DGA EV, Base Aérienne 120, 33260 Cazaux

Rapport de stage d'ingénieur de 2ème année

Filière F2 : « Génie Logiciel et Systèmes Informatiques »

Développement de l'application mobile EasyHover

Présenté par :

Anaël Courjaud

Maître de stage : Thibault L.

Tuteur académique : Vincent Limouzy

Date de la soutenance : 13.07.2023

Durée du stage : 4 mois

Campus Universitaire des Cézeaux • 1 Rue de la Chebarde • TSA 60026
63178 Aubière cedex

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage.

Tout d'abord, j'adresse mes remerciements à mon enseignant référent Vincent Limouzy qui m'a aidé dans ma recherche de stage. Son écoute et ses conseils m'ont permis de cibler mes candidatures, et de trouver ce stage qui était en totale adéquation avec mes attentes.

Je tiens à remercier mon maître de stage, Thibault L., pour son accueil, le temps et la confiance qu'il m'a accordés et le partage de son expertise, qui m'ont permis de m'accomplir totalement dans mes missions. Il fut d'une aide précieuse dans les moments les plus délicats.

Je remercie également toute l'équipe de l'innovation Flight Test Lab (IFTL) de la DGA EV pour leur accueil, leur esprit d'équipe et en particulier Marc qui m'a beaucoup aidé à comprendre le besoin des pilotes pour ne pas me tromper dans la réalisation de cette application.

Enfin, je tiens à remercier toutes les personnes qui m'ont conseillé et relu lors de la rédaction de ce rapport de stage : Xavier et mes camarades stagiaires.

Table des matières

Table des figures

Résumé/abstract

Stage effectué au sein de la **DGA Essais en Vol**, service IFTL du site de Cazaux. Le but est de développer une **application mobile** nommée « **EasyHover** » destinée à assister les **pilotes d'hélicoptères** dans le maintien de **vols stationnaires**. Lors d'essais en vol à la DGA EV, il peut y avoir un manque de repères visuels extérieurs fixes ce qui rend la manœuvre délicate sans assistance. L'application fonctionne sur les tablettes embarquées des pilotes. Cette application est disponible gratuitement sur le Play Store (Android) et l'App Store (iOS).

Après la validation de la **conception** grâce à des **maquettes** (**LibreOffice Draw**) et des **prototypes interactifs** (**Figma**), l'application a été développée à l'aide de **.Net MAUI**, framework utilisant à la fois du **C#** et **XAML**. Le développement a été réalisé sur l'EDI **Visual Studio Community 2022**. Des tests ont été effectués au fur-et-à-mesure du développement sur un téléphone **Android** Xiaomi Redmi Note 11 Pro. Des essais au sol puis en vol ont été prévus pour valider le bon fonctionnement de l'application avant sa livraison. Les versions successives du code sont stockées sur un dépôt **Git**. Utilisation de **Trello** pour la gestion de projet. A ce jour, l'application en est toujours à sa phase de développement.

Mots-clés : DGA / application mobile / EasyHover / .Net MAUI / C# / XAML / pilote d'hélicoptère / vol stationnaire / tablette embarquée / conception / maquettes / LibreOffice Draw / prototype interactif / Figma / Visual Studio Community 2022 / Android / essais en vol / Git / Trello

Internship at the **DGA Flight Tests**, IFTL department at the Cazaux site. The aim is to develop a **mobile application** called "**EasyHover**" to help **helicopter pilots** maintain **hovering flights**. During flight tests at DGA EV, there may be a lack of fixed external visual cues, making the manoeuvre tricky without assistance. The application runs on pilots' **onboard tablets**. The application is available free of charge from the Play Store (Android) and the App Store (iOS).

After validating the design using **mock-ups** (**LibreOffice Draw**) and **interactive prototypes** (**Figma**), the application was developed using **.Net MAUI**, a framework using both **C#** and **XAML**. Development was carried out using the **Visual Studio Community 2022** IDE. Tests were made as development progressed on a Xiaomi Redmi Note 11 Pro **Android** phone. Ground tests and then flight tests were scheduled to validate that the application was working properly before delivery. Successive versions of the code are stored on a **Git** repository. Using **Trello** for project management. To date, the application is still in the development phase.

Keywords: DGA EV / mobile application / EasyHover / .Net MAUI / C# / XAML / helicopter pilot / hovering flight / onboard tablet / design / mock-ups / LibreOffice Draw / interactive prototype / Figma / Visual Studio Community 2022 / Android / flight tests / Git / Trello

Glossaire

.net MAUI : C'est un framework qui permet aux développeurs de créer des applications multiplateformes avec un seul code source, leur permettant de concevoir des interfaces utilisateur pouvant s'exécuter sur plusieurs plateformes telles que Windows, macOS, iOS et Android.

Aéronef : Appareil capable de se déplacer dans les airs (avion, hélicoptère, aérostat…).

Assiette : L'assiette d'un hélicoptère fait référence à l'angle formé par l'inclinaison longitudinale de l'appareil par rapport à l'horizon.

Cadence : La cadence d'un hélicoptère fait référence à l'angle formé par l'inclinaison longitudinale de l'appareil par rapport à l'axe Nord/Sud du référentiel terrestre. La cadence varie autour de l'axe de lacet.

Cap/Cap vrai : Le cap fait référence à la direction vers laquelle un aéronef est orienté par rapport au nord magnétique de la planète Terre. Le cap vrai fait référence à la direction vers laquelle un aéronef est orienté par rapport au nord géographique de la planète Terre

Capteur GNSS : Un capteur GNSS (Global Navigation Satellite System) est un dispositif qui utilise des signaux satellites pour déterminer la position et la vitesse du récepteur à la surface de la Terre.

Cellule : Pour un hélicoptère, la cellule est constituée du fuselage et du train d'atterrissage

Conception UX/UI : La conception UX/UI (User Experience/User Interface) est un processus qui vise à créer des interfaces utilisateur conviviales et intuitives, en mettant l'accent sur l'expérience globale de l'utilisateur. Elle implique la recherche, l'analyse et la conception des interactions, de la disposition visuelle et des éléments graphiques pour améliorer l'utilisabilité et l'attrait esthétique des produits numériques. La conception UX/UI vise à optimiser la satisfaction de l'utilisateur en facilitant l'accès à l'information, en simplifiant les actions et en offrant une expérience cohérente et plaisante tout au long de l'interaction avec l'application ou le site web.

Débugger / Debug : Le Debug est le procédé par lequel un développeur cherche la source des dysfonctionnements dans son programme et tente de les corriger. Un Débugger est un outil, généralement intégré aux éditeurs modernes, qui permet de faciliter la recherche et l'identification de failles. Par exemple, il peut offrir un mode "pas à pas" pour exécuter le programme ligne par ligne, ou des aides pour trouver la source de fuites mémoire.

Design responsif : La conception responsive (ou design responsif) est une approche de conception qui vise à créer des interfaces utilisateur qui s'adaptent et se redimensionnent automatiquement en fonction des différents appareils et tailles d'écran utilisés par les utilisateurs, tels que les smartphones, tablettes et ordinateurs de bureau. Cette approche garantit une expérience utilisateur optimale, quelle que soit la plateforme utilisée, en ajustant la disposition, la taille des éléments et l'organisation du contenu pour une lecture et une navigation fluide. Le design responsif favorise

l'accessibilité et la convivialité en offrant une expérience cohérente et réactive, adaptée à chaque contexte d'utilisation.

EDI : Environnement de Développement Intégré. C'est un éditeur de code supplanté par un ensemble d'outils de développement comme un compilateur intégré, un éditeur de liens (permet de lier les différents fichiers entre eux), un débugger et bien d'autres fonctionnalités spécifiques à chaque EDI. On peut également dire IDE pour l'anglais Integrated Development Environment.

EPNER : Ecole du Personnel Navigant d'Essais et de Réception.

Expression régulière : Une expression régulière est un motif de recherche utilisé dans le domaine de l'informatique pour décrire et identifier des schémas de texte spécifiques, permettant ainsi d'effectuer des opérations de recherche, de validation ou de manipulation de données avec une grande flexibilité et précision.

Framework : Ensemble cohérent de composants logiciels qui sert à créer les fondations d'un logiciel (ou au moins en partie) et qui fournit une architecture. Il se distingue d'une simple bibliothèque par son caractère générique et faiblement spécialisé, et peut lui-même contenir un ensemble de bibliothèques ou modules. Un framework peut être basé sur un langage en particulier ou une plateforme spécifique. Il impose un certain cadre de travail qui force les développeurs à suivre certains modèles de conception recommandés.

Github : GitHub est une plateforme très populaire de développement collaboratif basée sur Git, permettant aux développeurs de stocker, gérer et partager leurs codes sources, de travailler en équipe et de suivre les modifications apportées aux projets.

IFTL : Innovation Flight Test Lab

IHM : Interface Homme-machine

Inclinaison: L'inclinaison d'un hélicoptère fait référence à l'angle formé par l'inclinaison transversale de l'appareil par rapport à l'horizon. L'inclinaison s'un hélicoptère varie autour de l'axe de roulis.

Maquette : Une maquette dans la conception d'une application est une représentation statique, généralement sous la forme d'une image ou d'un fichier PDF, qui illustre la disposition visuelle et l'agencement des éléments de l'interface utilisateur, fournissant ainsi une vue préliminaire de l'apparence de l'application.

Persona : Le persona en conception fonctionnelle est une représentation fictive d'un utilisateur ciblé, basée sur des recherches et des données, qui aide les concepteurs à comprendre les besoins, les attentes et les comportements des utilisateurs afin de créer des produits ou des services adaptés à leurs besoins spécifiques.

Prototype interactif : Un prototype interactif est une représentation fonctionnelle d'une application ou d'un produit numérique, qui permet aux utilisateurs d'interagir avec l'interface et de

tester les fonctionnalités de base. Il sert à recueillir des commentaires, à valider des concepts et à évaluer l'expérience utilisateur avant le développement complet de l'application ou du produit.

Refactorisation : La refactorisation de code consiste à réorganiser et à améliorer la structure interne du code source, sans en modifier le comportement externe, afin d'améliorer sa lisibilité, sa maintenabilité et sa performance.

RSE (Responsabilité Sociétale et Environnementale) : La Responsabilité Sociale des Entreprises (RSE) est un concept qui encourage les entreprises à prendre en compte les impacts de leurs activités sur la société et l'environnement, en adoptant des pratiques durables et éthiques. Cela comprend la gestion responsable des ressources, la promotion de l'équité sociale, la protection de l'environnement et l'engagement envers les parties prenantes.

Salle d'écoute : Les salles d'écoute sont des installations utilisées pour suivre et analyser les essais en vol des aéronefs, fournissant un environnement contrôlé pour les ingénieurs et les techniciens chargés de surveiller les communications, les données et les paramètres en temps réel. Elles sont équipées de systèmes de communication avancés et d'instruments de mesure pour recueillir des informations précieuses sur les performances, les systèmes et la sécurité des aéronefs pendant les essais en vol.

Sondes « Pitot » : Les sondes Pitot sont des dispositifs aérodynamiques utilisés dans l'aviation pour mesurer la pression dynamique de l'air, fournissant des informations cruciales sur la vitesse de l'aéronef.

Trello : Trello est une plateforme de gestion de projets en ligne qui permet aux individus et aux équipes de collaborer, d'organiser leurs tâches et de suivre leur progression de manière visuelle et intuitive à travers des tableaux, des listes et des cartes.

Vortex (hélicoptère) : L'effet de vortex en hélicoptère se produit lorsque l'air s'écoulant autour des pales du rotor principal commence à former des tourbillons d'air en rotation. Ces tourbillons, connus sous le nom de vortex, réduire significativement la portance des pales ce qui a un impact sur la stabilité et la performance de l'hélicoptère. Les vortex sont réputés pour créer des situations dangereuses comme la perte soudaine et brutale d'altitude par exemple.

Introduction de l'étude

Dans le cadre d'essais en vol spécifiques réalisés par la DGA EV, les aéronefs peuvent être amenés à devoir réaliser différentes manœuvres dans des conditions assez atypiques. Parmi ces dernières, le vol stationnaire en hélicoptère. Habituellement, les pilotes réussissent cette manœuvre à proximité du sol en s'appuyant sur des repères visuels extérieurs pour éventuellement corriger toutes dérives possibles. Néanmoins, certains stationnaires doivent être exécutés en l'absence totale de repères, lors de vols en haute altitude ou lors de conditions météorologiques dégradées par exemple. La plupart des hélicoptères récents disposent d'un système d'asservissement permettant de facilement éviter une dérive trop importante ce qui permet un maintien automatique de la position de consigne. Mais les aéronefs plus anciens en sont généralement dépourvus, ce qui est le cas des hélicoptères utilisés pour les vols d'essais à la DGA EV. Il a donc été demandé à l'IFTL de trouver une solution alternative simple à mettre en œuvre.

J'ai été chargé de développer une application nommée « EasyHover » (traduction littérale de « vol stationnaire simple » en anglais), destinée à assister les pilotes dans le maintien d'un vol stationnaire sans repères visuels extérieurs et fonctionnant sur leurs tablettes embarquées. De la compréhension fine du besoin à la programmation en passant par la conception fonctionnelle et le choix de la technologie, j'ai donc été amené à mener toutes les étapes d'un projet de développement d'application mobile.

Je vais dans un premier temps présenter l'entreprise et ses activités puis détailler les enjeux et le besoin à l'origine du projet EasyHover. Après une formalisation et analyse des fonctions principales et de contraintes se dégageant des différents cas d'usages de la solution, je décrirai mes conditions de travail ainsi que les protagonistes de ce projet. Je détaillerai ensuite mon approche du problème ainsi que la conception de ma solution avant de me plonger dans le détail de la réalisation. J'y expliquerai mes choix algorithmiques puis je présenterai les résultats avant de discuter de l'avenir du projet et de conclure.

Tout au long du rapport, les mots du glossaire seront suffixés d'un astérisque (*) lors de leur première apparition. Le glossaire est trié dans l'ordre alphabétique.

Travail Réalisé

I. Introduction du travail à faire

A. L'ENTREPRISE ET SON ACTIVITE

1. La Direction Générale de l'Armement

La Direction générale de l'Armement (DGA) est une direction du ministère des Armées (MinArm). Elle a été créée en 1961 par le général Charles De Gaulle dans le but de concevoir, acheter et évaluer les systèmes d'armes destinés à l'Armée française. Étant aujourd'hui une force d'expertise, d'essais et d'ingénierie au sein du ministère des Armées, la DGA a pour objectif principal de remplir les missions suivantes [1] :

- Équiper les forces armées de façon souveraine : piloter la réalisation des matériels militaires, les acquérir, les évaluer et les tester.
- Préparer le futur des systèmes de défense : toujours garder « un coup d'avance », garantir la disponibilité des technologies et des savoir-faire.
- Soutenir les exportations d'armement : l'Etat ne peut pas garantir à lui tout seul le bon remplissage du carnet de commande des industriels français sur le long terme et les exportations jouent également un rôle déterminant dans le dynamisme de notre économie.
- Promouvoir la coopération européenne : permet de partager les coûts de développement des matériels, de bénéficier d'économies d'échelle et facilite l'interopérabilité entre les armées des différents pays.
- Développer la Base Industrielle et Technique de Défense (BITD) française et européenne : plus de 200 000 emplois, hautement qualifiés et durables, sont générés en France par l'industrie de Défense.

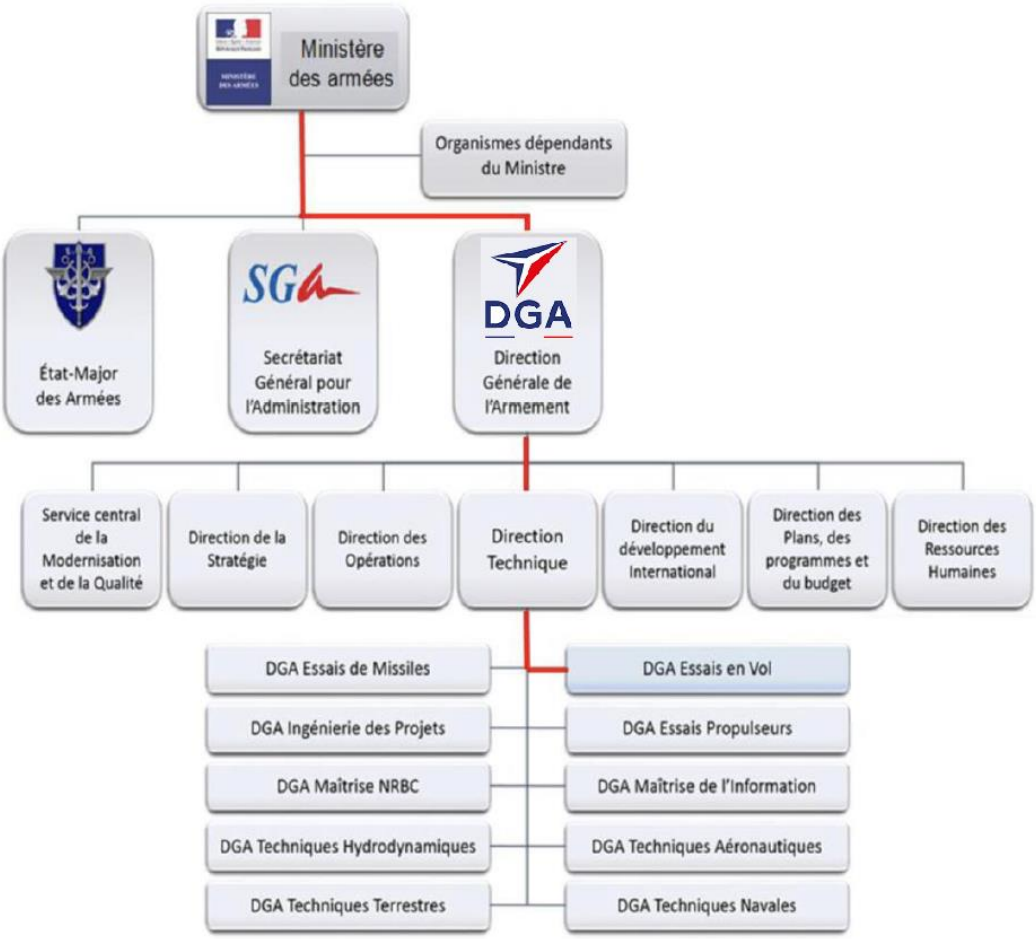


Figure 1 : Organigramme de la Direction générale de l'Armement (DGA) en 2023

La DGA intervient dans tous les domaines de la défense, que ce soit le combat terrestre, naval et aérien, les systèmes électroniques de communication et d'information, la dissuasion, l'espace, la cybersécurité ou encore la robotique. Elle finance chaque année d'importantes activités de recherche, de développement et d'acquisition de nouveaux équipements et logiciels à destination des forces armées.

La DGA est composée de plusieurs sous-divisions ayant chacune son propre domaine d'étude. Parmi ses quelques 10 000 employés civils ou militaires répartis sur 18 sites partout en France, on peut retrouver 62 % de cadres, ingénieurs ou experts.

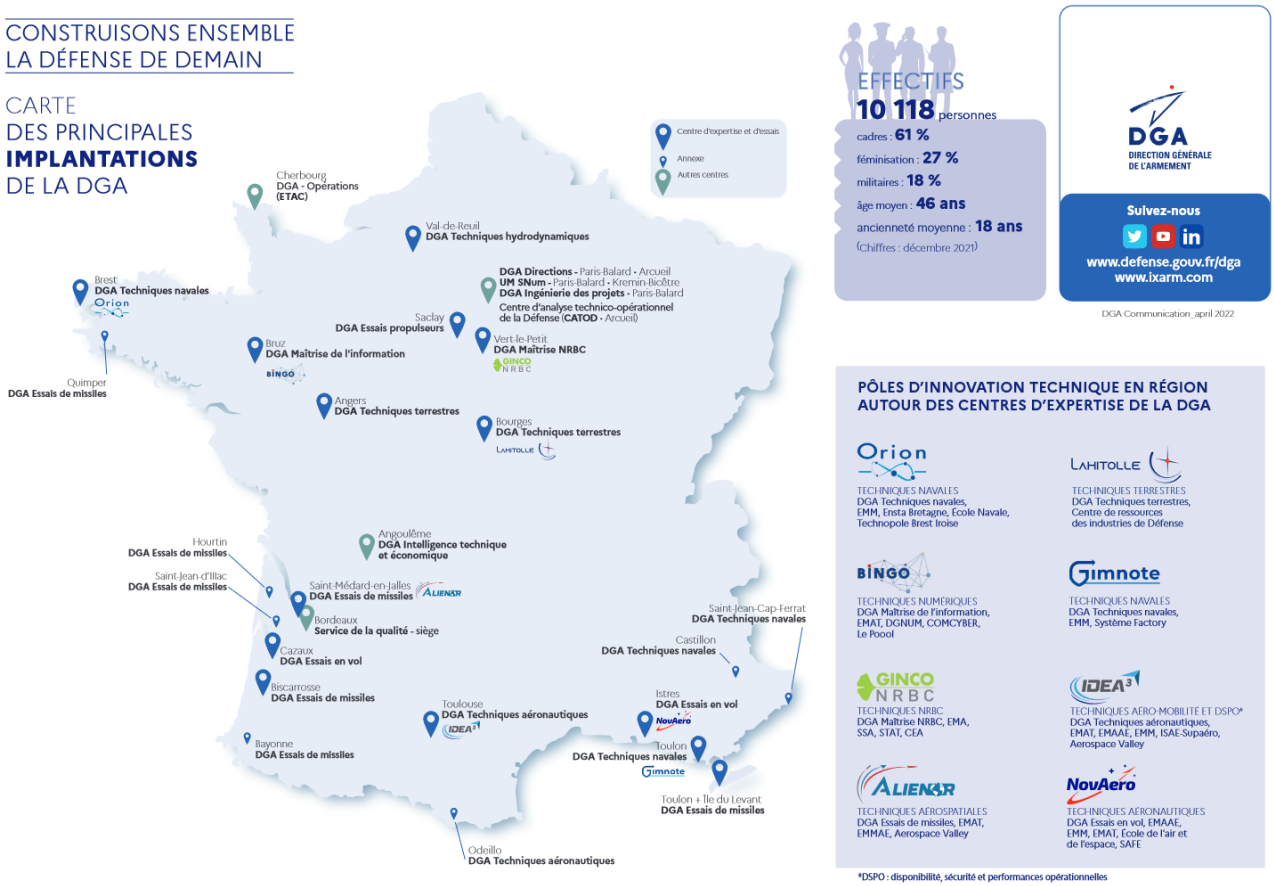


Figure 2 : Carte des principales implantations de la DGA en France en 2022 [2]

2. Les Essais en Vol

La Direction Générale de l'Armement Essais en Vol (DGA Essais en Vol ou DGA EV) est une sous-direction de la DGA implantée sur deux sites : le premier se trouve à Istres dans les Bouches-du-Rhône (13) et le second à Cazaux en Gironde (33). Ces deux sites sont complémentaires et indissociables.

La DGA EV est créée en 1944 sous la forme du Centre d'essais en Vol (CEV). Grâce à son expertise en matière de qualité et de sécurité des vols, elle représente aujourd'hui un acteur incontournable en termes de projets aéronautiques nationaux. La DGA EV remplit plusieurs missions : elle est chargée d'assurer l'expertise des aéronefs de l'Etat français (avions, hélicoptères, drones) et des équipements et armements associés, de conduire les différents essais en vols permettant de certifier ces matériels, et de former son propre personnel d'essai. Elle participe, avec le ministère des Transports et l'Agence européenne de la sécurité aérienne, aux travaux conduisant à la certification des aéronefs civils.

Pour la formation des personnels navigants, la DGA EV s'appuie sur l'EPNER* (Ecole du Personnel Navigant d'Essais et de Réception). Cette école forme les équipages d'essais destinés à l'industrie et aux forces françaises et étrangères. Au niveau mondial, il n'existe que trois autres écoles du même type, une en Grande-Bretagne et deux aux Etats-Unis.

La DGA EV de Cazaux est divisée en cinq divisions, chacune ayant une mission spécifique :

- « Moyens Aériens » (MA) : mise en œuvre et entretien des aéronefs, de leurs armements et de leurs équipements optionnels.

- « Moyens Sol » (MS) : organisation et réalisation du suivi au sol des différents essais, en mettant en place les moyens de mesure et l'exploitation, en temps réel ou différé, des résultats associés.

- « Installations Embarquées » (IE) : chargé de la conception, la validation, l'installation et la maintenance des outils de mesure et d'essais embarqués sur les aéronefs.

- « Essais Expertise » (EE) : regroupe les experts dans le domaine de la conduite d'essais. Ils sont chargés de la préparation des essais, leur organisation et supervisent l'exploitation des résultats ainsi que la réalisation des travaux d'expertise.

- « Personnel Navigant » (PN) : comprend les pilotes et les mécaniciens de bord chargés des essais en vol, des travaux d'expertise et de la formation des pilotes techniques.

Le site de Cazaux dispose de systèmes d'armement et d'équipements aéroportés. Il est spécialisé dans l'armement, les missiles et les tirs de qualification afin de vérifier leur conformité au cahier des charges. Son savoir-faire couvre aussi la sécurité des vols, le sauvetage, la guerre électronique, ainsi que la détection et l'identification des avions grâce aux radars et plus précisément l'IFF (Identification Friend or Foe) permettant l'identification de chaque avion.

Son champ d'action porte principalement sur les armements aéroportés, les équipements et l'ingénierie d'essais en vol. Cette dernière partie concerne la préparation, réalisation et exploitation des essais, mais aussi la modification et instrumentation d'aéronefs, la conception d'installations d'essais et l'informatique embarquée.

Le site de Cazaux dispose également d'un large espace aérien, maritime et terrestre lui donnant accès à différentes zones de tir : air-air, air-sol et air-surface.

Voici Figure 3 l'organigramme de la DGA EV sur le site de Cazaux, celui du site de Istres est extrêmement similaire, à quelques détails près.

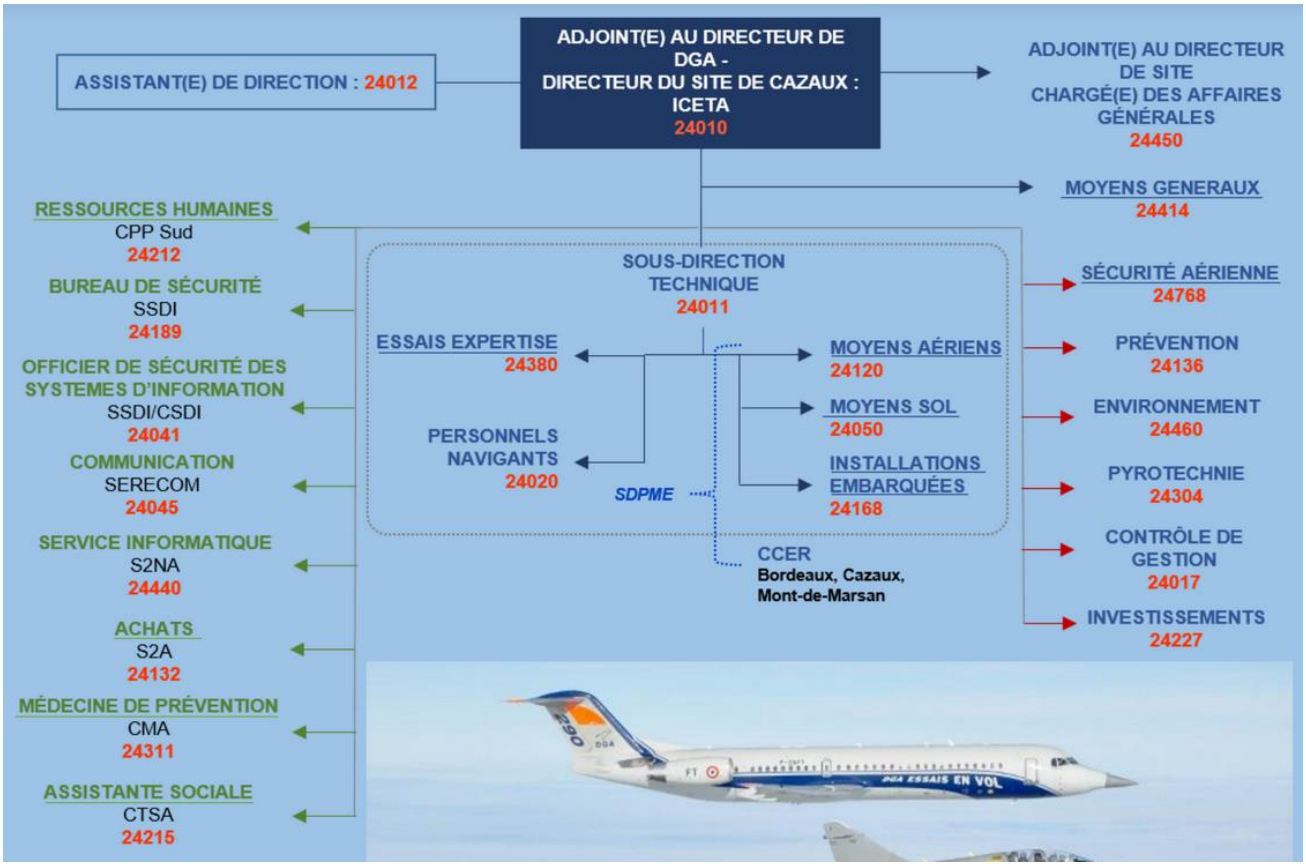


Figure 3 : Organigramme du site de Cazaux de la DGA EV (source : plaquette de bienvenue distribuée lors de l'arrivée des stagiaires)

3. L'Innovation Flight Test Lab

En 2018, Mme Florence Parly, ministre des Armées, signe officiellement la création de l'Agence Innovation Défense (AID). Cette agence, rattachée organiquement à la DGA, a pour mission de coordonner l'innovation au sein du ministère des armées. C'est dans ce contexte de structuration de l'innovation que DGA Essais en vol a créé l'IFTL en avril 2019 : l'Innovation Flight-Test Lab. L'IFTL, qui est rattachée à la sous-direction technique « Moyens Sol » (MS), est en relation avec l'AID et traite des innovations spécifiques aux essais en vol.

À l'instar de la DGA EV, répartie sur deux sites entre Cazaux et Istres, l'IFTL est divisée en deux cellules chacune composée de 2-3 collaborateurs. Malgré la distance séparant Cazaux et Istres, les deux cellules travaillent de concert et représentent un moteur d'innovation de la DGA EV. L'IFTL est chargée d'identifier et expérimenter à échelle réduite les innovations et idées pertinentes qui peuvent apporter un gain à l'activité des essais de la DGA EV et aux forces armées au sens large. Il représente également une force de changement qui accélère l'adoption de ces nouvelles technologies au sein de la DGA EV. La devise représentant l'IFTL est « Quick, Dirty but Safe » que l'on pourrait traduire par « Rapide, Sommaire mais Fiable » ce qui résume assez bien les valeurs décrites précédemment.

Quatre axes stratégiques d'innovation ont été identifiés pour guider la dynamique d'innovation dans les domaines spécifiques suivants essais en vol. Chacun des projets portés par l'IFTL est donc associé à au moins un de ces axes :

- Sécurité aérienne élargie
- Connectivité étendue
- Amélioration de la fonction actuelle
- Aviation verte

En quelques chiffres, l'IFTL dispose d'un budget de plus de 100 000 €/an et plus de 150 personnes sont impliquées dans les projets, qu'il s'agisse d'employés de la DGA EV ou de personnes extérieures. En moyenne, 5 projets durant environ 4 à 8 mois sont lancés chaque trimestre.

B. Rappel du sujet

Le but de mon stage au sein de l'IFTL est de développer une application mobile en .net MAUI* fonctionnant sur les tablettes embarquées afin d'assister les pilotes d'hélicoptères lors de la réalisation de vols stationnaires sans référence visuelle extérieure. Deux types de vols en stationnaire sont pris en compte dans ce stage :

- Priorité 1 : stationnaire sol par rapport à un point géographique, c'est-à-dire maintenir l'hélicoptère au-dessus d'un point de référence fixe au sol, à une altitude constante et sous un cap donné.
- Priorité 2 : stationnaire air par rapport à la masse d'air, c'est-à-dire maintenir l'hélicoptère fixe relativement à la masse d'air.

Le but de cette application est donc d'aider le pilote à maintenir son vol stationnaire en utilisant notamment la fonctionnalité de géolocalisation de la tablette de bord tout en restant le plus simple d'utilisation possible. Comme tout chantier aéronef est exclu pour des raisons de disponibilités des machines, la solution devra être parfaitement autonome mais adaptée à l'environnement de travail du pilote et facilement déployable.

C. Contexte et probleme existant

1. Focus technique sur le vol stationnaire en hélicoptère

a. Principales commandes d'un hélicoptère

Le pilotage d'un hélicoptère requiert un minimum de coordination, puisque l'utilisation des deux mains et des deux pieds est nécessaire, afin de contrôler les quatre systèmes principaux de commandes.

Tous les hélicoptères ont à peu près les mêmes commandes, ils sont équipés des éléments suivants que l'on retrouve sur la photo de la Figure 4 :

- Commande de pas collectif
- Poignée des gaz
- Commande de pas cyclique
- Pédales de commande de direction (palonnier)

Figure 4 : Photo du cockpit H225 de DGA EV

La commande de pas collectif, ou collectif, est située sur le côté gauche du siège du pilote, avec une commande à friction variable sélectionnée par le pilote pour empêcher tout mouvement involontaire. Ce levier se manœuvre de haut en bas et change l'angle d'incidence de toutes les pales du rotor principal « collectivement » (c'est-à-dire toutes en même temps) et indépendamment de leur position. En tirant le levier vers le haut, l'angle d'incidence de toutes les pales du rotor principal augmente ce qui entraîne une augmentation de portance et l'hélicoptère monte. En abaissant le levier, l'angle d'incidence de toutes les pales du rotor principal diminue ce qui entraîne une diminution de portance et l'hélicoptère descend.

La commande de puissance produite par le moteur est placée à l'avant de la commande du pas collectif, et se manœuvre comme la commande des gaz d'une moto. C'est en la faisant tourner que la puissance du moteur varie.

Le pilote tient la commande du pas cyclique (manche) avec la main droite. La commande cyclique fait basculer le disque rotor dans la direction où le cyclique est déplacé, que ce soit vers l'avant, vers l'arrière, ou dans un mouvement latéral, fournissant ainsi une poussée dans la direction où le rotor est basculé (voir Figure 5).

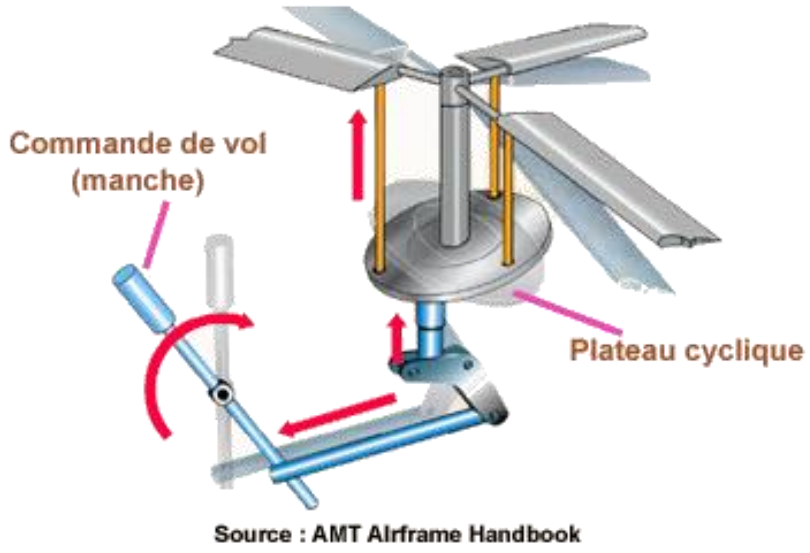


Figure 5 : Principe de fonctionnement de la commande du pas cyclique

Pour permettre le vol et compenser le couple dû au rotor principal, la plupart des hélicoptères sont équipés d'un rotor anti-couple ou d'un rotor de queue. C'est le palonnier qui contrôle la direction vers laquelle l'hélicoptère va, par l'entremise des deux pédales qui le constitue. En fait, le palonnier contrôle le rotor de queue, en modifiant l'angle d'incidence de ses pales pour contrecarrer l'effet de couple.

b. Vol stationnaire sol

Lors d'un vol en hélicoptère, le vol stationnaire géographique dit « stationnaire sol » consiste à se placer au-dessus d'un point fixe au sol et à compenser les différentes dérives que pourraient subir l'aéronef afin de maintenir sa position pendant un certain temps. La position de consigne à maintenir est nommée « le point de référence ». Cette position de consigne est éventuellement accompagnée d'une altitude et d'un cap* de consigne.

Pour assurer le maintien de l'hélicoptère lors d'un stationnaire sol, il faut contrôler en permanence les références extérieures : inclinaison*, assiette* (attitude de l'hélico) et cadence*, tout en maintenant une hauteur constante au-dessus du sol. À chaque variation des références extérieures ou de hauteur, il faut immédiatement corriger en sens contraire sur la commande correspondante.

En vol stationnaire sol, le déplacement de l'hélicoptère provient des variations d'inclinaison et d'assiette. Toutefois, ce déplacement n'est pas immédiat : il se produit d'abord la variation des références extérieures puis le déplacement. Il y a donc un temps de réponse des commandes qui est de l'ordre d'une seconde suivant le type d'hélicoptère. L'objectif pour le pilote étant de limiter au maximum les déplacements, tout en rattrapant ceux déjà subis. Le pilote constate ces déplacements en observant le défilement du sol à travers la verrière du cockpit.

En règle générale, les pilotes peuvent maintenir l'hélicoptère dans un rayon de 10 m voire descendre au niveau du mètre pendant des courtes périodes pour des missions demandant une très grande précision (ex : treuillage).

Vous trouverez en Annexes (Annexe 1) un protocole, détaillant les actions à effectuer sur les commandes d'un hélicoptère pour assurer le maintien d'un vol stationnaire sol.

c. Vol stationnaire air

Ce type de stationnaire équivaut à faire correspondre la vitesse de l'hélicoptère avec la direction et la vitesse de déplacement de la masse d'air (= « le vent ») afin de faire tendre la vitesse relative entre l'aéronef et le vent vers zéro.

Les hélicoptères sont équipés de sondes « Pitot » * permettant de mesurer cette vitesse relative. Le pilote doit donc effectuer des actions sur les commandes similaires à celles nécessaires au maintien de l'hélicoptère lors d'un stationnaire sol mais, cette fois, en surveillant les mesures de la sonde Pitot plutôt que le défilement du sol.

2. Problèmes existants

Nous avons vu que la réussite d'un vol stationnaire sol repose essentiellement sur la visualisation de repères extérieurs fixes permettant au pilote de correctement estimer les différentes variations provoquant des dérives. La difficulté apparaît donc lors de l'altération ou encore de la perte de ces fameux repères visuels. Cela arrive notamment en cas de vol en stationnaire à haute altitude ou encore dans des conditions météorologiques engendrant une visibilité dégradée : les repères visuels ne sont plus exploitables par le pilote.

Citons quelques scénarios faisant apparaître ce problème :

- Dans le cadre d'un essai en vol à la DGA EV, un dispositif a été suspendu à un hélicoptère par un câble de 200 m. L'aéronef a donc dû se placer en vol stationnaire au-dessus d'un point précis à une altitude d'environ 400 m, hauteur où les repères visuels fixes se font rares. L'exécution de la manœuvre a été délicate.

- Lors de la formation des pilotes d'hélicoptère à l'EPNER, un exercice consiste à recevoir des coordonnées géographiques, chacune accompagnée d'une altitude et d'un cap de consigne. Les apprentis pilotes ont donc pour mission de se rendre à une position, la tenir pendant un certain temps puis passer à la prochaine etc. Un dispositif leur permettant de gérer les différentes positions de consigne et de se chronométrer aurait été bien utile.

- Certaines manœuvres à proximité du sol peuvent provoquer ce qu'on appelle un « whiteout ». Cela peut arriver lors de missions de sauvetage en haute montagne où, à l'approche d'une surface enneigée, le souffle des pales a pour effet de soulever un épais nuage de neige. Ce nuage va « aveugler » le pilote en occultant considérablement sa visibilité sur l'extérieur. L'expression « whiteout » reprend le terme « blackout » qui fait référence à une panne de courant généralisée dans un quartier pendant la nuit, entraînant l'interruption de tout éclairage, ce qui a pour effet d'aveugler momentanément les personnes.

- Dans certaines régions très arides du globe comme le Sahel par exemple, les pilotes d'hélicoptères peuvent subir des « brownout ». Ce phénomène est extrêmement similaire au « whiteout » mais cette fois avec de la poussière ou du sable.

Concernant une aide pour les vols stationnaires air, le besoin provient essentiellement de l'EPNER afin de mettre en pratique les formations dispensées sur la mécanique du vol et autres cours théoriques sur l'hélicoptère.

3. Solutions actuelles

Tous ces problèmes ne s'appliquent pas vraiment aux modèles récents d'hélicoptère, d'ores et déjà équipés des accessoires nécessaires (coupleur de vol stationnaire) permettant d'automatiser la tenue du stationnaire sol. Pour les appareils un peu plus anciens en revanche, tout repose sur les talents du pilote et ce genre de situation arrive donc régulièrement à la DGA EV, qui utilise encore d'anciennes machines.

Des opérations de modernisation sont toutefois envisageables : la tenue du stationnaire sol se réaliserait par une veille permanente des instruments de bord conjugués à une installation d'essai basée sur un capteur GNSS*.

De même pour ce qui est du stationnaire air : les machines les plus récentes se voient équipées à la conception mais les plus anciennes nécessitent une installation d'essais appelée VIMI.

Ces installations d'essai seraient déployées sur demande et nécessiteraient une charge de travail pour les équipes d'intégration ainsi qu'une immobilisation de la machine (coûts importants). Il serait donc préférable de pouvoir se passer de tels chantiers aéronefs.

4. Le besoin d'une application et ses enjeux

Lors des vols d'essais à la DGA EV, les pilotes d'hélicoptère sont équipés d'un IPad fixé sur la cuisse. L'objectif poursuivi étant de développer un système alternatif à l'installation d'essai et qui soit facilement utilisable sans requérir un chantier aéronef, une application fonctionnant sur cet IPad pourrait représenter une solution tout à fait envisageable. En effet, l'application exploiterait les fonctionnalités de géolocalisation de l'IPad, tout en proposant une IHM ergonomique et intuitive à moindre coût.

Durant un vol d'essai en hélicoptère, le pilote est fortement sollicité : micro-ajustements à faire en permanence sur les commandes, contrôles incessants des différents instruments de navigation et indicateurs de bord ou encore entretien du contact radio avec les opérateurs de la salle d'écoute*

etc. Il est donc primordial de réduire le nombre d'actions à effectuer et d'épargner un maximum de distractions au pilote. L'objectif est de réduire les risques d'une surcharge cognitive qui pourrait conduire à des négligences ou, dans le pire des cas, à des accidents…

C'est ainsi que l'idée de l'application EasyHover est née. « Hover » signifie « survoler, voltiger » et fait donc référence à la notion de vol stationnaire, « Hover flight » en anglais. « Easy » signifie « facile, simple, sans effort » et fait référence à l'ergonomie et l'intuitivité de l'application, ainsi qu'à sa mise en place immédiate qui nécessite aucun dispositif préalable (chantier aéronef, formation du pilote). Les pilotes pourront ainsi, quel que soit le type de stationnaire souhaité et avec une saisie simplifiée des données nécessaires, profiter d'une assistance efficace tout en économisant de la charge mentale.

5. Des premières initiatives sur le sujet

L'idée de réaliser une application sur tablette pour aider les pilotes dans le maintien en stationnaire n'est pas nouvelle.

Un ancien collaborateur de la DGA EV avait déjà trouvé un moyen de répondre au besoin en développant, en 2021, un système qui tournait hors ligne sur un smartphone embarqué faisant office de serveur. Toutes les tablettes embarquées pouvaient se connecter en wifi sur ce smartphone, afin d'afficher sur une page web l'IHM, quel que soit le système d'exploitation de la tablette. Le problème de cette solution était qu'il fallait embarquer, en plus des tablettes servant d'IHM, un smartphone entièrement dédié à cette fonctionnalité. Or, dans le contexte d'essais en vol, tout objet embarqué représente de nombreuses contraintes supplémentaires : il faut chercher à réduire leur nombre au maximum. L'idéal aurait donc été d'avoir une appli autonome sur chaque tablette et d'ainsi supprimer le smartphone-serveur de l'équation.

C'est pour cela que l'IFTL s'est chargé de reprendre ce projet en le confiant à un stagiaire. C'est ainsi en juillet 2022 que Nathan Desnos, étudiant ISIMA de la promo 2023, ayant fini son premier sujet en avance, reprend le projet pendant les deux derniers mois de son stage de ZZ2. Le projet d'application n'étant pas achevé, je le continue à mon tour en tant que stagiaire, un an plus tard.

J'avais donc une belle base de travail à ma disposition en termes de spécification du besoin, conception fonctionnelle et autres étapes initiales de gestion de projet.

D. Analyse du Probleme

1. Formalisation du besoin et du cahier des charges

Voici une liste des principaux éléments du cahier des charges :

a. Livrables

- Fourniture des codes sources de l'application et de la documentation descriptive associée. Ce projet sera hébergé sur le GitHub DGAEV-IFTL.

- Dépôt de l'application sur l'App Store et le Play Store afin de la mettre à disposition gratuitement pour tout pilote rencontrant le même problème. L'application doit donc être compatible avec les environnements iOS et Android.

b. Pages et maquette

- L'interface utilisateur de l'application est en anglais.
- La page d'accueil est la première page visualisée par l'utilisateur. Sur cette page doit figurer le nom/logo de l'application, le logo du représentant étatique et éventuellement celui du titulaire.
- La page de travail prend la forme d'un graphique affichant une représentation de l'hélicoptère au centre. Ce graphique représente en fait l'hélicoptère de l'utilisateur et son environnement direct vu du dessus. En fonction des modes et des cas d'utilisation, le graphique est orienté selon le nord vrai (« North-up ») ou alors selon le cap vrai* (« Heads-up »). Différentes informations utiles à la manœuvre sont affichées.
- La page de tutoriel doit être accessible à tout moment et détaille toutes les informations à savoir pour avoir une bonne utilisation de l'application. On y découvre ses fonctionnalités ainsi que les différentes manières de l'utiliser et d'interagir avec.
- Une page de configuration doit être accessible à tout moment pour pouvoir configurer l'IHM* (unités de mesure, échelle, options d'affichage…)

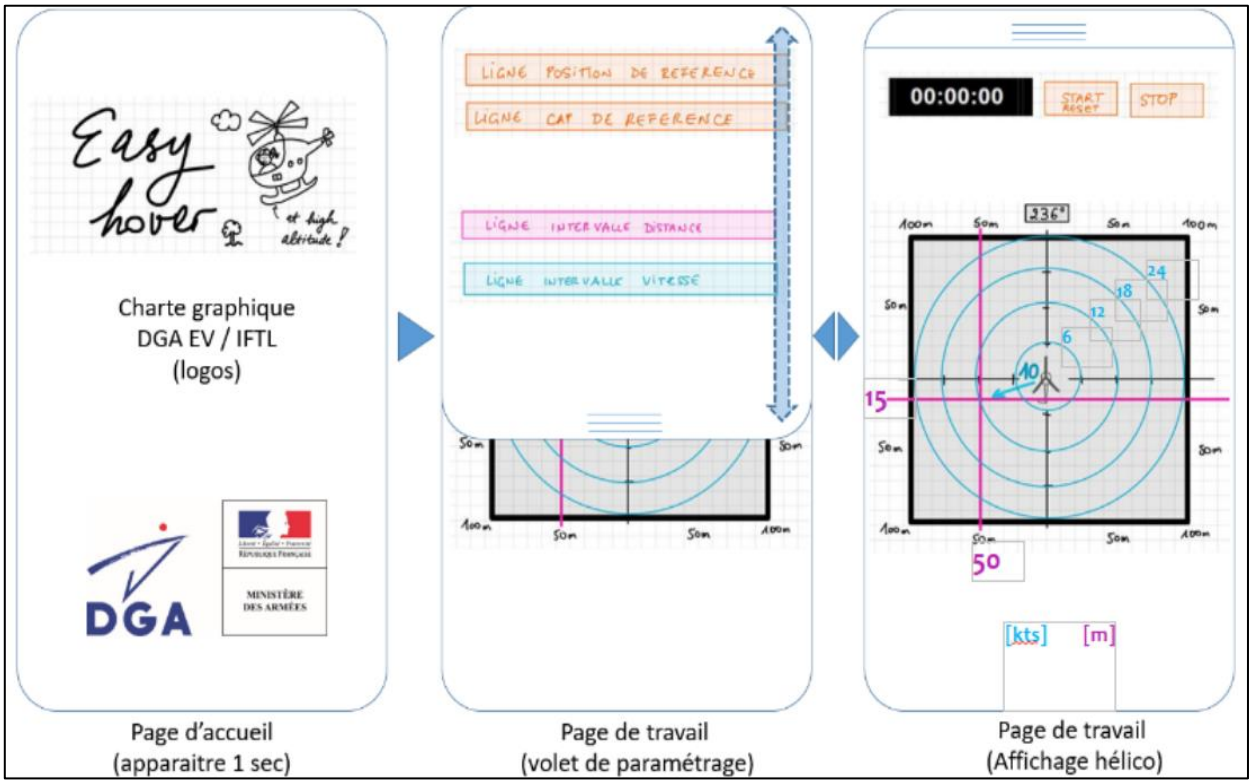


Figure 6 : Première proposition de maquette pour EasyHover

La maquette Figure 6 a été réalisée en 2021 en s'inspirant d'une IHM de cadrant aéronautique déjà existante.

c. Deux modes d'utilisation

- Le vol stationnaire par rapport à un point fixe géographique (= « stationnaire géographique » = « stationnaire sol » = « compenser les différentes dérives pour se maintenir au-dessus d'un point fixe au sol »)

- Le vol stationnaire par rapport à la masse d'air (= « stationnaire vent » = « stationnaire air » = « se laisser porter par le vent » = « avoir une vitesse relative au vent nulle »).

Pour chacun de ces deux modes, le pilote doit se charger de communiquer les informations (consignes) nécessaires au fonctionnement de l'application. Il peut à tout moment passer d'un mode à l'autre.

2. Les points durs du cahier des charges

a. Des conditions d'utilisation particulièrement difficiles

Cette application est destinée à être utilisée en vol, sur une tablette posée sur la cuisse d'un pilote d'hélicoptère : il faut donc considérer le grand nombre de contraintes que cela représente. Ce n'est qu'après une analyse en profondeur du contexte et des conditions d'utilisation que j'ai pu commencer à adapter la conception de l'application à tous niveaux : les fonctionnalités et l'ergonomie de l'IHM (UX design), l'intuitivité du graphique, la limitation de la charge mentale requise etc. Contrairement à d'autres applications à l'ergonomie moins exigeante et à l'affichage plus allégé, comme un service de messagerie par exemple, rien ne doit être laissé au hasard. Que ce soit au niveau du choix et de la disposition des éléments ou encore au niveau des différentes possibilités d'interactions offertes par chacun d'entre eux, tout doit être soigneusement choisi et justifié.

b. Une application tout-terrain

En plus d'être multiplateforme, l'application doit pouvoir être prise en charge par tous les types de smartphones et de tablettes et donc par tous les formats. Grossièrement, on peut considérer que ces formats vont du 4:3 (l'écran est presque carré) au 21:9 (l'appareil est plus de deux fois plus long que large). L'affichage doit être optimisé quel que soit le format, sans oublier les différentes tailles d'écran pouvant aller de 4 pouces à 13 pouces. L'application doit en revanche privilégier l'affichage en orientation portrait.

Pour atteindre le niveau de détail décrit précédemment pour chacun des appareils disponibles sur le marché, il faudra reconsidérer et réadapter la conception UX/UI* et raisonner par disjonction de cas. L'objectif final étant de rendre l'affichage et son design totalement responsifs.

Algorithmiquement, cela implique une analyse en temps réel des propriétés de l'écran afin d'identifier la disposition adaptée prévue en amont dans la disjonction de cas. Il « suffira » ensuite de sélectionner et d'exécuter le code correspondant pour obtenir le bon affichage.

c. Acquisition et traitement de données

L'application ne pouvant pas compter sur l'exploitation de données provenant des capteurs de l'aéronef, elle dispose uniquement des capteurs de la tablette embarquée (ou capteur GNSS porté par les pilotes et connecté en Bluetooth). L'idéal serait d'acquérir en permanence une géolocalisation et une orientation par rapport au nord vrai. La tablette a pour cela les capteurs nécessaires et pourrait ainsi permettre un affichage semblable à un GPS de voiture.

On considérera cependant la boussole inutilisable à cause d'une trop grande incertitude engendrée par la masse métallique constituant l'hélicoptère ainsi que par les interférences émises par les autres instruments de bord. L'exploitation de la boussole de la tablette est donc mise de côté dans un premier temps, tout en prenant soin de garder cette fonctionnalité facilement implémentable pour faciliter un potentiel changement dans le futur. La géolocalisation reste en revanche exploitable mais sa qualité est néanmoins variable en fonction du modèle et de la marque de la tablette. Le titulaire effectuera une analyse de ces variations de qualité ainsi que de l'impact qu'ont ces variations sur le fonctionnement de l'application et en tirera des recommandations.

II. Matériel, méthode, outils et objectifs

A. Revision des specifications

1. .NET : de Xamarin.Forms à MAUI

L'application devant être à la fois compatible avec Android et IOS, le développement de EasyHover a été réalisé avec le Framework Xamarin jusqu'à ce que je reprenne le projet. Il s'agit d'une plateforme open-source basée sur l'environnement .NET permettant de réaliser des applications multiplateformes en un seul langage de programmation, le C#. La majorité du code peut être écrit de façon indépendante à la plateforme sur laquelle il sera exécuté, et les fonctionnalités propres à chaque système d'exploitation sont accessibles avec des API relativement simples d'utilisation. Cela inclut l'interface utilisateur de l'application, qui est ensuite adaptée selon le support d'exécution.

Au niveau du code en lui-même, le Xamarin permet de réaliser des interfaces graphiques de 2 manières différentes. Le C# étant un langage orienté objet, il est possible de créer et de manipuler tous les éléments graphiques sous forme d'objets directement dans un fichier de code C#. La deuxième option, plus courante, revient à utiliser le langage XAML, un langage de balise déclaratif qui permet de facilement développer les interfaces utilisateurs pour les applications .NET. Le XAML permet de facilement séparer la définition des objets de l'interface de leur comportement, implémenté directement dans le code. Les fichiers XAML sont ainsi toujours liés à un fichier de code (ici écrit en C#).

Le Xamarin devenant aujourd'hui progressivement obsolète, j'ai préféré faire basculer le développement de EasyHover vers son récent successeur : .NET MAUI (Multi-platform App UI). Ce framework pousse le caractère multiplateforme du Xamarin encore plus loin en permettant un développement unique pour des applications fonctionnant sur toutes sortes d'appareils mobile (smartphone, tablette, Android, iOS) ET toutes sortes d'ordinateurs (Windows, Linux, MacOS).

.NET MAUI est intimement lié à la plateforme .NET 6 qui introduit plusieurs améliorations de performances significatives par rapport aux versions précédentes. Cela inclut des améliorations du temps de démarrage des applications, de l'utilisation de la mémoire, de la latence réseau, et de l'accès aux données. Le .NET MAUI étant donc une solution d'avenir, c'est ce qui m'a motivé à effectuer cette migration. N'étant pas plus à l'aise sur ce framework que sur le Xamarin, il faudra tout de même que je me réserve un temps dédié à la formation dans mon planning.

Si vous souhaitez plus d'informations sur ce framework, voici l'Url du site officiel de sa documentation et de celle de la plateforme .NET en général : https://learn.microsoft.com/fr-fr/dotnet/maui/what-is-maui

2. Android et/ou iOS ?

Le .NET MAUI permet certes le développement d'applications totalement multiplateforme, mais les applications iOS nécessitent d'être développées à partir d'une machine MacOS. Il existe cependant plusieurs solutions alternatives mais chacune d'entre elles nécessite soit des investissements de licence ou de matériel, soit des manipulations informatiques très poussées tendant vers le piratage.

On y reconnaît clairement la politique de l'entreprise Apple car le développement d'applications Android à partir d'une machine MacOS ne pose aucun problème, comme par hasard…

Il a donc été décidé de laisser pour l'instant de côté le développement iOS et de se concentrer sur le développement Android. Une fois l'application achevée, testée au sol, testée en vol et mise à disposition sur le Play Store, les pilotes devront confirmer que l'application est vraiment utile et efficace. C'est seulement à ce moment qu'une décision de budgétisation sera prise, en vue de se procurer le matériel nécessaire afin de réaliser le portage vers iOS et le dépôt sur App Store.

3. Objectifs pour ce stage

Voici, dans l'ordre chronologique, la liste des tâches à réaliser afin d'atteindre mes objectifs :

- Conception de l'application
    - Analyses préliminaires diverses et variées
    - Diagrammes décrivant le fonctionnement (navigation, boucle événementielle etc.)
    - Création de maquettes* des différents écrans prévus
    - Création d'un prototype interactif* complet pour valider le fonctionnement de l'application.
- Formation .NET MAUI
    - Suivre des tutoriels Youtube
    - Se documenter sur le site de Microsoft Learn
    - Télécharger des exemples de petits projets open-source pédagogiques et en observer, décortiquer et manipuler le code afin de gagner en aisance sur le langage.
    - Explorer au maximum les possibilités offertes par le framework afin d'y dénicher les outils nécessaires au développement des fonctionnalités prévues de l'application (exemple : faire en sorte que le graphique prenne la forme du plus gros carré possible en toutes circonstances est loin d'être aussi aisé que ce que l'on pourrait penser)
- Développement de l'application
    - Initialiser le dépôt Github* afin d'assurer un bonne gestion des versions
    - Mettre en place l'architecture de navigation
    - Création d'un graphique intuitif, fluide et robuste permettant de guider efficacement le pilote quelle que soit la situation
    - Rendre l'affichage de la page de travail configurable depuis la page de configuration (unités de distance, conventions d'écriture etc.)
    - Créer un gestionnaire d'événements permettant de détecter les différentes interactions de l'utilisateur et de comprendre l'action qu'il souhaiterait effectuer
    - Création des fonctions permettant géolocalisation de l'appareil

        - Récupération des données des capteurs GPS de l'appareil
        - Traitement et exploitation de ces données dans le graphique
        - Boucle permettant de réitérer ce protocole jusqu'à ce que l'arrêt soit demandé.
    - Mettre à disposition de l'utilisateur des outils lui permettant de donner des consignes à l'application (clavier tactile pour entrer les coordonnées d'un point de consigne par exemple).
    - Veiller à ce que l'implémentation de l'application respecte le design UX/UI déterminé lors de la phase de conception
    - Garantir le caractère responsif de l'application
- Essais et correctifs
    - Débogage de l'application
    - Effectuer des Essais au sol et apporter des corrections au code jusqu'à entière satisfaction
        - Essais fonctionnels, en imaginant que l'on se trouve dans un cockpit d'hélicoptère et en inventant différents scénarios
        - Essais avec différents types d'appareils, de différents formats pour tester la compatibilité et la responsivité de l'application
    - De même mais cette fois lors d'Essais en vol avec les pilotes
- Livraison sur le Play Store
    - Dernières petites retouches
    - Démarches pour l'ajout au catalogue du Play Store
    - Derniers tests pour déterminer si l'application se télécharge bien, si elle fonctionne bien etc.

En parallèle, pour les fonctionnalités offertes par les différentes pages, il ne faut pas oublier l'ordre de priorités suivant :

1. La page de travail doit offrir une assistance pour un vol stationnaire sol.
2. La page de configuration doit pouvoir être exploitable.
3. S'il n'y a pas trop de retard sur le planning, la page de travail doit pouvoir offrir une assistance pour un vol stationnaire air.
4. En cas d'avance sur le planning, la page de tutoriel doit être disponible. A défaut, on se contentera d'un fichier rédigé contenant la documentation nécessaire.

B. Équipe et gestion de projet

1. Conditions de travail et protagonistes

Mon poste de travail a été installé au sein des bureaux de l'IFTL, en compagnie de 3 autres stagiaires. L'ordinateur sur lequel j'ai mené l'intégralité du projet m'a été fourni mais j'ai dû me charger de l'installation des différents logiciels dont j'avais besoin. Les autres stagiaires et moi n'ayant pas les droits d'accès à l'intranet de la DGA EV et donc à la connexion Ethernet, nous avons dû faire des partages de connexion depuis nos téléphones portables. L'accès à Internet était donc restreint et peu performant.

Bien que travaillant seul sur le projet, je me suis référé tout au long du stage à mon maître de stage Thibault, co-fondateur et manager de l'IFTL. Nous faisions des points réguliers d'avancement et il a toujours été là pour répondre à mes questions, diverses et variées.

Sur le site de DGA EV se trouvent également les bureaux de toute l'équipe de pilotes chargés de piloter les hélicoptères utilisés pour les essais en vol. Le besoin pour l'application EasyHover émanant notamment de cette équipe de pilotes, l'un d'entre eux est considéré comme le client du projet : il s'appelle Marc. Nous avons donc régulièrement eu des petites entrevues lors desquelles je lui présentais l'avancement du projet, lui soumettait mes propositions ou encore l'on discutait de certains détails du besoin. Marc et son équipe ont toujours été présents pour répondre à mes questions et ils ont également été force de proposition pour certaines fonctionnalités.

2. Méthode de gestion de projet

Tout au long du projet, je me suis efforcé à tenir un tableau Trello* à jour. Ce tableau m'a permis d'organiser mon travail, de lister les tâches restantes et de leur associer une deadline.

Etant au courant que j'allais utiliser certaines technologies que je ne connaissais pas encore, j'ai réservé dans mon planning prévisionnel un temps pour me former.

Lors de la phase de conception, j'ai produit deux versions différentes de document de conception, maquette et prototype interactif. J'ai d'abord présenté la première version à Marc et Thibault. Leurs différentes remarques et corrections m'ont permises de faire aboutir une 2ème version, plus évoluée et détaillée que la première. Après une inspection de cette dernière version, j'ai procédé aux petites corrections qui en sont ressorties puis nous l'avons jugée satisfaisante. C'est alors que je suis vite passé à l'implémentation de l'application car un mois venait déjà de s'écouler depuis le début du stage. Dans la suite du rapport, je vais uniquement évoquer cette 2ème et dernière version de la conception.

3. Planning prévisionnel et contraintes

Mon stage ayant commencé le Lundi 3 Avril et finissant le soir du Jeudi 27 Juillet 2023, il aura en tout duré exactement 84 jours. En enlevant à cela les différents jours de congé et de RTT, cette somme tombe pour ma part à 71. Du fait de mon départ à l'étranger pour l'année scolaire prochaine, je me suis vu attribuer une date de soutenance de stage anticipée : le Jeudi 13 Juillet 2023. La date de remise du présent rapport de stage est donc fixée au Lundi 03 Juillet. Mon maître de stage

m'ayant permis de rédiger mon rapport de stage et de préparer ma soutenance sur les heures de travail à l'entreprise, je décidé de m'y suis consacrer entièrement à partir du Mercredi 21 Mai (48ème jour de travail). Après avoir passé ma soutenance, il me restera exactement 9 jours de travail avant de finir mon stage. J'aurai donc passé, en tout et pour tout 56 journées de 7h sur la conception et le développement de l'application EasyHover.

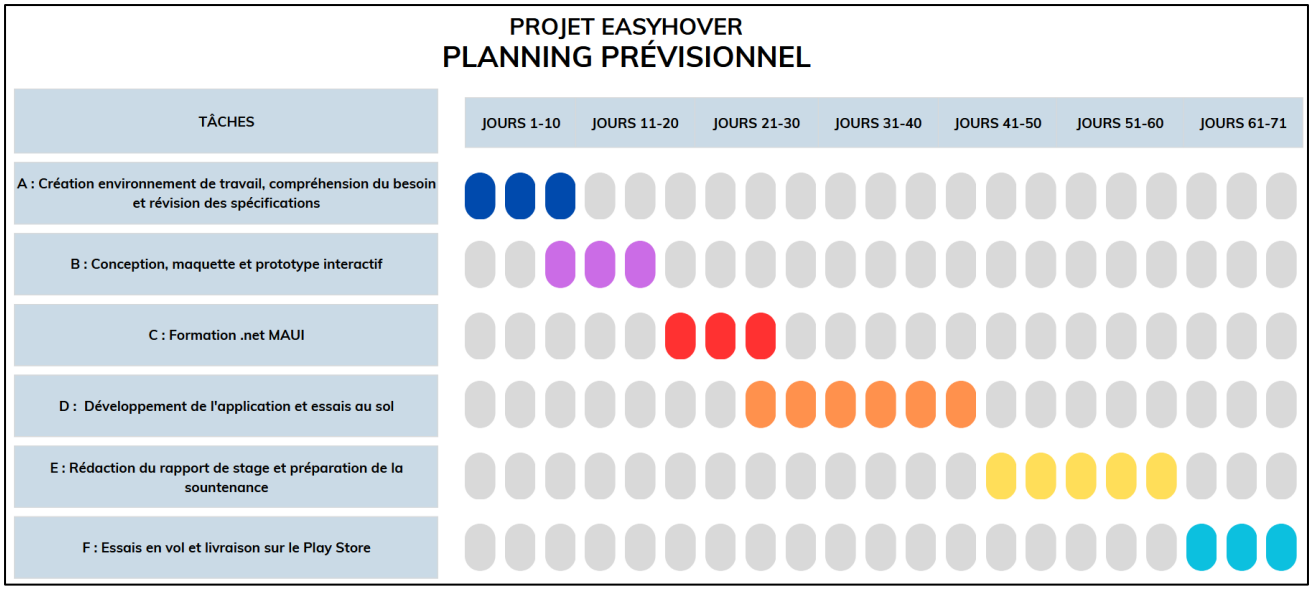


Figure 7 : Diagramme de Gantt prévisionnel du projet EasyHover

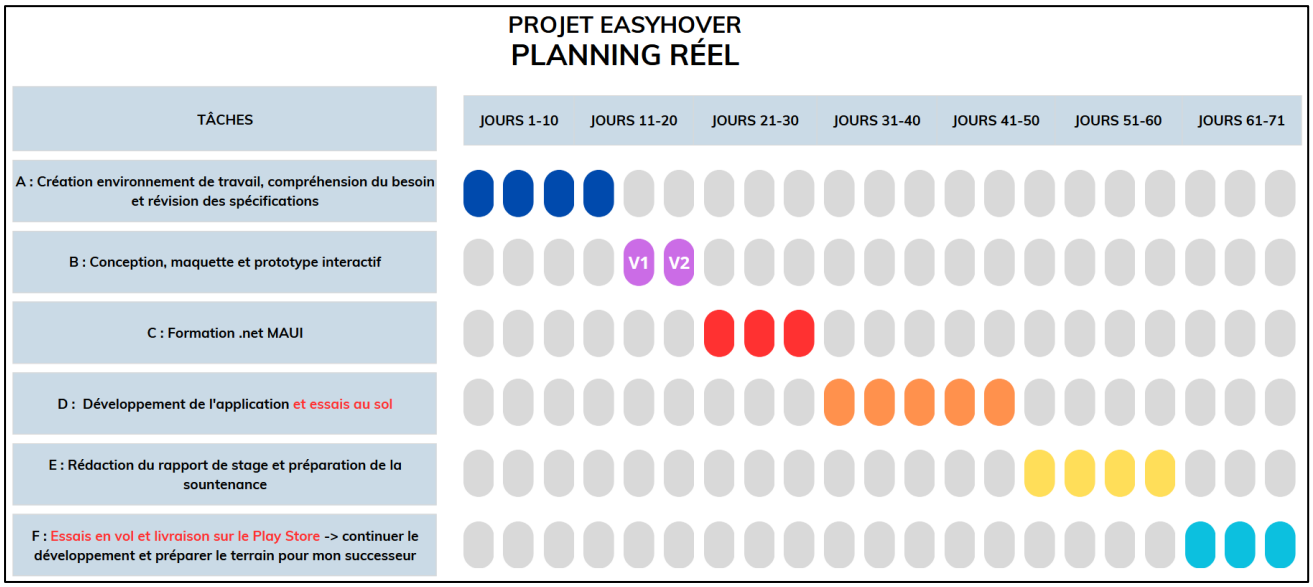


Figure 8 : Diagramme de Gantt réel du projet EasyHover

C. OUTILS UTILISES

1. Trello

Trello est un outil de gestion de projet en ligne permettant d'organiser des tâches, des projets ou encore des idées de manière visuelle et collaborative. Il est conçu pour aider à rester organisé et à collaborer efficacement avec une équipe.

La principale caractéristique de Trello est son tableau de bord flexible composé de listes et de cartes. On peut y créer des listes pour représenter les différentes étapes d'un projet, puis ajouter des cartes à chaque liste pour représenter des tâches ou des éléments spécifiques. Les cartes peuvent être déplacées d'une liste à l'autre pour refléter leur progression.

Trello facilite la coordination des activités et la communication entre les membres d'une même équipe puisqu'on peut attribuer des cartes à des personnes spécifiques, ajouter des commentaires, joindre des fichiers et définir des dates d'échéance pour les tâches.

Que ce soit pour planifier des projets professionnels, organiser des événements, gérer des listes de tâches personnelles ou collaborer avec des équipes, Trello permet de visualiser et de suivre facilement l'avancée du travail, ce qui aide à rester organisé et à coordonner les activités de manière efficace.

2. Figma

Figma est un outil gratuit en ligne qui permet aux designers de créer des maquettes et des prototypes interactifs pour la conception de sites web et d'applications mobiles. C'est comme un logiciel de dessin qui permet de montrer comment un site ou une application pourrait fonctionner avant même de commencer le développement. La réalisation d'une maquette est une étape essentielle dans la conception d'une application mobile, afin d'avoir une idée claire et un modèle sur lequel s'appuyer au moment de l'implémentation.

Avec Figma, il est possible de créer des maquettes d'interfaces utilisateur en y ajouter des boutons, des liens ou encore des animations. A cela s'ajoute des fonctionnalités de prototypage permettant de simuler l'expérience utilisateur en ajoutant des transitions entre les différentes pages ou écrans de votre projet. Par exemple, on peut créer un bouton qui, lorsqu'il est cliqué, montre une autre page ou affiche une animation. Cela permet de visualiser comment l'interface réagira lorsque les utilisateurs interagissent avec elle.

L'avantage du prototypage dans Figma est que cela permet de tester et valider des idées de conception avant de les mettre en œuvre. Il est possible de partager les prototypes avec des clients ou des membres de l'équipe pour obtenir des commentaires et apporter des modifications si nécessaire. C'est un outil utile pour les personnes qui souhaitent créer et partager des prototypes sans avoir besoin de compétences techniques avancées.

3. Visual Studio 2022 Community

Visual Studio 2022 Community est un environnement de développement intégré (EDI) populaire et puissant, développé par Microsoft. Il s'agit d'une version gratuite de Visual Studio, conçue spécifiquement pour les développeurs individuels, les étudiants, les petites équipes de développement et les projets open source.

Cet EDI offre une large gamme de fonctionnalités et d'outils pour faciliter le développement de logiciels. Il prend en charge de nombreux langages de programmation, notamment C#, C++, F#, Visual Basic et Python. On peut y développer des applications pour diverses plateformes, telles que Windows, Android, iOS, le cloud et le web.

Que ce logiciel soit utilisé par un développeur individuel ou une petite équipe de développement, Visual Studio 2022 Community permet de créer des applications de haute qualité de manière efficace.

4. Mon téléphone

J'ai utilisé mon téléphone lors du développement afin d'effectuer des tests de l'application. C'est une fonctionnalité proposée par Visual Studio, il suffit pour cela de connecter mon appareil à l'ordinateur via un câble USB. Cela m'a permis de vérifier le bon fonctionnement de chacune des fonctionnalités sur un périphérique réel.

5. Git & GitHub

Git est un logiciel libre de gestion de versions qui est réputé dans le milieu de l'informatique. Il a été créé en 2005 par Linus Torvalds et demeure libre et gratuit. Il permet de suivre les modifications apportées à des fichiers et des dossiers au fil du temps et il est utilisé principalement par les développeurs de logiciels pour gérer le code source de leurs projets. En d'autres termes, Git permet de garder une trace de l'évolution d'un projet, des modifications (résolution de bugs, ajout de fonctionnalités) apportées par différentes personnes et de revenir en arrière si nécessaire.

GitHub, quant à lui, est une plateforme en ligne basée sur Git. Elle offre un espace de stockage pour les projets Git, ainsi que des fonctionnalités de collaboration. GitHub permet aux développeurs de télécharger leurs projets Git sur la plateforme, de travailler dessus individuellement ou en équipe, de partager leur code avec d'autres personnes et de gérer les contributions des autres développeurs.

6. Synthèse sur les outils utilisés

En résumé, j'ai utilisé Visual Studio 2022 Community comme environnement de développement, Trello pour la gestion de projet, Git & GitHub pour le contrôle de version et le stockage en ligne du code, ainsi que Figma pour la conception d'interfaces utilisateur. Ces outils m'ont permis de m'accompagner tout au long du projet et développer efficacement.

III. Réalisation du projet : Conception

A. TRAITEMENT DES DONNEES DE GEOLOCALISATION

1. Coordonnées géographiques

L'application EasyHover exploite les données envoyées par la fonctionnalité de géolocalisation de la tablette sur laquelle elle fonctionne. Ces données de géolocalisation sont délivrées sous la forme de coordonnées géographiques. Par coordonnées géographiques d'un lieu sur la Terre, on entend un système de deux coordonnées qui sont le plus souvent (latitude, longitude). Ce système découle du système géodésique WGS84 qui modélise la forme de la surface de la Terre.

Comme représenté sur la

Figure 9, la latitude est une valeur angulaire, expression du positionnement Nord ou Sud d'un point sur Terre. Sa valeur varie entre -90° (Pôle Sud) et 90° (Pôle Nord) en passant par 0° (latitude à l'équateur). La longitude est également une valeur angulaire, expression cette fois du positionnement Est ou Ouest d'un point sur Terre. Sa valeur est de 0° sur le méridien de référence de l'IERS (souvent confondu avec le méridien de Greenwich (aussi appelé Premier Méridien) en raison de leur proximité (une centaine de mètres). L'erreur a d'ailleurs été faite sur la

Figure 9…). La valeur de la longitude varie entre -180° et 180°. En combinant les deux angles, la position à la surface de la Terre peut être spécifiée.

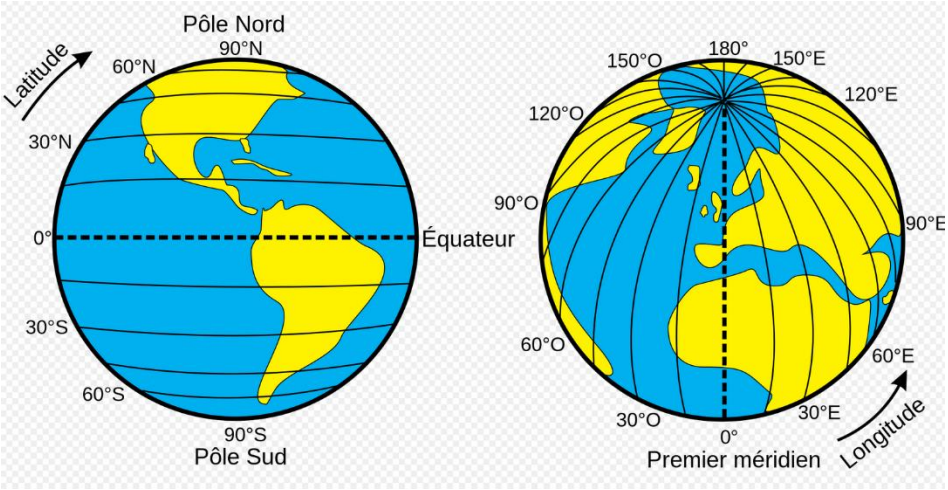


Figure 9 : Représentation des notions de latitude et de longitude permettant de situer un point à la surface de la planète Terre. [3]

Que ce soit pour la latitude ou la longitude, il existe plusieurs manières de les noter. Les coordonnées géographiques sont traditionnellement exprimées dans le système sexagésimal, parfois noté « DMS » : degrés (°) minutes (′) secondes (″). L'unité de base est le degré d'angle (1 tour complet = 360°), puis la minute d'angle (1° = 60′), puis la seconde d'angle (1° = 3 600″).

Voici plusieurs exemples de notation pour un même couple de coordonnées, le nombre maximal de décimales spécifié permet de garantir une précision au mètre près de la position :

- Degrés-décimales (DD),
    - Ex : Latitude = -77,508333°, Longitude = 164,754167° ;
    - Intervalle des valeurs : Latitude [-90°,90°] et Longitude [-180°, 180°] ;
    - Renseigner les coordonnées jusqu'à la 6ieme décimale.
- Degré-minutes-décimales (DDM),
    - Ex : Latitude = 77° 30,4998' S, Longitude = 164° 45,2500' E ;
    - Intervalle des valeurs : Degrés pour les latitudes [0,89], Degrés les longitudes [0,179] comme entiers et Minutes [0, 59.9999] jusqu'à 4 décimales après la virgule.
- Degré-minutes-secondes (DMS).
    - Ex : Latitude = 77° 30' 29,9998" S, Longitude = 164° 45' 15,0012" E ;
    - Intervalle des valeurs : Degrés [0,59] et Minutes [0, 59] comme entiers et Secondes [0,59.9999] jusqu'à 4 décimales après la virgule.

Pour les notations DDM et DMS, on ajoute une lettre après les valeurs. N ou S (Nord ou Sud) pour la latitude et E ou O (Est ou Ouest) pour la longitude. Cela permet d'éviter d'avoir des valeurs négatives. Exemple : en notation DD ci-dessus, on aurait pu écrire Latitude = 77,508333 S ; Longitude = 164,754167 E.

2. Distance entre deux points

La planète Terre n'est en réalité pas totalement sphérique : elle présente une forme un peu aplatie aux pôles. On la représente donc généralement par un ellipsoïde de révolution. Cependant, pour des raisons de simplification, nous allons à partir de maintenant la considérer totalement sphérique, et ce jusqu'à la fin de l'étude.

Le rayon de la Terre aux pôles et à l'équateur étant respectivement d'environ 6357 km et 6378 km nous allons donc choisir un rayon R = 6371 km pour notre représentation sphérique de la Terre. Cette distance est généralement considérée comme la moyenne des rayons de la planète.

Soit deux points quelconques A(lat1, long1) et B(lat2, long2) à la surface d'une sphère. Soit dr la distance angulaire en radians entre ces deux points. Alors, d'après la Figure 10,

$$dr = \cos^{-1}(\sin(lat1) * \sin(lat2) + \cos(lat1)\cos(lat2)\cos(long2 - long1))$$

Et donc D, la distance en km se trouve en multipliant dr par R :

$$D = dr * R$$

Cela est dû au fait que $D = a * C$ avec a la proportion de la circonférence de la Terre que représente l'arc dr ($a = \frac{dr}{2\pi}$) et C la circonférence de la sphère ($C = 2\pi R$)

Si l'on considère deux points A et B sur la sphère, de latitudes φ_A et φ_B et de longitudes λ_A et λ_B, alors la **distance angulaire en radians** $S_{A\text{-}B}$ entre A et B est donnée par la relation fondamentale de trigonométrie sphérique, utilisant $d\lambda = \lambda_B - \lambda_A$:

$$S_{A\text{-}B} = \arccos(\sin\varphi_A \sin\varphi_B + \cos\varphi_A \cos\varphi_B \cos d\lambda)$$

La distance S en mètres, s'obtient en multipliant $S_{A\text{-}B}$ par un rayon de la Terre conventionnel (6 378 137 mètres par exemple).

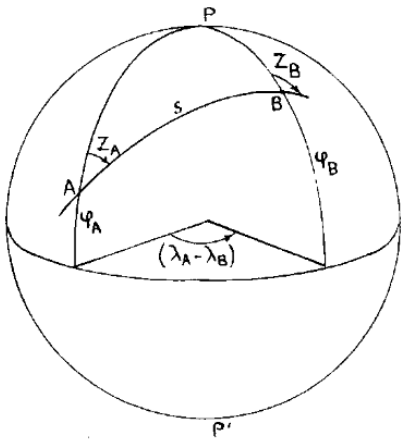


Figure 10 : Calcul d'une distance entre deux points à vol d'oiseau [4].

3. Erreur induite par l'altitude de l'aéronef ?

La formule précédente est fiable et donne des résultats très satisfaisants pour deux points situés à la surface de la représentation sphérique de la Terre. Mais pour deux points A' et B' situés cette fois à 10 km d'altitude, il faudrait multiplier dr par (R + 10) pour obtenir D'. La distance D' sera donc 0.15% supérieure à D ce qui représente 15 cm si D est de 100 m : autant dire que ceci est négligeable, surtout au vu de la marge d'erreur des données envoyées par le GPS (géolocalisation d'une précision de l'ordre du mètre).

J'ai pris pour cet exemple une altitude de 10 km or il n'existe pas d'hélicoptère pouvant aller plus haut que cela. Nous allons donc, pour la suite de l'étude, négliger l'altitude d'une localisation et nous considérerons donc que toutes les coordonnées envoyées par le GPS font référence à un point situé à la surface de notre représentation sphérique de la Terre.

4. Détermination de l'azimut

Toujours pour deux points non-confondus A(lat1, long1) et B(lat2, long2) de la surface de notre sphère, on appelle azimut ou cap vrai l'angle par rapport au nord géographique (nord vrai) selon lequel un aéronef situé au point A doit s'orienter pour se diriger vers le point B.

D'après la Figure 11, un tel cap vrai A se calcule à l'aide de la formule suivante :

$$A = 2 * \tan^{-1}\frac{y}{\sqrt{x^2 + y^2} + x}$$

Avec $x = \cos(lat1)\sin(lat2) - \sin(lat1)\cos(lat2)\cos(long2 - long1)$

Et $y = \sin(long2 - long1)\cos(lat2)$

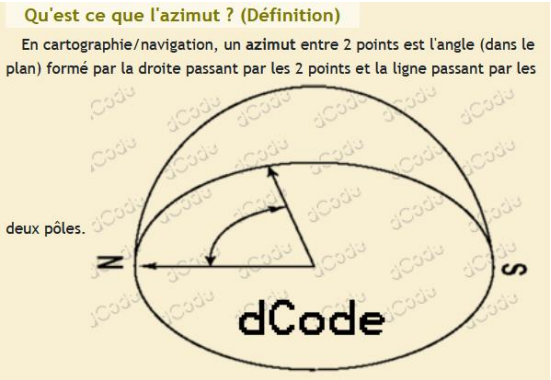


Figure 11 : Calcul de l'azimut entre 2 points à la surface d'une sphère [5].

B. Analyse preliminaire UX/UI

1. Persona*, support et équipement

Le pilote d'hélicoptère a généralement entre 20 et 60 ans, il est agile, a de bons réflexes, il est également très adaptable et autonome. Il a en outre l'habitude de manipuler des commandes exigeantes et demandant de la précision. Toutes les IHM prévues pour être utilisés par des pilotes d'hélicoptère, que ce soient des instruments de navigation ou alors des applications mobiles, respectent un ensemble de normes et de conventions bien connu et défini. Il faudra veiller à ce que EasyHover reprenne ces codes afin de proposer un environnement familier aux pilotes.

En guise de support, une tablette d'environ 7-8 pouces est utilisée pour avoir accès à certaines applications mobile facilitant la gestion de certaines situations. C'est ainsi que l'application EasyHover sera amenée à être utilisée lors des manœuvres suivantes : stationnaire sol sans repères visuels et stationnaire air.

Le pilote est bardé d'équipements en tout genre pouvant nuire à l'utilisation de la tablette : lunettes de soleil altérant la discrimination des couleurs, gants ignifugés restreignant la qualité des interactions avec un écran tactile, combinaison complète ignifugée accompagnée d'un gilet de bord et d'un casque offrants peu de mobilité etc.

2. Analyse des conditions d'utilisation

En vol, la tablette est fixée sur la cuisse du pilote (cuisse gauche pour 80% des pilotes) et tout le cockpit est soumis à d'importantes vibrations. Le pilote ne peut pas forcément se pencher en avant pour regarder l'écran de plus près et il est obligé de manipuler la tablette à une main (main gauche si tablette fixée à gauche et inversement). Dans quelques rares cas, le pilote peut manipuler la tablette à deux mains en activant le pilote automatique de l'aéronef. Comme décrit dans la partie Ces installations d'essai seraient déployées sur demande et nécessiteraient une charge de travail pour les équipes d'intégration ainsi qu'une immobilisation de la machine (coûts importants). Il serait donc préférable de pouvoir se passer de tels chantiers aéronefs.

Le besoin d'une application et ses enjeux, les pilotes sont déjà très fortement sollicités et ont très peu de charge mentale à consacrer à l'application.

Figure 12 : Photographie en position de vol d'un pilote d'hélicoptère avec sa tablette.

3. Analyse de faisabilité de certaines interactions

Sur mobile, il existe tout un tas d'interactions que les utilisateurs ont plus ou moins l'habitude d'utiliser. C'est au développeur de décider quelles interactions effectuent quelles actions dans l'application : c'est ce qu'on appelle un « bind ». Dans la plupart des applications par exemple, le fait d'écarter/pincer avec deux doigts est bindé avec les actions respectivement de zoom/dézoom…

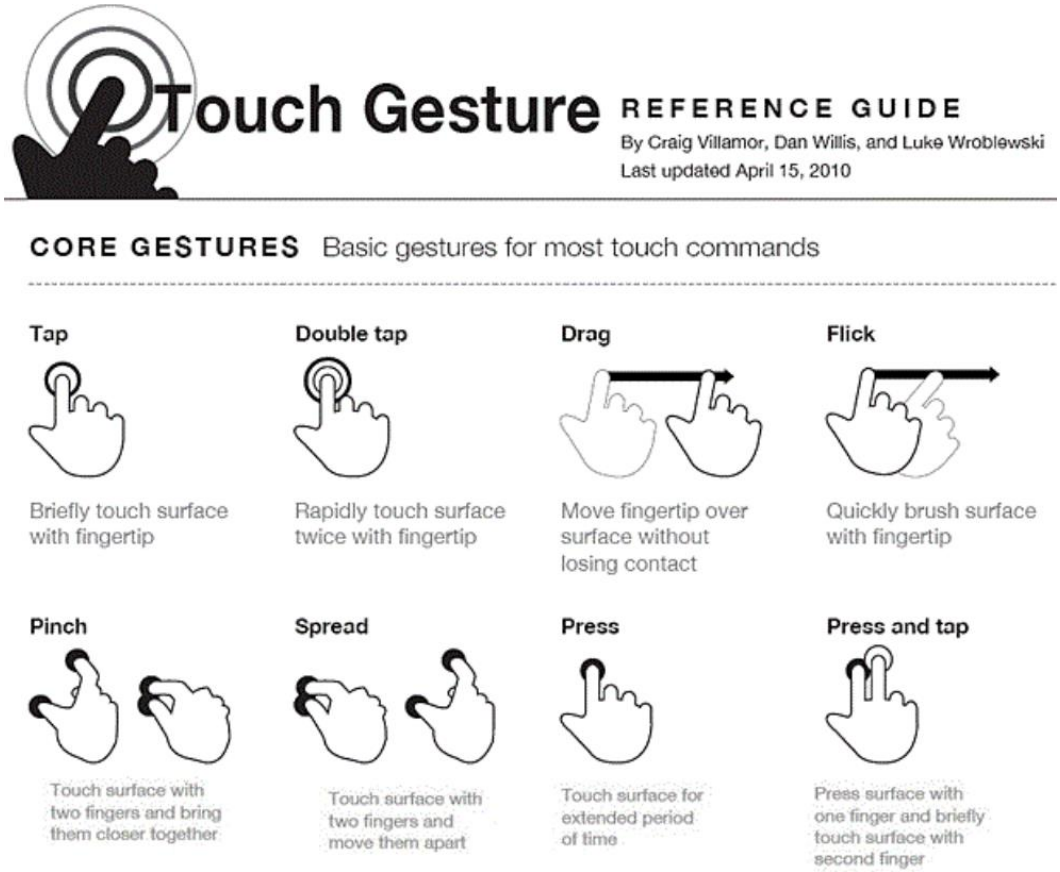


Figure 13 : Interactions tactiles les plus courantes [7].

La Figure 14 est un tableau passant en revue la plupart des possibilités d'interactions présentées sur la Figure 13 et déterminant leur faisabilité lors d'une utilisation selon les conditions détaillées dans la partie Analyse des conditions d'utilisation Pour la suite du rapport, il me semble plus parlant de garder les termes anglais pour désigner les différentes interactions. En ce qui concerne le tableau, la disponibilité des interactions est également prise en compte car elles peuvent parfois déjà être bindées avec des fonctionnalités plus générales de la tablette (détaillé dans la colonne « justification »).

Interaction	Disponible ?	Faisable en vol ?	Justification
Flick avec 3 doigts	Non	Bof	Souvent utilisé pour faire des captures d'écran.
Flick avec 2 doigts	Oui	Oui	
Flick avec 1 doigt	Oui	Oui	
Drag avec 3 doigts	Oui	Non	
Drag avec 2 doigts	Oui	Bof	
Drag avec 1 doigt	Oui	Bof	

Boutons de volume	Bof	Oui	Ferait apparaître une jauge de volume devant l'interface
Appui simultané volume et On/Off	Non	Non	Fastidieux et souvent utilisé pour captures d'écran
Press	Oui	Oui	
Tap	Oui	Oui	
Double-tap	Oui	Oui	
Pinch/Spread	Oui	Bof	

Figure 14 : Tableau traitant de la disponibilité et de la faisabilité en vol des différentes interactions présentées dans la figure précédente

4. Analyse d'accessibilité de certaines zones

En termes purement mécaniques et en prenant en compte la taille de la tablette (7-8 pouces) par rapport aux positions et aux morphologies des mains du pilote, certaines zones de l'écran tactile sont plus ou moins faciles à atteindre et à manipuler. L'emplacement sur l'écran de certaines commandes peut donc déterminer l'aisance avec laquelle le pilote réussira à interagir avec. Il est donc très important de cartographier et de quantifier le niveau d'accessibilité des différentes zones de l'écran tactile en fonction des cas d'utilisation : au sol, sans contraintes particulières et en vol, avec toutes les contraintes impliquées par le pilotage de l'hélicoptère. Après entretien avec un pilote, voici une représentation grossière des résultats de mon analyse :

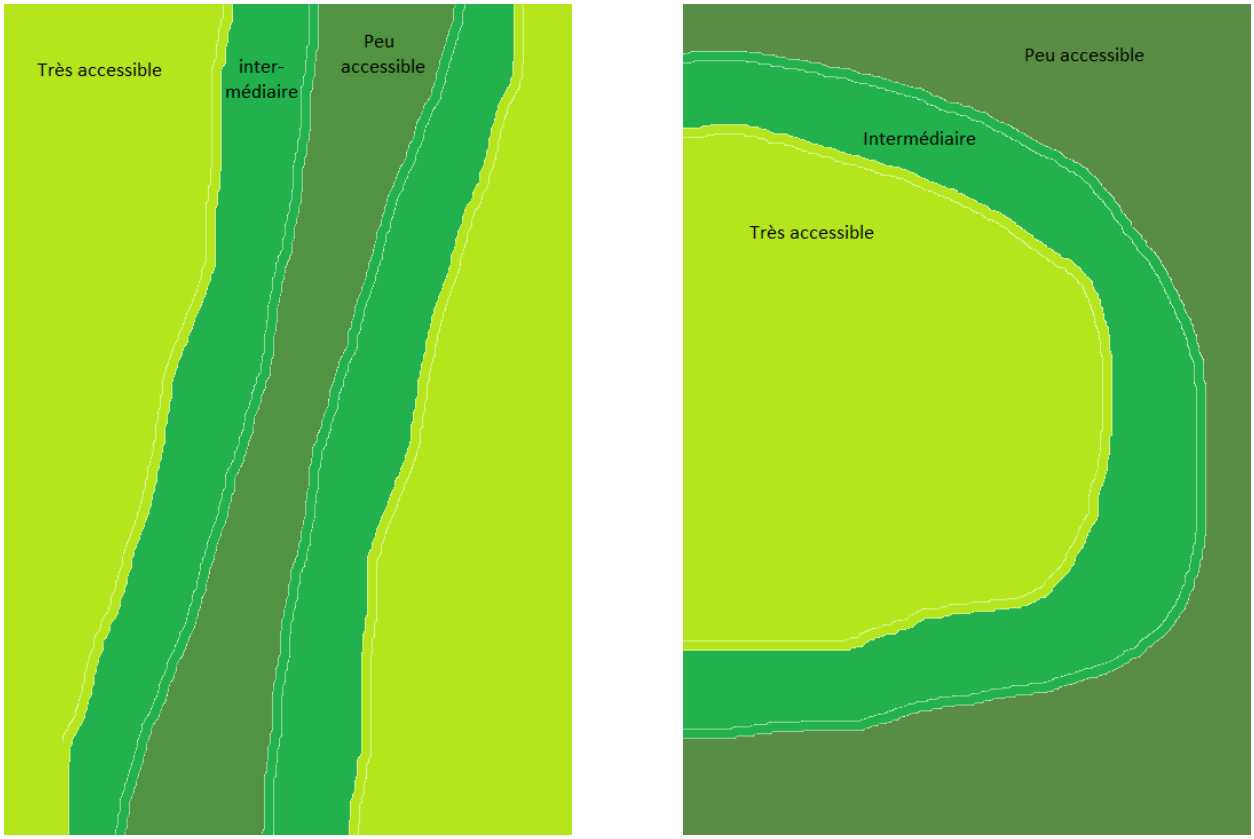


Figure 15 : Cartographie de l'accessibilité des zones de l'écran tactile d'une tablette sous différentes conditions d'utilisation

La disposition des éléments sur l'IHM devra donc tenir compte de cette cartographie, en particulier pour éviter de placer les boutons les plus utilisés dans des zones peu accessibles de l'écran.

C. UX DESIGN : NAVIGATION ENTRE LES PAGES

Il y a 4 pages différentes : La page d'accueil, le tutoriel, la page de configuration et la page de travail. La page de travail possède en outre deux états similaires mais bien distincts : Le stationnaire sol et le stationnaire air. Afin d'assurer une navigation correcte entre ces deux états, deux options s'offrent à moi :

- L'application détermine automatiquement et en temps réel dans quel état elle se trouve en fonction des inputs de l'utilisateur. Ceci implique une logique de transition plutôt complexe dans le comportement du graphique. Cette solution propose 3 champs directement accessibles depuis la page de travail à travers desquels le pilote peut saisir des données comme le cap, la force du vent etc. Cependant, vu qu'il est indispensable de renseigner la direction du vent pour un stationnaire air, cela va avoir tendance à surcharger la page de travail.

- L'utilisateur choisi sur la page d'accueil s'il veut effectuer un stationnaire air ou bien un stationnaire sol. Bien que cela rende la transition entre les deux états (stationnaire air ou sol) moins dynamique, cela permettrait d'épurer la page de travail, de rendre le graphique

plus intuitif tout en diminuant la complexité des logiques de transition de la page de travail. Je vais donc privilégier cette solution pour la conception de EasyHover.

L'application va donc par défaut s'ouvrir sur la page d'accueil. Sur cette page seront notamment disponibles 4 boutons qui permettront d'accéder aux autres pages. Depuis les autres pages, seul un bouton « retour » est proposé, permettant d'à tout moment retourner à la page d'accueil. C'est sur cette page que le pilote détermine s'il veut effecteur un vol stationnaire géographique ou bien un vol stationnaire par rapport à la masse d'air. La page d'accueil est donc centrale et il faudra tout le temps y repasser pour pouvoir naviguer au sein de l'application. Après une discussion avec le pilote, j'ai adopté sa proposition qui était de permettre un passage direct entre les modes stationnaire air et stationnaire sol en effectuant un clic long sur le bouton retour de la page de travail. J'ai essayé de représenter cette logique de transition sur Figure 16 :

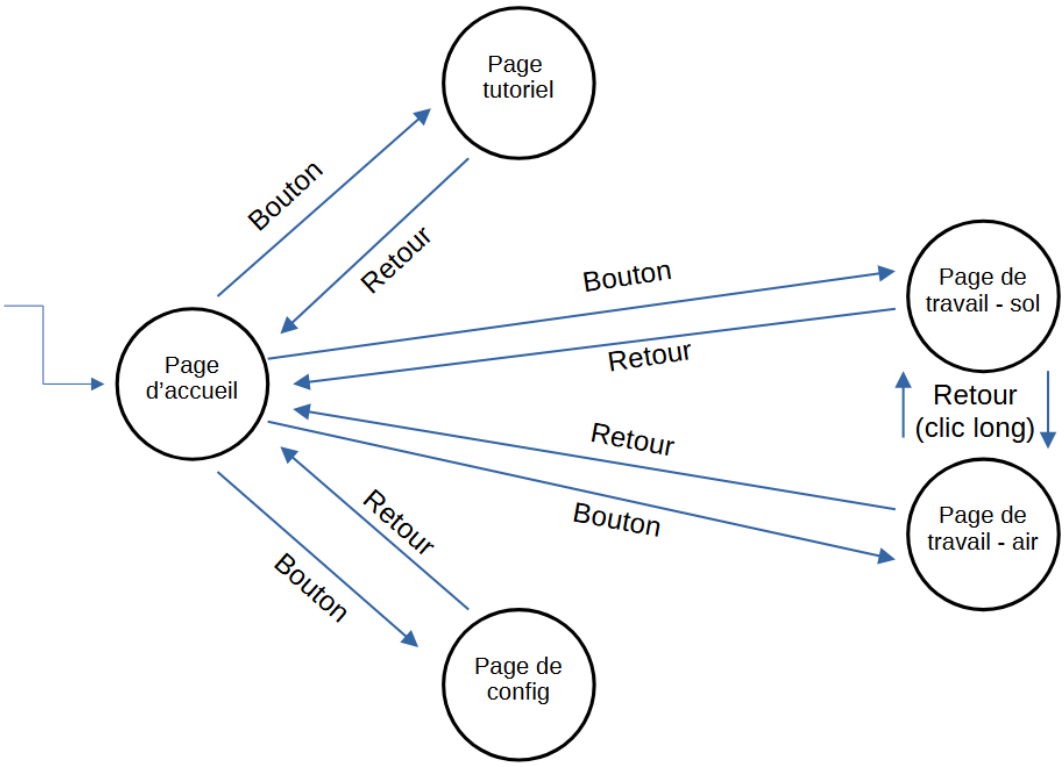


Figure 16 : Logique de transition pour la navigation entre les différentes pages de EasyHover

D. Comportement du graphique de la page de travail

1. Les différentes consignes fournies par le pilote

Pour que l'application puisse assister le pilote de façon efficace, il doit lui communiquer un certain nombre d'informations. Nous appellerons ces informations des consignes. Il existe 4 types de consignes :

- La position de référence : Dans le cadre d'un stationnaire sol, elle indique la position au-dessus de laquelle doit maintenir l'aéronef.

- Le cap vrai : Utile pour les deux types de stationnaires, c'est l'angle entre l'axe longitudinal de l'aéronef et le pôle Nord géographique de la planète Terre. Par exemple, si le cap vrai de l'aéronef est de 90°, alors cela veut dire que son nez pointe vers l'Est. Je rappelle que la fonction boussole de la tablette n'est pas exploitable donc c'est au pilote de renseigner à l'avance ou en temps réel le cap vrai qu'il souhaite maintenir ou qu'il maintient déjà actuellement.

- La force du vent : Dans le cadre d'un stationnaire air, elle exprime la vitesse du vent, considéré comme constant et homogène en haute altitude. C'est donc la vitesse à laquelle doit aller l'aéronef pour réussir son stationnaire air (= se laisser porter par le vent).

- La direction du vent : Pour éviter un effet de vortex* lors d'un stationnaire air, le pilote doit se placer face au vent. Cette information est donc intimement liée au cap vrai donné en consigne.

2. La façon de fournir ces consignes

Il existe plusieurs façons de communiquer à l'application les consignes détaillées ci-dessus :

a. La position de référence :

- Rentrer les coordonnées géographiques « à la main » avec un clavier tactile intégré à l'application se déployant sur demande.

- « Ploter » les coordonnées géographiques une fois arrivé sur place. Au moment où le pilote effectue l'action de ploter, l'application récupère la position actuelle de l'aéronef et l'utilise pour définir la position de référence. Cette commande équivaut à « je veux maintenir la position actuelle ».

b. Le cap vrai :

- Rentrer la valeur angulaire « à la main » avec un clavier tactile intégré à l'application se déployant sur demande.

- Sur le graphique est affiché un cadran numérique ressemblant fortement à une boussole. Ce dernier permet de lire intuitivement la valeur cap vrai et permet de comprendre d'un seul coup d'œil la direction vers laquelle le nez de l'aéronef est censé pointer. En outre, les pilotes sont habitués à ce genre d'affichage. Ce cadran permet également d'ajuster facilement le cap vrai de consigne à l'aide d'une simple interaction « drag » (voir Figure 13) : on ajuste en fait le cap vrai de consigne en faisant tourner le cadran à la manière d'un gros potentiomètre.

- On rappelle que le pilote doit se placer face au vent pour effectuer un stationnaire air. En conséquence, lorsque le pilote ajuste la consigne de direction du vent, l'application propose automatiquement de régler le cap vrai à une valeur de même direction mais de sens contraire. Nous appellerons cette manipulation « confirmer face au vent ».

c. La force du vent

Malheureusement pas de solutions plus ergonomiques que de rentrer la valeur de vitesse « à la main » avec un clavier tactile intégré à l'application se déployant sur demande.

d. La direction du vent

- Rentrer la valeur angulaire « à la main » avec un clavier tactile intégré à l'application se déployant sur demande.

- Le cadran numérique évoqué précédemment étant exclusivement réservé au réglage du cap vrai de consigne, cette solution n'est pas disponible pour le réglage de la direction du vent. On rappelle en revanche que le pilote doit se placer face au vent pour effectuer un stationnaire air. En conséquence, lorsque le pilote ajuste la consigne de cap vrai, l'application propose automatiquement de régler la direction du vent à une valeur de même direction mais de sens contraire. Nous appellerons cette manipulation « confirmer vent de face ».

3. Les différents cas d'utilisation

Nous avons vu qu'il existe 4 types de consigne à renseigner à l'application et qu'il y a différentes manières de renseigner ces consignes. En fait, ces consignes peuvent également être renseignées à différents moments. Il existe donc différentes approches et on peut recenser une dizaine de scénarios (cas d'utilisation) :

	Type de stationnaire	**Coordonnées géo**	**Cap vrai (nord géo)**	**Force vent**	**Direction vent**
Cas 1	Géographique	Rentrer avant décollage	Rentrer avant décollage	X	X
Cas 2	Géographique	Rentrer avant décollage	Rentrer sur place	X	X
Cas 3	Géographique	Rentrer sur place	Rentrer sur place	X	X
Cas 4	Géographique	Ploter sur place	Rentrer sur place	X	X
Cas 5	Géographique	Rentrer avant décollage	X	X	X
Cas 6	Géographique	Rentrer sur place	X	X	X
Cas 7	Géographique	Ploter sur place	X	X	X
Cas 8	Air	X	Rentrer sur place	Rentrer sur place	Rentrer sur place
Cas 9	Air	X	Confirmer face au vent	Rentrer sur place	Rentrer sur place

Cas 10	Air	X	Rentrer sur place	Rentrer sur place	Confirmer vent de face
Cas 11	Air	X	X	Rentrer sur place	Rentrer sur place

Figure 17 : Tableau recensant les différents cas d'utilisation du graphique en fonction du type d'infos rentrées et du moment auquel elles sont rentrées

Petites précisions :

- Cas 1 : Les coordonnées et le cap sont paramétrables au sol avant décollage car le pilote a reçu un objectif extrêmement précis comme par exemple « positionnez-vous au-dessus de telle zone, orienté face à telle cible ».

- Cas 2 : Le pilote sait quel point il doit rejoindre mais il ne sait pas encore quel cap il devra tenir une fois arrivé sur place. Exemple : « Positionnez-vous au-dessus de telle zone, orientez-vous face au vent une fois sur place ».

- Cas 4 et 7 : Une fois sur place, au lieu de rentrer manuellement les coordonnées, le pilote peut « ploter ». Ça veut dire qu'il récupère la localisation GPS actuelle et fait comprendre à l'application qu'il veut tenir cette position. Comme expliqué précédemment, cette fonctionnalité n'est pas disponible pour la détermination de l'orientation (cap) de la cellule car la boussole de la tablette est inexploitable. Le pilote devra donc tout de même rentrer le cap à la main, si besoin.

- Cas 8, 9, 10 et 11 : Avant d'être sur place, on ne peut pas obtenir d'infos fiables sur la force et la direction du vent en altitude, qui est pourtant considéré comme constant en force et en direction. Il ne peut donc pas y avoir de paramétrage au sol avant décollage.

- Cas 9 et 10 : Après avoir rentré la direction du vent, l'application émet l'hypothèse que le pilote a placé l'hélicoptère face au vent. Le pilote peut ainsi confirmer en un clic cette hypothèse via l'interface au lieu de rentrer manuellement le cap vrai et inversement pour le cas 10.

4. Orientation du graphique

Le graphique est orienté par défaut en mode « North-up » avec un objet rond en son centre symbolisant un hélicoptère dont on ignore l'orientation. Pour chacun des scénarios détaillés en Figure 17, si le cap vrai de la cellule est renseigné, alors le graphique s'orientera automatiquement en mode « Heads-up » avec une image d'hélicoptère en son centre. L'orientation « Heads-up » améliore la lisibilité du graphique en le rendant bien plus intuitif et cela permet d'économiser de la charge mentale pour le pilote.

E. Clavier Tactile ergonomique

Pour rentrer « à la main » les différentes consignes détaillées précédemment, je comptais lors de ma réflexion utiliser le clavier tactile traditionnel fourni par le système d'exploitation de la tablette.

Il se déploie automatiquement dès que la tablette comprend que l'utilisateur veut taper quelque chose dans une barre de recherche ou encore rédiger un message par exemple. Cependant, lors d'une discussion avec un pilote, il m'a dit que les conditions d'utilisation en vol (gants peu tactiles, tremblements, vibrations, peu de précision dans les interactions avec l'écran tactile) rendaient l'utilisation de ce genre de clavier impossible.

Nous avons donc convenu ensemble du besoin de l'implémentation d'un clavier ergonomique et adapté. Ce clavier utiliserait le plus de place possible à l'écran avec des boutons dont la taille serait maximisée. Pour diminuer le nombre de boutons, il faudra uniquement afficher les boutons nécessaires à la saisie des types de consigne suivants : valeur angulaire (pour le cap vrai et la direction du vent), valeur de vitesse (force du vent) et coordonnées géographiques (pour la position de référence) avec ses différentes conventions d'écriture (voir Coordonnées géographiques).

Les boutons à afficher sont donc les suivants :

- Les chiffres de 0 à 9
- Les 4 lettres N, E, S et W (pour North, East, South et West)
- Le point « . »
- Les boutons « OK » pour valider, « <- » pour effacer le dernier caractère et « Back » pour annuler la saisie et retourner en arrière.

Lors de l'appui sur « OK », la saisie sera vérifiée par l'expression régulière* correspondante au type de consigne concerné avant d'être validée. Si la vérification n'aboutit pas, alors un message d'erreur apparaît et le pilote doit corriger sa saisie.

F. MAQUETTES PROTOTYPE INTERACTIF

Pour permettre d'illustrer toutes ces réflexions autour de la conception de l'UX design de l'application, j'ai réalisé plusieurs maquettes du graphique avec libreOffice Draw (voir deux exemples sur la Figure 18, on voit bien le cadran numérique et les barres de tendance)

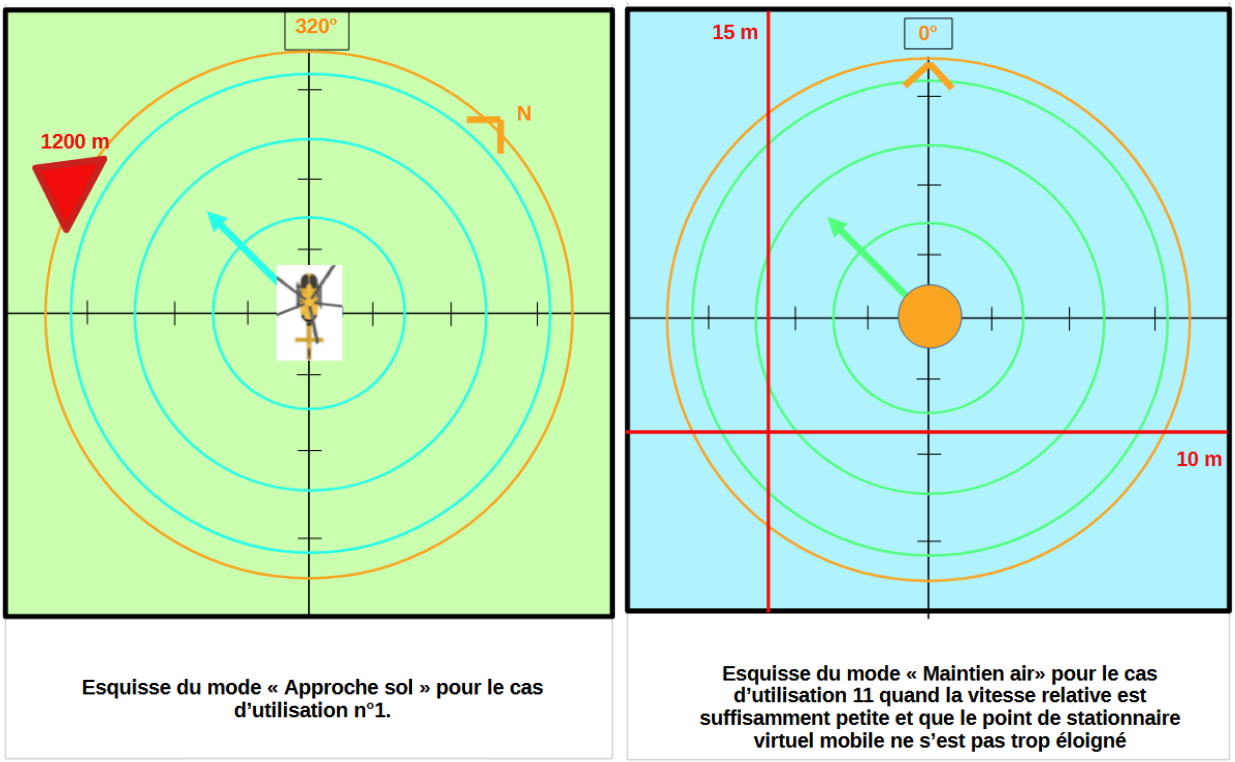


Figure 18 : Maquettes du graphique pour deux cas d'utilisation différents

J'ai ensuite intégré ces maquettes dans un prototype interactif pour leur donner un peu de vie et qu'un observateur extérieur puisse mieux comprendre l'utilité de chacune des 12 maquettes réalisées. Ce prototype interactif a été réalisé grâce au logiciel Figma, voici un lien permettant la consultation de ma dernière version en date (il vous faudra malheureusement créer un compte Figma pour pouvoir lancer le prototype) :

https://www.figma.com/file/iJeWqkIqHO2djKhV7JmXXC/EasyHover?type=design&node-id=45%3A10&mode=design&t=Y6Dnm4iHJvqwChrm-1

Ce prototype est très complet (voir Figure 19), le logiciel n'offrant pas la possibilité de reconnaître un clic long, je l'ai remplacé par un « drag and drop » dans le prototype.

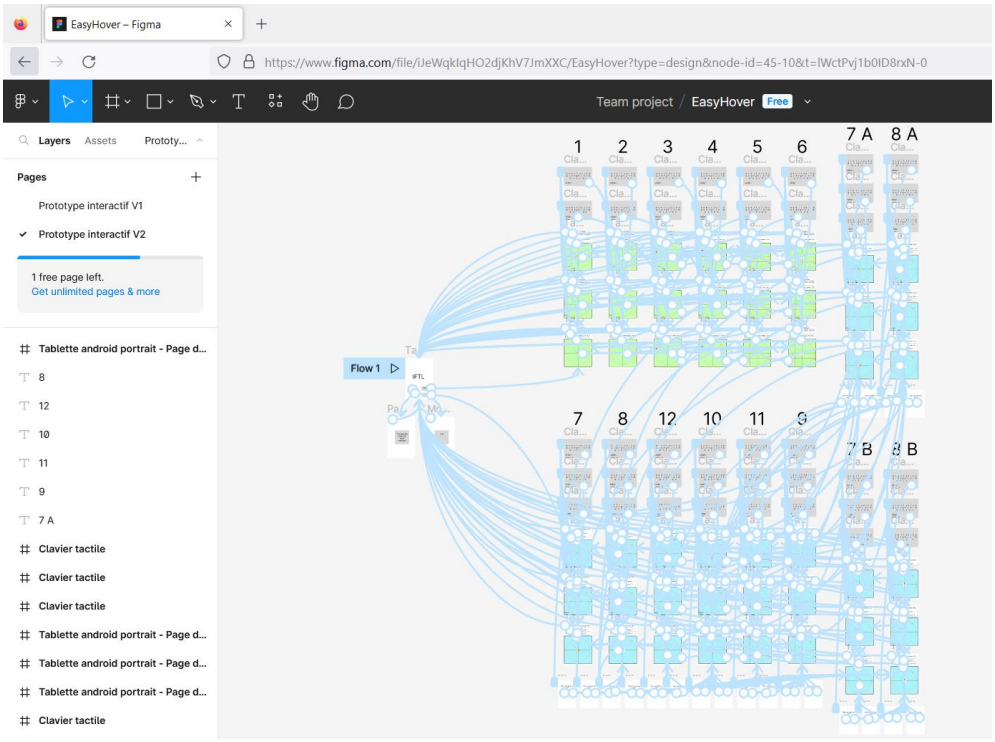


Figure 19 : Aperçu de la version finale du prototype interactif de EasyHover (capture d'un écran d'édition dans le logiciel Figma)

IV. Réalisation du projet : Implémentation

A. PRESENTATION GENERALE DU PROJET DE CODE

1. .Net MAUI : le C# et la Programmation orientée Objet

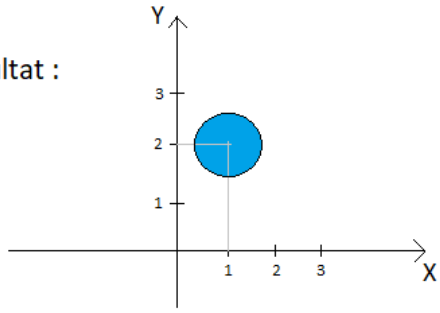
La programmation orientée objet (POO) est un paradigme de programmation qui utilise des objets pour représenter des entités ou des concepts. On distingue ainsi une classe, qui définit la structure et le comportement d'un type d'objet, d'une instance, qui est un objet spécifique créé à partir de cette classe. La classe sert de modèle ou de plan pour créer des instances, qui possèdent leurs propres données (attributs) et fonctionnalités (méthodes) définies par la classe. Les instances permettent de représenter des entités individuelles et d'interagir avec elles de manière spécifique. La POO repose sur l'encapsulation des données et des méthodes, l'héritage pour la réutilisation du code et le polymorphisme pour manipuler des objets de différentes classes de manière similaire. Elle permet une modélisation intuitive et une organisation modulaire du code.

```
1 référence
public class Balle
{
    // Attributs
    private int coordonneeX;
    private int coordonneeY;
    private Color couleurBalle;

    // Constructeur
    0 références
    public Balle(int x, int y, Color couleur)
    {
        this.coordonneeX = x;
        this.coordonneeY = y;
        this.couleurBalle = couleur;
    }

    // Méthode(s)
    0 références
    public void afficherBalle()
    {
        /* Code permettant de dessiner une
           balle de couleur couleurBalle et
           de coordonnées (coordonneeX, coordonneeY) */
    }
}
```

Exemple d'instanciation :

```
Balle maBalle = new Balle(1, 2, Colors.Blue);
maBalle.afficherBalle();
```

Résultat :



Ici, maBalle est une instance de la classe Balle et l'objet maBalle a exécuté la méthode afficherBalle() pour apparaître sur le graphique.

Figure 20 : Exemple de code C# (POO) exploitant notamment les notions de classe et d'instance

La plateforme de développement .Net MAUI se basant sur le langage C#, la programmation est donc orientée objet. A ce titre, mon code sera uniquement composé de différentes définitions de classes. Traditionnellement, un fichier de code contient une seule définition de classe. Lorsque la définition d'une classe nécessite énormément de lignes de code, alors on peut la scinder et la répartir dans plusieurs fichiers de code pour gagner en lisibilité. Les différents fichiers contiennent donc chacun ce une classe partielle, qui sont réunies automatiquement à la compilation pour obtenir une classe complète.

En .Net MAUI, chacune des pages de l'application est en fait définie par une classe dédiée. Ainsi, comme on peut le voir sur la Figure 21, les classes définies dans les fichiers de code du cadre bleu

correspondent aux différentes pages de EasyHover et la page de travail est définie par deux classes partielles, signe de sa richesse et de sa complexité. Les dossiers du cadre jaune contiennent des classes très spécifiques dont les canevas, qui ont notamment servis à l'affichage du graphique de la page de travail. Mais pas de panique, j'y reviendrai plus tard… Les fichiers encadrés en jaune assurent le bon fonctionnement du gestionnaire de versions Git et, résumé grossièrement, le reste des fichiers concernent des aspects très poussés de compilation : ils n'ont généralement pas besoin d'être retouchés.

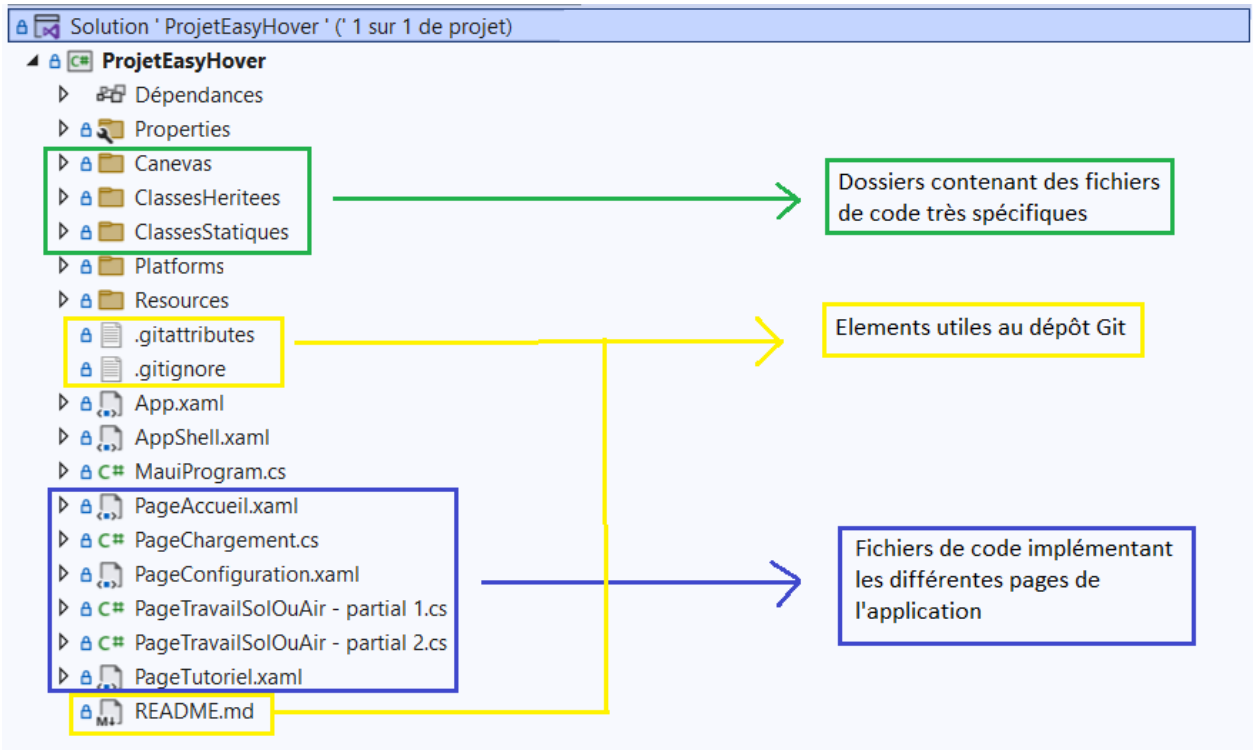


Figure 21 : Organisation des fichiers de code du projet EasyHover

2. Disposition et code-behind

Vous l'aurez remarqué sur la Figure 21, certaines de mes définitions de classes sont contenues avec des fichiers .xaml et d'autres avec des fichiers .cs. Le C# étant un langage orienté objet, il est possible de créer et de manipuler tous les éléments graphiques sous forme d'objets directement dans un fichier de code C#. La deuxième option, plus courante, revient à utiliser le langage XAML, un langage de balise déclaratif qui permet de facilement développer les interfaces utilisateurs pour les applications .NET. Le XAML permet de facilement séparer la définition des objets de l'interface de leur comportement, implémenté directement dans le code. Les fichiers XAML sont ainsi toujours liés à un fichier de code (ici écrit en C#). Le XAML gère donc la disposition des différents éléments visuels tandis que le fichier C#, appelé code-behind, gère le comportement de ces éléments visuels. Par exemple, le fichier PageAccueil.xaml est accompagné d'un fichier PageAccueil.xaml.cs. Les fichiers .cs ne sont en revanche pas forcément liés à un fichier .xaml.

Après compilation, le code de la Figure 22 et le code de la Figure 23 donnent exactement le même résultat : celui de la Figure 24. A chaque appui sur le bouton, un compteur s'incrémente. Ce comportement n'aurait pas pu être implémenté avec du XAML seul. Pour le code sans xaml, j'ai dû

instancier un par un tous les éléments visuels, puis les ajouter dans un container (l'objet layout) puis j'ai dû exécuter la commande « Content = layout » ce qui signifie que j'ajoute mon container dans l'affichage, ce qui revient à placer les balises <verticalStackLayout></VerticalStackLayout> dans les balises <ContentPage></ContentPage> dans un code XAML.

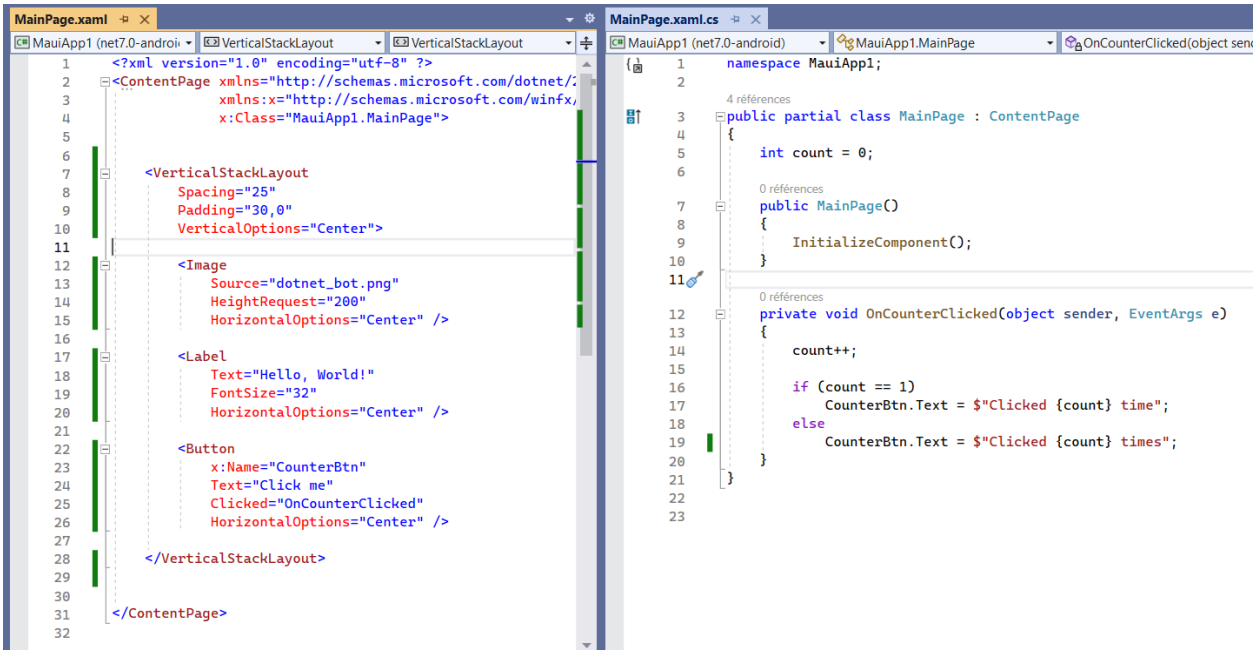


Figure 22 : Hello World codé avec un fichier .xaml lié avec un fichier .cs

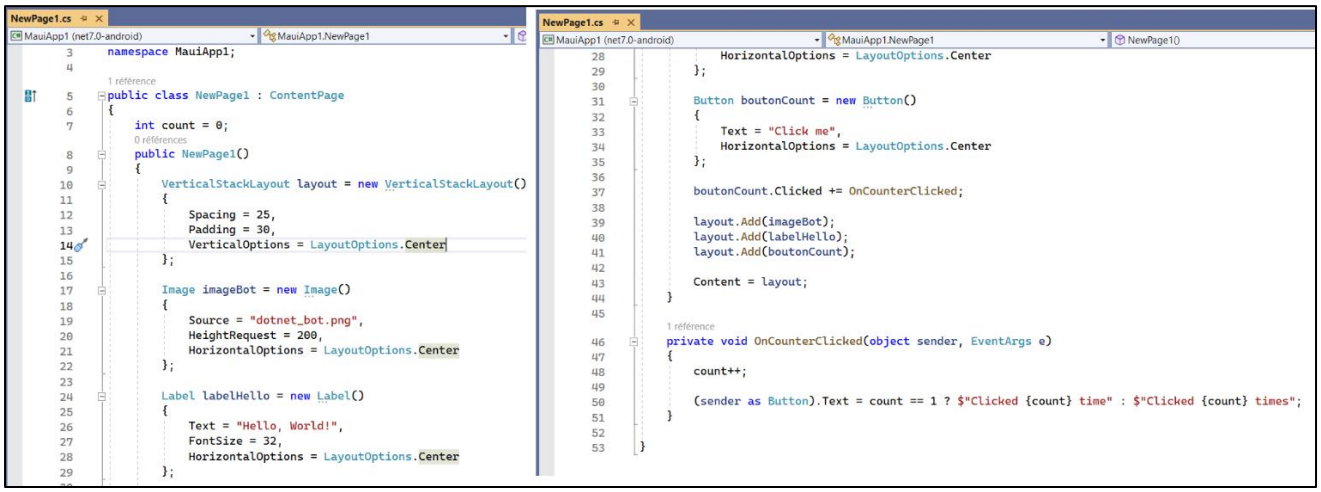


Figure 23 : Hello World codé uniquement avec un fichier .cs

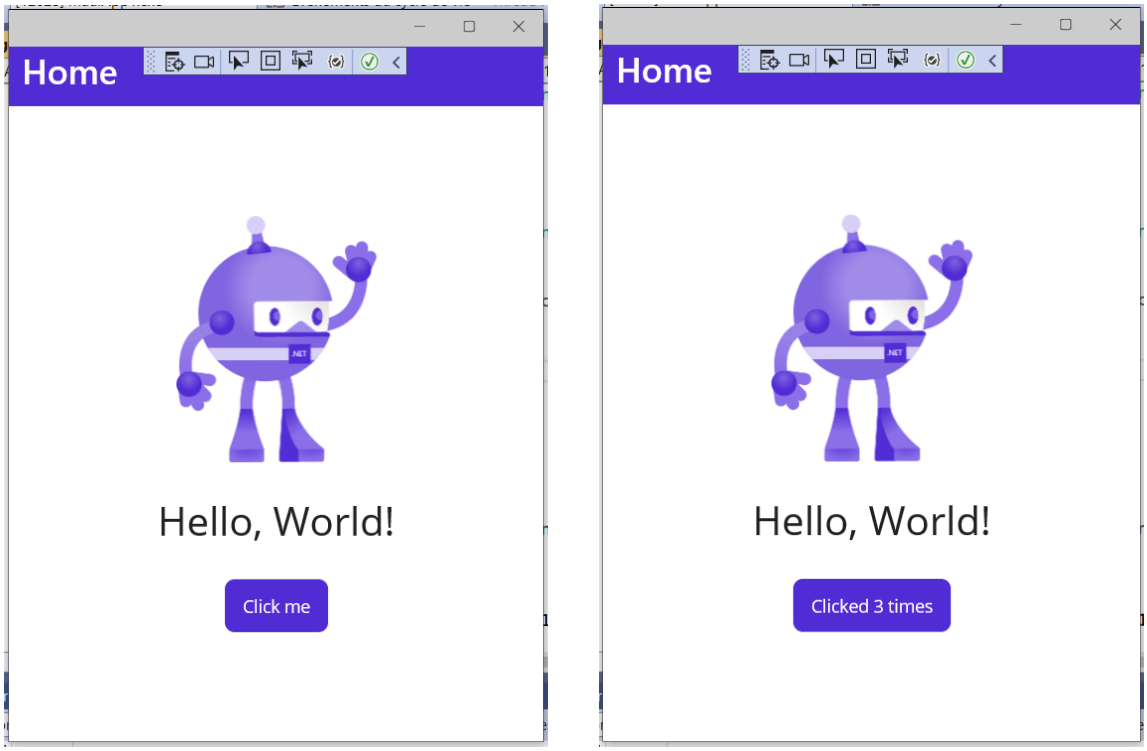


Figure 24 : Résultat du Hello World compilé sur un système Windows

On remarque que le XAML peut être très pratique dans certains cas comme celui étudié ci-dessus. Cependant, sa principale faiblesse est que l'on ne peut pas ajouter de conditions ou de boucles puisque c'est un langage à balises. On peut ainsi se retrouver limité dans certains cas et il devient nécessaire de remplacer le XAML par des commandes C#.

C'est ainsi que j'ai décidé de complétement me passer de XAML pour l'implémentation de la page de travail qui est extrêmement riche et complexe et nécessite toutes sortes de conditions. En revanche, pour la page de chargement qui est extrêmement basique, l'utilisation ou non de xaml ne faisait aucune différence donc mon choix a dépendu de mon humeur du moment.

3. La puissance de l'EDI

Lors de ma formation à l'ISIMA, beaucoup d'exercices portaient sur le développement de logiciels en C sans EDI. Nous avions pour cela un simple éditeur de texte à disposition, il fallait donc écrire l'intégralité des fichiers nécessaires à la compilation, taper les bonnes commandes dans le terminal etc. Ici, ayant codé mon projet .Net MAUI à l'aide de l'EDI Visual Studio, tout est automatiquement pris en charge. De plus, lors de la création de projet, l'EDI nous fournit un exemple simpliste d'application prête à l'emploi. Il suffit donc de créer le projet, modifier l'existant à sa guise en rajoutant des pages ou en modifiant les pages déjà présentes, appuyer sur le bouton compiler et c'est parti !

L'EDI propose même une interface graphique Git, très pratique pour effectuer efficacement la gestion des versions ce qui est généralement peu aisé sans interface graphique.

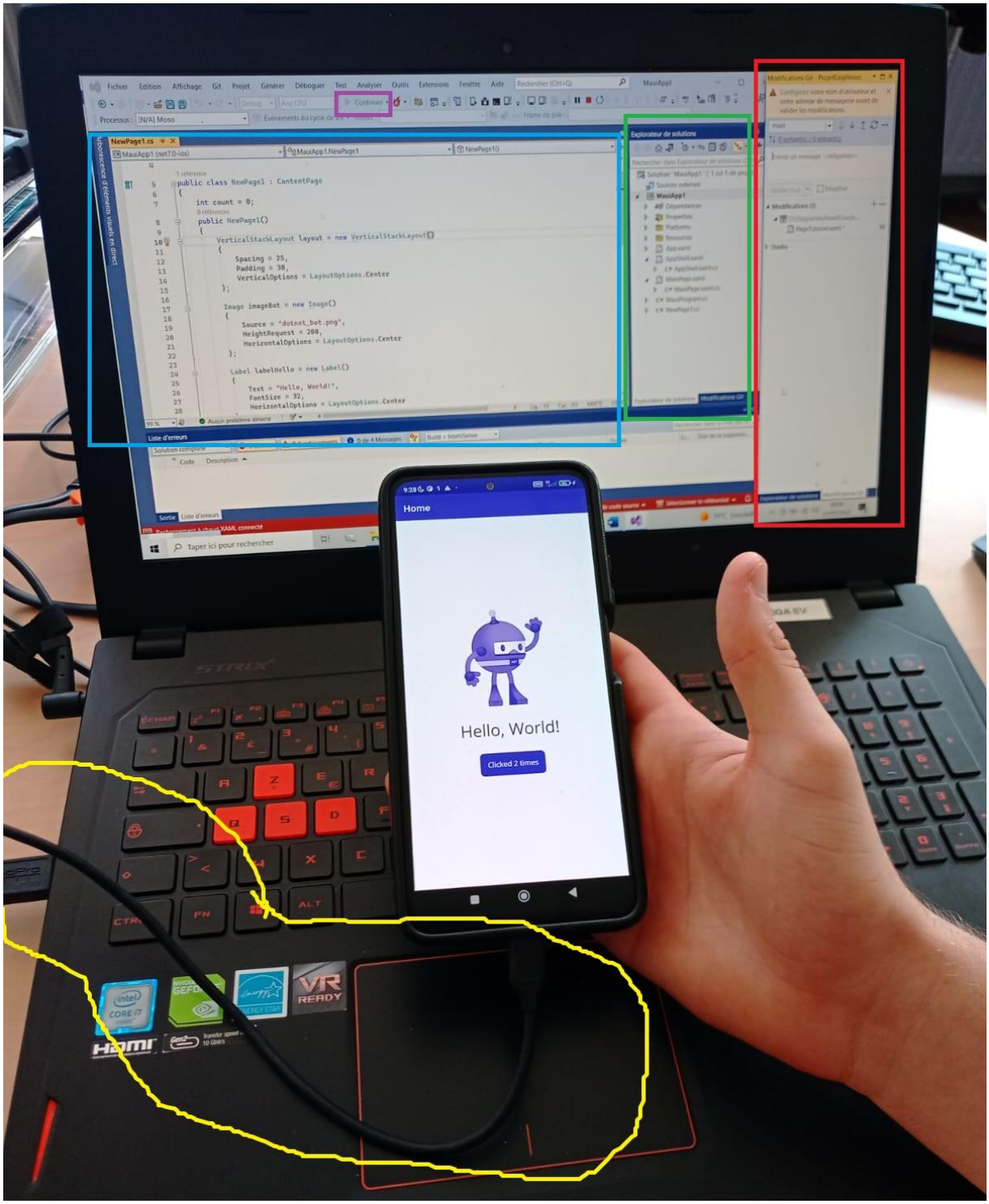


Figure 25 : Photo mettant en évidence la puissance de mon EDI

Sur la Figure 25, on peut voir les outils avec lesquels j'ai développé pendant mon stage. L'ordinateur portable mis à disposition par l'IFTL, mon téléphone personnel relié à l'ordinateur avec un câble USB (entouré en jaune). Sur l'écran de l'ordinateur est affiché l'EDI Visual Studio, la fenêtre d'édition de code est encadrée en bleu, la fenêtre encadrée en vert affiche l'arborescence des fichiers du projet. L'interface graphique Git mise à disposition par l'EDI est encadrée en rouge. Enfin, le

bouton permettant la compilation automatique ainsi que le déploiement sur mon téléphone personnel pour mener des tests est encadré en violet.

B. DISPOSITION DES ELEMENTS VISUELS

1. Les éléments visuels

Ce sont les objets constitutifs de l'interface utilisateur, les éléments que l'utilisateur voit et avec lesquels il interagit. .Net MAUI propose une très large bibliothèque d'éléments visuels, allant des plus basiques aux plus complexes. Pour le développement de EasyHover, je me suis contenté d'utiliser les plus courants :

- L'affichage de texte, de classe Label. Il permet d'afficher du texte, contenu dans des variables de type String (variables ne contenant pas des entiers ni des décimaux, mais plutôt du texte). On peut choisir la taille et la couleur de la police, la disposition du texte et son emplacement.

- Le bouton, de classe Button. Elément incontournable de toute interface utilisateur, la taille, la forme et l'emplacement du bouton sont personnalisables. Il porte le label de votre choix et une petite animation se lance automatiquement lors d'un clic (comme si c'était un bouton-poussoir et qu'il s'enfonçait au moment du clic avant de revenir à sa position initiale). Cette animation permet de confirmer à l'utilisateur qu'il a bien cliqué et elle est plutôt agréable. De très petites modifications du code permettent d'exécuter un retour haptique (petite vibration transmise au doigt de l'utilisateur) à chaque clic. Les méthodes à exécuter lors du clic sur un bouton sont définies dans le code-behind.

- L'interrupteur, de classe Switch. Généralement utilisé pour permettre le réglage On/Off d'une fonctionnalité. Également entièrement personnalisable, il possède deux positions. A chaque clic, l'indicateur glisse d'une position à l'autre.

- La zone de couleur, de classe BoxView. Permet de dessiner un rectangle simple, de largeur, de hauteur et de couleur spécifiées.

- Le canevas, de classe GraphicsView. Résumé grossièrement, cet élément fournit une zone dessinable. Le canevas a été extrêmement utile pour l'affichage du graphique de la page de travail.

Tous les classes définissant ces éléments visuels héritent de la classe View elle-même héritée de la classe VisualElement. D'après Microsoft Learn, cette classe définit un « élément qui occupe une zone de l'écran, qui possède une apparence visuelle et qui peut obtenir une entrée tactile. ».

2. Leur disposition

Afin de garantir un bon agencement de ces différents éléments visuels, on utilise des éléments de la classe Layout (« disposition » en anglais), elle aussi héritée de la classe View. Ces objets de type Layout se comportent comme des conteneurs pouvant accueillir des éléments visuels. Les objets contenus dans un objet Layout sont accessibles via la liste « Children » (« enfants » en anglais) avec la commande suivante : monLayout.Children ; Pour accéder à un enfant précis, dont on connaît

l'index, il faut exécuter la commande suivante : monLayout.Children[indexEnfant] ; puisque c'est une liste. Les objets layout étant eux-mêmes des objets View (héritage), on peut faire en sorte qu'un layout contienne un autre layout. On ne peut pas ajouter d'éléments visuels à une page de l'application sans qu'ils ne soient contenus dans layout. Ainsi, tous les éléments visuels sont contenus dans un « Parent », lui-même potentiellement contenu dans un autre Parent et ainsi de suite…

Il existe 4 principaux types de layout :

- La grille, de classe Grid. Elle permet de définir une grille en précisant le nombre de lignes et de colonnes. On peut ajuster de différentes manières la taille des différentes lignes et colonnes. Pour y ajouter des éléments visuels, on les « range » dans les différentes cases de la grille. On peut fusionner une case avec ses voisines afin d'y « ranger un enfant plus volumineux ». Ce layout est très utile aux UX/UI designers qui ont structuré leur interface utilisateur à l'aide d'une grille, ce qui est une pratique conseillée et très courante. Lors du développement de EasyHover, j'ai utilisé presque exclusivement des objets Grid pour agencer mes différents éléments visuels.

- Les agencements en tas, de classe StackLayout. Pouvant aller dans la direction horizontale ou bien verticale, pouvant aller du haut vers le bas, de la droite vers la gauche ou en sens inverse, cette disposition permet « d'entasser » les enfants selon différentes « règles d'entassement » : direction, sens, espacement etc. Le choix de ce type de layout très flexible et robuste est très courant pour les affichages simplistes, ne contenant pas énormément d'éléments visuels. Plusieurs HorizontalStackLayout entassés dans un VerticalStackLayout permet dans une certaine mesure de reproduire le comportement d'une Grid.

- L'affichage absolu, de classe AbsoluteLayout. Ici, chaque enfant possède un emplacement très précis repéré par des coordonnées ou des proportions qu'ils s'efforceront de respecter, quitte à chevaucher un élément voisin ou à sortir de l'enceinte du conteneur. Ce choix est excellent lorsque que l'on veut positionner des éléments de façon très précise et que l'on fait toujours tourner l'application sur un même appareil, mais on risque de se retrouver avec une disposition sens dessus-dessous au moindre changement de format d'écran. Cette disposition très rigide est plutôt risquée et très peu utilisée actuellement, surtout pour les applications multiplateformes devant être capables de tourner sur toutes sortes d'écran.

- L'affichage adaptatif, de classe FlexLayout. Cette disposition fera tout son possible pour faire apparaître ses enfants de la meilleure des façons possible, quitte à créer des nouvelles lignes d'affichage pour éviter les zones surchargées. Cette logique est similaire aux concepts FlexBox au sein des feuilles de style CSS (langage informatique permettant de mettre en forme des pages web (HTML ou XML)) en développement web.

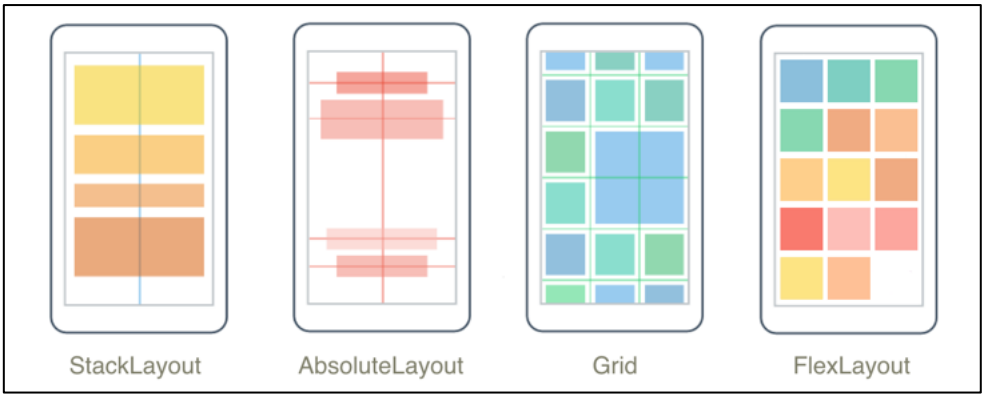


Figure 26 : Schéma imageant les 4 principaux types de disposition en .Net MAUI [6].

C. TIMERS ET PROGRAMMATION EVENEMENTIELLE

1. Les timers

Dans l'application, j'ai fait une utilisation massive des minuteurs (objet de la classe Timer de la bibliothèque System.Timers.Timer). Je vais appeler ces minuteurs des « timers ». Ces timers m'ont servi à créer des animations (déclencher un mouvement toutes les 100 millisecondes par exemple) ou alors à différencier différents types d'interactions.

Après avoir déclaré et initialisé un timer avec la valeur d'une variable de type TimeSpan (variable contenant une valeur de temps, exprimée en secondes ou millisecondes), le timer peut être démarré avec la commande monTimer.Start() ou alors arrêté avec la commande monTimer.Stop(). Dès que le timer est écoulé, il recommence à 0 et ainsi de suite. Tant qu'on n'arrête pas un timer, il continuera de s'égrener et de se réinitialiser à l'infini ce qui peut parfois poser des problèmes de mémoire. Il est donc préférable de stopper un timer dès qu'on en a plus besoin.

Le code C# suivant défini et déclare le timer « timerMajChrono » (timer permettant de mettre à jour l'affichage du chronomètre toutes les secondes afin de voir les secondes s'égrener). L'intervalle de ce timer est défini sur 1 seconde à l'aide de la commande TimeSpan.FromSeconds(1) :

```
System.Timers.Timer timerMajChrono ;

timerMajChrono = new System.Timers.Timer(TimeSpan.FromSeconds(1));
```

Une fois lancé, ce timer se réinitialise donc toutes les secondes. Nous verrons dans le point suivant qu'il est possible de détecter ce moment où il se réinitialise et d'exécuter du code en conséquence. On appelle cela un événement.

2. La programmation événementielle

a. Les événements

La programmation événementielle fait partie des piliers du développement en .Net MAUI. Cela consiste en l'exécution d'un morceau de code spécifique au moment du déclenchement de l'événement associé. L'appui sur un bouton par l'utilisateur ou la réinitialisation d'un timer sont par

exemple des événements. Les conditions de déclenchement d'un événement sont directement codées dans la définition de la classe des objets capables de déclencher ces événements. Chaque objet possède donc, s'il en possède, ses propres événements. Voici les principaux événements déclenchés par les objets utilisés dans EasyHover :

- Classe Button :
    - monBouton.Clicked : déclenché lors du clic sur un bouton (appui+relâchement, Clicked se déclenche lors du relâchement).
    - monBouton.Pressed : déclenché lors de l'appui sur un bouton (petite nuance).
    - monBouton.Released :déclenché lors du relâchement après l'appui sur un bouton.
- Classe Switch : monSwitch.Toggled → déclenché lors de l'inversion de la valeur de l'interrupteur, que ce soit par l'action de l'utilisateur (clic sur l'interrupteur), ou par une inversion algorithmique (monSwitch.IsToggled = ! monSwitch.IsToggled, l'attribut IsToggled étant un booléen public).
- Classe Timer : monTimer.Elapsed → déclenché au moment où monTimer (la commande monTimer.Start() ayant été exécutée au préalable) vient de s'écouler. Si monTimer.Interval = TimeSpan.FromSeconds(0.5), alors l'événement monTimer.Elapsed se déclenchera toutes les 500 millisecondes.

b. Les gestionnaires d'événements et abonnements

Il est possible d'abonner un ou plusieurs gestionnaires d'événements à l'événement d'un objet. On appelle gestionnaire d'événement le bout de code qui a pour but de s'exécuter au moment du déclenchement de l'événement au près duquel le gestionnaire a souscrit un abonnement. On peut ainsi attribuer à l'événement monBouton.Clicked le gestionnaire d'événement gestionMonBouton_clicked qui, lors de son exécution, imprime la phrase « Mon bouton a été cliqué » dans la console.

Il faut pour cela avoir exécuté la commande suivante au préalable. Elle a pour effet « d'abonner » le gestionnaire d'événement à l'événement :

```
monBouton.Clicked += gestionMonBouton_clicked;
```

Pour définir le gestionnaire d'événement :

```
private void gestionMonBouton_clicked(object sender, EventArgs e)
{
     Console.WriteLine("Mon bouton a été cliqué");
}
```

Le prototype d'un gestionnaire d'événement est donc généralement :

```
void monGestionnaire(object, EventArgs)
```

Par définition, toutes les classes héritent au moins de la classe Object, c'est le principe de la POO. Ainsi, quel que soit le type de l'objet sender, on sait qu'il est au moins du type Object. Comme le montre ce prototype, des informations sont passées automatiquement entre l'événement et le gestionnaire d'événement : il s'agit de l'objet sender (l'objet « envoyeur », donc l'objet dont l'événement a été déclenché) ainsi que la variable e de type EventArgs qui contient des informations parfois utiles sur les conditions de déclenchement de l'événement. Grâce à cette transmission automatique de données, on peut facilement accéder à certains éléments directement depuis le gestionnaire d'événement. L'objet monBouton est ainsi directement accessible dans le gestionnaire gestionMonBouton_clicked. Il est donc tout à fait envisageable de faire ce genre de choses :

Exemple de gestionnaire d'événement, qui lors du clic sur l'objet monBouton, a pour effet de changer la couleur de monBouton en rouge. Il faut pour cela « caster » le type de la variable sender en type Button (utilisation de la commande « as »). J'ai au passage rajouté un retour haptique dans ce gestionnaire d'événement, ce qui a pour effet de transmettre une petite vibration au doigt de l'utilisateur lors d'un clic sur monBouton :

```
private void gestionMonBouton_clicked(object sender, EventArgs e)
{
    (sender as Button).BackgroundColor = Colors.Red ;
    HapticFeedback.Default.Perform(HapticFeedbackType.Click);
    Console.WriteLine("Mon bouton a été cliqué. Il a donc changé de couleur");
}
```

Tout cela est bien joli mais les seuls événements à notre disposition sont ceux listés dans le paragraphe petit a. On peut vite se retrouver limité lorsque l'on a besoin de détecter des événements très spécifiques. Mais pas de panique ! Il est possible de créer ses propres événements.

c. La création d'événements personnalisés

La classe Button possède 3 événements (Clicked, Pressed et Released), pourtant aucun d'entre eux ne permet de détecter un appui prolongé sur un bouton… J'ai donc décidé d'implémenter moi-même cet événement à l'intérieur d'une nouvelle classe BoutonAppuiLong.

La classe BoutonAppuiLong hérite de la classe Button, elle possède donc toutes les caractéristiques et les comportements classiques d'un bouton. J'y ai cependant ajouté un timer « _timer » ainsi qu'un booléen « clicLongConfirmed ». J'y ai surtout ajouté deux nouveaux événements : ClicLong et ClicCourt. La durée nécessaire à un clic pour considérer qu'il soit long est personnalisable. J'ai abonné différentes méthodes aux différents événements (hérités de la classe Button) au sein même de la classe. Ainsi, pour tous les objets de type BoutonAppuiLong, il se passe des choses au sein de la classe lorsque les événements Clicked, Pressed ou Released sont déclenchés, même si le développeur n'a pas abonné de gestionnaire à ces événements.

Il y a deux scénarios :

- Déroulé du scénario « L'utilisateur effectue un clic court » :
    - Utilisateur touche l'écran
    - Evénement Pressed déclenché
        - La valeur false est affectée au booléen clicLongConfirmed
        - L'objet _timer d'intervalle tempsClicLong est démarré
    - L'utilisateur lève son doigt
    - Evénement Released déclenché
        - L'objet _timer est arrêté
    - Evénement Clicked déclenché
        - Comme le booléen clicLongConfirmed est toujours de valeur false, alors **l'événement ClicCourt est déclenché.**
- Déroulé du scénario « L'utilisateur effectue un clic long » :
    - Utilisateur touche l'écran
    - Evénement Pressed déclenché
        - La valeur false est affectée au booléen clicLongConfirmed
        - L'objet _timer d'intervalle tempsClicLong est démarré
    - Le temps s'écoule, l'utilisateur n'a toujours pas levé son doigt
    - tempsClicLong millisecondes se sont écoulées, l'événement _timer.Elapsed est donc déclenché
        - L'objet _timer est arrêté
        - La valeur True est affectée au booléen clicLongConfirmed
        - L'utilisateur ressent une vibration sous son doigt (retour haptique) ce qui confirme le fait qu'il a appuyé suffisamment longtemps pour que son clic soit considéré comme long.
    - Après un certain temps, l'utilisateur lève son doigt
    - Evénement Released déclenché
        - L'objet _timer est arrêté (même si ça ne sert à rien puisqu'il est déjà arrêté)

- Evénement Clicked déclenché
    - Comme le booléen clicLongConfirmed est de valeur True, alors **l'événement ClicLong est déclenché.**

Pour les objets de type BoutonAppuiLong, il est déconseillé de s'abonner à d'autres événements que ClicLong et ClicCourt. Vous trouverez l'intégralité du code permettant de définir cette classe en Annexes (Annexe 2). Au niveau de l'organisation des fichiers du projet, la classe BoutonAppuiLong est définie dans le fichier ClassesHeritees/BoutonAppuiLong.cs

D. LES CLASSES STATIQUES

Une classe statique, également appelée classe utilitaire ou classe de méthodes statiques, est une classe dans la programmation orientée objet qui ne peut pas être instanciée et dont toutes les méthodes et les membres sont déclarés comme étant statiques.

Les méthodes statiques sont associées à la classe elle-même plutôt qu'à une instance spécifique de la classe. Elles peuvent être appelées directement à partir de la classe sans avoir besoin de créer une instance de celle-ci. Par exemple, si vous avez une classe statique appelée « MathUtils » avec une méthode statique appelée « add » de prototype « int add(int, int) » permettant d'additionner deux entiers, vous pouvez l'appeler directement en utilisant la syntaxe "MathUtils.add(5, 3)" plutôt que de devoir créer une instance de la classe et appeler la méthode à partir de l'instance.

Les classes statiques sont souvent utilisées pour regrouper des méthodes et des fonctionnalités qui sont liées à une classe spécifique, mais qui n'ont pas besoin d'état ou de données propres à une instance particulière. Elles sont couramment utilisées pour implémenter des fonctions utilitaires qui effectuent des calculs mathématiques, des opérations de conversion de données ou d'autres tâches similaires.

Il est important de noter que les classes statiques ne peuvent pas être héritées, car elles ne peuvent pas être instanciées. Elles sont généralement déclarées avec un constructeur privé pour éviter toute tentative d'instanciation.

Dans mon projet, le dossier ClassesStatiques (voir Figure 21) contient 3 fichiers définissant des classes statiques.

- ConfigData.cs : La classe ConfigData est utilisée pour stocker différentes données de configuration exploitées dans différentes parties de l'application. J'y stocke par exemple l'unité dans laquelle les mesures de la page de travail sont exprimées (mètres, feets ou kilomètres). Dans mon cas, ces données de configuration sont modifiables par l'utilisateur via la page de configuration de l'application. L'avantage ici d'utiliser une classe statique est que les valeurs de configuration peuvent être facilement accessibles depuis n'importe quelle partie de l'application en utilisant la syntaxe ConfigData.nomDeMaVariable.

- FonctionsUtils.cs : La classe FonctionsUtils est utilisée pour regrouper des fonctions utilitaires qui ne nécessitent pas d'état spécifique d'une instance de classe. Ce sont majoritairement des fonctions permettant d'effectuer des calculs pour l'affichage du

graphique de la page de travail, calculs utilisant beaucoup de notions de géométrie comme la trigonométrie ou encore le Théorème de Pythagore. L'avantage d'utiliser une classe statique pour cela est que ces méthodes sont directement appelables en utilisant la syntaxe FonctionsUtils.NomDeLaMethode() sans avoir besoin d'instancier la classe. Voici un exemple de méthode contenu dans la classe FonctionsUtils :

```
public static float calculDistancePoints(PointF pointA, PointF pointB)
{
    float deltaX = pointB.X - pointA.X;
    float deltaY = pointB.Y - pointA.Y;
    float hypothenuse = (float)Math.Sqrt(deltaX * deltaX + deltaY * deltaY);

    return hypothenuse;
}
```

- ConvertUnit.cs : J'ai récupéré le code de la classe ConvertUnit en Open-Source sur Internet. Elle fournit des méthodes de conversion entre différentes unités de mesure ou de données. Voici quelques exemples de méthodes fournies par cette classe que j'utilise dans EasyHover :

```
const double metersToFeet = 3.28084;
const double feetToMeters = 0.3048;
const double kilometersToFeet = 3280.84;
const double feetToKilometers = 0.0003048;

public static double MetersToFeet(double distance) =>
distance * metersToFeet;
public static double FeetToMeters(double distance) =>
distance * feetToMeters;
public static double KilometersToFeet(double distance) =>
distance * kilometersToFeet;
public static double FeetToKilometers(double distance) =>
distance * feetToKilometers;
```

E. L'IMPLEMENTATION DES PAGES BASIQUES

1. La page d'accueil

La page d'accueil est composée d'un label et quatre boutons. Chacun des quatre boutons mènent vers une page spécifique :

- Le bouton « stationnaire sol » mène vers la page de travail en mode stationnaire sol
- Le bouton « stationnaire air » mène vers la page de travail en mode stationnaire air
- Le bouton « configuration » mène vers la page de configuration permettant de configurer l'IHM de la page de travail
- Le bouton « tutoriel » mène vers la page de tutoriel. Cette page fournit une présentation de EasyHover et explique en détail comment utiliser l'application.

Les deux boutons stationnaire sol et stationnaire air mènent sur la même page, mais en changeant un paramètre statique de la classe statique ConfigData :

```
private async void boutonStationnaire_clicked(object sender, EventArgs e)
{
    bool boolMode= (sender as Button).Text == "stationnaire sol" ? True : false;
    ConfigData.trueStationnaireSol_falseStationnaireAir = boolMode ;
    await Shell.Current.GoToAsync("PageTravailSolOuAir");
}
```

La commande commençant par « await » est la commande permettant de changer de page. Ainsi, une fois arrivé sur la page de travail, le programme sait s'il est en mode stationnaire sol ou air en consultant la variable statique « ConfigData.trueStationnaireSol_falseStationnaireAir ».

Voici Figure 27 une capture d'écran de l'état actuel de la page d'accueil, accompagnée d'un schéma mettant en évidence sa disposition. On voit que le layout racine est un StackLayout (encadré en vert), de type VerticalStackLayout. Il possède un padding de 30, ce qui signifie que ses enfants (encadrés en rouge) ne peuvent pas être affichés à moins de 30 unités de ses « parois » (zone « no man's Land » barbouillée en vert). Sa propriété « VerticalOptions » est réglée à « Start » ce qui signifie que ses enfants doivent s'entasser du bas vers le haut, en restant le plus haut possible (flèches vertes). Le premier enfant est un élément de type Label. Il possède un margin de 50, ce qui signifie qu'il s'efforce de mettre une distance de 50 unités entre lui et les autres éléments, y compris les parois de son parent (zone « espace vital » barbouillée en rouge dans le premier encadré rouge). Les deux autres enfants possèdent aussi cet espace vital mais cette fois de seulement 30 unités (zones barbouillées en rouge dans les deux encadrés rouge suivants). Les deux enfants suivants sont d'ailleurs eux aussi des StackLayout, mais cette fois de type HorizontalStackLayout. Leur propriété « HorizontalOptions » est réglée à « Center », ce qui signifie que leurs enfants doivent s'entasser de la gauche vers la droite, tout en restant les plus centrés possible (double-flèche orange). C'est ainsi que les enfants de ces layouts (enfants encadrés en orange) sont disposés. Ces enfants sont les boutons permettant la navigation dans EasyHover.

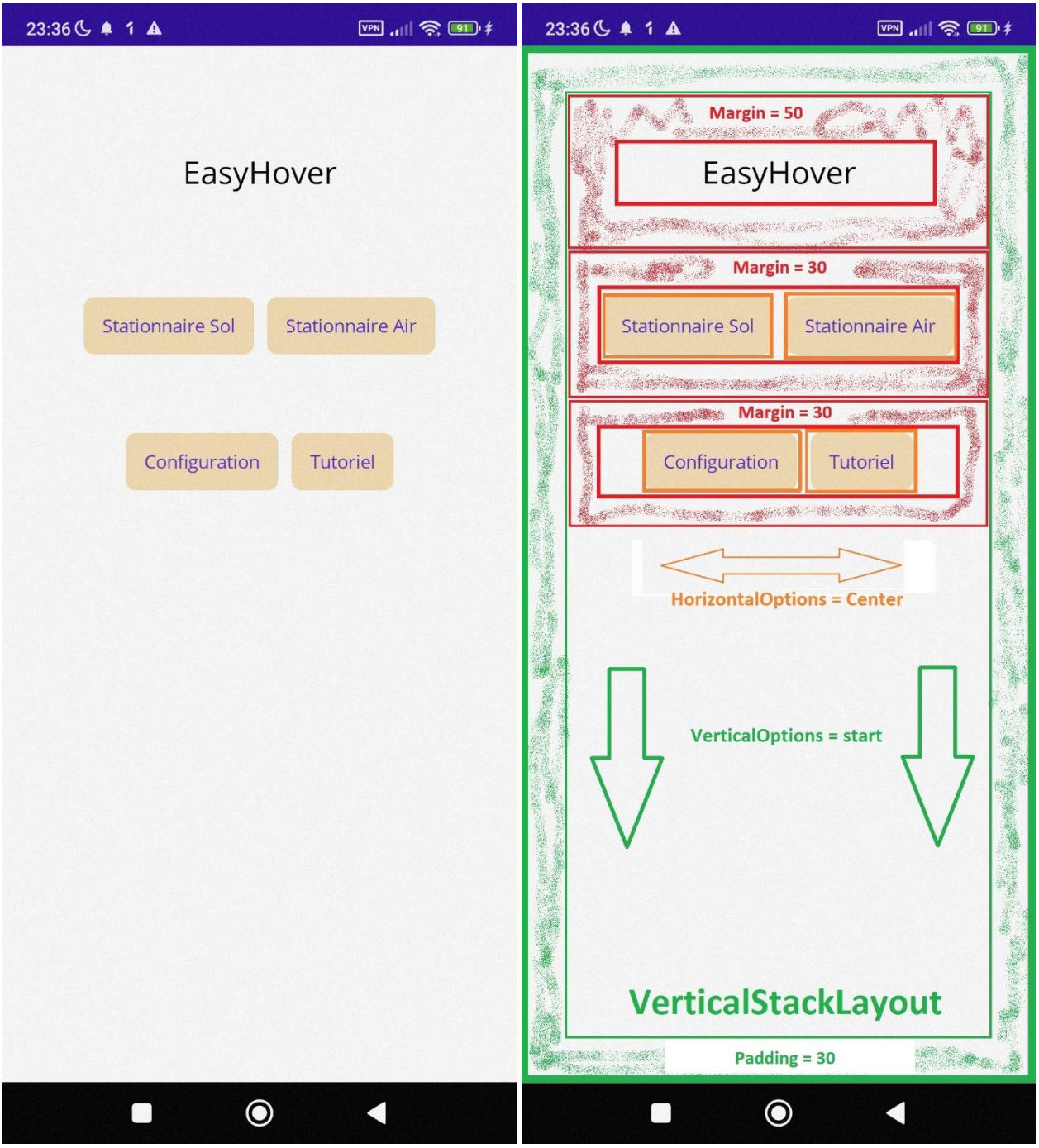


Figure 27 : Capture d'écran de l'état actuel de la page d'accueil de EasyHover, accompagnée d'un schéma mettant en évidence sa disposition

Voici le code XAML ayant servi à coder la mise en page de la page d'accueil :

```
<VerticalStackLayout BackgroundColor="White" Padding="30">

    <Label TextColor="Black"
        FontSize="Title"
        Margin="50"
        Text="EasyHover"
        VerticalOptions="Center"
```

```
        HorizontalOptions="Center" />

    <HorizontalStackLayout Margin="30"
        HorizontalOptions="Center"
        Spacing="10">

            <Button Text="Stationnaire Sol"
                    BackgroundColor="Wheat"
                    Clicked="stationnaireSol_clicked"/>
            <Button Text="Stationnaire Air"
                    BackgroundColor="Wheat"
                    Clicked="stationnaireAir_clicked"/>

    </HorizontalStackLayout>

    <HorizontalStackLayout Margin="30"
        HorizontalOptions="Center"
        Spacing="10">

            <Button Text="Configuration"
                    BackgroundColor="Wheat"
                    Clicked="configuration_clicked"/>
            <Button Text="Tutoriel"
                    BackgroundColor="Wheat"
                    Clicked="tutoriel_clicked"/>

    </HorizontalStackLayout>

</VerticalStackLayout>
```

2. La page tutoriel

Je n'ai pour l'instant pas encore eu le temps de réaliser le contenu de la page tutoriel. Elle est donc actuellement vide, à l'exception d'un simple bouton retour permettant de retourner sur la page d'accueil.

3. La page de configuration

Il n'y a pour l'instant qu'un seul paramètre configurable, l'unité dans laquelle les mesures de la page de travail sont exprimées (mètres, feets ou kilomètres). Ce paramètre est stocké dans la variable statique « ConfigData.paramDistanceUnit_0m_1ft_2km ». L'utilisateur a le choix entre trois boutons « Meters », « Feets » et « Kilometers ». Le bouton représentant l'unité actuellement configurée est de couleur rouge. Dès que l'utilisateur clique sur un autre bouton, le paramètre se met à jour et la couleur rouge se déplace au bouton fraîchement cliqué. L'utilisateur peut alors retourner à la page de travail (en retournant d'abord sur la page d'accueil) et constater que les mesures sont exprimées à l'aide de l'unité qu'i vient de sélectionner.

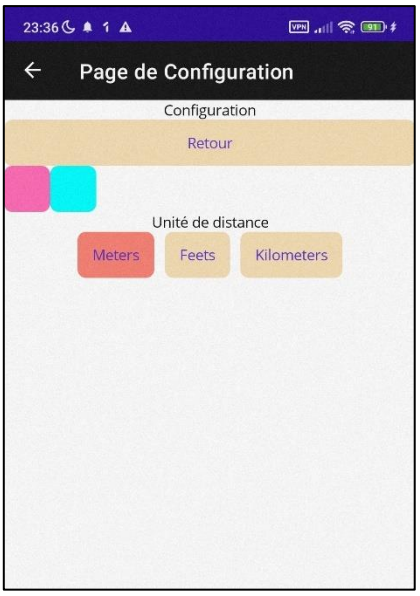


Figure 28 : Capture d'écran de l'état actuel de la page de configuration. La valeur "Meters" est actuellement sélectionnée.

Pour la suite du développement, il sera très simple de reprendre ce système pour assurer la configuration de nouveaux paramètres.

F. L'IMPLEMENTATION DE LA PAGE DE TRAVAIL

1. Les différents éléments et leur disposition

La Figure 29 est une capture d'écran de l'état actuel de la page de travail. Soit la longueur selon l'axe gauche-droite de la photo et soit la largeur selon l'axe haut-bas de la photo. Vu que cette figure contient beaucoup d'informations écrites en tout petit, vous retrouverez un gros plan de la partie droite de cette figure en Annexes (Annexe 3) : Figure 32.

On y retrouve les 4 principaux éléments visuels cités précédemment dans le rapport dont la disposition a été assurée par deux grilles (classe Grid). La grille verte n'ayant pas de parent, c'est donc le conteneur origine de cette page de l'application. Elle possède 2 lignes et 0 colonnes et elle est paramétrée de telle sorte que la deuxième ligne soit exactement 2 fois plus large que la première. Ainsi, quel que soit l'écran, la première ligne occupera 33% de sa hauteur et la deuxième le reste (donc 66% de la hauteur de l'écran). J'ai fait cela pour que, pour tout écran de format supérieur ou égal à 3:2 (ici mon écran est de format 20:9), la deuxième ligne soit plus large que longue. Ainsi, le graphique, de forme carrée, pourra maximiser son apparence sans dépasser sur le bas de l'écran.

La première ligne de la grille verte a deux enfants : une BoxView de couleur rouge en fond et la grille jaune. La grille jaune possède 6 lignes et 6 colonnes sans instructions de dimensions. Les 36 cellules résultantes ont donc toutes les mêmes dimensions. Nous repèrerons les cellules avec leurs coordonnées, l'origine étant le coin le plus en haut à gauche de la grille. La cellule la plus en haut à

gauche est donc de coordonnées (0,0) et la cellule contenant le label « manuel » est de coordonnées (5,5). La cellule la plus en bas à gauche de la grille est de coordonnées (0,5).

Les petits encadrés noirs signalent des regroupements de cellule, résultants ici toujours en grosses cellules de 2 de long et 2 de large (respectivement ColumnSpan et RowSpan). Nous repèrerons ces grosses cellules par la coordonnée de leur cellule la plus en haut à gauche. Ainsi la grosse cellule contenant le bouton « CHRONO » est de coordonnées (0,4). Ainsi, on peut distinguer les boutons « Retour » (←), « Coords » et « CHRONO » qui ont été centrés dans les grosses cellules respectivement (0,0), (0,2) et (0,4). Idem pour les labels « Tracking » et « Cap » qui ont eux aussi été centrés dans les cellules (3,2) et (3,4).

Dans les cellules (5,2), (5,3), (5,4) et (5,5) ont été rangés, dans l'ordre, un premier switch (switch_1), le label « Off », un deuxième switch (switch_2) et enfin le label « Manuel ». Les BoxViews de couleur rose situées sur toutes les cellules de la diagonale de la grille jaune aident justement les développeurs à visualiser la grille, il est bien entendu prévu de les enlever dans la version finale de EasyHover.

La deuxième ligne de la grille verte possède certes moins d'enfants, mais elle n'en est pas moins complexe… On retrouve en fond une BoxView de couleur bleu. Par-dessus se trouve un canevas, servant à dessiner le graphique.

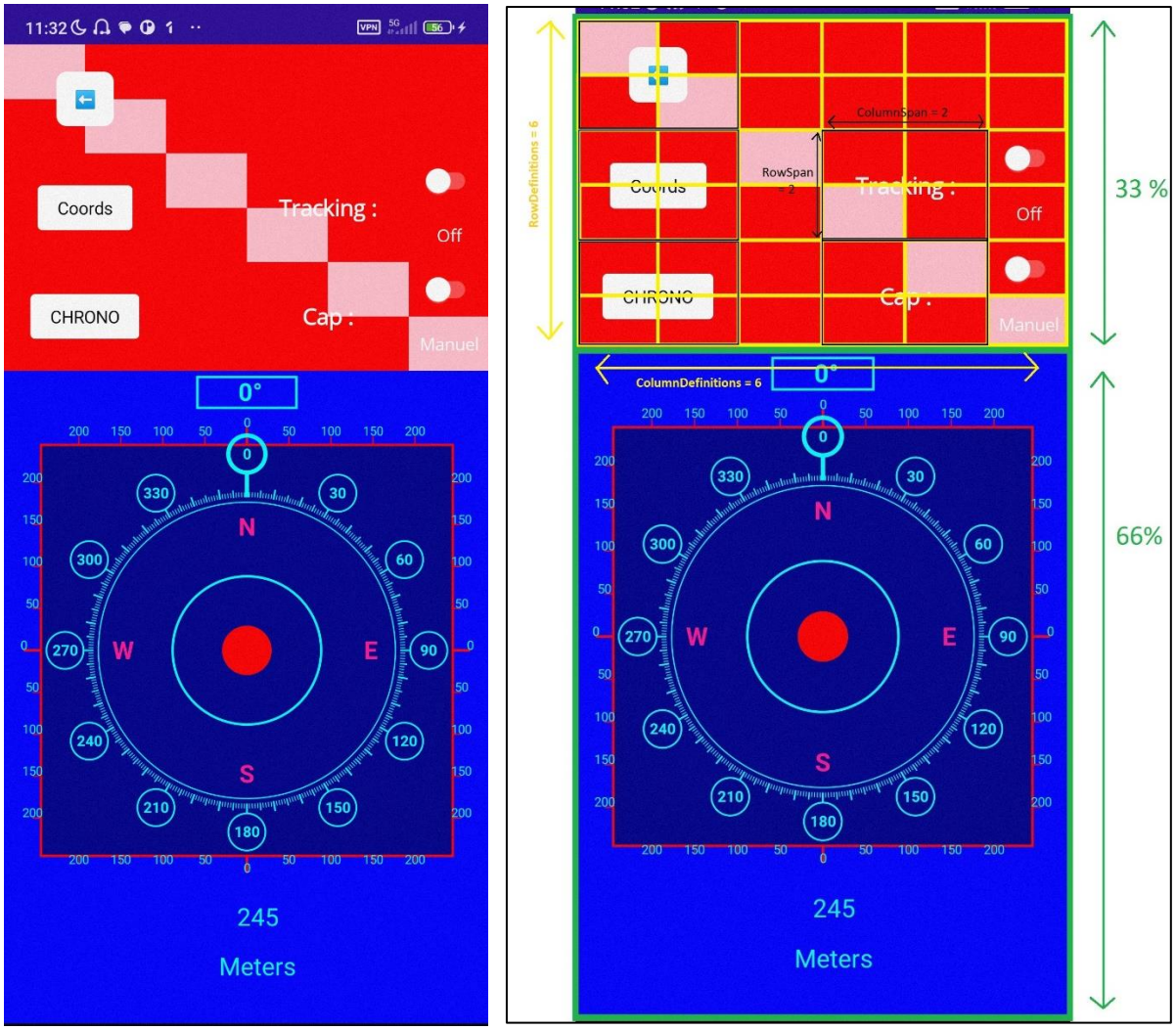


Figure 29 : Capture d'écran de la page de travail à gauche ; mise en évidence de la disposition des éléments visuels à droite

Voici le code C# permettant de déclarer et d'initialiser les grilles verte (grillePageDeTravail) et jaune (grilleHaut) :

```
var grillePageDeTravail = new Grid
{
    RowDefinitions =
    {
        new RowDefinition{Height = new GridLength(1, GridUnitType.Star)},
        new RowDefinition{Height = new GridLength(2, GridUnitType.Star)}
    }
};

var grilleHaut = new Grid
{
    ColumnSpacing = 0,
    RowSpacing = 0,
    RowDefinitions =
    {
        new RowDefinition{Height = new GridLength(1, GridUnitType.Star)},
        new RowDefinition{Height = new GridLength(1, GridUnitType.Star)},
        new RowDefinition{Height = new GridLength(1, GridUnitType.Star)},
```

```
            new RowDefinition{Height = new GridLength(1, GridUnitType.Star)},
            new RowDefinition{Height = new GridLength(1, GridUnitType.Star)},
            new RowDefinition{Height = new GridLength(1, GridUnitType.Star)}
        },

        ColumnDefinitions =
        {
            new ColumnDefinition{Width = new GridLength(1, GridUnitType.Star)},
            new ColumnDefinition{Width = new GridLength(1, GridUnitType.Star)},
            new ColumnDefinition{Width = new GridLength(1, GridUnitType.Star)},
            new ColumnDefinition{Width = new GridLength(1, GridUnitType.Star)},
            new ColumnDefinition{Width = new GridLength(1, GridUnitType.Star)},
            new ColumnDefinition{Width = new GridLength(1, GridUnitType.Star)}
        }
    }
```

Voici le code C# permettant de remplir les grosses cellules après les avoir fusionnées :

```
grilleHaut.Add(boutonRetour, 0, 0);
grilleHaut.SetColumnSpan(boutonRetour, 2);
grilleHaut.SetRowSpan(boutonRetour, 2);

grilleHaut.Add(boutonChrono, 0, 4);
grilleHaut.SetColumnSpan(boutonChrono, 2);
grilleHaut.SetRowSpan(boutonChrono, 2);

grilleHaut.Add(boutonCoords, 0, 2);
grilleHaut.SetColumnSpan(boutonCoords, 2);
grilleHaut.SetRowSpan(boutonCoords, 2);

grilleHaut.Add(lblTracking, 3, 2);
grilleHaut.SetColumnSpan(lblTracking, 2);
grilleHaut.SetRowSpan(lblTracking, 2);

grilleHaut.Add(lblCap, 3, 4);
grilleHaut.SetColumnSpan(lblCap, 2);
grilleHaut.SetRowSpan(lblCap, 2);
```

Voici le code C# permettant d'ajouter la diagonale de BoxViews roses ainsi que le contenu des petites cellules à droite de la grille jaune.

```
for (int i = 0; i < 6; i++)
{
    grilleHaut.Add(new BoxView { Color = Colors.Pink }, i, i);
}

grilleHaut.Add(switchGeoTrackingOnOff, 5, 2);
grilleHaut.Add(lblTrackingOnOff, 5, 3);
grilleHaut.Add(switchModeCapTrueAuto_falseManuel, 5, 4);
grilleHaut.Add(lblCapManuelOuAuto, 5, 5);
```

Enfin, voici le code C# permettant de remplir la grille verte et d'ajouter cette grille verte ainsi que son contenu à l'affichage de la page :

```
grillePageDeTravail.Add(boxHaut, 0, 0);
```

```
grillePageDeTravail.Add(grilleHaut, 0, 0);

grillePageDeTravail.Add(boxBas, 0, 1);
grillePageDeTravail.Add(monCanevas, 0, 1);

Content = grillePageDeTravail ;
```

Bien sûr, tout ceci aurait pu être écrit avec des balises XAML, mais vu que j'ai décidé de me passer de ce langage pour l'implémentation de la page de travail, j'ai tout fait en C#.

2. Le chronomètre

Le bouton « CHRONO » est en fait une instance de la classe BoutonAppuiLong. Cet objet fait donc la différence entre un clic court et un clic long lors de la détection d'un clic. Comme on peut le voir dans le paragraphe précédent, je le manipule à l'aide de la variable « boutonChrono ».

Pour le fonctionnement de la fonction chronomètre, j'utilise également l'objet « Stopwatch » de la classe System.Diagnostics.Stopawatch et l'objet « timerMajChrono » de la classe System.Timers.Timer. La Stopwatch permet de mesurer l'écoulement du temps et le Timer (dont j'ai réglé l'intervalle à 1 seconde) me permet de mettre régulièrement l'affichage du chrono à jour afin de voir les secondes s'égrener.

Les différentes méthodes de la Stopwatch :

- Stopwatch.Start() : permet de démarrer la mesure de l'écoulement du temps
- Stopwatch.Stop() : permet de mettre en pause la mesure de l'écoulement du temps
- Stopwatch.Restart() : permet de remettre le compteur à zéro tout en continuant à mesurer
- Stopwatch.Reset() : arrête la mesure du temps et remet le compteur à zéro

Le temps écoulé depuis le démarrage du compteur est accessible en lecture avec la propriété Stopwatch.Elapsed. Cette propriété délivre une variable de type TimeSpan.

Le chronomètre possède trois états : initial, isRunning et Stopped. A l'état Initial, on exécute Stopwatch.Reset(), à l'état isRunning, on exécute Stopwatch.Start() et à l'état Stopped, on exécute Stopwatch.Stop(). Dès que l'état est différent de Initial, le timer timerMajChrono est lancé et permet de rafraîchir l'affichage du chrono à chaque seconde. Le bouton boutonChrono, de type BoutonAppuiLong fait la différence entre un clic long et un clic court de l'utilisateur. Voici donc Figure 30 un diagramme UML décrivant son fonctionnement.

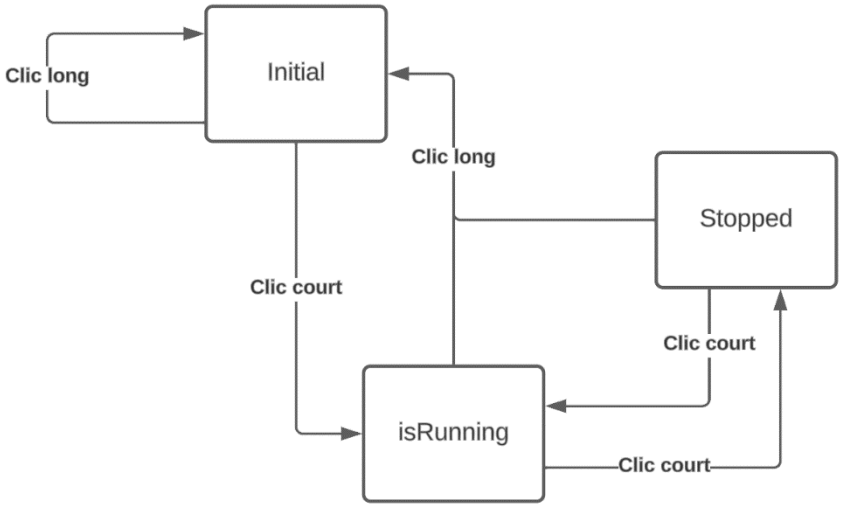


Figure 30 : Diagramme UML du fonctionnement du chronomètre

3. Le clavier tactile

Le clavier tactile est mis à disposition par la page de travail pour permettre à l'utilisateur de rentrer des consignes comme le cap, les coordonnées de référence, la force du vent ou encore sa direction. Le clavier tactile n'est pas une classe distincte, il apparaît grâce à la méthode faireClavierTactile() de la classe PageDeTravail qui s'occupe de le construire et de l'afficher. Pour cela, cette méthode crée un nouveau layout racine « grilleClavierTactile » (de type Grid), le remplit puis exécute la commande « Content = grilleClavierTactile » ce qui a pour effet « d'écraser » l'affichage de la page de travail. Ceci n'aurait pas été possible avec un affichage assuré par du code XAML. C'est donc une des raisons pourquoi j'ai décidé de coder l'intégralité de la classe PageDeTravail en C#.

Le clavier tactile est composé de différents boutons, tous rangés dans les différentes cellules de la grille racine (voir Figure 31). L'utilisateur peut ainsi rentrer sa consigne et appuyer sur le bouton « OK ». Si la consigne est conforme et ne contient pas d'erreur de syntaxe, alors la méthode faireLaPageDeTravail() s'exécute et construit à nouveau l'affichage de la page de travail, tout en conservant l'état qu'elle avait au moment de sa disparition. Si le chronomètre était en mode « isRunning », alors il l'est toujours et les secondes avaient bien continuées à s'égrener normalement pendant que l'utilisateur utilisait le clavier tactile.

Afin d'assurer la vérification de la bonne syntaxe des consignes entrées par l'utilisateur, des expressions régulières sont chargées de les valider ou non. Ses expressions régulières ne sont pas encore implémentées à l'heure où j'écris ces mots.

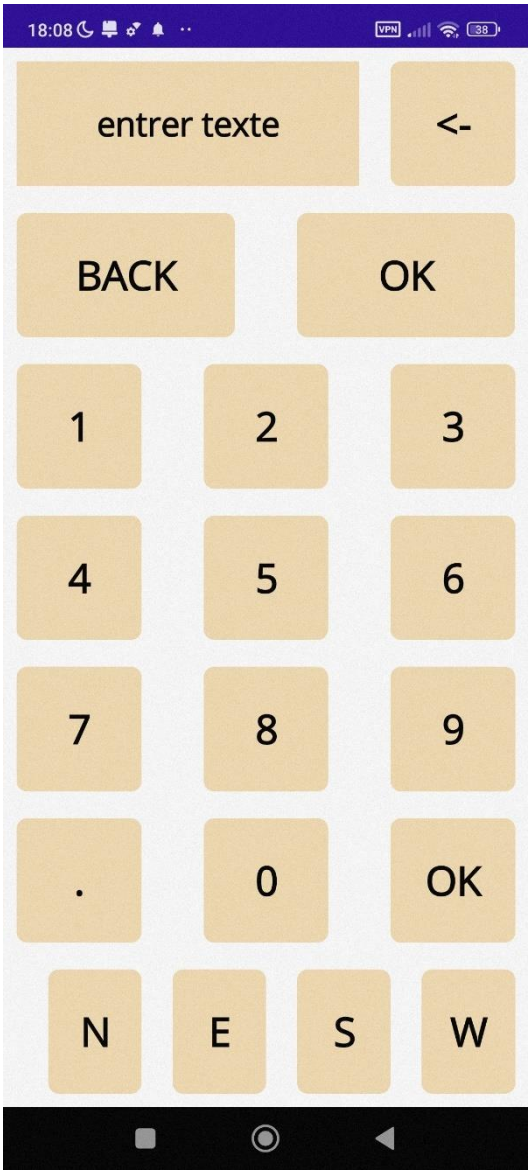


Figure 31 : Capture du clavier tactile proposé par la page de travail de EasyHover

G. La geolocalisation

a. La programmation asynchrone

La programmation asynchrone permet d'écrire du code qui peut être exécuté de manière non bloquante, ce qui améliore la réactivité et les performances des applications. Cela permet notamment d'éviter les « freezes » comme il peut parfois y en avoir lors d'une requête un peu trop complexe dans les vieilles applications. L'application reste réactive et ne bloque pas l'interface utilisateur pendant les opérations longues.

L'asynchronisme est utilisé lorsque des opérations prennent du temps, comme les appels réseau, les accès à la base de données ou les calculs longs. Au lieu de bloquer l'exécution du programme jusqu'à ce que ces opérations se terminent, on peut les exécuter de manière asynchrone pour permettre à l'application de continuer à fonctionner et à répondre aux autres interactions de l'utilisateur.

En .Net MAUI, il est possible d'utiliser le mot clé async pour marquer une méthode comme étant asynchrone. Le type Task ou Task<T> peut également être utilisé pour représenter une opération asynchrone qui retourne une valeur. Par exemple, si on a une méthode DoSomethingAsync qui effectue une opération longue, on peut la déclarer comme suit :

```
async Task<int> DoSomethingAsync()
{
    // L'opération longue est effectuée ici
    // Le mot clé "await" est utilisé pour attendre les résultats asynchrones

    // Un résultat est retourné
    return result;
}
```

Lorsqu'on appelle une méthode asynchrone, il faut utiliser le mot clé await pour attendre la fin de l'exécution de cette méthode sans bloquer le thread principal de l'application :

```
async Task RunAsync()
{
    // Une méthode asynchrone est appelée
    int result = await DoSomethingAsync();

    // On continue avec le reste du code après que l'opération asynchrone est
      terminée
    Console.WriteLine("Le résultat est : " + result);
}
```

b. Ma boucle de géolocalisation

Pour la récupération des données de géolocalisation, j'utilise la méthode « Geolocation.Default.GetLocationAsync() » qui retourne une variable de type « Location » contenant une latitude ainsi qu'une longitude. Cette méthode utilise les capteurs du téléphone, ces capteurs utilisent le réseau GPS afin de déterminer la position de l'appareil. Il doit pour cela avoir une constellation minimale de satellites et effectuer beaucoup de calculs de triangulation. Cette opération peut donc prendre plus ou moins de temps en fonction de la qualité de la constellation.

Afin de garantir une exécution fluide de l'application malgré ce genre d'opération, j'ai utilisé la programmation asynchrone. J'ai donc implémenté deux fonctions permettant la récupération des données de géolocalisation de manière cyclique et asynchrone : GetCurrentLocation() et lancerGeoTracking(). La première fournit les données de géolocalisation et la deuxième contient une boucle appelant GetCurrentLocation() de façon cyclique, tant que la surveillance de la

géolocalisation est activée. Ainsi, aussitôt que la méthode GetCurrentLocation() réussi à géolocaliser l'appareil, elle se retrouve instantanément chargée de recommencer et ainsi de suite.

Voici le code extrêmement simplifié de ces deux méthodes, on peut y noter l'utilisation des mots-clés « await », « async » ou encore « Task » (voir ci-dessus) :

```
    private async Task GetCurrentLocation()
    {
        GeolocationRequest request = new GeolocationRequest(GeolocationAccuracy.Best,
TimeSpan.FromSeconds(10));
        Location location = await Geolocation.Default.GetLocationAsync(request,
_cancelTokenSource.Token);
    }

    private async void lancerGeoTracking()
    {
        surveillanceGeolocalisation = true;
        while (surveillanceGeolocalisation)
        {
            await GetCurrentLocation();
        }
    }
```

Le booléen « surveillanceGeolocalisation » est une variable globale, accessible à n'importe quel endroit de la classe PageDeTravail. Ainsi, pour stopper la boucle lancée par lancerGeoTracking(), il suffit d'affecter la valeur False à ce booléen et la boucle n'effectuera pas d'itération supplémentaire.

V. Résultats, avenir du projet et discussion

A. Etat actuel

L'application n'est pas encore achevée mais presque complètement fonctionnelle, du moins pour ce qui est du stationnaire sol. Les principaux points manquants pour considérer l'application totalement utilisable pour un stationnaire sol sont :

- Les expressions régulières permettant de valider ou non les entrées de consigne de l'utilisateur.
- Corriger le calcul actuel de l'azimut (direction à prendre pour rejoindre le point de référence).
- Quelques petits réglages concernant les boucles de géolocalisation.

Le mode stationnaire air n'est en revanche pas encore implémenté, tout comme la page tutoriel, La charte graphique n'est encore harmonisée ni esthétique et il reste quelques petites modifications à faire sur la page de travail comme, par exemple, l'affichage des barres de tendance en pointillés sous certaines conditions.

L'état d'avancement actuel du développement est constatable grâce à certaines figures de la partie précédente (Figure 27, Figure 28, Figure 29 et Figure 31).

B. Revision des objectifs

Durant le stage, les objectifs fixés dans la partie Objectifs pour ce stage se sont avérés intenables dans le temps imparti : une fois ma soutenance passée, il me restera exactement 9 jours de stage. Cependant, il est toujours possible de se fixer de nouveaux objectifs à atteindre !

Il est désormais trop ambitieux d'envisager des campagnes d'essai et une mise à disposition sur le Play Store avant la fin de mon stage. J'aurai aimé terminer au moins la phase de développement afin que la personne qui reprendra le projet (si cela se produit) puisse directement passer à la phase d'essais. Mais il reste le mode stationnaire air et la page de tutoriel à coder, deux grosses étapes du développement ! De plus, même pendant la phase d'essais, des développements seront nécessaires pour corriger les bugs et améliorer les fonctionnalités, etc. La personne devra inévitablement se plonger dans le code.

Par conséquent, je dois rendre le code aussi clair que possible grâce à des nettoyages, des refactorisations*, des commentaires et des fichiers README En effet, malgré mes efforts pour donner des noms de variable explicites, respecter les conventions de mise en page ou encore organiser un minimum mon code, mes commentaires sont rares et il reste des bouts de code sauvages (ayant eu une utilité mais désormais inexploités…). Mon objectif est de garantir une passation fluide. J'appellerai cette phase « le grand nettoyage de printemps ».

C. Feuille de route pour la suite

1. Pour la fin du stage

Fort de cette redéfinition des objectifs, j'ai décidé de rédiger un nouveau planning afin d'organiser mes derniers jours de travail. Ce planning est sous la forme d'une liste de tâches, dont les points sont triés par ordre de priorité et accompagnés d'une durée prévisionnelle :

- Finir le mode stationnaire sol (Total : 3 jours)
    - Corriger le calcul actuel de l'azimut (0.5 jour)
    - Réviser la boucle de géolocalisation (1 jour)
    - Implémenter les expressions régulières et dernières petites retouches sur le clavier tactile (1 jour)
    - Petites retouches sur la page de travail et le graphique (0.5 jour)
- Effectuer le grand nettoyage de printemps (Total : 3 jours)
    - Nettoyage + commentaires + refactorisation (2 jours)
    - Ecrire un gros README à l'attention des développeurs donnant les grandes lignes de l'organisation et de l'utilité des différentes parties du code. J'y écrirai également quelques remarques et indications pertinentes, dans le but de rendre la passation la plus fluide possible (0.5 jour)
    - Ecrire un autre gros README à l'attention des utilisateurs faisant office de tutoriel temporaire (0.5 jour)

A ce stade-là, les nouveaux objectifs seront atteints et il restera normalement 3 jours avant d'arriver au terme du stage. Je considère ces 3 jours comme une marge de sécurité au cas où les tâches ci-dessus prendraient plus de temps que prévu. Je redoute tout particulièrement la première phase du nettoyage de printemps car on ne sait jamais sur quel genre de surprise on peut tomber, au détour d'un vieux couloir sombre…

2. Pour la suite

Voici les principales tâches à effectuer après la fin de mon stage pour mener le projet EasyHover à bout :

- Faire confirmer le besoin du mode stationnaire air. Si confirmé, alors en effectuer l'implémentation, sinon passer directement à la tâche suivante (Total : 7 jours)
    - Retravailler la conception du mode et faire valider par les pilotes. (2 jours)
    - Implémenter les méthodes orchestrant le bon fonctionnement de ce mode (2 jours)

    - Modifier l'affichage de la page de travail et du graphique en conséquence (3 jours)

- Mener une campagne d'essais avec les pilotes (Total : 10 jours)
    - Essais au sol + correctifs et débogages en conséquence et ainsi de suite (4 jours)
    - Essais en vol+ correctifs et débogages en conséquence et ainsi de suite (6 jours)
- Si la campagne d'essais est concluante, alors effectuer les dernières étapes avant la livraison. Ne m'y connaissant pas assez dans ce domaine, je ne peux pas prévoir les durées de façon satisfaisante :
    - Effectuer l'implémentation de la page de tutoriel. Le client décidera du format, du niveau de détails requis ainsi que du soin apporté à la réalisation du tutoriel. (Entre 1 et 10 jours en fonction du client)
    - Appliquer la charte graphique du MINARM, harmoniser l'intégralité de l'application et soigner tous les détails esthétiques (1 jour)
    - Faire valider la version finale par tous les acteurs du projet, effectuer les correctifs en fonction des retours
    - Mener une dernière campagne d'essais avec des testeurs et des pilotes afin de valider l'intégralité de l'application
    - Effectuer la livraison sur le Play Store (compilation, descriptifs, crédits etc.)
- Si le besoin d'une version iOS est confirmé, alors effectuer le portage et effectuer la livraison sur l'App Store.

D. RSE

Depuis quelques années et dans le contexte actuel de changement climatique, la responsabilité sociétale et environnementale (RSE*) est devenue un enjeu croissant pour les entreprises en France. En conséquence, on observe l'émergence d'initiatives intéressantes et même quelques progrès encourageants ! Mais c'était sans compter le contexte géopolitique mondial : les nombreuses tensions conduisent les acteurs étatiques à renforcer leurs forces militaires, dans une démarche à la fois dissuasive et défensive afin d'assurer leur souveraineté. Mais est-il seulement possible de mener un conflit armé de façon écologique et durable ?

La puissance militaire d'un pays est souvent définie par ses infrastructures et ses machines, très gourmandes en énergie. J'ai d'ailleurs pu le constater par moi-même : mon stage se déroulant dans la base aérienne de Cazaux, nous assistions tous les jours au décollage d'avions de chasse, laissant une épaisse traînée noire à la sortie de leurs réacteurs…

La source d'énergie privilégiée des armées étant du carburant dérivé de sources fossiles, la guerre implique donc des émissions importantes de gaz à effet de serre et contribue ainsi au changement

climatique. En effet, les énergies fossiles sont des énergies très concentrées : on peut tirer beaucoup d'énergie d'un litre de carburant. En parallèle, les énergies renouvelables sont généralement diffuses : il faut installer des infrastructures pour réussir à les capter en quantité suffisante. D'un point de vue logistique, les camions-citernes remplis de carburant sont une source d'énergie beaucoup plus maniable et sûre qu'un parc mobile de panneaux solaires dépendants des conditions météorologiques !

Savoir s'il est possible de mener une guerre de manière écologique et durable est complexe. Les activités militaires ont un impact significatif sur l'environnement, notamment par la destruction d'infrastructures, la pollution des ressources naturelles et la perturbation des écosystèmes. De plus, la production d'armes et la logistique militaire impliquent souvent une consommation importante de ressources naturelles et d'énergie. Certaines mesures pourraient atténuer ces effets néfastes. Par exemple, pour réduire la consommation d'énergie et les émissions de carbone associées aux opérations militaires, on pourrait alimenter les bases militaires et les véhicules avec des sources d'énergie renouvelable ou encore mettre en œuvre de pratiques d'écoconception pour réduire l'empreinte environnementale des équipements et des infrastructures.

Néanmoins, les conflits armés entraînent des conséquences dévastatrices sur les populations civiles, les droits de l'homme et la stabilité sociale, ils vont à l'encontre des principes fondamentaux de la RSE. L'impact humain et social des conflits est tout aussi crucial à prendre en compte que l'impact environnemental. La recherche de solutions pour minimiser ces impacts reste donc un défi complexe et controversé, car il est difficile de concilier les exigences d'une guerre avec les principes de durabilité environnementale et de responsabilité sociétale. S'il est essentiel de continuer à promouvoir des pratiques plus durables, même dans le contexte militaire, la solution écologique la plus efficace est de prévenir les conflits, les déploiements militaires à grande échelle et les courses à l'armement. L'idéal serait donc un contexte géopolitique apaisé.

Conclusion

Mon expérience de stage au sein de la DGA EV a été enrichissante à bien des égards. Mon projet consistait à développer une application mobile visant à assister les pilotes d'hélicoptère lors de vols stationnaires. Pour l'instant, malgré 48 jours de conception et de développement, aucun objectif n'a été atteint entièrement. Cependant, mon stage n'est pas encore arrivé à son terme et les 9 jours restants devraient suffire à atteindre l'objectif le plus important : offrir une assistance lors de vols stationnaires par rapport à un point géographique (un seul des deux types de vol stationnaire en hélicoptère). Il restera néanmoins l'implémentation du mode stationnaire air à réaliser, ainsi que la livraison de l'application sur le Play Store, après avoir mené différentes campagnes d'essais.

La non-réalisation des objectifs est principalement due à un manque de temps, mais aussi à un manque d'efficacité lors de certaines étapes et une mauvaise gestion des priorités. Je prévois toutefois de préparer l'avenir du projet en rendant mon code le plus digeste possible et en laissant les indications nécessaires à la personne qui reprendra le projet, pour qu'elle puisse rapidement s'approprier mon travail et réaliser les objectifs.

Mon stage se déroulant, en quelque sorte, au sein du pôle innovation d'une base aérienne de l'Armée, j'ai pu observer des aéronefs en tout genre et en apprendre plus sur leur fonctionnement ainsi que sur leurs utilisations. Mes fréquents contacts avec l'équipe de pilotes d'hélicoptère m'ont permis d'explorer les défis techniques et opérationnels spécifiques auxquels ils sont confrontés au quotidien.

En termes de compétences, j'ai tout d'abord pu appliquer mes connaissances académiques dans un contexte professionnel réel lors de la phase de conception fonctionnelle, tout en prenant en compte les retours des pilotes (clients du projet). Je suis ensuite monté en compétences sur le framework .Net MAUI (technologie permettant notamment le développement d'applications mobiles Android ou iOS) lors de mon auto-formation et lors de la phase de développement. J'ai été impliqué dans toutes les phases du projet, de la conception initiale à la livraison finale, ce qui m'a permis de tester et de perfectionner mes compétences en gestion de projet, tout en découvrant de nouveaux outils dédiés.

Enfin je remercie tout particulièrement mon maître de stage pour m'avoir à maintes reprises démontré combien la communication non-violente (même dans les phases de stress), la bienveillance (même dans les phases d'échec) et le fait d'être à l'écoute en toutes circonstances rendent agréables les relations avec la hiérarchie.

En conclusion, mon stage m'a permis de monter en compétences sur la technologie .Net MAUI, de gagner de l'expérience en gestion de projet, d'améliorer mes compétences en relation-client et de découvrir les enjeux de la DGA et des essais en vol. Je suis reconnaissant d'avoir eu cette opportunité de stage enrichissante qui m'a permis d'appliquer mes connaissances académiques dans un contexte professionnel réel. Ces apprentissages seront précieux pour mes expériences futures et m'aideront à mener à bien les projets à venir.

Références bibliographiques

1. « Les missions de la DGA », Direction générale de l'armement, Ministère des Armées, [URL :] https://www.defense.gouv.fr/dga/missions-dga, consulté le 28/06/2023.

2. « Carte des principales implantations de la DGA », Direction générale de l'armement, Ministère des Armées, [URL :] https://www.defense.gouv.fr/sites/default/files/dga/022_chiffres_cles_DGA_num_fr.pdf, consulté le 28/06/2023.

3. « Latitude et longitude sur la Terre », [dans :] « Coordonnées géographiques », Wikipédia, [URL :] https://fr.wikipedia.org/wiki/Coordonn%C3%A9es_g%C3%A9ographiques, consulté le 26/06/2023.

4. « Comment obtenir la distance entre deux points connus en longitude et latitude sur la sphère ? », IGN (Institut national de l'information géographique et forestière), [URL :] https://geodesie.ign.fr/contenu/fichiers/Distance_longitude_latitude.pdf, consulté le 22/06/2023.

5. « Calcul de la direction sachant 2 points », Azimut, dCode.fr, [URL :] https://www.dcode.fr/azimut, consulté le 22/06/2023.

6. « StackLayout, AbsoluteLayout, Grid, FlexLayout », [dans :] « Dispositions », learn.microsoft.com, 05/05/2023, [URL :] https://learn.microsoft.com/fr-fr/dotnet/maui/user-interface/layouts/, consulté le 22/06/2023.

7. VILLAMOR, Craig, WILLIS, Dan, WROBLEWSKI Luke, « Core Gestures », Touch Gesture : Reference Guide, 15/04/2010, [cité par :] JEANNOT, Vincent, « Vers une standardisation des gestes Multi-Touch ? », ThisIsMyBlog, 13/06/2012, [URL :] https://blog.vincentjeannot.fr/vers-une-standardisation-des-gestes-multi-touch/, consulté le 29/06/2023.

8. MORIEUX, Didier, « Le vol stationnaire de l'hélicoptère », Le portail de Didier Morieux, [URL :] https://dmorieux.pagesperso-orange.fr/volstationnaire0001.htm, consulté le 25/06/2023.

Annexes

Annexe 1 : Protocole pour le maintien d'un stationnaire sol [8] :

Pour assurer le maintien de l'hélicoptère lors d'un stationnaire sol, il faut contrôler en permanence les références extérieures : inclinaison, assiette et cadence, tout en maintenant une hauteur constante au-dessus du sol. À chaque variation des références extérieures ou de hauteur, il faut immédiatement corriger en sens contraire sur la commande correspondante.

En vol stationnaire sol, le déplacement de l'hélicoptère provient des variations d'inclinaison et d'assiette. Toutefois, ce déplacement n'est pas immédiat : il se produit d'abord la variation des références extérieures puis le déplacement. Il y a donc un temps de réponse des commandes qui est de l'ordre d'une seconde suivant le type d'hélicoptère.

Dans le cas d'une variation d'assiette ou d'inclinaison :

- On visualise la variation.
- On corrige à l'aide du manche en sens contraire avant que le déplacement sol n'apparaisse.
- On revient ensuite au neutre pour éviter un déplacement dans le sens contraire de la correction.

Les corrections de cadence à l'aide du palonnier s'effectuent de la même manière.

Le contrôle de la hauteur :

- Si la hauteur diminue, on augmente le pas général ou collectif.
- Si la hauteur augmente, on diminue le pas général ou collectif.

La correction du vent :

- On déplace le manche dans la direction d'où vient le vent. Ce déplacement du manche est d'autant plus grand que le vent est fort.
- Par vent de face, on bénéficie d'un gain de puissance.
- Par vent arrière, le pilotage devient plus délicat en raison de l'effet de girouette du rotor de queue.
- Par vent de côté, on met le manche dans le vent et le palonnier dans le sens opposé.

Annexe 2 : Code complet de la classe BoutonAppuiLong :

```
public class BoutonAppuiLong : Button
{
    private bool clicLongConfirmed = false;
    private TimeSpan tempsClicLong = TimeSpan.FromMilliseconds(1000);
    private System.Timers.Timer _timer;

    public event EventHandler ClicCourt;
    public event EventHandler ClicLong;


    public BoutonAppuiLong()
    {
        this.Pressed += BoutonAppuiLong_Pressed;
        this.Released += BoutonAppuiLong_Released;
        this.Clicked += BoutonAppuiLong_Clicked;
    }

    private void BoutonAppuiLong_Pressed(object sender, EventArgs e)
    {
        if (_timer != null)
        {
            _timer.Stop();
        }

        clicLongConfirmed = false;
        _timer = new System.Timers.Timer(tempsClicLong);
        _timer.Elapsed += _timer_Elapsed;
        _timer.Start();
    }

    private void _timer_Elapsed(object sender, ElapsedEventArgs e)
    {
        clicLongConfirmed = true;
        _timer.Stop();
        HapticFeedback.Default.Perform(HapticFeedbackType.LongPress);
    }

    private void BoutonAppuiLong_Released(object sender, EventArgs e)
    {
        _timer.Stop();
    }


    private void BoutonAppuiLong_Clicked(object sender, EventArgs e)
    {

        if (clicLongConfirmed)
        {
            ClicLong.Invoke(sender, e);
        }
        else
        {
            ClicCourt.Invoke(sender, e);
        }
    }
}
```

Annexe 3 : Capture d'écran de la page de travail

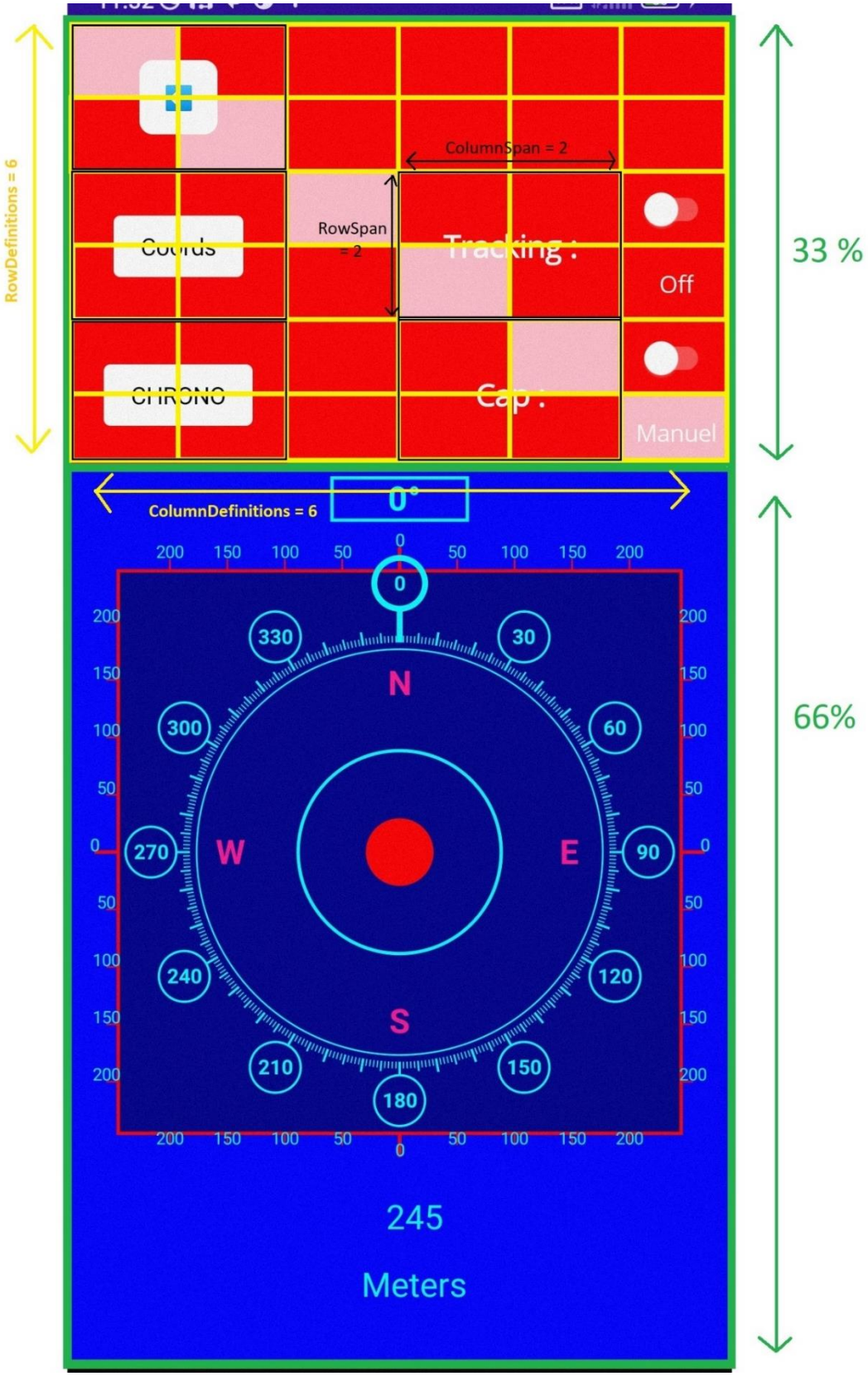


Figure 32 : Capture d'écran de la page de travail avec mise en évidence de la disposition des éléments visuels [ZOOM]