Finalisation du développement de prototypes destinés à la conception de systèmes agroforestiers

MASSON Gabriel

LEMIERE Laetitia & JAEGER Marc

2022-2023

Table des matières

0 Remerciements

Ce travail a bénéficié d'une aide de l'État gérée par l'Agence Nationale de la Recherche au titre du programme d'Investissements d'Avenir portant la référence ANR-16-CONV-0004.

Je remercie mes encadrants Marc Jaeger et Laetitia Lemiere pour avoir été d'incroyables maîtres de stage, de par la confiance qu'ils m'ont accordée et leur bienveillance.

De plus, je remercie l'UMR AMAP, son personnel ainsi que Clément Dahan et Houssem Triki qui ont été d'une sympathie sans précédent.

1 Introduction

L'agroforesterie est une méthode agricole mélangeant les arbres avec la culture avec parfois les animaux. Elle a grandement été remis en avant ces dernières années pour faire face aux problèmes de la monoculture. Effectivement, la monoculture entraîne, entre autre, une baisse de bio-diversité et génère les émissions à effets de serre. A l'inverse, l'agroforesterie favorise les interactions au sein d'une même parcelle en favorisant la biodiversité.

Mon stage s'est déroulé dans un laboratoire nommé UMR AMAP. C'est une unité interdisciplinaire qui s'occupe d'acquérir des connaissances sur les plantes et les végétations dans le but d'en apprendre plus sur la réponse des écosystèmes à la perturbation de leur équilibre. J'ai travaillé sur des outils ayant pour but de promouvoir l'agroforesterie.

Un des problèmes actuels, que les agriculteurs rencontrent quand ils entreprennent la création d'une parcelle agroforestière, est la conception de celle-ci. Contrairement à l'agriculture classique, l'agroforesterie demande de plus grandes connaissances dans le choix des plantes et du placement de celles-ci pour augmenter son efficacité. Il n'existe, actuellement, que des guides techniques expliquant les étapes à suivre pour accompagner l'agriculteur dans son projet d'agroforesterie mais pas d'ouvrages regroupant les plantes et leur arrangement favorisant tels but (augmentation de la production, amélioration de la biodiversité...). Une méthode pour aider les agriculteurs, est les ateliers participatifs. Ces ateliers regroupent plusieurs personnes pour réfléchir au projet et proposer un plan de la futur parcelle. Néanmoins, ces ateliers pour aider à la conception des parcelles existent mais sont rare. De plus, il n'existe pas d'outil de simulation pour les aider durant ces ateliers.

Mon stage a pour but de finaliser le développement des prototypes destinés à la conception de systèmes agroforestiers afin d'en autoriser un usage stable. Ils rendront l'élaboration d'un système agroforestier plus simple durant les ateliers de conception. Le prototype actuel nous permet de visualiser une parcelle en réalité augmentée et la croissance des plantes qui la constitue. De plus j'ai complété la documentation technique et j'ai rédigé une documentation utilisateur avec une forme descriptive et tutoriel avec des exemples.
La documentation technique devra être complétée et finalisée et je proposerai une documentation utilisateur avec une forme descriptive et tutoriel avec des exemples.

Je commencerai par vous présenter le projet dans son état actuel, pour ensuite vous expliquer les améliorations que j'ai apportées au projet. Pour continuer, j'entraînerai un réseau de neurones pour permettre la reconnaissance des formes que je rajouterai au premier projet. En second temps, je m'occuperai de développer un projet de "Vitrine numérique" permettant de visualiser un système agroforestier. Pour finir, je travaillerai sur une démonstration qui sera présentée à un éventail de personnes et évaluée.

2 Contexte

Le Cirad est un organisme qui a pour mission le développement rural des pays tropicaux et subtropicaux par des actions de recherche.

Il a été fondé en 1984, deux centres existent en France : celui qui m'accueille, à Montferrier-sur-Lez et l'autre à Montpellier.

Quelques chiffres :

1130 salariés	**544** cadres scientifiques	**315** doctorants chaque année
150 stagiaires chaque année	**340** scientifiques partenaire permanents	**6000** missions par an

Le CIRAD est implémenté dans plusieurs pays tel que :
— Costa Rica
— Trinidad
— Guyane
— Cameroun
— Kenya
— France

Le personnel de l'UMR AMAP, le laboratoire auquel j'étais rattaché physiquement durant mon stage, est composé de

49 chercheurs enseignants	**7** Chercheurs et Ingénieurs associés	**21** doctorants chaque année
34 techniciens et ingénieurs	**10** posts doctorants	**17** autre CDD

Durant ce stage, j'ai fait équipe avec un autre stagiaire, Clément DAHAN, qui fait son stage de fin d'études d'école d'ingénieur généraliste. Son sujet portait sur la conception de la visualisation dans une application en réalité augmentée. J'ai donc implémenté les maquettes qu'il avait développées en amont. On avait pour mission de finaliser les outils numériques permettant la conception de systèmes agroforestiers de Laetitia LEMIERE actuellement en fin de thèse. Son sujet portait sur la réalité augmentée pour accompagner la conception de systèmes agroforestiers.

2.1 Définition de l'existant

2.1.1 Réalité augmentée

Avant de vous présenter l'application existante à mon arrivée, je vais vous présenter la réalité augmentée.

La réalité augmentée fait partie de la grande famille des réalité étendues, elle se caractérise par la visualisation d'un environnement réel avec l'ajout d'informations ou de modèles fictifs.

Figure 1 – Exemple de réalité augmentée (AR)

2.1.2 Projet existant

Le projet auquel je me suis intéressé durant ce stage se nomme "AR Indoor" et se compose de 3 grandes parties schématisée à la Figure 2.

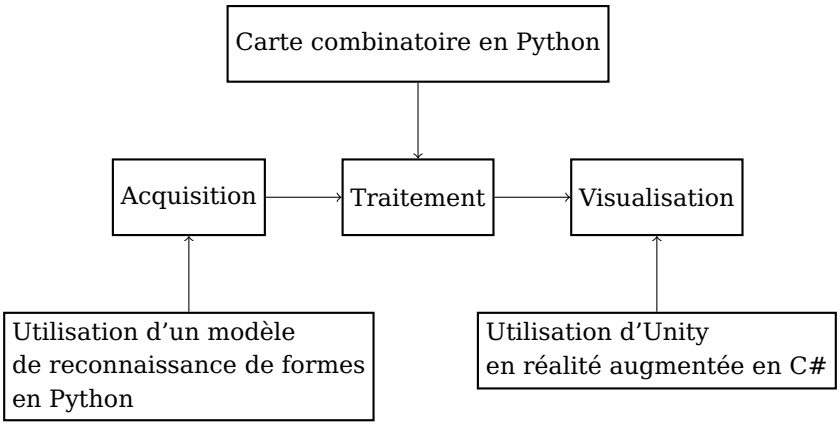


Figure 2 – Schéma des parties

Je ne me suis seulement intéressé sur la partie d'`acquisition` et de `visualisation` car le module de traitement nécessitait des notions sur une structure de données appelé "carte combinatoire". Le développement des différents modules ont des bases solides, mais fonctionnaient indépendamment. On utilisait des fichiers comportant des données pour pouvoir les lier artificiellement, ce qui n'était pas pratique. Ils devraient fonctionner ensemble pour permettre la meilleure expérience utilisateur possible.

2.1.3 Acquisition

Le module d'acquisition a pour objectif de reconnaitre les différents éléments d'une maquette de systèmes agroforestier. Chaque aimant de la Figure 3, correspond à un élément du système. Il reposait sur un réseau de neurones qui n'est pas parfait. En effet, il n'a pas reconnu toutes les formes de l'image visible à la Figure 3. Il manque le rectangle noir au centre, le cercle bleu en bas à gauche et la flèche noire à droite.

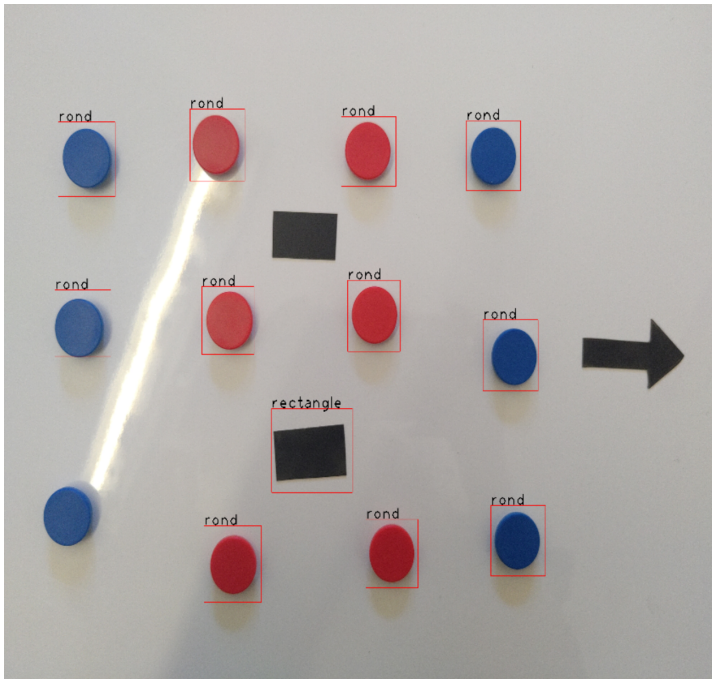
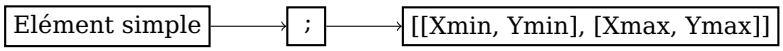


Figure 3 – Résultat du modèle existant

2.1.4 Visualisation

Le module de visualisation doit montrer le système agroforestier en réalité augmentée. Pour cela, grâce à un fichier texte, écrit à la main, comportant toutes les coordonnées de ce qui compose la scène.(Figure 4a) le module projette la scène comme sur la Figure 4b. On voit que le fichier sur la Figure 10 est composé de lignes qui se ressemblent. Une ligne peut soit représenter un élément simple (comme un arbre) soit une zone (comme une surface de culture).
Un élément simple se décompose en :

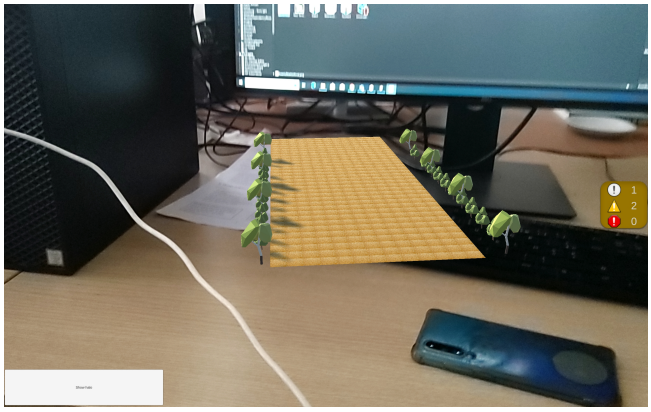
Pour une zone est décrite par :



Un script s'occupe de convertir ces lignes en éléments sur la scène. Puisque le module n'est pas connecté au module d'acquisition, la création du fichier à la main n'est clairement pas viable et prend beaucoup de temps.

```
TRZAX;10;10;[[2,0], [15,33]]
1BETG;1;0
1BETG;16;0
1BETG;1;8
1BETG;16;8
1BETG;1;16
1BETG;16;16
1BETG;1;24
1BETG;16;24
RUBID;1;2
RUBID;16;2
RUBID;1;4
RUBID;16;4
RUBID;1;6
RUBID;16;6
RUBID;1;10
RUBID;16;10
RUBID;1;12
RUBID;16;12
RUBID;1;14
RUBID;16;14
RUBID;1;18
RUBID;16;18
RUBID;1;22
RUBID;16;22
```

(a)



(b)

Figure 4 – Visualisation d'une parcelle en réalité augmentée

2.2 Installation de l'environement de travail

Avant de débuter le développement, j'ai dû installer quelques logiciels indispensables :

- Unity 2021.3
- Suite Jetbrain :
  - Rider
  - PyCharm
- Git

Unity est un moteur de jeu multiplateforme, un des plus répandus dans l'industrie du jeu vidéo. Ici, il sera utilisé avec le Framework **AR foundation** qui gère la réalité augmentée.

AR Foundation est un framework (c'est-à-dire un ensemble d'outils) développé par Unity qui permet de créer des expériences de réalité augmentée multiplateformes. Il offre des fonctionnalités telles que le suivi de la position et de l'orientation, la détection de surfaces, le placement d'objets virtuels dans le monde réel, ainsi que la prise en charge de l'interaction avec l'environnement en réalité augmentée (AR). Il permet aux développeurs de créer des applications AR de manière efficace et portable.

Rider & PyCharm sont des IDE (Intelligent Developpement Environement), PyCharm pour Python et Rider pour du C#, langage utilisé dans Unity.

Git & GitHub seront utilisé pour du "versionning". Cela permet de gérer les changements apportés à un code source au fil du temps.

2.2.1 Composition Unity

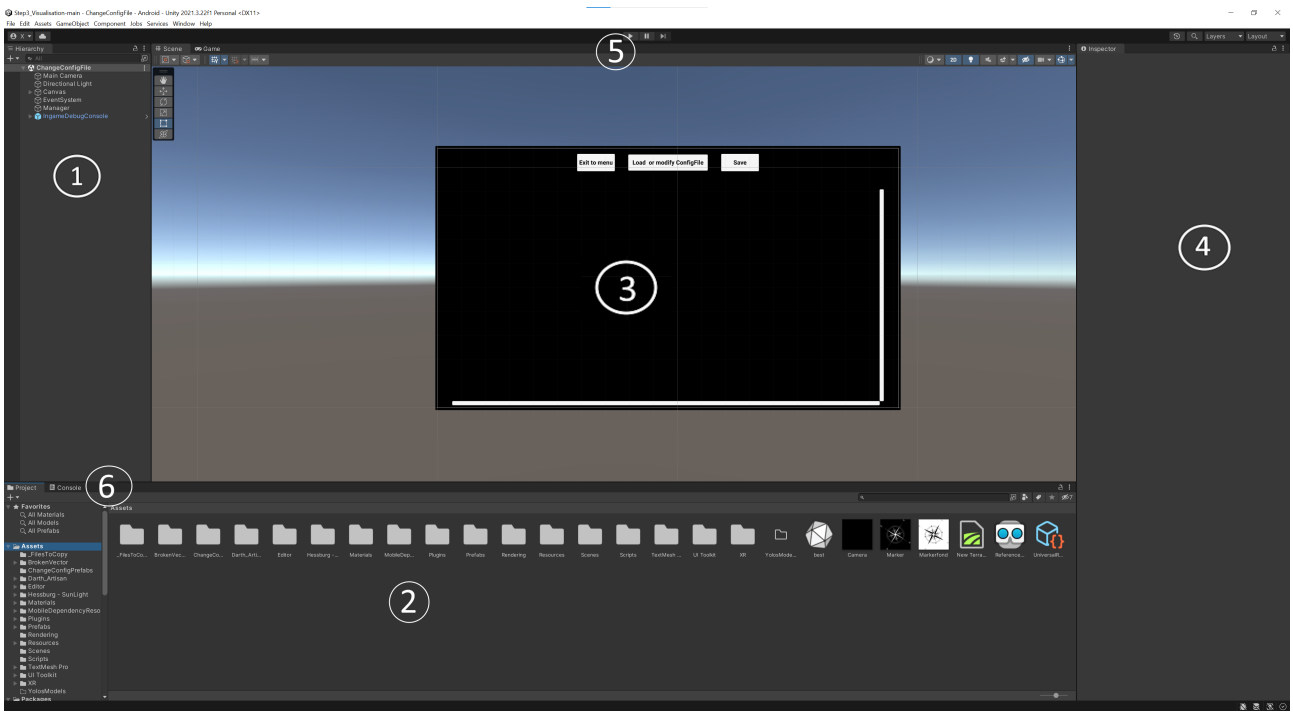


Figure 5 – Composition Unity

1	"Hierarchy", on y trouve les objets qui composent la scène
2	"Projet", gestionnaire de fichiers, objets, scripts qui composent le projet
3	"Scene", visualisation de la scène
4	"Inspector", apprait les paramètres de l'objet sélectionné
5	Boutons pour lancer la scène, la mettre en pause
6	Console, on peut y trouver les erreurs et les messages retournés par l'éditeur ou les scripts

2.2.2 Ouverture du projet "AR indoor"

Après avoir obtenu l'accès au dépôt GitHub du projet, j'ai pu le télécharger et commencer à le tester. Il était composé d'un seul menu (Figure 6) correspondant au menu principal de l'application et d'une visualisation en réalité augmentée.

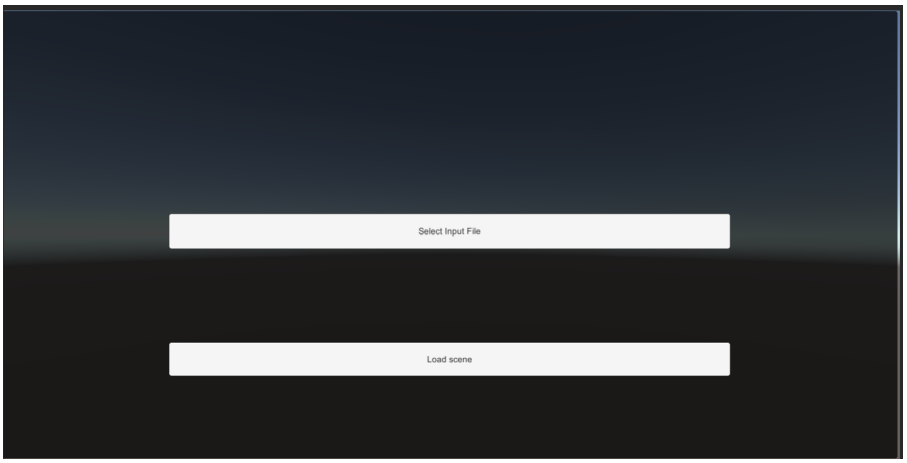


Figure 6 – Menu du projet

Figure 7 – Les scènes existantes sur le projet

Le projet est composé de 3 scènes (Figure 7) chacune représente une partie du projet. On peut voir la scène du menu renommée "Openning" et la scène qui gère la réalité augmentée "ARSceneTracking".

La scène qui comporte le menu est attribuée comme "défault" pour qu'au moment du lancement de l'application l'utilisateur arrive directement sur le menu. Depuis ce menu, l'utilisateur peut sélectionner la scène à visualiser grâce à l'interface de chargement de fichier d'entrée (Figure 8) décrit précédemment (partie visualisaiton)

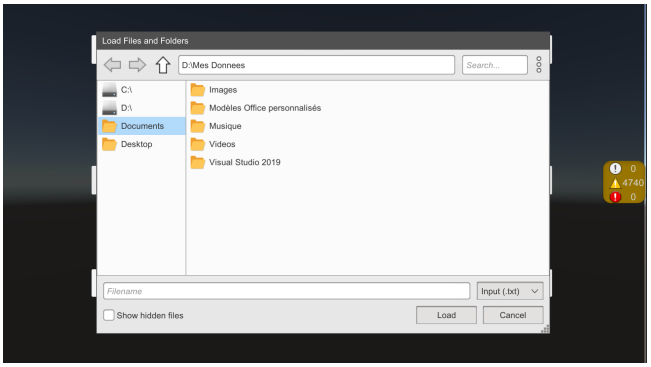


Figure 8 – Interface de chargement des données d'entrée

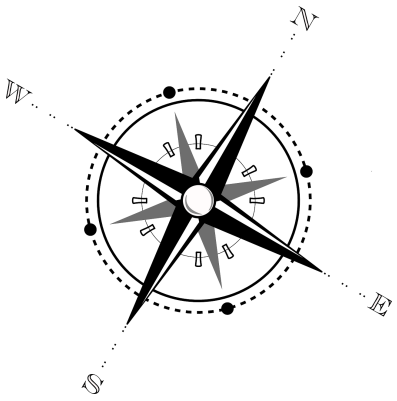


Figure 9 – Marqueur

Une fois le fichier sélectionné et que l'utilisateur appuie "Load scene", l'utilisateur voit ce qui se trouve devant lui grâce à sa caméra mais sans la scène en réalité augmentée. Pour pouvoir faire apparaître la parcelle, l'utilisateur doit scanner, un marqueur (Figure 9 représentant le fichier .txt chargé au préalable.

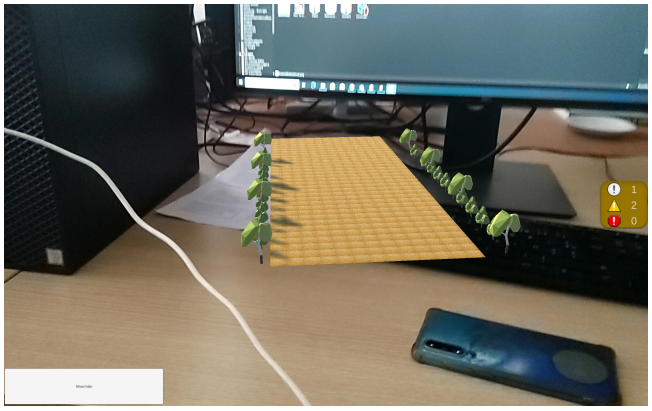


Figure 10 – Menu de chargement de fichier d'entrée

3 Missions et tâches accomplies

Mes missions se sont répartis sur 3 projets : AR Indoor que j'ai complété, Vitrine numérique que j'ai créé et une visualisation pour aider Clément Dahan. Je vais d'abords vous présenter mes ajouts dans AR Indoor, ensuite vous parler de la Vitrine numérique et enfin, vous montrer la visualisation crée en collaboration avec Clément.

3.1 Rotation aléatoire des arbres

3.1.1 Définition de la mission

Pour ma première mission, j'ai eu comme instruction de devoir appliquer une rotation aléatoire sur les arbres qui composent la scène pour augmenter le réalisme de celle-ci. Pour cela, j'ai attribué un attribut (appelé "Layer" dans Unity) "Tree" aux arbres afin de sélectionner uniquement les arbres parmi tous les éléments de la scène. J'ai également rajouté les deux lignes suivantes dans la fonction qui fait apparaître les arbres sur la scène qui appliquent une rotation aléatoire aux arbres.

```
1     if (visu.layer == LayerMask.NameToLayer("Tree"))
2           visu.transform.eulerAngles = new Vector3(0, Random.Range(0, 360), 0);
```

3.1.2 Problèmes rencontrés

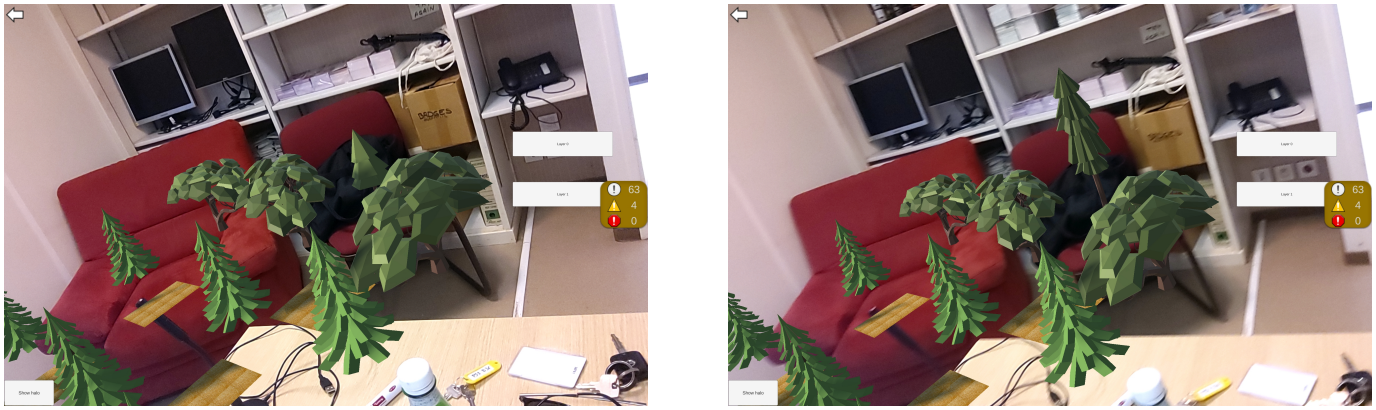
Il faut savoir que chaque objet dans la scène est composé d'un "Parent" invisible nommé "Plante". C'est dans ce "Parent" qu'on a imbriqué le modèle 3D de l'objet désiré (plante, arbre etc).

Toutes les 10 secondes, l'arbre ou la plante "grandit", on remplace donc l'ancien modèle de l'arbre par un nouveau modèle, c'est là où le parent de l'objet est utile, car c'est lui qui gère l'évolution de sa plante "Enfant" attribuée.

Le problème étant qu'au moment où la plante grandit on supprime l'ancien modèle pour le remplacer avec le nouveau, la rotation est donc réinitialisée. Pour y remédier, j'ai changé le programme que comporte l'objet parent :

```
1     newVisu.transform.rotation = current.transform.rotation;
```

Ici `newVisu` est le nouveau modèle, on lui attribue la rotation de l'ancien objet.



(a) Début de croissance (Arbre au fond de la parcelle)

(b) Fin de croissance (Arbre au fond de la parcelle)

Figure 11 – Rotation des arbres et leur croissance

Comme on peut le voir sur la Figure 11 les rotations sont sauvegardées et la croissance d'un arbre sur la parcelle n'atteint plus ceux qui ont une seule étape de croissance.

3.2 Fichier de configuration

Comme expliqué dans la section "2.1.2 Projet existant", le projet est composé de 3 grandes parties (acquisition, modélisation, visualisation). Pour pouvoir échanger des données entre ces 3 modules, un fichier de configuration sera indispensable, il servira à attribuer une forme et une couleur à un type d'objet.

On m'a donné un exemple de ce à quoi le fichier ressemble.

```
    {
  "Scene_size": [
    25,
    25
  ],
  "ratio_pixel_meter": 20,
  "plants": {
    "point": {
      "PEBAM": {
        "name": "Avocado",
        "3DFile": null,
        "color": "red",
        "figure": "circle"
      },
      "CIDLI": {
        "name": "Oak",
        "color": "black",
        "figure": "circle"
      },
      "FIUCA": {
        "name": "FIUCA",
        "color": "green",
        "figure": "circle"
      }
    },
    "area": {
      "TRZAX": {
        "name": "Wheat",
        "3DFile": null,
        "color": "blue",
        "figure": "rectangle"
      }
    }
  },
  "color": {
    "(255,0,0)": "red",
    "(0,255,0)": "green",
    "(0,0,255)": "blue",
    "(0,0,0)": "black",
    "(0,255,255)": "yellow"
  },
  "link": [
    "circle",
    "rectangle",
    "arrow"
  ]
}
```

Ce format de fichier est appelé `JSON` (JavaScript Object Notation), c'est un format de données très populaire auprès des développeurs.

Ici, mon objectif est de pouvoir afficher les données que contient ce fichier sur une interface graphique, sous Unity, permettant à l'utilisateur de changer les valeurs et de le sauvegarder.

Comme apparaît sur la Figure 12b un nouveau bouton "Select config file" a été rajouté. Il redirige sur une nouvelle scène.

(a) Ancien Menu

(b) Nouveau menu avec bouton "Select config file"

Figure 12 – Evolution du menu

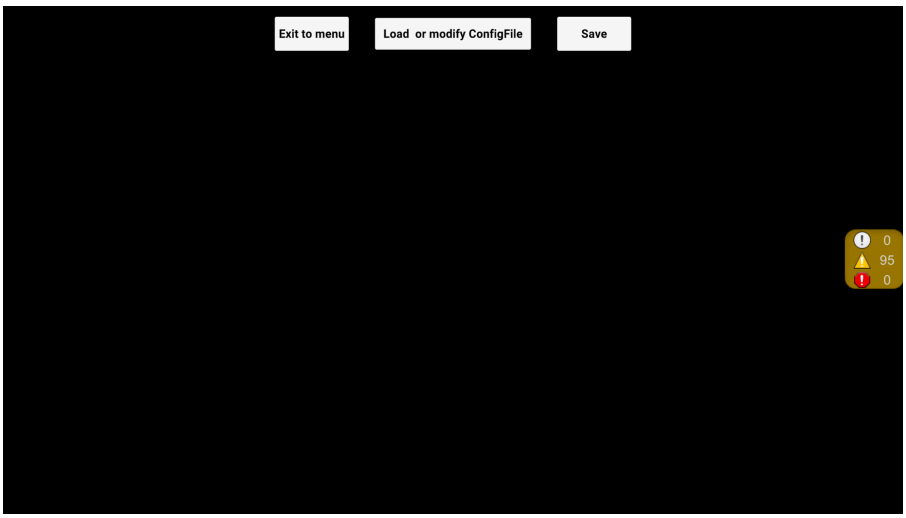


Figure 13 – Menu qui gère les actions pour le fichier de configuration

La Figure 13 est l'interface graphique que l'utilisateur rencontre à l'instant où la scène vient d'être chargée. On peut y voir 3 boutons :

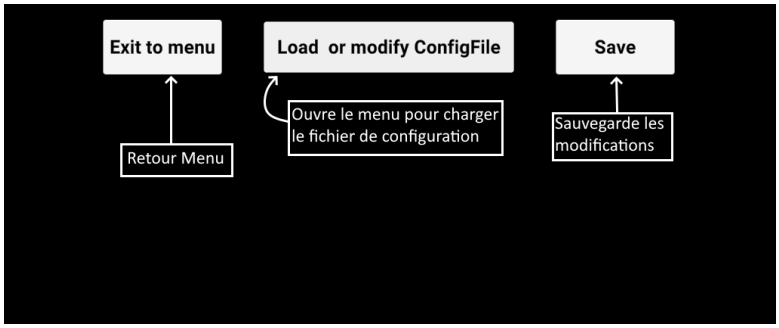


Figure 14 – Explications

Après avoir chargé le fichier de configuration, les données s'affichent sous une forme de tableau :

Nom de plante	Forme qui la représente	Couleur qui la représente

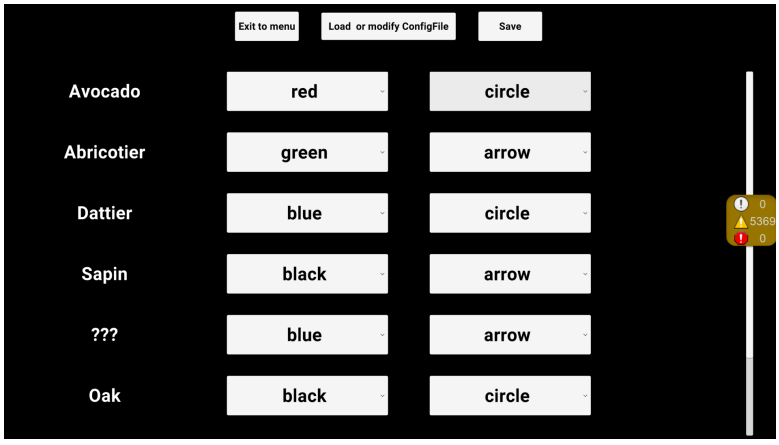


Figure 15 – Exemple de fichier de configuration chargé

Comme on peut le voir, sur la Figure 15 les valeurs du fichier JSON ont bien été affichées.
Pour y parvenir, j'ai créé une classe en C# qui fera référence au fichier de configuration et qui se remplit en fonction des données comprises dans le fichier.

```
public class PlantsDef
{
    public string name { get; set; }

    [JsonProperty("3DFile")]
    [CanBeNull] public string file { get; set; }
    [CanBeNull] public string color { get; set; }
    [CanBeNull] public string figure { get; set; }

    [JsonIgnore] public TMP_Dropdown dropdownColor { get; set; }

    [JsonIgnore] public TMP_Dropdown dropdownFigures { get; set; }
}

public class Plants
{
    public Dictionary<string, PlantsDef> point { get; set; }
    public Dictionary<string, PlantsDef> area { get; set; }
}
public class Config
{
    public int[] Scene_size { get; set; }

    public int ratio_pixel_meter { get; set; }

    public Plants plants { get; set; }

    public Dictionary<string, string> color { get; set; }

    public List<string> link { get; set; }

}
```

Pour convertir le fichier en .json en cette classe, une seule ligne suffit :

```
    _config = JsonConvert.DeserializeObject<Config>(json); //json <- texte lu du fichier
```

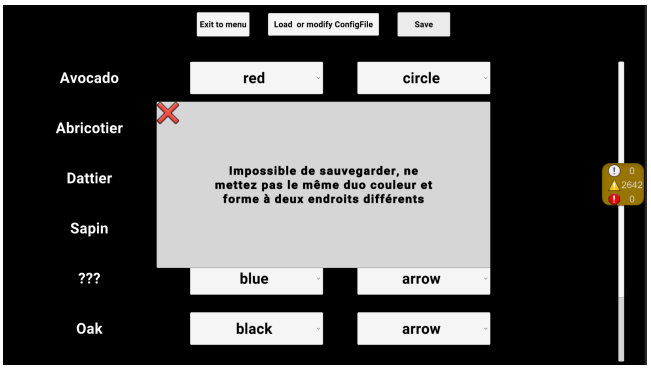


Figure 16 – Message d'erreur

Pour éviter la confusion, il n'est pas possible d'attribuer à deux plantes différentes un même couple de couleur et de forme. Sinon l'utilisateur sera prévenu avec une erreur comme sur la Figure 16.

3.3 Rajout de "Layers"

Un layer (couche), est un élément graphique ayant la capacité de se superposer. Ces layers peuvent être empilés dans un ordre précis, permettant ainsi de gérer la pertinence des informations disposés à l'écran.

Pour pouvoir faciliter la lecture de l'information, j'ai rajouté des couches contenant de l'information grâce à des couleurs sur la parcelle qui est visualisée en réalité augmentée. Le cahier des charges est simple, au clic d'un bouton, un layer apparaît au-dessus du terrain actuel avec des couleurs pouvant porter de l'information. Sur la Figure 17 un layer (rectangle rouge) est au-dessus du terrain et est composé de cases rouges, oranges et transparentes.

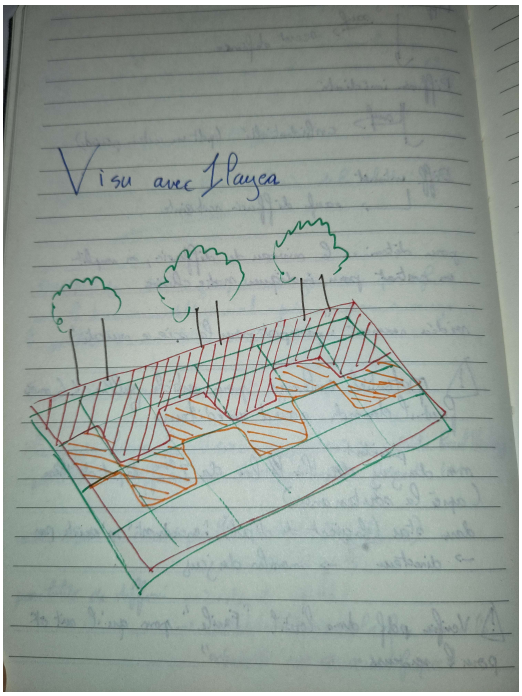


Figure 17 – Croquis

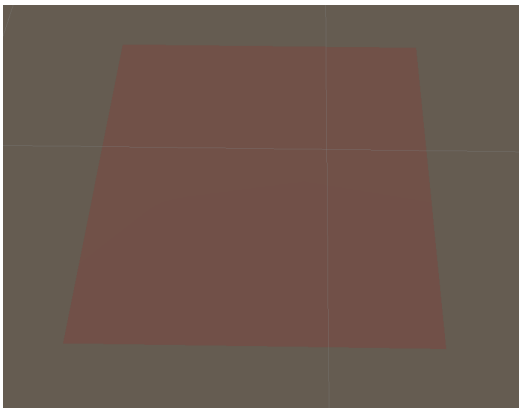
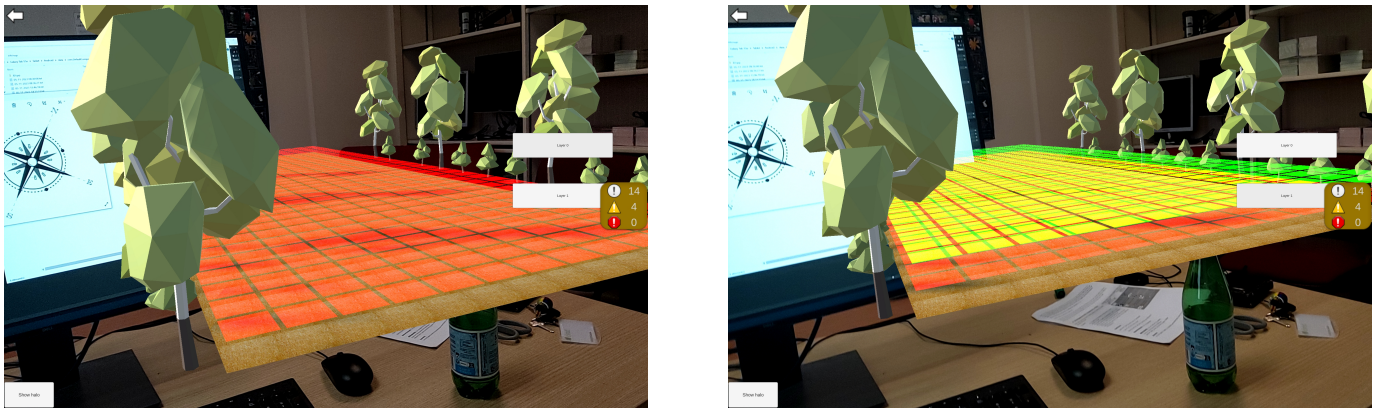


Figure 18 – Layer

J'ai commencé par créer un rectangle rouge et transparent sur Unity (Figure 18). J'ai ensuite créé le programme qui au clic sur un bouton génère, sur chaque parcelle, un layer d'une certaine couleur apparaît. J'ai rajouté la fonctionnalité d'empiler des layers, donc pour chacune des couches un bouton leur est attribuée.



(a) Avec un seul Layer

(b) Avec les deux layers

Figure 19 – Rendu du rajout des layers

Comme vous pouvez le voir sur la Figure 19, les couleurs ne portent pas d'information. Cela restera comme ci-dessus, car il faudrait implémenter des modèles qui calculent les informations que l'utilisateur voudrait voir apparaître (comme la croissance des racines). Actuellement, une étude est cours, menée par Clément DAHAN, qui s'assure que la visualisation est lisible.

3.4 Yolo & entraînement de réseau de neurones

3.4.1 Déroulement

YOLO (You Look At Once) est un modèle de détection d'objet en temps réel. Les réseaux de neurones sont composés de trois éléments principaux : les couches d'entrées, les couches cachées et la couche de sortie. Les connexions entre les neurones sont définies par des poids qui sont ajustés pendant l'apprentissage du réseau, permettant ainsi d'optimiser les performances du modèle. Les couches d'entrées reçoivent les données initiales, les couches cachées effectuent des calculs intermédiaires, et la couche de sortie produit les résultats finaux, tels que des prédictions ou des classifications.

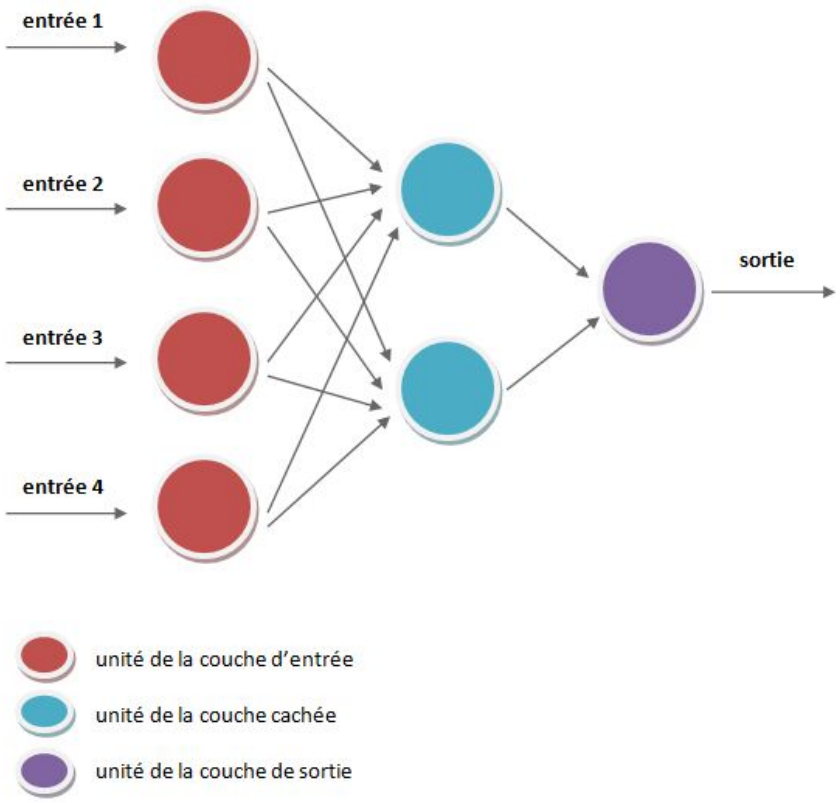


Figure 20 – Exemple d'architecture d'un réseau de neurones

Ici, le modèle devra détecter les classes suivantes qui correspondent aux différentes formes que peuvent prendre les éléments dans les maquettes :

— Carré
— Cercle
— Flèche

Pour pouvoir obtenir mon réseau de neurones il a fallu suivre les étapes décrites dans la Figure 21

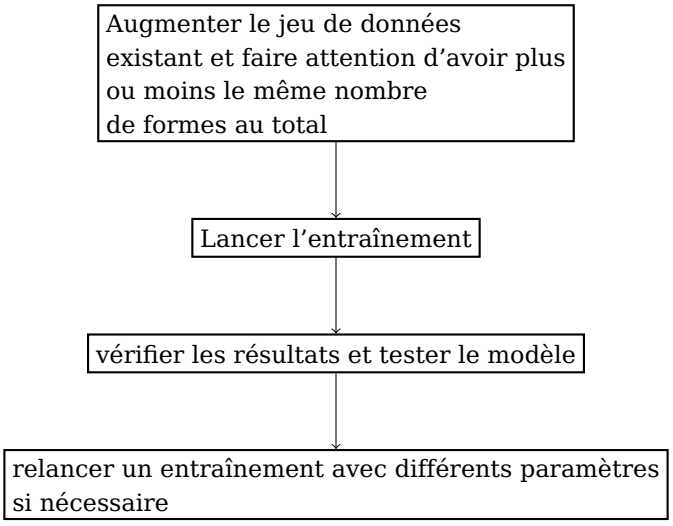
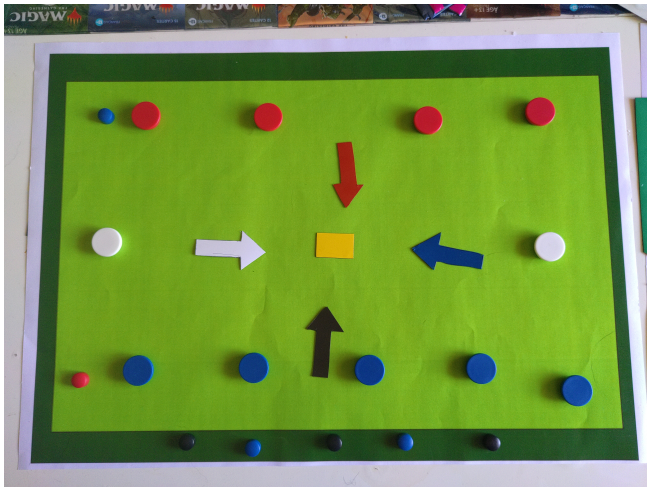


Figure 21 – Étapes à suivre pour obtenir un modèle fonctionnel.

3.4.2 Jeu de données

Pour pouvoir entraîner un réseau de neurones, il faut mettre à disposition du programme beaucoup d'images variées. Une image doit être accompagnée de son fichier texte qui répertorie la position des objets que le modèle doit reconnaître.



(a) Image qui va être utilisée comme entraînement

```
0 0.776474 0.832544 0.069418 0.052064
0 0.760066 0.657800 0.069418 0.050170
1 0.499432 0.514294 0.065632 0.070996
2 0.489335 0.351477 0.095923 0.120220
2 0.298750 0.497728 0.159031 0.064370
2 0.645841 0.534173 0.148933 0.053957
2 0.486811 0.689038 0.098448 0.122113
```

(b) fichier .txt qui comporte les informations requises

Figure 22 – Image et son fichier .txt

Comme on peut le voir sur la figure 22b chaque ligne est composée du numéro de la classe, de la position normalisée, par rapport à la dimension des images (ici 800x800px), des coordonnées "X" et "Y" du centre de la case, puis de sa longueur et de sa largeur.

Pour éviter de créer ce fichier manuellement j'ai utilisé un outil en ligne nommé Make Sens. Cette étape s'appelle le "Labelling".

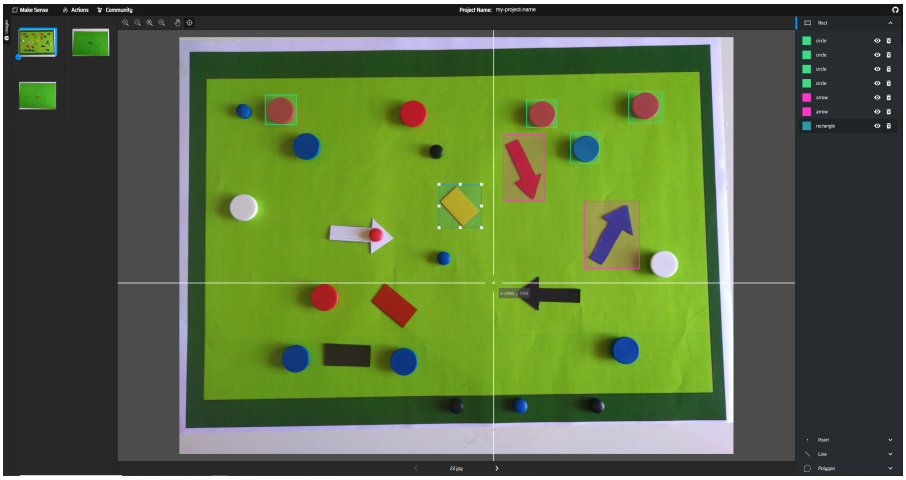


Figure 23 – Interface de Make Sens

Sur la Figure 23 on peut voir que j'ai manuellement encadré les formes avec une couleur différente qui correspond à leur "nom" de forme. Après avoir fait cette étape sur toute les images voulues, il suffit d'exporter le projet en YOLO et les fichiers textes seront créés automatiquement.

J’ai créé un petit programme pour connaître le nombre total de chaque classe :

	Avant	Après
Cercles	3589	3780
Rectangles	698	1385
Flèches	393	795

Le nombre de classes n’a pas totalement été rééquilibré mais cela n’a pas posé de problème, car dans les maquettes il y avait un déséquilibre dans l’utilisation des formes.

3.4.3 Entraînement

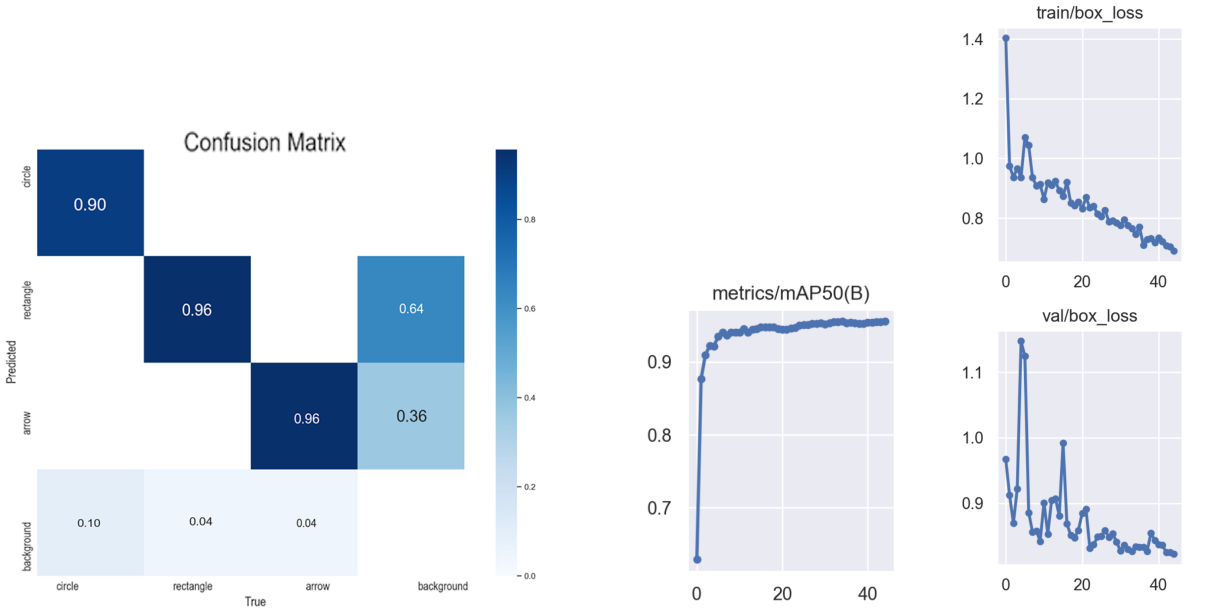
J'ai commencé par séparer toutes mes images et leurs fichiers texte en 3 dossiers : **Test, Train, Valid**. Aucune image ne doit être en double dans aucun dossiers. J'ai utilisé la méthode de "80-20" pour séparer mes images : 80% dans le dossier **train** et 20% dans le dossier **valid**. Cependant j'ai déplacé quelques images du dossier **train** pour le dossier **test** pour tester mon réseau de neurones.

Sur PyCharm, j'ai installé les librairies requises (Ultralytics pour Yolo et TorchCuda pour l'entraînement sur le GPU) pour l'entraînement du modèle. J'ai ensuite créé le fichier de configuration qui est utilisé par le programme, il contient les informations suivantes :

— le chemin d'accès pour les images
— le nombre total de classes (ici 3)
— Le nom des formes

```yaml
path: D:\GABRIEL_Masson\DeepL\Dataset
train: D:\GABRIEL_Masson\DeepL\Dataset\train
test: D:\GABRIEL_Masson\DeepL\Dataset\test
val: D:\GABRIEL_Masson\DeepL\Dataset\valid

nc: 3

names: ["circle", "rectangle", "arrow"]
```

3.4.4 Résultats



(a) Matrice de confusion

(b) Accuracy

(c) Train loss & Validation Loss

Figure 24

Il existe 3 grandes métriques pour analyser un réseau de neurones de type YOLO :

— Accuracy (Exactitude) : L'accuracy mesure la proportion d'échantillons correctement classés parmi tous les échantillons. C'est le rapport entre le nombre d'échantillons correctement classés et le nombre total d'échantillons. L'accuracy donne une idée globale de la performance du modèle, mais elle peut être trompeuse si les classes sont déséquilibrées.

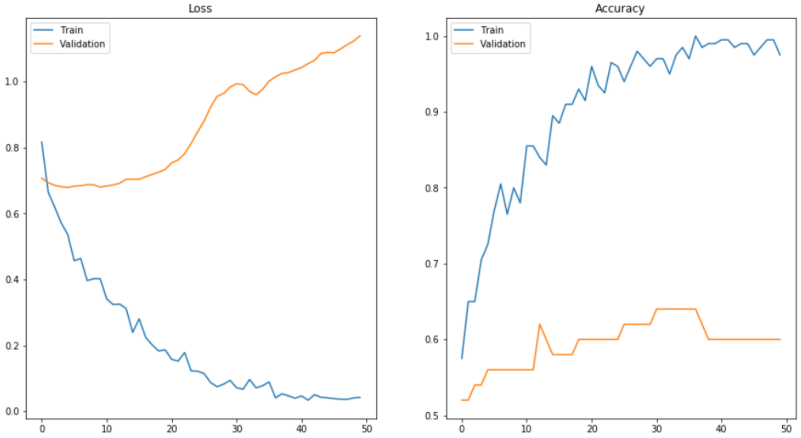
— Precision (Précision) : La précision est le rapport entre le nombre d'échantillons correctement classés comme positifs (vrais positifs) et le nombre total d'échantillons classés comme positifs (vrais positifs + faux positifs). Elle mesure la capacité du modèle à ne pas classifier à tort un échantillon comme positif.

— Recall (Rappel) : Le recall est le rapport entre le nombre d'échantillons correctement classés comme positifs (vrais positifs) et le nombre total d'échantillons qui sont réellement positifs (vrais positifs + faux négatifs). Il mesure la capacité du modèle à identifier correctement tous les échantillons positifs.

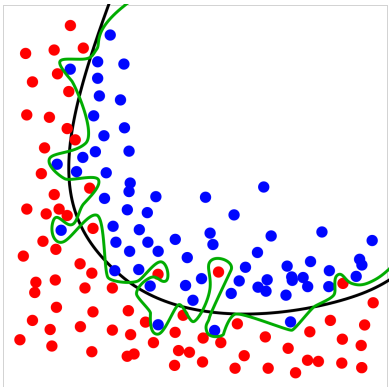
Après avoir attendu 1h30, j'ai obtenu les résultats suivants

Sur la Figure 24b on peut y voir la courbe d'accuracy qui tend vers 1. Une exactitude élevée signifie que le modèle est capable de classifier correctement la plupart, voire tous les exemples de test. On peut en conclure que le modèle suffira pour nos besoins.

Comme on peut le voir sur la Figure 24a la matrice de confusion est une table qui répertorie les prédictions du modèle par rapport aux vraies étiquettes de classes. Elle permet de visualiser et d'analyser les erreurs de classification faites par le modèle. Ici, le modèle confond un peu les rectangles et les flèches avec le fond de l'image, mais après avoir testé le modèle sur plusieurs photos ça n'arrivait que rarement, et seulement sur des images complexes. De plus, les courbes de "Loss" de la Figure 24c(*fonction perte est une mesure mathématique qui évalue l'écart entre les prédictions d'un modèle et les valeurs réelles, et elle est utilisée pour guider l'optimisation du modèle.*) tendent vers 0. Cela nous indique que le modèle n'est pas en surapprentissage (Overfitting). C'est suffisant pour affirmer que le modèle sera fonctionnel pour notre simple utilisation.



(a) Loss et accuracy



(b) Exemple d'un modèle d'overfitting (en vert)

Figure 25 – Overfitting

On peut voir l'exemple d'overfitting sur la Figure 25. Le surapprentissage peut être traduit par l'incapacité du modèle de généraliser ses prédictions car il se focalise trop sur les données d'apprentissage. La Figure 25b est une exemple d'overfitting. Avec la courbe verte qui correspond aux prédictions d'un modèle en overfitting et la courbe noire qui correspond à la courbe désirée.

Comme on peut le voir sur la Figure 26, des rectangles ont été rajoutés par-dessus l'image originale, le nombre à côté du nom de la forme est le niveau de confiance, entre 0 et 1. Il correspond à la probabilité que l'objet soit la classe auquel le nom de la forme fait référence.

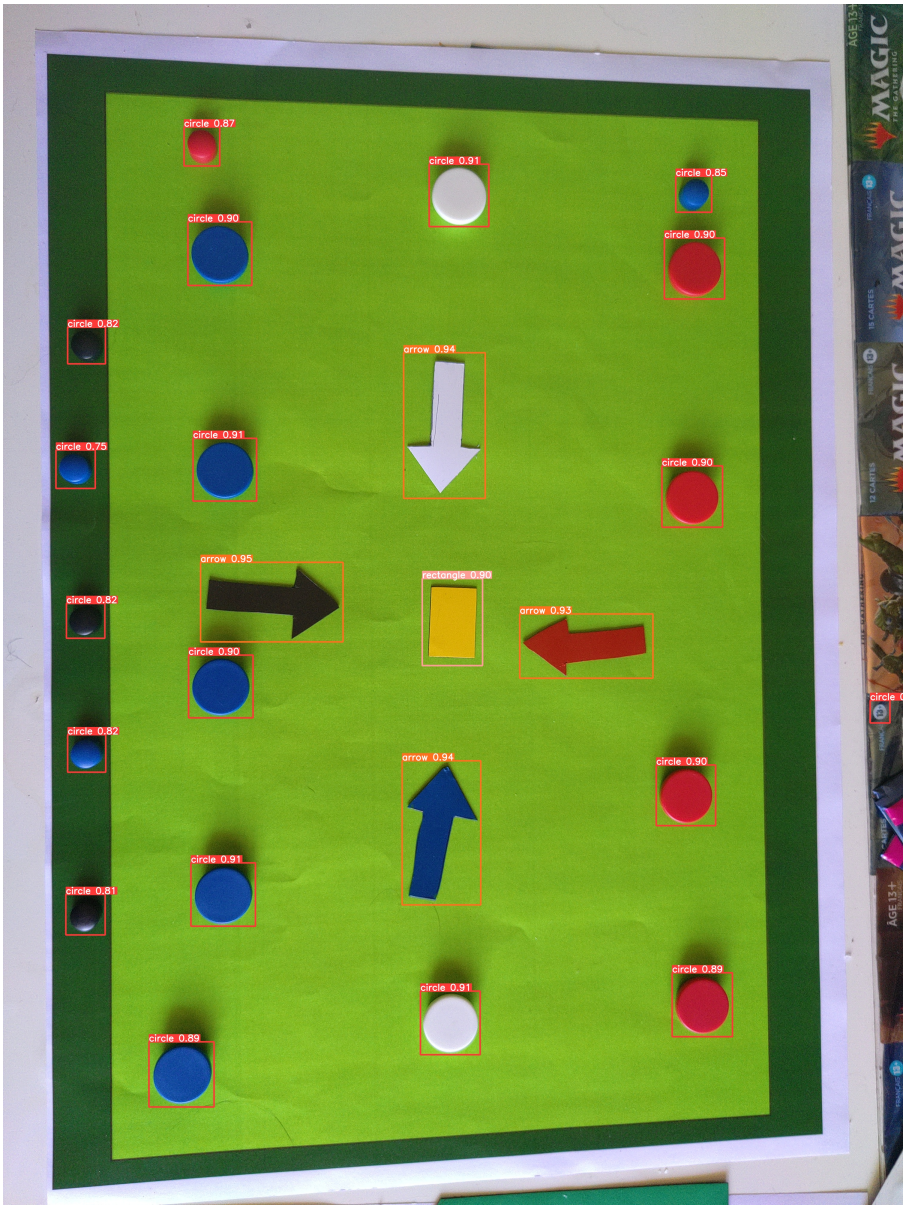


Figure 26 – photo analysée par le modèle

Vous pouvez observer sur la Figure 27 des erreurs à gauche. Comme l'a indiquée la matrice de confusion, le modèle confond un peu l'arrière-plan avec des formes. Mais cela est facilement traitable en filtrant la probabilité de la forme. Les cercles ont des probabilités qui tournent autour de 80% alors que les dessins à gauche sont au maximum à 50%. Ce sera possible en traitant les données en sortie du modèle. De plus, l'application devra être utilisée sur un plan qui ne comporte pas, ou très peu, de motifs.

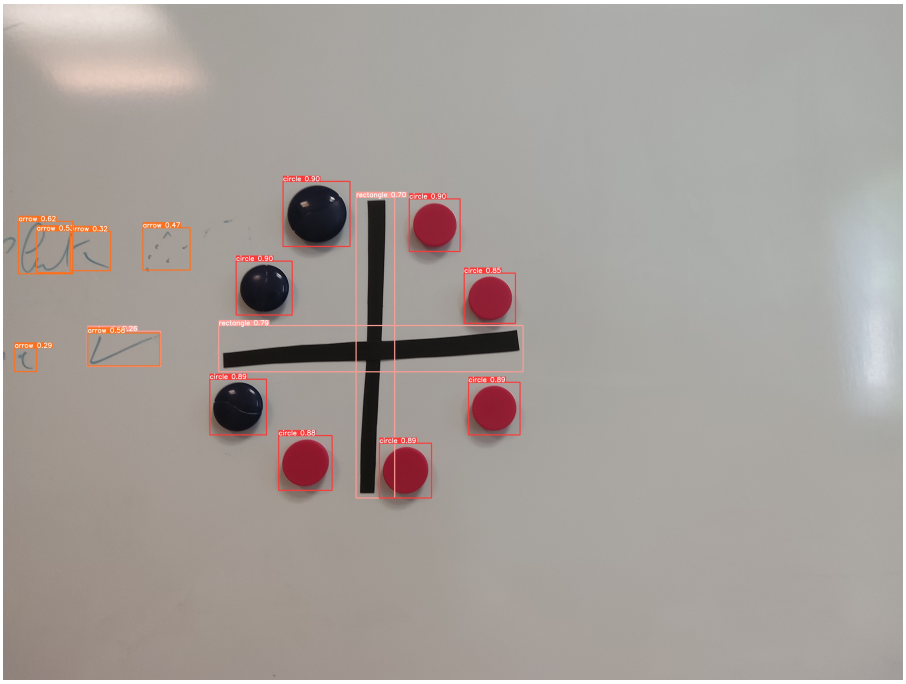


Figure 27 – Photo analysée par le réseau de neurones sur Unity

3.5 Rajout du modèle sur AR Indoor

Pour cette dernière mission, j'ai eu comme cahier des charges de rajouter le modèle de reconnaissance de formes au projet "AR Indoor". Le but est donc que l'utilisateur puisse prendre une photo de sa parcelle conçue avec les 3 formes à sa disposition avec son smartphone, l'utilisateur doit bien sur configurer l'application avec le bon code forme, couleur et plante avant cette étape. Une fois la photo prise et analysée, l'application devra également projeter la parcelle en réalité augmentée.

Unity ne peut comprendre les réseau de neurones sous le format ONNX. Or mon réseau est au format PyTorch. J'ai donc commencé par exporter le réseau de neurones au format ONNX, qui est un format que la librairie "Barracuda" comprend. Elle permet à Unity d'utiliser les modèles d'intelligence artificielle.

3.5.1 Analyse de l'image sur Unity

J'ai rajouté un bouton sur le Menu qui redirige l'utilisateur vers un écran et qui lui permet, soit de prendre une photo avec sa caméra, soit de charger une photo de son smartphone.J'ai chargé un fichier de configuration avant d'aller sur ce menu, on peut voir l'état de l'application en bas à droite de la Figure 28 avec la phrase "Fichier de configuration bien chargé".

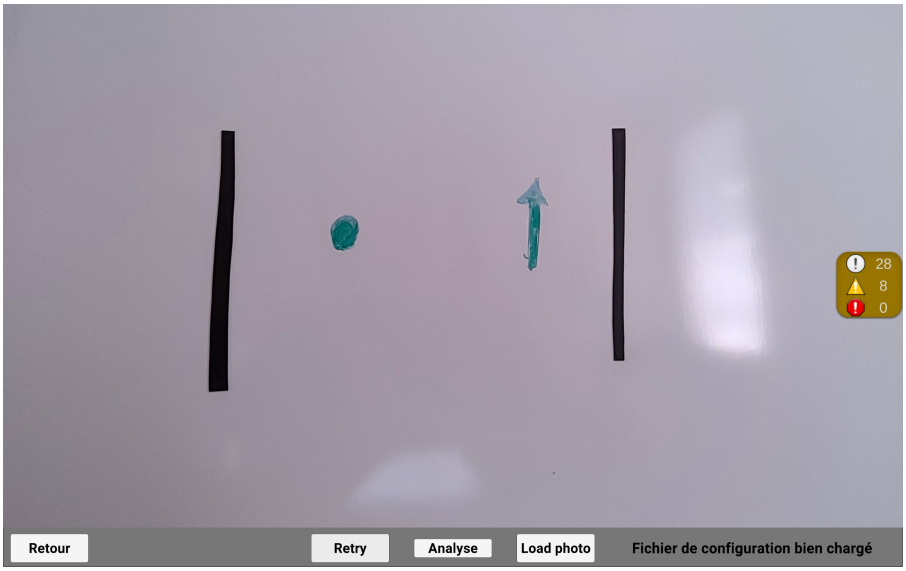


Figure 28 – Menu d'analyse de l'image
AR Indoor

Après avoir choisi la photo souhaitée, il suffit de cliquer sur le bouton `Analyse`. Le réseau de neurones cherche alors à reconnaitre les différentes formes et un algorithme reconnait les couleurs des forme. Les rectangles gris de la Figure 29 indiquent que le modèle a trouvé une forme à cet endroit.

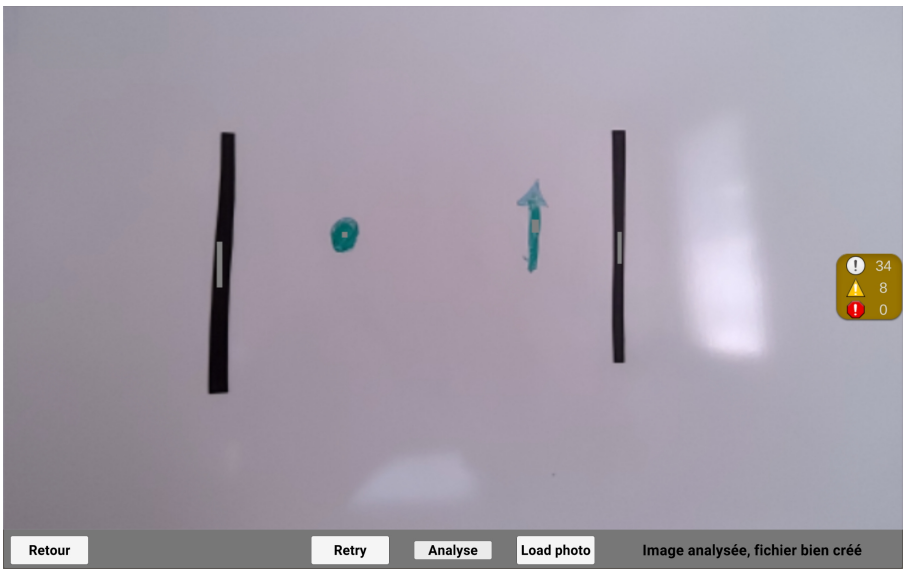


Figure 29 – Photo analysée par le modèle sur Unity

3.5.2 Transferts de l'image en fichier .txt

Après avoir analysé l'image, il faut créer le même fichier texte décrivant les différents éléments de la scène avec le nom des plantes associés. Pour cela, comme montre la Figure 4a, j'ai fais corréler la position des formes et des couleurs reconnues sur l'image en un fichier .txt correspondant.

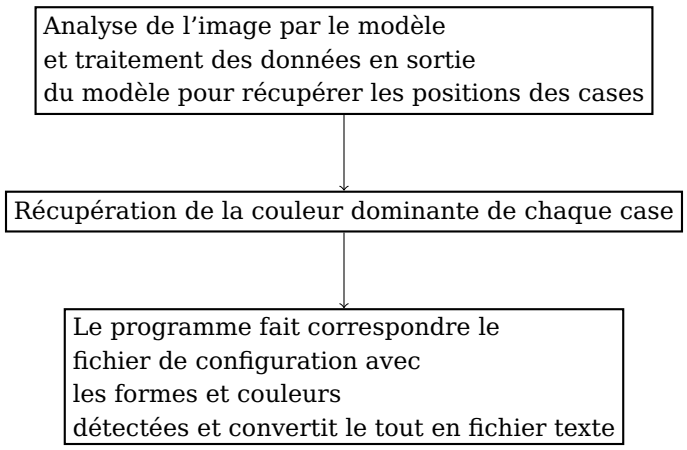


Figure 30 – Fonctionnement de l'analyse d'une image jusqu'à la création de la parcelle

3.5.3 Viusalisation de la scène

Pour projeter la scène en réalité augmentée, j'ai changé la phase de sélection de la scène (auparavant, il suffisait de choisir son fichier texte pour démarrer la scène en réalité augmentée). Maintenant, l'utilisateur est redirigé sur un menu où toutes les scènes disponibles sont affichées, avec comme nom, le moment où l'utilisateur a analysé l'image. Une petite visualisation de ce à quoi la scène va ressembler y est présente à côté (Figure 31 et la scène correspondante en réalité augmenté est visible à la Figure 32.

Figure 31 – Menu de présentation de scène disponible

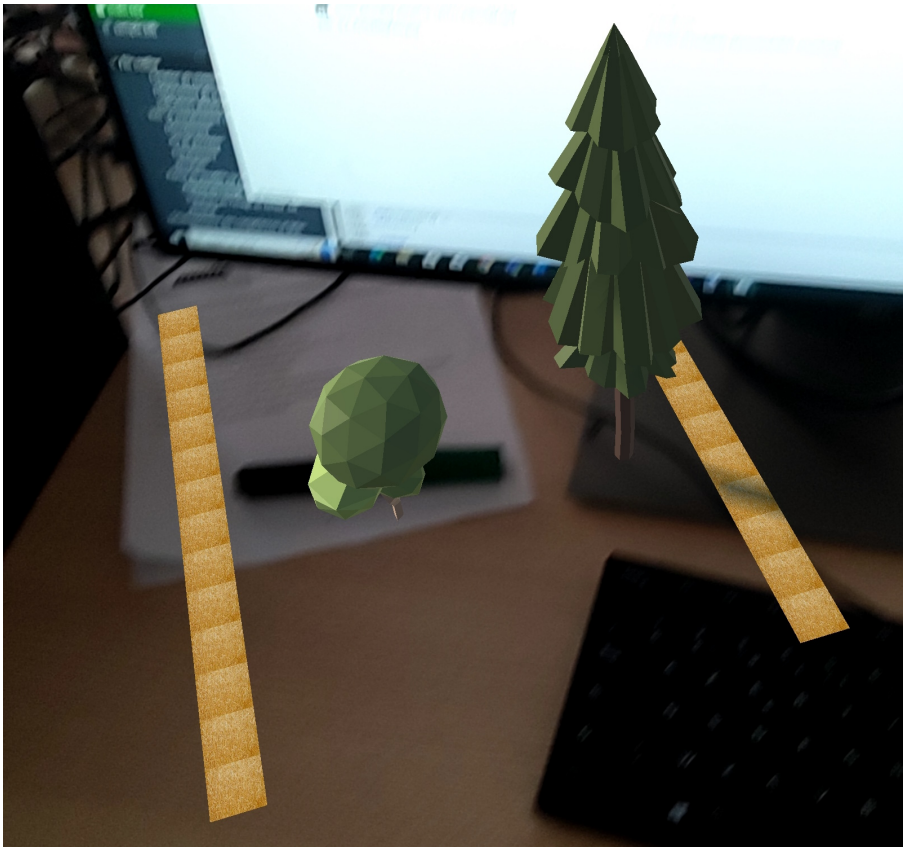


Figure 32 – Résultat de l'image analysée

3.6 Vitrine numérique

Il m'a été demandé de créer un projet en réalité augmentée de A à Z dont le cahier des charges est : Voici le cahier des charges :

— Création d'une interface pour sélectionner 1 système agroforestier parmi 3 (croquis visualisable à la Figure 33)
— Mettre un "slider" sur trois périodes de croissances (jeune, moyenne et avancée) des arbres pour simuler l'évolution temporelle dans l'écran avec la réalité augmentée
— Créer une ombre dynamique avec le package de gestion de l'ombrage (1 journée = 30 secondes)
— Projection de la scène à partir des coordonnées GPS de l'utilisateur
— Intégrer un slider pour les 4 saisons (été, automne, hiver, printemps) dans la visualisation en réalité augmentée
— Récupération de la position GPS de l'utilisateur

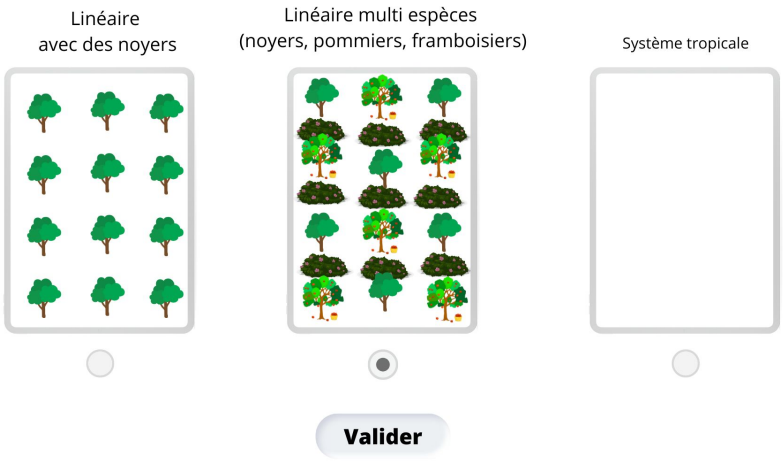


Figure 33 – Croquis du Menu

3.6.1 Création du menu de l'interface

J'ai donc commencé par la création du menu (Figure 34). Comme on peut le voir sur la Figure 34 c'est le système linéaire multi-espèces qui est sélectionné. Les cases rouge, bleue et verte devront être changées dans le futur, je n'ai actuellement pas les images à incruster.

Figure 34 – Menu principal de l'application

3.6.2 Récupération coordonnées GPS & création de la parcelle

J'ai enchaîné sur la récupération des coordonnées GPS de l'utilisateur.

Le fonctionnement se déroulera comme montré dans le schéma de la Figure 35

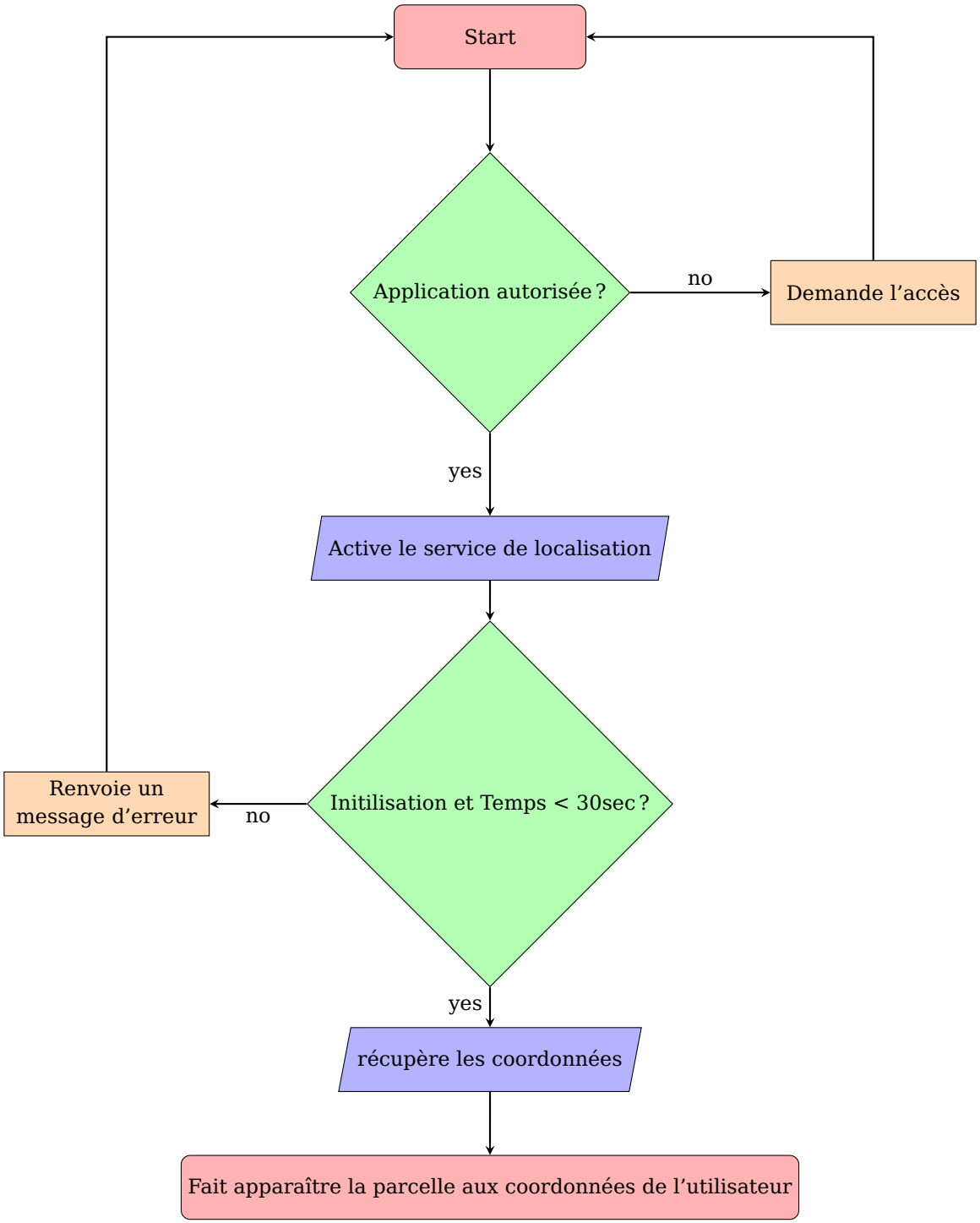


Figure 35 – Flowchart du fonctionnement de la fonction

L'application vérifie si l'utilisateur lui a bien autorisé l'accès à la géolocalisation. Si c'est le cas alors le service de géolocalisation est lancé. L'application attend au maximum 20 secondes pour que le service soit initialisé. Pour finir, elle récupère les coordonnées de l'utilisateur.

La fonction en C# :

```
IEnumerator GetPosition()
{
    if(!Input.location.isEnabledByUser)
    {
        print("location not enabled => permission request");
        Permission.RequestUserPermission(Permission.FineLocation);
    }

    if(!Input.location.isEnabledByUser)
        yield break;

    Input.location.Start(); //Demarre le service de geolocalisation integre a Unity

    // Waits until the location service initializes
    int maxWait = 20;
    while (Input.location.status == LocationServiceStatus.Initializing && maxWait > 0
        )
    {
        yield return new WaitForSeconds(1);
        maxWait--;
    }

    // If the service didn't initialize in 20 seconds this cancels location service
        use.
    if (maxWait < 1)
    {
        print("Timed out");
        yield break;
    }

    // If the connection failed this cancels location service use.
    if (Input.location.status == LocationServiceStatus.Failed)
    {
        print("Unable to determine device location");
        yield break;
    }

    _pLat = Input.location.lastData.latitude;
    _pLong = Input.location.lastData.longitude;
    _pAlt = Input.location.lastData.altitude;

    print("Lat : " + _pLat + " Long : " + _pLong + "Alt : " + _pAlt);

    StartCoroutine(InstantiateParcelle()); // cree la parcelle aux coordonnees de l'
        utilisateur
}
```

Après avoir récupéré les coordonnées, la scène se crée en conséquent. C'est une librairie nommée AR + GPS qui nous permet de faire apparaître des objets en réalité augmentée sur des coordonnées GPS.

```
1    ARAnchor anchor = anchorManager.AttachAnchor(plane, new Pose(plane.transform.position
         , plane.transform.rotation));
2    var newParcelle = Instantiate(_parcelles[systemChoose], anchor.transform);
3    anchorManager.anchorPrefab = newParcelle;
4
5    PlaceAtLocation.AddPlaceAtComponent(newParcelle, loc, opts); //Place la parcelle sur
         les coordonnees
```

La ligne 1 et 3 font en sorte que l'objet ait un meilleur ancrage dans la scène en réalité augmentée. Même si l'utilisateur bouge la camera, la parcelle restera à sa position initiale.

3.6.3 Slider croissance & saisons

J'ai eu besoin de sliders pour permettre à l'utilisateur de changer dynamiquement la croissance et les saisons de la parcelle. Les sliders sont un moyen simple et efficace et permettent aux utilisateurs d'ajuster et de sélectionner des valeurs de manière visuelle et interactive.

Qu'est-ce qu'un slider ?

Un slider peut être comparé à un potentiomètre, il peut avoir un nombre discrètes ou continues.

Figure 36 – Exemple Slider

Récupération des modèles 3D

J'ai téléchargé plusieurs modèles 3D d'arbres à partir de asset store d'Unity et je les ai importés dans le projet.

Le problème étant qu'il est impossible de changer le modèle 3D d'un arbre déjà existant sur la scène, il faut donc le supprimer et le remplacer par un nouveau. Le souci est qu'il faut garder la position de ce même arbre et connaître son modèle qui correspondant pour la combinaison saison et croissance qui lui est propre.

J'ai opté pour la solution de créer une liste de 3 dimensions (voir 37).

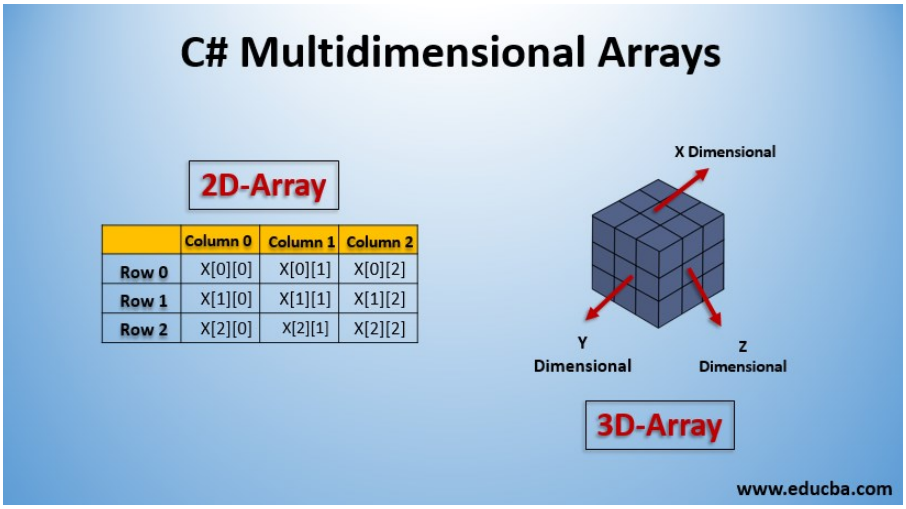


Figure 37 – Comparaison 2D & 3D

J’ai donc rangé mes modèles 3D comme sur le tableau 38

Liste de plantes												
Type de plante	Printemps			Été			Automne			Hiver		
	Jeune	Moyen	Avancé	Jeune	Moyen	Avancé	Jeune	Moyen	Avancé	Jeune	Moyen	Avancé
Plante 1	Modèle 1	Modèle 2	Modèle 3	Modèle 4	Modèle 5	Modèle 6	Modèle 7	Modèle 8	Modèle 9	Modèle 10	Modèle 11	Modèle 12
Plante 2	Modèle 13	Modèle 14	Modèle 15	Modèle 16	Modèle 17	Modèle 18	Modèle 19	Modèle 20	Modèle 21	Modèle 22	Modèle 23	Modèle 24

Figure 38 – Tableau représentant les dimensions de la liste

La première dimension représente le type de plante, la seconde sa saison et la troisième son état de croissance.

J’ai ensuite créé une classe pour faciliter la suite.

```
public class Plant
{
    public GameObject go { get; set; }

    private Position modelsPos;

    public Plant(int x, int y, int z, GameObject instantiateGo)
    {
        modelsPos = new Position { x = x, y = y, z = z};
        go = instantiateGo;
    }

    public class Position
    {
        public int x { get; set; }
        public int y { get; set; }
        public int z { get; set; }
    }

    public void Grow(int value)
    {
        modelsPos.z = value;
        ActualiseModel();
    }

    public void ChangeSeason(int value)
    {
        modelsPos.y = value;
        ActualiseModel();
    }

    private void ActualiseModel()
    {
        var oldPlant = go;

        var newGo = Instantiate(models[modelsPos.x, modelsPos.y, modelsPos.z],
            oldPlant.transform.parent, true);
        newGo.transform.position = oldPlant.transform.position;
        go = newGo;
        Destroy(oldPlant);
    }
}
```

Pour créer une nouvelle plante, il suffit de passer dans la classe, la position du modèle de la plante dans le tableau et l'objet plante.

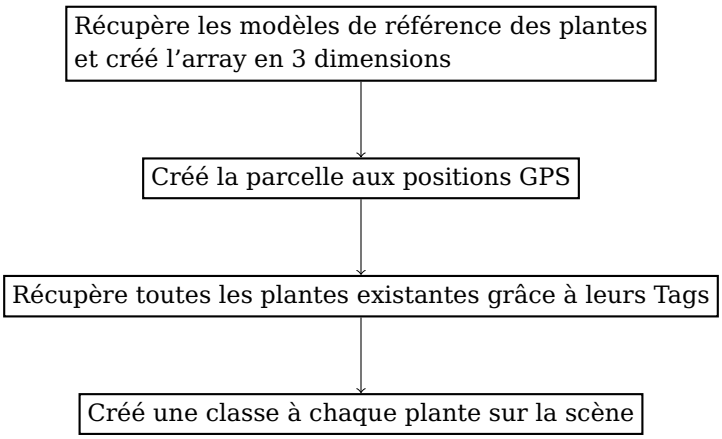


Figure 39 – Étapes du fonctionnement de l'application

Je rajoute les sliders. Comme vous pouvez le voir sur la Figure 40 le slider Saison comporte 4 pas pour chaque saison (ici été, automne et hiver). Le slider pour la croissance comporte 4 pas et a également des pastilles attribuées au niveau de croissance. Chaque des sliders a une fonction attribuée qui renvoie la valeur du slider. Elle se déclenche au moment du changement de la position de celui-ci par l'utilisateur. Elle provoque une mise à jour des modèles 3D de la parcelle.

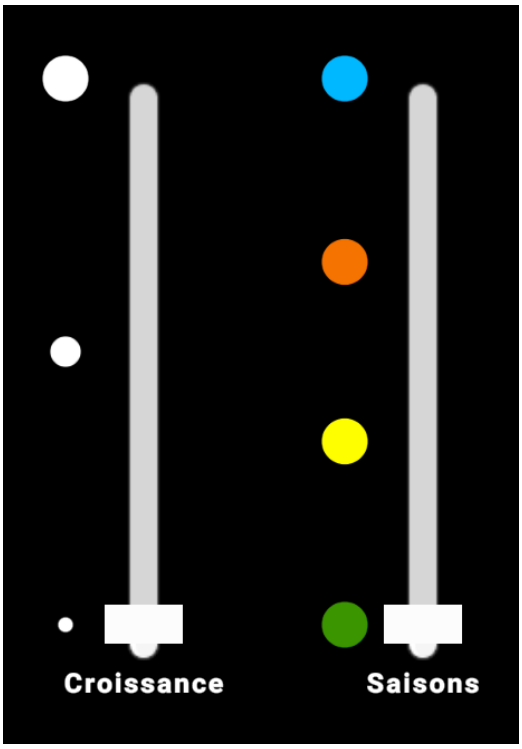


Figure 40 – Sliders croissance & saisons

```
    //Pour La croissance
    sliderGrow.onValueChanged.AddListener((value) => {
        foreach (var plant in _plants)
        {
            plant.Grow((int)value);
        }});

    //Pour les saisons
    sliderSeason.onValueChanged.AddListener((value) => {
        foreach (var plant in _plants)
        {
            plant.ChangeSeason((int)value);
        }});
```

3.6.4 Rajout des ombres

J'ai effectué le rajout des ombres grâce à un package nommé "Hessburg - Sunlight" (documentation) qui permet de simuler l'éclairage d'une journée à un endroit sur terre grâce à plusieurs paramètres qui nous permettent de changer la vitesse du soleil, sa couleur, sa position, etc... J'ai configuré le "soleil" de façon à ce qu'il prenne 30 secondes pour faire un cycle. J'ai créé une texture qui est transparente mais où les ombres peuvent se projeter pour augmenter le réalisme, comme visible sur la 41.

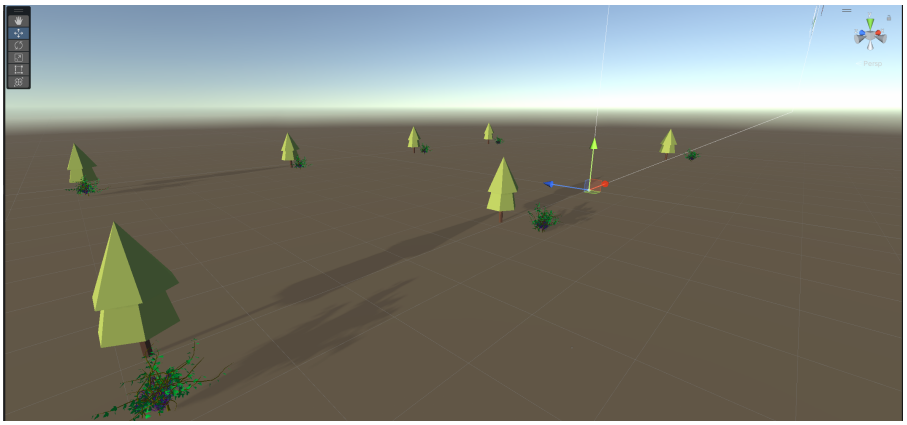
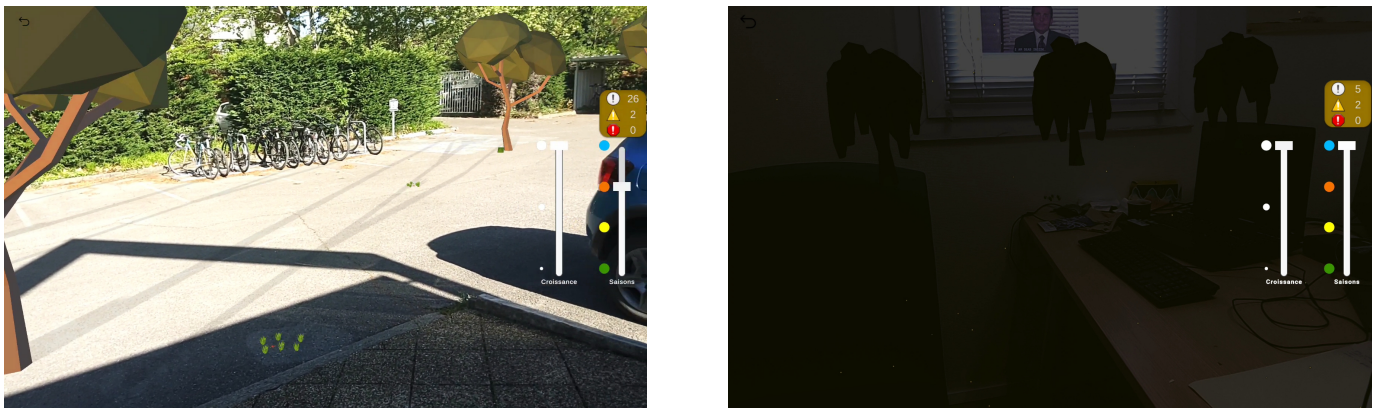


Figure 41 – Première vue de la parcelle sur Unity

Pour augmenter encore le réalisme, j'ai rajouté du post-processing sur le rendu.

Le processus consiste à appliquer des filtres et des effets plein écran à la mémoire tampon de l'image d'une caméra, avant qu'elle ne soit affichée à l'écran.

Quand il fera nuit sur la scène, j'ai fait en sorte d'augmenter progressivement les contrastes, l'exposition et la transparence des ombres pour moins les voir. Ce qui donne comme résultat final les deux images de la Figure 42. Comme on peut le voir sur la Figure 42a le soleil fait projeter les ombres sur le sol alors que sur la Figure 42b la vision est plus difficile et les ombres ne sont pas projetées.



(a) Rendu de jour

(b) Rendu de nuit

Figure 42 – Rendu du projet à deux moments différents

3.7 Démo tracteur

Dans cette partie, je vais vous présenter la visualisation que j'ai créé pour Clément Dahan. En effet, après mon stage, Clément et Laetitia LEMIERE vont organiser une journée pour demander l'avis d'agriculteurs et de conseillers, sur certaines fonctionnalités qui seront rajoutées au projet AR Indoor.

J'ai donc dû créer une démonstration d'une fonctionnalité qui doit permettre de visualiser un tracteur dans une parcelle et montrer lorsque ce dernier n'a pas la place pour manoeuvrer. Le cahier des charges regroupe donc :

- Un rendu en réalité augmentée
- Un tracteur suivant un chemin prédéfini sur une parcelle
- Si ce tracteur est trop proche d'un arbre alors mettre l'arbre en rouge pour illustrer sa collision (cela entraîne la baisse de la valeur marchande de l'arbre), sinon, laisser sa texture initiale
- L'endroit où le tracteur ne passe pas, le sol est en rouge
- Un bouton Play & Stop pour démarrer ou stopper les mouvements du tracteur
- L'utilisateur peut choisir entre 3 tailles de tracteurs
- L'utilisateur peut choisir la position du tracteur le tracé prédéfini

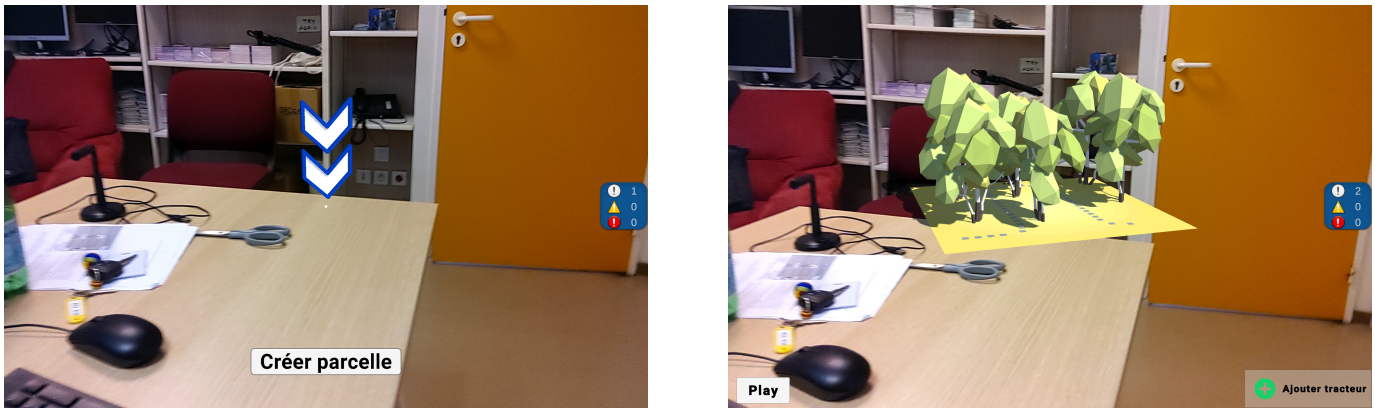
3.7.1 Système de création de la parcelle

Je me suis d'abord intéressé à comment j'allais faire apparaître la parcelle en réalité augmentée.

Les options que j'avais imaginées :

- La faire apparaître directement sur l'utilisateur (ne permets pas à l'utilisateur de choisir l'endroit souhaité)
- Faire un curseur qui représente la position où la parcelle va apparaître à l'endroit où l'utilisateur vise avec sa caméra
- Faire apparaître la parcelle sur une surface plane dès que celle-ci est reconnue

J'ai choisi l'option d'avoir un petit curseur au-dessus des zones planes, qui indiquera la position de la future parcelle. J'ai choisi cette option car c'était, pour moi, un défi que je voulais relever. Je pense aussi que c'est le moyen le plus simple pour l'utilisateur de visualiser où sa parcelle apparaîtra. Un petit bouton "Créer parcelle" apparaîtra à l'endroit où l'utilisateur vise avec sa caméra s'il est propice à l'apparition de la parcelle.



(a) Curseur d'apparition

(b) Parcelle

Figure 43 – Rendu du système d'apparition

Le schéma de la Figure 44 indique comment le programme gère ce système.

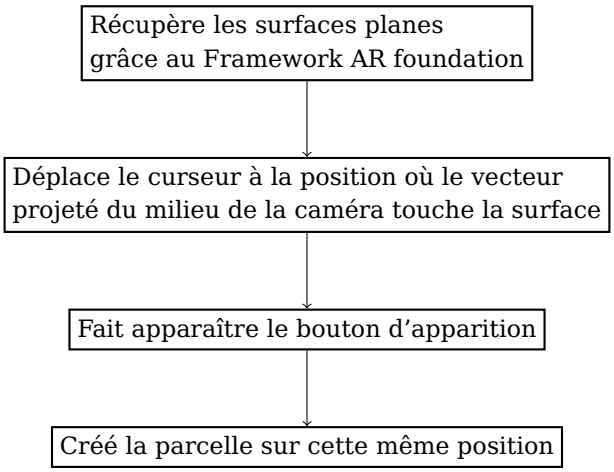


Figure 44 – Fonctionnement du curseur

3.7.2 Création du chemin & mouvements tracteur

Sur la Figure 45, j'ai commencé par créer la forme de la parcelle et positionner les arbres dans la parcelle et délimitant le futur chemin.

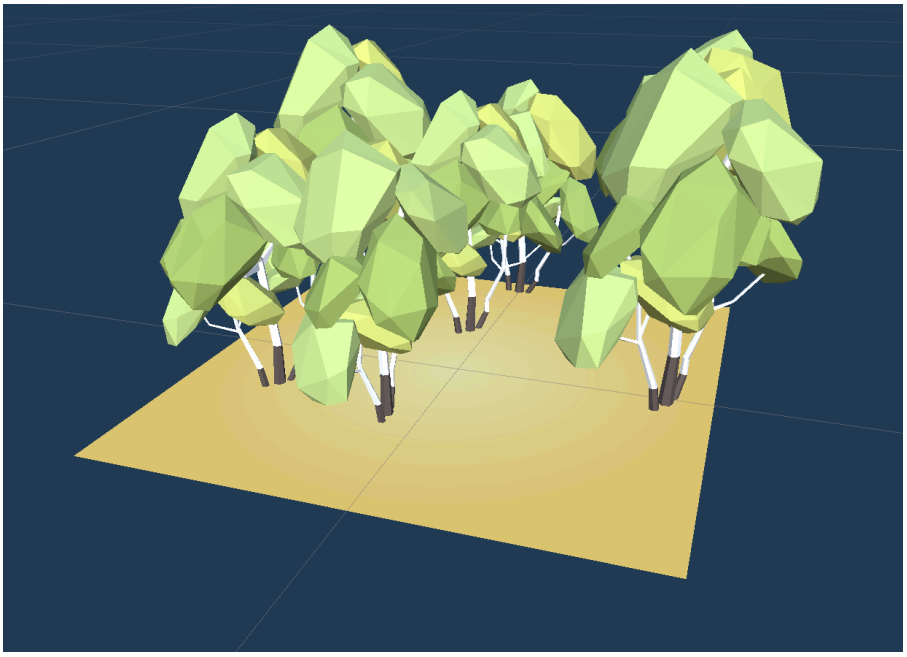


Figure 45 – Base parcelle

Comme on peut le voir sur la Figure 46, j'ai créé des objets "vides" qui me serviront de positions repères à chaque virages. Ils peuvent être assimilés à des checkpoint, ils sont utilisés pour amener le sujet d'une position à une autre en passant par tous les checkpoint.

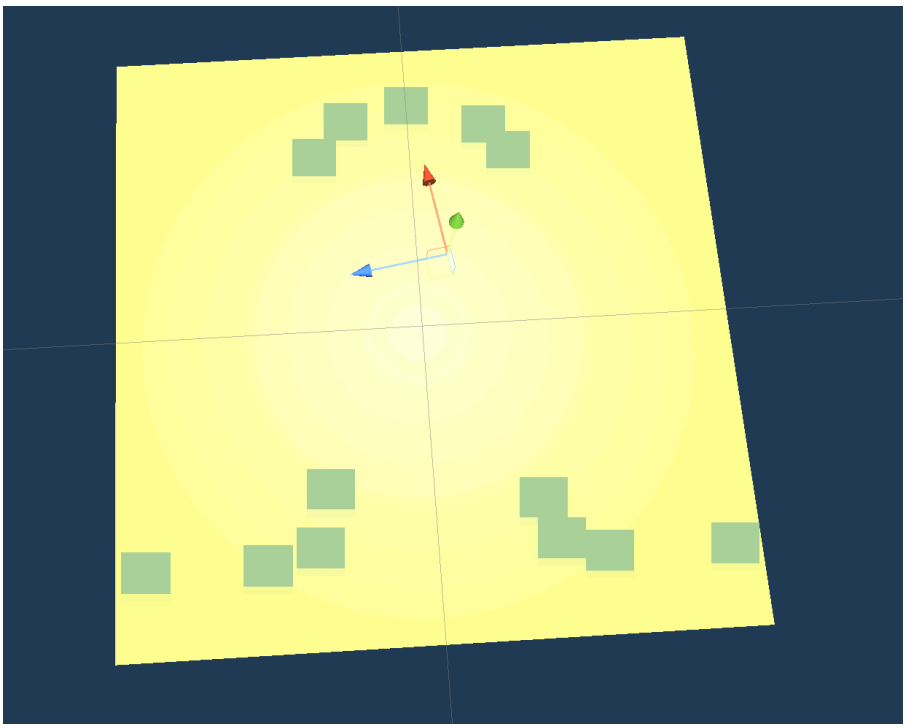


Figure 46 – [carrés vert] Objets qui marquent les virages

Pour que le tracteur se dirige d'une position à une autre, j'ai d'abord mis dans une liste toutes les positions des checkpoints dans l'ordre que le tracteur devra les suivre. J'ai ensuite programmé le tracteur pour qu'il suive le chemin si le mode "Play" est activé.

```
    while (_tractorMove)
    {
        if (_tractorTargetAngles >= _anglesTractor.Count - 1) //remet le tracteur au debut
        {
            _tractorTargetAngles = 0;
            _newtractor.transform.position = _anglesTractor[_tractorTargetAngles].transform.
                position;

            if (!TractorInitialisation)
                _tractorMove = false;
        }

        var position = _newtractor.transform.position;
        position = Vector3.MoveTowards(position, _anglesTractor[_tractorTargetAngles+1].transform.
            position, Time.deltaTime * speed); //donne la prochaine position du tracteur
        _newtractor.transform.position = position;

        _newtractor.transform.rotation = Quaternion.Lerp(_newtractor.transform.rotation,
            _anglesTractor[_tractorTargetAngles + 1].transform.rotation, Time.deltaTime * 7); //
            donne la bonne rotation au tracteur

        float distance = Vector3.Distance(position, _anglesTractor[_tractorTargetAngles+1].
            transform.position); //distance entre le tracteur et sa prochaine position

        if (distance < 0.001f) // passe au point suivant si la distance < 0.001
            _tractorTargetAngles++;
        yield return null;
    }
```

3.7.3 Déplacement du tracteur manuellement

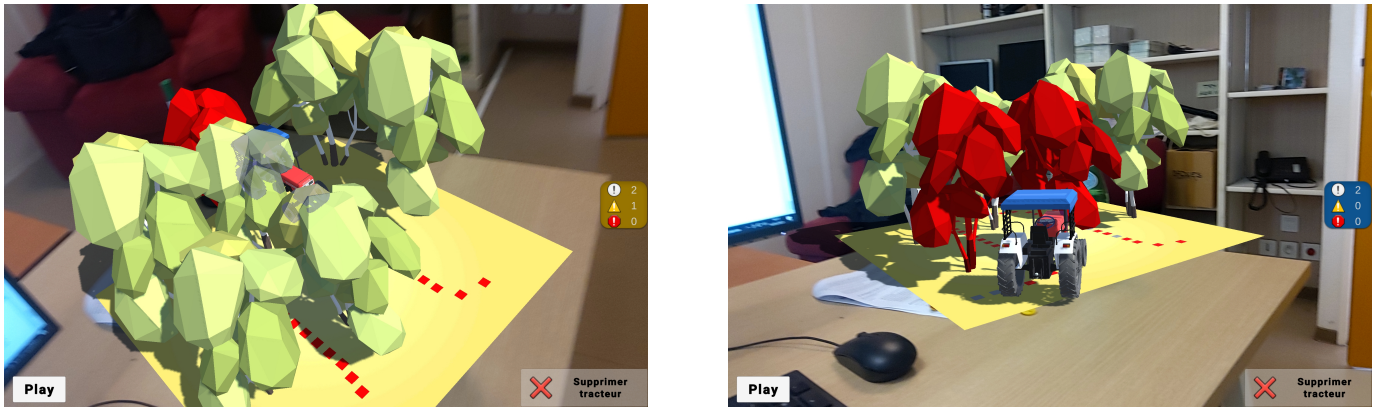
J'ai mis en place un mécanisme permettant de déplacer le tracteur vers l'endroit où l'utilisateur clique sur la parcelle. Lorsque l'utilisateur appuie sur l'écran, un vecteur qui démarre à partir de la position relative à l'écran, se projette vers la parcelle. Le tracteur se téléportera sur l'objet "vide" le plus proche, entre la position de collision du vecteur sur la parcelle et l'objet "vide".

```
    GameObject MoveTractor(Vector3 actualPosition)
    {
        float distance = Single.PositiveInfinity;
        GameObject nearestPosition = null;

        foreach (var anchor in _anglesTractor)
        {
            var newDistance = Vector3.Distance(anchor.transform.position, actualPosition)
                ;

            if (newDistance < distance)
            {
                distance = newDistance;
                nearestPosition = anchor;
                _tractorTargetAngles = _anglesTractor.IndexOf(anchor);
            }
        }

        return nearestPosition;
    }
```

La fonction ci-dessus prend comme paramètre la position de la collision sur la parcelle et renvoie l'objet "vide" le plus proche.

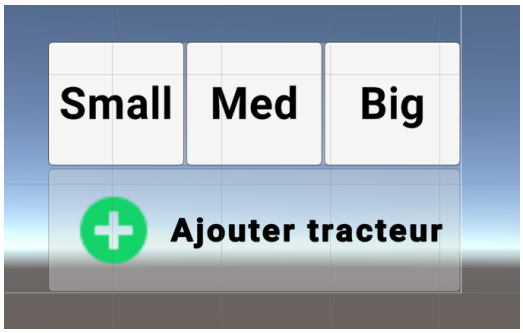
3.7.4 Indicateurs distance arbre - tracteur

Une fois que l'utilisateur a sélectionné la taille du tracteur grâce au menu de la Figure 48, l'objectif est d'indiquer à l'utilisateur si le tracteur peut passer ou non à un endroit spécifique. Pour ce faire, la zone au sol est représentée par des pointillés qui deviennent rouges si le tracteur ne peut pas passer, ou gris s'il peut passer. Quand l'utilisateur lance le mouvement du tracteur, au même endroit où le sol est rouge, l'arbre ou les arbres trop proches deviennent rouges temporairement. On peut l'observer sur la Figure 47.



(a) (b)

Figure 47 – Rendu des pointillés & et des arbres rouges



Voici le menu pour choisir entre les trois différentes tailles de tracteurs, sur la Figure 47 la taille **Med** a été choisie.

Figure 48 – Menu de choix taille tracteurs

4 Développement des compétences et bilan

4.1 Compétences GEII & nouvelles compétences

Les projets organisés durant les deux précédentes années m'ont permis d'avoir une approche optimisée et en adéquation avec les problèmes rencontrés. L'organisation et le travail en équipe étant au centre, ces projets m'ont permis de m'améliorer devantage sur ces points.

Les compétences apprises en mathématiques m'ont permis de mieux appréhender l'analyse des résultats du réseau de neurones, ainsi que les notions vectorielles. De plus j'ai suivi une formation obligatoire d'une journée en statistique au sein de l'UMR AMAP.

Sur mes compétences technique, même si j'avais déjà de bonnes bases dans le développement, j'ai amplifié mes compétences dans le domaine de l'intelligence artificielle et du développement, dans un environnement en réalité augmentée. En utilisant Python et Unity.

Une nouvelle tâche a été la création de la documentation, que ce soit côté utilisateur ou développeur, ce que je n'avais jamais fait auparavant. J'ai donc amélioré mes compétences de communication.

4.1.1 Formation : Module Image

Durant ce stage, j'ai assisté à un module au LIRMM nommé "Image". Comme son nom l'indique, il portait sur l'imagerie dans le numérique. Les sujets allaient de l'entreprise Golaem qui gère les effets de foule numériquement, jusqu'à la conduite automatisée. Même si la formation était plutôt destinée à des doctorants, cela ne m'a pas empêché d'en apprendre plus sur ce domaine. Les sujets étaient les suivants :

- Compression et évaluation de la qualité pour les environnements immersifs
- Les réseaux de neurones pour l'observation de la terre
- Asservissement visuel en robotique
- La recherche au service des effets spéciaux - la simulation de foules numériques
- Self-supervision on Wheels : Self-supervised representation learning from autonomous driving data
- Apprentissage et Mémorisation des Modèles Génératifs d'Images

4.1.2 Compétences transversales

De plus, avoir travaillé avec Clément Dahan m'a grandement enrichi dans mon expérience professionnelle. Nous avons oeuvré efficacement en équipe, partagé nos connaissances et résolu des problèmes de manière collaborative.

4.1.3 Parcours

Ce stage a beaucoup enrichi mes compétences et m'a permis de mettre un pied dans le monde du travail. Il a également renforcé ma décision de vouloir continuer dans le monde de l'informatique.

5 Conclusion

Mon travail aura permis l'aboutissement d'un prototype qui sera utilisé en démonstration durant des ateliers de conception. Il sera, peut-être, également utilisé comme base de travail si un autre sujet de thèse portant sur ce domaine.

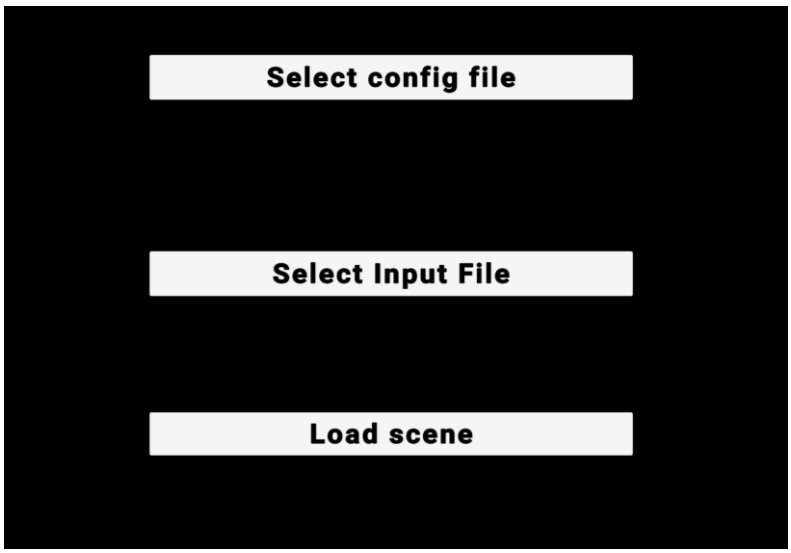
J'ai beaucoup aimé l'autonomie et la confiance qui m'ont été accordées ainsi que la liberté dans l'approche de mes missions. Je voyais le monde du travail bien plus hostile et compétitif, je me doute bien que mon expérience n'est pas une généralité mais je ne pensais pas une seule seconde que j'allais passer un aussi bon stage.

6 Annexes

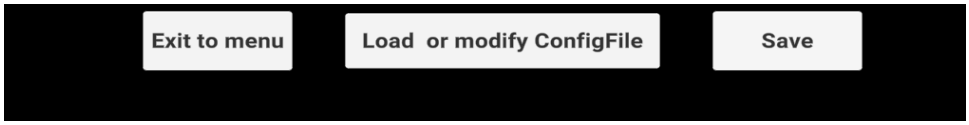
Lien GitHub des projets : https ://github.com/agroforestar

DOCUMENTATION AR INDOOR

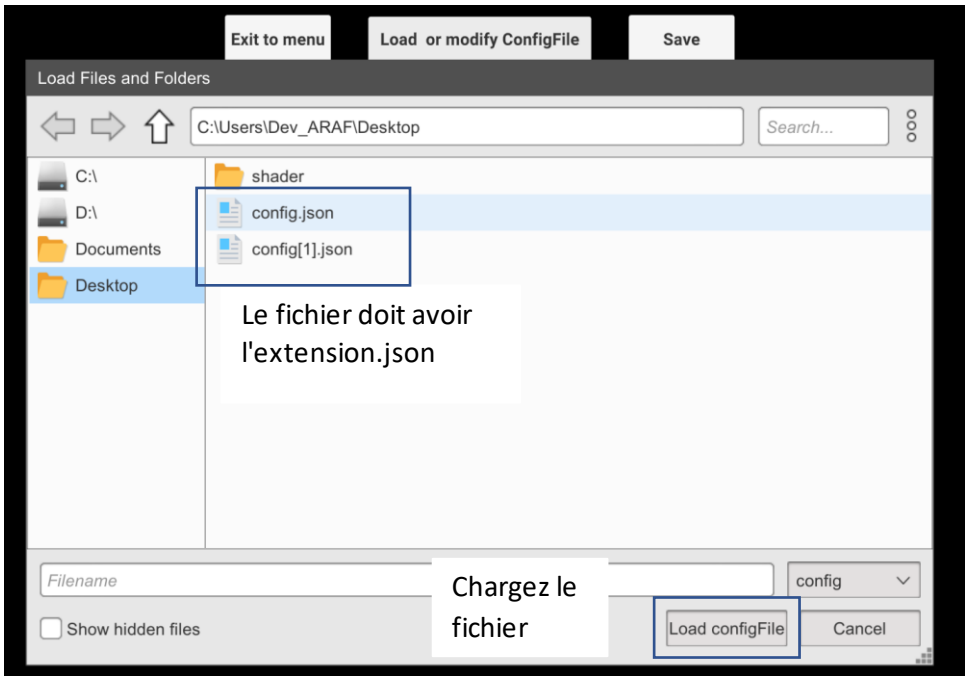
INITIALISATION ET CONFIGURATION



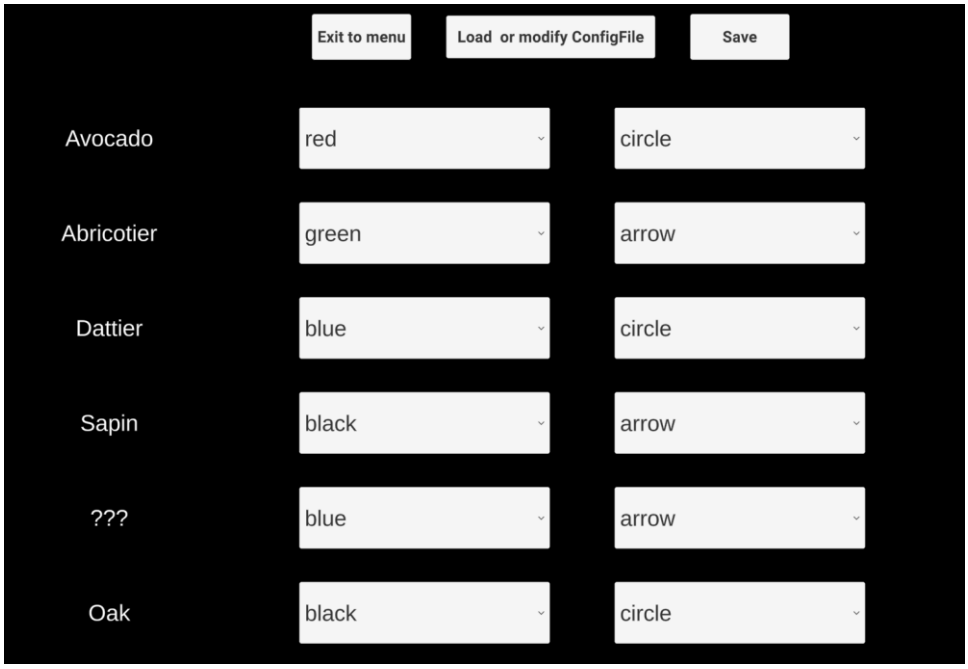
Vous avez trois menus à disposition. Si vous venez d'ouvrir l'application pour la première fois vous devrez impérativement vous rendre dans l'option "***Select config file***".



Chargez un fichier de configuration que vous avez au préalable déjà téléchargé grâce au bouton "***Load*** o***r modify ConfigFile***".



Après avoir chargé le fichier souhaité vous verrez une fenêtre comportant les informations du fichier.



Si les paramètres vous conviennent vous pouvez quitter sinon changez les à votre guise.

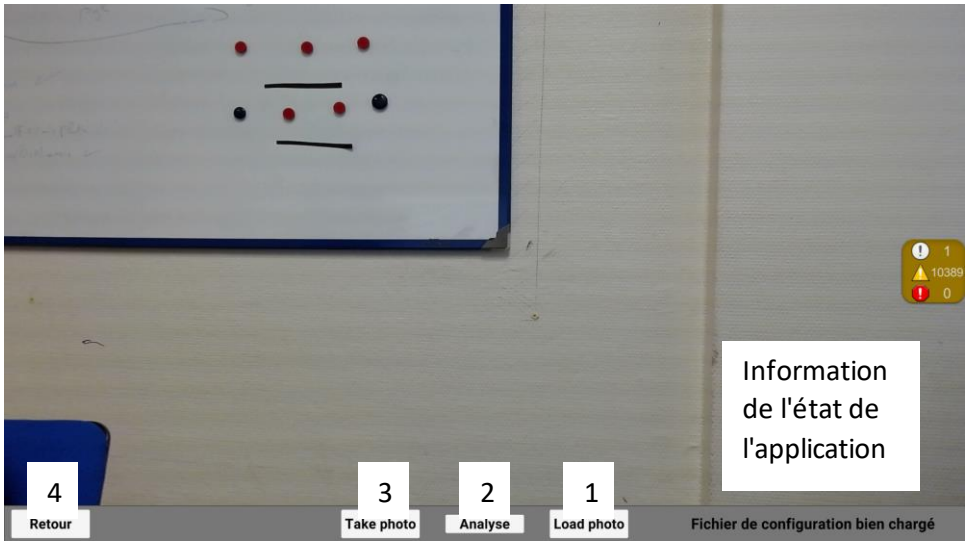
ATTENTION

Vous ne pouvez pas attribuer à deux plantes différentes une même couleur et une même forme.

Après vos modifications sauvegardez les modifications avec le bouton "***Save***", puis quittez avec le bouton "***Exit to menu***".

TRAITEMENT DE L'IMAGE

A partir du menu principal, dirigez-vous vers "***Select input file***".

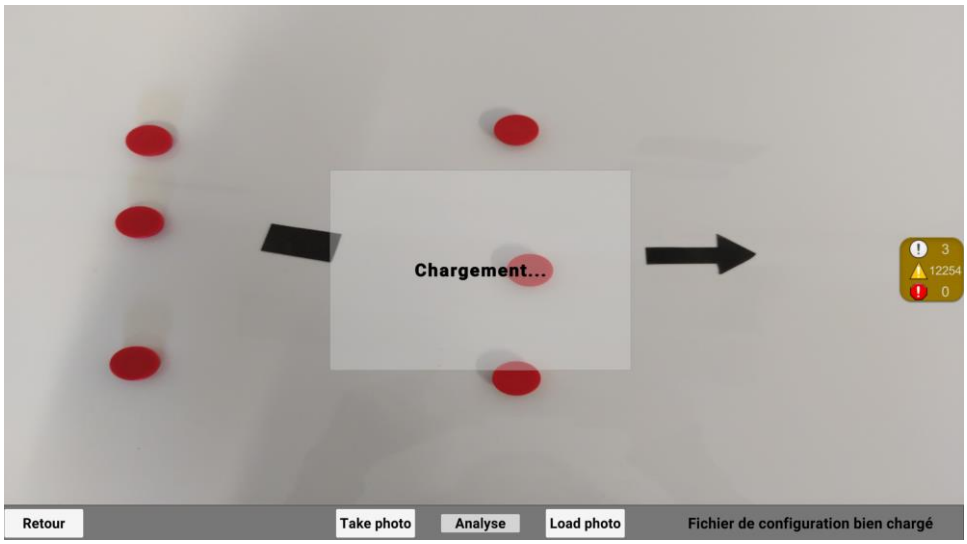


Si l'application à l'autorisation d'utiliser votre appareil photo vous devriez y apercevoir ses données.

Signification boutons :

1 – Vous chargez une photo en .jpg à partir de votre stockage.
2 – Vous analysez la photo que vous avez chargé ou prise avec l'application.
3 – Vous prenez en photo ce que votre caméra filme.
4 – Vous redirige sur le menu.

L'analyse dure une dizaine de seconde, durant ce temps aucunes interaction ne sera possible avec l'application.



Durant l'analyse une boite d'information apparaitra tant que l'application traite votre image.

Après ce chargement il vous sera indiqué en bas à droite,

l'état de l'application devra être comme ci-dessous.

Vous pourrez revenir au menu principal.

LANCEMENT DE LA PARCELLE

Après avoir analysé votre image vous devrez cliquer sur le bouton "***Load scene***".

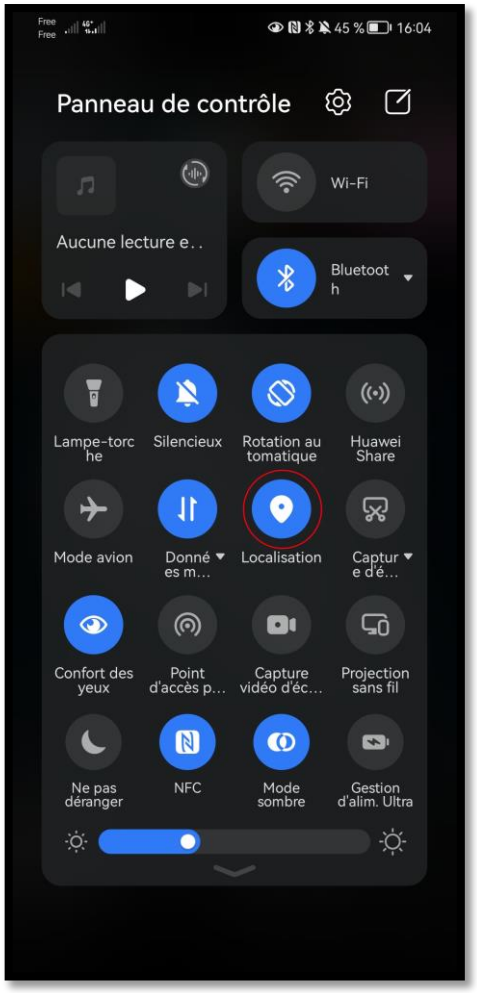
Vous pourrez, en cliquant sur le bouton où se situe la date d'analyse de votre image, lancer votre parcelle en réalité augmenté.

Vous pourrez également supprimer le fichier grâce à la **croix rouge** ou retourner au menu avec le bouton "***Quit***".

VITRINE NUMERIQUE DOCUMENTATION

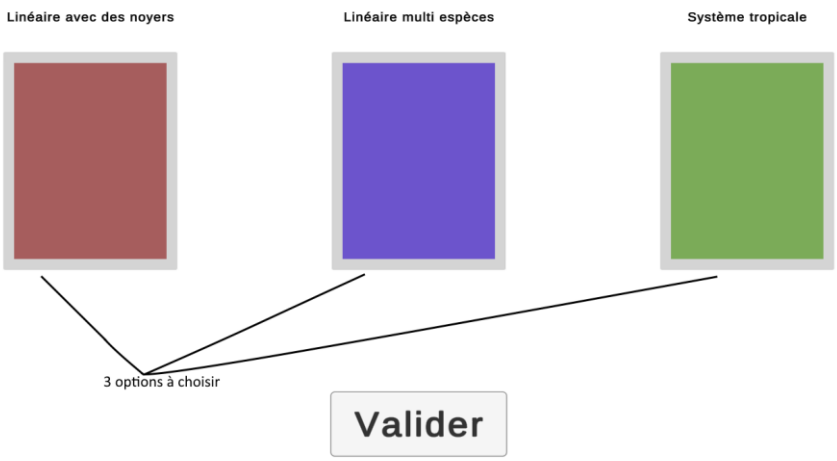
PRE-REQUIS

Avant de démarrer l'application n'oubliez pas d'activer la localisation de son smartphone.



- Si vous êtes connecté à un réseau (wifi ou données mobiles) la précision du GPS ne sera que meilleure.

MENU



Vous aurez le choix entre trois types de parcelles différentes :

- Une parcelle seulement constituée d'arbres.
- Une parcelle constituée d'arbres et de buissons.
- Une parcelle constituée de végétation tropicale.

Ici c'est la système linéaire multi-espèce qui est sélectionné.

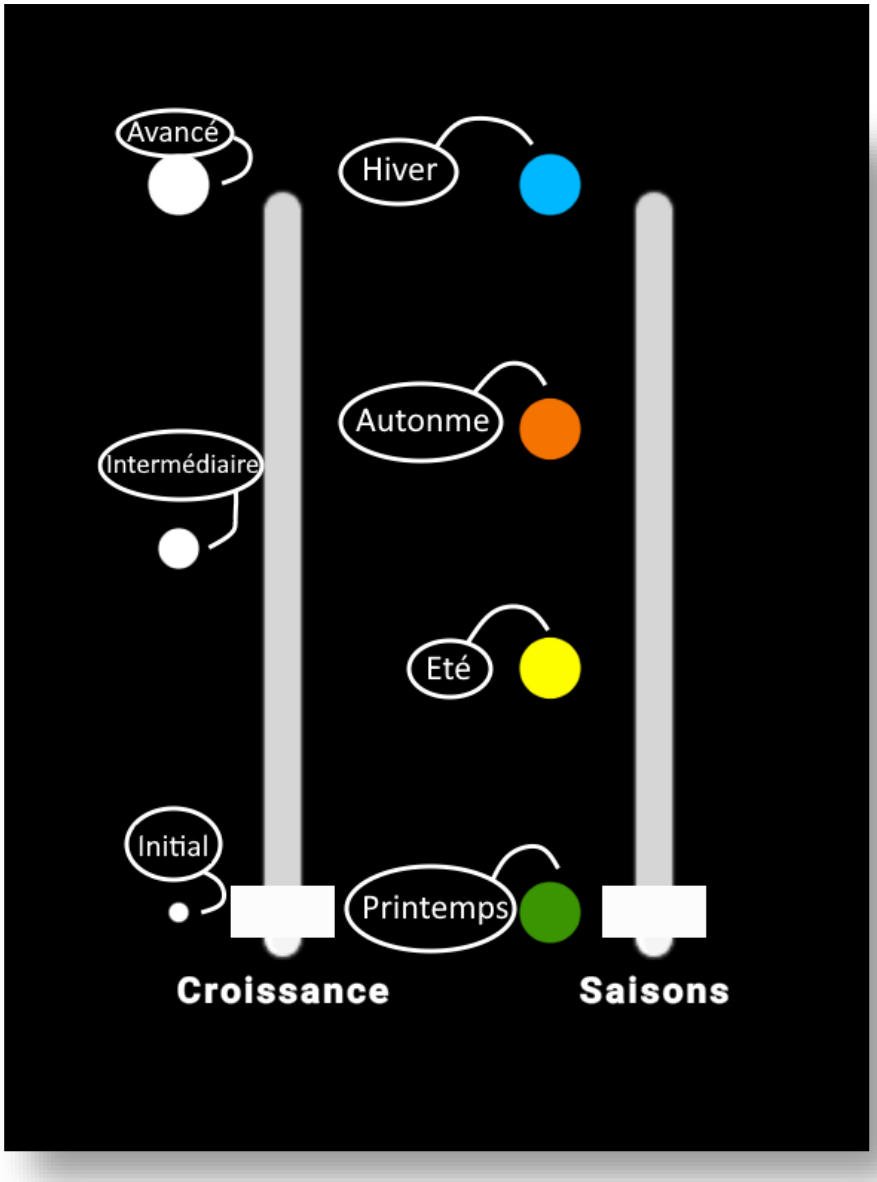
Après votre choix effectué le bouton "Valider", vous redirigera vers la scène principale.

DANS LA SCENE EN REALITE AUGMENTEE

Faire pivoter l'appareil verticalement au sol en forme de ∞ en face d'une surface plane jusqu'a que le texte "Veuillez avoir une surface plane en face de vous" disparaisse, cela signifie que l'application est bien calibrée.

Texte qui vous informe de l'état de calibration de l'application.

Vous pouvez changer l'apparences des plantes sur votre scène en changeant la valeur des curseurs.



- Saisons : Change l'apparence des plantes en fonction de la saison choisie par l'utilisateur.
- Croissance : Associe l'étape de croissance choisie par l'utilisateur aux modèle des plantes sur la scène.

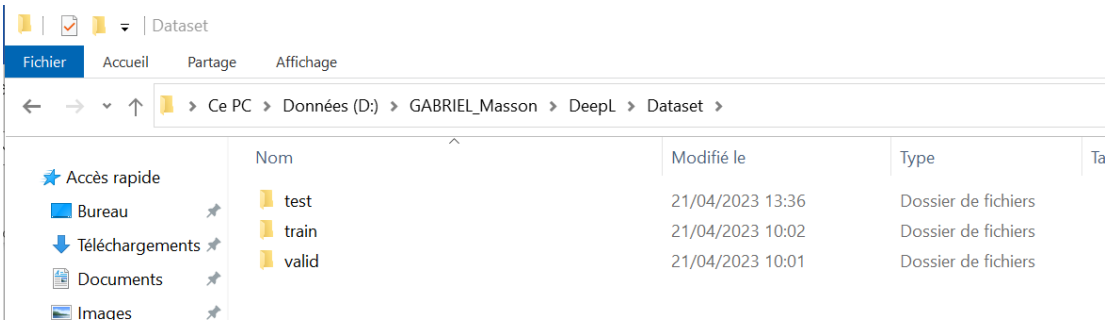
En haut à gauche, un bouton en forme de flèche vous permet de revenir au menu.

DOCUMENTATION YOLOTRAINING

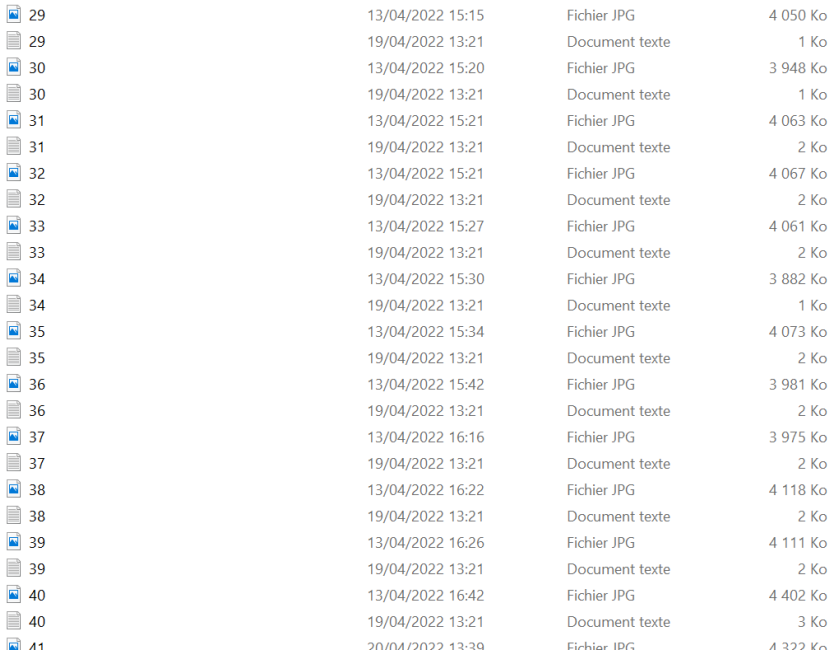
PRE-REQUIS

REPARTITION DE VOTRE DATASET

Avant tout, créez un dossier et 3 sous dossiers comme ci-dessous.



Dans chaque dossier placez-y les images souhaitées et leurs correspondances .txt.



Aucunes images ne devront apparaitre dans deux dossiers différents.

CREATION PROJET PY

Créez un nouveau projet dans votre environnement préféré et importez-y ultralytics.

De plus si vous avez un bon GPU vous pouvez installer CUDA & CuDNN puis installer Py-torch GPU pour pouvoir profiter d'une puissance de calcul optimale.

CREATION FICHIER YAML

Créez, dans le même dossier que votre main.py un nouveau fichier .yaml, renommez-le comme vous voulez.

Il devra être composé avec les éléments ci-dessous :

```yaml
path: D:\GABRIEL_Masson\DeepL\Dataset
train: D:\GABRIEL_Masson\DeepL\Dataset\train
test: D:\GABRIEL_Masson\DeepL\Dataset\test
val: D:\GABRIEL_Masson\DeepL\Dataset\valid

nc: 3

names: ["circle", "rectangle", "arrow"]
```

N'oubliez pas que l'ordre des noms de class dans **"name"** comptent.

LANCEMENT ENTRAINEMENT

Le code ci-dessous devra se trouver dans votre main.

N'oubliez pas de remplacer les chemins d'accès par les votre.

Seul "**model2 = YOLO("yolov8m.pt")**" devra rester comme il est, Ultralytics vous créera automatiquement votre model initial.

```
import torch.cuda
from ultralytics.yolo.engine.model import YOLO

model = YOLO("runs/detect/train11/weights/best.pt")
model2 = YOLO("yolov8m.pt")


def predict():
 results = model.predict(source="D:\GABRIEL_Masson\DeepL\Dataset\\test\\215.jpg", device=1,
stream=False, save=True)


def analyseVal():
 model.val(conf=0.5)
 model.val(data="custom.yaml", conf=0.5)


def exportONNX():
 model.export(format="onnx", opset=10)


def learning():
 model2.train(data="custom.yaml", imgsz=800, batch=8, epochs=45, device=1, plots=True)


if __name__ == '__main__':
 print(torch.cuda.get_device_name())
 print(torch.__version__)
 print(torch.version.cuda)
 learning()
```

Lancez le programme, attendez, et c'est tout.