2024.06.07

0475 프로그래밍 기초

14주차 팀 프로젝트 발표

– tkinter를 활용한 스네이크 게임

팀 K

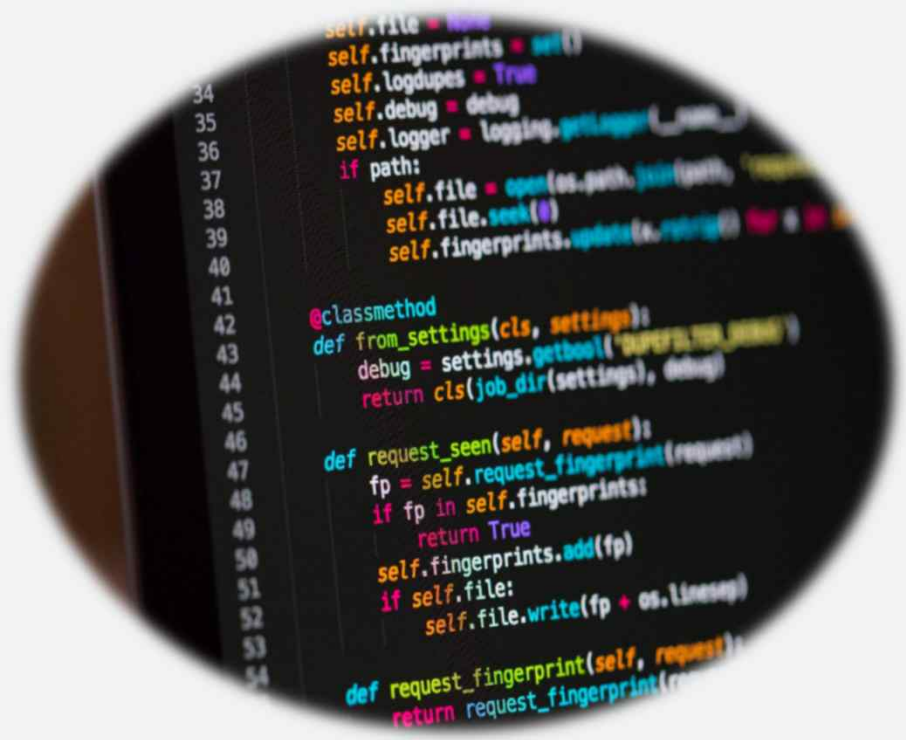
김찬영, 쩐꾸앙둥, 장수민, 김효찬

Contents

주제 선정 이유

이론 강의시간에 학습한 파이썬의 반복문, 함수, 리스트, 딕셔너리 등의 기능과 실습 시간에 실제 사용해본 tkinter 등의 라이브러리를 종합적으로 활용할 수 있는 주제를 선택하고자 함.

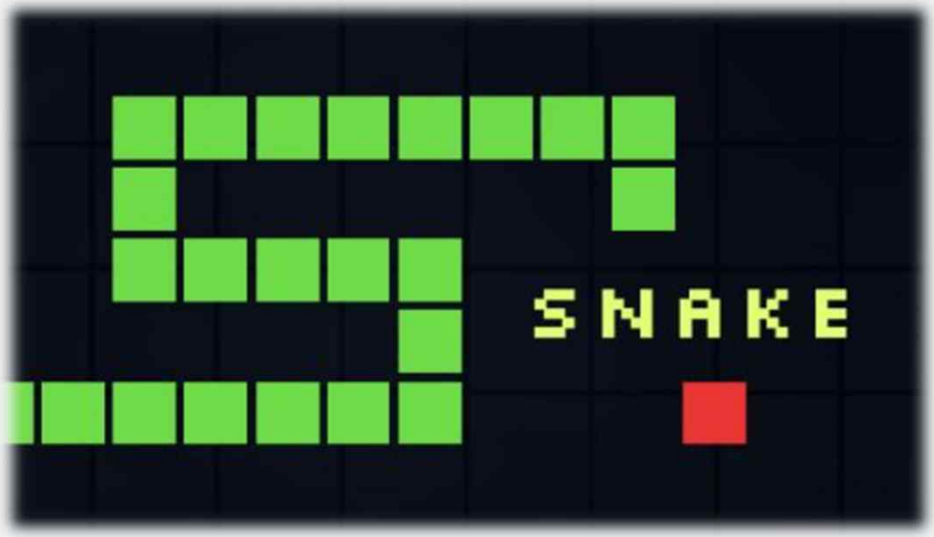
간단하고 재미있게 만들 수 있는 게임 GUI 중에서 상기한 조건에 부합하는 ‘스네이크 게임’ 을 선정하여 제작함.



프로그램 소개

스네이크 게임 기본 규칙

- 뱀은 멈추지 않고 끊임없이 이동
- 먹이를 먹으면 뱀의 길이 증가
- 한정된 크기의 공간, 뱀이 경계에 부딪히면 종료
- 뱀이 자기 자신에게 부딪히면 종료



추가로 구현한 기능

- 시간 제한 + 점수제(먹이)

→ 제한 시간 내 최대한 많은 점수를 올리는 게임성 부여

- 공간 내 장애물 생성, 뱀이 장애물에 부딪히면 종료
- 쉬움, 보통, 어려움의 난이도 차등 구현

소스코드

〈코드1. library and global variables〉

```
from tkinter import *
import random

width, height = 700, 700  # 캔버스 크기
rSize = 35  # 객체 크기
```

라이브러리 및 전역 변수 선언

- tkinter 라이브러리를 사용하여 GUI 생성
- random 라이브러리를 사용하여 음식을 랜덤한 위치에 생성
- width 와 height로 캔버스의 크기를 700x700로 정의
- rSize로 뱀과 음식의 크기를 35x35로 정의

소스코드

〈코드2. game_main()〉

```
def game_main():
    global snake, snake_direction, food, score, running, obstacles, time_left, img
    snake = [(70, 70), (60, 70)]  # 뱀 초기 위치 설정
    snake_direction = "Right"  # 초기 방향 설정
    food = None
    obstacles = []  # 장애물 리스트
    time_left = 30  # 타이머 초기화

    create_food()  # 음식 생성

    # 점수 초기화
    score = 0
    update_score()
    update_timer()  # 타이머 업데이트

    # 게임 상태 플래그
    running = True

    # 이미지 로드 (전역 변수로 설정)
    if 'img' not in globals():
        img = PhotoImage(file="C:/프로그래밍기초 팀프로젝트/Snake Game/KGU.gif")

    # 캔버스에 이미지 삽입
    canvas.create_image(590, 40, anchor=NW, image=img)
```

게임 초기화 함수

게임 시작시 다음의 값 설정

- 뱀의 초기 위치(snake)
- 뱀의 초기이동방향(snake_direction)
- 게임 상태 플래그(running)

게임 시작시 다음의 값 초기화

- 음식 생성 변수(food)
- 장애물 생성 리스트(obstacles)
- 시간 제한을 위한 타이머(time_left)
- 점수(score)

배경화면에 이미지 추가

- 이미지를 로드하여 캔버스에 삽입

소스코드

〈코드3. create_food())〉

```
def create_food():
    global food
    # 기존 음식 삭제
    if food is not None:
        canvas.delete(food)
        food = None
    # 새로운 음식 위치 설정
    while True:
        x = random.randint(0, 19) * rSize
        y = random.randint(0, 19) * rSize
        if (x, y) not in snake and (x, y) not in obstacles: # 좌표가 뱀의 몸통이나 장애물 좌표와 겹치지 않을 때 반복문 종료
            food = canvas.create_rectangle(x, y, x + rSize, y + rSize, fill="red")
            # x, y 랜덤 좌표에 rSize 크기인 사각형 먹이 객체 생성
            break
```

음식 생성 함수

- 기존 음식을 삭제하고 새로운 위치에 음식 생성
- 뱀이나 장애물과 겹치지 않는 위치에 음식 생성

소스코드

〈코드4. create_obstacles(count)〉

```
def create_obstacles(count):
    global obstacles
    # 기존 장애물 삭제 (게임 재시작시)
    for obstacle in obstacles:
        canvas.delete(obstacle)
    obstacles = [] # 장애물 리스트 생성
    # 장애물 생성
    for _ in range(count):
        while True:
            x = random.randint(0, 17) * rSize
            y = random.randint(0, 17) * rSize

            # 장애물이 뱀이나 음식과 겹치지 않도록
            if (x, y) not in snake and (x, y) != canvas.coords(food)[:2]:
                obstacle = canvas.create_rectangle(x, y, x + rSize, y + rSize, fill="gray")
                obstacles.append((x, y))  # 장애물 리스트에 좌표 저장
                break
```

장애물 생성 함수

- 장애물의 위치가 뱀과 음식의 위치와 겹치지 않도록 생성
- count 인수를 통해 함수가 호출될 때마다 다른 개수의 장애물 생성
- 선택한 난이도에 따라 장애물 생성 개수 조절

소스코드

〈코드5. update_score(), update_timer()〉

```
def update_score():
    global score_text
    # 점수 업데이트 시 객체 삭제하고 다시 생성
    if 'score_text' in globals():
        canvas.delete(score_text)

    # 점수 텍스트 업데이트
    score_text = canvas.create_text(50, 20, fill="black", text="점수: " + str(score), font=("Helvetica", 16))


def update_timer():
    global timer_text, time_left
    # 타이머 업데이트 시 객체 삭제하고 다시 생성
    if 'timer_text' in globals():
        canvas.delete(timer_text)

    # 타이머 텍스트 업데이트
    timer_text = canvas.create_text(600, 20, fill="black", text="남은 시간: " + str(time_left), font=("Helvetica", 16))
```

점수 및 타이머 업데이트 함수

- 게임이 진행되는 동한 점수와 타이머 업데이트

소스코드

〈코드6. countdown()〉

```
def countdown():
    global time_left, running, timer_id
    if running and time_left > 0: # 실행중일 때와 남은 시간이 1초 이상일 때 시간 감소
        time_left -= 1
        update_timer()
        timer_id = window.after(1000, countdown)
    elif time_left == 0: # 남은 시간이 0초일 때 게임 종료
        end_game()
```

타이머 함수

- 게임에 시간 제한 부여
- 1초마다 타이머를 감소시키고, 부여된 시간이 전부 소요되면 게임 종료

소스코드

〈코드7. end_game()〉

```
def end_game():
    global running
    running = False
    canvas.create_text(350, 300, fill="blue", text="게임 종료!", font=("Helvetica", 32))  # 중앙에 위치시키기
    canvas.create_text(350, 360, fill="blue", text="점수: " + str(score), font=("Helvetica", 24))  # 중앙에 위치
    canvas.create_text(350, 420, fill="blue", text="재시작하려면 R키를 누르세요", font=("Helvetica", 24))  # 중앙에 위치
```

게임 종료 함수

- 게임 종료 후 종료 메시지 출력
- 화면 크기가 700x700 이므로, 메시지를 중앙에 출력하기 위해 좌표를 (350, 350)으로 설정

소스코드

〈코드8. change_direction(event)〉

```
def change_direction(event):
    global snake_direction
    # 키보드 입력을 받아 뱀의 방향을 변경
    new_direction = event.keysym # 키 입력 받음
    all_directions = {"Up", "Down", "Left", "Right"} # 가능한 방향 정의
    opposites = {"Up": "Down", "Down": "Up", "Left": "Right", "Right": "Left"}
    # 현재 이동 방향의 반대 방향을 쉽게 확인하기 위한 딕셔너리 정의

    # 현재 방향과 반대 방향이 아닌지 확인
    if new_direction in all_directions and new_direction != opposites[snake_direction]:
        snake_direction = new_direction
```

뱀 방향 변경 함수

- event를 인수로 하여, 키보드 입력에 따라 뱀의 이동 방향을 변경하는 기능 구현

소스코드

〈코드9. restart_game(event)〉

```
def restart_game(event):
    # 게임 재시작 시 모든 요소 초기화
    global current_obstacle_count, timmer_id
    window.after_cancel(timer_id) # 카운트다운 정지
    canvas.delete("all")
    game_main()
    countdown()
    create_obstacles(current_obstacle_count)
```

게임 재시작 함수

- timer_id 변수를 통해 카운트다운 함수를 정지
- 캔버스 요소를 전부 삭제하고 game_main() 함수를 호출해 캔버스 요소 초기화와 동시에 게임 다시 실행
- countdown(), create_obstacles(current_obstacle_count) 함수를 호출해 다시 카운트다운을 실행하고 장애물 랜덤 생성

소스코드

〈코드10-1. move_snake()〉

```
def move_snake():
    global running, score, food
    # 뱀의 머리 위치 업데이트
    head_x, head_y = snake[0]
    if snake_direction == "Up":
        new_head = (head_x, head_y - rSize)
    elif snake_direction == "Down":
        new_head = (head_x, head_y + rSize)
    elif snake_direction == "Left":
        new_head = (head_x - rSize, head_y)
    elif snake_direction == "Right":
        new_head = (head_x + rSize, head_y)

    # 뱀의 위치 업데이트
    snake.insert(0, new_head)
    snake.pop()

    # 충돌 확인
    if check_collision():
        end_game()
        return

        .
        .
```

뱀 이동 함수(1)

- 뱀의 머리를 새로운 위치로 업데이트

  → 뱀을 이동시킴

뱀이 벽이나 자신과 충돌했는지 확인

- check_collision() 함수로 충돌 확인 시,

  → end_game() 함수를 호출하여 게임 종료 및 함수 실행 중단

소스코드

〈코드10-2.move_snake()〉

```
        .
        .
    # 뱀이 음식을 먹었는지 확인
    if canvas.coords(food)[:2] == [new_head[0], new_head[1]]:

    # coords로 food의 x, y좌표를 가져와 뱀의 머리 좌표와 비교
        snake.append(snake[-1])  # 뱀의 길이 증가
        canvas.delete(food)  # 기존 음식 삭제
        food = None
        create_food()  # 새로운 음식 생성
        score += 1  # 점수 증가
        update_score()  # 점수 업데이트

    # 뱀을 화면에 그리기
    canvas.delete("snake")
    for element in snake:
        canvas.create_rectangle(element[0], element[1], element[0] + rSize, element[1] + rSize, fill="green", tag="snake")
```

뱀 이동 함수(2)

- 뱀의 머리 위치가 음식 위치와 같는지 확인하여 만약 같다면,

  → 뱀의 길이 증가, 좌표값이 머리의 좌표와 겹친 기존의 음식은 삭제, 새로운 음식 생성, 점수 증가

- 뱀이 움직이는 것처럼 연출하기 위하여 뱀을 삭제하고 다시 그리기를 반복

소스코드

〈코드11. check_collision()〉

```
def check_collision():
    head_x, head_y = snake[0]
    # 벽이나 자기 자신과의 충돌 여부 확인
    return (
        head_x < 0 or head_x >= width or head_y < 0 or head_y >= height or
        (head_x, head_y) in snake[1:] or
        (head_x, head_y) in obstacles  # 장애물과의 충돌 확인
    )
```

충돌 확인 함수

- 뱀이 벽, 자기 자신 혹은 장애물과 충돌하였는지 확인

소스코드

〈코드12. game_loop(speed) & start_game(speed, obstacle_count)〉

```
def game_loop(speed):
    if running:
        move_snake()
    window.after(speed, game_loop, speed) # speed 밀리초후에 재귀적으로 game_loop 다시 실행

def start_game(speed, obstacle_count):
    global current_obstacle_count
    current_obstacle_count = obstacle_count # 현재 장애물 개수를 저장할 변수
    game_loop(speed)
    create_obstacles(obstacle_count)  # 장애물 생성
    countdown()  # 타이머 시작
    canvas.delete(button1_window)
    canvas.delete(button2_window)
    canvas.delete(button3_window)
```

게임 루프 및 시작 함수

- game_loop(speed) 함수로 speed를 인수로 받아 게임 루프의 반복 속도 제어

  → move_snake() 함수로 뱀의 이동 처리, speed를 통해 일정 시간 간격으로 게임 루프 반복 호출
- start_game(speed, obstacle_count) 함수로 게임 시작 시 장애물 생성, 타이머 시작, 게임 루프 및 초기화 버튼 삭제의 초기 설정 실행

  → speed를 인수로 받아 뱀의 속도 제어, obstacle_count를 인수로 받아 장애물 개수 제어
- 두 함수 모두 인수를 받아 난이도 별로 제어하여 게임 실행

소스코드

〈코드13-1.main window settings〉

```
# 게임 실행
window = Tk()
window.title("스네이크 게임")
canvas = Canvas(window, bg="white", width=700, height=700)  # 캔버스 생성
canvas.pack()  # 생성한 캔버스 윈도우에 배치

# 이미지 로드
img = PhotoImage(file="C:/프로그래밍기초 팀프로젝트/Snake Game/KGU.gif")

# 캔버스에 이미지 삽입
canvas.create_image(590, 40, anchor=NW, image=img)

game_main()  # 게임 초기화
window.bind("<KeyPress>", change_direction)  # 키보드 입력 처리
window.bind("<KeyPress-r>", restart_game)  # 게임 재시작 처리
          .
          .
```

메인 윈도우 설정(1)

- tkinter 윈도우와 캔버스의 제목, 크기와 색을 설정하고 이미지를 좌측 상단에 삽입
- game_main() 함수를 호출하여 게임을 초기화
- 임의의 키가 입력될 때 change_direction() 함수 호출, 'R' 키가 입력될 때 restart_game() 함수 호출

소스코드

〈코드13-2.main window settings〉

```
        .
        .

# 버튼 1 생성
button1 = Button(window, text="쉬움", command=lambda: start_game(125, 3), width=8, height=3)
button1_window = canvas.create_window(180, 300, anchor="nw", window=button1)

# 버튼 2 생성
button2 = Button(window, text="보통", command=lambda: start_game(100, 6), width=8, height=3)
button2_window = canvas.create_window(320, 300, anchor="nw", window=button2)

# 버튼 3 생성
button3 = Button(window, text="어려움", command=lambda: start_game(75, 9), width=8, height=3)
button3_window = canvas.create_window(460, 300, anchor="nw", window=button3)

# 게임 루프 시작
window.mainloop()
```

메인 윈도우 설정(2)

- 버튼 배치 후 '쉬움', '보통', '어려움 ' 을 눌렀을 때,

  → 각각 start_game(150, 6), start_game(100, 9), start_game(100, 12) 호출

- tkinter 이벤트 루프 실행, 사용자와의 상호작용 대기 및 처리

프로그램 구동

Q & A

감사합니다