Rapport détaillé du projet : Suivi d'objets avec YOLOv8 et la théorie des jeux

Realisé par :

- TAREB Selma
- KHOUAS Assia
- KEMMOUN Ramzy
- ABADLI Badreddine

Spécialité: M2 informatique Visuelle
Module : Théorie des jeux

1. Introduction

Le suivi d'objets est un domaine central de la vision par ordinateur, avec des applications variées allant de la surveillance à la robotique. Cependant, dans des environnements réels, la détection d'objets peut être perturbée par divers facteurs, tels que le bruit visuel ou les changements d'éclairage. Ce projet propose de modéliser le problème de suivi d'objets comme un jeu à somme nulle entre deux joueurs :

1. **Joueur 1 : YOLOv8** – un modèle de détection d'objets, utilisé pour détecter et suivre des objets dans une séquence d'images.
2. **Joueur 2 : Bruit de Perlin** – une technique générant du bruit visuel, utilisé pour perturber les détections effectuées par YOLO.

Ce modèle est analysé dans le cadre de la **théorie des jeux à somme nulle**, où chaque joueur cherche à maximiser ou minimiser un gain, dans ce cas mesuré par l'**Intersection over Union (IoU)** entre les prédictions de YOLO et les annotations réelles.

2. Objectifs du Projet

L'objectif principal de ce projet est de :

- Analyser l'interaction entre YOLOv8 et le bruit de Perlin dans un contexte de suivi d'objets.
- Remplir une matrice de gain à somme nulle pour modéliser différentes stratégies de YOLO et du bruit de Perlin.
- Trouver un **équilibre de Nash**, où chaque joueur (YOLO et Bruit) choisit une stratégie optimale en réponse à l'autre.

Les stratégies sont définies comme suit :

- **Pour YOLO (Joueur 1)** : Deux stratégies sont testées, la **précision faible** et la **précision élevée**, basées sur des seuils de confiance différents pour la détection.
- **Pour le Bruit (Joueur 2)** : Deux stratégies sont également définies, le **bruit faible** et le **bruit élevé**, représentant l'intensité du bruit de Perlin ajouté à l'image.

3. Méthodologie

3.1. YOLOv8

YOLOv8 (You Only Look Once version 8) est un modèle de détection d'objets en temps réel, utilisant des réseaux de neurones convolutifs (CNN). Il prédit les boîtes englobantes des objets en divisant l'image en une grille.

- **Stratégie 1 : Précision faible**
  Un **seuil de confiance bas** est utilisé, permettant au modèle de détecter plus d'objets. Toutefois, cela peut entraîner un plus grand nombre de détections incorrectes, comme des faux positifs. Cette stratégie favorise une couverture plus large, au détriment de la précision.

- **Stratégie 2 : Précision élevée**
  Un **seuil de confiance élevé** est appliqué, réduisant le nombre de détections mais augmentant leur précision. Le modèle ne détecte que les objets avec une forte certitude, minimisant ainsi les erreurs mais aussi les objets détectés.

3.2. Bruit de Perlin

Le **bruit de Perlin** est un bruit fractal utilisé pour ajouter des perturbations naturelles à des images, modifiant ainsi les détections de YOLO.

- **Stratégie 1 : Bruit faible**
  Un **bruit faible** est appliqué, perturbant légèrement les détections du modèle. L'impact est subtil, mais peut affecter la qualité de certaines détections.
- **Stratégie 2 : Bruit élevé**
  Un **bruit élevé** est ajouté, perturbant fortement les images et réduisant la qualité des détections de YOLO. Cela simule un environnement où les objets sont difficilement détectables à cause des distorsions.

3.3. Fonctions de Gain des Joueurs

Joueur 1 - Modèle (YOLOv8) :
Le gain du modèle est basé sur la performance de la détection d'objets. Plus précisément, il est calculé à partir de l'IoU entre la détection du modèle et la vérité terrain (ground truth). **L'IoU** est un indicateur de la précision du modèle : il mesure le rapport de l'intersection entre la boîte englobante prédite et celle de la vérité terrain, par rapport à leur union.

La fonction de gain du modèle est donc l'IoU moyen pour tous les objets détectés dans une image donnée.

$$G_{mod\_e} = \frac{1}{N} \sum_{i-1}^{N} IoU_i$$

Où :

- N est le nombre d'objets détectés dans l'image.
- IoU_i est l'IoU de chaque objet i.

Joueur 2 - Bruit (Perlin) :
Le bruit perturbe la détection du modèle en modifiant les images. Son but est de réduire l'IoU, et donc d'affecter négativement le gain du modèle. Le gain du bruit est l'inverse de celui du modèle, car plus le modèle perd, plus le bruit gagne.

$$G_{bruit} = 1 - G_{mod\_e}$$

Ainsi, la somme des gains des deux joueurs est toujours égale à zéro, ce qui en fait un jeu à somme nulle :

$$G_{mod\_e} + G_{bruit} = 0$$

3.4. Remplissage de la Matrice de Gain

La matrice de gains représente l'interaction entre les stratégies des deux joueurs (le modèle et le bruit Perlin). Chaque cellule de la matrice contient l'**IoU** (Intersection over Union) moyen pour une combinaison donnée de stratégies des deux joueurs. Par exemple, si le modèle choisit une détection précise et que le bruit est faible, l'IoU sera élevé, indiquant que le modèle a bien suivi les objets. Inversement, si le bruit est élevé, l'IoU sera faible. Ces valeurs aident à comprendre l'impact de chaque stratégie sur les performances du modèle.

3.5. Recherche du Maxmin et du Minmax

Dans ce projet, les concepts de **Maxmin** et **Minmax** ont été utilisés pour analyser les stratégies des deux joueurs (YOLO et le bruit de Perlin) dans un jeu à somme nulle.

1. **Maxmin pour YOLO** : Il s'agit de la stratégie la plus **sécurisée** pour YOLO, où il choisit la stratégie qui lui donne le **meilleur IoU minimum** possible, quel que soit le bruit. Cela garantit à YOLO une performance minimale dans le pire des cas, même si le bruit choisit la stratégie la plus perturbatrice.
2. **Minmax pour le Bruit** : Le **Minmax** représente la stratégie la plus **prudente** pour le bruit. Il s'agit du paiement minimal que le bruit peut obtenir, quelle que soit la stratégie de YOLO. Le bruit choisit la stratégie qui lui permet de minimiser l'IoU de YOLO tout en limitant ses propres pertes.

Intérêt de l'Analyse Maxmin et Minmax

L'analyse de Maxmin et Minmax permet de **comprendre les choix optimaux** de chaque joueur, en tenant compte des perturbations mutuelles. Si les valeurs de Maxmin et Minmax sont égales, cela signifie que le jeu atteint un **équilibre en stratégies pures**, où chaque joueur choisit une stratégie déterminée, garantissant une stabilité dans le jeu.

4. Explication du Code et de la Dataset

4.1. Dataset :
https://www.kaggle.com/datasets/angelosgiotis/consumers-bid?select=README.md

Vue d'ensemble:

La partie "Consumers" comprend des images de consommateurs dans des environnements de détail, capturées avec des caméras RGB. Chaque image contient des informations de vérité terrain, notamment des coordonnées de boîtes englobantes pour chaque consommateur.

Structure:

La dataset "Consumers" est divisée en deux partitions : 120 séquences vidéo pour l'entraînement et 25 séquences pour les tests, représentant un total de 8700 images/frames. La structure de la dataset est la suivante :

- Le fichier `gt.txt` contient des annotations globales, telles que l'ID de la frame, l'ID du consommateur et les coordonnées de la boîte englobante pour chaque séquence. Ce fichier est compatible avec le format du challenge MOT (Multiple Object Tracking).
- Chaque fichier `frameID.txt` contient des informations spécifiques par frame, avec l'ID du consommateur, les coordonnées de la boîte englobante, ainsi que des informations supplémentaires sur l'âge et le genre du consommateur.

Cette dataset est utilisée pour des tâches de vision par ordinateur telles que le suivi d'objets (tracking) et la détection de caractéristiques démographiques des consommateurs (par exemple, âge et genre). Vous avez utilisé uniquement la partition "Consumers" pour votre projet, qui est principalement axée sur l'extraction de ces informations à partir des séquences d'images.

4.1. Code :

- **Les fonction importantes :**

`1/ load_ground_truth`

La fonction `load_ground_truth` charge les annotations de vérité terrain à partir d'un fichier texte. Chaque ligne du fichier contient l'ID de la frame et les coordonnées d'une boîte englobante (x, y, largeur, hauteur). La fonction lit ces informations et les organise dans un dictionnaire, où chaque ID de frame est associé à une liste de boîtes englobantes au format `(x1, y1, x2, y2)` (coin supérieur gauche et coin inférieur droit).

Elle permet ainsi de récupérer facilement les boîtes de détection pour chaque frame, afin de comparer les prédictions du modèle avec les vérités terrain lors de l'évaluation des performances du suivi d'objets.

```python
def load_ground_truth(gt_file):
    gt_data = {}
    with open(gt_file, "r") as file:
        for line in file:
            parts = line.strip().split(",")
            frame_id = int(parts[0])
            x, y, w, h = map(float, parts[2:6])
            if frame_id not in gt_data:
                gt_data[frame_id] = []
            gt_data[frame_id].append([x, y, x + w, y + h])
    return gt_data
```

`2/ iou`

Cette fonction calcule l' Intersection over Union (IoU) entre deux boîtes de délimitation. L'IoU mesure le chevauchement entre les boîtes. Voici comment elle fonctionne :

1. Elle trouve les coordonnées de l'intersection des deux boîtes.
2. Elle calcule l'aire de l'intersection.
3. Elle calcule l'aire totale de chaque boîte et leur union (qui est l'aire des deux boîtes moins l'aire de l'intersection).
4. L'IoU est le rapport entre l'aire de l'intersection et l'aire de l'union.

```python
def iou(box1, box2):
    x1, y1, x2, y2 = box1
    x1g, y1g, x2g, y2g = box2
    xi1, yi1 = max(x1, x1g), max(y1, y1g)
    xi2, yi2 = min(x2, x2g), min(y2, y2g)
    inter_area = max(0, xi2 - xi1) * max(0, yi2 - yi1)

    box1_area = (x2 - x1) * (y2 - y1)
    box2_area = (x2g - x1g) * (y2g - y1g)

    union_area = box1_area + box2_area - inter_area

    return inter_area / union_area if union_area else 0
```

2/ add_permin_noise

La fonction `add_perlin_noise` ajoute du bruit de Perlin à une image pour créer des motifs aléatoires. Voici son fonctionnement :

1. Paramètres :
    - scale : Contrôle la taille des motifs de bruit (plus grand = motifs plus larges).
    - intensity : Détermine l'intensité du bruit ajouté à l'image.
2. Processus :
    - Elle génère une carte de bruit de Perlin de la même taille que l'image.
    - Ce bruit est normalisé et ajusté selon l'intensité.
    - Le bruit est ensuite ajouté à l'image, tout en s'assurant que les valeurs des pixels restent dans la plage [0, 255].

```
import numpy as np
from perlin_noise import PerlinNoise

def add_perlin_noise(image, scale=30, intensity=40):
    """
    Ajoute du bruit de Perlin à une image.
    - scale : échelle du bruit (plus grand = motifs plus larges)
    - intensity : intensité du bruit ajouté
    """
    h, w, _ = image.shape
    noise = PerlinNoise(octaves=4)

    noise_map = np.zeros((h, w))
    for i in range(h):
        for j in range(w):
            noise_map[i, j] = noise([i / scale, j / scale])

    noise_map = (noise_map - np.min(noise_map)) / (np.max(noise_map) - np.min(noise_map))
    noise_map = (noise_map * 2 - 1) * intensity

    noisy_image = np.clip(image.astype(np.float32) + noise_map[:, :, None], 0, 255).astype(np.uint8)

    return noisy_image
```

3/ display_video

- **`ArtistAnimation`** crée une animation à partir de la liste des images `frames`.
- **`to_html5_video()`** convertit l'animation en format vidéo HTML5.
- **`HTML`** permet d'afficher la vidéo directement dans Google Colab.

Cette méthode devrait te permettre d'afficher une vidéo à partir de tes images directement dans l'environnement Google Colab. Assure-toi que `frames` est une liste d'images sous forme de tableaux NumPy ou d'objets compatibles avec `imshow`.

```
def display_video(frames):
    fig = plt.figure(figsize=(8, 8))
    mov = [[plt.imshow(frame, animated=True)] for frame in frames]
    anime = animation.ArtistAnimation(fig, mov, interval=50, repeat_delay=1000)
    plt.close()

    # Convertir l'animation en vidéo et l'afficher dans Colab
    return HTML(anime.to_html5_video())
```

- **Les fonctions principales :**

Les fonctions principales de ce système sont conçues pour traiter différents scénarios de suivi d'objets en tenant compte du bruit et de la précision du modèle de détection. Elles sont réparties en quatre catégories :

1. Détection faible précision, bruit faible : Cette fonction est utilisée lorsque le modèle de détection a une faible précision et qu'il y a peu de bruit perturbant le suivi.
2. Détection faible précision, bruit élevé : Elle s'applique lorsque le modèle de détection est moins précis et que le bruit est important, nécessitant une gestion robuste des perturbations.
3. Détection précise, bruit faible : Dans ce cas, le modèle de détection offre une grande précision, et le bruit est faible, permettant un suivi plus stable et exact.
4. Détection précise, bruit élevé : Cette fonction est optimisée pour des modèles de détection très précis, tout en gérant un bruit élevé qui perturbe le suivi.

Ces fonctions permettent d'adapter le système à différentes conditions de détection et de bruit, assurant ainsi un suivi optimal des objets dans divers environnements.

1. Détection faible précision, bruit faible

La fonction `process_sequence` traite une séquence d'images pour effectuer une détection d'objets en utilisant un modèle YOLOv8 et évalue la qualité des détections en calculant les scores IoU (Intersection over Union) entre les boîtes prédites et les boîtes de vérité terrain (ground truth).

Voici les étapes principales de la fonction :

1. Chargement du modèle et des données de vérité terrain :
    - Le modèle YOLOv8 est chargé à partir du chemin `model_path` (par défaut, `'yolov8n.pt'`).
    - Les données de vérité terrain (ground truth) sont chargées depuis un fichier `gt.txt` situé dans le sous-dossier `gt` du dossier `sequence_folder`.
2. Génération de bruit de Perlin :
    - Un objet `PerlinNoise` est créé pour ajouter un bruit léger aux coordonnées des boîtes de détection, en perturbant légèrement les positions des objets détectés.
3. Traitement des frames de la séquence :
    - Les images de la séquence sont triées et lues une par une.
    - Pour chaque image (frame), la détection d'objets est effectuée à l'aide du modèle YOLOv8, et les boîtes de détection (bounding boxes) sont extraites.
4. Ajout de bruit aux boîtes de détection :
    - Le bruit de Perlin est ajouté aux coordonnées des boîtes détectées pour introduire des perturbations. Ce bruit est appliqué aux coordonnées `x1`, `y1`, `x2`, `y2` de la boîte prédite.
5. Affichage des boîtes de détection YOLO et des boîtes de vérité terrain :

    - Pour chaque frame, les boîtes détectées par YOLO sont dessinées en rouge avec l'étiquette "YOLO".
    - Les boîtes de vérité terrain (GT) sont également dessinées en vert sur la même image.
6. Calcul de l'IoU (Intersection over Union) :
    - Pour chaque boîte de vérité terrain, l'IoU avec chaque boîte prédite par YOLO est calculé.
    - Le score IoU le plus élevé est conservé pour chaque boîte de vérité terrain, et tous les scores IoU sont stockés.
7. Conversion des images et retour de la vidéo :
    - Chaque frame est convertie en format RGB et ajoutée à la liste `video_frames`.
    - Une fois toutes les frames traitées, la moyenne des scores IoU est calculée et affichée.
    - La vidéo résultante, composée des frames traitées, est retournée sous forme d'une vidéo HTML5 qui peut être affichée dans Google Colab.

2. Détection faible précision, bruit fort :

La fonction **`process_sequence`** traite une séquence d'images pour effectuer la détection d'objets et évaluer la qualité des détections en utilisant le modèle YOLOv8. Elle inclut l'ajout de bruit de Perlin via la fonction **`add_perlin_noise`** pour perturber les images. Voici un résumé détaillé de son fonctionnement :

1. **Chargement du modèle et des données de vérité terrain :**
    - Le modèle YOLOv8 est chargé depuis le chemin spécifié (`yolov8n.pt`).
    - Les données de vérité terrain (GT), qui contiennent les boîtes de détection correctes, sont lues à partir d'un fichier `gt.txt` situé dans le dossier `gt` du dossier `sequence_folder`.
2. **Génération du bruit de Perlin :**
    - Un objet **`PerlinNoise`** est créé avec un nombre d'octaves défini. Ce bruit est utilisé pour ajouter des perturbations aléatoires aux coordonnées des boîtes détectées.
3. **Traitement des images (frames) :**
    - Les images (frames) de la séquence sont lues, triées et traitées une par une.
    - La fonction **`add_perlin_noise`** est utilisée pour ajouter du bruit de Perlin aux images avant la détection d'objets. Ce bruit perturbe les images pour simuler des conditions réelles de bruit.

Les étapes suivantes sont similaires à celles de la fonction précédente :

- Affichage des boîtes de détection YOLO et des boîtes de vérité terrain.
- Calcul de l'IoU (Intersection over Union) entre les boîtes de vérité terrain et les boîtes détectées.
- Génération de la vidéo finale des frames traitées avec les résultats visuels.

3. Détection forte précision, bruit faible :

La fonction `process_sequence`, traite une séquence d'images pour effectuer la détection d'objets avec le modèle YOLOv8. Elle utilise un seuil de confiance élevé pour les détections et applique un bruit de Perlin faible pour perturber légèrement les boîtes de détection. Voici un résumé détaillé de son fonctionnement :

Fonctionnement de la fonction `process_sequence` :

1. Chargement du modèle et des données de vérité terrain :
    - Le modèle YOLOv8 est chargé à partir du chemin spécifié (`yolov8l.pt`), qui est une version plus grande du modèle YOLO.
    - Les données de vérité terrain (GT), contenant les boîtes de détection correctes pour chaque image, sont lues depuis un fichier `gt.txt`.
2. Création du bruit de Perlin :
    - Un objet `PerlinNoise` est utilisé pour générer du bruit de Perlin qui sera appliqué aux coordonnées des boîtes de détection afin de simuler une perturbation faible.
3. Traitement des images (frames) :
    - Les images de la séquence sont lues et triées par ordre de leur nom de fichier.
    - Pour chaque image, une détection d'objets est effectuée avec YOLOv8. Les résultats sont filtrés avec un seuil de confiance élevé de `0.5` et une valeur de Non-Maximum Suppression (NMS) de `0.4` pour améliorer la précision des détections.
    - Le bruit de Perlin est appliqué aux coordonnées des boîtes détectées pour ajouter une légère perturbation.

4. Détection forte précision, bruit faible :

La fonction `process_sequence` traite une séquence d'images d'une vidéo, applique un bruit de Perlin à chaque image via la fonction `add_perlin_noise`, effectue une détection d'objets avec le modèle YOLOv8, et évalue la qualité des détections en calculant l'IoU (Intersection over Union) entre les boîtes de prédiction et les boîtes de vérité terrain.

1. Chargement du modèle et des données de vérité terrain :
    - Le modèle YOLOv8 est chargé à partir du chemin spécifié (`yolov8l.pt`).
    - Les données de vérité terrain (`gt.txt`) sont chargées depuis un fichier dans le dossier `sequence_folder/gt`, contenant les coordonnées des boîtes de détection correctes pour chaque image.
2. Chargement et traitement des images :
    - La fonction `os.listdir()` récupère les fichiers d'images, triés par nom pour respecter l'ordre des frames.
    - Chaque image est lue avec `cv2.imread()`.

3. Application du bruit de Perlin :
    - La fonction `add_perlin_noise()` est utilisée ici pour appliquer un bruit de Perlin aux images avant la détection d'objets. Le bruit est ajouté pour simuler des perturbations réalistes dans les images.
    - Les paramètres d'échelle (`scale=30`) et d'intensité (`intensity=40`) sont utilisés pour contrôler l'effet du bruit sur l'image.
4. Détection d'objets avec YOLOv8 :
    - Le modèle YOLOv8 effectue la détection d'objets sur chaque image bruitée avec un seuil de confiance de `0.6` et un seuil de Non-Maximum Suppression (NMS) de `0.7`. Ces valeurs sont utilisées pour améliorer la précision de la détection.
    - Pour chaque boîte détectée, les coordonnées sont extraites, et une boîte est dessinée autour de l'objet détecté dans l'image.

Résumé des seuils et paramètres :

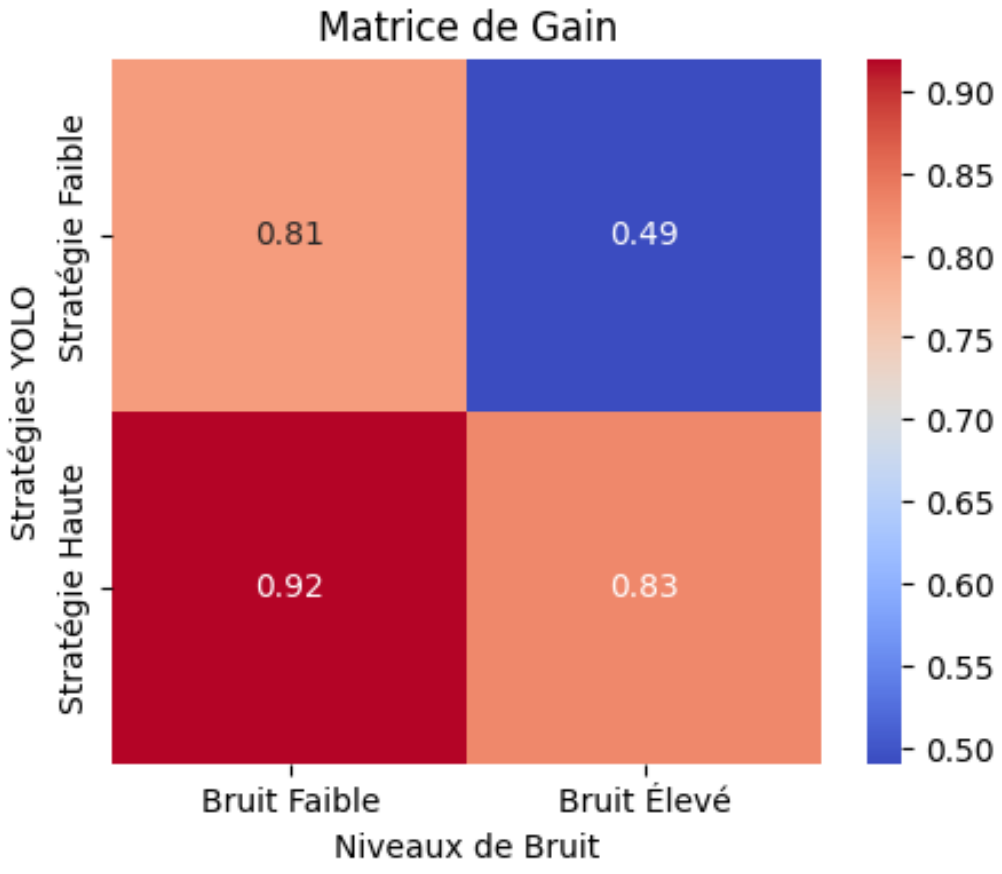
- **Conf** (confiance) : seuil de confiance pour les prédictions YOLO, fixé à `0.6`.
- **IoU** : seuil de NMS fixé à `0.7` pour éliminer les boîtes de faible qualité.
- **Bruit de Perlin** : **La fonction `add_perlin_noise` est utilisée pour ajouter des perturbations aléatoires** dans les images avec des paramètres `scale=30` et `intensity=40`.

5. Interpretation des resultats :

4.1. La matrice des gains finale :

Perlin noise \ YOLO	Perturbation faible	Perturbation forte
Précision faible	Nombre total de frames traitées : 54 Moyenne IoU : 0.8160	Nombre total de frames traitées : 54 Moyenne IoU : 0.4907
Précision forte	Nombre total de frames traitées : 54 Moyenne IoU : 0.9210	Nombre total de frames traitées : 54 Moyenne IoU : 0.8398

Représentation de la matrice de gain :



Quand YOLO choisit une précision faible et le bruit choisit un bruit faible :

- L'IoU est relativement élevé (0.81), ce qui signifie que YOLO a une bonne performance de détection même avec un bruit faible. Le bruit, dans ce cas, n'a pas perturbé beaucoup les détections.

Quand YOLO choisit une précision faible et le bruit choisit un bruit fort :

- L'IoU chute à 0.49. Cela montre que le bruit fort perturbe de manière significative la détection de YOLO, ce qui réduit la performance de détection.

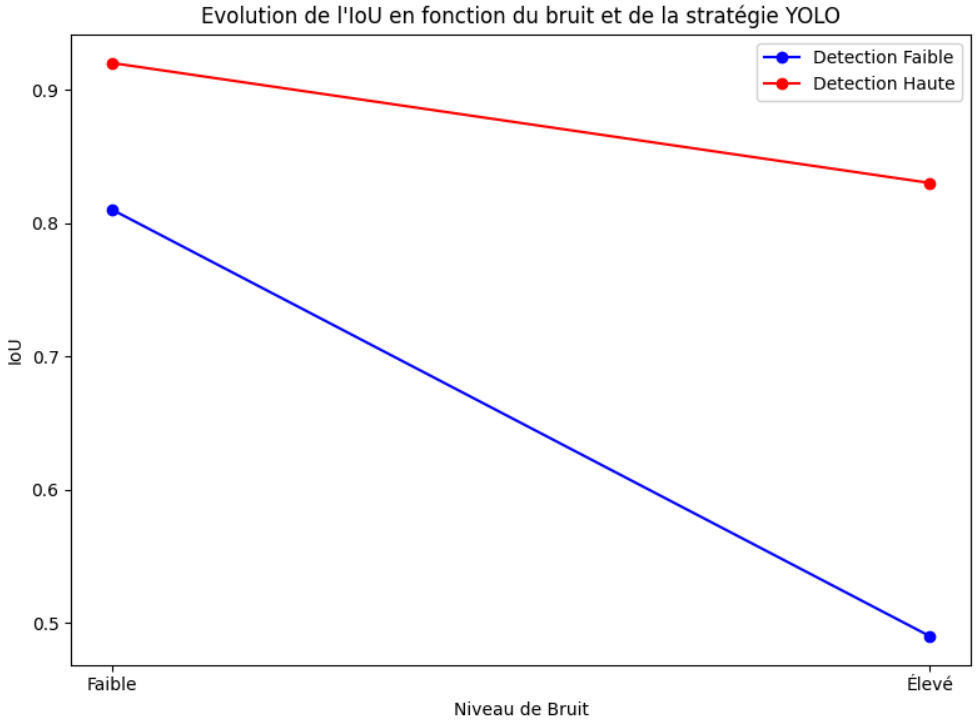
Quand YOLO choisit une précision élevée et le bruit choisit un bruit faible :

- L'IoU est très élevé (0.92), indiquant que YOLO est très performant dans sa détection avec une précision élevée et un bruit faible. Le bruit a un effet minimal sur la détection de YOLO.

Quand YOLO choisit une précision élevée et le bruit choisit un bruit fort :

- L'IoU est encore assez élevé (0.83), ce qui signifie que même si le bruit est fort, YOLO reste performant grâce à sa précision élevée. Le bruit perturbateur est moins efficace que dans le cas précédent.

Les résultats obtenus à partir des différentes combinaisons de stratégies montrent que la stratégie de haute précision pour YOLO (utilisant YOLOv8l avec un seuil de confiance de 0.6 et un seuil NMS de 0.7) donne les meilleures performances en termes d'IoU. Cela souligne l'importance de l'ajustement des paramètres pour optimiser les performances de détection. De plus, le bruit de Perlin, qui perturbe les détections de manière optimale, démontre l'impact des perturbations sur la qualité des prédictions.



4.1. Le calcule du MaxMin et MinMax :

```python
import numpy as np

payment_matrix = np.array([
    [0.81, 0.49],
    [0.92, 0.83]
])

# Calcul du Maxmin pour YOLO (maximum des paiements minimaux dans chaque ligne)
maxmin_yolo = np.max(np.min(payment_matrix, axis=1))
print(f"Maxmin pour YOLO (stratégie la plus sécurisée de YOLO) : {maxmin_yolo}")

# Calcul du Minmax pour le Bruit (minimum des paiements maximaux dans chaque colonne)
minmax_bruit = np.min(np.max(payment_matrix, axis=0))
print(f"Minmax pour Bruit (stratégie la plus sécurisée du Bruit) : {minmax_bruit}")

if maxmin_yolo == minmax_bruit:
    print("Le jeu a une valeur en stratégie pure !")
else:
    print("Le jeu n'a pas de valeur en stratégie pure.")
```

Après exécution :

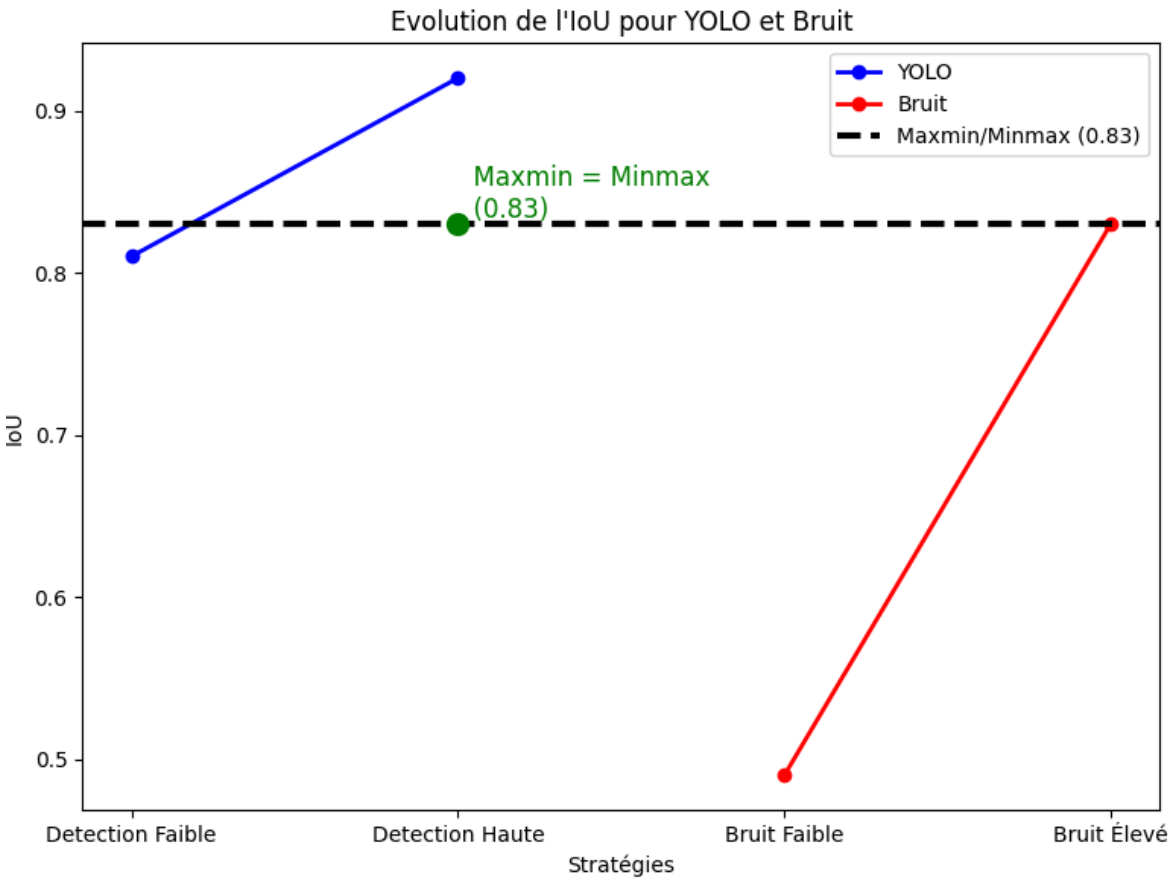
```
Maxmin pour YOLO : 0.83
Minmax pour Bruit : 0.83
Le jeu a une valeur en stratégie pure!
```

- **Maxmin pour YOLO** = 0.83
- **Minmax pour Bruit** = 0.83

Cela signifie que **le jeu a une valeur en stratégie pure**. En d'autres termes, chaque joueur (YOLO et Bruit) peut choisir une stratégie déterminée, et le jeu a un équilibre stable dans lequel les deux joueurs choisissent leurs meilleures réponses sans nécessiter de stratégies mixtes.

Interprétation du Résultat

- **Maxmin pour YOLO** (0.83) : Cela indique que la **stratégie la plus sécurisée de YOLO** est celle où il obtient un IoU de 0.83, quelle que soit la stratégie choisie par Bruit. YOLO peut garantir un IoU d'au moins 0.83 en choisissant cette stratégie.
- **Minmax pour Bruit** (0.83) : Cela signifie que la **stratégie la plus sécurisée pour Bruit** (en cherchant à perturber le plus possible l'IoU de YOLO) est celle où il réduit l'IoU de YOLO à un minimum de 0.83. Bruit choisira cette stratégie pour s'assurer de perturber l'IoU de YOLO tout en minimisant les dommages pour lui-même.



- Le graphe confirme que le jeu entre YOLO et Bruit est équilibré, chaque joueur adoptant une stratégie optimale garantissant leur meilleur résultat dans le pire des cas.
- YOLO garantit un IoU de 0.83, même sous perturbation maximale par Bruit.
- Bruit choisit une stratégie qui limite le mieux possible l'IoU de YOLO à 0.83.
- Ce résultat suggère que les stratégies mixtes ne sont pas nécessaires, et le jeu est "stable" et "sécurisé".

Conclusion

Puisque **Maxmin = Minmax** (les deux égaux à 0.83), le jeu admet une **valeur en stratégie pure**. Cela veut dire qu'il existe un équilibre stable dans lequel chaque joueur choisit la stratégie optimale de manière déterminée.

Les joueurs vont donc suivre une **stratégie pure** sans avoir besoin de mélange de stratégies. Le jeu est "prudent" et "sûr", chaque joueur ayant une stratégie qui lui garantit le meilleur résultat dans le pire des scénarios.

6. Conclusion générale

Ce projet a exploré l'interaction entre un modèle de détection d'objets avancé (YOLOv8) et le bruit de Perlin dans un contexte de suivi d'objets, en appliquant la théorie des jeux à somme nulle pour analyser les stratégies des deux joueurs. En traitant les objets comme des jeux entre le modèle YOLO (Joueur 1) et le bruit de Perlin (Joueur 2), nous avons pu modéliser différentes stratégies, évaluer l'impact du bruit sur la précision du modèle, et étudier les gains associés à chaque choix stratégique.

Les résultats ont permis de développer une matrice de gain à somme nulle, servant à déterminer les interactions entre les différentes stratégies, telles que la détection à faible ou haute précision de YOLO et les niveaux de bruit de faible ou haute intensité. L'analyse des stratégies Maxmin et Minmax a contribué à une meilleure compréhension des choix optimaux des deux joueurs, soulignant l'importance d'adapter le système en fonction des conditions environnementales.

Les performances du système ont été évaluées à travers des métriques telles que l'IoU, permettant de mesurer l'efficacité du suivi d'objets et de quantifier les perturbations causées par le bruit. Le projet a démontré que le bruit de Perlin, en perturbant les détections du modèle, pouvait affecter significativement la qualité du suivi d'objets, en particulier dans des environnements à forte perturbation visuelle.

En somme, ce projet offre une nouvelle perspective sur la manière de gérer les perturbations dans les systèmes de suivi d'objets, en utilisant la théorie des jeux comme outil de modélisation stratégique. Les résultats obtenus ouvrent la voie à des améliorations potentielles, telles que l'intégration de stratégies de correction d'erreurs basées sur l'analyse des perturbations, pour renforcer la robustesse des systèmes de vision par ordinateur dans des environnements complexes.