Groupe n°7
ABOUTARIK Ali
DUDIK Sviatoslav
FISSEL-LARAKI Mohamed
TAIBI Sofiane

Conduite et gestion de projet

Aide à la vision sur des lunettes de réalité virtuelle

2020-2021

Client : Olivier Bodini
Équipe de suivi : Thierry Hamon, Sophie Toulouse
École d'ingénieurs Sup Galilée

Journal des modifications

Version 5.0

- Ajout d’explications sur la méthode utilisée pour faire la cartographie (p.4-5)
- Description détaillée des derniers cas d’utilisation (p.13-14)
- Nouvelles captures d’écran (p.15-17)
- Ajout de diagrammes de séquences de de leur description (p.19-24)
- Ajout d’explications sur des choix techniques (Watershed, DICOM, Amsler) (p.25-27)

Version 6.0

- Ajout de diagrammes de séquences de de leur description (p. 27)
- Ajout du bilan global du projet (p. 33)

Introduction

Dans le cadre du cours de gestion de projet enseigné en spécialité Informatique de l'école d'ingénieurs Sup Galilée, nous sommes amenés à réaliser le projet d'aide à la vision sur des lunettes de réalité virtuelle.
Ce document sert de cadre à ce projet. Dans un premier temps, il va nous permettre de définir les objectifs. Puis nous allons déterminer les fonctionnalités nécessaires à la réalisation de ce projet. Ce document nous servira aussi à mettre en place une organisation au sein de l'équipe. Il sera aussi un moyen de partager notre vision du projet avec notre client.

Nom	Prénom	Rôle	Mail
ABOUTARIK	Ali	Chargé d'interface graphique	medaboutarik@gmail.com
DUDIK	Sviatoslav	Chef de projet	sviatoslav.dudik@gmail.com
FISSEL LARAKI	Mohamed	Architecte	m.fissel.laraki@gmail.com
TAIBI	Sofiane	Chargé d'algorithmique	sofiane.st98@gmail.com

Analyse du contexte

Contexte

En France, près de 1,3 million de personnes sont atteintes de DMLA.
La DMLA (dégénérescence maculaire liée à l'âge) est une maladie liée à un vieillissement de la zone centrale de la rétine. Elle se traduit par une perte progressive de la vision centrale. C'est une maladie qui touche essentiellement les personnes de tranche d'âge comprise entre 60 et 75 ans.
Ainsi notre objectif sera de créer une application qui sera destinée à aider ces personnes à voir mieux.

Objectifs

Notre objectif consiste à venir en aide à ces personnes atteintes de DMLA. Pour cela, on compte munir des lunettes de réalité virtuelle d'un logiciel permettant de

déformer les images captées par la caméra afin de les réparer selon l'œil de l'utilisateur.
Notre objectif est de développer une application permettant de détecter les zones lésées de la rétine, afin de pouvoir déformer l'image au mieux pour qu'elle envoie le maximum d'informations à l'utilisateur. Pour ce faire, il va falloir créer dans un premier temps un programme qui sera amené à envoyer un signal lumineux dans le champ de vision de l'utilisateur en commençant par le centre car la personne atteinte ne voit plus au centre de son champ de vision (la taille de la tâche dépend de la sévérité de la maladie). Le but étant de délimiter la zone lésée de la rétine. Ensuite, nous développerons un algorithme permettant de déformer l'image prise par la caméra des lunettes afin de déplacer l'information dans la partie non touchée de la rétine.

Analyse du besoin

Dans ce projet, l'idée est de développer une application (et une Interface graphique) qui servira à cartographier les zones lésées de l'œil. On pourra utiliser une télécommande pour recevoir les réponses de l'utilisateur. Cette application devra être conçue par l'intermédiaire du langage Java. Le développement de l'application devra :

- Être capable d'afficher en temps réel des points lumineux dans le champ visuel.
- Faire en sorte que l'utilisateur soit capable de transmettre ses réponses à l'application.
- Faire en sorte que l'application crée une cartographie des zones lésées grâce au réponse de l'utilisateur.
- Proposer un système de modification d'image afin de prendre en compte les zones lésées de la rétine.

Analyse de l'existant

Avant de commencer, il faut savoir que nous avons à notre disposition les lunettes de réalité augmentée Epson BT-300. Elles sont fournies avec une télécommande et des caches. Comme le verre est transparent, l'utilisateur aura besoin de ces caches pour ne voir que l'écran des lunettes.
Les lunettes Epson BT-300 ont un système d'exploitation basé sur Android permettant de lancer des applications Android.

Après avoir fait quelques recherches nous avons trouvé que la cartographie du champ visuel peut être réalisée de plusieurs manières:

1) Relevé du champ par périmétrie cinétique
2) Relevé du champ par périmétrie statique
3) La technique flicker (papillotement)

La première méthode nécessite la présence d'un expert pour déterminer les problèmes en temps réel, donc ne convient pas pour notre utilisation. La technique flicker utilise une fréquence spécifique de rafraîchissement de l'écran et est principalement utilisée pour le dépistage des glaucomes. Nous avons décidé d'utiliser le relevé du champ par périmétrie statique.

La méthode qu'on utilise consiste à afficher des points lumineux un par un en faisant varier leurs intensités. À la sortie nous aurons une matrice qui contient la sensibilité de l'œil (en dB) dans chaque zone. Ensuite, on soustrait de cette matrice celle de la population normale de même âge (déterminée statistiquement) et on obtient la carte de déviation. Puis, on peut construire la carte de déviation individuelle qui reflète surtout les anomalies liées à la maladie.

Fonctionnalité

Titre	Générer une cartographie
Objectif	Obtenir une cartographie des zones lésées
Description	Des points lumineux sont affichés un par un sur un fond noir et l'utilisateur sélectionne à l'aide de la télécommande s'il voit ou pas le point.
Sous-fonctionnalités	Sauvegarde de la cartographie dans le stockage système.
Contraintes	Précision de la zone lésée, s'adapter à l'usager.
Niveau de priorité	Maximale

Titre	Afficher une cartographie
Objectif	Afficher la cartographie sélectionnée par l'utilisateur.
Description	On sélectionne une cartographie générée par l'utilisateur, puis on l'affiche à l'écran.
Niveau de priorité	Moyenne

Titre	Choisir une cartographie
Objectif	Détermine la cartographie que la caméra utilisera.
Description	On sélectionne une cartographie créée préalablement par l'utilisateur, pour l'utiliser dans la caméra.
Niveau de priorité	Moyenne

Titre	Vision en temps réel
Objectif	Afficher une image corrigée.
Description	On active la caméra et pour chaque image, on essaye de corriger les zones lésées en fonction de la cartographie utilisée.
Contraintes	Nombre d'images par seconde, latence due au traitement des images.
Niveau de priorité	Élevée

Solution proposée

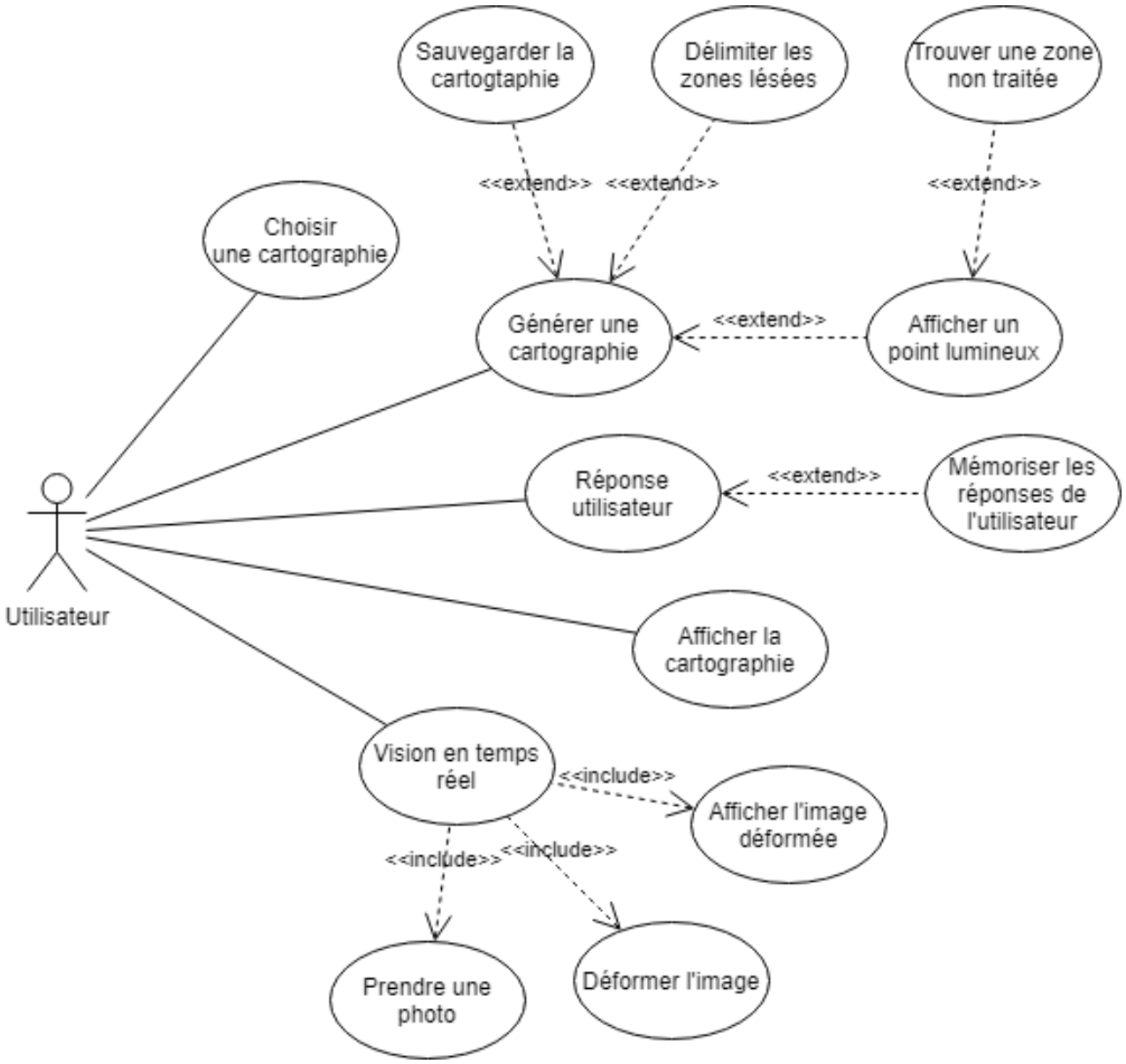
Découpage en parties

Nous présenterons dans cette partie la liste des différents fonctionnalités de l'application :

- Création de point lumineux aléatoire
- Garder en mémoire les zones déjà explorées
- Délimiter les zones lésées
- Création du cartographie des zones lésées
- Développement de l'interface utilisateur
- Télécommande envoyant un signal aux lunettes
- Déformer l'image en fonction de la cartographie

Cas d’utilisations

Voici les différents cas d’utilisation que peuvent effectuer les différents acteurs de notre application.



Hiérarchisation des cas d'utilisation

Nous avons hiérarchisé les cas d’utilisations présentés ci-dessus de la manière suivante, par ordre de priorité:

1. Générer une cartographie
    a. Afficher un point lumineux
    b. Réponse utilisateur
    c. Mémoriser les réponses de l’utilisateur
    d. Délimiter les zones lésées
    e. Sauvegarder la cartographie
2. Vision en temps réel
    a. Prendre une photo
    b. Déformer l’image
    c. Afficher l’image déformée
3. Choisir une cartographie
4. Afficher la cartographie

Description détaillée

Titre	Afficher un point lumineux
Niveau	Sous-fonction
Résumé	Affiche un point lumineux sur l’écran de l’utilisateur
Acteurs secondaires	Écran
Scénario principal	1- L’application fait appel à une fonction permettant de trouver une zone non traitée 2- La fonction renvoie une zone non traitée 3- L’application affiche un point sur l’écran
Extensions	1.a. Toutes les zones ont été explorées 1.a.1. La fonction renvoie un message d’arrêt

Titre	Réponse utilisateur
Niveau	Sous-fonction
Résumé	Réception des réponses affirmatives de l'utilisateur, ou l'absence de réponse
Acteurs secondaires	Télécommande
Scénario principal	1- L'application est en attente d'une réponse de l'utilisateur pendant un cours laps de temps 2- L'utilisateur répond oui 3- L'application marque la zone comme visible jusqu'au prochain test
Extensions	2.a. L'utilisateur dépasse le délai accordé pour répondre 2.a.1. Le point sera traité à nouveau plus tard 3.a. Il n'y a pas assez d'espace 3.a.1. La fonction renvoie un code d'erreur

Titre	Mémoriser les réponses de l'utilisateur
Niveau	Sous-fonction
Résumé	Conserve dans une structure de données l'ensemble des réponses de l'utilisateur
Acteurs secondaires	Stockage
Déclencheur	Réponse utilisateur
Scénario principal	1- L'application enregistre, dans une structure de données, les coordonnées du point et si celui-ci est visible ou non par l'utilisateur 2- Vérification de la cohérence des réponses 3- Les réponses sont sauvegardées dans la structure de données
Extensions	2.a Les réponses ne sont pas cohérentes 2.a.1 On vérifie à nouveau les points concernés 3.a. Il n'y a pas assez d'espace 3.a.1. La fonction renvoie un code d'erreur

Titre	Délimiter les zones lésées
Niveau	Sous-fonction
Résumé	Détermine les zones lésées de l'utilisateur
Acteurs secondaires	Écran
Scénario principal	1- L'application utilise la structure de données créée grâce aux réponses de l'utilisateur 2- Un algorithme délimitera les zones lésées à partir des informations contenues dans la structure de données 3- L'application génère la cartographie

Titre	Sauvegarder la cartographie
Niveau	Sous-fonction
Résumé	Sauvegarde de la cartographie de l'utilisateur dans l'espace de stockage des lunettes
Acteurs secondaires	Stockage
Scénario principal	1- L'application affiche une boîte de dialogue demandant à l'utilisateur s'il veut sauvegarder sa cartographie 2- L'utilisateur répond oui 3- L'application sauvegarde la cartographie

Titre	Prendre une photo
Niveau	Sous-fonction
Résumé	Prise d'une photo par la caméra
Acteurs secondaires	Caméra
Scénario principal	1- L'application demande à la caméra d'effectuer une photo 2- La caméra prend la photo 3- La caméra envoie la photo à l'application.

Extensions	1.a. Il n'y a pas assez d'espace 1.a.1. L'application renvoie un code d'erreur 1.b. L'application n'a pas la permission d'utiliser la caméra 1.b.1. L'application demande l'autorisation au système d'exploitation 1.c. La caméra est utilisée par une autre application 1.c.1. L'application renvoie un code d'erreur

Titre	Déformer l'image
Niveau	Sous-fonction
Résumé	L'application déforme l'image en fonction de la cartographie de l'utilisateur
Acteurs secondaires	Application
Scénario principal	1- L'application reçoit la photo de la caméra 2- L'application traite la photo et la déforme 3- L'application retourne la photo déformée

Titre	Afficher l'image déformée
Niveau	Sous-fonction
Résumé	L'image capturée par la caméra est déformée par un algorithme afin que tout élément de l'image soit visible
Acteurs secondaires	Écran
Déclencheur	Vision en temps réel
Scénario principal	1- L'activité transmet l'image déformée à afficher 2- L'application affiche l'image déformée

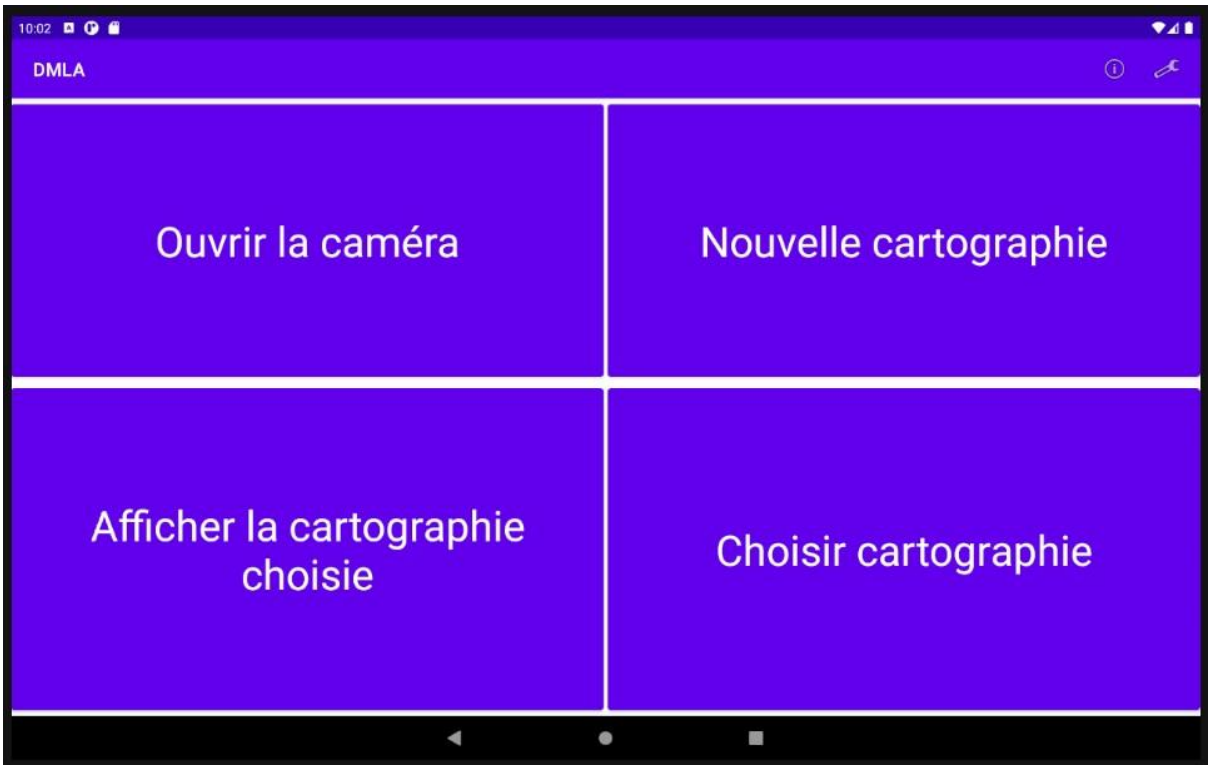
Titre	Choisir une cartographie
Niveau	Sous-fonction
Résumé	L'utilisateur choisit une autre cartographie
Acteurs secondaires	Stockage
Préconditions	Une cartographie a déjà été enregistrée
Scénario principal	1- L'utilisateur clique sur un bouton pour choisir une autre cartographie 2- L'application charge toute les cartographies sauvegardées sur le stockage 3- L'utilisateur choisi une des cartographies
Extensions	2.a L'application ne peut pas afficher les cartographies sauvegardées sur le stockage 2.a.1 L'application renvoie un code d'erreur

Titre	Générer une cartographie
Niveau	But Utilisateur
Résumé	L'application teste la vision de l'utilisateur en affichant des points pour détecter les zones lésées et enregistrer les résultats dans un fichier
Acteurs secondaires	Écran, Stockage
Scénario principal	1- L'utilisateur souhaite générer la cartographie de son champ de vision 2- L'application affiche des points lumineux 3- L'application attend la réponse de l'utilisateur 4- L'application délimite les zones lésées grâce aux données récoltées 5- Les réponses sont sauvegardées dans un fichier
Extensions	3.a Il n'y a pas encore assez de données 3.a.1 On retourne à l'étape 2

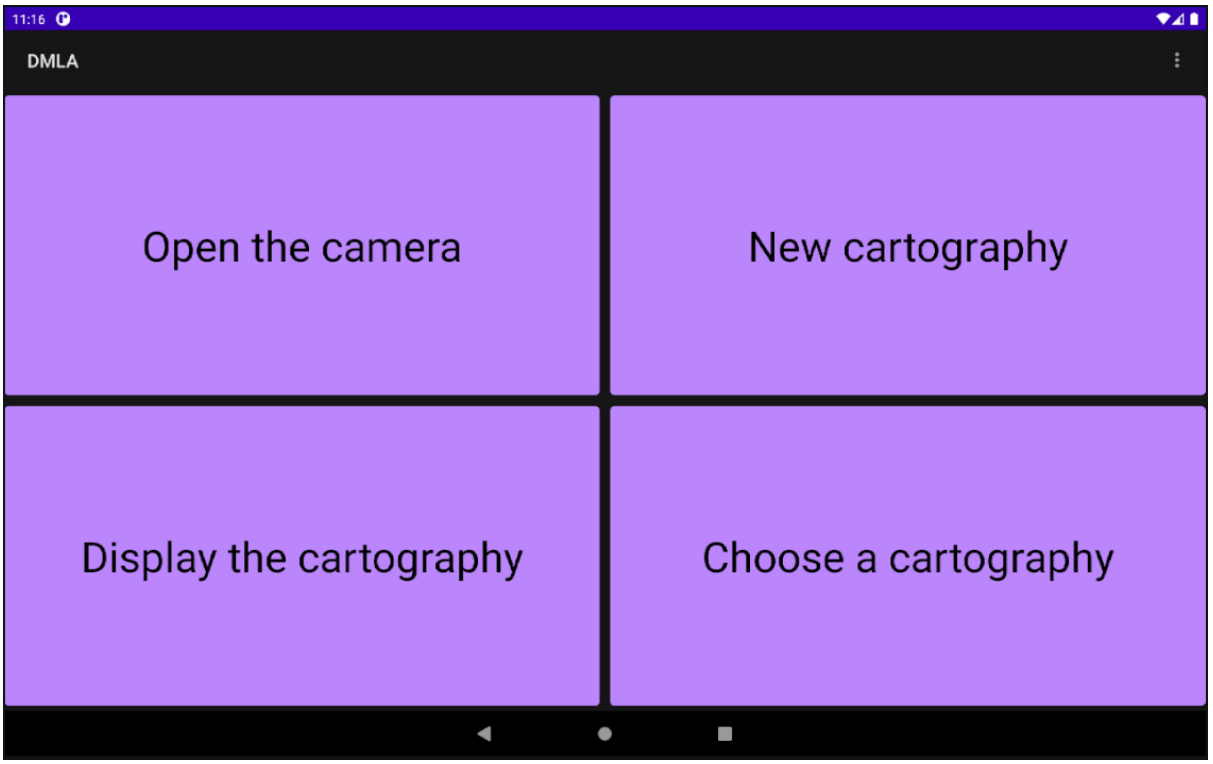
Titre	Vision en temps réel
Niveau	But Utilisateur
Résumé	L'utilisateur voit à travers la caméra des lunettes pour avoir une vision corrigée
Acteurs secondaires	Caméra, Stockage, Écran
Scénario principal	1- L'utilisateur ouvre la caméra 2- La caméra capture une photo 3- L'application déforme l'image 4- L'application affiche l'image déformée
Extensions	1.a La caméra ne s'ouvre pas 1.a.1 La fonction renvoie un code d'erreur 2.a Il n'y a plus assez d'espace pour capturer de photos 2.a.1 La fonction renvoie un code d'erreur

Titre	Afficher la cartographie
Niveau	But Utilisateur
Résumé	L'utilisateur peut voir la cartographie du champ de vision de ses yeux
Acteurs secondaires	Stockage, Écran
Scénario principal	1- L'utilisateur souhaite visualiser la cartographie 2- Un explorateur de fichiers s'ouvre 3- L'utilisateur sélectionne la cartographie à afficher 4- L'application affiche la cartographie sur l'écran
Extensions	2.a L'explorateur de fichiers ne s'ouvre pas 2.a.1 La fonction renvoie un code d'erreur 3.a. Il n'y a pas de cartographie 3.a.1. L'application demande à l'utilisateur d'en enregistrer une d'abord

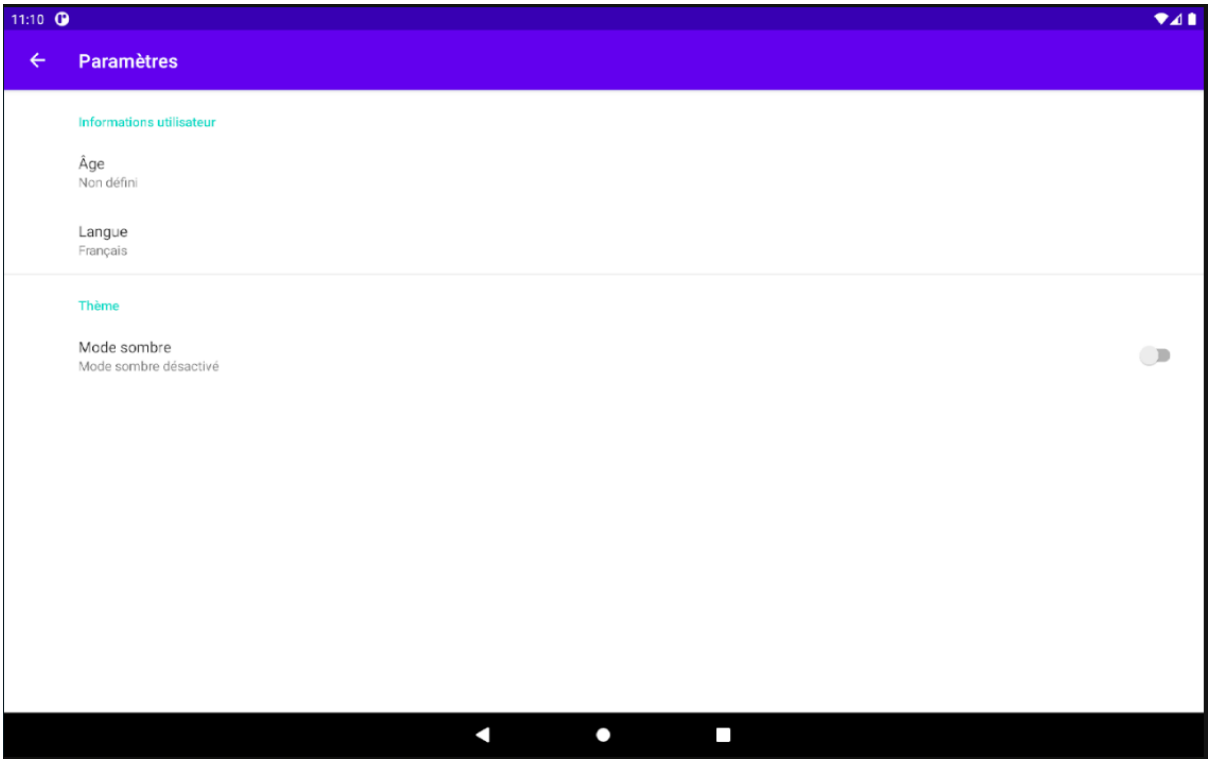
Interface homme-machine



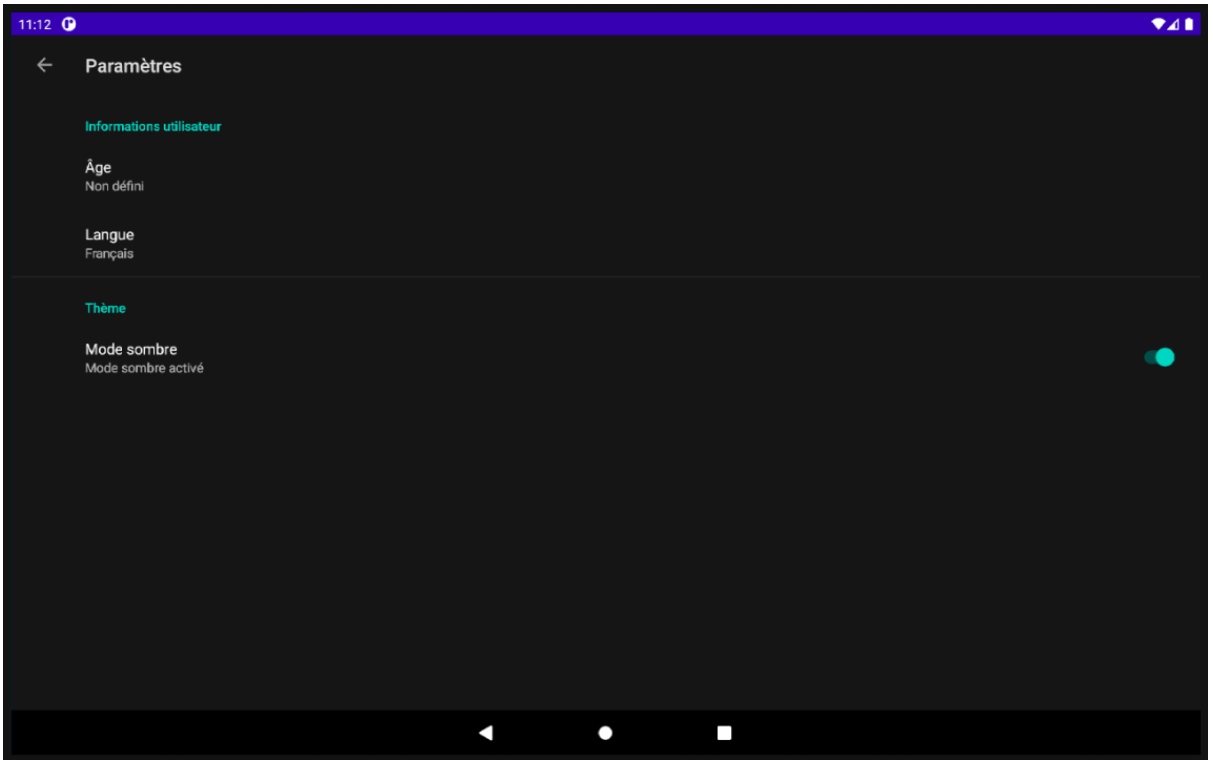
Menu principal



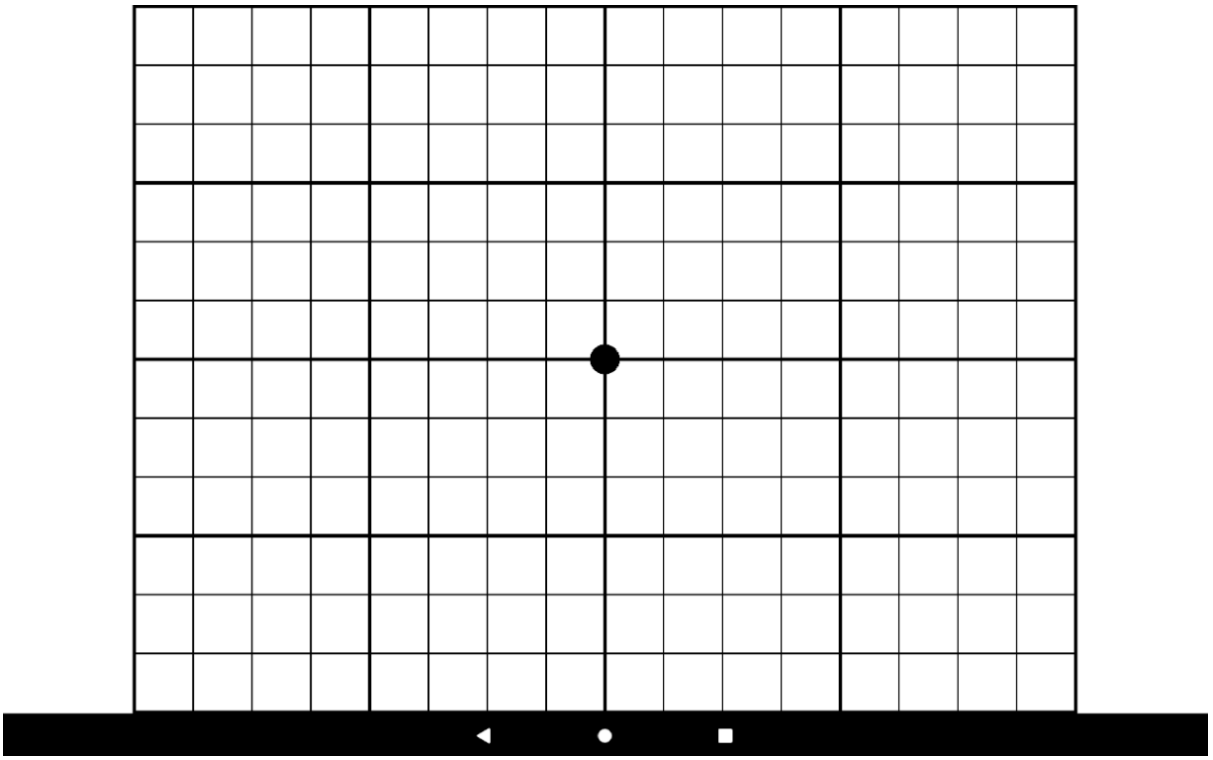
Menu principal en anglais



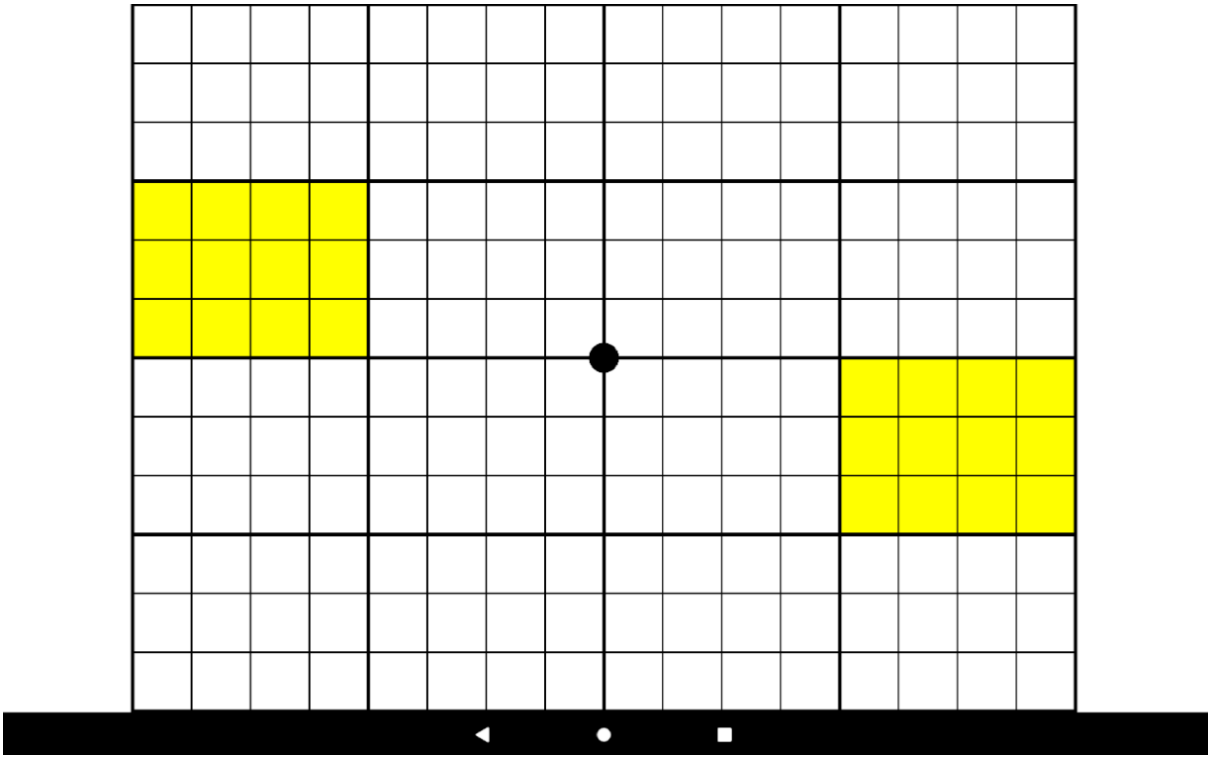
Paramètres de l'application



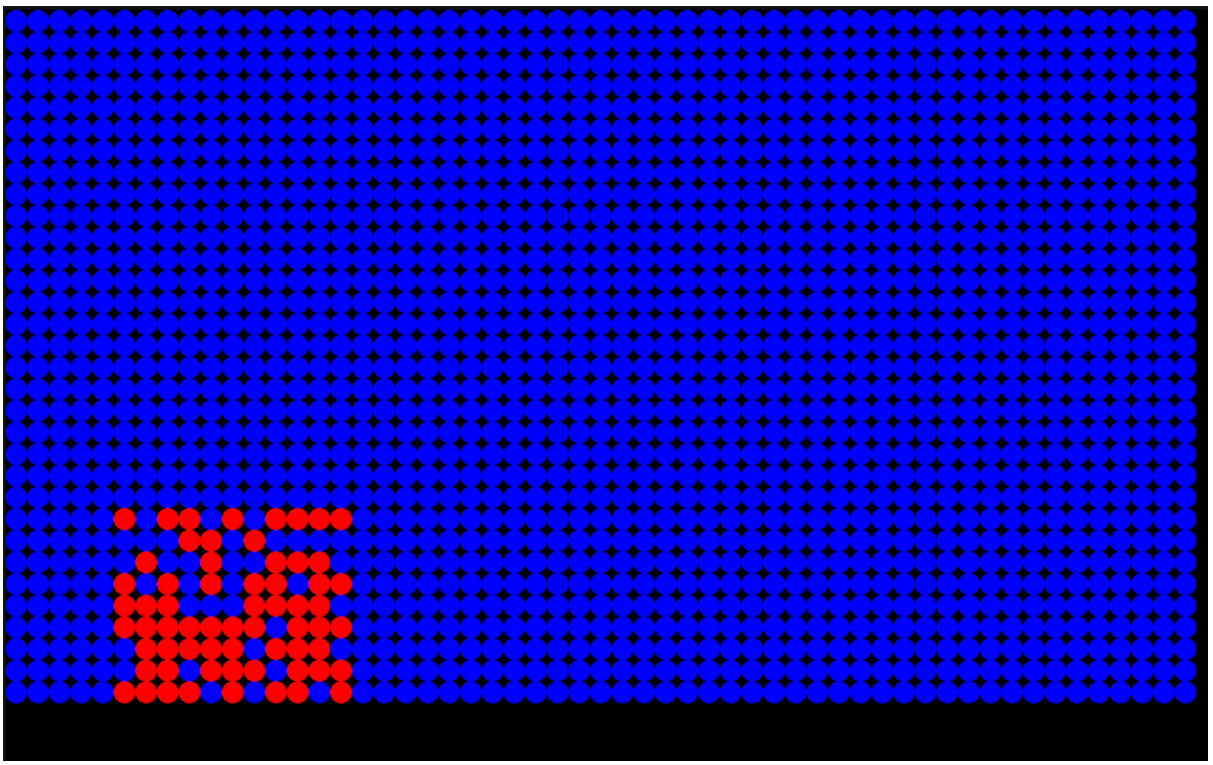
Mode sombre



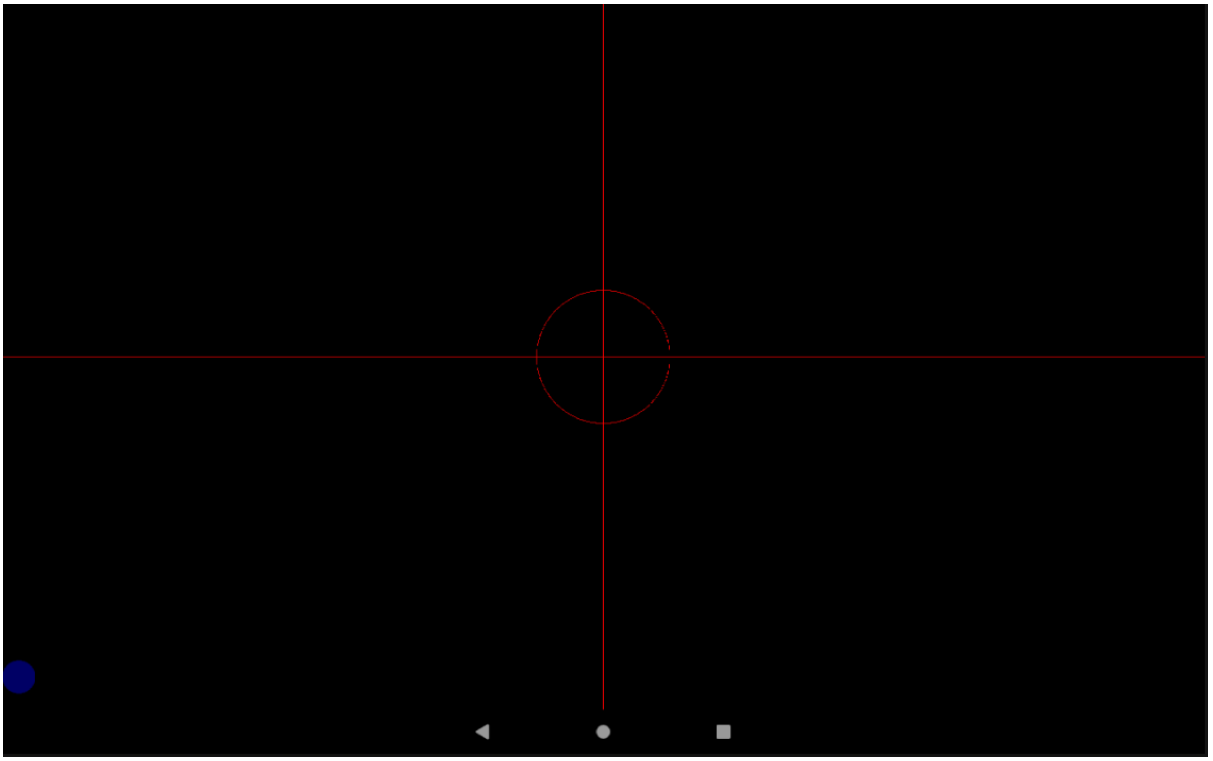
Grille d’amsler (aucune zone sélectionnée)



Grille d’amsler (deux zones sélectionnées)



Exemple d’affichage de cartographie



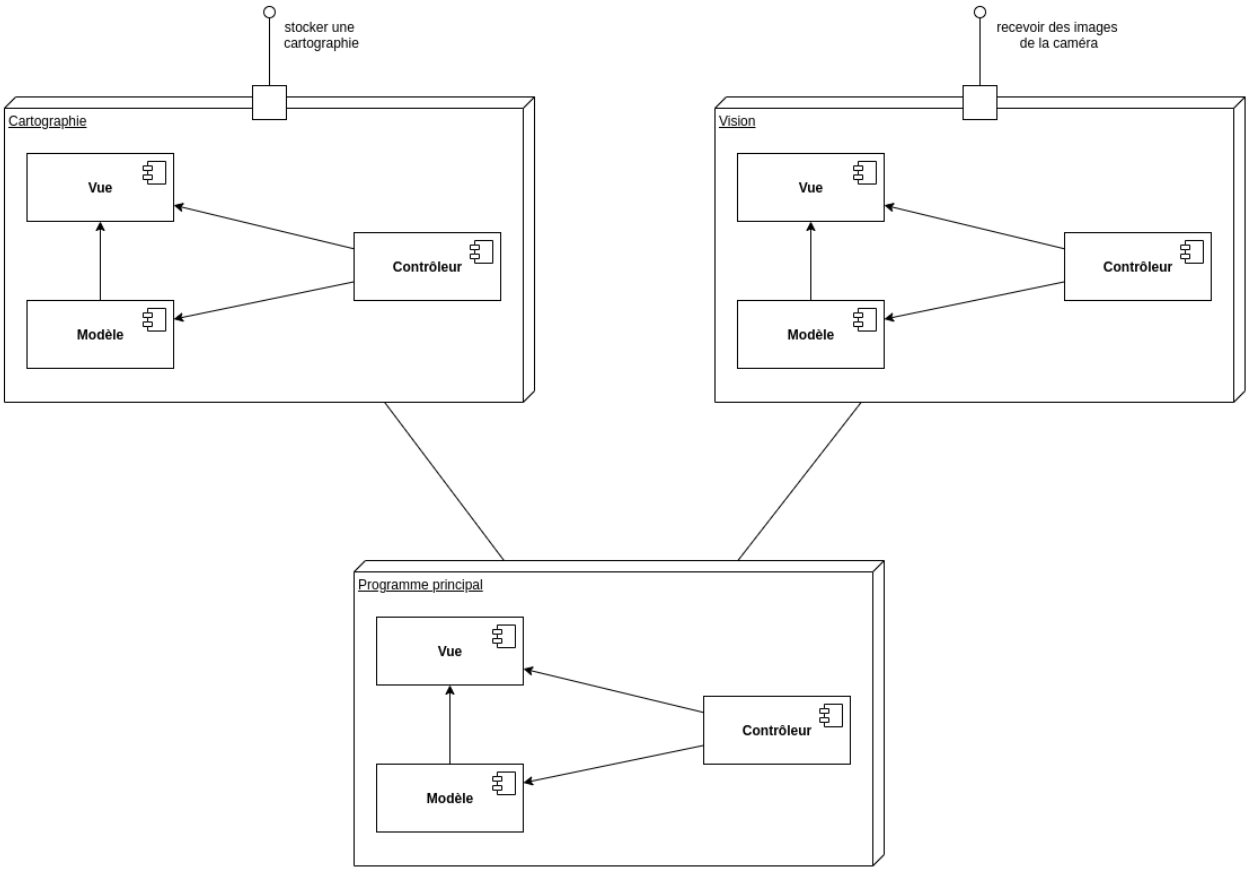
Nouvelle Cartographie en cours

Conception

Architecture globale

L'application consiste en un ensemble de 3 modules : programme principal, cartographie et vision. Chaque module a l'architecture Modèle-Vue-Contrôleur (MVC). Elle est appropriée pour les systèmes interactifs, particulièrement ceux impliquant plusieurs vues du même modèle de données.

Le programme principal est en quelque sorte le point d'entrée de l'application. Il permet d'accéder aux autres modules, qui eux ont permettent d'effectuer certaines actions telles que l'utilisation de la caméra pour avoir une vision corrigée mais également de générer une cartographie, d'en choisir une ou de la stocker en mémoire.
Le module "cartographie" permet de générer une cartographie selon les réponses de l'utilisateur à l'affichage des points sur l'écran.
La vision se charge d'afficher l'image lue par la caméra ainsi que la déformer selon un certain algorithme afin d'obtenir les résultats attendus.



Architecture globale

Conception détaillée

Afficher un point

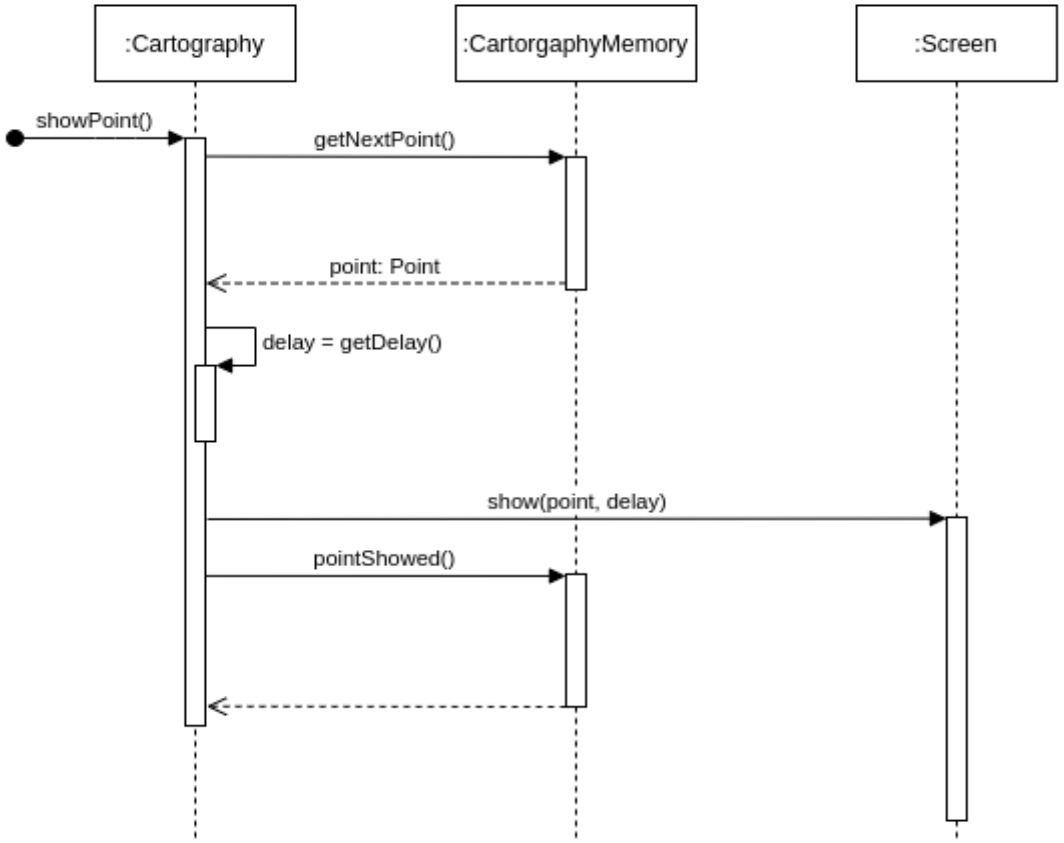


Diagramme de séquence du cas d'utilisation "afficher un point"

Le but ici est d'afficher un point lumineux sur l'écran. Lors de l'appel de la fonction show Point(), on fait appel à la fonction getNextPoint(), cette fonction renvoie les coordonnées du points à afficher. On fait appel par la suite à la fonction getdelay() afin de déterminer le temps d'attente entre deux points. On affiche le point sur l'écran puis on laisse le temps d'attente démarrer.

Réponse utilisateur

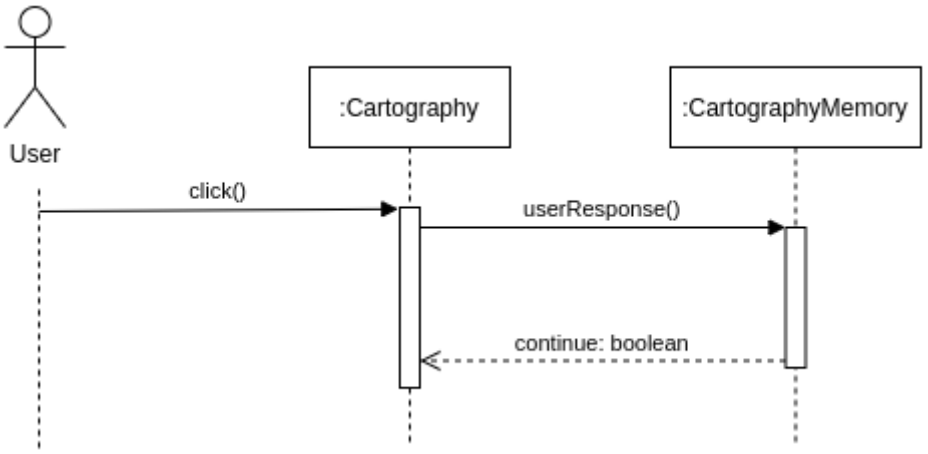


Diagramme de séquence du cas d'utilisation "réponse utilisateur"

Le but de cette fonction est d'enregistrer la réponse de l'utilisateur, l'absence de réponse pendant le temps d'attente signifie que l'utilisateur n'a pas vu le point ou a mal répondu. Quand l'utilisateur appuie sur le bouton, la fonction click() est appelée, userResponse() enregistre la réponse.

Mémoriser les réponses de l'utilisateur

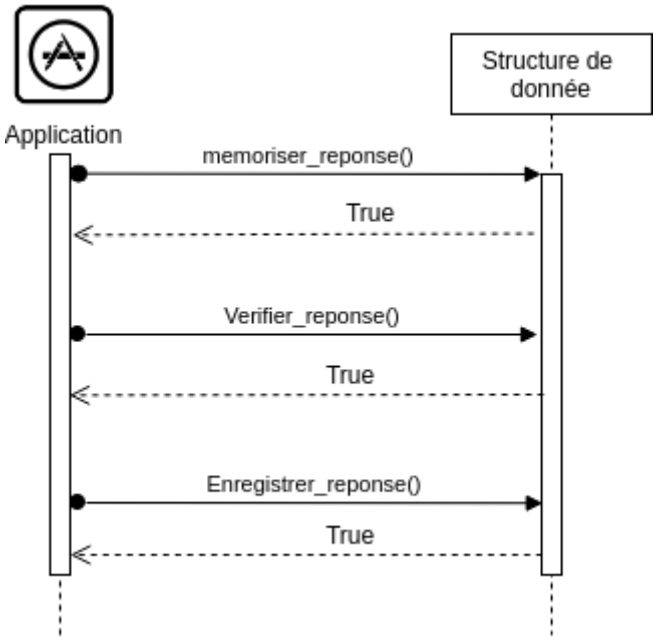


Diagramme de séquence du cas d'utilisation "mémoriser les réponses de l'utilisateur"

Le but de cette fonction est d'enregistrer la réponse de l'utilisateur dans la mémoire. La première fonction appelée est memoriser_reponse qui va sauvegarder les réponses de l'utilisateur lors de la création d'une cartographie. Une fois qu'on a fini de traiter tous les points, on passe à l'étape de vérification durant laquelle on s'assure de la validité des

réponses de l'utilisateur en lui affichant de nouveau certains points. Enfin, une fois la vérification établie, on enregistre définitivement les réponses.

Générer une cartographie

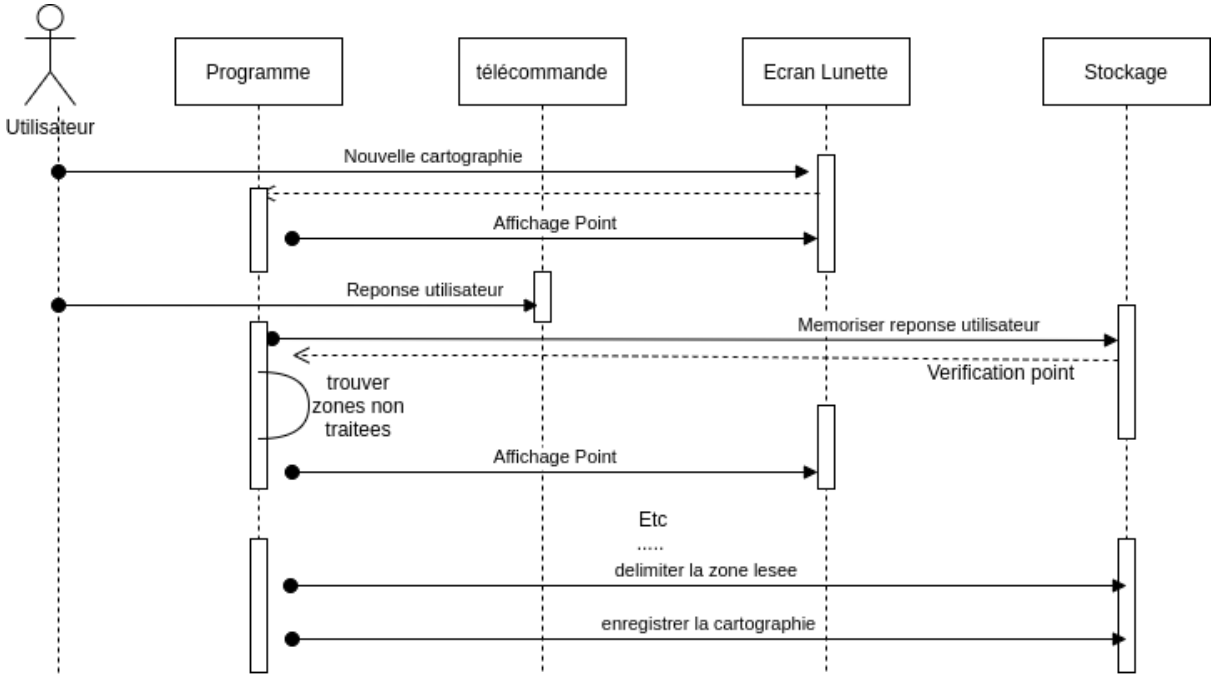


Diagramme de séquence du cas d'utilisation "générer une cartographie"

Le but de cette fonction est de générer une cartographie, on commence par générer une cartographie vide, on génère un point lumineux, on traite la réponse utilisateur (qui peut être une réponse ou une absence de réponse), elle est mémorisée puis vérifiée. On détermine par la suite une nouvelle zone pour afficher un point, puis on réitère l'opération.

Délimiter les zones lésées

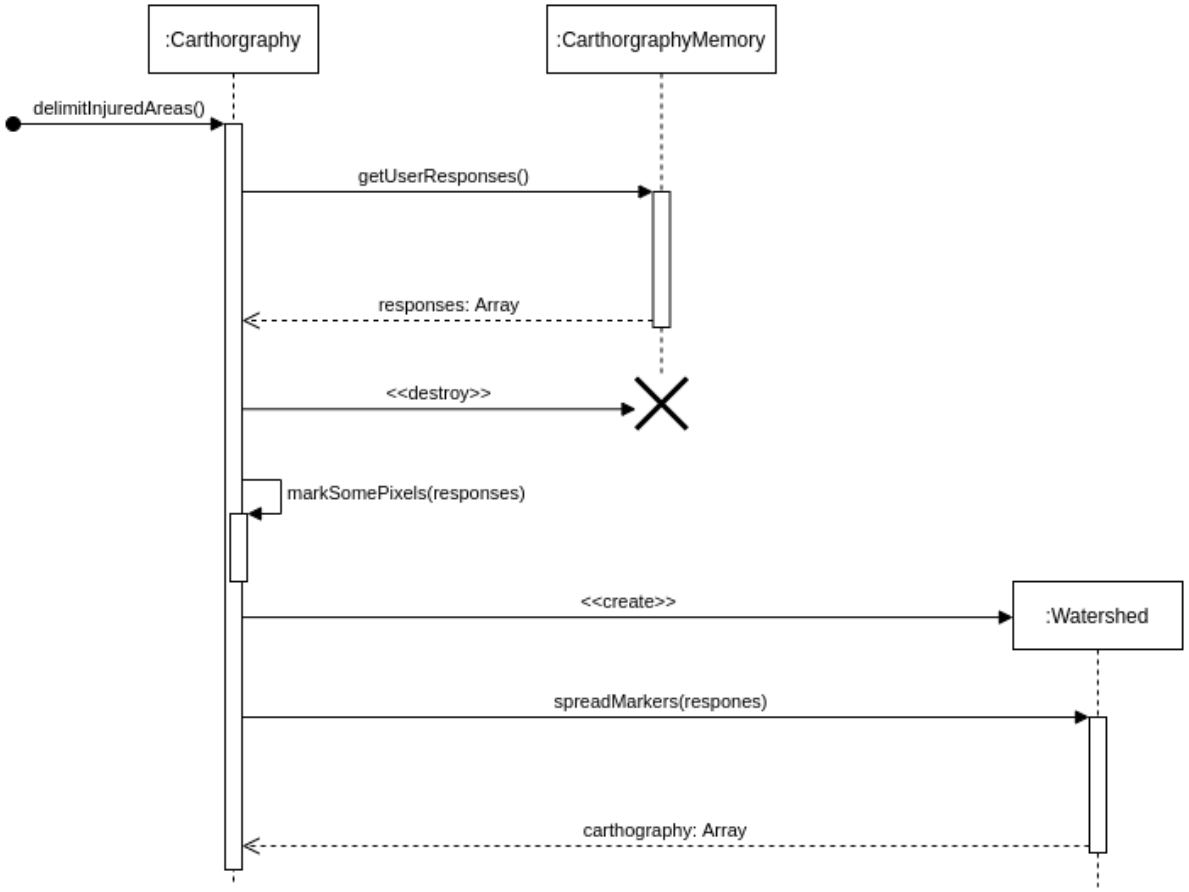


Diagramme de séquence du cas d'utilisation "délimiter les zones lésées"

Afin de délimiter les zones lésées, on récupère d'abord les réponses de l'utilisateur sous forme de tableau. On marque certaines zones (ou pixels) en fonction des données du tableau et enfin on initialise le composant Watershed qui s'occupera d'étaler les marqueurs afin d'établir une cartographie.

Sauvegarder la cartographie:

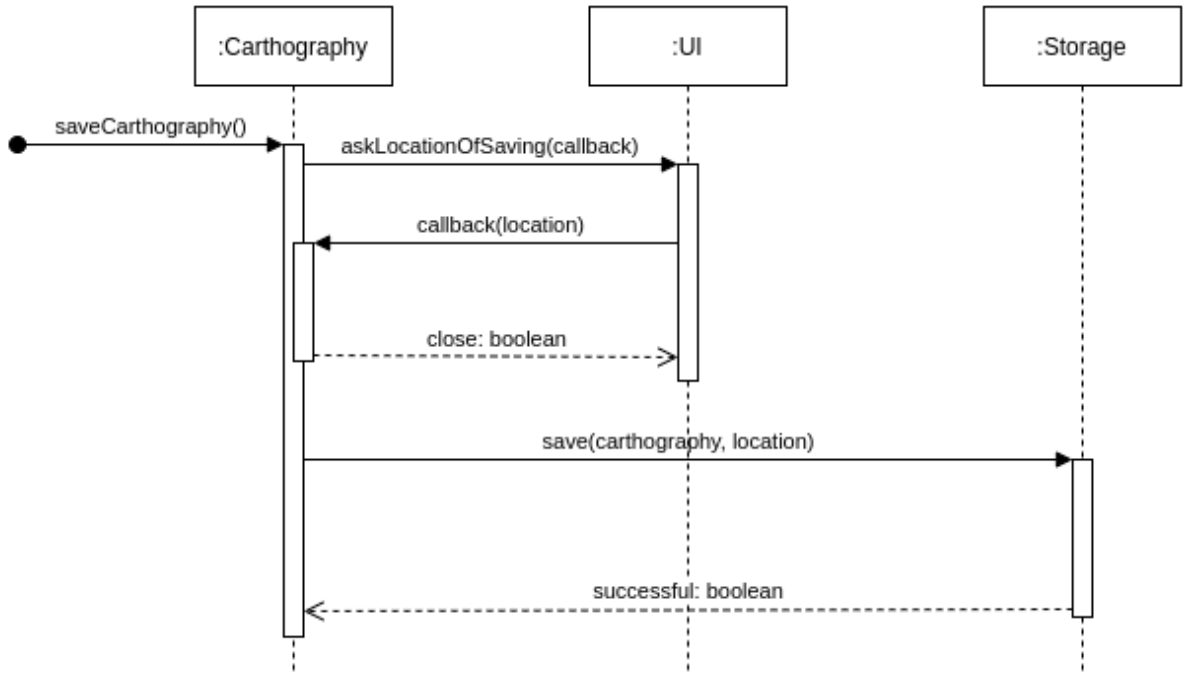


Diagramme de séquence du cas d'utilisation "sauvegarder la cartographie"

Le but de cette fonction est de sauvegarder la cartographie. Tout d'abord on demande à l'utilisateur où souhaite t il enregistrer ses fichiers, une fois qu'il a répondu, le callback est déclenché avec comme paramètre l'emplacement choisi. Ensuite on demande au composant Storage d'enregistrer nos données en spécifiant la cartographie et l'emplacement. Enfin, on reçoit une réponse qui indique si oui ou non la sauvegarde a été correctement effectuée.

Vision en temps réel

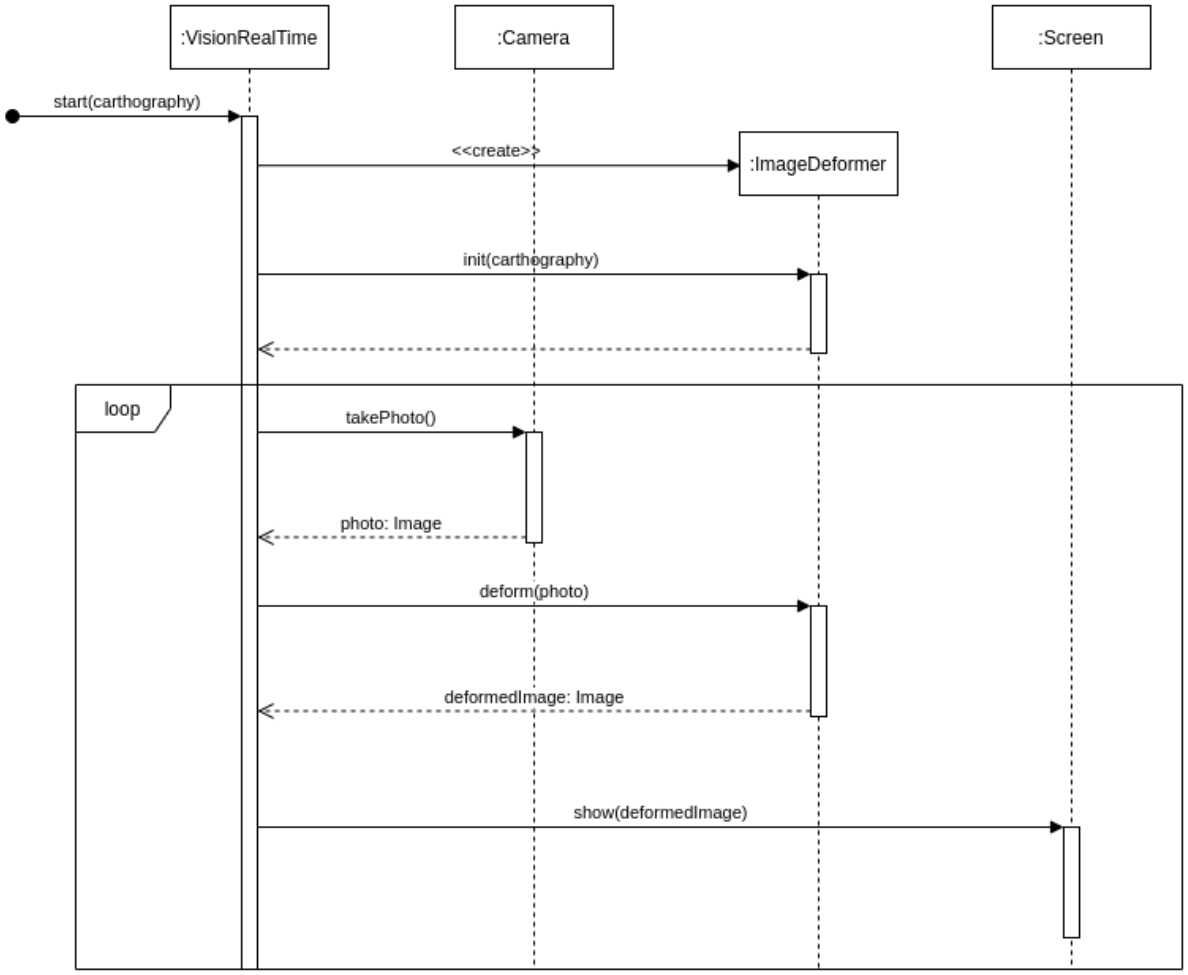
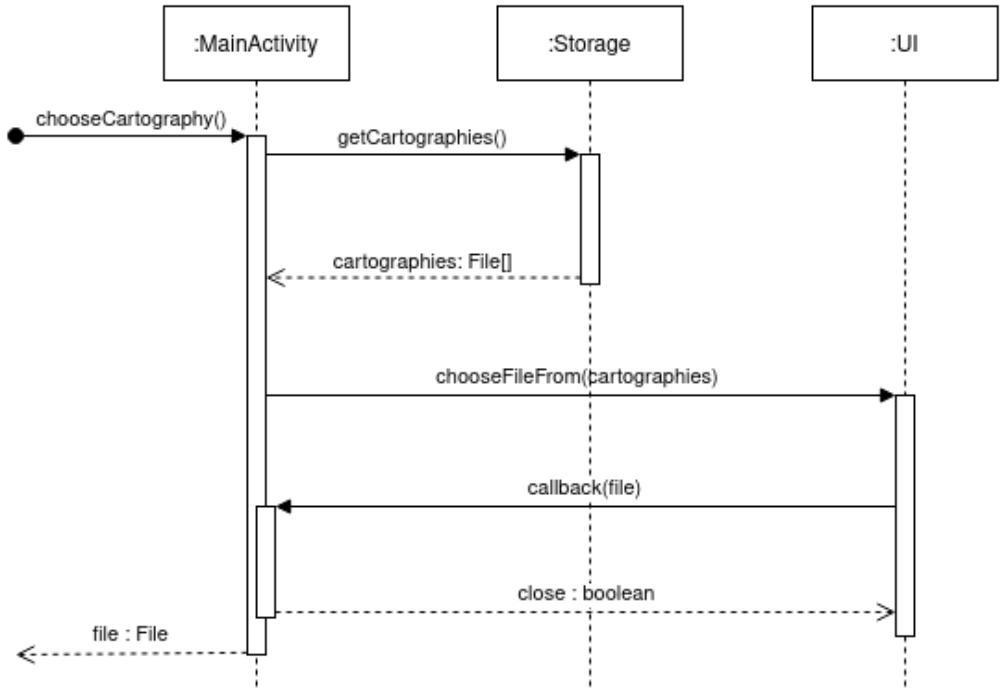


Diagramme de séquence du cas d'utilisation "vision en temps réel"

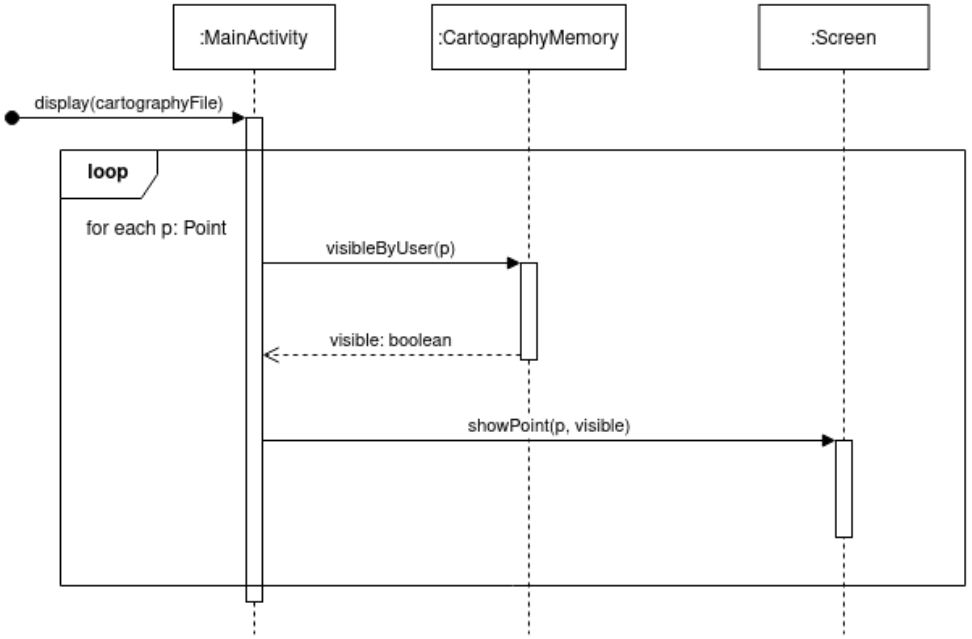
Dans ce diagramme, on représente la vision en temps réel. Tout d'abord, il faut savoir quelle cartographie sera utilisée, c'est pour cela qu'on la spécifie en paramètre au début. On commence par initialiser le composant ImageDeformer en fonction de la cartographie qui, comme son nom l'indique, s'occupera de déformer l'image. Ensuite, on commence une boucle infinie (tant que l'utilisateur ne décide pas d'arrêter). Dans cette boucle, il s'agit de demander une photo au composant Caméra, une fois la photo reçue, on l'a transmets au composant ImageDeformer pour qu'il applique un algorithme de déformation et nous renvoie l'image déformée. Enfin, on envoie l'image déformée à l'écran de l'utilisateur pour qu'il l'affiche.

Choisir une cartographie



Ce diagramme montre comment on choisit une cartographie. D'abord, on récupère tous les fichiers de cartographies depuis le stockage. On demande, ensuite, à l'utilisateur d'en choisir un et on renvoie le fichier sélectionné.

Afficher la cartographie



Le but de ce cas d'utilisation est d'afficher la cartographie choisie à l'aide du cas d'utilisation choisir une cartographie. Lors de l'appel de la fonction qui affiche la cartographie on passe le fichier de cartographie. Dans la fonction on affiche chaque point avec une couleur en fonction de si l'utilisateur voit le point ou pas.

Choix techniques

Étant donné que les technologies de développement de l'application et l'interface ont déjà été fixées, nous allons nous attarder sur l'environnement de l'application et celui de développement.

- Type de l'application : Application android
- Langage de programmation : Java
- Plateforme de versonning: GitHub
- Plateforme de développement d'application : Android Studio
- Plateformes de communication : E-mailing, Discord, WhatsApp, Google Drive

Méthodes choisies

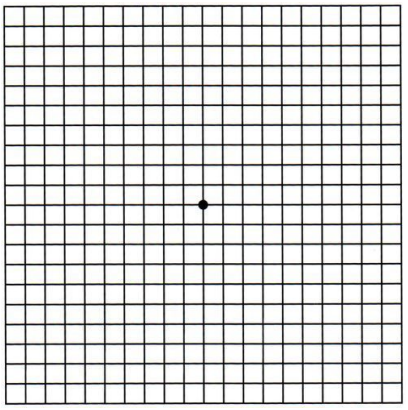
- Watershed

En ce qui concerne la détection des zones lésées, nous avons opté pour l'utilisation d'un algorithme de ligne de partage des eaux pour détecter les contours des zones lésées sur une image donnée.
Ensuite, en ayant connaissance des zones lésées, un autre algorithme sera appliqué sur l'image pour la déformer afin que toutes les zones lésées soient visibles.
Pour segmenter l'image, on réalise deux tâches à la fois : la reconnaissance des tâches d'intérêt et la délimitation de leurs contours. Ces deux tâches sont étroitement liées. En effet, pour reconnaître un objet dans une image, il faut avoir déterminé son contour et pour déterminer le contour d'un objet il faut l'avoir au préalable reconnu. L'intensité du point lumineux reste importante car elle permet de savoir si un point donné appartient au contour de la tâche.
Avec cet algorithme, on étudie un ensemble fini de points, munis d'une intensité, en un certain nombre de régions (dans le cas de présence de plusieurs tâches). Dans ce cadre, Watershed permet de partitionner les pixels en un ensemble de régions connexes séparées par un contour fermé. Elle constitue donc, par nature, un outil bien adapté à la segmentation.

- DICOM (Digital imaging and communications in medicine)

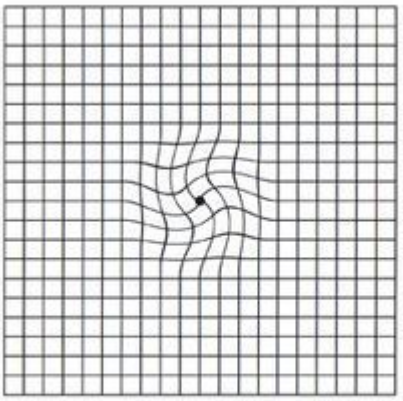
Pour la sauvegarde de la cartographie, nous avons pensé au format de fichier DICOM. En effet, ce format est un standard pour la gestion informatique des données issues de l'imagerie médicale. Indépendant des technologies (scanner, IRM, etc.), et des constructeurs, il permet de standardiser l'accès aux résultats d'imagerie médicale, ce format est donc propice à l'utilisation médicale.
Le fichier est constitué d'une suite de champs, et en fonction des données que l'on souhaite enregistrer (dans notre cas, la cartographie du champ de vision de l'utilisateur) nous rempliront ces champ là.
Nous allons utiliser le champ 'Ophtalmic Photography' pour notre application, dans lequel on pourra préciser quelques détails sur le patient, sur les pixels de l'image, etc.
Par ailleurs, ce format de fichier est celui utilisé par les ophtalmologues, il sera donc possible d'importer un fichier DICOM afin de l'utiliser dans notre application.
Ensuite, pour afficher la cartographie, nous utiliserons les données stockées dans ce même fichier.

- Amsler Grid

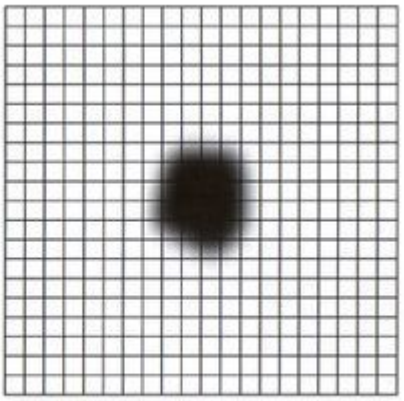


Après avoir implémenté et codé les différentes idées pour délimiter les zones lésés, nous avons réalisé plusieurs tests pour délimiter une zone (imaginaire, que nous avons choisis). La méthode utilisée s'est avérée être très longue et assez ennuyante pour l'utilisateur, la zone de départ est beaucoup trop grande pour être traitée par la méthode choisie, dans notre cas, une absence de réponse signifie que l'utilisateur n'a pas vu le point ou a mal répondu et dans le cas où il y a plusieurs zones non visibles, la méthode reste peu efficace. Nous avons effectué des recherches pour trouver une nouvelle approche pour mieux délimiter la ou les zones non visibles, la grille d'Amsler reste la plus adaptée, une grille de lignes horizontales et verticales, utilisée pour évaluer la partie centrale du champ visuel d'une personne. Une personne atteinte de DMLA voit une tache noire et des lignes ondulées autour de cette tâche, cette grille sera pour nous un prétraitement afin de mieux délimiter la zone de départ, elle sera divisée en un certain nombre de parties qui dépend de la taille de l'écran.

On demandera alors à l'utilisateur de renseigner les zones déformées et non visibles, si ces zones sont différentes et éloignées les unes des autres, cela signifie qu'elles sont distinctes, on traiterai chaque zone séparément, mais si elles sont collées, la tâche s'étale sur plusieurs zones, on fusionne ces zones et on les traitera comme une.



Les lignes vous paraissent déformées, ondulées



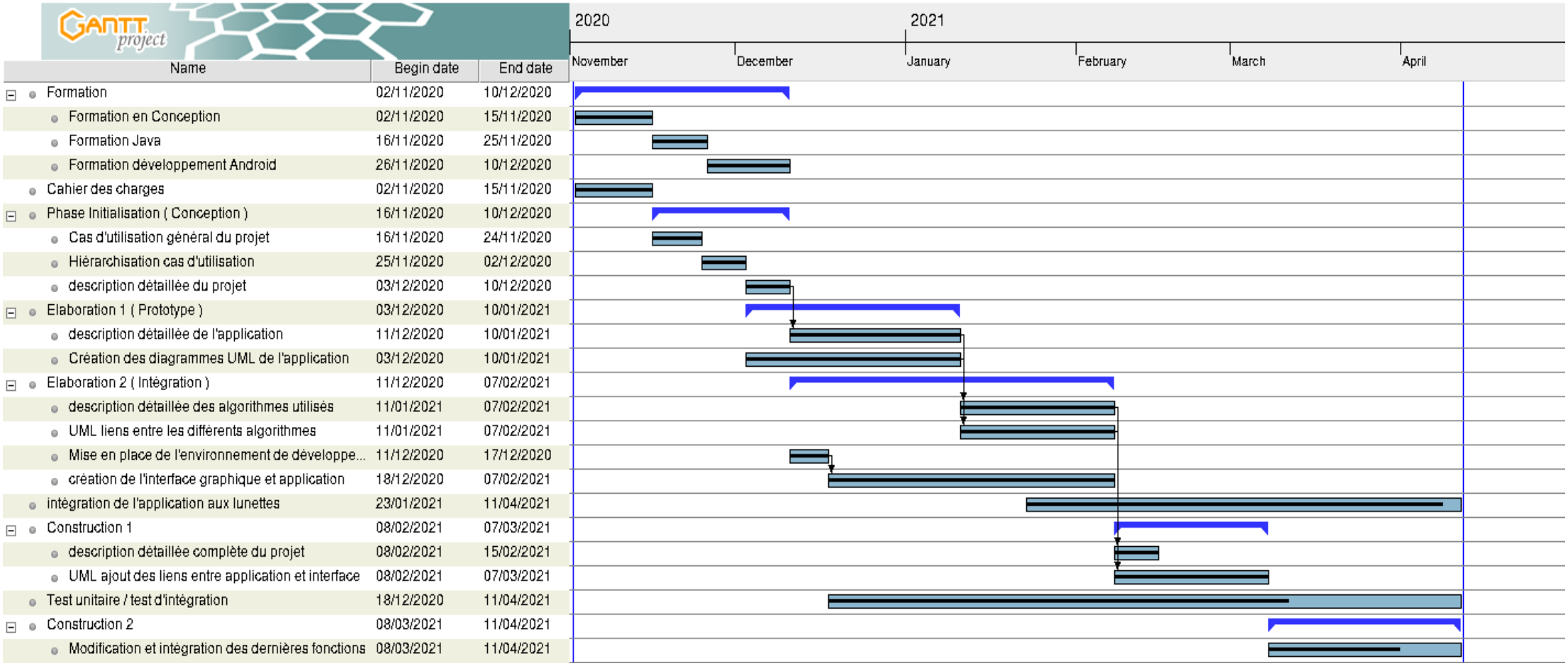
Il existe une tache sombre ou floue dans une zone de la grille.

Planning prévisionnel

Estimation temporelle

L’équipe sera composée de cinq élèves ingénieurs en deuxième année d’ingénieurs informatique (ABOUTARIK Ali, DENGUIR Guilane, DUDIK Sviatoslav, FISSEL-LARAKI Mohamed, TAIBI Sofiane). Chaque personne étant capable de fournir 35 jours de travail ce qui nous donne un total de 175 jours homme dont 20 jours homme de formation.

Planning-Tableau



Name	Begin date	End date
Formation	02/11/2020	10/12/2020
Formation en Conception	02/11/2020	15/11/2020
Formation Java	16/11/2020	25/11/2020
Formation développement Android	26/11/2020	10/12/2020
Cahier des charges	02/11/2020	15/11/2020
Phase Initialisation (Conception)	16/11/2020	10/12/2020
Cas d'utilisation général du projet	16/11/2020	24/11/2020
Hiérarchisation cas d'utilisation	25/11/2020	02/12/2020
description détaillée du projet	03/12/2020	10/12/2020
Elaboration 1 (Prototype)	03/12/2020	10/01/2021
description détaillée de l'application	11/12/2020	10/01/2021
Création des diagrammes UML de l'application	03/12/2020	10/01/2021
Elaboration 2 (Intégration)	11/12/2020	07/02/2021
description détaillée des algorithmes utilisés	11/01/2021	07/02/2021
UML liens entre les différents algorithmes	11/01/2021	07/02/2021
Mise en place de l'environnement de développe...	11/12/2020	17/12/2020
création de l'interface graphique et application	18/12/2020	07/02/2021
intégration de l'application aux lunettes	23/01/2021	11/04/2021
Construction 1	08/02/2021	07/03/2021
description détaillée complète du projet	08/02/2021	15/02/2021
UML ajout des liens entre application et interface	08/02/2021	07/03/2021
Test unitaire / test d'intégration	18/12/2020	11/04/2021
Construction 2	08/03/2021	11/04/2021
Modification et intégration des dernières fonctions	08/03/2021	11/04/2021

BILAN

On obtient un résultat de cartographie plutôt correct, mais pas ‘scientifique’. En effet, le fichier obtenu ne peut être exploité et utilisé pour d’autres fins si ce n’est pour développer la partie de déformation d’image. Cette dernière nécessite seulement de connaître quelles zones de l’écran, l’utilisateur ne peut voir correctement.

Une des principales difficultés de ce projet a été de comprendre la méthode utilisée pour cartographier un œil pour ensuite l’écrire sous forme de code. En effet plusieurs paramètres entrent en compte tels que la durée d’affichage, le temps de réponse,...
Malheureusement, au cours du développement de notre application, un membre de l’équipe a quitté le projet au mois de novembre, nous avons donc perdu un temps précieux, qui représente 20% du temps total de travail, cela nous aurait permis d’aller plus loin dans notre projet.

Certaines fonctionnalités ont été implémentées mais ne sont toujours pas utilisées. Cependant, elles seront indispensables pour un usage futur.
Par exemple, la caméra servira à regarder à travers les lunettes, mais il faudra pour cela créer un filtre, pour déformer l’image, à partir de la cartographie que l’on peut générer pour corriger les zones lésées.
L’utilisateur peut aussi renseigner son âge, qui permettra plus tard de générer une cartographie encore plus précise en se calibrant grâce aux cartographies d’autres personnes saines du même âge.
Enfin, Watershed (ligne de partage des eaux) a été implémenté pour permettre de faire un post-traitement à la cartographie générée afin d’avoir des résultats plus uniformes.