

**Python – Gaming**

*Course Plan*

**Course Developer**: Darshil Shah

**Last update:** 12/12/2020

# **7 Golden Rules to code**

* Learn by doing. Always play with the code while learning. ...
* Grasp the fundamentals for long-term benefits. ...
* Code by hand. ...
* Ask for help. ...
* Seek out more online resources. ...
* Don't just read the sample code. ...
* Take breaks when debugging. ...

**Concept :- OOPS**

# **Problem Statement**

Introduction to OOPS

# **Instruction**

Create a Common base class for all Students

Add Method :- displayCount

Add Method :- displayStudent

# **Hint**

Class − A user-defined blueprint for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

Instance − An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.

# **Activity**

class Student:

'Common base class for all Students'

studCount = 0

def \_\_init\_\_(self, name, grade):

self.name = name

self.grade = grade

Student.studCount += 1

def displayCount(self):

print(f"Total Student {self.studCount}")

def displayStudent(self):

print(f"Name : {self.name} , grade: {self.grade}")

"This would create first object of Student class"

emp1 = Student("Zara", 2)

"This would create second object of Student class"

emp2 = Student("Mani", 5)

emp1.displayStudent()

emp2.displayStudent()

print("Total Student %d" % Student.studCount)

**Game I :- Man and Enemy**

# **Problem Statement**

Getting Started with Pygame by developing a basic Man V/S Enemy Game

# **Instruction**

Generate the following components of the game:

* Basic Movement
* Jumping & Boundaries
* Character Animation
* Optimization & OOP
* Projectiles
* Enemies
* Collision & Hitboxes
* Scoring & Health Bars
* Finishing Touches

# **Hint**

* Load the images
* Set & Update (X,Y) positions of the character
* Set Frames per Second
* Have a main game Loop
* Redraw the window every frame with the required surfaces.

# **Activity**

*import* pygame

pygame.init()

win = pygame.display.set\_mode((500,480))

pygame.display.set\_caption("First Game")

walkRight = [pygame.image.load('R1.png'), pygame.image.load('R2.png'), pygame.image.load('R3.png'), pygame.image.load('R4.png'), pygame.image.load('R5.png'), pygame.image.load('R6.png'), pygame.image.load('R7.png'), pygame.image.load('R8.png'), pygame.image.load('R9.png')]

walkLeft = [pygame.image.load('L1.png'), pygame.image.load('L2.png'), pygame.image.load('L3.png'), pygame.image.load('L4.png'), pygame.image.load('L5.png'), pygame.image.load('L6.png'), pygame.image.load('L7.png'), pygame.image.load('L8.png'), pygame.image.load('L9.png')]

bg = pygame.image.load('bg.jpg')

char = pygame.image.load('standing.png')

clock = pygame.time.Clock()

bulletSound = pygame.mixer.Sound('bullet.wav')

hitSound = pygame.mixer.Sound('hit.wav')

music = pygame.mixer.music.load('music.mp3')

pygame.mixer.music.play(-1)

score = 0

class player(object):

def \_\_init\_\_(self,x,y,width,height):

self.x = x

self.y = y

self.width = width

self.height = height

self.vel = 5

self.isJump = False

self.left = False

self.right = False

self.walkCount = 0

self.jumpCount = 10

self.standing = True

self.hitbox = (self.x + 17, self.y + 11, 29, 52)

def draw(self, win):

*if* self.walkCount + 1 >= 27:

self.walkCount = 0

*if* not(self.standing):

*if* self.left:

win.blit(walkLeft[self.walkCount//3], (self.x,self.y))

self.walkCount += 1

*elif* self.right:

win.blit(walkRight[self.walkCount//3], (self.x,self.y))

self.walkCount +=1

*else*:

*if* self.right:

win.blit(walkRight[0], (self.x, self.y))

*else*:

win.blit(walkLeft[0], (self.x, self.y))

self.hitbox = (self.x + 17, self.y + 11, 29, 52)

*#pygame.draw.rect(win, (255,0,0), self.hitbox,2)*

def hit(self):

self.isJump = False

self.jumpCount = 10

self.x = 100

self.y = 410

self.walkCount = 0

font1 = pygame.font.SysFont('comicsans', 100)

text = font1.render('-5', 1, (255,0,0))

win.blit(text, (250 - (text.get\_width()/2),200))

pygame.display.update()

i = 0

*while* i < 200:

pygame.time.delay(10)

i += 1

*for* event *in* pygame.event.get():

*if* event.type == pygame.QUIT:

i = 201

pygame.quit()

class projectile(object):

def \_\_init\_\_(self,x,y,radius,color,facing):

self.x = x

self.y = y

self.radius = radius

self.color = color

self.facing = facing

self.vel = 8 \* facing

def draw(self,win):

pygame.draw.circle(win, self.color, (self.x,self.y), self.radius)

class enemy(object):

walkRight = [pygame.image.load('R1E.png'), pygame.image.load('R2E.png'), pygame.image.load('R3E.png'), pygame.image.load('R4E.png'), pygame.image.load('R5E.png'), pygame.image.load('R6E.png'), pygame.image.load('R7E.png'), pygame.image.load('R8E.png'), pygame.image.load('R9E.png'), pygame.image.load('R10E.png'), pygame.image.load('R11E.png')]

walkLeft = [pygame.image.load('L1E.png'), pygame.image.load('L2E.png'), pygame.image.load('L3E.png'), pygame.image.load('L4E.png'), pygame.image.load('L5E.png'), pygame.image.load('L6E.png'), pygame.image.load('L7E.png'), pygame.image.load('L8E.png'), pygame.image.load('L9E.png'), pygame.image.load('L10E.png'), pygame.image.load('L11E.png')]

def \_\_init\_\_(self, x, y, width, height, end):

self.x = x

self.y = y

self.width = width

self.height = height

self.end = end

self.path = [self.x, self.end]

self.walkCount = 0

self.vel = 3

self.hitbox = (self.x + 17, self.y + 2, 31, 57)

self.health = 10

self.visible = True

def draw(self,win):

self.move()

*if* self.visible:

*if* self.walkCount + 1 >= 33:

self.walkCount = 0

*if* self.vel > 0:

win.blit(self.walkRight[self.walkCount //3], (self.x, self.y))

self.walkCount += 1

*else*:

win.blit(self.walkLeft[self.walkCount //3], (self.x, self.y))

self.walkCount += 1

pygame.draw.rect(win, (255,0,0), (self.hitbox[0], self.hitbox[1] - 20, 50, 10))

pygame.draw.rect(win, (0,128,0), (self.hitbox[0], self.hitbox[1] - 20, 50 - (5 \* (10 - self.health)), 10))

self.hitbox = (self.x + 17, self.y + 2, 31, 57)

*#pygame.draw.rect(win, (255,0,0), self.hitbox,2)*

def move(self):

*if* self.vel > 0:

*if* self.x + self.vel < self.path[1]:

self.x += self.vel

*else*:

self.vel = self.vel \* -1

self.walkCount = 0

*else*:

*if* self.x - self.vel > self.path[0]:

self.x += self.vel

*else*:

self.vel = self.vel \* -1

self.walkCount = 0

def hit(self):

*if* self.health > 0:

self.health -= 1

*else*:

self.visible = False

print('hit')

def redrawGameWindow():

win.blit(bg, (0,0))

text = font.render('Score: ' + str(score), 1, (0,0,0))

win.blit(text, (350, 10))

man.draw(win)

goblin.draw(win)

*for* bullet *in* bullets:

bullet.draw(win)

pygame.display.update()

*#mainloop*

font = pygame.font.SysFont('comicsans', 30, True)

man = player(200, 410, 64,64)

goblin = enemy(100, 410, 64, 64, 450)

shootLoop = 0

bullets = []

run = True

*while* run:

clock.tick(27)

*if* goblin.visible == True:

*if* man.hitbox[1] < goblin.hitbox[1] + goblin.hitbox[3] and man.hitbox[1] + man.hitbox[3] > goblin.hitbox[1]:

*if* man.hitbox[0] + man.hitbox[2] > goblin.hitbox[0] and man.hitbox[0] < goblin.hitbox[0] + goblin.hitbox[2]:

man.hit()

score -= 5

*if* shootLoop > 0:

shootLoop += 1

*if* shootLoop > 3:

shootLoop = 0

*for* event *in* pygame.event.get():

*if* event.type == pygame.QUIT:

run = False

*for* bullet *in* bullets:

*if* bullet.y - bullet.radius < goblin.hitbox[1] + goblin.hitbox[3] and bullet.y + bullet.radius > goblin.hitbox[1]:

*if* bullet.x + bullet.radius > goblin.hitbox[0] and bullet.x - bullet.radius < goblin.hitbox[0] + goblin.hitbox[2]:

hitSound.play()

goblin.hit()

score += 1

bullets.pop(bullets.index(bullet))

*if* bullet.x < 500 and bullet.x > 0:

bullet.x += bullet.vel

*else*:

bullets.pop(bullets.index(bullet))

keys = pygame.key.get\_pressed()

*if* keys[pygame.K\_SPACE] and shootLoop == 0:

bulletSound.play()

*if* man.left:

facing = -1

*else*:

facing = 1

*if* len(bullets) < 5:

bullets.append(projectile(round(man.x + man.width //2), round(man.y + man.height//2), 6, (0,0,0), facing))

shootLoop = 1

*if* keys[pygame.K\_LEFT] and man.x > man.vel:

man.x -= man.vel

man.left = True

man.right = False

man.standing = False

*elif* keys[pygame.K\_RIGHT] and man.x < 500 - man.width - man.vel:

man.x += man.vel

man.right = True

man.left = False

man.standing = False

*else*:

man.standing = True

man.walkCount = 0

*if* not(man.isJump):

*if* keys[pygame.K\_UP]:

man.isJump = True

man.right = False

man.left = False

man.walkCount = 0

*else*:

*if* man.jumpCount >= -10:

neg = 1

*if* man.jumpCount < 0:

neg = -1

man.y -= (man.jumpCount \*\* 2) \* 0.5 \* neg

man.jumpCount -= 1

*else*:

man.isJump = False

man.jumpCount = 10

redrawGameWindow()

pygame.quit()

page break

**Game II :- Classical Snake**

# **Problem Statement**

Game II : Develop the classical snake game using Pygame Module

# **Instruction**

Generate the following components of the game:

* Basic Movement
* Random Food & Boundaries
* Character Animation
* Optimization & OOP
* Collision & Hitboxes
* Scoring
* Finishing Touches

# **Hint**

* Load the images
* Set & Update (X,Y) positions of the Snake and Food
* Set Frames per Second
* Have a main game Loop
* Redraw the window every frame with the required surfaces.

# **Activity**

*#Snake Tutorial Python*

*import* math

*import* random

*import* pygame

*import* tkinter *as* tk

*from* tkinter *import* messagebox

class cube(object):

rows = 20

w = 500

def \_\_init\_\_(self,start,dirnx=1,dirny=0,color=(255,0,0)):

self.pos = start

self.dirnx = 1

self.dirny = 0

self.color = color

def move(self, dirnx, dirny):

self.dirnx = dirnx

self.dirny = dirny

self.pos = (self.pos[0] + self.dirnx, self.pos[1] + self.dirny)

def draw(self, surface, eyes=False):

dis = self.w // self.rows

i = self.pos[0]

j = self.pos[1]

pygame.draw.rect(surface, self.color, (i\*dis+1,j\*dis+1, dis-2, dis-2))

*if* eyes:

centre = dis//2

radius = 3

circleMiddle = (i\*dis+centre-radius,j\*dis+8)

circleMiddle2 = (i\*dis + dis -radius\*2, j\*dis+8)

pygame.draw.circle(surface, (0,0,0), circleMiddle, radius)

pygame.draw.circle(surface, (0,0,0), circleMiddle2, radius)

class snake(object):

body = []

turns = {}

def \_\_init\_\_(self, color, pos):

self.color = color

self.head = cube(pos)

self.body.append(self.head)

self.dirnx = 0

self.dirny = 1

def move(self):

*for* event *in* pygame.event.get():

*if* event.type == pygame.QUIT:

pygame.quit()

keys = pygame.key.get\_pressed()

*for* key *in* keys:

*if* keys[pygame.K\_LEFT]:

self.dirnx = -1

self.dirny = 0

self.turns[self.head.pos[:]] = [self.dirnx, self.dirny]

*elif* keys[pygame.K\_RIGHT]:

self.dirnx = 1

self.dirny = 0

self.turns[self.head.pos[:]] = [self.dirnx, self.dirny]

*elif* keys[pygame.K\_UP]:

self.dirnx = 0

self.dirny = -1

self.turns[self.head.pos[:]] = [self.dirnx, self.dirny]

*elif* keys[pygame.K\_DOWN]:

self.dirnx = 0

self.dirny = 1

self.turns[self.head.pos[:]] = [self.dirnx, self.dirny]

*for* i, c *in* enumerate(self.body):

p = c.pos[:]

*if* p in self.turns:

turn = self.turns[p]

c.move(turn[0],turn[1])

*if* i == len(self.body)-1:

self.turns.pop(p)

*else*:

*if* c.dirnx == -1 and c.pos[0] <= 0: c.pos = (c.rows-1, c.pos[1])

*elif* c.dirnx == 1 and c.pos[0] >= c.rows-1: c.pos = (0,c.pos[1])

*elif* c.dirny == 1 and c.pos[1] >= c.rows-1: c.pos = (c.pos[0], 0)

*elif* c.dirny == -1 and c.pos[1] <= 0: c.pos = (c.pos[0],c.rows-1)

*else*: c.move(c.dirnx,c.dirny)

def reset(self, pos):

self.head = cube(pos)

self.body = []

self.body.append(self.head)

self.turns = {}

self.dirnx = 0

self.dirny = 1

def addCube(self):

tail = self.body[-1]

dx, dy = tail.dirnx, tail.dirny

*if* dx == 1 and dy == 0:

self.body.append(cube((tail.pos[0]-1,tail.pos[1])))

*elif* dx == -1 and dy == 0:

self.body.append(cube((tail.pos[0]+1,tail.pos[1])))

*elif* dx == 0 and dy == 1:

self.body.append(cube((tail.pos[0],tail.pos[1]-1)))

*elif* dx == 0 and dy == -1:

self.body.append(cube((tail.pos[0],tail.pos[1]+1)))

self.body[-1].dirnx = dx

self.body[-1].dirny = dy

def draw(self, surface):

*for* i, c *in* enumerate(self.body):

*if* i ==0:

c.draw(surface, True)

*else*:

c.draw(surface)

def drawGrid(w, rows, surface):

sizeBtwn = w // rows

x = 0

y = 0

*for* l *in* range(rows):

x = x + sizeBtwn

y = y + sizeBtwn

pygame.draw.line(surface, (255,255,255), (x,0),(x,w))

pygame.draw.line(surface, (255,255,255), (0,y),(w,y))

def redrawWindow(surface):

global rows, width, s, snack

surface.fill((0,0,0))

s.draw(surface)

snack.draw(surface)

drawGrid(width,rows, surface)

pygame.display.update()

def randomSnack(rows, item):

positions = item.body

*while* True:

x = random.randrange(rows)

y = random.randrange(rows)

*if* len(list(filter(lambda z:z.pos == (x,y), positions))) > 0:

*continue*

*else*:

*break*

*return* (x,y)

def message\_box(subject, content):

root = tk.Tk()

root.attributes("-topmost", True)

root.withdraw()

messagebox.showinfo(subject, content)

*try*:

root.destroy()

*except*:

*pass*

def main():

global width, rows, s, snack

width = 500

rows = 20

win = pygame.display.set\_mode((width, width))

s = snake((255,0,0), (10,10))

snack = cube(randomSnack(rows, s), color=(0,255,0))

flag = True

clock = pygame.time.Clock()

*while* flag:

pygame.time.delay(50)

clock.tick(10)

s.move()

*if* s.body[0].pos == snack.pos:

s.addCube()

snack = cube(randomSnack(rows, s), color=(0,255,0))

*for* x *in* range(len(s.body)):

*if* s.body[x].pos in list(map(lambda z:z.pos,s.body[x+1:])):

print('Score: ', len(s.body))

message\_box('You Lost!', 'Play again...')

s.reset((10,10))

*break*

redrawWindow(win)

main()

page break

**Game III :- Hangman**

# **Problem Statement**

# **Instruction**

Generate the following components of the game:

* Basic Display
* Layout of the Character Engraved Buttons
* Character Animation
* Optimization & OOP
* ]Game Logic
* Selection of the Game Word
* Finishing Touches

# **Hint**

* Load the images as per the logic
* Set & Update (X,Y) positions of the characters
* Respond as per Click Location
* Have a main game Loop
* Redraw the window every frame with the required surfaces.

# **Activity**

*import* pygame

*import* math

*import* random

run = True

*# setup display*

pygame.init()

WIDTH, HEIGHT = 800, 500

win = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Hangman Game!")

*# button variables*

RADIUS = 20

GAP = 15

letters = []

startx = round((WIDTH - (RADIUS \* 2 + GAP) \* 13) / 2)

starty = 400

A = 65

*for* i *in* range(26):

x = startx + GAP \* 2 + ((RADIUS \* 2 + GAP) \* (i % 13))

y = starty + ((i // 13) \* (GAP + RADIUS \* 2))

letters.append([x, y, chr(A + i), True])

*# fonts*

LETTER\_FONT = pygame.font.SysFont('comicsans', 40)

WORD\_FONT = pygame.font.SysFont('comicsans', 60)

TITLE\_FONT = pygame.font.SysFont('comicsans', 70)

*# load images.*

images = []

*for* i *in* range(7):

image = pygame.image.load("hangman" + str(i) + ".png")

images.append(image)

*# game variables*

hangman\_status = 0

words = ["IDE", "REPLIT", "PYTHON", "PYGAME"]

word = random.choice(words)

guessed = []

*# colors*

WHITE = (255,255,255)

BLACK = (0,0,0)

def draw():

win.fill(WHITE)

*# draw title*

text = TITLE\_FONT.render("DEVELOPER HANGMAN", 1, BLACK)

win.blit(text, (WIDTH/2 - text.get\_width()/2, 20))

*# draw word*

display\_word = ""

*for* letter *in* word:

*if* letter in guessed:

display\_word += letter + " "

*else*:

display\_word += "\_ "

text = WORD\_FONT.render(display\_word, 1, BLACK)

win.blit(text, (400, 200))

*# draw buttons*

*for* letter *in* letters:

x, y, ltr, visible = letter

*if* visible:

pygame.draw.circle(win, BLACK, (x, y), RADIUS, 3)

text = LETTER\_FONT.render(ltr, 1, BLACK)

win.blit(text, (x - text.get\_width()/2, y - text.get\_height()/2))

win.blit(images[hangman\_status], (150, 100))

pygame.display.update()

def display\_message(message):

pygame.time.delay(1000)

win.fill(WHITE)

text = WORD\_FONT.render(message, 1, BLACK)

win.blit(text, (WIDTH/2 - text.get\_width()/2, HEIGHT/2 - text.get\_height()/2))

pygame.display.update()

pygame.time.delay(3000)

def main():

global hangman\_status, run

FPS = 60

clock = pygame.time.Clock()

*while* run:

clock.tick(FPS)

*for* event *in* pygame.event.get():

*if* event.type == pygame.QUIT:

run = False

*if* event.type == pygame.MOUSEBUTTONDOWN:

m\_x, m\_y = pygame.mouse.get\_pos()

*for* letter *in* letters:

x, y, ltr, visible = letter

*if* visible:

dis = math.sqrt((x - m\_x)\*\*2 + (y - m\_y)\*\*2)

*if* dis < RADIUS:

letter[3] = False

guessed.append(ltr)

*if* ltr not in word:

hangman\_status += 1

draw()

won = True

*for* letter *in* word:

*if* letter not in guessed:

won = False

*break*

*if* won:

display\_message("You WON!")

*break*

*if* hangman\_status == 6:

display\_message("You LOST!")

*break*

*while* (True & run):

*for* event *in* pygame.event.get():

*if* event.type == pygame.QUIT:

pygame.display.quit()

pygame.quit()

exit(1)

main()

pygame.display.quit()

pygame.quit()

page break

**Game IV :- Side Scroller**

# **Problem Statement**

# **Instruction**

Generate the following components of the game:

* Basic Movement
* Jumping & Boundaries
* Character Animation
* Optimization & OOP
* Projectiles
* Collision & Hit Boxes
* Scoring
* Finishing Touches

# **Hint**

* Load the images
* Set & Update (X,Y) positions of the character
* Set Frames per Second
* Have a main game Loop
* Redraw the window every frame with the required surfaces.

# **Activity**

Starter File:-

*import* pygame

*from* pygame.locals *import* \*

*import* os

*import* sys

*import* math

pygame.init()

W, H = 800, 447

win = pygame.display.set\_mode((W,H))

pygame.display.set\_caption('Side Scroller')

bg = pygame.image.load(os.path.join('images','bg.png')).convert()

bgX = 0

bgX2 = bg.get\_width()

clock = pygame.time.Clock()

class player(object):

run = [pygame.image.load(os.path.join('images', str(x) + '.png')) *for* x *in* range(8,16)]

jump = [pygame.image.load(os.path.join('images', str(x) + '.png')) *for* x *in* range(1,8)]

slide = [pygame.image.load(os.path.join('images', 'S1.png')),pygame.image.load(os.path.join('images', 'S2.png')),pygame.image.load(os.path.join('images', 'S2.png')),pygame.image.load(os.path.join('images', 'S2.png')), pygame.image.load(os.path.join('images', 'S2.png')),pygame.image.load(os.path.join('images', 'S2.png')), pygame.image.load(os.path.join('images', 'S2.png')), pygame.image.load(os.path.join('images', 'S2.png')), pygame.image.load(os.path.join('images', 'S3.png')), pygame.image.load(os.path.join('images', 'S4.png')), pygame.image.load(os.path.join('images', 'S5.png'))]

jumpList = [1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4]

def \_\_init\_\_(self, x, y, width, height):

self.x = x

self.y = y

self.width = width

self.height = height

self.jumping = False

self.sliding = False

self.slideCount = 0

self.jumpCount = 0

self.runCount = 0

self.slideUp = False

def draw(self, win):

*if* self.jumping:

self.y -= self.jumpList[self.jumpCount] \* 1.2

win.blit(self.jump[self.jumpCount//18], (self.x,self.y))

self.jumpCount += 1

*if* self.jumpCount > 108:

self.jumpCount = 0

self.jumping = False

self.runCount = 0

*elif* self.sliding or self.slideUp:

*if* self.slideCount < 20:

self.y += 1

*elif* self.slideCount == 80:

self.y -= 19

self.sliding = False

self.slideUp = True

*if* self.slideCount >= 110:

self.slideCount = 0

self.slideUp = False

self.runCount = 0

win.blit(self.slide[self.slideCount//10], (self.x,self.y))

self.slideCount += 1

*else*:

*if* self.runCount > 42:

self.runCount = 0

win.blit(self.run[self.runCount//6], (self.x,self.y))

self.runCount += 1

Final File:

*import* pygame

*from* pygame.locals *import* \*

*import* os

*import* random

pygame.init()

W, H = 800, 437

win = pygame.display.set\_mode((W,H))

pygame.display.set\_caption('Side Scroller')

bg = pygame.image.load(os.path.join('images', 'bg.png')).convert()

bgX = 0

bgX2 = bg.get\_width()

clock = pygame.time.Clock()

class player(object):

run = [pygame.image.load(os.path.join('images', str(x) + '.png')) *for* x *in* range(8, 16)]

jump = [pygame.image.load(os.path.join('images', str(x) + '.png')) *for* x *in* range(1, 8)]

slide = [pygame.image.load(os.path.join('images', 'S1.png')), pygame.image.load(os.path.join('images', 'S2.png')), pygame.image.load(os.path.join('images', 'S2.png')), pygame.image.load(os.path.join('images', 'S2.png')), pygame.image.load(os.path.join('images', 'S2.png')),pygame.image.load(os.path.join('images', 'S2.png')), pygame.image.load(os.path.join('images', 'S2.png')), pygame.image.load(os.path.join('images', 'S2.png')), pygame.image.load(os.path.join('images', 'S3.png')), pygame.image.load(os.path.join('images', 'S4.png')), pygame.image.load(os.path.join('images', 'S5.png'))]

fall = pygame.image.load(os.path.join('images', '0.png'))

jumpList = [1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-3,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4,-4]

def \_\_init\_\_(self, x, y, width, height):

self.x = x

self.y = y

self.width = width

self.height = height

self.jumping = False

self.sliding = False

self.falling = False

self.slideCount = 0

self.jumpCount = 0

self.runCount = 0

self.slideUp = False

def draw(self, win):

*if* self.falling:

win.blit(self.fall, (self.x, self.y + 30))

*elif* self.jumping:

self.y -= self.jumpList[self.jumpCount] \* 1.3

win.blit(self.jump[self.jumpCount//18], (self.x, self.y))

self.jumpCount += 1

*if* self.jumpCount > 108:

self.jumpCount = 0

self.jumping = False

self.runCount = 0

self.hitbox = (self.x+ 4, self.y, self.width-24, self.height-10)

*elif* self.sliding or self.slideUp:

*if* self.slideCount < 20:

self.y += 1

self.hitbox = (self.x+ 4, self.y, self.width-24, self.height-10)

*elif* self.slideCount == 80:

self.y -= 19

self.sliding = False

self.slideUp = True

*elif* self.slideCount > 20 and self.slideCount < 80:

self.hitbox = (self.x, self.y+3, self.width-8, self.height-35)

*if* self.slideCount >= 110:

self.slideCount = 0

self.runCount = 0

self.slideUp = False

self.hitbox = (self.x+ 4, self.y, self.width-24, self.height-10)

win.blit(self.slide[self.slideCount//10], (self.x, self.y))

self.slideCount += 1

*else*:

*if* self.runCount > 42:

self.runCount = 0

win.blit(self.run[self.runCount//6], (self.x,self.y))

self.runCount += 1

self.hitbox = (self.x+ 4, self.y, self.width-24, self.height-13)

*#pygame.draw.rect(win, (255,0,0),self.hitbox, 2)*

class saw(object):

rotate = [pygame.image.load(os.path.join('images', 'SAW0.png')), pygame.image.load(os.path.join('images', 'SAW1.png')), pygame.image.load(os.path.join('images', 'SAW2.png')), pygame.image.load(os.path.join('images', 'SAW3.png'))]

def \_\_init\_\_(self, x, y, width, height):

self.x = x

self.y = y

self.width = width

self.height = height

self.rotateCount = 0

self.vel = 1.4

def draw(self, win):

self.hitbox = (self.x + 10, self.y + 5, self.width - 20, self.height - 5)

*# pygame.draw.rect(win, (255,0,0), self.hitbox, 2)*

*if* self.rotateCount >= 8:

self.rotateCount = 0

win.blit(pygame.transform.scale(self.rotate[self.rotateCount//2], (64,64)), (self.x,self.y))

self.rotateCount += 1

def collide(self, rect):

*if* rect[0] + rect[2] > self.hitbox[0] and rect[0] < self.hitbox[0] + self.hitbox[2]:

*if* rect[1] + rect[3] > self.hitbox[1]:

*return* True

*return* False

class spike(saw):

img = pygame.image.load(os.path.join('images', 'spike.png'))

def draw(self, win):

self.hitbox = (self.x + 10, self.y, 28,315)

*# pygame.draw.rect(win, (255,0,0), self.hitbox, 2)*

win.blit(self.img, (self.x, self.y))

def collide(self, rect):

*if* rect[0] + rect[2] > self.hitbox[0] and rect[0] < self.hitbox[0] + self.hitbox[2]:

*if* rect[1] < self.hitbox[3]:

*return* True

*return* False

def updateFile():

f = open('scores.txt','r')

file = f.readlines()

last = int(file[0])

*if* last < int(score):

f.close()

file = open('scores.txt', 'w')

file.write(str(score))

file.close()

*return* score

*return* last

def endScreen():

global pause, score, speed, obstacles

pause = 0

speed = 30

obstacles = []

run = True

*while* run:

pygame.time.delay(100)

*for* event *in* pygame.event.get():

*if* event.type == pygame.QUIT:

run = False

pygame.quit()

*if* event.type == pygame.MOUSEBUTTONDOWN:

run = False

runner.falling = False

runner.sliding = False

runner.jumpin = False

win.blit(bg, (0,0))

largeFont = pygame.font.SysFont('comicsans', 80)

lastScore = largeFont.render('Best Score: ' + str(updateFile()),1,(255,255,255))

currentScore = largeFont.render('Score: '+ str(score),1,(255,255,255))

win.blit(lastScore, (W/2 - lastScore.get\_width()/2,150))

win.blit(currentScore, (W/2 - currentScore.get\_width()/2, 240))

pygame.display.update()

score = 0

def redrawWindow():

largeFont = pygame.font.SysFont('comicsans', 30)

win.blit(bg, (bgX, 0))

win.blit(bg, (bgX2,0))

text = largeFont.render('Score: ' + str(score), 1, (255,255,255))

runner.draw(win)

*for* obstacle *in* obstacles:

obstacle.draw(win)

win.blit(text, (700, 10))

pygame.display.update()

pygame.time.set\_timer(USEREVENT+1, 500)

pygame.time.set\_timer(USEREVENT+2, 3000)

speed = 30

score = 0

run = True

runner = player(200, 313, 64, 64)

obstacles = []

pause = 0

fallSpeed = 0

*while* run:

*if* pause > 0:

pause += 1

*if* pause > fallSpeed \* 2:

endScreen()

score = speed//10 - 3

*for* obstacle *in* obstacles:

*if* obstacle.collide(runner.hitbox):

runner.falling = True

*if* pause == 0:

pause = 1

fallSpeed = speed

*if* obstacle.x < -64:

obstacles.pop(obstacles.index(obstacle))

*else*:

obstacle.x -= 1.4

bgX -= 1.4

bgX2 -= 1.4

*if* bgX < bg.get\_width() \* -1:

bgX = bg.get\_width()

*if* bgX2 < bg.get\_width() \* -1:

bgX2 = bg.get\_width()

*for* event *in* pygame.event.get():

*if* event.type == pygame.QUIT:

pygame.quit()

run = False

*if* event.type == USEREVENT+1:

speed += 3

*if* event.type == USEREVENT+2:

r = random.randrange(0,2)

*if* r == 0:

obstacles.append(saw(810, 310, 64, 64))

*elif* r == 1:

obstacles.append(spike(810, 0, 48, 310))

*if* runner.falling == False:

keys = pygame.key.get\_pressed()

*if* keys[pygame.K\_SPACE] or keys[pygame.K\_UP]:

*if* not(runner.jumping):

runner.jumping = True

*if* keys[pygame.K\_DOWN]:

*if* not(runner.sliding):

runner.sliding = True

clock.tick(speed)

redrawWindow()

page break

**Game V :- Space Shooter**

# **Problem Statement**

# **Instruction**

Generate the following components of the game:

* Basic Movement
* Jumping & Boundaries
* Spaceship Animation
* Optimization & OOP
* Projectiles
* Enemies
* Collision & Hit Boxes
* Scoring & Health Bars
* Finishing Touches

# **Hint**

* Load the images
* Set & Update (X,Y) positions of the character
* Set Frames per Second
* Have a main game Loop
* Redraw the window every frame with the required surfaces.

# **Activity**

*import* pygame

*import* os

*import* time

*import* random

pygame.font.init()

WIDTH, HEIGHT = 750, 700

WIN = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Space Shooter Tutorial")

*# Load images*

RED\_SPACE\_SHIP = pygame.image.load(os.path.join("assets", "pixel\_ship\_red\_small.png"))

GREEN\_SPACE\_SHIP = pygame.image.load(os.path.join("assets", "pixel\_ship\_green\_small.png"))

BLUE\_SPACE\_SHIP = pygame.image.load(os.path.join("assets", "pixel\_ship\_blue\_small.png"))

*# Player player*

YELLOW\_SPACE\_SHIP = pygame.image.load(os.path.join("assets", "pixel\_ship\_yellow.png"))

*# Lasers*

RED\_LASER = pygame.image.load(os.path.join("assets", "pixel\_laser\_red.png"))

GREEN\_LASER = pygame.image.load(os.path.join("assets", "pixel\_laser\_green.png"))

BLUE\_LASER = pygame.image.load(os.path.join("assets", "pixel\_laser\_blue.png"))

YELLOW\_LASER = pygame.image.load(os.path.join("assets", "pixel\_laser\_yellow.png"))

*# Background*

BG = pygame.transform.scale(pygame.image.load(os.path.join("assets", "background-black.png")), (WIDTH, HEIGHT))

class Laser:

def \_\_init\_\_(self, x, y, img):

self.x = x

self.y = y

self.img = img

self.mask = pygame.mask.from\_surface(self.img)

def draw(self, window):

window.blit(self.img, (self.x, self.y))

def move(self, vel):

self.y += vel

def off\_screen(self, height):

*return* not(self.y <= height and self.y >= 0)

def collision(self, obj):

*return* collide(self, obj)

class Ship:

COOLDOWN = 30

def \_\_init\_\_(self, x, y, health=100):

self.x = x

self.y = y

self.health = health

self.ship\_img = None

self.laser\_img = None

self.lasers = []

self.cool\_down\_counter = 0

def draw(self, window):

window.blit(self.ship\_img, (self.x, self.y))

*for* laser *in* self.lasers:

laser.draw(window)

def move\_lasers(self, vel, obj):

self.cooldown()

*for* laser *in* self.lasers:

laser.move(vel)

*if* laser.off\_screen(HEIGHT):

self.lasers.remove(laser)

*elif* laser.collision(obj):

obj.health -= 10

self.lasers.remove(laser)

def cooldown(self):

*if* self.cool\_down\_counter >= self.COOLDOWN:

self.cool\_down\_counter = 0

*elif* self.cool\_down\_counter > 0:

self.cool\_down\_counter += 1

def shoot(self):

*if* self.cool\_down\_counter == 0:

laser = Laser(self.x, self.y, self.laser\_img)

self.lasers.append(laser)

self.cool\_down\_counter = 1

def get\_width(self):

*return* self.ship\_img.get\_width()

def get\_height(self):

*return* self.ship\_img.get\_height()

class Player(Ship):

def \_\_init\_\_(self, x, y, health=100):

super().\_\_init\_\_(x, y, health)

self.ship\_img = YELLOW\_SPACE\_SHIP

self.laser\_img = YELLOW\_LASER

self.mask = pygame.mask.from\_surface(self.ship\_img)

self.max\_health = health

def move\_lasers(self, vel, objs):

self.cooldown()

*for* laser *in* self.lasers:

laser.move(vel)

*if* laser.off\_screen(HEIGHT):

self.lasers.remove(laser)

*else*:

*for* obj *in* objs:

*if* laser.collision(obj):

objs.remove(obj)

*if* laser in self.lasers:

self.lasers.remove(laser)

def draw(self, window):

super().draw(window)

self.healthbar(window)

def healthbar(self, window):

pygame.draw.rect(window, (255,0,0), (self.x, self.y + self.ship\_img.get\_height() + 10, self.ship\_img.get\_width(), 10))

pygame.draw.rect(window, (0,255,0), (self.x, self.y + self.ship\_img.get\_height() + 10, self.ship\_img.get\_width() \* (self.health/self.max\_health), 10))

class Enemy(Ship):

COLOR\_MAP = {

"red": (RED\_SPACE\_SHIP, RED\_LASER),

"green": (GREEN\_SPACE\_SHIP, GREEN\_LASER),

"blue": (BLUE\_SPACE\_SHIP, BLUE\_LASER)

}

def \_\_init\_\_(self, x, y, color, health=100):

super().\_\_init\_\_(x, y, health)

self.ship\_img, self.laser\_img = self.COLOR\_MAP[color]

self.mask = pygame.mask.from\_surface(self.ship\_img)

def move(self, vel):

self.y += vel

def shoot(self):

*if* self.cool\_down\_counter == 0:

laser = Laser(self.x-20, self.y, self.laser\_img)

self.lasers.append(laser)

self.cool\_down\_counter = 1

def collide(obj1, obj2):

offset\_x = obj2.x - obj1.x

offset\_y = obj2.y - obj1.y

*return* obj1.mask.overlap(obj2.mask, (offset\_x, offset\_y)) != None

def main():

run = True

FPS = 60

level = 0

lives = 5

main\_font = pygame.font.SysFont("comicsans", 50)

lost\_font = pygame.font.SysFont("comicsans", 60)

enemies = []

wave\_length = 5

enemy\_vel = 1

player\_vel = 5

laser\_vel = 5

player = Player(300, 630)

clock = pygame.time.Clock()

lost = False

lost\_count = 0

def redraw\_window():

WIN.blit(BG, (0,0))

*# draw text*

lives\_label = main\_font.render(f"Lives: {lives}", 1, (255,255,255))

level\_label = main\_font.render(f"Level: {level}", 1, (255,255,255))

WIN.blit(lives\_label, (10, 10))

WIN.blit(level\_label, (WIDTH - level\_label.get\_width() - 10, 10))

*for* enemy *in* enemies:

enemy.draw(WIN)

player.draw(WIN)

*if* lost:

lost\_label = lost\_font.render("You Lost!!", 1, (255,255,255))

WIN.blit(lost\_label, (WIDTH/2 - lost\_label.get\_width()/2, 350))

pygame.display.update()

*while* run:

clock.tick(FPS)

redraw\_window()

*if* lives <= 0 or player.health <= 0:

lost = True

lost\_count += 1

*if* lost:

*if* lost\_count > FPS \* 3:

run = False

*else*:

*continue*

*if* len(enemies) == 0:

level += 1

wave\_length += 5

*for* i *in* range(wave\_length):

enemy = Enemy(random.randrange(50, WIDTH-100), random.randrange(-1500, -100), random.choice(["red", "blue", "green"]))

enemies.append(enemy)

*for* event *in* pygame.event.get():

*if* event.type == pygame.QUIT:

quit()

keys = pygame.key.get\_pressed()

*if* keys[pygame.K\_a] and player.x - player\_vel > 0: *# left*

player.x -= player\_vel

*if* keys[pygame.K\_d] and player.x + player\_vel + player.get\_width() < WIDTH: *# right*

player.x += player\_vel

*if* keys[pygame.K\_w] and player.y - player\_vel > 0: *# up*

player.y -= player\_vel

*if* keys[pygame.K\_s] and player.y + player\_vel + player.get\_height() + 15 < HEIGHT: *# down*

player.y += player\_vel

*if* keys[pygame.K\_SPACE]:

player.shoot()

*for* enemy *in* enemies[:]:

enemy.move(enemy\_vel)

enemy.move\_lasers(laser\_vel, player)

*if* random.randrange(0, 2\*60) == 1:

enemy.shoot()

*if* collide(enemy, player):

player.health -= 10

enemies.remove(enemy)

*elif* enemy.y + enemy.get\_height() > HEIGHT:

lives -= 1

enemies.remove(enemy)

player.move\_lasers(-laser\_vel, enemies)

def main\_menu():

title\_font = pygame.font.SysFont("comicsans", 70)

run = True

*while* run:

WIN.blit(BG, (0,0))

title\_label = title\_font.render("Press the mouse to begin...", 1, (255,255,255))

WIN.blit(title\_label, (WIDTH/2 - title\_label.get\_width()/2, 350))

pygame.display.update()

*for* event *in* pygame.event.get():

*if* event.type == pygame.QUIT:

run = False

*if* event.type == pygame.MOUSEBUTTONDOWN:

main()

pygame.quit()

main\_menu()

page break

**Game VI : AI Plays Flappy Birds**

# **Problem Statement**

# **Instruction**

Generate the following components of the game:

* Basic Movement
* Jumping & Boundaries
* Character Animation
* Optimization & OOP
* Collision & Hitboxes
* Scoring
* Finishing Touches
* AI Configurations

# **Hint**

* Load the images
* Set & Update (X,Y) positions of the character
* Set Frames per Second
* Have a main game Loop
* Redraw the window every frame with the required surfaces.

# **Activity**

*"""*

*The classic game of flappy bird. Make with python*

*and pygame. Features pixel perfect collision using masks :o*

*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\_file):

*"""*

*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\_file)

*# 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)

page break

**Hello world with Turtles**

# **Activity**

page break