Documentation de la partie « plan de vol » :

1 Création du plan de vol :

Fichiers : PlanDeVolVC.swift, Obstacle.swift, Objectif.swift

Langage : swift

Utilité : Création d'un plan de vol

Fonctionnement :

L'utilisateur crée un plan de vol en appuyant sur les différentes commandes. Chaque commande est alors ajoutée à la variable *listeCommande*.

L'utilisateur peut également rentrer des objectifs ou des obstacles en spécifiant les coordonnées (x,y,z). Ils sont alors stockés dans *listeObstacle* ou *listeObjectif*. Il est également possible de sauvegarder un plan de vol. (voir la documentation sur le gestionnaire)

L'affichage des commandes et de la positon des obstacles et objectifs se font dans une tableView avec 3 sections : Une pour chaque catégorie.

En appuyant sur le bouton simulation, une vérification grammaticale du programme est faite, et si elle est validée, nous arrivons alors sur la view simulation.

Grammaire du langage :

```
<expr> -> décollage + <terme>
<terme> -> <commande> + <terme> | fin
<commande> -> droite | gauche | avancer | reculer | monter | descendre
<fin> -> atterrissage
```

2 Simulation :

Fichiers : SimulationView.swift

Langage : swift

Utilité : Simulation d'un plan de vol

Fonctionnement :

Mise en place :

Utilise la technologie de sceneKit. Le nœud principal est un cube sceneKit d'arrête 20 où une unité correspond à environ 10cm dans la réalité. Dans lequel on a ajouté un nœud (sphère) représentant le drone (positionné en : *sphereNode.position = SCNVector3(x: 0, y: -10, z: 0)*). Cette position correspond au sol, centré sur l'axe x et y (Chaque axe va de -10 à 10)

Pour chaque obstacle/objectif, on ajoute un nœud représenté par des sphères/anneaux.

Simulation :

La fonction *simulate* est appelée lorsqu'on appuie sur le bouton simuler. On va alors parcourir la liste de commande, et ajouter les *SCNAction* dans une liste sequences. Pour chaque action, on met à jour la position du drone dans la variable position. Cela nous permet de vérifier s'il est toujours dans le cube. Dans le cas où il sort, on lève alors le flagPosition qui nous permettra d'indiquer l'erreur à l'utilisateur.

Cette position nous permet également de tester le passage d'objectifs et l'impact avec un obstacle un obstacle.

Une fois la liste sequence complétée, on va alors la jouer avec *runAction* ce qui va donner la simulation tel que l'utilisateur la verra. Si la simulation se déroule bien, alors le bouton pour lancer sur le drone apparaîtra.

3 Le vol réel :

Fichiers : simulation.swift

Langage : swift

Utilité : Le drone effectue le plan de vol programmer par l'utilisateur.

Fonctionnement :

La variable *chercheur*, qui est une instance de *« DroneDiscover »* va s'occuper de la recherche de drone dès le chargement de la vue. De cette manière on pourra actualiser la tableView. Une fois un drone sélectionné, le drone sera représenté par la variable *bebopDrone*.

Une fois que l'utilisateur appuie sur le bouton « *Lancer Drone »*, la fonction *launchDrone* sera exécutée. Dans cette fonction, on lit une liste d'entier qui contient les commandes. Chaque entier est alors envoyé au drone et interprété (avec un délai afin de voir le drone effectué les différentes commandes). Il est nécessaire que ce soit le thread principal de l'application qui envoie les commandes au drone.

```
Lien entre numéro et les commandes
0: Decollage
1: Aterrisage
2: Droite
3: Gauche
4: Avancer
5: Reculer
6: Monter
7: Descendre
```