Report for ‘The battle of balls’

Team name: The Sky Is The Limit

Team members: 王皆宜 Wang Jieyi

张瑜滢 Zhang Yuying

朱迅泽 ZHU XUNZE

PAUL THIRIOT

Teacher & Mentor: Bao Yang

1st/06/2019

Contents

Abstract

Our game is inspired by the *Battles of Balls*, in which the ball can become bigger and bigger by eating smaller balls. We later improved the game by adding *Object Tracking*, namely, we use the movement of the object in the real world to control the movement of the ball in the game. The report will show the design of our game and the process of developing the game as well as some limitations and further works in detail.

Problem and Tasks Description

0.1 Background

Our project is adapted from a popular game *Battles of Balls* developed by *Superpop & Lollipop*. The game is quite simple, in which your ball can become bigger by eating smaller balls. We aim to recreate the classic game by adding *Object Tracking* of *OpenCV* library.

Fig0: Battle of Balls

Instead of applying traditional game controllers, like a mouse, we would like to adopt a more innovative controller to manipulate the ball. Entering the game, the user's image is displayed in the right corner of the game interface, and the user can choose any item in it by dragging a selected frame. Then the movement of the ball in the game will follow that of the selected item. If the ball the player is controlling is unfortunately eaten by a bigger ball, the game is over.

The *Object Tracking* is initially used to locate a moving object over time in a video. We will apply this technique into our game to trace the movement of the selected item in the camera.

0.2 Our tasks in this project

1. Create a game interface

We used *pygame* to draw circles to represent the balls in the game. In the interface, instructions and feedbacks can also be shown to players. Also, dyeing different balls in different colors is primarily necessary. For the game interface we used the code for Vilami available on the following address: https://github.com/Viliami/agario. We also improved the game interface adding a dynamic scoreboard and adding the display of numbers of enemies remaining.

2. Define the motion mode of the object and collisions.

In the whole game, there are several different movement methods of circles. We define several functions to describe their movement, like the random movement of the computer-controlled balls (so-called NPCs). We also define a few determining statements to judge whether the player's ball is covered by a larger one or a smaller ball is 'eaten' by it.

3. Understand and add the *Object Tracking* of *OpenCV*

While the original version of Battle of Balls is controlled by user's mouse, we are also inspired by another advanced technology that can locate a moving object over time (often applied by the police to track criminals). So, we decided to apply *Object Tracing* to our game to increase interaction between the game and the player.

4. Define the winning and losing conditions.

We detected if the player wins or loses the game. In case of victory or loss, we created screens to be displayed in case of victory or lose.

Chapter 1 Problem analysis and Basic Module

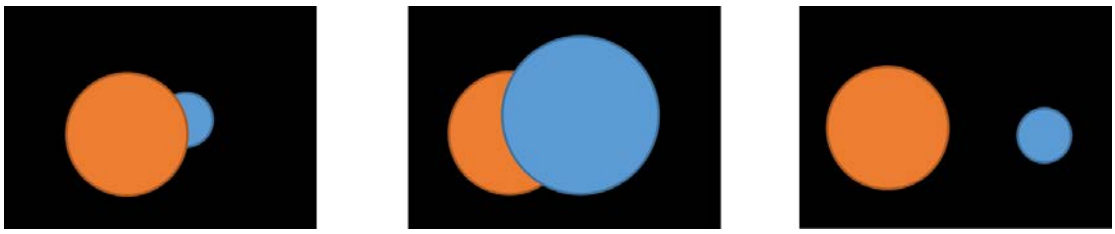
1.1 Problem Analysis

The tasks reveal that the project could be divided into three stages.

1. Define the functions needed

We first need to define and class some critical variables in the program. e.g. computer-controlled balls (random movement); the player-controlled ball; score and highest score

In addition, we apply some control flows. Actually the balls only have three conditions——being covered by a larger one, covering a smaller one or being separated with other balls. Therefore, whenever the player-controlled ball meets with other balls, the game executes a judgement to compare the distance between the balls and their radius.



covering a smaller one　　being covered by a larger one　　being separated with other balls

Fig1: Three conditions of Balls(the red ball represents the player)

2. Add the *Object Tracking* of *OpenCV*

Actually, in *pygame*, there are prestored functions to allow touch pad to control movement of the ball, but now in order to realize image control, we've learned a new module called *OpenCV*, by which we can get real-time image from the player and then it is stored in a variable.

3.Test the smooth operation of the game

As the leading-in of the player's image and tracking the selected item takes time, the operation of the game can be quite slow. So finally, we should test and develop it to ensure its smooth operation.

1.2 Module and theorem: how we learned them

1.2.1 *Pygame*

Pygame is a python programming language library for making multimedia programs, especially designing electronic games. We learned it by referring to the official documentations on its official website https://www.pygame.org/docs/.

Generally, we use several build-in functions in *pygame* to help us decorate the game's interface and provide the gamer a better playing experience. For example, we use 'init' to initialize the game engine, 'display' to creat the interface of our game, 'surface' to represent the leaderboard, 'font' to set the font, 'draw' to draw the cells, and etc.

1.2.2 *OpenCV*: Open Source Computer Vision Library

We used object tracking, so we turned to the demo codes provided by Mr Bao during class for help. Here are the initial codes we refer to:

```python
import cv2
import sys
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()
    if not ok:
        print("Cannot read video file")
        sys.exit()
    # define an initial bounding box
    # bbox = (287, 23, 86, 320)
    bbox = cv2.selectROI(frame, False)
    # initialize tracker with first frame and bounding box
    ok = tracker.init(frame, bbox)
    while True:
        # read a new frame
        ok, frame = video.read()
        if not ok:
            break
        # start timer
        timer = cv2.getTickCount()
        # update tracker
        ok, bbox = tracker.update(frame)
        # calculate Frames per second (FPS)
        fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer)
        # draw bounding box
        if ok:
            # tracking success
            p1 = (int(bbox[0]), int(bbox[1]))
            p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
            cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)
        else:
            # tracking failure
            cv2.putText(frame, "Tracking failure detected", (100, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)

        # display tracker type on frame
        cv2.putText(frame, " Tracker", (100, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

        # Display FPS on frame
        cv2.putText(frame, "FPS : " + str(int(fps)), (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

        # display result
        cv2.imshow("Tracking", frame)
```

However, the problem is that how we can draw the characteristic parameters in the tracking process to control the movement of the player in the game.

Chapter 2 Solution design

2.1 Import some libraries

Initially, we import six libraries, which are *pygame*, *random*, *math*, *cv2*, *sys* and *copy*.

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

2.2 Some preparations for the game

We use RGB coordinates to select the colors of different objects. In order to choose the most agreeable colors of various objects, we consulted the RGB color codes chart on https://www.rapidtables.com/web/color/RGB_Color.html.

```
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)]
```

We 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))
```

To provide the user a better game experience, we draw a leaderboard, actual of player and the number of ennemies remaining in the game.

```
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))
```

We make our group name as the window title.

```
pygame.display.set_caption("Agar.io The sky is the limit")
```

We give the game an initial situation.

```
cell_list = list()
actors_list= list()
clock = pygame.time.Clock()
actorsNameList=()
winLose=10
```

We set the font of our game. Considering that the font may not be available in some situations, we use the try except.

```
try:
    font = pygame.font.Font("Ubuntu-B.ttf",20)
    big_font = pygame.font.Font("Ubuntu-B.ttf",24)
except:
    print("Font file not found: Ubuntu-B.ttf")
    font = pygame.font.SysFont('Ubuntu',20,True)
    big_font = pygame.font.SysFont('Ubuntu',24,True)
    huge_font = pygame.font.SysFont('Ubuntu', 60, True)
```

2.3 Track the object we choose to control

We set up the tracking process by taking a video through the computer camera. The user should choose the tracking object by selecting the bounding box. The window will be destroyed once finishing the selecting process.

```
#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
```

We begin the tracking process. In order to tell the gamer that how sensitive he or she is controlling the object, we display the tracking speed by calculating the frames per second.

```
def roundTrack(ok, bbox, frame, tracker, video):
    # read a new frame
    ok, frame = video.read()
    frame = cv2.flip(frame, 1)
    # start timer
    timer = cv2.getTickCount()
    # update tracker
    ok, bbox = tracker.update(frame)
    # calculate Frames per second (FPS)
    fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer)
    # draw bounding box
    if ok:
        # tracking success
        p1 = (int(bbox[0]), int(bbox[1]))
        p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
        cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)
        centerM=(((int(bbox[0])+(int(bbox[0] + bbox[2])/2)),(int(bbox[1])+int(bbox[1] + bbox[3]))/2))
    else:
        # tracking failure
        cv2.putText(frame, "Tracking failure detected", (100, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)

    # display tracker type on frame
    cv2.putText(frame, " Tracker", (100, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

    # Display FPS on frame
    cv2.putText(frame, "FPS : " + str(int(fps)), (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50, 170, 50), 2);

    # display result
    cv2.imshow("Tracking", frame)

    # Exit if ESC pressed
    k = cv2.waitKey(1) & 0xff
```

2.4 Describe the objects

The program of our game is quite complicated, so we use class to group our attributes and methods. This enables us to elevate the level of thinking while designing the game.

2.4.1 Description of camera

We implement the movement of the object tracked to control the player, so we calculate the movement of the object tracked in real time situation.

```
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
```

We use the value of object movement in real life situation to make the ball in the game move the same value.

2.4.2 Description of player

We use one circle to represent the player and then give it an initial position and its initial mass. We use the function to update the attributes of the player.

```
class Player:
    def __init__(self,surface,name = ""):
        self.startX = self.x = 1000
        self.startY = self.y = 1000
        self.mass = 20
        self.surface = surface
        self.color = colors_players[random.randint(0,len(colors_players)-1)]
        self.name = name
        self.pieces = list()
        piece = Piece(surface,(self.x,self.y),self.color,self.mass,self.name)

    def update(self):
        self.move(setUp[0], setUp[1], setUp[2], setUp[3], setUp[4])
        self.collisionDetection()
```

We define the function collisionDetection to detect whether the player eats a cell and increase its mass.

```
def collisionDetection(self):
    for cell in cell_list:
        if(getDistance((cell.x,cell.y),(self.x,self.y)) <= self.mass/2):

            self.mass+=0.5
            cell_list.remove(cell)
```

We define the function move (in player class) to determine the movement of the players. We compute the center of the rectangle which correspond to the object tracked and then determine the directions of the move. We then compute the next position of the player.

```
def move(self,ok, bbox, frame, tracker, video):
    # read a new frame
    ok, frame = video.read()
    frame = cv2.flip(frame, 1)
    # start timer
    timer = cv2.getTickCount()
    # update tracker
    ok, bbox = tracker.update(frame)
    # calculate Frames per second (FPS)
    fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer)
    # draw bounding box
    # tracking success
    centerM = (((int(bbox[0]) + (int(bbox[0] + bbox[2]) / 2)), (int(bbox[1]) + int(bbox[1] + bbox[3])) / 2))
    dX,dY = centerM
    rotation = math.atan2(dY-(float(screen_height)/2),dX-(float(screen_width)/2))*180/math.pi
    speed = 8
    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
```

We define the draw function (in player class) to determine the movement of the ball in the game. We use the value computed in the function move to determine the movement of the balls in the game. We then compute the next position of the ball.

```
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
```

2.4.3 Description of cell

We use class cell to describe the attributes and methods of the relatively small ‘cells’. In order to win the game, the player should eat as many cells as possible to make itself grow bigger. Initially, we distribute and dye the cells randomly. We also give it an initial mass. To ensure that the player can eat the cells initially, we give the cells a relatively smaller mass than the player’s.

```
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)]
```

We define the function draw to display the cells as small circles to the player.

```
def draw(self,cam):
    pygame.draw.circle(self.surface,self.color,(int((self.x*cam.zoom+cam.x)),int(self.y*cam.zoom+cam.y)),int(self.mass*cam.zoom))
```

We define a function by using iteration to generate sufficient cells.

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

This function draws the gird of the game. The gird is adapted to the camera zoom.

```
def draw_grid():
    gaming_zone=[0,2001]
    for i in range(gaming_zone[0],gaming_zone[1],25):
        pygame.draw.line(surface,(200,200,200),(0+camera.x,i*camera.zoom+camera.y),(2001*camera.zoom+camera.x,i*camera.zoom+camera.y),3)
        pygame.draw.line(surface,(200,200,200),(i*camera.zoom+camera.x,0+camera.y),(i*camera.zoom+camera.x,2001*camera.zoom+camera.y),3)
```

2.4.4 Description of bot

We use class bot to describe the attributes and methods of the relatively big bots. The bots refer to the enemies in the game. In order to win the game, the player should avoid being eaten by them. We give the bots an initial position and dye the bots randomly. We also give it an initial mass. To ensure that the bots can eat the player initially, we give the bots a relatively bigger mass than the player's.

```
class bot():
    def __init__(self, surface, name="", xstart=0, ystart=0):
        self.startX = self.x = xstart
        self.startY = self.y = ystart
        self.mass = 32
        self.surface = surface
        self.color = colors_players[random.randint(0, len(colors_players) - 1)]
        self.name = name
        self.pieces = list()
        self.dX = 0
        self.dY = 0
        self.lastupdate = 0
        piece = Piece(surface, (self.x, self.y), self.color, self.mass, self.name)
```

We define the moving pattern of the bots. To ensure the bots stay in the limits of gaming area, we use the if statements to realize the process. If the bots goes outside the gaming area, the bots follow a defined pattern to come back in the area 0,0: 2000,2000. Else, if the distance between the player and the bots is inferior to 200, there are two possibilities. If the bot is bigger than the player, the bot will aim at move in the direction of the player in order to eat him. Otherwise, if the bot is smaller than the player, the bot will aim at fleeing in the opposite direction of the player to not be eat. Finally, if the bot is in the gaming are not close to the player, namely, superior to 200, the bot will adopt a random pattern.

```
def move(self):
    speed = 5
    if ((self.x < 0) or (self.x > 2000) or (self.y > 2000) or (self.y < 0)):
        if self.lastupdate > 150:
            self.lastupdate = 0
            if (self.x < 0) and (self.y < 0):
                dX, dY = (800, 500)
                self.dX = dX
                self.dY = dY
            elif (self.x < 0) and (self.y > 2000):
                dX, dY = (800, 0)
                self.dX = dX
                self.dY = dY
            elif (self.x > 2000) and (self.y < 0):
                dX, dY = (0, 500)
                self.dX = dX
                self.dY = dY
            elif (self.x > 2000) and (self.y > 2000):
                dX, dY = (0, 0)
                self.dX = dX
                self.dY = dY
            elif (self.x > 2000):
                dX, dY = (0, 250)
                self.dX = dX
                self.dY = dY
            elif (self.x < 0):
                dX, dY = (800, 250)
            elif (self.y < 0):
                dX, dY = (400, 500)
                self.dX = dX
                self.dY = dY
            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
```

2.5 Manage the collisions

We define the function collisionDetectionActors to detect whether the player will be eaten by the bots it encounters or otherwise it will eat the bots. It also determines if the bots eat each other. The actorsNameList and actors_list of existing stakeholders (player and bot) are also updated. The actorsNameList is used for defining the condition of victory and lose. The actors_list is used for Collision Detections, the display of current enemies in game and the display of leaderboard.

```
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
```

2.6 Define the win or lose

This is how we define the victory or loss. We use “actorsNameList” to store the remaining stakeholders in the game (player and bots). If the player is in the “actorsNameList”, -which means it has been eaten by a bot-, he lost the game. Else, if the length of list “actorsNameList” is equal to one, only the player remains in the game -all the bots have been eaten-, the player won the game.

```
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")
```

2.7 Update of the game

The draw_HUD function draws the scoreboard numbers of enemies remaining and score displayed. This function also ranks the player according to their score using the lists: orderedScore, orderedName and the dictionary dicoScore.

```
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
```

2.8 Main program: updating values and coordinating function

The part in the While loop is the main part of the program which manages the continuous execution of the program and the updating of the different values through the call of the different functions. If the key ESC is pressed, the game ends.

```
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)
```

Chapter 3 Limitations and further improvements

3.1 Limitations

We finally design the game successfully and the result seems to be perfect when the player plays the game. However, if the player looks inside of our game, our code seems to be a little mixed and long. We should learn to omit the part of codes that are not necessary, and give each line of the codes a remark. In further study, we will try to get command of the coding regulation of python and try our best to make our future codes more readable and tidy.

3.2 Further improvements

3.2.1 Optimize the tracking process

In our game, we are able to make the player in the game follow our movement of objects in most cases. However, when the color of the object we choose is close to the background color, the game might fail to track it. Therefore, for further improvement, we should think about how to optimize the tracking process and make it more sensitive.

3.2.2 Add historic records

In our game, to simplify the problem, we add some scores on the leaderboard initially instead of recording the historic records of the players. It may be even ideal for the game to actually record the scores of the previous players.

3.2.3 Add some sound effects

Generally, as we play games, music and sound effects are important parts of a game. This can be especially attractive to young kids. A matched music can provide the player a better experience of the game, and even remember it as a characteristic. For example, we may add some exciting background music to make the player immerse in the gaming process. Also, we can add some sound effects of congratulations for getting a highest score and some negative sounds for a low score.

Acknowledgement

First, we need to extend our gratitude to the websites like Github, CSDN, which provides us the community to find shared resources and allows to learn python better.

Next, we are sincerely grateful to Mr. Bao, who has introduced us to the amazing python. His tutorial lessons have helped a lot of this project.