Digital Maker - Bouncy Ball Animation

Stage 0: Boilerplate Code

Importing Modules

```
import pygame
from pygame.locals import *
import sys
```

Here, we are simply loading in two modules, a module being some prewritten code that we want to make use of. The first module we are importing, `pygame`, gives us various graphics features which python does not natively support. We are also importing the `sys` module, which we will use for closing our graphics window.

```
pygame.init()

window_size = (1000, 1000)
display = pygame.display.set_mode(window_size)
```

Here, we initialise pygame and create a graphical display of a specified size (1000px by 1000px in my case).

```
import pygame
from pygame.locals import *
import sys

pygame.init()

window_size = (1000, 1000)
display = pygame.display.set_mode(window_size)

running = True
while running:

    # draw onto screen
```

```
    display.fill((20, 20, 20))
    pygame.display.flip() # display the frame
```

Stage 1: A Basic Bounce

```
import pygame
from pygame.locals import *
import sys

pygame.init()

window_size = (1000, 1000)
display = pygame.display.set_mode(window_size)

# define constants
g = 0.005
r = 30
ground = window_size[1]*0.9
elasticity = 1

# initialise variables
x = window_size[0] // 2
y = 50
v = 0

running = True
while running:

  # check for exit button press
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

  # update velocity
    if y > ground - r and v > 0:
        v *= -elasticity
    else :
        v += g

  # update y-pos
    y += v

  # draw onto screen
    display.fill((20, 20, 20))
    pygame.draw.circle(display, (255, 0, 0), (x, int(y)), r)
    pygame.draw.line(display, (255, 255, 255), (0, ground), (window_size[0], ground))
    pygame.display.flip() # display the frame
```

Stage 2: Control Frame Rate

Added lines are:

```
clock = pygame.time.Clock()

############################

# check whether ball will
# bounce back above ground
  if g * (y + r - ground) > 0.5 * v**2:
  # stop the animation at this point
    y = ground - r
    v = 0

############################

clock.tick(30) # limits fps
```

In the second code block, we are determining whether ΔGPE, the energy required for the ball to bounce above the surface (since it will usually rebound below) is greater than the ball's kinetic energy, E_k

In general, $\Delta GPE \approx mg\Delta h,$ and $E_k = \frac{1}{2}mv^2$

Therefore, we want to the animation to stop when
$mg\Delta h > \frac{1}{2}mv^2$

Dividing through by the (nonzero!) mass, we get

$g\Delta h > \frac{1}{2}v^2$

Since we want Δh to be the distance between the bottom of the ball and the ground, we end up with
$g \cdot (y + r - y_{ground}) > \frac{1}{2}v^2$

```
import pygame
from pygame.locals import *
import sys
```

```
pygame.init()

window_size = (1000, 1000)
display = pygame.display.set_mode(window_size)
clock = pygame.time.Clock()

# define constants
a = 1
r = 30
ground = window_size[1]*0.9
elasticity = 0.1

# initialise variables
x = window_size[0] // 2
y = 50
v = 0

running = True
while running:

  # check for exit button press
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

  # update velocity
    if y > ground - r and v > 0:
        v *= -elasticity
    else :
        v += a

  # update y-pos
    y += v

  # check whether ball will
  # bounce back above ground
    if g * (y + r - ground) > 0.5 * v**2:
    # stop the animation at this point
        y = ground - r
        v = 0

  # draw onto screen
    display.fill((20, 20, 20))
    pygame.draw.circle(display, (255, 0, 0), (x, int(y)), r)
    pygame.draw.line(display, (255, 255, 255), (0, ground), (window_size[0], ground))
    pygame.display.flip() # display the frame
    clock.tick(30) # limits fps
```

Stage 3: Encapsulating the Ball in a Class

```
import pygame
from pygame.locals import *
import sys
```

```
class Ball:

    def __init__(self, x, y, r, elasticity):
        self.x = x
        self.y = y
        self.r = r
        self.elasticity = elasticity
        self.v = 0

    def update_velocity(self, g, ground):
        if self.y > ground - self.r and self.v > 0:
            self.v *= -self.elasticity
        else :
            self.v += g

    def update_y_pos(self):
        self.y += self.v

    def check_energy(self, g, ground):
        if (self.y - (ground-self.r)) * g > 0.5 * self.v**2:
            # stop the animation at this point
            self.y = ground - self.r
            self.v = 0

    def update_dynamics(self, g, ground):
        self.update_velocity(g, ground)
        self.update_y_pos()
        self.check_energy(g, ground)

    def draw(self, surface):
        pygame.draw.circle(surface, (255, 0, 0), (self.x, int(self.y)), self.r)


pygame.init()

window_size = (1000, 1000)
display = pygame.display.set_mode(window_size)
clock = pygame.time.Clock()

# define constants
g = 1
ground = window_size[1]*0.9

# initialise balls
balls = []
n = 10 # number of balls
sep = window_size[0] / n # the seperation between balls
for i in range(n):
    x = int((i + 0.5) * sep)
    ball = Ball(x, 50 + 60 * i , 30, 1)
    balls.append(ball)


running = True
while running:

  # check for exit button press
```

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

    for ball in balls:
        ball.update_dynamics(g, ground)

  # draw onto screen
    display.fill((20, 20, 20))
    pygame.draw.line(display, (255, 255, 255), (0, ground), (window_size[0], ground))

    #draw balls
    for ball in balls:
        ball.draw(display)

    pygame.display.flip() # display the frame
    clock.tick(30) # limits fps
```

Stage 4: A Splash of Colour

```
import pygame
from pygame.locals import *
import sys

class Ball:

    def __init__(self, x, y, r, elasticity, colour):
        self.x = x
        self.y = y
        self.r = r
        self.elasticity = elasticity
        self.v = 0
        self.colour = colour

    def update_velocity(self, g, ground):
        if self.y > ground - self.r and self.v > 0:
            self.v *= -self.elasticity
        else :
            self.v += g

    def update_y_pos(self):
        self.y += self.v

    def check_energy(self, g, ground):
        if (self.y - (ground-self.r)) * g > 0.5 * self.v**2:
            # stop the animation at this point
            self.y = ground - self.r
            self.v = 0

    def update_dynamics(self, g, ground):
        self.update_velocity(g, ground)
        self.update_y_pos()
        self.check_energy(g, ground)
```

```
    def draw(self, surface):
        pygame.draw.circle(surface, self.colour, (self.x, int(self.y)), self.r)

def fromHSLA(h, s, l, a):
    '''returns a pygame.Color object with the specified HSLA vals'''
    colour = pygame.Color(0, 0, 0)
    colour.hsla = (h,s,l,a)
    return colour

pygame.init()

window_size = (1000, 1000)
display = pygame.display.set_mode(window_size)
clock = pygame.time.Clock()

# define constants
g = 1
ground = window_size[1]*0.9

# initialise balls
balls = []
n = 10 # number of balls
sep = window_size[0] / n # the seperation between balls
hue_increment = 360 / n
for i in range(n):
    colour = fromHSLA(i * hue_increment, 40, 50, 100)
    x = int((i + 0.5) * sep)
    ball = Ball(x, 50 + 60 * i , 30, 0.95, colour)
    balls.append(ball)


running = True
while running:

  # check for exit button press
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    for ball in balls:
        ball.update_dynamics(g, ground)

  # draw onto screen
    display.fill((20, 20, 20))
    pygame.draw.line(display, (255, 255, 255), (0, ground), (window_size[0], ground))

    #draw balls
    for ball in balls:
        ball.draw(display)

    pygame.display.flip() # display the frame
    clock.tick(30) # limits fps
```

Stage 5: Drawing Trails

Additions

```
def draw_trail(self, surface, n, k):
        N = min(n, len(self.past_positions))
        for i in range(0, N, k):
            position = self.past_positions[-N+i]
            colour = adjust_hsla(self.colour, 0, 0, 50/n*(N-i), 0)
            pygame.draw.circle(surface, colour, (position[0], position[1]), self.r)
```

```
def adjust_hsla(colour, h_pcent, s_pcent, l_pcent, a_pcent):
    pcent_changes = [h_pcent, s_pcent, l_pcent, a_pcent]
    [h, s, l, a] = [int(colour.hsla[i] * (1 + pcent_changes[i]/100)) for i in range(4)]
    # new_l = int(l * (1 + h_pcent/100))
    new_colour = pygame.Color(0, 0, 0)
    new_colour.hsla = (h % 360, s, l, a)
    return new_colour
```

```
import pygame
from pygame.locals import *
import sys
import numpy as np
import matplotlib.pyplot as plt
import pandas

class Ball:

    def __init__(self, x, y, r, elasticity, colour):
        self.x = x
        self.y = y
        self.r = r
        self.elasticity = elasticity
        self.v = 0
        self.colour = colour
        self.past_positions = [[int(x), int(y)]]

    def update_velocity(self, g, ground):
        if self.y > ground - self.r and self.v > 0:
            self.v *= -self.elasticity
        else :
            self.v += g

    def update_y_pos(self):
        self.y += self.v
        self.past_positions.append([int(self.x), int(self.y)])
```

```
    def check_energy(self, g, ground):
        if (self.y - (ground-self.r)) * g > 0.5 * self.v**2:
            # stop the animation at this point
            self.y = ground - self.r
            self.v = 0

    def update_dynamics(self, g, ground):
        self.update_velocity(g, ground)
        self.update_y_pos()
        self.check_energy(g, ground)

    def draw(self, surface):
        pygame.draw.circle(surface, self.colour, (self.x, int(self.y)), self.r)

    def draw_trail(self, surface, n, k):
        N = min(n, len(self.past_positions))
        for i in range(0, N, k):
            position = self.past_positions[-N+i]
            colour = adjust_hsla(self.colour, 0, 0, 50/n*(N-i), 0)
            pygame.draw.circle(surface, colour, (position[0], position[1]), self.r)


def fromHSLA(h, s, l, a)
    '''returns a pygame.Color object with the specified HSLA vals'''
    colour = pygame.Color(0, 0, 0)
    colour.hsla = (h,s,l,a)
    return colour

def adjust_hsla(colour, h_pcent, s_pcent, l_pcent, a_pcent):
    pcent_changes = [h_pcent, s_pcent, l_pcent, a_pcent]
    [h, s, l, a] = [int(colour.hsla[i] * (1 + pcent_changes[i]/100)) for i in range(4)]
    # new_l = int(l * (1 + h_pcent/100))
    new_colour = pygame.Color(0, 0, 0)
    new_colour.hsla = (h % 360, s, l, a)
    return new_colour

pygame.init()

window_size = (1920, 1080)
display = pygame.display.set_mode(window_size)
clock = pygame.time.Clock()

# define constants
g = 1
ground = window_size[1]*0.9

# initialise balls
balls = []
n = 25 # number of balls
sep = window_size[0] / n # the seperation between balls
hue_increment = 360 / n
for i in range(n):
    colour = fromHSLA(i * hue_increment, 40, 50, 100)
    x = int((i + 0.5) * sep)
    ball = Ball(x, 50 + 33 * i , 30, 1, colour)
    balls.append(ball)
```

```
running = True
while running:

  # check for exit button press
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    for ball in balls:
        ball.update_dynamics(g, ground)

  # draw onto screen
    display.fill((20, 20, 20))
    pygame.draw.line(display, (255, 255, 255), (0, ground), (window_size[0], ground))

    #draw balls
    for ball in balls:
        ball.draw_trail(display, 10, 2)
        ball.draw(display)

    pygame.display.flip() # display the frame
    clock.tick(30) # limits fps
```

Stage 6: Controlling Release Time

```
import pygame
from pygame.locals import *
import sys
import numpy as np
import matplotlib.pyplot as plt
import pandas

class Ball:

    def __init__(self, x, y, r, elasticity, colour):
        self.x = x
        self.y = y
        self.r = r
        self.elasticity = elasticity
        self.v = 0
        self.colour = colour
        self.past_positions = [[int(x), int(y)]]

    def update_velocity(self, g, ground):
        if self.y > ground - self.r and self.v > 0:
            self.v *= -self.elasticity
        else :
            self.v += g

    def update_y_pos(self):
        self.y += self.v
        self.past_positions.append([int(self.x), int(self.y)])
```

```
    def check_energy(self, ground):
        if (self.y - (ground-self.r)) * g > 0.5 * self.v**2:
            # stop the animation at this point
            self.y = ground - self.r
            self.v = 0

    def update_dynamics(self, g, ground):
        self.update_velocity(g, ground)
        self.update_y_pos()
        self.check_energy(ground)

    def draw(self, surface):
        pygame.draw.circle(surface, self.colour, (self.x, int(self.y)), self.r)

    def draw_trail(self, surface, n, k):
        N = min(n, len(self.past_positions))
        for i in range(0, N, k):
            position = self.past_positions[-N+i]
            colour = adjust_hsla(self.colour, 0, 0, 50/n*(N-i), 0)
            pygame.draw.circle(surface, colour, (position[0], position[1]), self.r)


def fromHSLA(h, s, l, a):
    '''returns a pygame.Color object with the specified HSLA vals'''
    colour = pygame.Color(0, 0, 0)
    colour.hsla = (h,s,l,a)
    return colour

def adjust_hsla(colour, h_pcent, s_pcent, l_pcent, a_pcent):
    pcent_changes = [h_pcent, s_pcent, l_pcent, a_pcent]
    [h, s, l, a] = [int(colour.hsla[i] * (1 + pcent_changes[i]/100)) for i in range(4)]
    # new_l = int(l * (1 + h_pcent/100))
    new_colour = pygame.Color(0, 0, 0)
    new_colour.hsla = (h % 360, s, l, a)
    return new_colour

pygame.init()

window_size = (1920, 1080)
display = pygame.display.set_mode(window_size)
clock = pygame.time.Clock()

# define constants
g = 1
ground = window_size[1]*0.9

# initialise balls
balls = []
n = 25 # number of balls
sep = window_size[0] / n # the seperation between balls
hue_increment = 360 / n
for i in range(n):
    colour = fromHSLA(i * hue_increment, 40, 50, 100)
    x = int((i + 0.5) * sep)
    ball = Ball(x, 50, 30, 1, colour)
    balls.append(ball)

release_pause = 100
```

```
start_time = pygame.time.get_ticks()
running = True
while running:

  # check for exit button press
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    released = (pygame.time.get_ticks() - start_time) // release_pause

    for ball in balls[:released]:
        ball.update_dynamics(g, ground)

  # draw onto screen
    display.fill((20, 20, 20))
    pygame.draw.line(display, (255, 255, 255), (0, ground), (window_size[0], ground))

    #draw balls
    for ball in balls:
        ball.draw_trail(display, 20, 2)
        ball.draw(display)

    pygame.display.flip() # display the frame
    clock.tick(30) # limits fps
```