Lab 1 (Setting up Pygame and Matplotlib)

Simulation and Modeling (CSCI 3010U)

Faisal Qureshi

Introduction

The goal of this lab is to set up Pygame and Matplotlib python packages on your machines, and get some experience with these packages.

Installation steps

You are asked to complete the following steps:

- Download and install Anaconda python distribution.
- Install pygame. It seems that Anaconda pygame installer is broken. You probably need pip to install pygame. Use the following command `pip install pygame`. Do ensure that `pip` command that you use belongs to the anaconda installation.
- Install matplotlib.

Programming

Check out the following code that simulates a ball falling under gravity. Friction due to air is missing. The simulation also collects position data and plots position vs. time once the simulation is completed.

1D ball in free-fall

```
import pygame, sys
import matplotlib.pyplot as plt
import numpy as np

# set up the colors
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)

# clock object that ensure that animation has the same
# on all machines, regardless of the actual machine speed.
```

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

def load_image(name):
    image = pygame.image.load(name)
    return image

class MyCircle(pygame.sprite.Sprite):
    def __init__(self, color, width, height):
        pygame.sprite.Sprite.__init__(self)

        self.image = pygame.Surface([width, height])
        self.rect = self.image.get_rect()
        self.image.fill(WHITE)
        cx = self.rect.centerx
        cy = self.rect.centery
        pygame.draw.circle(self.image, color, (width/2, height/2), cx, cy)
        self.rect = self.image.get_rect()

    def update(self):
        pass

class Simulation:
    def __init__(self):
        self.y = 0
        self.vy = 0
        self.mass = 0
        self.g = -9.8 # gravity acts downwards
        self.dt = 0.033 # 33 millisecond, which corresponds to 30 fps
        self.cur_time = 0

        self.paused = True # starting in paused mode

    def setup(self, y, vy, mass):
        self.y = y
        self.vy = vy
        self.mass = mass

        self.times = [self.cur_time*1000]
        self.positions = [self.y]

    def step(self):
        self.y += self.vy
        self.vy += self.mass * self.g * self.dt
        self.cur_time += self.dt

        self.times.append(self.cur_time * 1000)
        self.positions.append(self.y)

    def pause(self):
```

```
        self.paused = True

    def resume(self):
        self.paused = False

def sim_to_screen_y(win_height, y):
    '''flipping y, since we want our y to increase as we move up'''
    return win_height - y

def main():

    # initializing pygame
    pygame.init()

    # top left corner is (0,0) top right (640,0) bottom left (0,480)
    # and bottom right is (640,480).
    win_width = 640
    win_height = 480
    screen = pygame.display.set_mode((win_width, win_height))
    pygame.display.set_caption('1D Ball in Free Fall')

    # setting up a sprite group, which will be drawn on the
    # screen
    my_sprite = MyCircle(RED, 30, 30)
    my_group = pygame.sprite.Group(my_sprite)

    # setting up simulation
    sim = Simulation()
    sim.setup(460, 0, 1)

    print '--------------------------------'
    print 'Usage:'
    print 'Press (r) to start/resume simulation'
    print 'Press (p) to pause simulation'
    print 'Press (space) to step forward simulation when paused'
    print '--------------------------------'

    while True:
        # 30 fps
        clock.tick(30)

        # update sprite x, y position using values
        # returned from the simulation
        my_sprite.rect.x = win_width/2
        my_sprite.rect.y = sim_to_screen_y(win_height, sim.y)

        event = pygame.event.poll()
        if event.type == pygame.QUIT:
            pygame.quit()
```

```
            sys.exit(0)

        if event.type == pygame.KEYDOWN and event.key == pygame.K_p:
            sim.pause()
            continue
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_r:
            sim.resume()
            continue
        else:
            pass

        # clear the background, and draw the sprites
        screen.fill(WHITE)
        my_group.update()
        my_group.draw(screen)
        pygame.display.flip()

        if sim_to_screen_y(win_height, sim.y) > win_height:
            pygame.quit()
            break

        # update simulation
        if not sim.paused:
            sim.step()
        else:
            if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
                sim.step()

    # Lets move our lists to numpy array
    # first row contains times, second row contains positions
    pos_vs_times = np.vstack([sim.times, sim.positions])

    # Using matplotlib to plot simulation data
    plt.figure(1)
    plt.plot(pos_vs_times[0,:], pos_vs_times[1,:])
    plt.xlabel('Time (ms)')
    plt.ylabel('y position')
    plt.title('Height vs. Time')
    plt.show()

if __name__ == '__main__':
    main()
```

The code is available at lab-1.py

Tasks

You are asked to complete the following tasks:

- Modify the code so as to also plot Velocity vs. Time.

- Modify the code such that keystroke 'q' stop the simulation, and plots to result. *Note that simulation automatically stops when balls falls below a certain height.*
- (**Bonus**) Modify the code so as to save the results "Position vs. Time" and "Velocity vs. Time" to a file at the end of the simulation. The results will be saved when 'q' is pressed.
- (**Bonus**) Modify the code so as to load the results "Position vs. Time" and "Velocity vs. Time" from the file, and start the simulation from where you stopped it last time.

Submission

Via Blackboard.