Document d'élaboration 1

Implémentation d'un système d'aide à la vision sur des lunettes de réalité virtuelle

- **Client :** Olivier BODINI <olivier.bodini@univ-paris13.fr>

- **Équipe de suivi :**

  Thierry HAMON <thierry.hamon@univ-paris13.fr>

  Sophie TOULOUSE <sophie.toulouse@lipn.univ-paris13.fr>

- **Groupe :**

  Mohamed Ali YACOUBI <mohamedali.yacoubi@edu.univ-paris13.fr>

  Flavien HAMELIN <flavien.hamelin@edu.univ-paris13.fr>

  Safa KASSOUS <safa.kassous@edu.univ-paris13.fr>

  Farah CHERIF <farah.cherif1@edu.univ-paris13.fr>

  Saad AMMARI <saad.ammari@edu.univ-paris13.fr>

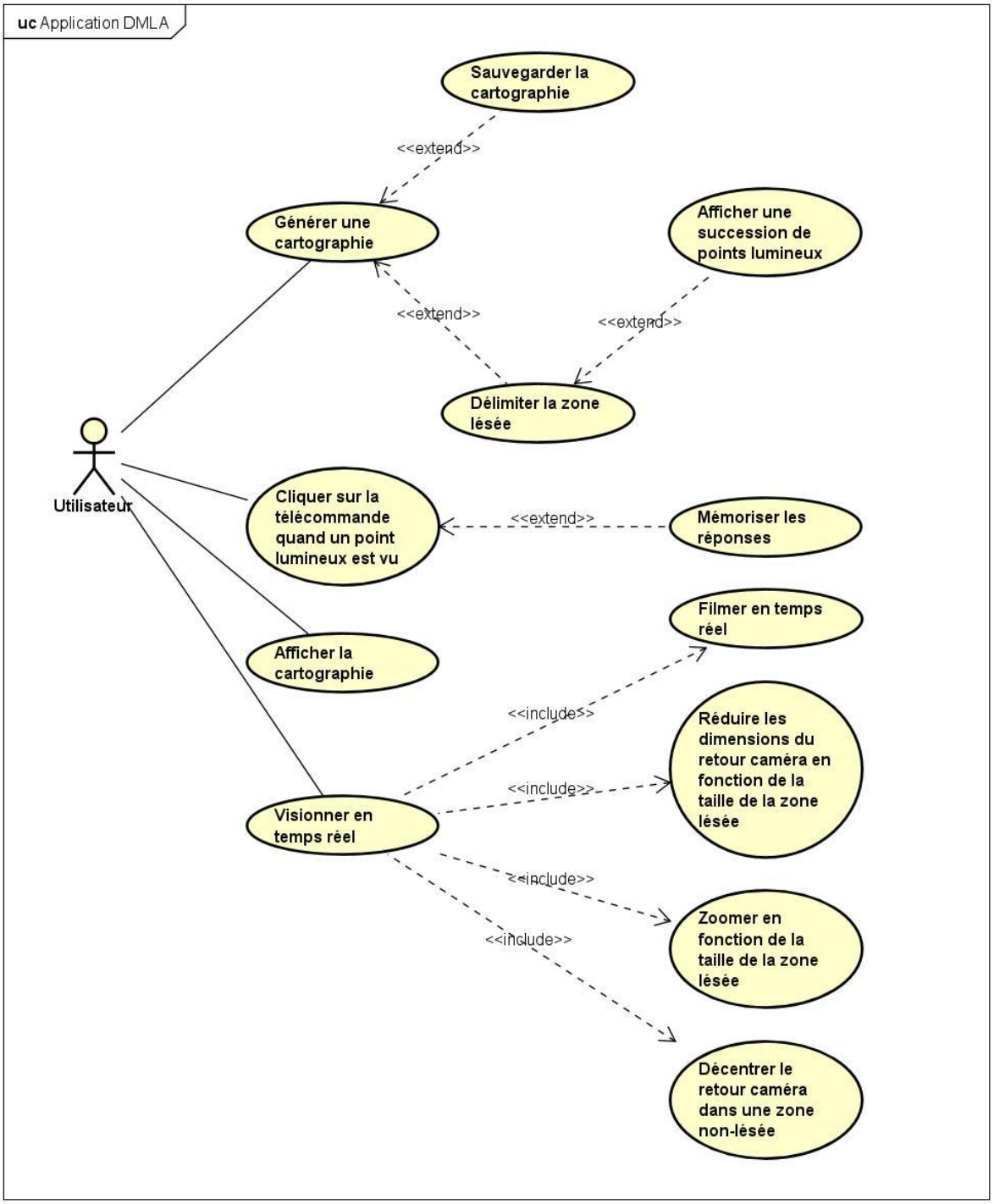
12/01/2022

Sommaire

I) Recontextualisation

Nous souhaitons réaliser un système d'aide à la vision pour les personnes atteintes de dégénérescence maculaire liée à l'âge (DMLA). Ce système utilise les lunettes intelligentes Epson Moverio BT 300. Nous travaillons sur le système d'exploitation open source utilisé par les lunettes : Moverio OS, qui est une surcouche utilisant le système d'exploitation Android. L'environnement de développement choisi est Android Studio. Grâce à cet environnement nous pouvons créer une application Android compatible avec le système d'exploitation des lunettes.

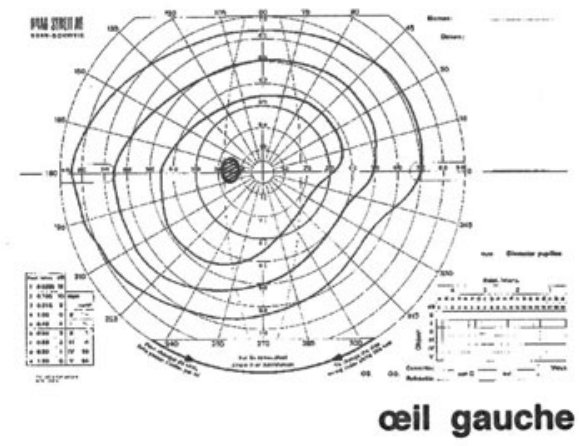
Afin de visualiser les cas d'utilisations de l'application aidant les personnes atteintes de DMLA, nous vous proposons le diagramme de cas d'utilisation ci-après :


uc Application DMLA
Sauvegarder la cartographie
<<extend>>
Générer une cartographie
Afficher une succession de points lumineux
<<extend>>
<<extend>>
Délimiter la zone lésée
Utilisateur
Cliquer sur la télécommande quand un point lumineux est vu
<<extend>>
Mémoriser les réponses
Filmer en temps réel
Afficher la cartographie
<<include>>
Réduire les dimensions du retour caméra en fonction de la taille de la zone lésée
<<include>>
Visionner en temps réel
<<include>>
Zoomer en fonction de la taille de la zone lésée
<<include>>
Décentrer le retour caméra dans une zone non-lésée

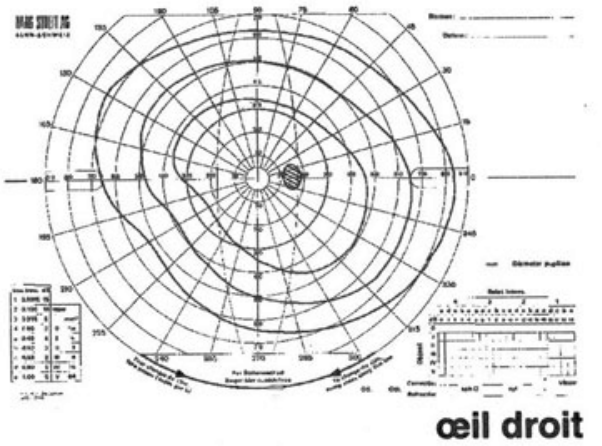
En ouvrant l'application, l'utilisateur doit pouvoir dans un premier temps : "générer une cartographie". Cette option permet de lancer un test d'une durée de 20 minutes, ce test affiche , sur l'écran des lunettes intelligentes, une succession de points lumineux de tailles, positions et luminosités différentes. A l'issue de ce test les zones lésées des yeux qui sont atteintes par la DMLA sont déterminées et l'ensemble des données collectées est sauvegardé.

L'utilisateur peut agir pendant le test précédemment décrit, si l'utilisateur clique sur la télécommande alors qu'un point lumineux vient d'apparaître, on considérera ce point comme vu. Si l'utilisateur clique alors qu'un point n'est pas encore apparu, on considérera que la réponse est faussée et que l'information ne doit pas être stockée. Si l'utilisateur ne clique pas alors qu'un point est apparu, on considérera que ce point n'est pas vu.

Dans l'application, il doit y avoir une possibilité de visionner la cartographie réalisée à partir du test lancé via : "générer une cartographie". De ce fait, le cas : "Afficher la cartographie", affiche l'ensemble des données récoltées pendant la durée du test sous la forme de deux champs visuels. Vous pouvez visionner ci-après les champs visuels de deux yeux normaux non atteints par la DMLA :



œil gauche



œil droit

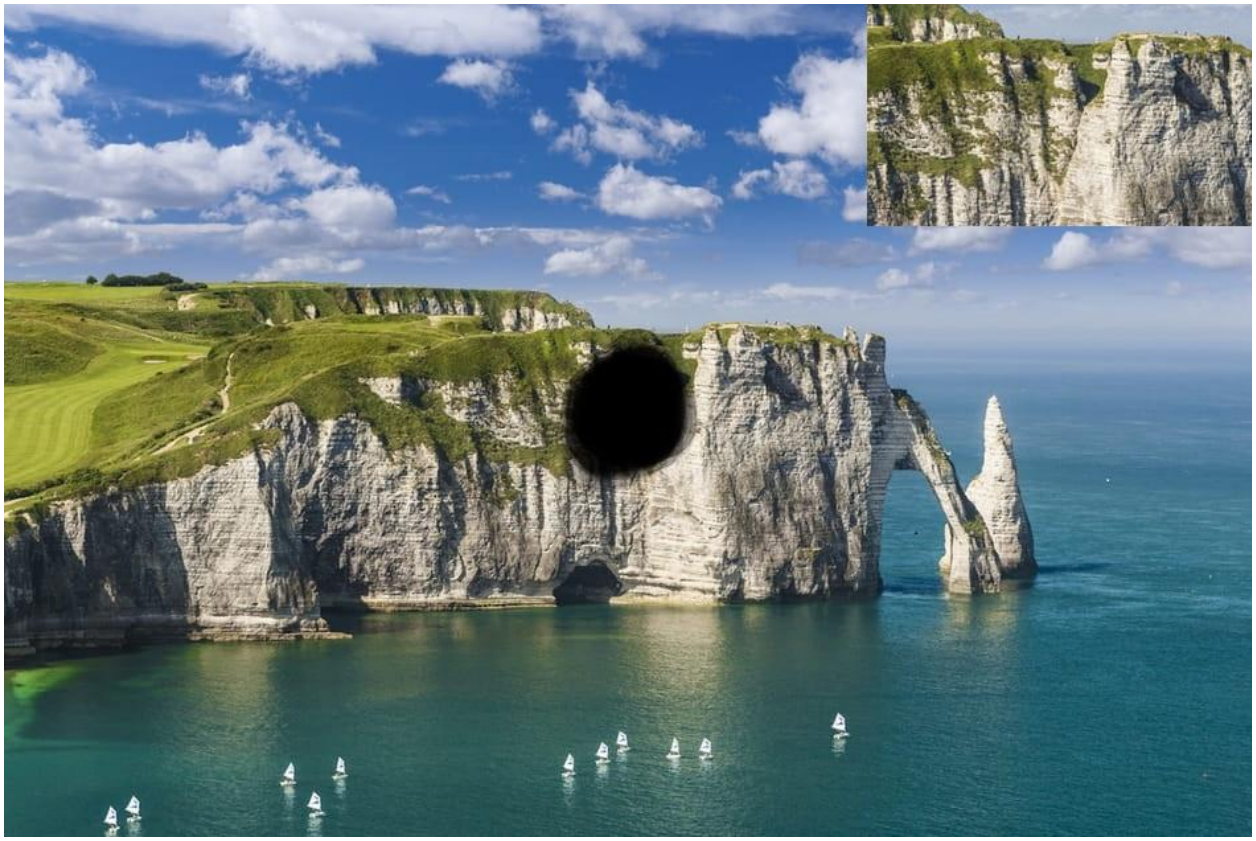
On constate que les yeux possèdent déjà à l'origine une zone aveugle caractérisée par une tâche noire. L'affichage de la cartographie consisterait à préciser les zones lésées des yeux en ajoutant de nouvelles tâches sur ce modèle représentatif.

Lorsque les zones lésées sont détectées et stockées, on peut désormais aider l'utilisateur à améliorer sa vision. Le cas "visionner en temps réel" précise la solution primaire considérée pour aider un malade de DMLA, il s'agit de filmer la zone de l'espace qu'il ne peut pas voir, de lui afficher un retour caméra sur les lunettes intelligentes dans une zone non lésée.

II) Description des fonctionnalités implémentées

1) Retour caméra dans les lunettes intelligentes

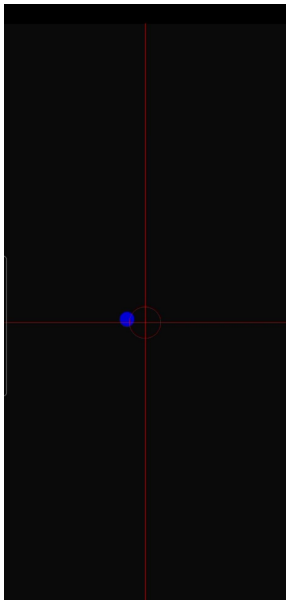
A ce jour, nous avons proposé une première implémentation à la problématique d'affichage d'un retour caméra dans les lunettes intelligentes. Nous sommes capables d'afficher un retour caméra dans le coin supérieur droit de la zone d'affichage des lunettes. Les dimensions sont choisies arbitrairement et ne dépendent pas encore de la forme et de la taille de la zone non vue par l'utilisateur affecté par la maladie. On applique un zoom à ce retour caméra car la zone lésée se trouve généralement au centre de la vision, il faut pouvoir zoomer sur la zone qui est non vue. La valeur de ce zoom dépendra à l'avenir des données récoltées pendant le test : "générer une cartographie". Ci-dessous, vous pouvez constater un exemple de ce que permet l'implémentation que nous avons réalisé à ce jour si vous êtes atteints de DMLA et que vous mettez les lunettes intelligentes :

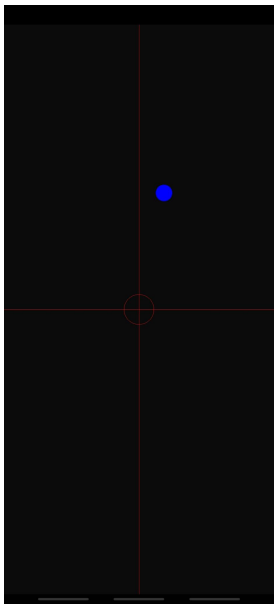


2) Affichage des points lumineux

Dans la partie “Générer Cartographie”, nous avons réussi à afficher des points lumineux aléatoirement avec des variations de luminosité et de taille sur toute l’interface afin d’étudier toutes les zones pour avoir une cartographie améliorée et plus précise. Le temps consacré pour afficher un point et attendre la réponse de l’utilisateur est de 6 secondes. Dans un premier lieu, nous commençons à afficher un point P de luminosité l, de position (x,y) et de rayon r à partir d’un temps db dans l’intervalle [0s,3s] (date_début_affichage). La date de la fin de l’affichage date_fin_affichage sera dans l’intervalle [date_début_affichage + 1,4]. Nous consacrons une période dr qui est égale à : 6 - date_fin_affichage pour la réponse de l’utilisateur.

Les figures ci-dessous montrent l’affichage des points lumineux ayant une position, une luminosité et une taille différentes.



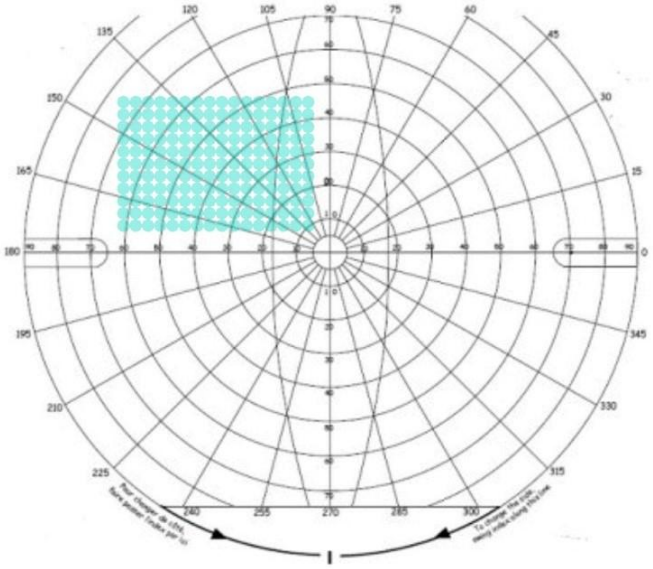


Algorithme:

```
#dispatchKeyEvent va mettre clique = true si l'utilisateur clique sur
#la télécommande

dispatchKeyEvent(@NotNull KeyEvent event) {
      clique = True
}


#now() retourne le temps courant en millisecondes
#random(x,y) retourne un nombre aléatoire entre x et y
#sleep(x) met le système en état d'attente pendant x millisecondes

T <- now()
#La durée totale du test ne doit pas dépasser 20 minutes
Tant que (now()-T) < 20*60*1000
      debut_affichage <- random(0,3000)
      fin_affichage <- random(debut+1000,4000)
      tclique_max <- now()+6000
      epsilon <- 100

      Tant que now()<tclique_max
            zero = now()
            si abs(now()-debut_affichage)<epsilon alors
                  affiche_point()
            fin si
            si abs(now()-fin_affichage)<epsilon alors
                  effacer_point()
            fin si

            si clique && (tclique-zero)<debut_affichage alors
                  #faux_positif
                  #point non stocké
            sinon
                  si clique && debut_affichage < (tclique-zero) <
            tclique_max alors
                              stocker_point_comme_vu()
                  sinon
                              stocker_point_comme_non_vu()
                  fin si
            fin si

            sleep(200)
      Fin tant que
Fin tant que
```

3) Affichage d’une cartographie

Pour l’instant au niveau de la partie “afficher cartographie”, nous avons bel et bien un affichage en champ de vision comme mentionné dans le I) sauf que pour le moment nous ne prenons pas en compte l'œil gauche et droit séparément. Nous avons donc un seul champ de vision où sont représentées les zones lésées des deux yeux (voir figure ci-dessous). Les petits cercles bleus ciels représentent les zones lésés de l’utilisateur après avoir généré une cartographie.

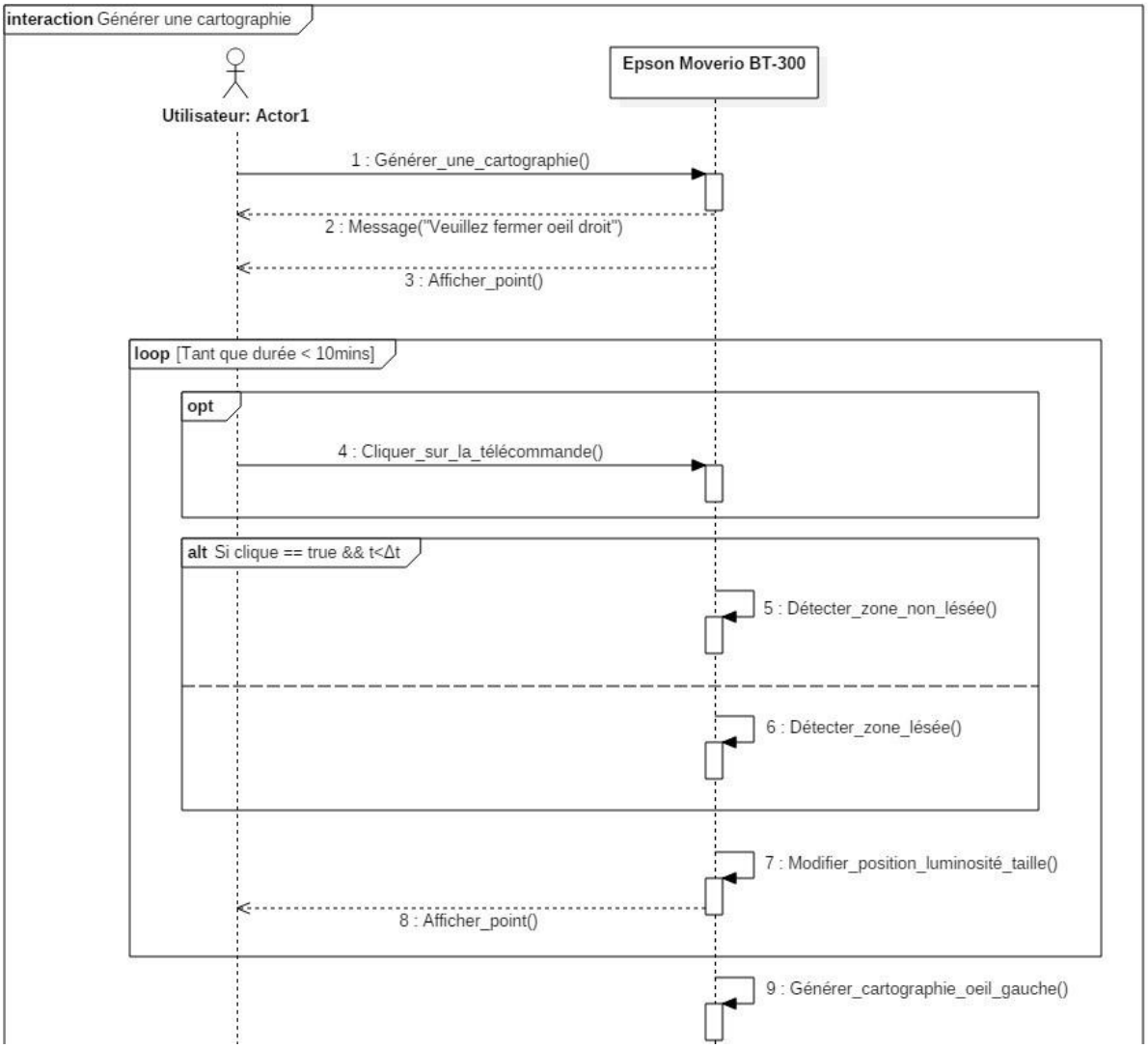


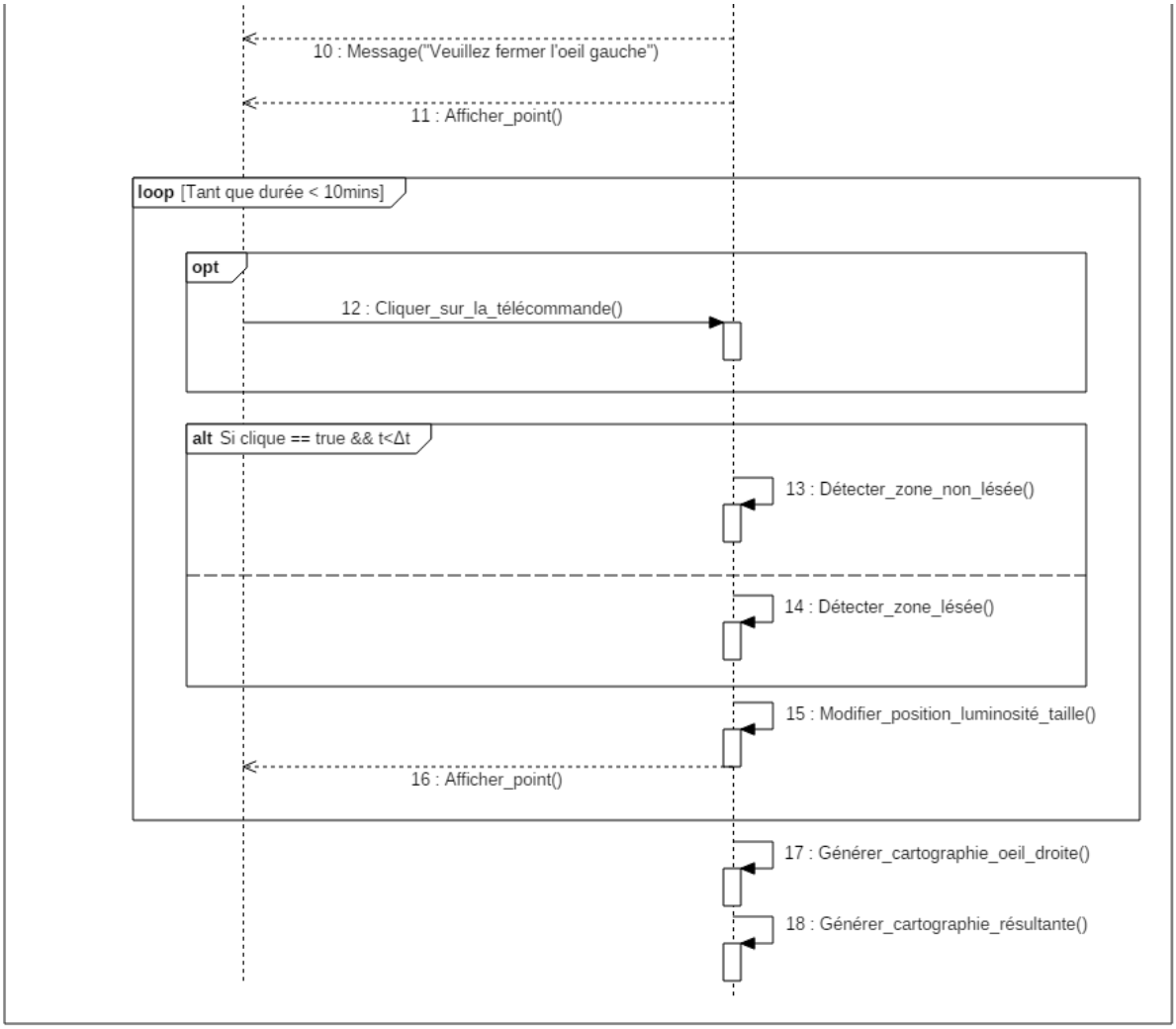
III) Description des fonctionnalités à implémenter

1) Génération d'une cartographie

En fonction de la réponse de l'utilisateur, une cartographie sera générée qui va délimiter les zones lésées, les zones non lésées et les frontières de brouillage de vision entre ces zones (générer des différents niveaux de gris : Noir pour les zones lésées, blanc pour les zones non lésées et gris pour les zones de brouillage de vision). Nous allons réaliser le test de vision pour chaque œil pendant 10 minutes afin de stocker deux cartographies : cartographie oeil gauche et cartographie oeil droit. Finalement, nous générons une cartographie résultante en superposant les deux cartographies oeil gauche et oeil droit afin de déterminer les zones lésées quand l'utilisateur observe avec ses deux yeux.

Le diagramme de séquence du cas d'utilisation "Générer cartographie" suivant montre les différentes intéractions de l'utilisateur avec les lunettes :


interaction Générer une cartographie
Utilisateur: Actor1
Epson Moverio BT-300
1 : Générer_une_cartographie()
2 : Message("Veuillez fermer oeil droit")
3 : Afficher_point()
loop [Tant que durée < 10mins]
opt
4 : Cliquer_sur_la_télécommande()
alt Si clique == true && t<Δt
5 : Détecter_zone_non_lésée()
6 : Détecter_zone_lésée()
7 : Modifier_position_luminosité_taille()
8 : Afficher_point()
9 : Générer_cartographie_oeil_gauche()



2) Affichage d’une cartographie (adaptation de l'affichage d'une cartographie pour 2 champs de visions)

Après la génération d’une cartographie, nous aurons trois structures de données où seront stockés les résultats des tests pour l'œil gauche, l'œil droit et une structure où nous aurons les deux résultats cumulés. Les deux premières structures nous permettront d’afficher grâce à un modèle de champ de vision les zones lésées et non lésées des deux yeux.

IV) Diagramme de Gantt

Comme le montre ce diagramme, nous avons pu avancer sur les différentes tâches qui constituent notre projet, à savoir la mise en place de l'affichage des points lumineux et la mise en œuvre de l'affichage d'un retour caméra dans les lunettes intelligentes. En parallèle, nous avons commencé à mettre en place la cartographie pour chaque œil avec un modèle de champ visuel.

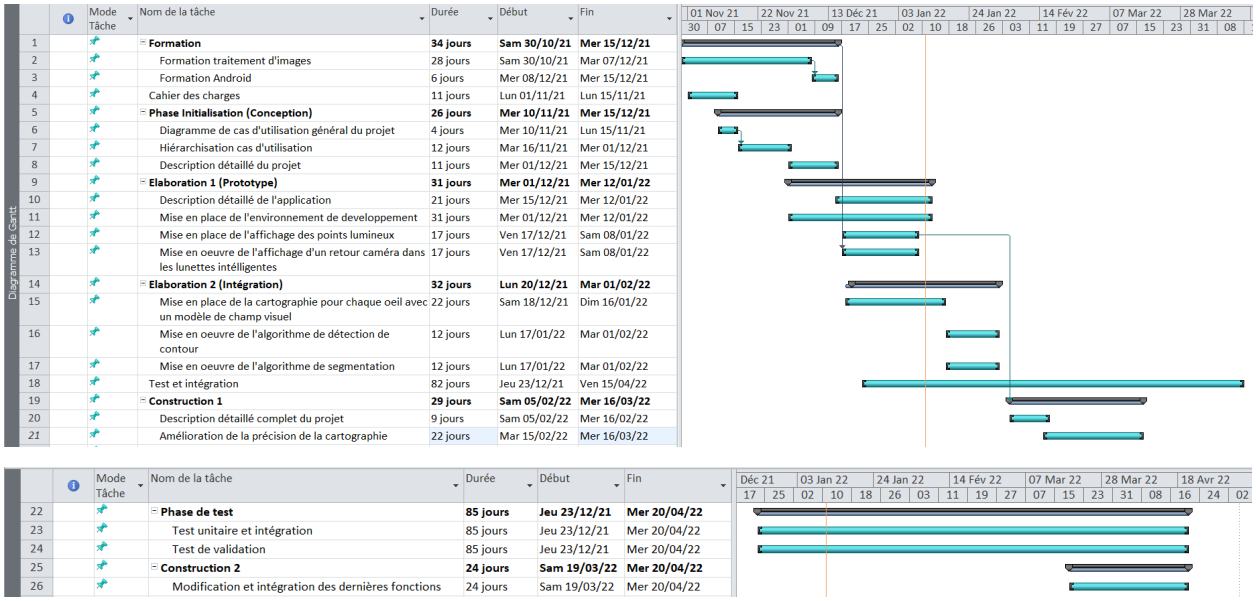


Diagramme de Gantt

	Mode Tâche	Nom de la tâche	Durée	Début	Fin
1		**Formation**	**34 jours**	**Sam 30/10/21**	**Mer 15/12/21**
2		Formation traitement d'images	28 jours	Sam 30/10/21	Mar 07/12/21
3		Formation Android	6 jours	Mer 08/12/21	Mer 15/12/21
4		Cahier des charges	11 jours	Lun 01/11/21	Lun 15/11/21
5		**Phase Initialisation (Conception)**	**26 jours**	**Mer 10/11/21**	**Mer 15/12/21**
6		Diagramme de cas d'utilisation général du projet	4 jours	Mer 10/11/21	Lun 15/11/21
7		Hiérarchisation cas d'utilisation	12 jours	Mar 16/11/21	Mer 01/12/21
8		Description détaillé du projet	11 jours	Mer 01/12/21	Mer 15/12/21
9		**Elaboration 1 (Prototype)**	**31 jours**	**Mer 01/12/21**	**Mer 12/01/22**
10		Description détaillé de l'application	21 jours	Mer 15/12/21	Mer 12/01/22
11		Mise en place de l'environnement de developpement	31 jours	Mer 01/12/21	Mer 12/01/22
12		Mise en place de l'affichage des points lumineux	17 jours	Ven 17/12/21	Sam 08/01/22
13		Mise en oeuvre de l'affichage d'un retour caméra dans les lunettes intélligentes	17 jours	Ven 17/12/21	Sam 08/01/22
14		**Elaboration 2 (Intégration)**	**32 jours**	**Lun 20/12/21**	**Mar 01/02/22**
15		Mise en place de la cartographie pour chaque oeil avec un modèle de champ visuel	22 jours	Sam 18/12/21	Dim 16/01/22
16		Mise en oeuvre de l'algorithme de détection de contour	12 jours	Lun 17/01/22	Mar 01/02/22
17		Mise en oeuvre de l'algorithme de segmentation	12 jours	Lun 17/01/22	Mar 01/02/22
18		Test et intégration	82 jours	Jeu 23/12/21	Ven 15/04/22
19		**Construction 1**	**29 jours**	**Sam 05/02/22**	**Mer 16/03/22**
20		Description détaillé complet du projet	9 jours	Sam 05/02/22	Mer 16/02/22
21		Amélioration de la précision de la cartographie	22 jours	Mar 15/02/22	Mer 16/03/22
22		**Phase de test**	**85 jours**	**Jeu 23/12/21**	**Mer 20/04/22**
23		Test unitaire et intégration	85 jours	Jeu 23/12/21	Mer 20/04/22
24		Test de validation	85 jours	Jeu 23/12/21	Mer 20/04/22
25		**Construction 2**	**24 jours**	**Sam 19/03/22**	**Mer 20/04/22**
26		Modification et intégration des dernières fonctions	24 jours	Sam 19/03/22	Mer 20/04/22

V) En direction de l’élaboration 2

Nous avons commencé à améliorer l’affichage des points lumineux et le retour de la caméra sur les lunettes de réalité virtuelle pendant l’élaboration 1. Nous nous consacrons principalement à améliorer la génération de la cartographie finale tout en montrant les différents niveaux de gris afin de déterminer les zones lésées en utilisant Android Studio comme environnement de développement.