0. 라이브러리와 전역 변수 정의

```
import tkinter.messagebox
import tkinter
import random

# 전역 변수
index = 0
timer = 0
score = 0
difficulty = 0
tsugi = 0
timerCount = 0 # 진행 시간 체크
noClickTimer = 0  # 클릭 없는 시간 (프레임 단위)
turnCount = 0
bonusCount = 0  # 지금까지 지급한 보너스 횟수
autoPlace = False
jokerHold = False  # 조커 한 턴 유지 여부 체크용 변수

cursor_x = 0
cursor_y = 0
mouse_x = 0
mouse_y = 0
mouse_c = 0
```

[가상현실프로그래밍] 기말 프로젝트

AR·VR미디어디자인전공 202420409 황채린


```python
# 최고 기록 불러오기 (게임 시작 시)
hisc = 100
inFile = open('hiscRecord.txt', 'r')
inStr = inFile.read()
if inStr.isdigit():
    hisc = int(inStr)
inFile.close()
```

```python
elif index == 4:  # 나란히 놓인 고양이 블록이 있다면
    # 최고 기록 저장 (게임 도중 점수가 갱신될 경우)
    if score > hisc:
        hisc = score
        outFile = open("hiscRecord.txt", "w")
        outFile.write(str(hisc))
        outFile.close()
```

1. HISC를 기록하고 불러오도록 합니다.

- 게임 시작 시 hiscRecord.txt에서 최고 점수를 불러오기
- 게임 도중 최고 점수를 경신할 경우 이를 파일에 저장
- if score > hisc 조건을 통해 점수 갱신 여부를 확인하고, 파일에 새 기록을 저장하도록 구현

```python
#공간정보 저장
neko = []
check = []
for i in range(12):
    neko.append([0, 0, 0, 0, 0, 0, 0, 0, 0, 0]) # 리스트 안쪽에 리스트(2차원 배열)
    check.append([0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0])
```

2. 게임 공간을 8x10에서 10x12 공간으로 바꿉니다.

- 2차원 배열 neko와 check를 10x12으로 정의

10칸

12칸

```python
def draw_neko():
    cvs.delete("NEKO")  # 캔버스에서 태그 'NEKO'을 삭제 / 지우고 밑에서 다시 그리는 것을 반복
    for y in range(12):  # 세로
        for x in range(10):  # 가로
            if neko[y][x] > 0:  # 모든 칸에 대해서 실행 / y에 있는 값 중에 x번째 값 / neko[y]가 변수
                cvs.create_image(x * 72 + 60, y * 72 + 60, image=img_neko[neko[y][x]], tag="NEKO") #'NEKO' 생성
```

3. 바뀐 공간에서도 동일하게 게임이 동작하도록 합니다.

for y in range(12):
for x in range(10):

- 모든 루프나 위치 조건을 10x12 크기에 맞춰 수정
- 블럭 낙하, 매칭, 제거가 전체 범위에서 정상 동작하도록 구현

```
# 블럭 정보 저장
blockCount = [0,0,0,0,0,0,0]
```

```
def sweep_neko():
    num = 0
    for y in range(12):
        for x in range(10):  # 모든 칸에 대해서 실행
            if neko[y][x] == 7:
                neko[y][x] = 0  # 빈칸
                num = num + 1   # 파괴된 블럭 개수를 표현
    return num
```

```
draw_txt("SCORE " + str(score), 160, 60, 32, "blue", "INFO")
draw_txt("HISC " + str(hisc), 450, 60, 32, "yellow", "INFO")
#파괴된 블록 개수 합산
total = sum(blockCount)
draw_txt("BlockCount  " + str(total), 890, 100, 24, "blue", "INFO")
```

4. 파괴된 블럭을 화면에 표현하고, 10개가 넘을 때 마다 보너스 10점씩 추가합니다.

- sweep_neko() 함수에서 파괴된 블럭 개수를 반환하고, 점수 계산
- blockCount 배열을 통해 각 블록별 파괴 횟수를 누적하고, sum(blockCount)로 총 파괴 개수를 실시간 계산

score = 15(sc) X 4점(난이도) X 2 +10(보너스) = 130

```
sc = sweep_neko()
score = score + sc * difficulty * 2 # 기본 점수

# 보너스 점수
if total // 10 > bonusCount:
    score += 10
    bonusCount += 1
```

4. 파괴된 블럭을 화면에 표현하고, 10개가 넘을 때 마다 보너스 10점씩 추가합니다.

- bonusCount에 보너스 지급횟수 저장
- total = sum(blockCount) / 게임 전체에서 파괴된 블록 수를 누적 계산
- total // 10 > bonusCount 조건을 이용해, 10개 단위로 블록이 부서질 때마다 10점 보너스를 한 번씩만 지급 (중복 방지)

```python
# 조커 블록이 있을 때의 나머지 블록들은 같아야 함 / 4개의 블록
def is_match_four(a, b, c, d):
    if 0 in (a, b, c, d):
        return False
    fourBlocks = [a, b, c, d]
    filtered = [x for x in fourBlocks if x != 8]
    first = filtered[0]
    return all(x == first for x in filtered)
```

```python
for y in range(0, 11):
    for x in range(0, 9):
        if check[y][x] > 0:  # 네모 블록
            if is_match_four(check[y + 1][x], check[y][x + 1], check[y][x], check[y+1][x+1]):
                if neko[y][x] != 8: blockCount[neko[y][x]-1] +=4
                neko[y][x] = 7
                neko[y][x+1] = 7
                neko[y + 1][x] = 7
                neko[y + 1][x + 1] = 7
```

5. 4개 블럭이 네모로 놓여 있을 때에도 블럭이 파괴됩니다.

- 함수 is_match_four()로 check_neko()에서 2x2 정사각형 블럭 비교

```python
# 타이머 증가 및 표시 (프레임 단위)
if index in [2,3,4,5]: # 게임 플레이 하는 시간 동안 타이머 표시
    timerCount += 1
    min = timerCount // 60
    sec = timerCount % 60
    draw_txt(f"TIME {min:02}:{sec:02}", 890, 60, 24, "white", "INFO")
root.after(100, game_main)
```

6. 게임 시작 시 경기 시간을 화면에 표시합니다.

- 게임을 진행하는 index 2-5까지만 타이머 작동
- root.after(100, game_main)을 통해 game_main() 함수가 0.1초마다 한 번씩 재귀호출
=> timerCount가 1씩 증가
- 경기 시간을 mm:ss 형식으로 화면에 표시

```python
# 자동 배치 타이머
noClickTimer+=1
if noClickTimer >= 50:
    autoPlace = True
    set_neko()
    if tsugi !=0:
        neko[random.randint(0, 11)][random.randint(0, 9)] = tsugi
        tsugi = 0
        index = 2
        noClickTimer = 0 # 클릭 안할 시 타이머 리셋
else:
    autoPlace = False
```

7. 5초 이상 블럭을 배치하지 않는 경우 자동으로 블럭이 떨어집니다.

- 클릭 없이 5초(=50프레임)가 지나면 자동으로 랜덤으로 블록이 떨어지며, 이 자동 배치는 조커 블록 등장 조건에서는 제외
- 자동 배치가 발생한 경우 autoPlace가 True가 되어, 이후 턴에서는 turnCount 증가 없이 진행되어 조커 등장 조건에 포함되지 않음


```python
#esc 눌렀을 때 게임 리셋
def key_down(e):
    global index, score, timerCount, tsugi, noClickTimer, jokerHold
    if e.keysym == 'Escape':
        answer = tkinter.messagebox.askyesno('종료 확인','게임을 종료하시겠습니까?')
        if answer:
            index=0
            score=0
            timerCount=0
            tsugi = 0


            # 화면 클리어
            cvs.delete("NEKO")
            cvs.delete("CURSOR")
            cvs.delete("INFO")
            cvs.delete("OVER")
```

8. 'esc' 버튼을 누르는 경우 '게임을 종료하시겠습니까?' 라는 경고창이 뜨며, '예' 버튼을 누르는 경우 메인 화면으로 돌아갑니다.

- answer을 받았을 시 score, timerCount, tsugi 값 초기화 후 index=0으로 돌아가기


```
#이미지와 관련된 영역
bg = tkinter.PhotoImage(file="neko_bg_EX.png")
cursor = tkinter.PhotoImage(file="neko_cursor.png")
img_neko = [
    None,
    tkinter.PhotoImage(file="neko1.png"),
    tkinter.PhotoImage(file="neko2.png"),
    tkinter.PhotoImage(file="neko3.png"),
    tkinter.PhotoImage(file="neko4.png"),
    tkinter.PhotoImage(file="neko5.png"),
    tkinter.PhotoImage(file="neko6.png"),
    tkinter.PhotoImage(file="neko_niku.png"),
    tkinter.PhotoImage(file="neko0.png") # 조커 블록 이미지
]

cvs.create_image(528, 456, image=bg)
game_main()
root.mainloop()
```

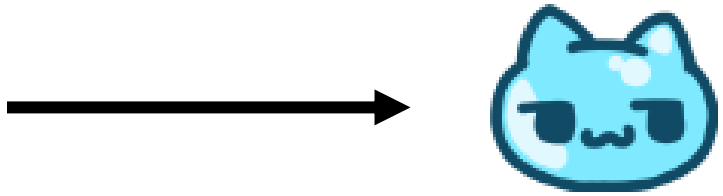
```
if sc > 0:
    index = 2
else:
    if not over_neko():
        # autoPlace가 True면 조커 등장 조건 무시
        if not autoPlace and turnCount%5 == 0 and turnCount>0: # 5턴 때마다 조커 블록 등장
            tsugi = 8 # 조커 블록 등장
        else:
            tsugi = random.randint(1, difficulty)
        index = 5
    else:
        index = 6
        timer = 0
draw_neko()
```

조커 블록

9. 블럭 배치 5턴 마다 조커 블럭을 배치할 수 있게 됩니다. 조커 블럭은 모든 블럭으로 해석 될 수 있습니다. (5번 횟수에 시간초과로 블럭이 배치되는 경우는 포함되지 않습니다.)

- 조커 이미지 추가
- 자동 낙하 시 조커 등장 조건 무시 => turnCount 증가X
- turnCount가 5의 배수일 때마다 조커 등장
- is_match(), is_match_three(), is_match_four() 함수 내부에서 조커 블럭(숫자 8)은 어떤 블럭과도 매칭되도록 설계

```python
# 한 턴 유지 → 이후 변환
if jokerHold:
    for y in range(12):
        for x in range(10):
            if neko[y][x] == 8:
                neko[y][x] = random.randint(1, difficulty)
    jokerHold = False
else:
    # 조커가 존재할 시 유지 시작
    for y in range(12):
        for x in range(10):
            if neko[y][x] == 8:
                jokerHold = True
                break
        if jokerHold:
            break
```



10. 조커 블럭은 한 턴 유지 후 일반 블럭으로 변경됩니다.

- jokerHold = True일 때 유지 / False일 때는 랜덤 블럭으로 전환
- 조커 블럭이 생성된 다음 drop_neko()에서 더 이상 낙하가 발생하지 않으면, 그 턴까지 유지되고, 이후 턴에서 jokerHold가 False로 전환되어 랜덤 블록으로 변환