UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Département d'informatique
Université de Genève
Travail de bachelor

DroneProgrammer

Application iOS
github: mathiasTonini/droneProgrammer

Étudiant:
Mathias Tonini

Professeur:
Didier Buchs

semestre de printemps 2020

Liste des figures

Table des matières

Chapitre 1: Introduction

1.1 Utilisation des drones

Les drones seront de plus en plus présents dans nos vies. De nos jours, les drones sont déjà utilisés pour de nombreuses activités différentes telles que l'agriculture, la prise de vidéo pour des films/vidéos/documentaires, la recherche de survivants etc... À l'avenir, les drones prendront encore plus de place dans nos vies. Les entreprises de livraison cherchent à les utiliser afin de pouvoir livrer des colis à moindre coût et plus rapidement, les systèmes de santé se penchent sur leur possible utilisation en tant qu'ambulances [4]. Les domaines dans lesquels les drones peuvent être utiles semblent ainsi infinis.

1.2 Ce travail

L'objectif de ce travail est d'apporter une nouvelle utilisation aux drones: les utiliser dans un but éducatif, notamment dans le milieu de la programmation. Ce travail consiste donc principalement au développement d'une application iOS qui permet la programmation en bloc de plans de vol et le test ainsi que la simulation de ces plans de vols avant leur exécution par le drone.

1.3 État de l'art

1.3.1 Playground

Dans l'univers de la programmation de drone, notamment par bloc il existe *Swift Playground*, une application développée par Apple dans le but d'apprendre la programmation de manière ludique autour d'un jeu vidéo d'aventure [2]. Il est possible de programmer des drones Parrot depuis playgrounds. Cependant, playgrounds pour Parrot n'est disponible que pour les mini drones. Or, l'Université de Genève

possédant déjà un Parrot Bebop 2, il a été préféré de développer notre propre moyen de programmer le drone, avec l'application droneProgrammer [3].

1.3.2 Tynker

Tynker est une plateforme d'apprentissage de la programmation. La grande particularité de Tykner et le fait d'être de la programmation par bloc. C'est à dire qu'à la place d'écrire des lignes de codes, l'utilisateur *drag and drop* des blocs qui représentent des instructions[6].

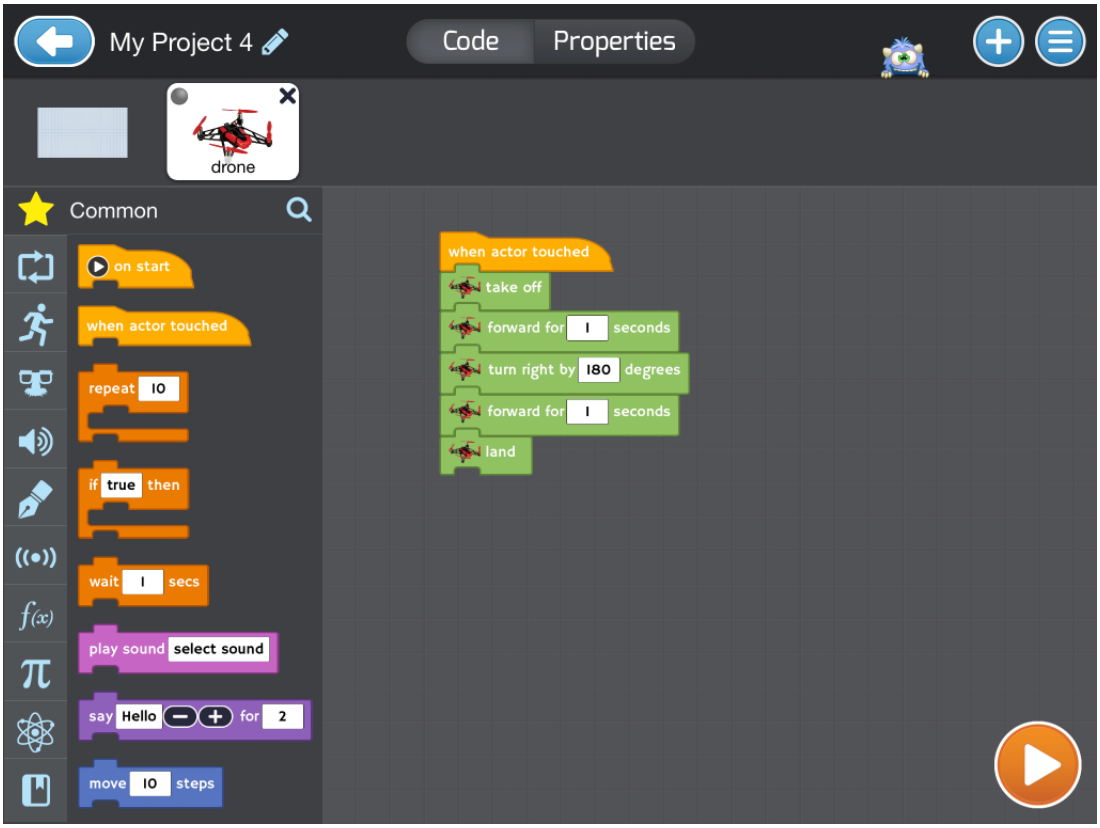

Figure 1.1: Programme Tynker pour un Parrot Rolling Spider minidrone

Source: site web de Tynker

Cette plateforme se rapproche beaucoup de ce que nous cherchons à faire, cependant elle n'est malheureusement ni gratuite, ni compatible avec le drone bebop 2.

1.3.3 Workbench Education

Workbench est une plateforme d'apprentissage orientée sur tout ce qui est technologie. La Parrot Flight school permet de créer des projets collaboratifs ou de programmer le drone directement depuis un navigateur web comme le montre l'image suivante:

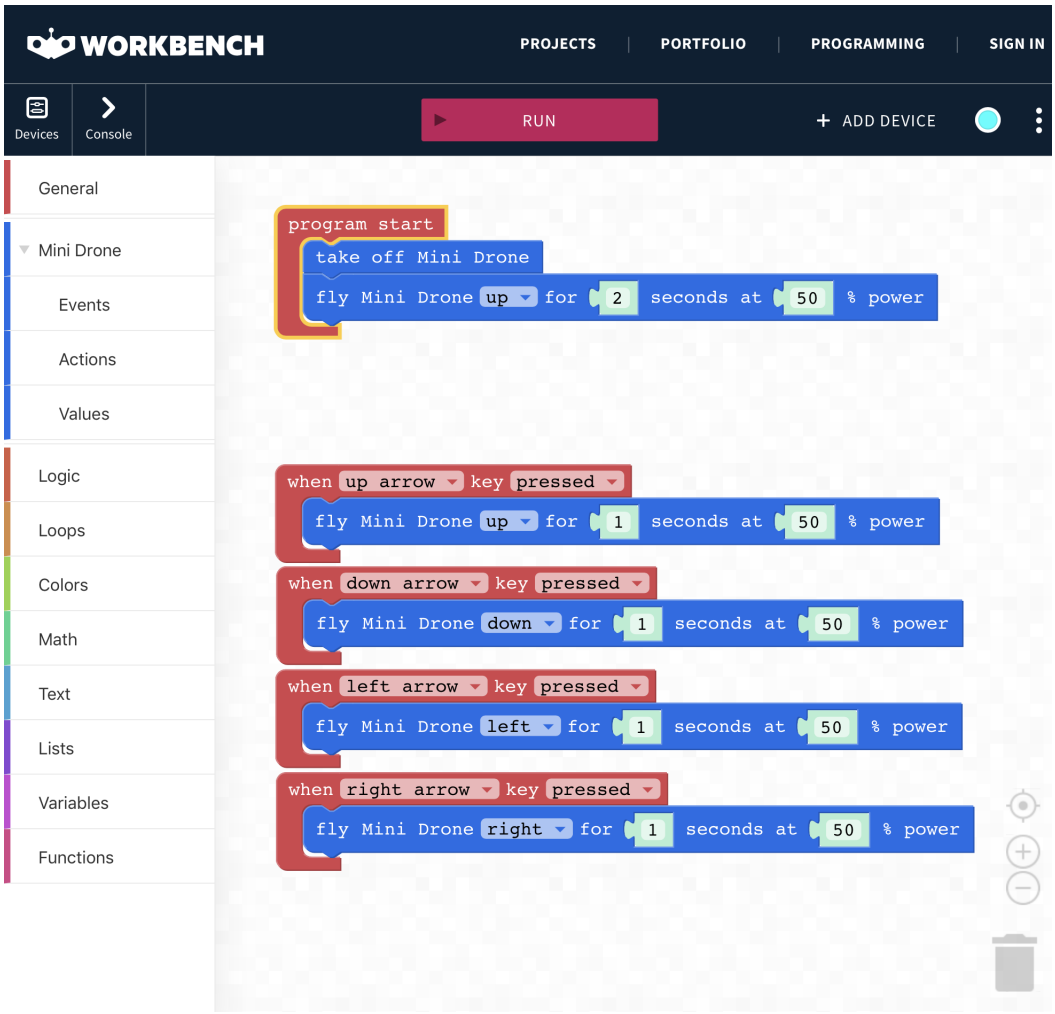

Figure 1.2: Programme workbench sur un navigateur web

Source: Site web de Parrot

Cependant avec cette option nous rencontrons le même problème que pour les deux autres, elle n'est malheureusement disponible que pour les mini-drones Parrot et non pas avec le bebop2 [5]

1.3.4 Bilan

Au final nous voyons qu'il existe déjà des solutions sont déjà proposées pour la programmation par bloc pour les drones Parrot. Cependant, aucune des propositions mentionnées ci-dessus ne permet de le faire avec le Bebop 2. Parrot propose également des package Simulink, python et javascript. Cependant, utiliser des langages de programmation comme cela n'est pas le but du projet puisque nous cherchons a faire de la programmation par bloc [1].
C'est pour cela qu'au final nous nous sommes tournés vers le SDK proposé par Parrot afin de créer notre propre application programmation par bloc pour le drone Parrot Bebop 2.

Chapitre 2: Le point de départ

L'application ne partait pas de nulle part. En effet, un certain nombre de chose existait déjà et nous nous sommes donc appuyés dessus pour ce projet

2.1 L'ancienne application: drone-control

Une ancienne application qui s'appelle drone-control, existait déjà. Elle était composée de deux parties distinctes.

2.1.1 Le vol libre

Cette fonctionnalité se rapproche beaucoup de celle qui existe sur l'application droneProgrammer. Elle permet de piloter en direct le drone avec de simples instructions telle que "Monter", "Descendre", "Droite" etc...

2.1.2 Le plan de vol

La partie plan de vol de drone-control est illustrée par la figure 2.1: afin de faire un programme exécutable par le drone, l'utilisateur doit effectuer la suite d'actions suivante:

1. Écrire les instructions possibles (visibles sur la droite de l'image) dans le carré gris en haut à gauche de la *view* et les valider avec le bouton prévu.

2. Rentrer le nombre d'obstacles que le drone doit éviter ainsi que leurs coordonnées et valider les entrées.

3. Délimiter les coordonnées minimales et maximales de la zone de vol du drone. Cela représente un cube duquel le drone ne pourra pas sortir.

4. L'utilisateur peut ensuite essayer son plan de vol. L'application interprétera alors les commandes rentrées et vérifiera qu'aucun obstacle ne sera percuté et que le drone restera dans la zone délimitée.

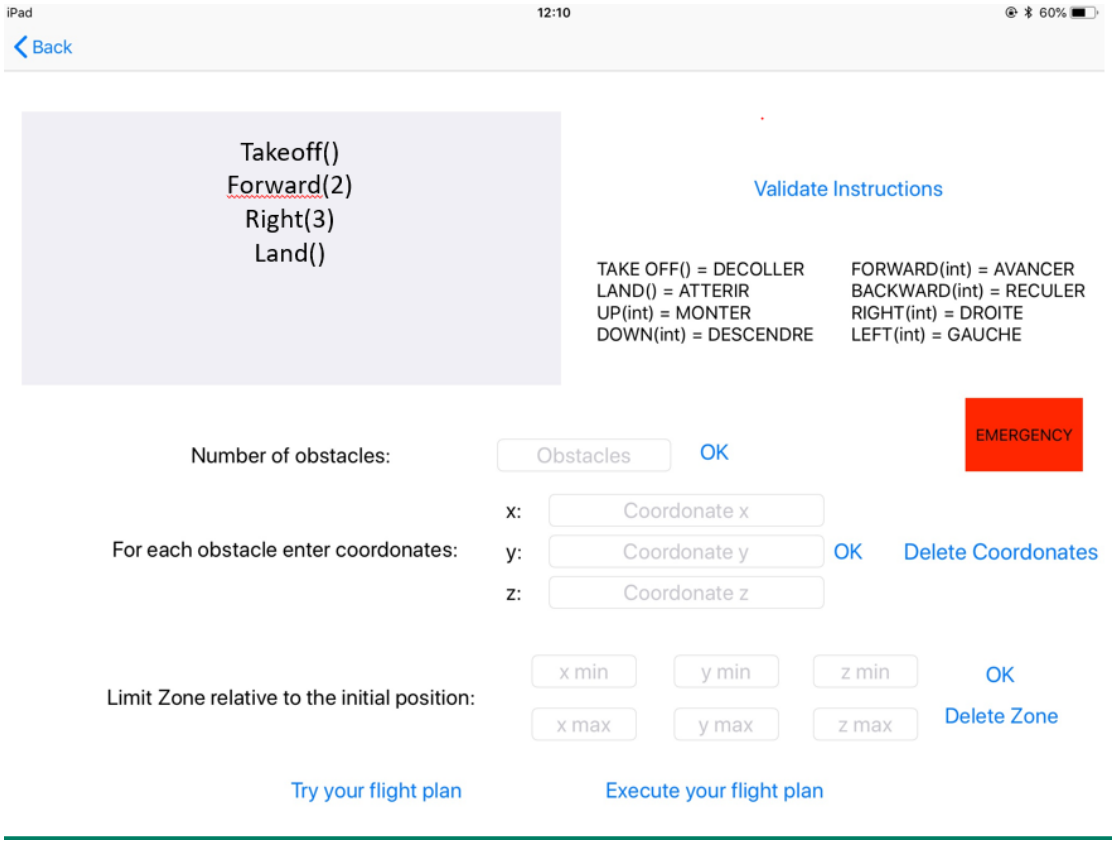

Figure 2.1: Capture d'écran de l'éditeur plan de vol de drone-control

Nous verrons que ce projet apporte une simplification de la programmation de plan de vol en utilisant notamment des blocs simples de commandes, ainsi qu'une partie simulation de vol plus visuelle.

2.2 Le drone

Pour le développement de l'application, nous avons utilisé un *drone Parrot bebop 2*[1], illustré sur la figure 2.2. Les principaux éléments du drone sont:

- Une caméra grand angle de 14 Mpx.
- Un gps
- Quatres moteurs permettant de se diriger

[1] *Site officel de Parrot*

Figure 2.2: Photo d'un drone Parrot Bebop 2

Source: Site officiel de Parrot

2.2.1 Le SDK Parrot

Parrot fournit un SDK [2] qui permet d'accéder à différents éléments de l'état du drone où encore de lui envoyer des instructions. Ce SDK étant très complet nous avons préférer utiliser les samples (que l'on peut voir comme une version allégée et facilitée su SDK) pour développer l'application droneProgrammer

2.2.2 Les samples

Parrot met à disposition des samples disponibles afin de faciliter l'utilisation du SDK, disponibles sur *github* et dont le modèle UML est illustré par la figure 2.3. Ces samples sont cependant écrits en Objective-C alors que pour le développement d'une application nous souhaitions développé en Swift. Pour faire le lien entre les samples, l'application nous avons donc utilisé un *bridging header* qui nous permet d'utiliser des méthodes écrites en Objective-C en swift.

[2] Un kit de développement logiciel, aussi appelé trousse de développement logiciel, est un ensemble d'outils logiciels destiné aux développeurs, facilitant le développement d'un logiciel sur une plateforme donnée. *(Wikiepdia)*

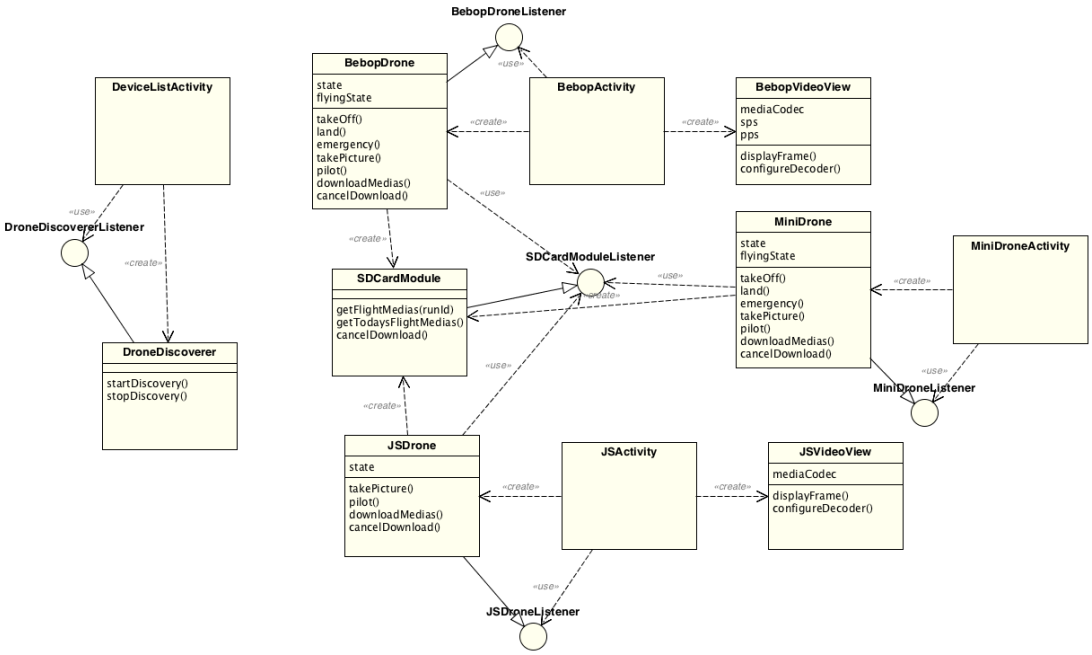

Figure 2.3: Représentation des samples fournis directement par Parrot

Chapitre 3: L'application

3.1 Le développement de l'application

Pour commencer le projet, nous avons tout d'abord commencé par faire des esquisses de story board afin d'avoir une visualisation du projet dont on peut voir un exemple à la figure 3.1. Nous avons également modéliser la structure du projet

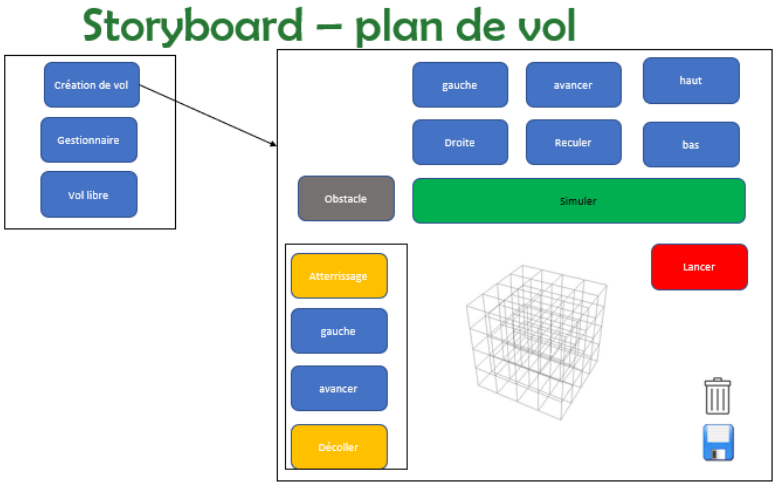

Figure 3.1: Exemple d'une storyboard de présentation

comme on peut le voir à la figure 3.2. Ce modèle que nous avons fait au début n'a pas changé et nous a permis dès le début d'estimer la charge de travail de manière à finir le projet dans les temps.

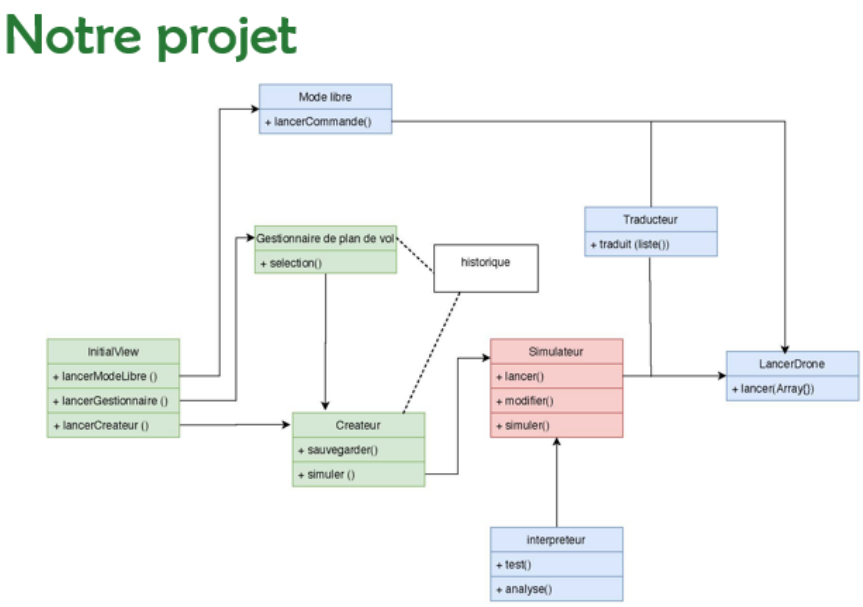

Figure 3.2: Modèle de l'application

Nous avons travaillé en *sprint* de deux semaines. Cela nous a permis de pouvoir faire le point sur l'avancement, de se donner un certain nombre de tâches à faire toutes les deux semaines et de vérifier que nous avancions dans le bon sens, ainsi que d'actualiser notre planning au fur et à mesure.

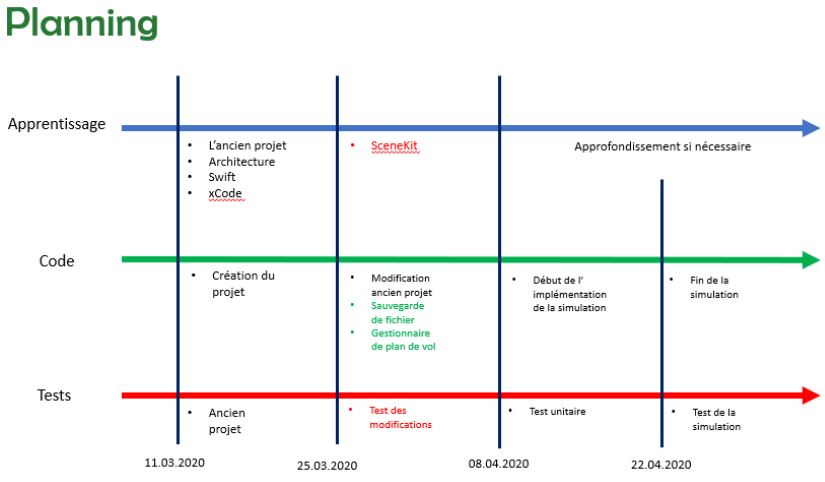

Figure 3.3: Exemple d'un planning

3.2 Structure générale

Lorsque l'on ouvre l'application, l'utilisateur arrive sur un menu qui nous permet de choisir quel mode de l'application il souhaite utiliser. Comme on le voit sur

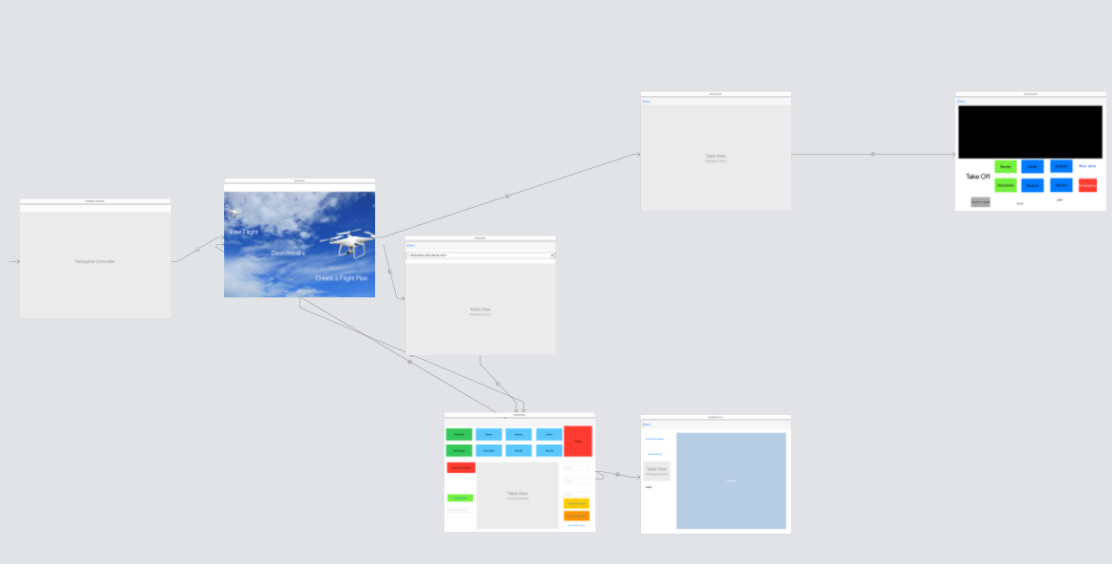

Figure 3.4: Storyboard droneProgrammer

la figure 3.4, l'application est séparée en trois parties distinctes: La partie "*free flight*" indépendante et les partie "*gestionnaire*" et "*Create a flight plan*" qui sont reliées entre elles. Ainsi, l'utilisateur peut soit piloter le drone directement depuis l'application soit aller chercher un plan de vol précédemment créer dans le gestionnaire ou alors écrire un nouveau plan de vol, le simuler et s'il est accepté alors exécuter le plan de vol sur le drone.

3.2.1 Vol libre

Cette partie de l'application est là pour piloter le drone en direct. Comme on le voit dans la figure 3.5, on a plusieurs commandes pour piloter le drone: On a d'abord un bouton "TakeOff/Land" qui permet donc de décoller ou d'atterrir avec le drone. D'autres boutons permettent au drone d'avancer, reculer, monter, descendre, aller à droite, ou encore aller à gauche. On affiche également la batterie du drone. Le bouton de prise de photo n'est malheureusement pas opérationnel en raisoin d'un problème avec le drone durant le développement. Dans le code, toutes les fonctions qui y sont liées sont commentées. L'état actuel des fonctions est qu'elles permettent de prendre des photos et de les télécharger sur l'iPad.

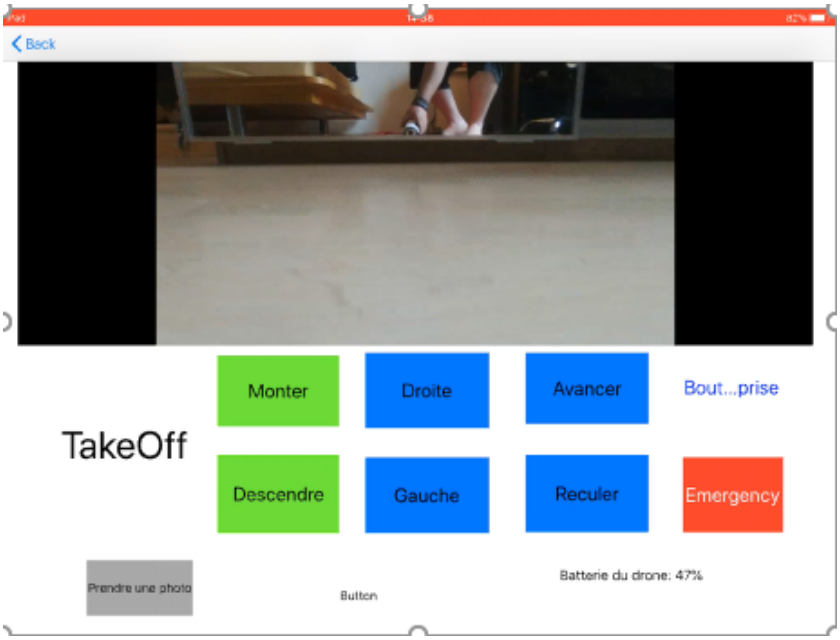

Figure 3.5: Capture d'écran de la partie free flight

Cependant elle restent dans le système de fichier de l'application et pas celui de l'iPad. On ne peut donc pas y accéder dpuis la galerie.

3.2.2 Gestionnaire

Le gestionnaire de plan de vol permet de sauvegarder et récupérer des plans de vol écrits précédemment.

Figure 3.6: Capture d'écran du gestionnaire

Comme on le voit dans la figure 3.6, visuellement il consiste en une simple tableView qui affiche les nom des plans de vol. En appuyant sur un nom, on sera redirigé vers la view d'édition avec le plan de vol chargé. Chaque plan de vol est encodé en Json de la manière suivante:

```
struct Fichier: Decodable {
    let listeCommande: [Int]
    let listeObstacle: [[Int]]
    let listeObjectif: [[Int]]
    let nom: String

    enum CodingKeys: String, CodingKey {
        case listeCommande = "ListeCommande"
        case listeObstacle = "ListeObstacle"
        case listeObjectif = "ListeObjectif"
        case nom
    }
}
```

De cette manière la tableView ne récupère que le nom du plan de vol et passera le reste du plan de vol par les segues lors du chargement de la page d'édition.

3.2.3 Programmation du plan de vol

La partie plan de vol se compose tout d'abord en un langage de programmation défini par la grammaire suivante:

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

A partir de cette grammaire l'utilisateur utilise l'interface graphique que l'on voit à la figure 3.7 pour programmer un plan de vol. Ainsi en appuyant sur les différentes

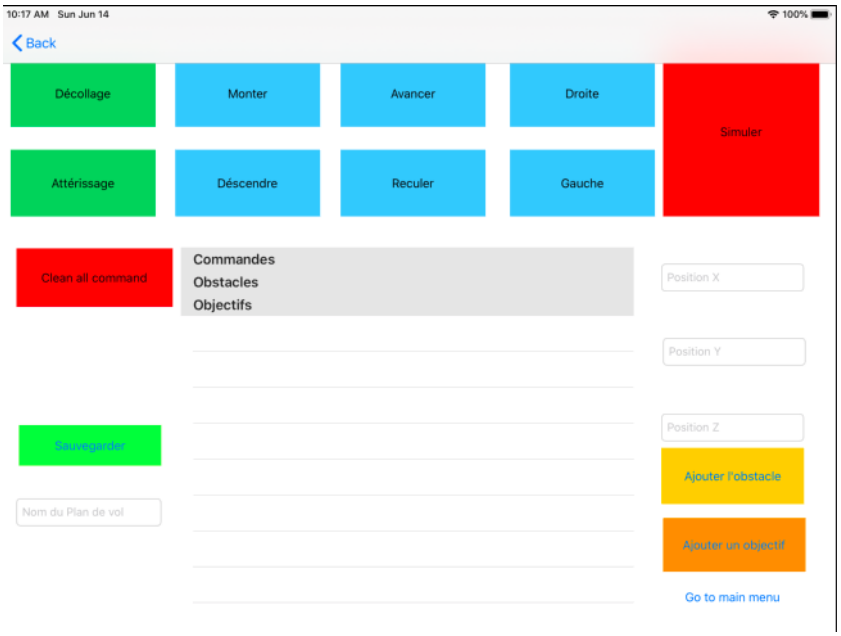

Figure 3.7: Interface de programmation d'un plan de vol

commandes sur le haut de la vue, l'utilisateur peut ainsi créer un programme pour le drone. Chaque commande déplace le drone d'une unité (Une unité correspond à un mouvement d'environ 10cm).
Un plan de vol est également définit par une liste d'obstacles et d'objectifs. L'utilisateur pourra rentrer les coordonnées sur la droite de la vue et préciser s'il souhaite ajouter un obstacle ou un objectif.
L'utilisateur peut également enregistrer un plan de vol qu'il vient de programmer en utilisant le champ sur la gauche de la vue.
La figure 3.8 nous donne un exemple d'un plan de vol complet. Lorsque l'utilisateur va appuyer sur le bouton simuler, une analyse syntaxique sera faite pour vérifier

que le programme correspond à la grammaire ci-dessus. Si l'analyse est réussie, l'utilisateur arrivera alors sur la view simulation.

Figure 3.8: Exemple d'un plan de vol complété

3.2.4 La simulation

Lorsque l'utilisateur arrive sur la simulation, la view qui apparaît est donnée par la figure 3.9. La simulation est représentés par un cube de 20 unités d'arêtes.

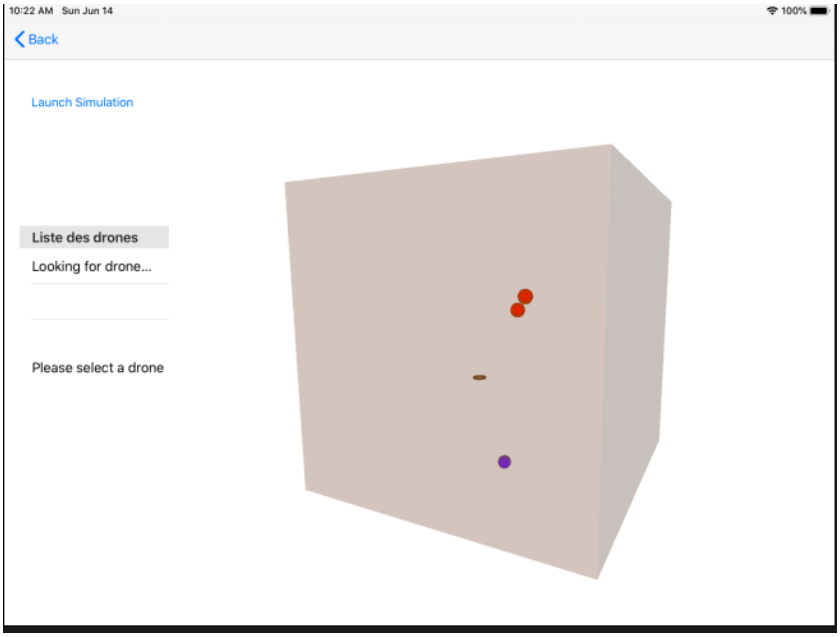

Figure 3.9: View d'une simulation

La sphère violette représente le drone, l'anneau brun un objectif tandis que les sphères rouges représentent les obstacles. Les figures 3.10 , 3.11 et 3.12 illustrent les différents problème qui peuvent être soulevés par la simulation. La figure 3.13 représente quand à elle une simulation réussie. On voit d'ailleurs qu'un bouton pour lancer le drone apparaît à ce moment.

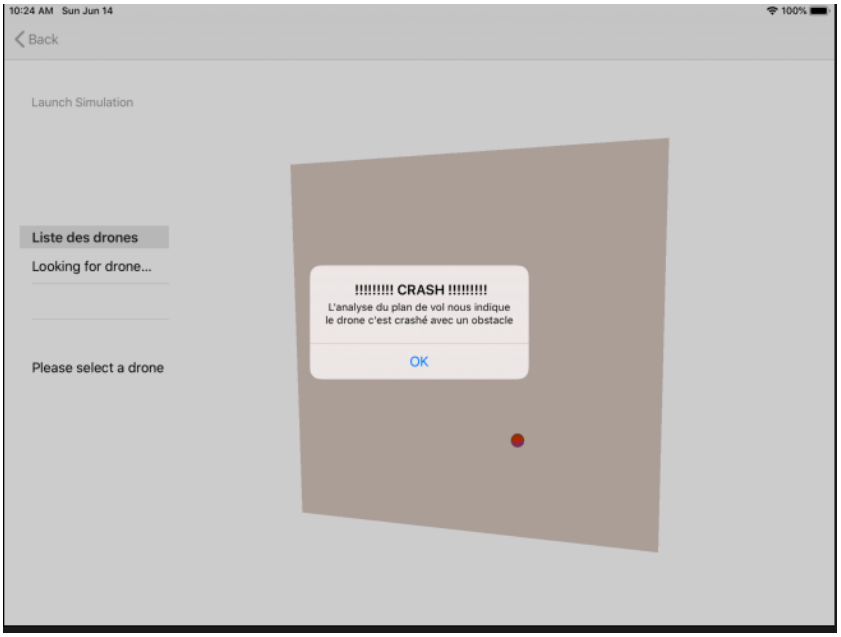

Figure 3.10: Exemple d'un crash lors d'une simulation

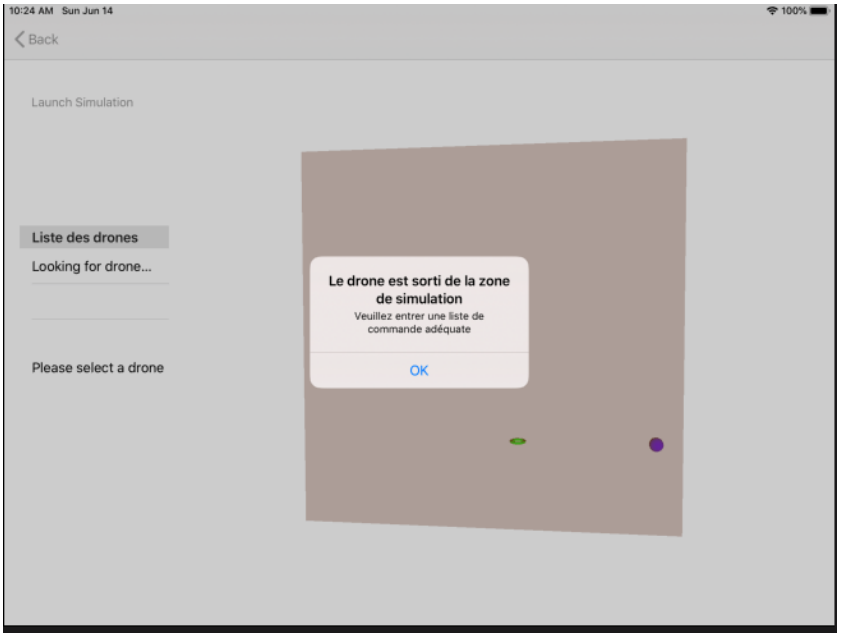

Figure 3.11: Exemple d'une sortie de zone lors de la simulation

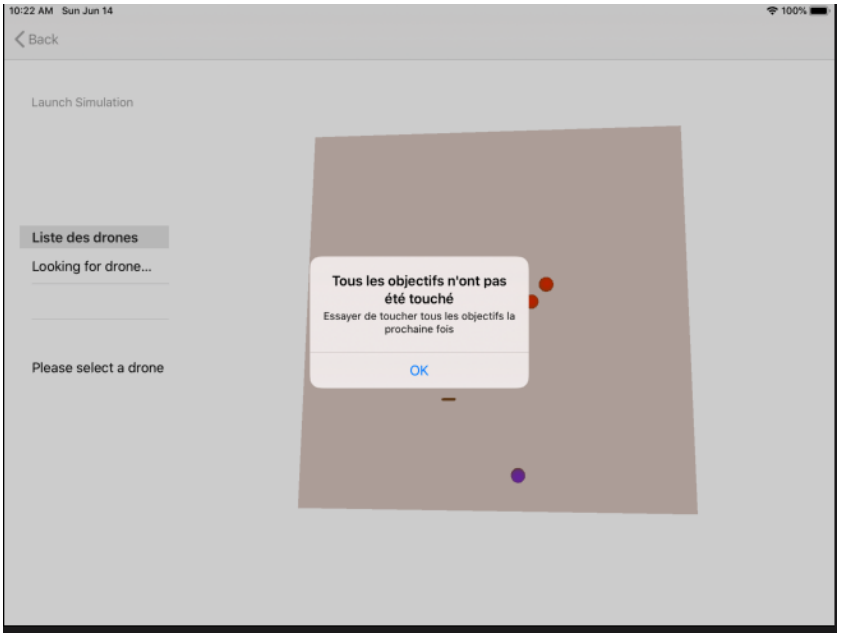

Figure 3.12: Exemple d'une simulation dans laquelle les objectifs ne sont pas tous atteints

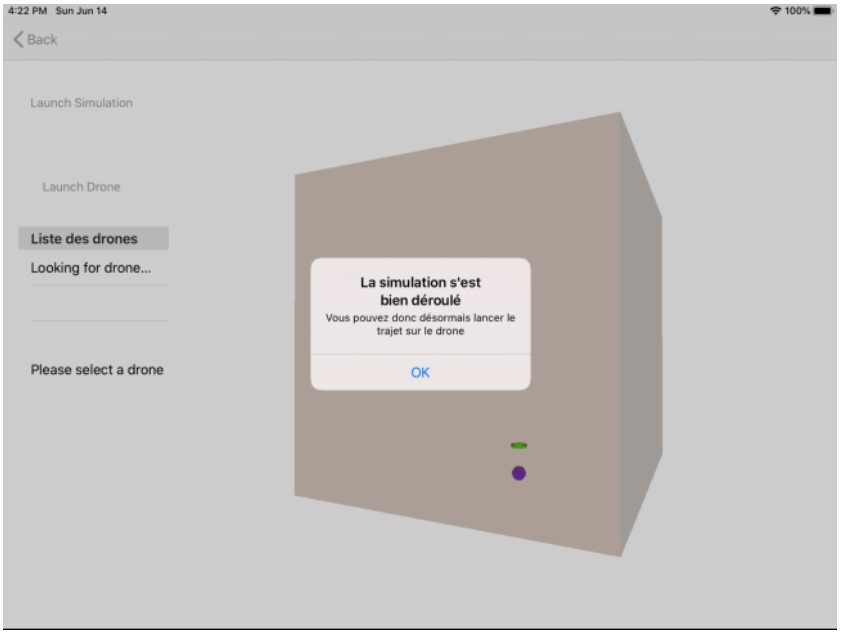

Figure 3.13: Exemple d'une simulation réussie

3.2.4.1 Le développement de la simulation

Afin de développer la simulation nous avons utilisé la libraire sceneKit[1]. La page est composée d'une seule scène. Le cube principal ainsi que le drone sont représentés par un noeud comme pour chaque obstacle et objectif. À ces noeuds se rajoute celui de la caméra qui représente la manière dont on voit la scène. Ainsi, L'utilisateur a la possibilité de bouger la caméra (ce qui aura pour conséquence de déplacer le cube de son point de vue). L'application va ensuite boucler sur chaque commande que l'utilisateur a rentré et effectuer un certain nombre de tests (tel que vérifier si le drone est encore dans la zone de vol, s'il percute un obstacle ou encore si un objectif est traversé. En fonction de ces conditions on va créer une liste de *SCNaction* qu'on va remplir en même temps qu'on lit les commandes.

```
1 self.sphereNode.runAction(moveSequence)
```

La ligne ci-dessus permet au *sphereNode* (qui représente donc notre drone) d'effectuer les séquences comprises dans la liste *moveSequence* qui contient les instructions. Ainsi, on à la simulation du vol. La méthode est également attachée à un *completionHandler* qui permet d'afficher les différentes *alertView* en fonction de la lecture des commandes.

[1]Lien vers le site internet du framework

Chapitre 4: Les problèmes

Dans ce chapitre, je vais décrire les différents problèmes auxquels nous avons été confrontés durant ce projet.

4.1 Les technologies

4.1.1 Swift et l'éco-système Apple

Afin de développer droneProgrammer, nous avons du apprendre de nombreuses choses. En effet, nous n'étions pas familiés avec le langage de programmation Swift ni avec l'éco-système d'Apple. Cependant, au fur et à mesure que le projet avançait, cette difficulté devenait de moins en moins importante, car nous maîtrisions de mieux en mieux le langage et prenions l'habitude de travailler avec les appareils Apple.

4.1.2 SceneKit

Cette librairie est une libraire très utile et puissante, cependant son apprentissage est relativement long et difficile. Comme la simulation a été un des derniers éléments à être ajouté à l'application, nous avons pu appréhender ce problème en étant déjà à l'aise avec Swift. De cette manière, en nous renseignant bien et cherchant la documentation, nous avons pu assez rapidement maîtriser les éléments qui étaient nécessaires à réalisation de ce projet.

4.1.3 Les samples

L'utilisation de samples écrits en Objective-C nous semblait être une partie relativement technique de la programmation de l'application. Néanmoins la documentation était complète, de plus, étant donné que nous avons repris de nombreux éléments de l'ancienne application, nous avions des exemples concrets de comment les utiliser pour ce genre de projet.

4.2 Reprise de l'ancienne application

Comme dit précedemment, l'application droneProgrammer est basée sur une application déjà existante (disponible *ici*). Comme cette application était écrite en Swift et que nous avions justement peu de connaissances de ce langage nous avons eu beaucoup de peine à la comprendre au début. Or comme cette application était notre point de départ, nous voulions vraiment la comprendre afin de pouvoir profiter au maximum de ce travail pouvant nous être utile.
Nous avons donc pris contact avec les étudiants qui l'ont développée afin de comprendre un peu mieux le fonctionnement. Cela nous a beaucoup aidé et permis de savoir d'où nous partions et de mieux comprendre le travail qu'il nous fallait faire pour arriver où nous le souhaitions.

4.3 COVID-19

Comme beaucoup de travaux ce printemps, ce projet a également été impacté par le COVID-19 et ses conséquences. En effet, lors de la fermeture des bâtiments universitaires, nous avons dû travailler sans avoir accès au drone. Ce virus nous a également empêché de nous rencontrer autant avec le professeur et les assistants afin de montrer la continuité du projet qu'entre nous afin de discuter de la répartition des tâches et de l'organisation du travail en général.
Nous avons finalement pu trouver une solution et ramener le drone chez nous, ce qui nous a permis de pouvoir tester l'application des fonctionnalités

4.4 Erreur du drone

Alors que nous arrivions à la fin du projet, le drone a eu un souci que nous n'avons malheureusement pas pu réparer par nous même. De ce fait, les dernières parties que nous avons implémentées n'ont pas pu être testées. Cela concerne la recherche, la connexion au drone dans la partie simulation ainsi que l'exécution du plan de vol. C'est également la raison pour laquelle nous n'avons pas pu terminer l'implémentation de la prise de photo.

Chapitre 5: Conclusion

5.1 En résumé

Dans ce projet, nous avons pris une ancienne application que nous avons améliorée. Dans la partie de vol libre, nous avons ajouter le flux vidéo du vol ainsi qu'une majeure partie de l'implémentation de la prise de photo.
Nous avons également créer un gestionnaire de plan de vol qui permet de sauvegarder les plans de vols écrits et de les charger directement dans l'éditeur de plan de vol.
Le créateur de plan de vol a été refait à partir de zéro afin d'avoir une programmation par bloc plus claire que celle qui existait déjà dans l'ancienne application. En plus des obstacles déjà existants dans l'ancienne application, nous avons ajouté la possibilité d'avoir des objectifs à traverser avec le drone.
Nous avons également ajouté toute la partie simulation du vol qui permet à l'utilisateur d'avoir une représentation graphique de comment le drone se déplace. Cela permet également de visualiser à quel moment se fera une éventuelle collision avec un obstacle ou pourquoi il ne le traverse pas. Elle permet également de voir à quel moment du plan de vol le drone sort de la zone définie.

5.2 Personnelle

Ce projet m'a personnellement beaucoup apporté. Pour commencer il m'a permis de comprendre avant tout l'importance de la documentation lors d'un projet d'une certaine ampleur. L'ancienne application n'étant que peu documentée, nous avons perdu beaucoup de temps à comprendre son fonctionnement.
C'est également avec plaisir que j'ai pu apprendre le fonctionnement et la création d'application, particulièrement pour iOS dans le cadre de ce projet.
Au final, ce projet m'a également permis d'apprendre de nouvelles compétences telles que la programmation en Swift, certaines bases pour l'utilisation de SceneKit, l'utilisation de sample, ou plus général encore, le travail en groupe.

5.3 Pour aller plus loin

5.3.1 Terminer la prise de photo

Pour continuer avec cette application, une idée serait de terminer le développement de la prise de photo dans le mode *Free Flight*. La majeure partie de la fonctionnalité est déjà implémentée. En effet le drone prend les photos et les renvoie à l'iPad à chaque fin de vol. Il ne reste plus qu'à afficher les photos soit dans l'application même ou alors dans la galerie de l'iPad.

5.3.2 Ajout d'une view lors de l'exécution d'un plan de vol

Une amélioration possible de l'application serait également d'avoir une view dédiée durant l'exécution d'un plan de vol dans laquelle on pourrait avoir accès au flux vidéo du drone ainsi que la possibilité de prendre des photos. De cette manière, on pourrait ajouter comme objectif des prises de photos à certains endroits durant le vol de drone et amélioré le coté ludique de l'application.

5.3.3 Approfondir le langage

Afin d'obtenir un application plus complète, un élément qui pourrait être améliorer serait le langage de programmation du drone. En effet, le langage est très simple pour l'instant, mais il serait possible de l'étoffer en ajoutant notamment des boucles d'instructions.

5.3.4 Simulation sur une carte

Un autre élément qu'il serait possible d'améliorer est la simulation. En effet, depuis le drone il est possible de récupérer les coordonnées GPS. Il serait dès lors possible de faire la simulation du plan de vol directement sur une carte (p. ex. utiliser l'API d'openStreetMaps) afin que l'utilisateur puisse avoir une meilleure visualisation du vol qu'il a programmé.

[1] *ARDroneSDK3 API Reference*. URL: https://developer.parrot.com/docs/SDK3/#general-information (visited on 06/13/2020).

[2] BFMTV. *Swift Playgrounds: l'appli gratuite d'Apple pour apprendre à coder aux enfants est là*. BFMTV. Library Catalog: hightech.bfmtv.com Publisher: BFMTV. URL: https://hightech.bfmtv.com/logiciel/swift-playgrounds-apple-veut-apprendre-les-enfants-a-coder-et-gagner-leur-coeur-1036814.html (visited on 05/29/2020).

[3] *Coding Apps for Parrot Mambo*. URL: https://edu.parrot.com/apps.html (visited on 05/29/2020).

[4] Yan Hélin. *Quelles sont les applications et usages des drones ?* Dronedecole. Apr. 7, 2016. URL: https://dronedecole.fr/differentes-utilisations-drones/ (visited on 05/28/2020).

[5] *Parrot*. Library Catalog: edu.workbencheducation.com. URL: https://edu.workbencheducation.com/partners/parrot (visited on 06/11/2020).

[6] *Tynker | Coding for Kids*. Tynker.com. Library Catalog: www.tynker.com. URL: https://www.tynker.com/learn-to-code/code-this-drone/ (visited on 06/11/2020).