0 is neutral.

Steering: Steering value: -1 and +1 means respectively full left and right, that corresponds to an angle of 0.785398 rad.

Meta: This is meta-control command: 0 do nothing, 1 ask competition server to restart the race.

In the evaluation lap, the performance achieved is recorded and used to compute the fitness of the network as follows: $F = C1 - Tout + C2 \cdot \bar{s} + d$,

After 2000 game tics, corresponding to 40s of simulated time, the evaluation is over and the performance of the controller is computed as: $P = C - \alpha \cdot Tout - \beta \cdot Tcollision - \gamma \cdot \Delta$,

Our results show that NEAT is able to evolve a driving skill that is competitive with the best human programmed controller available. Similarly, the overtaking skill evolved by NEAT outperformed both the best programmed and the best evolved controllers. In the final experiments we compared the best controllers of the Simulated Car Racing Competition to the ones evolved with our approach: a controller with only the driving skill, a controller with only the overtaking skill, and a controller that combines both the skills. The results suggest that either the driving skill or the overtaking skill alone does not lead to a very competitive and reliable car controller.

Controller	Overtaking Time	Tcollision	Tout	Success Rate
NEAT driver	351.77 ± 151.21	109.03 ± 107.99	1.11 ± 11.00	55.2%
NEAT overtaker	271.49 ± 135.57	47.10 ± 97.85	13.50 ± 39.38	86.7%

Paper-2

Problem analysis

The presented T2 fuzzy control structure is composed of two important blocks which are the reference generator and Single Input Interval T2 Fuzzy Logic Controller (SIT2-FLC). The reference generator is the mechanism which uses the bird's position and the pipes' positions to generate an appropriate reference signal to be tracked. the generated reference signal is tracked via the presented SIT2-FLC that can be easily tuned while also provides a certain degree of robustness to system. The game formed the basis for several researches to achieve autonomously flying bird

Proposed algorithm:

Single Input Interval T2 Fuzzy Logic Controller (SIT2-FLC).

Evaluation of proposed algorithm

The proposed T2 fuzzy control system is composed of three main parts which are the reference generator, the SIT2-FLC, and the system dynamics of the bird.

The reference generator provides the trajectory for the bird by taking account the gap between the pipes and the bird's position. The reference trajectory (r_i) is updated when the bird reaches the end of the pipe set (T_i) automatically Then, the new reference trajectory is generated as follows:

$r_{i+1} = \underline{p}_h + 0.3\left(\overline{p}_h - \underline{p}_h\right)$ where i is the frame when the bird reaches the end of the pipe. The generation of the value r_{i+1}.

It can be clearly observed that the T2 fuzzy control scheme improved the average score almost by 55% in comparison with its baseline counterpart.

Code:

```
"""
The classic game of flappy bird. Make with python
and pygame. Features pixel perfect collision using masks :o

Date Modified:  Jul 30, 2019
Author: Tech With Tim
Estimated Work Time: 5 hours (1 just for that damn collision)
"""
import pygame
import random
import os
import time
import neat
import visualize
import pickle
pygame.font.init()  # init font

WIN_WIDTH = 600
WIN_HEIGHT = 800
FLOOR = 730
```

```
STAT_FONT = pygame.font.SysFont("comicsans", 50)
END_FONT = pygame.font.SysFont("comicsans", 70)
DRAW_LINES = False

WIN = pygame.display.set_mode((WIN_WIDTH, WIN_HEIGHT))
pygame.display.set_caption("Flappy Bird")

pipe_img = pygame.transform.scale2x(pygame.image.load(os.path.join("imgs","pipe.png")).convert_alpha())
bg_img = pygame.transform.scale(pygame.image.load(os.path.join("imgs","bg.png")).convert_alpha(), (600, 900))
bird_images = [pygame.transform.scale2x(pygame.image.load(os.path.join("imgs","bird" + str(x) + ".png"))) for x in range(1,4)]
base_img = pygame.transform.scale2x(pygame.image.load(os.path.join("imgs","base.png")).convert_alpha())

gen = 0

class Bird:
    """
    Bird class representing the flappy bird
    """
    MAX_ROTATION = 25
    IMGS = bird_images
    ROT_VEL = 20
    ANIMATION_TIME = 5

    def __init__(self, x, y):
        """
        Initialize the object
        :param x: starting x pos (int)
        :param y: starting y pos (int)
        :return: None
        """
        self.x = x
        self.y = y
        self.tilt = 0  # degrees to tilt
        self.tick_count = 0
        self.vel = 0
        self.height = self.y
        self.img_count = 0
        self.img = self.IMGS[0]

    def jump(self):
        """
        make the bird jump
        :return: None
        """
        self.vel = -10.5
        self.tick_count = 0
        self.height = self.y

    def move(self):
        """
        make the bird move
        :return: None
        """
        self.tick_count += 1

        # for downward acceleration
        displacement = self.vel*(self.tick_count) + 0.5*(3)*(self.tick_count)**2  # calculate
displacement

        # terminal velocity
        if displacement >= 16:
            displacement = (displacement/abs(displacement)) * 16

        if displacement < 0:
            displacement -= 2

        self.y = self.y + displacement

        if displacement < 0 or self.y < self.height + 50:  # tilt up
            if self.tilt < self.MAX_ROTATION:
                self.tilt = self.MAX_ROTATION
        else:  # tilt down
            if self.tilt > -90:
                self.tilt -= self.ROT_VEL

    def draw(self, win):
        """
        draw the bird
        :param win: pygame window or surface
        :return: None
        """
        self.img_count += 1

        # For animation of bird, loop through three images
        if self.img_count <= self.ANIMATION_TIME:
            self.img = self.IMGS[0]
        elif self.img_count <= self.ANIMATION_TIME*2:
            self.img = self.IMGS[1]
        elif self.img_count <= self.ANIMATION_TIME*3:
            self.img = self.IMGS[2]
        elif self.img_count <= self.ANIMATION_TIME*4:
            self.img = self.IMGS[1]
        elif self.img_count == self.ANIMATION_TIME*4 + 1:
            self.img = self.IMGS[0]
            self.img_count = 0

        # so when bird is nose diving it isn't flapping
        if self.tilt <= -80:
            self.img = self.IMGS[1]
            self.img_count = self.ANIMATION_TIME*2



        # tilt the bird
        blitRotateCenter(win, self.img, (self.x, self.y), self.tilt)

    def get_mask(self):
        """
        gets the mask for the current image of the bird
        :return: None
        """
        return pygame.mask.from_surface(self.img)



class Pipe():
    """
    represents a pipe object
    """
```

```
GAP = 200
VEL = 5

def __init__(self, x):
    """
    initialize pipe object
    :param x: int
    :param y: int
    :return" None
    """
    self.x = x
    self.height = 0

    # where the top and bottom of the pipe is
    self.top = 0
    self.bottom = 0

    self.PIPE_TOP = pygame.transform.flip(pipe_img, False, True)
    self.PIPE_BOTTOM = pipe_img

    self.passed = False

    self.set_height()

def set_height(self):
    """
    set the height of the pipe, from the top of the screen
    :return: None
    """
    self.height = random.randrange(50, 450)
```

```
        self.top = self.height - self.PIPE_TOP.get_height()
        self.bottom = self.height + self.GAP

    def move(self):
        """
        move pipe based on vel
        :return: None
        """
        self.x -= self.VEL

    def draw(self, win):
        """
        draw both the top and bottom of the pipe
        :param win: pygame window/surface
        :return: None
        """
        # draw top
        win.blit(self.PIPE_TOP, (self.x, self.top))
        # draw bottom
        win.blit(self.PIPE_BOTTOM, (self.x, self.bottom))


    def collide(self, bird, win):
        """
        returns if a point is colliding with the pipe
        :param bird: Bird object
        :return: Bool
        """
        bird_mask = bird.get_mask()
        top_mask = pygame.mask.from_surface(self.PIPE_TOP)
        bottom_mask = pygame.mask.from_surface(self.PIPE_BOTTOM)
        top_offset = (self.x - bird.x, self.top - round(bird.y))
        bottom_offset = (self.x - bird.x, self.bottom - round(bird.y))

        b_point = bird_mask.overlap(bottom_mask, bottom_offset)
        t_point = bird_mask.overlap(top_mask,top_offset)

        if b_point or t_point:
            return True

        return False

class Base:
    """
    Represnts the moving floor of the game
    """
    VEL = 5
    WIDTH = base_img.get_width()
    IMG = base_img

    def __init__(self, y):
        """
        Initialize the object
        :param y: int
        :return: None
        """
        self.y = y
        self.x1 = 0
        self.x2 = self.WIDTH
    def move(self):
        """
        move floor so it looks like its scrolling
        :return: None
        """
        self.x1 -= self.VEL
        self.x2 -= self.VEL
        if self.x1 + self.WIDTH < 0:
            self.x1 = self.x2 + self.WIDTH

        if self.x2 + self.WIDTH < 0:
            self.x2 = self.x1 + self.WIDTH

    def draw(self, win):
        """
        Draw the floor. This is two images that move together.
        :param win: the pygame surface/window
        :return: None
        """
        win.blit(self.IMG, (self.x1, self.y))
        win.blit(self.IMG, (self.x2, self.y))


def blitRotateCenter(surf, image, topleft, angle):
    """
    Rotate a surface and blit it to the window
    :param surf: the surface to blit to
    :param image: the image surface to rotate
    :param topLeft: the top left position of the image
    :param angle: a float value for angle
    :return: None
    """
    rotated_image = pygame.transform.rotate(image, angle)
    new_rect = rotated_image.get_rect(center = image.get_rect(topleft = topleft).center)

    surf.blit(rotated_image, new_rect.topleft)

def draw_window(win, birds, pipes, base, score, gen, pipe_ind):
    """
    draws the windows for the main game loop
    :param win: pygame window surface
    :param bird: a Bird object
    :param pipes: List of pipes
    :param score: score of the game (int)
    :param gen: current generation
    :param pipe_ind: index of closest pipe
    :return: None
    """
    if gen == 0:
        gen = 1
    win.blit(bg_img, (0,0))

    for pipe in pipes:
        pipe.draw(win)

    base.draw(win)
    for bird in birds:
        # draw lines from bird to pipe
        if DRAW_LINES:
            try:
                pygame.draw.line(win, (255,0,0), (bird.x+bird.img.get_width()/2, bird.y + bird.img.get_height()/2), (pipes[pipe_ind].x + pipes[pipe_ind].PIPE_TOP.get_width()/2, pipes[pipe_ind].height), 5)

                pygame.draw.line(win, (255,0,0), (bird.x+bird.img.get_width()/2, bird.y + bird.img.get_height()/2), (pipes[pipe_ind].x + pipes[pipe_ind].PIPE_BOTTOM.get_width()/2, pipes[pipe_ind].bottom), 5)

            except:

                pass

        # draw bird

        bird.draw(win)


    # score

    score_label = STAT_FONT.render("Score: " + str(score),1,(255,255,255))

    win.blit(score_label, (WIN_WIDTH - score_label.get_width() - 15, 10))


    # generations

    score_label = STAT_FONT.render("Gens: " + str(gen-1),1,(255,255,255))

    win.blit(score_label, (10, 10))


    # alive

    score_label = STAT_FONT.render("Alive: " + str(len(birds)),1,(255,255,255))

    win.blit(score_label, (10, 50))


    pygame.display.update()



def eval_genomes(genomes, config):

    """

    runs the simulation of the current population of

    birds and sets their fitness based on the distance they

    reach in the game.

    """
```

```
global WIN, gen
win = WIN
gen += 1

# start by creating lists holding the genome itself, the
# neural network associated with the genome and the
# bird object that uses that network to play
nets = []
birds = []
ge = []
for genome_id, genome in genomes:
    genome.fitness = 0  # start with fitness level of 0
    net = neat.nn.FeedForwardNetwork.create(genome, config)
    nets.append(net)
    birds.append(Bird(230,350))
    ge.append(genome)

base = Base(FLOOR)
pipes = [Pipe(700)]
score = 0

clock = pygame.time.Clock()

run = True
while run and len(birds) > 0:
    clock.tick(30)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
```

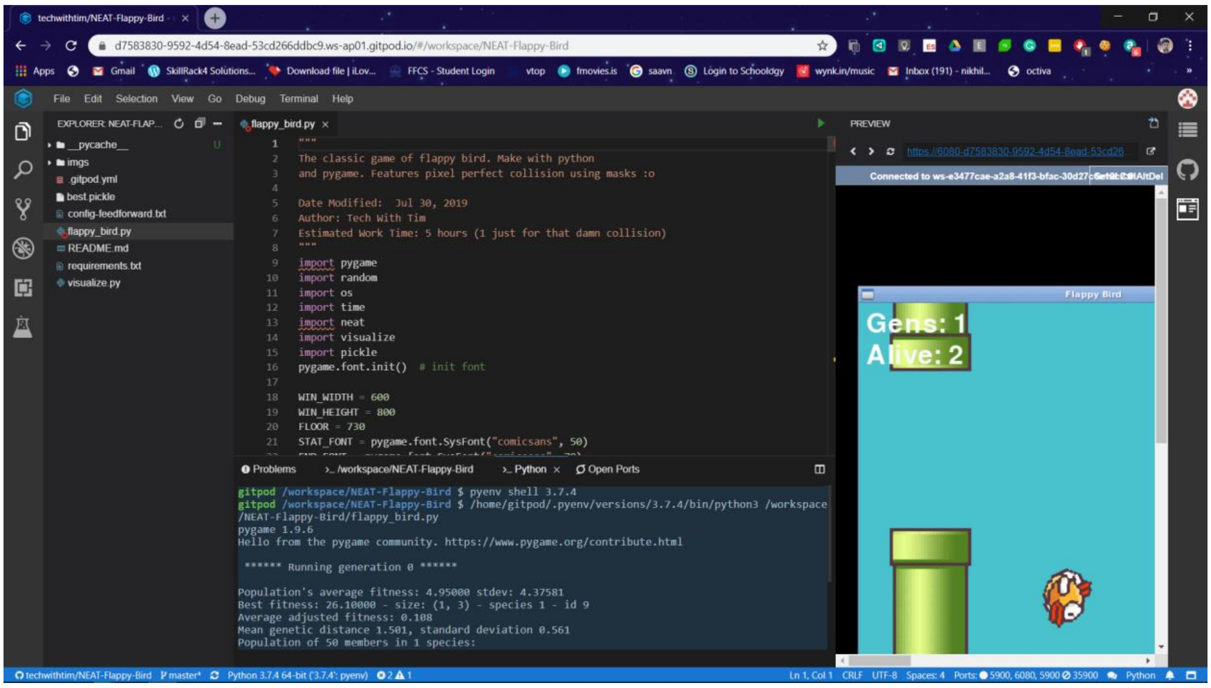
```
                pygame.quit()
                quit()
                break

        pipe_ind = 0
        if len(birds) > 0:
            if len(pipes) > 1 and birds[0].x > pipes[0].x + pipes[0].PIPE_TOP.get_width():  #
determine whether to use the first or second
                pipe_ind = 1                                                                 # pipe on the screen for neural
network input

        for x, bird in enumerate(birds):  # give each bird a fitness of 0.1 for each frame it stays
alive
            ge[x].fitness += 0.1
            bird.move()

            # send bird location, top pipe location and bottom pipe location and determine from
network whether to jump or not
            output = nets[birds.index(bird)].activate((bird.y, abs(bird.y - pipes[pipe_ind].height),
abs(bird.y - pipes[pipe_ind].bottom)))

            if output[0] > 0.5:  # we use a tanh activation function so result will be between -1 and
1. if over 0.5 jump
                bird.jump()

        base.move()

        rem = []
        add_pipe = False
        for pipe in pipes:
            pipe.move()
            # check for collision
```

```
    for bird in birds:
        if pipe.collide(bird, win):
            ge[birds.index(bird)].fitness -= 1
            nets.pop(birds.index(bird))
            ge.pop(birds.index(bird))
            birds.pop(birds.index(bird))

    if pipe.x + pipe.PIPE_TOP.get_width() < 0:
        rem.append(pipe)

    if not pipe.passed and pipe.x < bird.x:
        pipe.passed = True
        add_pipe = True

if add_pipe:
    score += 1
    # can add this line to give more reward for passing through a pipe (not required)
    for genome in ge:
        genome.fitness += 5
    pipes.append(Pipe(WIN_WIDTH))

for r in rem:
    pipes.remove(r)

for bird in birds:
    if bird.y + bird.img.get_height() - 10 >= FLOOR or bird.y < -50:
        nets.pop(birds.index(bird))
        ge.pop(birds.index(bird))
        birds.pop(birds.index(bird))
        draw_window(WIN, birds, pipes, base, score, gen, pipe_ind)

        # break if score gets large enough
        '''if score > 20:
            pickle.dump(nets[0],open("best.pickle", "wb"))
            break'''



def run(config_path):
    """
    runs the NEAT algorithm to train a neural network to play flappy bird.
    :param config_file: location of config file
    :return: None
    """
    config = neat.config.Config(neat.DefaultGenome, neat.DefaultReproduction,
                         neat.DefaultSpeciesSet, neat.DefaultStagnation,
                         config_path)

    # Create the population, which is the top-level object for a NEAT run.
    p = neat.Population(config)

    # Add a stdout reporter to show progress in the terminal.
    p.add_reporter(neat.StdOutReporter(True))
    stats = neat.StatisticsReporter()
    p.add_reporter(stats)
    #p.add_reporter(neat.Checkpointer(5))

    # Run for up to 50 generations.
    winner = p.run(eval_genomes, 50)
    # show final stats
    print('\nBest genome:\n{!s}'.format(winner))



if __name__ == '__main__':
    # Determine path to configuration file. This path manipulation is
    # here so that the script will run successfully regardless of the
    # current working directory.
    local_dir = os.path.dirname(__file__)
    config_path = os.path.join(local_dir, 'config-feedforward.txt')
    run(config_path)
```

Output:

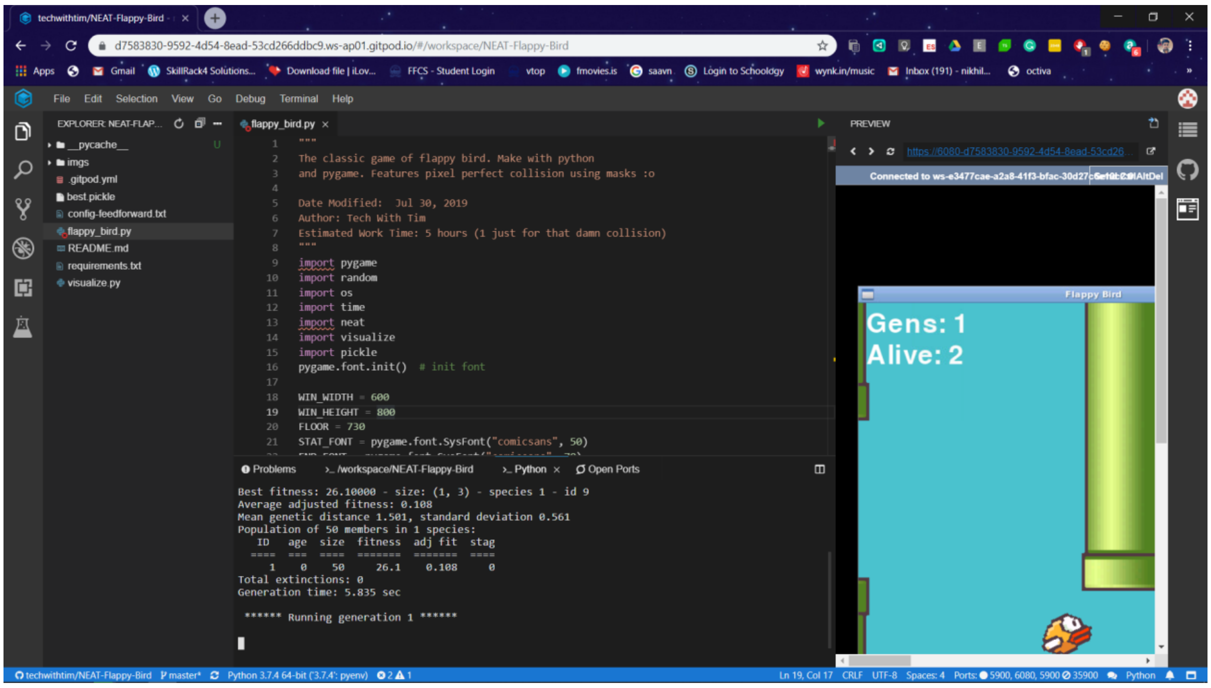
```
gitpod /workspace/NEAT-Flappy-Bird $ pyenv shell 3.7.4
gitpod /workspace/NEAT-Flappy-Bird $ /home/gitpod/.pyenv/versions/3.7.4/bin/python3 /workspace/NEAT-Flappy-Bird/flappy_bird.py
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html

 ****** Running generation 0 ******

Population's average fitness: 4.95000 stdev: 4.37581
Best fitness: 26.10000 - size: (1, 3) - species 1 - id 9
Average adjusted fitness: 0.108
Mean genetic distance 1.501, standard deviation 0.561
Population of 50 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    0    50     26.1    0.108     0
Total extinctions: 0
Generation time: 5.835 sec
```

****** Running generation 1 ******



```python
"""
The classic game of flappy bird. Make with python
and pygame. Features pixel perfect collision using masks :o

Date Modified:  Jul 30, 2019
Author: Tech With Tim
Estimated Work Time: 5 hours (1 just for that damn collision)
"""
import pygame
import random
import os
import time
import neat
import visualize
import pickle
pygame.font.init()  # init font

WIN_WIDTH = 600
WIN_HEIGHT = 800
FLOOR = 730
STAT_FONT = pygame.font.SysFont("comicsans", 50)
```

```
gitpod /workspace/NEAT-Flappy-Bird $ pyenv shell 3.7.4
gitpod /workspace/NEAT-Flappy-Bird $ /home/gitpod/.pyenv/versions/3.7.4/bin/python3 /workspace
/NEAT-Flappy-Bird/flappy_bird.py
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html

 ****** Running generation 0 ******

Population's average fitness: 4.95000 stdev: 4.37581
Best fitness: 26.10000 - size: (1, 3) - species 1 - id 9
Average adjusted fitness: 0.108
Mean genetic distance 1.501, standard deviation 0.561
Population of 50 members in 1 species:
```



```python
"""
The classic game of flappy bird. Make with python
and pygame. Features pixel perfect collision using masks :o

Date Modified:  Jul 30, 2019
Author: Tech With Tim
Estimated Work Time: 5 hours (1 just for that damn collision)
"""
import pygame
import random
import os
import time
import neat
import visualize
import pickle
pygame.font.init()  # init font

WIN_WIDTH = 600
WIN_HEIGHT = 800
FLOOR = 730
STAT_FONT = pygame.font.SysFont("comicsans", 50)
```

```
Best fitness: 26.10000 - size: (1, 3) - species 1 - id 9
Average adjusted fitness: 0.108
Mean genetic distance 1.501, standard deviation 0.561
Population of 50 members in 1 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    0    50     26.1    0.108     0
Total extinctions: 0
Generation time: 5.835 sec

 ****** Running generation 1 ******

█
```