Universidad Peruana de Ciencias Aplicadas
Facultad de Ingeniería

INFORME DE TRABAJO FINAL

1ASI0404-2510-2699– INTELIGENCIA ARTIFICIAL-
Carrera de Ciencias de la Computación

Integrantes:
Tejada Ramirez, Ricardo Martin
Meza Alfaro, Julio Cesar Guillermo
Núñez Del Arco, Ilan André
Romero Delgado, Cristopher Daniel

Sección: 2510-2699
Docente:
Javier Ulises Rosales Huamanchumo

Índice

1. Descripción del problema

En la actualidad, las personas con alguna discapacidad visual, específicamente las personas con visión reducida, necesitan de una buena accesibilidad en entornos físicos y virtuales. El entorno virtual más popular después de las redes sociales son los videojuegos, donde se estima que el 40% de la población mundial ha interactuado dentro este entorno en algún momento. Tal como se menciona:

> Según la OMS, se calcula que hay aproximadamente 253 millones de personas con limitaciones visuales y discapacidad visual en todo el planeta. Sin embargo, se sabe que aproximadamente el 40 por ciento de la población global dedica su tiempo libre a jugar. Esto implica que la cantidad de jugadores que tienen discapacidad visual es probablemente de varios millones (Cristian Suarez, 2023).

Por otro lado, la tecnología evoluciona dando pasos agigantados y acelerando. Este avance ha generado herramientas potenciadas con inteligencia artificial que se pueden aplicar en muchas industrias. La industria de los videojuegos es uno de los sectores económicos más importantes del mundo y aplica inteligencia artificial en una gran cantidad de sus productos. Pero a pesar de este avance, los entornos de juego rara vez consideran la accesibilidad desde la perspectiva de personas con discapacidad visual.

En el ámbito del Machine Learning, una de las áreas de investigación más desafiantes es el desarrollo de agentes capaces de detectar objetos o patrones eficazmente en entornos dinámicos e inciertos, basándose únicamente en información visual. Un caso particular de este reto es el diseño de algoritmos que ayuden a la detección de humanos como potenciales supervivientes entre escombros, luego de un desastre como puede ser un edificio colapsado o un alúd. Este escenario simula condiciones cercanas a la percepción humana, donde el agente no tiene acceso a información estructurada del entorno como son las coordenadas, sino que debe interpretar imágenes en tiempo real.

Teniendo en cuenta este desafío, se propone una solución innovadora para mejorar la accesibilidad del público con discapacidad visual, considerando como candidato al clásico videojuego DOOM para desarrollar dicho experimento, junto al entorno experimental VizDoom, ampliamente utilizado como plataforma de investigación para el aprendizaje reforzado visual de agentes. Nuestra propuesta es crear un asistente de juego llamado “Doom Helper” que al presionar un botón pueda tomar cierto control del juego para ayudar al jugador utilizando modelos de IA basados en computer vision.

2. **Descripción y visualización del conjunto de datos.**

   Origen de los datos: Las imágenes fueron generadas desde VizDoom mediante episodios en los que el jugador se mueve con controles manuales. Las capturas se realizaron cada 30 frames para evitar redundancia, se capturaron alrededor de 2100 imágenes durante el gameplay de todos los niveles de *Doom 2*, luego con la herramienta LabelImg se etiquetaron manualmente las imágenes definiendo bounding boxes para las clases "enemy", "pickups", "weapons" y "misc".

   Formato: Las imágenes están en formato PNG (RGB), tienen un tamaño 1024 x 768 píxeles. Se almacenan en una carpeta llamada "frames" que contiene las imágenes y un archivo formato .txt que contienen las etiquetas de las imágenes en un formato de índice de etiqueta y coordenadas de su bounding box, luego se divide el dataset siguiendo la estructura de entrenamiento de YOLOv5. En las carpetas images con los archivos png y labels para los archivos .txt, como estamos implementando dos modelos tenemos una copia de ambas carpetas llamadas images 2 y labels 2.

   En la imagen se pueden ver múltiples ejemplos de la supervisión manual realizada, donde los números representan:

   0: Enemy (Zombi, Espectro, Demonio, etc)

   1: Pickup (Botiquines de vida, Munición, etc)

   2: Weapon (Escopeta, BFG 9000, Motosierra, etc)

   2: Misc (Barriles explosivos)

3. **Propuesta**

   El objetivo del proyecto es **desarrollar un asistente inteligente para videojuegos**, específicamente en el entorno de VizDoom, que sea capaz de:

   - Detectar visualmente enemigos u objetos importantes en el entorno.
   - Permitir que el jugador pueda moverse de manera manual.

   Este sistema busca facilitar la accesibilidad para personas con discapacidad visual parcial, la función se podrá activar y desactivar mediante un botón cuando el usuario lo desee.

a. Descripción de la técnicas utilizadas para el desarrollo del trabajo:

YOLOv5:
Es un modelo de detección de objetos en tiempo real desarrollado por Ultralytics en 2020, conocido por su rapidez, precisión y facilidad de implementación. Utiliza una arquitectura basada en CSP-Darknet (backbone), FPN + PANet (neck) y un head que predice las cajas delimitadoras, clases y niveles de confianza. Realiza la detección en una sola pasada, lo que lo hace altamente eficiente para aplicaciones en tiempo real. Se entrena fácilmente mediante PyTorch, con soporte para exportación a ONNX, TensorRT y otros formatos. A diferencia de versiones posteriores como YOLOv6, YOLOv7 y YOLOv8, YOLOv5 no soporta segmentación de instancias ni clasificación, y usa una arquitectura más clásica.

Vizdoom:
Es un entorno de simulación basado en el videojuego clásico DOOM, utilizado para entrenar agentes de inteligencia artificial en tareas de visión por computadora y aprendizaje por refuerzo. Los agentes interactúan con el entorno desde una vista en primera persona, aprendiendo a moverse, disparar o recolectar objetos a partir de imágenes del juego. Es ligero, personalizable y permite probar algoritmos en escenarios del videojuego, lo que lo convierte en una herramienta muy útil para desarrollar y evaluar agentes inteligentes en entornos visuales.

Para este trabajo se siguió la **metodología CRISP-DM**:

Comprensión del negocio
▪ Definir el problema y los objetivos del proyecto desde la perspectiva del negocio.

Comprensión de los datos
▪ Recolectar, explorar y familiarizarse con los datos disponibles.

Preparación de los datos
▪ Limpiar, transformar y seleccionar los datos relevantes para el análisis.

Modelado
▪ Aplicar algoritmos de machine learning y ajustar los parámetros para obtener los mejores resultados.

Evaluación
▪ Verificar si el modelo cumple con los objetivos del negocio y si los resultados son válidos.

Despliegue
▪ Implementar la solución en el entorno real para que genere valor práctico.

Elección del modelo:
Se seleccionó el modelo YOLOv5 (You Only Look Once versión 5), específicamente la variante yolov5s, por su balance entre precisión y velocidad. Este modelo es ideal para tareas de detección en tiempo real como la identificación de enemigos, armas y objetos dentro de un entorno de videojuego. Además, la versión 5 de YOLO presenta una ventaja sobre las otras porque presenta el balance entre ser lo suficientemente liviano y tener la complejidad justa para cumplir nuestro objetivo.

Preparación de los datos (Data Preparation)

Organización del dataset:
Para estructurar correctamente el proceso de entrenamiento y evaluación, el dataset fue organizado en distintas carpetas según la función que cumplen:

- images/train: contiene las imágenes utilizadas para entrenar el modelo.
- images/val: almacena las imágenes reservadas para la validación durante el entrenamiento.
- images/test: incluye las imágenes empleadas para la evaluación final del desempeño del modelo.
- Esta división nos permite separar los datos destinados al entrenamiento, validación y las pruebas finales, asegurando así una evaluación objetiva y evitando un sobreajuste.

Las anotaciones correspondientes hechas en LabelImg a cada imagen se almacenan en carpetas las cuales son labels/train, labels/val, labels/test, etc, donde cada archivo especifica los objetos presentes en la imagen y su ubicación.

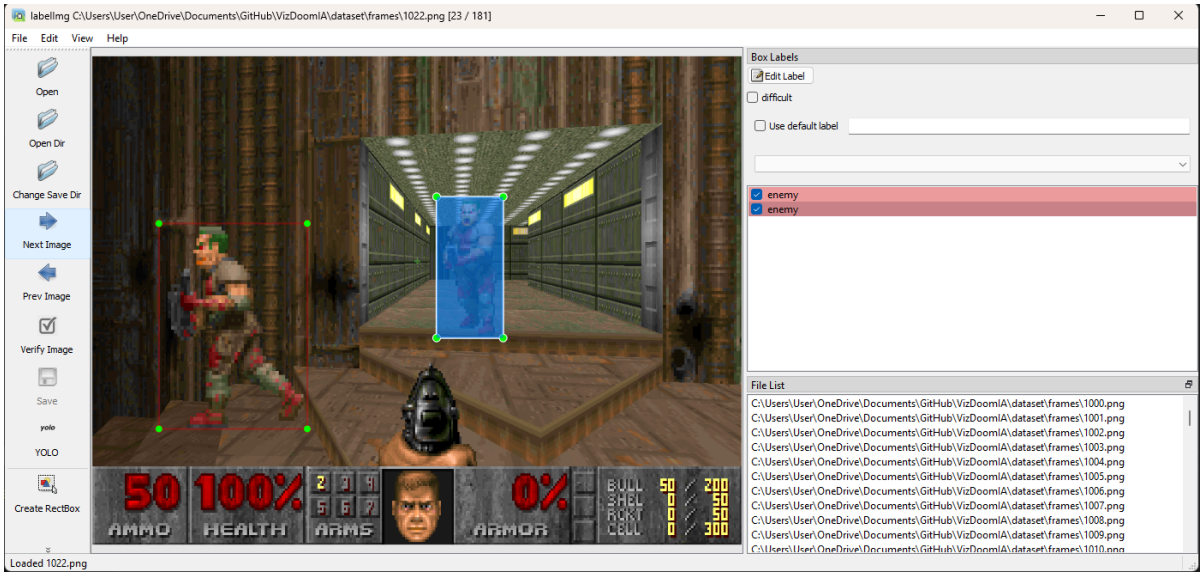


Imagen 1. Fuente: labelImg

Conversión al formato YOLO:

Todas las etiquetas se realizaron en formato YOLO, donde cada línea de los archivos de la etiqueta representa un objeto detectado. Este formato incluye la clase del objeto como enemigo o pickup y las coordenadas normalizadas de su bounding box, lo que facilita la integración directa con el modelo.

Archivo de configuración data.yaml:

Por último se elaboró el archivo de configuración data.yaml, en el cual se definió la ruta a las carpetas de imágenes y etiquetas, así como el nombre y orden de las clases.Este archivo garantiza que el proceso de entrenamiento utilice correctamente la estructura y la información del dataset.

Entrenamiento del modelo:

Configuramos un virtual environment de python 3.10 ya que ademas de ser compatible con VizDoom nos permite utilizar el repositorio de YOLOv5 para el entrenamiento. Utilizamos el script train.py de YOLOv5 (de Ultralytics), que permite entrenar modelos personalizados. A través la consola de visual studio desde nuestro virtual environment utilizamos el comando:

Para el modelo de detección de items, weapons y misc:

```
python train.py --img 416 --batch 16 --epochs 100 --data ../data.yaml
--weights yolov5s.pt --project runs/train --name doom_exp5
```

Para el modelo de detección de enemigos:

```
python train.py --img 416 --batch 16 --epochs 10 --data ../data.yaml
--weights yolov5s.pt --project runs/train --name doom_exp5
```

Argumentos del comando:

- img 416: redimensionamos las imágenes de entrada a 416 x 416 píxeles.
- batch 16: usa un tamaño de lote (batch size) de 16.
- epochs 100/10: entrena durante 100/10 épocas.
- data ../data.yaml: especifica la ruta al archivo de configuración del dataset (que contiene las rutas a las imágenes y clases).
- weights yolov5s.pt: parte del modelo preentrenado "yolov5s".
- project runs/train: guarda los resultados en la carpeta runs/train/.
- name doom_expxx: el nombre del experimento será doom_exp5/9.

Hiperparametros del Entrenamiento:
El archivo de hiperparámetros utilizado fue el predeterminado: hyp.scratch-low.yaml. A continuación se detallan los principales hiperparámetros y su propósito: Parametros base del yolo5

- lr0 = 0.01: Tasa de aprendizaje inicial alta para una convergencia más rápida.
- momentum = 0.937: Ayuda a estabilizar el descenso del gradiente.
- weight_decay = 0.0005: Regularización para evitar sobreajuste.
- warmup_epochs = 3.0: Fase inicial donde la tasa de aprendizaje sube gradualmente.
- box = 0.05: Peso de la pérdida asociada a la predicción de las cajas (bounding boxes).
- cls = 0.5: Pérdida asociada a la clasificación de la clase.
- obj = 1.0: Pérdida asociada a la presencia de un objeto en una celda.
- iou_t = 0.2: Umbral de IoU para considerar una predicción válida.

- fliplr = 0.5: Probabilidad de aplicar volteo horizontal a la imagen (aumento de datos).

- scale, translate = 0.5, 0.1: Aumentaciones para mejorar la generalización del modelo.

Resultados del modelo:

Modelo de detección de ítems, armas y barriles después de 100 épocas:

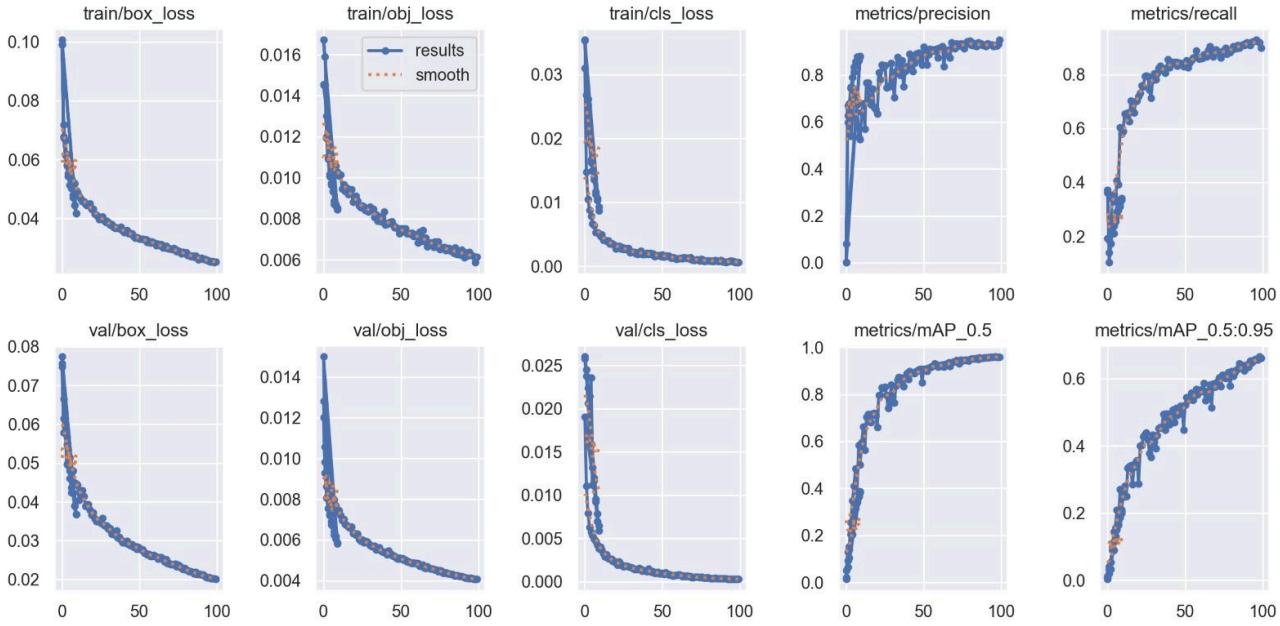


Imagen 2. Fuente: creación propia

Precisión: Llega hasta 0.88 por lo que puede concluir que el modelo cometa pocos falsos positivos.
Recall: Llega hasta 0.85 detecta la mayoría de los objetos con pocos falsos negativos.
mAP@0.5: Llega hasta 0.90 y tiene buena calidad en las detecciones con un IoU razonable (50%).
mAP@0.5:0.95: Llega hasta 0.75 y mantiene buena precisión incluso en condiciones más exigentes.

Modelos de detección de enemigos después de 10 épocas:

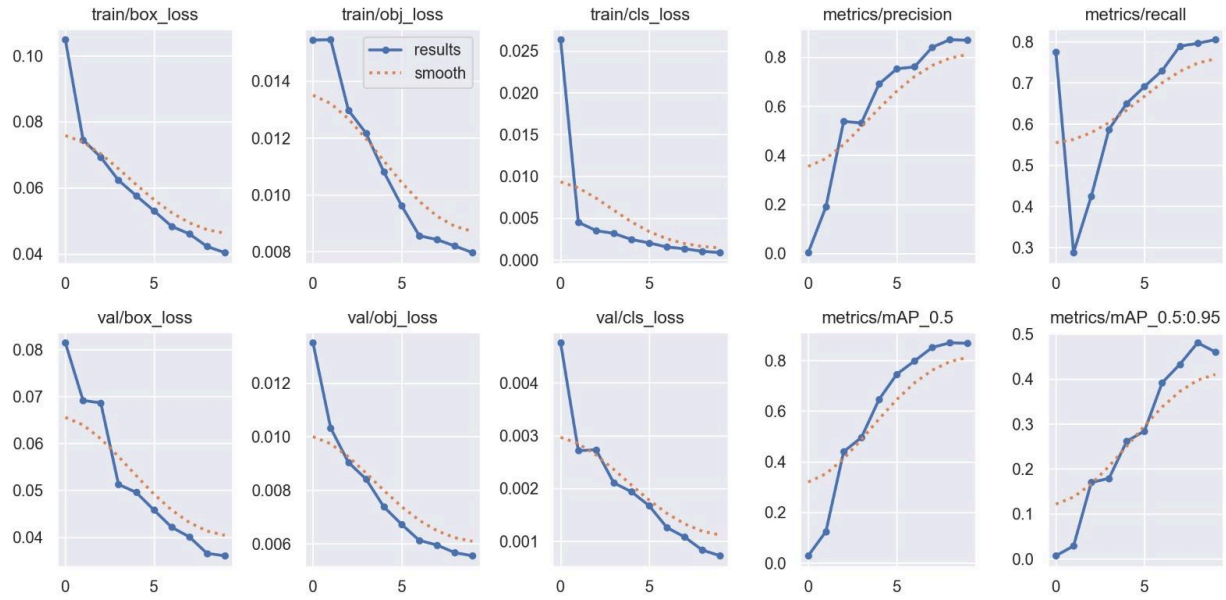


Imagen 3. Fuente: creación propia

Precisión: Llega hasta 0.95 por lo que puede concluir que el modelo cometa pocos falsos positivos.
Recall: Llega hasta 0.75 detecta la mayoría de los objetos con con algunos falsos negativos.
mAP@0.5: Llega hasta 0.85 y tiene calidad decente en las detecciones con un IoU
mAP@0.5:0.95: Llega hasta 0.65 y mantiene precisión decente incluso en condiciones más exigentes.

4. Diseño del aplicativo

Interfaz de usuario:
Teniendo en cuenta nuestro segmento objetivo, diseñamos un sistema simple para nuestros usuarios.

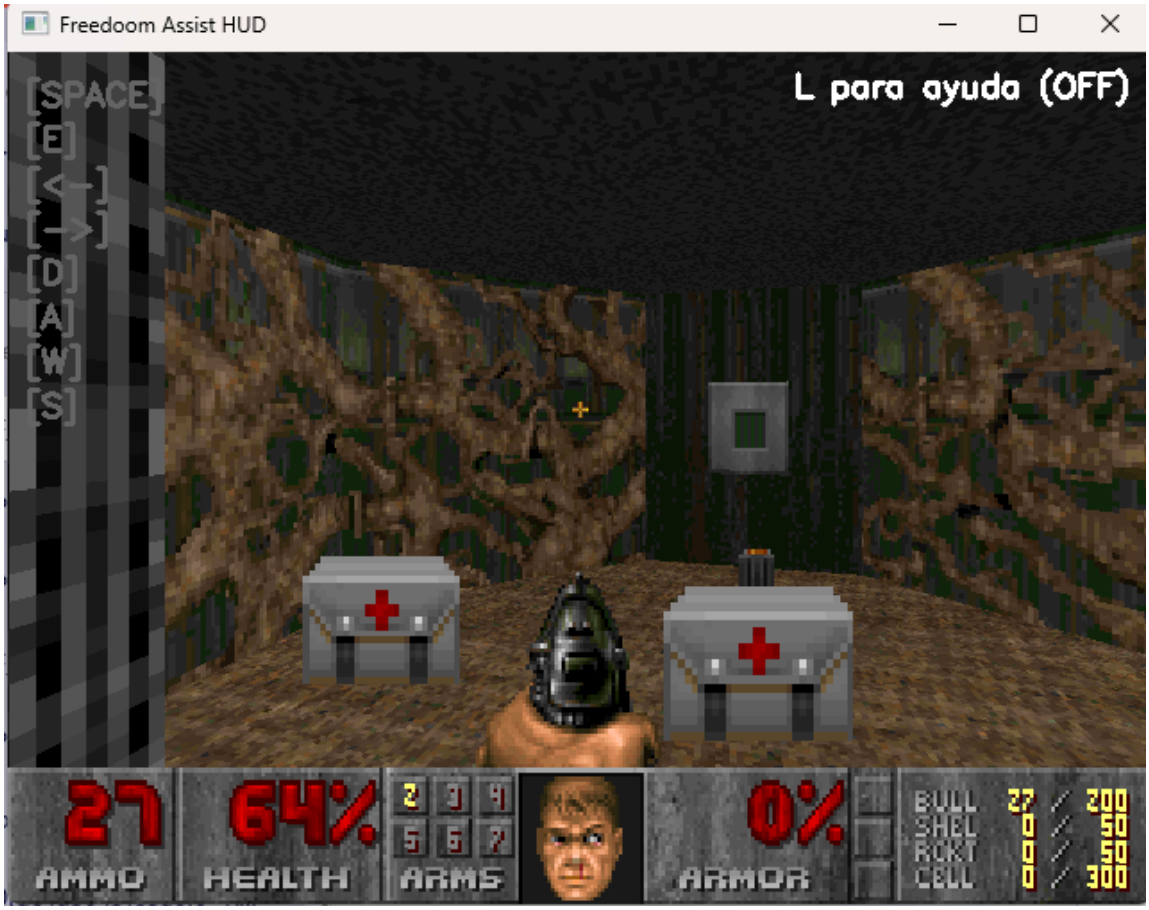


Imagen 4. Fuente: creación propia

El jugador va a poder experimentar el gameplay normalmente con los mismos controles del juego original, pero tendrá un pop-up en su pantalla diciendo “L para ayuda” para activar. Una vez el usuario presione esta tecla nuestra interfaz aparecerá.

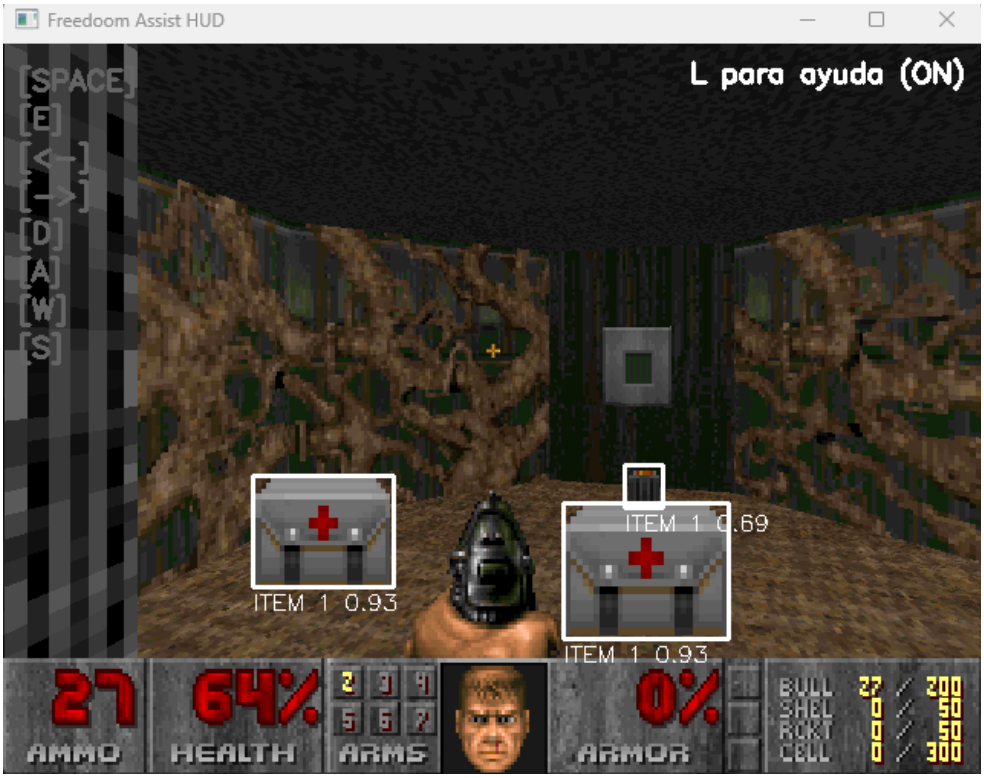


Imagen 5. Fuente: creación propia

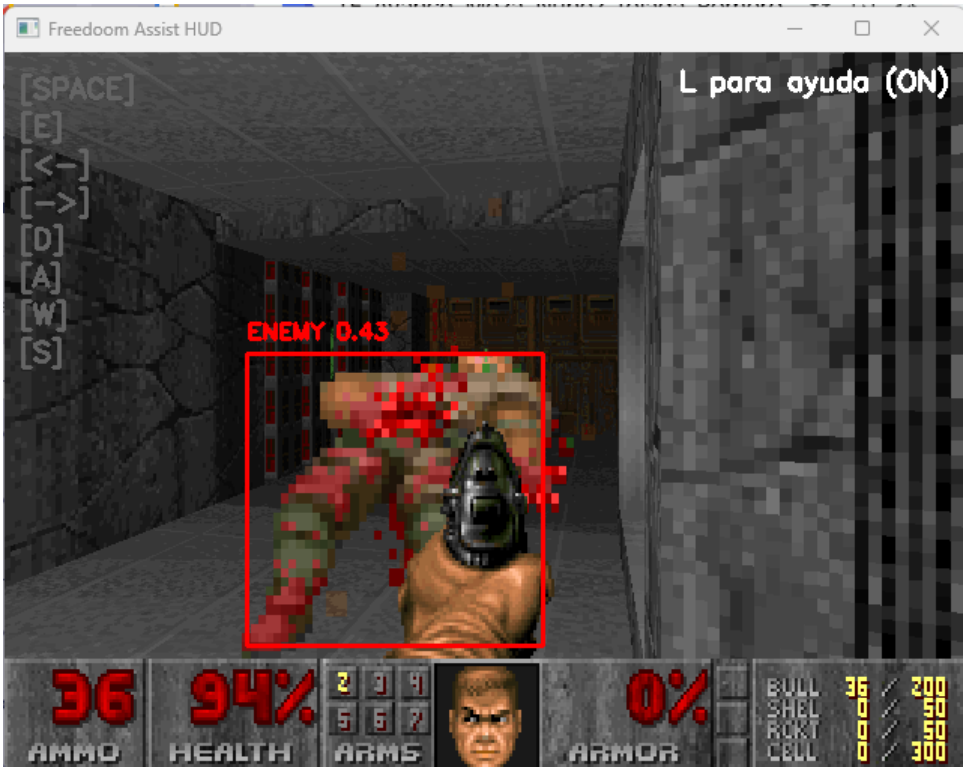


Imagen 6. Fuente: creación propia

En esta interfaz se podrá ver los controles del usuario y un bounding box que resalte los enemigos, los pickups, las armas y los barriles explosivos. Además, los controles cambiarán y a través de nuestro modelo se tendrá apuntado y disparo automático hacia los enemigos.

Implementacion:

Nuestros modelos entrenados están almacenados en la carpeta VizDoomIA\yolov5\runs\train\ desde esta carpeta cargamos nuestros dos modelos doom_exp5 para la identificación de los pickups y doom_exp9 para la identificación de los enemigos. Luego cargamos los modelos y a través de Vizdoom configuramos las variables de nuestro entorno de juego.

```
# === Configuración de modelos ===
MODEL_ITEMS_PATH = os.path.join(YOLOV5_PATH, 'runs', 'train', 'doom_exp5', 'weights', 'best.pt')
MODEL_ENEMIES_PATH = os.path.join(YOLOV5_PATH, 'runs', 'train', 'doom_exp9', 'weights', 'best.pt')

DEVICE = select_device('cpu')
IMG_SIZE = 416

model_items = attempt_load(MODEL_ITEMS_PATH, device=DEVICE)
model_enemies = attempt_load(MODEL_ENEMIES_PATH, device=DEVICE)
model_items.eval()
model_enemies.eval()
print("Modelos YOLOv5 cargados correctamente")

# === Parámetros generales ===
SCREEN_WIDTH = 640
CENTER_X = SCREEN_WIDTH // 2
CENTER_TOLERANCE = 30
TARGET_FPS = 144
FRAME_DURATION = 1.0 / TARGET_FPS
```

Luego cargamos el juego utilizando el archivo DOOM2.WAD que contiene las texturas del juego Doom II original e inicializamos el programa.

```python
# === Inicializar VizDoom con nivel ===
def init_game(map_name):
    game = DoomGame()
    game.load_config(os.path.join(BASE_DIR, "doom1.cfg"))
    game.set_doom_scenario_path(os.path.join(BASE_DIR, "DOOM2.WAD"))
    game.set_doom_map(map_name)
    game.set_window_visible(True)
    game.set_mode(Mode.ASYNC_PLAYER)
    game.add_game_args("+snd_volume 1")
    game.add_game_args("+snd_sfxvolume 1")
    game.set_screen_resolution(ScreenResolution.RES_640X480)
    game.init()
    game.new_episode()
    print(f" Iniciando mapa: {map_name}")
    return game
```

Luego tenemos el loop principal donde validamos las teclas del usuario así como la carga de los mapas.

```python
while True:
    if game.is_episode_finished():
        map_index += 1
        try:
            game.close()
            time.sleep(0.5)
            game = init_game(f"map{map_index:02d}")
        except Exception as e:
            print(f" No se pudo cargar el mapa map{map_index:02d}: {e}")
            map_index = 1
            game = init_game("map01")

    actions = [0] * 9
    if "space" in pressed_keys: actions[0] = 1
    if "e"     in pressed_keys: actions[1] = 1
    if "left"  in pressed_keys: actions[2] = 1
    if "right" in pressed_keys: actions[3] = 1
    if "d"     in pressed_keys: actions[4] = 1
    if "a"     in pressed_keys: actions[5] = 1
    if "w"     in pressed_keys: actions[6] = 1
    if "s"     in pressed_keys: actions[7] = 1
    if "f"     in pressed_keys: actions[8] = 1
```

Se utilizó **YOLOv5s** como base por su eficiencia en CPU y tamaño reducido.

```
if state and state.screen_buffer is not None and state.screen_buffer.shape[0] == 3:
    frame = state.screen_buffer.transpose(1, 2, 0)
    img = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

    # === YOLOv5 detección con letterbox ===
    img_letterboxed = letterbox(img, new_shape=IMG_SIZE)[0]
    img_tensor = torch.from_numpy(img_letterboxed).permute(2, 0, 1).float().to(DEVICE) / 255.0
    img_tensor = img_tensor.unsqueeze(0)

    with torch.no_grad():
        pred = model(img_tensor)[0]
        detections = non_max_suppression(pred, conf_thres=0.5)[0]

    manually_turning = "left" in pressed_keys or "right" in pressed_keys
    manually_attacking = "space" in pressed_keys

    closest_detection = None
    closest_offset = float('inf')

    if detections is not None and len(detections):
        detections[:, :4] = scale_boxes(img_letterboxed.shape[:2], detections[:, :4], img.shape).round()

        for *xyxy, conf, cls in detections:
            x1, y1, x2, y2 = map(int, xyxy)
            center_x = (x1 + x2) // 2
            offset = abs(center_x - CENTER_X)
            class_id = int(cls.item())
            label = f"{class_id} {conf:.2f}"

            # Asignar colores
            if class_id == 0:  # enemigo
                color = (0, 255, 0)
                if offset < closest_o  (variable) offset: int
                    closest_offset = offset
                    closest_detection = (x1, y1, x2, y2, center_x)
```

Finalmente, utilizamos el buffer de VizDoom para pasarle la imagen del frame actual a nuestros modelos para predecir y dibujar su bounding box.

```
if closest_detection:
    x1, y1, x2, y2, center_x = closest_detection
    offset = center_x - CENTER_X

    if not manually_turning:
        if abs(offset) > CENTER_TOLERANCE:
            actions[2] = int(offset < 0)
            actions[3] = int(offset > 0)

    if abs(offset) <= CENTER_TOLERANCE and not manually_attacking:
        actions[0] = 1  # disparar

draw_pressed_keys(img)
cv2.imshow("Freedoom AutoAim HUD", img)
if cv2.waitKey(1) == 27: break
```

Además, para los enemigos calculamos su posición en pantalla y aplicamos acciones simulando al jugador para que la cámara busque estar centrado con el enemigo y que

dispare una vez el enemigo esté lo suficientemente en el centro. Finalmente se dibujan las teclas presionadas y nuestra interfaz de usuario.

5. Validación de resultados y pruebas

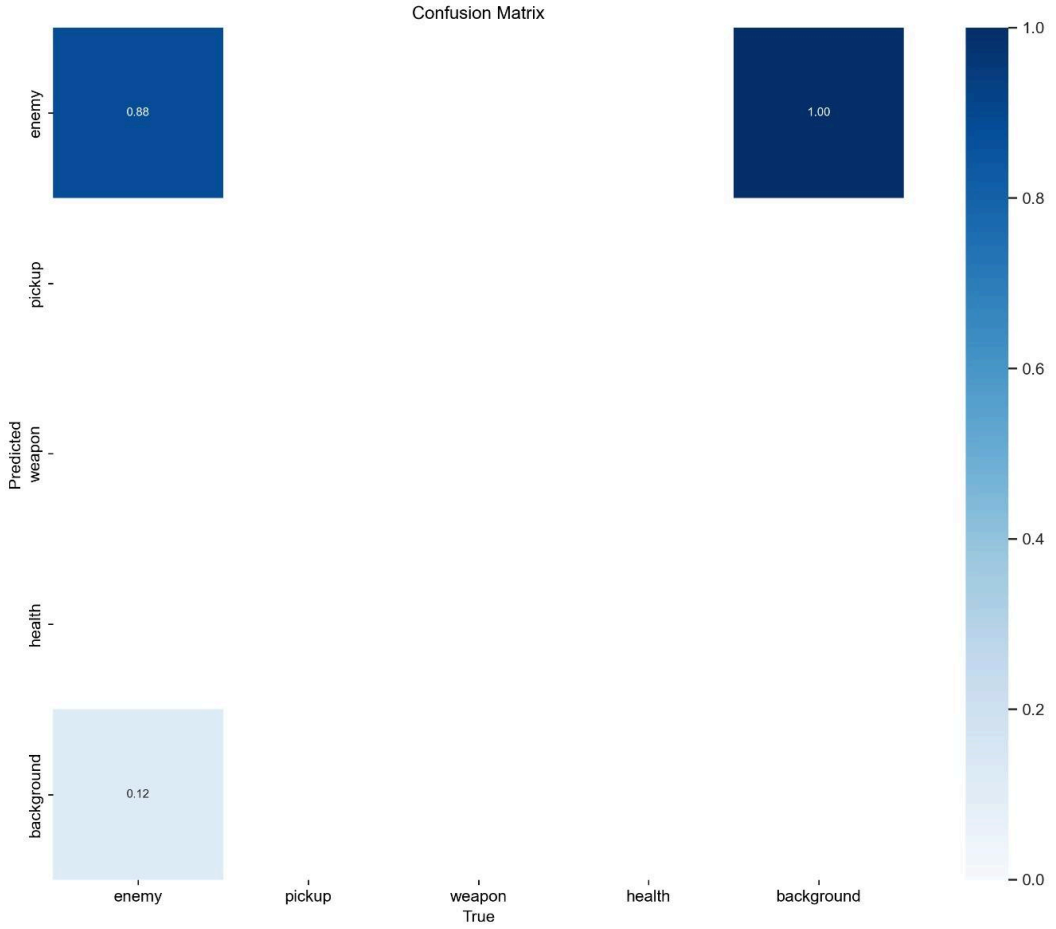
Video de prueba de funcionalidades: https://youtu.be/bgbbelQBRos

METRICAS:

- **Matriz de confusión:** Cuántas predicciones fueron correctas o incorrectas para cada clase. Diagonal = predicciones correctas (True Positives). Fuera de la diagonal = errores (False Positives o False Negatives).
- **F1 Confidence Curve:** La F1-score (promedio armónico de precisión y recall) en función del umbral de confianza. Ayuda a determinar a qué nivel de confianza (confidence threshold) el modelo logra su mejor equilibrio entre precisión y recall. Pico máximo = mejor compromiso entre ambos.
- **Labels:** Histograma de la cantidad de objetos por clase (etiquetas reales del dataset). Permite ver el desbalance de clases.
- **Labels Correlogram:** Correlación entre clases que aparecen juntas en una misma imagen. Mapa de calor donde celdas más claras indican mayor concurrencia de esas clases.
- **Precisión-Confidence Curve:** Cómo varía la precisión a medida que se aumenta el umbral de confianza. Una curva que cae indica que al ser más exigente (alta confianza), se cometen menos errores (más precisión), pero también se pierden más detecciones.
- **Precision-Recall Curve:** Relación entre precisión y recall para distintos umbrales. Idealmente, la curva debe estar cerca de la esquina superior derecha.
- **Recall-Confidence Curve:** Cómo cambia el recall (tasa de verdaderos positivos) en función del umbral de confianza. A mayor confianza, el recall suele bajar (el modelo se vuelve más estricto).

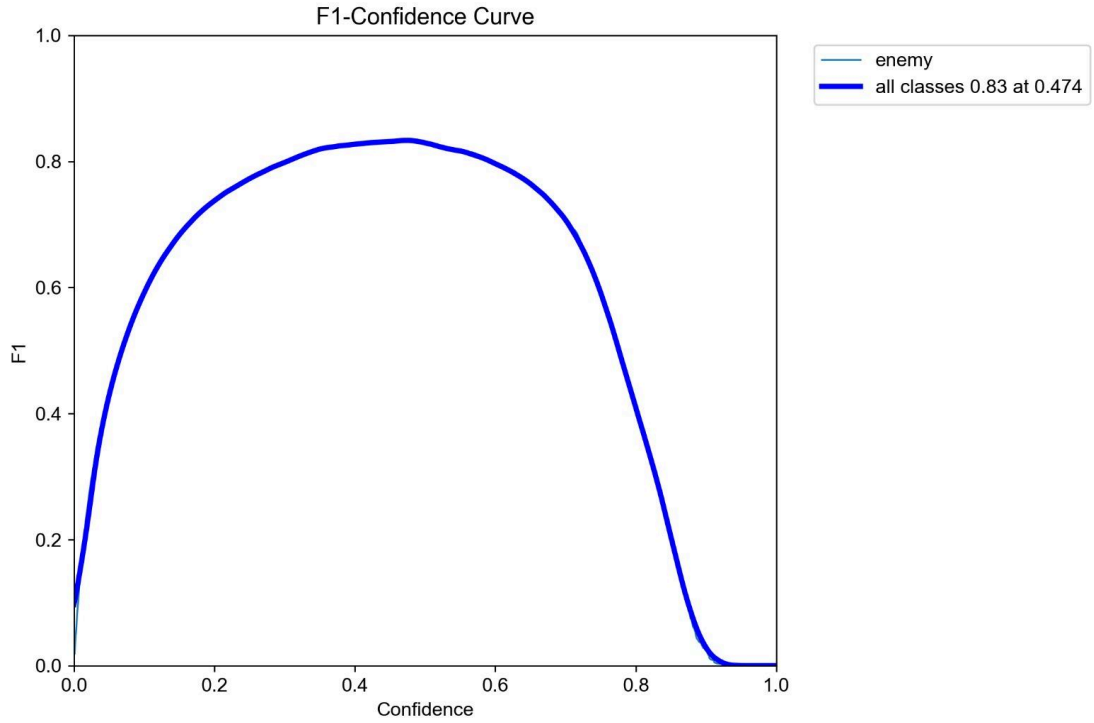
Modelo de enemigos:

Matriz de confusion:



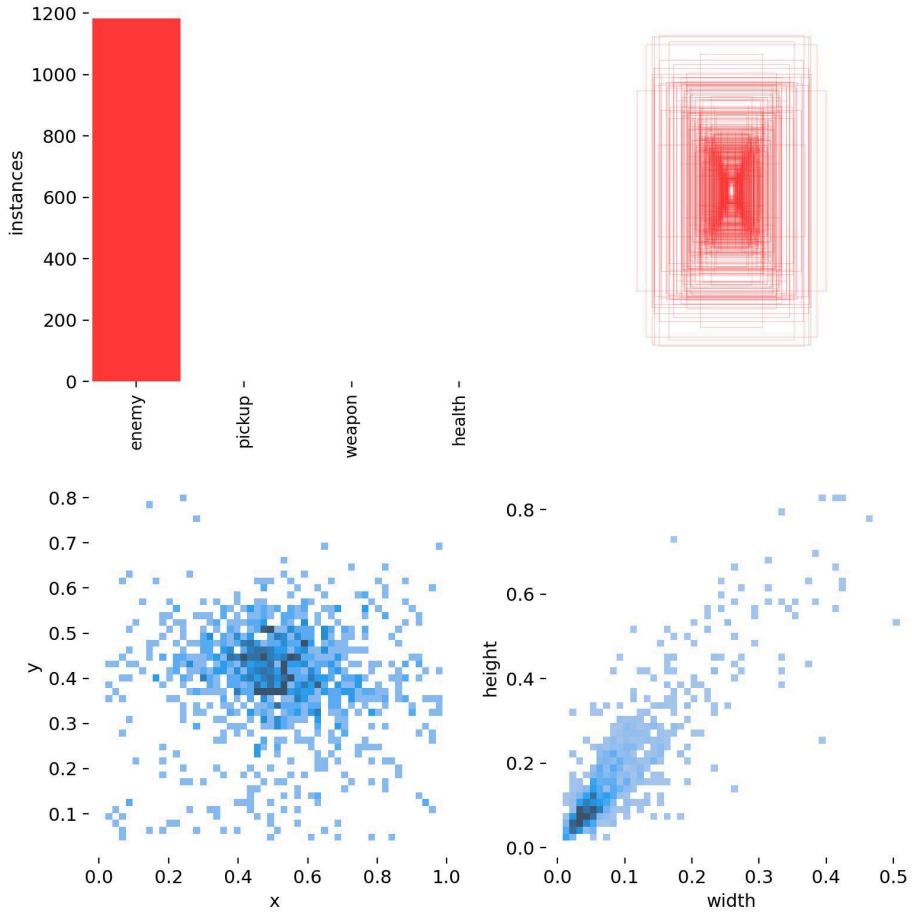
Interpretación: Las predicciones de “enemy” fueron correctas en su mayoría (0.88) pero se tuvieron falsos positivos etiquetados como background.

F1 curva:



Interpretación: El mejor equilibrio de precisión y recall que tuvo la clase “enemy” con las demás clases fue entre 0.4 y 0.6 en nivel de confianza.

Labels:



Interpretación:

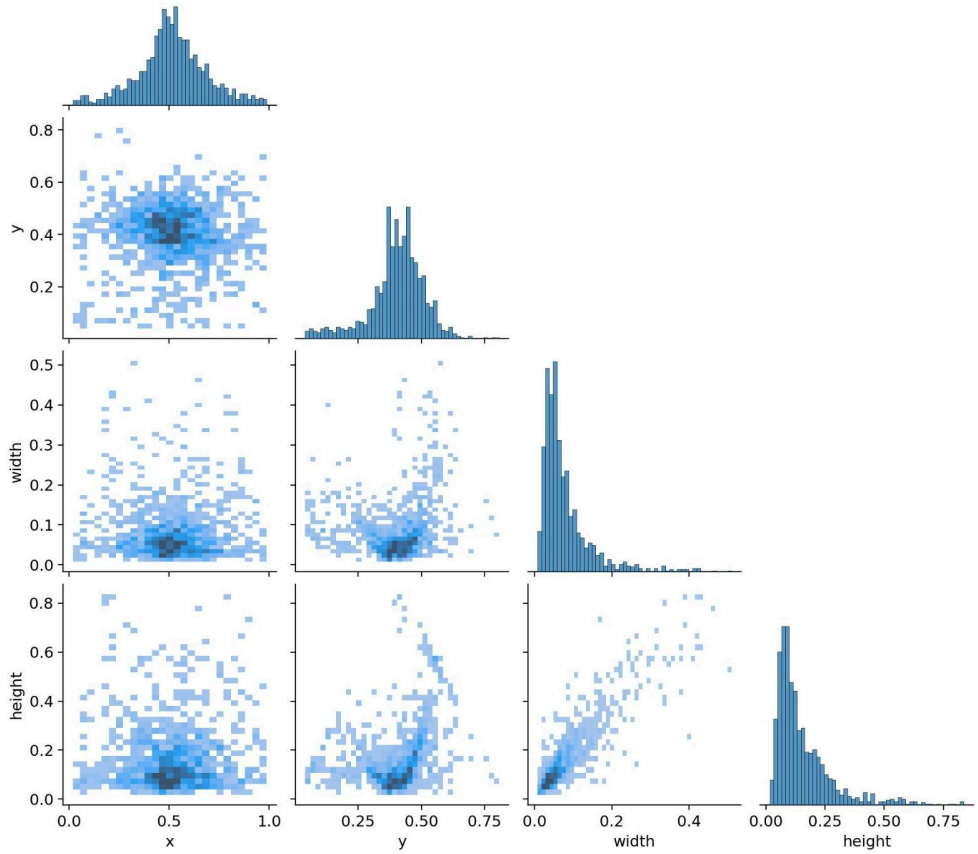
Histograma de clases: Hay una clara desproporción en la cantidad de instancias por clase.

Distribución de bounding boxes: Los objetos anotados (en su mayoría enemigos) están generalmente centrados en la pantalla. Esto sugiere que los enemigos tienden a aparecer en una zona central del campo visual del agente de VizDoom.

Distribución espacial X-Y de los objetos: La mayoría de los objetos aparecen entre x = 0.4 y 0.6 y y = 0.3 y 0.6, es decir, cerca del centro de la imagen.

Distribución de tamaños: La mayoría de los objetos tienen un tamaño pequeño a medio.

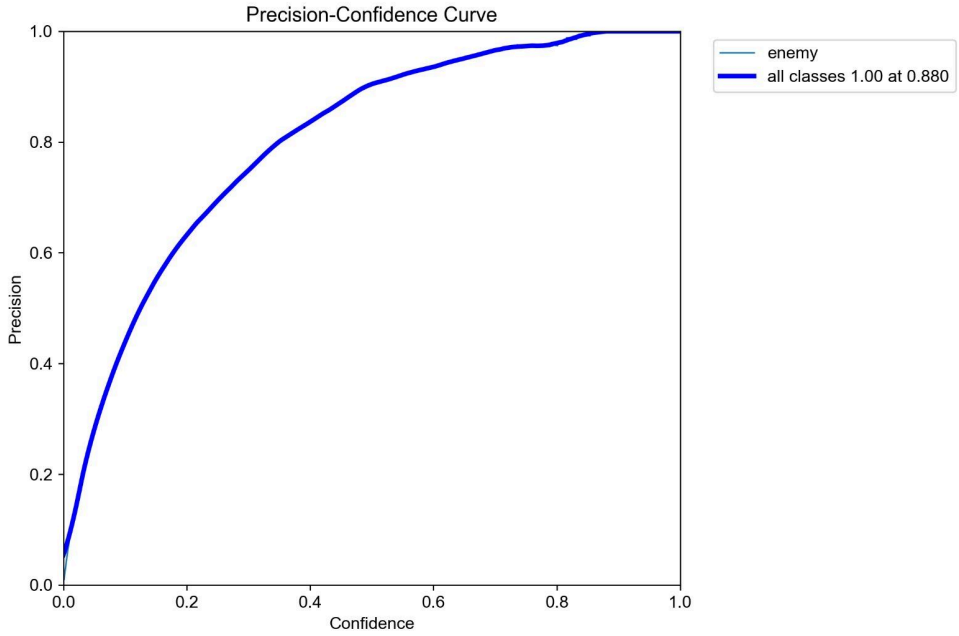
Labels correlogram:



Interpretacion:

Nuestros objetos aparecen centrados en la imagen (x ≈ 0.5, y ≈ 0.4). Varían los tamaños entre pequeños y medianos. Hay una relación directa entre ancho y alto de los objetos (bounding boxes).

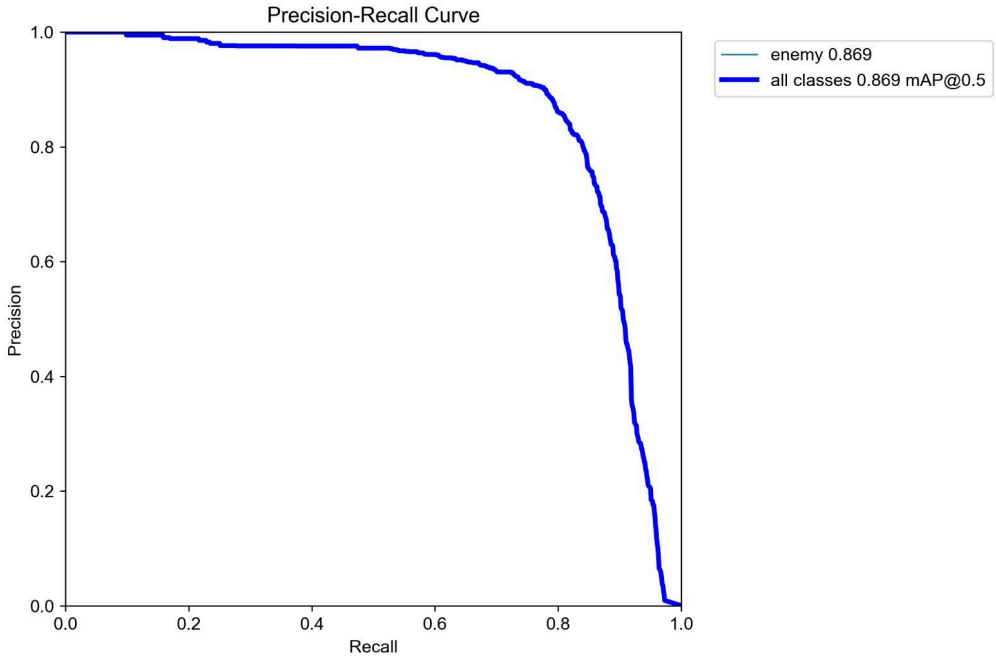
P_curve:



Interpretacion:

La clase enemy muestra tener más capacidad de detección y precisión a medida que aumenta el umbral de confianza en general.

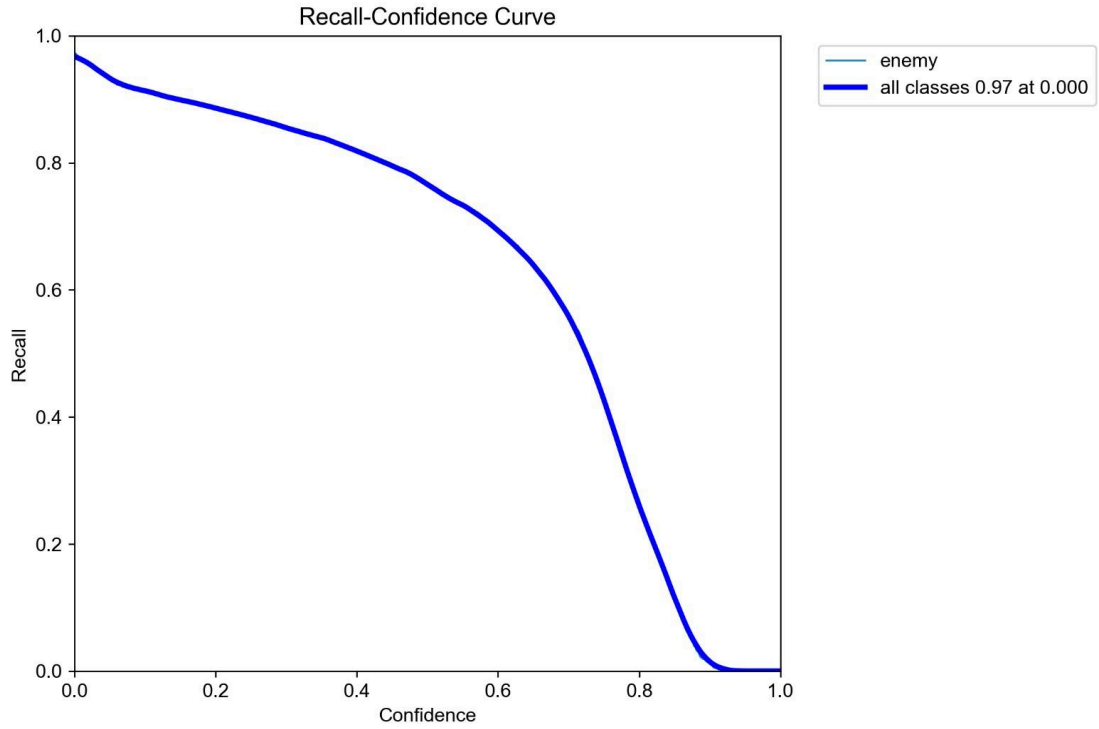
PR curve:



Interpretacion:

Las clases empiezan a tener un mal equilibrio recall-precision a cuando el recall (cobertura de detecciones) pasa por el valor 0.88.

R?curve



Interpretación: La clase enemy empieza a tener menos verdaderos positivos mayormente a partir del nivel de confianza 0.6.

Training Batch:

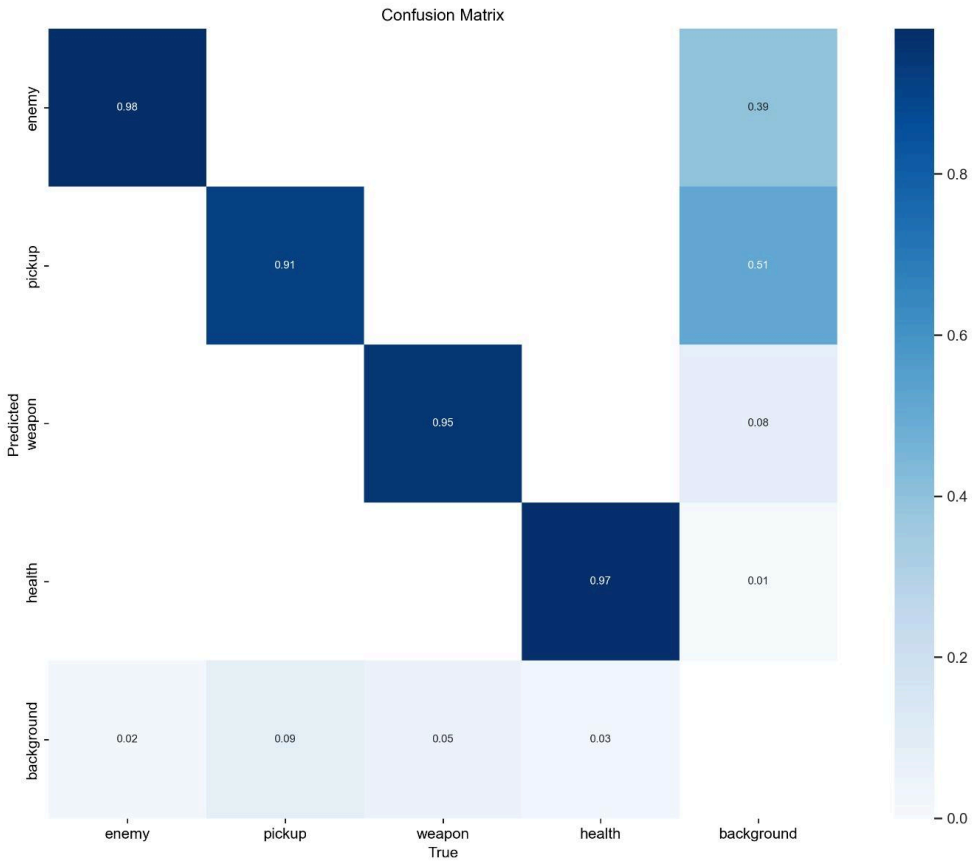
Interpretación: Ejemplos de imágenes para el entrenamiento.

Validation y prediction:

Interpretación: Las predicciones en el resultado final son muy prometedoras, la mayoría mostrando un alto nivel de predicción y seguridad.

Modelo de Items, armas y barriles explosivos:

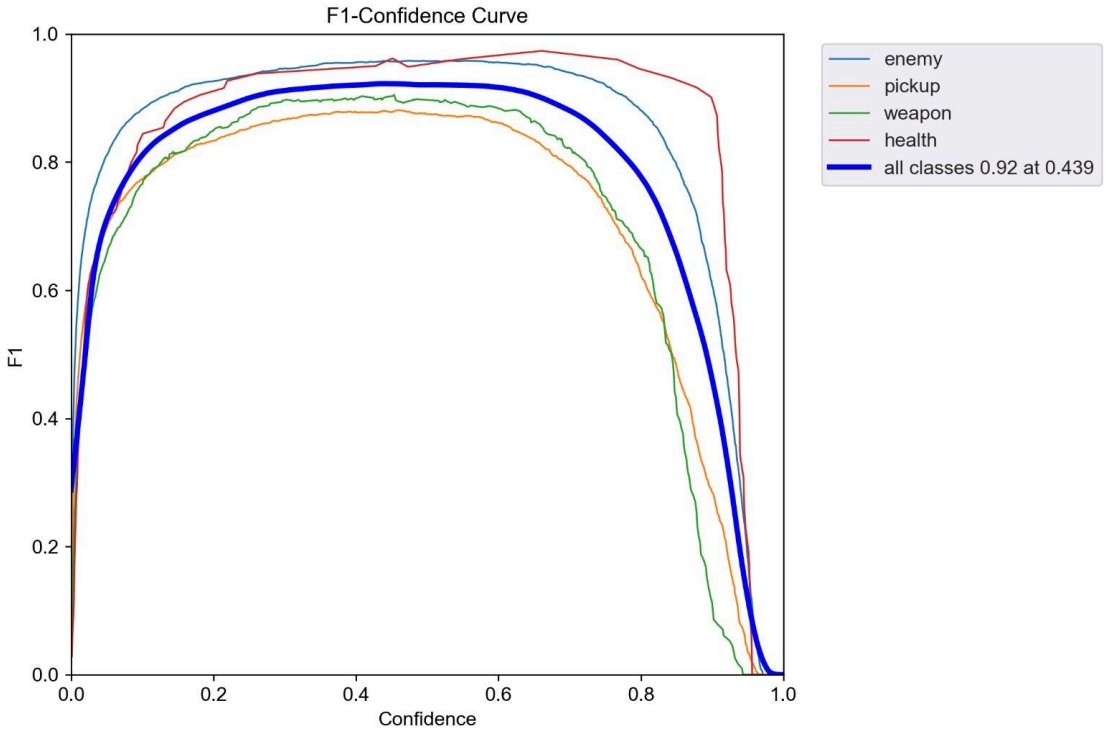
Matriz de confusión:



Interpretación:

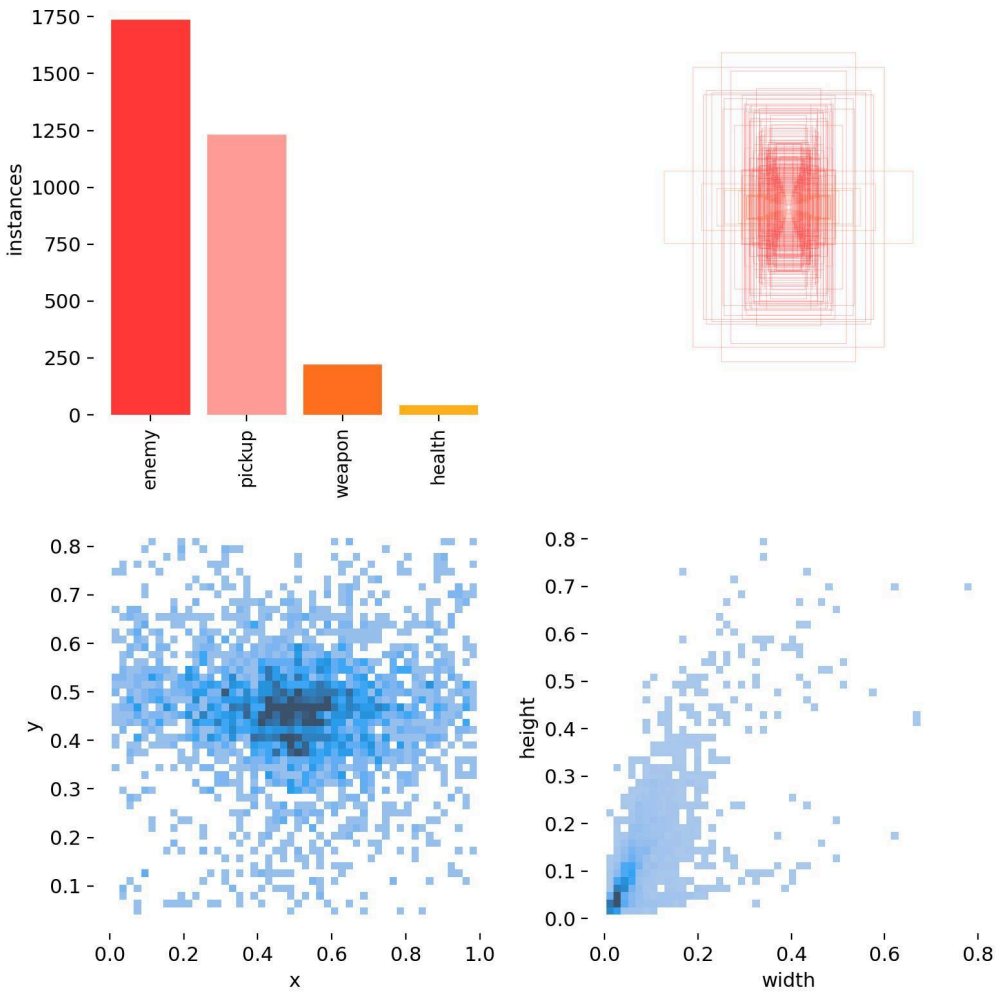
Todas las clases presentan una gran precisión a la hora de predecir, habiendo muy pocos falsos positivos o falsos negativos.

F1 curva



Interpretación: En promedio todas las clases alcanzan su mejor equilibrio Precision-Recall en el nivel de confianza 0.5, luego tiene un declive ya que el umbral se vuelve más estricto.

Labels



Interpretacion:

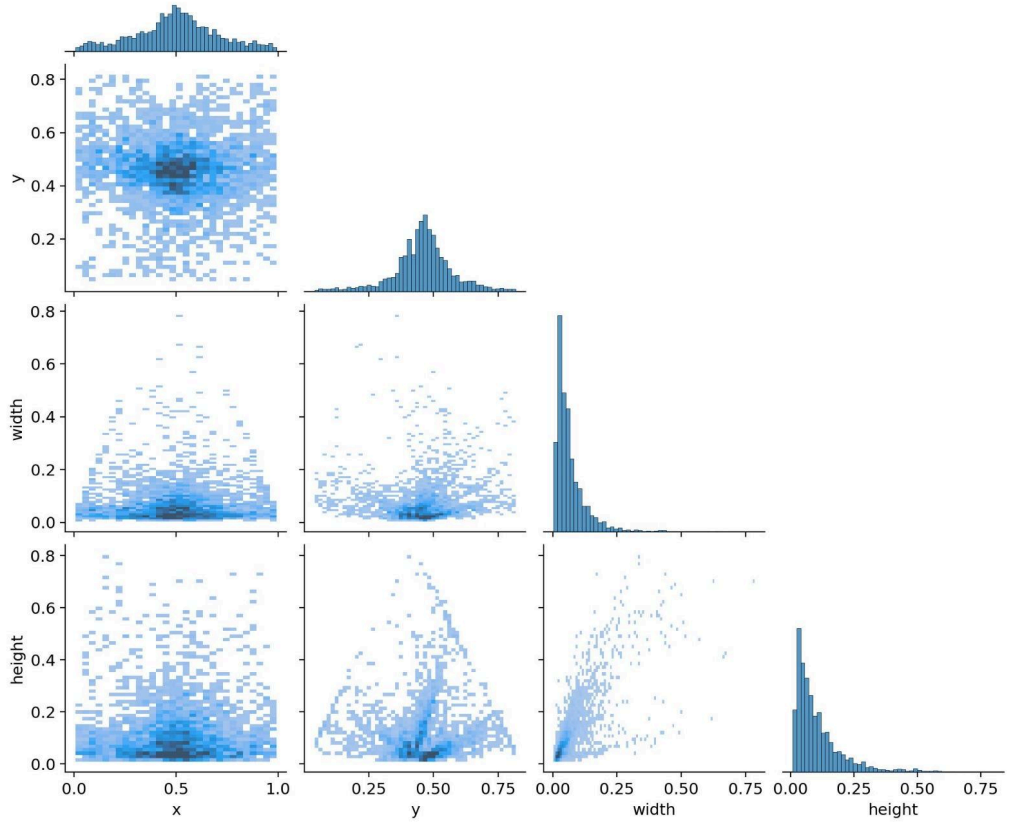
Distribución por clase: El dataset está desbalanceado. Las clases enemy y pickup tienen muchas más instancias que weapon or health, lo que podría afectar el desempeño del modelo.

Distribución de cajas: Se muestran las ubicaciones y tamaños relativos de las cajas, la mayoría están en el frente, lo cual es normal ya que en Doom es lo habitual.

Heatmap de coordenadas x vs y: los objetos (enemigos, pickups, etc.) están mayormente en el centro de la pantalla.

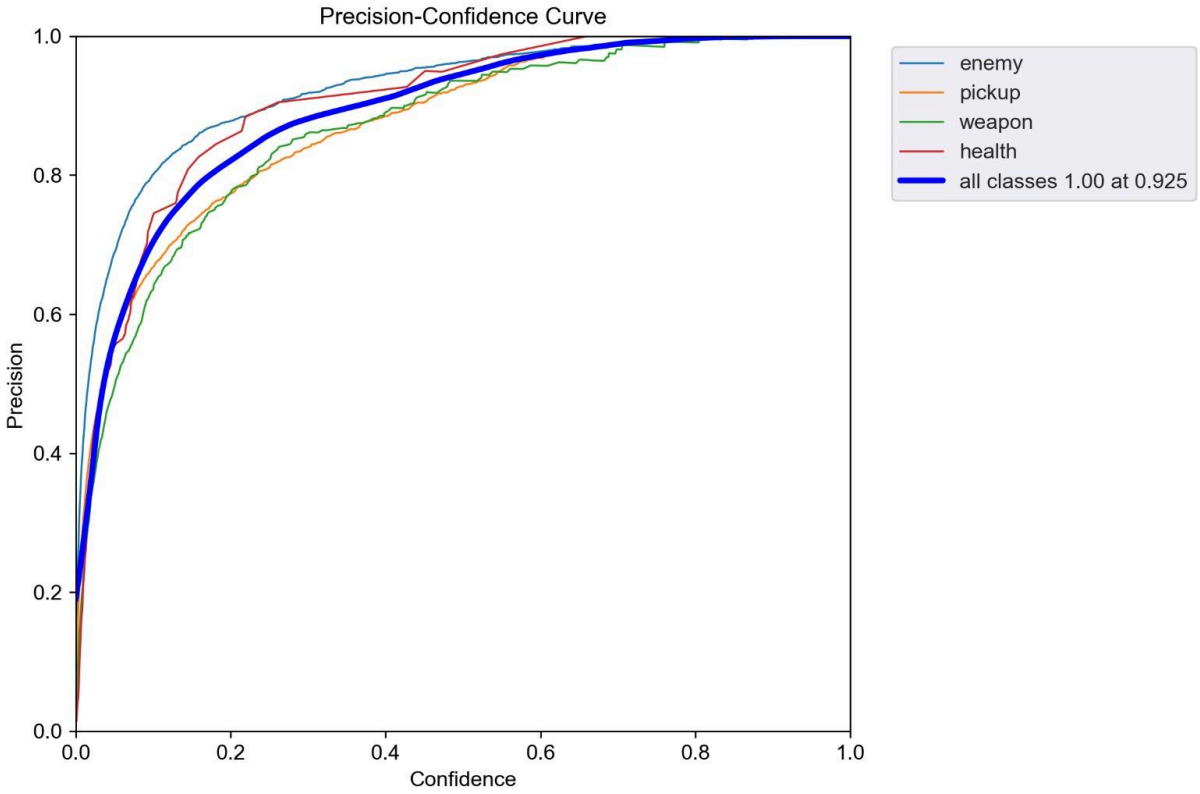
Width y height: La mayoría de los objetos son pequeños (width y height < 0.2), pero algunos son más grandes.

Labels correlogram:



Interpretación: Nuestros objetos aparecen centrados o abajo en la imagen. Varían los tamaños entre grandes y medianos. Hay una relación directa entre ancho y alto de los objetos (bounding boxes).

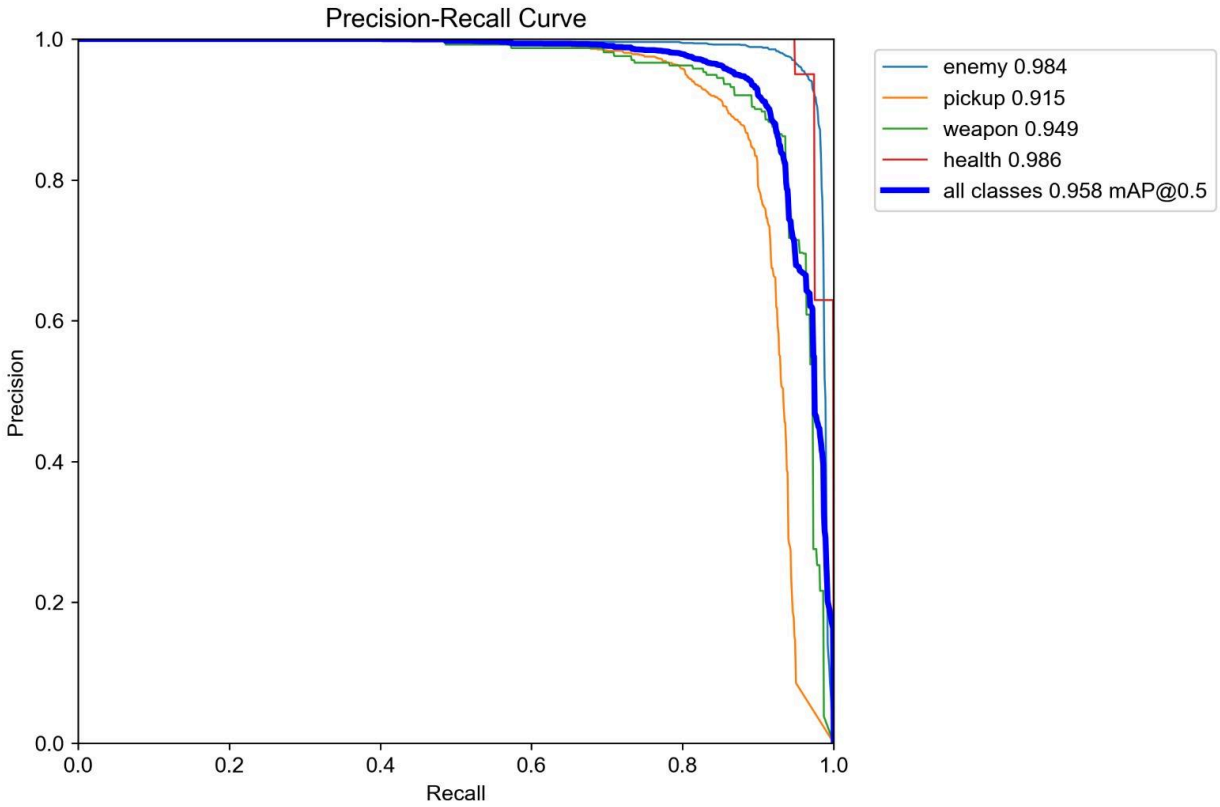
P_curve



Interpretacion:

Todas las clases indican tener buena capacidad de precisión.

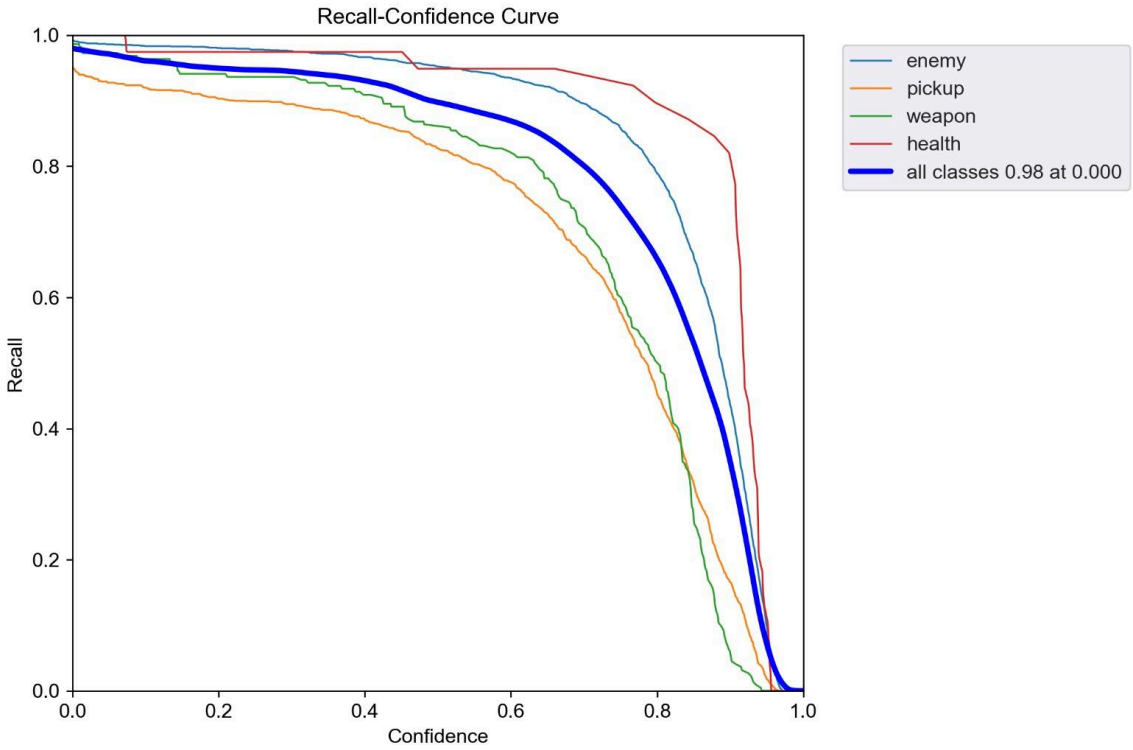
PR curve



Interpretación:

Las clases empiezan a tener un mal equilibrio recall-precision a cuando el recall (cobertura de detecciones) pasa por el valor 0.9.

R?curve



Interpretación:

Las clases empiezan a tener menos verdaderos positivos mayormente a partir del nivel de confianza 0.7.

Training Batch:

Interpretación:

Algunos ejemplos de imágenes con sus etiquetas correspondientes para el entrenamiento.

Validation y prediction:

Interpretación: Las predicciones en el resultado final son muy prometedoras, la mayoría mostrando un alto nivel de predicción y seguridad.

6. Conclusiones

Se logró implementar un asistente inteligente funcional para el entorno de VizDoom, capaz de detectar en tiempo real tanto enemigos como pickups, utilizando modelos personalizados basados en YOLOv5s. El sistema integra la inferencia visual y permite al usuario activar o desactivar la asistencia según necesidad, contribuyendo a mejorar la accesibilidad para personas con discapacidad visual parcial.

Los modelos entrenados alcanzaron métricas de precisión y recall muy altas, así como valores de mAP superiores a lo esperado en la validación (mAP@0.5 > 0.85 para enemigos y > 0.90 para pickups). Esto demuestra que la arquitectura seleccionada, junto con la organización y etiquetado del dataset, permitió una detección robusta y eficiente incluso en recursos limitados como CPU.

El sistema ha sido diseñado de manera modular y escalable, permitiendo la incorporación sencilla de nuevas funcionalidades, como el auto-aim, el entrenamiento continuo mediante

aprendizaje por refuerzo y la integración con otras plataformas. Además, el flujo de datos y el etiquetado permiten agregar nuevas clases o expandir el dataset fácilmente. Sería bastante interesante investigar cómo podríamos implementar modelos similares en juegos con gráficos más fotorrealistas.

Impacto social y proyección:

La solución desarrollada aporta una mejora concreta en la accesibilidad de los videojuegos, facilitando la inclusión de personas con discapacidad visual en entornos de juego interactivos. Además, la metodología empleada puede servir como base para futuras aplicaciones tanto en videojuegos como en otros ámbitos que requieran asistencia visual inteligente.