Report on Circling Smile Animation

20B11806
Saw Kay Khine Oo

1. Goal

   The goal of this animation code is to make our smiley face (or anything created in the list, but in this report, I will refer to as smiley face) go circling round and round. In this program, I specifically made it to circle for 3 complete turns. After 3 complete turns, the smiley face along with any other digit should disappear diagonally from bottom right corner.

2. Thinking Process

   The smiley face will be assigned in a list named imgs and we will operate on the list to achieve the goal. Since I am new to creating animation, I cannot just create circling animation at once. So, firstly, I start thinking about how to make the smiley face go up, down, left and right. I will elaborate the code details in the following section. Then, I decide whether my smiley face will turn around in anticlockwise direction or clockwise direction. I choose anticlockwise direction. As the smiley face cannot move in curves, but only in straight line, I plan to make smiley face go in the anticlockwise direction following a rhombus route, that is, up right, up left, down left, and down right.

   Given that I have figured out each movement in anticlockwise direction, I need to think of a way to combine those movements and make them turn around continuously as many times as I want. Even though the smiley face moves in rhombus route, if the speed is fast enough, it will look like that the smiley face is turning around in circle. Lastly, after the smiley face has turned 3 complete rounds, the program is supposed to finish. However, since the digits will still be on the screen, I do not feel like it is finished. So, I want to make all the digits disappear diagonally as a proper exiting function. To make all the digits disappear diagonally from bottom right corner, one digit must disappear from bottom line in the first loop and 2 in the second loop and so on. Moreover, I need to make sure that the digits will disappear completely and not be left as 0 or 1.

3. Code details

   3.1 First part of thinking process

      3.1.1 Moving up

      The number i line is going to be replaced by number i+1 line. We will keep repeating this until the last line. However, to keep the simplicity, we set the smiley face in a way that the first and the last lines will be the same, so, when moving up, the last line will remain the same.

```
for i in range(len(imgs)):
  if i<len(imgs)-1:
    imgs[i] = imgs[i+1]
```

3.1.2 Moving down

The number i+1 line is going to be replaced by number i line. The last line will still remain unaffected. I also have to assign the first line with the value of the last line to keep it unaffected.

```
for i in range(len(imgs)):
  if i>1:
    imgs[len(imgs)-i]=imgs[len(imgs)-i-1]
imgs[0]=imgs[len(imgs)-1]
```

3.1.3 Moving left

In order to move to the left side, constant base value will be necessary. The base will be assigned the value with the first digit 1 and the remaining digits 0. If we take modulo operation of every line with the base, we will get the result with the first non-zero digit in the line (excluding the first digit 1 of each line). Multiplying that result with 10 will shift the result to left after adding with the base again.

```
imgs[i] = ((imgs[i] % base)*10)+base
```

It is possible to replace modulo operation with basic operations. However, since modulo operation is used in a complex formula as provided, replacing modulo operation with basic operations would only make it complex. As for how to replace it, details will be provided in section 3.3 Combinations.

3.1.4 Moving right

Moving to the right is similar to moving to the left. The only difference is that after taking the modulo operation, the result will be divided by 10 instead of multiplication so that it will shift to the right.

```
imgs[i] = ((imgs[i] % base)//10)+base
```

3.2 Second part of thinking process

3.2.1 Moving up right

After figuring out how to move up, down, left, right, diagonal movements are possible by combining any of those 2.

```
for i in range(len(imgs)):
   if i<len(imgs)-1:
      imgs[i] = ((imgs[i+1]%base)//10)+base
```

3.2.2 Moving up left

The process of moving up left is similar to that of moving up right.

```
for i in range(len(imgs)):
    if i<len(imgs)-1:
        imgs[i] = ((imgs[i+1]*base)//10)+base
```

3.2.3 Moving down left

The smiley face will move down left if we combine the process of moving down and moving left.

```
for i in range(len(imgs)):
  if i>1:
    imgs[len(imgs)-i]=((imgs[len(imgs)-i-1]%base)*10)+base
imgs[0]=imgs[len(imgs)-1]
```

3.2.4 Moving down right

The process of moving down right is similar to that of moving down left.

```
for i in range(len(imgs)):
  if i>1:
    imgs[len(imgs)-i]=((imgs[len(imgs)-i-1]%base)//10)+base
imgs[0]=imgs[len(imgs)-1]
```

3.3 Combinations

Since there are 4 different animations, a special method is required to combine them. The count *t* will increase by one after every loop. One animation will display in one count. However, we need 4 animations to display a full round. We use *if* statements and modulo operation to determine which animation should be displayed in that loop. In this way when the count t is 12 times, the smiley face will make 3 complete rounds.

```
if t%4== 1: display up right
if t%4== 2: display up left
if t%4== 3: display down left
if t%4== 0: display down right
```

Using modulo operation makes the code compact and easy to understand. We can obtain the same result by the following code without using modulo operation.

```
r = t
while r>=4:
    r -= 4
if r == 1:
if r == 2:
if r == 3:
if r == 0:
```

3.4 Exiting

Since the digits form a rectangular box of digits, I need to take care that the bottom line will become zero first while the other lines still have a lot of digits left. To solve this, I will assign the lines in the list with the value of an empty space “” when the line has only zero left. Then, as long as the value of any line is not equal to the empty space “”, that line will be divided by 10 making the last digit disappear. However, since this exiting function is supposed to make the digits disappear diagonally, when the count is less than the length of the list, the number of digits to disappear depends on the count. For example, first time, only one digit disappears from last line. Second time, one digit from last line and one digit from the line above last line disappear, and so on.

```
if k<= len(imgs):
  for i in range(k):
    if imgs[len(imgs)-i-1]!= "":
      imgs[len(imgs)-i-1] = imgs[len(imgs)-i-1] //10
    if imgs[len(imgs)-i-1] == 0:
      imgs[len(imgs)-i-1] = ""

if k> len(imgs):
  for i in range(len(imgs)):
    if imgs[len(imgs)-i-1]!= "":
      imgs[len(imgs)-i-1] = imgs[len(imgs)-i-1] //10
    if imgs[len(imgs)-i-1] == 0:
      imgs[len(imgs)-i-1] = ""
```

4. Points for improvement

As I mentioned before, in order for this program to work successfully, the smiley face cannot be in the last line. Improvements can be made so that such restrictions won’t be necessary. Furthermore, the circle is so small that it doesn’t actually look like it’s circling. So, a larger scale, that is, divided by 100 or 1000 instead of 10, would be desirable for circling.