Documentación para el Desarrollo del Backend del Juego

(Clases Player, Enemy y Movimiento)

Desarrollo de la lógica de negocio y algoritmos:

Player (Jugador):

- Inicialización del jugador: Al iniciar el juego, se crea una instancia de la clase **Player** que representa al jugador dentro del juego. Se definen sus atributos, como posición inicial y velocidad.
- Movimiento del jugador: El jugador puede moverse horizontalmente dentro del área de juego. Se implementa un algoritmo que actualiza la posición del jugador en función de la entrada del usuario.

Enemy (Enemigo):

- Generación de enemigos: Los enemigos aparecen aleatoriamente en la parte superior de la pantalla y se desplazan hacia abajo. Se implementa un mecanismo para generar nuevos enemigos en intervalos regulares.
- Comportamiento de los enemigos: Los enemigos se eliminan cuando salen de la pantalla o colisionan con el jugador. Se implementa un algoritmo de detección de colisiones para manejar estas situaciones.
- Velocidad de los enemigos: La velocidad de los enemigos aumenta progresivamente durante el juego para aumentar la dificultad. Se implementa un sistema para ajustar dinámicamente la velocidad de los enemigos.

Movimiento:

- Interpretación del movimiento: El movimiento de la cabeza se traduce en movimientos horizontales del jugador en el juego. Se implementa un sistema para convertir el ángulo de inclinación de la cabeza en una acción de movimiento del jugador.
- Suavizado del movimiento: Se implementa un algoritmo de suavizado para evitar movimientos bruscos del jugador en la pantalla. Esto mejora la experiencia del usuario al hacer que el movimiento del jugador sea más fluido.

Implementación de las Funciones de Procesamiento de Datos:

Inicialización del juego:

- Al iniciar el juego, se inicializan los componentes necesarios, incluyendo la detección de movimiento de la cabeza y la creación de instancias de las clases **Player** y

Enemy. Se establecen los parámetros iniciales del juego, como la velocidad y la posición inicial de los elementos.

Integración con la lógica del juego:

- Control del jugador: El movimiento detectado se utiliza para controlar el movimiento horizontal del jugador en el juego. Se actualiza la posición del jugador en función del ángulo de inclinación de la cabeza.
- Actualización en tiempo real: Las acciones de movimiento del jugador se actualizan en tiempo real según el movimiento detectado. Esto permite al jugador controlar el movimiento del jugador simplemente moviendo la cabeza.

Manejo de las Solicitudes y Respuestas del Cliente:

Solicitudes del Cliente:

- El cliente envía solicitudes al backend del juego para controlar el movimiento del jugador utilizando la detección de movimientos faciales. Se establece una conexión entre el cliente y el servidor para enviar y recibir datos.

Respuestas del Cliente:

- El backend procesa las imágenes capturadas por la cámara web del jugador para determinar el movimiento de la cabeza. Se identifican los landmarks faciales relevantes y se calcula el ángulo de inclinación de la cabeza.
- El movimiento detectado se traduce en acciones de movimiento del jugador en el juego. Se envían actualizaciones de la posición del jugador al cliente para reflejar los movimientos en la pantalla del juego.

Pruebas Unitarias y de Integración:

Pruebas Unitarias:

- Se diseñan casos de prueba para verificar la precisión del cálculo del movimiento de la cabeza. Se crean situaciones de prueba que cubren una variedad de escenarios, como movimientos de cabeza hacia la izquierda, hacia la derecha y movimientos neutros.
- Se desarrollan casos de prueba para evaluar el correcto funcionamiento del procesamiento de imágenes de la cámara web. Se simulan diferentes condiciones de iluminación y posiciones de la cabeza para verificar la robustez del sistema de detección facial.

Pruebas de Integración:

- Se realizan pruebas para verificar la sincronización adecuada entre los movimientos detectados y las acciones del jugador en el juego. Se asegura que el sistema responda de manera coherente y precisa a los movimientos faciales del jugador en tiempo real.
- Se comprueba que el sistema de detección de movimiento facial esté correctamente integrado con el resto del backend del juego. Se verifican las interfaces de comunicación entre los diferentes componentes del backend para garantizar una interacción fluida y eficiente.

Código:

```
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()



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

        self.surf = pygame.image.load(r"C:\Users\servidor\Desktop\PROYECTO
ALGORITMO\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 al jugador en el centro
        self.rect = self.surf.get_rect(
            center=(
                (SCREEN_WIDTH/2) - (WIDTH/2),
                (SCREEN_HEIGHT - HEIGHT)
            )
```

```
        )


    def update(self, movement, delta_time):
        self.rect.move_ip(SPEED * movement * delta_time, 0)


        # Restringir el movimiento dentro del área de la pista permitida
        if not allowed_area.contains(self.rect):
            self.rect.clamp_ip(allowed_area)


        # Rotación
        rotation = 45 * movement * -1
        self.surf = pygame.transform.rotate(self.original_surf, self.lerp(self.lastRotation,
rotation, .5))
        self.lastRotation = rotation
        self.update_mask()


        if self.rect.left < 0: self.rect.left = 0
        if self.rect.right > SCREEN_WIDTH: self.rect.right = SCREEN_WIDTH
        if self.rect.top <= 0: self.rect.top = 0
        if self.rect.bottom >= SCREEN_HEIGHT: self.rect.bottom = SCREEN_HEIGHT


    def lerp(self, a: float, b: float, t: float) -> float:
        return (1 - t) * a + t * b


    def update_mask(self):
        #Mascara tiene un 80% del tamano para 'perdonar' al jugador en ciertas colisiones
        maskSurface = self.surf
        maskSurface = pygame.transform.scale(maskSurface, (WIDTH*.8,HEIGHT*.8))
```

```
    self.mask = pygame.mask.from_surface(maskSurface)
```

Resultados de las Pruebas:

- Las pruebas unitarias y de integración confirman que el sistema funciona de manera consistente y confiable en una variedad de situaciones.
- Se valida que el juego proporciona una experiencia de juego inmersiva y satisfactoria, permitiendo al jugador controlar el movimiento del jugador de forma intuitiva mediante el movimiento de la cabeza.
- Estas pruebas son fundamentales para garantizar la calidad y la fiabilidad del sistema de detección de movimiento facial, asegurando que el juego pueda ser disfrutado sin problemas por los usuarios finales.

Ejecución Local del Juego

Requisitos del Sistema:

- Python 3.6 o superior instalado en el sistema.
- Bibliotecas necesarias: Pygame, MoviePy, OpenCV, MediaPipe, y otras bibliotecas mencionadas en el código.

Pasos para la Ejecución:

1. **Descargar el Código:**
    - Descarga el código fuente en tu sistema local.
2. **Instalar Dependencias:**
    - Asegúrate de tener todas las dependencias necesarias instaladas.
3. **Interactuar con el Juego:**
    - Una vez que el juego se inicie, podrás interactuar con él a través de los movimientos faciales detectados por la cámara web. Sigue las instrucciones en pantalla para jugar y disfrutar del juego.

Notas Adicionales:

- Asegúrate de proporcionar las rutas de archivo correctas para los recursos como imágenes, videos y archivos de audio en el código, especialmente si estás ejecutando el juego desde un directorio diferente.
- Si encuentras algún problema durante la ejecución, verifica los mensajes de error en la consola y asegúrate de haber seguido todos los pasos correctamente, incluyendo la instalación de las dependencias necesarias.
- El juego puede requerir permisos de acceso a la cámara web en tu sistema. Asegúrate de permitir el acceso cuando se te solicite para que el juego pueda utilizar la cámara correctamente.