The battle of balls

Final Report

2019

王皆宜
张瑜滢
朱迅泽
PAUL THIRIOT

The sky is the

CONTENT

Background

Background

The Battle of Balls

- developed by Superpop & Lollipop
- A kind of agile games
- The aim of the game:
  Avoid big balls and eat more small ones

Now We Have Made
Improvement to Realize
A New Way of Control

click here

Challenges

Challenges

We faced these difficulties:

1.Create a game interface

2.Defining the motion mode of the object and collisions

3.Understanding and adding the *Object Tracking* of *OpenCV*

4. Define the winning and losing conditions

Programming

Programming

Step 1:Import some libraries

Install five libraries:pygame, random, math, cv2, sys and copy.

```
import pygame
import random
import math
import cv2
import sys
import copy
```

Programming

Step 2:Some preparations for the game

Use RGB coordinates to select the colors of different objects.

```
colors_players = [(37,7,255),(35,183,253),(48,254,241),(19,79,251),(255,7,230),(255,7,23),(6,254,13)]
colors_cells = [(80,252,54),(36,244,255),(243,31,46),(4,39,243),(254,6,178),(255,211,7),(216,6,254),(145,255,7),(7,255,182),(255,6,86),(147,7,255)]
colors_viruses = [(66,254,71)]
```

Determine the size of the screen and create the first window of our game.

```
screen_width, screen_height = (1000,625)
surface = pygame.display.set_mode((screen_width,screen_height))
```

Programming

Step 2:Some preparations for the game

Draw a leaderboard.

```
t_surface = pygame.Surface((95,25),pygame.SRCALPHA) #transparent rect for score
t_lb_surface = pygame.Surface((155,200),pygame.SRCALPHA) #transparent rect for leaderboard
win_surface = pygame.Surface((1000,625),pygame.SRCALPHA) #transparent rect for WIN
lose_surface = pygame.Surface((1000,625),pygame.SRCALPHA) #transparent rect for lose
t_surface.fill((50,50,250,150))
t_lb_surface.fill((50,50,250))
lose_surface.fill((0,0,0))
win_surface.fill((0,250,0))
surface.fill((250,250,250))
```

Programming

Step 3:Track the object we choose to control

Set up the tracking process by taking a video through the computer camera

```
#set up tracking
def track_object(video_file):
    # set up tracker: https://docs.opencv.org/3.4.0/d0/d0a/classcv_1_1Tracker.html
    # tracker = cv2.TrackerMIL_create()
    tracker = cv2.TrackerBoosting_create()
    # read video
    video = cv2.VideoCapture(video_file)
    # exit if video not opened.
    if not video.isOpened():
        print("Could not open video")
        sys.exit()
    # read first frame
    ok, frame = video.read()
    frame = cv2.flip(frame, 1)
    if not ok:
        print("Cannot read video file")
        sys.exit()
    # define an initial bounding box
    bbox = cv2.selectROI(frame, False)
    # initialize tracker with first frame and bounding box
    ok = tracker.init(frame, bbox)
    initCam1=[ok, bbox, frame, tracker, video]
    cv2.destroyWindow("ROI selector")
    return initCam1
```

Programming

Step 4:Describe the objects

Description of the camera

```
class Camera:
    def __init__(self):
        self.x = 0
        self.y = 0
        self.width = screen_width
        self.height = screen_height
        self.zoom = 0.5

    def centre(self,blobOrPos):
        if(isinstance(blobOrPos,Player)):
            p = blobOrPos
            self.x = (p.startX-(p.x*self.zoom))-p.startX+((screen_width/2))
            self.y = (p.startY-(p.y*self.zoom))-p.startY+((screen_height/2))
        elif(type(blobOrPos) == tuple):
            self.x,self.y = blobOrPos
```

Programming

Step 4:Describe the objects

Description of the player

```
def draw(self,cam):
    col = self.color
    zoom = cam.zoom
    x = cam.x
    y = cam.y
    pygame.draw.circle(self.surface,(col[0]-int(col[0]/3),int(col[1]-col[1]/3),int(col[2]-col[2]/3)),(int(self.x*zoom+x),int(self.y*zoom+y)),int((self.mass/2+3)*zoom))
    pygame.draw.circle(self.surface,col,(int(self.x*cam.zoom+cam.x),int(self.y*cam.zoom+cam.y)),int(self.mass/2*zoom))
    if(len(self.name) > 0):
        fw, fh = font.size(self.name)
        drawText(self.name, (self.x*cam.zoom+cam.x-int(fw/2),self.y*cam.zoom+cam.y-int(fh/2)),(50,50,50))

def Playerxy(self):
    xy = [self.x, self.y]
    return xy
```

Programming

Step 4:Describe the objects

Description of cell

smaller ones:

```
class Cell:
    def __init__(self,surface):
        self.x = random.randint(20,1980)
        self.y = random.randint(20,1980)
        self.mass = 7
        self.surface = surface
        self.color = colors_cells[random.randint(0,len(colors_cells)-1)]
```

sufficient ones:

```
def spawn_cells(numOfCells):
    for i in range(numOfCells):
        cell = Cell(surface)
        cell_list.append(cell)
```

Programming

Step 4:Describe the objects

```
        elif (self.y > 2000):
            dX, dY = (400, 0)
            self.dX = dX
            self.dY = dY

        rotation = math.atan2(dY - (float(screen_height) / 2), dX - (float(screen_width) / 2)) * 180 / math.pi
        vx = (speed * (90 - math.fabs(rotation)) / 90)
        vy = (0)
        if (rotation < 0):
            vy = -speed + math.fabs(vx)
        else:
            vy = speed - math.fabs(vx)
        self.x += vx
        self.y += vy
    else:
        self.lastupdate = self.lastupdate + 1
        dX = self.dX
        dY = self.dY
elif getDistance((blob.x, blob.y), (self.x, self.y)) < 250:
    if self.mass > blob.mass:

        radians = math.atan2(blob.y - self.y, blob.x - self.x)
        distance = math.hypot(blob.x - self.x, blob.y - self.y) / speed
        distance = int(distance)

        vx = math.cos(radians) * speed
        vy = math.sin(radians) * speed

        self.x += vx
        self.y += vy
    else:
        radians = math.atan2(blob.y - self.y, blob.x - self.x)
        distance = math.hypot(blob.x - self.x, blob.y - self.y) / speed
        distance = int(distance)

        vx = math.cos(radians) * speed
        vy = math.sin(radians) * speed

        self.x -= vx
        self.y -= vy
else:
    dX, dY = (random.randrange(0, 801), random.randrange(0, 500))
    self.dX = dX
    self.dY = dY

    rotation = math.atan2(dY - (float(screen_height) / 2), dX - (float(screen_width) / 2)) * 180 / math.pi
    vx = (speed * (90 - math.fabs(rotation)) / 90)
    vy = (0)
    if (rotation < 0):
        vy = -speed + math.fabs(vx)
    else:
        vy = speed - math.fabs(vx)
    self.x += vx
    self.y += vy
```

Programming

Step 5:Manage the collisions

```
def collisionDetectionActors(actors_list):
    bb=0
    for i in range(len(actors_list)):
        if bb==1:
            break
        for j in range(len(actors_list)):
            if bb==1:
                break
            if i == j:
                pass
            else :
                if (getDistance((actors_list[i].x, actors_list[i].y), (actors_list[j].x, actors_list[j].y)) <= actors_list[i].mass / 2)
                    if  ((actors_list[i].mass)*1.10) > actors_list[j].mass:
                        actors_list[i].mass = actors_list[j].mass + actors_list[i].mass
                        actors_list.remove(actors_list[j])
                        actorsNameList.remove(actorsNameList[j])

                        bb=1
                        break

                    elif ((actors_list[i].mass) < (actors_list[j].mass*1.10)):
                        actors_list[j].mass = actors_list[j].mass + actors_list[i].mass
                        actors_list.remove(actors_list[i])
                        actorsNameList.remove(actorsNameList[i])

                        bb=1
                        break
```

Programming

Step 6:Define the win or lose

```
def winLose():
    if "You" not in actorsNameList:
        font = pygame.font.SysFont("comicsansms", 120)
        text = font.render("You Died", True, (250, 0, 0))
        lose_surface.blit(text, (screen_width/4, screen_height/3))
        surface.blit(pygame.transform.scale(lose_surface,(1000,650)),(0,0))
    elif len(actorsNameList) == 1 :
        font = pygame.font.SysFont("comicsansms", 50)
        text = font.render("Congratulations !", True, (0, 0, 0))
        win_surface.blit(text, (screen_width / 3.25, screen_height / 2.5))
        surface.blit(pygame.transform.scale(win_surface, (1000, 650)), (0, 0))
        print("win")
```

```
def draw_HUD():
    w,h = font.size("Score: "+str(int(blob.mass*2))+" ")
    surface.blit(pygame.transform.scale(t_surface,(w,h)),(8,screen_height-30))
    drawText("Score: " + str(int(blob.mass*2)),(10,screen_height-30))
    w, h = font.size("Ennemies in game: " +str((int((len(actors_list)-1))))+ " ")
    surface.blit(pygame.transform.scale(t_surface,(w,h)),(8,screen_height-60))
    drawText("Ennemies in game: " +str((int((len(actors_list)-1)))),(10,screen_height-60))
    dicoScore= {}
    orderedScore= []
    orderedName = []
    for actor in actors_list:
       dicoScore.update({actor.name:actor.mass})
    for key in dicoScore:
        orderedScore.append([key,dicoScore[key]])
    orderedScore.sort(key=takeSecond, reverse=True)
    for key in orderedScore:
        orderedName.append(str(key[0]))
    surface.blit(t_lb_surface, (screen_width - 160, 15))
    surface.blit(big_font.render("Leaderboard", 0, (0, 0, 0)), (screen_width - 157, 20))
    i=0
    for actor in orderedName:
        drawText((orderedName[i]), (screen_width - 157, 20 + 25*(i+1)))
        i += 1
```

Programming

Step 7:Update the game

```
def draw_HUD():
    w,h = font.size("Score: "+str(int(blob.mass*2))+" ")
    surface.blit(pygame.transform.scale(t_surface,(w,h)),(8,screen_height-30))

    drawText("Score: " + str(int(blob.mass*2)),(10,screen_height-30))

    w, h = font.size("Ennemies in game: " +str((int((len(actors_list)-1))))+ " ")
    surface.blit(pygame.transform.scale(t_surface,(w,h)),(8,screen_height-60))

    drawText("Ennemies in game: " +str((int((len(actors_list)-1)))),(10,screen_height-60))

    dicoScore= {}
    orderedScore= []
    orderedName = []
    for actor in actors_list:
       dicoScore.update({actor.name:actor.mass})
    for key in dicoScore:
        orderedScore.append([key,dicoScore[key]])
    orderedScore.sort(key=takeSecond, reverse=True)
    for key in orderedScore:
        orderedName.append(str(key[0]))


    surface.blit(t_lb_surface, (screen_width - 160, 15))
    surface.blit(big_font.render("Leaderboard", 0, (0, 0, 0)), (screen_width - 157, 20))
    i=0
    for actor in orderedName:
        drawText((orderedName[i]), (screen_width - 157, 20 + 25*(i+1)))
        i += 1
```

Programming

Step 8:Develop the main program

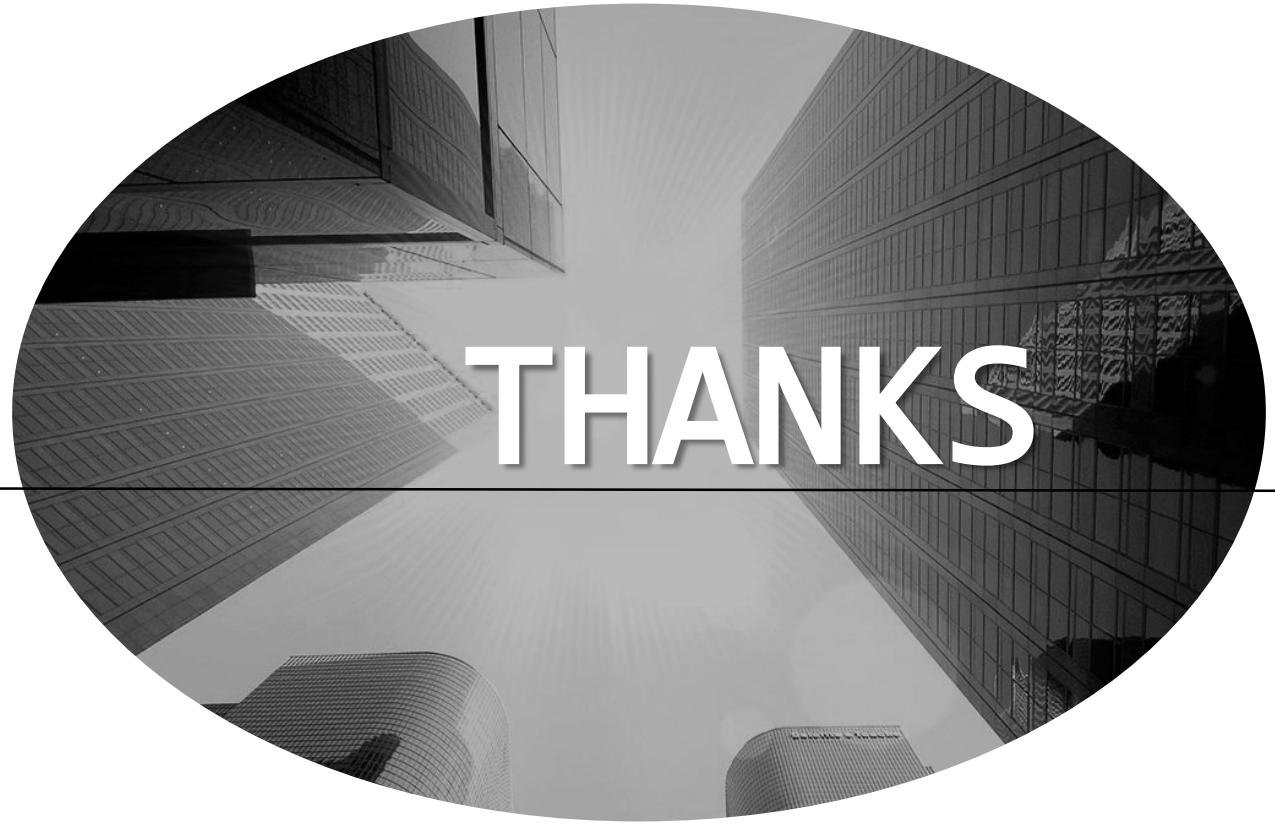
```
while(True):

    clock.tick(30)
    for e in pygame.event.get():
        if(e.type == pygame.KEYDOWN):
            if(e.key == pygame.K_ESCAPE):
                pygame.quit()
                quit()
        if(e.type == pygame.QUIT):
            pygame.quit()
            quit()
    roundTrack(setUp[0], setUp[1], setUp[2], setUp[3], setUp[4])
    for actor in actors_list:
        actor.update()
    camera.zoom = 100/(blob.mass)+0.3
    camera.centre(blob)
    surface.fill((150,150,150))
    draw_grid()
    for c in cell_list:
        c.draw(camera)
    for actor in actors_list:
        actor.draw(camera)
    draw_HUD()
    winLose()
    pygame.display.flip()
    collisionDetectionActors(actors_list)
    print(actorsNameList)
```

Further Improvements

Further Improvements

- Optimize the tracking process
- Add historic records
- Add some sound effects



THANKS