Step1

**1. シーンの準備と道・障害物の配置**

**ステップ1: 新規プロジェクトを作成**

* **Unityを起動し、新しい3Dプロジェクトを作成。**

**ステップ2: 地面（道路）の作成**

* **Hierarchy（ヒエラルキー）ウィンドウの空白部分を右クリック → 3D Object → Planeを選択。**
* **これが道路や地面になります。**
* **サイズを変えるには、選択してInspectorウィンドウでScaleを調整（例：X=10, Z=50など）。**

**ステップ3: 道のライン・車線の作成**

* **3D Object → Cubeを追加して、道路上に配置。**
* **例えば、車線の白線を表現するために白色のCubeを細長くして道路の端に配置。**

**ステップ4: 障害物の配置**

* **3D Object → CubeやSphereを追加して、障害物に見立てる。**
* **位置、サイズ、色を変えて配置。**
* **衝突判定のために\*\*Collider（Box Collider, Sphere Collider）\*\*も自動で追加されます。**

**2. 車の作成と配置**

**ステップ1: 車のモデルを作る**

* **GameObjectの中からCubeまたはより詳細なモデル（無料のアセットも利用可）を使って車を作る。**
* **サイズや色を調整。**

**ステップ2: 車の移動制御**

* **追加した車にRigidbodyコンポーネントを付与（Physics用）。**
* **必要に応じてBox ColliderやWheel Colliderを追加。**

**3. カメラの配置と視界の取得**

**ステップ1: カメラの追加**

* **Hierarchy → Right Click → Cameraを選択。**
* **これを車の子オブジェクトにします。**

**ステップ2: カメラの位置と角度設定**

* **車の上または前方にカメラを配置（例えば、車の上に置き、少し前に傾ける）。**
* **位置例：**
  + **Position: (0, 1.5, -2)（車の中心から少し上方・後方）**
  + **Rotation: (10, 0, 0)（少し前傾角度）**

**ステップ3: カメラの視点**

* **カメラのProjectionはPerspective。**
* **視野角やクリッピング距離も適宜調整。**

**4. スクリプトで画像取得の準備**

* **カメラの映像をキャプチャするには、RenderTextureを使います。**
* **Hierarchy → Right Click → Create Empty → 名前は例：CameraCapture。**
* **カメラのTarget TextureにこのRenderTextureを設定。**

**こうすることで、カメラの映像を画像として取得できます。**

**5. まとめ**

* **道と障害物はプリミティブのObjectから配置**
* **車を作り、制御スクリプトを追加**
* **カメラを車に取り付けて映像を取得**
* **これらを基に、Pythonと連携しやすい状態を作ります。**

Step2

そうだね。超大元の土台ができた感じだ。ここからは、

0.画像を撮って？pythonに送るスクリプト 1.画像を認識して処理し、結果をunityに送るスクリプト 2.unity上で、送られてきた情報ヲ基に、車の動作を制御するスクリプト これら3つを作る感じ？

車のbody mainのMesh Coliderのconvexのチェックを入れる。(non convexは形を正確に、凹凸まで再現する感じで、これとrigidbodyは共存できない。 Convexは抽象度あげた感じで、これによって物理演算が可能になる。)

***あなたのPC上に、Pythonで「サーバ」プログラム（Flaskで作ったもの）を起動*** *それが「通信のハブ」になっている。*

***Unityはそのサーバを「IPアドレスとポート番号」で呼び出している****。*

**1. システムの基本構築**

* Unity内に**車とカメラ**を配置
* カメラは車の前に設置し、RenderTextureを使って映像を取得

**2. 画像送信部分の作成**

* Unityのスクリプトで**RenderTextureの画像をJPEGに圧縮**
* UnityWebRequestを使って、ローカルのPythonサーバに**画像データをPOST送信**

**3. Pythonサーバの構築**

* Flaskでサーバを立ち上げ
* /uploadエンドポイントを作り、画像受信→画像認識→結果返すコードを実装

**サーバーに対する質問**

なるほど。立てたサーバーはまさにハブで、unityからの情報をpythonに送り、その結果をまたunityに返す　というただそれだけの役割をしているということか。そのサーバーを立てる方法として、コマンドプロント上で「python ファイル名」とすることがあるのか。、(また、暗にIPアドレスとポート番号を指定しており、unity側からはそれを通じてアクセスする。）今回は仮想環境上(でなくてもいいけど)にサーバーを立てたが、サーバーを立てるということはファイルを作るわけではないんだな。見えないのか？

**【あなたの理解のポイント】**

**1. サーバは「ハブ」役割**

* Unityから送られた情報（画像やコマンド）を一旦受け取って
* Python側で認識・解析し、その結果をUnityに返す
* それだけの役割を【"待ち受ける"】形で常に動いている

**2. サーバの起動方法**

* **コマンドプロンプトだけで**  
  python your\_server.py と打つと起動
* そうすると、そのファイルの中に書いたコード（Flaskサーバ）**が実行される**  
  → これが「見えないけど動作しているサーバ」

**3. このファイルは「見えない」？ → そんなことはない！**

* 実体は**ソフトウェア**として起動して、**メモリ上に展開されている**
* そのコードは「ファイルに書いてある内容」だけど、自動的に「ウェブサーバ」になって動き出す
* \*\*「裏側の仕組み」\*\*なので、見えないだけ

**4. IPアドレスとポート番号は**

* **コードの中（app.run(host='0.0.0.0', port=5000)）で決めている**
* **Unityはそれを通じてアクセスしている**

そして、そのサーバに情報とともにアクセスすると、情報をパイソンスクリプトに渡したうえでパイソンスクリプトを実行し、結果を返hしてくれる。そんな感じか。

**4. 通信の成功と結果の確認**

* Unityから画像を正常に送信できて、
* Pythonからの返答（例：左折推奨の文字列）が、**UnityのConsoleに正しく表示**された

**5. 次のステップ（車の制御部分）**

* Pythonの結果をUnityが受信
* if文を使って「左に曲がる」動作や「直進」動作を

**画像処理編**

RGB空間(三原色の強さで表現) HSV空間は明度とか彩度とかで表現、より直感的

白線を検出するには、HSV空間における、明度が重要となり、これでわかりやすく白線を検出できる。

RGB空間で白線を検出しようとした場合、どれが白色っぽいかわかりにくい(RGBではすべてMax , HSVでは明度のみが明るい)

引継ぎ0620 17:24

まず、Mainについて、くっついている変なスクリプトをはがす。

Interval,2つのスクリプトについて調整する。

Native Leak Detection

void Start() { NativeLeakDetection.Mode = NativeLeakDetectionMode.EnabledWithStackTrace; }

このコードを入れたスクリプトをメインとかに張り付けることで？？デバッグが詳細なものになる。Chatgptとかは全然違うこと言うけど、それをやっても何も変わらない。これで変わった。