

Document d'élaboration 2

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>

—



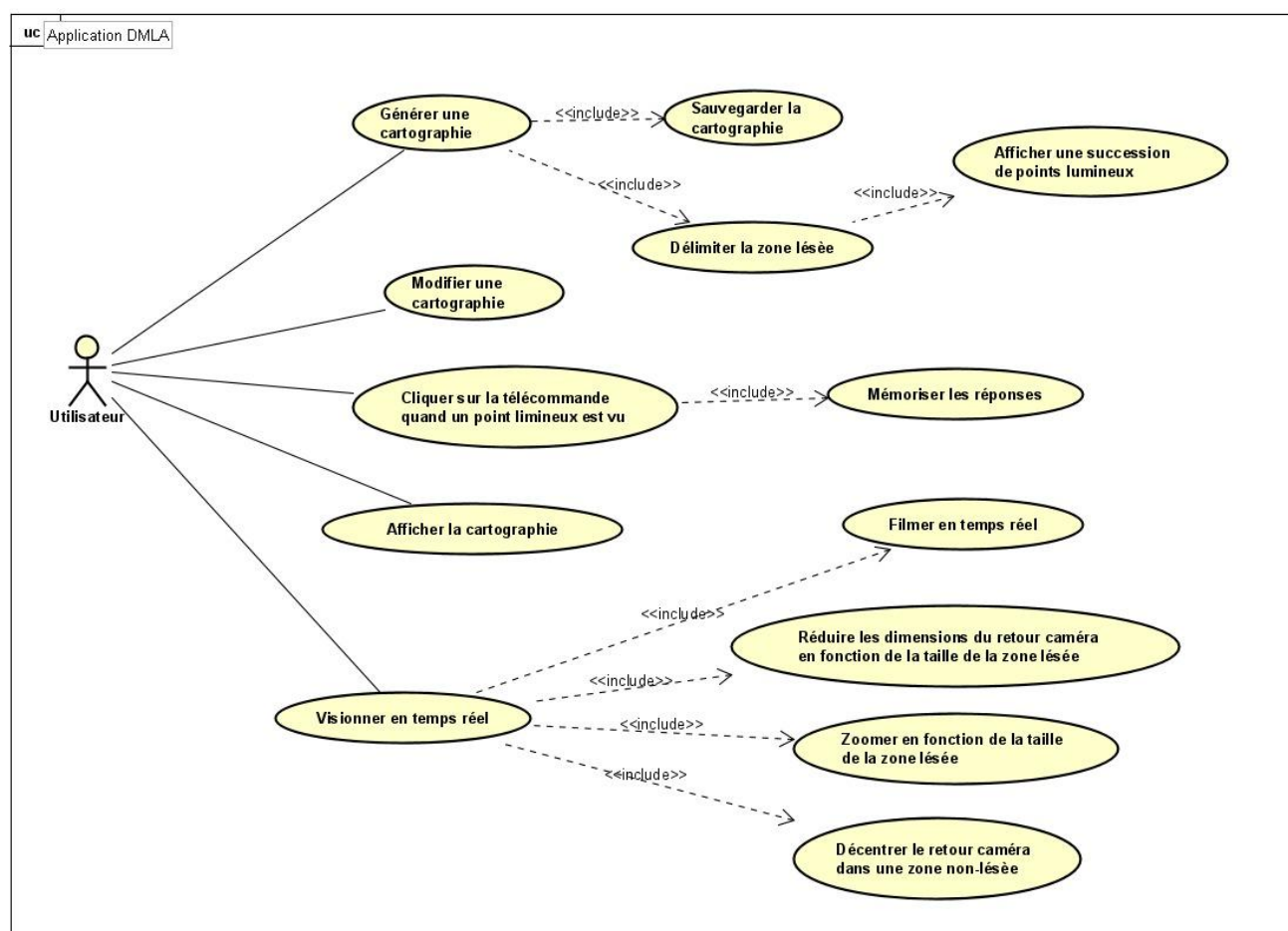
Sommaire

I) Recontextualisation	3
II) Description des fonctionnalités implémentées	4
III) Description des fonctionnalités à implémenter	8
IV) Diagramme de Gantt	11
V) En direction de la construction 1	12

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 :



II) Description des fonctionnalités implémentées


1) Affichage des points lumineux

Lors de la première élaboration, nous avons réussi à implémenter l'affichage des points lumineux avec des positions, luminosités et tailles choisies aléatoirement. En effet, Le temps consacré pour afficher un point et attendre la réponse de l'utilisateur est de 6 secondes. Dans un premier temps, nous commençons à afficher un point P de luminosité I , de position (x,y) et de rayon r , à partir d'un temps "date_début_affichage" qui est choisi aléatoirement dans l'intervalle $[0s, 3s]$. La date de la fin de l'affichage $date_fin_affichage$ est choisie aléatoirement dans l'intervalle $[date_début_affichage + 1s, 4s]$. Nous consacrons une période de réponse utilisateur qui est égale à : $6 - date_fin_affichage$. De ce fait, l'utilisateur a au moins 2 secondes pour signaler qu'il a vu un point en cliquant sur la télécommande.

Dans la partie "Générer Cartographie", nous avons réussi à séparer les phases de tests pour les deux yeux : un test de vision pour chaque œil qui dure 10 minutes à travers l'affichage des points lumineux.

A partir des réponses de l'utilisateur, nous avons considéré :

- Point vu : lorsque l'utilisateur clique au moment de l'affichage du point ou lorsqu'il a cliqué après la disparition du point mais qu'il n'a pas encore dépassé la date de fin d'affichage.
- Point non vu : lorsque l'utilisateur n'a pas cliqué pendant la période de 6 secondes contenant l'affichage d'un point.
- Point traité : c'est un point vu ou bien non vu.
- Point non sauvegardé : lorsqu'on a un point qui est un faux positif, si l'utilisateur clique alors que le point n'a même pas encore été affiché.



Nous avons réussi aussi à stocker les réponses de l'utilisateur dans plusieurs listes :

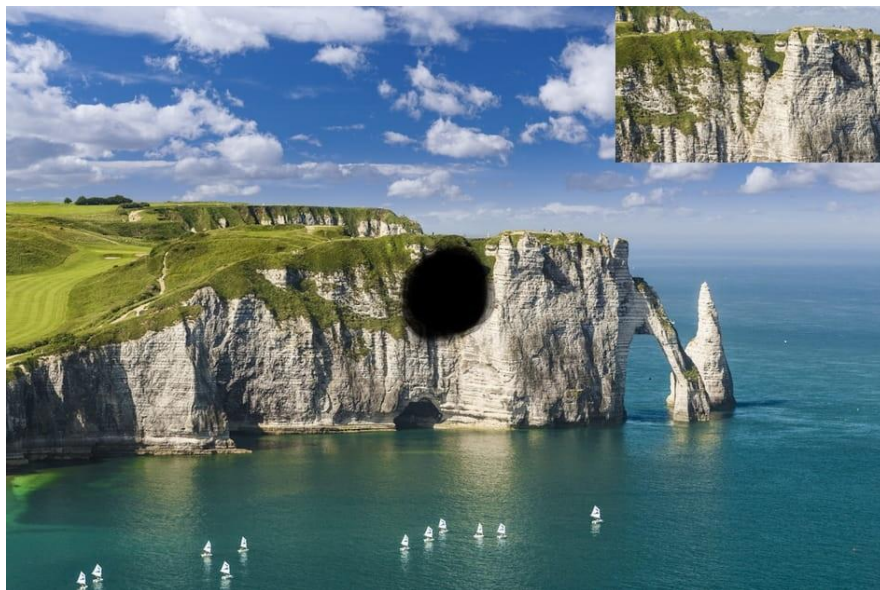
- `liste_points_vus_oeil_gauche[]`
- `liste_points_non_vus_oeil_gauche[]`
- `liste_points_traités_oeil_gauche[]`
- `liste_points_vus_oeil_droit[]`
- `liste_points_non_vus_oeil_droit[]`
- `liste_points_traités_oeil_droit[]`

Pour la synchronisation et la gestion du temps nous avons manipulé les threads. Nous avons utilisé `Timer` et `TimerTask` qui sont des classes utilitaires Java utilisées pour planifier des tâches dans un thread d'arrière-plan.

Ainsi, le stockage dans les différentes listes nous permettra de générer deux champs visuels (oeil gauche et oeil droit) pour générer à la fin une cartographie résultante afin de déterminer les zones lésées quand l'utilisateur observe avec ses deux yeux.

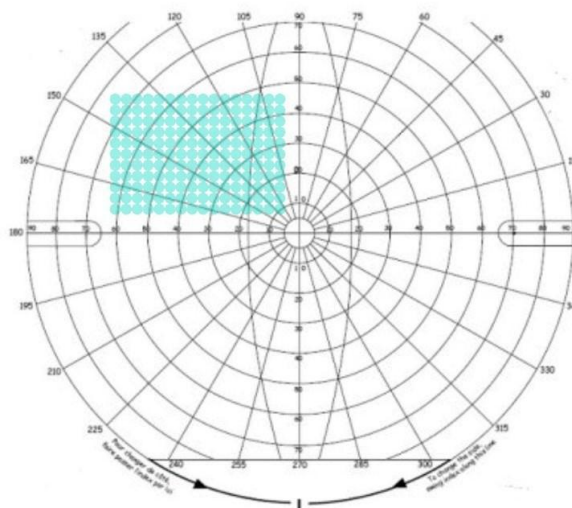
2) Retour caméra sur les lunettes intelligentes

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 :



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 sommes capables d'afficher une cartographie pour un seul œil. Après avoir terminé la partie "générer une cartographie" nous serons en mesure d'utiliser les deux cartographies différentes générées pour les deux yeux afin de proposer à l'utilisateur de visualiser le champ visuel de l'œil gauche et le champ visuel de l'œil droit. Les cercles bleus ciel dans la figure ci-dessous représentent les zones lésées de l'utilisateur après avoir généré une cartographie pour un œil.



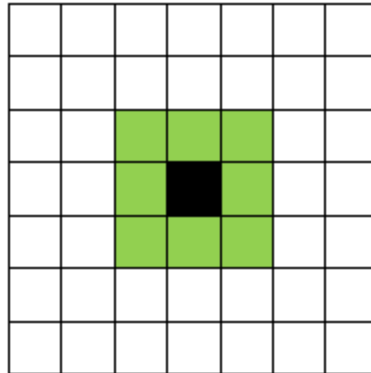
III) Description des fonctionnalités à implémenter

- **Modification d'une cartographie :** Dans le cas où il y a une évolution de la maladie, nous allons offrir à l'utilisateur la possibilité de modifier une cartographie déjà existante. La nouvelle cartographie générée sera plus précise.
- **L'envoi d'une cartographie par e-mail :** Notre application va permettre à l'utilisateur de sélectionner une cartographie, et de saisir l'adresse email du destinataire. Un email contenant la cartographie sera expédié automatiquement à ce dernier.
- **Stockage des points traités dans une matrice creuse :** Vu le grand nombre de points à traiter, il est préférable de stocker ces informations dans une matrice creuse. Une matrice creuse est une matrice contenant beaucoup de zéros. Donc, au lieu d'avoir plusieurs listes, on stocke ces informations dans une matrice creuse et on utilise moins d'octets de stockage.
- **Génération d'une cartographie :**

En fonction de la réponse de l'utilisateur, une cartographie sera générée. Cette cartographie 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 avons réalisé le test de vision pour chaque œil pendant 10 minutes afin de stocker six listes qui nous permettront de générer la cartographie de l'œil gauche et la cartographie de l'œil droit. Finalement, nous générerons une cartographie résultante en superposant les deux cartographies œil gauche et œil droit afin de déterminer les zones lésées quand l'utilisateur observe avec ses deux yeux.

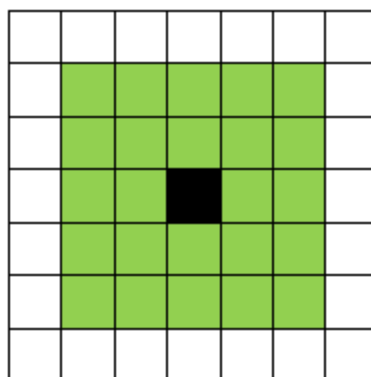
Pour générer une cartographie il faut déduire les nuances de gris de toutes les positions possibles sur l'affichage. Or, un problème se pose : le temps d'un test de vision entier dure 20 minutes et la période de temps que nous utilisons pour l'affichage d'un point fait que l'on affiche 200 points à l'utilisateur durant toute la durée du test. 200 points ne suffisent pas pour couvrir toute la surface de l'affichage et il y aura des zones non traitées. De ce fait, nous devons réaliser un calcul des nuances de gris des points non traités. Pour réaliser ce calcul nous proposons la méthode suivante :

Pour un point non traité, nous considérons les 8 points donnés par son voisinage de Moore d'ordre 1 :

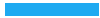


Si on a N points qui ont été traités avec $N \geq 4$ sur les 8 points du voisinage de Moore d'ordre 1 alors on réalise la moyenne des nuances de gris de ces N points pour déterminer la nuance de gris du point central non traité.

Sinon, si moins de 4 points sur les 8 points du voisinage ont été traités alors on augmente l'ordre du voisinage de Moore jusqu'à ce que notre valeur N soit supérieure à 4 :



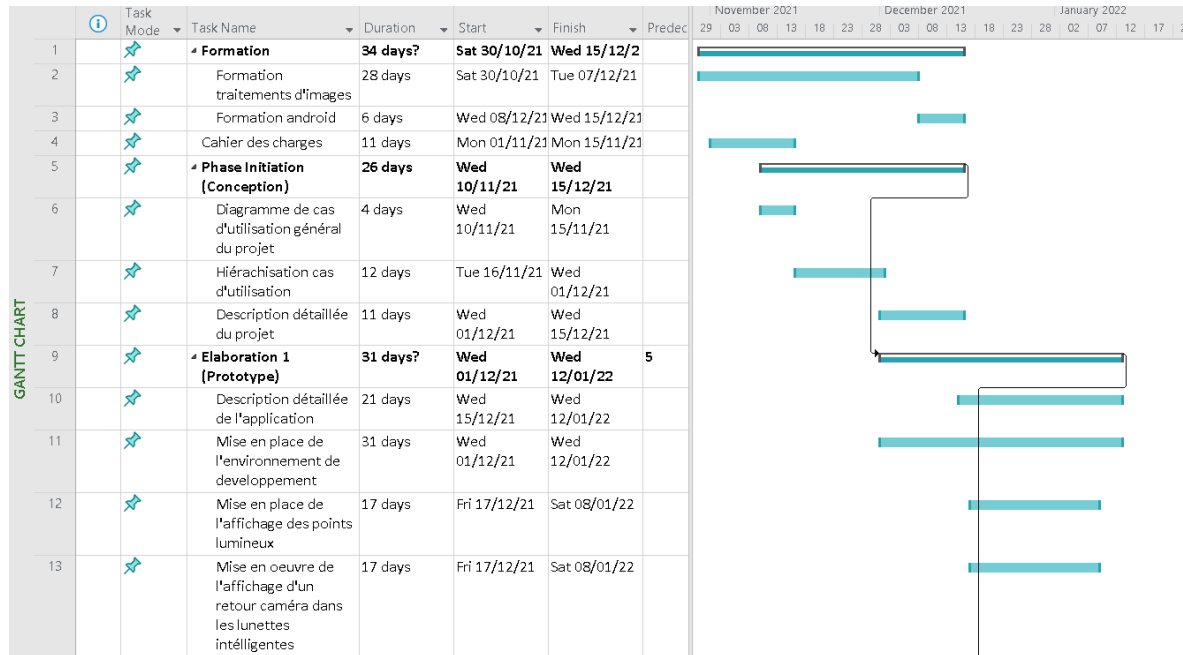
Sur cette itération suivante, on chercherait à avoir N points qui ont été traités avec $N \geq 4$ sur les 24 points du voisinage de Moore d'ordre 2, si ce n'est pas le cas, on considère le voisinage de Moore d'ordre 3.

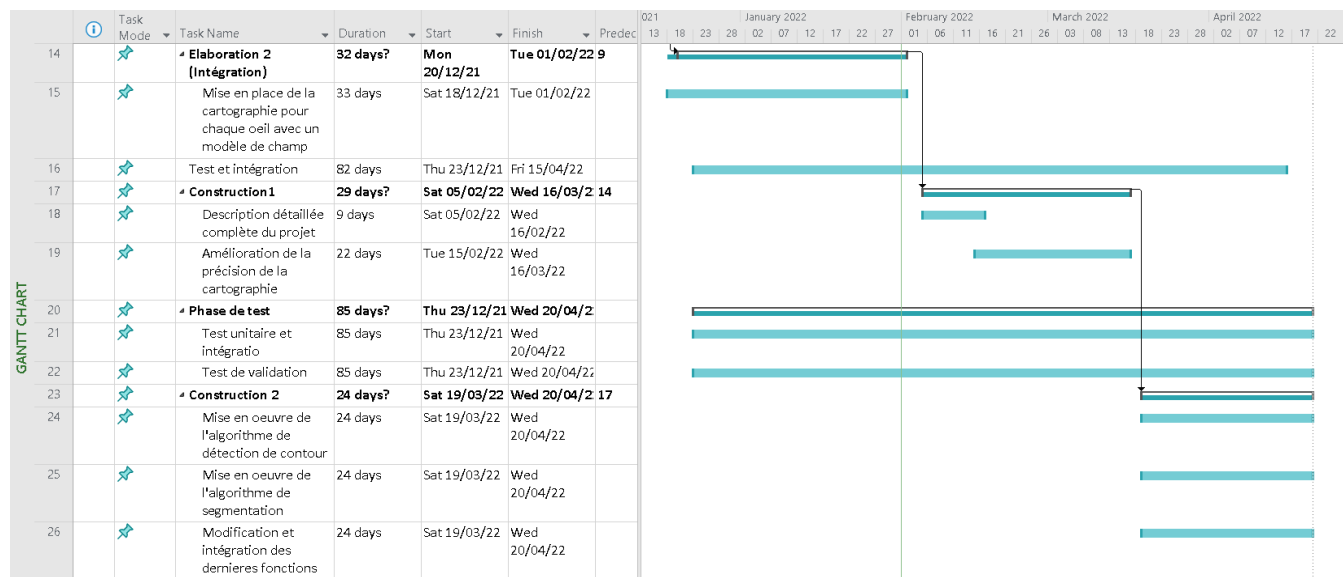


Cette méthode permettrait de pallier le problème des informations manquantes et s'adapte à notre implémentation si on augmente le temps de test total en passant par exemple de 20 minutes à 30 minutes.

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 la cartographie pour chaque œil avec un modèle de champ visuel.





V) En direction de la construction 1

Nous avons réussi à améliorer l'affichage des points lumineux et la mise en œuvre de l'affichage d'un retour caméra sur les lunettes intelligentes ainsi que la mise en place de la cartographie pour chaque œil avec un modèle de champ de vision pendant l'élaboration 1 et l'élaboration 2. A partir des réponses de l'utilisateur que nous avons stocké dans plusieurs listes, nous nous consacrons principalement à améliorer la précision de la cartographie finale tout en montrant les différents niveaux de gris afin de déterminer les zones lésées.