```
#This will be the code that I turn in for the create performance task
#This code will be similar to the original one that I have but will
#include functions and more organization
#
#this is still the in progress code, so there will be version floating
around
#for example as of right now I have a v1 that is updated to the drop
row()
# stopping at the bottom but the game not ending
#
# March 31, 2017
#

import pygame
import intersects
import math
import random

# Initialize game engine
////////////////////////////////////////////////////////
pygame.init()

# Open file
/////////////////////////////////////////////////////////////////////
file = open('highScore.txt', 'r+')
content = file.read()

# Window
////////////////////////////////////////////////////////////////////////
WIDTH = 835
HEIGHT = 800
SIZE = (WIDTH, HEIGHT)
TITLE = "Brick Breaker"
screen = pygame.display.set_mode(SIZE)
pygame.display.set_caption(TITLE)

# Timer
/////////////////////////////////////////////////////////////////////////
clock = pygame.time.Clock()
refresh_rate = 60

# Colors
////////////////////////////////////////////////////////////////////////
RED = (255, 4, 20)
WHITE = (255, 255, 255)
PINK = (254, 199, 204)
LIGHT = (232, 237, 223)
BLUE2 = (8, 65, 92)

BLACK = (0, 0, 0)
YELLOW = (255, 255, 0)
GREEN = (115, 255, 0)
BLUE = (0, 167, 225)
DARK_MINT = (0, 103, 111)
```

```
MINT2 = (0, 164, 176)
MINT3 = (0, 226, 244)
MINT4 = (113, 244, 255)
MINT5 = (134, 254, 255)
randco = (35, 54, 62)

# Font
///////////////////////////////////////////////////////////////////////////
/
font = pygame.font.Font(None, 48)
font2 = pygame.font.Font(None, 35)
font3 = pygame.font.Font(None, 70)
font4 = pygame.font.Font(None, 100)

# Stuff to set outside game loop
////////////////////////////////////////////////



speed = 7
score = 0
clicking = False
stage = 'delay'

''' draw the board '''
def draw_board():
    screen.fill(LIGHT)

    '''borders '''
    pygame.draw.rect(screen, BLUE2, [0, 0, WIDTH, 50])
    pygame.draw.rect(screen, BLUE2, [0, HEIGHT - 50, WIDTH, 50])
    pygame.draw.rect(screen, BLACK, [0, 50, WIDTH, 10])
    pygame.draw.rect(screen, BLACK, [0, HEIGHT - 60, WIDTH, 10])

    ''' score '''
    scoring = font.render("Score: " + str(score), 1, WHITE)
    screen.blit(scoring, [5, 7])

    ''' ball count '''
    ballCount = font.render("Ball Count: " + str(len(balls)), 1, WHITE)
    screen.blit(ballCount, [300, 7])

    ''' title '''
    name = font3.render("Click Brick Break", 1, WHITE)
    screen.blit(name, [200, HEIGHT - 50])

'''displays the game_over message'''
def game_over(blocks):
    pygame.draw.rect(screen, WHITE, [105, 100, 625, 600])
    pygame.draw.rect(screen, BLACK, [105, 100, 625, 600], 10)
    pygame.draw.rect(screen, RED, [120, 230, 585, 20])
    if len(blocks) > 0:
        lose = font4.render("GAME OVER", 1, BLACK)
        screen.blit(lose, [200, 150])
    finalScore = font3.render("FINAL SCORE: " + str(score), 1, BLACK)
    screen.blit(finalScore, [200, 280])
    finalCount = font3.render("FINAL BALL COUNT: " + str(len(balls)), 1,
BLACK)
    screen.blit(finalCount, [160, 360])
    img = pygame.image.load('finger_guns.jpg')
    screen.blit(img, [160, 425])



''' Intersects function '''
def intersects(rect1, rect2):
    left1 = rect1[0]
    right1 = rect1[0] + rect1[2]
    top1 = rect1[1]
    bottom1 = rect1[1] + rect1[3]

    left2 = rect2[0]
    right2 = rect2[0] + rect2[2]
    top2 = rect2[1]
    bottom2 = rect2[1] + rect2[3]

    return not (right1 <= left2 or
                left1 >= right2 or
                bottom1 <= top2 or
                top1 >= bottom2)

''' This allows for the next click to be made '''
def all_stopped(balls):

    for b in balls:
        if b.vx != 0 or b.vy != 0:
            return False

    return True

''' this removes the blocks when all of the hits have been made '''
def remove(blocks):

    to_remove = []

    for b in blocks:
        if b.hits <= 0:
            to_remove.append(b)


    for t in to_remove:
        blocks.remove(t)


def remove_powerup(powerups):

    to_remove = []
    for p in powerups:
        if p.hits <= 0:
            to_remove.append(p)

    for t in to_remove:
        powerups.remove(t)


''' this gets the ball slope from the mouse click '''
def get_vel(bx, by, mx, my, speed):
    a = mx - bx
    b = my - by
    c = math.sqrt((a**2) + (b**2))

    vx = int(speed) * (a/c)
    vy = int(speed) * (b/c)

    return vx, vy

''' drops a new row of blocks at the end of each turn '''
def get_new_row(blocks):

    b1 = Block(0, 100, 100, 35, int(score))
    b2 = Block(105, 100, 100, 35, int(score))
    b3 = Block(210, 100, 100, 35, int(score))
    b4 = Block(315, 100, 100, 35, int(score))
    b5 = Block(420, 100, 100, 35, int(score))
    b6 = Block(525, 100, 100, 35, int(score))
    b7 = Block(630, 100, 100, 35, int(score))
    b8 = Block(735, 100, 100, 35, int(score))

    row = [b1, b2, b4, b5, b7, b8]

    rlist = [ row[i] for i in random.sample(range(len(row)), 4) ]

    blocks.append(rlist[0])
    blocks.append(rlist[1])
    blocks.append(rlist[2])

def get_new_powerup(powerups):

    p3 = Powerup(210, 100, 100, 35, 1)
    p6 = Powerup(525, 100, 100, 35, 1)

    row =  [p3, p6]

    num = random.randint(0, 1)

    rlist = [ row[i] for i in random.sample(range(len(row)), num) ]

    if num > 0:
        powerups.append(rlist[0])
```

```
# Make a Player
////////////////////////////////////////////////////////////////

class Ball:

    def __init__(self, x, y, width, height, delay):

        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.delay = delay
        self.vx = 0
        self.vy = 0

    def get_rect(self):
        return [self.x, self.y, self.width, self.height]

    def update(self):
        ''' move ball in x direction '''
        if self.delay <= 0:
            self.x += self.vx

        ''' resolve x edge detection '''
        if self.x < 0:
            self.vx *= -1

        if self.x > WIDTH - self.width:
            self.vx *= -1

        ''' resolve x block collisions '''
        for b in blocks:
            if intersects(self.get_rect(), b.get_rect()):
                if self.vx > 0:
                    self.x = b.x - self.width
                else:
                    self.x = b.x + b.width
                self.vx *= -1
                b.hits -= 1

        ''' resolve x powerup collisions '''
        for p in powerups:
            if intersects(self.get_rect(), p.get_rect()):
                p.hits -= 1
                balls.append(Ball(WIDTH/2, 715, 25, 25, 50)) #come back
to solve powerup problem

        '''move ball in y direction '''
        if self.delay <= 0:
            self.y += self.vy

        ''' resolve y edge detection '''
        if self.y < 60:
                self.vy *= -1

        if self.y > HEIGHT - 85:
                self.vy  = 0
                self.vx  = 0
                self.y = HEIGHT - 85

        '''resolve y block collisions '''
        for b in blocks:
            if intersects(self.get_rect(), b.get_rect()):
                if self.vy > 0:
                    self.y = b.y - self.height
                else:
                    self.y = b.y + b.height
                self.vy *= -1
                b.hits -= 1

        '''resolve y powerup collisions '''
        '''for p in powerups:
            if intersects(self.get_rect(), p.get_rect()):
                p.hits -= 1
                balls.append(Ball(WIDTH/2, 715, 25, 25))'''


    def draw(self):
        pygame.draw.ellipse(screen, RED, [self.x, self.y, self.width,
self.height])


# Make Blocks
//////////////////////////////////////////////////////////////////

class Block:

    def __init__(self, x, y, width, height, hits):

        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.hits = hits

    def get_rect(self):
        return [self.x, self.y, self.width, self.height]


    def drop_row(self, blocks):
        if len(blocks) > 0:
            first = blocks[0].y
            for b in blocks:
                if b.y >= first:
                    first = b.y

            if first <= HEIGHT - 120:
                self.y += 40




    def draw(self):
        pygame.draw.rect(screen, BLUE, [self.x, self.y, self.width,
self.height])
        bhits = font2.render(str(self.hits), 1, WHITE)
        '''if self.hits < 10:
            screen.blit(bhits, [self.x + 45, self.y + 5])
        elif self.hits < 100:
            screen.blit(bhits, [self.x + 40, self.y + 5])
        elif self.hits >= 100:
            screen.blit(bhits, [self.x + 35, self.y + 5])'''


        linex = self.width / 2 - bhits.get_width() / 2

        screen.blit(bhits, [self.x + linex, self.y + 5])






# Make Power ups
class Powerup:

    def __init__(self, x, y, width, height, hits):

        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.hits = hits

    def get_rect(self):
        return[self.x, self.y, self.width, self.height]

    def drop_row(self, powerups):
        self.y += 40


    def draw(self):
        pygame.draw.rect(screen, GREEN, [self.x, self.y, self.width,
self.height])




# List of Blocks
///////////////////////////////////////////////////////////////
```

```
blocks = []

blocks.append(Block(0, 100, 100, 35, 1))
blocks.append(Block(105, 100, 100, 35, 1))
blocks.append(Block(210, 100, 100, 35, 1))
blocks.append(Block(315, 100, 100, 35, 1))
blocks.append(Block(420, 100, 100, 35, 1))
blocks.append(Block(525, 100, 100, 35, 1))
blocks.append(Block(630, 100, 100, 35, 1))
blocks.append(Block(735, 100, 100, 35, 1))




# List of Balls
////////////////////////////////////////////////////////////////

balls = []

balls.append(Ball(WIDTH/2, 715, 25, 25, 0))

# List of Power Ups

powerups = []



# Game Loop
////////////////////////////////////////////////////////////////////
done = False

while not done:
    # Event processing
/////////////////////////////////////////////////////////
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
        if stage == 'delay':
                if event.type == pygame.MOUSEBUTTONUP:
                    mx, my = pygame.mouse.get_pos()
                    count = 0
                    for b in balls:
                        b.vx, b.vy = get_vel(b.x, b.y, mx, my, speed)
                        b.delay = count
                        count += 5

                    print(mx, my)
                    stage = 'playing'
                    score += 1

    pressed = pygame.mouse.get_pressed()
    if pressed:
        m_x, m_y = pygame.mouse.get_pos()


    # Game Logic
////////////////////////////////////////////////////////////////

    showline = pressed[0]

    ''' move balls '''
    if stage == 'playing':
        for b in balls:
            if b.delay <= 0:
                b.update()
            b.delay -= 1



    remove(blocks)
    remove(powerups)

    if all_stopped(balls) == True:
        if stage == 'playing':
            for b in blocks:
                 b.drop_row(blocks)
            for p in powerups:
                p.drop_row(powerups)
            get_new_row(blocks)
            get_new_powerup(powerups)
            print(len(balls))
            begin = balls[0].x
            for b in balls:
                b.x = begin

            stage = 'delay'

            if len(blocks) > 0:
                first = blocks[0].y
                for b in blocks:
                    if b.y >= first:
                        first = b.y
                if first >= HEIGHT - 120:
                    stage = 'end'

            if len(powerups) > 0:
                for p in powerups:
                    if p.y >= HEIGHT - 120:
                        powerups.remove(p)
            '''else:
                stage = 'end'''
    # Drawing code
//////////////////////////////////////////////////////////////
    draw_board()

    for b in balls:
        b.draw()

    for b in blocks:
        b.draw()

    for p in powerups:
        p.draw()

    if showline:
        pygame.draw.line(screen, RED, [balls[0].x + 10, balls[0].y + 10],
[m_x, m_y,], 1)

    if stage == 'end':
        game_over(blocks)

    #update screen
////////////////////////////////////////////////////////////
    pygame.display.flip()

    # Limit refresh rate of game loop
//////////////////////////////////////////
    clock.tick(refresh_rate)

# Close window and quit
pygame.quit()
```