빅데이터 프로그래밍1

과제 7

조 번호 : 1 조

조원명단 : 김미소

고아라

곽민지

김 원

1. main

```
from app import * #app 모듈을 임포트

def on_closing():
    global interrupt
    if messagebox.askokcancel("Quit", "Quit?"):
        root.quit()
        interrupt = 0

if __name__ == "__main__":
    global interrupt #인터럽트
    interrupt = 1
    width = 4
    root = Tk() #root 생성

    while True:
        app = App(root, width) ####app 모듈의 App을 실행한다.
        root.protocol("WM_DELETE_WINDOW", on_closing)
        root.mainloop()
        if app.continue_game == 0 or interrupt == 0:
            break
    root.destroy()
```

2. app

```
from maintable import *
from conveyor import *
from PIL import Image, ImageTk
from tkinter import messagebox
import sys
import os

class App(Frame):    #프레임을 상속 받는 클래스
    picture_image = [] # 메인 테이블용 도형 이미지
    alphabet_image = [] # 메인 테이블용 알파벳 이미지
    resized_picture = [] # 컨베이어용 도형 이미지
```

```
    continue_game = 1

    def __init__(self, master, n): #constructor : 부모가 마스터, n이라는 것을 보유
        super(App, self).__init__() #Frame()을 실행.
        self.master = master # root를 parent로
        self.n = n
        self.create_images()

        # Maintable frame widget 생성 #self.table에 메인테이블이라는 객체 생성함 (maintable.py로)
        self.table = Maintable(self, self.picture_image, self.alphabet_image, self.n)
        self.table.grid(row=0, column=0, pady=(10, 20))  # 테이블은 위 공백 10, 아래 공백 20
        # 테이블을 프레임 안에 위치시킨다.

        # Conveyor frame widget 생성
        self.conveyor = Conveyor(self, self.resized_picture, self.n)  # (conveyor.py로)
        self.conveyor.grid(row=1, column=0)  # 콘베이어는 아래편에 넣는다
        # 콘베이어를 프레임 안에 위치시킨다.

        self.table.conveyor=self.conveyor

    # 이미지 파일을 읽어와서 이미지 객체를 생성하고 리스트에 저장하는 부분
    # 생성된 이미지 객체 리스트에는 추가적인 변경이 없고, index를 통해 randomize
    def create_images(self):
        self.picture_image = list(Image.open("picture₩₩%d.JPG" % (i+1)) for i in range(self.n*self.n))
        #게임판 n*n개 만큼 1~n*n번까지 오픈해서 그림리스트로 만듦
        self.alphabet_image = list(PhotoImage(file="alphabet₩₩%d.GIF" % (i+1)) for i in
range(self.n*self.n))
        # 게임판 n*n개 만큼 1~n*n번까지 오픈해서 알파벳그림리스트로 만듦
        self.resized_picture = list(self.picture_image[i].resize((50,50), Image.ANTIALIAS) for i in
range(self.n*self.n))
        #picture image를 리사이즈해서 다시 리스트로 만듦
        self.resized_picture = list(ImageTk.PhotoImage(self.resized_picture[i]) for i in range(self.n*self.n))
        self.picture_image = list(ImageTk.PhotoImage(self.picture_image[i]) for i in range(self.n*self.n))

    # 게임 종료시 처리 부분
    def quit_game(self, win):
        if win == True:
            messagebox.showinfo("게임 종료", "성공하였습니다")
        else:
            messagebox.showinfo("게임 종료", "실패하였습니다")
```

```
        result = messagebox.askquestion("다시 시작", "다시 시작하시겠습니까?", icon='warning')
        if result == 'no':
                App.continue_game = 0

        self.conveyor.quit()
        self.table.quit()
        self.quit()
```

3. imagebutn

```
from tkinter import *

class ImageButton(Button):    #버튼의 일종
    def __init__(self, parent = None, **kw):
        Button.__init__(self, parent, kw)

    # 해당 widget에 숨겨진 이미지 추가
    def add_hidden(self, alphabet, hidden):
        self.alphabet = alphabet
        self.hidden = hidden

    # 해당 widget의 숨겨진 이미지 값 return
    def get_hidden(self):
        return self.hidden
```

4. maintable

```
from imagebtn import *
from tkinter import *
from random import *
import time

class Maintable(Frame):
    n=0
    selected_image=0

    def __init__(self, master, picture, alphabet, width):
```

```
        #app class에서 콜됨
        #self.table = Maintable(self, self.picture_image, self.alphabet_image, self.n)
        super(Maintable, self).__init__() #frame의 생성.
        self.image_number_list = []  # 셔플된 이미지의 번호를 저장하기 위한 리스트. 16개
        self.master = master # maintable frame의 parent 설정
        self.width = width # maintable의 넓이. = 4
        self.n = width * width # maintable에 추가될 이미지 수. = 16
        self.picture = picture # # app에서 생성한 이미지 받아와서 저장

        self.conveyor=None
        # 숨겨진 이미지 셔플링
        self.random_shuffle() #이미지를 섞음. 아래에서 정의 해야 한다 TODO

        # TODO
        # ImageButton widget 생성하고 각 widget에 숨겨진 이미지 추가
        buttonlst=[]
        for i in range(self.width*self.width):
            abutton=ImageButton(self,image=alphabet[i])
            abutton.add_hidden(alphabet[i],self.picture[self.image_number_list[i]])
            buttonlst.append(abutton)
            abutton.grid(row=i//4,column=i%4)

        # 이미지 클릭시 이벤트 bind
        for i in range(0,self.width):
            for j in range(0,self.width):pass

        for i in range(self.width*self.width):
            buttonlst[i].bind('<Button-1>', self.show_hidden)
            buttonlst[i].bind('<ButtonRelease-1>', self.hide_picture)

    # TODO
    # hidden 이미지 셔플링
    def random_shuffle(self):
        self.image_number_list=list(range(self.width*self.width))
        shuffle(self.image_number_list)


    # 선택된 알파벳 ImageButton의 숨겨진 이미지 출력
    def show_hidden(self, event):
        event.widget.config(image=event.widget.get_hidden())
```

```
    # TODO
    # 숨겨진 이미지 숨기고 알파벳 이미지로 변환
    # 선택된 이미지와 컨베이어의 현재 이미지와 비교하고, 비교 결과에 따른 명령어 실행 부분
    def hide_picture(self, event):
        #time.sleep(10)
        #time.sleep(1.5)
        self.selected_image = self.picture.index(event.widget.hidden) #picture에서 어떤 번호인가
        event.widget.config(image=event.widget.alphabet)
        if self.selected_image!=self.conveyor.image_number_list[self.conveyor.cur_idx]:
            self.conveyor.wrong_match()
        else:
            self.conveyor.correct_match()
```

5. conveyor

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

class Conveyor(Frame):
    def __init__(self, master, picture, width):
        #self.conveyor = Conveyor(self, self.resized_picture, self.n) #(conveyor.py로)
        super(Conveyor, self).__init__()
        self.image_number_list = [] # 셔플된 이미지의 번호를 저장하기 위한 리스트. 13개
        self.labels = [] # 컨베이어 frame에 추가되는 이미지 label 위젯의 리스트
        self.master = master # 컨베이어 frame의 parent 설정
        self.width = width # 메인 테이블의 가로 길이. = 4
        self.n = width*(width-1)+1 # 컨베이어에 넣을 이미지의 수. = 13
        self.picture = picture # app에서 생성한 이미지 받아와서 저장
        self.image_flags = list(False for i in range(self.width*self.width))
        #현재 가리키는 그림의 번호 =  image_number_list[self.cur_idx]
        #현재 가리키는 그림객체 = self.picture[image_number_list[self.cur_idx]]
        self.conveyor_canvas = Canvas(self, height=30,width=50)

        # 컨베이어에 올릴 이미지 셔플링
        self.random_shuffle()

        # TODO
```

```
        # 셔플 결과대로 이미지 label 생성하여 리스트에 저장
        self.finallabel=Label(self,text='final',fg='red')
        self.finallabel.grid(row=0,column=12)

        for i in range(self.n):
            self.labels.append(Label(self,image=self.picture[self.image_number_list[i]]))
            self.labels[i].grid(row=1,column=i)

        # 현재 index 설정 = 시작 위치 설정
        self.cur_idx = int(self.n/self.width*(self.width-1)) #13/4*3=9

        # 현재 이미지 설정 = 시작 이미지 설정

        # 선택한 이미지와 비교 목적으로 저장
        self.cur_image = self.picture.index(self.picture[self.image_number_list[self.cur_idx]])
        #현재 화살표 그림의 원래 번호



        # TODO
        # 캔버스 세팅
        self.conveyor_canvas.grid(row=0, column=self.cur_idx)
        self.pointer=self.conveyor_canvas.create_polygon(10, 2, 40, 2, 25, 30, fill='yellow',
outline='black', width=2)

    # TODO
    # 이미지 셔플 함수
    def random_shuffle(self):
        self.image_number_list=sample(list(range(16)),13) #[0,1,2,....15] 중 임의의 13개의 숫자를 선택
        for i in self.image_number_list:
            self.image_flags[i]=True

    # TODO
    # 선택한 그림이 현재 위치의 그림과 일치하는 경우
    def correct_match(self):
        # 마지막 이미지를 찾은 경우
        if self.cur_idx == self.n-1:
            self.master.quit_game(True)
        #최종 그림 상태에서 정답을 맞췄다

        # 캔버스 위젯
```

```
            # 현재 위치 표시 도형 우측 이동

        else:
            # 현재 위치가 컨베이어의 가장 우측 도형을 지목할 때
            # FINAL 글씨를 가리지 않도록 도형 수정
            if self.cur_idx == self.n-2:
                self.conveyor_canvas.delete(self.pointer)
                self.finallabel.destroy()
                self.finallabel = Label(self, text='final', fg='red',bg='yellow')
                self.finallabel.grid(row=0, column=12)
                self.cur_idx+=1
            #끝 두번째 상태에서 정답을 맞춰서 final로 화살표가 가야 한다

            # 그 외 도형 이동
            else:
                self.conveyor_canvas.grid(row=0, column=self.cur_idx + 1)
                self.cur_idx += 1
            # 현재 이미지 및 현재 위치 수정

    # TODO
    # 선택한 그림이 현재 위치의 그림과 일치하지 않는 경우
    def wrong_match(self):
        # 마지막 기회에서 틀린 경우
        if(self.cur_idx == 0):
            self.master.quit_game(False)
            #마지막에서 틀려서 게임 오버 해야함

        # 캔버스 위젯
        # 가장 왼쪽의 이미지를 제거
        # 기존 이미지들 좌측으로 한 칸씩 이동
        # 컨베이어에 추가되지 않은 이미지 중 하나 선택하여 가장 우측에 추가

        else:
            # FINAL에서 오답 선택했을 때 도형 복구
            if self.cur_idx == self.n-1:
                self.cur_idx-=1
                self.pointer = self.conveyor_canvas.create_polygon(10, 2, 40, 2, 25, 30, fill='yellow',
outline='black',
                                                                   width=2)
                self.finallabel.destroy()
```

```
        self.finallabel = Label(self, text='final', fg='red')
        self.finallabel.grid(row=0, column=12)

# 그 외 도형 이동
else:#그냥 틀려서 왼쪽으로 이동 해야함.
        self.conveyor_canvas.grid(row=0,column=self.cur_idx-1)
        self.cur_idx-=1

# 새 이미지 추가
while True:
        new_image = randint(0, self.width*self.width-1)
        if new_image not in self.image_number_list :
                break #랜덤 사진 뽑기해서 기존 사진에 없으면 새 이미지로 채택

# 기존 이미지 좌측으로 한 칸씩 이동
# label.config(parameter = configuration) 기존의 label 위젯 변경 가능
for i in range(0,self.n-1):
        self.labels[i].config(image=self.picture[self.image_number_list [i+1]])
        #모든 레이블에 대해 그 뒤 레이블로 바꿔치기 한다.
        self.image_number_list [i] = self.image_number_list [i+1]
        #그림 숫자 리스트도 바꿔치기 한다

# 새 이미지 추가
self.image_number_list[self.n-1] = new_image #새이미지 리스트 끝에 새 번호를 넣는다
self.labels[self.n-1].config(image=self.picture[self.image_number_list [self.n-1]])
#레이블 리스트의 마지막을 새 이미지로 바꾼다.
```