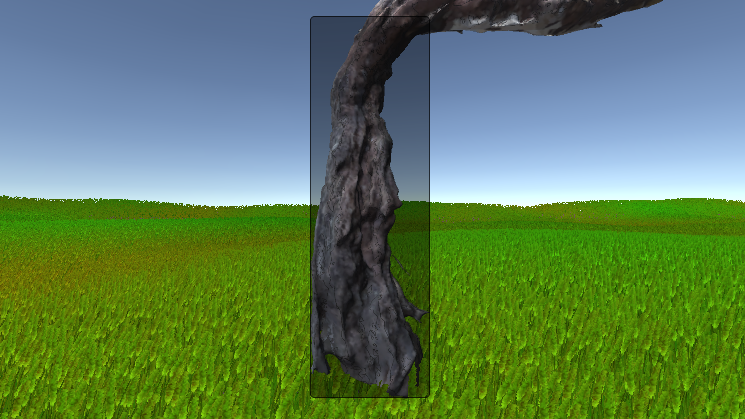
# Szőlőtőke annotálása Unity-ben YOLO neurális hálóhoz



. ábra, annotált tőke Unity-ben

*2021.07*

# Tartalomjegyzék

Feladat leírása 3

Használt technológiák 3

* Unity 3
* Meshroom 3
* .OBJ 4
* Blender 5

3D-s modell importálása Unity-be 6

Unity projekt megoldások 8

* Bounding box kiszámítása 8
* Forgatás megoldása 8
* Pillanatkép készítésének megoldása 8
* Fájl készítésének megoldása 9
* Folyamat vezérlése 9

Elért eredmények 11

Ábrajegyzék 13

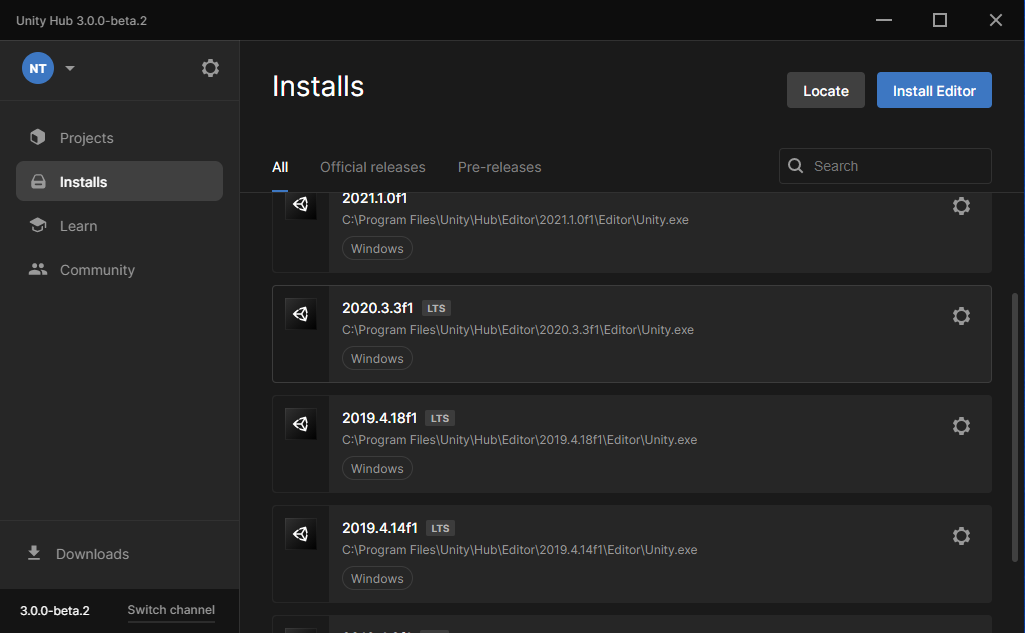
# Feladat leírása

Az alkalmazás célja, hogy egy digitalizált 3D-s szőlőtőkét több nézőpontból körbeforgasson adott fokonként és minden forgatáskor készítsen pillanatképet és egy szöveges fájlt, ami tartalmazza a tőke alsó részét körülölelő bounding box adatait a YOLO számára értelmezhető formátumban. Az így kapott képekből és a hozzá tartozó szöveges fájlokból pedig képesek leszünk feltanítani egy neurális hálót.

# Használt technológiák

## Unity

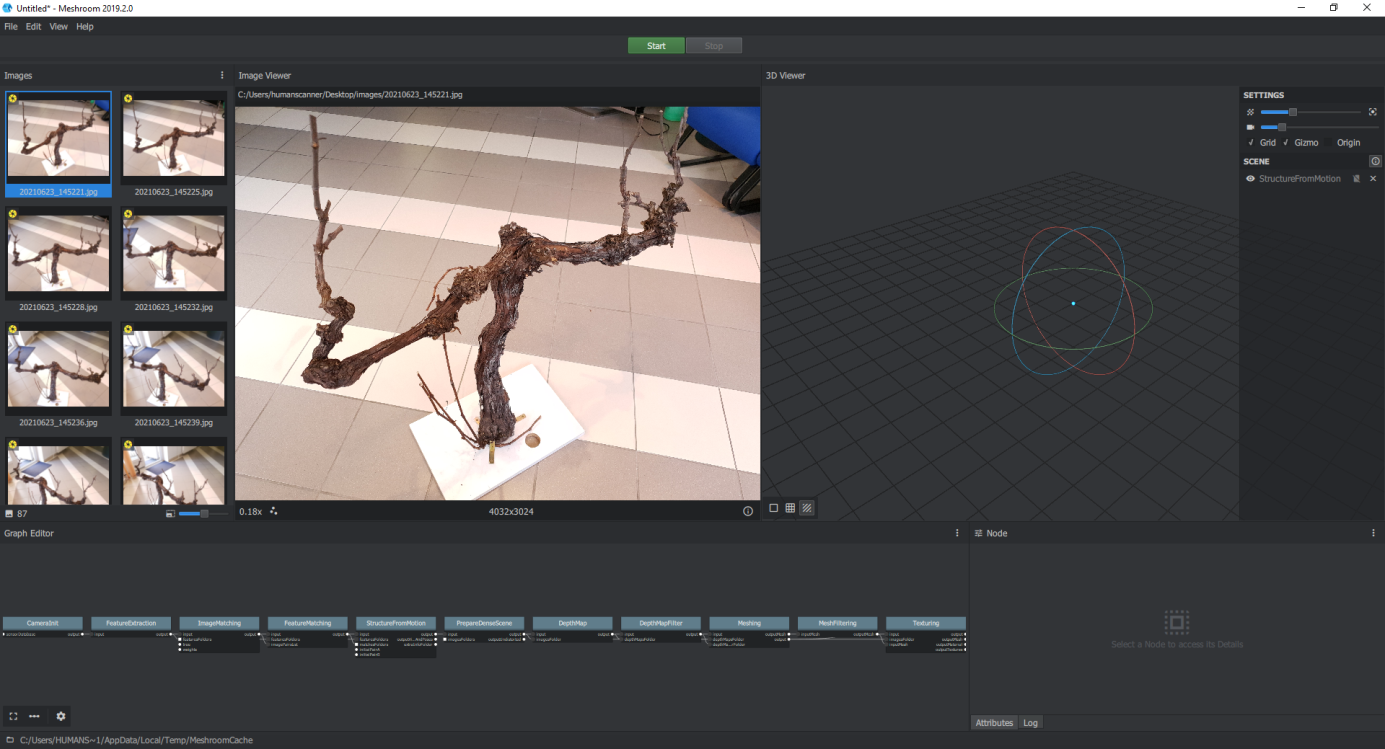
A Unity egy videójáték-motor, amelyet a Unity Technologies fejleszt. A Unity segítségével háromdimenziós illetve kétdimenziós videojátékokat, ezen kívül egyéb interaktív jellegű tartalmakat lehet létrehozni, például építészeti látványterveket, valós idejű háromdimenziós animációkat vagy szimulációkat. A projekt megnyitásához Unity 2020.3.3f1 vagy újabb verzióra van szükségünk. Ezt a Unity Hub-on belül letölthető



2. ábra, Unity Hub

## Meshroom

Ezzel az alkalmazással több 2D-s képből fotogrammetria segítségével 3D-s objektumot tudunk előállítani. A projektben használt 3D-s objektumok rendre 50-100 darab 2D-s színes képből lettek rekonstruálva.



3. ábra, Meshroom

## .OBJ

Ez a fájlkiterjesztés az egyik legelterjedtebb formátum a 3D-s objektumok leírására. A Unity natívan támogatja ezt a fájltípust. Ha később új modelleket szeretnénk importálni a projektbe, akkor azoknak is ilyen formátumban kell lenniük.



4. ábra, OBJ fájlkiterjesztés

## Blender

A Blender egy szabad, nyílt forráskódú, háromdimenziós grafikai program. Felhasználható animációs filmek, vizuális effektek létrehozására, használják művészek, alkalmazható 3D nyomtatásra, interaktív 3D alkalmazások és videojátékok létrehozására. Az általunk rekonstruált 3D-s objektumokat ezzel az alkalmazással tudjuk forgatni és szétdarabolni.

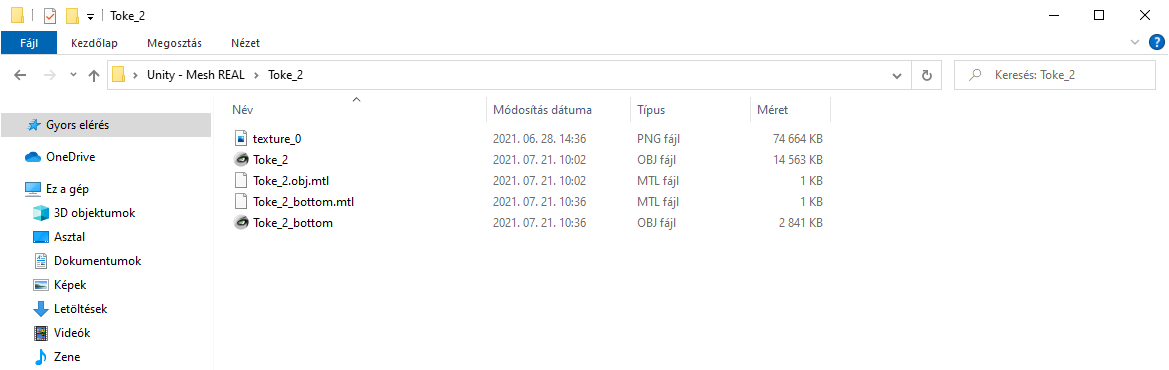


5. ábra, Blender

# 3D-s modell importálása Unity-be

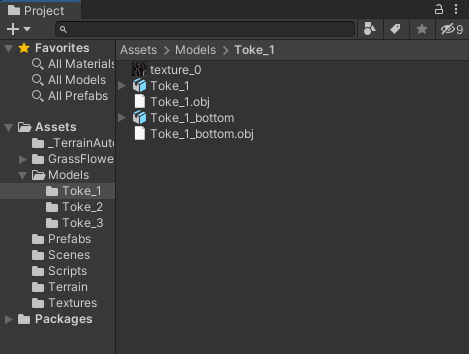
Ha szeretnénk importálni egy 3D-s modellt Unity-be előtte még szét kell bontanunk két különálló részre. Az egyik rész a mi esetünkben a szőlőtőke modellnek az alsó része, ami köré szeretnénk, hogy a bounding box kerüljön. Ennek a résznek a levágását Blenderben oldottam meg, majd külön obj kiterjesztésű fájlba mentettem. A másik részbe pedig a „maradék” kerül, ami az alsó részen kívül volt. Ezt is külön obj fájlba mentettem ki.

Így néz ki egy feldarabolt 3D-s szőlőtőke obj modell a fájlrendszerben:



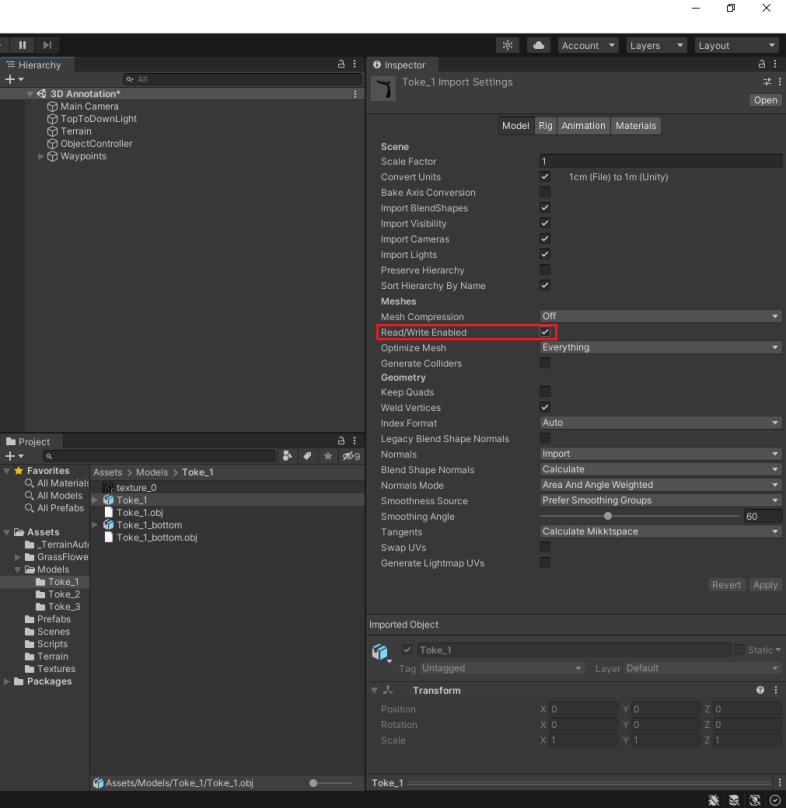
6. ábra, szétdarabolt obj modell

Az importálandó objektumokat egyszerűen húzzuk be a Unity editor szerkesztői felületébe. Ilyenkor automatikusan importálódni fognak a projektbe. Érdemes a modelleknek külön mappát létrehozni az átláthatóság végett. Fontos, hogy be kell importálnunk az obj fájlhoz tartozó mtl fájlt és ha van akkor a textúrát is.



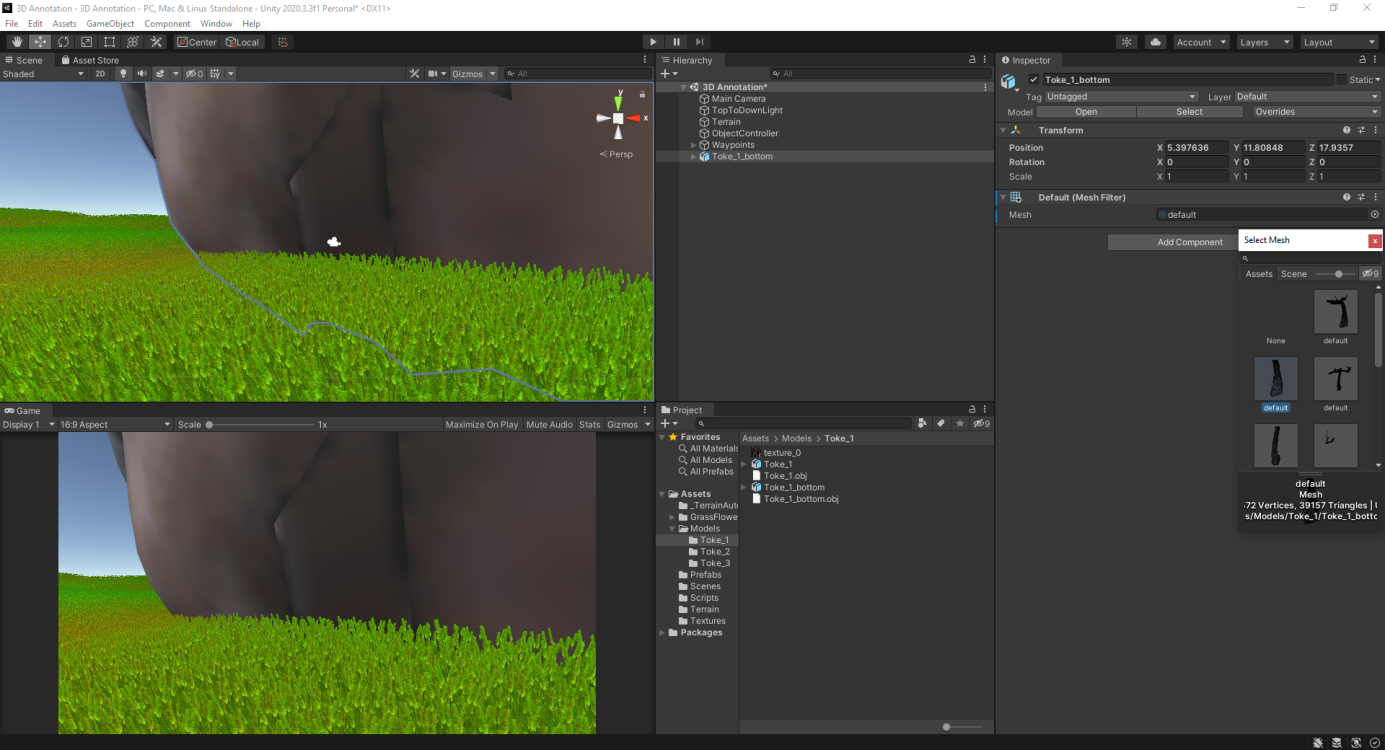
7. ábra, importált modellek

Következő lépésben engedélyeznünk kell a beimportált obj fájlokon az írás/olvasást. Ha ez nincs beállítva, akkor nem tudjuk lekérdezni a fájloz tartozó csúcs koordináta értékeit.



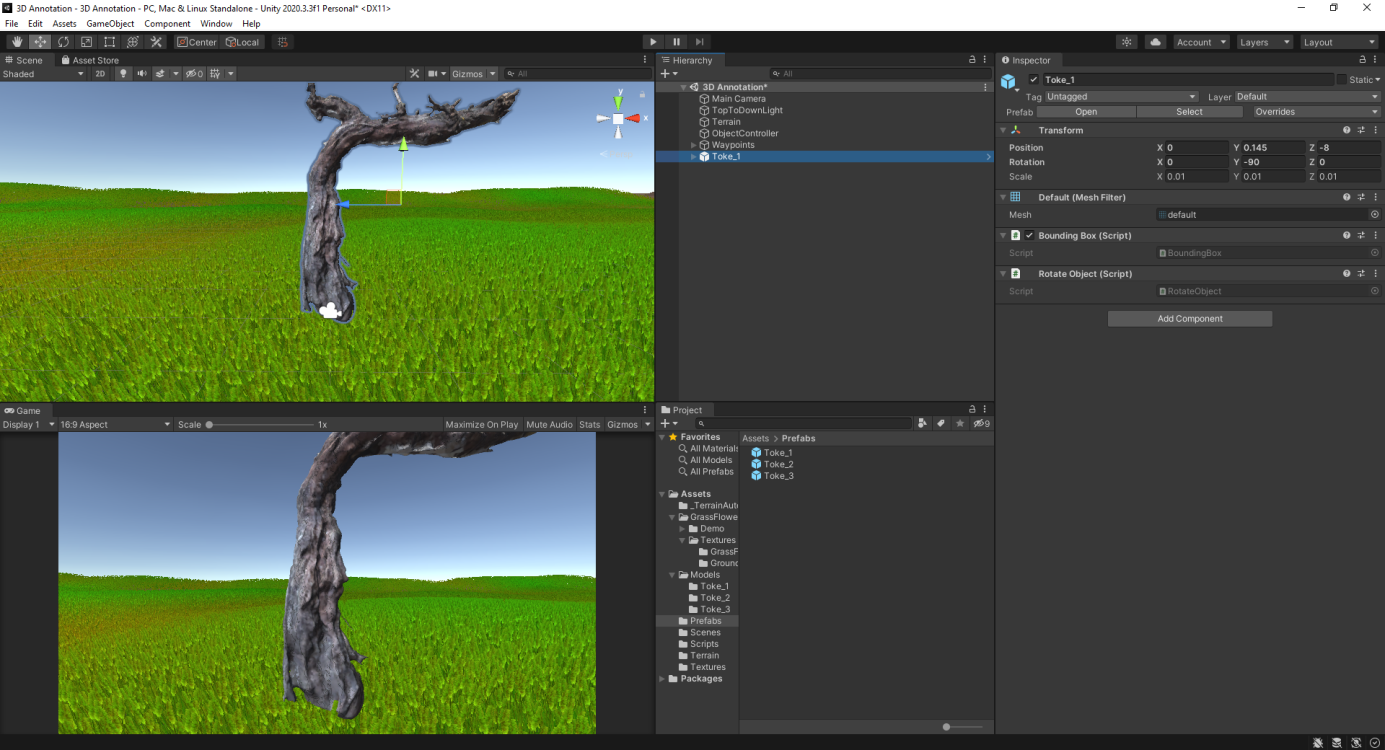
8. ábra, obj Read/Write Enabled

Ezek után húzzuk be a modelleket a hierarchiába, majd adjuk hozzájuk komponensként Mesh Filter-t, majd a Mesh Filter-hez adjuk hozzá az adott obj fájlhoz tartozó default Mesh-t.



9. ábra, Mesh Filter

A modell alsó részébe húzzuk bele a modell felső részét, azaz adjuk meg gyerekének. Így a forgatás során együtt fognak forogni a modellek, de mégis külön-külön tudjuk kezelni őket. Transzformáljuk, forgassuk és skálázzuk úgy az objektumot, hogy az számunkra megfelelő legyen, majd a kész objektumból csináljunk prefab-et, oly módon, hogy behúzzuk Hierarchy részből az általunk beállított modellt a Project mappába. Majd a Hierarchy részből töröljük ki a modellt.



10. ábra, kész 3D-s prefab

# Unity projekt megoldások

## Bounding box kiszámítása

A bounding box kiszámítását a BoundingBox.cs script-ben oldottam meg. Ezt a script-et minden prefab modellhez hozzá kell rendelni. A script-ben a GUI2dRectWithObject függvény átmegy a modell pontjain és megnézni, hogy melyek a minimum és maximum x, y koordináták. A WorldToGUIPoint pedig a Unity világ koordinátákból állít elő pixel koordinátákat. Ezek a pixelkoordináták lesznek később átkonvertálva YOLO-nak megfefelő adattá.

public Rect GUI2dRectWithObject(GameObject go)

{

Vector3[] vertices = go.GetComponent<MeshFilter>().mesh.vertices;

float x1 = float.MaxValue, y1 = float.MaxValue, x2 = 0.0f, y2 = 0.0f;

foreach (Vector3 vert in vertices)

{

Vector2 tmp = WorldToGUIPoint(go.transform.TransformPoint(vert));

if (tmp.x < x1) x1 = tmp.x;

if (tmp.x > x2) x2 = tmp.x;

if (tmp.y < y1) y1 = tmp.y;

if (tmp.y > y2) y2 = tmp.y;

}

Rect bbox = new Rect(x1, y1, x2 - x1, y2 - y1);

return bbox;

}

Vector2 WorldToGUIPoint(Vector3 world)

{

Vector2 screenPoint = Camera.main.WorldToScreenPoint(world);

screenPoint.y = (float)Screen.height - screenPoint.y;

return screenPoint;

}

## Forgatás megoldása

A forgatást a RotateObject.cs script-ben oldottam meg. A RotateGameObject egy paraméterben megkapott értékkel forgatja el az objektumot. Ezt a script-et a forgatandó prefab-nek kell komponensként megadni.

public void RotateGameObject(float rotationDegree)

{

transform.Rotate(0, -rotationDegree, 0);

}

## Pillanatkép készítésének megoldása

A forgatások közötti pillanatképek készítését a TakeSnapshot.cs script-ben oldottam meg. A CaptureSnapshot paraméterként megkap egy kamerát, egy fájl indexet egy snapshot indexet és a felbontásokat. Ezen paraméterek függvényében készíti el a pillanatképet. Ezt a script-et a Main Camera-hoz rendeltem.

public void CaptureSnapshot(Camera snapshotCamera, int fileIndex, int snapshotIndex, int resWidth, int resHeight)

{

RenderTexture rt = new RenderTexture(resWidth, resHeight, 24);

snapshotCamera.targetTexture = rt;

Texture2D screenShot = new Texture2D(resWidth, resHeight, TextureFormat.RGB24, false);

snapshotCamera.Render();

RenderTexture.active = rt;

screenShot.ReadPixels(new Rect(0, 0, resWidth, resHeight), 0, 0);

snapshotCamera.targetTexture = null;

RenderTexture.active = null;

Destroy(rt);

byte[] bytes = screenShot.EncodeToPNG();

string filename = "C:/Users/humanscanner/Desktop/Pictures/" + fileIndex + "\_" + snapshotIndex + ".png";

System.IO.File.WriteAllBytes(filename, bytes);

}

## Fájl készítésének megoldása

A szöveges file készítését a CreateFile.cs script-ben oldottam meg. A script-ben 2 függvény van. A CreateTextFile a fájl elkészítéséért felel, a DarknetLabelFormat pedig a YOLO-nak megfelelő formátum előállításáért.

public void CreateTextFile(Rect boundingBox, int objectIndex, int fileIndex, int imageWidth, int imageHeight)

{

string filePath = "C:/Users/humanscanner/Desktop/Pictures/" + objectIndex + "\_" + fileIndex + ".txt";

System.IO.File.WriteAllText(filePath, DarknetLabelFormat(boundingBox, imageWidth, imageHeight));

}

string DarknetLabelFormat(Rect boundingBox, int imageWidth, int imageHeight)

{

float xMin = boundingBox.xMin;

float yMin = boundingBox.yMin;

float w = boundingBox.width;

float h = boundingBox.height;

float w\_img = imageWidth;

float h\_img = imageHeight;

float xCenter = (xMin + w / 2) / w\_img;

float yCenter = (yMin + h / 2) / h\_img;

w = w / w\_img;

h = h / h\_img;

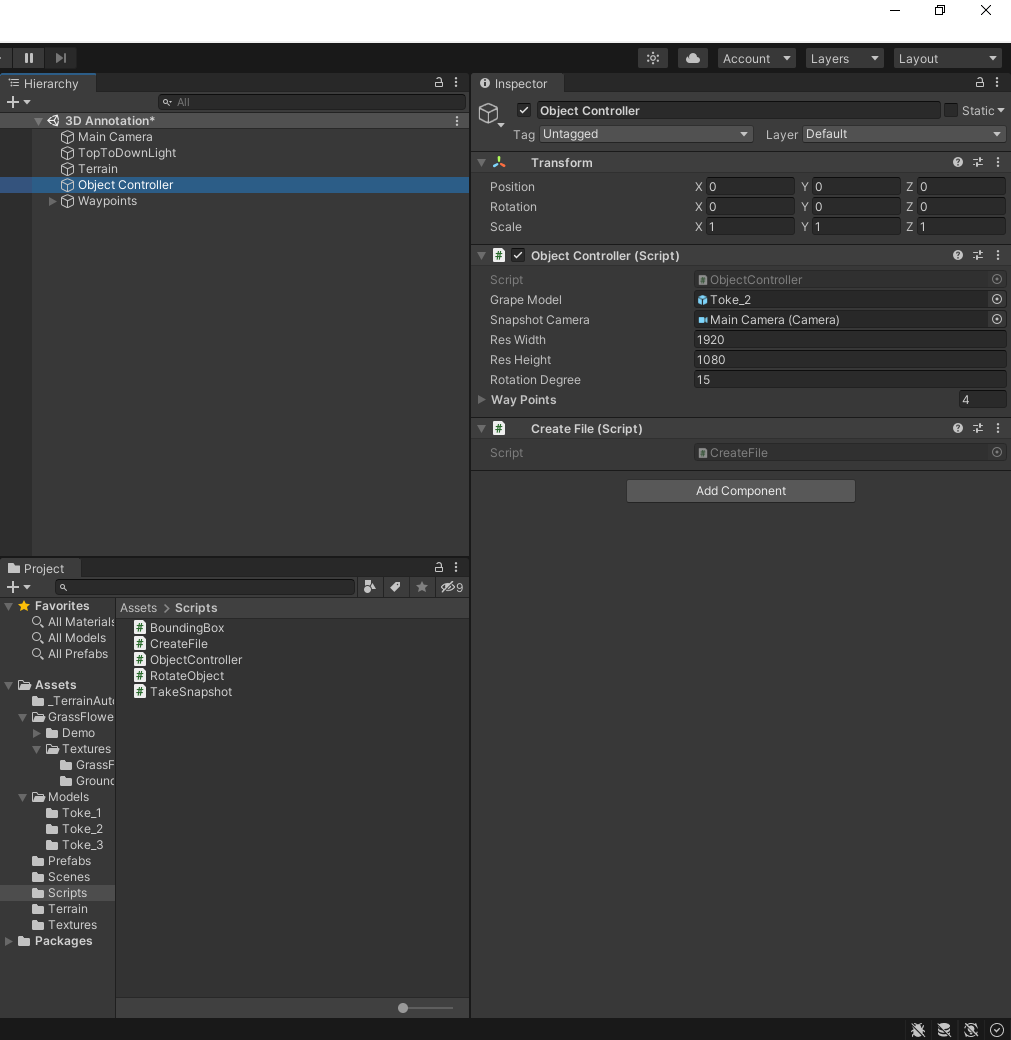
string spaceing = " ";

return (0 + spaceing + xCenter + spaceing + yCenter + spaceing + w + spaceing + h).Replace(",",".");

}

## Folyamat vezérlése

A forgatást, a pillanatkép készítését és a fájl készítését egy helyről lehet kezelni, méghozzá a Object Controller objektumból. Itt megadhatjuk, hogy melyik 3D-s objektumról szeretnénk képeket készíteni, milyen felbontásban és hány fokonként.



11. ábra, Object Controller

Az ObjectController.cs script vezérlésért felelős része az alábbiakban látható. A Start metódusban inicializálom az összes változót. Itt kerül meghatározásra, hogy hány részből fog állni egy teljes körbeforgatás. A RotateObjectAndTakeSnapshot függvényben pedig a pillanatkép és fájl készítés történik.

void Start()

{

grapeInstance = Instantiate(grapeModel, grapeModel.transform.position, grapeModel.transform.rotation);

takeSnapshot = snapshotCamera.GetComponent<TakeSnapshot>();

boundingBox = grapeInstance.GetComponent<BoundingBox>();

createFile = GetComponent<CreateFile>();

rotateObject = grapeInstance.GetComponent<RotateObject>();

int rotationNumber = (int)Mathf.Ceil(360 / rotationDegree);

RotateObjectAndTakeSnapshot(rotationNumber);

}

void RotateObjectAndTakeSnapshot(int rotationNumber)

{

for(int viewPointCounter = 0; viewPointCounter < wayPoints.Length; viewPointCounter++)

{

Camera.main.transform.position = wayPoints[viewPointCounter].transform.position;

Camera.main.transform.rotation = wayPoints[viewPointCounter].transform.rotation;

for (int rotationCounter = 0; rotationCounter < rotationNumber; rotationCounter++)

{

takeSnapshot.CaptureSnapshot(snapshotCamera, viewPointCounter, rotationCounter, resWidth, resHeight);

createFile.CreateTextFile(boundingBox.GUI2dRectWithObject(grapeInstance), viewPointCounter, rotationCounter, resWidth, resHeight);

rotateObject.RotateGameObject(rotationDegree);

}

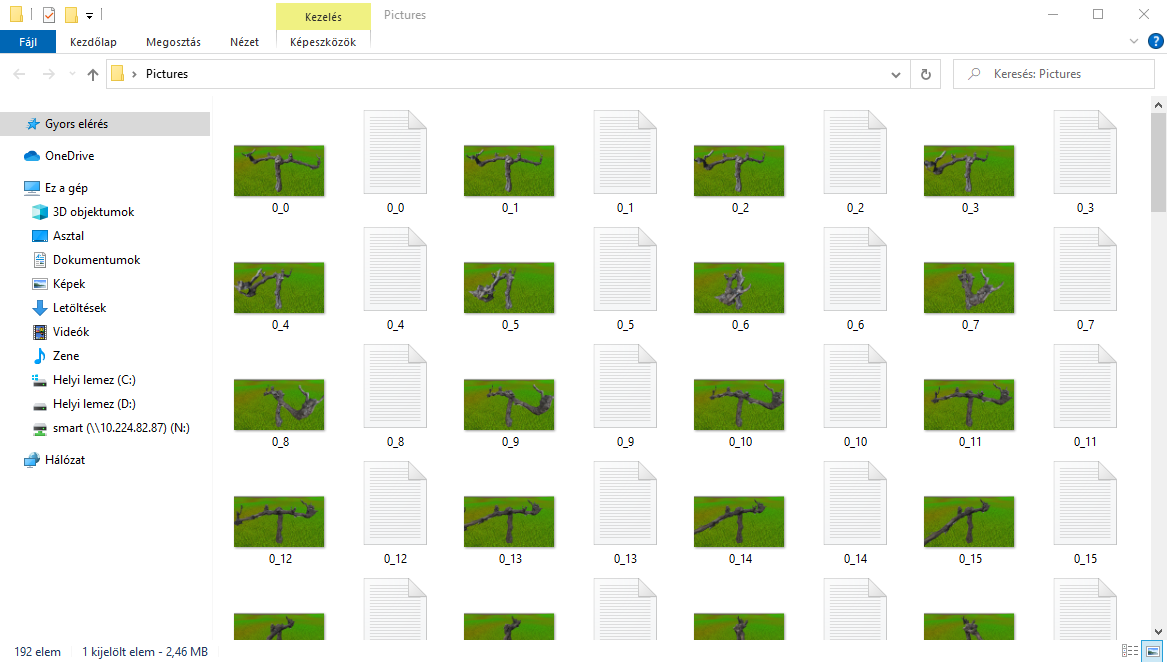
}

}

# Elért eredmények

Ha beállítottuk a forgatási szöget és kiválasztottuk a megfelelő 3D-s modellt, akkor indítsuk el a Unity editor-t a play gombra kattintva. Beállítástól függően pár másodperctől, akár több percig is eltarthat egy szőlőtőke annotálása. Szerintem a legoptimálisabb, 10-15°-onként forgatni a szőlőtőkét. Ekkor fél percig tart az annotálás.

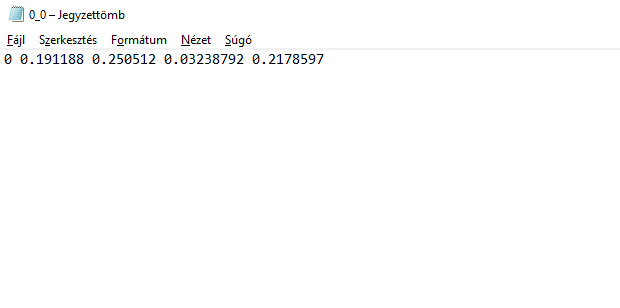
A folyamat végén a megadott mappában lesznek a képek és a hozzátartozó szöveges fájlok, amik a tőke körüli bounding box adatait tárolják a YOLO neurális hálónak értelmezhető formátumban. Ezeket már lehet bármiféle változtatás nélkül használni YOLO neurális hálózat feltanításához.



12. ábra, Annotált eredmények



13. ábra, Unity által készített kép



14. ábra, YOLO formátum

# Ábrajegyzék

[1. ábra, annotált tőke Unity-ben 1](#_Toc78463571)

[2. ábra, Unity Hub 3](#_Toc78463572)

[3. ábra, Meshroom 4](#_Toc78463573)

[4. ábra, OBJ fájlkiterjesztés 4](#_Toc78463574)

[5. ábra, Blender 5](#_Toc78463575)

[6. ábra, szétdarabolt obj modell 5](#_Toc78463576)

[7. ábra, importált modellek 6](#_Toc78463577)

[8. ábra, obj Read/Write Enabled 6](#_Toc78463578)

[9. ábra, Mesh Filter 7](#_Toc78463579)

[10. ábra, kész 3D-s prefab 7](#_Toc78463580)

[11. ábra, Object Controller 10](#_Toc78463581)

[12. ábra, Annotált eredmények 11](#_Toc78463582)

[13. ábra, Unity által készített kép 12](#_Toc78463583)

[14. ábra, YOLO formátum 12](#_Toc78463584)