03/06/2014

Eye tracker pour visualiser des images

Projet d'approfondissement

Master of Science HES-SO in Engineering

Orientation : Technologies de l'information et de la communication (TIC)

William Droz

Sous la direction de

Prof. Olivier, Hüsser

Haute école ARC

Accepté par la HES-SO//Master (Suisse, Lausanne) sur proposition de

Prof. David Grunenwald, conseiller du projet d'approfondissement

Neuchâtel, le 25 mai 2014

Prof. Olivier Hüsser	Prof. Didier Rizzotti
Conseiller	Responsable de la filière MRU TIC-Arc

Table des matières

1 Introduction

L'oculométrie[1] a un grand potentiel que de plus en plus d'acteurs tentent t'exploiter. Ce projet a pour but de démontrer ce qu'il est possible de faire avec une technologie accessible (Tobii Rex[2]) sans pour autant en exclure d'autres.

Il a été décidé de faire une application qui permet de naviguer dans des dossiers d'images. Ce projet permettra ainsi d'avoir un aperçu des capacités des appareils d'oculométries. La problématique de l'intégration d'un tel dispositif dans un logiciel sera aussi abordée.

1.1 Définition de l'oculométrie

« L'oculométrie (en anglais Eye-tracking) regroupe un ensemble de techniques permettant d'enregistrer les mouvements oculaires. Les oculomètres les plus courants analysent des images de l'œil humain capturées par une caméra, souvent en lumière infrarouge, pour calculer la direction du regard du sujet. En fonction de la précision souhaitée, différentes caractéristiques de l'œil sont analysées. D'autres techniques sont basées sur les variations de potentiels électriques à la surface de la peau du visage ou encore sur les perturbations induites par une lentille spéciale sur un champ magnétique »

Source : wikipédia révision du 10 juillet 2013

[1] http://fr.wikipedia.org/wiki/Oculom%C3%A9trie
[2] http://www.tobii.com/en/eye-experience/buy/rex/

1.2 Cahier des charges

Le projet consiste en un explorateur d'images. Celui-ci doit permettre la navigation facilitée par le Tobii Rex (eyesTracker). Il doit être possible d'ouvrir les images afin de les visualiser en grand format. L'utilisateur doit être capable de zoomer de façon dirigé (avec les yeux).

1.2.1 Spécifications fonctionnelles

- Permettre la navigation dans des dossiers d'images
- Permettre la sélection d'images
- Permettre d'afficher en grand format les images
- Permettre de zoomer là où le regard se pose en fonction d'une tâche d'action spécifique
- Permettre de naviguer uniquement avec le Tobii Rex
- Flouter légèrement là où le regard ne se pose pas*
- Permettre l'ouverture du programme avec les Tobii Rex*
- Permettre de scroller dans l'image après un zoom*

1.2.2 Spécifications non-fonctionnelles

- Doit fonctionner sur Windows 8
- Doit fonctionner avec le .NET 4.5

[*] -> Facultatif

1.2.3 Modification

Durant la réunion du 14 mai 2014[3] il a été convenu de modifier le cahier de charges. Voici la nature des changements :

- Simplification de la partie de navigation (plus besoin d'avoir un explorateur de fichier)
- Utilisateur du capteur "THEEYETRIBE[4]"
- Si le temps le permet, ajouter un bac-à-sable pour restreindre les mouvements de l'image à son cadre.

Il a été jugé plus intéressant de pouvoir changer de capteur facilement que d'avoir une exploration du système de fichier complète.

[3] https://github.com/wdroz/eyePA/wiki/notesPV-14-mai
[4] http://theeyetribe.com/

2 État de l'art

Voici un état des lieux de ce qu'il se fait actuellement en matière d'oculométrie. Ce chapitre aborde les bonnes pratiques à prendre en compte, une brève introduction aux techniques de captures « du regard » et terminera avec une sélection de domaines d'applications.

2.1 Les bonnes pratiques

En matière d'oculométrie il est nécessaire de faire attention à plusieurs points. En effet, il pourrait être tentant de dire « Quand l'utilisateur cligne des yeux, alors fait l'action X ». Alors que justement, ceci est une erreur.

Chez Tobii par exemple, ils utilisent une touche d'activation (sur le clavier). Ainsi il suffit de regarder la zone que l'on désire activer, puis enfin appuyer sur cette touche.

2.1.1 Ce qu'il ne faut pas faire

Voici une liste non-exhaustive des choses à éviter

Piège	Raison
Utiliser le clignement de l'œil pour interagir	L'œil n'est pas fait pour faire des actions
Utiliser une gestuelle des yeux pour interagir	L'œil n'est pas fait pour faire des actions
Utiliser des zones d'interactions de surface faible	Les outils ne sont pas assez précis
Demander l'attention de l'utilisateur trop longtemps	L'utilisateur peut être distrait et/ou se fatiguer.

2.2 Les différentes technologies

Au fil des années, plusieurs technologies ont vu le jour avec plus ou moins de succès. Nous avons essayé de les catégoriser.

2.2.1 Les lunettes

Les lunettes ne sont pas à proprement parler une technologie d'enregistrement oculaire, mais plutôt un moyen d'exploiter des techniques.

+	−
Permet d'être très proche des yeux et donc d'être plus précis	Conflit avec des lunettes correctives Demande l'achat d'un appareil

2.2.2 Les caméras - vidéo-oculographie

Utiliser des caméras semblent une bonne idée. Malheureusement cela ne donne pas de bons résultats. C'est pour ça qu'on combine les caméras avec des lumières infrarouges pour amplifier la brillance de la pupille.

Les premières versions ne sont pas très sexy

Figure 1 vidéo-oculographie

Cela nous donne un aperçu des progrès réalisés avec les Tobii Rex.

2.2.3 électro-oculographique

Technique qui mesure les différences de potentiels électriques provoqués par la rotation des yeux. Cette technique est très peu utilisée car ce n'est pas précis pour reconnaître où l'utilisateur regarde.

2.2.4 Galvanométrique

Mesure des signaux électriques à l'aide d'un champ magnétique. Cette technique est très précise mais a le désavantage de nécessiter une lentille.

2.2.5 Reflet cornéen

Cette technique ressemble un peu à celle de la vidéo-oculographie. On utilise plusieurs éclairages infrarouges afin de pouvoir détecter des variations sur le reflet de la cornée.

Les Tobii Rex utilisent cette technique

2.3 Domaines d'applications

Voici les domaines principaux où est utilisés l'oculométrie.

2.3.1 Médical

Il y a plusieurs sous-domaines, voici quelques exemples :

- Détecter des dyslexies chez un patient[5]
- Etudier la compétence d'un chirurgien[6]
- Evaluer des étudiants en médecine[7]

2.3.2 Conception d'interfaces graphiques[8]

Il n'est pas toujours évident de bien concevoir une interface graphique. En posant des eyes tracker sur un certain nombre de volontaire, nous pouvons recueillir des informations précieuses qui peuvent nous permettre de modifier notre interface.

Voici quelques informations qui peuvent nous intéresser :

- Qu'est-ce que l'utilisateur regarde en premier ?
- Quel chemin parcourt ces yeux ?
- Est-ce qu'il fait des allez-retours ?
- Est-ce qu'il doit regarder longtemps le menu pour comprendre comment ça marche ?

2.3.3 Les jeux

Les jeux-vidéos essayent sans arrêt d'innover et de proposer des nouvelles façons de jouer.

2.3.4 Militaire

Les pilotes utilisent des casques[9] qui leurs permettent d'être plus efficaces.

[55] http://www.club-44.ch/media-c44/c44-p-53315f6501d43.pdf
[6] http://eyecomresearch.com/eyetrackingresearch/eye-tracking-as-a-medical-training-device/
[7] http://www.smivision.com/fileadmin/user_upload/downloads/case_studies/cs_smi_medicaldiagnosis.pdf
[8] http://www.tobii.com/fr/eye-tracking-research/global/research/usability/
[9] http://en.wikipedia.org/wiki/Helmet-mounted_display

2.4 Les nouveautés à attendre

Toutes les entreprises ne sont pas bavardes sur leurs recherches et développements avec l'oculométrie. Tobii nous ont quand même renseigné sur le sujet.

2.4.1 Tobii pour les jeux[10]

Chez Tobii ils veulent tenter de fournir du matériel suffisamment réactif pour les joueurs. Car les Tobii Rex ne sont pas adaptées aux joueurs. Si ceux-ci font des gestes brusques, les yeux ne seront plus détectés pendant quelques secondes.

Toutefois certain jeux ne nécessitent pas de prendre en compte ces contraintes pour joueurs « nerveux ». Citons par exemple le jeu gratuit « HearthStone [11]».

Figure 2 HearthStone - interface durant un combat

Un travail d'automne au niveau Bachelor serait peut-être suffisant pour créer une interface entre le jeu et l'appareil d'oculométrie. Par exemple déplacer la souris avec « AutoIt »[12] là où le regard se pose et ajouter deux-trois contrôles.

[10] http://www.engadget.com/2014/01/03/tobii-steelseries-eye-tracking-game-accessory/
[11] http://eu.battle.net/hearthstone/fr/
[12] http://www.autoitscript.com/site/autoit/

3 Planification

Ce chapitre arborera tout ce qui touche à la planification, telle que la méthode de développement, la planification initial et après la première itération. En fin de chapitre, nous avons effectué une analyse des déviations.

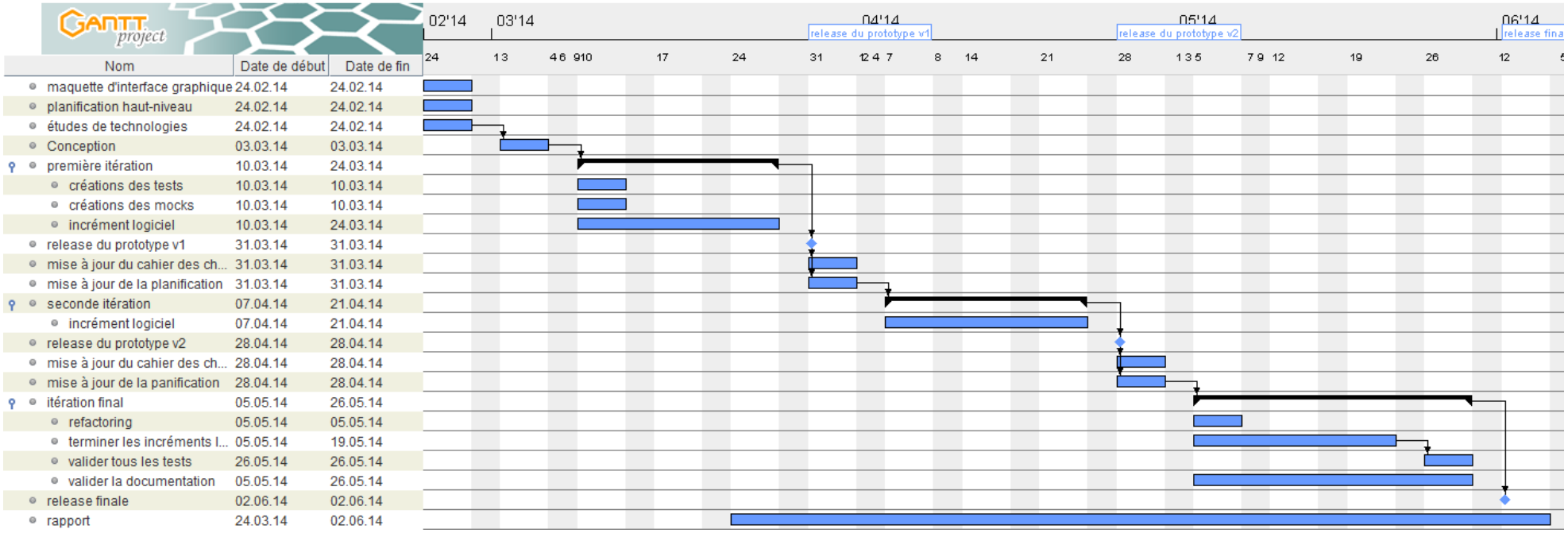
3.1 Méthode de développement

Nous avons choisi une méthode semi-itérative. À Chaque nouvelle itération, il y a un incrément logiciel (dans le sens ajout de features) ainsi qu'une amélioration des features précédentes.

Cette technique va permettre d'avoir une application utilisable dès la première itération.

Avec plus de temps, la méthode RAD avec prototype jetable aurait été meilleure, car elle permet de se baser sur l'expérience acquise sur le prototype précédant pour améliorer le prochain. Ceci est très utile quand on ne maîtrise pas ou peu la technologie dans laquelle on développe.

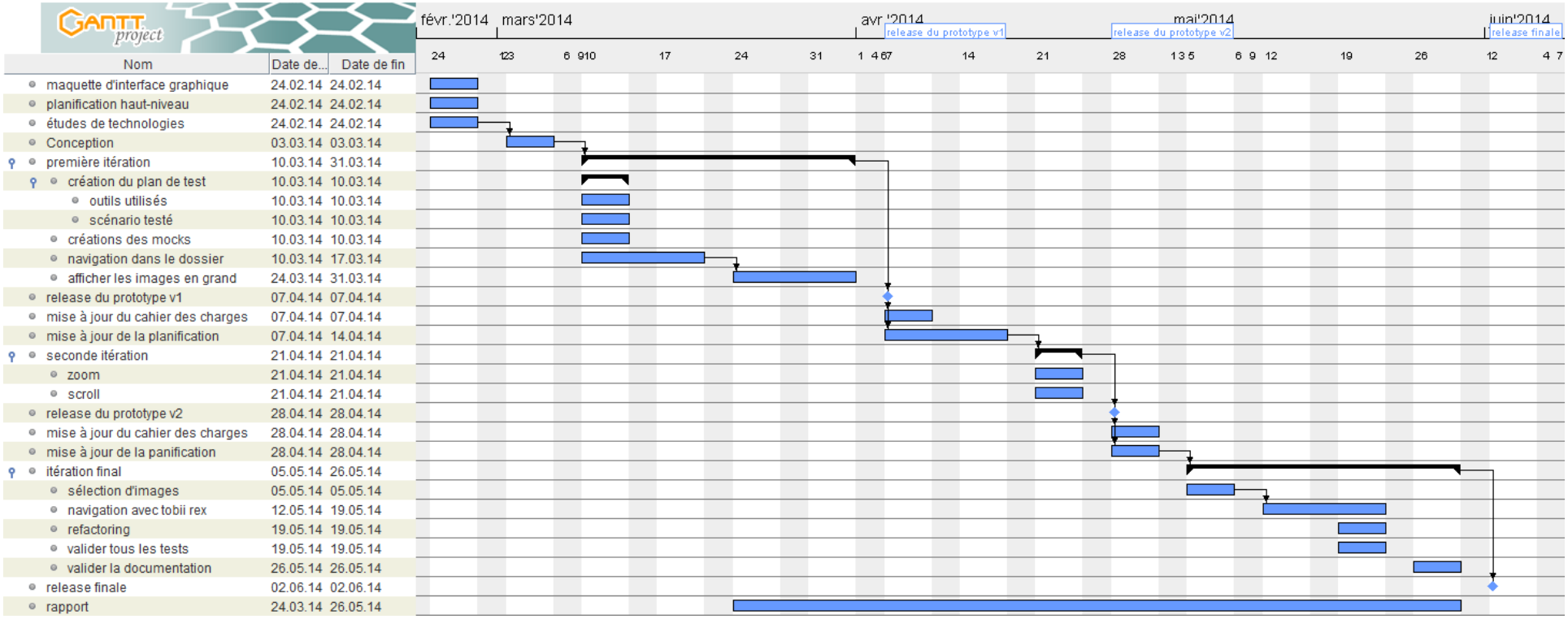
3.2 Planification initiale



Nom	Date de début	Date de fin
maquette d'interface graphique	24.02.14	24.02.14
planification haut-niveau	24.02.14	24.02.14
études de technologies	24.02.14	24.02.14
Conception	03.03.14	03.03.14
première itération	10.03.14	24.03.14
créations des tests	10.03.14	10.03.14
créations des mocks	10.03.14	10.03.14
incrément logiciel	10.03.14	24.03.14
release du prototype v1	31.03.14	31.03.14
mise à jour du cahier des ch...	31.03.14	31.03.14
mise à jour de la planification	31.03.14	31.03.14
seconde itération	07.04.14	21.04.14
incrément logiciel	07.04.14	21.04.14
release du prototype v2	28.04.14	28.04.14
mise à jour du cahier des ch...	28.04.14	28.04.14
mise à jour de la panification	28.04.14	28.04.14
itération final	05.05.14	26.05.14
refactoring	05.05.14	05.05.14
terminer les incréments l...	05.05.14	19.05.14
valider tous les tests	26.05.14	26.05.14
valider la documentation	05.05.14	26.05.14
release finale	02.06.14	02.06.14
rapport	24.03.14	02.06.14

3.3 Planification après la première itération

Nous avons modifié le planning afin de le rendre plus cohérant. Nous avons décidé de faire le zoom et le scroll dans la seconde itération afin d'avoir l'écran principale complet dès la fin de celle-ci. Il nous a fallu donc inverser avec le menu d'exploration des fichiers qui a été relayé à la dernière itération.



Nom	Date de...	Date de fin
maquette d'interface graphique	24.02.14	24.02.14
planification haut-niveau	24.02.14	24.02.14
études de technologies	24.02.14	24.02.14
Conception	03.03.14	03.03.14
première itération	10.03.14	31.03.14
création du plan de test	10.03.14	10.03.14
outils utilisés	10.03.14	10.03.14
scénario testé	10.03.14	10.03.14
créations des mocks	10.03.14	10.03.14
navigation dans le dossier	10.03.14	17.03.14
afficher les images en grand	24.03.14	31.03.14
release du prototype v1	07.04.14	07.04.14
mise à jour du cahier des charges	07.04.14	07.04.14
mise à jour de la planification	07.04.14	14.04.14
seconde itération	21.04.14	21.04.14
zoom	21.04.14	21.04.14
scroll	21.04.14	21.04.14
release du prototype v2	28.04.14	28.04.14
mise à jour du cahier des charges	28.04.14	28.04.14
mise à jour de la panification	28.04.14	28.04.14
itération final	05.05.14	26.05.14
sélection d'images	05.05.14	05.05.14
navigation avec tobii rex	12.05.14	19.05.14
refactoring	19.05.14	19.05.14
valider tous les tests	19.05.14	19.05.14
valider la documentation	26.05.14	26.05.14
release finale	02.06.14	02.06.14
rapport	24.03.14	26.05.14

3.4 Les déviations

Nous avons eu quelques décalages entre ce que nous avions planifié et ce qui s'est réellement passé. Voici donc la description et la raison de ces décalages.

La première itération a pris plus de temps que nous l'avions prévu. L'affichage des images en grand format via un canevas n'a pas été aussi facile que nous pensions.

La seconde itération a pris beaucoup plus de temps que prévu. L'utilisation planifiée de la semaine de vacance durant cette itération n'a pas suffi à terminer rapidement celle-ci. La gestion du zoom et du scroll a été revu à plusieurs reprises.

La modification du cahier de charges a modifié complétement la nature de la dernière itération. Celle-ci n'est donc plus à prendre en compte.

4 Analyse

Ce chapitre concerne l'analyse et les réflexions effectuées afin de répondre aux exigences du projet. Nous avons analysé le fonctionnement du SDK fournit par Tobii, les choix des technologies et des diagrammes des cas d'utilisations.

4.1 Fonctionnement du EyeX SDK

Le SDK fournit par Tobii permet de simplifier considérablement le développement d'une application « standard ». Ce chapitre va synthétiser son fonctionnement. Son but n'est toutefois par de remplacer la lecture du « developer's Guide » fournit par Tobii.

Il y a trois composants qui interagissent entre eux :

- **EyeX Engine**
  - Sélectionne le contrôle[13] que l'utilisateur regarde et/ou actionne.
  - Informe l'application via un « Event » des intentions de l'utilisateur.
- **EyeX Controller**
  - Fournit au eyeX Engine les coordonnés du regard de l'utilisateur.
- **Application**
  - Indique à l'EyeX Engine les coordonnés de ces contrôles.
  - Réagit aux « Events » envoyés par l'EyeX Engine.

[13] Appelé Interactor dans le jargon Tobii

4.1.1 Interactor

Les interactors définissent les zones à l'écran avec lesquelles l'utilisateur peut interagir. Voici un exemple de l'interface de Google avec trois Interactors :

Figure 3 Interface de google.ch

4.1.2 Query

Quand l'utilisateur regarde l'écran, l'EyeX Engine demande s'il y a des interactors dans la zone du regard de l'utilisateur. Cette demande est adressée à l'application plusieurs fois par secondes.

L'application est sensé lui répondre une liste d'interactors.

4.1.3 Events

Lors que l'utilisateur désire activer un contrôle, il va appuyer sur la touche d'activation. Celle-ci est interceptée par l'EyeX Engine. Celui-ci détermine quel est l'interactor que l'utilisateur regarde (grâce aux query vu avant).

L'EyeX Engine va ensuite envoyer un Event à l'application l'informant quel interactor a été activé.

4.1.4 Déroulement d'un scénario simple

Nous allons montrer pas à pas le déroulement sur l'interface Google vu précédemment.

Figure 4 Interface google.ch avec Interactors et le regard de l'utilisateur

Comme dirait le visiteur du futur[14], voilà ce qui va se passer :

Figure 5 Le visiteur du futur qui dit "voilà ce qui va se passer"

1. L'EyeX Controller va informer l'EyeX Engine où regarde l'utilisateur
2. L'EyeX Engine va faire une Query à l'application pour lui demander s'il y a des interactors dans la zone que regarde l'utilisateur (en rouge sur la Figure 4).
3. L'application va répondre avec une liste contenant les interactors 1 et 2.
4. L'utilisateur appui sur la touche d'activation
5. L'EyeX Engine détermine quel est l'interactor le plus probable entre le 1 et le 2. Il choisit le 2 car l'intersection est plus importante.
6. L'EyeX Engine envoi un Event à l'application lui indiquant que l'interactor 2 vient d'être activé.
7. L'application fait le code que le développeur a déterminé.

[14] http://www.levisiteurdufutur.com/

4.1.5 Limitations

Ce SDK simplifie le travail du développeur, sauf si celui-ci veut sortir des sentiers battus (ce qui est notre cas).

En effet, si on désire définir plusieurs comportements différents en fonctions de plusieurs touches différentes… Cela n'est pas pris en charge. Il n'est pas possible d'avoir des events du genre « Interactor X activé avec la touche A » et « Interactor Y activé avec la touche B ».

Pour notre application, nous avons donc choisit de nous en tenir aux query comme interaction avec l'EyeX Engine. Ceux-ci nous informent de la zone regardée par l'utilisateur.

4.2 Choix des technologies

Il y a deux choix de SDK pour les Tobii Rex. L'un est en C++, l'autre en C#. Pour chaque langage, des exemples sont fourni.

Pour le C#, les exemples uilisent des « WinForms [15]». Pour le C++ ils sont en GDI[16].

Dans ce chapitre, nous allons tenter de comparer les deux voies possibles.

4.2.1 C++ avec Qt[17]

La première solution consiste à utiliser C++ avec le Framework Qt pour l'interface graphique. Nous n'évoquons par GDI ici, car nous avons moins de 30 ans. 

Qt est la seule manière moderne de créer des interfaces graphiques en C++ que nous maîtrisons.

Voici les avantages et les inconvénients pour cette méthode :

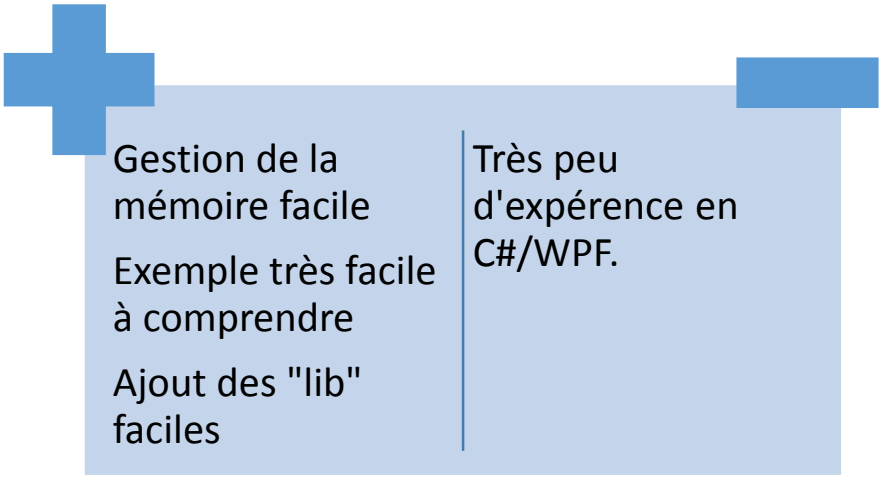
Déjà fait de nombreux projets avec Qt	Gestion de la mémoire
Permet de faire des traitements rapide (si on veut faire des effets sur les images, par exemple).	Les exemple sont très difficiles à lire/comprendre[vor annexe "*Code C++ pour Tobii Rex*"].

[15] http://en.wikipedia.org/wiki/Windows_Forms
[16] http://en.wikipedia.org/wiki/Graphics_Device_Interface
[17] http://qt-project.org/

4.2.2 C# avec WPF[18]

Les WinForms sont de moins en moins utilisés au profit de WPF, car ceux-ci sont plus malléables en séparant bien le code et l'interface graphique. Nous avons donc opté pour WPF comme technologie qui va avec C#.

Voici les avantages et les inconvénients pour cette méthode :



4.2.3 Conclusion

Le projet étant dans un cadre des études, nous choisissons « **C# avec WPF** » afin de pouvoir nous perfectionner avec cette technologie qui est assez présente dans les entreprises.

Malheureusement, il y a un « coût » avec cette méthode : En effet, vu que nous ne maitrisons pas très bien la technologie, le risque de faire un logiciel peu maintenable et peu optimisé est grand.

[18] http://en.wikipedia.org/wiki/Windows_Presentation_Foundation

4.3 Diagrammes des cas d'utilisations

Voici le diagramme « UseCase » sur lequel nous allons nous baser pour la conception :

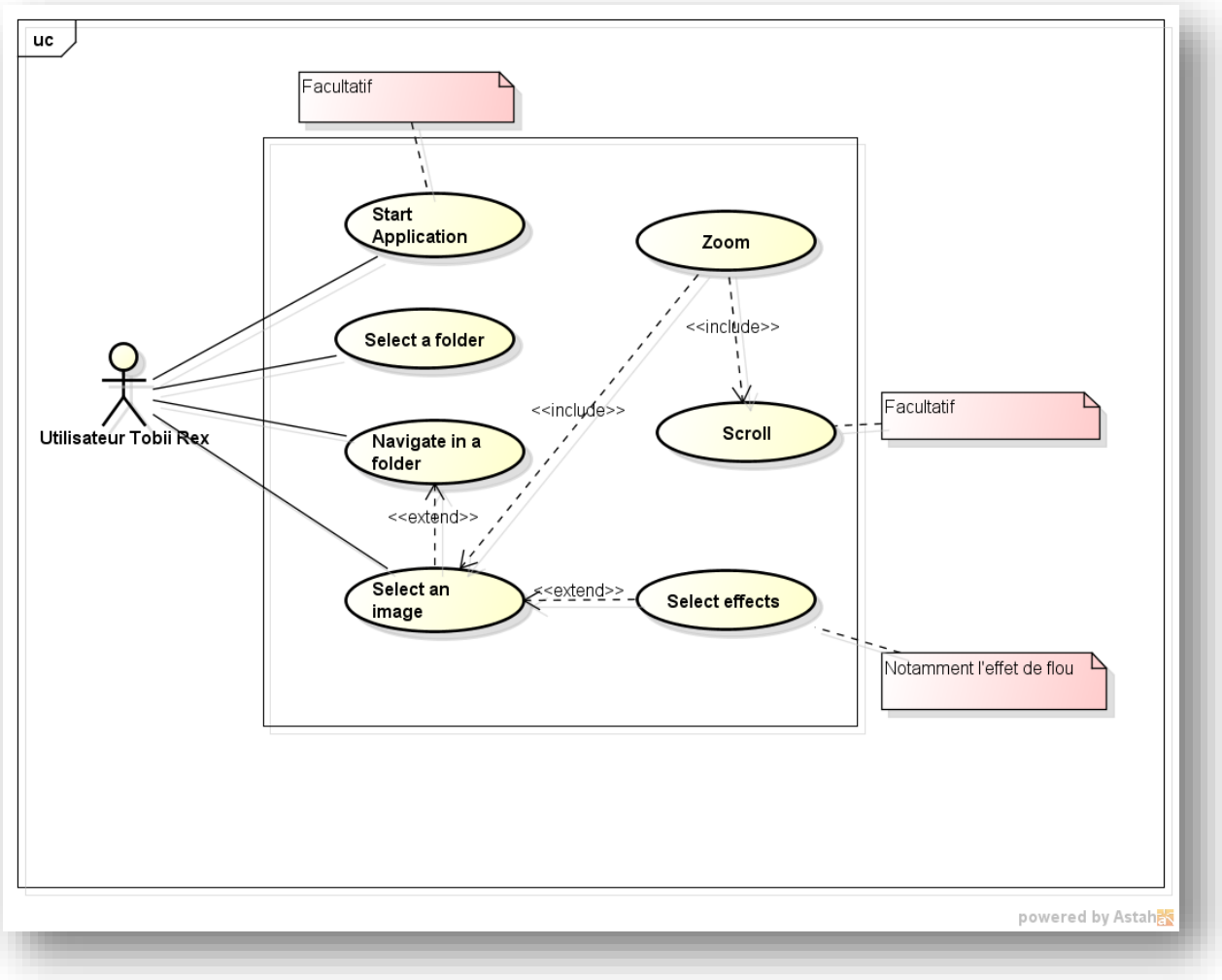


Figure 6 UseCase de l'analyse

5 Conception

Ce chapitre contient les éléments qui ont guidé l'implémentation. Nous avons donc les diagrammes de classes et les interfaces graphiques

5.1 Diagrammes de classes

Pour plus de lisibilité, nous avons séparé le diagramme en deux parties.

5.1.1 Première partie

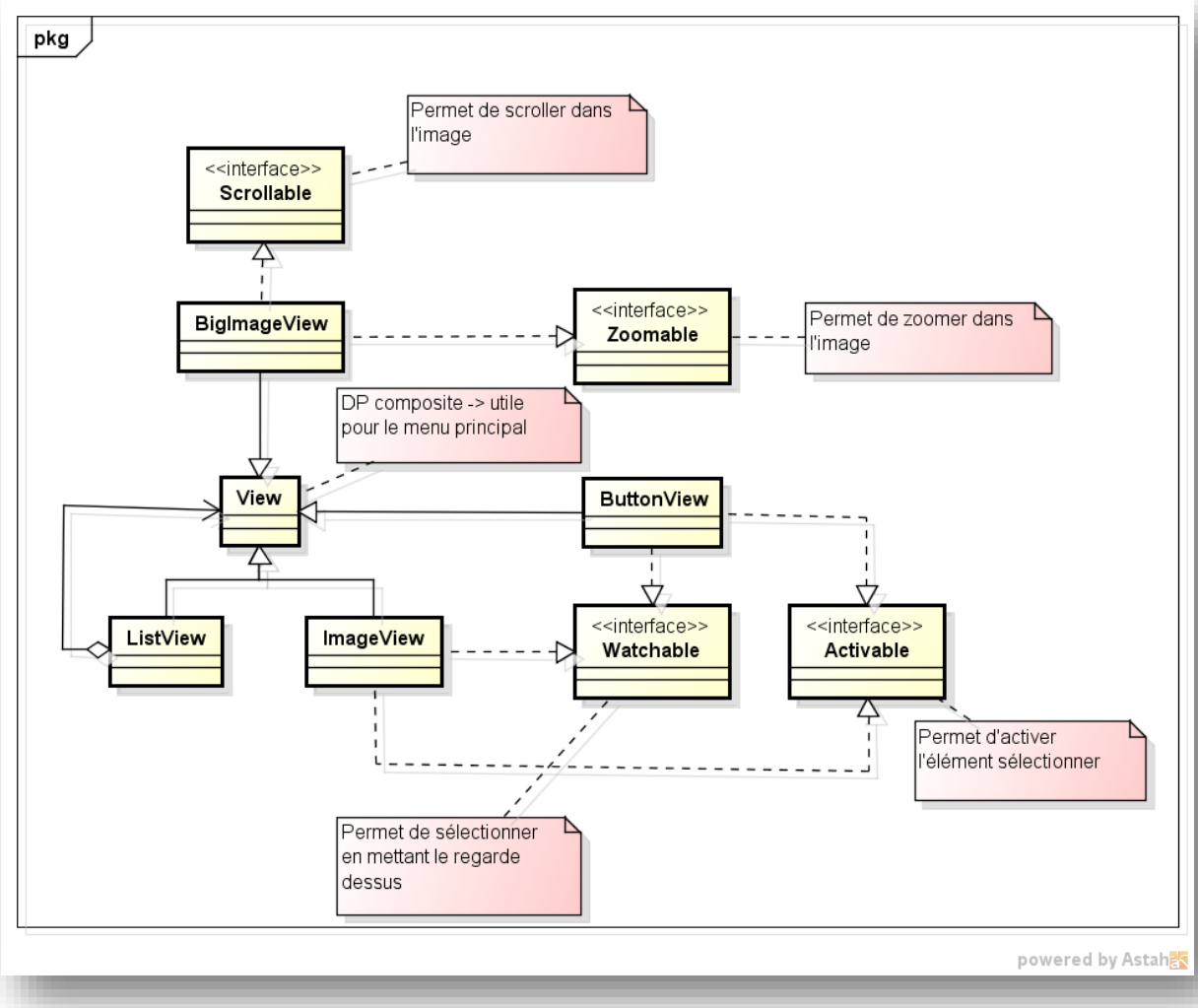


Figure 7 Diagramme de classe partie I

5.1.2 Seconde partie

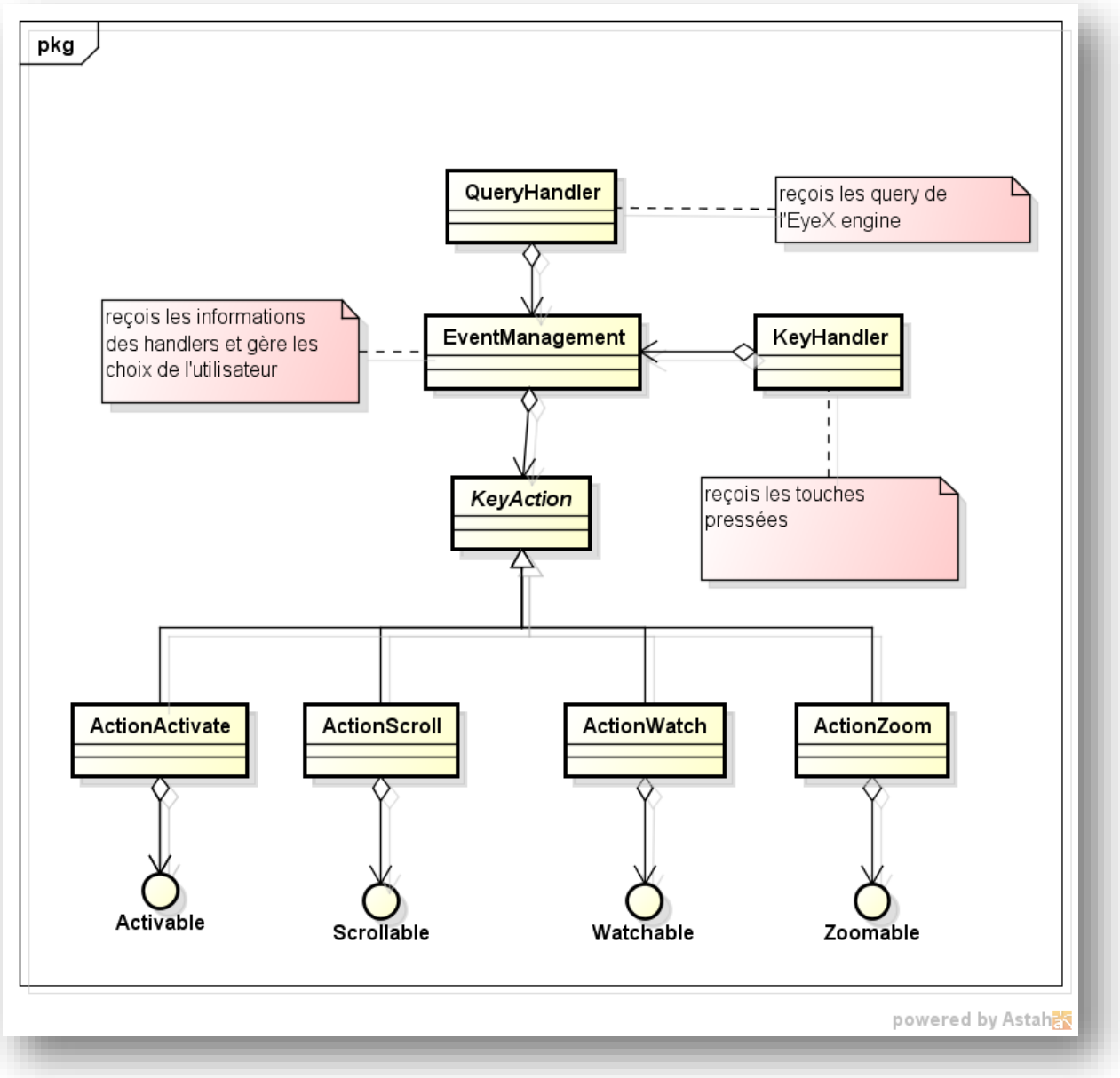


Figure 8 Diagramme de classe partie II

5.2 Interfaces graphiques

Voici les maquettes des interfaces graphiques pour le menu principal ainsi que pour la partie de navigation.

5.2.1 Menu principal

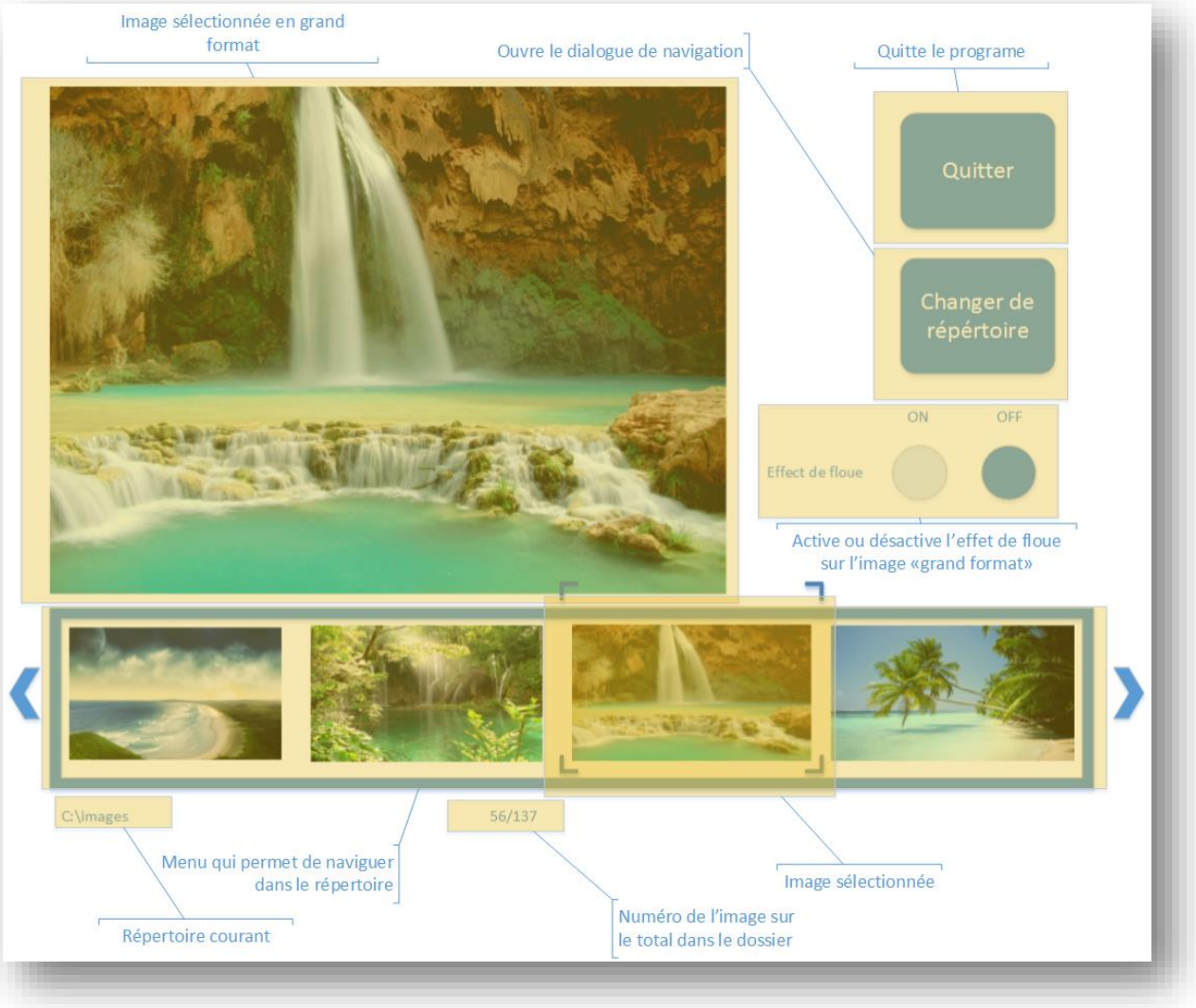


Figure 9 Maquette du menu principal

5.2.2 Menu navigation

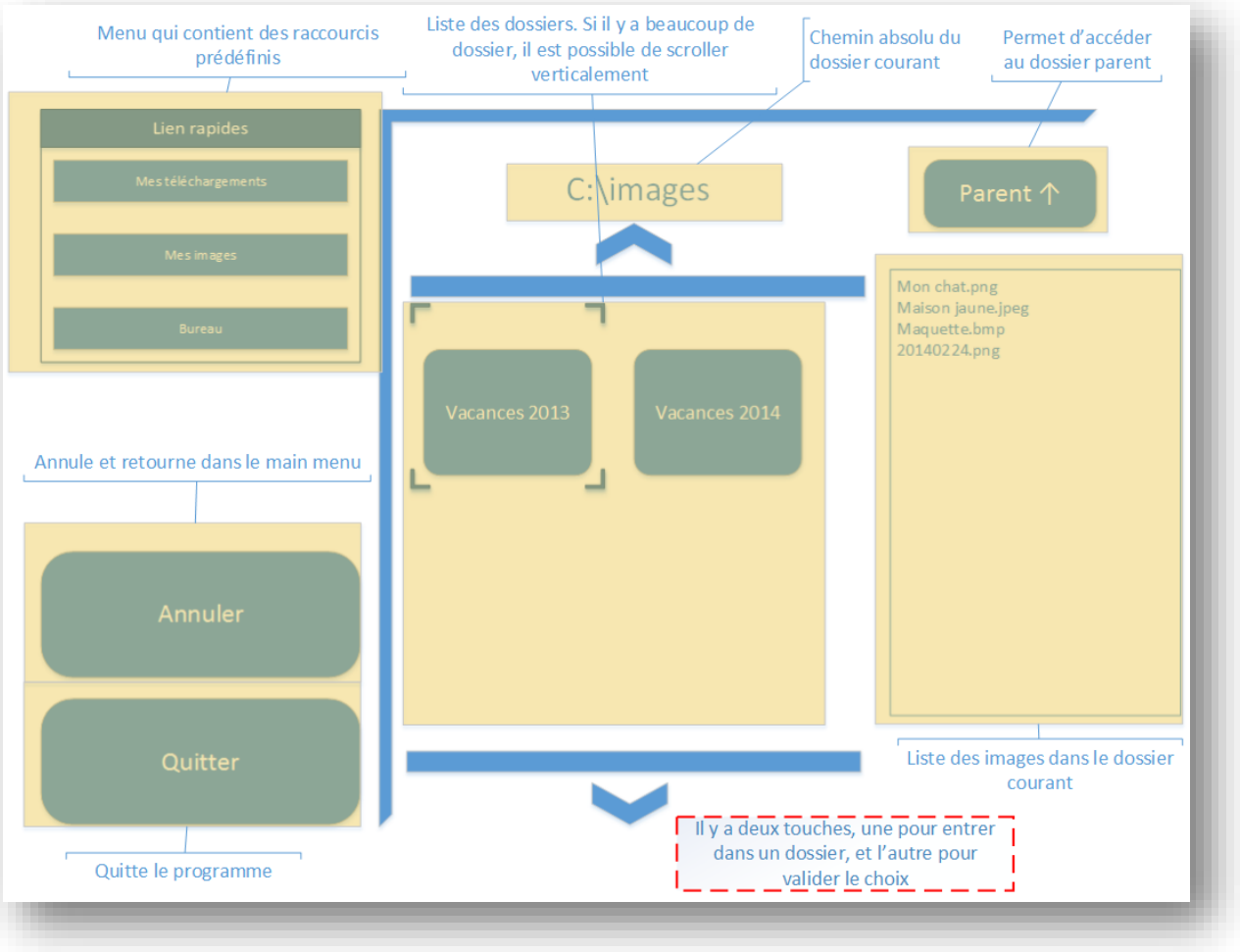


Figure 10 Menu de navigation

5.2.3 Pièges à éviter

Comme on peut s'en douter, utiliser des boutons de petite taille n'est pas une bonne idée. Tobii n'a pas de taille recommandée[19], ils indiquent juste d'éviter d'utiliser des contrôles trop petits car il y a du bruit (dans le sens imprécision).

Là où ça devient moins évident, c'est de penser à minimiser l'angle entre le Tobii Rex et le contrôle. En effet, si nous mettons des éléments en haut de l'écran, il y aura plus d'imprécisions qu'en bas de l'écran.

Il y a une petite astuce qu'on peut utiliser dans certain cas pour augmenter l'efficacité de l'eyes tracking. Celle-ci consiste à utiliser une touche, par exemple 'Q'. Quand celle-ci est appuyée, un nouveau menu apparait. Il est possible d'utiliser plusieurs touches pour plusieurs menus. Cela permet de **minimiser** le nombre de contrôles affichés à l'écran en même temps.

[19] http://developer.tobii.com/community/forums/topic/design-of-gui-buttonobject-sizes/

6 Implémentation

Lors de l'implémentation, certaines parties méritent de figurer dans le rapport. Notamment les parties qui sont difficiles à comprendre d'un simple coup d'œil, ou encore celles qui permettent des features importantes.

6.1 Trouver quel control est « regardé »

Il n'est pas toujours facile de déterminer quel contrôle l'utilisateur regarde. Reprenons pour exemple l'image de Google.

Figure 11 Quel est l'interactor choisit ?

Le regard est posé entre deux interactors (sur la barre de recherche et sur le bouton « Recherche Google »). Mais lequel l'utilisateur veut réellement voir ?

Nous allons parler de deux techniques possibles.

6.1.1 Surface absolue (technique naïve)

Cette première technique est présentée comme naïve pour des raisons qui seront bientôt évidentes.

Pour connaître l'interactor regardé, il suffit de comparer l'intersection et choisir celui dont la surface d'intersection est la plus grande.

Exemple : Z = surface du regard de l'utilisateur, a et b sont deux interactors.

$$A = Z \cap a$$
$$B = Z \cap b$$

Si A > B, alors l'interactor a est choisi, sinon c'est le b.

Cette technique est très simple et permet d'avoir de bons résultats. Néanmoins, il y a un inconvénient majeur. Prenons le cas suivant :

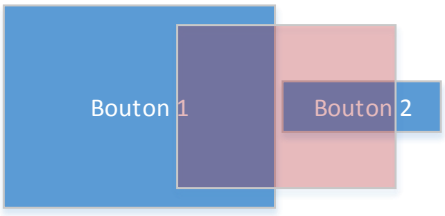


Figure 12 Bouton 1 ou Bouton 2 ?

Avec cette technique, il est très difficile d'atteindre le Bouton 2, car ça surface personnelle est beaucoup plus petite que celle du bouton 1. L'intersection sera plus grande chez Bouton 1 que chez Bouton 2.

Nous allons donc essayer de pallier ce problème avec la solution suivante.

6.1.2 Surface relative

L'astuce dans cette technique est de comparer « à quel pourcentage a ou b est intersecté ».

Sur la figure 12, le bouton 2 est intercepté à ~70%, tandis que le bouton 1 est intercepté à ~20%. Dans cette technique, c'est bouton 2 qui est donc choisi.

C'est donc cette technique qui nous parait la plus juste que nous avons décidé d'implémenter.

Nous avons remarqué que cette technique ressemble beaucoup à l'indice de Jaccard[20], sauf qu'on prend en compte uniquement la surface de l'interactor au dénominateur.

[20] http://fr.wikipedia.org/wiki/Indice_et_distance_de_Jaccard

6.2 Partie visualisation des images

Le dialogue qui permet de visualiser les images d'un dossier est la partie principale du projet. Nous aborderons deux principes clés de notre implémentation : Le zoom et le scroll.

6.2.1 Zoom

Le zoom est l'action qui consiste à se focaliser sur un point précis de l'image et d'en augmenter le niveau de détail en l'agrandissant.

Nous avons essayé plusieurs techniques, certaines ont eu plus de succès que d'autres. Nous allons présenter deux d'entre-elles. Celle qui sera présenté en premier est moins performante que la suivante, néanmoins elle mérite d'être présentée.

Pour expliquer avec le plus de clarté, nous allons présenter un scénario simple.

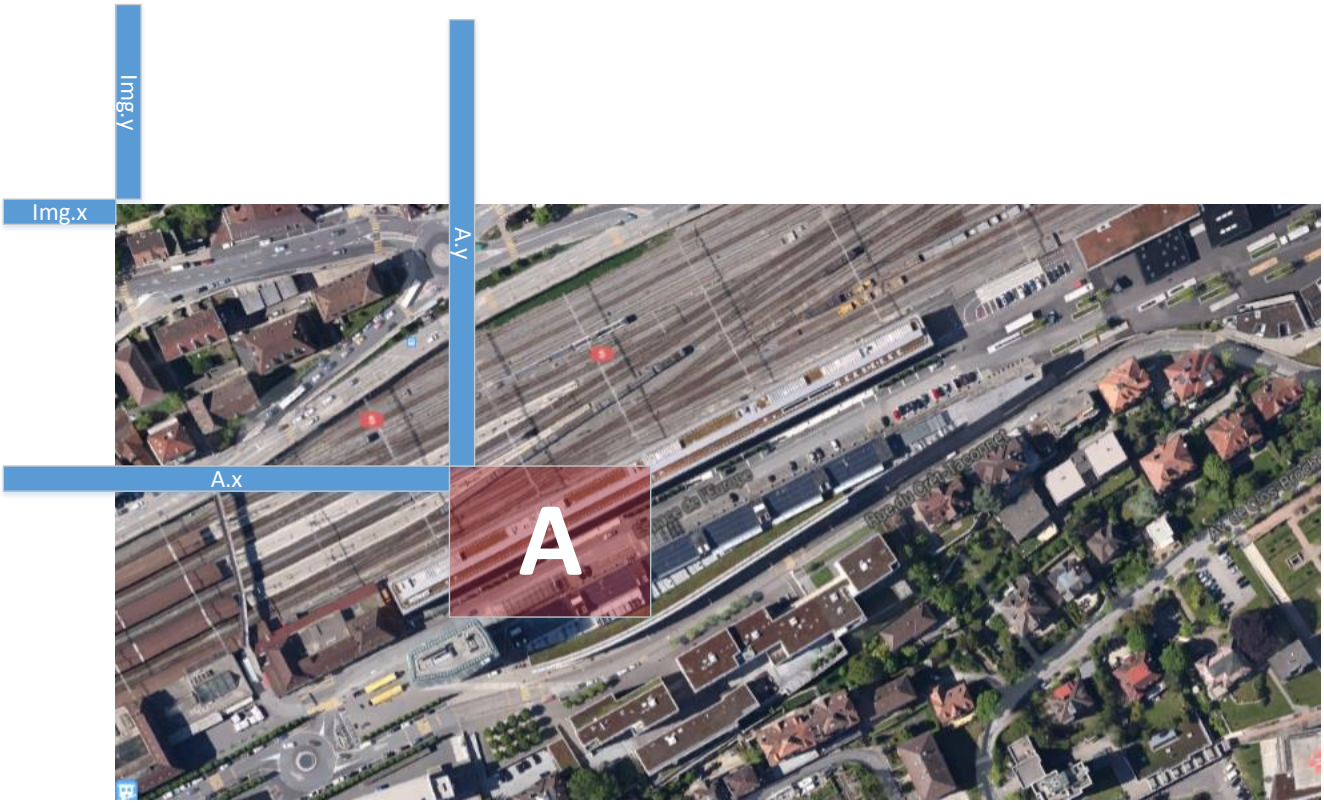


Figure 13 Scénario d'un zoom

La zone 'A' est la surface du regard de l'utilisateur. Celui-ci veut donc zoomer dans l'image, voici comment nous allons tenter d'effectuer cette action.

Nous créons un nouveau point au centre de l'image. Ensuite nous allons déduire le vecteur qui se trouve entre le centre de l'image et le centre de la zone 'A', nommons celui-ci « VZOOM ».

C'est ici que va commencer à diverger les deux techniques. Pour la première, le vecteur « VZOOM » va être normalisé. Ensuite nous allons faire une translation depuis le centre de l'image dans le sens du « VZOOM » normalisé en le multipliant par un scalaire qui permet de choisir la vitesse de déplacement.

Ensuite nous effectuons un changement d'échelle afin d'agrandir l'image.

Cette technique permet d'avoir un déplacement fluide qui est assez agréable. Toutefois celle-ci a un inconvénient majeur : la vitesse de déplacement ne prend pas en compte si l'utilisateur regarde loin ou prêt du centre de l'image. Cela rend difficile d'accéder rapidement aux endroits éloigné de centre de l'image.

Pour pallier cet inconvénient, nous avons mis au point une seconde technique. Dans celle-ci, nous ne normalisons plus le vecteur « VZOOM » et le scalaire a un espace plus petit que '1' afin de limite la vitesse (cela permet d'éviter des changements brusques).

Cette seconde technique permet de zoomer beaucoup plus rapidement que la première et est plus agréable à utiliser.

6.2.2 Scroll

Le scroll est organisé comme le zoom, mais sans faire de changement d'échelle.

6.3 Partie navigation

Un menu de navigation qui permet de changer de répertoire est disponible dans l'application. Il est possible de modifier les dossiers qui seront affichés. Pour faire cela, il faut éditer le fichier « App.config » comme ci-dessous :

```
<add key="listDossiers" value="
     %HOMEDRIVE%\\%HOMEPATH%\\Pictures
     C:\\images3
     C:\\imagesBig
     D:\\image1
     D:\\image2
     D:\\image3
     D:\\image4
     D:\\image5"/>
```

Figure 14 App.config liste des dossiers

Pour avoir le résultat suivant :

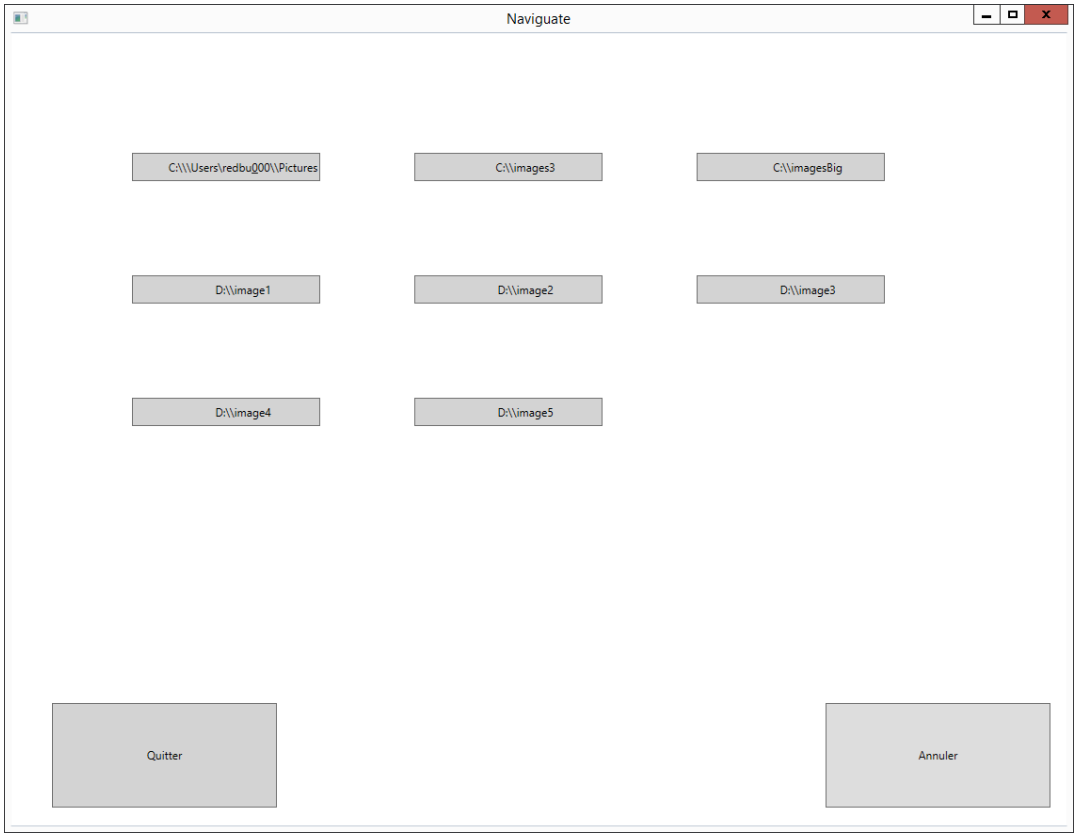


Figure 15 Interface résultante de navigation

7 Evolutions

Lors du développement dans une technologie que l'on ne maitrise pas, il est difficile de faire du code « maintenable » et « évolutif ». Néanmoins, nous avons réussi à créer un code qui permet de facilement faire évoluer l'application de manière significative, sans pour autant devoir tout réécrire.

L'aspect évolutif le plus intéressant, c'est la possibilité d'utiliser une autre technologie que les Tobii Rex afin de capturer les mouvements des yeux.

7.1 Utiliser une autre technologie que Tobii Rex

Nous avons eu la chance d'avoir accès à un autre appareil d'oculométrie : « THEEYETRIBE [21]». Les deux appareils sont assez similaires. Ils ont tous les deux des caractéristiques uniques, mais ils ont un socle commun.

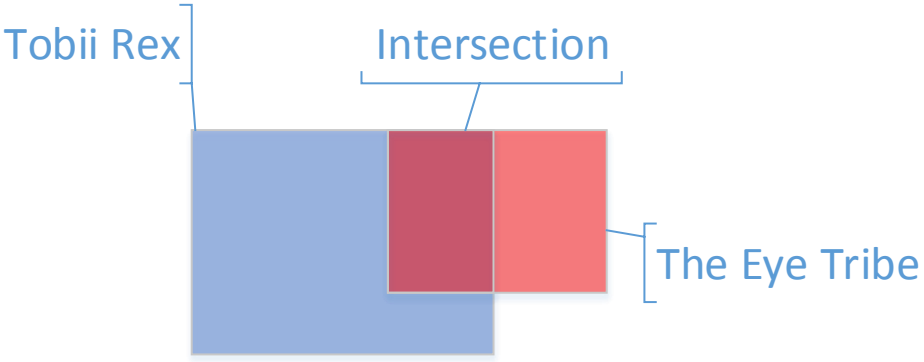


Figure 16 Intersection entre Tobii Rex et The Eye Tribe

C'est justement sur ce socle commun (intersection) que se situent nos besoins pour notre application (nous avons seulement besoin de savoir où se pose le regard de l'utilisateur).

[21] https://theeyetribe.com/

8 Tests et vérifications

Ce chapitre présente les différents tests qui devront être effectué afin d'assurer le fonctionnement de ce projet. Étant donné que l'application à rendre est un « démonstrateur », les tests ne sont pas centraux. Toutefois, nous avons créé un protocole de test afin d'avoir une vision rapide de ce qui fonctionne.

8.1 Tests des vues

Le logiciel contient deux vues, voici les éléments à valider dans chacune d'elles (Toutes les interactions se font avec le Tobii Rex ou avec Tribe).

8.1.1 Vue principale

Elément à tester	Résultat	Itération	Commentaire
Il est possible de quitter le logiciel	Juste	I	
Les images du dossier s'affichent dans la listView	Juste	I	
Le répertoire courant est affiché	Juste	I	
Le nombre d'image et l'indice de l'image actuelle sont affichés	Juste	I	Peut-être plus judicieux d'afficher l'indice de l'image sélectionnée plutôt que regardée.
Un effet visuel permet de distinguer l'élément regardé	Juste	I	Effet orange
Un bouton permet d'aller sur la vue de navigation	Juste	I	
L'image sélectionnée doit s'afficher en grand	Juste	I	
Il est possible de zoomer dans l'image sélectionnée	Juste	III	
Il est possible de scroller dans l'image sélectionnée	Juste	III	
Il est possible d'activer ou désactiver l'effet de flou	Faux	III	Point facultatif non implémenté
Il est possible de scroller dans le choix des images dans la liste	Juste	III	Deux boutons, un de chaque côté

8.1.2 Vue de navigation

Elément à tester	Résultat	Itération	Commentaire
Un menu avec des liens rapide permet d'accéder rapidement à certains dossiers	Juste	II	
Il est possible d'annuler cette vue et retourner à la vue principale	Juste	II	
Il est possible de quitter le logiciel	Juste	II	

Le chemin du dossier courant doit s'afficher	Changé	II	Ne s'applique plus avec le changement du cahier des charges
Un bouton permet d'accéder au dossier parent	Changé	II	Ne s'applique plus avec le changement du cahier des charges
Une liste permet d'afficher les dossiers du répertoire courant	Changé	II	Ne s'applique plus avec le changement du cahier des charges
Une liste permet d'afficher le nom des images du répertoire courant	Changé	II	Ne s'applique plus avec le changement du cahier des charges
Un effet visuel permet de distinguer le dossier regardé	Juste	II	Effet orange
Il est possible d'entrer dans un sous-dossier	Changé	II	Ne s'applique plus avec le changement du cahier des charges
Il est possible de choisir un sous-dossier et de l'ouvrir pour la vue principale	Changé	II	Ne s'applique plus avec le changement du cahier des charges

8.2 Technologies utilisées pour les tests

Les technologies peuvent apporter une valeur ajoutée important quand celles-ci sont disponibles.

8.2.1 Les tests unitaires

Le Framework .NET permet de faire ces tests nativement. Toutefois nous n'en utilisons pas.

8.2.2 Les tests de scénarios

Ils devront se faire manuellement, nous n'avons pas trouvé d'équivalent à « Selenium [22]» (pour les interfaces web) pour les interfaces normales.

[22] http://docs.seleniumhq.org/

9 Problèmes rencontrés

Ce chapitre énumère les difficultés majeures rencontrées durant le projet

9.1 Responsabilité des objets

Lors de la conception, nous voulions que chaque objet qui s'affiche à l'écran ait la responsabilité de s'afficher.

Le problème, c'est qu'en WPF c'est aux objets parents (dans le sens de la composition) qui incombent cette responsabilité.

Nous avons pu pallier ce problème en modifiant la conception de sorte que les objets retournent un « FrameworkElement » et ainsi les parents peuvent les afficher facilement.

9.2 Lacunes C# et WPF

N'étant pas très familier avec la technologie C#/WPF, à plusieurs reprises nous nous sommes dits : « j'aurais dû faire comme ceci ou comme cela » après l'implémentation effective.

De toute évidence, nous aurions une implémentation sensiblement différente si nous devions recommencer.

9.3 Bugs connus

Voici la liste des bugs recensés

- Plante si on appuie une touche dès l'ouverture du programme
- On ne peut pas zoomer / scroller dès l'ouverture du programme (il faut choisir une image, même si la première image est affichée en grand format
- Les noms des images ne doivent pas contenir des espaces.

9.4 Bac-à-sable

Il aurait été souhaitable de mettre en place un confinement de l'image afin que l'utilisateur ne puisse pas faire disparaître l'image de l'écran.

Le problème est que cela n'est pas évident de le faire dans l'implémentation actuelle. Voici l'illustration du problème :

Figure 17 Dépassement du canevas

On peut connaître la position absolue du canevas, mais pas celle du Background contenu de celui-ci (c'est là-dessus qu'est appliqué les transformations pour le zoom et le scroll).

Il est peut-être possible de s'en sortir mathématiquement avec la matrice de transformation, mais nous allons laisser ça à notre éventuel successeur.

10 Conclusion

Développer une application s'utilisant avec des appareils d'oculométries a été une expérience enrichissante. En effet, ça demande de la réflexion sur la conception des interfaces graphiques. Il n'est pas facile de placer les éléments de manière que l'utilisateur les sélectionne le plus facilement possible. L'exemple d'une liste d'éléments à afficher entre exactement dans cette problématique.

Peut-être devrions-nous concevoir des interfaces complétement différentes. Par exemple, avec une vue d'ensemble puis un zoom automatique sur la surface regardée par l'utilisateur. Il reste encore beaucoup à faire pour les interfaces graphiques utilisant ces appareils.

Pour une personne handicapée, l'oculométrie est très utiles car cela peut être le seul moyen d'interaction avec un ordinateur. Par contre, pour une personne valide, cela n'apporte pas une vraie valeur ajoutée[23]. Le clavier et la souris offrent une approche très rapide et intuitive.

Eventuellement, si dans le futur il est possible de concevoir des appareils d'une précision redoutable, cela pourrait changer la donne.

La commande via la voix pourrait offrir une alternative intéressante, par exemple nous regardons une image et disons « zoom » pour zoomer. Cela permettrait d'éviter de devoir cliquer sur des touches différentes pour les actions.

Concernant le projet, comme nous pouvons le voir, ça fonctionne plutôt bien, malgré l'imprécision des capteurs. Dommage cependant que le périphérique de « Tribe » ait une connectique défectueuse, nous n'avons pu faire que peu d'essais. Pour les comparer efficacement, il aurait fallu plus de tests.

Le côté le plus intéressant de l'application est le zoom et le scroll dans la grande image. L'utilisation est fluide et agréable.

Neuchâtel, le mardi 3 juin 2014

William Droz

[23] Pour les cas devant les ordinateurs

11 Références

http://acces.ens-lyon.fr/acces/ressources/neurosciences/vision/comprendre/VisionMarseille/pages_techniques

- Permet d'avoir des bonnes explications en Français sur l'enregistrement des yeux

http://www.tobii.com/fr/eye-tracking-research/global/research/usability/

- Montre le potentiel des eyes tracker pour les interfaces graphiques

12 Annexes

12.1 Code C++ pour Tobii Rex

```
EyeXHost::EyeXHost()
	: _state(Initializing),
	_hWnd(nullptr),
	_statusChangedMessage(0), _focusedRegionChangedMessage(0),
_regionActivatedMessage(0),
	_focusedRegionId(-1),
	_context(TX_EMPTY_HANDLE),
	_connectionStateChangedTicket(0), _queryHandlerTicket(0), _eventHandlerTicket(0)
{
	// initialize the EyeX Engine client library.
	txInitializeSystem(TX_SYSTEMCOMPONENTOVERRIDEFLAG_NONE, nullptr, nullptr);

	// create a context and register event handlers, but don't enable the connection to
the engine just yet.
	// we'll enable the connection in the Init method, when we're ready to handle the
	// connection-status-changed notifications.
	bool success = txCreateContext(&_context, TX_FALSE) == TX_RESULT_OK;
	success &= RegisterConnectionStateChangedHandler();
	success &= RegisterQueryHandler();
	success &= RegisterEventHandler();

	if (!success)
	{
		SetState(Failed);
	}
}
```