Turtle Advantage
(공 피하기 게임)

과목명	파이선 프로그래밍
담당교수	김동근 교수님
학과	컴퓨터공학부 컴퓨터공학전공
학번	201401932
이름	노 명 환

목 차

Ⅰ. 서론

이번 Turtle Graphic을 활용하여 프로그램을 만드는 것에 있어서 가장 기본적인 부분은 그래픽이라는 생각이 들었다. 기존의 프로그램들이 간단하게, Console 혹은 IDLE 내에서 작동하는 프로그램이었다면, turtle을 활용한 프로그램은 빈 도화지에 자신이 그림을 직접 그려나가는 과정이다. 이러한 그래픽 부분은 이전까지 접해보지 못하였던 부분으로, 누구나 즐길 수 있는 게임을 만들어보자는 생각이 들었다.

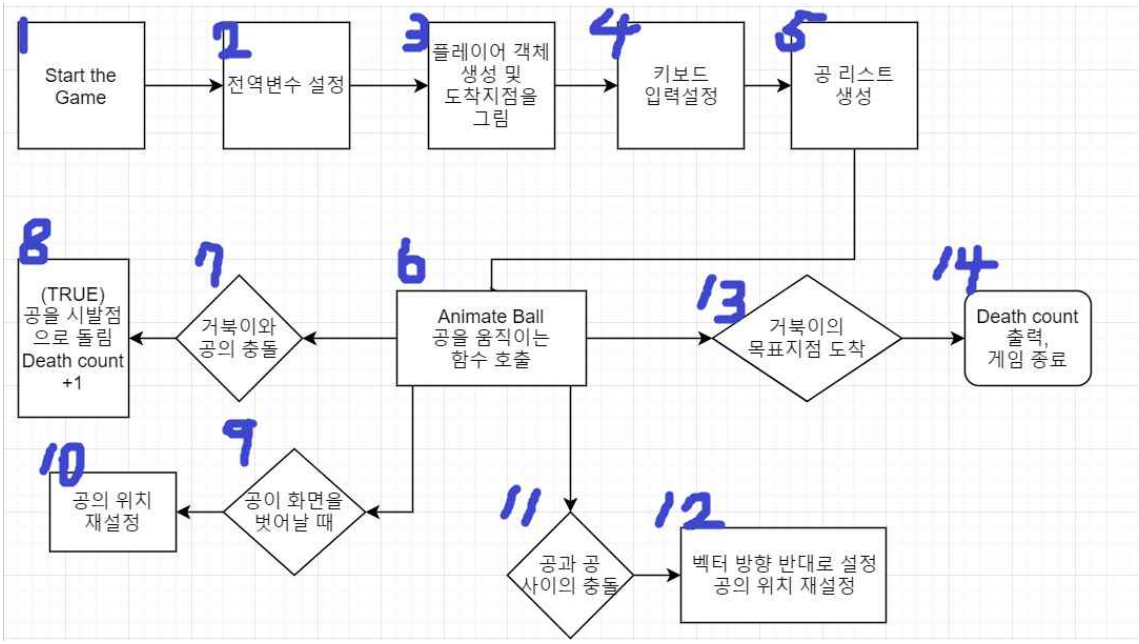
이번에 구현한 Turtle Advantage는 말 그대로 거북이의 모험이라는 뜻으로, 거북이가 목적지까지 장애물을 피하여 이동하는 게임이다. 움직이는 거북이를 표현하기 위하여 독립적인 터틀 객체를 만들었으며, 장애물은 공들의 움직임으로 표현하였고 이들도 하나의 터틀 객체로 표현하였다.

게임이 시작하면 거북이는 자신의 도착점을 그린 후에 출발점으로 이동한다. 이후, 공이 랜덤하게 생성되며 이 공들은 벡터 값을 가지고 있으며, 이동을 하게 된다. 이 과정에서, 공이 맵 범위 밖으로 벗어날 수 있으므로, 처음에 설정해놓은 맵의 범위를 이용하여 맵의 범위와 공의 위치를 서로 비교하여, 공이 맵 밖으로 벗어날 수 없도록 하였다. 또한, 공과 공 사이의 충돌에서는 충돌 시 서로의 벡터 값에 -1을 곱하여 방향을 반대로 움직이도록 설정하였다. 추가로, 범위가 많이 겹쳐지는 경우 같은 자리에서 맴도는 상황이 발생하여 이를 예방하기 위하여 충돌 즉시 서로의 위치 값을 수정하여 겹쳐지는 경우를 방지하였다.

거북이의 이동은 키보드를 입력받아 수행하며, 각각의 입력에 따른 함수를 정의하였다. 정의한 키보드 입력은 onkey함수를 이용하여 활성화 시켰으며, 이로 인하여 거북이의 움직임을 구현하였다. 공과 마찬가지로, 거북이 또한 맵 밖으로 벗어나지 않도록 각 키보드 입력 시마다 맵의 좌표와 비교하여 맵을 벗어날 경우 맵의 끝 좌표로 다시 위치하도록 설계하였다. 거북이가 이동 중, 공을 만나면 Death count가 1 증가하며, 원점으로 돌아가도록 하였다.

마지막으로, 거북이가 최초에 그려놓은 네모 칸에 도착하면 좌표 검사를 통하여 도착 지점에 도착하였는지 검사를 실행한다. 만약 도착 지점에 도착한 것이 맞다면, Death count를 출력하며 게임의 상태가 False로 변경되어 게임이 끝난다.

Ⅱ. 본론



1. Start the Game

프로그램에 진입

2. 전역변수 설정

게임에 필요한 전역변수를 설정한다. Turtle Graphic의 좌표계, 공의 크기, 목표 지점의 좌표, 키보드 입력 시 움직일 거리를 초기화 시켜준다.

3. 플레이어 객체 생성 및 도착지점을 그림

플레이어 객체를 Turtle Graphic으로 새로 생성하고, 이를 사용하여 도착지점을 그린다.

4. 키보드 입력설정

키보드 이벤트를 처리 할 함수를 정의하고 onkey함수를 통하여 키 입력을 활성화 시킨다.

5. 공 리스트 생성

전역 변수에서 설정한 값을 넘겨주어 공을 생성하고 리스트 안에 넣어준다.

6. 공을 움직이는 함수 호출 (Animate Ball)

공 리스트에 넣은 공들을 움직이는 함수를 호출한다. 함수 내에서 거북이는 key 이벤트를 통하여 이동을 할 수 있다.

7. 거북이와 공의 충돌

거북이의 좌표와 공의 좌표를 미리 정의해놓은 거리 계산함수를 통하여 서로 겹쳐지는지 확인한다.

8. if(True) 공을 시발점으로 되돌림, Death count 증가

공이 만약 거북이와 만난다면, 거북이를 시발점으로 되돌리고 Death count를 1 증가시킨다.

9. 공이 화면을 벗어날 때

공이 처음에 설정해놓은 전역변수 map의 x, y 좌표 밖으로 벗어나는지 체크한다.

10. 공의 위치 재설정

공이 화면 밖으로 벗어났으면, 방향 벡터에 -1을 곱하여 진행방향을 반대로 만들어 다시 화면 안쪽에서 운동을 할 수 있도록 만든다.

11. 공과 공 사이의 충돌

공과 공 사이의 충돌이 일어나는지 검사한다. 공과 공 사이의 충돌은 서로의 중심좌표를 확인하여 만약 중심좌표 사이의 거리가 두 원의 반지름의 합보다 작다면 충돌한 것이다.

12. 벡터 방향 반대로 설정 및 공의 위치 재설정

공 사이에 충돌이 일어났다면 서로의 방향 벡터에 -1을 곱하여 진행방향을 바꾸어주도록 한다. 추가로, 겹치는 영역이 클 경우, 탈출을 하지 못하고 같은 자리에서 움직이는 경우를 생각하여 현재 위치에서 벡터 값*2를 더하여 서로 떨어질 수 있도록 설정하여준다.

13. 거북이의 목표지점 도착

거북이가 처음에 설정한 목표지점에 도착하였는지 체크한다.

14. Death count 출력 및 게임 종료

만약, 거북이가 목표지점에 도착하였다면 도착할 때까지의 Death count를 출력시키고 게임의 상태를 False로 설정하여 게임을 종료시킨다.

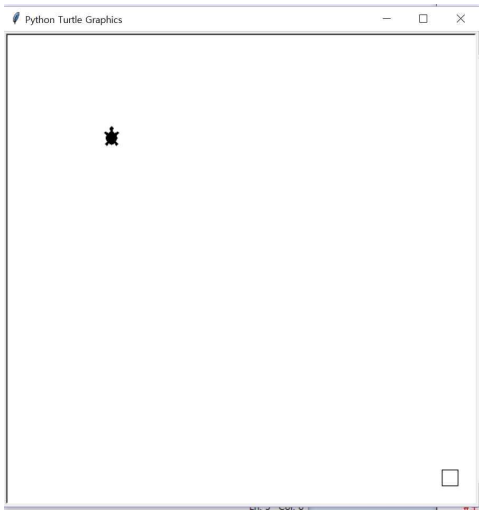
Ⅲ. 실험결과

*프로그램 작성환경

OS : Microsoft Windows 10 Home

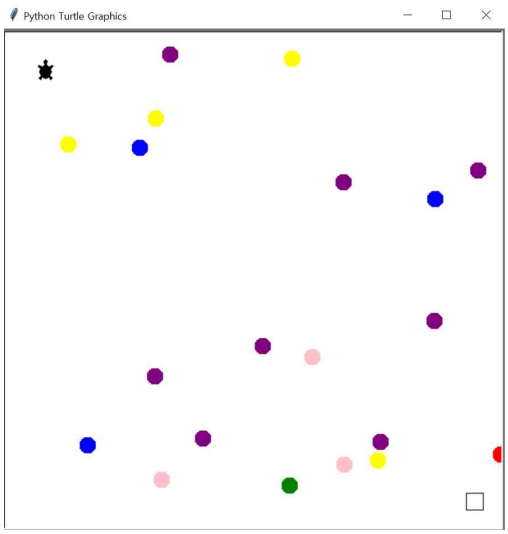
사용 언어 : Python 3.6.2

사용 패키지 : turtle, random



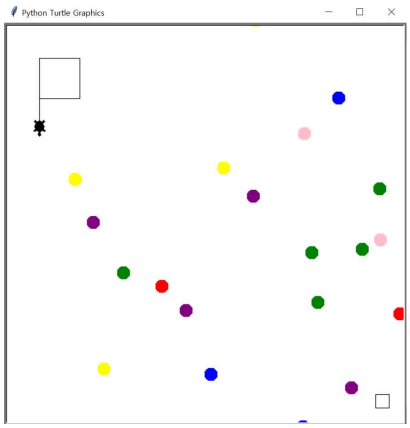
1. 게임 시작 시

터틀 객체를 생성하고 도착지점(우하단 네모)을 그리고 시발점으로 이동



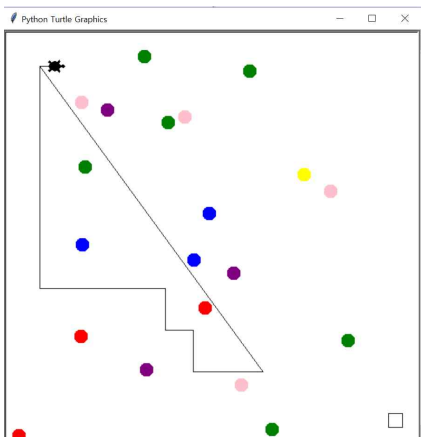
2. 공 생성

공을 생성하고 공 리스트의 볼을 움직임



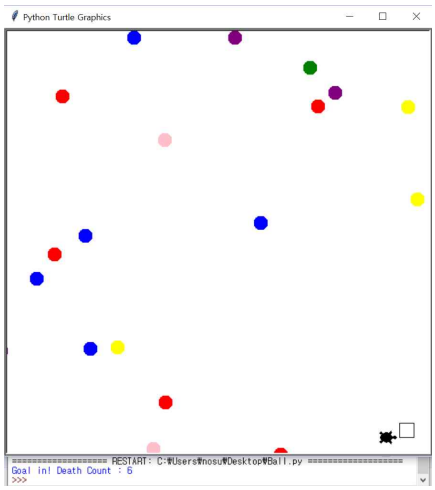
3. 키보드 입력을 통한 이동

키보드 입력을 통하여 거북이가 이동한다. (사진에서는 down()함수를 이용하여 이동로 표시)



4. 공과 충돌 시 시발점으로 복귀

거북이와 공 사이의 거리를 계산하여 서로 만났을 경우, 거북이를 원점으로 위치시킨다.



5. 목표지점 도착

목표지점 도착 시 게임을 정지시키고 Death count를 출력한다.

Ⅳ. 결론

이번 파이썬 Turtle Graphic을 이용하여 게임을 구현하면서 가장 크게 다가온 부분은 언어의 유연성이었다. 파이썬이라는 언어 자체가 범용성이 뛰어나고 필요한 거의 모든 함수들을 이미 구현해놓았기 때문에 소스 코드를 작성함에 있어서 굉장히 편리함을 느꼈다. 함수를 사용하는 방법도 간단하였고, 무엇보다 직관적인 부분이 크게 작용하여 유연하게 소스 코드를 작성할 수 있었던 것 같다.

사실 이번 파이썬 프로젝트에 있어서 가장 먼저 떠올랐던 아이디어는 갈릴레이의 사고실험이었다. 마찰력이 작용하지 않는 공간 내에서 공을 굴렸을 때, 공은 다시 출발했던 높이만큼 올라오며, 만약 경사가 없이 직선로라면 공은 무한하게 운동을 계속한다는 이론이다. 이를 위하여, 삼각함수를 활용하여 그래프를 그리고 마찰력까지 구현하는 것에 성공하였으나, 가장 큰 문제점은 가속도를 표현할 방법을 모른다는 것이었다. 파이선 내에서 공이 이동하는 속도를 설정하기 위하여 다양한 방법을 시도해보았지만, 결국 가속도를 표현하지 못하여 중간에 포기하였으나 나중에 기회가 된다면 다시 한 번 만들어보고 싶다는 생각이 들었다.

그래픽이라는 분야는 나에게 있어 상당히 생소한 분야이다. 미술에 일가견이 없는 부분이 일단 가장 크고, 시각적인 부분에 많이 약한 나에게 있어 그래픽을 활용한 프로그래밍은 어려울 것이라는 생각이 많이 들었다. 하지만, 이번 기회를 통하여 직접 프로그램을 작성하며 이 부분을 많이 극복하고 실력도 많이 증진되는 기회가 된 것 같다. 파이썬의 가장 큰 장점을 이 부분에서 많이 느끼게 되었는데, 패키지를 활용하여 숙련되지 못 한 부분도 많이 보완할 수 있다는 것을 느꼈다. 앞으로는 다양한 패키지를 통하여 다양한 분야에 대하여 공부해보고 싶다는 생각이 들었다.

Ⅴ. 참고자료

쉽게 배우는 파이썬 프로그래밍 (가메출판사, 김동근 저)

원 충돌 알고리즘
(https://m.blog.naver.com/PostView.nhn?blogId=winterwolfs&logNo=10165506488&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F)

Ⅵ. 부록 (소스 코드)

```
from turtle import *
import random

def key_left():
    my.seth(180)
    my.fd(step)
    if(my.xcor()<-300):
        my.setx(-300)

def key_right():
    my.seth(0)
    my.fd(step)
    if(my.xcor()>300):
        my.setx(300)

def key_up():
    my.seth(90)
    my.fd(step)
    if(my.ycor()>300):
        my.sety(300)

def key_down():
    my.seth(270)
    my.fd(step)
    if(my.ycor()<-300):
        my.sety(-300)


def make_ball(size):
    x = random.randint(-200, max_x)
    y = random.randint(min_y, 200)
    color = random.choice(["red", "green", "blue", "yellow", "pink", "purple"])
    while True:
        dx = random.randint(-1, 1)  # 이동벡터
        dy = random.randint(-1, 1)  # 이동벡터
        if dx != 0 or dy != 0:
            break
    return [x, y, size, color, dx, dy]

#거리계산 함수
def distance(x1, y1, x2, y2):
    return ((x2-x1)**2+(y2-y1)**2)**0.5
```

```
#움직이는 공
def animate_ball(pen, ball_list):
    tracer(False)
    game=True
    death=0
    while game:
        listen()
        pen.clear()  # 공을 지움
        for i in range( len(ball_list) ): # 모든 공을 움직임
            ball_x, ball_y, size, color, dx, dy =  ball_list[i]
            ball_x += dx
            ball_y += dy
            ball_list[i][0] = ball_x
            ball_list[i][1] = ball_y
            #거북이와 공의 충돌
            if(distance(my.xcor(), my.ycor(), ball_x, ball_y)<=15):
                my.setpos(-goal, goal)
                death = death+1
            #화면을 벗어날 때
            if (ball_x<min_x) or (ball_x > max_x): ball_list[i][4] *= -1
            if (ball_y<min_y) or (ball_y > max_y): ball_list[i][5] *= -1
            #공과 공의 충돌 판정
            for j in range(len(ball_list)) :
                if(not i==j):
                    ball2_x=ball_list[j][0]
                    ball2_y=ball_list[j][1]
                    if(distance(ball_x, ball_y, ball2_x, ball2_y)<=ball_size):
                        ball_list[i][4] *= -1
                        ball_list[i][5] *= -1
                        ball_list[j][4] *= -1
                        ball_list[j][5] *= -1
                        ball_list[i][0] += ball_list[i][4]*2
                        ball_list[i][1] += ball_list[i][5]*2
                        ball_list[j][0] += ball_list[j][4]*2
                        ball_list[j][1] += ball_list[j][5]*2
            pen.setpos(ball_x, ball_y)
            pen.dot(size, color)
        #목표지점 도착
        if(goal<=my.xcor()<=goal+20 and -goal-20<=my.ycor()<=-goal):
                game=False
                return death
        update() # 스크린 갱신

def main():
    global min_x, max_x, min_y, max_y, ball_size, goal, step
    max_x = 300; min_x=  -max_x
```

```
max_y = 300; min_y = -max_y
ball_size=20
goal=250
step=20
setup(max_x*2, max_y*2)

global my
my=Turtle()
my.up()
#도착지점을 그림
my.setpos(goal, -goal)
my.down()
my.seth(0)
my.fd(20)
my.seth(270)
my.fd(20)
my.seth(180)
my.fd(20)
my.seth(90)
my.fd(20)
my.up()

#기본 설정
my.shape("turtle")
my.setpos(-goal,goal)
onkey(key_right, "Right")
onkey(key_left, "Left")
onkey(key_up, "Up")
onkey(key_down, "Down")

my_pen = Turtle()
my_pen.up()
my_pen.ht()
ball_list = []
#공 생성
for i in range(20):
    ball = make_ball(ball_size)
    ball_list.append(ball)
deathCount=animate_ball(my_pen, ball_list) # 에니메이션
print("Goal in! Death Count : %d"%(deathCount))
```