

Universidad Autónoma de Tamaulipas

Facultad de Ingeniería Tampico



ASIGNATURA

Automatización y Robótica

9no. Semestre – Grupo “J”
2025 - 3

Ejercicios

Evidencia de ejercicios complementarios

Docente: Alejandro H. García Ruiz

Integrantes:

Aldama Trinidad Alfonso Rene
Clemente Villegas José Adán
Cristóbal Francisco Jesús Marcelino
Domínguez Reyes Pavel Noel



Índice

Evidencia de trabajos	3
Repository(s).....	3
Ejercicios.....	3
CNN Modelos > 85%	3
CNN Tiempo real	7
Modelos OpenCV	11
MediaPipe 1	12
MediaPipe 5 gestos con direcciones	14
Carrito entorno virtual	20
Circuito realizado	26
Prueba de circuito con manejo de teclado	27
Laberinto procedural	33
Laberinto completado de forma manual	38
Laberinto completado de forma automática	44
Karel letra	47
Karel come 3 Beepers	50
Reconocimiento de voz y simulación	53
Camino con opciones*	60
Grafo ponderado búsqueda local	66
Cadenas de Markov	73
PID	74



Evidencia de trabajos

Repositorio(s)

Ejercicios	Repository
Unidad 2 – Unidad 3	https://github.com/adanclev/AyR_2025_3.git

Ejercicios

CNN Modelos > 85%

Descripción

Este ejercicio se dividió en dos: el entrenamiento y la evaluación del modelo creado. En la primera etapa se construyó y entrenó una red neuronal convolucional (CNN) utilizando imágenes organizadas por clases.

En la segunda etapa se cargó el modelo entrenado y se probó contra un conjunto de imágenes para medir su desempeño. El objetivo era obtener un modelo con una precisión mayor al 85%.

Código

Entrenamiento de una CNN para reconocer rostros

```
import os
#from tensorflow.keras .... o .... from tensorflow.python.keras ...
#from keras.preprocessing.image import ImageDataGenerator # preprocesamiento de
imagenes
from keras import optimizers # algoritmos para entrenar
from keras.models import Sequential # para crear redes neuronales secuenciales
from keras.layers import Dropout, Flatten, Dense, Activation #
from keras.layers import Convolution2D, MaxPooling2D # capas de la red neuronal
from keras import backend as K
from keras.src.legacy.preprocessing.image import ImageDataGenerator

K.clear_session()

data_entrenamiento = "../../Archivos/Clases-individuos/F1-Entrenamiento"
data_validacion = "../../Archivos/Clases-individuos/F2-Validacion"

#Parametros
epocas = 5
alto, largo = 300, 300 #dimensiones de las imagenes. Para redimensionar
batch_size = 5 # numero de imagenes que se mandara a procesar por cada paso # 16
pasos = 35 # 100imagenes / batch => max pasos #60
pasos_validacion = 15 # 50imagenes /batch => max pasos validacion #25

#To make sure that you have "at least steps_per_epoch * epochs batches", set the
steps_per_epoch to
#steps_per_epoch = len(X_train)//batch_size
#validation_steps = len(X_test)//batch_size # if you have validation data
########

kernel1 = (3, 3)
```

```

kernel2 = (2, 2)
kernel3 = (3, 3)

tot_kernels1 = 48
tot_kernels2 = 64
tot_kernels3 = 128

stride = (2, 2) #para MaxPooling

clases = 4 #total de clases a clasificar

lr = 0.0005 #learning rate # 0.1 ---- 0.01 0.001 0.002

#preprocesamiento de imagenes --> aumento de datos...
entrenamiento_datagen = ImageDataGenerator(
    rescale=1./255, # cada px va de 0 a 255, con esto se pasara al rango de 0 a
1, para procesar mas facil
    shear_range=0.3, #inclinacion
    zoom_range= 0.3,
    vertical_flip=True,
    horizontal_flip=True #inversion horizontal
)

validacion_datagen = ImageDataGenerator(
    rescale=1./255
)

imagen_entrenamiento = entrenamiento_datagen.flow_from_directory(
    data_entrenamiento,
    target_size= (alto, largo),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode="grayscale"
)

imagen_validacion = validacion_datagen.flow_from_directory(
    data_validacion,
    target_size=(alto, largo),
    batch_size=batch_size,
    class_mode='categorical',
    color_mode="grayscale"
)

#red convolucional

cnn = Sequential()

##capa 1
cnn.add(Convolution2D(tot_kernels1, kernel1, padding='same', input_shape=(alto,
largo, 1), activation='relu'))
##capa 2
cnn.add(MaxPooling2D(pool_size=stride))
##capa 3
cnn.add(Convolution2D(tot_kernels2, kernel2, padding='same', activation='relu'))
##capa 4
cnn.add(MaxPooling2D(pool_size=stride))
##capa 5
cnn.add(Convolution2D(tot_kernels3, kernel3, padding='same', activation='relu'))
##capa 6
cnn.add(MaxPooling2D(pool_size=stride))

```



```
cnn.add(Flatten()) # aplana la informacion

##capa 7
cnn.add(Dense(256, activation='relu')) #

# 0.2 - 0.6
cnn.add(Dropout(0.4)) #porcentaje de neuronas apagadas en cada paso (0.5 = 50%)
# permite aprender caminos alternos para clasificar.. evita sobreentrenamiento

#capa 8 - salida'
cnn.add(Dense(clases, activation='softmax'))

cnn.compile(loss='categorical_crossentropy',
optimizer=optimizers.Adam(learning_rate=lr), metrics=['accuracy'])

#entrena el modelo de la red neuronal
cnn.fit(imagen_entrenamiento, steps_per_epoch=pasos, epochs=epocas,
validation_data= imagen_validacion, validation_steps=pasos_validacion)

dir = "/modelo/"
if not os.path.exists(dir):
    os.mkdir(dir)

cnn.save(dir + 'modelo.keras') #estructura
cnn.save_weights(dir + 'pesos.weights.h5') #pesos en las capas
```

Evaluá la eficiencia del modelo probando con las imágenes de prueba

```
import os
import numpy as np
#from keras.preprocessing.image import load_img, img_to_array #deprecated en tf
2.9
#from tensorflow.keras.utils import load_img #alternative 1
from keras.utils import load_img, img_to_array #alternative 2

from keras.models import load_model

# Convolutional Neuronal Network
alto, largo = 300, 300
modelo = "../E04_CNN/modelo/modelo.keras" #./modelo/modelo.h5'
pesos = '../E04_CNN/modelo/pesos.weights.h5'

cnn = load_model(modelo)
cnn.load_weights(pesos)
UMBRAL = 0.85

def predict(file):
    imagen_a_predecir = load_img(file, target_size = (alto, largo),
color_mode="grayscale")
    imagen_a_predecir = img_to_array(imagen_a_predecir)
    imagen_a_predecir = imagen_a_predecir / 255.0 ## nueva
    imagen_a_predecir = np.expand_dims(imagen_a_predecir, axis=0) #agrega una
dimension adicional
    arreglo = cnn.predict(imagen_a_predecir) ## [[1,0,0,0,0,0]]
    resultado = arreglo[0]
    #print(resultado)

    probabilidad_maxima = np.max(arreglo[0])

    respuesta = np.argmax(resultado) #indice del valor mas alto
```



```
if probabilidad_maxima >= UMBRAL:
    match respuesta:
        case 0:
            return 'C1-Adan'
        case 1:
            return 'C2-Poncho'
        case 2:
            return 'C3-Pavel'
        case 3:
            return 'C4-Cristobal'
        case _:
            return '----'
else:
    print("Desconocido")

#predict('/Users/alejandrohumbertogarciaruiz/Desktop/introTensorFlow/F3-Prueba/2-
cisne_cisne_31.jpg')

def get_folders_name_from(from_location):
    list_dir = os.listdir(from_location)
    # folders = [archivo for archivo in listDir if os.path.splitext(archivo)[1]
    == ""]
    # the above is equals to ....
    folders = []
    for file in list_dir:
        temp = os.path.splitext(file)
        if temp[1] == "":
            folders.append(temp[0])
    folders.sort()
    # folders.remove('.DS_Store') #solo en mac
    return folders

#foto cuando fue creado el modelo, foto eficiencia y foto
# Proyecto 1 - Tensorflow -> 85 %
# proyecto 2 - Entrenar
def probar_red_neuronal():
    base_location = "../../Archivos/Clases-individuos/F3-Prueba/" # ./F3-Prueba/
    #base_location = "../E04_CNN/F3-Prueba/" # ./F3-Prueba/

    folders = get_folders_name_from(base_location)

    correct = 0
    count_predictions = 0
    for folder in folders:
        files = [archivo for archivo in os.listdir(base_location + '/' + folder)
if archivo.endswith(".jpg") or archivo.endswith(".jpeg") or
archivo.endswith(".png")]
        for file in files:
            composed_location = base_location + folder + '/' + file
            prediction = predict(composed_location)
            print('Folder Name: ', folder, ' Prediction: ', prediction, "
Resultado: ", prediction in folder)
            count_predictions += 1
            if prediction in folder:
                correct += 1

    print("Efficiency: ", (correct/count_predictions*100))

probar_red_neuronal()
```

Ejecución

```

Epoch 1/10
35/35    0s 1s/step - accuracy: 0.2810 - loss: 2.0148C:\Users\toto_\OneDrive\Documentos\Inteligencia arti
          self._warn_if_super_not_called()
35/35    43s 1s/step - accuracy: 0.3657 - loss: 1.5581 - val_accuracy: 0.4533 - val_loss: 1.1644
Epoch 2/10
35/35    35s 1s/step - accuracy: 0.6343 - loss: 0.8299 - val_accuracy: 0.8000 - val_loss: 0.5166
Epoch 3/10
35/35    35s 1s/step - accuracy: 0.7771 - loss: 0.5077 - val_accuracy: 0.8667 - val_loss: 0.2802
Epoch 4/10
35/35    37s 1s/step - accuracy: 0.7886 - loss: 0.4787 - val_accuracy: 0.7067 - val_loss: 0.4021
Epoch 5/10
35/35    36s 1s/step - accuracy: 0.8343 - loss: 0.3834 - val_accuracy: 0.7200 - val_loss: 0.7192
Epoch 6/10
35/35    34s 970ms/step - accuracy: 0.9029 - loss: 0.2909 - val_accuracy: 0.7467 - val_loss: 0.5283
Epoch 7/10
30/35    4s 918ms/step - accuracy: 0.8647 - loss: 0.3880C:\Users\toto_\OneDrive\Documentos\Inteligencia a
          self._interrupted_warning()
35/35    29s 835ms/step - accuracy: 0.8267 - loss: 0.4222 - val_accuracy: 0.7600 - val_loss: 0.4441
Epoch 8/10
35/35    34s 983ms/step - accuracy: 0.8171 - loss: 0.4037 - val_accuracy: 0.9733 - val_loss: 0.1587
Epoch 9/10
35/35    33s 954ms/step - accuracy: 0.9429 - loss: 0.1635 - val_accuracy: 0.6933 - val_loss: 0.7972
Epoch 10/10
35/35    34s 967ms/step - accuracy: 0.9029 - loss: 0.2394 - val_accuracy: 1.0000 - val_loss: 0.0484

Process finished with exit code 0
Folder Name: C4-Cristobal Prediction: C4-Cristobal Resulto: True
1/1    0s 78ms/step
Folder Name: C4-Cristobal Prediction: C4-Cristobal Resulto: True
Efficiency: 99.25

```

CNN Tiempo real

Descripción

El programa implementa un sistema de reconocimiento facial en tiempo real usando PyQt5 y una red neuronal convolucional (CNN). La UI indica si la persona fue reconocida.

Código

Manejo de UI de la cámara

```

from PyQt5 import uic, QtWidgets, QtGui, QtCore
from PyQt5.QtSvg import QSvgRenderer
from camera_thread import CameraThread
from config import STYLES, MODELO, PESOS

qtCreatorFile = "Interfaz_Verificacion.ui" # Nombre del archivo aquí.
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)

class CNNTiempoRealView(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        self.setWindowTitle("CNN Tiempo Real")

        self.lbl_predict.setText("Esperando cámara ...")
        self.btn_action.clicked.connect(self.toggle_camera)

```



```
    self.is_running = False

    self.Worker = None

# Área de los Slots
def toggle_camera(self):
    if self.Worker is None or not self.Worker.isRunning():
        self.is_running = True
        self.Worker = CameraThread(model_path=MODELO, weights_path=PESOS)

        self.Worker.Prediction.connect(self.worker_conn)
        self.Worker.start()
        self.btn_action.setText("Detener cámara")
        self.btn_action.setStyleSheet(STYLES.get("btn_action", "") +
"background-color: rgb(255, 0, 0);")
    elif self.Worker is not None:
        self.Worker.stop()
        self.is_running = False
        self.btn_action.setText("Iniciar cámara")
        self.btn_action.setStyleSheet(STYLES.get("btn_action", "") +
"background-color: rgb(85, 85, 255);")

        self.lbl_predict.setText("Esperando cámara ...")
        self.lbl_predict.setStyleSheet(STYLES.get("lbl_predict", "") +
"background-color: rgb(122, 122, 122);")

    renderer = QSvgRenderer(":svgs/Archivos/Images/placeholder.svg")
    pixmap = QtGui.QPixmap(self.lbl_cam.width(), self.lbl_cam.height())
    pixmap.fill(QtCore.Qt.transparent)

    painter = QtGui.QPainter(pixmap)
    renderer.render(painter)
    painter.end()

    self.lbl_cam.setPixmap(pixmap)

def worker_conn(self, img, predict):
    if not self.is_running:
        return
    self.update_label_frame(img)
    self.update_label_predict(predict)

def update_label_frame(self, img):
    h, w, ch = img.shape
    bytesPerLine = ch * w
    convertToQtFormat = QtGui.QImage(img.data, w, h, bytesPerLine,
QtGui.QImage.Format_RGB888)
    pixmap = QtGui.QPixmap.fromImage(convertToQtFormat)
    self.lbl_cam.setPixmap(pixmap.scaled(self.lbl_cam.size(),
QtCore.Qt.KeepAspectRatio))

def update_label_predict(self, prediction):
    if prediction[0]:
        self.lbl_predict.setStyleSheet(STYLES.get("lbl_predict", "") +
"background-color: rgb(0, 170, 0);")
    else:
        self.lbl_predict.setStyleSheet(STYLES.get("lbl_predict", "") +
"background-color: rgb(255, 0, 0);")

    self.lbl_predict.setText(prediction[1])
```



Hilo que procesa la cámara, detecta rostro y predice

```
from PyQt5.QtCore import QThread, pyqtSignal
import numpy as np
import cv2
from keras.models import load_model
from keras.utils import load_img, img_to_array
import time

UMBRAL = 0.7

class CameraThread(QThread):
    Prediction = pyqtSignal(np.ndarray, tuple)

    def __init__(self, model_path, weights_path, width=300, height=300):
        super().__init__()
        self.running = True
        self.width = width
        self.height = height

        try:
            self.cnn = load_model(model_path)
            self.cnn.load_weights(weights_path)
        except Exception as e:
            print(f"Error al cargar el modelo/pesos: {e}")
            self.cnn = None

        self.face_classifier = cv2.CascadeClassifier(
            cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
        )

    def run(self):
        cam = cv2.VideoCapture(0)

        PREDICT_INTERVAL = 0.7 # s
        last_prediction_time = time.time()

        predict = (False, "X Esperando primera predicción...")

        while self.running:
            ret, frame = cam.read()

            if not ret:
                break

            frame = cv2.flip(frame, 1)
            rgbImage = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            if time.time() - last_prediction_time >= PREDICT_INTERVAL:
                last_prediction_time = time.time()
                predict = self.predict_face(frame)
                self.Prediction.emit(rgbImage, predict)

        cam.release()

    def stop(self):
        self.running = False
        self.wait() # Espera a que el hilo termine

    def predict_face(self, image):
        # Detección de la cara en la imagen
```



```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = self.face_classifier.detectMultiScale(gray_image, 1.1, 5,
minSize=(40, 40))

if len(faces) > 0:
    (x, y, w, h) = faces[0] # Tomamos la primera cara detectada
    face_frame = image[y:y + h, x:x + w]

    # Preprocesa la imagen de la cara
    face_frame = cv2.resize(face_frame, (self.height, self.width),
interpolation=cv2.INTER_CUBIC)
    face_frame = cv2.cvtColor(face_frame, cv2.COLOR_BGR2GRAY)
    face_frame = img_to_array(face_frame)
    face_frame = face_frame / 255.0
    face_frame = np.expand_dims(face_frame, axis=0)

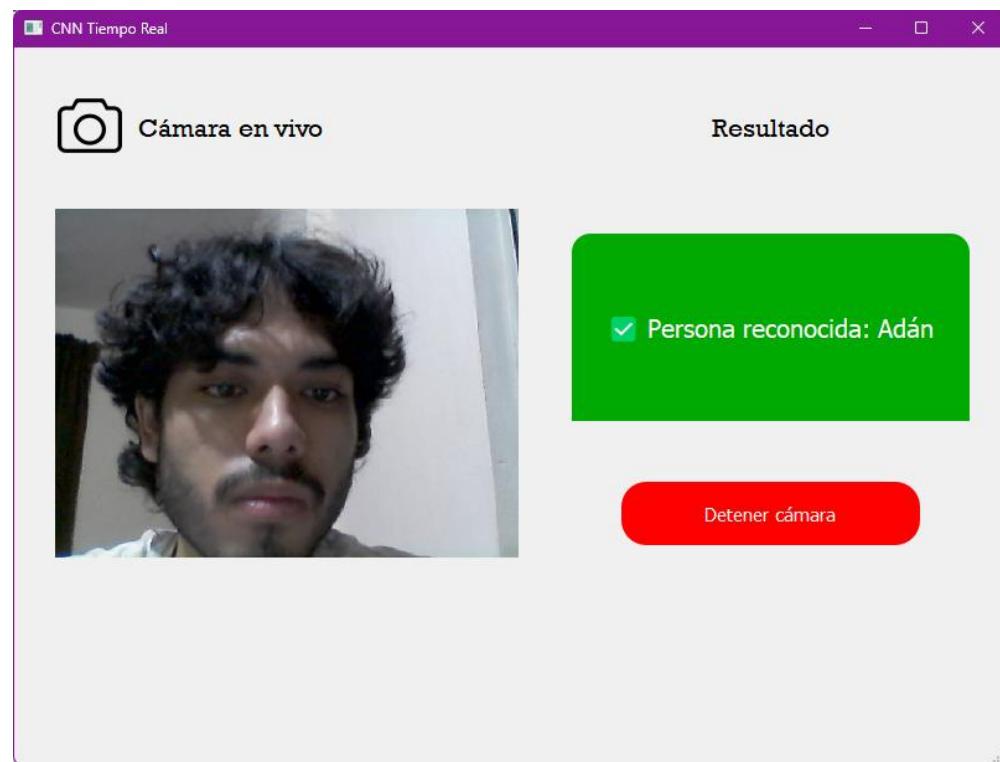
    # Realiza la predicción
    arreglo = self.cnn.predict(face_frame)

    probabilidad_maxima = np.max(arreglo[0])

    respuesta = np.argmax(arreglo[0])

    if probabilidad_maxima >= UMBRAL:
        resultado = "☑ Persona reconocida: "
        match respuesta:
            case 0:
                return (True, resultado + "Adán")
            case 1:
                return (True, resultado + "Poncho")
            case 2:
                return (True, resultado + "Pavel")
            case 3:
                return (True, resultado + "Cristobal")
            case _:
                return (False, "✗ Persona NO reconocida: idx inválido")
    else:
        return (False, "✗ Persona NO reconocida: Desconocido")
else: return (False, "✗ No se detecta rostro")
```

Ejecución



Modelos OpenCV

Descripción

Este programa mediante OpenCV, construye modelos entrenados capaz de reconocer rostros, utiliza algoritmos de reconocimiento facial (EigenFace, FisherFace, LBPH).

Código

```
import cv2
import os
import numpy as np
import entrenar_EigenFace as entrenador
# import entrenar_FisherFace as entrenador
# import entrenar_LBPHFace as entrenador

data_entrenamiento = "../../Archivos/Clases-individuos/F1-Entrenamiento"

alto, largo = 300, 300

def get_folders_name_from(from_location):
    list_dir = os.listdir(from_location)
    # folders = [archivo for archivo in listDir if os.path.splitext(archivo)[1]
    == ""]
    # the above is equals to ....
    folders = []
    for file in list_dir:
        temp = os.path.splitext(file)
        if temp[1] == "":
            folders.append(temp[0])
    folders.sort()
    # folders.remove('.DS_Store') #solo en mac
```

```

return folders

folders = get_folders_name_from(data_entrenamiento)
print('Nombre de las carpetas (clases): ', folders)

labels = []
images = []
label = 0

for folder in folders:
    full_dir = data_entrenamiento + '/' + folder
    print('Leyendo las imágenes')

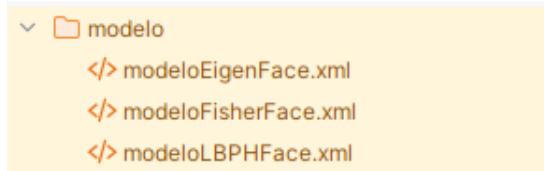
    for fileName in os.listdir(full_dir):
        print('Faces: ', folder + '/' + fileName)
        if fileName != ".DS_Store":
            labels.append(label)
            img = cv2.imread(full_dir + '/' + fileName, 0) # 0 = escala de grises
            img = cv2.resize(img, (alto, largo), interpolation=cv2.INTER_CUBIC)
            images.append(img)
            #cv2.imshow('img',img)
            #cv2.waitKey(10)
    label = label + 1

print('labels= ',labels)
for clase in range(label):
    print('Imagenes de clase ' + str(clase) +
      ':',np.count_nonzero(np.array(labels)==clase))

entrenador.train(images, labels)

```

Ejecución



MediaPipe 1

Descripción

Este programa utiliza OpenCV y MediaPipe para detectar una o dos manos mediante la cámara y contar cuantos dedos están levantados. Cuando solo se detecta una mano, muestra en pantalla el número correspondiente; si se detectan dos manos, suma los dedos de ambas y muestra el resultado.

Código

```

import cv2
import mediapipe as mp

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils

def contar_dedos(hand_landmarks, handedness):
    dedos = []

```



```
# Pulgar
if handedness == "Right":
    if hand_landmarks.landmark[4].x < hand_landmarks.landmark[3].x:
        dedos.append(1)
    else:
        dedos.append(0)
else: # Left
    if hand_landmarks.landmark[4].x > hand_landmarks.landmark[3].x:
        dedos.append(1)
    else:
        dedos.append(0)

# Índice, medio, anular, meñique (compara eje Y)
for tip in [8, 12, 16, 20]:
    if hand_landmarks.landmark[tip].y < hand_landmarks.landmark[tip - 2].y:
        dedos.append(1)
    else:
        dedos.append(0)

return sum(dedos)

cap = cv2.VideoCapture(0)

with mp_hands.Hands(max_num_hands=2, min_detection_confidence=0.75) as manos:
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        frame = cv2.flip(frame, 1)
        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = manos.process(rgb)

        if results.multi_hand_landmarks:
            numeros = []
            for hand_landmarks, hand_handedness in
zip(results.multi_hand_landmarks,
results.multi_handedness):
                # Saber si es mano izquierda o derecha
                label = hand_handedness.classification[0].label # "Left" o
                "Right"

                # Contar dedos con función que recibe también el label
                dedos = contar_dedos(hand_landmarks, label)
                numeros.append(dedos)

                mp_drawing.draw_landmarks(frame, hand_landmarks,
mp_hands.HAND_CONNECTIONS)

                # Si detecta dos manos → suma
                if len(numeros) == 2:
                    suma = numeros[0] + numeros[1]
                    cv2.putText(frame, f"numeros[0] + numeros[1] = {suma}",
(50, 100), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0),
3)

                # Si detecta una mano → solo muestra número
                elif len(numeros) == 1:
                    cv2.putText(frame, f"Numero: {numeros[0]}",
(50, 100), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0),
3)
```

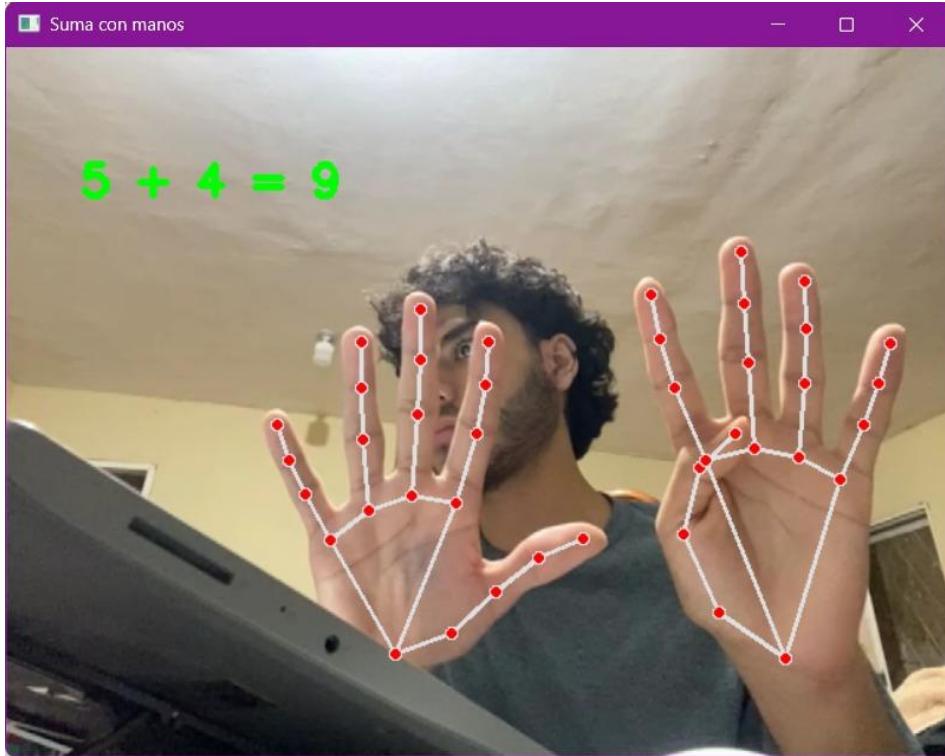
```

cv2.imshow("Suma con manos", frame)
if cv2.waitKey(1) & 0xFF == 27: # ESC para salir
    break

cap.release()
cv2.destroyAllWindows()

```

Ejecución



MediaPipe 5 gestos con direcciones

Descripción

Este programa utiliza OpenCV, MediaPipe y PyQt5 para detectar una mano mediante la cámara y de acuerdo con el gesto realizado muestra que dirección corresponde al gesto hecho.

Código

Manejo de la UI

```

from PyQt5 import uic, QtWidgets, QtGui, QtCore
from PyQt5.QtSvg import QSvgRenderer
from config import STYLES
from camera_thread import CameraThread

qtCreatorFile = "Ui_Gestos.ui" # Nombre del archivo UI
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)

class MediaPipe5(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)

```



```
self.setupUi(self)
self.setWindowTitle("MediaPipe 5 Gestos")

self.btn_action.clicked.connect(self.toggle_camera)

self.is_running = False

self.Worker = None

def toggle_camera(self):
    if self.Worker is None or not self.Worker.isRunning():
        self.is_running = True
        self.Worker = CameraThread()

        self.Worker.Statement.connect(self.worker_conn)
        self.Worker.start()
        self.btn_action.setText("Detener cámara")
        self.btn_action.setStyleSheet(STYLES.get("btn_action", "") +
"background-color: rgb(255, 0, 0);")
    elif self.Worker is not None:
        self.Worker.stop()
        self.is_running = False
        self.btn_action.setText("Iniciar cámara")
        self.btn_action.setStyleSheet(STYLES.get("btn_action", "") +
"background-color: rgb(85, 85, 255);")

    renderer = QSvgRenderer(":/svgs/Archivos/Images/placeholder.svg")
    pixmap = QtGui.QPixmap(self.lbl_cam.width(), self.lbl_cam.height())
    pixmap.fill(QtCore.Qt.transparent)

    painter = QtGui.QPainter(pixmap)
    renderer.render(painter)
    painter.end()

    self.lbl_cam.setPixmap(pixmap)

def worker_conn(self, img, moves):
    if not self.is_running:
        return
    self.update_label_frame(img)
    self.update_label_movement(moves)

def update_label_frame(self, img):
    h, w, ch = img.shape
    bytesPerLine = ch * w
    convertToQtFormat = QtGui.QImage(img.data, w, h, bytesPerLine,
QtGui.QImage.Format_RGB888)
    pixmap = QtGui.QPixmap.fromImage(convertToQtFormat)
    self.lbl_cam.setPixmap(pixmap.scaled(self.lbl_cam.size(),
QtCore.Qt.KeepAspectRatio))

def update_label_movement(self, moves):
    cadena = " + ".join(moves)
    self.lbl_moves.setText(cadena)
```

Hilo de procesamiento de cámara y reconocimiento de gestos

```
from PyQt5.QtCore import QThread, pyqtSignal
import cv2
import mediapipe as mp
import numpy as np
```



```
import math
from config import *

class CameraThread(QThread):
    Statement = pyqtSignal(np.ndarray, list)

    def __init__(self):
        super().__init__()
        self.running = True

    try:
        self.mp_hands = mp.solutions.hands
        self.hands = self.mp_hands.Hands(
            min_detection_confidence=0.5,
            min_tracking_confidence=0.5,
            max_num_hands=1
        )
        self.mp_draw = mp.solutions.drawing_utils
    except Exception as e:
        print(f'Error al cargar el modelo: {e}')
        self.mp_hands = None
        self.mp_draw = None

    def run(self):
        cam = cv2.VideoCapture(0)

        while self.running:
            ret, frame = cam.read()
            if not ret: return

            frame = cv2.flip(frame, 1)
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            height, width, _ = frame.shape
            states = list()

            # self.draw_lines(rgb_frame, height, width)

            if self.hands:
                results = self.hands.process(rgb_frame)
                if results.multi_hand_landmarks:
                    self.draw_landmarks(rgb_frame, results, height, width,
states)

                self.Statement.emit(rgb_frame, states)

            cam.release()

    def stop(self):
        self.running = False
        self.wait()

    def draw_lines(self, frame, height, width):
        x_lim_izq = int(width * MARGEN_IZQUIERDO)
        cv2.line(frame, (x_lim_izq, 0), (x_lim_izq, height), (255, 255, 0), 2)

        x_lim_der = int(width * MARGEN_DERECHO)
        cv2.line(frame, (x_lim_der, 0), (x_lim_der, height), (255, 255, 0), 2)

    def draw_landmarks(self, frame, results, height, width, states):
        for hand_landmarks in results.multi_hand_landmarks:
            self.mp_draw.draw_landmarks(frame, hand_landmarks,
```



```
self.mp_hands.HAND_CONNECTIONS)
    # self.move_detector(frame, hand_landmarks, height, width, states)
    self.analizar_gesto(hand_landmarks, states)

def move_detector(self, frame, hand_landmarks, height, width, states):
    x4 = int(hand_landmarks.landmark[4].x * width)
    y4 = int(hand_landmarks.landmark[4].y * height)

    x8 = int(hand_landmarks.landmark[8].x * width)
    y8 = int(hand_landmarks.landmark[8].y * height)
    distancia = math.hypot(x8 - x4, y8 - y4)

    # Punto medio del pellizco
    cx, cy = (x4 + x8) // 2, (y4 + y8) // 2

    pos_relativa = cx / width

    if distancia < UMBRAL_PELLIZCO:
        states.append(STATES.get("FORWARD"))

    if pos_relativa < MARGEN_IZQUIERDO:
        states.append(STATES.get("LEFT"))
    elif pos_relativa > MARGEN_DERECHO:
        states.append(STATES.get("RIGHT"))

def analizar_gesto(self, landmarks, states):
    dedos = []

    # --- 0. PULGAR ---
    if landmarks.landmark[4].x < landmarks.landmark[3].x:
        dedos.append(1)
    else:
        dedos.append(0)

    # --- 1. ÍNDICE ---
    # Si la punta (8) está más ARRIBA que la articulación (6) en el eje Y
    if landmarks.landmark[8].y < landmarks.landmark[6].y:
        dedos.append(1)
    else:
        dedos.append(0)

    # --- 2. MEDIO ---
    if landmarks.landmark[12].y < landmarks.landmark[10].y:
        dedos.append(1)
    else:
        dedos.append(0)

    # --- 3. ANULAR ---
    if landmarks.landmark[16].y < landmarks.landmark[14].y:
        dedos.append(1)
    else:
        dedos.append(0)

    # --- 4. MENIQUE ---
    if landmarks.landmark[20].y < landmarks.landmark[18].y:
        dedos.append(1)
    else:
        dedos.append(0)

    # Caso: Todos levantados
    if dedos == [1, 1, 1, 1, 1] or dedos == [0, 1, 1, 1, 1]:
```



```
states.append(STATES.get("STOP"))
return

# Caso: Pulgar
if dedos[0] == 1:
    states.append(STATES.get("FORWARD"))

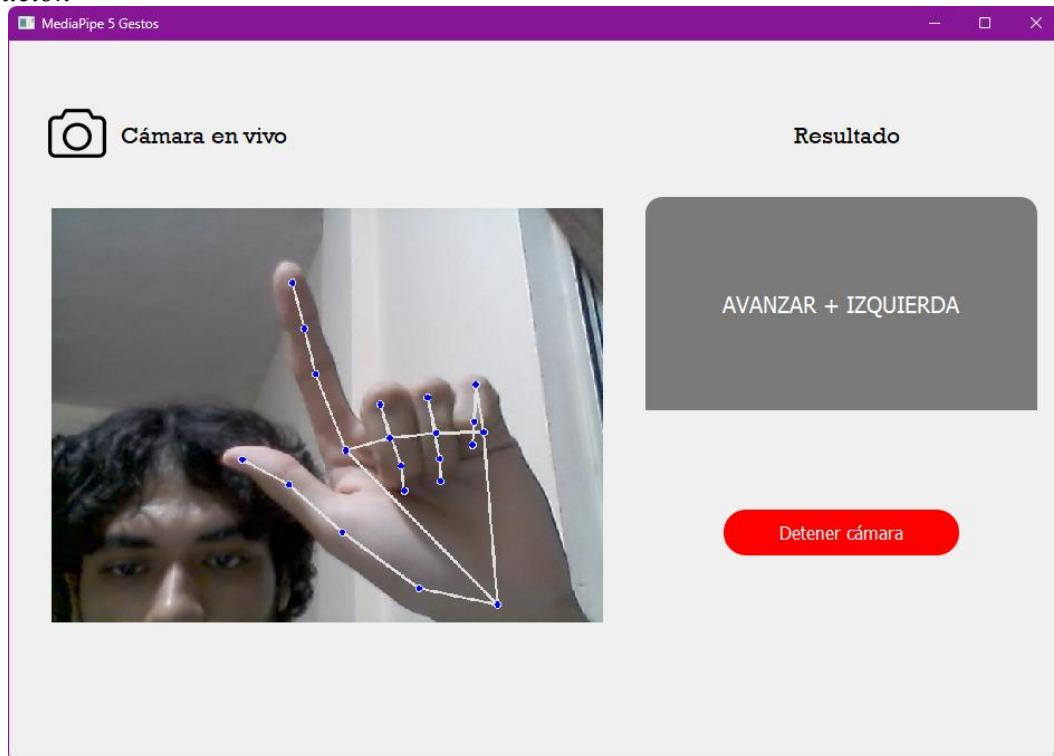
# Caso: Mano Cerrada
if dedos == [0, 0, 0, 0, 0] or dedos == [0, 1, 0, 0, 0] or dedos == [0,
0, 0, 0, 1]:
    states.append(STATES.get("BACKWARD"))

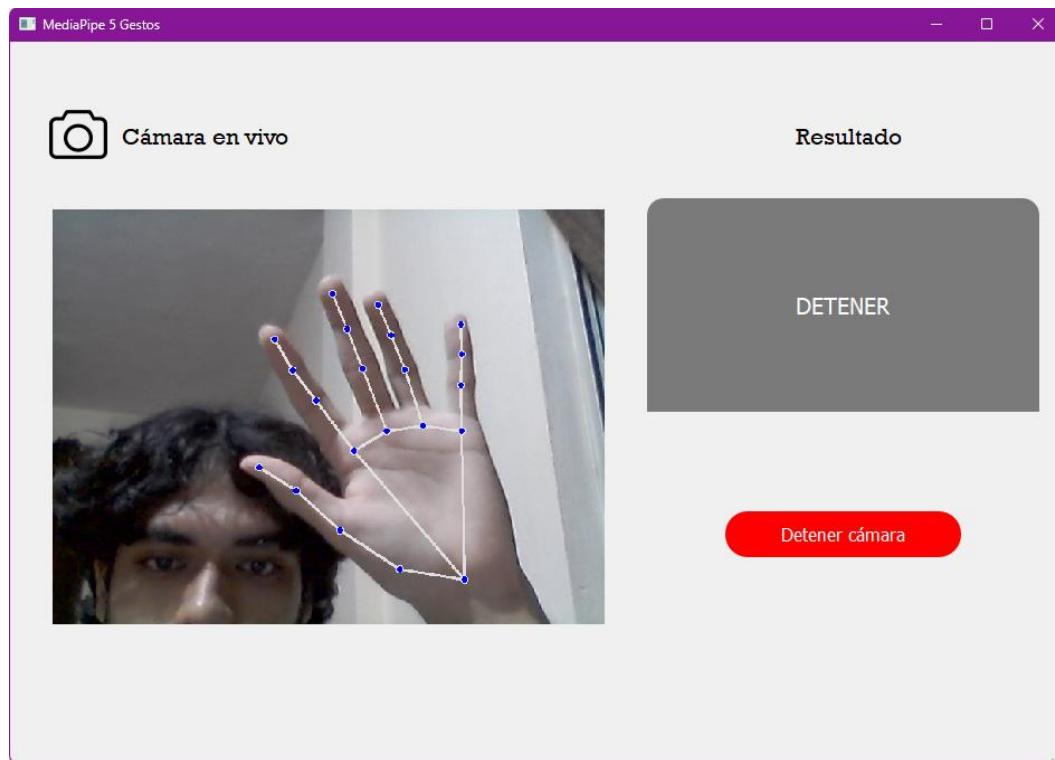
# Caso: Índice
if dedos[1] == 1:
    states.append(STATES.get("LEFT"))

# Caso: Meñique
if dedos[4] == 1:
    states.append(STATES.get("RIGHT"))

return
```

Ejecución





Carrito entorno virtual

Descripción

Este es un proyecto de simulación de vehículos 2D desarrollado en Unity, que se centra en una jugabilidad simple y una mecánica de movimiento de coches con características especiales. El script CarController implementa un sistema de control de vehículo basado en física, con aceleración y freno/reversa automáticos que se activan con alternancia en lugar de una pulsación constante. La característica distintiva es un "Giro Asistido" que aplica una pequeña aceleración automática al girar sin necesidad de acelerar manualmente, permitiendo movimientos más fluidos y cerrados, lo que sugiere un enfoque en la maniobrabilidad ágil para un juego de circuitos. También se muestra la velocidad del carrito que cambia si es hacia adelante o hacia atrás.

Código

CarController.cs

```
using UnityEngine; using UnityEngine.InputSystem; // Necesario para
InputAction.CallbackContext
public class CarController : MonoBehaviour
{ // Parámetros configurables [SerializeField] private float accelerationFactor
= 30.0f; [SerializeField] private float turnFactor = 3.5f; [SerializeField]
private float maxSpeed = 20f;
    // Parámetros para el giro asistido
    [Header("Asistencia de Giro")]
    [SerializeField] private float assistedAcceleration = 0.6f; // Aceleración
extra cuando solo se gira
    [SerializeField] private float assistedTurnMultiplier = 2.0f; // Factor
para hacer el giro más cerrado/derrape

    // Estado interno
    private float rotationAngle = 0f;
    private float velocityVsUp = 0f;
    private bool isAccelerating = false; // NUEVO: Estado de
aceleración automática
    private bool isBrakingOrReversing = false; // NUEVO: Estado de
Reversa/Freno automático

    // Componentes
    private Rigidbody2D carRb2D;
    private CarInputHandler inputHandler;

    private void Awake()
    {
        carRb2D = GetComponent<Rigidbody2D>();
        inputHandler = GetComponent<CarInputHandler>();
    }

    private void FixedUpdate()
    {
        ApplyEngineForce();
        KillOrthogonalVelocity();
        ApplySteering();
    }
```

```
// --- MÉTODOS DE TOGGLE PARA EL INPUT ---

/// <summary>
/// Alterna el estado de aceleración automática con un solo toque.
/// </summary>
public void ToggleAcceleration(InputAction.CallbackContext context)
{
    if (context.started)
    {
        isAccelerating = !isAccelerating;
        // Si activamos la aceleración, desactivamos el freno para evitar
conflicto        if (isAccelerating)
        {
            isBrakingOrReversing = false;
        }
        Debug.Log($"Aceleración automática: {isAccelerating}");
    }
}

/// <summary>
/// Alterna el estado de reversa/freno automático con un solo toque.
/// </summary>
public void ToggleBrake(InputAction.CallbackContext context)
{
    if (context.started)
    {
        isBrakingOrReversing = !isBrakingOrReversing;
        // Si activamos el freno, desactivamos la aceleración para evitar
conflicto        if (isBrakingOrReversing)
        {
            isAccelerating = false;
        }
        Debug.Log($"Freno/Reversa automática: {isBrakingOrReversing}");
    }
}

// ----

/// <summary>
/// Aplica la fuerza del motor, usando los estados de toggle.
/// </summary>
private void ApplyEngineForce()
{
    velocityVsUp = Vector2.Dot(transform.up, carRb2D.linearVelocity);

    // Limitar velocidad si está acelerando (automáticamente)
    if (velocityVsUp > maxSpeed && isAccelerating) return;
    if (velocityVsUp < -maxSpeed * 0.5f && isBrakingOrReversing) return;
    if (carRb2D.linearVelocity.sqrMagnitude > maxSpeed * maxSpeed &&
isAccelerating) return;

    // 🚗 CÁLCULO DE FUERZA FINAL
    float throttleInput = 0f;
```



```
if (isBrakingOrReversing)
{
    throttleInput = -1f; // Aplicar reversa/freno total
}
else if (isAccelerating)
{
    throttleInput = 1f; // Aplicar aceleración total
}

float finalThrottle = throttleInput;

// Bandera para saber si estamos usando la aceleración asistida
bool isAssistedTurning = false;

// Giro asistido: Solo si NO hay aceleración o freno activos Y se está
girando
if (!isAccelerating && !isBrakingOrReversing &&
Mathf.Approximately(inputHandler.move.x, 0f))
{
    if (velocityVsUp < maxSpeed * 0.75f)
    {
        finalThrottle = assistedAcceleration;
        isAssistedTurning = true;
    }
}

// Aplicación del Drag (Frenado pasivo): Solo si no hay fuerza activa
ni asistencia
if (Mathf.Approximately(finalThrottle, 0f) && !isAssistedTurning)
{
    // Aplicar fricción pasiva para detener el coche
    carRb2D.linearDamping = Mathf.Lerp(carRb2D.linearDamping, 3.0f,
Time.fixedDeltaTime * 3);
}
else
{
    // Eliminar fricción pasiva si hay movimiento activo
    carRb2D.linearDamping = 0f;
}

// Vector de fuerza del motor
Vector2 engineForceVector = transform.up * finalThrottle *
accelerationFactor;

// Aplicar fuerza
carRb2D.AddForce(engineForceVector, ForceMode2D.Force);
}

/// <summary>
/// Aplica el giro del vehículo.
/// </summary>
private void ApplySteering()
{
    float speed = carRb2D.linearVelocity.magnitude;
    float turnMultiplier = speed < 0.1f ? 0.3f : Mathf.Clamp01(speed / 8f);
```



```
// Giro más cerrado/asistido (solo si NO hay aceleración manual y SOLO
se está girando)
// Nota: Como 'inputHandler.move.y' ahora siempre es 0, solo
comprobamos el giro.
if (!Mathf.Approximately(inputHandler.move.x, 0f))
{
    // Si no hay aceleración o freno activos (el giro asistido es más
cerrado)
    if (!isAccelerating && !isBrakingOrReversing)
    {
        turnMultiplier *= assistedTurnMultiplier;
    }
}

// Determinar dirección según movimiento (para invertir la dirección
del giro en reversa)
float direction = Vector2.Dot(carRb2D.linearVelocity, transform.up) >=
0 ? 1f : -1f;

// Ajustar ángulo de rotación
rotationAngle -= inputHandler.move.x * turnFactor * turnMultiplier *
direction;

carRb2D.MoveRotation(rotationAngle);
}

/// <summary>
/// Elimina la velocidad ortogonal para evitar derrapes irreales.
/// </summary>
private void KillOrthogonalVelocity()
{
    Vector2 forwardVelocity = transform.up *
Vector2.Dot(carRb2D.linearVelocity, transform.up);
    carRb2D.linearVelocity = forwardVelocity;
}
```

```
CarInputHandler.cs
using UnityEngine;
using UnityEngine.InputSystem;

public class CarInputHandler : MonoBehaviour
{
    private CarController carController; // Referencia al componente
CarController
    private InputAction m_MoveAction;
    private Vector2 _move;

    private void Awake()
    {
        carController = GetComponent<CarController>();
        if (carController == null)
        {
```



```
        Debug.LogError("CarInputHandler requiere un CarController en el
mismo GameObject.");
        enabled = false;
        return;
    }

    m_MoveAction = InputSystem.actions.FindAction("Player/Move");
    m_MoveAction.Enable();

    // 🚗 CONEXIÓN CLAVE: Toggle de Aceleración (valor Y positivo)
    m_MoveAction.started += context => {
        // Solo llamar a ToggleAcceleration si la entrada vertical es
positiva (Acelerar: W o Flecha Arriba)
        if (context.ReadValue<Vector2>().y > 0.5f)
        {
            carController.ToggleAcceleration(context);
        }
    };

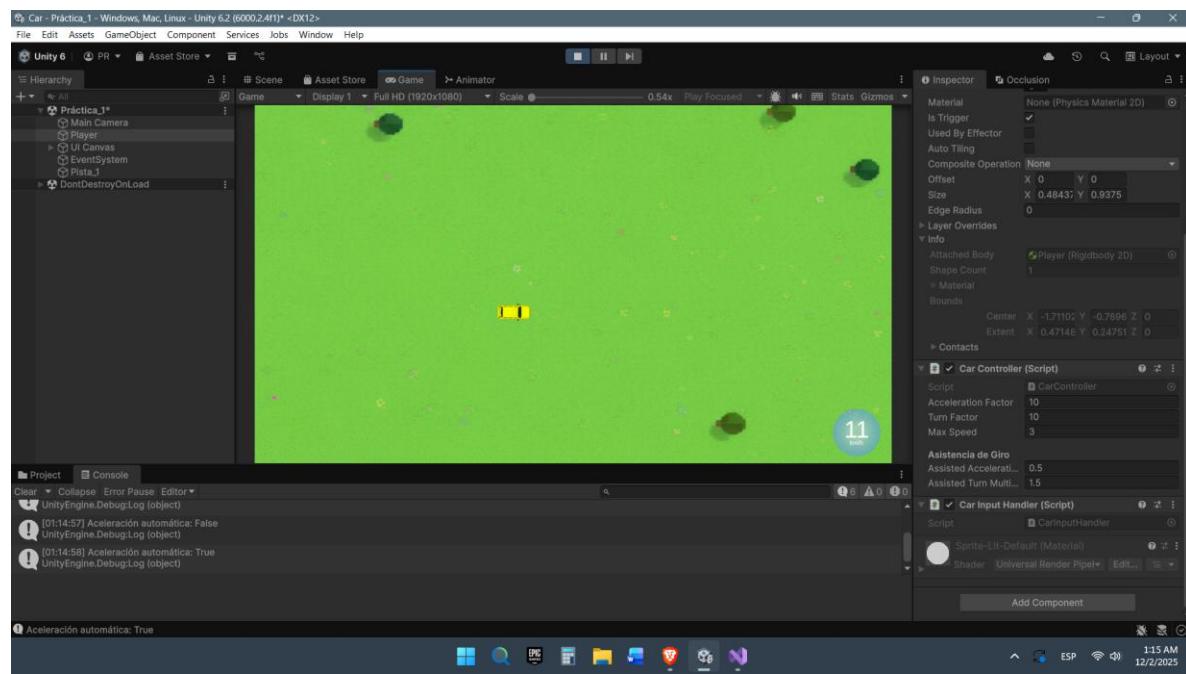
    // 🚗 CONEXIÓN CLAVE: Toggle de Reversa/Freno (valor Y negativo)
    m_MoveAction.started += context => {
        // Solo llamar a ToggleBrake si la entrada vertical es negativa
(Frenar: S o Flecha Abajo)
        if (context.ReadValue<Vector2>().y < -0.5f)
        {
            carController.ToggleBrake(context);
        }
    };
}

public Vector2 move { get { return _move; } }

private void Update()
{
    // 1. Leemos el valor del Vector2 completo
    Vector2 rawMove = m_MoveAction.ReadValue<Vector2>();

    // 2. Filtramos la entrada:
    // El eje Y (aceleración/freno) ahora se maneja por estados booleanos
en CarController.
    // Aquí solo necesitamos el valor X (giro) y el Y debe ser 0.
    _move = new Vector2(rawMove.x, 0f);
}
```

Ejecución

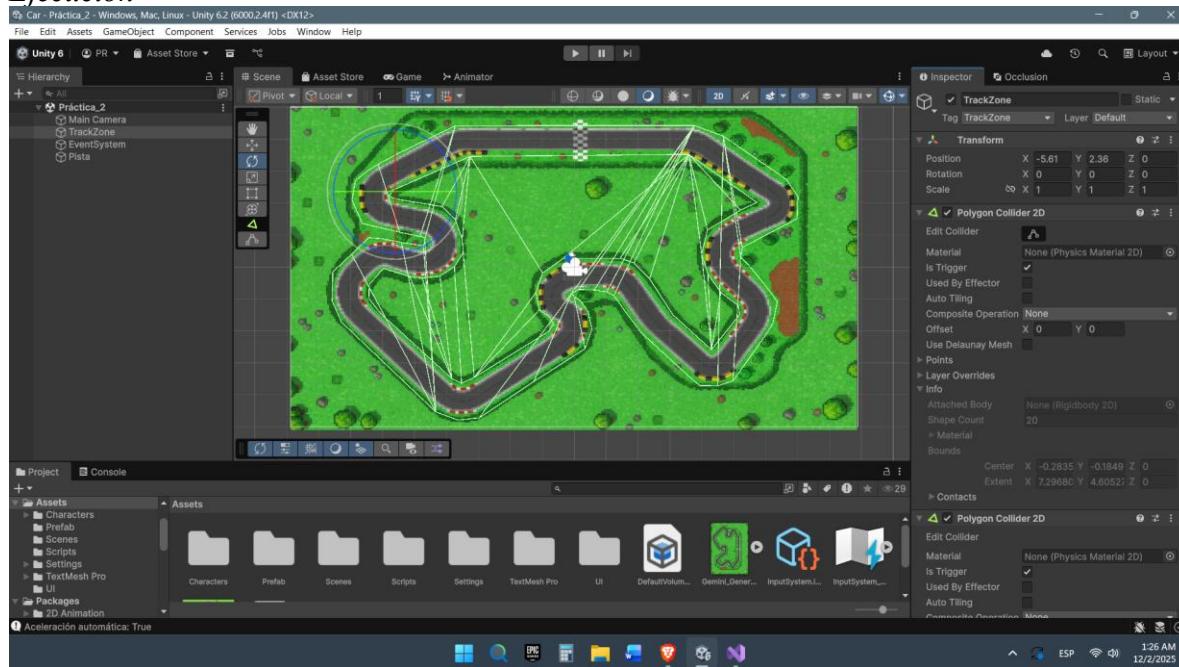


Circuito realizado

Descripción

Este ejercicio se enfocó exclusivamente en el diseño y la implementación del mapa del circuito de carreras dentro del entorno 2D de Unity. Este circuito fue creado sobre un sprite de mapa visible, utilizando el componente Polygon Collider 2D para definir con precisión la geometría de la pista. Este collider no solo da forma al trazado de la carrera, sino que también es crucial para establecer la zona de la pista ("TrackZone"), permitiendo al sistema de juego determinar si el vehículo está dentro o fuera de los límites, lo cual es fundamental para futuras mecánicas como la detección de vueltas o la aplicación de penalizaciones por salirse del camino.

Ejecución



Prueba de circuito con manejo de teclado

Descripción

Este ejercicio se centró en integrar la lógica de detección de límites y el estado de Fin del Juego dentro del simulador de carreras. Esto se logró al configurar el collider del circuito (etiquetado como TrackZone) como un Is Trigger y utilizando el método OnTriggerEnter2D en el script CarroController para detectar cuando el vehículo abandona la pista. Una innovación clave fue la implementación de una Corutina para el Tiempo de Gracia al Inicio (startGracePeriod), que deshabilita temporalmente los colliders de los límites durante 0.5 segundos, previniendo un Game Over inmediato al empezar la carrera sobre la línea de salida. Al detectar el límite, el coche se detiene inmediatamente (GameOver()), marcando la finalización de la partida.

Código

CarroController.cs

```
using UnityEngine; using UnityEngine.InputSystem;
using System.Collections; // NECESARIO para usar Corutinas (IEnumerator)
public class CarroController : MonoBehaviour // CLASE PRINCIPAL { // Parámetros
configurables [SerializeField] private float accelerationFactor = 30.0f;
[SerializeField] private float turnFactor = 3.5f; [SerializeField] private
float maxSpeed = 20f; [SerializeField] private string trackZoneTag =
"TrackZone"; // Tag usado para detectar el límite
{// NUEVA VARIABLE: Tiempo para ignorar los límites al inicio
[SerializeField] private float startGracePeriod = 0.5f;

// Parámetros para el giro asistido
[Header("Asistencia de Giro")]
[SerializeField] private float assistedAcceleration = 0.6f;
[SerializeField] private float assistedTurnMultiplier = 2.0f;

// Estado interno
private float rotationAngle = 0f;
private float velocityVsUp = 0f;
private bool isAccelerating = false;
private bool isBrakingOrReversing = false;
public bool IsCarReversing => isBrakingOrReversing;

// Componentes
private Rigidbody2D carRb2D;
private CarroInputHandler inputHandler;

private void Awake()
{
    carRb2D = GetComponent<Rigidbody2D>(); inputHandler =
GetComponent<CarroInputHandler>();
    rotationAngle = transform.eulerAngles.z;
}

private void Start()
{
```

```

// Detiene el coche, resetea el estado y garantiza que no haya
velocidad inicial
    carRb2D.linearVelocity = Vector2.zero;
    carRb2D.angularVelocity = 0f;
    isAccelerating = false;

    // INICIA LA CORUTINA: Desactiva los límites por 0.5 segundos
    StartCoroutine(ToggleBoundaryColliders(false, startGracePeriod));
}

private void FixedUpdate()
{
    ApplyEngineForce();
    KillOrthogonalVelocity();
    ApplySteering();
}

// --- MÉTODOS DE TOGGLE Y LÓGICA DE MOVIMIENTO (Sin cambios) ---

public void ToggleAcceleration(InputAction.CallbackContext context)
{
    if (context.started)
    {
        isAccelerating = !isAccelerating;
        if (isAccelerating)
        {
            isBrakingOrReversing = false;
        }
        Debug.Log($"Aceleración automática: {isAccelerating}");
    }
}

public void ToggleBrake(InputAction.CallbackContext context)
{
    if (context.started)
    {
        isBrakingOrReversing = !isBrakingOrReversing;
        if (isBrakingOrReversing)
        {
            isAccelerating = false;
        }
        Debug.Log($"Freno/Reversa automática: {isBrakingOrReversing}");
    }
}

private void ApplyEngineForce()
{
    velocityVsUp = Vector2.Dot(transform.up, carRb2D.linearVelocity);
    if (velocityVsUp > maxSpeed && isAccelerating) return;
    if (velocityVsUp < -maxSpeed * 0.5f && isBrakingOrReversing) return;
    if (carRb2D.linearVelocity.sqrMagnitude > maxSpeed * maxSpeed &&
isAccelerating) return;

    float throttleInput = 0f;
    if (isBrakingOrReversing) { throttleInput = -1f; }
    else if (isAccelerating) { throttleInput = 1f; }
}

```



```
float finalThrottle = throttleInput;
bool isAssistedTurning = false;

if (!isAccelerating && !isBrakingOrReversing &&
!Mathf.Approximately(inputHandler.move.x, 0f))
{
    if (velocityVsUp < maxSpeed * 0.75f)
    {
        finalThrottle = assistedAcceleration;
        isAssistedTurning = true;
    }
}

if (Mathf.Approximately(finalThrottle, 0f) && !isAssistedTurning)
{
    carRb2D.linearDamping = Mathf.Lerp(carRb2D.linearDamping, 3.0f,
Time.fixedDeltaTime * 3);
}
else
{
    carRb2D.linearDamping = 0f;
}

Vector2 engineForceVector = transform.up * finalThrottle *
accelerationFactor;
carRb2D.AddForce(engineForceVector, ForceMode2D.Force);
}

private void ApplySteering()
{
    float speed = carRb2D.linearVelocity.magnitude;
    float turnMultiplier = speed < 0.1f ? 0.3f : Mathf.Clamp01(speed / 8f);

    if (!Mathf.Approximately(inputHandler.move.x, 0f))
    {
        if (!isAccelerating && !isBrakingOrReversing)
        {
            turnMultiplier *= assistedTurnMultiplier;
        }
    }

    float direction = Vector2.Dot(carRb2D.linearVelocity, transform.up) >=
0 ? 1f : -1f;
    rotationAngle -= inputHandler.move.x * turnFactor * turnMultiplier * direction;
    carRb2D.MoveRotation(rotationAngle);
}

private void KillOrthogonalVelocity()
{
    Vector2 forwardVelocity = transform.up *
Vector2.Dot(carRb2D.linearVelocity, transform.up);
    carRb2D.linearVelocity = forwardVelocity;
}
```



```
// -----  
  
// CORUTINA: Desactiva los colliders de límite para evitar la detección al  
inicio  
private IEnumerator ToggleBoundaryColliders(bool enable, float delay)  
{  
    // 1. Encuentra los límites  
    GameObject[] boundaryZones =  
    GameObject.FindGameObjectsWithTag(trackZoneTag);  
  
    // 2. Deshabilita los colliders inmediatamente  
    foreach (GameObject zone in boundaryZones)  
    {  
        Collider2D col = zone.GetComponent<Collider2D>();  
        if (col != null)  
        {  
            col.enabled = false;  
        }  
    }  
  
    yield return new WaitForSeconds(delay); // 3. Espera el tiempo de  
gracia  
  
    // 4. Vuelve a habilitar los colliders después del tiempo de gracia  
    foreach (GameObject zone in boundaryZones)  
    {  
        Collider2D col = zone.GetComponent<Collider2D>();  
        if (col != null)  
        {  
            col.enabled = true;  
        }  
    }  
}  
  
// --- LÓGICA DE JUEGO (SENSOR/TRIGGER) ---  
  
/// <summary>  
/// Detecta la entrada al sensor de límite. Requiere que el límite tenga Is  
Trigger MARCADO.  
/// </summary>  
private void OnTriggerEnter2D(Collider2D other)  
{  
    // Verifica si entramos en el área del límite (TrackZone)  
    if (other.CompareTag(trackZoneTag))  
    {  
        Debug.Log("¡Límite detectado por sensor! Juego terminado.");  
        GameOver();  
    }  
}  
  
// --- Lógica de Fin de Juego ---  
  
private void GameOver()  
{  
    carRb2D.linearVelocity = Vector2.zero;  
    carRb2D.angularVelocity = 0f;
```



```
        enabled = false;
    }

    public void StopCarOnWin()
    {
        carRb2D.linearVelocity = Vector2.zero;
        carRb2D.angularVelocity = 0f;
        enabled = false;
    }

}
```

CarroInputHandler.cs

```
using UnityEngine;
using UnityEngine.InputSystem;

public class CarroInputHandler : MonoBehaviour // <-- CORRECCIÓN CLAVE: Hereda
de MonoBehaviour
{
    private CarroController carController; // <-- Referencia al Carro
Controller
    private InputAction m_MoveAction;
    private Vector2 _move;

    private void Awake()
    {
        // Busca el componente de control
        carController = GetComponent<CarroController>();

        if (carController == null)
        {
            // Mensaje de error personalizado, ahora preciso
            Debug.LogError("El script CarroInputHandler requiere un
CarroController en el mismo GameObject.");
            enabled = false;
            return;
        }

        m_MoveAction = InputSystem.actions.FindAction("Player/Move");
        m_MoveAction.Enable();

        // Conexiones de Input (Toggles)
        m_MoveAction.started += context => {
            if (context.ReadValue<Vector2>().y > 0.5f)
            {
                carController.ToggleAcceleration(context);
            }
        };

        m_MoveAction.started += context => {
            if (context.ReadValue<Vector2>().y < -0.5f)
            {
                carController.ToggleBrake(context);
            }
        };
    }
}
```

```

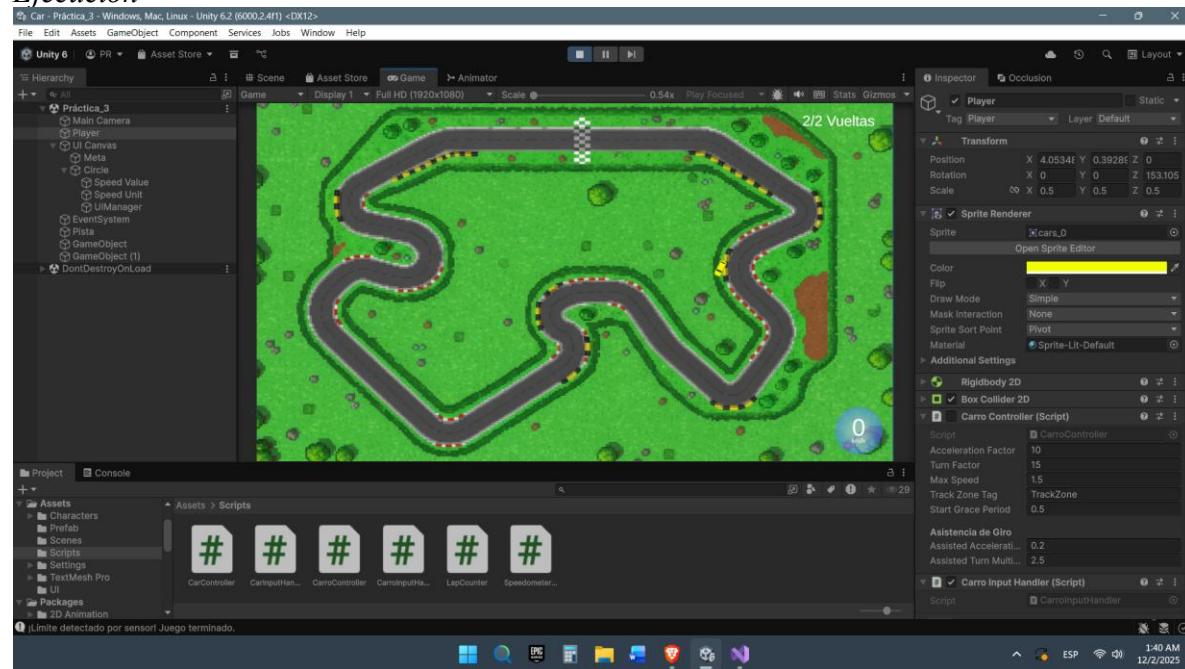
    };
}

public Vector2 move { get { return _move; } }

private void Update()
{
    Vector2 rawMove = m_MoveAction.ReadValue<Vector2>();
    _move = new Vector2(rawMove.x, 0f); // Solo propagamos el giro (X)
}
}

```

Ejecución



Laberinto procedural

Descripción

Este programa genera un laberinto procedural en unity. Se desarrolló un algoritmo para crear caminos aleatorios dentro de una cuadrícula. A partir de esta lógica, el sistema construye visualmente el laberinto usando prefabs de piso y pared.

Código

```
using System.Collections.Generic; using UnityEngine;
public class GeneradorLaberinto : MonoBehaviour
{
    public int filas = 10;
    public int columnas = 10;
    public GameObject paredPrefab;
    public GameObject pisoPrefab;
    public Material materialInicio;
    public Material materialFin;

    private Celda[,] celdas;
    private Transform laberintoHolder;

    void Start()
    {
        celdas = new Celda[columnas, filas];
        for (int x = 0; x < columnas; x++)
        {
            for (int z = 0; z < filas; z++)
            {
                celdas[x, z] = new Celda();
            }
        }
        GenerarLaberintoLogico();
        ConstruirLaberintoVisual();
    }

    private void OnDrawGizmos()
    {
        if (celdas == null) return;

        for (int x = 0; x < columnas; x++)
        {
            for (int z = 0; z < filas; z++)
            {

                Vector3 centro = new Vector3(x, 0, z);
                Celda celda = celdas[x, z];
                if (celda.esPiso)
                {
                    Gizmos.color = Color.green;
                    Gizmos.DrawWireCube(centro, new Vector3(0.5f, 0.1f, 0.5f));
                    if (celda.visitada && !celda.esPiso)
                    {
                        Gizmos.color = Color.yellow;
                        Gizmos.DrawSphere(centro, 0.1f);
                    }
                }
            }
        }
    }
}
```

```

        else
    {
        Gizmos.color = new Color(0.3f, 0, 0, 0.5f);
        Gizmos.DrawCube(centro, new Vector3(0.3f, 0.3f, 0.3f));
    }
    Gizmos.color = Color.red;
    if (celda.paredNorte)
    {
        Vector3 pos = new Vector3(x, 0.5f, z + 0.5f);
        Gizmos.DrawWireCube(pos, new Vector3(1, 1, 0.1f));
    }
    if (z == 0 && celda.paredSur)
    {
        Vector3 pos = new Vector3(x, 0.5f, z - 0.5f);
        Gizmos.DrawWireCube(pos, new Vector3(1, 1, 0.1f));
    }
    if (celda.paredEste)
    {
        Vector3 pos = new Vector3(x + 0.5f, 0.5f, z);
        Gizmos.DrawWireCube(pos, new Vector3(0.1f, 1, 1));
    }
    if (x == 0 && celda.paredOeste)
    {
        Vector3 pos = new Vector3(x - 0.5f, 0.5f, z);
        Gizmos.DrawWireCube(pos, new Vector3(0.1f, 1, 1));
    }
}
}

void GenerarLaberintoLogico()
{
    Stack<Vector2Int> pila = new Stack<Vector2Int>();
    Vector2Int posActual = new Vector2Int(0, 0);
    celdas[posActual.x, posActual.y].visitada = true;
    pila.Push(posActual);
    while (pila.Count > 0)
    {

        posActual = pila.Peek();
        List<Vector2Int> vecinos = ObtenerVecinosNoVisitados(posActual);
        if (vecinos.Count > 0)
        {

            Vector2Int vecinoElegido = vecinos[Random.Range(0,
            vecinos.Count)];

            TumbarPared(posActual, vecinoElegido);
            celdas[vecinoElegido.x, vecinoElegido.y].visitada = true;
            pila.Push(vecinoElegido);
        }
        else
        {
            pila.Pop();
        }
    }
}

```



```
        celdas[columnas - 1, filas - 1].paredEste = false;  
    }  
  
    List<Vector2Int> ObtenerVecinosNoVisitados(Vector2Int pos)  
{  
    List<Vector2Int> listaVecinos = new List<Vector2Int>();  
    int x = pos.x;  
    int z = pos.y;  
    if (z + 1 < filas && !celdas[x, z + 1].visitada)  
        listaVecinos.Add(new Vector2Int(x, z + 1));  
    if (z - 1 >= 0 && !celdas[x, z - 1].visitada)  
        listaVecinos.Add(new Vector2Int(x, z - 1));  
    if (x + 1 < columnas && !celdas[x + 1, z].visitada)  
        listaVecinos.Add(new Vector2Int(x + 1, z));  
  
    if (x - 1 >= 0 && !celdas[x - 1, z].visitada)  
        listaVecinos.Add(new Vector2Int(x - 1, z));  
  
    return listaVecinos;  
}  
  
void TumbarPared(Vector2Int actual, Vector2Int vecina)  
{  
    celdas[actual.x, actual.y].esPiso = true;  
    celdas[vecina.x, vecina.y].esPiso = true;  
    if (vecina.x > actual.x)  
    {  
        celdas[actual.x, actual.y].paredEste = false;  
        celdas[vecina.x, vecina.y].paredOeste = false;  
    }  
    else if (vecina.x < actual.x)  
    {  
        celdas[actual.x, actual.y].paredOeste = false;  
        celdas[vecina.x, vecina.y].paredEste = false;  
    }  
    else if (vecina.y > actual.y)  
    {  
        celdas[actual.x, actual.y].paredNorte = false;  
        celdas[vecina.x, vecina.y].paredSur = false;  
    }  
    else if (vecina.y < actual.y)  
    {  
        celdas[actual.x, actual.y].paredSur = false;  
        celdas[vecina.x, vecina.y].paredNorte = false;  
    }  
}  
void ConstruirLaberintoVisual()  
{  
    if (laberintoHolder != null) Destroy(laberintoHolder.gameObject);  
    laberintoHolder = new GameObject("Laberinto").transform;  
    laberintoHolder.parent = this.transform;  
  
    // AJUSTES VISUALES  
    float grosorPared = 0.1f;  
    float alturaPared = 1.0f;  
    float largoPared = 1.0f;
```

```

        for (int x = 0; x < columnas; x++)
    {
        for (int z = 0; z < filas; z++)
        {
            Celda celda = celdas[x, z];
            if (celda.esPiso)
            {
                Vector3 posPiso = new Vector3(x, -0.1f, z);
                GameObject piso = Instantiate(pisoPrefab, posPiso,
                    Quaternion.identity, laberintoHolder);
                piso.transform.localScale = new Vector3(1, 0.2f, 1);
                piso.name = $"Piso_{x}_{z}";
                MeshRenderer pisoRenderer =
                    piso.GetComponent<MeshRenderer>();

                if (pisoRenderer != null)
                {
                    // Inicio
                    if (x == 0 && z == 0)
                    {
                        if (materialInicio != null)
                            pisoRenderer.material = materialInicio;
                        else
                            pisoRenderer.material.color = Color.red;

                        piso.name = "Piso_INICIO";
                    }
                    // Fin
                    else if (x == columnas - 1 && z == filas - 1)
                    {
                        if (materialFin != null)
                            pisoRenderer.material = materialFin;
                        else
                            pisoRenderer.material.color = Color.green;
                        piso.name = "Piso_FIN";
                        piso.tag = "Meta";
                        BoxCollider trigger =
                            piso.AddComponent<BoxCollider>();
                        trigger.isTrigger = true;
                        trigger.size = new Vector3(1, 10, 1);
                        trigger.center = new Vector3(0, 5, 0);
                    }
                }
            }
        }
        // Pared Norte (Arriba)
        if (celda.paredNorte)
        {
            Vector3 pos = new Vector3(x, 0.5f, z + 0.5f);
            GameObject pared = Instantiate(paredPrefab, pos,
                Quaternion.identity, laberintoHolder);
            // ESCALA: Ancho X=1, Alto Y=1, Grosor Z=0.1
            pared.transform.localScale = new Vector3(largoPared,
                alturaPared, grosorPared);
            pared.name = $"ParedNorte_{x}_{z}";
        }
    }
}

```



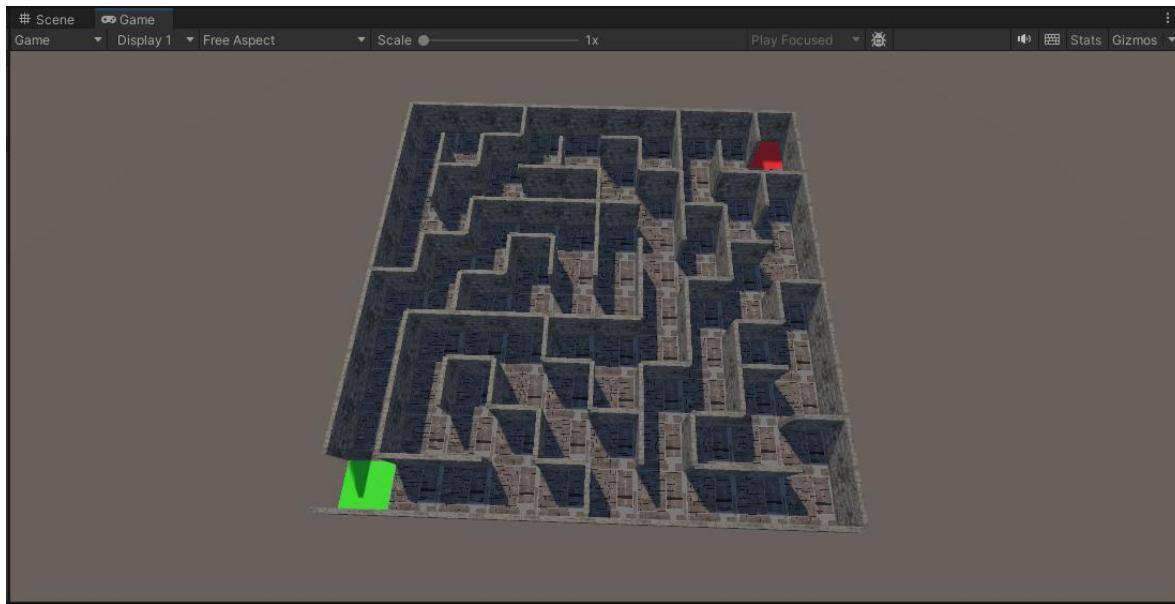
```
// Pared Sur
if (z == 0 && celda.paredSur)
{
    Vector3 pos = new Vector3(x, 0.5f, z - 0.5f);
    GameObject pared = Instantiate(paredPrefab, pos,
Quaternion.identity, laberintoHolder);
    pared.transform.localScale = new Vector3(largoPared,
alturaPared, grosorPared);
    pared.name = $"ParedSur_{x}_{z}";
}
// Pared Este
if (celda.paredEste)
{
    Vector3 pos = new Vector3(x + 0.5f, 0.5f, z);
    GameObject pared = Instantiate(paredPrefab, pos,
Quaternion.identity, laberintoHolder);

    pared.transform.localScale = new Vector3(grosorPared,
alturaPared, largoPared);
    pared.name = $"ParedEste_{x}_{z}";
}
// Pared Oeste
if (x == 0 && celda.paredOeste)
{
    Vector3 pos = new Vector3(x - 0.5f, 0.5f, z);
    GameObject pared = Instantiate(paredPrefab, pos,
Quaternion.identity, laberintoHolder);
    pared.transform.localScale = new Vector3(grosorPared,
alturaPared, largoPared);
    pared.name = $"ParedOeste_{x}_{z}";
}
}
}
}

}

[System.Serializable] public class Celda { public bool visitada = false; public
bool paredNorte = true; public bool paredSur = true; public bool paredEste =
true; public bool paredOeste = true; public bool esPiso = false; }
```

Ejecución



Laberinto completado de forma manual

Descripción

Este programa genera un laberinto procedural en unity. Se desarrollo un algoritmo para crear caminos aleatorios dentro de una cuadrícula con inicio y fin y se agrego un jugador para poder jugarlo controlado por teclado para completarlo.

Código

Generador del laberinto:

```
using System.Collections.Generic;
using UnityEngine;

public class GeneradorLaberinto : MonoBehaviour
{
    public int filas = 10;
    public int columnas = 10;
    public GameObject paredPrefab;
    public GameObject pisoPrefab;
    public Material materialInicio;
    public Material materialFin;

    private Celda[,] celdas;
    private Transform laberintoHolder;

    void Start()
    {
        celdas = new Celda[columnas, filas];
        for (int x = 0; x < columnas; x++)
        {
            for (int z = 0; z < filas; z++)
            {
                if (x == 0 && z == 0)
                    celdas[x, z] = new Celda(x, z, materialInicio);
                else
                    celdas[x, z] = new Celda(x, z, materialPiso);
            }
        }
    }
}
```



```
{  
    celdas[x, z] = new Celda();  
}  
}  
GenerarLaberintoLogico();  
ConstruirLaberintoVisual();  
}  
private void OnDrawGizmos()  
{  
    if (celdas == null) return;  
  
    for (int x = 0; x < columnas; x++)  
    {  
        for (int z = 0; z < filas; z++)  
        {  
  
            Vector3 centro = new Vector3(x, 0, z);  
            Celda celda = celdas[x, z];  
            if (celda.esPiso)  
            {  
                Gizmos.color = Color.green;  
                Gizmos.DrawWireCube(centro, new Vector3(0.5f, 0.1f, 0.5f));  
                if (celda.visitada && !celda.esPiso)  
                {  
                    Gizmos.color = Color.yellow;  
                    Gizmos.DrawSphere(centro, 0.1f);  
                }  
            }  
            else  
            {  
                Gizmos.color = new Color(0.3f, 0, 0, 0.5f);  
                Gizmos.DrawCube(centro, new Vector3(0.3f, 0.3f, 0.3f));  
            }  
            Gizmos.color = Color.red;  
            if (celda.paredNorte)  
            {  
                Vector3 pos = new Vector3(x, 0.5f, z + 0.5f);  
                Gizmos.DrawWireCube(pos, new Vector3(1, 1, 0.1f));  
            }  
            if (z == 0 && celda.paredSur)  
            {  
                Vector3 pos = new Vector3(x, 0.5f, z - 0.5f);  
                Gizmos.DrawWireCube(pos, new Vector3(1, 1, 0.1f));  
            }  
            if (celda.paredEste)  
            {  
                Vector3 pos = new Vector3(x + 0.5f, 0.5f, z);  
                Gizmos.DrawWireCube(pos, new Vector3(0.1f, 1, 1));  
            }  
            if (x == 0 && celda.paredOeste)  
            {  
                Vector3 pos = new Vector3(x - 0.5f, 0.5f, z);  
                Gizmos.DrawWireCube(pos, new Vector3(0.1f, 1, 1));  
            }  
        }  
    }  
}
```



```
}

void GenerarLaberintoLogico()
{

    Stack<Vector2Int> pila = new Stack<Vector2Int>();
    Vector2Int posActual = new Vector2Int(0, 0);
    celdas[posActual.x, posActual.y].visitada = true;
    pila.Push(posActual);
    while (pila.Count > 0)
    {

        posActual = pila.Peek();
        List<Vector2Int> vecinos = ObtenerVecinosNoVisitados(posActual);
        if (vecinos.Count > 0)
        {

            Vector2Int vecinoElegido = vecinos[Random.Range(0,
            vecinos.Count)];
            TumbarPared(posActual, vecinoElegido);
            celdas[vecinoElegido.x, vecinoElegido.y].visitada = true;
            pila.Push(vecinoElegido);
        }
        else
        {
            pila.Pop();
        }
    }
    celdas[columnas - 1, filas - 1].paredEste = false;
}

List<Vector2Int> ObtenerVecinosNoVisitados(Vector2Int pos)
{
    List<Vector2Int> listaVecinos = new List<Vector2Int>();
    int x = pos.x;
    int z = pos.y;
    if (z + 1 < filas && !celdas[x, z + 1].visitada)
        listaVecinos.Add(new Vector2Int(x, z + 1));
    if (z - 1 >= 0 && !celdas[x, z - 1].visitada)
        listaVecinos.Add(new Vector2Int(x, z - 1));
    if (x + 1 < columnas && !celdas[x + 1, z].visitada)
        listaVecinos.Add(new Vector2Int(x + 1, z));

    if (x - 1 >= 0 && !celdas[x - 1, z].visitada)
        listaVecinos.Add(new Vector2Int(x - 1, z));

    return listaVecinos;
}

void TumbarPared(Vector2Int actual, Vector2Int vecina)
{
    celdas[actual.x, actual.y].esPiso = true;
    celdas[vecina.x, vecina.y].esPiso = true;
    if (vecina.x > actual.x)
    {
        celdas[actual.x, actual.y].paredEste = false;
```



```
        celdas[vecina.x, vecina.y].paredOeste = false;
    }
    else if (vecina.x < actual.x)
    {
        celdas[actual.x, actual.y].paredOeste = false;
        celdas[vecina.x, vecina.y].paredEste = false;
    }
    else if (vecina.y > actual.y)
    {
        celdas[actual.x, actual.y].paredNorte = false;
        celdas[vecina.x, vecina.y].paredSur = false;
    }
    else if (vecina.y < actual.y)
    {
        celdas[actual.x, actual.y].paredSur = false;
        celdas[vecina.x, vecina.y].paredNorte = false;
    }
}
void ConstruirLaberintoVisual()
{
    if (laberintoHolder != null) Destroy(laberintoHolder.gameObject);
    laberintoHolder = new GameObject("Laberinto").transform;
    laberintoHolder.parent = this.transform;

    // AJUSTES VISUALES
    float grosorPared = 0.1f;
    float alturaPared = 1.0f;
    float largoPared = 1.0f;

    for (int x = 0; x < columnas; x++)
    {
        for (int z = 0; z < filas; z++)
        {
            Celda celda = celdas[x, z];
            if (celda.esPiso)
            {
                Vector3 posPiso = new Vector3(x, -0.1f, z);
                GameObject piso = Instantiate(pisoPrefab, posPiso,
Quaternion.identity, laberintoHolder);
                piso.transform.localScale = new Vector3(1, 0.2f, 1);
                piso.name = $"Piso_{x}_{z}";
                MeshRenderer pisoRenderer =
piso.GetComponent<MeshRenderer>();

                if (pisoRenderer != null)
                {
                    // Inicio
                    if (x == 0 && z == 0)
                    {
                        if (materialInicio != null)
                            pisoRenderer.material = materialInicio;
                        else
                            pisoRenderer.material.color = Color.red;
                    }
                    piso.name = "Piso_INICIO";
                }
            }
        }
    }
}
```

```

        // Fin
    else if (x == columnas - 1 && z == filas - 1)
    {
        if (materialFin != null)
            pisoRenderer.material = materialFin;
        else
            pisoRenderer.material.color = Color.green;
        piso.name = "Piso_FIN";
        piso.tag = "Meta";
        BoxCollider trigger =
    piso.AddComponent<BoxCollider>();
        trigger.isTrigger = true;
        trigger.size = new Vector3(1, 10, 1);
        trigger.center = new Vector3(0, 5, 0);
    }
}
// Pared Norte (Arriba)
if (celda.paredNorte)
{
    Vector3 pos = new Vector3(x, 0.5f, z + 0.5f);
    GameObject pared = Instantiate(paredPrefab, pos,
Quaternion.identity, laberintoHolder);
    // ESCALA: Ancho X=1, Alto Y=1, Grosor Z=0.1
    pared.transform.localScale = new Vector3(largoPared,
alturaPared, grosorPared);
    pared.name = $"ParedNorte_{x}_{z}";
}
// Pared Sur
if (z == 0 && celda.paredSur)
{
    Vector3 pos = new Vector3(x, 0.5f, z - 0.5f);
    GameObject pared = Instantiate(paredPrefab, pos,
Quaternion.identity, laberintoHolder);
    pared.transform.localScale = new Vector3(largoPared,
alturaPared, grosorPared);
    pared.name = $"ParedSur_{x}_{z}";
}
// Pared Este
if (celda.paredEste)
{
    Vector3 pos = new Vector3(x + 0.5f, 0.5f, z);
    GameObject pared = Instantiate(paredPrefab, pos,
Quaternion.identity, laberintoHolder);

    pared.transform.localScale = new Vector3(grosorPared,
alturaPared, largoPared);
    pared.name = $"ParedEste_{x}_{z}";
}
// Pared Oeste
if (x == 0 && celda.paredOeste)
{
    Vector3 pos = new Vector3(x - 0.5f, 0.5f, z);
    GameObject pared = Instantiate(paredPrefab, pos,
Quaternion.identity, laberintoHolder);
}

```



```
        pared.transform.localScale = new Vector3(grosorPared,
alturaPared, largoPared);
        pared.name = $"ParedOeste_{x}_{z}";
    }
}
}

[System.Serializable]
public class Celda
{
    public bool visitada = false;
    public bool paredNorte = true;
    public bool paredSur = true;
    public bool paredEste = true;
    public bool paredOeste = true;
    public bool esPiso = false;
}

Jugador:
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Jugador : MonoBehaviour
{
    public float velocidadMovimiento = 3.0f;
    public float velocidadRotacion = 150.0f;

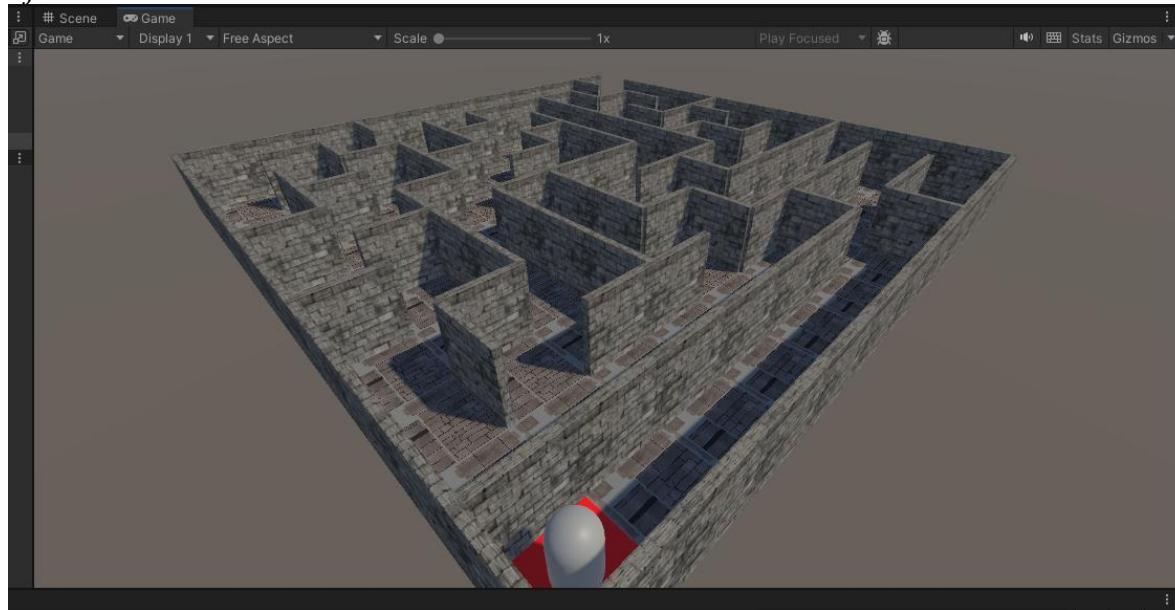
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        float moverZ = Input.GetAxis("Vertical") * velocidadMovimiento *
Time.deltaTime;
        float rotarY = Input.GetAxis("Horizontal") * velocidadRotacion *
Time.deltaTime;
        transform.Translate(0, 0, moverZ);
        transform.Rotate(0, rotarY, 0);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Meta"))
        {
            Debug.Log(";GANASTE! Juego Pausado.");
            Time.timeScale = 0;
        }
    }
}
```

Ejecución



Laberinto completado de forma automática

Descripción

Este programa genera un laberinto procedural en unity. Se desarrollo un algoritmo para crear caminos aleatorios dentro de una cuadrícula con inicio y fin y se agrego un jugador con un algoritmo para jugarlo de forma automática y completar el laberinto.

Código

Jugador con algoritmo:

```
using UnityEngine;
using System.Collections;

public class SolverManoDerecha : MonoBehaviour
{
    public float tiempoEspera = 0.5f;
    public bool haTerminado = false;

    void Start()
    {
        StartCoroutine(ResolverLaberinto());
    }

    IEnumerator ResolverLaberinto()
    {
        yield return new WaitForSeconds(1f);

        while (!haTerminado)
        {
            if (!HayPared(Vector3.right))
            {
                yield return Girar(90);
                yield return Moverse();
            }
        }
    }
}
```



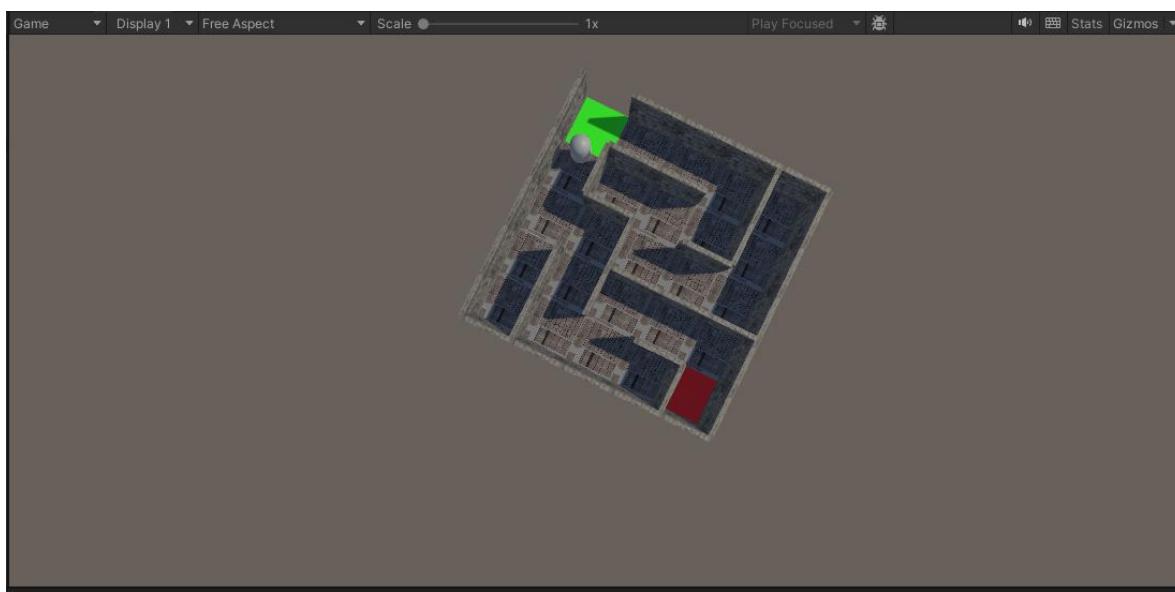
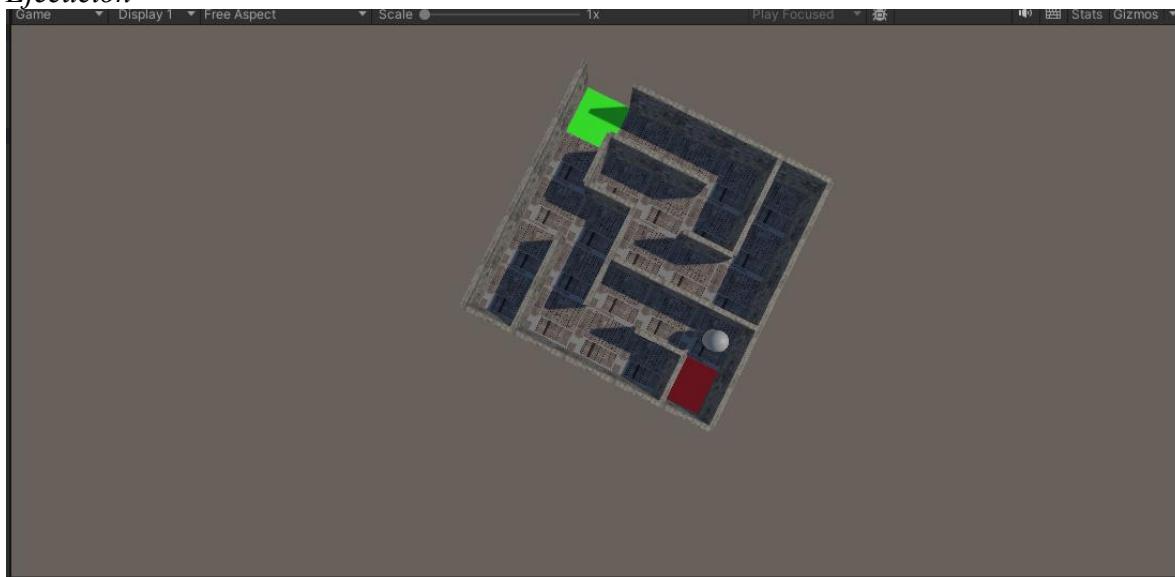
```
        }
        else if (!HayPared(Vector3.forward))
        {
            yield return Moverse();
        }
        else
        {
            yield return Girar(-90);
        }
        yield return new WaitForSeconds(tiempoEspera);
    }
}
bool HayPared(Vector3 direccionLocal)
{
    Vector3 direccionMundo = transform.TransformDirection(direccionLocal);
    if (Physics.Raycast(transform.position, direccionMundo, out RaycastHit
hit, 0.8f))
    {
        if (hit.collider.isTrigger) return false;

        return true;
    }
    return false;
}
IEnumerator Moverse()
{
    Vector3 posicionInicial = transform.position;
    Vector3 destino = transform.position + transform.forward;
    float tiempoPasado = 0;
    float duracion = 0.3f;
    while (tiempoPasado < duracion)
    {
        transform.position = Vector3.Lerp(posicionInicial, destino,
tiempoPasado / duracion);
        tiempoPasado += Time.deltaTime;
        yield return null;
    }
    transform.position = destino;
}
IEnumerator Girar(float angulo)
{
    Quaternion rotacionInicial = transform.rotation;
    Quaternion rotacionFinal = rotacionInicial * Quaternion.Euler(0,
angulo, 0);

    float tiempoPasado = 0;
    float duracion = 0.3f;

    while (tiempoPasado < duracion)
    {
        transform.rotation = Quaternion.Lerp(rotacionInicial,
rotacionFinal, tiempoPasado / duracion);
        tiempoPasado += Time.deltaTime;
        yield return null;
    }
    transform.rotation = rotacionFinal;
```

```
        }
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Meta"))
        {
            Debug.Log("¡EL BOT HA GANADO!");
            haTerminado = true;
            StopAllCoroutines(); // Detener el cerebro
        }
    }
}
```

Ejecución



Karel letra

Descripción

Este ejercicio consistió en el uso del entorno educativo Karel el Robot, se desarrolló un algoritmo que controló el desplazamiento y las acciones de Karel para formar la letra B con zumbadores, utilizándolos como puntos que forman la letra dentro del mundo.

Código

```
class program {
    void turnright(){
        iterate(3){
            turnleft();
        }
    }

    void direccion(dir){
        if(iszero(dir)){
            turnright();
            move();
            turnright();
        }else{
            turnleft();
            if(frontIsClear){
                move();
            }
            turnleft();
        }
    }

    void go(){
        while(frontIsClear){
            move();
        }
    }

    void draw(n){
        iterate(n){
            putbeeper();
            move();
        }
    }

    void blanks(n){
        iterate(n){
            move();
        }
    }

    void columnup(){
        direccion(0);
        blanks(1);
        draw(1);
        blanks(3);
        draw(1);
        blanks(2);
        draw(1);
    }
}
```



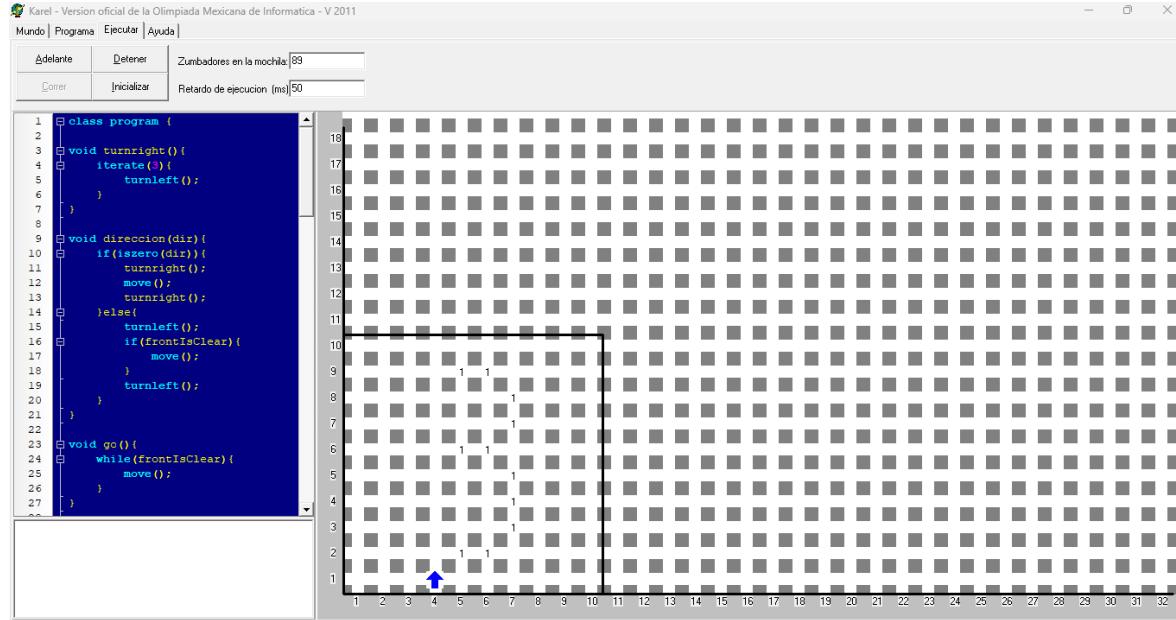
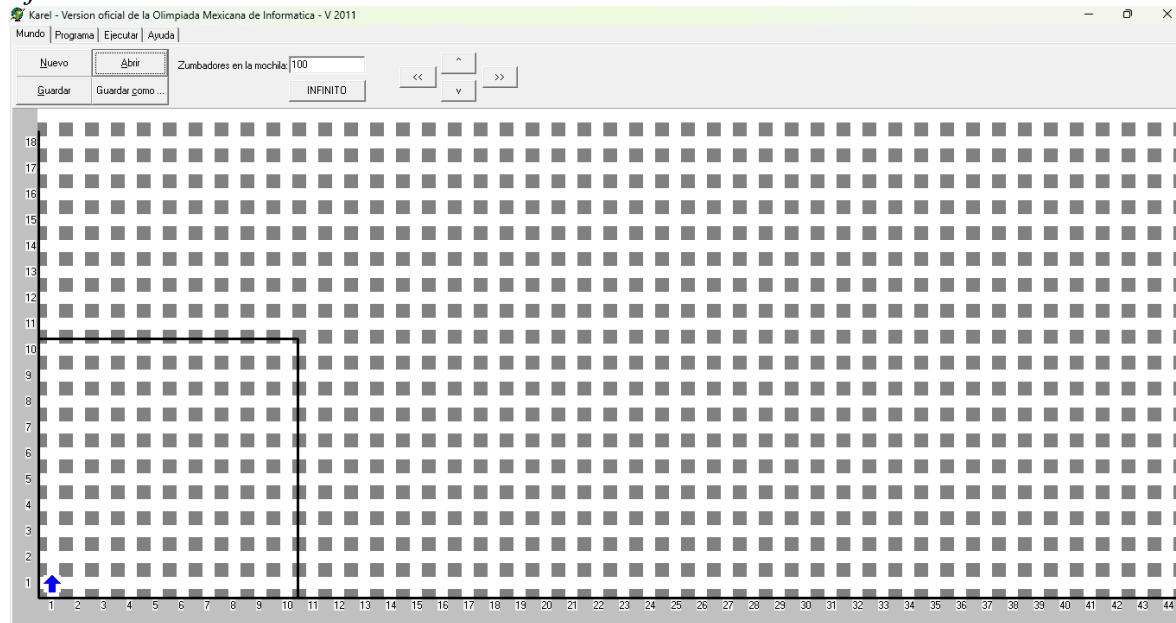
```
program() {  
    iterate(5){  
        go();  
        direccion(0);  
        go();  
        direccion(1);  
    }  
  
    go();  
    direccion(1);  
    go();  
    direccion(0);  
  
    go();  
  
    direccion(1);  
    blanks(2);  
    draw(2);  
    blanks(1);  
    draw(3);  
  
    go();  
  
    columnup();  
  
    go();  
  
    direccion(1);  
    blanks(1);  
    draw(1);  
    blanks(2);  
    draw(1);  
    blanks(3);  
    draw(1);  
  
    go();  
  
    columnup();  
  
    go();  
  
    direccion(1);  
    blanks(1);  
    draw(8);  
  
    go();  
  
    direccion(0);  
  
    go();  
    direccion(1);  
    go();  
    turnleft();  
    turnleft();  
    ;
```

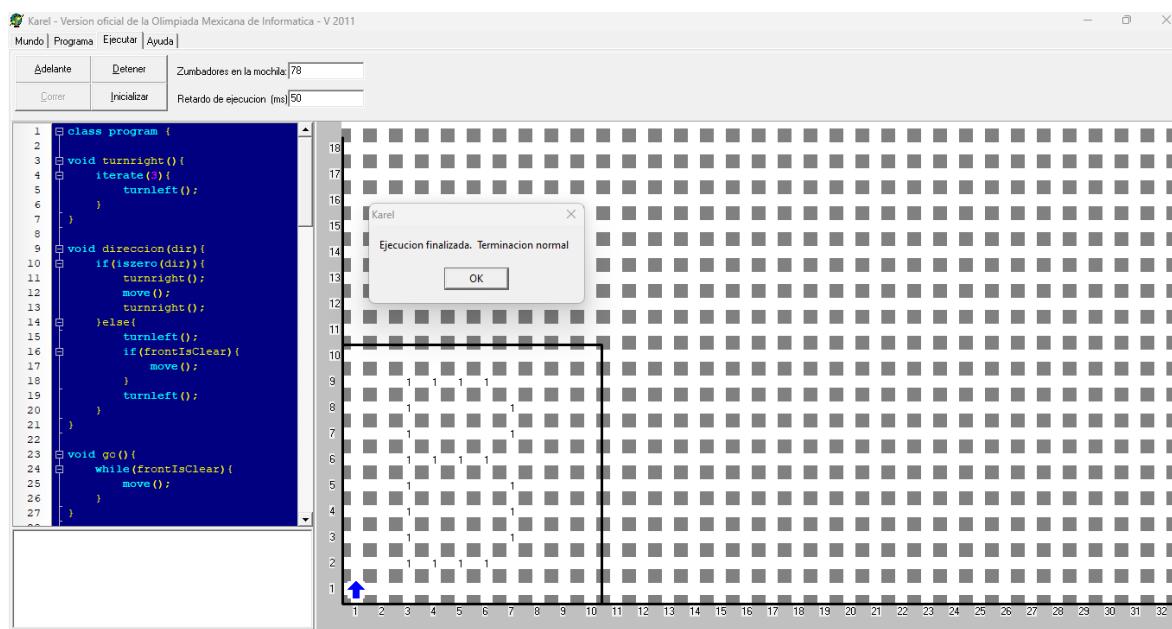
```

        turnoff();
    }
}

```

Ejecución





Karel come 3 Beepers

Descripción

Este ejercicio se desarrolló en el entorno Karel el Robot, donde se implementó el algoritmo para que el robot recorriera el mundo creado identificando y analizando los montones de zumbadores que se encuentran en el camino recorrido, con el objetivo de solo recoger aquellos montones que contengan exactamente 3 zumbadores.

Código

```
class program {
    void turnright(){
        iterate(3){
            turnleft();
        }
    }

    void retur(dir){
        if(iszero(dir)){
            turnright();
            move();
            turnright();
        }else{
            turnleft();
            if(frontIsClear){
                move();
            }
            turnleft();
        }
    }

    void go(){
        while (frontIsClear){
            if(nextToABeeper){

```



```
pickbeeper();
if(nextToABeeper){
    pickbeeper();
    if(nextToABeeper){
        pickbeeper();

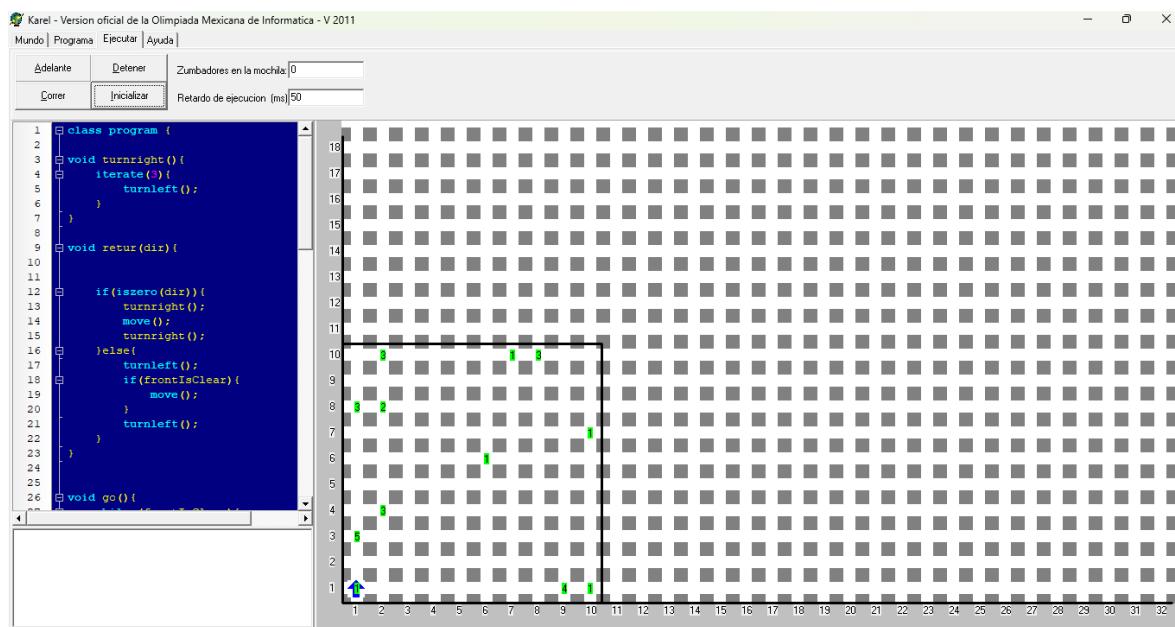
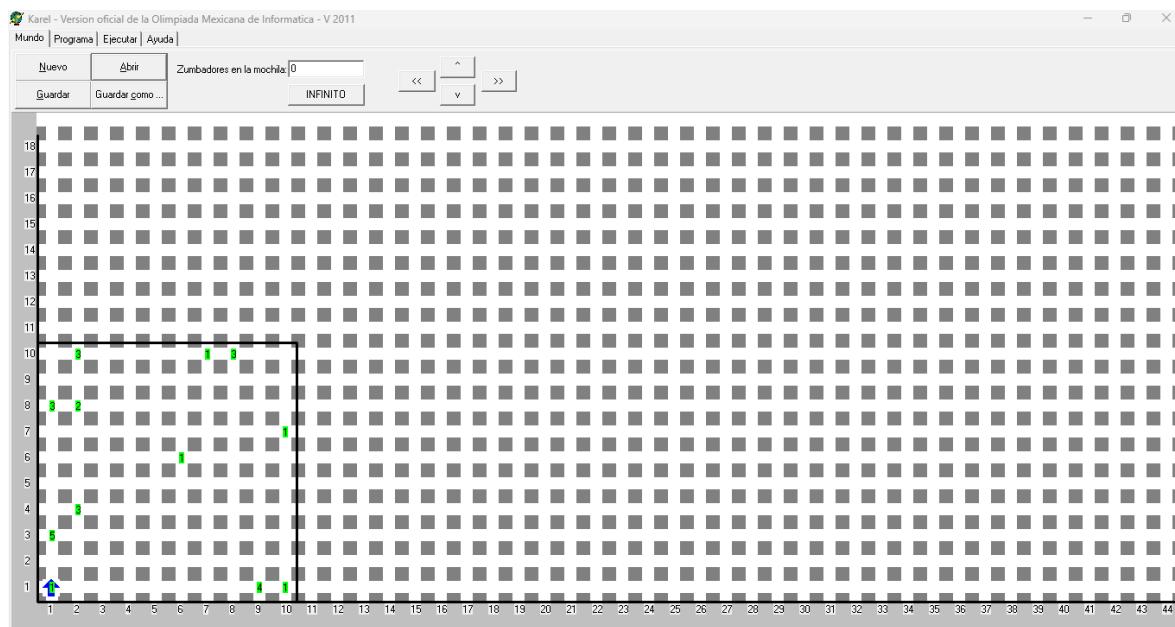
        if(nextToABeeper){
            iterate(3){
                putbeeper();
            }
        }
        }else{
            iterate(2){
                putbeeper();
            }
        }
    }else{
        putbeeper();
    }
}
move();
}

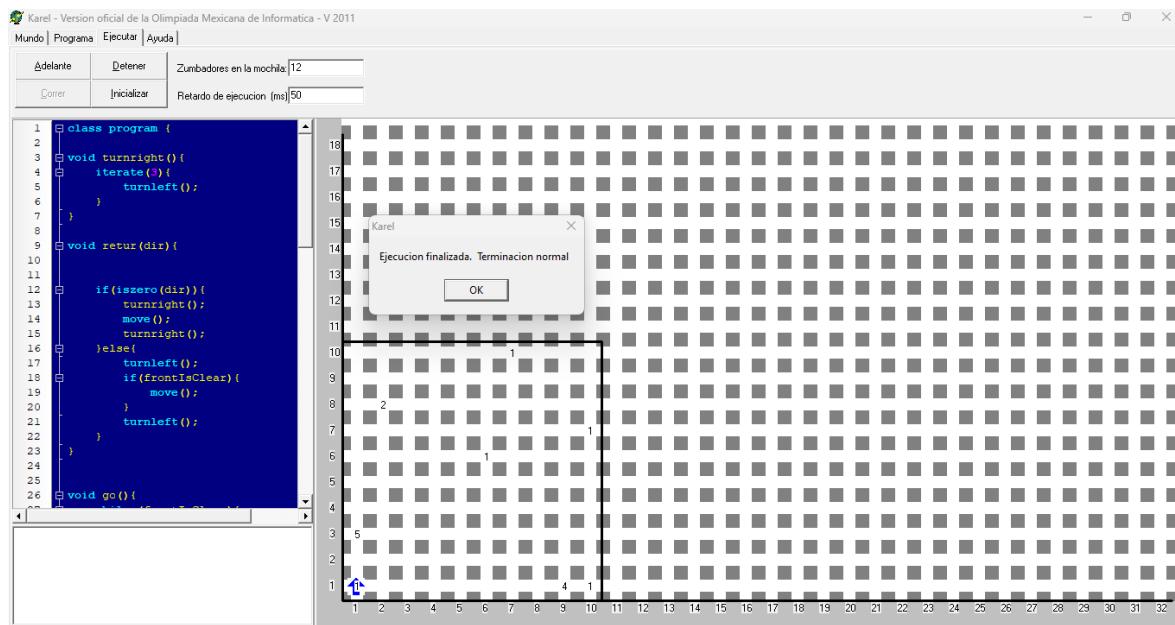
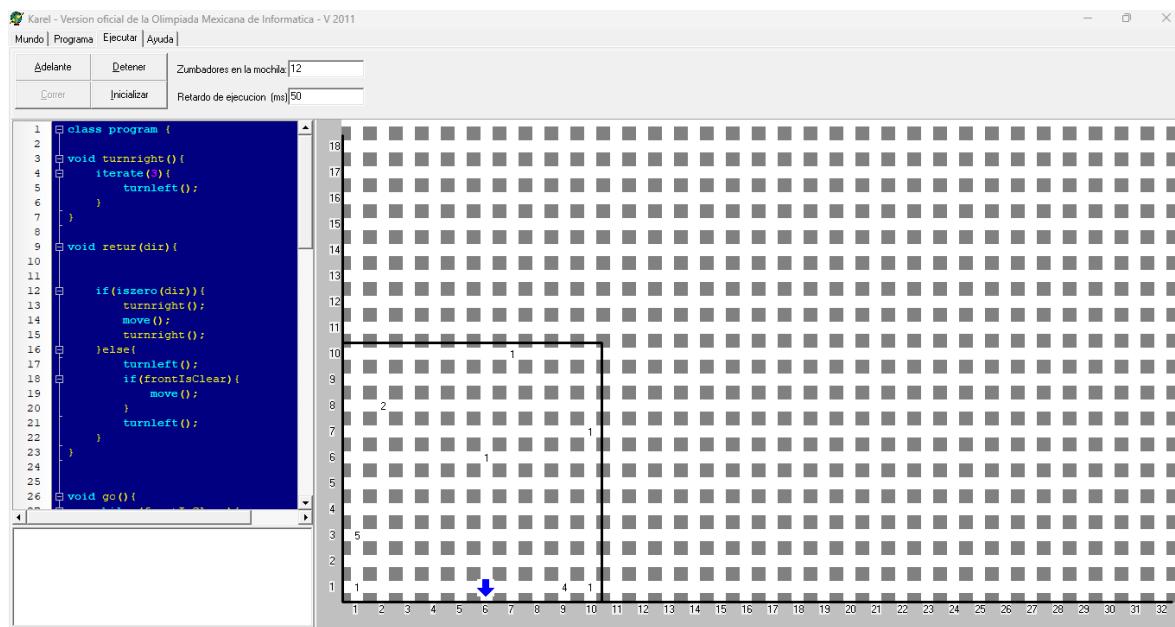
program() {
    iterate(5){
        go();
        return(0);
        go();
        return(1);
        go();
    }

    iterate(4){
        return(1);
        go();
        return(0);
        go();
    }
    turnleft();
    move();
    turnleft();
    go();
    turnleft();
    turnleft();

    turnoff();
}
}
```

Ejecución





Reconocimiento de voz y simulación

Descripción

Este proyecto implementa un sistema de control por voz para manejar una bocina utilizando reconocimiento de voz y un análisis semántico de comandos.

Código

Análisis semántico

```

class AnalizadorDeComandos :
    def __init__(self, diccionario_, reglas_):
        self.diccionario = diccionario_

```



```
self.reglas = reglas_
self.lex = {}
self.accion_key = None
self.objeto_key = None
self.valor_key = None

def _analisis_lexico(self, commands_normalizados):
    self.lex = {}
    for key, values in self.diccionario.items():
        for command in commands_normalizados:
            if command in values:
                self.lex[key] = command

    self.accion_key = None
    self.objeto_key = None
    self.valor_key = None

    for key in self.lex.keys():
        if key.startswith("accion_"):
            self.accion_key = key
        elif key.startswith("objeto_"):
            self.objeto_key = key
        elif key == "valor":
            self.valor_key = key

def _validacion(self, frase):
    if not self.accion_key or not self.objeto_key:
        return False, f"Faltan la Acción en la frase: '{frase}'"

    roles_compatibles = self.reglas[self.accion_key]

    if self.objeto_key not in roles_compatibles:
        return False, (f"La Acción '{self.lex[self.accion_key]}'"
        f"({self.accion_key}) no es compatible "
        f"con el Objeto '{self.lex[self.objeto_key]}'"
        f"({self.objeto_key}).")

    return True, "FRASE VÁLIDA."

def analizar(self, frase):
    commands = frase.split(" ")
    commands_minus = [c.lower() for c in commands]

    self._analisis_lexico(commands_minus)

    valido, mensaje = self._validacion(frase)

    print("-" * 40)
    print(f"FRASE ANALIZADA: '{frase}'")
    print(f"Roles Léxicos Encontrados: {self.lex}")
    print("-" * 40)

    if valido:
        print(f"{mensaje}")
        accion = self.lex.get(self.accion_key)
        objeto = self.lex.get(self.objeto_key)
        valor = self.lex.get(self.valor_key)

        print(f"Acción: {accion.upper()}, Objeto: '{objeto.upper()}'",
end=""")
```



```
        print(f", Valor: {valor.upper()}")
    else: print("")
else:
    print(f"ERROR: {mensaje}")

if __name__ == '__main__':
    from config import DICCIONARIO, REGLAS
    analizador = AnalizadorDeComandos(DICCIONARIO, REGLAS)

    analizador.analizar("Sube el volumen ")
```

Manejo UI

```
import sys
import os
from vosk import Model
from PyQt5 import uic, QtWidgets, QtMultimedia
from PyQt5.QtCore import QUrl
from config import DICCIONARIO, REGLAS, PORCENTOS, VOSK_MODEL_PATH
from analizador_comandos import AnalizadorDeComandos
from voice_thread import VoiceThread

if not os.path.exists(VOSK_MODEL_PATH):
    print("Descargar modelo de https://alphacepheli.com/vosk/models")
    sys.exit(1)

VOSK_MODEL = Model(VOSK_MODEL_PATH)

qtCreatorFile = "Control_bocina.ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)

class ControlBocinaView(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        self.analizador = AnalizadorDeComandos(DICCIONARIO, REGLAS)

        # Inicializar reproductor y playlist
        self.playlist = QtMultimedia.QMediaPlaylist()
        self.player = QtMultimedia.QMediaPlayer()
        self.player.setPlaylist(self.playlist)

        self.carpeta_canciones = "sounds"
        self.lista_canciones = []
        self.cargar_canciones()

        # Conectar botones
        self.btn_Encender.clicked.connect(self.encender)
        self.btn_Reproducir.clicked.connect(self.reproducir)
        self.btn_Pausar.clicked.connect(self.pausar)
        self.btn_Siguiente.clicked.connect(self.siguiente)
        self.btn_Anterior.clicked.connect(self.anterior)
        self.btn_Silencio.clicked.connect(self.silencio)
        self.btn_Aumenta.clicked.connect(self.aumenta_volumen)
        self.btn_Disminuye.clicked.connect(self.disminuye_volumen)

        self.slider_Volumen.setValue(50)
        self.player.setVolume(50)
        self.slider_Volumen.valueChanged.connect(self.player.setVolume)
        self.encendida = False
```



```
# Inicializar y Conectar Hilo de Voz
self.voice_thread = VoiceThread(model=VOSK_MODEL)
# Conectamos la señal de voz a la función de procesamiento
self.voice_thread.command_recognized.connect(self.process_voice_command)
self.voice_thread.start()

# Sobreescribir el metodo de cierre para detener el hilo
def closeEvent(self, event):
    print("\nCerrando aplicación y deteniendo hilo de voz...")
    self.voice_thread.stop()
    super().closeEvent(event)

def process_voice_command(self, command):
    self.analizador.analizar(command)

    # Obtenemos los roles funcionales clave del resultado del análisis
    accion = self.analizador.lex.get(self.analizador.accion_key)
    # objeto = self.analizador.lex.get(self.analizador.objeto_key)
    valor = self.analizador.lex.get(self.analizador.valor_key)

    # Si el análisis semántico falló, detenemos la ejecución
    valido, mensaje = self.analizador._validacion(command)
    if not valido: return

    if self.analizador.accion_key == "accion_estado":
        if accion in ["encender", "prende"]:
            self.encender()
        else:
            if self.encendida: self.encender()

    elif not self.encendida:
        print("Bocina apagada")
        return

    elif self.analizador.accion_key == "accion_play":
        if accion in ["reproducir", "reproduce"]:
            self.reproducir()
        elif accion in ["pausar", "pausa"]:
            self.pausar()
        elif accion == "siguiente":
            self.siguiiente()
        elif accion == "anterior":
            self.anterior()

    elif self.analizador.accion_key == "accion_ajuste":
        value = PORCENTOS.get(valor, None)
        if accion in ["sube", "aumenta"]:
            if valor == "máximo":
                self.player.setVolume(100)
                self.slider_Volumen.setValue(100)
            else:
                self.aumenta_volumen(value)

        elif accion in ["baja", "disminuye"]:
            if valor == "cero":
                self.player.setVolume(0)
                self.slider_Volumen.setValue(0)
            else:
                self.disminuye_volumen(value)
```



```
        elif accion in ["establece", "establecer"]:
            if not valor: return
            if valor == 'máximo':
                value = 100
            # print(value)
            self.establece_volumen(value)

        elif self.analizador.accion_key == "accion_mute":
            self.silencio()

        else:
            self.statusbar.showMessage("Comando reconocido, pero sin acción
mapeada.")
            print("INFO: Comando no mapeado a una función de la GUI.")

    def cargar_canciones(self):
        """Carga los archivos .mp3 o .wav de la carpeta canciones"""
        if os.path.exists(self.carpeta_canciones):
            for archivo in os.listdir(self.carpeta_canciones):
                if archivo.endswith(".mp3") or archivo.endswith(".wav"):
                    ruta = os.path.join(self.carpeta_canciones, archivo)
                    self.lista_canciones.append(ruta)

        # Usa QListWidget.addItem() (Asumiendo que has corregido el
        UI)
        self.listView.addItem(archivo)

        url = QUrl.fromLocalFile(os.path.abspath(ruta))
        self.playlist.addMedia(QtMultimedia.QMediaContent(url))

    if len(self.lista_canciones) > 0:
        self.playlist.setCurrentIndex(0)
        self.listView.setCurrentRow(0)

    def encender(self):
        """Encender o apagar la bocina"""
        if not self.encendida:
            self.encendida = True
            self.statusbar.showMessage("Bocina encendida ☑")
            self.btn_Encender.setText("Apagar")
        else:
            self.encendida = False
            self.player.stop()
            self.statusbar.showMessage("Bocina apagada ✗")
            self.btn_Encender.setText("Encender")

    def reproducir(self):
        """Reproducir canción seleccionada o la actual"""
        if self.encendida and self.lista_canciones:
            indice = self.listView.currentRow()
            if indice >= 0:
                self.playlist.setCurrentIndex(indice)
                self.player.play()

            if self.playlist.mediaCount() > 0 and
self.playlist.currentMedia().canonicalUrl().fileName():
                actual = self.playlist.currentMedia().canonicalUrl().fileName()
                self.statusbar.showMessage(f"Reproduciendo: {actual} ⏪")
```



```
        else:
            self.statusbar.showMessage("Bocina encendida, lista vacía 🔊")

    def pausar(self):
        if self.encendida:
            self.player.pause()
            self.statusbar.showMessage("Música en pausa ||")

    def siguiente(self):
        if self.encendida and self.playlist.mediaCount() > 0:
            self.playlist.next()
            self.player.play()
            indice = self.playlist.currentIndex()
            if indice >= 0:
                self.listWidget.setCurrentRow(indice)
                actual = self.playlist.currentMedia().canonicalUrl().fileName()
                self.statusbar.showMessage(f"Siguiente: {actual} ►►")

    def anterior(self):
        if self.encendida and self.playlist.mediaCount() > 0:
            self.playlist.previous()
            self.player.play()
            indice = self.playlist.currentIndex()
            if indice >= 0:
                self.listWidget.setCurrentRow(indice)
                actual = self.playlist.currentMedia().canonicalUrl().fileName()
                self.statusbar.showMessage(f"Anterior: {actual} ◀◀")

    def silencio(self):
        if self.encendida:
            self.player.setMuted(not self.player.isMuted())
            if self.player.isMuted():
                self.statusbar.showMessage("Silencio 🔇")
            else:
                self.statusbar.showMessage(f"Volumen: {self.player.volume()} % 🔈")
        else:
            self.statusbar.showMessage(f"Volumen: {self.player.volume()} % 🔈")

    def aumenta_volumen(self, value):
        if self.encendida:
            if self.player.isMuted():
                self.silencio()
            else:
                cantidad = 10 if not value else value
                vol = min(self.player.volume() + cantidad, 100)
                self.player.setVolume(vol)
                self.slider_Volumen.setValue(vol)
                self.statusbar.showMessage(f"Volumen: {vol} % 🔈")
        else:
            self.statusbar.showMessage(f"Volumen: {vol} % 🔈")

    def disminuye_volumen(self, value):
        if self.encendida:
            cantidad = 10 if not value else value
            vol = max(self.player.volume() - cantidad, 0)
            self.player.setVolume(vol)
            self.slider_Volumen.setValue(vol)
            self.statusbar.showMessage(f"Volumen: {vol} % 🔈")
        else:
            self.statusbar.showMessage(f"Volumen: {vol} % 🔈")

    def establece_volumen(self, value):
        if self.encendida:
            if self.player.isMuted():
                self.silencio()
            else:
```



```
        self.silencio()
    self.player.setVolume(value)
    self.slider_Volumen.setValue(value)
    self.statusbar.showMessage(f"Volumen: {value}% 🔊")
```

Hilo que escucha el micrófono, procesa audio y lo transcribe a texto usando el modelo de Vosk

```
from vosk import KaldiRecognizer
import pyaudio
from PyQt5.QtCore import QThread, pyqtSignal
import time

class VoiceThread(QThread):
    command_recognized = pyqtSignal(str)

    def __init__(self, parent=None, model=None):
        super(VoiceThread, self).__init__(parent)
        self.vosk_model = model
        self.running = True

    def run(self):
        recognizer = KaldiRecognizer(self.vosk_model, 16000)
        mic = pyaudio.PyAudio()

        stream = mic.open(format=pyaudio.paInt16,
                           channels=1,
                           rate=16000,
                           input=True,
                           frames_per_buffer=8192)
        stream.start_stream()

        print("⚡ Hilo de Voz iniciado. Escuchando...")

        while self.running:
            try:
                data = stream.read(4096, exception_on_overflow=False)

                if recognizer.AcceptWaveform(data):
                    result = recognizer.Result()

                    result = result.replace("\\"", "").replace("\n", "")
                    posDosPuntos = result.index(":") + 2
                    comando = result[posDosPuntos:-1].strip()

                    if comando:
                        print(f"COMANDO DETECTADO: {comando}")
                        self.command_recognized.emit(comando)
            except:
                pass
            except ValueError:
                print("Error de valor en stream. Reintentando...")
                time.sleep(0.1)

        # Limpieza al detener el hilo
        stream.stop_stream()
        stream.close()
```



```
mic.terminate()
print("● Hilo de Voz detenido.")

def stop(self):
    self.running = False
    self.wait()
```

Ejecución

Camino con opciones*

Descripción

Este programa implementa una representación gráfica en donde se exploran diferentes caminos entre secciones para encontrar la ruta de menor costo.

El sistema genera caminos con costos aleatorios y, mediante un proceso iterativo, selecciona, bloquea o reabre rutas según su costo acumulado.

Código

Manejo de la UI de los caminos

```
from PyQt5 import uic
from PyQt5.QtWidgets import QMainWindow
from PyQt5.QtGui import QPainter, QColor, QFont
from PyQt5.QtCore import Qt, QRect
from sapo_thread import SapoThread
from config import STATES

qtCreatorFile = "path_view.ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)

class PathView(QMainWindow, Ui_MainWindow):
    def __init__(self):
        QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)
        self.setWindowTitle("Window")

        self.levels = list()
        self.connections = list()

        self.Worker = SapoThread()
        self.Worker.Signal.connect(self.worker_conn)
        self.Worker.start()

    def worker_conn(self, nodes, tupla):
        self.levels = nodes
        self.define_connections()

        self.lbl_best.setText(str(tupla[0]))
        self.lbl_current.setText(str(tupla[1]))

        self.update()

    def define_connections(self):
        if not self.levels:
            return

        self.connections.clear()
```



```
for i in range(len(self.levels)):
    current_level = self.levels[i]
    next_level_index = i + 1
    if next_level_index > len(self.levels) - 1:
        break

    if len(self.levels[i]) == 1:
        for j in range(len(self.levels[next_level_index])):
            self.connections.append((i, 0, next_level_index, j))
    else:
        for j in range(len(current_level)):
            self.connections.append((i, j, next_level_index, 0))

def paintEvent(self, event):
    if not self.levels: # No hay nada que dibujar
        return

    painter = QPainter(self)
    painter.translate(10, 25)
    painter.setFont(QFont('Arial', 12))

    # Calcular posiciones de nodos
    self.posiciones = []
    ancho = self.width()
    alto_total = self.height()
    niveles_totales = len(self.levels)
    for nivel_index, nivel in enumerate(self.levels):
        x = 50 + nivel_index * (ancho // (niveles_totales+1))
        n_nodos = len(nivel)
        fila = []
        for i, cost_and_state in enumerate(nivel):
            y = (i+1) * (alto_total // (n_nodos+1))
            fila.append((x, y, cost_and_state[0], cost_and_state[1]))
        self.posiciones.append(fila)

    # Dibujar conexiones
    painter.setPen(QColor(0, 0, 0))
    for con in self.connections:
        n1, i1, n2, i2 = con
        x1, y1, *_ = self.posiciones[n1][i1]
        x2, y2, *_ = self.posiciones[n2][i2]
        painter.drawLine(x1, y1, x2, y2)

    # Dibujar nodos
    for nivel in self.posiciones:
        for x, y, cost, state in nivel:
            if state == STATES.get("SELECTED"):
                painter.setBrush(QColor(56, 176, 0))
            elif state == STATES.get("BLOCKED"):
                painter.setBrush(QColor(186, 24, 27))
            else:
                painter.setBrush(QColor(113, 17, 171))
            painter.drawEllipse(x-20, y-20, 40, 40)

            if cost == 0:
                continue

            painter.setPen(QColor(255, 255, 255))
            rect = QRect(x - 20, y - 20, 40, 40)
            painter.drawText(rect, Qt.AlignCenter, str(cost))
```

Modelo del nodo

```
class Node:
    def __init__(self, cost):
        self.cost = cost
        self.closed = False

    def toggle_closed_state(self):
        self.closed = not self.closed

    def __str__(self):
        return f"Node(cost={self.cost}, closed={self.closed})"
```

Hilo de búsqueda del camino óptimo

```
from PyQt5.QtCore import QThread, pyqtSignal
from config import N_SECTIONS, MAX_COST, MAX_PATHS, STATES, MAX_IT_SIN_MEJORA,
INTERVAL
from node import Node
import time
import random as rand
import copy

class SapoThread(QThread):
    Signal = pyqtSignal(list, tuple)
    # [[(cost, state), ...], ...], (best, current)

    def __init__(self, n_sections=N_SECTIONS, max_paths=MAX_PATHS,
max_cost=MAX_COST):
        super().__init__()
        self.running = True
        self.n_sections = n_sections
        self.max_paths = max_paths
        self.max_cost = max_cost

    def run(self):
        sections_og = self.create_paths()
        min_cost, min_paths = self.get_min_cost_and_paths(sections_og)
        print("====")
        print(f"Mejor Costo: {min_cost}")
        print("Caminos Óptimos:")
        for index, camino in enumerate(min_paths):
            print(str(index+1) + ". " + str(camino))
        print("====")

        it_sin_mejora = 0
        best_race = 9999

        while self.running and it_sin_mejora < MAX_IT_SIN_MEJORA:
            it = 0
            offset = 1
            current_total_cost = 0
            sections = copy.deepcopy(sections_og)

            self.update_graph(sections, best=best_race,
current=current_total_cost)
            time.sleep(INTERVAL)

            previous_costs = [(0, 0) for _ in range(N_SECTIONS)] # [(prev cost,
```

```

last path), ...
    while it < N_SECTIONS:
        section = sections[it]
        random_idx = rand.randrange(len(section)) # pos path
        selected_path = section[random_idx]
        checked = self.anyone_open(section)
        while checked and selected_path.closed:
            random_idx = rand.randrange(len(section))
            selected_path = section[random_idx]

        if not checked:
            for path in section:
                path.toggle_closed_state()
        it = it - 1
        offset = offset - 1
        self.update_graph(sections,
                          idx_section=it+offset,
                          idx_path=previous_costs[it][1],
                          best=best_race,
                          current=current_total_cost
                          )
        time.sleep(INTERVAL)
        self.update_graph(sections,
                          idx_section=it+offset-1,
                          best=best_race,
                          current=current_total_cost
                          )
        current_total_cost = previous_costs[it][0]
        time.sleep(INTERVAL)
        continue

        selected_path.toggle_closed_state()
        current_total_cost += selected_path.cost

        self.update_graph(sections,
                          idx_section=it+offset,
                          idx_path=random_idx,
                          best=best_race,
                          current=current_total_cost
                          )
        time.sleep(INTERVAL)

        if current_total_cost <= best_race:
            previous_costs[it] = (current_total_cost -
selected_path.cost, random_idx)
            it = it + 1
            offset = offset + 1
            self.update_graph(sections,
                              idx_section=it+offset-1,
                              best=best_race,
                              current=current_total_cost
                              )
        else:
            current_total_cost -= selected_path.cost
            self.update_graph(sections,
                              idx_section=it+offset-1,
                              best=best_race,
                              current=current_total_cost
                              )

```



```
        time.sleep(INTERVAL)

    if current_total_cost == best_race:
        it_sin_mejora += 1
    elif current_total_cost < best_race:
        best_race = current_total_cost
        it_sin_mejora = 0

    print(f"Mejor Costo encontrado: {best_race}")
    print("Caminos tomados:")
    #print(previous_costs)
    for index, camino in enumerate(previous_costs):
        print(str(index + 1) + ". " + str(sections_og[index][camino[1]]))
    print("=====")
    self.stop()

def stop(self):
    self.running = False
    self.wait() # Espera a que el hilo termine

def create_paths(self):
    sections = list()
    for _ in range(self.n_sections):
        n_paths = rand.randint(1, self.max_paths)
        paths = [Node(rand.randint(1, self.max_cost)) for _ in
range(n_paths)]
        sections.append(paths)

    return sections

def anyone_open(self, paths, idx=0):
    if not paths[idx].closed:
        return True

    if idx < len(paths) - 1:
        return self.anyone_open(paths, idx + 1)

    return False

def get_min_cost(self, sections):
    min_cost = 0
    for section in sections:
        min_cost += min([path.cost for path in section])

    return min_cost

def get_min_cost_and_paths(self, sections):
    total_min_cost = 0
    min_cost_path_indices = []

    for section in sections:
        min_cost_path = min(section, key=lambda path: path.cost)
        total_min_cost += min_cost_path.cost
        #print(min_cost_path)

        #min_index = section.index(min_cost_path)
        min_cost_path_indices.append(min_cost_path)

    return total_min_cost, min_cost_path_indices

def create_graph_levels(self, sections, idx_section, idx_path):
```

```

graph_levels = list()
for section in sections:
    graph_levels.append([(0 , STATES.get("AVAILABLE", 0))]) # Inicio /
descanso [(cost, state), ...]
    graph_levels.append([(path.cost, STATES.get("BLOCKED", 0) if
path.closed else STATES.get("AVAILABLE",0))
        for path in section])

graph_levels.append([(0 , STATES.get("AVAILABLE", 0))]) # Meta [(cost,
state), ...]

cost, _ = graph_levels[idx_section][idx_path]
graph_levels[idx_section][idx_path] = (cost, STATES.get("SELECTED", 0))

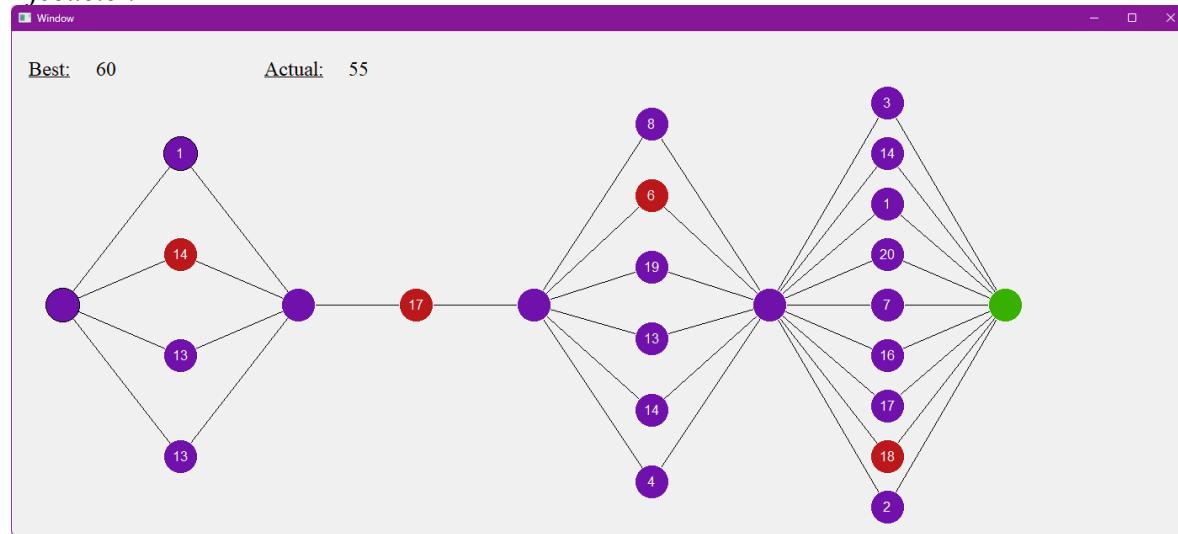
return graph_levels

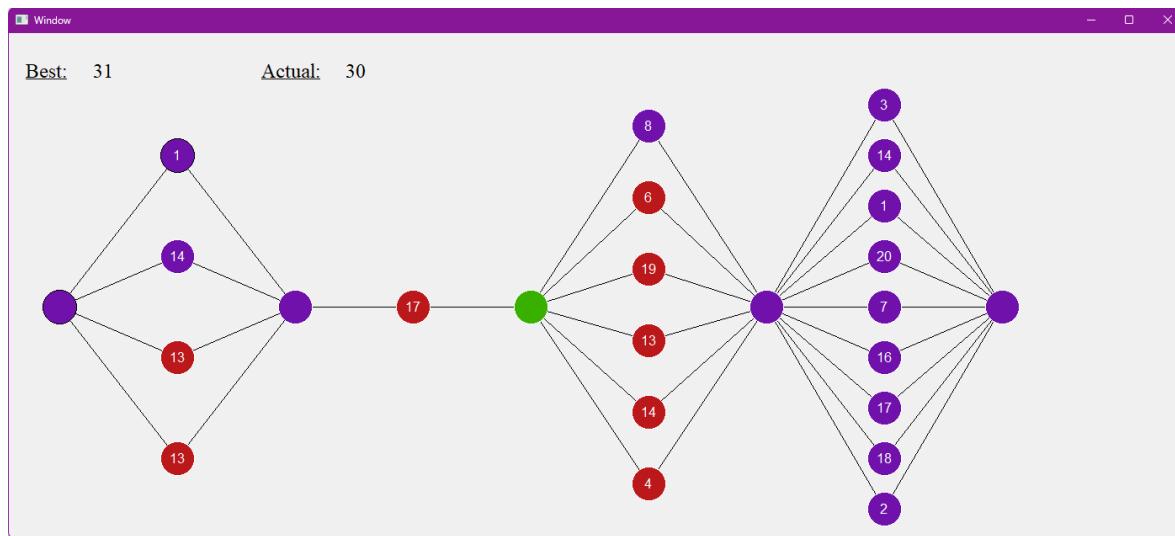
def update_graph(self, sections, idx_section=0, idx_path=0, best=9999,
current=0):
    new_list = self.create_graph_levels(sections, idx_section, idx_path)
    self.Signal.emit(new_list, (best, current))

def timeout(self):
    pass

```

Ejecución





Grafo ponderado búsqueda local

Descripción

Este programa implementa una búsqueda local para encontrar un camino de menor costo entre dos nodos dentro de un grafo generado aleatoriamente.

Código

UI de grafos generados

```
import random
import math
from PyQt5 import uic
from PyQt5.QtWidgets import (
    QGraphicsScene, QGraphicsEllipseItem,
    QGraphicsLineItem, QGraphicsTextItem, QMainWindow
)
from PyQt5.QtGui import QPen, QBrush, QPainter, QFont
from PyQt5.QtCore import Qt, QPointF, QLineF
from local_search_thread import LSThread
from config import NODE_TYPE

qtCreatorFile = "graph_view.ui"
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtCreatorFile)

class GraphView(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()
        self.setupUi(self)

        self.scene = QGraphicsScene()
        self.graph_view.setScene(self.scene)
        self.graph_view.setRenderHint(QPainter.Antialiasing)

        self.btn_new.setEnabled(False)
        self.btn_new.clicked.connect(self.new_graph)

        self.nodes = [] # [(id, type), ...]
        self.edges = [] # [(nodo1, nodo2, peso, is_selected), ...]
        self.posiciones = []
```



```
self.nodes_to_draw = {} # {id: (pos, type)}
```

```
self.Worker = None
self.new_graph()
```

```
def new_graph(self):
    self.btn_new.setEnabled(False)
```

```
# Detener thread anterior si existe
if self.Worker is not None and self.Worker.isRunning():
    self.Worker.terminate()
    self.Worker.wait()
```

```
# Crear y arrancar nuevo thread
self.nodes_to_draw = {}
self.Worker = LSThread()
self.Worker.Graph.connect(self.worker_conn)
self.Worker.finished.connect(self.thread_finished) # señal cuando
termina
self.Worker.start()
```

```
def thread_finished(self):
    # Rehabilitar botón cuando el thread termine
    self.btn_new.setEnabled(True)
```

```
def worker_conn(self, nodes, edges, weight, path):
    self.nodes = nodes
    self.edges = edges
    self.btn_new.setEnabled(not self.Worker.running)

    if not self.nodes_to_draw:
        self.posiciones = self.generar_posiciones_circulo()
        self.nodes_to_draw = self.asignar_posiciones_aleatorias()

    self.lbl_best.setText(str(weight))
    self.lbl_path.setText(path)

    self.scene.clear()
    self.dibujar_grafo()
```

```
def generar_posiciones_circulo(self):
    n = len(self.nodes)
    center_x = self.graph_view.width() / 2
    center_y = self.graph_view.height() / 2
    radio = min(self.graph_view.width(), self.graph_view.height()) / 2 - 80
    posiciones = []
    for i in range(n):
        angulo = 2 * math.pi * i / n
        x = center_x + radio * math.cos(angulo)
        y = center_y + radio * math.sin(angulo)
        posiciones.append(QPointF(x, y))
    return posiciones
```

```
def asignar_posiciones_aleatorias(self):
    posiciones_disponibles = self.posiciones.copy()
    random.shuffle(posiciones_disponibles)
    return {id: (posiciones_disponibles[i], type)
            for i, (id, type) in enumerate(self.nodes)}
```

```
def dibujar_grafo(self):
    if not self.nodes or not self.edges:
```



```
return

outline_color_green = Qt.green
outline_color_red = Qt.red
outline_color_black = Qt.black
brush_nodo = QBrush(Qt.white)
fuente_nodo = QFont("Arial", 14)
fuente_peso = QFont("Arial", 16)

# Dibujar aristas
for origen, destino, peso, is_selected in self.edges:
    p1 = self.nodes_to_draw[origen][0]
    p2 = self.nodes_to_draw[destino][0]

    linea = QGraphicsLineItem(p1.x(), p1.y(), p2.x(), p2.y())
    linea.setPen(QPen(Qt.yellow if is_selected else Qt.black, 2))
    self.scene.addItem(linea)

    # Texto del peso
    line = QLineF(p1, p2)
    mid_x = (p1.x() + p2.x()) / 2
    mid_y = (p1.y() + p2.y()) / 2
    dx = line.dx()
    dy = -line.dy()
    longitud = (dx**2 + dy**2) ** 0.5
    if longitud != 0:
        dx /= longitud
        dy /= longitud
    desplazamiento = 20
    peso_pos_x = mid_x + dx * desplazamiento
    peso_pos_y = mid_y + dy * desplazamiento
    texto_peso = QGraphicsTextItem(str(peso))
    texto_peso.setFont(fuente_peso)
    texto_peso.setPos(peso_pos_x - 10, peso_pos_y - 10)
    self.scene.addItem(texto_peso)

    # Dibujar nodos
    for id, (pos, typeof) in self.nodes_to_draw.items():
        if typeof == NODE_TYPE.get("START"):
            pen_outline = QPen(outline_color_green, 3)
        elif typeof == NODE_TYPE.get("END"):
            pen_outline = QPen(outline_color_red, 3)
        else:
            pen_outline = QPen(outline_color_black, 3)

        node = QGraphicsEllipseItem(pos.x() - 25, pos.y() - 25, 50, 50)
        node.setBrush(brush_nodo)
        node.setPen(pen_outline)
        self.scene.addItem(node)

        text = QGraphicsTextItem(id)
        text.setFont(fuente_nodo)
        text.setPos(pos.x() - 10, pos.y() - 15)
        self.scene.addItem(text)

bounding_rect = self.scene.itemsBoundingRect()
padding = 50
bounding_rect.adjust(-padding, -padding, padding, padding)
self.scene.setSceneRect(bounding_rect)
self.graph_view.fitInView(self.scene.sceneRect(), Qt.KeepAspectRatio)
```



Estructura y generador de grafos

```
import random as rand

class Node:
    def __init__(self, id):
        self.id = id
        self.adjacents = {} # { id: weight, ... }

    def add_adjacent(self, node_id, weight=1):
        self.adjacents[node_id] = weight

    def __repr__(self):
        return f"Node({self.id}, adjacents={self.adjacents})"

    def __str__(self):
        return f"Node {self.id} -> {self.adjacents}"

class Graph:
    def __init__(self, n_vertices, max_weight, prob_path):
        self.n = n_vertices
        self.max_weight = max_weight
        self.p = prob_path
        self.nodes = []

    def create_graph(self):
        ids = [chr(65 + i) for i in range(self.n)] # ['A', 'B', 'C', ...]
        self.nodes = [Node(id) for id in ids]

        for i in range(self.n):
            node = self.nodes[i]
            for j in range(self.n):
                adjacent_node = self.nodes[j]
                condition = i != j and not ids[i] in adjacent_node.adjacents

                if condition and rand.random() < self.p:
                    weight = rand.randint(1, self.max_weight)
                    node.add_adjacent(ids[j], weight)

                adjacent_node.add_adjacent(ids[i], weight)

    def get_nodes(self):
        return self.nodes

    def get_node(self, id):
        for node in self.nodes:
            if node.id == id:
                return node
        return None

    def __str__(self):
        return "\n".join(str(node) for node in self.nodes)

if __name__ == "__main__":
    g = Graph(n_vertices=5, max_weight=10, prob_path=0.4)
    g.create_graph()
    print(g)

    nodos = g.get_nodes()
```

```

print(nodos)

print(nodos[0].id, nodos[0].adjacents)

# Cantidad de nodos, c/nodo tendra: id, adjacents
# Grafo: Matriz de adyacencia/costos

```

Hilo de búsqueda local para optimización de caminos

```

import time
from PyQt5.QtCore import QThread, pyqtSignal
from config import MAX_IT, MAX_WEIGHT, PROB_PATH, N_NODES, NODE_TYPE
from graph import Graph
import random as rand

class LSThread(QThread):
    # [ ('A', 'start'), ('B', 'end'), ...], (NodoId, node_type)
    # [ ('A', 'B', 12), ('A', 'F', 3), ...], Edges(Nodo1, Nodo2, peso, selected)
    # costo
    # path
    Graph = pyqtSignal(list, list, int, str)

    def __init__(self, max_it=MAX_IT, n=N_NODES, max_weight=MAX_WEIGHT,
p=PROB_PATH):
        super().__init__()
        self.running = True
        self.n = n
        self.max_weight = max_weight
        self.p = p
        self.max_it = max_it

    def update_view(self, nodes, edges=None, best_weight=9999, best_path=""):
        self.Graph.emit(nodes, edges, best_weight, best_path)

    def run(self):
        it = 0
        graph = Graph(self.n, self.max_weight, self.p)
        graph.create_graph()
        print(graph, end='\n\n')

        start_node, end_node = self.select_start_end_nodes(graph)
        print("Nodo inicio: " + str(start_node))
        print("Nodo fin: " + str(end_node))

        _nodes = self.get_nodes_from_graph(graph, start_node, end_node)
        edges = self.get_edges(graph)
        self.update_view(_nodes, edges)

        path = self.initial_path(graph.nodes, start_node, end_node)
        temp_weight = self.evaluation(path)
        best_weight = 9999 if temp_weight is None else temp_weight
        best_path = None if temp_weight is None else path

        print(f"Solución inicial (Costo: {best_weight}):")
        print(" -> ".join([n.id for n in path]), end='\n')

        time.sleep(1)

        while self.running and it < self.max_it:
            it += 1
            path = self.neighborhood(graph.nodes, path)

```



```
temp_weight = self.evaluation(path)

    if temp_weight is not None and temp_weight < best_weight:
        best_path = path
        best_weight = temp_weight
        print(f"Iteración {it}: Mejora encontrada. Nuevo mejor peso:
{best_weight}")

    if best_path is not None:
        print(f"\nMejor camino encontrado (Costo: {best_weight}):")
        str_path = " -> ".join([n.id for n in best_path])
        edges = self.get_edges(graph, best_path)

        print(str_path)
    else:
        str_path = "No se encontró un camino válido"
        print("\nNo se pudo encontrar un camino válido.")
    self.update_view(_nodes, edges, best_weight, str_path)
    self.stop()

def stop(self):
    self.running = False
    self.wait()

def get_edges(self, graph, path=None):
    if path is not None:
        path_edges = set()
        for i in range(len(path) - 1):
            # sorted para que sea no dirigido
            path_edges.add(tuple(sorted([path[i].id, path[i + 1].id])))

    edges = []
    seen = set() # Para grafo no dirigido

    for node in graph.nodes:
        for neighbor, weight in node.adjacents.items():
            key = tuple(sorted([node.id, neighbor]))
            if key not in seen:
                seen.add(key)
                selected = False if path is None else key in path_edges
                edges.append((node.id, neighbor, weight, selected))

    return edges

def get_nodes_from_graph(self, graph, start_node, end_node):
    nodes = list()
    for node in graph.nodes:
        if node.id == start_node.id:
            nodes.append((node.id, NODE_TYPE["START"]))
        elif node.id == end_node.id:
            nodes.append((node.id, NODE_TYPE["END"]))
        else:
            nodes.append((node.id, NODE_TYPE["REGULAR"]))

    return nodes

def initial_path(self, nodes, start, end): # ([Node, ...], Node, Node)
    solution = nodes.copy()
    solution.remove(start)
    solution.remove(end)
    n = randint(0, len(solution))
```



```
if n == 0:
    return [start, end]
middle_path = rand.sample(solution, n)
return [start] + middle_path + [end]

def select_start_end_nodes(self, graph):
    n = len(graph.nodes)
    start_node = rand.randrange(n) # returns integer
    end_node = rand.randrange(n)
    while end_node == start_node:
        end_node = rand.randrange(n)

    return graph.nodes[start_node], graph.nodes[end_node]

def evaluation(self, solution):
    weight = 0
    for i in range(len(solution)-1):
        node = solution[i]

        adjacent_node = solution[i + 1].id
        if adjacent_node in node.adjacents:
            weight += node.adjacents.get(adjacent_node)
        else:
            return None

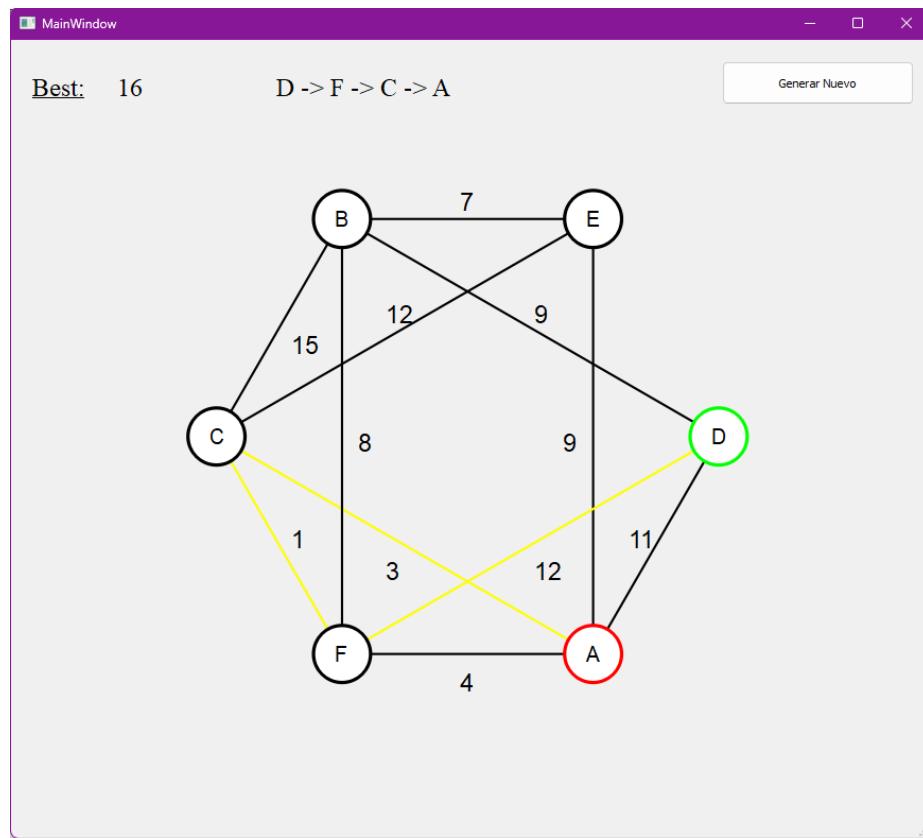
    return weight

def neighborhood(self, nodes, current_path):
    # Cambiar un path por otro
    # Tomar uno random de middle path y cambiarlo por uno que no este en el
    current path
    if len(current_path) <= 2:
        return current_path
    middle_path = current_path[1:-1]
    set_current_path = set(current_path)
    random_path_idx = rand.randrange(len(middle_path))
    set_current_path.remove(middle_path[random_path_idx])

    new_path = rand.choice(nodes)
    while new_path in set_current_path:
        new_path = rand.choice(nodes)

    middle_path[random_path_idx] = new_path
    return [current_path[0]] + middle_path + [current_path[-1]]
```

Ejecución



Cadenas de Markov

Descripción

El ejercicio consistió en implementar una simulación de Cadenas de Markov para modelar la evolución probabilística de tres posibles estados de comportamiento: hacer la tarea (A), dormir (B) y jugar (C). A partir de un vector de estado inicial y una matriz de transición previamente definida, se desarrolló un algoritmo que realizó multiplicaciones matriciales iterativas durante cinco períodos, permitiendo observar cómo cambian las probabilidades del sistema en cada paso.

Código

```
#vector inicial
p_actual = [0.3, 0.2, 0.5]

# A = Hacer la tarea
# B = Dormir
# C = Jugar
#   A     B     C
T = [
    [0.1, 0.3, 0.6],
    [0.2, 0.5, 0.3],
    [0, 0.5, 0.5]
]

for paso in range(5):
    p_siguiente = []

    for i in range(3):
        suma = 0
        for j in range(3):
            suma += p_actual[j] * T[j][i]
        p_siguiente.append(suma)

    p_actual = p_siguiente
```

```

for i in range(len(T)):    #fila
    probabilidad = 0
    for j in range(len(T[i])): #columna
        suma = p_actual[j] * T[j][i]
        probabilidad += suma
    p_siguiente.append(probabilidad)

print("          A   B   C")
print(f" Resultado del paso {paso + 1}: {p_siguiente}")
print(f" El valor mas grande es: {max(p_siguiente)})")

p_actual = p_siguiente

```

Ejecución

```

C:\Users\fatim\AppData\Local\Programs\Python\Python310\python.exe C:/Users/fatim/Documents/AyR_2025_3\Unidad_3\CadenaMarkov\morkov_nveces.py
          A   B   C
Resultado del paso 1: [0.07, 0.44, 0.49]
El valor mas grande es: 0.49
          A   B   C
Resultado del paso 2: [0.0950000000000001, 0.486, 0.4190000000000004]
El valor mas grande es: 0.486
          A   B   C
Resultado del paso 3: [0.1067000000000002, 0.4810000000000004, 0.4123]
El valor mas grande es: 0.4810000000000004
          A   B   C
Resultado del paso 4: [0.10687, 0.4786600000000003, 0.41447]
El valor mas grande es: 0.4786600000000003
          A   B   C
Resultado del paso 5: [0.1064190000000001, 0.478626, 0.414955]
El valor mas grande es: 0.478626

Process finished with exit code 0

```

PID

Introducción

El controlador PID (Proporcional-Integral-Derivativo) es uno de los mecanismos de control por retroalimentación más utilizados en la ingeniería y la automatización industrial. Su función principal es corregir el comportamiento de un sistema para que una variable física (como velocidad, temperatura o posición) se mantenga en un valor deseado o “setpoint”.

El algoritmo opera calculando continuamente la diferencia entre el valor medido y el valor deseado (el **error**) y aplicando una corrección basada en tres términos:

- **Proporcional (P):** Reacciona al error actual.
- **Integral (I):** Corrige la acumulación de errores pasados.
- **Derivativo (D):** Predice el error futuro basándose en la tasa de cambio.

En sistemas de movilidad autónoma, el PID es esencial para lograr transiciones suaves, evitando oscilaciones bruscas al acelerar o frenar.

Objetivo de la práctica

El objetivo de esta práctica es implementar y sintonizar un algoritmo de control PID dentro de un entorno de simulación física en Unity 2D para gobernar la cinemática de un vehículo autónomo, enfocándose específicamente en la gestión eficiente de la velocidad y el frenado adaptativo. La actividad busca demostrar la capacidad del controlador para reaccionar ante la aparición repentina de un obstáculo (un peatón) en una trayectoria continua, modulando la fuerza de desaceleración en tiempo real para detener el automóvil a una distancia segura sin producir colisiones y, posteriormente, gestionar la reincorporación suave y estable a la velocidad de crucero una vez despejado el camino.

Código

Coche

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CarControllerSimple : MonoBehaviour
{
    [Header("Configuración de la Pista")]
    public TrackPath track;
    public float radioX = 10f;
    public float radioY = 5f;

    [Header("Configuración del Coche")]
    public float motorSpeed = 20f; // Velocidad Máxima
    public float acceleration = 10f; // Qué tan rápido acelera/frena
    public float rotationSpeed = 300f; // Velocidad de giro (importante si vas
    rápido)

    [Header("PIDs")]
    // Kp=3, Ki=0, Kd=0.5 funcionan bien
    public PIDController steeringPid = new PIDController();

    [Header("Sensores")]
    public float detectionDistance = 4f;
    public LayerMask obstacleLayer;

    // Variable privada para guardar la velocidad real
    private float currentSpeed = 0f;

    void Start()
    {
        PlaceCarOnTrack();
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.R)) PlaceCarOnTrack();

        // --- 1. VELOCIDAD Y OBSTÁCULOS ---
        float targetSpeed = motorSpeed;
        RaycastHit2D hit = Physics2D.Raycast(transform.position, transform.up,
        detectionDistance, obstacleLayer);
```



```
if (hit.collider != null) targetSpeed = 0f;

// Esto permite que la velocidad se mantenga y aumente correctamente
currentSpeed = Mathf.MoveTowards(currentSpeed, targetSpeed, acceleration *
Time.deltaTime);

// Mover el coche
transform.Translate(Vector3.up * currentSpeed * Time.deltaTime);

// --- 2. LÓGICA PID (DIRECCIÓN) ---
if (track != null)
{
    Vector3 closestPoint = track.GetClosestPointOnPath(transform.position);
    float cte = CalculateCTE(closestPoint);

    // Debug visual
    Debug.DrawLine(transform.position, closestPoint, Color.green);

    // PID
    float pidOutput = steeringPid.Calculate(0, cte);

    // Aplicar giro
    float rotationAmount = pidOutput * rotationSpeed * Time.deltaTime;
    transform.Rotate(0, 0, rotationAmount);
}
}

public void PlaceCarOnTrack()
{
    float randomAngle = Random.Range(0f, 360f) * Mathf.Deg2Rad;
    float x = Mathf.Cos(randomAngle) * radioX;
    float y = Mathf.Sin(randomAngle) * radioY;
    transform.position = new Vector3(x, y, 0);

    // Alinear
    float nextAngle = randomAngle + 0.1f;
    Vector3 lookDir = (new Vector3(Mathf.Cos(nextAngle) * radioX,
Mathf.Sin(nextAngle) * radioY, 0) - transform.position).normalized;
    transform.up = lookDir;

    steeringPid.Reset();
    currentSpeed = motorSpeed; // Arrancar con velocidad
}

float CalculateCTE(Vector3 closestPoint)
{
    float distance = Vector3.Distance(transform.position, closestPoint);
    Vector3 directionToPoint = (closestPoint - transform.position).normalized;
    float side = Vector3.Dot(transform.right, directionToPoint);
    return distance * Mathf.Sign(side);
}

Trayectoria de la via
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
[RequireComponent(typeof(LineRenderer))]
public class TrackPath : MonoBehaviour
{
    public float radioX = 10f; // Ancho del óvalo
    public float radioY = 5f; // Alto del óvalo
    public int segmentos = 100;
    private LineRenderer lineRenderer;

    void Awake()
    {
        lineRenderer = GetComponent<LineRenderer>();
        lineRenderer.positionCount = segmentos + 1;
        lineRenderer.useWorldSpace = true;
        DrawOval();
    }

    void DrawOval()
    {
        for (int i = 0; i <= segmentos; i++)
        {
            float angulo = (float)i / (float)segmentos * 2 * Mathf.PI;
            float x = Mathf.Sin(angulo) * radioX;
            float y = Mathf.Cos(angulo) * radioY;
            lineRenderer.SetPosition(i, new Vector3(x, y, 0));
        }
    }

    // --- ¡LA FUNCIÓN MÁS IMPORTANTE! ---
    // Encuentra el punto más cercano en el óvalo a una posición dada (la del
    coche)
    public Vector3 GetClosestPointOnPath(Vector3 carPosition)
    {
        Vector3 closestPoint = Vector3.zero;
        float minDistance = float.MaxValue;

        // Iteramos por todos los segmentos para encontrar el más cercano
        for (int i = 0; i < segmentos; i++)
        {
            Vector3 p1 = lineRenderer.GetPosition(i);
            Vector3 p2 = lineRenderer.GetPosition(i + 1);

            // Función mágica que encuentra el punto más cercano en un segmento de
            // línea
            Vector3 pointOnSegment = ClosestPointOnLineSegment(p1, p2,
                carPosition);
            float distance = Vector3.Distance(carPosition, pointOnSegment);

            if (distance < minDistance)
            {
                minDistance = distance;
                closestPoint = pointOnSegment;
            }
        }
        return closestPoint;
    }

    // Función auxiliar para GetClosestPointOnPath
    Vector3 ClosestPointOnLineSegment(Vector3 p1, Vector3 p2, Vector3 point)
    {

```



```
    Vector3 p1_to_point = point - p1;
    Vector3 p2_to_p1 = p2 - p1;
    float l2 = p2_to_p1.sqrMagnitude;
    if (l2 == 0) return p1;
    float t = Mathf.Clamp01(Vector3.Dot(p1_to_point, p2_to_p1) / l2);
    return p1 + t * p2_to_p1;
}

}

Gestor de juegos
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameManager : MonoBehaviour
{
    public GameObject personPrefab;
    public TrackPath track;

    // Esta función la conectarás al OnClick() del botón en el Inspector
    public void SpawnPerson()
    {
        // Elige un punto aleatorio en el óvalo para aparecer
        int randomSegment = Random.Range(0, track.segmentos);
        Vector3 spawnPoint =
track.GetComponent<LineRenderer>().GetPosition(randomSegment);

        Instantiate(personPrefab, spawnPoint, Quaternion.identity);
    }
}

Crear persona
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Person : MonoBehaviour
{
    // La persona se destruye sola después de 5 segundos.
    public float lifeTime = 5f;

    void Start()
    {
        // El coche la verá, frenará, y luego la persona desaparecerá
        // y el coche volverá a acelerar.
        Destroy(gameObject, lifeTime);
    }
}
```

Ejecución

