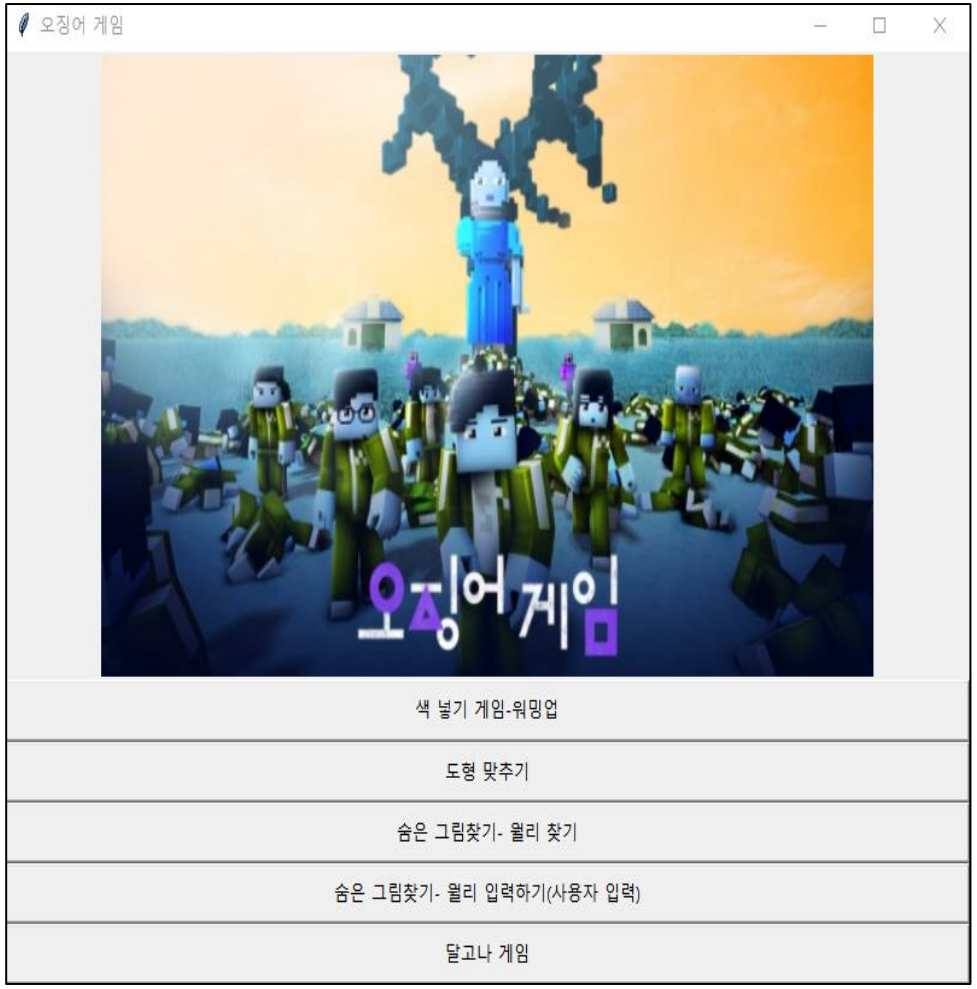
Playdata 미니프로젝트 - 홍훈표, 문주현

오징어 게임 프로젝트 목차



1. 색 넣기 게임 - 워밍업
2. 도형 맞추기
3. 숨은 그림찾기 - 윌리를 찾아라
4. 숨은 그림찾기 - 윌리 지정하기(사용자 입력)
5. 달고나 게임

〈오징어 게임 프로젝트 역할분담 - 2인〉

색 넣기 게임, 도형 맞추기

〈색 넣기 게임 - 워밍업〉

트랙바, 마우스 콜백함수, 스레시홀드

색 넣기(floodfill)

〈도형 맞추기〉

도형 그리기(polyline, circle, rectangle)

도형 중심좌표 매칭

문주현 담당

숨은 그림 찾기, 달고나 게임

〈숨은 그림찾기 - 윌리를 찾아라〉

이미지 합성하기

비트와이즈 연산자 및 mask 활용

〈달고나 게임〉

컨투어 경계 검출(findContours, drawContours)

이미지 매칭

홍훈표 담당

최대한 복습 위주의 프로젝트를 진행하였으며, 1인당 2개의 게임을 담당하여 구현함

색 입히기 게임 - 워밍업

<구현 이미지_img>

1. 트랙바로 색을 지정하고 마우스로 원하는 위치를 색칠함

2. 트랙바로 움직이면, 현재 해당하는 (B,G,R) 색상을 확인할 수 있음

스레시홀드 - cv2.threshold()

트랙바 - cv2.createTrackbar()

색 채우기 - cv2.floodfil()

트랙바로 원하는 색을 고른 후, 원하는 위치에 마우스로 클릭하면 해당 부분의 색이 변경됨

색 입히기 게임 - 워밍업

<스레시홀드 처리 전>

<스레시홀드 처리 후>

```python
# 색칠 이미지
img = cv2.imread('./coloring1.png', cv2.COLOR_BGR2GRAY)
# 외곽선을 뚜렷하게 하기 위해
ret, imthres = cv2.threshold(img, 30, 255, cv2.THRESH_BINARY)
```

(문제발생)

Img의 외곽선이 명확하지 않아 색칠 시, 흰 부분의 노이즈 발생

(해결방법)

외곽선을 조금 더 명확하게 하기 위해서 이미지를 Grayscale로

Imread한 후에 Threshold처리를 진행함

색 입히기 게임 - 워밍업

<구현 이미지_img>

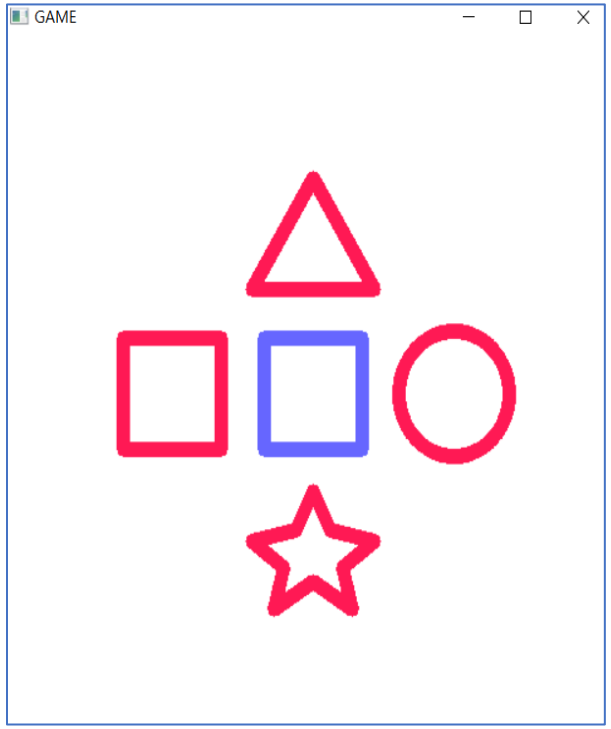
```python
def onChange(x):
    trackbar[:] = [newVal[0],newVal[1],newVal[2]]
    cv2.imshow(win_trackbar,trackbar)
```

```python
def onMouse(event, x, y, flags, param):
    global mask, imthres, newVal
    if event == cv2.EVENT_LBUTTONDOWN:
        seed = (x,y)
        retval = cv2.floodFill(imthres, mask, seed, newVal, loDiff, upDiff)
        cv2.imshow(win_name,imthres)
```

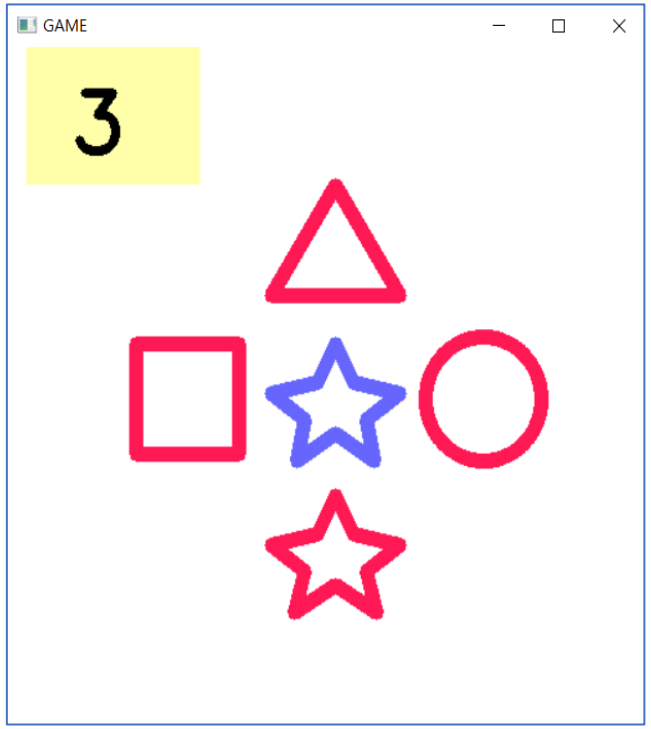
1. Color pick 창에서 트랙바를 활용하여 현재 색을 확인할 수 있음

2. 왼쪽 마우스(콜백함수처리)를 활용하여 해당 위치에 색칠

도형 맞추기



〈시작_img〉

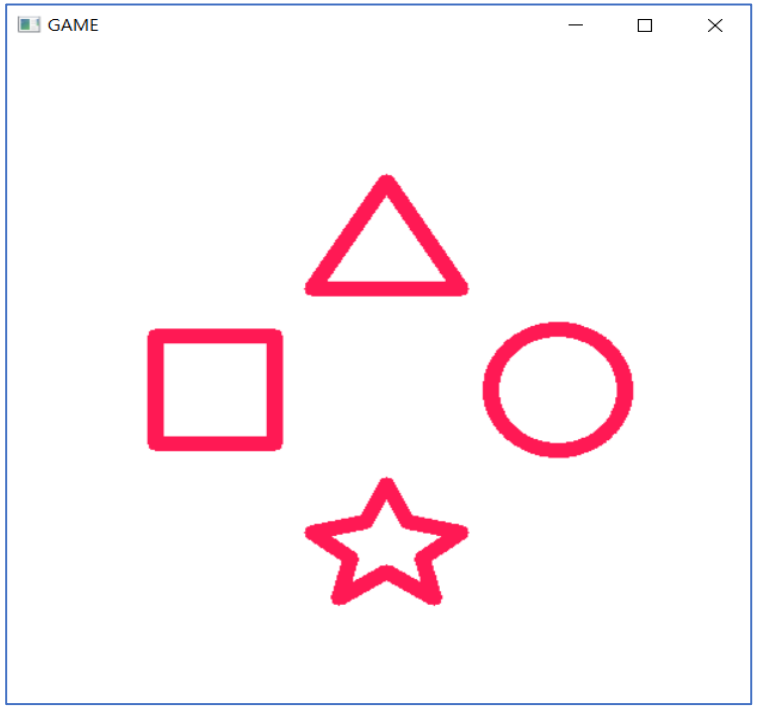


〈구현 후_img〉

〈게임 룰 설명〉

1. 파란색 도형(삼각형, 사각형, 원, 별모양)이 랜덤으로 출현함
2. 해당 도형을 빨간색 선으로 이루어진 도형에 맞게 대칭하면 성공!(카운트 발생)
3. 파란색 도형 대칭은 각 도형의 중심점에 가까워야 인정됨

도형 맞추기



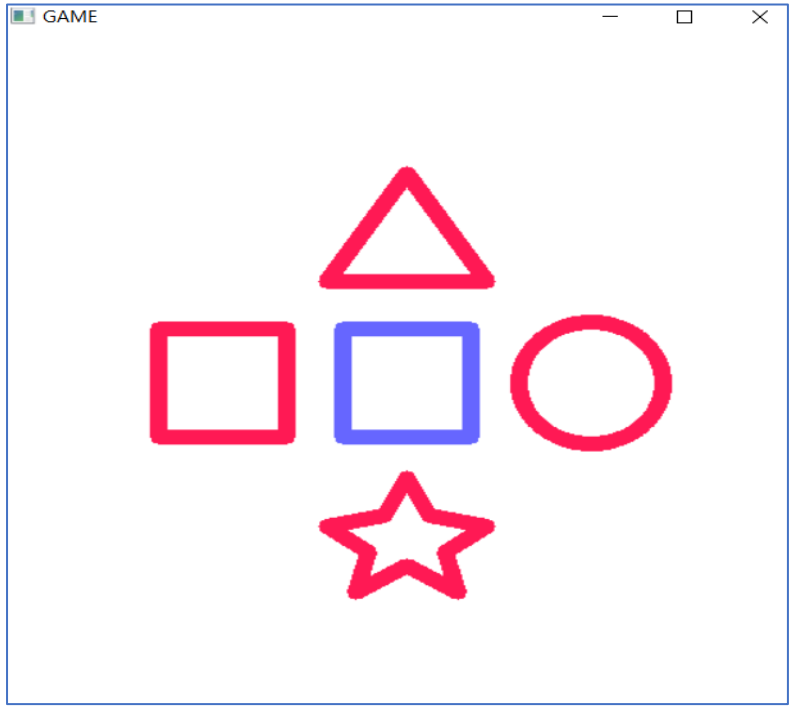
<기본 도형_img>

```
# 기본 배경 삼각형, 별 좌표
pts1 = np.array([[255,100],[205,180],[305,180]],dtype=np.int32)
pts2 = np.array([[255,325],[241,353],[205,361],[231,380],
                 [223,410],[255,390],[287,410],[279,380],
                 [305,361],[269,353]],dtype=np.int32)
```

```
# 게임 기본 화면
cv2.rectangle(img, (100,215),(180,295),(84,25,255),10) # 사각형
cv2.circle(img, (370,255),45,(84,25,255),10) #원
cv2.polylines(img, [pts1], True, (84,25,255),10) # 삼각형
cv2.polylines(img, [pts2], True, (84,25,255),10) # 별모양
```

기본 도형 그리기(500X500) 배경의 원하는 도형을 그림

도형 맞추기



<기본 도형_img>

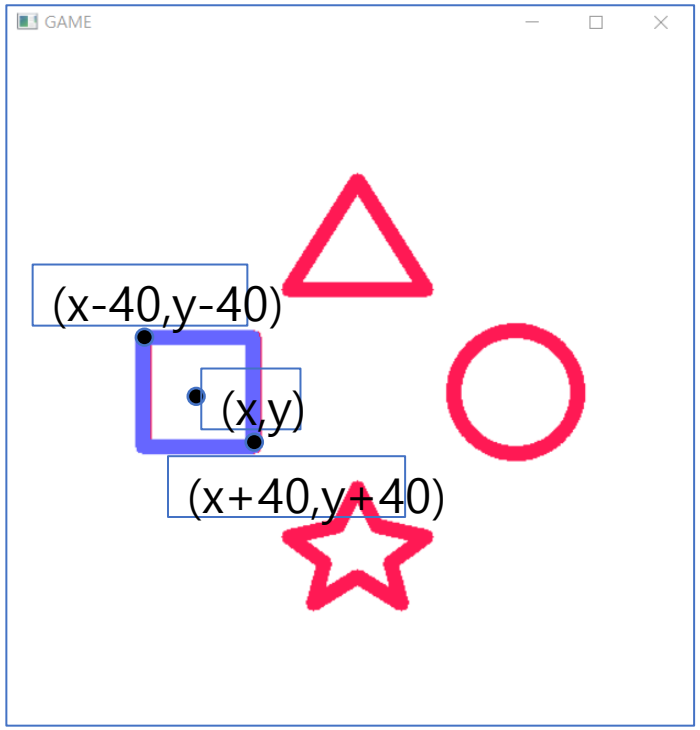
```
# 랜덤 도형 생성
now = np.random.randint(0,4)

if now == 0:
    cv2.polylines(img_draw1, [random1], True, (255,102,102),10)
elif now == 1:
    cv2.circle(img_draw1, (255,255),45,(255,102,102),10)
elif now == 2:
    cv2.rectangle(img_draw1, (215,215),(295,295),(255,102,102),10)
elif now == 3:
    cv2.polylines(img_draw1, [random2], True, (255,102,102),10)
cv2.imshow('GAME', img_draw1)
```

1. random.randint()를 활용하여 반복문에서 도형을 랜덤으로 생성함
2. now : 현재의 랜덤 도형 출현
3. 0: 삼각형, 1: 원, 2: 사각형, 3: 별모양

사용자가 마우스로 위치를 변동시킬 수 있는 도형을 랜덤으로 생성함

도형 맞추기



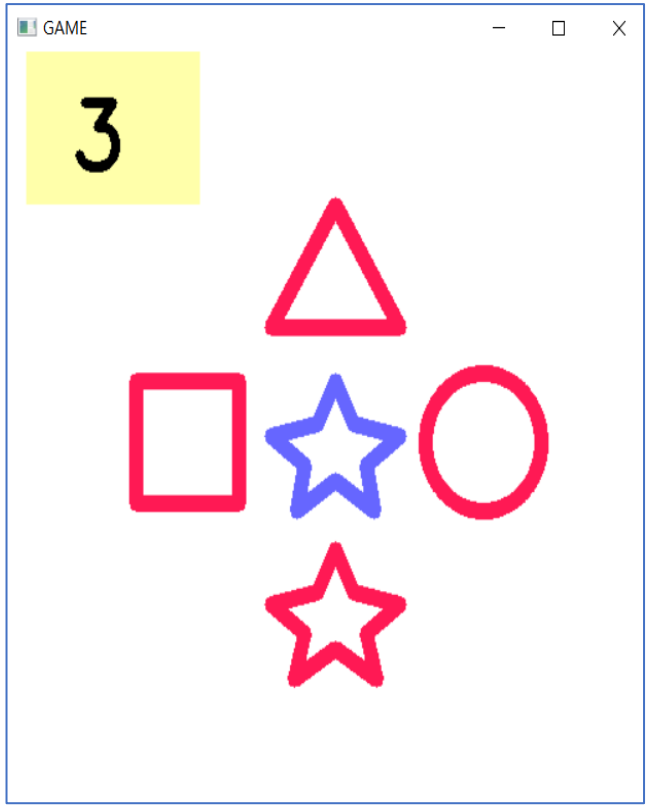
〈기본 도형_img〉

```
if isDragging:
    img_draw2 = img.copy()
    if now == 0:
        random1 = np.array([[x,y-40],[x-50,y+40],[x+50,y+40]],dtype=np.int32)
        cv2.polylines(img_draw2, [random1], True, (255,102,102),10)
    elif now == 1:
        cv2.circle(img_draw2, (x,y),45,(255,102,102),10)
    elif now == 2:
        cv2.rectangle(img_draw2, (x-40,y-40),(x+40,y+40),(255,102,102),10)
    elif now == 3:
        random2 = np.array([[x,y-54],[x-14,y-26],[x-50,y-18],[x-24,y+1],[x-30,y+31],₩
                            [x,y+11],[x+30,y+31],[x+24,y+1],[x+50,y-18],[x+14,y-26]],dtype=np.int32)
        cv2.polylines(img_draw2, [random2], True, (255,102,102),10)
    cv2.imshow('GAME',img_draw2)
```

onMouse 함수에서 얻은 마우스의 x,y 좌표를 각 도형의 중심좌표로 설정하고,

마우스의 이동에 따라 도형을 그리는 방식을 채택함

도형 맞추기



<기본 도형_img>

```
elif event == cv2.EVENT_LBUTTONUP:
    if isDragging:
        isDragging = False
        # 도형 매칭 확인
        if now == 0: # 삼각형
            if 235<x<275 and 120<y<160:
                cnt+=1
        elif now==1: #원
            if 350<x<390 and 235<y<275:
                cnt+=1
        elif now==2: #사각형
            if 120<x<160 and 235<y<275:
                cnt+=1
        elif now==3:
            if 235<x<275 and 360<y<400:
                cnt+=1
```

1. 랜덤 도형과 기본화면의 sample 도형을 매칭시키기 위해 좌표값을 이용함
2. 두 도형을 비교할 때, 위치 값이 +- 오차 범위 이내 랜덤한 도형이 도달하면,

   카운트 점수 +1 획득

3. 해당 카운트 점수를 cv2.putText()를 활용하여 표기함

숨은 그림찾기 - 윌리 지정하기

<기본 back_img>

+

<wally_img>

사용자가 지정한 곳에 wally_img를 입력하여 이미지를 숨긴다

숨은 그림찾기 - 윌리 지정하기

<wall_img를 합성한 back_img>

```python
def onMouse(event, x, y, flags, param):
    global dst, img2, cx, cy
    if event == cv2.EVENT_LBUTTONDOWN:
        #이미지 합성하기
        _, mask = cv2.threshold(dst[:,:,3], 1, 255, cv2.THRESH_BINARY)
        mask_inv = cv2.bitwise_not(mask)

        dst = cv2.cvtColor(dst, cv2.COLOR_BGRA2BGR)
        h, w=dst.shape[:2]
        roi = img2[y:y+h, x:x+w] #x좌표 y좌표
        #윌리 좌표
        cx = int((x+(x+w))/2)
        cy = int((y+(y+h))/2)
        print(cx,cy)

        masked_dst=cv2.bitwise_and(dst, dst, mask=mask)
        masked_img = cv2.bitwise_and(roi, roi, mask = mask_inv)

        added = masked_dst + masked_img
        img2[y:y+h, x:x+w]=added
        cv2.imshow("img1", img2)
        cv2.imwrite("./img/santa_wally.png", img2)
```

사용자 편의를 위한 이벤트 처리와 비트와이즈 연산자를 활용한 합성을 구현

숨은 그림찾기 - 윌리를 찾아라

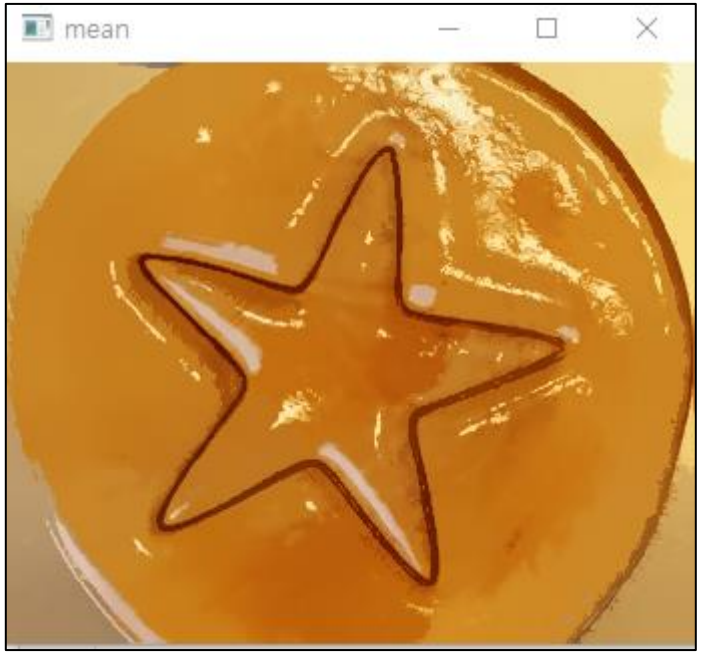
<wall_img를 합성한 back_img>

```
def onSearch(event, x, y, flags, param):
    global cx, cy, user_img, count, cuo, a
    if event==cv2.EVENT_LBUTTONDOWN:
        count+=1
        if abs(x-cx)<20 and abs(y-cy)<20: #위치가 가까우면 그냥..pass
            #cv2.circle(user_img, (x, y), 10, (0,255,0), 3)
            cv2.circle(user_img, (cx, cy), 10, (0,255,255), 3)
            succeed_rate = ((count-cuo)/count) *100
            #cv2.rectangle(user_img, (50,100),(430,170), (255,255,255),-1)
            #cv2.putText(user_img, "Succeed! {0:.1f}%".format(succeed_rate),
            cv2.imshow("user_img", user_img)
        else:
            cuo+=1
            cv2.line(user_img, (x-a,y-a),(x+a,y+a),(0,0,255),3)
            cv2.line(user_img, (x-a,y+a),(x+a,y-a),(0,0,255),3)
            cv2.imshow("user_img", user_img)
```

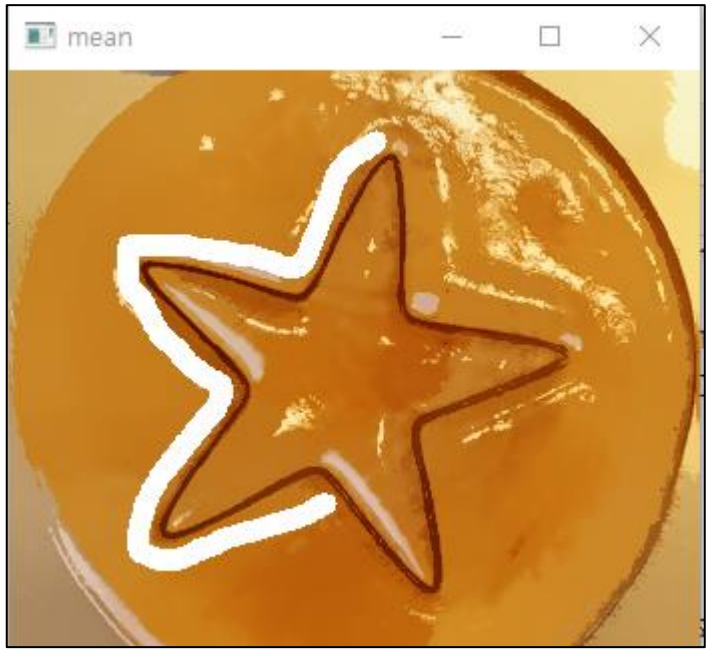
<cx, cy는 wally_img의 좌표>

사용자의 이벤트 x,y 좌표가 wally_img의 좌표와의 오차가 20미만이면 성공!

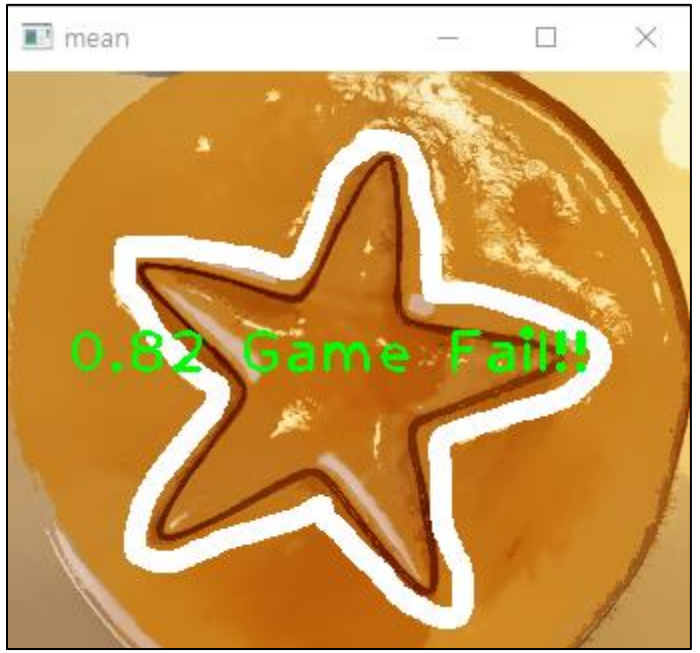
달고나 게임 - 컨투어



〈기본 달고나_img〉



〈사용자 입력 - 마우스 콜백〉



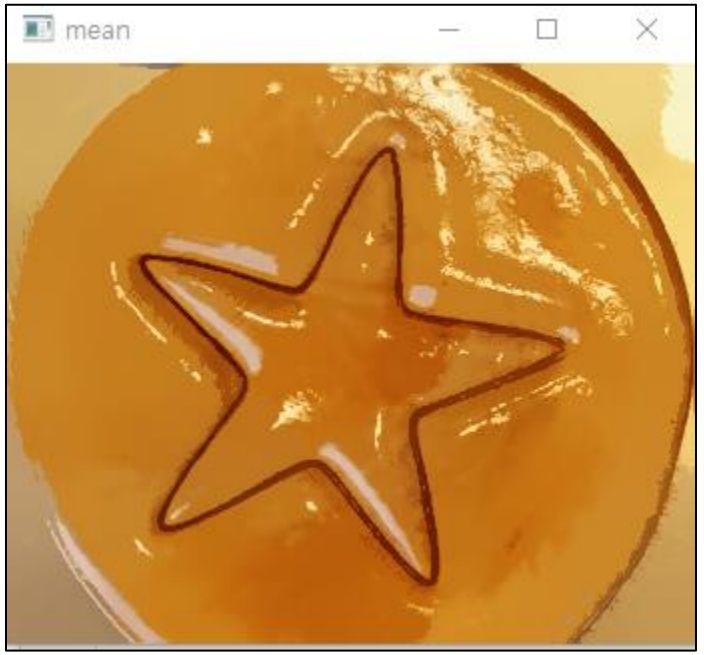
〈게임 결과 출력 - 매칭률〉

사용자가 그린 그림이 달고나 윤곽선과 (위치, 방향, 크기 등)

얼마나 비슷한 모양을 띄고 있는지 cv2.mathShapes()를 활용하여 매칭률 추출

달고나 게임 - 컨투어

<원본 달고나_img>



<달고나_img - 블러처리>

```python
#달고나 표면을 흐릿하게..피라미드 평균 시프트 적용
mean = cv2.pyrMeanShiftFiltering(img, 20, 30) #첫 이미지..
mean1=mean.copy()
gray = cv2.cvtColor(mean1, cv2.COLOR_BGR2GRAY)
gray=cv2.GaussianBlur(gray, (3,3), 0)
edges = cv2.Canny(gray, 100, 370)
```

평균 블러링

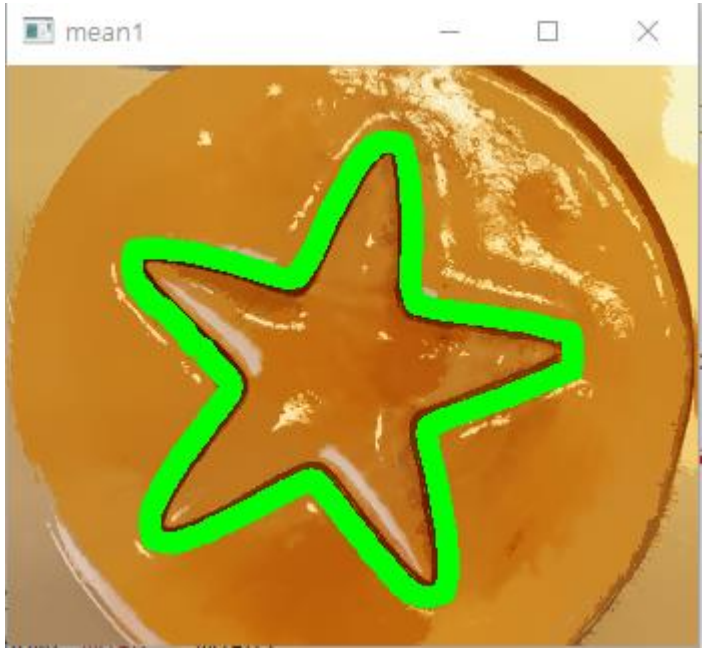
가우시안 블러 처리

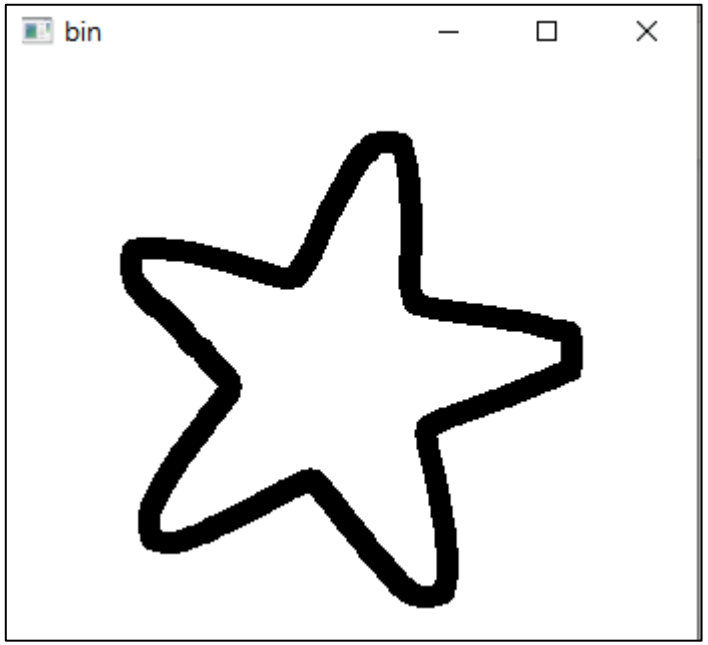
캣니엣지 검출 - for 뚜렷한 컨투어 검출

원본 달고나_img의 표면의 노이즈를 제거하고

컨투어를 원활하게 추출하기 위해 다양한 영상 필터 처리를 진행함

달고나 게임 - 컨투어






```
thresh_star = cv2.adaptiveThreshold(edges, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BIN
cntr_stars, _ = cv2.findContours(thresh_star, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

#for i in range(len(cntr_stars)):
cntr_star = cntr_stars[4]
cv2.drawContours(mean1, [cntr_star], -1, (0,255,0), 10)
cv2.drawContours(sample_star, [cntr_star], -1, (0,0,0), 10)
```

findContours()

drawContours()

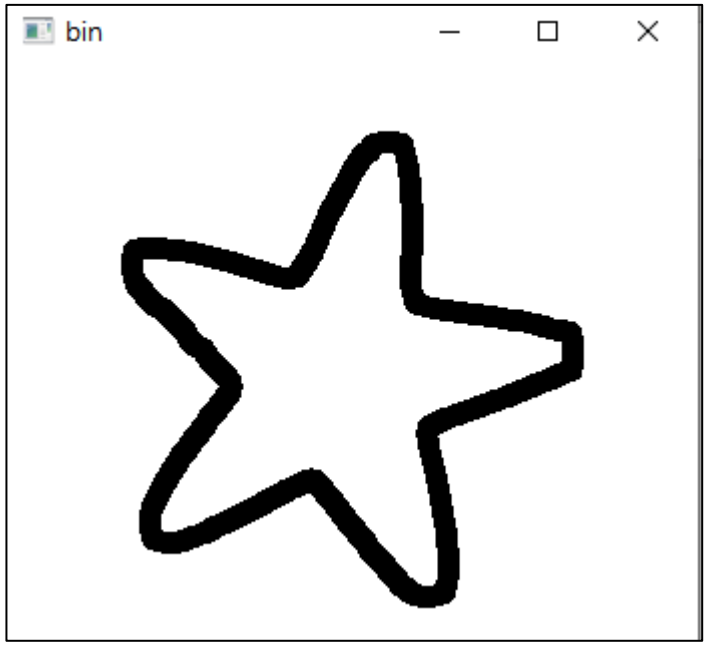
〈컨투어 추출〉

〈컨투어 sample_img〉

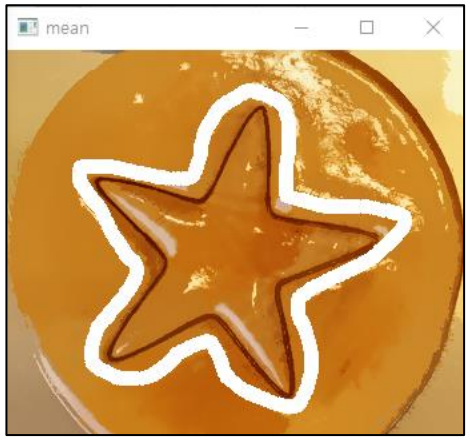
사실, 기본 달고나_img의 노이즈가 완전히 제거되지 않아, 많은 컨투어(약 138개)가 검출됨

캣니검출과 적응형 스레시홀드의 인자값을 활용하여 컨투어 추출을 조절에 성공

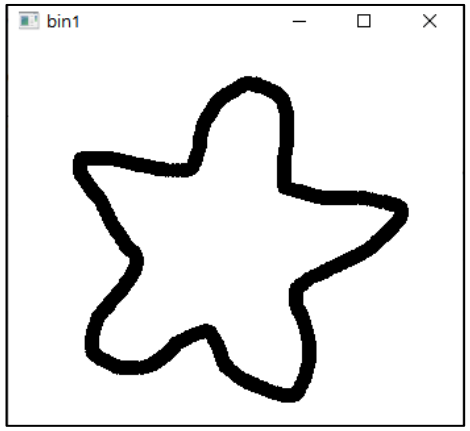
달고나 게임 - 컨투어



<컨투어 sample_img>



<사용자가 입력 컨투어_img>

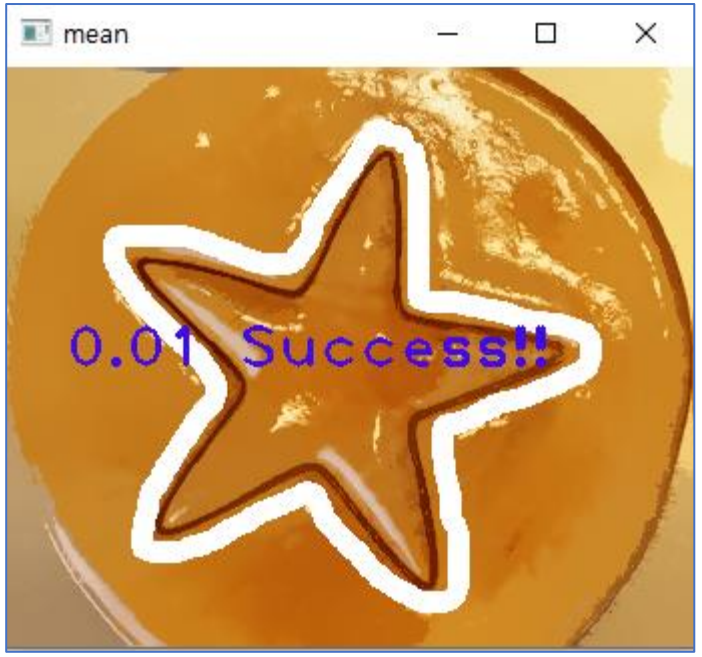



```
#매치율 확인하기
matchs=[]
for contr in user_cnt:
    match=cv2.matchShapes(sample_cnt[0], contr, cv2.CONTOURS_MATCH_I2,
    matchs.append((match, contr))
    #print(match)
    #print(tuple(contr[0][0]))
    if match<=0.05:
        cv2.putText(mean, "%.2f Success!!"%match, (30,150), cv2.FONT_H
    else:
        cv2.putText(mean, "%.2f Game Fail!!"%match, (30,150), cv2.FONT
cv2.imshow("mean", mean)
cv2.imshow("bin1", user_star)
```

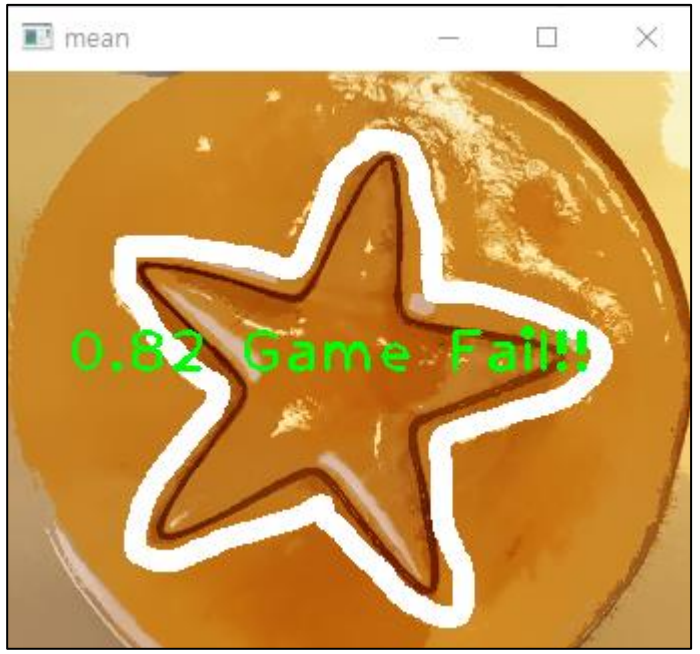
기본 컨투어 sample_img와 사용자가 입력한 컨투어(target_img)를

cv2.matShapes()에 활용하여 match결과가 0.05이하이면 합격

달고나 게임 - 컨투어



<달고나 게임 성공(예)>



<달고나 게임 실패(예)>

감사합니다!