


Informe técnico “Hands Battle”


1. Funcionamiento General del Programa

El programa implementa un juego por turnos para dos jugadores. El objetivo es ser el primero en ganar 3 rondas. En cada ronda:

Instrucciones: Al inicio, se muestran las instrucciones del juego, explicando qué gesto de la mano (número de dedos levantados) corresponde a cada arma y qué arma vence a cuál.

Un dedo: "Quick Sword" (Espada Rápida) 

Dos dedos: "Double Daggers" (Dagas Dobles) 

Cinco dedos (o cuatro o más): "Great Shield" (Gran Escudo) 

La Espada Rápida vence a las Dagas Dobles.

Las Dagas Dobles vencen al Gran Escudo.

El Gran Escudo vence a la Espada Rápida.

Turno del Jugador: Cada jugador, en su turno, tiene 3 segundos para mostrar su mano a la cámara.

Captura y Detección: El programa captura una imagen de la cámara y utiliza OpenCV para procesarla y detectar el número de dedos levantados, determinando así el "arma" elegida.

Determinación del Ganador de la Ronda: Se comparan las armas elegidas por ambos jugadores para determinar el ganador de la ronda, o si es un empate.

Puntuación: Se actualiza la puntuación. Si un gesto no es reconocido, se considera una ronda inválida para esa elección.

Fin del Juego: El juego continúa hasta que un jugador alcanza 3 puntos. Se declara al ganador y se muestra la puntuación final.

1. Introducción

En este proyecto echo en base en OpenCV con el lenguaje de C++ se realizo un programa que detectara gestos de mano para un juego, se utilizo de referencias videos adjuntados por el profesor Gerardo Franco y también se utilizo la ayuda de paginas web y ciertos videos para la comprensión de el código y posteriormente su realización.

2. Implementación de OpenCV

OpenCV (Open Source Computer Vision Library) es fundamental para la funcionalidad de detección de gestos. Las siguientes son las etapas clave de su implementación en la función **detectWeapon(Mat frame)**:

Captura de Imagen:

Se utiliza VideoCapture cap(0); para acceder a la cámara web predeterminada.

cap.set(CAP_PROP_FRAME_WIDTH, 640); y cap.set(CAP_PROP_FRAME_HEIGHT, 480); establecen la resolución de la captura.

cap >> frame; captura un fotograma de la cámara.

Preprocesamiento de la Imagen:

cvtColor(frame, gray, COLOR_BGR2GRAY); Convierte la imagen capturada de formato BGR (color) a escala de grises. Esto simplifica el análisis al reducir la cantidad de información de color.

GaussianBlur(gray, blurred, Size(9, 9), 0); Aplica un filtro Gaussiano para suavizar la imagen y reducir el ruido. El **kernel de Size(9, 9)** indica el tamaño de la matriz de convolución; valores mayores producen más desenfoque.

threshold(blurred, thresholded, 100, 255, THRESH_BINARY_INV);: Realiza una **umbralización (thresholding)** para convertir la imagen en escala de grises a una imagen binaria (blanco y negro).

Píxeles con intensidad mayor a 100 se convierten a 0 (negro), y los menores o iguales a 100 se convierten a 255 (blanco) debido a **THRESH_BINARY_INV** (umbral binario invertido). Esto ayuda a segmentar la mano del fondo, asumiendo que la mano es más oscura que el fondo o viceversa, dependiendo de la iluminación y el umbral.

Detección de Contornos:

findContours(thresholded, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);:
Encuentra los contornos en la imagen binaria.

RETR_EXTERNAL recupera solo los contornos externos.

CHAIN_APPROX_SIMPLE comprime los segmentos horizontales, verticales y diagonales, dejando solo sus puntos finales, ahorrando memoria.

La función **findMaxContour(contours)** itera sobre todos los contornos encontrados y devuelve aquel con el área más grande, asumiendo que este es el contorno de la mano.
contourArea(contour) calcula el área de un contorno.

Análisis de la Mano (**Convex Hull y Defectos de Convexidad**):

convexHull(handContour, hull, false, false);: Calcula la envolvente convexa (**convex hull**) del contorno de la mano. La envolvente convexa es el polígono convexo más pequeño que encierra todos los puntos del contorno. El hull resultante es un vector de índices de los puntos del contorno que forman la envolvente convexa.

convexityDefects(handContour, hull, defects);: Encuentra los defectos de convexidad. Un defecto de convexidad es una desviación del contorno respecto a su envolvente convexa. Estos defectos son cruciales para identificar los espacios entre los dedos.

defects es un vector de Vec4i, donde cada elemento **defect** contiene:

defect[0]: Índice del punto de inicio del defecto en **handContour**.

defect[1]: Índice del punto final del defecto en **handContour**.

defect[2]: Índice del punto más lejano del defecto (el valle entre los dedos) en **handContour**.

defect[3]: Distancia aproximada del punto más lejano al casco convexo (profundidad del defecto).

Conteo de Dedos:

Se itera sobre los **defects** encontrados.

float depth = defect[3] / 256.0f; Se calcula la profundidad del defecto. El valor original está escalado, por lo que se divide por 256.0.

Se filtran los defectos por profundidad (**if (depth > 20)**): Solo los defectos suficientemente profundos se consideran espacios entre dedos. Este umbral (20) es empírico y puede necesitar ajuste según las condiciones de iluminación y la distancia de la mano a la cámara.

Para cada defecto válido, se calculan los puntos de inicio (start), fin (end) y el punto más lejano (farPt).

Se calcula el ángulo formado por estos tres puntos usando la ley de los cosenos: **double angle = acos((b * b + c * c - a * a) / (2 * b * c)) * 180 / CV_PI;** donde a, b, y c son las longitudes de los lados del triángulo formado por start, end, y farPt.

if (angle < 90): Si el ángulo en el punto más lejano (el valle entre los dedos) es menor a 90 grados, se considera un dedo levantado. Este es un criterio común para distinguir los dedos extendidos.

int totalFingers = fingerCount + 1; El número total de dedos se asume como el conteo de espacios válidos más uno (ya que, por ejemplo, un puño no tiene defectos pero representa un estado, aunque en este juego no se usa, y una mano con un dedo levantado no tiene "espacios" entre dedos contados por **convexityDefects** de esta manera). Nota: Esta lógica es una simplificación común en la detección de dedos. La fiabilidad puede variar.

Asignación de Arma:

Según el totalFingers calculado:

1 dedo: "sword"

2 dedos: "daggers"

4 o más dedos: "shield" (el código dice >= 4, lo que incluye 5 dedos)

Cualquier otro conteo (o si no se detectan contornos/defectos): "unknown"

Visualización y Limpieza:

imshow("Player X", frame); Muestra la imagen capturada en una ventana.

waitKey(500); Espera 500 milisegundos. Esto permite que la ventana se actualice y sea visible.

destroyWindow("Player X"); Cierra la ventana de visualización después de la selección del jugador.

3. Estructuras de Control Utilizadas

El programa utiliza diversas estructuras de control de C++ para gestionar el flujo del juego, la lógica de decisión y las iteraciones:

Bucles while:

while (p1Score < 3 && p2Score < 3): Es el bucle principal del juego. Continúa ejecutando rondas mientras ninguno de los jugadores haya alcanzado los 3 puntos necesarios para ganar.

Bucles for:

for (const auto& contour : contours) (en findMaxContour): Itera sobre cada contorno en la lista de contornos para encontrar el de mayor área. Es un bucle for-each basado en rango.

for (const auto& defect : defects) (en detectWeapon): Itera sobre cada defecto de convexidad para analizar los espacios entre los dedos.

for (int i = 3; i > 0; --i) (en main): Implementa la cuenta regresiva antes de que cada jugador muestre su mano.

Declaraciones if-else if-else:

En **findMaxContour**: **if (area > maxArea)** para actualizar el contorno de área máxima.

En detectWeapon:

if (contours.empty()) return "unknown"; y if (handContour.empty()) return "unknown";
para manejar casos donde no se detectan contornos.

if (hull.size() > 3) para asegurar que haya suficientes puntos en la envolvente convexa para calcular defectos.

if (depth > 20) para filtrar defectos de convexidad por su profundidad.

if (angle < 90) para contar un dedo basado en el ángulo del defecto.

if (totalFingers == 1) ... else if (totalFingers == 2) ... else if (totalFingers >= 4) para asignar el arma basada en el conteo de dedos.

En **determineWinner**: Se utiliza una cadena de if-else if para determinar el ganador de la ronda basado en las reglas del juego (espada > dagas, dagas > escudo, escudo > espada).

if (weapon1 == "unknown" || weapon2 == "unknown") return -1;

if (weapon1 == weapon2) return 0;

if ((weapon1 == "sword" && weapon2 == "daggers") || ...)

En weaponCodeToName: Se usa if-else if para convertir el código interno del arma a un nombre legible.

En main:

if (!cap.isOpened()) para verificar si la cámara se abrió correctamente.

if (frame.empty()) para verificar si la captura de imagen fue exitosa.

if (result == 1) ... else if (result == 2) ... else if (result == 0) ... else para procesar el resultado de la ronda y actualizar las puntuaciones.

if (p1Score == 3) ... else para anunciar el ganador final del juego.

Constantes y Arreglos:

const string **WEAPONS[3]** y const string **WEAPON_CODES[3]** almacenan los nombres y códigos de las armas, proporcionando una forma centralizada de gestionar estas cadenas.

Funciones: El código está bien modularizado en funciones, cada una con una responsabilidad específica:

findMaxContour: Aísla la lógica de encontrar el contorno más grande.

detectWeapon: Encapsula toda la lógica de procesamiento de imágenes y detección de gestos.

determineWinner: Contiene las reglas para decidir el resultado de una ronda.

weaponCodeToName: Facilita la presentación de la información al usuario.

showInstructions: Muestra las reglas al inicio.

main: Orquesta el flujo general del juego.

Manejo de Tiempo y Concurrencia (Básica):

#include <thread> y **#include <chrono>** se utilizan para las pausas en el juego.

this_thread::sleep_for(chrono::seconds(1)); y
this_thread::sleep_for(chrono::seconds(2)); introducen retrasos para mejorar la jugabilidad (cuenta regresiva, pausa entre rondas).

Trabajo realizado por:

Jose Manuel Niño Ibarra

Jose Manuel González Echavarría