**Universidad Autónoma de Tamaulipas**

**Facultad de Ingeniería Tampico**

A black background with a black square

Description automatically generated with medium confidence

**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](#_Toc215560506)

[Repositorio(s) 3](#_Toc215560507)

[Ejercicios 3](#_Toc215560508)

[CNN Modelos > 85% 3](#_Toc215560509)

[CNN Tiempo real 7](#_Toc215560510)

[Modelos OpenCV 11](#_Toc215560511)

[MediaPipe 1 12](#_Toc215560512)

[MediaPipe 5 gestos con direcciones 14](#_Toc215560513)

[Carrito entrono virtual 20](#_Toc215560514)

[Circuito realizado 26](#_Toc215560515)

[Prueba de circuito con manejo de teclado 27](#_Toc215560516)

[Laberinto procedural 33](#_Toc215560517)

[Laberinto completado de forma manual 38](#_Toc215560518)

[Laberinto completado de forma automática 44](#_Toc215560519)

[Karel letra 47](#_Toc215560520)

[Karel come 3 Beepers 50](#_Toc215560521)

[Reconocimiento de voz y simulación 53](#_Toc215560522)

[Camino con opciones\* 60](#_Toc215560523)

[Grafo ponderado búsqueda local 66](#_Toc215560524)

[Cadenas de Markov 73](#_Toc215560525)

[PID 74](#_Toc215560526)

# Evidencia de trabajos

## Repositorio(s)

|  |  |
| --- | --- |
| Ejercicios | Repositorio |
| 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*

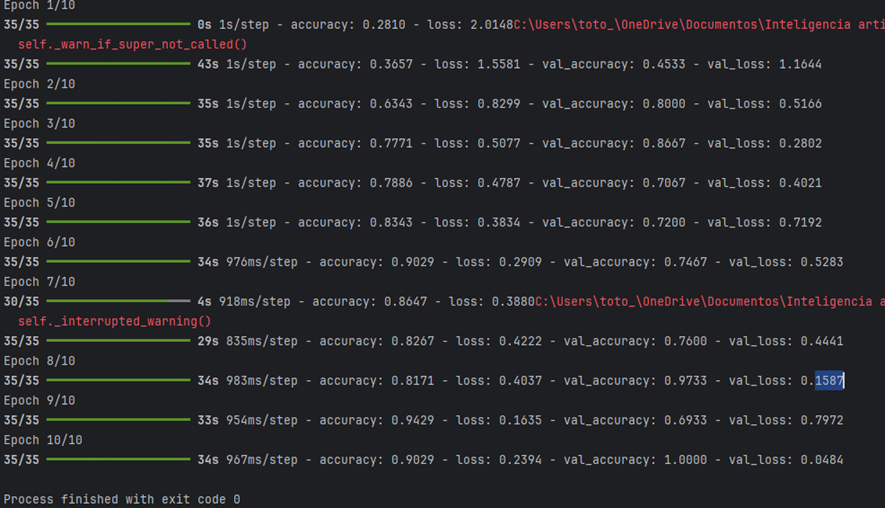
Entrenamiento de una CNN para reconocer rostros

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 redimenzionar*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*

Evalúa 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, " Resulto: ", prediction in folder)  
 count\_predictions += 1  
 if prediction in folder:  
 correct +=1  
  
 print("Efficiency: ", (correct/count\_predictions\*100))  
  
probar\_red\_neuronal()

*Ejecución*

**

**

### 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, "✖️ 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*

Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto.

### 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*

*Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.*

### 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*

Imagen que contiene interior, juguete, tabla, pequeño

El contenido generado por IA puede ser incorrecto.

### 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. MEÑIQUE ---* 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*

Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto.

*Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto.*

Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto.

### Carrito entrono 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.

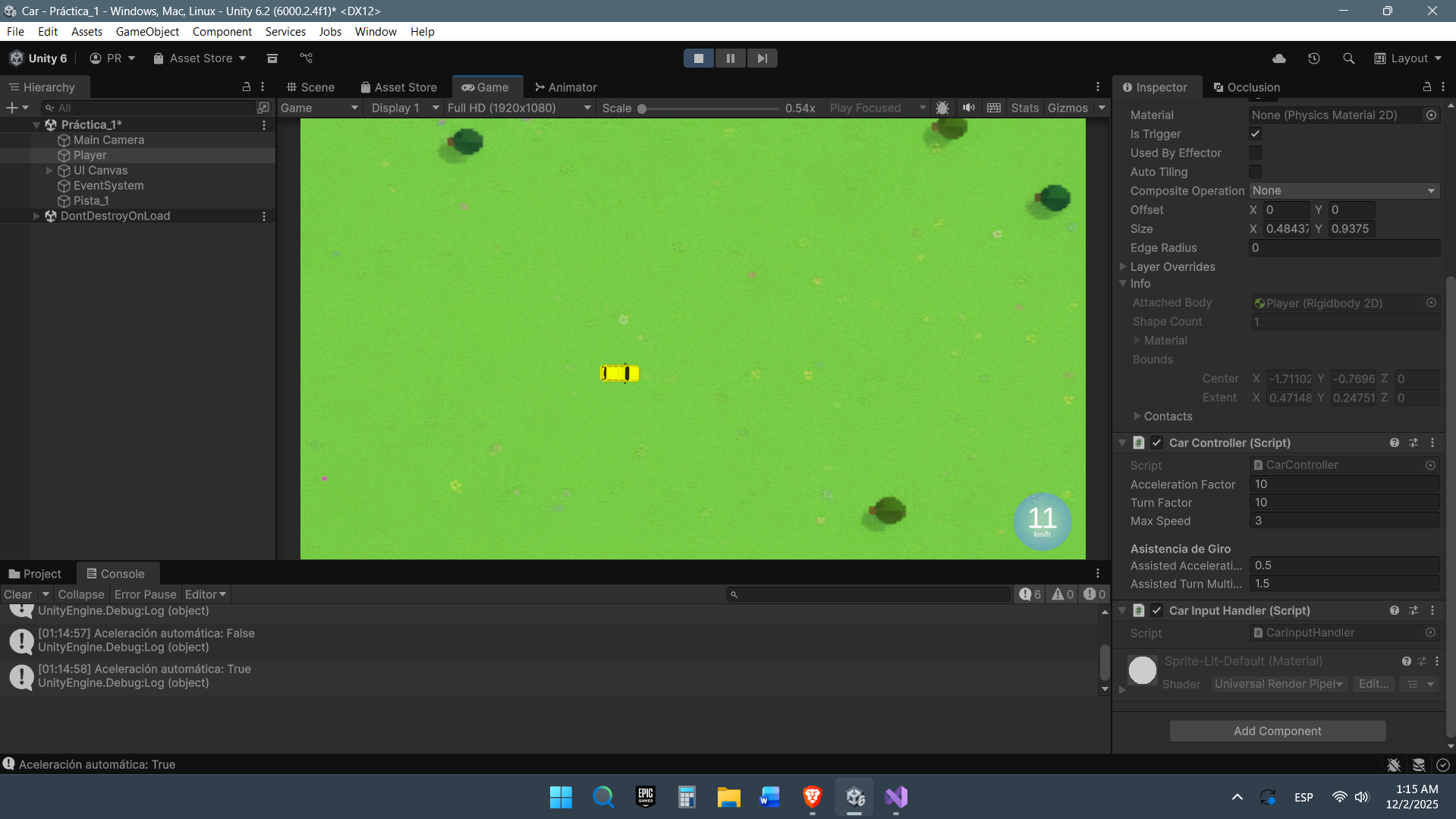
// Aquí solo necesitamos el valor X (giro) y el Y debe ser 0.

\_move = new Vector2(rawMove.x, 0f);

}

}

*Ejecución*

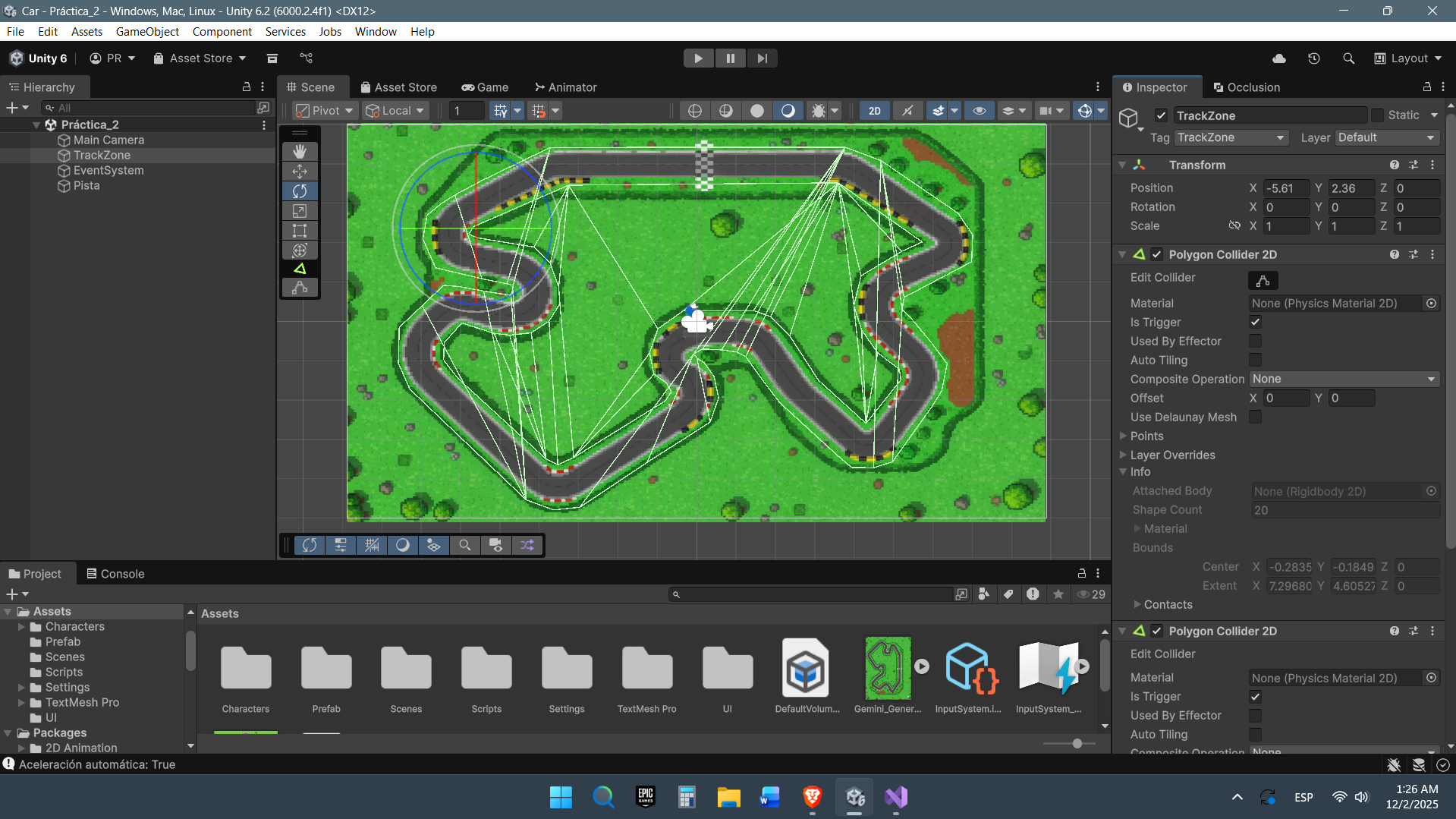


### Circuito realizado

*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()

{

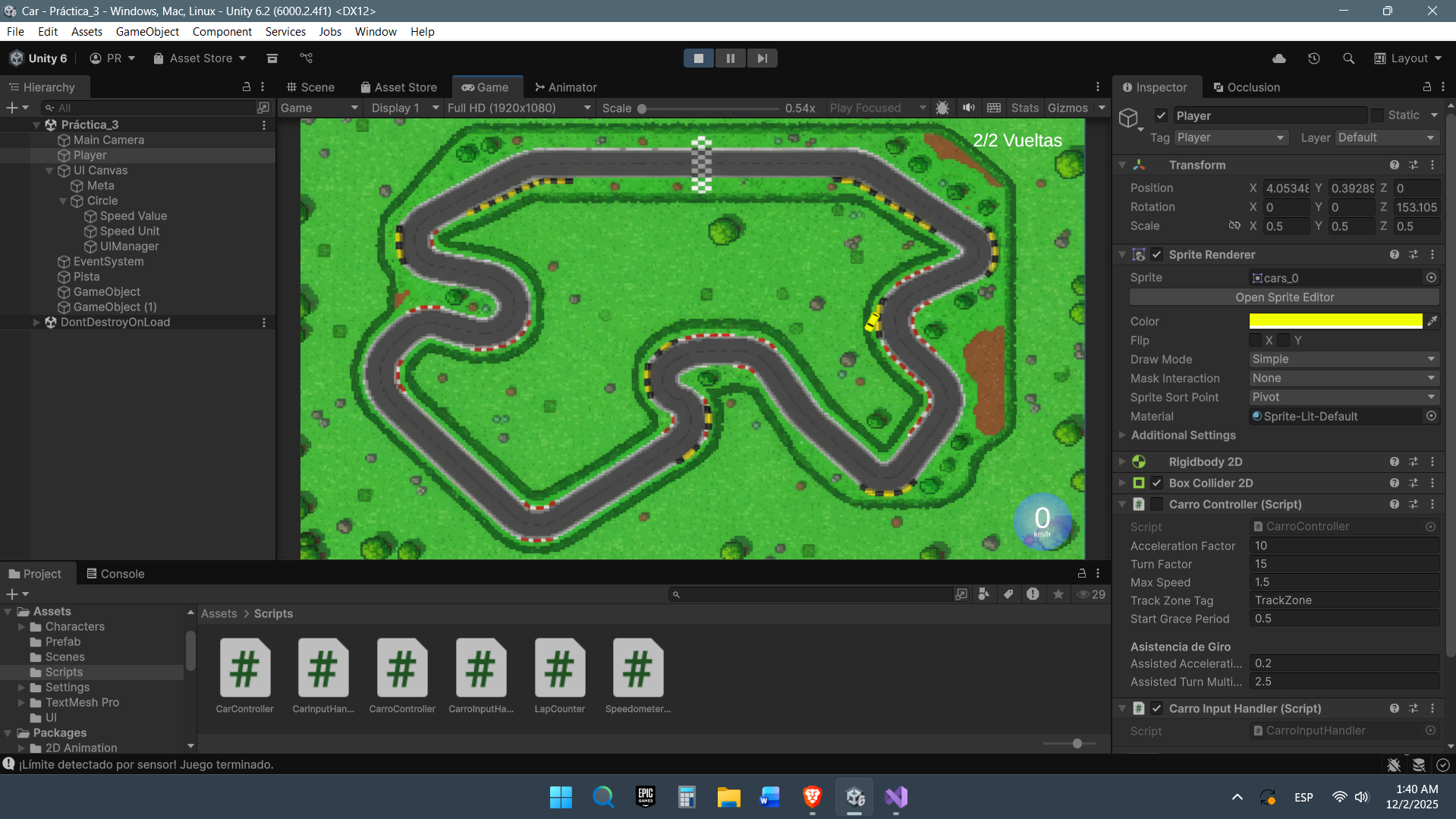
Vector2 rawMove = m\_MoveAction.ReadValue<Vector2>();

\_move = new Vector2(rawMove.x, 0f); // Solo propagamos el giro (X)

}

}

*Ejecución*



### Laberinto procedural

*Descripción*

Este programa genera un laberinto procedural en unity. Se desarrollo 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*

*Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.*

### 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++)

{

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*

Pantalla de un computador

El contenido generado por IA puede ser incorrecto.

### 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*

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

### 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();

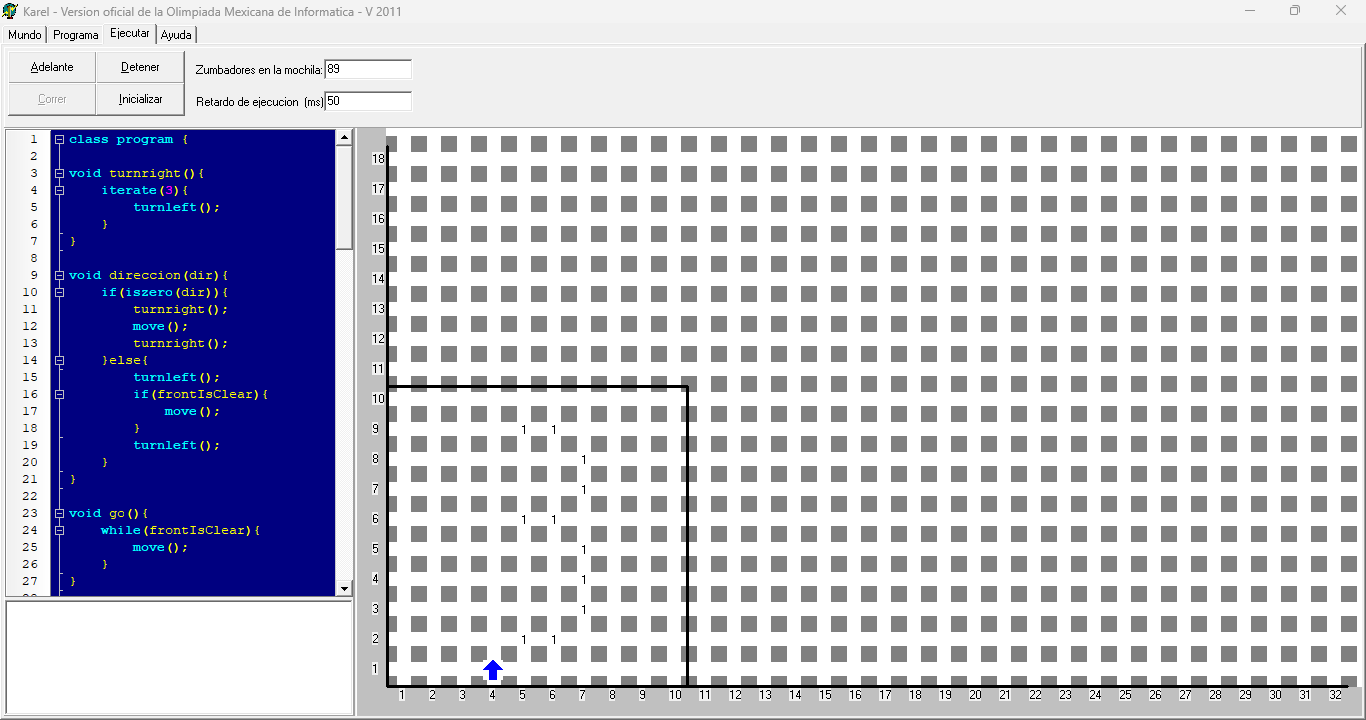
}

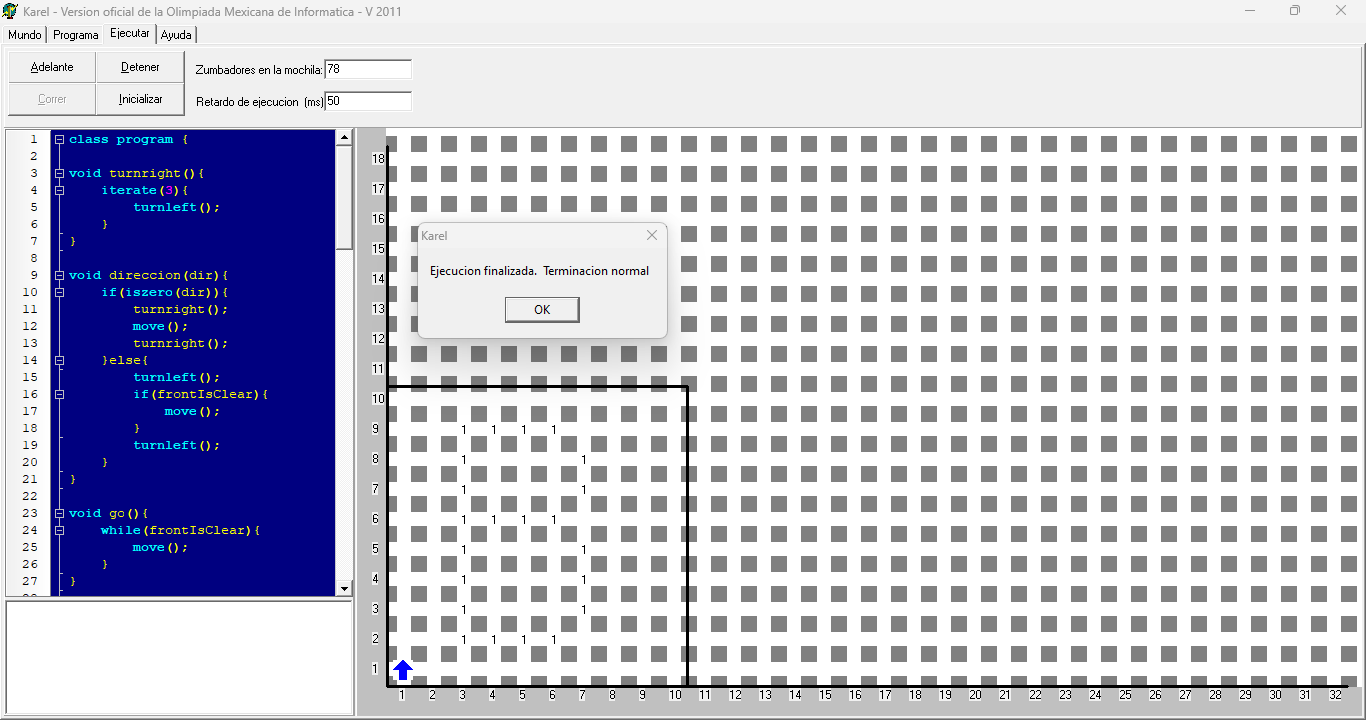
}

*Ejecución*

Patrón de fondo

El contenido generado por IA puede ser incorrecto.





### 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();

retur(0);

go();

retur(1);

go();

}

iterate(4){

retur(1);

go();

retur(0);

go();

}

turnleft();

move();

turnleft();

go();

turnleft();

turnleft();

turnoff();

}

}

*Ejecución*

Patrón de fondo

El contenido generado por IA puede ser incorrecto.

Imagen que contiene computadora

El contenido generado por IA puede ser incorrecto.

Imagen que contiene computadora

El contenido generado por IA puede ser incorrecto.

Imagen de la pantalla de un computador

El contenido generado por IA puede ser incorrecto.

### 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]}' ({self.accion\_key}) no es compatible "  
 f"con el Objeto '{self.lex[self.objeto\_key]}' ({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="")  
 if valor:  
 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, PORCIENTOS, 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://alphacephei.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.siguiente()  
 elif accion == "anterior":  
 self.anterior()  
  
 elif self.analizador.accion\_key == "accion\_ajuste":  
 value = PORCIENTOS.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.listWidget.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.listWidget.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.listWidget.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()}% 🔊")  
  
 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}% 🔊")  
  
 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}% 🔉")  
  
 def establece\_volumen(self, value):  
 if self.encendida:  
 if self.player.isMuted():  
 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)  
 else:  
 pass  
  
 except IOError as e:  
 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*

*Gráfico, Gráfico radial

El contenido generado por IA puede ser incorrecto.*

*Gráfico, Gráfico radial

El contenido generado por IA puede ser incorrecto.*

### 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.dy()  
 dy = -line.dx()  
 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 = rand.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*

*Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.*

### 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 periodos, 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(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*

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

### 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 practica*

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

Pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Pantalla de computadora

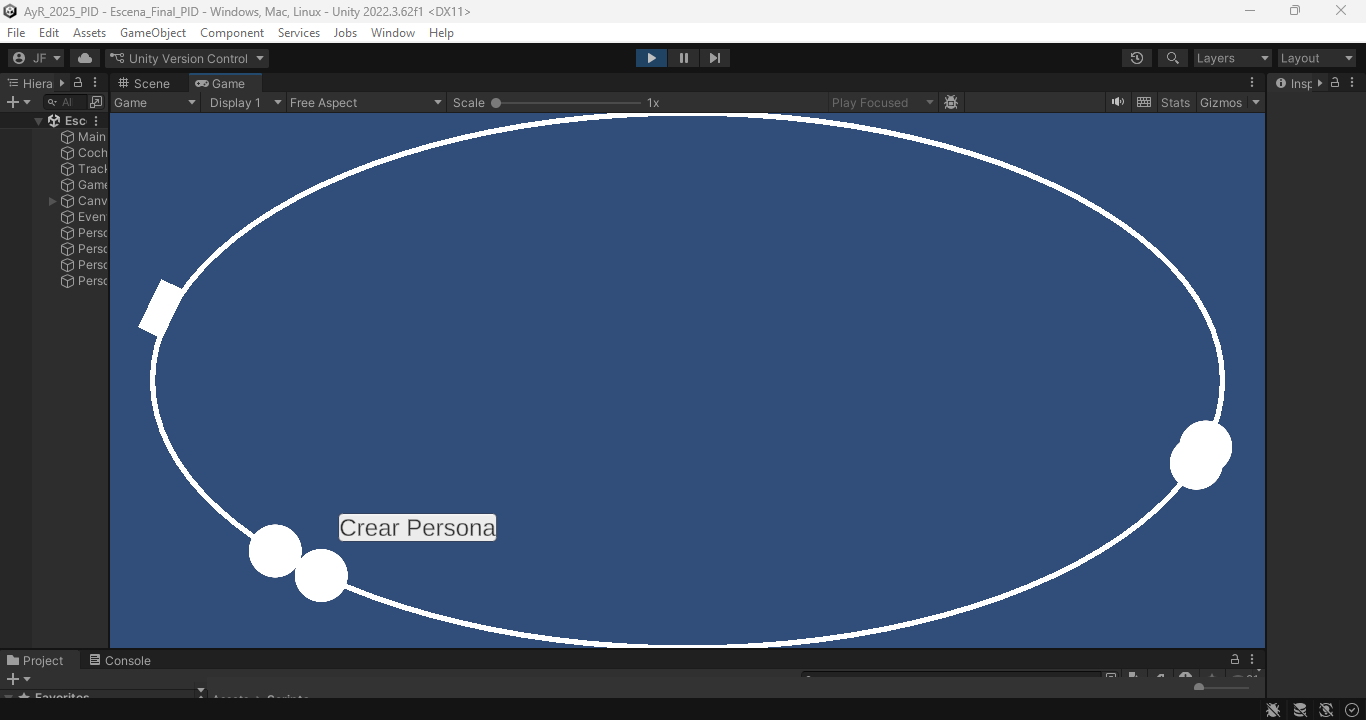
El contenido generado por IA puede ser incorrecto.

Pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Pantalla de computadora

El contenido generado por IA puede ser incorrecto.



Pantalla de computadora

El contenido generado por IA puede ser incorrecto.