REPORT 

팀 프로젝트 : Dynamic틀린그림 찾기

오픈소스SW개론(002) 3조

­­­­­­목차

[개요 3](#_Toc73351659)

[프로젝트 정보 3](#_Toc73351660)

[프로젝트 요약 3](#_Toc73351661)

[역할 분담 및 기여도 4](#_Toc73351662)

[개발 일정 4](#_Toc73351663)

[구현 내용 및 방법(백엔드) 5](#_Toc73351664)

[구현 원리 5](#_Toc73351665)

[객체 탐지 5](#_Toc73351666)

[객체 삭제 6](#_Toc73351667)

[시행 착오 및 최종 결정 방법 6](#_Toc73351668)

[코드 모듈화(백엔드) 10](#_Toc73351669)

[모듈화 10](#_Toc73351670)

[객체 탐지 10](#_Toc73351671)

[객체 삭제 11](#_Toc73351672)

[퀴즈 전달 11](#_Toc73351673)

[구현 내용 및 방법(프론트엔드) 12](#_Toc73351674)

[소개 12](#_Toc73351675)

[시작 화면 12](#_Toc73351676)

[게임 오버, 클리어 화면 13](#_Toc73351677)

[게임 화면 13](#_Toc73351678)

[코드 모듈화(프론트엔드) 17](#_Toc73351679)

[모듈화 17](#_Toc73351680)

[화면 17](#_Toc73351681)

[윈도우 프레임 18](#_Toc73351682)

[버튼 이벤트 18](#_Toc73351683)

[결과 분석 19](#_Toc73351684)

[백엔드 19](#_Toc73351685)

[프론트엔드 19](#_Toc73351686)

[결론 21](#_Toc73351687)

개요

프로젝트 정보

3조 조원은 소프트웨어학과 17011648 배정준, 17011695­ 김동역으로 Dynamic 틀린그림찾기를 구현하였습니다.

프로젝트github주소<https://github.com/GalaxyOverMe/Dynamic_FindTheDifference>

프로젝트 요약

Dynamic 틀린그림찾기와 일반적인 틀린그림찾기의 차이점은 일반적인 틀린그림찾기는 서로 다른 그림 2장을 준비해야하지 만, Dynamic 틀린그림찾기는 그림을 1장만 준비해도 문제를 만들어 줍니다.



 [원본 그림]

[원본 그림] [틀린 그림] [정답 그림]

프로젝트 수행 방법

프로젝트 수행을 위해 python의 OpenCV와 wxPython을 이용했습니다. OpenCV로 틀린그림을 만들고, wxPython으로 GUI를 구현하였습니다.

역할 분담 및 기여도

프로젝트 구현을 위해 백엔드(OpenCV를 이용한 틀린그림 생성), 프론트엔드(GUI구현) 으로 역할을 분담하였습니다. 배정준은 백엔드를 담당하고 김동역은 프론트엔드를 담당했습니다.

배정준 : 백엔드, 이미지 샘플 수집

김동역 : 프론트엔드

개발 일정

9 주차(4월 26일 ~ 5월 2일) : 기획

객체 탐지(Object Detection) KNN제안

객체 삭제(Object Deletion) erosion 제안

10주차(5월 3일 ~ 5월 9일) : 구현 방법 설계

객체 탐지: contour 방식 제안

Canny Detection 개선(Gaussian Filter -> Bilateral Filter)

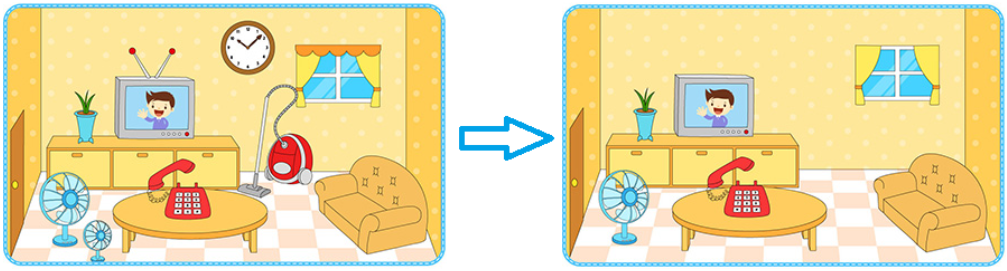
객체 삭제: Image Inpainting 방식 제안

사각형 -> 다각형

11주차~13주차 (5월 10일~5월 30일) : 구현 방법 설계 및 개발

객체 탐지: Morphological Edge 최종 결정

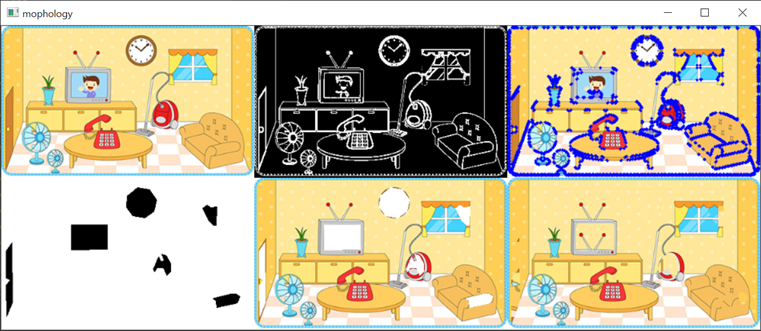
구현 내용 및 방법(백엔드)

구현 원리

[원본 그림] [틀린 그림]

Dynamic 틀린그림찾기는 원본 그림에서 **객체를 탐지**하여 **객체를 삭제**하는 방식으로 틀린 그림을 생성합니다.

객체 탐지



틀린 그림을 얻기 위해서 contour[[1]](#footnote-1)를 폐곡선으로 둘러싼 다각형을 객체로 하고,

이 객체를 퀴즈 후보로 인정합니다.

그러나 모든 퀴즈 후보를 퀴즈로 인정할 수 없기 때문에 (1)육안 식별이 가능할 정도의 크기, (2)너무 크지 않은 크기, (3)객체간 최소 거리를 만족하는 객체를 퀴즈로 인정합니다.

객체 삭제

객체 탐지를 통해 얻은 퀴즈를 원본 그림에서 삭제합니다. 이 때 삭제한 부분이 주변의 이미지와 조화를 이루어야 합니다. (이질감이 느껴지면 안 됩니다.) Image Inpainting 기법은 단순히 주변의 점이 아닌 주변의 영역을 가져오기 때문에 자연스러운 틀린 그림을 만들 수 있습니다.

시행 착오 및 최종 결정 방법

객체 탐지 방법이 2번, 객체 삭제 방법이 1번 바뀌었습니다.

먼저 객체 탐지 방법 결정 간 시행 착오에 대해 소개하겠습니다.



방법1에서 방법2으로의 개선은 KNN알고리즘이 아닌 contour를 통해 객체 탐지를 하는 것입니다. KNN 알고리즘은 계산량도 많고, 결과도 좋지 않았습니다.

방법2에서 방법3으로의 개선은 edge를 얻기 위해 사용하는 방법을 Canny Edge가 아닌 Morphological Edge를 사용하는 것입니다.

|  |  |  |
| --- | --- | --- |
| 원본 그림 |  |  |
| Canny-Edge 적용 이미지 |  |  |
| Morphological-Edge  적용 이미지 |  |  |

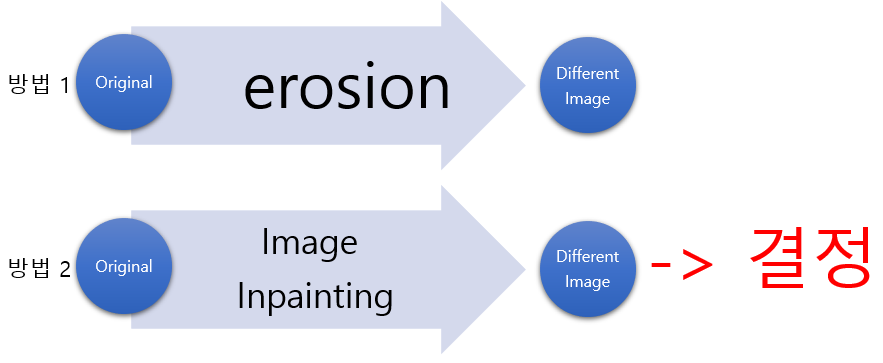
[Canny Edge와 Morphological Edge의 비교][[2]](#endnote-1)[1]

틀린그림찾기에서 퀴즈후보로는 객체의 내부 보다는 객체 전체가 적합합니다.

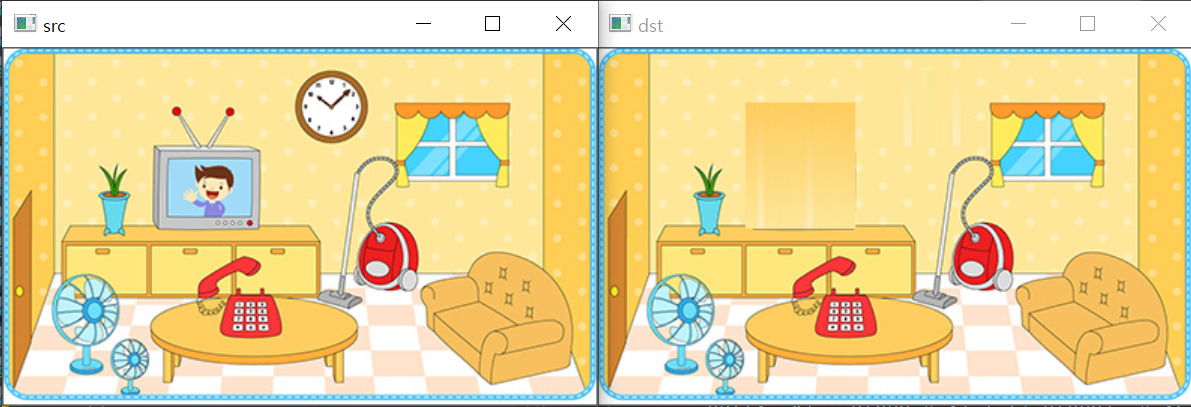
추가로 Canny Edge방법은 Gaussian Filter 이미지에서 얻어지기 때문에 Gaussian Filter이미지가 필요없는 Morphological Edge방법은 계산량에서 유리합니다.

|  |
| --- |
|  |
| Canny 방법 |
|  |
| Morphological Edge방법 |

Canny Edge 방식 보다 Morphological Edge 방법이 객체 탐지에 더 적합함을 볼 수 있습니다.



객체 삭제를 주변색을 이용하는 침식을 통해 해결할 수 있을 것이라고 생각했습니다. 그러나 그 결과가 너무 부자연스러웠습니다.



[원본 그림] [침식을 적용한 틀린 그림]

결과적으로 객체삭제로 Image Inpainting을 이용하였습니다.

[원본 그림] [틀린 그림] [정답 그림]

Image Inpainting의 결과가 Erosion 방식보다 자연스러운 것을 알 수 있습니다.

코드 모듈화(백엔드)

모듈화

src, dst, ref 는 원본그림, 틀린그림, 정답 그림 입니다.

## 객체 탐지

def normalize\_size(src, dst, ref, width = 400, height = 600)

in : 원본 이미지 (src, ref, dst)

out : 크기 정규화된 이미지 (src, ref, dst)

def get\_morphological\_edge(src)

in : src 이미지

out : edge 이미지

def get\_contours(edge, imgContour)

in : edge 이미지, Contour를 저장할이미지

out : 퀴즈 후보

def quiz(ptsCandidate, approxCandidate, num, imgMask, ref)

in : 퀴즈 후보(pts와 approx), 최대 퀴즈 개수, 마스크 이미지, 정답그림

out : 퀴즈 정답 좌표(직사각형 좌표 리턴 [x1, y1, x2, y2])

## 객체 삭제

cv2\_contib.inpaint(src, mask, dst, algorithmType)

in : 원본 그림, 마스크 이미지, 틀린 그림, Inpainting 방식

out : None

## 퀴즈 전달

def centering\_image(src, dst, ref, width = 400, height = 600)

in : normarlize\_size 된 이미지 (src, ref, dst)

out : centering 된 이미지(src, ref, dst), 이동한 좌표(x,y)

def get\_pts\_center(pts)

in : ptr[[x1,y1,x2,y2],[x1,y1,x2,y2],...,]

out : ptr\_c[[cx,cy],[cx,xy],...]

def load\_images()

in : None

out : 이미지 경로 배열

def get\_next\_quiz(NextImagePath)

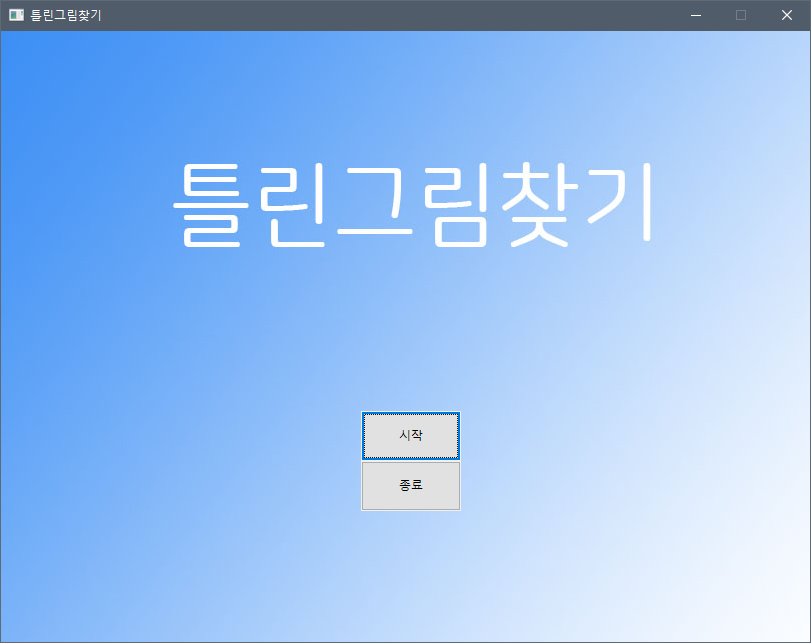
in : 다음 이미지 주소

out : (원본, 틀린그림, 정답그림)순서의 병합된 이미지, 정답 좌표, 이동한 좌표

구현 내용 및 방법(프론트엔드)

소개

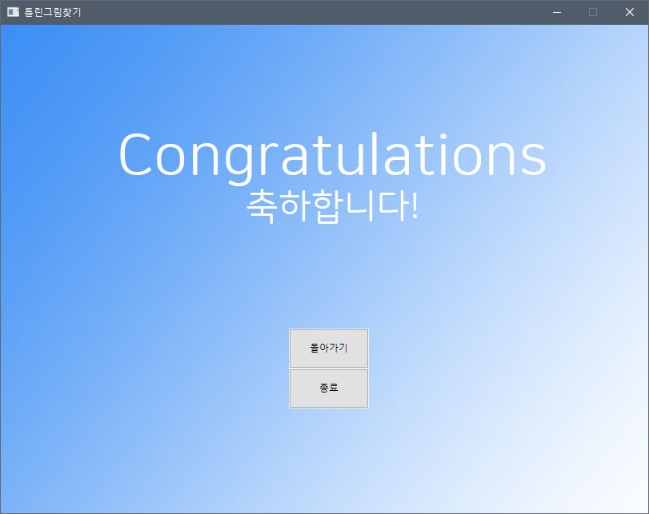
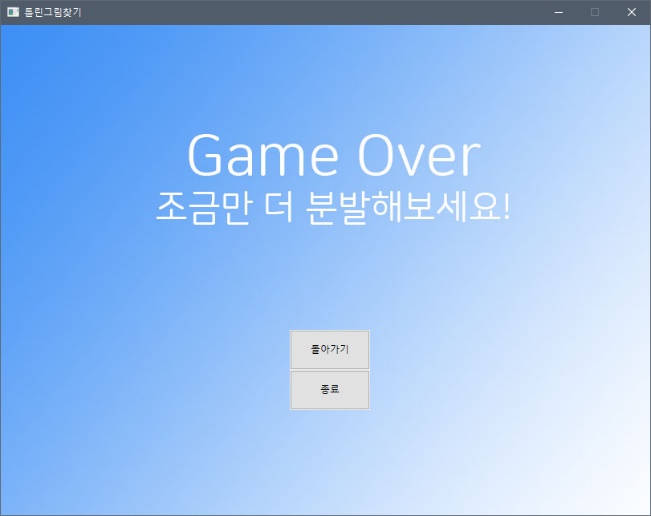
백엔드에서 이미지를 가져와 게임을 진행하는 프론트엔드 환경입니다. 크게 시작 화면, 게임 화면, 게임 오버 화면, 게임 클리어 화면으로 이루어져 있습니다. 프론트엔드의 구현은 파이썬 GUI 라이브러리인 wxPython을 활용하였습니다.

시작 화면

처음 실행할 때, 출력되는 화면입니다. 이후 클리어 또는 게임 오버 시에도 돌아가기 버튼을 클릭하면 출력됩니다. 시작, 종료 버튼을 배치했고, 타이틀 배경은 별도의 이미지 파일을 만들어 이미지 형태로 화면에 출력했습니다. 이 구성은 게임 오버 화면과 클리어 화면도 같습니다.

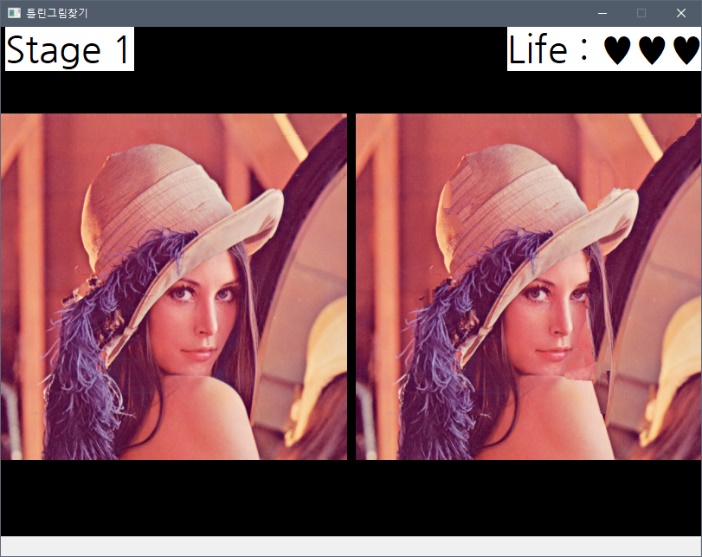
## 게임 오버, 클리어 화면

게임은 한 스테이지에서 3번 틀릴 경우 게임 오버입니다. 모든 스테이지를 통과하는 경우 게임을 클리어할 수 있습니다. 아래는 각각 게임 오버, 게임 클리어 시 출력되는 화면입니다.

시작 화면과 거의 유사하게 구성되어 있습니다. 차이점은 시작 버튼 자리에 시작 화면으로 돌아가는 돌아가기 버튼이 위치해 있다는 것입니다.

게임 화면

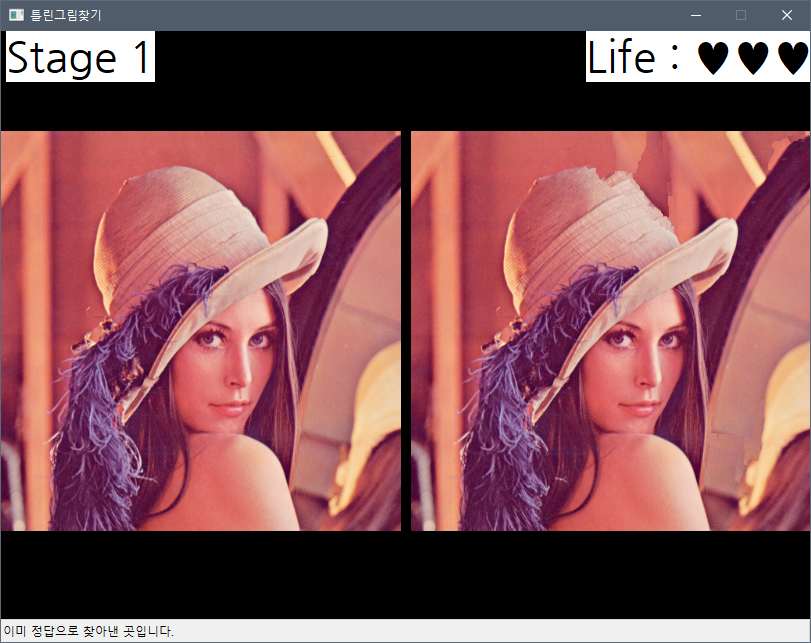
시작 화면에서 시작 버튼을 누르면 틀린그림찾기가 시작됩니다. 게임의 목표는 틀린 부분 3곳(이미지에 따라 3개 미만인 경우 그 수 만큼, 3개 초과인 경우 3개로 고정)을 찾는 것입니다. 게임을 시작하면, 스테이지, 목숨, 그리고 왼쪽에 원래 그림과 오른쪽에 틀린 그림이 출력됩니다. 다음 화면이 게임 화면 예시입니다.



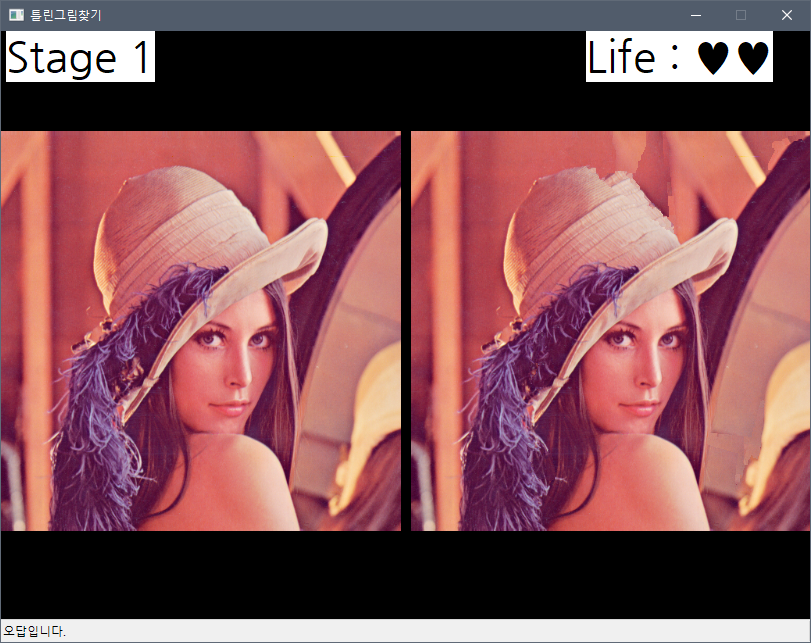
플레이어는 틀린 부분을 찾아 클릭하여 정답을 확인합니다. 틀린 곳을 찾을 때, 왼쪽의 원래 그림을 클릭해도 되고, 틀린 그림의 틀린 부분을 클릭해도 됩니다. 마우스로 클릭한 좌표와 틀린 부분의 거리가 30 이내인 경우 정답으로 처리됩니다. 정답인 경우 아래 화면과 같이, 정답과 찾아야하는 남은 정답 수가 화면 하단 상태 바에 표시됩니다.



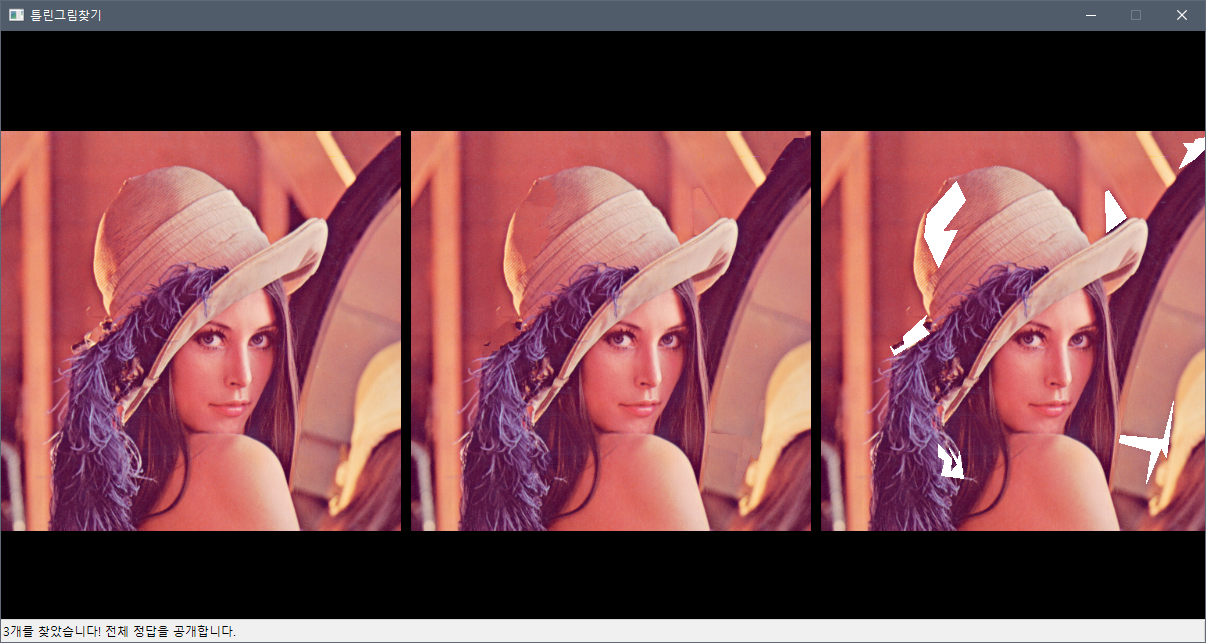
이 상태에서 이미 찾아낸 곳을 클릭해도 정답으로 처리되지 않습니다. 이 경우 하단 상태 바에 다음과 같이 표시됩니다.



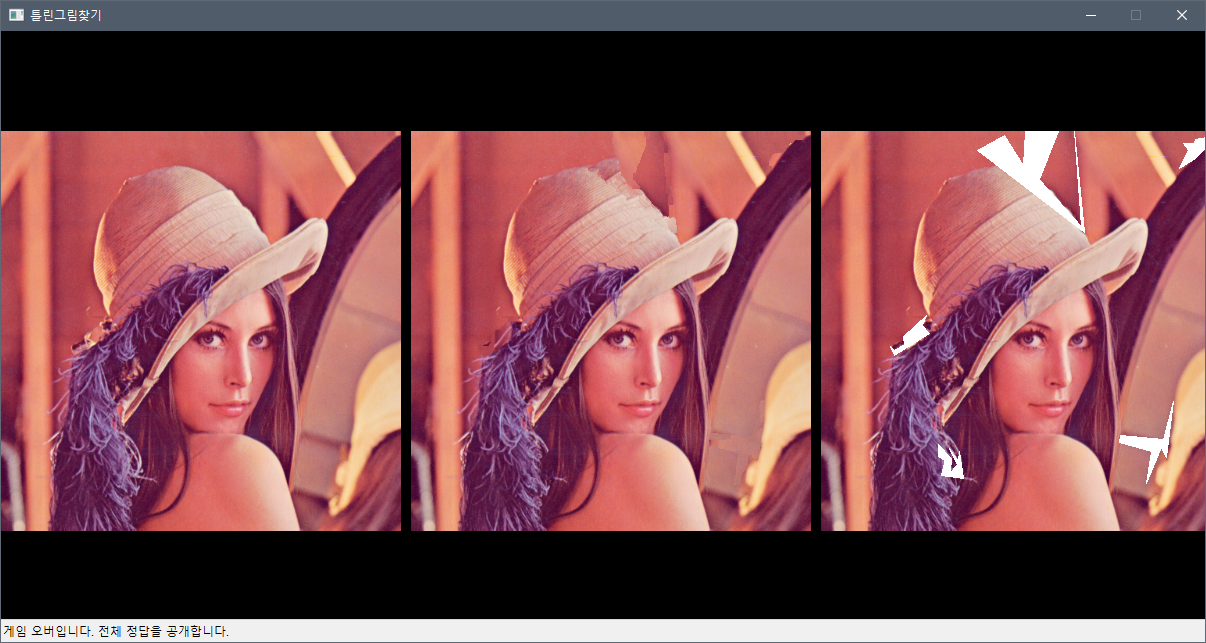
정답이 아닌 곳을 클릭했을 경우 오답 처리됩니다. 오답인 경우 라이프가 하나 감소하고(하트 사라짐), 하단 상태 바에 오답이라고 표시됩니다.



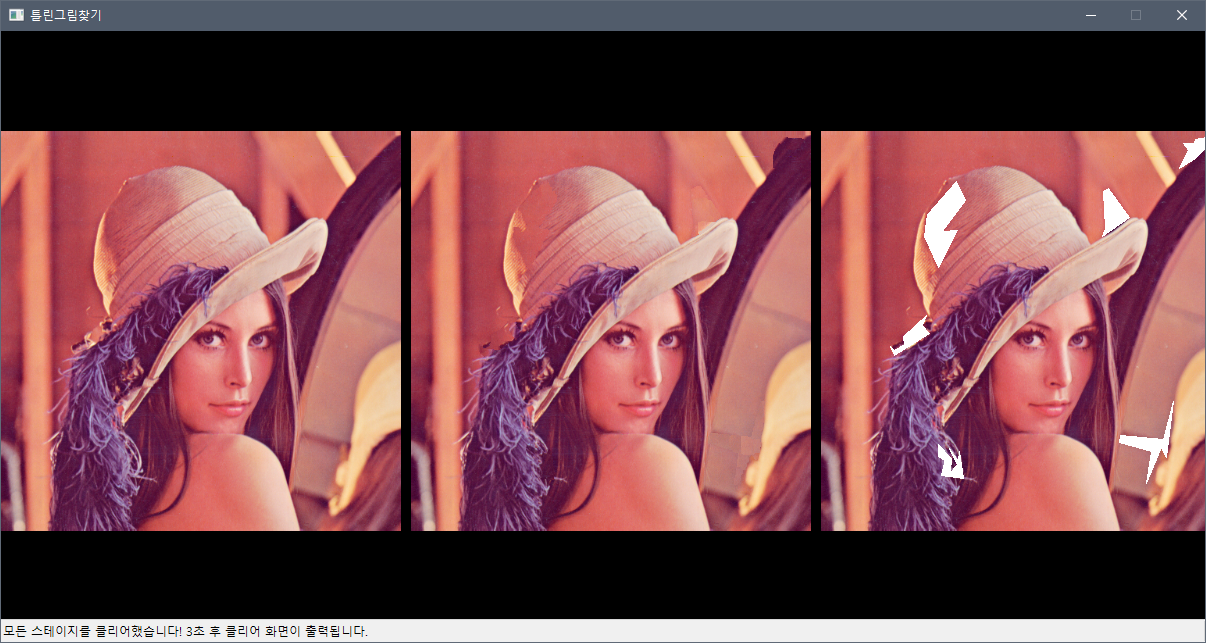
정답 세 곳을 전부 찾는 경우, 현재 스테이지를 통과할 수 있으며 찾지 못했던 전체 정답이 공개됩니다. 전체 정답을 3초 보여준 후, 다음 스테이지로 넘어갑니다.



라이프가 0이 되는 경우 게임 오버입니다. 이 때에도 마찬가지로 전체 정답을 보여준 후, 게임 오버 화면을 출력합니다. 다음은 게임 오버 시 화면 예시입니다.



모든 스테이지를 클리어했을 경우, 스테이지 클리어 화면에서 하단 상태 바의 메시지가 아래 화면과 같이 변경됩니다. 이후 게임 클리어 화면을 출력합니다.



코드 모듈화(프론트엔드)

모듈화

화면

gamePanel.gameProgress

게임 진행 처리. 그림을 마우스로 클릭할 때 실행됨. 클릭한 좌표를 이용하여 정/오답 처리 및 게임 오버, 클리어 시 처리도 포함

gamePanel.goToNextStage

스테이지를 클리어한 경우 다음 이미지와 정답 좌표를 새롭게 설정. 스테이지 값 증가 및 라이프 3으로 초기화.

윈도우 프레임

wx.Sizer.Add(Panel, …) (wxPython 라이브러리 메소드)

화면 배치를 관리하는 Sizer의 각각 화면 추가. Panel에 게임을 위해 생성한 Panel 입력

Bind(event, handler) (wxPython 라이브러리 메소드)

버튼 및 마우스 클릭에 이벤트 설정. event자리에 이벤트 유형 결정, handler에 설정할 이벤트 입력. Event가 EVT\_BUTTON 인 경우 버튼 클릭, EVT\_LEFT\_DOWN 인 경우 왼쪽 마우스 클릭.

wx.Statusbar (wxPython 라이브러리)

여러가지 정보를 표시할 화면 하단 상태 바를 생성

mainFrame.showGameover, mainFrame.showClear

각각 게임 화면에서 게임 오버, 클리어 화면으로 전환

버튼 이벤트

gamePanel.resetAndStartGame

시작 화면에서 게임을 시작할 때 실행. 이미지, 스테이지, 라이프 등 모두 초기화 후 게임 시작을 위한 준비 후 게임 화면 출력

wx.MessageBox (wxPython 라이브러리)

게임을 종료하기 전 정말 종료할 것인지 의사를 묻는 메시지 창 출력. ‘예’ 클릭시 게임 종료, ‘아니오’ 클릭 시 아무 일도 일어나지 않음.

answer = MessageBox('게임을 종료할까요?', '종료',  
 YES\_NO | NO\_DEFAULT, self)

‘게임을 종료할까요?’ : 질문

‘종료’ : 메시지 창 제목

YES\_NO : 네 혹은 아니오를 묻는 유형의 메시지 창 설정

NO\_DEFAULT : 기본값은 ‘아니오’

예, 아니오 클릭에 따라 answer에 값 저장 후 answer가 yes인 경우 종료.

# 결과 분석

## 백엔드

1. 사이즈 정규화를 통해 크기가 큰 이미지에 대해서도 동일한 수행시간이 소요된다. Morphologic Edge의 Threshold와 contour를 설정하는 함수의 매개변수의 최적값을 찾는다면 더 나은 결과를 얻을 수 있을 것이다.

2. 틀린그림찾기 이용자는 사물의 왜곡[그림 1] ( 축구공이 있던 자리에 농구공으로 교체) 보다 인물의 왜곡[그림 2] ( 얼굴이 있던 자리를 농구공으로 교체)에서 더 큰 괴리감을 느끼고 불만족을 느끼게 된다.

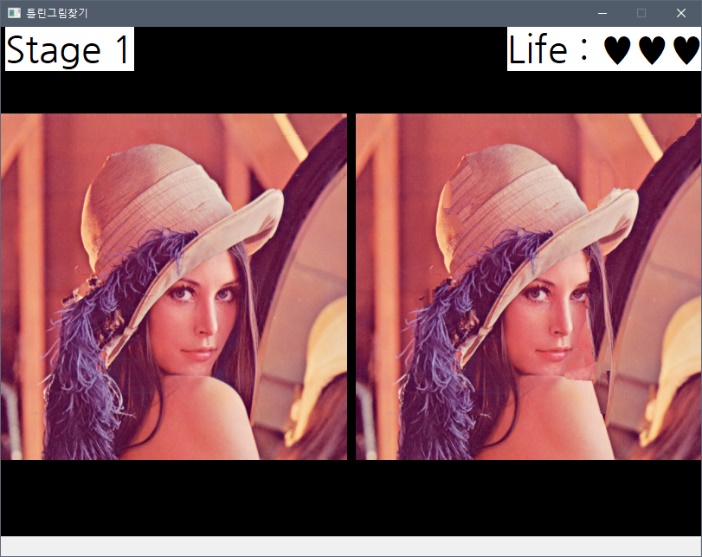
  

[원본그림] [그림 1] [그림 2]

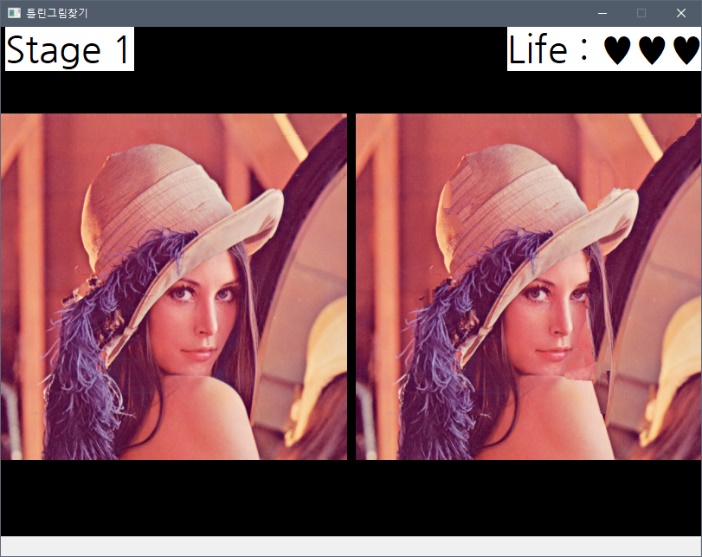
이러한 문제점 때문에 인물이 나오는 그림에 적용하기 어려운 점이 있다.

## 프론트엔드

1. 정답인 부분에 원으로 정답임을 표시하고 싶었습니다. 그러나 wxPython 라이브러리의 이미지 출력 처리의 한계로, 원 이외에 배경이 투명한 이미지를 이용하여도, 비트맵으로 전환해야 출력이 가능해져 배경이 특정 색으로 자동으로 색칠되어 그림을 가리게 되었습니다. 구현이 어렵다고 판단하여 정답과 오답 등 현재 상태를 확인할 수 있게 하는 하단 상태 바로 정/오답 확인을 대체하였습니다.



위 화면처럼 출력되기를 희망했으나, 원이 아래 화면처럼 출력되었습니다.



2. 라이프 부분을 동그라미와 마찬가지로 이미지로 처리하고 싶었습니다. (아래의 화면 예시처럼) 다만 그 경우 하트 모양 텍스트(♥)를 사용하는 것보다 처리가 번거로웠습니다. (위에서 발생하는 배경이 투명한 이미지가 출력이 안되는 현상이 여기에서도 발생하였습니다.) 그래서 현재 완성된 화면처럼 레이블 텍스트(wx.StaticText) 로 처리하여, 라이프 수 만큼 텍스트를 변경하는 방식으로 변경하였습니다.



# 결론

임의의 이미지로 틀린그림찾기를 생성하는 것을 제안하였습니다. 백엔드의 매개변수 최적화와 GUI 개선이 이루어진 후, 웹, 어플리케이션으로 확장한다면 긍정적인 반응을 얻을 수 있을 것입니다.

1. 같은 색 강도를 연결한 선, canny image를 입력으로 받는다. [↑](#footnote-ref-1)
2. [1] P. S. Venugopala, “An Approach to Improvise Canny Edge Detection using Morphological Filters”, IJCA vol. 116, pp. 39, Apr 2015. [↑](#endnote-ref-1)