# TRABAJO PRÁCTICO INTEGRADOR

*IA4.4 Procesamiento de Imagenes*

*Tecnicatura Universitaria en Inteligencia Artificial*

**Tomás Navarro Miñón**

**Mateo Gravi Fiorino**

**Alejo Lo Menzo**

**Ramiro Sagrera**

20/11/2023

Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

## INTRODUCCIÓN

Este es el segundo trabajo practico de la materia de procesamiento de imagenes de la Tecnicatura Universitaria en Inteligencia Artificial (TUIA).

El mismo consiste de 2 ejercicios:

El primero que se debe detectar y clasificar monedas y dados:

La imagen monedas.jpg, adquirida con un smartphone, consiste de monedas de distinto valor y tamaño, y de dados sobre un fondo de intensidad no uniforme (ver Figura 1).

a) Procesar la imagen de manera de segmentar las monedas y los dados de manera automática.

b) Clasificar los distintos tipos de monedas y realizar un conteo, de manera automática.

c) Determinar el número que presenta cada dado mediante procesamiento automático.

Mientras que el segundo, su finalidad es obtener, procesar y detectar patentes:
La carpeta Patentes contiene imágenes de la vista anterior o posterior de diversos vehículos donde se visualizan las correspondientes patentes. En Figura 2 puede verse una de las imágenes.
a) Implementar un algoritmo de procesamiento de las imágenes que detecte automáticamente las patentes y segmente las mismas. Informar las distintas etapas de procesamiento y mostrar los resultados de cada etapa.
b) Implementar un algoritmo de procesamiento que segmente los caracteres de la patente detectada en el punto anterior. Informar las distintas etapas de procesamiento y mostrar los resultados de cada etapa.

## EJERCICIO 1:

En este primer ejercicio el desafío era identificar las monedas y clasificarlas. Por otro lado, debíamos identificar los dados que aparecían e indicar el número que mostraban.

Primero importamos las librerías necesarias. También se pueden encontrar en requirements.txt

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Comenzamos declarando una función para graficar las imágenes.

```
def imshow(img, nueva_figura=True, titulo=None, img_a_color=False, bloqueante=True, barra_de_color=False, sin_ticks=False):
    print(img.shape)
    if nueva_figura:
        plt.figure()
    if img_a_color:
        plt.imshow(img)
    else:
        plt.imshow(img, cmap='gray')
    plt.title(titulo)
    if not sin_ticks:
        plt.xticks([]), plt.yticks([])
    if barra_de_color:
        plt.colorbar()
    if nueva_figura:
        plt.show(block=bloqueante)
```

A continuación cargamos la imágen, la pasamos a color gris. Luego definimos un kernel para usar en la función para dilatar las imágenes que será utilizada más adelante.

```
imagen = cv2.imread("monedas.jpg")
imagen_gris = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)

# kernel dilatación para cerrar los bordes
kernel_dilatacion = np.ones((10, 10), np.uint8)

# funcion para dilatar
Comment Code
def dilate_image(img, iterations=1):
    return cv2.dilate(img, kernel_dilatacion, iterations=iterations)
```

Seguimos con un leve suavizado de la imágen, aplicamos el Método Canny para los bordes y luego dilatamos la imagen. Por último realizamos una erosión para terminar de pulir.

```python
# Suavizar con Median/ probamos con Gaussian pero era mucho
imagen_suavizada = cv2.medianBlur(imagen_gris, 5)

# Usamos canny para los bordes
canny = cv2.Canny(imagen_suavizada, 50, 150)

# Dilatamos los bordes
bordes_dilatados = dilate_image(canny, iterations=3)

# Aplicamos erosión para eliminar pequeñas imperfecciones
img_er = cv2.erode(
    bordes_dilatados,
    cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (17, 17)),
    iterations=1,
)
```

Ahora podemos buscar los contornos, a su vez, creamos una máscara en blanco del mismo tamaño que la imagen original y dibujamos los contornos sobre ella.

```python
# Ahora si buscamos contornos
contornos, _ = cv2.findContours(img_er, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
# Contornos encontrados
len(contornos)

# Crear una máscara en blanco del mismo tamaño que la imagen original
mascara = np.zeros_like(imagen_gris)

# Dibujamos los contornos en la mascara
cv2.drawContours(mascara, contornos, -1, (255), thickness=cv2.FILLED)
```

Ya estamos listos para encontrar los componentes conectados, definimos las variables necesarias. (Los índices de las estadísticas se encuentran detallados en el código)

```python
# Encontramos componentes conectados en la máscara
(
    componentes_conectadas,
    etiquetas,
    estadisticas,
    centroides,
) = cv2.connectedComponentsWithStats(mascara, cv2.CV_32S, connectivity=8)
```

Una vez encontrados los componentes conectados, creamos una nueva imágen en blanco para palpar el resultado. Después iteramos para filtrar y dibujar los contornos según el área de los componentes. (El área la obtuvimos con las estadísticas.)

```python
# Crear una imagen en blanco para el resultado
resultado = np.zeros_like(img_er)

# Filtrar y dibujar contornos basados en el área de los componentes conectados
for i in range(1, componentes_conectadas):
    area = estadisticas[i, cv2.CC_STAT_AREA]

    if area > 600:
        mascara = np.uint8(etiquetas == i)
        contorno, _ = cv2.findContours(
            mascara, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
        )
        cv2.drawContours(resultado, contorno, -1, 120, 2)
```

Ya podemos rellenar los contornos en la imagen resultante.

```python
def fill_contours(img):
    seed_point = (0, 0)
    blanco = (255, 255, 255)

    flags = 4
    lo_diff = (10, 10, 10)
    up_diff = (10, 10, 10)

    cv2.floodFill(img, None, seed_point, blanco, lo_diff, up_diff, flags)
    return ~img

# Rellenar contornos en la imagen resultante
resultado = fill_contours(resultado)
```

Ahora toca repetir los pasos, aplicamos dilatación a la imagen resultante y encontramos sus componentes conectados.

```python
# Aplicar dilatación a la imagen resultante
result = cv2.dilate(
    resultado, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5)), iterations=5
)

# Encontrar componentes conectadas en la imagen resultante
(
    componentes_conectadas,
    etiquetas,
    estadisticas,
    centroides,
) = cv2.connectedComponentsWithStats(result, cv2.CV_32S, connectivity=8)
```

Ya podemos encontrar los cuadrados que representan los dados. Entonces definimos listas y variables e iteramos sobre los componentes que encontramos.

Utilizamos una máscara y contornos, y en base al área, el perímetro y el rho determinamos si es un cuadrado y lo dibujamos.

```
# Inicializar listas y variables para el procesamiento de cuadrados
squares_masks = []
aux = np.zeros_like(result)
labeled_image = cv2.merge([aux, aux, aux])
RHO_TH = 0.83

# Iterar sobre los componentes conectados y clasificar cuadrados
for i in range(1, componentes_conectadas):
    mascara = np.uint8(etiquetas == i)
    contorno, _ = cv2.findContours(mascara, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    area = cv2.contourArea(contorno[0])
    perimetro = cv2.arcLength(contorno[0], True)
    rho = 4 * np.pi * area / (perimetro**2)
    flag_circ = rho > RHO_TH
    if flag_circ:
        if area > 123000:
            labeled_image[mascara == 1, 0] = 255
        elif area > 85000:
            labeled_image[mascara == 1, 1] = 255
        else:
            labeled_image[mascara == 1, 2] = 255
    else:
        labeled_image[mascara == 1, 2] = 120
        labeled_image[mascara == 1, 1] = 120

        square_mask = np.zeros_like(result)
        cv2.drawContours(square_mask, contorno, -1, 120, 2)
        squares_masks.append(square_mask)
```

Combinamos ambas imágenes y establecemos un nuevo umbral para los cuadrados

```
# Combinar la imagen original con la imagen etiquetada
img_final = cv2.addWeighted(imagen, 0.7, labeled_image, 0.3, 0)

# Establecer un nuevo umbral para la circularidad de los cuadrados
RHO_TH = 0.78
```

Queda el paso final iteramos sobre las máscaras de los cuadrados y realizamos el procesamiento. Rellenamos los contornos y aplicamos canny a los cuadrados.
Luego dilatamos los cuadrados encontrados para procesarlos mejor. Volvemos a encontrar los componentes conectados en los cuadrados. Para ver qué número encontramos. Para eso seteamos un contador a 0. Iteramos en el rango de los componentes, seteamos el área a las estadísticas encontradas y una máscara. Filtramos los cuadrados Y vamos contando los cuadrados circulares.

```
# Iterar sobre las máscaras de los cuadrados y realizar operaciones de procesamiento
for id, square_mask in enumerate(squares_masks):
    square_mask = fill_contours(square_mask)
    squares = cv2.bitwise_and(canny, canny, mask=square_mask)

    # Aplicar dilatación a los cuadrados
    squares_dilatados = dilate_image(squares)

    # Encontrar componentes conectadas en los cuadrados
    (
        componentes_conectadas,
        etiquetas,
        estadisticas,
        centroides,
    ) = cv2.connectedComponentsWithStats(squares_dilatados, cv2.CV_32S, connectivity=8)

    count = 0 # Contador para numeros de dados
    for i in range(1, componentes_conectadas):
        area = estadisticas[i, cv2.CC_STAT_AREA]
        mascara = np.uint8(etiquetas == i)
        contorno, _ = cv2.findContours(
            mascara, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
        )

        # Filtrar cuadrados por área
        if area < 10000 and area > 500:
            area = cv2.contourArea(contorno[0])
            perimetro = cv2.arcLength(contorno[0], True)
            rho = 4 * np.pi * area / (perimetro**2)
            flag_circ = rho > RHO_TH

            # Contar cuadrados circulares
            if flag_circ:
                count += 1

    print(f"Dado {id + 1}, muestra el numero: {count}")

# Mostrar el resultado
imshow(img_final)
```

## EJERCICIO 2:

Este ejercicio, como mencionamos anteriormente, consiste en poder identificar y detectar patentes de autos. Para realizar el mismo, desde la cátedra nos dieron fotos obtenidas donde debíamos obtener la patente y obtener cada uno de los caracteres de la misma.

Lo primero que realizamos es importar las librerías que vamos a utilizar

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
```

Vamos a usar numpy para realizar operaciones matemáticas complejas. Matplotlib.pyplot para realizar plots en python (o sea poder visualizar la imagen). Cv2 la usamos para el procesamiento de imágenes y OS es para poder verificar archivos desde el sistema.

Para poder comprender mejor el codigo y lo que hace, lo que resolvimos fue crear funciones donde en cada una se realicen operaciones diferentes y se muestre como va quedando la imagen paso a paso

Para poder visualizar la imagen, definimos una función imshow (la misma la hemos usado muchas veces en clases). El objetivo de la misma es que nos imprima la imagen en pantalla

```
Comment Code
def imshow(img, new_fig=True, title=None, color_img=False, blocking=True, colorbar=False, ticks=False):
    print(img.shape)
    if new_fig:
        plt.figure()
    if color_img:
        plt.imshow(img)
    else:
        plt.imshow(img, cmap='gray')
    plt.title(title)
    if not ticks:
        plt.xticks([]), plt.yticks([])
    if colorbar:
        plt.colorbar()
    if new_fig:
        plt.show(block=blocking)
```

También definimos una función gray_scale(), esta se utiliza para transformar la imagen a escala de grises ya que este normalmente es el primer paso que usamos cuando empezamos a procesar imágenes

```
def gray_scale(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imshow(gray)
    return gray
```

Posterior a este paso de transformar la imagen en gris, lo que hacemos es utilizar el método de umbralado adaptativo de Gauss, y muestra la imagen umbralizada. La elección de un umbral adaptativo es útil para manejar variaciones locales en la iluminación de la imagen.

```
def adaptive_threshold(img):
    tresh = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 9, -10)
    imshow(tresh, new_fig=True, title="Tresh")
    return tresh
```

La función connected_components() identifica y filtra componentes conectados en una imagen binarizada (pq le realizamos un umbralado previo) según parámetros específicos de área y relación de aspecto que les pasemos en una variable declarada, devolviendo una imagen binarizada filtrada.

```python
def connected_components(img_tresh, min_area, max_area, min_aspect_ratio, max_aspect_ratio):
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(img_tresh, connectivity=4)

    filtered_area = np.zeros_like(img_tresh)

    for i in range(1, num_labels):
        aspect_ratio = stats[i, cv2.CC_STAT_WIDTH] / stats[i, cv2.CC_STAT_HEIGHT]

        if (min_area < stats[i][4] and stats[i][4] < max_area and min_aspect_ratio < aspect_ratio and aspect_ratio < max_aspect_ratio):
            filtered_area[labels == i] = img_tresh[labels == i]

    return filtered_area
```

Continuando definimos a la función remove_noise_and_smooth(). Esta función lo que realiza es utilizar operaciones morfológicas (apertura y cierre) después de eliminar el ruido para mejorar la calidad de la imagen binarizada, haciendo que los objetos sean más definidos y eliminando pequeños elementos no deseados.

```python
def remove_noise_and_smooth(filtered_area, distance_threshold):
    clean_area = remove_noise(filtered_area.copy(), distance_threshold)

    se = cv2.getStructuringElement(cv2.MORPH_RECT, (1, 1))
    clean_area_open = cv2.morphologyEx(clean_area, cv2.MORPH_OPEN, se)
    final_image = cv2.morphologyEx(clean_area_open, cv2.MORPH_CLOSE, se)
    imshow(final_image)

    return final_image
```

ya casi finalizando definimos la función extract_characters() donde esta función lo que hace es identificar y mostrar cada componente conectado en una región de interés de la imagen binarizada después de aplicar operaciones morfológicas. Cada componente se muestra como una máscara binaria en una figura separada. Esto lo hacemos para que nos muestre caracter por caracter de la patente.

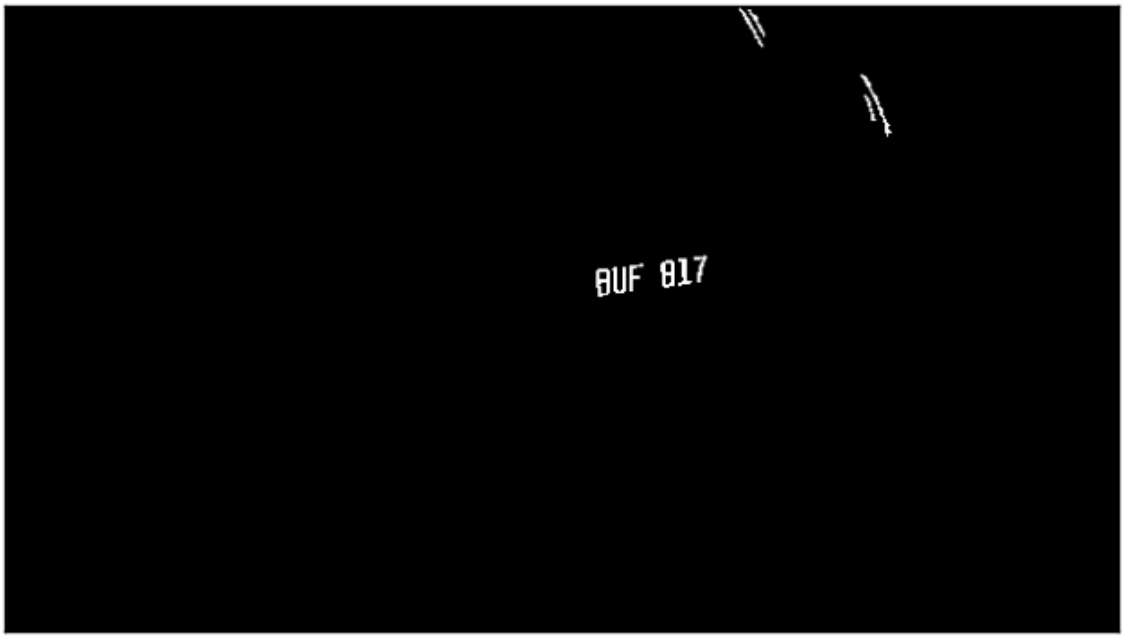
```python
def extract_characters(final_image):
    w, h = final_image.shape
    num_labels, labels, _, _ = cv2.connectedComponentsWithStats(final_image[80:w-50, 170:h-50], connectivity=4)

    for label in range(1, num_labels):
        component_mask = (labels == label).astype(np.uint8) * 255
        imshow(component_mask)
```

y su funcionamiento:

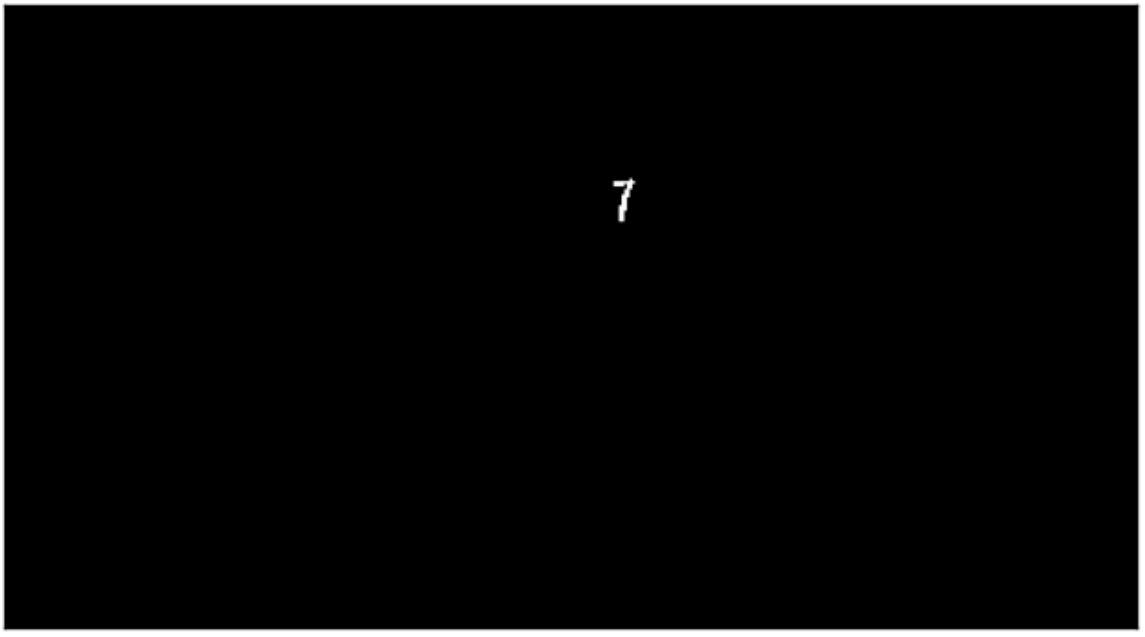
Imagen con escalado de grises


Imagen con umbralado adaptativo

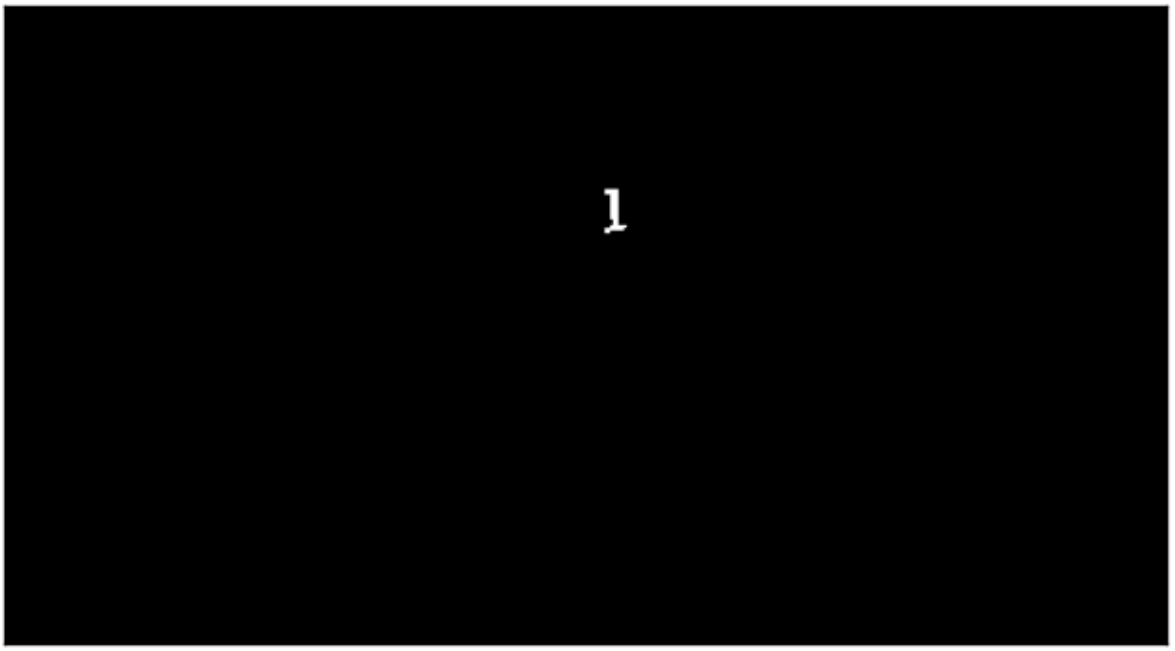
Imagen con filtrado final



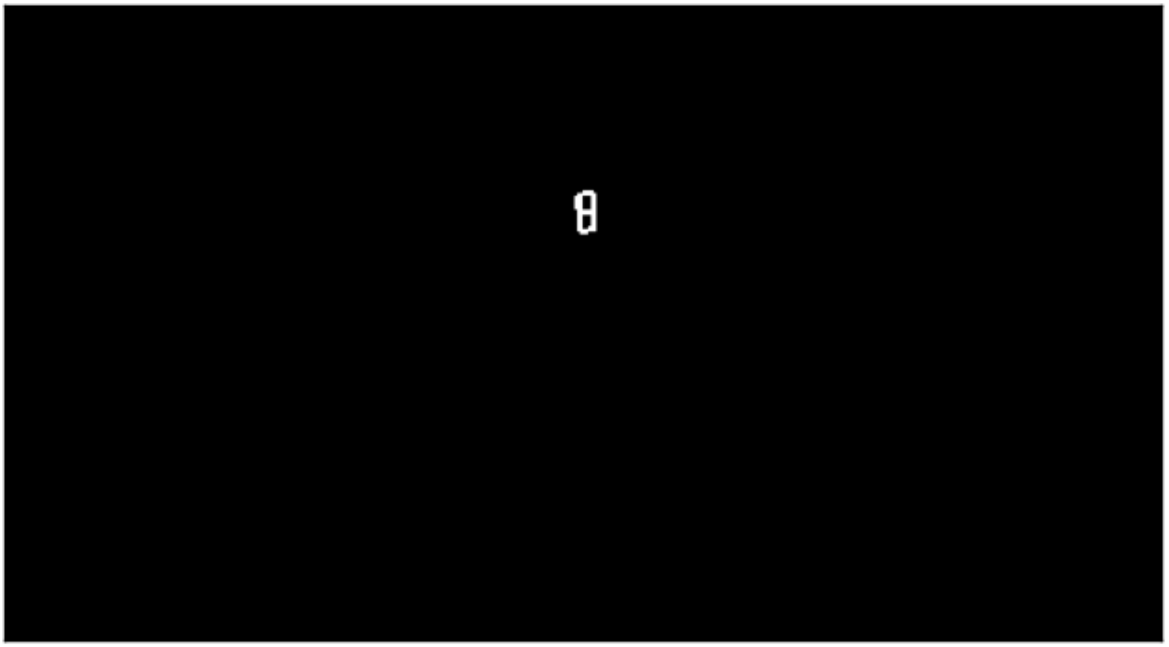

Componentes



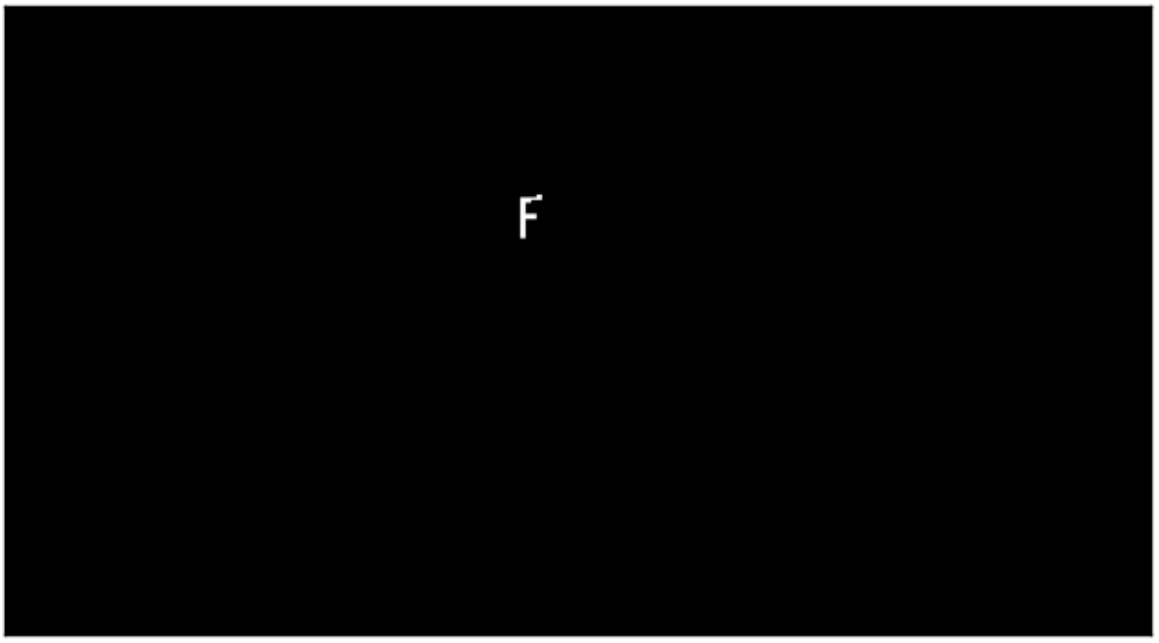
Componentes



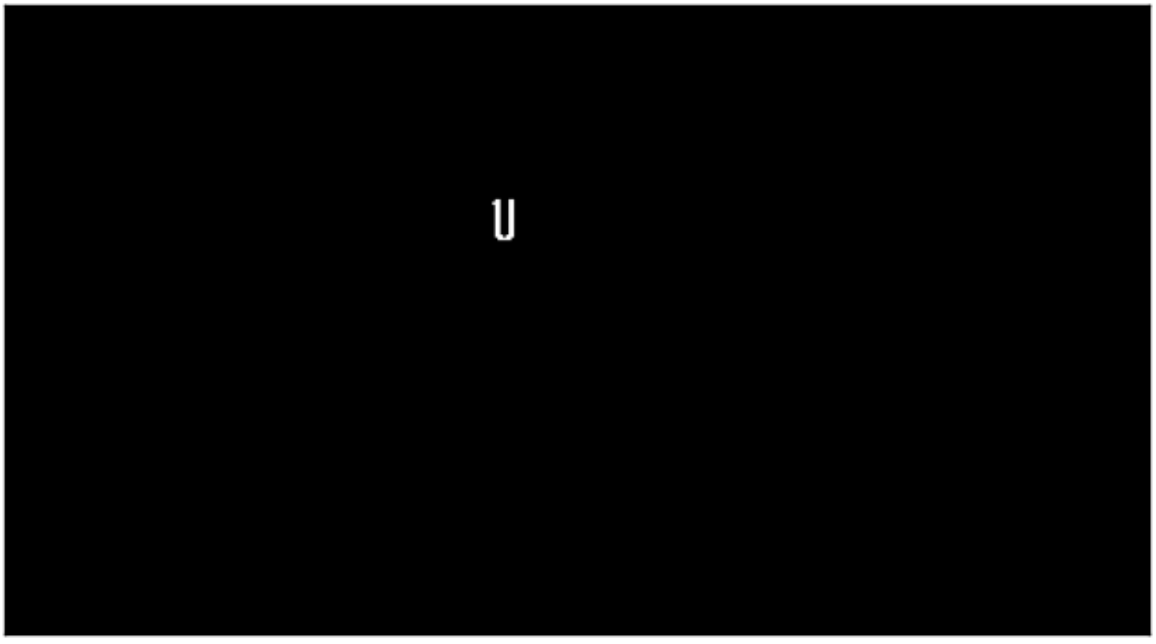

Componentes



Componentes




Componentes



Componentes