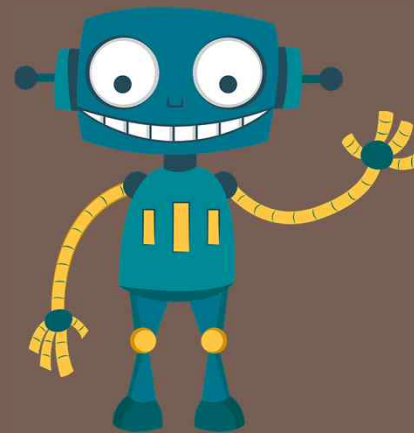


파이썬 익스프레스



13장 파이썬을 이용한 게임 작성

Q & A

- 언제라도 질문하세요
 1. 강의시간의 채팅창 (수신인: 모두, 수강생모두에게 답변하기에)
 2. 네이버카페 질의응답게시판

- 강사의 1번 선생님은 여러분의 질문

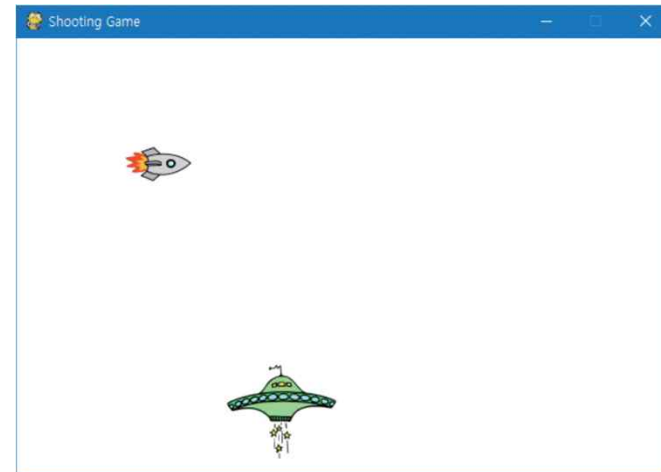
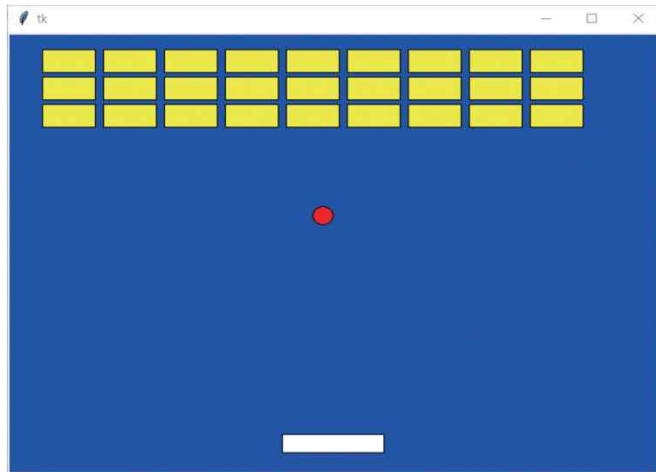


아으로 나는 할 수 있다

1. 나는 `tkinter`를 이용하여 벽돌깨기 게임을 작성할 수 있다
2. 나는 `pygame`을 이용하여 외계 우주선을 피하는 게임을 작성할 수 있다

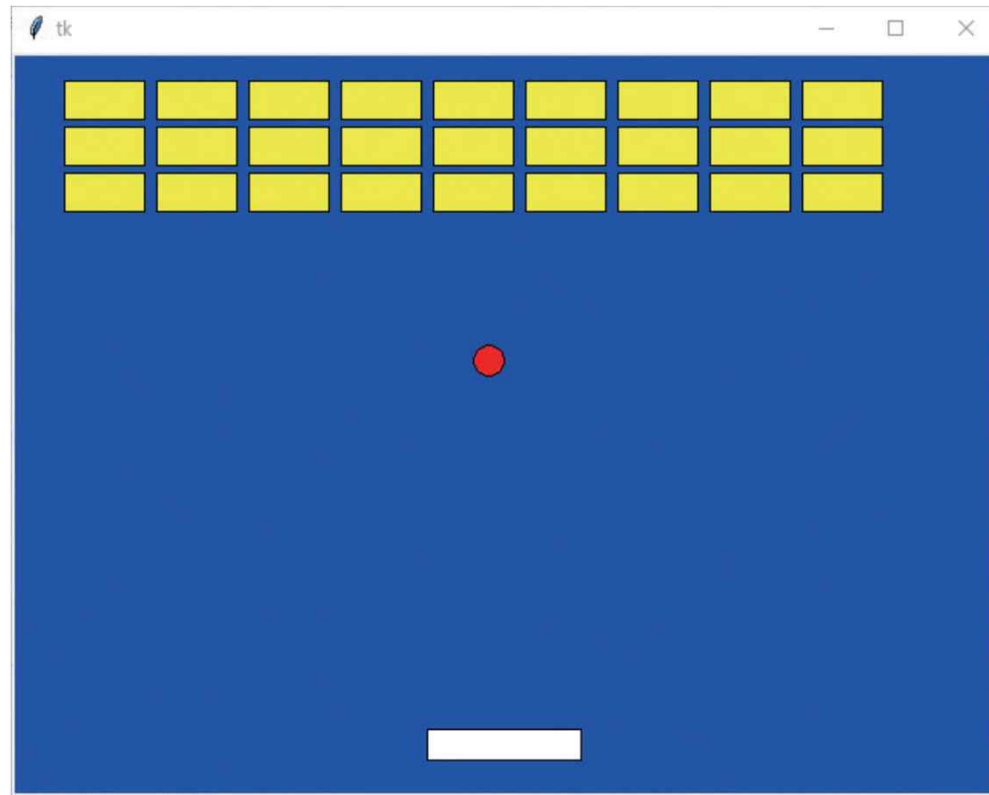


이번장에서 만들 프로그램



tkinter를 이용한 벽돌깨기 게임 작성

- 공을 이용해서 벽돌을 깨는 고전 게임을 작성해보자.



STEP #1: 화면을 작성해보자.

- 다음과 같이 배경색이 청색으로 칠해진 윈도우를 생성해보자. 프레임 만들고 프레임 안에 캔버스를 생성하면 된다.

```
class BrickBreaker(Frame):
    def __init__(self, root):
        super().__init__(root)
        self.width = 640
        self.height = 480
        self.canvas = Canvas(self, bg='blue',
                             width=self.width,
                             height=self.height,)
        self.canvas.pack()
        self.pack()
```

STEP #2: Sprite 클래스를 정의해보자.

- Sprite 클래스를 만들어서 공통적인 속성과 메소드는 여기에서 정의하자.

```
class Sprite():
    def __init__(self, canvas, item):
        self.canvas = canvas # 캔버스 객체
        self.item = item     # 캔버스 안에 있는 도형의 식별 번호
        self.speedx = 3      # x 방향 속도
        self.speedy = 3      # y 방향 속도
        self.x = 0           # 현재 x좌표
        self.y = 0           # 현재 y좌표

        # 도형의 위치와 크기를 반환한다.
    def get_coords(self):
        return self.canvas.coords(self.item)
```

STEP #2: Sprite 클래스를 정의해보자.

```
# 도형의 위치를 반환한다.
def get_position(self):
    pos = self.canvas.coords(self.item)          # 튜플로 반환
    x = pos[0]
    y = pos[1]
    return x, y

# 객체의 상태를 변경한다.
def update(self):
    self.x = self.x + self.speedx
    self.y = self.y + self.speedy

# 객체를 움직인다.
def move(self):
    self.canvas.move(self.item, self.speedx, self.speedy)

# 객체를 캔버스에서 삭제한다.
def delete(self):
    self.canvas.delete(self.item)
```


STEP #3: Ball 클래스를 정의해보자.

```
class Ball(Sprite):
    def __init__(self, canvas, x, y, radius):
        self.radius = radius
        item = canvas.create_oval(x-self.radius, y-self.radius,
                                   x+self.radius, y+self.radius,
                                   fill='red')

        self.x = x
        self.y = y
        super().__init__(canvas, item)          # 부모 클래스 생성자 호출

    def update(self):
        x, y = self.get_position()
        width = self.canvas.winfo_width()

        # 벽에 부딪히면 방향을 변경한다.
        if x <= 0 or x >= width:
            self.speedx *= -1                  # x 방향 변경
        if y <= 0:
            self.speedy *= -1                  # y 방향
```

STEP #4: 패들을 화면에 그려보자.

```
class Paddle(Sprite):
    def __init__(self, canvas, x, y):
        self.width = 100
        self.height = 20
        item = canvas.create_rectangle(x - self.width / 2, y - self.height / 2,
                                       x + self.width / 2, y + self.height / 2,
                                       fill='white')
        super().__init__(canvas, item) # 부모 클래스 생성자 호출
        self.x = x                    # 현재 위치 저장
        self.y = y

# 패들을 dx, dy만큼 이동한다. 키보드 이벤트에서 호출된다.
def move(self, dx, dy):
    self.x = self.x + dx
    self.y = self.y + dy
    self.canvas.move(self.item, dx, dy)
```

STEP #5: 벽돌을 화면에 그려보자.

```
class Brick(Sprite):
    def __init__(self, canvas, x, y):
        self.width = 52
        self.height = 25
        item = canvas.create_rectangle(x - self.width / 2, y - self.height / 2,
                                       x + self.width / 2, y + self.height / 2,
                                       fill='yellow', tags='brick')
        super().__init__(canvas, item)

    # 벽돌과 공이 충돌하면 벽돌을 삭제한다.
    def handle_collision(self):
        self.delete()
```

STEP #6: 여러 개의 벽돌을 생성하자.

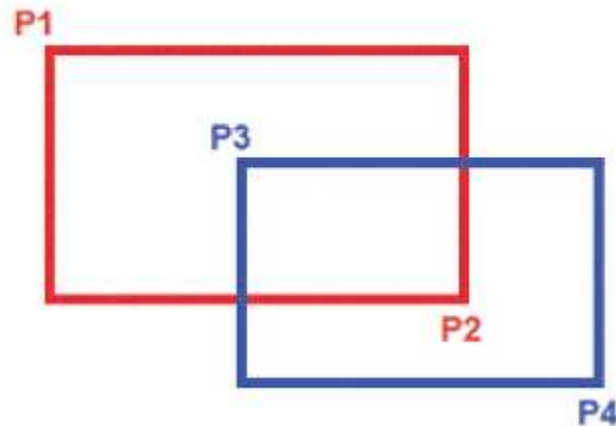
```
# Brick 객체를 2차원 모양으로 생성한다.  
for r in range(1, 4):  
    for c in range(1, 10):  
        brick = Brick(self.canvas, c*60, r*30)  
        # Brick 객체를 shapes에 저장한다.  
        self.shapes[brick.item] = brick
```

STEP #7: 패들을 움직이자.

```
# 캔버스가 키보드 이벤트를 받을 수 있도록 설정한다.  
self.canvas.focus_set()  
# 화살표키와 스페이스키에 이벤트를 붙인다.  
self.canvas.bind('<Left>',  
                 lambda _: self.paddle.move(-10, 0))  
self.canvas.bind('<Right>',  
                 lambda _: self.paddle.move(10, 0))  
self.canvas.bind('<space>', lambda _: self.start())
```

STEP #8: 충돌을 처리하자.

- 공이 패들에 반사되게 하려면 공이 패들과 충돌하였는지를 검사하여야 한다.
- 게임에서 충돌 검사는 아주 중요한 부분으로 공을 감싸는 사각형과 패들을 감싸는 사각형이 겹치는 지를 검사하면 된다.



STEP #8: 충돌을 처리하자.

```
coords = self.ball.get_coords() # Ball 객체의 위치를 구한다.  
# 겹치는 모든 도형을 찾는다. 식별 번호가 저장된다.  
items = self.canvas.find_overlapping(*coords) # *coords를 풀어서 인수로 전달  
  
# 겹치는 도형의 식별 번호로 객체를 찾아서 리스트에 저장한다.  
objects = [self.shapes[x] for x in items if x in self.shapes]  
  
# 충돌 처리 메소드를 호출한다.  
self.ball.collide(objects)  
self.ball.update()  
self.ball.move()  
  
# game_loop()를 50밀리초 후에 호출한다.  
self.after(50, self.game_loop)
```

STEP #8: 충돌을 처리하자.

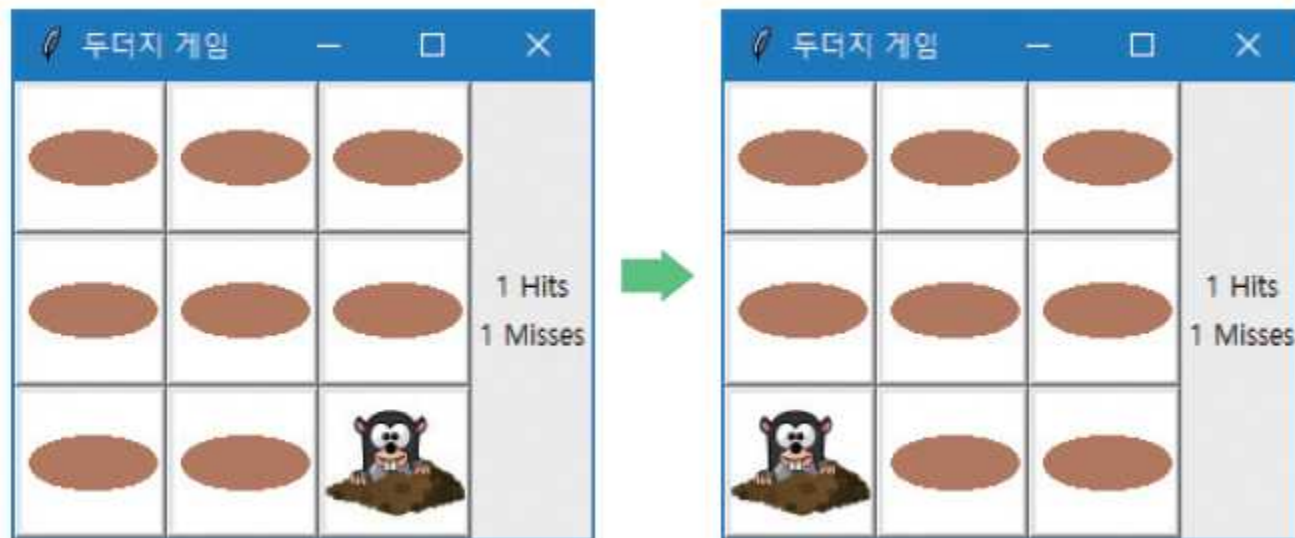
```
# 충돌을 처리하는 메소드
# Ball과 충돌이 일어난 객체들의 리스트가 매개변수로 전달된다.
def collide(self, obj_list):
    x, y = self.get_position()
    # 충돌이 하나라도 일어난다면
    if len(obj_list):
        self.speedy *= -1
        # 충돌은 벽돌이나 패들
        # y 방향 변경

        # 충돌이 일어난 객체가 벽돌이면 Brick의 충돌 처리 메소드를 호출한다.
    for obj in obj_list:
        if isinstance(obj, Brick):
            obj.handle_collision()
```

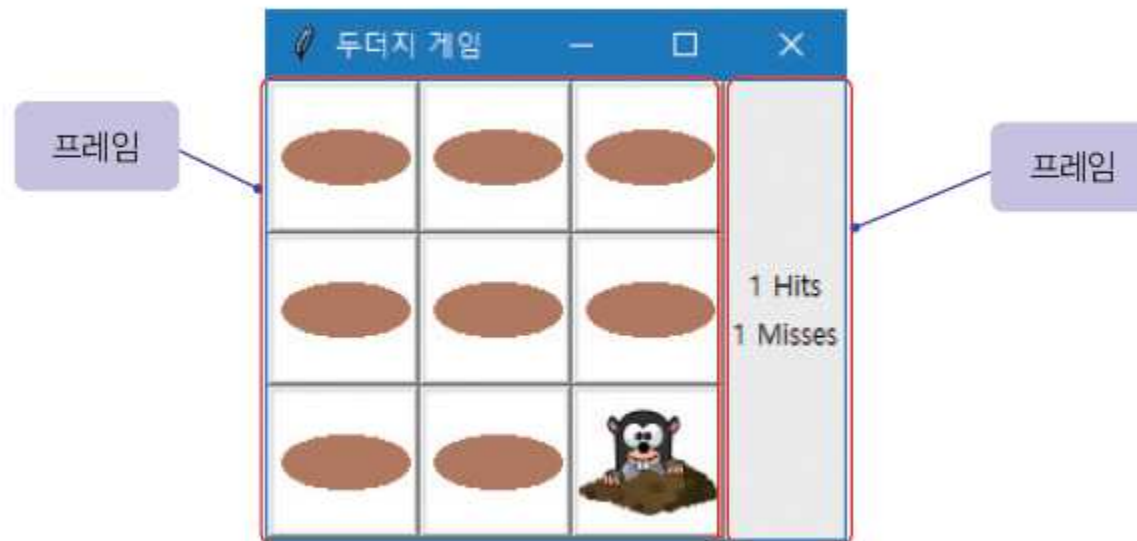

STEP #8: 충돌을 처리하자.

```
def collide(self, obj_list):
    x, y = self.get_position()
    # 공이 패드이나 벽돌에 맞으면 y방향을 반대로 한다.
    if len(obj_list):
        self.speedy *= -1
```

Lab: 두더지 게임



두더지 게임의 사용자 인터페이스 -



Sol: 두더지 게임

```
from tkinter import *  
from random import *
```

```
NUM_MOLES = 3
```

```
# 두더지 개수
```

```
window = Tk()  
window.title("두더지 게임")
```

```
# 루트 윈도우 생성
```

```
moleFrame = Frame(window)  
moleFrame.grid(row=0, column=0)
```

```
# 첫 번째 프레임 컨테이너 생성  
# 첫 번째 프레임을 루트 윈도우에 배치
```

```
statusFrame = Frame(window)  
statusFrame.grid(row=0, column=1)  
hitsLabel = Label(statusFrame, text="0 Hits")  
hitsLabel.pack()  
missedLabel = Label(statusFrame, text="0 Misses")  
missedLabel.pack()
```

```
# 두 번째 프레임 컨테이너 생성  
# 두 번째 프레임을 루트 윈도우에 배치
```

```
mole_image = PhotoImage(file="mole.png") # 이미지를 읽는다.  
no_mole_image = PhotoImage(file="no_mole.png")
```

```
numHits=0 # 획득 점수
numMissed=0 # 실패 횟수
```

```
def mole_hit(c): # 사용자가 두더지를 잡았는지 체크한다.
```

```
    global numHits, numMissed, molesList, missedLabel, hitsLabel
```

```
    if molesList[c]["text"] == "mole":
```

```
        numHits += 1
```

```
        hitsLabel["text"] = str(numHits)+" Hits"
```

```
    else:
```

```
        numMissed += 1
```

```
        missedLabel["text"] = str(numMissed)+" Misses"
```

```
molesList = [] # 버튼들이 저장된다.
```

```
def init():
```

```
    count=0
```

```
    for r in range(NUM_MOLES): # 버튼을 만들어서 격자 형태로 배치한다.
```

```
        for c in range(NUM_MOLES):
```

```
            button = Button(moleFrame, command=lambda
```

```
c=count: mole_hit(c))
```

```
            button["image"] = no_mole_image
```

```
            button["text"] = "no mole"
```

```
            button.grid(row=r, column=c)
```

```
            molesList.append(button)
```

```
            count += 1
```

Sol: 두더지 게임

```
def update(): # 랜덤하게 버튼에 두더지를 심는다.
    global molesList
    for i in range(NUM_MOLES*NUM_MOLES): # 전체 버튼을 초기화한다.
        button = molesList[i]
        button["text"] = "no mole"
        button["image"] = no_mole_image
    x = randint(0, NUM_MOLES*NUM_MOLES-1) # 난수를 발생한다.
    molesList[x]["image"] = mole_image # 두더지 영상으로 바꾼다.
    molesList[x]["text"] = "mole"
    window.after(3000, update) # 3초 지나면 다시 호출되게 한다.

init()
update()
window.mainloop()
```

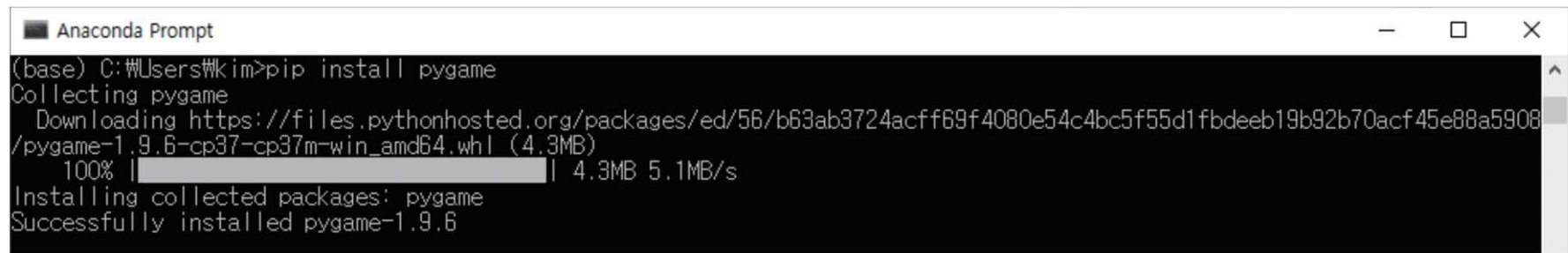
pygame을 이용한 게임 작성

- pygame은 SDL(Simple DirectMedia Layer) 라이브러리의 파이썬 래퍼이다.
- SDL은 사운드, 비디오, 마우스, 키보드, 조이스틱과 같은 시스템의 기본 멀티미디어 하드웨어 구성 요소에 대한 크로스 플랫폼 액세스를 제공한다.



pygame 설치

- pygame을 설치하려면 [시작]→[Anaconda3]→[Anaconda prompt]를 실행해서 “pip install pygame”을 입력한다.



```
Anaconda Prompt
(base) C:\Users\kim>pip install pygame
Collecting pygame
  Downloading https://files.pythonhosted.org/packages/ed/56/b63ab3724acff69f4080e54c4bc5f55d1fbdeeb19b92b70acf45e88a5908/pygame-1.9.6-cp37-cp37m-win_amd64.whl (4.3MB)
    100% |#####| 4.3MB 5.1MB/s
Installing collected packages: pygame
Successfully installed pygame-1.9.6
```


STEP #1: 기본적인 pygame 프로그램

pygame 라이브러리를 포함한다.

```
import pygame
```

pygame 라이브러리를 초기화한다.

```
pygame.init()
```

```
WIDTH = 600
```

```
HEIGHT = 400
```

그림이 그려지는 화면 설정

```
mydisplay = pygame.display.set_mode([WIDTH, HEIGHT])
```

```
pygame.display.set_caption('Shooting Game')
```

```
black = (0,0,0)
```

```
white = (255,255,255)
```

```
blue = (0,0,255)
```

```
spaceshipImage = pygame.image.load('d:/spaceship.png')
```

```
# 사용자가 중단할 때까지 반복 실행한다.
```

```
running = True
```

```
while running:
```

```
    # 사용자가 중단 버튼을 누르면
```

```
    for event in pygame.event.get():
```

```
        if event.type == pygame.QUIT:
```

```
            running = False
```

```
    # 배경을 흰색으로 채운다.
```

```
    mydisplay.fill(white)
```

```
    # 중앙에 이미지를 그린다.
```

```
    mydisplay.blit(spaceshipImage, (WIDTH/2, HEIGHT/2))
```

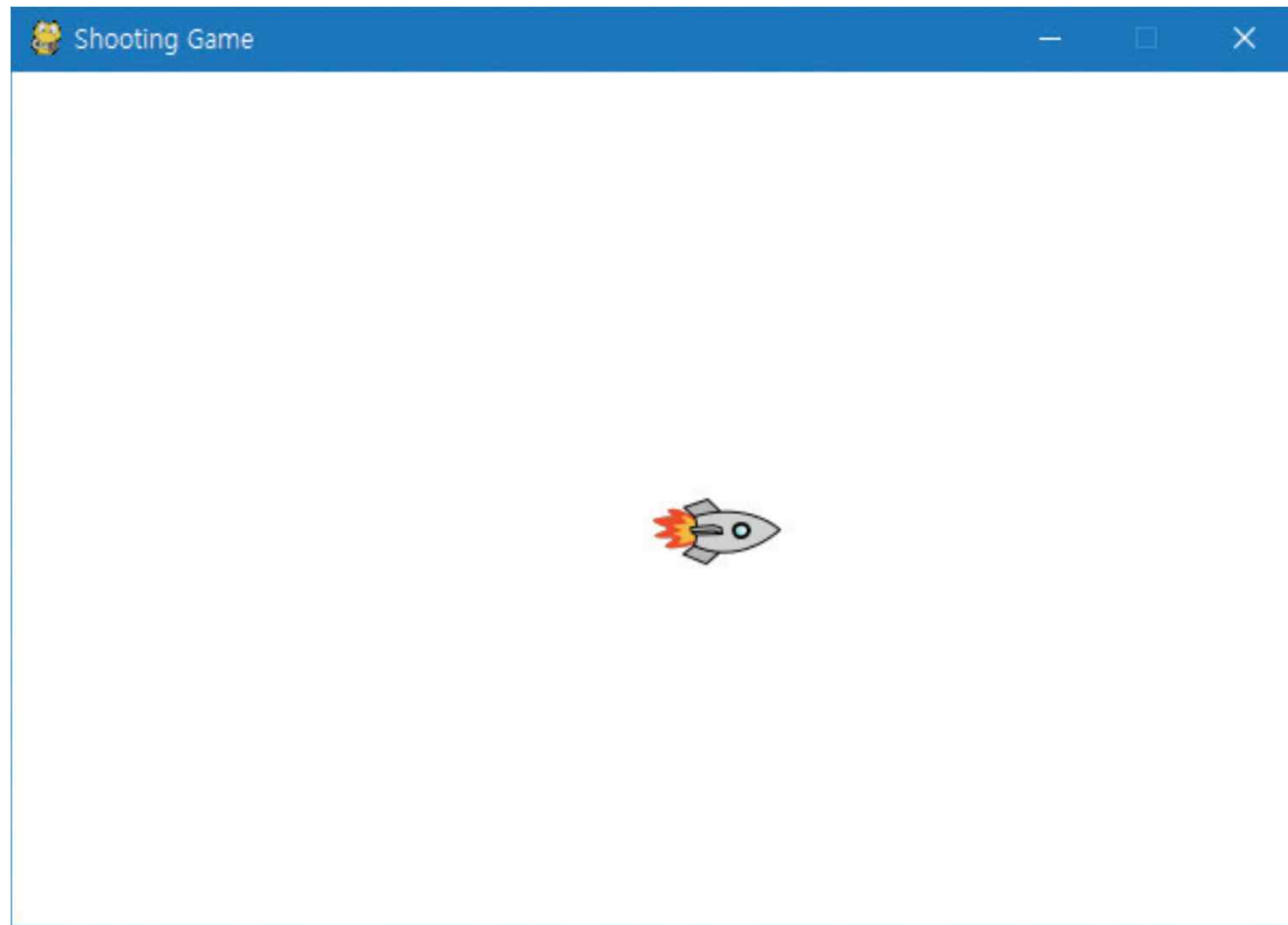
```
    # 화면을 업데이트한다.
```

```
    pygame.display.update()
```

```
# 종료한다.
```

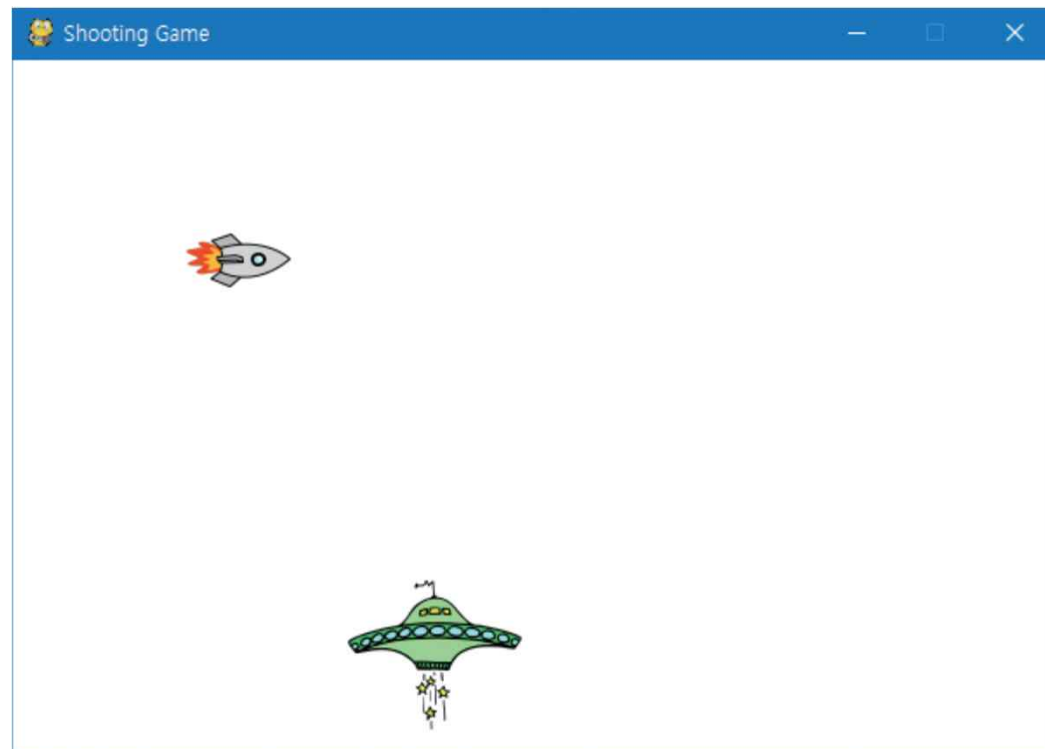
```
pygame.quit()
```

실행 결과



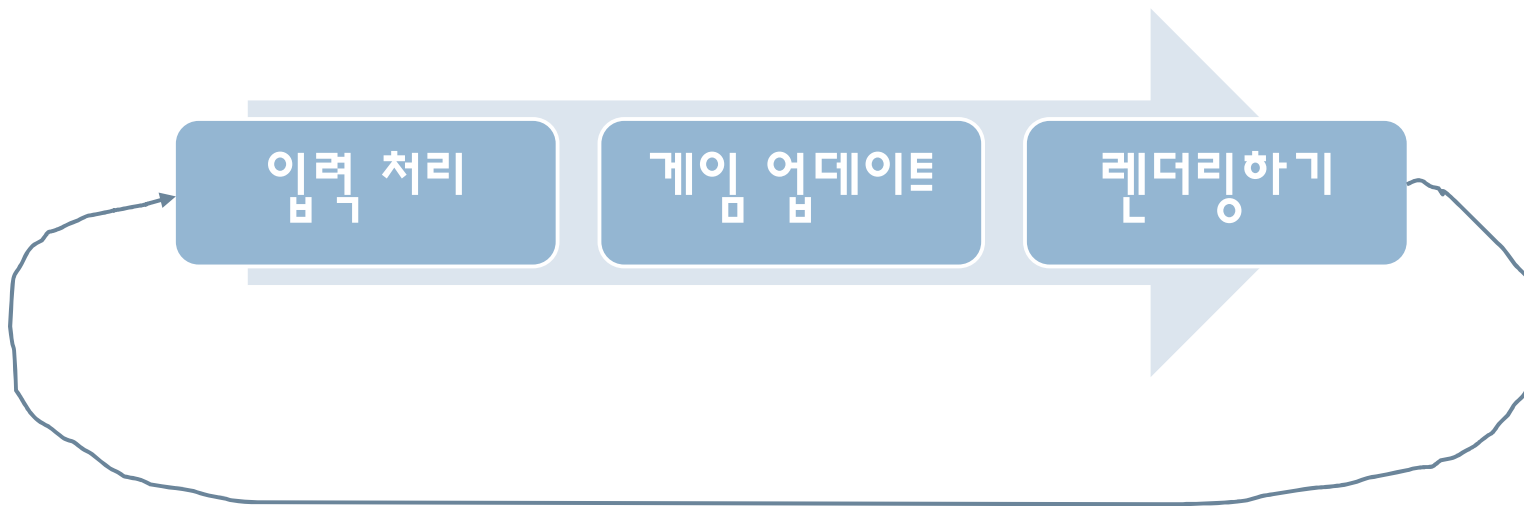
STEP #2: 게임 디자인

- 게임의 목표는 우리 우주선이 외계 우주선을 피하는 것이다.
- 우리 우주선은 화면 왼쪽에 있다.
- 외계 우주선은 오른쪽에서 왼쪽으로 이동한다.
- 우리 우주선은 외계 우주선을 피하기 위해 위, 아래로만 움직일 수 있다.



게임 루프

- 게임 루프는 다음과 같은 네 가지 중요한 작업을 처리한다.
 - 사용자의 입력을 처리한다.
 - 모든 게임 객체의 상태를 업데이트하고 이동시킨다.
 - 디스플레이 및 오디오 출력을 업데이트한다.
 - 게임의 속도를 조절한다.



STEP #3: 우주선 움직이기

```
# 사용자가 중단할 때까지 반복 실행한다.
running = True
x = WIDTH/2
y = HEIGHT/2
while running:

    # 사용자가 중단 버튼을 누르면
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    key = pygame.key.get_pressed()
    if key[pygame.K_UP]:
        y += -1
    if key[pygame.K_DOWN]:
        y += 1
```

STEP #3: 우주선 움직이기

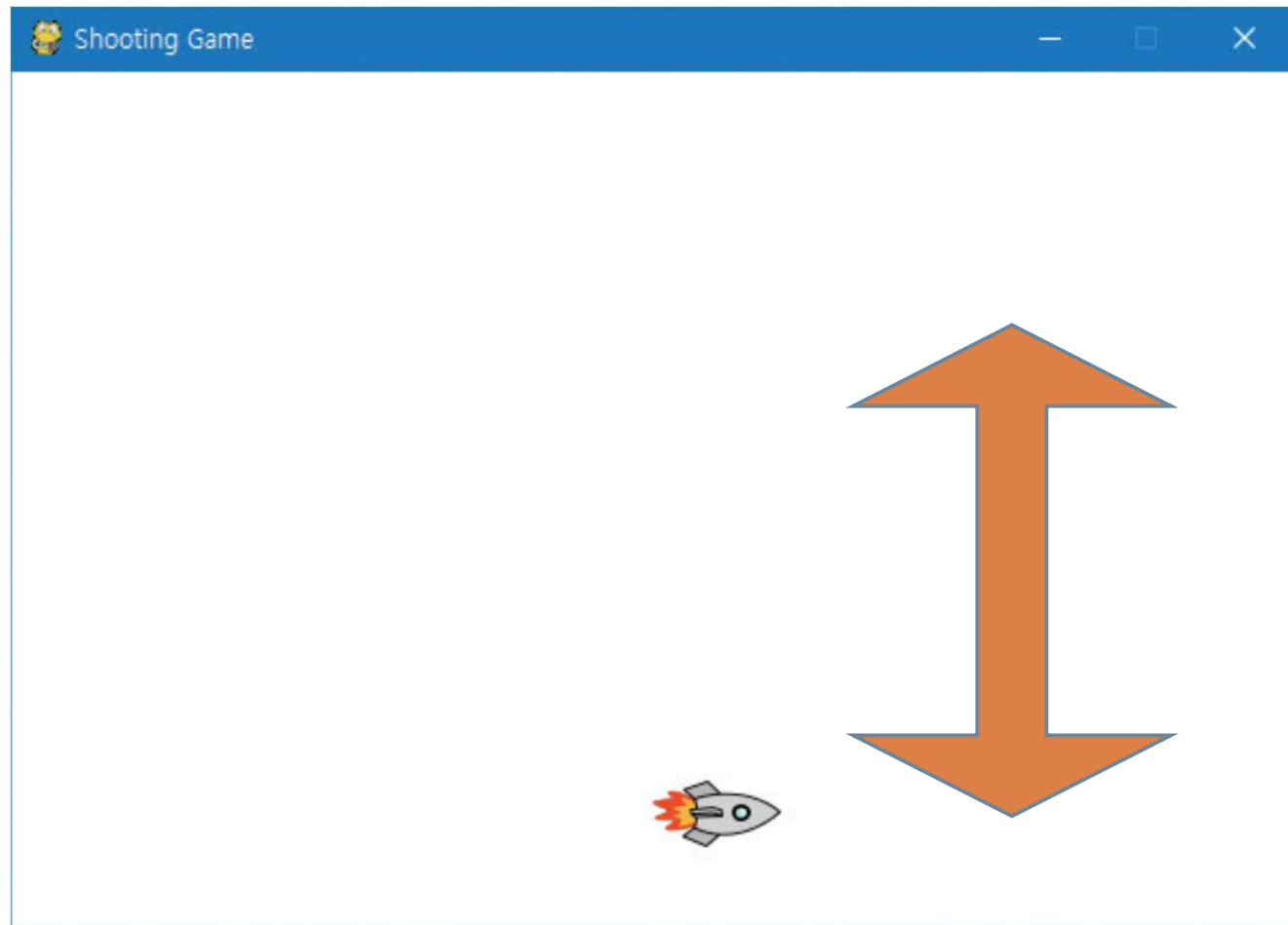
```
# 배경을 흰색으로 채운다.  
mydisplay.fill(white)
```

```
# 중앙에 이미지를 그린다.  
mydisplay.blit(spaceshipImage, (x, y))
```

```
# 화면을 업데이트한다.  
pygame.display.update()
```

```
# 종료한다.  
pygame.quit()
```

실행 결과



STEP #4: 클래스로 만들어보자.

```
class SpaceShip(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load('d:/spaceship.png')
        self.dx = 1
        self.dy = 1
        self.rect = self.image.get_bounding_rect()
        self.rect.x = 100
        self.rect.y = 100

    def move(self, dx, dy):
        self.rect.x += dx
        self.rect.y += dy
```

STEP #4: 클래스로 만들어보자.

```
class EnemyShip(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load('d:/saucer.png')
        self.dx = -1
        self.dy = 0
        self.rect = self.image.get_bounding_rect()
        self.rect.x = 500
        self.rect.y = 300

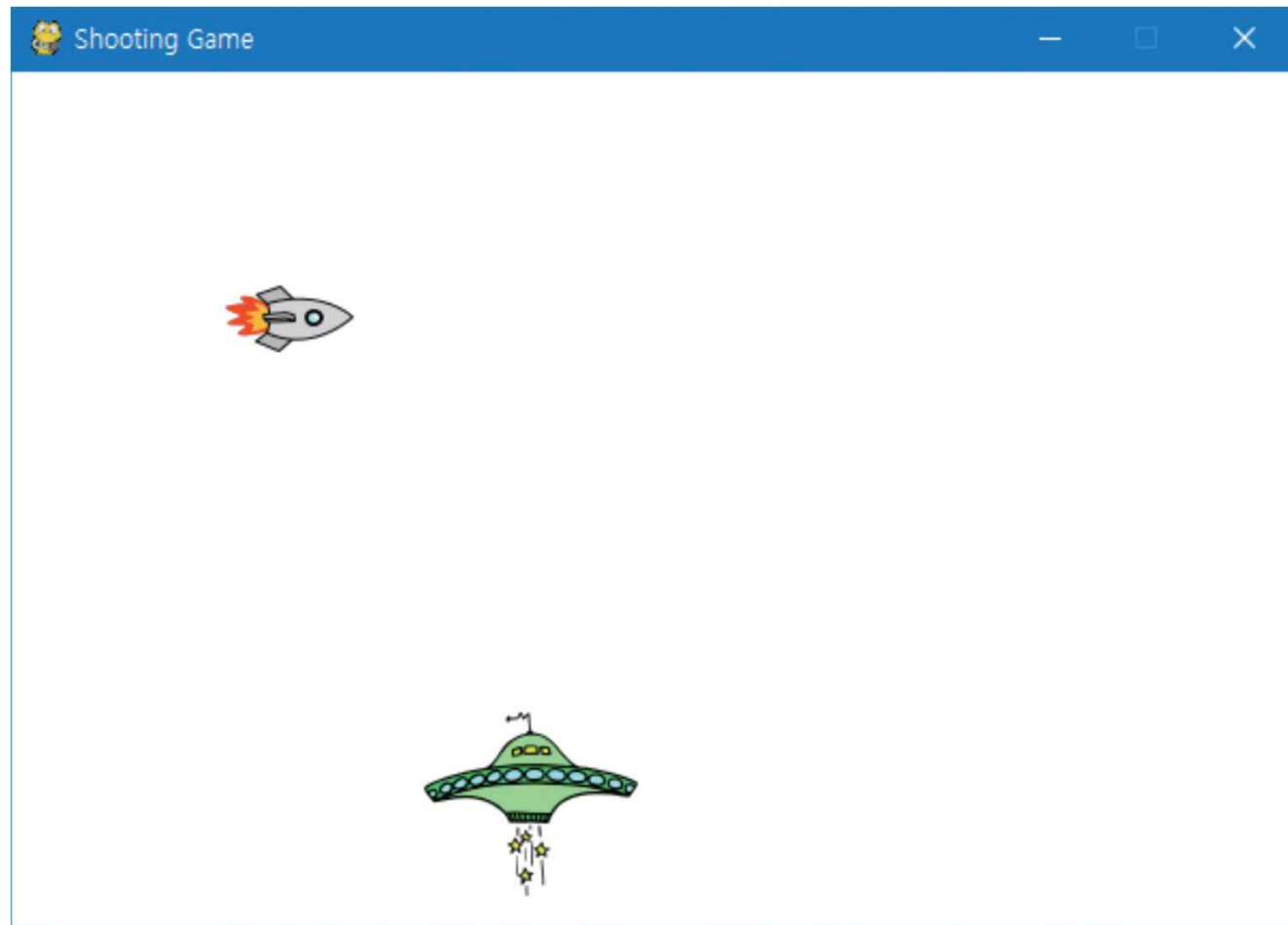
    def move(self):
        self.rect.x += self.dx
        if self.rect.x < 0:
            self.rect.x = 500
```

STEP #5: 충돌을 처리하자.

- pygame에서는 게임에는 사용할 수 있는 많은 충돌 감지 방법을 제공한다.
- 여기서는 `spritecollideany()`라는 메소드를 사용한다. 이 메소드는 `Sprite`와 `Group`을 매개 변수로 허용한다. 즉 `Group`의 모든 객체의 `rect`가 `Sprite`의 `rect`와 교차하는지 확인한다.

```
if pygame.sprite.spritecollideany(player, [enemy]):  
    player.kill()  
    running = False
```

실행 결과



이제 나는 할 수 있다

1. 나는 `tkinter`를 이용하여 벽돌깨기 게임을 작성할 수 있다
2. 나는 `pygame`을 이용하여 외계 우주선을 피하는 게임을 작성할 수 있다

