05

미로 게임

1. 배열과 화면 구성
2. 이미지 추가
3. 키보드 이벤트 추가
4. 실시간 움직임 처리
5. 클리어 판정
6. 다시 시작 추가
7. 개선점

5-1. 배열과 화면 구성

- 배열(Array): 행렬(Matrix), 표(Table)의 개념
  - 1차원의 형태: 파이썬의 리스트(list)

```
a = [1, 2, 3]
a[0] = 0
```

저번 학기에 배운 리스트

  - 2차원의 형태: 리스트를 중첩해서 사용하면 됨

```
a = [
        [1, 2, 3, 4],
        [5, 6, 7, 8],
        [9, 10, 11, 12]
]
a[0] = ?
a[0][1] = ?
```

행이 우선

5-1. 배열과 화면 구성

- 배열(Array): 행렬(Matrix), 표(Table)의 개념
  - 2차원의 형태: 2D로 화면 구성할 경우 배경 데이터 관리
    → 이미지가 2차원 형태이기 때문
  - 먼저 생각할 것: 미로의 구성을 어떻게 할까?
    → 10 × 7 배열, 흰색이 통로(0), 회색이 벽(1)

1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	0	0	1
1	0	1	1	0	0	1	0	0	1
1	0		1	0	0	0	0	0	1
1	0		1	1	1	1	1	0	1
1	0	0	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1	1	1

5-1. 배열과 화면 구성

- 미로의 구성을 코드로 표현

```
maze = [
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 0, 0, 1, 0, 0, 1],
    [1, 0, 1, 1, 0, 0, 1, 0, 0, 1],
    [1, 0, 0, 1, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 1, 1, 1, 1, 1, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
]
```

1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	0	0	1
1	0	1	1	0	0	1	0	0	1
1	0	0	1	0	0	0	0	0	1
1	0	0	1	1	1	1	1	0	1
1	0	0	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1	1	1

5-1. 배열과 화면 구성

- 미로를 윈도우에 뿌리기: canvas와 이중 for문을 사용해야 함

2차원 배열(리스트)와 짝꿍이 이중(중첩) for문

```
import tkinter as tk

... maze 리스트 ...

root = tk.Tk()
root.title("Maze Game")
# 800x500 화면 구성 -> 화면 구성이 먼저임
canvas = tk.Canvas(width=800, height=560, bg="white")
canvas.pack()

# 아래부터 2차원 리스트를 사각형으로 뿌리기
for y in range(7):
    for x in range(10):
        if maze[y][x] == 1: # 벽이면
            canvas.create_rectangle(y*80, x*80, y*80 + 80, x*80 + 80, fill="gray")

root.mainloop()
```

5-2. 이미지 추가

- 실시간 처리, 키 입력, 미로 - 3가지 조합으로 만들기
  - 강아지 이미지 추가 (이미지 resize도 함께)

```
import tkinter as Tk
from PIL import Image, ImageTk  # pip install pillow (image resize)

... 미로 리스트 생략 ...
# 강아지 위치 좌표
dog_x = 1
dog_y = 1

... 윈도우 구성과 사각형 뿌리는 코드 생략 ...

# 강아지를 미로 안에 넣어야 함
dog = Image.open("dog.png")
dog = dog.resize((80, 80)) # 이미지 크기는, 사각형 블록의 크기와 같아야 함
dog = ImageTk.PhotoImage(dog)
canvas.create_image(dog_x*80 + 40, dog_y*80 + 40, image=dog, tag="DOG")

root.mainloop()
```

프로그래밍에서 대부분
중요한 변수들은 상단에 배치하는 것이 관례

아래 코드는 오리 게임에서도 한 내용

5-3. 키보드 이벤트 추가

- 실시간 처리, 키 입력, 미로 - 3가지 조합으로 만들기
  - 강아지에게 키보드 이벤트 추가: 움직이는 코드는 없음!

```
... 미로 리스트, 강아지 위치 좌표 생략 ...
# 키보드 이벤트를 위한 변수와 이벤트 함수 추가
key = ""
def key_down(e): # 키를 누를 때 발생하는 이벤트 처리 함수
    global key   # key는 global(전역) 변수야! local(지역)로 쓰지마!
    key = e.keysym

def key_up(e):   # 키를 뗄 때 발생하는 이벤트 처리 함수
    global key
    key = ""
... 윈도우, 캔버스 코드 바로 다음에 ...
# 키보드 이벤트 등록
root.bind("<KeyPress>", key_down)
root.bind("<KeyRelease>", key_up)
... 사각형 뿌리는 코드 생략 ...
... 강아지 이미지 넣는 코드 생략 ...
root.mainloop()
```

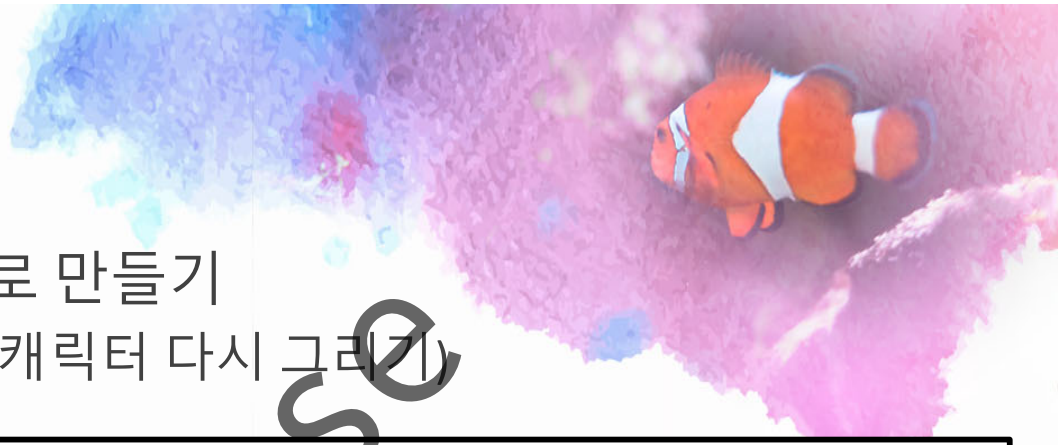
5-4. 실시간 움직임 처리

- 실시간 처리, 키 입력, 미로 - 3가지 조합으로 만들기
  - 강아지 키보드 이벤트를 확인하고 실시간으로 움직임 처리

```
... 미로 리스트, 강아지 위치 좌표 생략 ...
...키보드 이벤트를 위한 변수와 이벤트 함수 추가 바로 다음부터 ...
def main_action():
    global dog_x, dog_y
    if key == "Up" and maze[dog_y - 1][dog_x] == 0: # 위키를 누르고 통로이면
        dog_y -= 1  # 위로 한 칸 가기
    if key == "Down" and maze[dog_y + 1][dog_x] == 0:  # 아래키를 누르고 통로이면
        dog_y += 1  # 아래로 한 칸 가기
    if key == "Left" and maze[dog_y][dog_x - 1] == 0: # 왼쪽키를 누르고 통로이면
        dog_x -= 1  # 왼쪽으로 한 칸 가기
    if key == "Right" and maze[dog_y][dog_x + 1] == 0: # 오른쪽키를 누르고 통로이면
        dog_x += 1  # 오른쪽으로 한 칸 가기
    canvas.coords("DOG", dog_x*80 + 40, dog_y*80 + 40)
    # 새로운 위치로 이동 (80픽셀 움직임)
    root.after(200, main_action)  # 0.2초후 이 함수 재실행
... 이후 생략 끝에 쪽 줄 다음부터 ...
main_action()       # main_action 함수 첫 시작점
root.mainloop()
```

돌다리를 두들겨 보고건너자

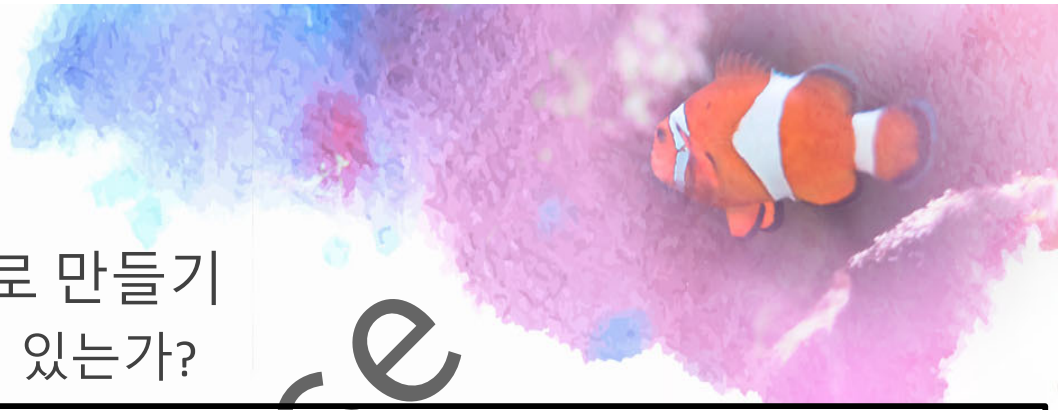
5-4. 실시간 움직임 처리



- 실시간 처리, 키 입력, 미로 - 3가지 조합으로 만들기
  - 강아지가 지나온 길을 표시하기 (사각형과 캐릭터 다시 그리기)

```
def main_action():
    ... 생략 ...
    if key == "Right" and maze[dog_y][dog_x + 1] == 0: # 오른쪽키를 누르고 통로이면
        dog_x += 1  # 오른쪽으로 한 칸 가기
    if maze[dog_y][dog_x] == 0: # 캐릭터가 있는 장소가 통로라면
        maze[dog_y][dog_x] = 2  # 리스트 값을 2로 변경하고 사각형 다시 그리기
        canvas.create_rectangle(dog_x*80, dog_y*80, dog_x*80 + 79, dog_y*80 + 79, \
                                fill="yellow", width=0)
    # 사각형을 그렸으니 그 장소에 있는 강아지가 안보이므로 삭제하고 다시 그리기
    canvas.delete("DOG")
    canvas.create_image(dog_x*80 + 40, dog_y*80 + 40, image=dog, tag="DOG")
    #canvas.coords("DOG", dog_x*80 + 40, dog_y*80 + 40) # 새로운 위치로 이동 (80픽셀
움직임)
    root.after(200, main_action)  # 0.2초후 이 함수 재실행
```

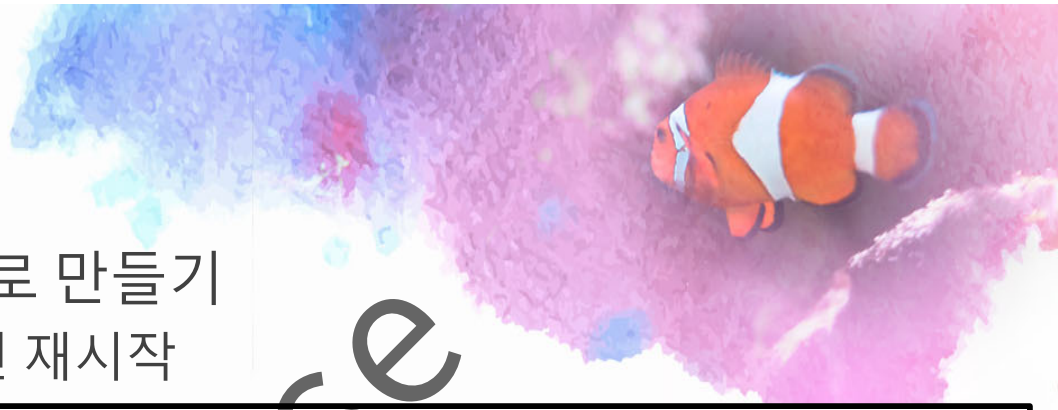
5-5. 클리어 판정



- 실시간 처리, 키 입력, 미로 - 3가지 조합으로 만들기
  - 게임 클리어 판정: 모든 통로 바닥이 칠해져 있는가?

```
import tkinter.messagebox as msgbox
is_clear = 0  # 게임 클리어 판정 변수 추가 (맨 위쪽에)
... 생략 ...
def main_action():
    ... 생략 ...
    if key == "Right" and maze[dog_y][dog_x + 1] == 0: # 오른쪽키를 누르고 통로이면
        dog_x += 1  # 오른쪽으로 한 칸 가기
    if maze[dog_y][dog_x] == 0: # 캐릭터가 있는 장소가 통로라면
        maze[dog_y][dog_x] = 2  # 리스트 값을 2로 변경하고 사각형 다시 그리기
        is_clear += 1  # 칠한 횟수 1 증가
        canvas.create_rectangle(dog_x*80, dog_y*80, dog_x*80 + 79, dog_y*80 + 79, \
                                fill="yellow", width=0)
    ... 다시 그리는 코드 생략 ...
    if is_clear == 30: # 30개 칸이 모두 칠해졌으면
        canvas.update()
        msgbox.showinfo("Game Clear!", "모든 곳을 칠했습니다!")
    else:
        root.after(200, main_action)  # 0.2초후 이 함수 재실행
```

5-6. 다시 시작 추가

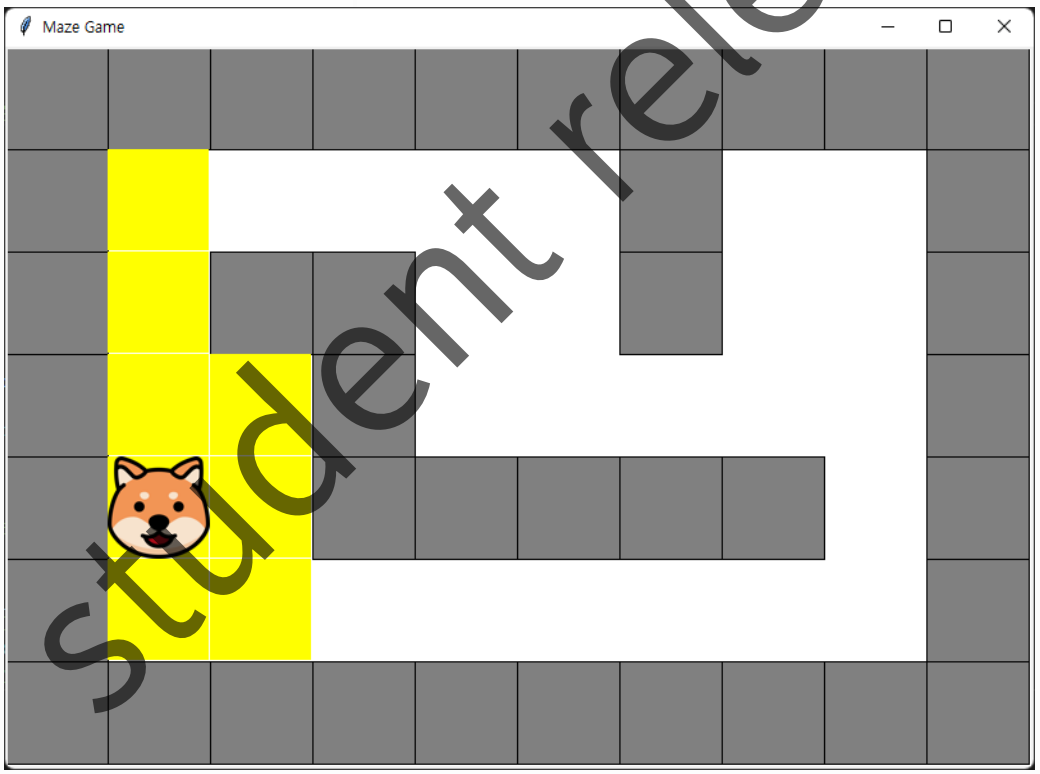


- 실시간 처리, 키 입력, 미로 - 3가지 조합으로 만들기
  - 게임 다시 시작 추가: 왼쪽 Shift 키를 누르면 재시작

```
... 생략 ...
def main_action():
    global dog_x, dog_y, is_clear
    if key == "Shift_L" and is_clear > 1: # 왼쪽Shift키가 눌리고 2칸 이상 칠했으면
        canvas.delete("PAINT")
        dog_x = dog_y = 1
        is_clear = 0
        for y in range(7):
            for x in range(10):
                if maze[y][x] == 2:
                    maze[y][x] = 0
    ... 강아지 이동 코드 생략 ...
    if maze[dog_y][dog_x] == 0: # 캐릭터가 있는 장소가 통로라면
        maze[dog_y][dog_x] = 2 # 리스트 값을 2로 변경하고 사각형 다시 그리기
        is_clear += 1  # 칠한 횟수 1 증가
        canvas.create_rectangle(dog_x*80, dog_y*80, dog_x*80 + 79, dog_y*80 + 79, \
                                fill="yellow", width=0, tag="PAINT")
    ... 그 이하 생략 ...
```

5-7. 개선점

- 이 게임의 문제점이나 업그레이드할 것은?
  - 문제점을 개선하는 것이 프로그래머가 하는 일
  - 이미 갔던 길을 빠져나올 방법이 없음 - 처리 코드가 없기 때문!



5-7. 개선점

- 이 게임의 문제점이나 업그레이드할 것은?
  - 문제점을 개선하는 것이 프로그래머가 하는 일

  - 만약, 미로를 10 by 7이 아니고 다르게 한다면?
    → Canvas의 width와 height를 변경해야 하고,
    이중 for문에서 range()를 변경해야 함
    → 사각형 박스 픽셀(pixel)이 80인 것도 변경해야 할 것임
    → 이를 상수로 처리해서
    width와 height를 자동으로 계산하는 것이 효율적일 것임
  - 미로 안에 숫자를 꼭 0과 1만으로?
    → 이 숫자를 다르게 해서, 0은 수면, 1은 숲, 2는 돌판 등
    다양한 지형 처리를 할 수 있음
    → 대부분 2D 게임의 경우 배경이나 맵에 뭐가 존재하는지를
    숫자로 관리함 (이 숫자는 그리고 대부분 상수로 관리함)

5-7. 개선점

- 이 게임의 문제점이나 업그레이드할 것은?
  - 게임 시작 시 타이틀 화면이 없음 (첫 시작 화면)
    → 다양한 기능 구현 시 메뉴 타이틀 화면이 필요할 수 있음
    → 메뉴 선택을 어떤 것을 했는지 관리하는 변수와 함수가 존재해야 함
    → 이러한 것을 따로 캔버스로 그리고 이를 삭제해야 함

  - 게임에는 난이도나 단계(stage)가 존재할 수도 있음
    → 단계에 따라 미로 데이터(리스트)를 변경하고 초기 위치를 설정해줘야 함
    → 미로 데이터를 변경했으면 이를 또 다시 캔버스에 그려야 함
    → 캔버스에 그리는 이러한 과정을 함수로 따로 빼면 편리할 것임!

06

고양이 게임

1. 계획과 리소스 준비
2. 게임 화면 구성
3. 커서와 마우스 이벤트
4. 고양이 위치 관리
5. 낙하 알고리즘 구현
6. 판정 알고리즘 구현
7. 타이틀과 게임 오버 화면 구현

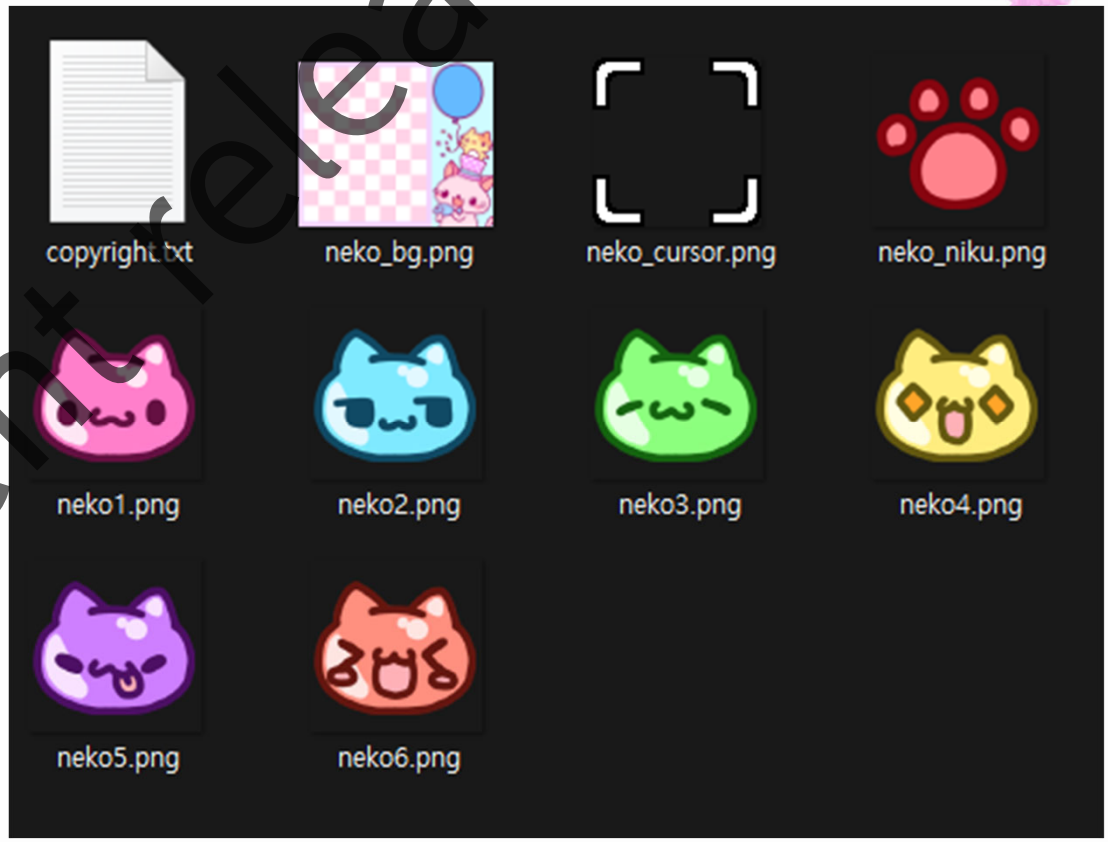
6-1. 계획과 리소스 준비

- 게임 규칙 정하기
  - 화면을 클릭한 위치에 고양이를 1개 배치
    → 배치되는 고양이는 화면 오른쪽 위에 표시되며, 랜덤으로 변경
  - 고양이를 배치하면 화면 위에서 고양이가 떨어짐
  - 떨어진 고양이는 바닥에 아무것도 없으면 화면 아래부터 위까지 쌓임
  - 가로, 세로, 대각선 중 한 방향에 같은 고양이가 3개 이상 모이면 없앨 수 있음
  - 한 줄이라도 맨 위 화면에 닿으면 게임 클리어

6-1. 계획과 리소스 준비

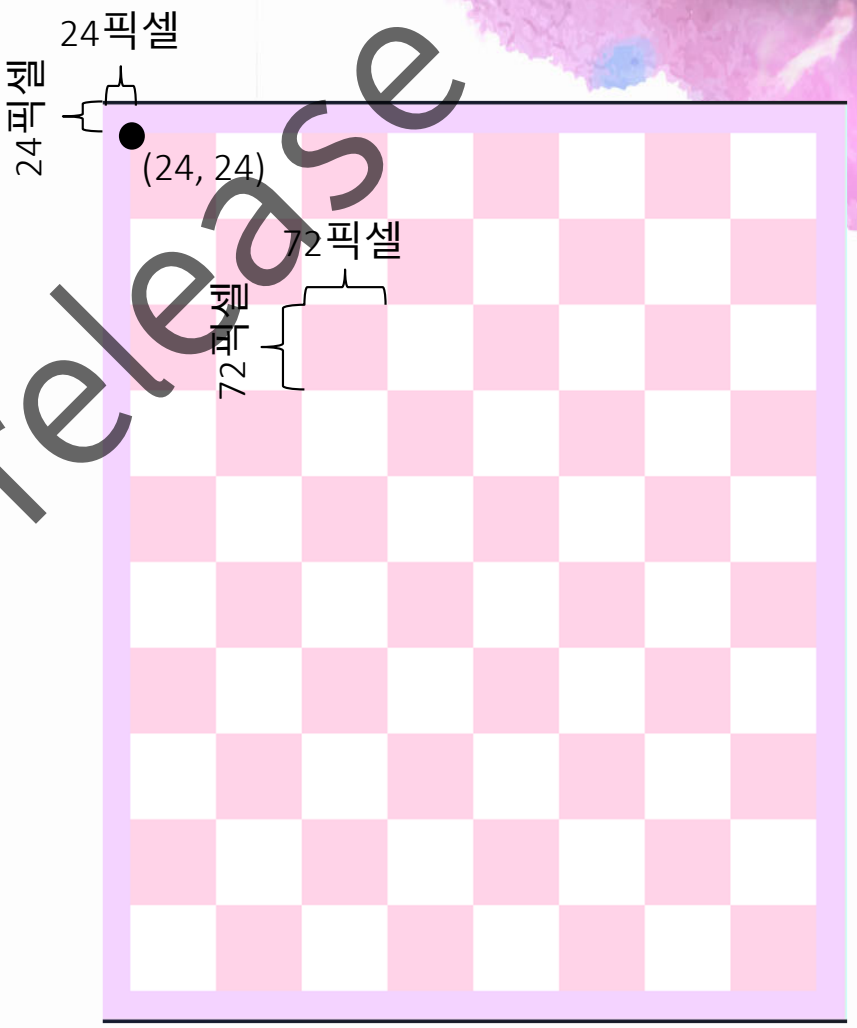
- 게임 리소스 준비
  - https://github.com/Marlrero/Lecture_game_example/tree/master/tkinter_primary_adv/cat_game

  - 배경화면
  - 마우스 커서 표시
  - 고양이 6마리
  - 고양이 발바닥



6-2. 게임 화면 구성

- 게임 화면 구성하기
  - 고양이를 밑으로 내려 보내려면 영역의 백그라운드 화면 생각
  - 가로, 세로 몇 칸을 둘 지 생각
  - 각각의 픽셀을 생각
  - 저 칸 안에서 마우스 커서가 놀아야 함!



6-2. 게임 화면 구성

- 게임 화면 구성하기
  - Tkinter 윈도우 구성 먼저!

```
import tkinter as tk

root = tk.Tk()
root.title("AniPang")
root.resizable(False, False)
cvs = tk.Canvas(root, width=912, height=768) # background size
cvs.pack()

root.mainloop()
```

6-2. 게임 화면 구성

- 게임 화면 구성하기
  - 배경 이미지와 커서 이미지 불러오기

```
import tkinter as tk

BG_SIZE = (912, 768)

root = tk.Tk()
root.title("AniPang")
root.resizable(False, False)
cvs = tk.Canvas(root, width=BG_SIZE[0], height=BG_SIZE[1]) #
background size
cvs.pack()

# 경로 설정 잘해야 함!
background = tk.PhotoImage(file="resource/neko_bg.png")
cursor = tk.PhotoImage(file="resource/neko_cursor.png")
cvs.create_image(BG_SIZE[0] // 2, BG_SIZE[1] // 2, image=background)

root.mainloop()
```

파이썬에서는
변하지 않는 수(상수)는 대문자로 이름 구성이 관례

이렇게 상수에 이름을 붙이면 관리하기 편리함

6-3. 커서와 마우스 이벤트

- 게임 화면 구성하기
  - 마우스 이벤트 추가

```
BG_SIZE = (912, 768)
CUR_LIMIT = 24
BLOCK_SIZE = (8, 10)
BLOCK_PIXEL = 72

cursor_x = 0  # 커서의 좌표
cursor_y = 0
mouse_x = 0   # 실제 마우스 좌표
mouse_y = 0

def mouse_move(e): # 마우스 움직일 때 이벤트 함수
    global mouse_x, mouse_y
    mouse_x = e.x
    mouse_y = e.y

... 다음 장 이어서 ...
```

6-3. 커서와 마우스 이벤트

- 게임 화면 구성하기
  - 마우스 이벤트 추가

```
def main_action(): # 실시간 처리 함수
    global cursor_x, cursor_y
    if CUR_LIMIT <= mouse_x < CUR_LIMIT + BLOCK_PIXEL * BLOCK_SIZE[0] \
        and CUR_LIMIT <= mouse_y < CUR_LIMIT + BLOCK_PIXEL * BLOCK_SIZE[1]:
        cursor_x = int((mouse_x - CUR_LIMIT) / BLOCK_PIXEL)
        cursor_y = int((mouse_y - CUR_LIMIT) / BLOCK_PIXEL)
    cvs.delete("CURSOR")
    cvs.create_image(cursor_x * BLOCK_PIXEL + 60, cursor_y * BLOCK_PIXEL + 60,\
                     image=cursor, tag="CURSOR")
    root.after(100, main_action)

... 윈도우, 캔버스, 이미지 불러오는 코드 생략 ...

root.bind("<Motion>", mouse_move) # 마우스 이벤트 처리
main_action()  # 실시간 처리 함수 시작점
root.mainloop()
```

1칸의 크기로 나누기

최종: 몇 번째 칸이지?

여백 부분의 픽셀 수를 빼줘야 원점 고정 가능

60픽셀?

create_image 좌표는 중심점

여백 24 + (블록 픽셀 72/2=36)

= 60

6-4. 고양이 위치 관리

- 게임 화면 구성하기
  - 고양이를 리스트로 관리해 위치가 어디에 있는지 확인

```
import tkinter as tk

root = tk.Tk()  # root를 일로 올려야 아래 IMG_CAT 에러 발생 안함
... 생략 ...

IMG_CAT = [
    None,       # 0은 아무것도 없음(cat_loc에서)
    tk.PhotoImage(file="resource/neko1.png"),
    tk.PhotoImage(file="resource/neko2.png"),
    tk.PhotoImage(file="resource/neko3.png"),
    tk.PhotoImage(file="resource/neko4.png"),
    tk.PhotoImage(file="resource/neko5.png"),
    tk.PhotoImage(file="resource/neko6.png"),
    tk.PhotoImage(file="resource/neko_niku.png")
]
```

6-4. 고양이 위치 관리

- 게임 화면 구성하기
  - 고양이를 리스트로 관리해 위치가 어디에 있는지 확인

```
cat_loc = [
    [1, 0, 0, 0, 0, 0, 7, 7],
    [0, 2, 0, 0, 0, 0, 7, 7],
    [0, 0, 3, 0, 0, 0, 0, 0],
    [0, 0, 0, 4, 0, 0, 0, 0],
    [0, 0, 0, 0, 5, 0, 0, 0],
    [0, 0, 0, 0, 0, 6, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 2, 3, 4, 5, 6]
]
```

고양이가 화면에 나오는지 확인
나중에 모두 0으로 바꿔야 함

6-4. 고양이 위치 관리

- 게임 화면 구성하기
  - 고양이를 리스트로 관리해 위치가 어디에 있는지 확인

```
... 생략 ...
def draw_cat():
    for y in range(BLOCK_SIZE[1]):
        for x in range(BLOCK_SIZE[0]):
            if cat_loc[y][x] > 0:
                cvs.create_image(x*BLOCK_PIXEL + 60, y*BLOCK_PIXEL + 60, \
                    image=IMG_CAT[cat_loc[y][x]])

def main_action(): # 실시간 처리 함수
    global cursor_x, cursor_y
    ... 생략 ...
    draw_cat()   # 실시간으로 계속 고양이 위치를 찾아 그려야 함
    root.after(100, main_action)

... 생략 ...
```

6-5. 낙하 알고리즘 구현

- 알고리즘: 고양이가 아래로 떨어지는 상황 구현

```
def drop_cat(): # 고양이가 위에서 떨어져야 함
    for y in range(BLOCK_SIZE[0], -1, -1): # 8부터 0까지 1씩 감소
        for x in range(BLOCK_SIZE[0]):   # 0부터 7까지 1씩 증가
            # 고양이가 있어야 하고, 아래로 내려갈 때 비어 있으면
            if cat_loc[y][x] != 0 and cat_loc[y + 1][x] == 0:
                cat_loc[y + 1][x] = cat_loc[y][x] # 아래로 내려가
                cat_loc[y][x] = 0 # 원래 있던 곳은 고양이가 없어짐

def draw_cat(): # 고양이를 그리는 함수
    for y in range(BLOCK_SIZE[1]):
        for x in range(BLOCK_SIZE[0]):
            if cat_loc[y][x] > 0:
                cvs.create_image(x*BLOCK_PIXEL + 60, y*BLOCK_PIXEL + 60, \
                    image=IMG_CAT[cat_loc[y][x]], tag="CAT")
```

6-5. 낙하 알고리즘 구현

- 알고리즘: 고양이가 아래로 떨어지는 상황 구현

```
def main_action(): # 실시간 처리 함수
    global cursor_x, cursor_y
    drop_cat()
    if CUR_LIMIT <= mouse_x < CUR_LIMIT + BLOCK_PIXEL * BLOCK_SIZE[0] \
        and CUR_LIMIT <= mouse_y < CUR_LIMIT + BLOCK_PIXEL * BLOCK_SIZE[1]:
        cursor_x = int((mouse_x - CUR_LIMIT) / BLOCK_PIXEL)
        cursor_y = int((mouse_y - CUR_LIMIT) / BLOCK_PIXEL)

    cvs.delete("CAT")
    cvs.delete("CURSOR")
    cvs.create_image(cursor_x * BLOCK_PIXEL + 60, cursor_y * BLOCK_PIXEL + 60,\
                     image=cursor, tag="CURSOR")
    draw_cat() # 실시간으로 계속 고양이 위치를 찾아 그려야 함
    root.after(100, main_action)
```

6-5. 낙하 알고리즘 구현

- 알고리즘: 클릭해서 고양이를 떨어뜨리는 상황 구현

```
cat_loc = [
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0]
]
```

6-5. 낙하 알고리즘 구현

- 알고리즘: 클릭해서 고양이를 떨어뜨리는 상황 구현

```
cursor_x = 0  # 커서의 좌표
cursor_y = 0
mouse_x = 0   # 실제 마우스 좌표
mouse_y = 0
mouse_c = 0   # 실제 마우스 클릭 여부

... 생략 ...

def mouse_press(e): # 마우스 클릭 시 이벤트 함수
    global mouse_c
    mouse_c = 1

... 생략 ...
root.bind("<Motion>", mouse_move) # 마우스 이벤트 처리
root.bind("<ButtonPress>", mouse_press)
main_action()  # 실시간 처리 함수 시작점
root.mainloop()
```

6-5. 낙하 알고리즘 구현

- 알고리즘: 클릭해서 고양이를 떨어뜨리는 상황 구현

```
def main_action(): # 실시간 처리 함수
    global cursor_x, cursor_y, mouse_c
    drop_cat()
    if CUR_LIMIT <= mouse_x < CUR_LIMIT + BLOCK_PIXEL * BLOCK_SIZE[0] \
        and CUR_LIMIT <= mouse_y < CUR_LIMIT + BLOCK_PIXEL * BLOCK_SIZE[1]:
        cursor_x = int((mouse_x - CUR_LIMIT) / BLOCK_PIXEL)
        cursor_y = int((mouse_y - CUR_LIMIT) / BLOCK_PIXEL)
        if mouse_c == 1: # 마우스를 클릭했으면
            mouse_c = 0  # 마우스 클릭 변수 초기화
            # 커서 위치 칸에 무작위 고양이 놓기
            cat_loc[cursor_y][cursor_x] = random.randint(1, 6)
         ... 생략 ...
```


랜덤하게 고양이가 떨어져야 하므로
import random**을 하고**
1~6**번까지 무작위 고르기!**

6-6. 판정 알고리즘 구현

- 알고리즘: 고양이가 3마리 모일 때 판정
  - cat_loc 리스트를 간단히 초기화 하는 방법 (list comprehension)

```
cat_loc = []
for i in range(BLOCK_SIZE[1]):
    cat_loc.append([0*x for x in range(BLOCK_SIZE[0])])
```

append: **리스트에 추가** $0 \times x \leftarrow x \in \{0, 1, 2, 3, 4, 5, 6, 7\}$

  - 가로로 나란히 있는 3마리 고양이 판별법: 상대좌표 이용

cat[y][x - 1]cat[y][x] cat[y][x + 1]

6-6. 판정 알고리즘 구현

- 알고리즘: 고양이가 3마리 모일 때 판정
  - 가로로 나란히 있는 3마리 고양이 판별법: 상대좌표 이용

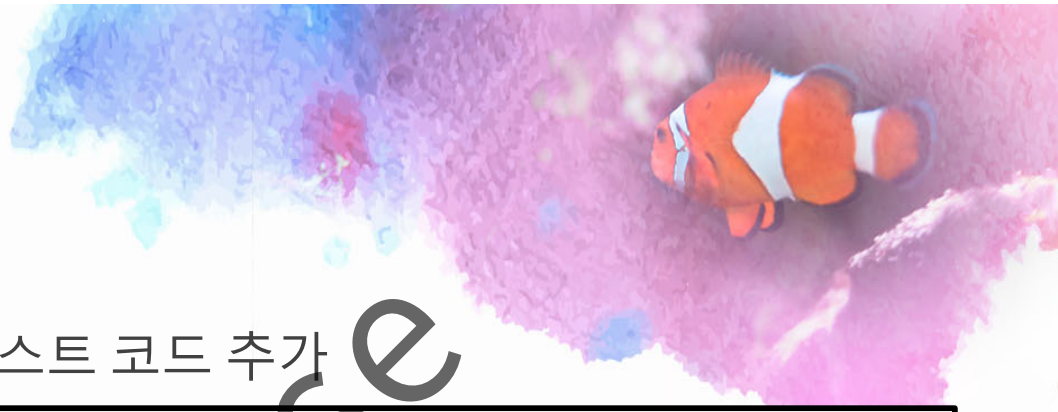
```
... 생략 ...

def decide_cat():  # 고양이 3마리 판별
    for y in range(BLOCK_SIZE[1]):
        for x in range(1, BLOCK_SIZE[0] - 1): # x는 1~7
            # 가로 방향 3개가 놓여 있으면
            if cat_loc[y][x] > 0: # 고양이가 아닌 경우 제외
                if cat_loc[y][x - 1] == cat_loc[y][x] == cat_loc[y][x + 1]:
                    cat_loc[y][x - 1] = cat_loc[y][x] = cat_loc[y][x + 1] = 7
                    # 발자국으로 변경
```

6-6. 판정 알고리즘 구현

- 알고리즘: 고양이가 3마리 모일 때 판정
  - 가로로 나란히 있는 3마리 고양이 판별 - 테스트 코드 추가

```
... 생략 ...
def main_action(): # 실시간 처리 함수
    global cursor_x, cursor_y, mouse_c
    #drop_cat()
    # 테스트 코드: 풍선을 클릭하면
    if 660 <= mouse_x < 840 and 100 <= mouse_y < 160 and mouse_c == 1:
        mouse_c = 0
        decide_cat() # 고양이 3마리 판별 코드 작동

    if CUR_LIMIT <= mouse_x < CUR_LIMIT + BLOCK_PIXEL * BLOCK_SIZE[0] \
        and CUR_LIMIT <= mouse_y < CUR_LIMIT + BLOCK_PIXEL * BLOCK_SIZE[1]:
        ... 생략 ...
        if mouse_c == 1: # 마우스를 클릭했으면
            mouse_c = 0  # 마우스 클릭 변수 초기화
            # 커서 위치 칸에 무작위 고양이 놓기
            #cat_loc[cursor_y][cursor_x] = random.randint(1, 6)
            # 고양이 3마리 판별 테스트 코드
            cat_loc[cursor_y][cursor_x] = random.randint(1, 2)
... 생략 ...
```

6-6. 판정 알고리즘 구현

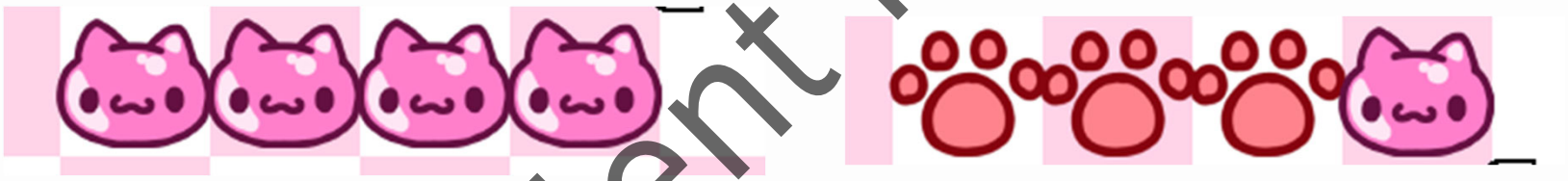


- 알고리즘: 고양이가 3마리 모일 때 판정
  - 가로로 나란히 있는 3마리 고양이 판별 - 테스트 코드 추가

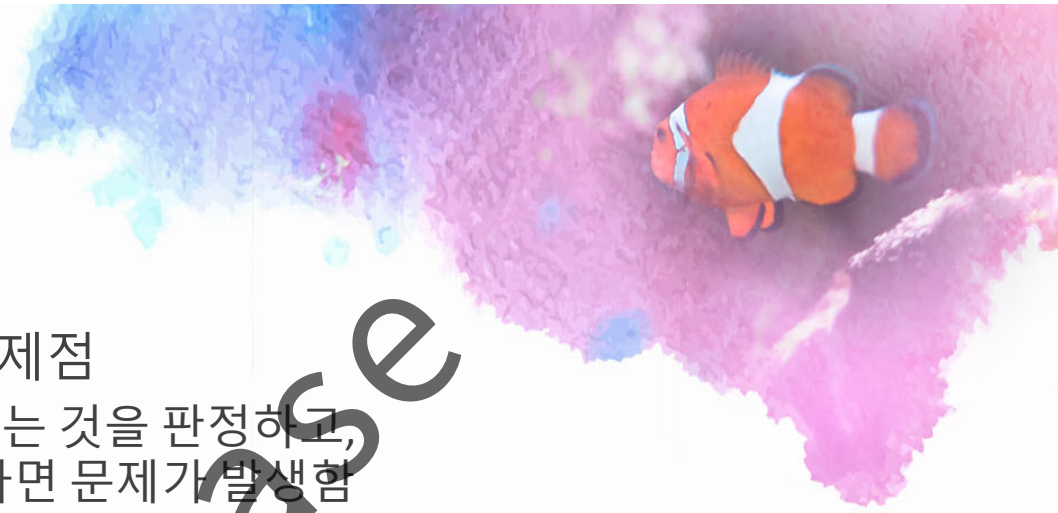
```
... 생략 ...
# 경로 설정 잘해야 함!
background = tk.PhotoImage(file="resource/neko_bg.png")
cursor = tk.PhotoImage(file="resource/neko_cursor.png")
cvs.create_image(BG_SIZE[0] // 2, BG_SIZE[1] // 2, image=background)
# 테스트 코드 (고양이 3마리 판별)
cvs.create_rectangle(660, 100, 840, 160, fill="white")
cvs.create_text(750, 130, text="test", fill="red", font=("Times New Roman", 30))

root.bind("<Motion>", mouse_move) # 마우스 이벤트 처리
root.bind("<ButtonPress>", mouse_press)
main_action()  # 실시간 처리 함수 시작점
root.mainloop()
```

6-6. 판정 알고리즘 구현

- 알고리즘: 고양이가 3마리 모일 때 판정
  - 가로로 나란히 있는 3마리 고양이 판별 - 문제점
    - 가로로 3개가 나란히 있을 경우 판별에는 문제가 없음
    - 마찬가지 방법으로 세로 방향, 대각선 방향으로 해도 됨
    - 다만, 4마리 이상의 경우 처리가 없음
      → 고양이가 한마리 남음!



6-6. 판정 알고리즘 구현



- 알고리즘: 고양이가 3마리 모일 때 판정
  - 가로로 나란히 있는 3마리 고양이 판별 - 문제점
    - 또한, 고양이가 가로로 3마리 나란히 놓여있는 것을 판정하고, 이후 세로로 나란히 놓여있는 것을 판정한다면 문제가 발생함

```
def decide_cat():  # 고양이 3마리 판별
    for y in range(BLOCK_SIZE[1]):
        for x in range(1, BLOCK_SIZE[0] - 1): # x는 1~7
            # 가로 방향 3개가 놓여 있으면
            if cat_loc[y][x] > 0: # 고양이가 아닌 경우 제외
                if cat_loc[y][x - 1] == cat_loc[y][x] == cat_loc[y][x + 1]:
                    cat_loc[y][x - 1] = cat_loc[y][x] = cat_loc[y][x + 1] = 7
                    # 발자국으로 변경

    # 세로 판정 코드
    for y in range(1, BLOCK_SIZE[1] - 1): # y는 1~9
        for x in range(BLOCK_SIZE[0]):
            if cat_loc[y][x] > 0:
                if cat_loc[y - 1][x] == cat_loc[y][x] == cat_loc[y + 1][x]:
                    cat_loc[y - 1][x] = cat_loc[y][x] = cat_loc[y + 1][x] = 7
```

6-6. 판정 알고리즘 구현

- 알고리즘: 고양이가 3마리 모일 때 판정
  - 가로로 나란히 있는 3마리 고양이 판별 - 문제점
    - 또한, 고양이가 가로로 3마리 나란히 놓여있는 것을 판정하고, 이후 세로로 나란히 놓여있는 것을 판정한다면 문제가 발생함

6-6. 판정 알고리즘 구현

- 알고리즘: 고양이가 3마리 모일 때 판정 - **올바르게 대체**
  - 판정용 리스트를 하나 복사하고 이를 통해 판정해야 함!
    → 가로, 세로, 대각선 모두를 반영해야 하기 때문

```
cat_loc = []
check = []    # 판정용 리스트 추가
for i in range(BLOCK_SIZE[1]):
    cat_loc.append([0*x for x in range(BLOCK_SIZE[0])])
    check.append([0*x for x in range(BLOCK_SIZE[0])])
```

6-6. 판정 알고리즘 구현

- 알고리즘: 고양이가 3마리 모일 때 판정 - **올바르게 대체**
  - 판정용 리스트를 하나 복사하고 이를 통해 판정해야 함!
    → 가로, 세로, 대각선 모두를 반영해야 하기 때문

```
def decide_cat():  # 고양이 3마리 판별
    # 판정용 리스트로 복사하기 (주의: check = cat_loc로 안됨!)
    for y in range(BLOCK_SIZE[1]):
        for x in range(BLOCK_SIZE[0]):
            check[y][x] = cat_loc[y][x]

    # 가로 판정 코드
    for y in range(BLOCK_SIZE[1]):
        for x in range(1, BLOCK_SIZE[0] - 1): # x는 1~7
            if check[y][x] > 0: # 고양이가 아닌 경우 제외
                if check[y][x - 1] == check[y][x] == check[y][x + 1]:
                    cat_loc[y][x - 1] = cat_loc[y][x] = cat_loc[y][x + 1] = 7

    ... 다음 장 이어서 ...
```

6-6. 판정 알고리즘 구현

- 알고리즘: 고양이가 3마리 모일 때 판정 - **올바르게 대체**
  - 판정용 리스트를 하나 복사하고 이를 통해 판정해야 함!
    → 가로, 세로, 대각선 모두를 반영해야 하기 때문

```
    # 세로 판정 코드
    for y in range(1, BLOCK_SIZE[1] - 1): # y는 1~9
        for x in range(BLOCK_SIZE[0]):
            if check[y][x] > 0:
                if check[y - 1][x] == check[y][x] == check[y + 1][x]:
                    cat_loc[y - 1][x] = cat_loc[y][x] = cat_loc[y + 1][x] = 7

    # 대각선 판정 코드
    for y in range(1, BLOCK_SIZE[1] - 1): # y는 1~9
        for x in range(1, BLOCK_SIZE[0] - 1): # x는 1~7
            if check[y][x] > 0:
                if check[y - 1][x - 1] == check[y][x] == check[y + 1][x + 1]:
                    cat_loc[y - 1][x - 1] = cat_loc[y][x] = cat_loc[y + 1][x + 1] = 7

                if check[y + 1][x - 1] == check[y][x] == check[y - 1][x + 1]:
                    cat_loc[y + 1][x - 1] = cat_loc[y][x] = cat_loc[y - 1][x + 1] = 7
```

6-7. 타이틀과 게임 오버 화면 구현

- 타이틀 화면과 게임 오버 화면 처리
  - index라는 변수로 게임의 처음부터 끝까지 흐름 관리
    - 0: 타이틀 화면을 문자로 표시하고 1로 이동함
    - 1: 게임 시작을 위한 입력을 기다림
      화면을 클릭하면 가장 먼저 떨어질 고양이를 셋하고
      2로 이동함
    - 2: [게임 중 처리 1] 고양이를 떨어뜨림
      → 모든 고양이가 떨어지면 3으로 이동
    - 3: [게임 중 처리 2] 고양이가 3개 이상 나란히 있는지 확인
      → 나란히 있다면, 발자국으로 변경. 4로 이동
    - 4: [게임 중 처리 3] 발자국으로 변환된 칸이 있으면,
      삭제하고 점수에 더한 뒤, 2로 이동
      발자국으로 변환된 칸이 없다면,
      가장 위쪽 칸까지 쌓여 있지 않다면,
      5로 입력 처리 대기 이동
      가장 위쪽 칸까지 쌓여있다면 6으로 이동

6-7. 타이틀과 게임 오버 화면 구현

- 타이틀 화면과 게임 오버 화면 처리
  - index라는 변수로 게임의 처음부터 끝까지 흐름 관리
    - 5: [게임 중 처리 4] 플레이어 입력 대기
      → 마우스 커서를 이동해 클릭하고 고양이를 놓으면 2로 고양이를 떨어뜨리는 처리로 이동
    - 6: 게임 오버 화면
      → 변수로 시간 카운트하고 5초 후 0으로 이동

```
root = tk.Tk()

index = 0  # 게임 진행 관련 인덱스
timer = 0  # 타이머
score = 0  # 점수
next_cat = 0  # 다음에 놓을 고양이
```

6-7. 타이틀과 게임 오버 화면 구현

- 타이틀 화면과 게임 오버 화면 처리
  - 발자국 제거 함수

```
def footprint_delete():  # 발자국 제거
    num = 0  # 지워야 할 발자국 카운트 (점수 계산 위해)
    for y in range(BLOCK_SIZE[1]):
        for x in range(BLOCK_SIZE[0]):
            if cat_loc[y][x] == 7:
                cat_loc[y][x] = 0
                num += 1

    return num
```

6-7. 타이틀과 게임 오버 화면 구현

- 타이틀 화면과 게임 오버 화면 처리
  - 고양이가 떨어지는 함수 변경

```
def drop_cat(): # 고양이가 위에서 떨어져야 함
    flag = False  # 낙하 여부 (떨어지지 않았음)
    for y in range(BLOCK_SIZE[0], -1, -1): # 8부터 0까지 1씩 감소
        for x in range(BLOCK_SIZE[0]):  # 0부터 7까지 1씩 증가
            # 고양이가 있어야 하고, 아래로 내려갈 때 비어 있으면
            if cat_loc[y][x] != 0 and cat_loc[y + 1][x] == 0:
                cat_loc[y + 1][x] = cat_loc[y][x] # 아래로 내려가
                cat_loc[y][x] = 0 # 원래 있던 곳은 고양이가 없어짐
                flag = True  # 고양이가 떨어진 것임
    return flag
```

만약 모든 고양이가 떨어지면(떨어질 고양이가 없으면) 3으로 이동해야 함

6-7. 타이틀과 게임 오버 화면 구현

- 타이틀 화면과 게임 오버 화면 처리
  - 고양이가 맨 윗 줄까지 도달했는지 여부 확인

```
def over_cat_line():  # 맨 윗줄에 도달했는가?
    for x in range(BLOCK_SIZE[0]):
        if cat_loc[0][x] > 0: # 맨 윗줄이면 y가 0일 때임
            return True
    return False
```

  - 맨 윗 줄에 고양이를 놓을 경우

```
def top_line_cat(): # 맨 윗 줄에 고양이를 놓는 경우
    for x in range(BLOCK_SIZE[0]):
        cat_loc[0][x] = random.randint(0, 6)
```

  - 저번에 작성한 테스트 코드 주석 처리하기

6-7. 타이틀과 게임 오버 화면 구현

- 타이틀 화면과 게임 오버 화면 처리
  - 텍스트 추가 코드 자동화로 만들기(함수화)

```
def draw_txt(txt, x, y, size, color, tag):
    fnt = ("Times New Roman", size, "bold")
    cvs.create_text(x + 2, y + 2, text=txt, fill="black", font=fnt, tag=tag) # 그림자
    cvs.create_text(x, y, text=txt, fill=color, font=fnt, tag=tag)
```

6-7. 타이틀과 게임 오버 화면 구현

- 타이틀 화면과 게임 오버 화면 처리
  - 게임 인덱스 조정으로 흐름 처리 (main_action 함수 변경)

```
def main_action(): # 실시간 처리 함수
    global cursor_x, cursor_y, mouse_c
    global index, timer, score, next_cat
    #drop_cat()
    # 테스트 코드: 풍선을 클릭하면
    #if 660 <= mouse_x < 840 and 100 <= mouse_y < 160 and mouse_c == 1:
    #    mouse_c = 0
    #    decide_cat() # 고양이 3마리 판별 코드 작동

    if index == 0: # 타이틀 로고
        draw_txt("Cat game!", 312, 240, 100, "violet", "TITLE")
        draw_txt("Click to start.", 312, 560, 50, "orange", "TITLE")
        index = 1
        mouse_c = 0

... 다음 장 이어서 ...
```

6-7. 타이틀과 게임 오버 화면 구현

- 타이틀 화면과 게임 오버 화면 처리
  - 게임 인덱스 조정으로 흐름 처리 (main_action 함수 변경)

```
    elif index == 1: # 타이틀 화면, 시작 대기
        if mouse_c == 1: # 클릭하면
            # 고양이 위치 관리 초기화
            for y in range(BLOCK_SIZE[1]):
                for x in range(BLOCK_SIZE[0]):
                    cat_loc[y][x] = 0

            mouse_c = 0
            score = 0
            next_cat = 0
            cursor_x = cursor_y = 0
            top_line_cat()
            draw_cat()
            cvs.delete("TITLE") # 타이틀 제거
            index = 2
    elif index == 2: # 고양이 낙하
        if drop_cat() == False: # 낙하할 고양이가 없으면
            index = 3
        draw_cat()
```

6-7. 타이틀과 게임 오버 화면 구현

- 타이틀 화면과 게임 오버 화면 처리
  - 게임 인덱스 조정으로 흐름 처리 (main_action 함수 변경)

```
    elif index == 3: # 고양이가 나란히 놓였는지 판정
        decide_cat()
        draw_cat()
        index = 4
    elif index == 4: # 나란히 놓인 고양이가 있으면 삭제
        sc = footprint_delete()  # 발자국 삭제하고, 몇 개 발자국이 없어졌는지 담기
        score += sc*10  # 발자국 1개당 10점씩 추가
        if sc > 0:    # 삭제한 발자국이 있다면
            index = 2
        else:
            if over_cat_line() == False: # 가장 윗줄에 도달하지 않으면
                next_cat = random.randint(1, 6) # 다음 고양이를 랜덤하게 정함
                index = 5
            else:  # 가장 윗줄에 도달했다면
                index = 6  # 게임 오버
                timer = 0  # 타이머 초기화

        draw_cat()
```

6-7. 타이틀과 게임 오버 화면 구현

- 타이틀 화면과 게임 오버 화면 처리
  - 게임 인덱스 조정으로 흐름 처리 (main_action 함수 변경)

```
    elif index == 5: # 마우스 입력 대기
        if CUR_LIMIT <= mouse_x < CUR_LIMIT + BLOCK_PIXEL * BLOCK_SIZE[0] \
            and CUR_LIMIT <= mouse_y < CUR_LIMIT + BLOCK_PIXEL * BLOCK_SIZE[1]:
            cursor_x = int((mouse_x - CUR_LIMIT) / BLOCK_PIXEL)
            cursor_y = int((mouse_y - CUR_LIMIT) / BLOCK_PIXEL)
            if mouse_c == 1: # 마우스를 클릭했으면
                mouse_c = 0  # 마우스 클릭 변수 초기화
                # 커서 위치 칸에 무작위 고양이 놓기
                #cat_loc[cursor_y][cursor_x] = random.randint(1, 6)
                # 고양이 3마리 판별 테스트 코드
                #cat_loc[cursor_y][cursor_x] = random.randint(1, 2)
                top_line_cat() # 가장 윗줄 고양이 설정
                cat_loc[cursor_y][cursor_x] = next_cat # 현재 커서에 고양이 놓기
                next_cat = 0
                index = 2

        cvs.delete("CURSOR")
        cvs.create_image(cursor_x * BLOCK_PIXEL + 60, cursor_y * BLOCK_PIXEL + 60,\
                         image=cursor, tag="CURSOR")
        draw_cat()
```

6-7. 타이틀과 게임 오버 화면 구현

- 타이틀 화면과 게임 오버 화면 처리
  - 게임 인덱스 조정으로 흐름 처리 (main_action 함수 변경)

```
    #cvs.delete("CAT")
    #cvs.delete("CURSOR")
    #cvs.create_image(cursor_x * BLOCK_PIXEL + 60, cursor_y * BLOCK_PIXEL + 60,\
    #                  image=cursor, tag="CURSOR")
    #draw_cat() # 실시간으로 계속 고양이 위치를 찾아 그려야 함
    cvs.delete("INFO")  # 점수표시 삭제
    draw_txt("SCORE: " + str(score), 160, 60, 32, "blue", tag="INFO")
    if next_cat > 0: # 다음에 배치할 고양이가 있다면
        cvs.create_image(752, 128, image=IMG_CAT[next_cat], tag="INFO")
    root.after(100, main_action)
```

6-7. 타이틀과 게임 오버 화면 구현

- 조정 작업: 맨 윗 줄에서 고양이가 6마리나 낙하!
  → 매우 어려운 게임이 됨
  - 최고 점수 관리
  - 난이도 변경 (Easy 4개, Normal 5개, Hard 6개)

  - game_main_adv.py에 있음

참고 자료

- 윤성우, 열혈 파이썬(기본편/중급편), 오렌지미디어, 2017.
- 히로세 츠요시, 파이썬으로 배우는 게임 개발 (입문편/실전편), 제이펍, 2020.
- 폴 데이텔, 하비 데이텔, 안진섭, 프로그래머를 위한 Python, 성안당, 2021.
- 루시아누 하말류, 전문가를 위한 파이썬, O'REILLY, 2016.
- 브렛 슬라킨, Effective Python, 2nd edition, 길벗, 2020.