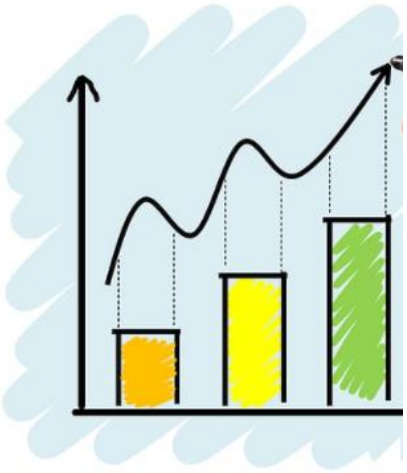
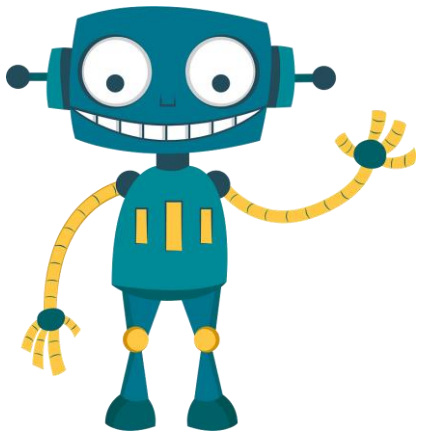


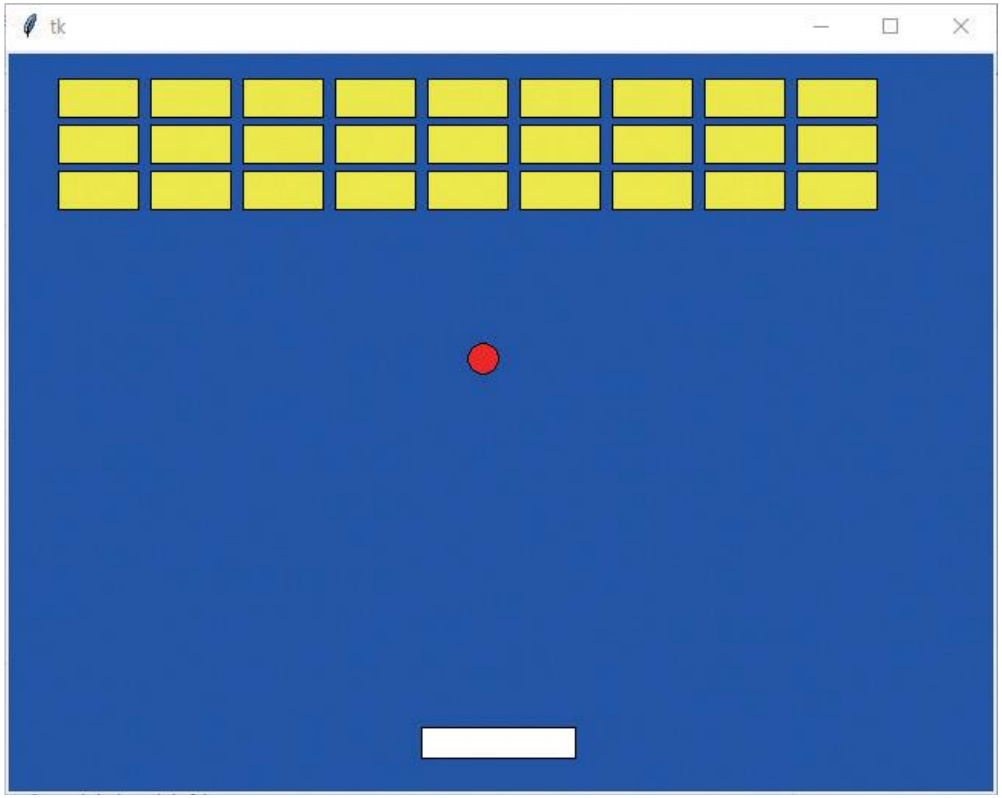


13장 파이썬을 이용한 게임 작성

tkinter를 이용한 벽돌깨기 게임 작성

- 공을 이용해서 벽돌을 깨는 고전 게임을 작성해보자.
- 스페이바 시작. 좌우 화살표키로 패들을 이동

Sprite
Brick: 벽돌
Ball
Paddle



STEP #2: Sprite 클래스를 정의해보자.

- Sprite 클래스를 만들어서 공통적인 속성과 메소드는 여기에서 정의.

```
from tkinter import *

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

                # 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)
    x = pos[0]
    y = pos[1]
    return x, y

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

```
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)

    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 변경
```

STEP #8: 충돌을 처리하자.

```
 # 충돌을 처리하는 메소드
 # Ball과 충돌이 일어난 객체들의 리스트가 매개변수로 전달된다.
def collide(self, obj_list):
    x, y = self.get_position()

    # 공이 패들이나 벽돌에 맞으면 y방향을 반대로 한다.
    if len(obj_list):
        self.speedy *= -1

    for obj in obj_list:
        if isinstance(obj, Brick):
            obj.handle_collision()
```

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 = 24
        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 #1: 화면을 작성해보자.

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

```
class BrickBreaker(Frame):              # help(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 #6: 여러 개의 벽돌을 생성하자.

```
# shapes에는 화면에 있는 모든 객체가 저장된다.
# 키는 도형 식별 번호이고 값은 객체이다.
self.shapes = {}

# 패들 객체를 생성하고 shapes에 저장한다.
self.paddle = Paddle(self.canvas, self.width/2, 450)
self.shapes[self.paddle.item] = self.paddle

# Ball 객체를 생성한다.
self.ball = Ball(self.canvas, 310, 200, 10)            # x, y, radius

# 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())

    def start(self):
        self.game_loop()
```

STEP #8: 충돌을 처리하자.

```
# 게임 루프를 작성한다.
    def game_loop(self):
        coords = self.ball.get_coords()    # Ball 객체의 위치를 구한다.
        # 겹치는 모든 도형을 찾는다. 식별 번호가 저장된다.
        items = self.canvas.find_overlapping(*coords)

        # 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)

window = Tk()
game = BrickBreaker(window)
window.mainloop()
```