Reporte Técnico del Proyecto ChecoGame

1. Introducción

El proyecto ChecoGame es un videojuego desarrollado utilizando la biblioteca Pygame en Python. El juego consiste en controlar un automóvil y esquivar enemigos que se desplazan por la pantalla. Además, el juego incorpora tecnologías de visión por computadora para interactuar con la cámara y detectar movimientos faciales.

2. Arquitectura del Sistema

2.1 Componentes Principales

El sistema está compuesto por los siguientes componentes:

Pygame: Biblioteca principal utilizada para crear el juego y manejar gráficos, eventos y sonido.

MoviePy: Utilizado para reproducir un video de introducción antes de iniciar el juego.

OpenCV: Utilizado para la captura y procesamiento de video desde la webcam.

Mediapipe: Utilizado para detectar y procesar movimientos faciales.

2.2 Estructura del Código

El código está organizado de la siguiente manera:

main(): Función principal que inicializa el juego.

play_video(): Reproduce un video de introducción.

show_menu(): Muestra el menú inicial del juego.

Enemy: Clase que define a los enemigos del juego.

Background: Clase que maneja el fondo del juego.

Player: Clase que define al jugador y su interacción.

Game: Clase principal que contiene la lógica del juego.

Webcam: Clase para manejar la captura de video desde la webcam.

3. Descripción de la Arquitectura del Sistema

3.1 Clases y Funciones

3.1.1 Clase Enemy

La clase Enemy representa a los enemigos que el jugador debe esquivar. Cada enemigo es una imagen que se mueve verticalmente por la pantalla. La clase maneja la posición, velocidad y colisiones con otros enemigos.

3.1.2 Clase Background

La clase Background maneja el desplazamiento del fondo del juego. Utiliza dos superficies que se mueven en bucle para crear una ilusión de movimiento continuo.

3.1.3 Clase Player

La clase Player representa al jugador. Maneja la carga de la imagen del jugador, la actualización de su posición y la rotación basada en la entrada del usuario.

3.1.4 Clase Game

La clase Game es la más compleja y contiene la lógica principal del juego:
Inicialización de Pygame y configuración de la pantalla.
Manejo de eventos y entrada del usuario.
Actualización de la lógica del juego y renderización de los gráficos.
Detección de colisiones y gestión del puntaje

.

3.1.5 Clase Webcam

La clase Webcam maneja la captura de video desde la cámara. Utiliza OpenCV para acceder a la cámara y proporciona el último fotograma capturado para ser procesado.

4. Explicación de los Algoritmos Utilizados

4.1 Algoritmo de Movimiento del Jugador

El movimiento del jugador se basa en la entrada del usuario a través del movimiento de la cabeza. La posición del jugador se actualiza cada frame según la velocidad y la dirección indicadas por el usuario. Además, se aplica una rotación a la imagen del jugador para simular el giro del automóvil.

4.2 Algoritmo de Generación de Enemigos

Los enemigos se generan aleatoriamente en la parte superior de la pantalla y se mueven hacia abajo. La frecuencia de generación y la velocidad de los enemigos aumenta con el tiempo para incrementar la dificultad del juego.

4.3 Algoritmo de Detección de Colisiones

Se utiliza la función pygame.sprite.spritecollide para detectar colisiones entre el jugador y los enemigos. Si se detecta una colisión, el juego se detiene y se muestra la pantalla de Game Over.

4.4 Algoritmo de Detección de Movimiento Facial

Se utiliza Mediapipe para detectar la posición de la cara del jugador. La posición se utiliza para calcular el movimiento del jugador en el juego, permitiendo controlar el automóvil con movimientos de la cabeza.

5. Documentación del Código

```
import pygame
import sys
from moviepy.editor import VideoFileClip
from pygame.locals import *
from threading import Thread
import cv2
import platform
```

```
import random
import os
import mediapipe as mp
import math
from datetime import datetime

# Define las limitaciones del área permitida en la pista del jugador
allowed_area = pygame.Rect(100, 300, 600, 200)

# Define las limitaciones del área permitida en la pista de los enemigos
enemy_allowed_area = pygame.Rect(400, 10, 500, 300)

# Definir un nuevo evento personalizado para agregar enemigos al juego
ADD_ENEMY = pygame.USEREVENT + 1

# Configurar la velocidad inicial del juego
game_speed = 1

# Tamaño de la pantalla
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600

# Tamaño original de las imágenes
ORIGINAL_SIZE = (272, 512)
WIDTH = 50
HEIGHT = (WIDTH / ORIGINAL_SIZE[0]) * ORIGINAL_SIZE[1]
SPEED = 1

# Directorio donde se encuentran las imágenes de los enemigos
ENEMY_DIRECTORY =
r'C:\Users\nesto\OneDrive\Escritorio\proyecto_algoritmos\sprites\corredore
s'

# Función principal
def main():
    # Reproducir el video con sonido antes de iniciar el juego
    play_video()

    # Mostrar el menú y esperar a que el usuario inicie el juego
    show_menu()
```

```
    # Iniciar el juego
    Game().loop()

# Función para reproducir el video de la intro
def play_video():
    pygame.display.set_caption("ChecoGame")
    # Ruta del archivo de video
    video_path =
r"C:\Users\nesto\OneDrive\Escritorio\proyecto_algoritmos\sprites\f1_intro.
mp4"

    # Reproducir el video con sonido
    video_clip = VideoFileClip(video_path)

    # Redimensionar el video al tamaño deseado y centrarlo en una ventana
de 800x600
    video_clip_resized = video_clip.resize((800, 600))
    video_clip_resized = video_clip_resized.set_position(('center',
'center'))

    # Reproducir el video con sonido
    video_clip_resized.preview()

# Función del menú
def show_menu():
    pygame.init()
    screen = pygame.display.set_mode((800, 600))
    pygame.display.set_caption("Menú")

    # Cargar imagen de fondo del menú
    background_image =
pygame.image.load(r"C:\Users\nesto\OneDrive\Escritorio\proyecto_algoritmos
\sprites\back_menu.jpg").convert()
    background_image = pygame.transform.scale(background_image, (800,
600))  # Redimensionar la imagen al tamaño de la pantalla

    # Cargar imagen del botón de inicio
```

```
    start_button_image =
pygame.image.load(r"C:\Users\nesto\OneDrive\Escritorio\proyecto_algoritmos
\sprites\boton.png").convert_alpha()
    start_button_image = pygame.transform.scale(start_button_image, (200,
100))  # Redimensionar el botón

    # Inicializar el módulo de sonido de pygame
    pygame.mixer.init()
    # Cargar el archivo de audio
    background_music =
pygame.mixer.Sound(r"C:\Users\nesto\OneDrive\Escritorio\proyecto_algoritmo
s\sprites\intro.wav")
    # Reproducir el audio en bucle
    background_music.play(-1)

    clock = pygame.time.Clock()

    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
            elif event.type == MOUSEBUTTONDOWN:
                # Verificar si el botón de inicio fue presionado
                if start_button_rect.collidepoint(event.pos):
                    background_music.stop()  # Detener la música antes de
salir del menú
                    return  # Salir de la función para iniciar el juego

        # Dibujar la imagen de fondo del menú
        screen.blit(background_image, (0, 0))

        # Obtener el rectángulo del botón de inicio
        start_button_rect = start_button_image.get_rect()
        # Centrar el botón de inicio en la parte inferior de la pantalla
        start_button_rect.center = (400, 550)
        # Dibujar el botón de inicio
        screen.blit(start_button_image, start_button_rect)

        pygame.display.flip()
```

```
        clock.tick(30)

# Clase enemigo
class Enemy(pygame.sprite.Sprite):
    def __init__(self, enemies_group):
        super(Enemy, self).__init__()

        self.enemies_group = enemies_group

        # Lista de nombres de archivos de imágenes de enemigos
        enemy_images = os.listdir(ENEMY_DIRECTORY)

        # Selecciona aleatoriamente una imagen de la lista
        image_name = random.choice(enemy_images)
        image_path = os.path.join(ENEMY_DIRECTORY, image_name)

        # Carga la imagen y la escala
        self.surf = pygame.image.load(image_path).convert_alpha()
        self.surf = pygame.transform.scale(self.surf, (70, 90))  # Escala
fija
        self.mask = pygame.mask.from_surface(self.surf)

        # Configura el rectángulo y la posición inicial
        self.rect = self.surf.get_rect(
            center=(
                random.randint(0, SCREEN_WIDTH),
                random.randint(-100, -20)
            )
        )
        # Limites de pista
        if not enemy_allowed_area.contains(self.rect):
            # Si el enemigo sale de esa área, lo reposiciona dentro de
ella
            self.rect.clamp_ip(enemy_allowed_area)

        # Velocidad semi-aleatoria, aumenta con la velocidad global del
juego
        self.speed = (random.randint(1, 2) / 10) * (1 + (game_speed / 2))
```

```
        self.rect.bottomleft = (random.randint(0, SCREEN_WIDTH -
self.rect.width), -self.rect.height)

    # Representa los movimientos de los enemigos en la pantalla
    def update(self, delta_time):
        # Movimiento hacia abajo
        self.rect.move_ip(0, self.speed * delta_time)
        self.mask = pygame.mask.from_surface(self.surf)

        # Verificación de colisiones con otros enemigos
        for enemy in self.enemies_group:
            if enemy != self and self.rect.colliderect(enemy.rect):
                # Si hay colisión con otro enemigo, ajusta la posición
vertical
                if self.speed > 0:
                    self.rect.bottom = min(enemy.rect.top,
self.rect.bottom) - 1
                else:
                    self.rect.top = max(enemy.rect.bottom, self.rect.top)
+ 1

        # Eliminación cuando sale de la pantalla
        if self.rect.top > SCREEN_HEIGHT:
            self.kill()

# Clase de fondo
class Background(pygame.sprite.Sprite):
    def __init__(self):
        super(Background, self).__init__()

        # Cargar la imagen del fondo de la pista
        self.surf =
pygame.image.load(r"C:\Users\nesto\OneDrive\Escritorio\proyecto_algoritmos
\sprites\pista.png")
        background_width = SCREEN_WIDTH  # doble ancho
        background_height = (background_width / self.surf.get_width()) *
self.surf.get_height()
        self.surf = pygame.transform.scale(self.surf, (background_width,
background_height))
        self.rect = self.surf.get_rect(
```

```
            bottomleft=(0, SCREEN_HEIGHT)
        )
        # Implementamos un bucle de fondo en el juego que se reposicionan
cuando una de ellas se mueve fuera de la pantalla
        # Usamos 2 surface para hacer un buen loop del fondo
        # Se mueven juntos y cuando uno desaparece de pantalla se mueve
        # hasta abajo, y así se van repitiendo
        self.surf2 = self.surf
        self.rect2 = self.surf2.get_rect(
            bottomleft=self.rect.topleft
        )
        self.ypos = 0
        self.ypos2 = background_height - SCREEN_HEIGHT

    # Gestiona el fondo de la pista
    def update(self, delta_time):
        self.ypos += .05 * delta_time
        self.ypos2 += .05 * delta_time
        self.rect.y = int(self.ypos)
        self.rect2.y = int(self.ypos2)
        if self.rect.y > SCREEN_HEIGHT:
            self.ypos = self.rect2.y - self.surf2.get_height()
            self.rect.y = self.ypos
        if self.rect2.y > SCREEN_HEIGHT:
            self.ypos2 = self.rect.y - self.surf.get_height()
            self.rect2.y = self.ypos2

    # Dibuja los fondos de la pista en la pantalla
    def render(self, dest):
        dest.blit(self.surf, self.rect)
        dest.blit(self.surf2, self.rect2)

# Clase jugador
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super(Player, self).__init__()

        # Cargar la imagen del jugador
```

```
        self.surf =
pygame.image.load(r"C:\Users\nesto\OneDrive\Escritorio\proyecto_algoritmos
\sprites\carrito.png").convert_alpha()
        self.surf = pygame.transform.scale(self.surf, (WIDTH, HEIGHT))
        self.update_mask()

        # Guardar para manejar bien las rotaciones
        self.original_surf = self.surf
        self.lastRotation = 0

        # Posicionamos el coche en el centro
        self.rect = self.surf.get_rect()
        self.rect.center = (SCREEN_WIDTH / 2, SCREEN_HEIGHT / 1.2)

        # Centroide del objeto
        self.update_centroid()

        # Guardar para el control del movimiento con MediaPipe
        self.reference_point = None

    def update_centroid(self):
        self.centroid_x = self.rect.x + self.rect.width // 2
        self.centroid_y = self.rect.y + self.rect.height // 2

    def update_mask(self):
        self.mask = pygame.mask.from_surface(self.surf)

    def update(self, key_pressed, delta_time):
        pressed_keys = pygame.key.get_pressed()

        if pressed_keys[K_LEFT]:
            self.rect.move_ip(-5, 0)
        if pressed_keys[K_RIGHT]:
            self.rect.move_ip(5, 0)
        if pressed_keys[K_UP]:
            self.rect.move_ip(0, -5)
        if pressed_keys[K_DOWN]:
            self.rect.move_ip(0, 5)

        # Limitar el movimiento del jugador al área permitida
```

```
        self.rect.clamp_ip(allowed_area)
        self.update_centroid()
        self.update_mask()

    def rotate(self, angle):
        # Ajustar la rotación
        if angle == 0:
            self.surf = self.original_surf
        else:
            self.surf = pygame.transform.rotate(self.original_surf, angle)
        self.update_mask()

    # Define la rotación del jugador con mediaPipe
    def rotate_with_mediapipe(self, face_landmarks):
        if self.reference_point is None:
            self.reference_point = (face_landmarks.landmark[5].x,
face_landmarks.landmark[5].y)

        current_point = (face_landmarks.landmark[5].x,
face_landmarks.landmark[5].y)
        dx = current_point[0] - self.reference_point[0]
        dy = current_point[1] - self.reference_point[1]

        # Convertir coordenadas de normalizadas a píxeles
        dx_pixels = dx * SCREEN_WIDTH
        dy_pixels = dy * SCREEN_HEIGHT

        angle = math.atan2(dy_pixels, dx_pixels) * (180 / math.pi)

        # Rotar el jugador
        self.rotate(angle)

# Clase del juego principal
class Game:
    def __init__(self):
        pygame.init()
        pygame.mixer.init()

        self.screen = pygame.display.set_mode((SCREEN_WIDTH,
SCREEN_HEIGHT))
```

```
        pygame.display.set_caption("ChecoGame")
        self.clock = pygame.time.Clock()

        self.background = Background()

        self.player = Player()
        self.enemies = pygame.sprite.Group()

        self.sprites = pygame.sprite.Group()
        self.sprites.add(self.player)

        self.face_mesh =
mp.solutions.face_mesh.FaceMesh(min_detection_confidence=0.5,
min_tracking_confidence=0.5)
        self.cap = cv2.VideoCapture(0)

        pygame.time.set_timer(ADD_ENEMY, 500)

    def loop(self):
        running = True
        while running:
            delta_time = self.clock.tick(60)  # Time between frames
            for event in pygame.event.get():
                if event.type == QUIT:
                    running = False
                elif event.type == ADD_ENEMY:
                    new_enemy = Enemy(self.enemies)
                    self.enemies.add(new_enemy)
                    self.sprites.add(new_enemy)

            key_pressed = pygame.key.get_pressed()
            self.player.update(key_pressed, delta_time)

            # Leer el frame desde la cámara
            ret, frame = self.cap.read()
            if not ret:
                print("No se puede leer el frame de la cámara")
                continue

            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

```
            results = self.face_mesh.process(frame_rgb)

            # Procesar resultados de MediaPipe
            if results.multi_face_landmarks:
                for face_landmarks in results.multi_face_landmarks:
                    self.player.rotate_with_mediapipe(face_landmarks)

            self.sprites.update(delta_time)
            self.background.update(delta_time)

            self.background.render(self.screen)
            for entity in self.sprites:
                self.screen.blit(entity.surf, entity.rect)

            pygame.display.flip()

        self.cap.release()
        pygame.quit()
        sys.exit()

if __name__ == "__main__":
    main()
```