UNIVERSITÉ ANTONINE
Faculté d'ingénieurs en Informatique, Multimédia, Réseaux et Télécommunications

Web Cam Modes

Matière: Atelier de Génie Logiciel

Présenté par: Elie Matta et al.

Table de matières

Phase 1 :
Spécifications des exigences fonctionnelles

Chapitre 1 :
Description du projet

1) Introduction

Ce projet est « Webcam Modes » permettant à un utilisateur de choisir un mode désiré et de prendre des photos selon le mode choisi.
L'utilisateur doit tout d'abord démarrer le programme et de choisir un des sept modes présents.
Ce « Webcam Modes » offre aussi la possibilité de charger la photo prise sur un serveur ou le site construit et spécialement dédié au chargement des photos, en leur offrant une interface facile.

Ce service de webcam offre la possibilité aux utilisateurs de consulter la liste des photos dans une librairie. En fait le regroupement des photos sera élaboré selon la date de prise de la photo.

Il faut signaler que ce service est une « Windows application » qui doit être installée sur la machine client.

2) Les acteurs du système :

Les utilisateurs du « Webcam Modes» ainsi que le système lui-même sont les acteurs. En effet les utilisateurs peuvent effectuer plusieurs fonctions afin d'interagir avec ce système comme par exemple prendre une photo, choisir un mode, faire un croquis… et en contre partie le système répond aux actions des utilisateurs. Quand on parle du système on désigne l'application ainsi que le serveur de bases de données.

3) Les services attendus :

Ce service de photo favorise la prise d'une photo selon plusieurs modes ou l'élaboration d'un croquis quelconque. Les fonctions principales des différents modes sont les suivantes :
-Mode Slow Motion : ce mode permet de prendre des photos avec le temps du slow motion qu'on décidera nous même.
-Mode Paint : ce mode sert à dessiner une photo donnée en introduisant un objet distinguée
-Mode Color : ce mode sert à choisir la couleur du fond de l'image qu'on veut prendre.
-Mode Spy : ce mode sert à détecter la personne qui passe devant la camera.
-Mode Face Detection : ce mode sert à voire le degré de ressemblance entre deux images.
-Environment Mode : ce mode sert à choisir un cadre pour l'image qu'on veut prendre.
-Personnalization Mode : ce mode sert à choisir des accessoires pour une image

En d'autres termes le but de ce service de photo est d'assurer à l'utilisateur différents modes, couleurs, accessoires et facettes pour la prise d'une photo. En plus d'une librairie pour le stockage des photos.

4) Les fonctionnalités

Ce «Webcam Mode » a un objectif de c'est pourquoi il présente plusieurs fonctions qui sont les suivantes :

- **Slow Motion :** l'utilisateur est capable de prendre des photos successives séparées par un délai de temps précisé à l' avance.

- **Take Picture :** Cette fonction est présente dans les différents modes du programme. Elle permet a l'utilisateur de prendre des photos successives, à l'instant désiré, et dans le mode désiré. Les photos peuvent être prises même s'il est dans le slow motion mode.

- **Paint:** Le programme demande lors de l'exécution de cette fonction de choisir une certaine couleur d'objet et de mettre en relief l'objet désiré pour élaborer le croquis. Une fois l'objet détecté, l'utilisateur n'a qu'à déplacer l'objet dans une certaine direction pour dessiner le croquis.

- **Color :** Cette fonction permet à l'utilisateur de choisir la couleur désirée pour le fond de l'image. L'utilisateur dispose d'une palette pour choisir la couleur préférée.
  L'utilisateur peut donc personnaliser la fenêtre en choisissant sa couleur «Background », en précisant les caractéristiques du font ainsi que sa couleur.

- **Spy**: Même si l'utilisateur n'est pas présent devant son ordinateur, cette fonction démarre la webcam, garde cette dernière en mode démarré et prend des photos à chaque fois que quelqu'un passe devant la camera ou qu'elle détecte un changement dans la photo.

- **Face Detection :** Cette fonction permet la comparaison de deux photos et affiche le pourcentage de ressemblance entre ces deux dernières. La comparaison consiste à comparer les pixels des deux photos.

- **Environment:** Cette fonction permet a l'utilisateur de choisir le « Backgroud » désiré (une photo par exemple) et prendre la photo avec le « Backgroud » choisi.

- **Personalization :** l'utilisateur peut choisir des accessoires (chapeau, moustache, cheuveux...) et prendre la photo en portant ces accessoires. Il peut de même choisir un déguisement prédéfinit dans la librairie de la fonction (visage de loup par exemple).

5) Description technique

La technologie, l'outil, le système d'exploitation ainsi que l'architecture technique utilisée pour réaliser ce projet sont :
Technologie : La technologie utilisée est le langage c#; en effet on est familier avec ce langage, c# fournit une interface graphique facile à utiliser. Le langage UML (**Unified Modeling Language**) sera utilisé pour la conception de ce projet.
Outil : L'outil est le Visual studio 2005 qui nous fournit une interface capable de faciliter nos taches de programmation. Le rational rose sera utilisé en tant que outil de conception.
Système d'exploitation : Ce projet sera réalisé sur Windows vista.
Processus technique : RUP « Rational Unified Process ».

6) Cahier des charges

Le cahier des charges est un document recensant les spécifications/exigences. Il résulte de l'analyse et est contractuel entre le client et l'entreprise informatique qui va réaliser le logiciel. Il doit donc être validé par les deux.
Qualités attendues : non ambigu, complet, vérifiable, cohérent, modifiable, traçable, utilisable durant la maintenance, indépendant des solutions.
Plan type :
1. introduction : présentation générale, motivations, définitions des termes ;
2. contexte : environnement matériel et humain, acteurs et utilisateurs, interaction avec d'autres systèmes et logiciels, existant ;
3. spécifications fonctionnelles : grandes fonctionnalités du système, acteurs et autres systèmes qu'elles impliquent ;
4. spécifications non fonctionnelles, contraintes :
– carte graphique,
– matériel : marques, RAM, débit de connexion Internet. . .
– interfaçage : protocoles de communication, formats de fichiers, etc., pour l'interaction avec des matériels, logiciels, systèmes d'exploitation. . .
– performances : temps réel. . .
– sécurité : sauvegardes, confidentialité. . .
– charge à supporter : volume des données, nombre d'utilisateurs simultanés. . .
– comportement en cas de panne. . .
5. priorités relatives des spécifications, versions à prévoir, délais ;
6. évolutions à prévoir ;
7. annexes.

7)

Dans le contexte des projets de grande envergure ont émergé des rôles pour partager et désigner les responsabilités :

Tâche	Niveau	Ressources	Etat	Temps de construction	Commentaires (techniques et fonctionnels)
Analyse des besoins	s	privacy applied	a	3	On a réalisé que les fonctions sont présentes sur internet mais le projet comme une seule entité n'existe pas
Analyse des besoins	s	privacy applied	a	3	On a réalisé que les fonctions sont présentes sur internet mais le projet comme une seule entité n'existe pas
Analyse des besoins	s	privacy applied	a	3	On a réalisé que les fonctions sont présentes sur internet mais le projet comme une seule entité n'existe pas
Analyse des besoins	s	privacy applied	a	3	On a réalisé que les fonctions sont présentes sur internet mais le projet comme une seule entité n'existe pas
Identification des fonctions	m	privacy applied	a	2	
Identification des fonctions	m	privacy applied	a	2	
Identification des fonctions	m	privacy applied	a	2	
Identification des fonctions	m	privacy applied	a	2	
Elaboration du code de la fonction: Slow_motion()	c	privacy applied	c		Cette fonction sert a prendre des photos avec slow motion qu'on décidera nous même
Elaboration du code de la fonction: Take_picture()	c	privacy applied	c		Cette fonction sert a prendre des photos en cliquant sur le bouton "Take Picture"
Elaboration du code de la fonction: Paint()	c	privacy applied	c		Cette fonction sert à dessiner une photo donnée en introduisant un objet distinguée
Elaboration du code de la fonction: Color()	c	privacy applied	c		cette fonction sert à choisir la couleur du fond de l'image qu'on veut prendre
Elaboration du code de la fonction: Spy()	c	privacy applied	c		cette fonction sert à détecter la personne qui passe devant la camera

Elaboration du code de la fonction: Face_detection()	c	privacy applied	c		cette fonction sert à détecter une image d'une série qui se trouve dans notre database
Elaboration du code de la fonction: Environment_mode()	c	privacy applied	c		cette fonction sert à choisir un cadre pour l'image qu'on veut prendre
Elaboration du code de la fonction: Personalization_mode( )	c	privacy applied	c		cette fonction sert à choisir des accessoires pour une image
élaboration du DCU	m	privacy applied	a	3	Diagramme de Use Case
élaboration du DCU	m	privacy applied	a	3	Diagramme de Use Case
élaboration des SN/SA/SN	m	privacy applied	c		Scenario Nominal et Scenario Alternatif et Scenario d'Erreurs
élaboration des SN/SA/SN	m	privacy applied	c		Scenario Nominal et Scenario Alternatif et Scenario d'Erreurs

Dans notre projet nous avons partagé les rôles aux différents concepteurs du projet. A chaque tache sont assignés :

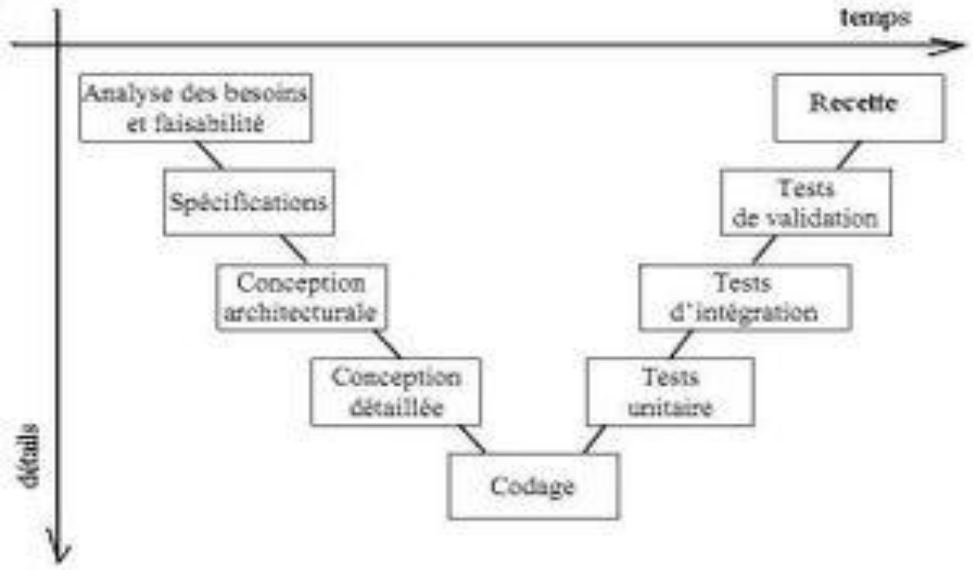
- Un niveau : simple, moyen ou complexe
- Les ressources : privacy applied
  privacy applied
  privacy applied
  privacy applied
- Un état : Achevé ou en cours
- Le temps de construction eh heures
- Les commentaires ou explication de chaque tache.

8) Cycle en V :

Le **modèle du cycle en V** est un modèle conceptuel de gestion de projet imaginé suite au problème de réactivité du modèle en cascade. Il permet, en cas d'anomalie, de limiter un retour aux étapes précédentes. Les phases de la partie montante doivent renvoyer de l'information sur les phases en vis-à-vis lorsque des défauts sont détectés, afin d'améliorer le logiciel.

Le cycle en V est devenu un standard de l'Industrie logicielle depuis les années 1980 et depuis l'apparition de l'Ingénierie des Systèmes est devenu un standard conceptuel dans tous les domaines de l'Industrie.

La figure suivante décrit les phases du cycle en V a travers le temps et le niveau de détails.



Les différentes étapes du cycle en V sont les suivantes :

1) Analyse des besoins et faisabilité

C'est-à-dire l'expression, le recueil et la formalisation des besoins du demandeur (le client) et de l'ensemble des contraintes.

Cette première étape, a pour but de dégager du cahier des charges, toutes les contraintes nécessaires à l'élaboration du logiciel. Trois sortes de contraintes logicielles sont à prendre en considération :

- Les contraintes externes définissent les caractéristiques d'entrée/sortie du logiciel attendues par le client (données à utiliser et à afficher, les exigences ergonomiques, le parc informatique, etc.).
- Les contraintes fonctionnelles caractérisent le fonctionnement interne du logiciel, c'est-à-dire, quels moyens seront mis en œuvre pour traiter les informations d'entrée/sortie du logiciel (méthode de calcul, intervalle des données, etc.).
- Les contraintes de performances indiquent la vitesse d'exécution du logiciel ou de ses modules, la résolution d'affichage, la précision des données, etc..

2) Spécification logicielle
3) Conception architecturale

Il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel.
La conception générale (ou analyse organique générale) a pour objectif de déduire de la spécification, l'architecture du logiciel. Lors de cette phase, plusieurs solutions peuvent être envisagées afin d'en étudier leur faisabilité. A l'issue, un document de conception général du logiciel est réalisé afin de décrire la structure générale de l'alternative approuvée.
Lors de cette phase, il peut être décidé de découper le logiciel en plusieurs modules distincts afin de les sous-traiter par plusieurs équipes de développement. Un module possède une interface permettant son intégration au logiciel global ou à d'autres modules, et un corps pour son fonctionnement interne. Ils sont hiérarchisés de telle façon que des modules de bas niveau s'emboîtent dans des modules intermédiaires, lesquels s'intègrent à un module de haut niveau (noyau logiciel).
Un autre découpage peut être aussi utilisé pour scinder le logiciel en tâches distinctes. L'application contient alors plusieurs sous-ensembles ayant en charge des traitements spécifiques (tâches externes et tâches internes au logiciel), lesquels s'interconnectent entre eux

4) Conception détaillée

Consistant à définir précisément chaque sous-ensemble du logiciel. La phase de conception détaillée (ou analyse organique détaillée) en s'appuyant sur le document de conception générale, énumère l'architecture approfondie du logiciel jusqu'à parvenir à une description externe de chaque sous-ensemble et information utilisable dans le futur logiciel. A partir de cette étape, seront connus toutes les données (variables, constantes, attributs, champs, etc.) et fonctions (procédures, méthodes, etc.) de l'application vue de l'extérieur. Le logiciel peut être entièrement écrit en algorithme. Un langage de programmation est en général validé lors de cette phase. Un document de conception détaillée, ainsi qu'un manuel d'utilisation sont édités afin de respectivement de décrire l'architecture détaillée et la mise en oeuvre du logiciel. Les phases de conception doivent prendre environ 25 pourcents du temps total du cycle de développement.

5) Codage

(Implémentation ou programmation), soit la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception.
Le codage consiste à écrire avec un langage de programmation chacune des sous-programmes du logiciel. Le développement peut être confié à une seule personne dans le cas d'une application simple ou divisé entre plusieurs équipes de développeurs dans le cas de projets importants. Cette phase durant environ 15 pourcents du temps total du cycle de vie se termine par la production d'un code source.

6) Test unitaire

Les tests unitaires ont pour objectif de vérifier individuellement la conformité de chaque élément du logiciel (fonctions et variables) par rapport aux documents de conception détaillée. Toutes les fonctionnalités internes et externes de chaque sous-programme sont contrôlées méthodiquement. En outre, un contrôle des performances globales et locales est également entrepris. Cette phase consomme aux alentours de 5 pourcents du temps total du cycle de vie et se finalise par la rédaction des résultats des tests.

7) Test d'intégration

Son objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel. Elle fait l'objet de *tests d'intégration* consignés dans un document.

La phase d'intégration permet de vérifier l'assemblage des différentes parties du logiciel. Les différents modules du logiciel sont successivement intégrés jusqu'à aboutir à la construction complète, en respectant rigoureusement les spécifications des tests d'intégration. Chaque module doit parfaitement être assimilé sans que le fonctionnement des modules précédemment intégrés n'en soit aucunement affecté. Les résultats de cette phase sont consignés dans un document des tests d'intégration. En général, une présentation du logiciel est également réalisée. Les tests d'intégration représentent en moyenne 20 pourcents du temps total du cycle de développement. **Des spécifications de tests d'intégration et unitaire sont également produites, à l'issue des deux phases de conception.** Elles permettront de confronter le fonctionnement de l'application à son architecture générale et détaillée

8) Test de validation (Recette Usine, Validation Usine - VAU)
9) Recette (Vérification d'Aptitude au Bon Fonctionnement - VABF)

La dernière phase a pour vocation de valider le logiciel dans son environnement extérieur. Le produit applicatif est mis en situation d'utilisation finale afin de vérifier s'il répond parfaitement aux besoins énoncés dans les spécifications textuelles (première phase). Un document appelé résultat de la recette est produit au terme de la phase de validation qui dure 10 pourcents du temps total du cycle de vie du développement du logiciel.

La finalité du cycle de vie en V consiste à parvenir sans incident à livrer un logiciel totalement conforme au cahier des charges. Lors de la phase de spécification textuelle, une négociation avec le client permet d'affiner ou d'enrichir les besoins à propos de certains points techniques omis ou obscurs dans le cahier des charges. **Lorsque la spécification est validée, la suite du processus de développement doit être parfaitement encadré, contrôlé et approuvé de telle sorte qu'à aucun moment, il ne soit possible de diverger des règles énoncées lors de la première phase.**

9) <u>Répartition des différents documents selon les étapes :</u>

Documents en fonction des étapes								
Besoins et Faisabilité	**Spécification**	**Conception Architecturale**	**Conception Détaillée**	**Codage**	**Test unitaire**	**Test d'intégration**	**Test de Validation**	**Recette**

Spécification des Besoins Utilisateur Cahier des charges								Rapport de Recette
	Spécifications Générales Spécification Technique des Besoins						Procès Verbal de Validation	
		Dossier de Définition du Logiciel Dossier d'Architecture Technique Plan de Test				Rapport de Tests d'Intégration		
			Rapport de Conception Détaillée		Rapport de Tests Unitaires			
				Code source				

Phase 2 :
Analyse des besoins et fiabilité

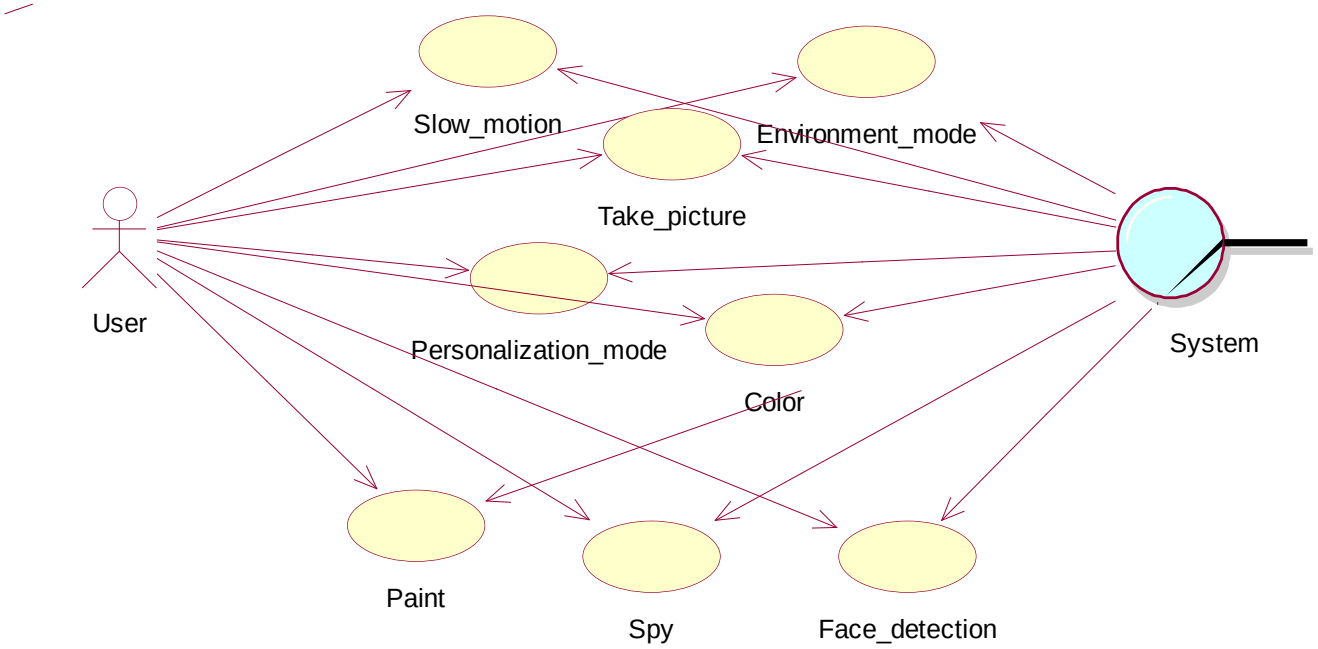
Chapitre 1 :
Description des fonctions

I) Introduction :

Dans cette partie, nous allons décrire en détail toutes les fonctions dont notre application doit accomplir. Ainsi, nous allons commencer par présenter le diagramme des cas d'utilisation pour mieux voir les fonctions que l'utilisateur pourra exécuter à travers cette application.
Ensuite, pour chaque cas d'utilisation, nous allons présenter son scenario nominal, diagramme d'activités et de séquences.
8 fonctions forment les taches principales à accomplir par notre projet.

II) Description des fonctions :

Diagramme des cas d'utilisations :



Ce DCU (Diagramme des cas d'utilisations) met en relief l'interaction entre l'utilisateur et le système : Webcam.
En effet, l'utilisateur peut :

- -Prendre une photo.
- -Effectuer une opération de spy.
- -Changer la couleur d'une photo.
- -Détecter un visage.
- -Modifier une image.
- -Effectuer un effet de slow motion.
- -Mettre un background pour une image.
- -Dessiner une photo.

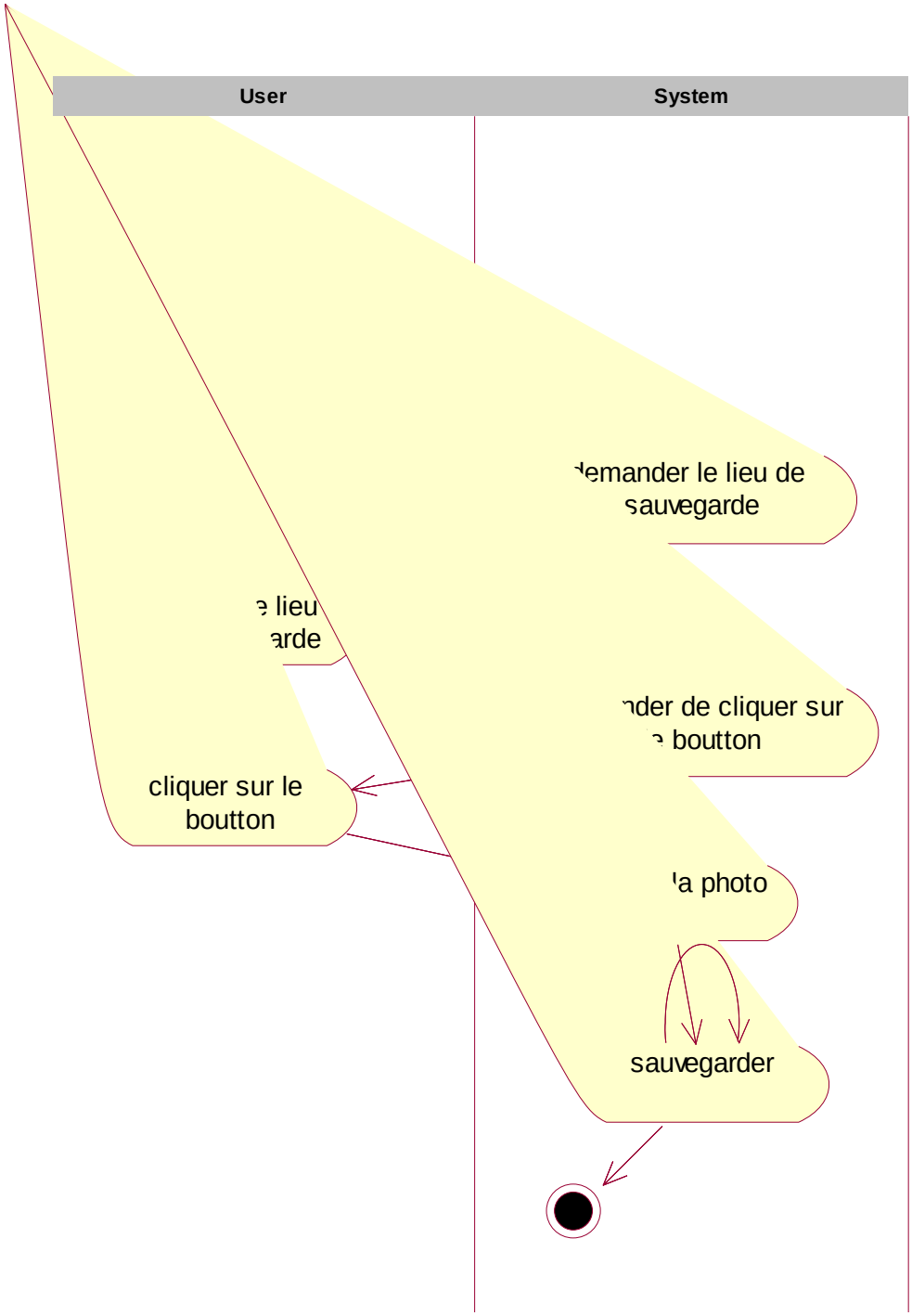
1. **<u>Prendre une photo</u> :**
   Cette fonction permet à chaque utilisateur de prendre une photo en cliquant sur un boutton dans le site web.

Titre de la tache: Prendre une photo.
Responsables de la tache : L'utilisateur et le système.

Acteur (Utilisateur)	**Système (Webcam)**
1) L'utilisateur ouvre l'application.	
2) Il choisit l'option "take picture".	
	3) Le système demande à l'utilisateur où il veut sauvegarder les images.
4) L'utilisateur précise l'endroit du sauvegarde .	
	5) Le système demande de l'utilisateur de cliquer sur le boutton quand il est prêt a prendre la photo.
6) L'utilisateur clique sur le boutton quand il est prêt.	
	7) Le système effectue alors l'opération de prendre la photo.
	8) Le système sauvegarde la photo dans l'endroit demandé.

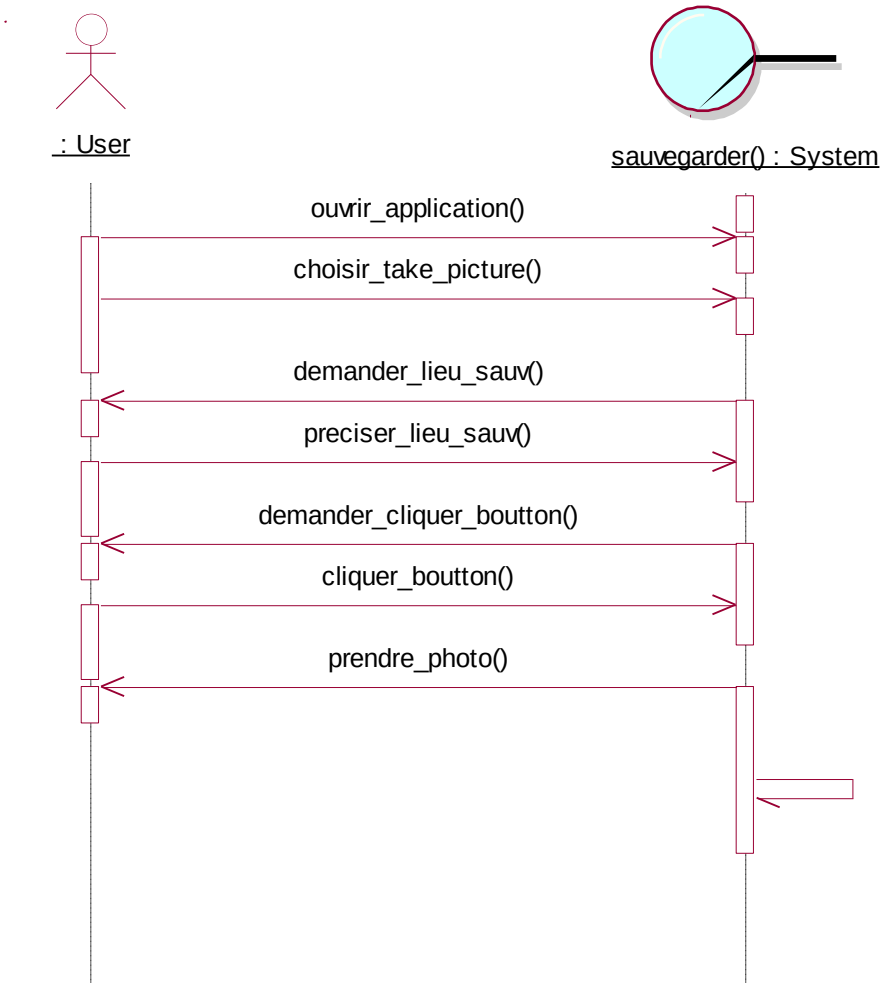
DAC :

Le diagramme d'activité correspondant a cette fonction est le suivant :



DES :

Le diagramme de séquences correspondant a la fonction « take picture » est le suivant :



2. Effectuer une opération de spy:

Cette fonction permet de faire une opération de spy en utilisant l'un des deux modes « invisible » ou « normal ».

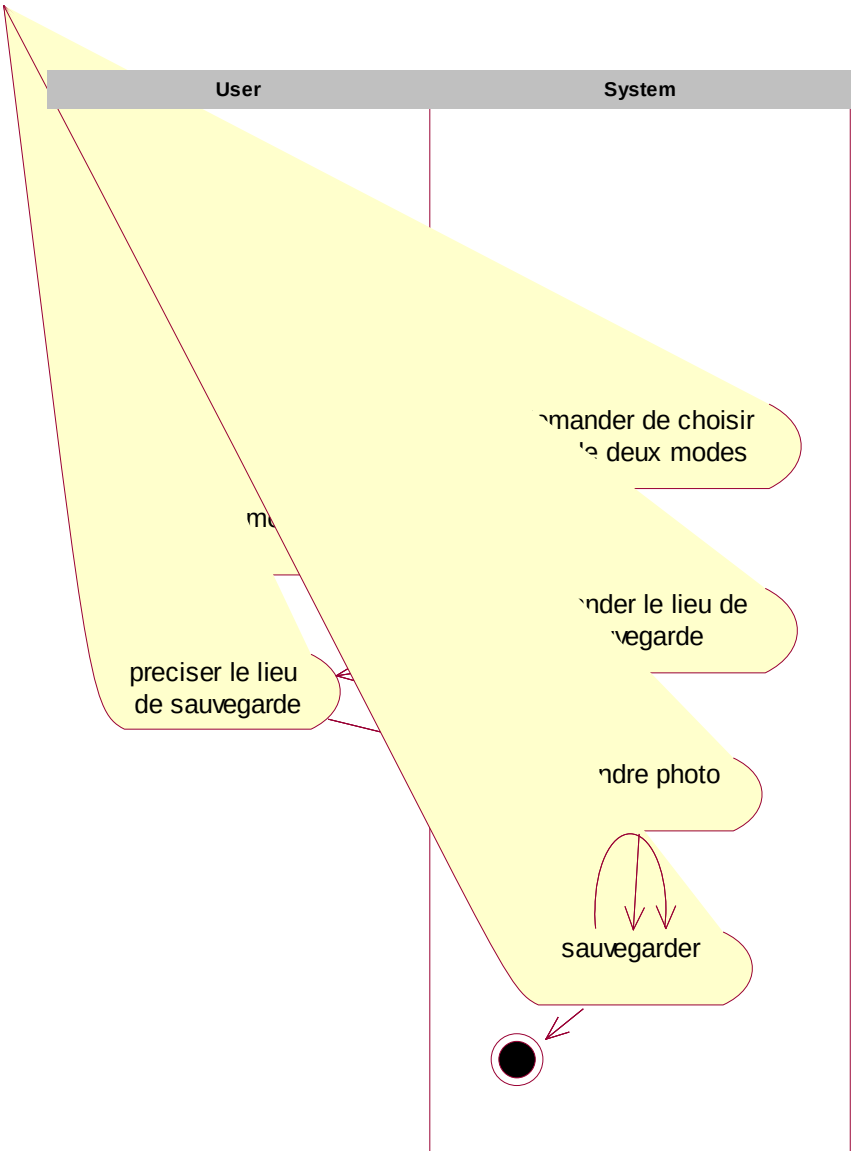
Le scénario nominal pour cette fonction est comme suit :

Titre de la tache: Effectuer une opération de spy.
Responsables de la tache : L'utilisateur et le système.

Acteur (Utilisateur)	**Système (Webcam)**
1) L'utilisateur ouvre l'application.	
2) Il choisit l'option "spy ".	
	3) Le système demande à l'utilisateur de choisir l'un des deux modes « invisible » ou « normal ».
4) L'utilisateur précise le mode.	
	5) Le système demande à l'utilisateur où il veut sauvegarder les images.
6) L'utilisateur précise l'endroit du sauvegarde .	
	7) Le système prend les photos selon le mode demandé.
	8) Le système sauvegarde les images dans l'endroit demandé.

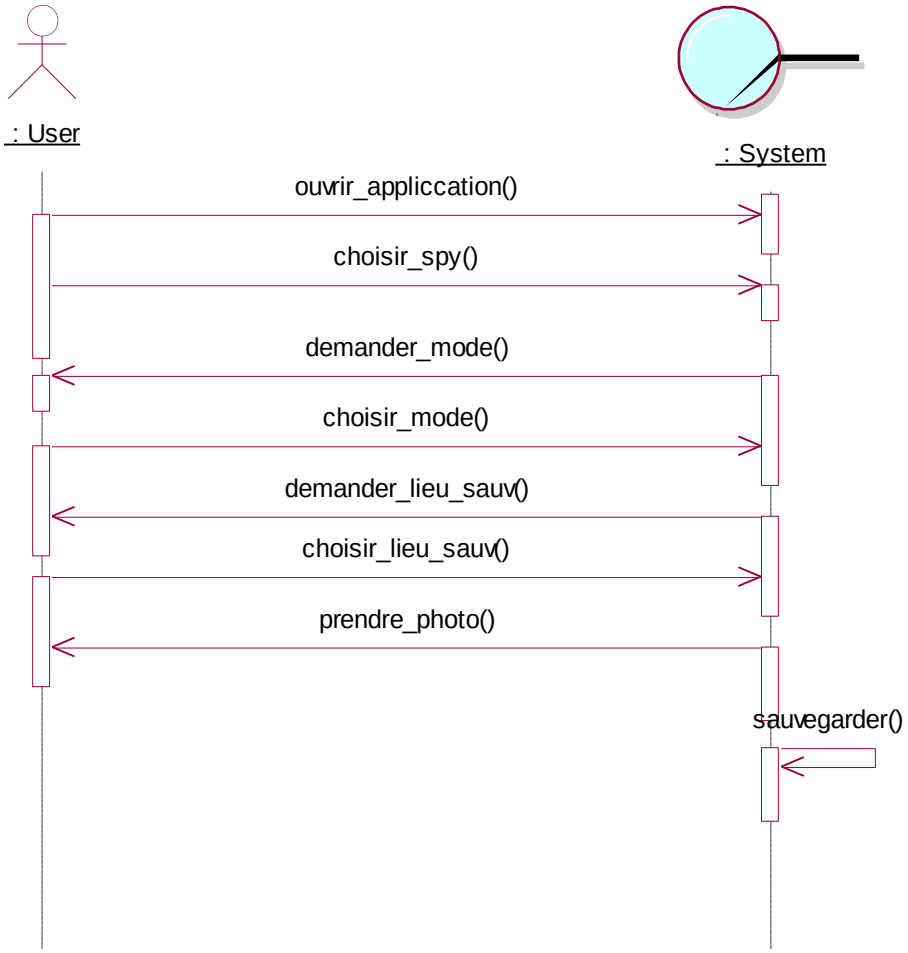
DAC :

Le diagramme d'activités décrivant les actions se déroulant tout au long de l'opération spy est :



DES :

La figure suivante montre le diagramme de séquence de cette fonction :



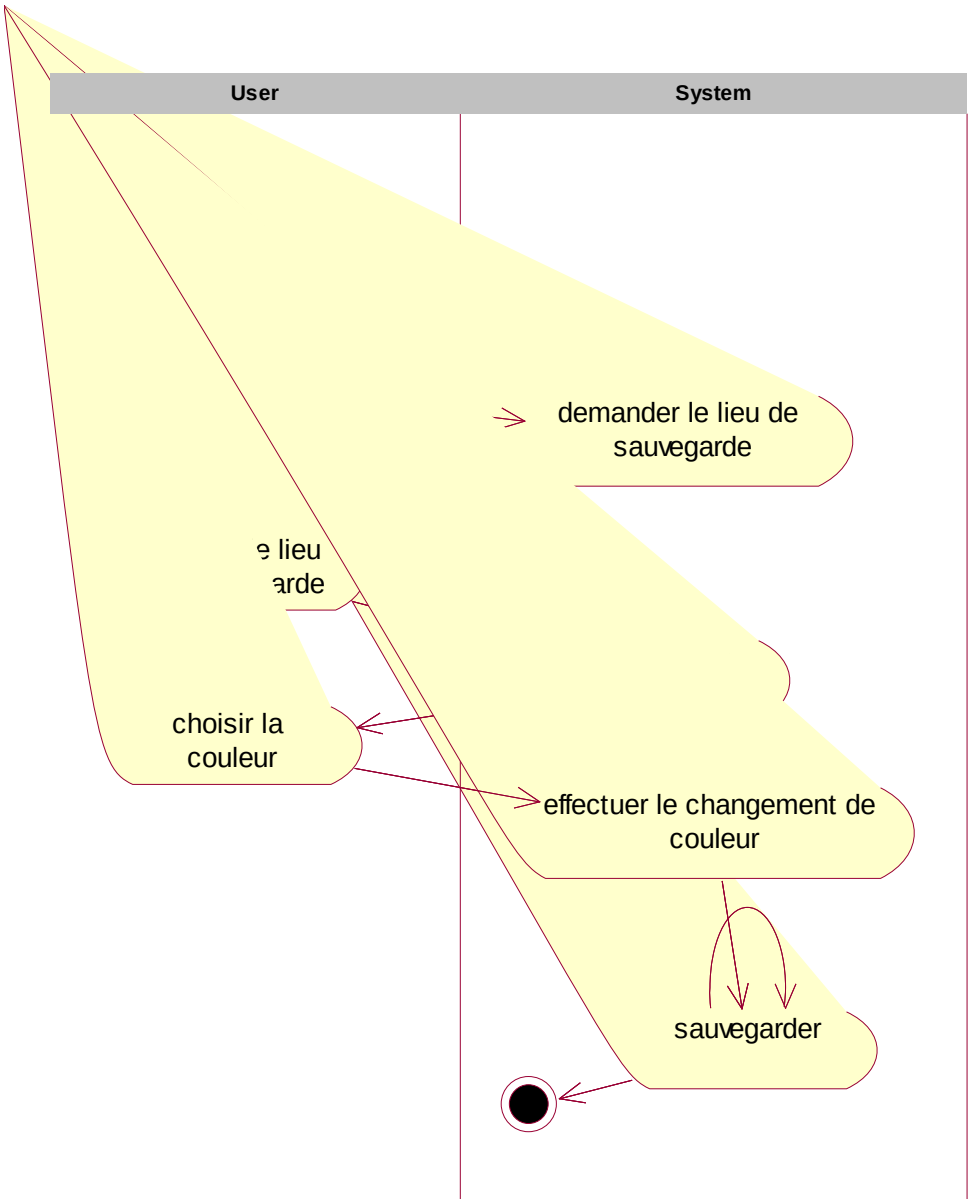
3. Changer la couleur d'une photo :

Titre de la tache: Changer la couleur d'une photo.
Responsables de la tache : L'utilisateur et le système.
Scénario alternatif : S'il n'y a pas de photo selectionnée pour effectuer le changement de couleur. Dans ce cas il le système demande de selectionner une photo.

Acteur (Utilisateur)	**Système (Webcam)**
1) L'utilisateur ouvre l'application.	
2) Il choisit l'option "color".	
	3) Le système demande à l'utilisateur où il veut sauvegarder les images.
4) L'utilisateur précise l'endroit du sauvegarde .	
	5) Le système demande de choisir la couleur.
6) L'utilisateur choisit la couleur qu'il veut.	
	7) Le système effectue alors l'opération de changer la couleur de la photo.
	8) Le système sauvegarde l'image dans l'endroit demandé.

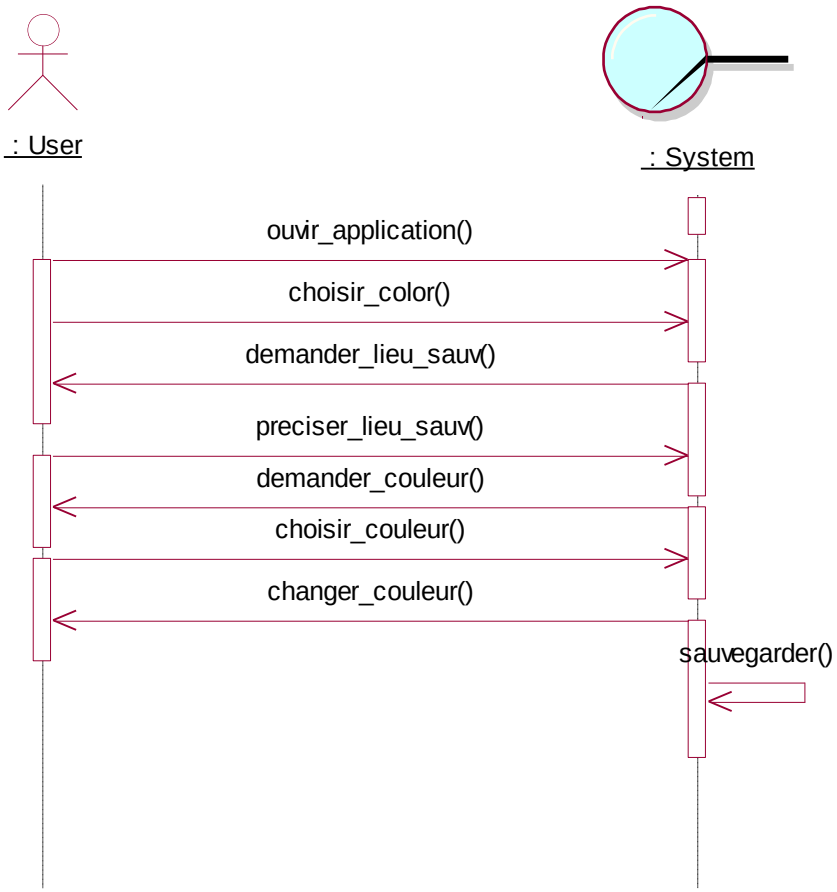
DAC :

Le diagramme d'activités est représenté par la figure ci-dessous :



DES :

Le diagramme de séquence est aussi représenté comme suit :



4. Détecter un visage:

Cette action permet de detecter un visage déjà existant dans une photo dans le système.

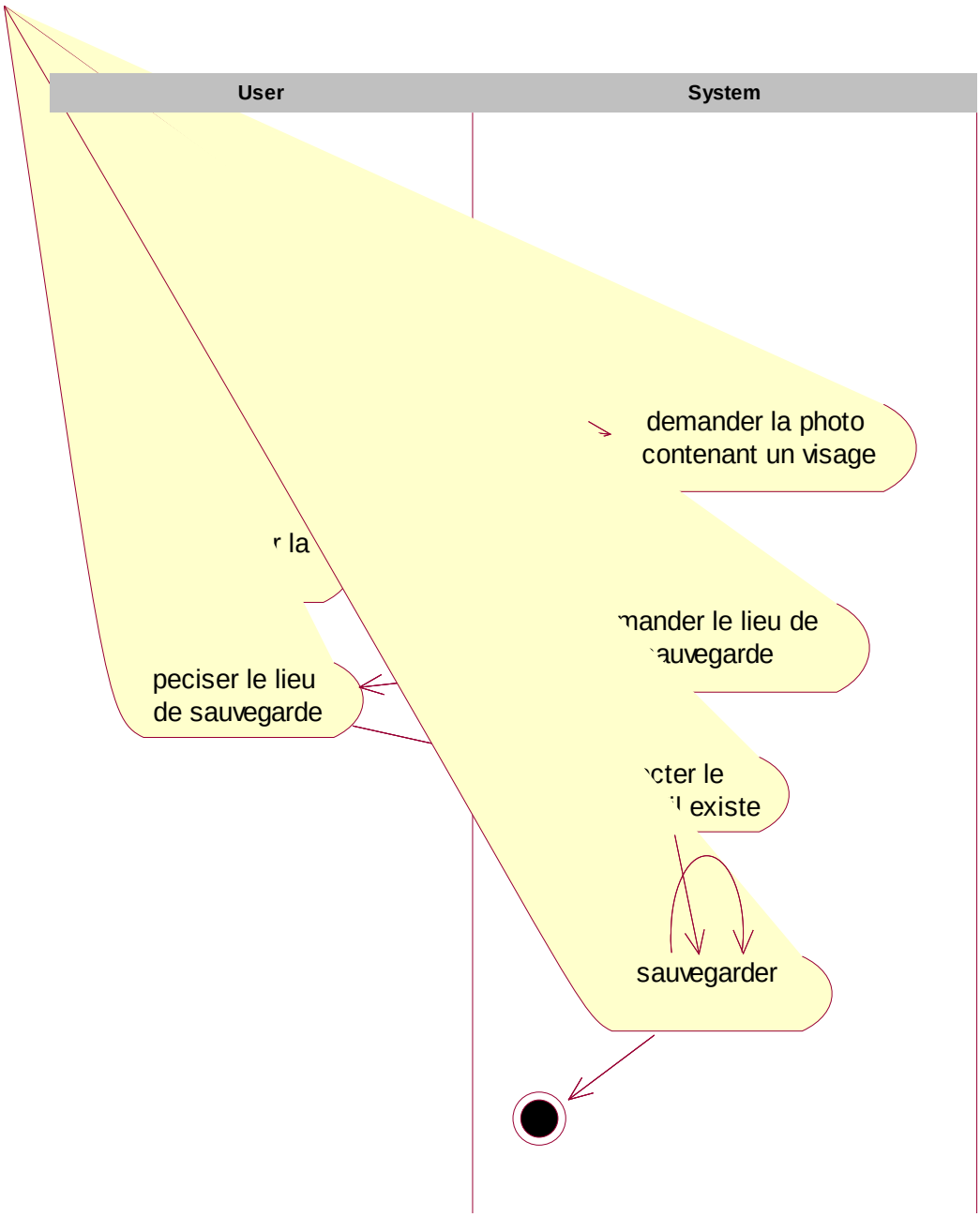
Le scénario nominal correspondant est comme suit :

Titre de la tache: Détecter un visage.
Responsables de la tache : L'utilisateur et le système.
Pré-conditions: - Présence d'une photo pour détecter le visage de cette photo.

Acteur (Utilisateur)	**Système (Webcam)**
1) L'utilisateur ouvre l'application.	
2) Il choisit l'option "face detection ".	
	3) Le système demande de choisir la photo qui contient le visage qu'il doit détecter.
4) L'utilisateur précise le mode.	
	5) Le système demande à l'utilisateur où il veut sauvegarder les images.
6) L'utilisateur précise l'endroit du sauvegarde .	
	7) Le système effectue alors l'opération de détecter ce visage s'il le trouve.
	8) Le système sauvegarde les images dans l'endroit demandé.

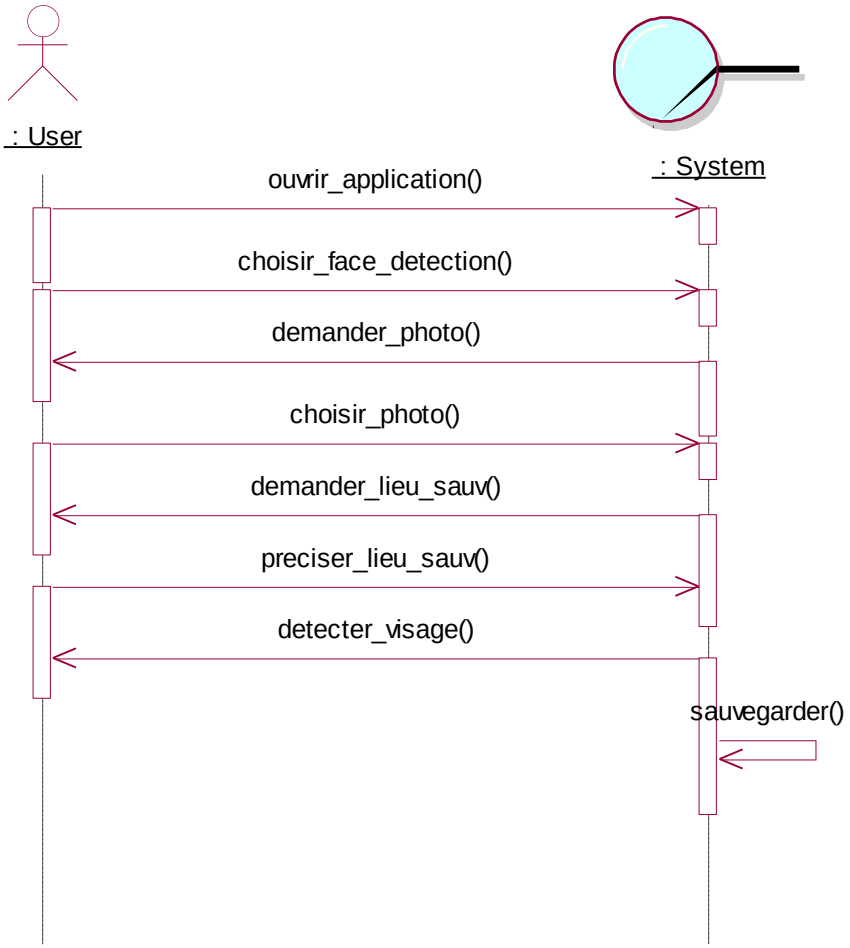
DAC :

Le diagramme d'activité de la conversation a temps réel est comme suit :



DES :

Le diagramme de séquences de la conversation a temps réel est représenté comme suit :



5. Modifier une image :

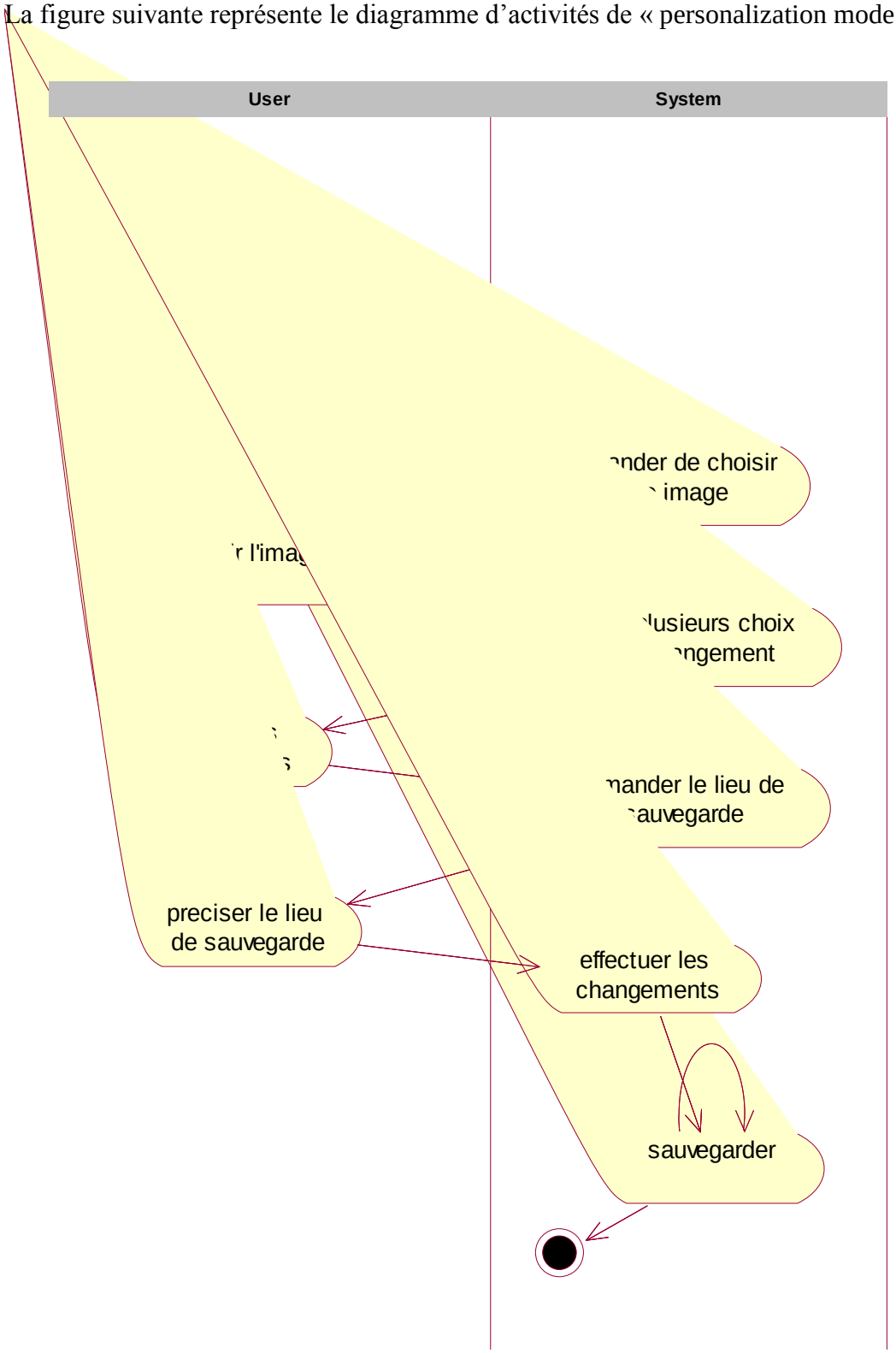
Cette fonction permet à un utilisateur de modifier une image en ajoutant des détails extérieurs
Le scénario nominal de cette fonction est :

Titre de la tache: Modifier une image
Responsables de la tache : L'utilisateur et le système.
Pré-conditions: - L'existence d'une image

Acteur (Utilisateur)	**Système (Webcam)**
1) L'utilisateur ouvre l'application.	
2) Il choisit l'option "personalization mode".	
	3) Le système demande de choisir une image.
4) L'utilisateur choisit l'image.	
	5) Le système offre plusieurs choix de changement à l'utilisateur.
6) L'utilisateur effectue le changement qu'il veut.	
	7) Le système demande à l'utilisateur où il veut sauvegarder les images.
8) L'utilisateur précise l'endroit du sauvegarde .	
	9) Le système effectue l'opération de changement pour l'image.
	10) Le système sauvegarde l'image dans l'endroit demandé.

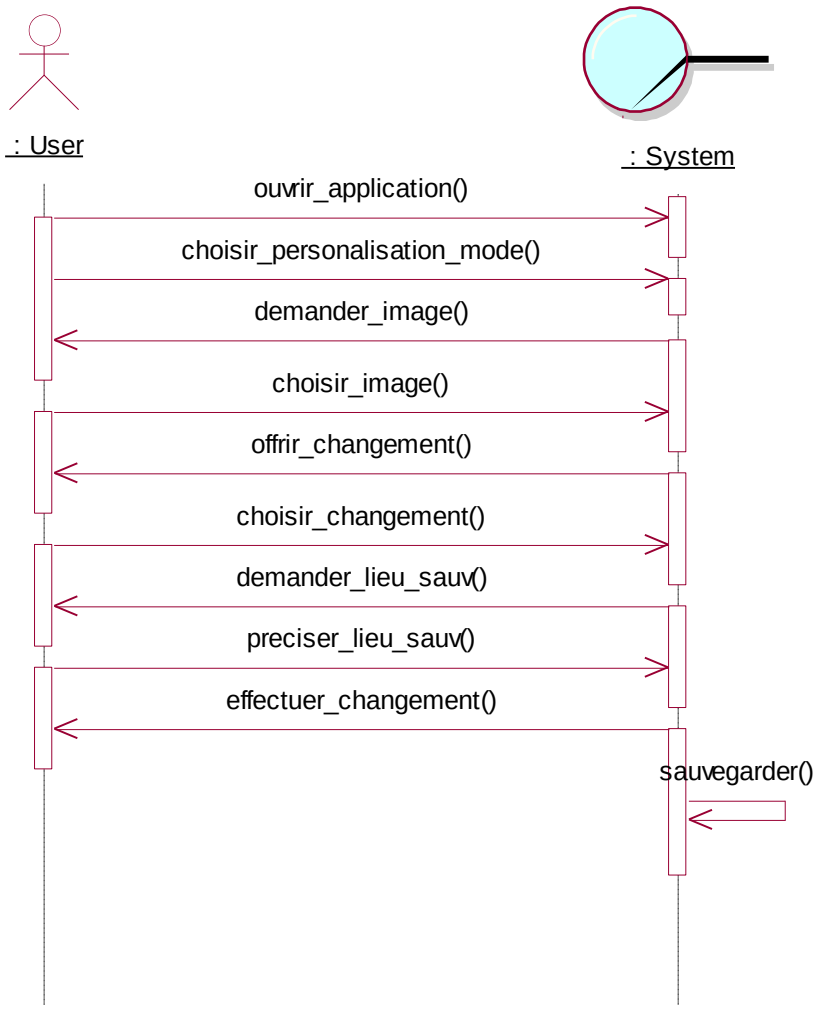
DAC :

La figure suivante représente le diagramme d'activités de « personalization mode » :



DES :

La figure suivante représente le diagramme de séquences de changement pour une image :



6. Effectuer un effet de slow motion :

A partir de cette fonction, l'utilisateur peut demander du système d'effectuer un effet de slow motion.

Cette fonction a comme scénario nominal :

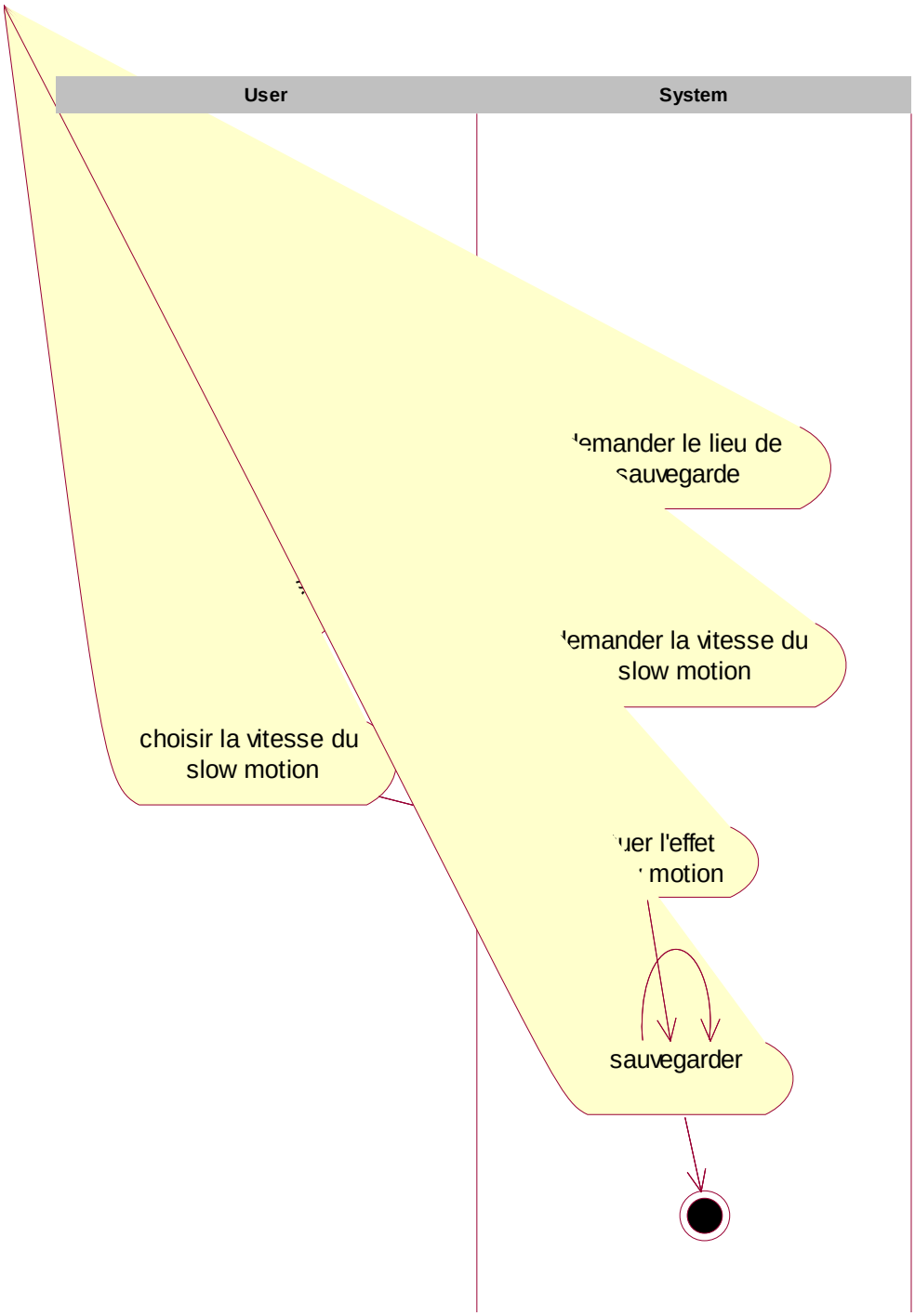
Titre de la tache: Effectuer un effet de slow motion.

Responsables de la tache : L'utilisateur et le système.

Acteur (Utilisateur)	**Système (Webcam)**
1) L'utilisateur ouvre l'application.	
2) Il choisit l'option "slow motion".	
	3) Le système demande à l'utilisateur où il veut sauvegarder les images.
4) L'utilisateur précise l'endroit de la sauvegarde.	
	5) Le système demande la vitesse de l'effet du slow motion.
6) L'utilisateur entre la vitesse qu'il préfère.	
	7) Le système effectue alors l'effet du slow motion avec la vitesse demandée par l'utilisateur.
	8) Le système sauvegarde les images dans l'endroit demandé.

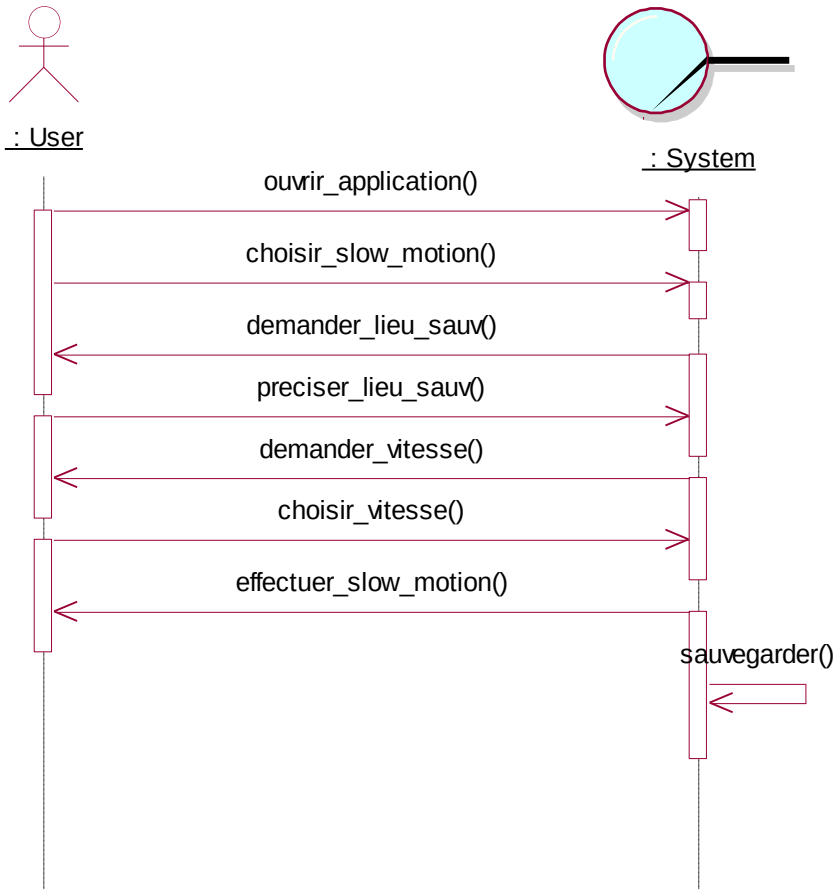
DAC :

La figure ci-dessous represente le diagramme d'activités de « slow motion».



DES :

La figure ci-après, représente le DES du slow motion :

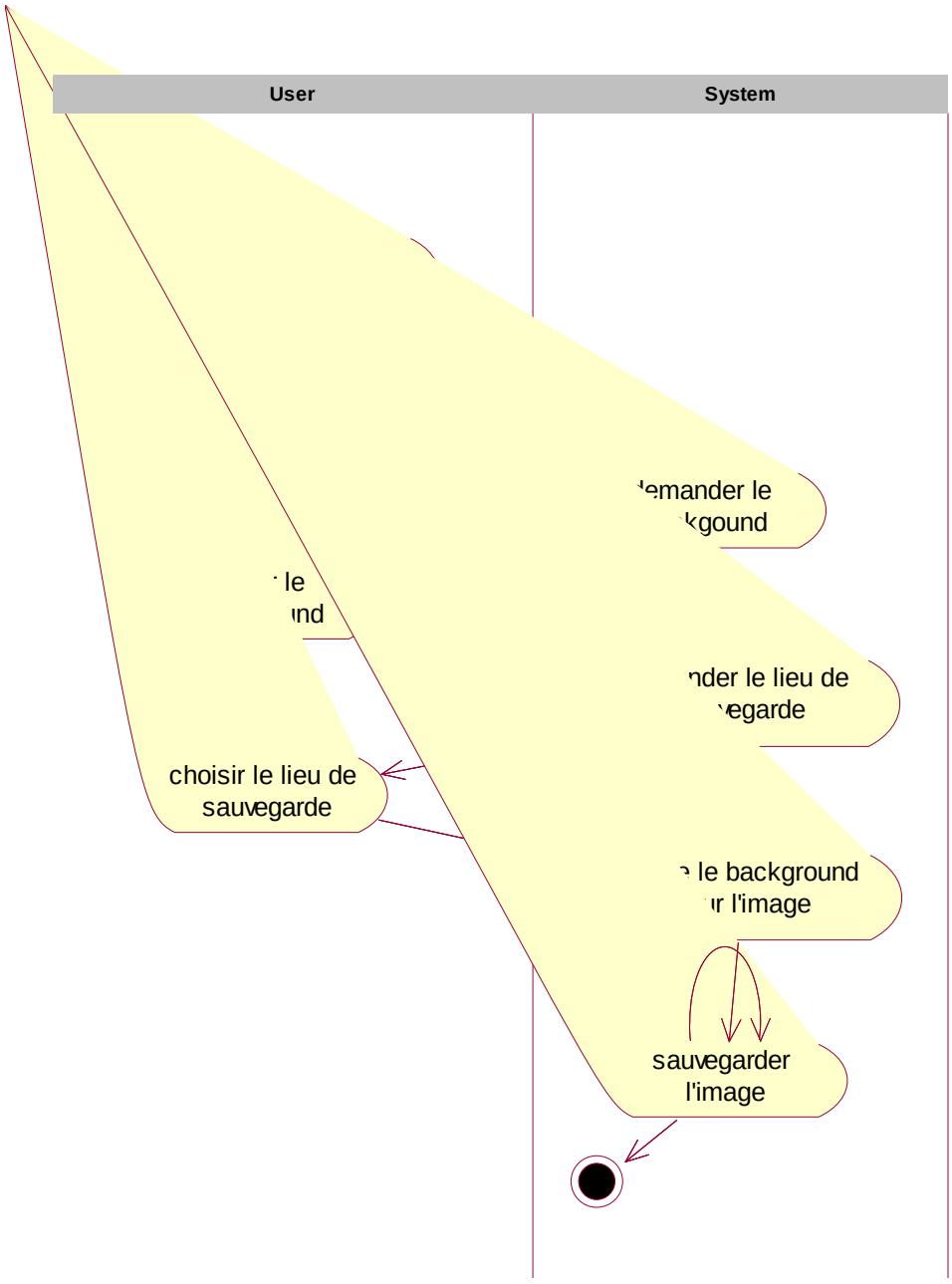


7. Mettre un background pour une image :
L'utilisateur peut mettre un background à une photo.

Le scénario nominal correspondant est comme suit :
Titre de la tache: Mettre un background pour une image
Responsables de la tache : L'utilisateur et le système.
Pré-conditions: - Présence d'une photo

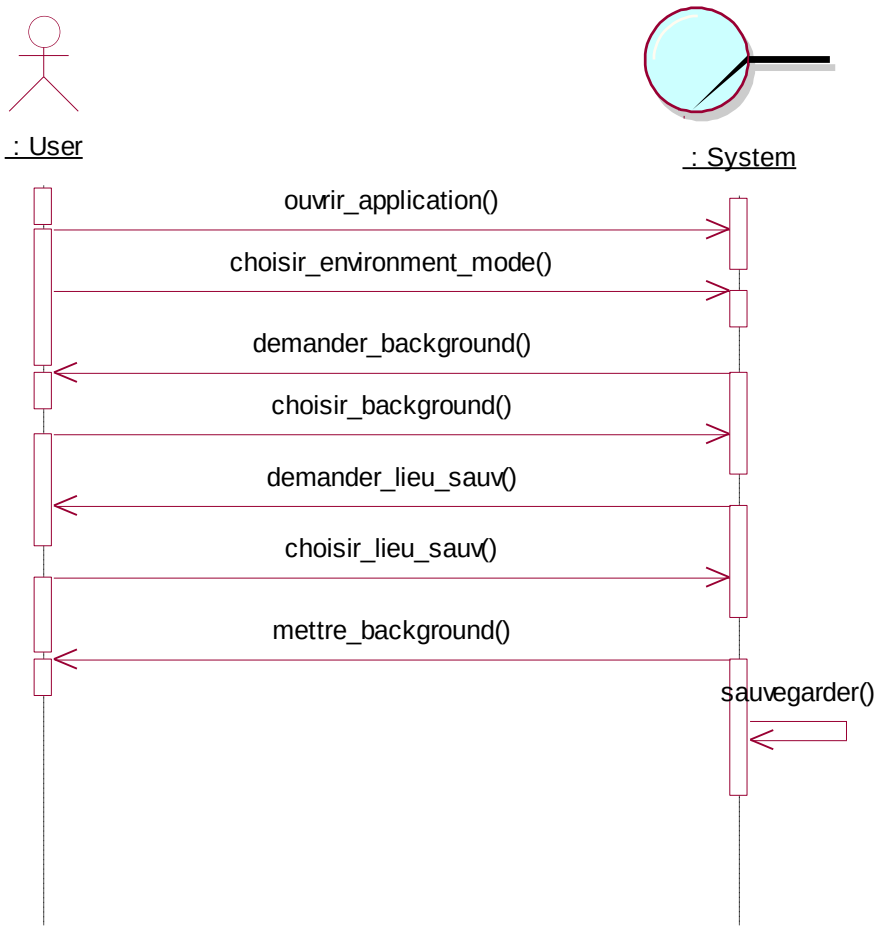
Acteur (Utilisateur)	**Système (Webcam)**
1) L'utilisateur ouvre l'application.	
2) Il choisit l'option "environment mode".	
	3) Le système demande de choisir un background pour une image.
4) L'utilisateur choisit le background.	
	5) Le système demande à l'utilisateur où il veut sauvegarder les images.
6) L'utilisateur précise l'endroit du sauvegarde.	
	7) Le système effectue l'opération de mettre un background pour l'image.
	8) Le système sauvegarde les images dans l'endroit demandé.

Le DAC correspondant est représenté par la figure suivante :



DES :

Le DES correspondant est le suivant :



8. Dessiner une photo :

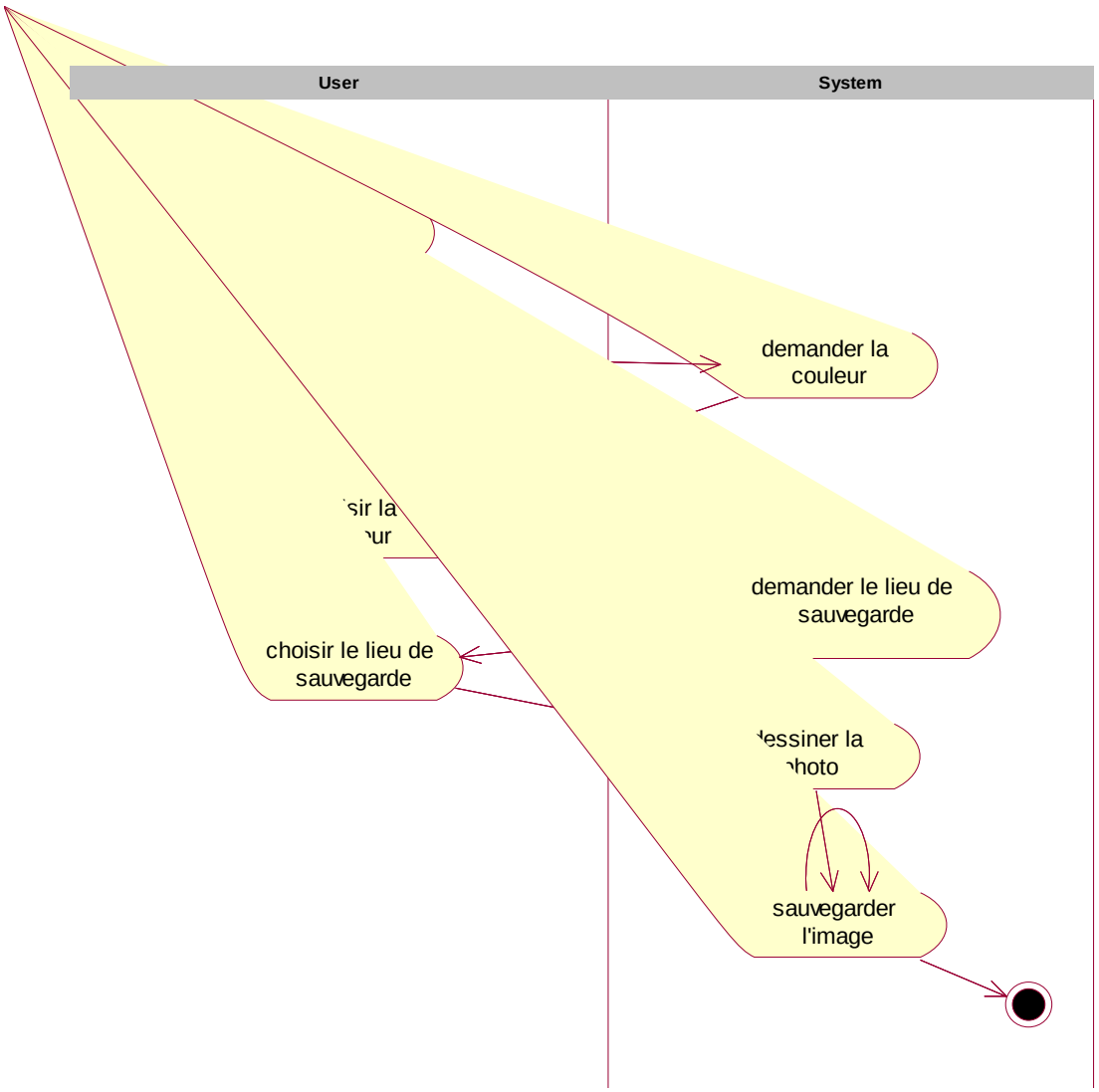
Cette fonction sert à dessiner une image qui existe déjà sous forme d'une photo avec la couleur que l'utilisateur décide.

Le scénario nominal de cette fonction est le suivant :
Titre de la tache: Dessiner une photo
Responsables de la tache : L'utilisateur et le système.
Pré-conditions: - L'existence d'une photo

Acteur (Utilisateur)	**Système (Webcam)**
1) L'utilisateur ouvre l'application.	
2) Il choisit l'option "paint".	
	3) Le système demande à l'utilisateur de lui montrer sur la camera la couleur demandée.
4) L'utilisateur lui montre la couleur qu'il veut.	
	5) Le système demande à l'utilisateur où il veut sauvegarder les images.
6) L'utilisateur précise l'endroit du sauvegarde.	
	7) Le système effectue alors l'opération de dessiner la photo.
	8) Le système sauvegarde la photo dans l'endroit demandé.

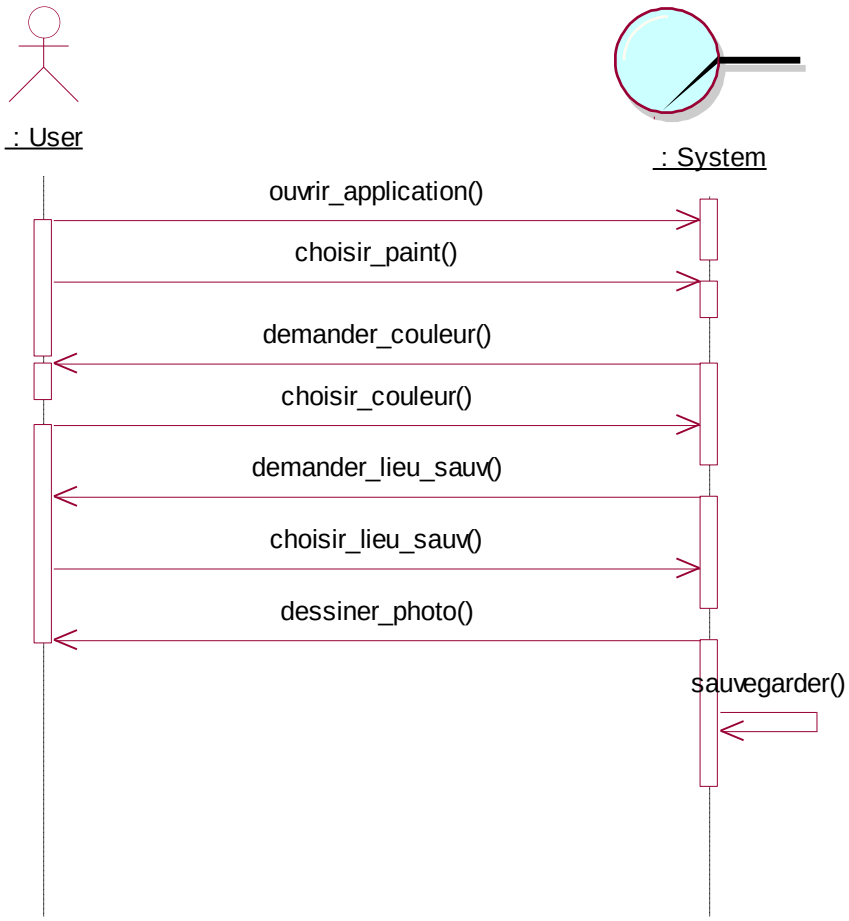
DAC :

Ci-dessous est représenté le DAC de la fonction paint :



DES :

Le DES de la fonction Paint est représenté par la figure suivante :



Conclusion :

Après la représentation des différents scénarios nominaux et diagrammes d'activités et de séquences, nous pouvons maintenant construire une idée claire sur la totalité des fonctions à accomplir par cette application.
Alors, ce projet facilitera l'utilisation du webcam en beneficiant de plusieurs autres services.
Dans le chapitre suivant, nous allons etablir un premier prototype qui illustrera plus clairement et d'une manière graphique toutes les fonctionnalités de l'application.

Chapitre 2 :
Prototype

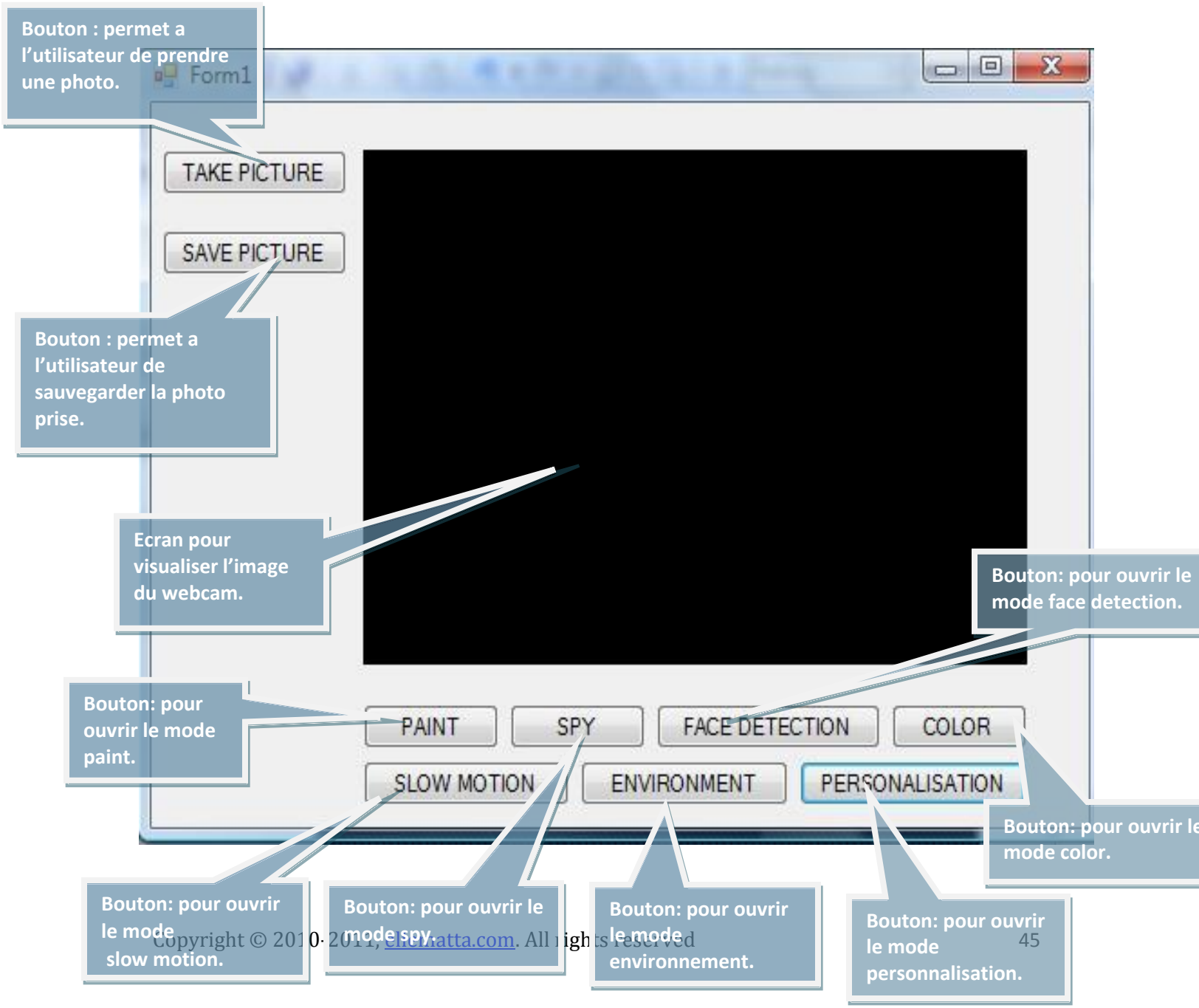
Introduction :

Pour démontrer plus précisément comment fonctionne ce WebcamMode, nous allons présenter le prototype suivant sans le code, et base de données ; puisqu'ils seront détaillés plus tard dans la phase suivante.
Ainsi, ce prototype formera uniquement une représentation graphique qui sera capable d'exécuter d'une manière statique pour le moment les taches à compléter par l'application.

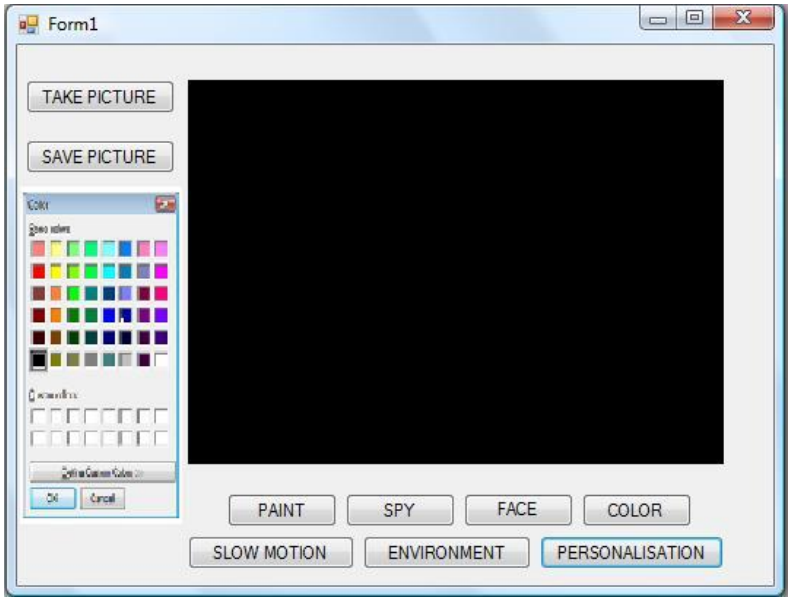
Prototype (GUI):

Voici le prototype représentant l'application à mettre sur les machines client. Il comprend les écrans suivants:

1. Ecran principal :

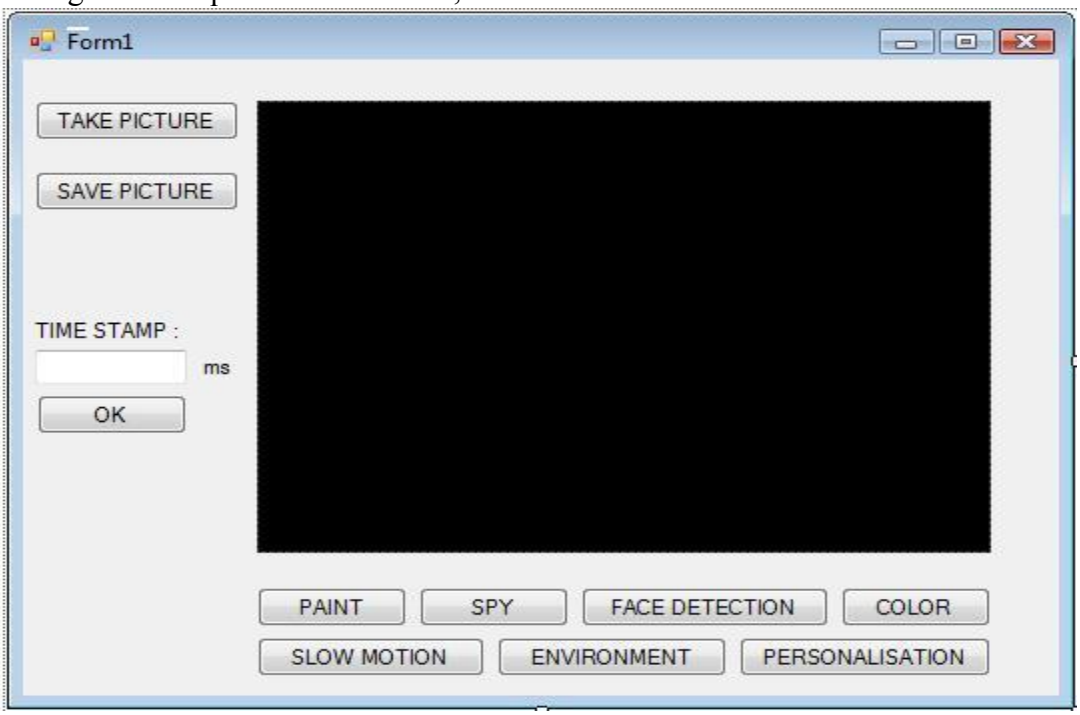


Le bouton color va nous permettre de choisir la couleur du background. En cliquant sur ce bouton, on aura la forme suivante :



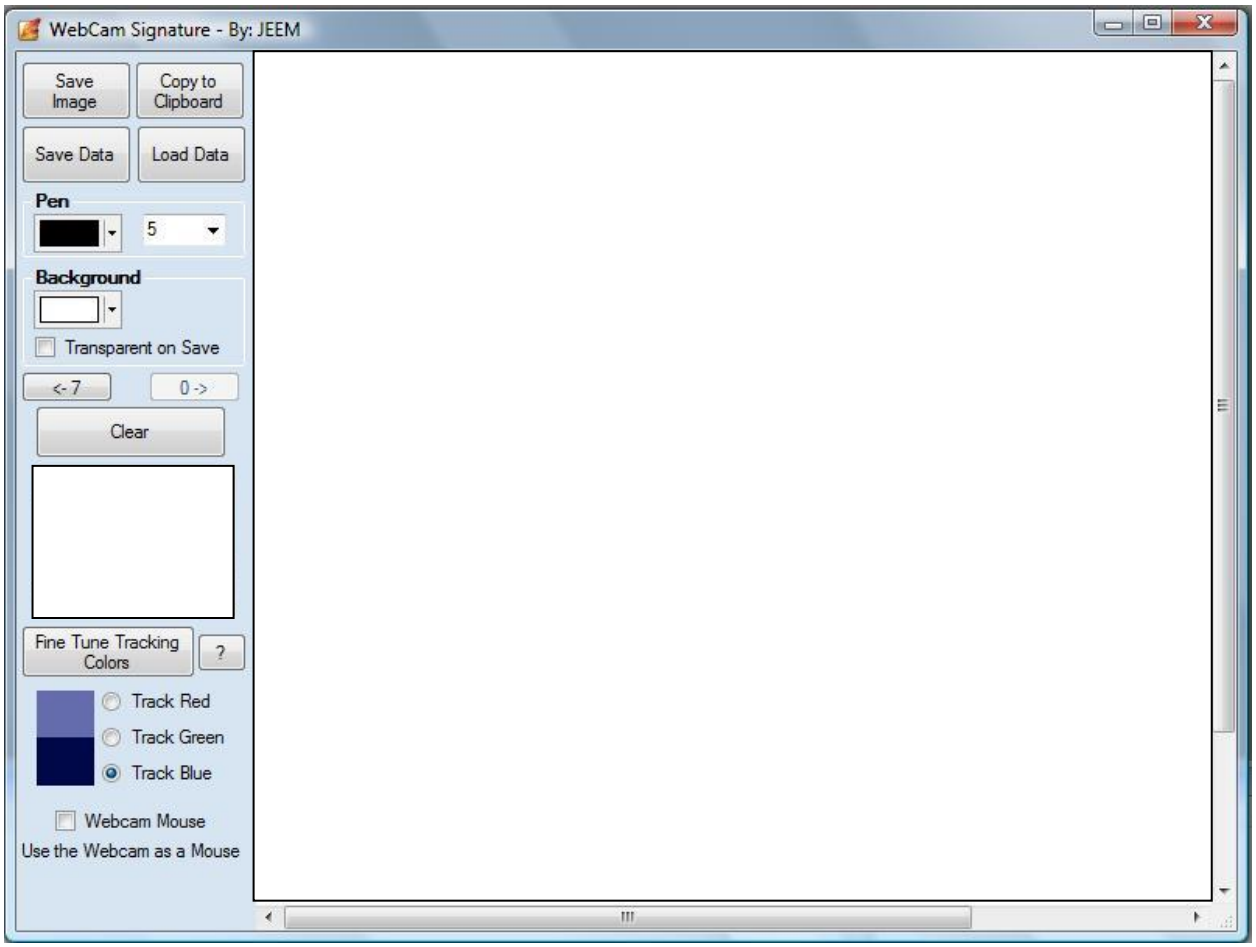
L'utilisateur a la possibilité de spécifier la couleur désirée en la choisissant de la liste des couleurs présente à gauche. En cliquant sur le bouton OK la couleur sera sélectionnée.

Le bouton slow motion va nous permettre de choisir le délai du temps entre deux prises d'images. En cliquant sur ce bouton, on aura la forme suivante :



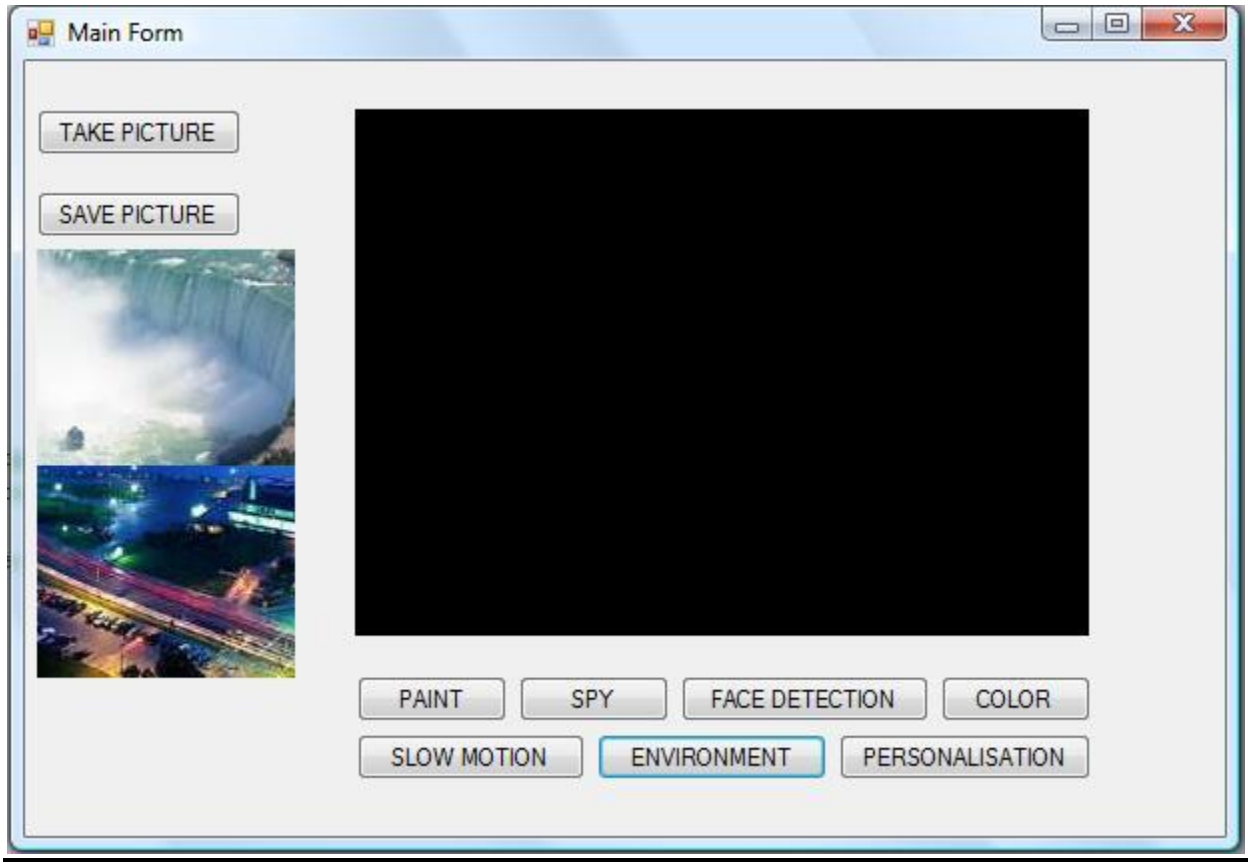
L'utilisateur a la possibilité de spécifier le temps désiré en le spécifiant dans le textbox présente à gauche de la fenêtre. En cliquant sur le bouton OK le temps sera sélectionné.

Le bouton Paint va nous permettre de dessiner a l'aide d'un crayon et du webcam, on aura la forme suivante :

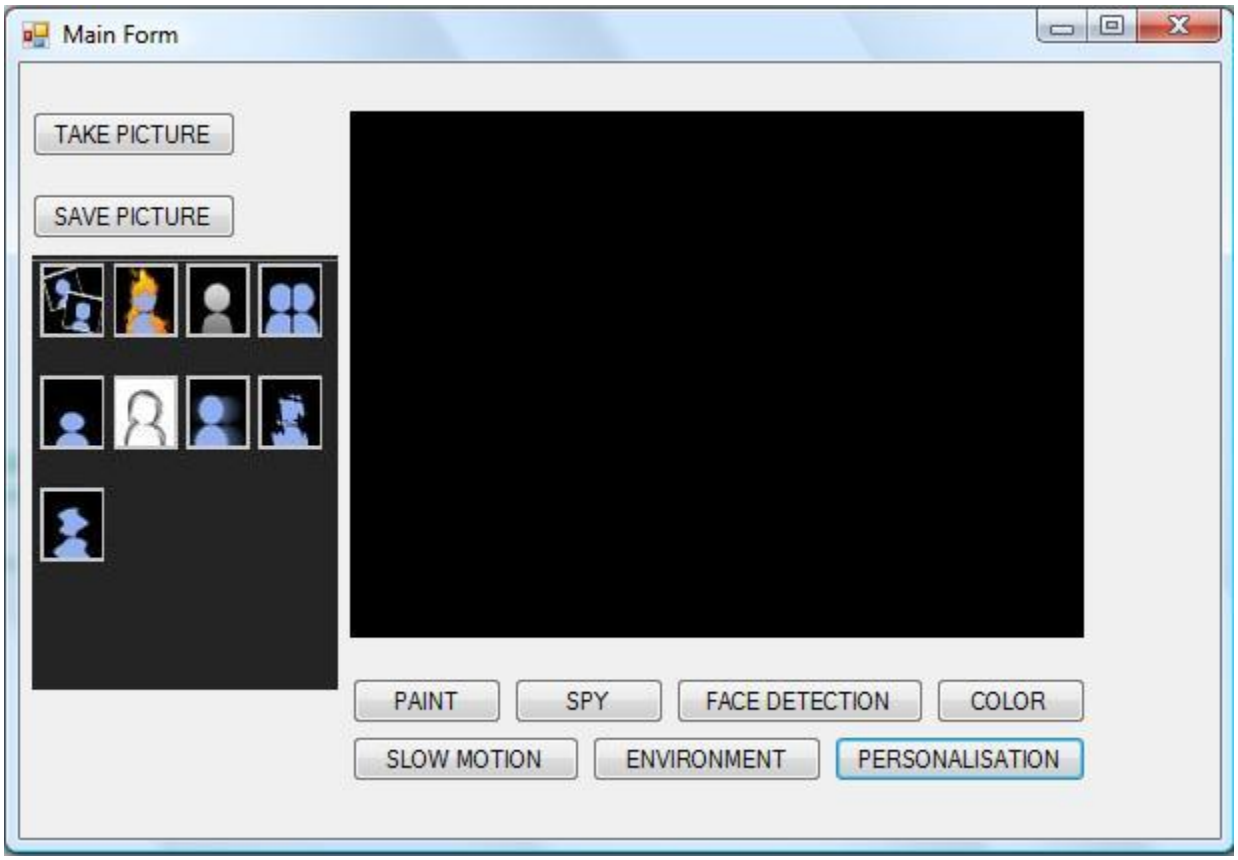


Le programme demande de choisir une certaine couleur d'objet et de mettre en relief l'objet désiré pour élaborer le croquis. Une fois l'objet détecté, l'utilisateur n'a qu'à déplacer l'objet dans une certaine direction pour dessiner le croquis.

Le bouton Environment va nous permettre de dessiner a l'aide d'un crayon et du webcam, on aura la forme suivante :



Le bouton Personalisation va nous permettre de dessiner a l'aide d'un crayon et du webcam, on aura la forme suivante :



Conclusion :

Pour conclure, ce "WebcamModes" favorise la communication entre l'utilisateur qui interagissent avec le système et son webcam. La communication peut etre sous formes de plusieurs modes: **Slow Motion**, **Take Picture** , **Paint**, **Color, Spy, Face Detection , Environment, Personalization**.Le DCU, les SN, DAC, DES et le DCL permettent de décrire les différentes fonctionnalités du système. Le prototype qui utilise la base de données décrite précédemment donne une idée claire de l'application, de son interface graphique, de ses fonctions, du squelette de son code en détaillant certaines taches. Finalement on peut dire que ce document couvre les 2 premières phases du RUP (la phase de lancement et la phase d'élaboration.

Chapitre 3 :

Squelette du code

1) Introduction

On va nommer dans cette partie tous les espaces des noms utilisés dans notre programme et définir les espaces des noms les plus utilisés.

2) Les espaces des noms utilisés et explication

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Runtime.InteropServices;
using System.Threading;
using System.IO;
using SampleGrabberNET;
using DShowNET;
using DShowNET.Device;
using FancyColorDialog;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using System.Reflection;
```

Parlons alors de l'espace de nom les plus important:

A) DShowNET

DShowNet est un excellent emballeur qui permet une interface simple avec la webcam. On a utilise la version 1.0 de DShowNet car elle est plus vite de la version 2.0

B) System.Runtime.InteropServices

L'espace de noms System.Runtime.InteropServices fournit une grande variété de membres qui prennent en charge COM Interop et les services d'appel de plate-forme.

Les membres de cet espace de noms fournissent plusieurs catégories de fonctionnalités. Les attributs contrôlent le marshaling, notamment le mode d'organisation des structures et de représentation des chaînes. Les principaux attributs sont DllImportAttribute, qui permet de définir les méthodes d'appel de plate-forme pour accéder aux API non managées, et MarshalAsAttribute, qui permet de spécifier comment les données doivent être marshalées entre la mémoire managée et non managée.

C) System.Threading

L'espace de noms System.Threading fournit des classes et des interfaces permettant la programmation multithread. En plus des classes destinées à la synchronisation des activités des threads et de l'accès aux données (Mutex, Monitor, Interlocked, AutoResetEvent, etc.), cet espace de noms comprend une classe ThreadPool qui permet d'utiliser un pool de threads fournis par le système et une classe Timer qui exécute des méthodes de rappel sur les threads du pool.

3) Squelette du code

Program.cs

```
//Class 1

using System;
using System.Collections.Generic;
//using System.Linq;
using System.Windows.Forms;

namespace WebCam_Signature
{
    static class Program
    {

        // The main entry point for the application.
        /* (1) We are going to use Application instance that provides static
methods and properties
         * to manage an application, such as methods to start and stop an
application, to process Windows messages,
         * and properties to get information about an application.
         * (2) STAThread indicates that the COM threading model for an
application is single-threaded which allow to
         * create only one thread to the webcam */

        [STAThread] // (2)
        static void Main()
        {
            Application.EnableVisualStyles(); //(1)
            Application.SetCompatibleTextRenderingDefault(false); //(1)

            // Allow only one instance of the application to run at a time
            if (SingleInstance.SingleApplication.Run(new MainForm()))
//Creates an instance of the class MainForm
            {
                return;
            }
        }
    }
}
```

MainForm.cs

```
//Class 2


// This program devided to 9 related regions allows a user to write his/her
name with a webcam and a colored object.
// The program tracks colors to determine what to draw and creates an array
of points (ArrayList SignaturePoints)
// based on the path that the colored object moves.
//
// The program creates smooth signatures by drawing cardinal splines through
the array of points
// where possible.
//
// The original purpose of Webcam Signature is to create a mouse moving
program
// that just utilizes a person's hands with no other objects needed.

// The tracking code is located in the region "Core Code".
// The most important function is private int GetPoint(....)
// This function applies the background filtering and looks at the pixels in
the images
// captured from the webcam to find the pixels with the wanted colors.
// It then averages the location of the valid pixels to determine at what
point on the screen
// the user is referencing.

// (1) The program utilizes DShowNet to interface with the webcam. It uses a
really
// old version (1.0), because we found it to be faster than the new version
(2.0).

// This version of Webcam Signature introduces background filtering.

// In the next version, we need to improve the tracking of the program near
the edges of the captured
// webcam images. We also need to code the dialation to reduce noise.
// Right now, the program tracks really accurately when there is even
lighting in the room and the color
// of the object stands out. It also works really well when you use a
flashlight in place of a pen.

using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Runtime.InteropServices;
using System.Threading;
using System.IO;
using SampleGrabberNET;
using DShowNET; //(1)
```

```
using DShowNET.Device; //(1)
using FancyColorDialog;


namespace WebCam_Signature
{
    public partial class MainForm : Form, ISampleGrabberCB //We need to
inherit from the ISampleGrabberCB interface that handles the buffering in the
webcam
    {
        #region Constants & Variables
        //Region 1 - Constants & Variables: This region will hold the
constans and variables used all along the main form

        //Webcam Resolution: 640x480 with 20 fps
        const int iHeight = 480;
        const int iWidth = 640;
        const int FrameRate = 20;

        int RedL, RedH, GreenL, GreenH, BlueL, BlueH;
        float PenThickness;
        bool isKeyDown;
        bool DoneProcessing;
        public bool FormActivated;
        int DrawingWidth, DrawingHeight;
        bool WebcamMouseMode;

        // Variables used to store the Bitmap images of the drawing area
        Bitmap SignaturePixels;
        Bitmap ProcessPixels;
        UnsafeBitmap WebCamBitmap;

        // Variables to store the color for the signature and the signature
background color
        PixelData SignatureColor;
        Color SignatureColorSystem;
        PixelData backColor;
        Color backColorSystem;

        // SignaturePoints stores the points recorded for the movement path
of the colored object.
        // Undo and Redo are arrays of SignaturePoints at various states.
        ArrayList SignaturePoints = new ArrayList();
        ArrayList Undo = new ArrayList();
        ArrayList Redo = new ArrayList();

        public FineTuneTrackColorsForm FineTuneTC; //Instance on the class
FineTuneTrackColorsForm
        ColorButton PenColorButton = new ColorButton();
        ColorButton BackgroundColorButton = new ColorButton();

        // Variables used to store data for the background average colors
        // and the behavior for background filtering
        double[, ,] BackgroundAverage;
        int RequiredNumBackgroundFrames;
        int NumBackgroundFramesCaptured;
        int BackgroundDiffThreshold;
```

```
        int BackgroundCalibrateTime;
        int NumBackgroundFramesRequiredCalibration;
        bool NoBackgroundFilter;
        bool UseBackgroundStaticFilter;
        bool UseBackgroundDynamicFilter;

        #endregion

        #region Mouse Control
        //Region 2 - Mouse Control: Mouse simulation to simulate and follow
the mouse movement

        private const UInt32 MOUSEEVENTF_LEFTDOWN = 0x0002;
        private const UInt32 MOUSEEVENTF_LEFTUP = 0x0004;
        private const UInt32 MOUSEEVENTF_RIGHTDOWN = 0x0008;
        private const UInt32 MOUSEEVENTF_RIGHTUP = 0x0010;
        private const UInt32 MOUSEEVENTF_MIDDLEDOWN = 0x20;
        private const UInt32 MOUSEEVENTF_MIDDLEUP = 0x40;
        private const UInt32 MOUSEEVENTF_WHEEL = 0x800;

        [DllImport("user32.dll")]
        private static extern void mouse_event(UInt32 dwFlags, UInt32 dx,
UInt32 dy, UInt32 dwData, IntPtr dwExtraInfo);
        /*
         *  UInt32 dwFlags, // motion and click options
            UInt32 dx, // horizontal position or change
            UInt32 dy, // vertical position or change
            UInt32 dwData, // wheel movement
            IntPtr dwExtraInfo // application-defined information
         */
        public static void SendClick(Point location)
        {
            Cursor.Position = location;
            mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, new IntPtr());
            mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, new IntPtr());
        }

        #endregion

        #region WebCam API
        //Region 3 - WebCam API: WebCam Application Programming Interface(API
is an interface that a software program implements in order to allow other
software to interact with it
        //much in the same way that software might implement a user interface
in order to allow humans to use it.)

        [DllImport("user32.dll")]
        static extern IntPtr GetForegroundWindow();

        // The following code in region WebCam API is a modified version of
the image capturing code
        // from the first version of the DShowNet library. It utilizes
DirectX 8.1 or higher.
        // DShowNet is an excellent wrapper to allow easy interfacing with
the webcam.
        // Most other image capturing code utilizes the clipboard as the
interface to the webcam.
```

```
        // Flag to detect first Form appearance.
        private bool firstActive;

        // Base filter of the actually used video devices : DShowNET
        private IBaseFilter capFilter;

        // Graph builder interface : DShowNET
        private IGraphBuilder graphBuilder;

        // Capture graph builder interface : DShowNET
        private ICaptureGraphBuilder2 capGraph;
        private ISampleGrabber sampGrabber;

        // Control interface : DShowNET
        private IMediaControl mediaCtrl;

        // Event interface : DShowNET
        private IMediaEventEx mediaEvt;

        // Video window interface : DShowNET
        private IVideoWindow videoWin;

        // Grabber filter interface : DShowNET
        private IBaseFilter baseGrabFlt;

        // Structure describing the bitmap to grab : DShowNET
        private VideoInfoHeader videoInfoHeader;
        private bool captured = true;
        private int bufferedSize;

        // Buffer for bitmap data.
        private byte[] savedArray;

        // List of installed video devices.
        private ArrayList capDevices;

        private const int WM_GRAPHNOTIFY = 0x00008001;      // message from
graph

        private const int WS_CHILD = 0x40000000;      // attributes for video
window
        private const int WS_CLIPCHILDREN = 0x02000000;
        private const int WS_CLIPSIBLINGS = 0x04000000;

        //Event when callback has finished (ISampleGrabberCB.BufferCB).
        private delegate void CaptureDone();

        private void InitializeCamera()
        {
            if (!DsUtils.IsCorrectDirectXVersion()) //Direct X Version
            {
                MessageBox.Show(this, "DirectX 8.1 or higher NOT
installed!\n" + "Webcam functionality not available.\n", "Webcam Signature",
MessageBoxButtons.OK, MessageBoxIcon.Stop);
                //this.Close();
                return;
```



```
            }

            if (!DsDev.GetDevicesOfCat(FilterCategory.VideoInputDevice, out
capDevices)) //If there is no video input device
            {
                MessageBox.Show(this, "No video capture devices found!\n" +
"Please enable your webcam and restart the program.\n", "Webcam Signature",
MessageBoxButtons.OK, MessageBoxIcon.Stop);
                return;
            }

            DsDevice dev = null;
            if (capDevices.Count == 1)
                dev = capDevices[0] as DsDevice;
            else
            {
                DeviceSelector selector = new DeviceSelector(capDevices);
                selector.ShowDialog(this);
                dev = selector.SelectedDevice;
            }

            if (dev == null)
            {
                this.Close(); return;
            }

            if (!StartupVideo(dev.Mon))
                this.Close();
        }

        // Start all the interfaces, graphs and preview window.
        bool StartupVideo(UCOMIMoniker mon)
        {
            int hr;
            try
            {
                if (!CreateCaptureDevice(mon))
                    return false;

                if (!GetInterfaces())
                    return false;

                if (!SetupGraph())
                    return false;

                if (!SetupVideoWindow())
                    return false;

                hr = mediaCtrl.Run();
                if (hr < 0)
                    Marshal.ThrowExceptionForHR(hr);
                        return true;
            }
            catch (Exception ee)
            {
                MessageBox.Show(this, "Could not start video stream\r\n" +
ee.Message, "Webcam Signature", MessageBoxButtons.OK, MessageBoxIcon.Stop);
```



```
                return false;
            }
        }

        // Make the video preview window to show in videoPanel.
        bool SetupVideoWindow()
        {
            int hr;
            try
            {
                // Set the video window to be a child of the main window
                hr = videoWin.put_Owner(CameraPreviewPanel.Handle);
                if (hr < 0)
                    Marshal.ThrowExceptionForHR(hr);

                // Set video window style
                hr = videoWin.put_WindowStyle(WS_CHILD | WS_CLIPCHILDREN);
                if (hr < 0)
                    Marshal.ThrowExceptionForHR(hr);

                // Use helper function to position video window in client
rect of owner window
                ResizeVideoWindow();

                // Make the video window visible, now that it is properly
positioned
                hr = videoWin.put_Visible(DsHlp.OATRUE);
                if (hr < 0)
                    Marshal.ThrowExceptionForHR(hr);

                hr = mediaEvt.SetNotifyWindow(this.Handle, WM_GRAPHNOTIFY,
IntPtr.Zero);
                if (hr < 0)
                    Marshal.ThrowExceptionForHR(hr);
                return true;
            }
            catch (Exception ee)
            {
                MessageBox.Show(this, "Could not setup video window\r\n" +
ee.Message, "Webcam Signature", MessageBoxButtons.OK, MessageBoxIcon.Stop);
                return false;
            }
        }

        // Build the capture graph for grabber.
        bool SetupGraph()
        {
            int hr;
            try
            {
                hr = capGraph.SetFiltergraph(graphBuilder);
                if (hr < 0)
                    Marshal.ThrowExceptionForHR(hr);

                hr = graphBuilder.AddFilter(capFilter, "Ds.NET Video Capture
Device");
                if (hr < 0)
```

```
                Marshal.ThrowExceptionForHR(hr);


            AMMediaType media = new AMMediaType();
            media.majorType = MediaType.Video;
            media.subType = MediaSubType.RGB24;
            media.formatType = FormatType.VideoInfo;

            // The following hard codes the video capture size to 640 x
480
            media.formatSize = 0;
            media.sampleSize = 0;

            hr = sampGrabber.SetMediaType(media);
            if (hr < 0)
                Marshal.ThrowExceptionForHR(hr);

            hr = graphBuilder.AddFilter(baseGrabFlt, "Ds.NET Grabber");
            if (hr < 0)
                Marshal.ThrowExceptionForHR(hr);

            Guid cat = PinCategory.Preview;
            Guid med = MediaType.Video;
            hr = capGraph.RenderStream(ref cat, ref med, capFilter, null,
null); // baseGrabFlt
            if (hr < 0)
                Marshal.ThrowExceptionForHR(hr);

            cat = PinCategory.Capture;
            med = MediaType.Video;
            hr = capGraph.RenderStream(ref cat, ref med, capFilter, null,
baseGrabFlt); // baseGrabFlt
            if (hr < 0)
                Marshal.ThrowExceptionForHR(hr);

            media = new AMMediaType();
            hr = sampGrabber.GetConnectedMediaType(media);
            if (hr < 0)
                Marshal.ThrowExceptionForHR(hr);
            if ((media.formatType != FormatType.VideoInfo) ||
(media.formatPtr == IntPtr.Zero))
                throw new NotSupportedException("Unknown Grabber Media
Format");

            videoInfoHeader =
(VideoInfoHeader)Marshal.PtrToStructure(media.formatPtr,
typeof(VideoInfoHeader));
            Marshal.FreeCoTaskMem(media.formatPtr); media.formatPtr =
IntPtr.Zero;

            hr = sampGrabber.SetBufferSamples(false);
            if (hr == 0)
                hr = sampGrabber.SetOneShot(false);
            if (hr == 0)
                hr = sampGrabber.SetCallback(null, 0);
            if (hr < 0)
                Marshal.ThrowExceptionForHR(hr);
```

```
                return true;
            }
            catch (Exception ee)
            {
                MessageBox.Show(this, "Could not setup graph\r\n" +
ee.Message, "Webcam Signature", MessageBoxButtons.OK, MessageBoxIcon.Stop);
                return false;
            }
        }


        // Create the used COM components and get the interfaces.
        bool GetInterfaces()
        {
            Type comType = null;
            object comObj = null;
            try
            {
                comType = Type.GetTypeFromCLSID(Clsid.FilterGraph);
                if (comType == null)
                    throw new NotImplementedException(@"DirectShow
FilterGraph not installed/registered!");
                comObj = Activator.CreateInstance(comType);
                graphBuilder = (IGraphBuilder)comObj; comObj = null;

                Guid clsid = Clsid.CaptureGraphBuilder2;
                Guid riid = typeof(ICaptureGraphBuilder2).GUID;
                comObj = DsBugWO.CreateDsInstance(ref clsid, ref riid);
                capGraph = (ICaptureGraphBuilder2)comObj; comObj = null;

                comType = Type.GetTypeFromCLSID(Clsid.SampleGrabber);
                if (comType == null)
                    throw new NotImplementedException(@"DirectShow
SampleGrabber not installed/registered!");
                comObj = Activator.CreateInstance(comType);
                sampGrabber = (ISampleGrabber)comObj; comObj = null;

                mediaCtrl = (IMediaControl)graphBuilder;
                videoWin = (IVideoWindow)graphBuilder;
                mediaEvt = (IMediaEventEx)graphBuilder;
                baseGrabFlt = (IBaseFilter)sampGrabber;
                return true;
            }
            catch (Exception ee)
            {
                MessageBox.Show(this, "Could not get interfaces\r\n" +
ee.Message, "Webcam Signature", MessageBoxButtons.OK, MessageBoxIcon.Stop);
                return false;
            }
            finally
            {
                if (comObj != null)
                    Marshal.ReleaseComObject(comObj); comObj = null;
            }
        }
```

```
        // Create the user selected capture device.
        bool CreateCaptureDevice(UCOMIMoniker mon)
        {
            object capObj = null;
            try
            {
                Guid gbf = typeof(IBaseFilter).GUID;
                mon.BindToObject(null, null, ref gbf, out capObj);
                capFilter = (IBaseFilter)capObj; capObj = null;
                return true;
            }
            catch (Exception ee)
            {
                MessageBox.Show(this, "Could not create capture device\r\n" +
ee.Message, "Webcam Signature", MessageBoxButtons.OK, MessageBoxIcon.Stop);
                return false;
            }
            finally
            {
                if (capObj != null)
                    Marshal.ReleaseComObject(capObj); capObj = null;
            }

        }

        // Do cleanup and release DirectShow.
        void CloseInterfaces()
        {
            int hr;
            try
            {

                if (mediaCtrl != null)
                {
                    hr = mediaCtrl.Stop();
                    mediaCtrl = null;
                }

                if (mediaEvt != null)
                {
                    hr = mediaEvt.SetNotifyWindow(IntPtr.Zero,
WM_GRAPHNOTIFY, IntPtr.Zero);
                    mediaEvt = null;
                }

                if (videoWin != null)
                {
                    hr = videoWin.put_Visible(DsHlp.OAFALSE);
                    hr = videoWin.put_Owner(IntPtr.Zero);
                    videoWin = null;
                }

                baseGrabFlt = null;
                if (sampGrabber != null)
                    Marshal.ReleaseComObject(sampGrabber); sampGrabber =
null;
```



```
                if (capGraph != null)
                    Marshal.ReleaseComObject(capGraph); capGraph = null;

                if (graphBuilder != null)
                    Marshal.ReleaseComObject(graphBuilder); graphBuilder =
null;

                if (capFilter != null)
                    Marshal.ReleaseComObject(capFilter); capFilter = null;

                if (capDevices != null)
                {
                    foreach (DsDevice d in capDevices)
                        d.Dispose();
                    capDevices = null;
                }
            }
            catch (Exception)
            { }
        }

        // Resize preview video window to fill client area.
        void ResizeVideoWindow()
        {
            if (videoWin != null)
            {
                Rectangle rc = new Rectangle(0, 0, iWidth / 5, iHeight / 5);
                videoWin.SetWindowPosition(0, 0, rc.Right, rc.Bottom);
            }
        }

        // Override window fn to handle graph events.
        protected override void WndProc(ref Message m)
        {
            if (m.Msg == WM_GRAPHNOTIFY)
            {
                if (mediaEvt != null)
                    OnGraphNotify();
                return;
            }
            base.WndProc(ref m);
        }

        // Graph event (WM_GRAPHNOTIFY) handler.
        void OnGraphNotify()
        {
            DsEvCode code;
            int p1, p2, hr = 0;
            do
            {
                hr = mediaEvt.GetEvent(out code, out p1, out p2, 0);
                if (hr < 0)
                    break;
                hr = mediaEvt.FreeEventParams(code, p1, p2);
            }
            while (hr == 0);
        }
```

```
        private void GetAFrame()
        {
            int hr;
            if (savedArray == null)
            {
                int size = videoInfoHeader.BmiHeader.ImageSize;
                if ((size < 1000) || (size > 16000000))
                    return;
                savedArray = new byte[size + 64000];
            }

            captured = false;
            hr = sampGrabber.SetCallback(this, 1);
        }

        // Capture event, triggered by buffer callback.
        void OnCaptureDone()
        {
            GCHandle handle;
            try
            {
                int hr;
                if (sampGrabber == null)
                    return;
                hr = sampGrabber.SetCallback(null, 0);

                //return;
                int w = videoInfoHeader.BmiHeader.Width;
                int h = videoInfoHeader.BmiHeader.Height;
                if (((w & 0x03) != 0) || (w < 32) || (w > 4096) || (h < 32)
|| (h > 4096))
                    return;
                int stride = w * 3;

                handle = GCHandle.Alloc(savedArray, GCHandleType.Pinned);
                int scan0 = (int)handle.AddrOfPinnedObject();
                scan0 += (h - 1) * stride;

                if (DoneProcessing)
                    ProcessPixels = new Bitmap(w, h, -stride,
PixelFormat.Format24bppRgb, (IntPtr)scan0);

                handle.Free();
                savedArray = null;
            }
            catch (Exception ee)
            {
                savedArray = null;
                MessageBox.Show(this, "Could not grab picture\r\n" +
ee.Message, "Webcam Signature", MessageBoxButtons.OK, MessageBoxIcon.Stop);
            }
        }

        // Sample callback, NOT USED.
        int ISampleGrabberCB.SampleCB(double SampleTime, IMediaSample
pSample)
```



```
        {
            return 0;
        }

        // Buffer callback, COULD BE FROM FOREIGN THREAD.
        int ISampleGrabberCB.BufferCB(double SampleTime, IntPtr pBuffer, int
BufferLen)
        {
            if (captured || (savedArray == null))
            {
                return 0;
            }

            captured = true;
            bufferedSize = BufferLen;
            if ((pBuffer != IntPtr.Zero) && (BufferLen > 1000) && (BufferLen
<= savedArray.Length))
                Marshal.Copy(pBuffer, savedArray, 0, BufferLen);
            this.BeginInvoke(new CaptureDone(this.OnCaptureDone));
            return 0;
        }
        #endregion

        #region Initialize
        //Region 4 - Initialize: In this region we will call most of the main
controls to launch and initialize to start(e.g. clearing undo and redo,
setting RGB to default or from the UserSettings Custom Control)

        public MainForm()
        {
            InitializeComponent();

            // Currently, Webcam Signature is a fixed size Window
            // Webcam Signature will allow resizing
            //this.Size = new Size(iWidth + splitContainer1.Panel1.Width +
SystemInformation.FrameBorderSize.Width,
            //    iHeight + SystemInformation.FrameBorderSize.Height * 1 +
SystemInformation.CaptionHeight);
            this.Size = new Size(800, 600);
            splitContainer1.SplitterDistance = 150;
            //Signature.Width = iWidth;
            //Signature.Height = iHeight;
            //DrawingWidth = SystemInformation.PrimaryMonitorSize.Width;
            //DrawingHeight = SystemInformation.PrimaryMonitorSize.Height;
            DrawingWidth = 800;
            DrawingHeight = 600;
            Signature.Width = DrawingWidth;
            Signature.Height = DrawingHeight;

            // Load the color of signature and background from the
UserSettings
            ColorConverter colconv = new ColorConverter();
            SignatureColorSystem =
(Color)colconv.ConvertFromString(UserSettings.Default.LineColor);
            SignatureColor.blue = (byte)SignatureColorSystem.B;
            SignatureColor.green = (byte)SignatureColorSystem.G;
            SignatureColor.red = (byte)SignatureColorSystem.R;
```



```
            backColorSystem =
(Color)colconv.ConvertFromString(UserSettings.Default.BackColor);
            backColor.blue = (byte)backColorSystem.B;
            backColor.green = (byte)backColorSystem.G;
            backColor.red = (byte)backColorSystem.R;

            // Create the special color selection buttons
            PenColorButton.Size = new Size(57, 25);
            PenColorButton.Location = new Point(8, 15);
            PenColorButton.Click += new EventHandler(PenColorButton_Click);
            PenColorButton.BackColor = SystemColors.Control;
            PenColorButton.CenterColor = SignatureColorSystem;
            PenColorButton.FlatStyle = FlatStyle.Popup;
            groupBoxPen.Controls.Add(PenColorButton);

            BackgroundColorButton.Size = new Size(57, 25);
            BackgroundColorButton.Location = new Point(8, 15);
            BackgroundColorButton.Click += new
EventHandler(BackgroundColorButton_Click);
            BackgroundColorButton.BackColor = SystemColors.Control;
            BackgroundColorButton.CenterColor = backColorSystem;
            BackgroundColorButton.FlatStyle = FlatStyle.Popup;
            groupBoxBackground.Controls.Add(BackgroundColorButton);

            if (UserSettings.Default.TransparentOnSave)
                MakeBackTransparent.Checked = true;
            else
                MakeBackTransparent.Checked = false;

            WebCamBitmap = null;
            Signature.BringToFront();
            ShowPointLocation.BringToFront();
            SignaturePixels = new System.Drawing.Bitmap(DrawingWidth,
DrawingHeight);
            InitiatilizeSignaturePixels(ref SignaturePixels);

            ProcessPixels = new System.Drawing.Bitmap(iWidth, iHeight);

            Signature.Image = SignaturePixels;

            LoadTrackColor();
            UpdateTrackColorSample();

            SignaturePoints.Clear();
            SignaturePoints.Add(null);
            Undo.Clear();
            Redo.Clear();
            Undo.Add(SignaturePoints);

            // Tooltips are used sparingly
            toolTipTransparentOnSave.SetToolTip(this.MakeBackTransparent,
                "Transparent backgrounds are NOT compatible with MS
Paint/Photoshop.\n"+
                "Works with GIMP/MS Office 2007/PDF's.\n" + "The clipboard
does not support transparent backgrounds.");
            HelpAboutToolTip.SetToolTip(this.HelpAbout, "Help & About");
```

```
            // A timer is used to control the analysis of the video from the
webcam
            timer1.Interval = (int)(1.0 / (double)FrameRate * 1000.0);
            timer1.Enabled = true;
            timer1.Start();

            isKeyDown = false;
            DoneProcessing = true;
            FormActivated = true;
            FineTuneTC = null;

            PenThickness = UserSettings.Default.LineThickness;
            PenSizeDropBox.Text =
UserSettings.Default.LineThickness.ToString();

            ShowPointLocation.Location = new Point(-100, -100);
            InitializeBackgroundAverage();
            RefreshBackFilteringParams();

            WebcamMouseMode = false;
            WebcamMouseCB.Checked = false;
        }

        private void InitializeBackgroundAverage()
        {
            BackgroundAverage = new double[iWidth, iHeight, 3];
            for (int i = 0; i < iWidth; i++)
            {
                for (int j = 0; j < iHeight; j++)
                {
                    for (int k = 0; k < 3; k++)
                    {
                        BackgroundAverage[i, j, k] = 0.0;
                    }
                }
            }
        }

        private void MainForm_Shown(object sender, EventArgs e)
        {
            //if (UserSettings.Default.ActivateWebcam)
            //{
                InitializeCamera();
            //}
        }

        private void MainForm_FormClosing(object sender, FormClosingEventArgs
e)
        {
            CloseInterfaces();
        }

        // Show the colors that the program is tracking
        public void UpdateTrackColorSample()
        {
            if (UserSettings.Default.TrackRed)
```



```
            {
                this.TrackRed.Checked = true;
                this.TrackGreen.Checked = false;
                this.TrackBlue.Checked = false;
                SampleTrackColor.BackColor = Color.FromArgb(RedH, GreenH,
BlueH);
                SampleTrackColorLow.BackColor = Color.FromArgb(RedL, GreenL,
BlueL);
            }
            else if (UserSettings.Default.TrackGreen)
            {
                this.TrackRed.Checked = false;
                this.TrackGreen.Checked = true;
                this.TrackBlue.Checked = false;
                SampleTrackColor.BackColor = Color.FromArgb(RedH, GreenH,
BlueH);
                SampleTrackColorLow.BackColor = Color.FromArgb(RedL, GreenL,
BlueL);
            }
            else
            {
                this.TrackRed.Checked = false;
                this.TrackGreen.Checked = false;
                this.TrackBlue.Checked = true;
                SampleTrackColor.BackColor = Color.FromArgb(RedH, GreenH,
BlueH);
                SampleTrackColorLow.BackColor = Color.FromArgb(RedL, GreenL,
BlueL);
            }
        }

        // Get the colors to track from UserSettings
        public void LoadTrackColor()
        {
            if (UserSettings.Default.TrackRed)
            {
                RedL = UserSettings.Default.TRLowRed;
                RedH = UserSettings.Default.TRHighRed;
                GreenL = UserSettings.Default.TRLowGreen;
                GreenH = UserSettings.Default.TRHighGreen;
                BlueL = UserSettings.Default.TRLowBlue;
                BlueH = UserSettings.Default.TRHighBlue;
            }
            else if (UserSettings.Default.TrackGreen)
            {
                RedL = UserSettings.Default.TGLowRed;
                RedH = UserSettings.Default.TGHighRed;
                GreenL = UserSettings.Default.TGLowGreen;
                GreenH = UserSettings.Default.TGHighGreen;
                BlueL = UserSettings.Default.TGLowBlue;
                BlueH = UserSettings.Default.TGHighBlue;
            }
            else
            {
                RedL = UserSettings.Default.TBLowRed;
                RedH = UserSettings.Default.TBHighRed;
                GreenL = UserSettings.Default.TBLowGreen;
```

```
                GreenH = UserSettings.Default.TBHighGreen;
                BlueL = UserSettings.Default.TBLowBlue;
                BlueH = UserSettings.Default.TBHighBlue;
            }
        }

        #endregion

        #region Core Code
        //Region 4 - Core Code: Contains the functions that will allow us to
actually draw and redraw pixels, points, lines
        //and ofcourse the most important function GetPoint that takes a
color bitmap image and filters out everything but the given RBG color range.

        // timer1 controls when the program draws pixels to the screen
        private void timer1_Tick(object sender, EventArgs e)
        {
            if (DoneProcessing && FormActivated)
                DrawPixels();
        }

        // This is the main function that coordinates the drawing of the
Signature pixels
        // It also filters the input to prevent jittering of the image.
        // This is reason why the signatures come out smooth.
        private void DrawPixels()
        {
            DoneProcessing = false;
            try
            {
                Point pt = new Point(0, 0);
                Point ScaledPoint = new Point(0, 0);

                if (ProcessPixels != null)
                {
                    GetAFrame();

                    if (GetPoint(ref pt, ref ProcessPixels, RedL, RedH,
GreenL, GreenH, BlueL, BlueH) == 1)
                    {
                        if (WebcamMouseMode == false)
                        {
                            ScaledPoint.X = pt.X - (int)((double)iWidth * 0.1
/ 2.0);
                            ScaledPoint.Y = pt.Y - (int)((double)iHeight *
0.1 / 2.0);
                            //ScaledPoint = TranslateWebcamPTtoScreen(iWidth,
iHeight,
                            //    (int)((double)iWidth * 0.9),
(int)((double)iHeight * 0.9), ScaledPoint);
                            ScaledPoint =
TranslateWebcamPTtoScreen(SystemInformation.PrimaryMonitorSize.Width,
SystemInformation.PrimaryMonitorSize.Height,
                                (int)((double)iWidth * 0.9),
(int)((double)iHeight * 0.9), ScaledPoint);
```

```
                            if (isKeyDown && !(SignaturePoints.Count > 0 &&
SignaturePoints[SignaturePoints.Count - 1] != null && ScaledPoint ==
(Point)SignaturePoints[SignaturePoints.Count - 1]))
                            {
                                SignaturePoints.Add(ScaledPoint);

                                // Do a rough representation of the signature
while the user draws it
                                if (SignaturePoints.Count >= 2 &&
SignaturePoints[SignaturePoints.Count - 2] != null &&
                                    SignaturePoints[SignaturePoints.Count -
1] != null)
                                {

DrawLine((Point)SignaturePoints[SignaturePoints.Count - 2],
(Point)SignaturePoints[SignaturePoints.Count - 1],
                                        ref SignaturePixels, true);
                                }
                                else if (SignaturePoints.Count >= 1 &&
SignaturePoints[SignaturePoints.Count - 1] != null)
                                {
                                    DrawAPoint(ScaledPoint, ref
SignaturePixels, true);
                                }
                            }

                            ShowPointLocation.Location = new
Point(ScaledPoint.X, ScaledPoint.Y);
                        }
                        else
                        {
                            ScaledPoint.X = pt.X - (int)((double)iWidth * 0.1
/ 2.0);
                            ScaledPoint.Y = pt.Y - (int)((double)iHeight *
0.1 / 2.0);
                            Cursor.Position =
TranslateWebcamPTtoScreen(SystemInformation.PrimaryMonitorSize.Width,
SystemInformation.PrimaryMonitorSize.Height,
                                (int)((double)iWidth * 0.9),
(int)((double)iHeight * 0.9), ScaledPoint);
                        }
                    }
                }

            }
            catch (Exception)
            {
                // Do nothing
            }

            ProcessPixels = null;
            GC.Collect();
            DoneProcessing = true;
        }

        // Set the background color of the Signature
        private void InitiatilizeSignaturePixels(ref Bitmap NewBitmap)
```



```
        {
            UnsafeBitmap WebCamBitmap = new UnsafeBitmap(ref NewBitmap,
true);
            WebCamBitmap.LockBitmap();

            for (int i = 0; i < WebCamBitmap.Bitmap.Width; i++)
            {
                for (int j = 0; j < WebCamBitmap.Bitmap.Height; j++)
                {
                    WebCamBitmap.SetPixel(i, j, backColor);
                }
            }
            WebCamBitmap.UnlockBitmap();
        }

        // Draw a circular point with a radius of PenThickness / 2
        public void DrawAPoint(Point pt, ref Bitmap image, bool
RefreshRegion)
        {
            UnsafeBitmap WebCamBitmap = new UnsafeBitmap(ref image, true);
            WebCamBitmap.LockBitmap();

            int left = pt.X - (int)(PenThickness / (float)2.0) + 1;
            if (left < 0)
                left = 0;

            int top = pt.Y - (int)(PenThickness / (float)2.0) + 1;
            if (top < 0)
                top = 0;

            int right = pt.X + (int)(PenThickness / (float)2.0) - 1;
            if (right > image.Width)
                right = image.Width;

            int bottom = pt.Y + (int)(PenThickness / (float)2.0) - 1;
            if (bottom > image.Height)
                bottom = image.Height;

            double radius = (double)(bottom - top) / 2.0;

            for (int i = left; i < right; i++)
            {
                for (int j = top; j < bottom; j++)
                {
                    if (Math.Sqrt(Math.Pow(Math.Abs(i - pt.X), 2.0) +
Math.Pow(Math.Abs(j - pt.Y), 2.0)) <= radius)
                    {
                        WebCamBitmap.SetPixel(i, j, SignatureColor);
                    }
                }
            }
            WebCamBitmap.UnlockBitmap();

            // Redraw only the region necessary to increase speed
            if (RefreshRegion)
            {
                Rectangle tempRect = new Rectangle(left, top, right, bottom);
```

```
                Region r = new Region(tempRect);
                Signature.Invalidate(r);
                Signature.Update();
            }
        }

        // Draws a line with a start and end point on a given image.
        private void DrawLine(Point start, Point end, ref Bitmap image, bool
RefreshRegion)
        {
            if (Math.Sqrt(Math.Pow(Math.Abs(start.X - end.X), 2.0) +
Math.Pow(Math.Abs(start.Y - end.Y), 2.0)) <= (double)PenThickness)
            {
                Point middle = new Point((start.X + end.X) / 2, (start.Y +
end.Y) / 2);
                DrawAPoint(middle, ref image, RefreshRegion);
            }
            else
            {
                Graphics g = null;
                g = Graphics.FromImage((Image)image);
                Pen pen = new Pen(SignatureColorSystem, PenThickness);
                pen.MiterLimit = 0.0f;
                pen.StartCap = System.Drawing.Drawing2D.LineCap.Round;
                pen.EndCap = System.Drawing.Drawing2D.LineCap.Round;
                g.DrawLine(pen,start.X, start.Y, end.X, end.Y);
                pen.Dispose();
                g.Dispose();

                // Refresh only the region drawn to speed up the program
                // instead of using Signature.Refresh() to update the whole
image
                if (RefreshRegion)
                {
                    int tempLeft, tempTop, tempRight, tempBottom;
                    if (start.X < end.X)
                    {
                        tempLeft = start.X;
                        tempRight = end.X;
                    }
                    else
                    {
                        tempLeft = end.X;
                        tempRight = start.X;
                    }

                    if (start.Y < end.Y)
                    {
                        tempTop = start.Y;
                        tempBottom = end.Y;
                    }
                    else
                    {
                        tempTop = end.Y;
                        tempBottom = start.Y;
                    }
```

```
                    Rectangle tempRect = new Rectangle(tempLeft, tempTop,
tempRight, tempBottom);
                    Region r = new Region(tempRect);
                    Signature.Invalidate(r);
                    Signature.Update();
                }
            }
        }



        // Called by ApplySmoothing_MouseClick to redraw the signature area
        private void RedrawPixels(ref Bitmap TargetBitmap)
        {
            int lastIndex = 0;
            while (lastIndex < SignaturePoints.Count - 1)
            {
                int null1 = SignaturePoints.IndexOf(null, lastIndex);
                if (null1 < 0 || null1 == SignaturePoints.Count - 1)
                {
                    break;
                }

                int null2 = SignaturePoints.IndexOf(null, null1 + 1);
                lastIndex = null2;
                if (null2 < 0)
                {
                    break;
                }

                int curvePointsSize = null2 - null1 - 1;

                if (curvePointsSize >= 3)
                {
                    Point[] curvePoints;
                    curvePoints = new Point[curvePointsSize];

                    int tempCounter = 0;
                    for (int i = null1 + 1; i < null2; i++)
                    {
                        curvePoints[tempCounter] = (Point)SignaturePoints[i];
                        tempCounter++;
                    }

                    ApplySmoothingFunction(curvePoints, ref TargetBitmap);
                }
                else if (curvePointsSize == 2)
                {
                    DrawLine((Point)SignaturePoints[null1 + 1],
(Point)SignaturePoints[null1 + 2], ref TargetBitmap, false);
                }
                else if (curvePointsSize == 1)
                {
                    DrawAPoint((Point)SignaturePoints[null1 + 1], ref
TargetBitmap, false);
                }
```

```
            }
            Signature.Refresh();
        }

        // ApplySmoothingFunction creates smooth cardinal curves for an array
of points
        private void ApplySmoothingFunction(Point[] curvePoints, ref Image
image)
        {
            Graphics g = null;
            g = Graphics.FromImage(image);
            Pen pen = new Pen(SignatureColorSystem, PenThickness);
            pen.MiterLimit = 0.0f;
            pen.StartCap = System.Drawing.Drawing2D.LineCap.Round;
            pen.EndCap = System.Drawing.Drawing2D.LineCap.Round;
            g.DrawCurve(pen, curvePoints, (float)0.5);

            pen.Dispose();
            g.Dispose();
        }

        private void ApplySmoothingFunction(Point[] curvePoints, ref Bitmap
image)
        {
            Graphics g = null;
            g = Graphics.FromImage((Image)image);
            Pen pen = new Pen(SignatureColorSystem, PenThickness);
            pen.MiterLimit = 0.0f;
            pen.StartCap = System.Drawing.Drawing2D.LineCap.Round;
            pen.EndCap = System.Drawing.Drawing2D.LineCap.Round;
            g.DrawCurve(pen, curvePoints, (float)0.5);

            pen.Dispose();
            g.Dispose();
        }

        // MovingAverage is not currently used
        private void MovingAverage(ref Point[] curvePoints)
        {
            long sumX = 0;
            long sumY = 0;
            long numPoints = 0;

            for (int i = 0; i < curvePoints.Length; i++)
            {
                Point pt = curvePoints[i];
                sumX = sumX + (long)pt.X;
                sumY = sumY + (long)pt.Y;
                numPoints++;

                pt.X = (int)((double)sumX / (double)numPoints);
                pt.Y = (int)((double)sumY / (double)numPoints);

                curvePoints[i] = pt;
            }
        }
```

```
        // GetPoint takes a color bitmap image and filters out everything but
the given RBG color range.
        // It returns 1 if it finds a point, 0 other wise.
        // The point it finds is placed in CurrentPoint.
        private int GetPoint(ref Point CurrentPoint, ref Bitmap RawPixels,
int RedLow, int RedHigh, int GreenLow, int GreenHigh, int BlueLow, int
BlueHigh)
        {
            int SumX = 0;
            int SumY = 0;
            int NumPoints = 0;
            int neighbors = 0;

            // The UnsafeBitmap greatly speeds up the processing time.
            WebCamBitmap = new UnsafeBitmap(RawPixels);
            WebCamBitmap.LockBitmap();

            // We don't actually look at every pixel because that consumes
too much processing time
            // We look at every other one
            for (int i = 0; i < WebCamBitmap.Bitmap.Width; i += 2)
            {
                for (int j = 0; j < WebCamBitmap.Bitmap.Height; j += 2)
                {
                    PixelData pixel = WebCamBitmap.GetPixel(i, j);

                    // Update the background averages
                    // Can be used for static background filtering or
                    // it can be used for dynamic background filtering
                    // where it will constantly update itself
                    if (NoBackgroundFilter == false)
                    {
                        if (NumBackgroundFramesCaptured <
RequiredNumBackgroundFrames)
                        {
                            if (NumBackgroundFramesCaptured == 0)
                            {
                                //red
                                BackgroundAverage[i, j, 0] =
(double)pixel.red;
                                //green
                                BackgroundAverage[i, j, 1] =
(double)pixel.green;
                                //blue
                                BackgroundAverage[i, j, 2] =
(double)pixel.blue;
                            }
                            else
                            {
                                //red
                                BackgroundAverage[i, j, 0] =
(((double)BackgroundAverage[i, j, 0] * (double)(NumBackgroundFramesCaptured)
+ (double)pixel.red)
                                    / (double)(NumBackgroundFramesCaptured +
1));
                                //green
```

```
                                BackgroundAverage[i, j, 1] =
(((double)BackgroundAverage[i, j, 1] * (double)(NumBackgroundFramesCaptured)
+ (double)pixel.green)
                                    / (double)(NumBackgroundFramesCaptured +
1));
                                //blue
                                BackgroundAverage[i, j, 2] =
(((double)BackgroundAverage[i, j, 2] * (double)(NumBackgroundFramesCaptured)
+ (double)pixel.blue)
                                    / (double)(NumBackgroundFramesCaptured +
1));
                            }
                        }
                        else if (UseBackgroundDynamicFilter == true)
                        {
                            //red
                            BackgroundAverage[i, j, 0] =
(((double)BackgroundAverage[i, j, 0] * (double)(NumBackgroundFramesCaptured)
+ (double)pixel.red)
                                / (double)(NumBackgroundFramesCaptured + 1));
                            //green
                            BackgroundAverage[i, j, 1] =
(((double)BackgroundAverage[i, j, 1] * (double)(NumBackgroundFramesCaptured)
+ (double)pixel.green)
                                / (double)(NumBackgroundFramesCaptured + 1));
                            //blue
                            BackgroundAverage[i, j, 2] =
(((double)BackgroundAverage[i, j, 2] * (double)(NumBackgroundFramesCaptured)
+ (double)pixel.blue)
                                / (double)(NumBackgroundFramesCaptured + 1));
                        }
                    }


                    if ((Math.Abs(BackgroundAverage[i, j, 0] -
(int)pixel.red) > BackgroundDiffThreshold ||
                        Math.Abs(BackgroundAverage[i, j, 1] -
(int)pixel.green) > BackgroundDiffThreshold ||
                        Math.Abs(BackgroundAverage[i, j, 2] -
(int)pixel.blue) > BackgroundDiffThreshold || NoBackgroundFilter) &&
                        pixel.red >= (byte)RedLow && pixel.red <=
(byte)RedHigh && pixel.green >= (byte)GreenLow &&
                        pixel.green <= (byte)GreenHigh && pixel.blue >=
(byte)BlueLow && pixel.blue <= (byte)BlueHigh)
                    {
                        // We now know that the pixel is a valid color
                        // We now need to check if there is a neighboring
pixel of a valid color
                        // Purpose is to filter out errant pixels
                        // Right now, it is coded to look in a region of 100
pixels for
                        // each valid pixel found to see if there is a
neighbor
                        int ktemp = i - 4;
                        if (ktemp < 0)
                            ktemp = 0;
```

```
                        int khigh = i + 4;
                        if (khigh > WebCamBitmap.Bitmap.Width - 1)
                        {
                            khigh = WebCamBitmap.Bitmap.Width - 1;
                        }

                        int ltemp = j - 4;
                        if (ltemp < 0)
                            ltemp = 0;

                        int lhigh = j + 4;
                        if (lhigh > WebCamBitmap.Bitmap.Height - 1)
                        {
                            lhigh = WebCamBitmap.Bitmap.Height - 1;
                        }

                        neighbors = 0;

                        for (int k = ktemp; k <= khigh; k++)
                        {
                            for (int l = ltemp; l <= lhigh; l++)
                            {
                                PixelData NeighborPixel =
WebCamBitmap.GetPixel(k, l);
                                if (NeighborPixel.red >= (byte)RedLow &&
NeighborPixel.red <= (byte)RedHigh && NeighborPixel.green >= (byte)GreenLow
&&
                                    NeighborPixel.green <= (byte)GreenHigh &&
NeighborPixel.blue >= (byte)BlueLow && NeighborPixel.blue <= (byte)BlueHigh)
                                {
                                    neighbors++;
                                }
                            }
                            if (neighbors >= 1)
                                break;
                        }

                        if (neighbors >= 1)
                        {
                            NumPoints++;
                            SumX += WebCamBitmap.Bitmap.Width - i;
                            SumY += j;
                        }
                    }
                }
            }
            WebCamBitmap.UnlockBitmap();
            WebCamBitmap.Dispose();

            if (NoBackgroundFilter == false)
            {
                if (NumBackgroundFramesCaptured <
RequiredNumBackgroundFrames)
                {
                    NumBackgroundFramesCaptured++;
                    if (NumBackgroundFramesCaptured >=
NumBackgroundFramesRequiredCalibration)
```

```
                        NumBackgroundFramesCaptured =
RequiredNumBackgroundFrames;
                }
            }

            if (NumPoints >= 1)
            {
                // The point we return is simply the average of the points
seen
                // No need to over complicate this by calculating the
centroid
                CurrentPoint = new Point(SumX / NumPoints, SumY / NumPoints);
                return 1;
            }
            else
            {
                return 0;
            }
        }

        private Point TranslateWebcamPTtoScreen(int ScreenWidth, int
ScreenHeight, int WebcamWidth, int WebcamHeight, Point CurrPoint)
        {
            Point ScreenPoint = new Point();

            ScreenPoint.X = (int)((double)CurrPoint.X / (double)WebcamWidth *
(double)ScreenWidth);
            ScreenPoint.Y = (int)((double)CurrPoint.Y / (double)WebcamHeight
* (double)ScreenHeight);

            if (ScreenPoint.X < 0)
                ScreenPoint.X = 0;
            if (ScreenPoint.X > ScreenWidth)
                ScreenPoint.X = ScreenWidth;
            if (ScreenPoint.Y < 0)
                ScreenPoint.Y = 0;
            if (ScreenPoint.Y > ScreenHeight)
                ScreenPoint.Y = ScreenHeight;

            //Console.WriteLine(CurrPoint + "  " + ScreenPoint);
            return ScreenPoint;
        }

        public void RefreshBackFilteringParams()
        {
            RequiredNumBackgroundFrames =
(int)(UserSettings.Default.BackgroundRefreshTime * (double)FrameRate);
            NumBackgroundFramesCaptured = 0;
            BackgroundDiffThreshold =
UserSettings.Default.BackgroundDiffThreshold;
            BackgroundCalibrateTime =
UserSettings.Default.BackgroundCalibrateTime;
            NumBackgroundFramesRequiredCalibration =
(int)(UserSettings.Default.BackgroundCalibrateTime * (double)FrameRate);
            NoBackgroundFilter = UserSettings.Default.NoBackgroundFilter;
            UseBackgroundStaticFilter =
UserSettings.Default.UseStaticBackFilter;
```

```
            UseBackgroundDynamicFilter =
UserSettings.Default.UseDynamicBackFilter;
            //InitializeBackgroundAverage();
        }

        // Creates a Bitmap from the array of values for the background
average
        public Bitmap GetStaticBackBitmap()
        {
            Bitmap StaticBackground = new Bitmap(iWidth, iHeight);

            if (mediaCtrl != null)
            {
                int state1, state2, state3;
                state2 = 500;
                state1 = mediaCtrl.GetState(state2, out state3);

                if (state3 == 2)
                {
                    UnsafeBitmap WebCamBitmap = new UnsafeBitmap(ref
StaticBackground, true);
                    WebCamBitmap.LockBitmap();

                    for (int i = 0; i < WebCamBitmap.Bitmap.Width; i += 2)
                    {
                        for (int j = 0; j < WebCamBitmap.Bitmap.Height; j +=
2)
                        {
                            PixelData currPixel = new PixelData();
                            currPixel.red = (byte)BackgroundAverage[i, j, 0];
                            currPixel.green = (byte)BackgroundAverage[i, j,
1];
                            currPixel.blue = (byte)BackgroundAverage[i, j,
2];
                            WebCamBitmap.SetPixel(i, j, currPixel);
                        }
                    }
                    WebCamBitmap.UnlockBitmap();
                }
            }

            GC.Collect();
            return StaticBackground;
        }

        // Returns a full frame captured from the webcam
        // This is used to pass a frame to FineTuneTrackColorsForm
        public Bitmap GetFullFrameBitmap()
        {
            Bitmap ReturnBitmap = new Bitmap(iWidth, iHeight);

            try
            {
                if (mediaCtrl != null)
                {
                    int state1, state2, state3;
                    state2 = 500;
```



```
                    state1 = mediaCtrl.GetState(state2, out state3);

                    if (state3 == 2)
                    {
                        DoneProcessing = false;
                        try
                        {
                            if (ProcessPixels != null)
                            {
                                GetAFrame();
                            }
                        }
                        catch (Exception)
                        {
                        }

                        //UnsafeBitmap WebCamBitmap = new
UnsafeBitmap(ProcessPixels);
                        //WebCamBitmap.LockBitmap();
                        //UnsafeBitmap UnsafeReturnBitmap = new
UnsafeBitmap(ref ReturnBitmap, true);
                        //UnsafeReturnBitmap.LockBitmap();

                        //for (int i = 0; i < WebCamBitmap.Bitmap.Width; i++)
                        //{
                        //    for (int j = 0; j < WebCamBitmap.Bitmap.Height;
j++)
                        //    {
                        //        UnsafeReturnBitmap.SetPixel(i, j,
WebCamBitmap.GetPixel(i, j));
                        //    }
                        //}
                        //WebCamBitmap.UnlockBitmap();
                        //UnsafeReturnBitmap.UnlockBitmap();
                        if (ProcessPixels != null)
                            ReturnBitmap = new Bitmap(ProcessPixels);

                        ProcessPixels = null;
                        DoneProcessing = true;
                    }
                }
            }
            catch (Exception ee)
            {
                Console.WriteLine("Error render frame. " + ee);
            }

            GC.Collect();
            return ReturnBitmap;
        }

        public int RefreshingBackgroundFilterProgress()
        {
            if (mediaCtrl != null)
            {
                int state1, state2, state3;
                state2 = 500;
```

```
                state1 = mediaCtrl.GetState(state2, out state3);

                if (state3 == 2)
                {
                    if (NoBackgroundFilter == false)
                    {
                        if (NumBackgroundFramesCaptured <
RequiredNumBackgroundFrames)
                        {
                            return (int)((double)NumBackgroundFramesCaptured
/ (double)RequiredNumBackgroundFrames * (100.0));
                        }
                        else
                        {
                            return 100;
                        }
                    }
                    else
                    {
                        return 100;
                    }
                }
                else
                {
                    return 100;
                }
            }
            else
            {
                return 100;
            }
        }

        #endregion

        #region Change Tracking Colors
        //Region 5 - Change Tracking Colors: We will choose here the color
from one of the three radio buttons shown at the left of the splitContainer.

        private void TrackRed_CheckedChanged(object sender, EventArgs e)
        {
            if (this.TrackRed.Checked == true)
            {
                UserSettings.Default.TrackRed = true;
                UserSettings.Default.TrackGreen = false;
                UserSettings.Default.TrackBlue = false;
                UserSettings.Default.Save();
                LoadTrackColor();
                UpdateTrackColorSample();

                if (FineTuneTC != null)
                {
                    FineTuneTC.LoadSettings();
                }
            }
        }
```

```
        private void TrackGreen_CheckedChanged(object sender, EventArgs e)
        {
            if (this.TrackGreen.Checked == true)
            {
                UserSettings.Default.TrackRed = false;
                UserSettings.Default.TrackGreen = true;
                UserSettings.Default.TrackBlue = false;
                UserSettings.Default.Save();
                LoadTrackColor();
                UpdateTrackColorSample();

                if (FineTuneTC != null)
                {
                    FineTuneTC.LoadSettings();
                }
            }
        }

        private void TrackBlue_CheckedChanged(object sender, EventArgs e)
        {
            if (this.TrackBlue.Checked == true)
            {
                UserSettings.Default.TrackRed = false;
                UserSettings.Default.TrackGreen = false;
                UserSettings.Default.TrackBlue = true;
                UserSettings.Default.Save();
                LoadTrackColor();
                UpdateTrackColorSample();

                if (FineTuneTC != null)
                {
                    FineTuneTC.LoadSettings();
                }
            }
        }

        private void FineTuneTrackColors_MouseClick(object sender,
MouseEventArgs e)
        {
            if (FineTuneTC == null)
            {
                FineTuneTC = new FineTuneTrackColorsForm(this);
                FormActivated = true;
            }
            else
            {
                FineTuneTC.Show();
                FineTuneTC.BringToFront();
            }
        }

        #endregion

        #region File Processing
        //Region 6 - File Processing: All the functions and buttons that
allow us to wheter Save, Load or Copy the image to our local hard drive or to
another image browsing software goes in this region.
```



```
        private void CopytoClipboard_MouseClick(object sender, MouseEventArgs
e)
        {
            Clipboard.SetImage((Image)SignaturePixels);
        }

        // Save the signature to an image file
        private void saveFileDialog1_FileOk(object sender, CancelEventArgs e)
        {
            Image tempImage;

            if (UserSettings.Default.TransparentOnSave == true)
            {
                Bitmap transparentVersion = new
Bitmap((Image)SignaturePixels);
                for (int i = 0; i < transparentVersion.Width; i++)
                {
                    for (int j = 0; j < transparentVersion.Height; j++)
                    {
                        transparentVersion.SetPixel(i, j,
SignatureColorSystem);
                    }
                }

                transparentVersion.MakeTransparent(SignatureColorSystem);
                RedrawPixels(ref transparentVersion);
                tempImage = (Image)transparentVersion;
                tempImage.Save(saveFileDialog1.FileName);
            }
            else
            {
                tempImage = (Image)SignaturePixels;
                tempImage.Save(saveFileDialog1.FileName);
            }
        }

        private void SavetoFile_MouseClick(object sender, MouseEventArgs e)
        {
            saveFileDialog1.ShowDialog();
        }

        // Save the signature data points to a .wsd file
        private void SaveDataPoints_MouseClick(object sender, MouseEventArgs
e)
        {
            SaveDataPointsDialog.ShowDialog();
        }

        private void SaveDataPointsDialog_FileOk(object sender,
CancelEventArgs e)
        {
            try
            {
                TextWriter tw = new
StreamWriter(SaveDataPointsDialog.FileName);
```

```
                tw.WriteLine("[Signature Points Start]");
                for (int i = 0; i <= SignaturePoints.Count - 1; i++)
                {
                    if (SignaturePoints[i] == null)
                        tw.WriteLine("null");
                    else
                        tw.WriteLine(SignaturePoints[i].ToString());
                }
                tw.WriteLine("[Signature Points End]");
                tw.Close();
            }
            catch (Exception ee)
            {
                MessageBox.Show(this, "Could not save to file. File access
error.", "Webcam Signature", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        // Load signature data points from a .wsd file
        private void LoadDataPoints_MouseClick(object sender, MouseEventArgs
e)
        {
            OpenDataPointsDialog.ShowDialog();
        }

        private void OpenDataPointsDialog_FileOk(object sender,
CancelEventArgs e)
        {
            try
            {
                ArrayList SigDataPoints = new ArrayList();
                TextReader tr = new
StreamReader(OpenDataPointsDialog.FileName);
                while (true)
                {
                    String dataline = tr.ReadLine();
                    if (dataline == null)
                    {
                        break;
                    }
                    else if (dataline == "[Signature Points Start]")
                    {
                        String SigPoint = tr.ReadLine();
                        if (SigPoint == null)
                            break;

                        while (SigPoint != "[Signature Points End]")
                        {
                            if (SigPoint == "null")
                            {
                                SigDataPoints.Add(null);
                                SigPoint = tr.ReadLine();
                            }
                            else
                            {
                                Point pt = new Point(0, 0);
```

```
                                pt.X =
int.Parse(SigPoint.Substring(SigPoint.IndexOf("X=") + 2,
SigPoint.IndexOf(",") - (SigPoint.IndexOf("X=") + 2)));
                                pt.Y =
int.Parse(SigPoint.Substring(SigPoint.IndexOf("Y=") + 2,
SigPoint.IndexOf("}") - (SigPoint.IndexOf("Y=") + 2)));
                                SigDataPoints.Add(pt);
                                SigPoint = tr.ReadLine();
                            }
                        }
                    }
                }
                tr.Close();

                SignaturePoints.Clear();
                SignaturePoints = SigDataPoints;
                int begin = 1;

                while (begin < SigDataPoints.Count)
                {
                    ArrayList tempList2 = new ArrayList();
                    begin = SigDataPoints.IndexOf(null, begin);

                    if (begin < 0)
                        break;

                    for (int i = 0; i <= begin; i++)
                    {
                        tempList2.Add(SigDataPoints[i]);
                    }

                    SetUndo(tempList2);

                    begin += 1;
                }
                InitiatilizeSignaturePixels(ref SignaturePixels);
                RedrawPixels(ref SignaturePixels);
            }
            catch (Exception ee)
            {
                MessageBox.Show(this, "Could not open file / invalid file." +
ee, "Webcam Signature", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        #endregion

        #region Undo/Redo Functions
        //Region 7 - Undo/Redo Functions: Undo and Redo button functions

        private void buttonUndo_MouseClick(object sender, MouseEventArgs e)
        {
            if (Undo.Count > 1 && isKeyDown == false)
            {
                SetRedo(SignaturePoints);
                ArrayList tempUndoPoint = new
ArrayList((ArrayList)Undo[Undo.Count - 2]);
```

```
                if (tempUndoPoint.Count == 0)
                    tempUndoPoint.Add(null);
                SignaturePoints.Clear();
                SignaturePoints = tempUndoPoint;
                Undo.RemoveAt(Undo.Count - 1);
                InitiatilizeSignaturePixels(ref SignaturePixels);
                RedrawPixels(ref SignaturePixels);
                UpdateUndoRedoButtons();
            }
        }

        private void buttonRedo_MouseClick(object sender, MouseEventArgs e)
        {
            if (Redo.Count > 0 && isKeyDown == false)
            {
                ArrayList tempRedoPoint = new
ArrayList((ArrayList)Redo[Redo.Count - 1]);
                if (tempRedoPoint.Count == 0)
                    tempRedoPoint.Add(null);
                SignaturePoints.Clear();
                SignaturePoints = tempRedoPoint;
                SetUndo(SignaturePoints);
                Redo.RemoveAt(Redo.Count - 1);
                InitiatilizeSignaturePixels(ref SignaturePixels);
                RedrawPixels(ref SignaturePixels);
                UpdateUndoRedoButtons();
            }
        }

        private void UpdateUndoRedoButtons()
        {
            if (Undo.Count > 1)
                buttonUndo.Enabled = true;
            else
                buttonUndo.Enabled = false;

            if (Redo.Count > 0)
                buttonRedo.Enabled = true;
            else
                buttonRedo.Enabled = false;

            buttonUndo.Text = "<- " + (Undo.Count - 1);
            buttonRedo.Text = Redo.Count + " ->";
        }

        private void SetUndo(ArrayList DataPoints)
        {
            while (Undo.Count > 200)
            {
                Undo.RemoveAt(0);
            }
            ArrayList tempList = new ArrayList(DataPoints);
            Undo.Add(tempList);
            UpdateUndoRedoButtons();
        }

        private void SetRedo(ArrayList DataPoints)
```

```
        {
            while (Redo.Count > 200)
            {
                Redo.RemoveAt(0);
            }
            ArrayList tempList = new ArrayList(DataPoints);
            Redo.Add(tempList);
            UpdateUndoRedoButtons();
        }

        #endregion

        #region User Actions
        //Region 8 - User Actions: All the user actions like choosing the pen
color, the background color, pressing the shit, ctrl will trigger the
following functions

        private void MainForm_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode.ToString() != "Return" && PenSizeDropBox.Focused ==
false &&
                (e.KeyCode.ToString() == "ShiftKey" || e.KeyCode.ToString()
== "ControlKey"))
            {
                isKeyDown = true;
                e.Handled = true;
            }
        }

        private void MainForm_KeyUp(object sender, KeyEventArgs e)
        {
            if (e.KeyCode.ToString() != "Return" && PenSizeDropBox.Focused ==
false &&
                (e.KeyCode.ToString() == "ShiftKey" || e.KeyCode.ToString()
== "ControlKey"))
            {
                isKeyDown = false;
                if (SignaturePoints.Count > 0 &&
SignaturePoints[SignaturePoints.Count - 1] != null)
                {
                    SignaturePoints.Add(null);
                }
                e.Handled = true;

                //ApplySmoothing_MouseClick(this, null);

                // Create an undo point on each keyup
                SetUndo(SignaturePoints);
            }
        }

        // Clear the drawing but not the Undo/Redo points
        private void Clear_MouseClick(object sender, MouseEventArgs e)
        {
            SignaturePixels = new System.Drawing.Bitmap(DrawingWidth,
DrawingHeight);
            InitiatilizeSignaturePixels(ref SignaturePixels);
```



```
            SignaturePoints.Clear();
            SignaturePoints.Add(null);

            Signature.Image = SignaturePixels;

            SetUndo(SignaturePoints);
        }

        private void PenSizeDropBox_TextChanged(object sender, EventArgs e)
        {
            if (PenSizeDropBox.Text.Length > 0)
            {
                if (PenSizeDropBox.Text == "Hide")
                {
                    ShowPointLocation.Hide();
                    PenSizeDropBox.Text =
UserSettings.Default.LineThickness.ToString();

                    if (mediaCtrl != null)
                    {
                        int state1, state2, state3;
                        state2 = 500;
                        state3 = 0;
                        state1 = mediaCtrl.GetState(state2, out state3);

                        if (state3 == 2)
                        {
                            mediaCtrl.Stop();
                        }
                    }

                    UserSettings.Default.ActivateWebcam = false;
                    UserSettings.Default.Save();
                }
                else if (PenSizeDropBox.Text == "Show")
                {
                    ShowPointLocation.Show();
                    PenSizeDropBox.Text =
UserSettings.Default.LineThickness.ToString();
                    if (mediaCtrl != null)
                    {
                        int state1, state2, state3;
                        state2 = 500;
                        state3 = 0;
                        state1 = mediaCtrl.GetState(state2, out state3);

                        if (state3 != 2)
                        {
                            mediaCtrl.Run();
                        }
                    }
                    else
                    {
                        InitializeCamera();
                    }
```

```
                    UserSettings.Default.ActivateWebcam = true;
                    UserSettings.Default.Save();
                }
                else
                {
                    try
                    {
                        float SelectedThickness =
float.Parse(PenSizeDropBox.Text);
                        if (SelectedThickness > 0)
                        {
                            UserSettings.Default.LineThickness =
SelectedThickness;
                            UserSettings.Default.Save();
                            PenThickness = SelectedThickness;
                        }
                        InitiatilizeSignaturePixels(ref SignaturePixels);
                        RedrawPixels(ref SignaturePixels);
                    }
                    catch (Exception)
                    {
                        if (PenSizeDropBox.Text.Length >= 1)
                            PenSizeDropBox.Text =
PenSizeDropBox.Text.Substring(0, PenSizeDropBox.Text.Length - 1);
                    }
                }
            }
        }

        private void PenSizeDropBox_DropDownClosed(object sender, EventArgs
e)
        {
            Signature.Focus();
        }

        private void PenSizeDropBox_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Enter)
            {
                Signature.Focus();
                e.Handled = true;
            }
        }

        private void splitContainer1_Panel1_MouseClick(object sender,
MouseEventArgs e)
        {
            splitContainer1.Panel1.Focus();
        }

        private void Signature_MouseClick(object sender, MouseEventArgs e)
        {
            Signature.Focus();
        }

        void PenColorButton_Click(object sender, System.EventArgs e)
        {
```

```
            ColorButton callingButton = (ColorButton)sender;
            Point p = new Point(groupBoxPen.Location.X + callingButton.Left,
groupBoxPen.Location.Y + callingButton.Top + callingButton.Height);
            p = PointToScreen(p);

            ColorPaletteDialog clDlg = new ColorPaletteDialog(p.X, p.Y);

            clDlg.ShowDialog();

            if (clDlg.DialogResult == DialogResult.OK)
            {
                ColorConverter colconv = new ColorConverter();
                UserSettings.Default.LineColor =
colconv.ConvertToString(clDlg.Color);
                UserSettings.Default.Save();
                SignatureColorSystem =
(Color)colconv.ConvertFromString(UserSettings.Default.LineColor);
                SignatureColor.blue = (byte)clDlg.Color.B;
                SignatureColor.green = (byte)clDlg.Color.G;
                SignatureColor.red = (byte)clDlg.Color.R;

                InitiatilizeSignaturePixels(ref SignaturePixels);
                RedrawPixels(ref SignaturePixels);
            }

            callingButton.CenterColor = SignatureColorSystem;

            Invalidate();

            clDlg.Dispose();
        }

        void BackgroundColorButton_Click(object sender, System.EventArgs e)
        {
            ColorButton callingButton = (ColorButton)sender;
            Point p = new Point(groupBoxBackground.Location.X +
callingButton.Left, groupBoxBackground.Location.Y + callingButton.Top +
callingButton.Height);
            p = PointToScreen(p);

            ColorPaletteDialog clDlg = new ColorPaletteDialog(p.X, p.Y);

            clDlg.ShowDialog();

            if (clDlg.DialogResult == DialogResult.OK)
            {
                ColorConverter colconv = new ColorConverter();
                UserSettings.Default.BackColor =
colconv.ConvertToString(clDlg.Color);
                UserSettings.Default.Save();
                backColorSystem =
(Color)colconv.ConvertFromString(UserSettings.Default.BackColor);
                backColor.blue = (byte)clDlg.Color.B;
                backColor.green = (byte)clDlg.Color.G;
                backColor.red = (byte)clDlg.Color.R;

                InitiatilizeSignaturePixels(ref SignaturePixels);
```

```
                RedrawPixels(ref SignaturePixels);
            }

            callingButton.CenterColor = backColorSystem;

            Invalidate();

            clDlg.Dispose();
        }

        public void MainForm_Deactivate(object sender, EventArgs e)
        {
            //if (FineTuneTC != null && GetForegroundWindow() ==
FineTuneTC.Handle)
            //    FormActivated = true;
            //else
            //    FormActivated = false;
        }

        public void MainForm_Activated(object sender, EventArgs e)
        {
            FormActivated = true;
        }

        private void MakeBackTransparent_CheckedChanged(object sender,
EventArgs e)
        {
            if (MakeBackTransparent.Checked == true)
            {
                UserSettings.Default.TransparentOnSave = true;
            }
            else
            {
                UserSettings.Default.TransparentOnSave = false;
            }
            UserSettings.Default.Save();
        }

        private void HelpAbout_MouseClick(object sender, MouseEventArgs e)
        {

System.Diagnostics.Process.Start("http://sites.google.com/site/jeemgroup/");
        }

        #endregion

        #region More Features
        //Region 9 - More Features: You can fix the tracking at the edges of
the window, reduce the window size of the program...

        // For Webcam Signature 2.0:
        // Explore the possiblity of tracking brightness
        // Brightness = (299 * red + 587 * green + 114 * blue) / 1000

        // Code the dialation and erosion functions to smooth out the pixel
blobs
```



```
        long MouseMoveCounter = 0;
        private void Signature_MouseMove(object sender, MouseEventArgs e)
        {
            Point pt = e.Location;
            if (MouseIsDown && !(SignaturePoints.Count > 0 &&
SignaturePoints[SignaturePoints.Count - 1] != null && pt ==
(Point)SignaturePoints[SignaturePoints.Count - 1]))
            {
                long remainder = 0;
                Math.DivRem(MouseMoveCounter, 2, out remainder);
                if (remainder == 0)
                {
                    SignaturePoints.Add(pt);

                    // Do a rough representation of the signature while the
user draws it
                    if (SignaturePoints.Count >= 2 &&
SignaturePoints[SignaturePoints.Count - 2] != null &&
                        SignaturePoints[SignaturePoints.Count - 1] != null)
                    {
                        DrawLine((Point)SignaturePoints[SignaturePoints.Count
- 2], (Point)SignaturePoints[SignaturePoints.Count - 1],
                            ref SignaturePixels, true);
                    }
                    else if (SignaturePoints.Count >= 1 &&
SignaturePoints[SignaturePoints.Count - 1] != null)
                    {
                        DrawAPoint(pt, ref SignaturePixels, true);
                    }
                }
                else
                {
                    // Do a rough representation of the signature while the
user draws it
                    //if (SignaturePoints.Count >= 1 &&
SignaturePoints[SignaturePoints.Count - 1] != null)
                    //{
                    //    DrawLine(pt,
(Point)SignaturePoints[SignaturePoints.Count - 1],
                    //        ref SignaturePixels, true);
                    //}
                    //else if (SignaturePoints.Count >= 1 &&
SignaturePoints[SignaturePoints.Count - 1] != null)
                    //{
                    //    DrawAPoint(pt, ref SignaturePixels, true);
                    //}
                }



                MouseMoveCounter++;
            }


        }
```



```
        bool MouseIsDown = false;
        private void Signature_MouseUp(object sender, MouseEventArgs e)
        {
            MouseIsDown = false;

            if (SignaturePoints.Count > 0 &&
SignaturePoints[SignaturePoints.Count - 1] != null)
            {
                SignaturePoints.Add(null);
            }

            //ApplySmoothing_MouseClick(this, null);

            // Create an undo point on each keyup
            SetUndo(SignaturePoints);

            MouseMoveCounter = 0;
        }

        private void Signature_MouseDown(object sender, MouseEventArgs e)
        {
            MouseIsDown = true;
        }

        private void Signature_MouseLeave(object sender, EventArgs e)
        {
        }

        #endregion

        private void MainForm_SizeChanged(object sender, EventArgs e)
        {
            //Splitter distance set to a fixed value: 150
            splitContainer1.SplitterDistance = 150;
        }

        private void WebcamMouseCB_CheckedChanged(object sender, EventArgs e)
        {
            //Button to choose the WebCam mouse option
            if (WebcamMouseCB.Checked == true)
            {
                ShowPointLocation.Location = new Point(-100, -100);
                WebcamMouseMode = true;
            }
            else
            {
                WebcamMouseMode = false;
            }
        }
    }
}
```

ColorButton.cs

```
//Class 3

using System;
using System.Windows.Forms;
using System.Drawing;

namespace FancyColorDialog
{

//All the Drawings with automatic rectification to edges, Paintings and
EventHandlers will be initialized in this class and referred by the Class2
public class ColorButton : Button
{
      Color centerColor;

      public ColorButton()
      {
            MouseEnter += new EventHandler(OnMouseEnter);
            MouseLeave += new EventHandler(OnMouseLeave);

            MouseUp += new MouseEventHandler(OnMouseUp);

            Paint += new PaintEventHandler(ButtonPaint);
      }

      public Color CenterColor
      {
            get{ return centerColor; }
            set{ centerColor = value; }
      }

      void OnMouseEnter(object sender, EventArgs e)
      {
            Invalidate();
      }

      void OnMouseLeave(object sender, EventArgs e)
      {
            Invalidate();
      }

      void OnMouseUp(object sender, MouseEventArgs e)
      {
            Invalidate();
      }

      void ButtonPaint(Object sender, PaintEventArgs e)
      {
            Graphics g = e.Graphics ;

            Rectangle r = ClientRectangle;

            byte border = 4;
```

```
            byte right_border = 15;

            Rectangle rc = new Rectangle(r.Left + border, r.Top + border,
                                         r.Width - border - right_border - 1,
r.Height - border * 2 - 1);

            SolidBrush centerColorBrush = new SolidBrush( centerColor );
            g.FillRectangle(centerColorBrush, rc);

            Pen pen = new Pen( Color.Black );
            g.DrawRectangle( pen, rc );

            //draw the arrow
            Point p1 = new Point( r.Width - 9, r.Height / 2 - 1 );
            Point p2 = new Point(r.Width - 5, r.Height / 2 - 1 );
            g.DrawLine(pen, p1, p2);

            p1 = new Point( r.Width - 8, r.Height / 2 );
            p2 = new Point(r.Width - 6, r.Height / 2 );
            g.DrawLine(pen, p1, p2);

            p1 = new Point( r.Width - 7, r.Height / 2 );
            p2 = new Point(r.Width - 7, r.Height / 2 + 1 );
            g.DrawLine(pen, p1, p2);

            //draw the divider line
            pen = new Pen( SystemColors.ControlDark );
            p1 = new Point( r.Width - 12, 4 );
            p2 = new Point(r.Width - 12, r.Height - 5 );
            g.DrawLine(pen, p1, p2);

            pen = new Pen( SystemColors.ControlLightLight );
            p1 = new Point( r.Width - 11, 4 );
            p2 = new Point(r.Width - 11, r.Height - 5 );
            g.DrawLine(pen, p1, p2);
        }
}

}
```

ColorDialog.cs

```
//Class 4
using System;
using System.Windows.Forms;
using System.Drawing;

namespace FancyColorDialog
{
    //All the color palet diaolog with mouse over text are built and written
in this class
public class ColorPaletteDialog : Form
{
      byte max = 40;
      Panel[] panel = new Panel[40];

      Color[] color = new Color[40]
      {
            //row 1
            Color.FromArgb(0,0,0), Color.FromArgb(153,51,0),
Color.FromArgb(51,51,0), Color.FromArgb(0,51,0),
            Color.FromArgb(0,51,102), Color.FromArgb(0,0,128),
Color.FromArgb(51,51,153), Color.FromArgb(51,51,51),

            //row 2
            Color.FromArgb(128,0,0), Color.FromArgb(255,102,0),
Color.FromArgb(128,128,0), Color.FromArgb(0,128,0),
            Color.FromArgb(0,128,128), Color.FromArgb(0,0,255),
Color.FromArgb(102,102,153), Color.FromArgb(128,128,128),

            //row 3
            Color.FromArgb(255,0,0), Color.FromArgb(255,153,0),
Color.FromArgb(153,204,0), Color.FromArgb(51,153,102),
            Color.FromArgb(51,204,204), Color.FromArgb(51,102,255),
Color.FromArgb(128,0,128), Color.FromArgb(153,153,153),

            //row 4
            Color.FromArgb(255,0,255), Color.FromArgb(255,204,0),
Color.FromArgb(255,255,0), Color.FromArgb(0,255,0),
            Color.FromArgb(0,255,255), Color.FromArgb(0,204,255),
Color.FromArgb(153,51,102), Color.FromArgb(192,192,192),

            //row 5
            Color.FromArgb(255,153,204), Color.FromArgb(255,204,153),
Color.FromArgb(255,255,153), Color.FromArgb(204,255,204),
            Color.FromArgb(204,255,255), Color.FromArgb(153,204,255),
Color.FromArgb(204,153,255), Color.FromArgb(255,255,255)

      };

      string[] colorName = new string[40]
      {
            "Black", "Brown", "Olive Green", "Dark Green", "Dark Teal", "Dark
Blue", "Indigo", "Gray-80%",
```

```
            "Dark Red", "Orange", "Dark Yellow", "Green", "Teal", "Blue",
"Blue-Gray", "Gray-50%",
            "Red", "Light Orange", "Lime", "Sea Green", "Aqua", "Light Blue",
"Violet", "Gray-40%",
            "Pink", "Gold", "Yellow", "Bright Green", "Turquoise", "Sky
Blue", "Plum", "Gray-25%",
            "Rose", "Tan", "Light Yellow", "Light Green", "Light Turquoise",
"Pale Blue", "Lavender", "White"
      };

      Button moreColorsButton = new Button();
      Button cancelButton = new Button();
      Color selectedColor;

      public ColorPaletteDialog(int x, int y)
      {
            Size = new Size(158, 155);
            FormBorderStyle = FormBorderStyle.FixedDialog;
            MinimizeBox = MaximizeBox = ControlBox = false;
            ShowInTaskbar = false;
            CenterToScreen();
            Location = new Point(x, y);

            BuildPalette();

            moreColorsButton.Text = "More colors ...";
            moreColorsButton.Size = new Size(142, 22);
            moreColorsButton.Location = new Point(5, 99);
            moreColorsButton.Click += new
EventHandler(moreColorsButton_Click);
            moreColorsButton.FlatStyle = FlatStyle.Popup;
            Controls.Add(moreColorsButton);

            //"invisible" button to cancel at Escape
        cancelButton.Text = "Close";
            cancelButton.Size = new Size(142, 22);
            cancelButton.Location = new Point(5, 125);
            cancelButton.Click += new EventHandler(cancelButton_Click);
        cancelButton.FlatStyle = FlatStyle.Standard;
            Controls.Add(cancelButton);
            cancelButton.TabIndex = 0;
            cancelButton.DialogResult = DialogResult.Cancel;
            this.CancelButton = cancelButton;
      }

      public Color Color
      {
            get {return selectedColor;}
      }

      void BuildPalette()
      {
            byte pwidth = 16;
            byte pheight = 16;
            byte pdistance = 2;
            byte border = 5;
            int x = border, y = border;
```

```
            ToolTip toolTip = new ToolTip();

            for(int i = 0; i < max; i++)
            {
                  panel[i] = new Panel();
                  panel[i].Height = pwidth;
                  panel[i].Width = pheight;
                  panel[i].Location = new Point(x, y);
                  toolTip.SetToolTip(panel[i], colorName[i]);

                  this.Controls.Add(panel[i]);

                  if(x < ( 7 * (pwidth + pdistance)))
                        x += pwidth + pdistance;
                  else
                  {
                        x = border;
                        y += pheight + pdistance;
                  }

                  panel[i].BackColor = color[i];
                  panel[i].MouseEnter += new EventHandler(OnMouseEnterPanel);
                  panel[i].MouseLeave += new EventHandler(OnMouseLeavePanel);

                  panel[i].MouseDown += new
MouseEventHandler(OnMouseDownPanel);
                  panel[i].MouseUp += new MouseEventHandler(OnMouseUpPanel);

                  panel[i].Paint += new PaintEventHandler(OnPanelPaint);

            }
      }

      void moreColorsButton_Click(object sender, System.EventArgs e)
    {
      ColorDialog colDialog = new ColorDialog();
      colDialog.FullOpen = true;
      if(colDialog.ShowDialog() == DialogResult.OK)
      {
            selectedColor = colDialog.Color;
            DialogResult = DialogResult.OK;
      }
      colDialog.Dispose();

      Close();
    }

    void cancelButton_Click(object sender, System.EventArgs e)
    {
      Close();
    }

      void OnMouseEnterPanel(object sender, EventArgs e)
      {
            DrawPanel(sender, 1);
      }
```



```
        void OnMouseLeavePanel(object sender, EventArgs e)
        {
              DrawPanel(sender, 0);
        }

        void OnMouseDownPanel(object sender, MouseEventArgs e)
        {
              DrawPanel(sender, 2);
        }

        void OnMouseUpPanel(object sender, MouseEventArgs e)
        {
              Panel panel = (Panel)sender;
              selectedColor = panel.BackColor;
              DialogResult = DialogResult.OK;
              Close();
        }

        void DrawPanel(object sender, byte state)
        {
              Panel panel = (Panel)sender;

              Graphics g = panel.CreateGraphics();

              Pen pen1, pen2;

              if(state == 1)          //mouse over
              {
                    pen1 = new Pen( SystemColors.ControlLightLight );

                    pen2 = new Pen( SystemColors. ControlDarkDark);
              }
              else if(state == 2)     //clicked
              {
                    pen1 = new Pen( SystemColors.ControlDarkDark );

                    pen2 = new Pen( SystemColors.ControlLightLight );

              }
              else                    //neutral
              {
                    pen1 = new Pen( SystemColors.ControlDark );

                    pen2 = new Pen( SystemColors.ControlDark );

              }

              Rectangle r = panel.ClientRectangle;
              Point p1 = new Point( r.Left, r.Top );                        //top
left
              Point p2 = new Point( r.Right -1, r.Top );                    //top
right
              Point p3 = new Point( r.Left, r.Bottom -1 );          //bottom
left
              Point p4 = new Point( r.Right -1, r.Bottom -1 );      //bottom
right
```



```
            g.DrawLine( pen1, p1, p2 );
            g.DrawLine( pen1, p1, p3 );
            g.DrawLine( pen2, p2, p4 );
            g.DrawLine( pen2, p3, p4 );
      }

      void OnPanelPaint(Object sender, PaintEventArgs e)
      {
            DrawPanel(sender, 0);
      }
}

}
```

DeviceSelector.cs

```
//Class 5

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Runtime.InteropServices;

using DShowNET;
using DShowNET.Device;

namespace SampleGrabberNET
{


//In this class a dialog will be displayed let user select a capture device
if more then one installed.
public class DeviceSelector : System.Windows.Forms.Form
{
	private System.Windows.Forms.Button okButton;
	private System.Windows.Forms.Button cancelButton;
	private System.Windows.Forms.ListView deviceListVw;
	private System.Windows.Forms.ColumnHeader nameColHd;
	//Required designer variable.
	private System.ComponentModel.Container components = null;

	public DeviceSelector( ArrayList devs )
	{
		// Required for Windows Form Designer support
		InitializeComponent();

		ListViewItem item = null;
		foreach( DsDevice d in devs )
		{
			item = new ListViewItem( d.Name );
			item.Tag = d;
			deviceListVw.Items.Add( item );
		}
	}

	// Clean up any resources being used.
	protected override void Dispose( bool disposing )
	{
		if( disposing )
		{
			if(components != null)
			{
				components.Dispose();
			}
		}
		base.Dispose( disposing );
	}
```

```
            #region Windows Form Designer generated code
            // Required method for Designer support - do not modify
            // the contents of this method with the code editor.

            private void InitializeComponent()
            {
                  this.deviceListVw = new System.Windows.Forms.ListView();
                  this.okButton = new System.Windows.Forms.Button();
                  this.cancelButton = new System.Windows.Forms.Button();
                  this.nameColHd = new System.Windows.Forms.ColumnHeader();
                  this.SuspendLayout();
                  //
                  // deviceListVw
                  //
                  this.deviceListVw.Columns.AddRange(new
System.Windows.Forms.ColumnHeader[] {

                                                      this.nameColHd});
                  this.deviceListVw.FullRowSelect = true;
                  this.deviceListVw.GridLines = true;
                  this.deviceListVw.HeaderStyle =
System.Windows.Forms.ColumnHeaderStyle.Nonclickable;
                  this.deviceListVw.HideSelection = false;
                  this.deviceListVw.Location = new System.Drawing.Point(8,
8);
                  this.deviceListVw.MultiSelect = false;
                  this.deviceListVw.Name = "deviceListVw";
                  this.deviceListVw.Size = new System.Drawing.Size(344, 112);
                  this.deviceListVw.TabIndex = 0;
                  this.deviceListVw.View = System.Windows.Forms.View.Details;
                  this.deviceListVw.DoubleClick += new
System.EventHandler(this.deviceListVw_DoubleClick);
                  //
                  // okButton
                  //
                  this.okButton.Anchor =
((System.Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Left)
                        | System.Windows.Forms.AnchorStyles.Right);
                  this.okButton.DialogResult =
System.Windows.Forms.DialogResult.OK;
                  this.okButton.Location = new System.Drawing.Point(108,
130);
                  this.okButton.Name = "okButton";
                  this.okButton.Size = new System.Drawing.Size(64, 24);
                  this.okButton.TabIndex = 1;
                  this.okButton.Text = "OK";
                  this.okButton.Click += new
System.EventHandler(this.okButton_Click);
                  //
                  // cancelButton
                  //
                  this.cancelButton.Anchor =
((System.Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Left)
                        | System.Windows.Forms.AnchorStyles.Right);
```



```
                this.cancelButton.DialogResult =
System.Windows.Forms.DialogResult.Cancel;
                this.cancelButton.Location = new System.Drawing.Point(188,
130);
                this.cancelButton.Name = "cancelButton";
                this.cancelButton.Size = new System.Drawing.Size(64, 24);
                this.cancelButton.TabIndex = 1;
                this.cancelButton.Text = "Cancel";
                this.cancelButton.Click += new
System.EventHandler(this.cancelButton_Click);
                //
                // nameColHd
                //
                this.nameColHd.Text = "Name";
                this.nameColHd.Width = 400;
                //
                // DeviceSelector
                //
                this.AcceptButton = this.okButton;
                this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
                this.CancelButton = this.cancelButton;
                this.ClientSize = new System.Drawing.Size(360, 159);
                this.Controls.AddRange(new System.Windows.Forms.Control[] {

                                this.okButton,

                                this.deviceListVw,

                                this.cancelButton});
                this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
                this.MaximizeBox = false;
                this.MinimizeBox = false;
                this.Name = "DeviceSelector";
                this.ShowInTaskbar = false;
                this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterParent;
                this.Text = "Select video capture device";
                this.ResumeLayout(false);

            }
            #endregion


      private void deviceListVw_DoubleClick(object sender, System.EventArgs
e)
      {
            this.okButton_Click( sender, e );
      }

      private void okButton_Click(object sender, System.EventArgs e)
      {
            if( deviceListVw.SelectedItems.Count != 1 )
                  return;
            ListViewItem selitem = deviceListVw.SelectedItems[0];
            SelectedDevice = selitem.Tag as DsDevice;
            Close();
```



```
        }

        private void cancelButton_Click(object sender, System.EventArgs e)
        {
              Close();
        }

        public DsDevice          SelectedDevice;
}

}
```


FineTuneTrackColorsForm.cs

```
//Class 6

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
//using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using SampleGrabberNET;

//In this class the UserSettings will be loaded and referred to its parent
form MainForm, we can choose the color that should be tracked manually
//by choosing the RGB colours as well as we can reset the color that was set,
filter the background to check how well is our choice of the color and
//the type of the filtering.
namespace WebCam_Signature
{
    public partial class FineTuneTrackColorsForm : Form
    {
        [DllImport("user32.dll")]
        static extern IntPtr GetForegroundWindow();

        MainForm parentForm;
        Bitmap SampleStaticBack;
        Bitmap SelectColor;
        PixelData PixelFromPoint;

        public FineTuneTrackColorsForm(MainForm qparent)
        {
            parentForm = qparent;
            GC.KeepAlive(parentForm);
            InitializeComponent();

            this.Show();
            this.BringToFront();
        }

        private void FineTuneTrackColorsForm_FormClosing(object sender,
FormClosingEventArgs e)
        {
            parentForm.FineTuneTC = null;
        }

        private void FineTuneTrackColorsForm_Shown(object sender, EventArgs
e)
        {
            LoadSettings();
            groupBox1.Focus();
        }
```

```
        public void LoadSettings()
        {
            ColorRangeThresNB.Value =
(decimal)UserSettings.Default.TrackColorRangeThres;
            if (UserSettings.Default.TrackRed)
            {
                FineTuningColor.Text = "RED RGB Settings";

                TBRedLow.Value = UserSettings.Default.TRLowRed;
                TBRedHigh.Value = UserSettings.Default.TRHighRed;
                TBGreenLow.Value = UserSettings.Default.TRLowGreen;
                TBGreenHigh.Value = UserSettings.Default.TRHighGreen;
                TBBlueLow.Value = UserSettings.Default.TRLowBlue;
                TBBlueHigh.Value = UserSettings.Default.TRHighBlue;
            }
            else if (UserSettings.Default.TrackGreen)
            {
                FineTuningColor.Text = "GREEN RGB Settings";

                TBRedLow.Value = UserSettings.Default.TGLowRed;
                TBRedHigh.Value = UserSettings.Default.TGHighRed;
                TBGreenLow.Value = UserSettings.Default.TGLowGreen;
                TBGreenHigh.Value = UserSettings.Default.TGHighGreen;
                TBBlueLow.Value = UserSettings.Default.TGLowBlue;
                TBBlueHigh.Value = UserSettings.Default.TGHighBlue;
            }
            else
            {
                FineTuningColor.Text = "BLUE RGB Settings";

                TBRedLow.Value = UserSettings.Default.TBLowRed;
                TBRedHigh.Value = UserSettings.Default.TBHighRed;
                TBGreenLow.Value = UserSettings.Default.TBLowGreen;
                TBGreenHigh.Value = UserSettings.Default.TBHighGreen;
                TBBlueLow.Value = UserSettings.Default.TBLowBlue;
                TBBlueHigh.Value = UserSettings.Default.TBHighBlue;
            }

            SelectColor = parentForm.GetFullFrameBitmap();
            SelectColorPB.Image = SelectColor;

            // For Filter Background
            if (UserSettings.Default.NoBackgroundFilter == true)
                NoBackFilterRB.Checked = true;
            else
                NoBackFilterRB.Checked = false;

            if (UserSettings.Default.UseStaticBackFilter == true)
                UseBackStaticFilterRB.Checked = true;
            else
                UseBackStaticFilterRB.Checked = false;

            if (UserSettings.Default.UseDynamicBackFilter == true)
                UseBackDynamicFilterRB.Checked = true;
            else
                UseBackDynamicFilterRB.Checked = false;
```

```
            BackRefreshTimeNB.Value =
(decimal)UserSettings.Default.BackgroundRefreshTime;
            BackCalibrationTimeNB.Value =
(decimal)UserSettings.Default.BackgroundCalibrateTime;
            BackDiffThreshNB.Value =
(decimal)UserSettings.Default.BackgroundDiffThreshold;
            timer1.Enabled = false;
            RefreshBackFilterProgress.Value = 100;
            CaptureBackgroundButton_MouseClick(this, null);
            BackFilteringInfo.Clear();
            BackFilteringInfo.Text = "* Static filtering takes a still
picture of your background and utilizes that to filter out the background. It
is more effective than dynamic filtering if you are sure that your webcam
won't be moved. \r\n"
                +"* Dynamic filtering is recommended because it will update
the background over a period of time (Refresh Time) so that new objects or
movements of the webcam are taken into account.";
            RGBSettingsInfo.Text = "* This section allows you to adjust the
range of colors that the program tracks. To change the range of colors,
either adjust the sliders at the left to" +
                " manually change the RGB values or click \"Grab A Frame\"
and and click on the color you want from the image. \r\n" +
                "* Even with selecting a color from the picture, you may
still have to fine tune the track bars above to get a stable tracking. \r\n"+
                "* The program comes with default settings for Red, Blue, and
Green colors for BIC pens. \r\n" +
                "* Note that you can change the \"Red\", \"Green\", and
\"Blue\" tracking colors to what ever you want them to be. "+
                "You do not have to keep them restricted to the color of
their names. Try using a flashlight and set them to white!";
        }

        private void TBRedLow_Scroll(object sender, EventArgs e)
        {
            if (UserSettings.Default.TrackRed)
            {
                UserSettings.Default.TRLowRed = TBRedLow.Value;
                //UserSettings.Default.TRHighRed = TBRedHigh.Value;
                //UserSettings.Default.TRLowGreen = TBGreenLow.Value ;
                //UserSettings.Default.TRHighGreen = TBGreenHigh.Value;
                //UserSettings.Default.TRLowBlue = TBBlueLow.Value;
                //UserSettings.Default.TRHighBlue = TBGreenHigh.Value;
            }
            else if (UserSettings.Default.TrackGreen)
            {
                UserSettings.Default.TGLowRed = TBRedLow.Value;
                //UserSettings.Default.TGHighRed = TBRedHigh.Value;
                //UserSettings.Default.TGLowGreen = TBGreenLow.Value;
                //UserSettings.Default.TGHighGreen = TBGreenHigh.Value;
                //UserSettings.Default.TGLowBlue = TBBlueLow.Value;
                //UserSettings.Default.TGHighBlue = TBGreenHigh.Value;
            }
            else
            {
                UserSettings.Default.TBLowRed = TBRedLow.Value;
                //UserSettings.Default.TBHighRed = TBRedHigh.Value;
                //UserSettings.Default.TBLowGreen = TBGreenLow.Value;
```



```
            //UserSettings.Default.TBHighGreen = TBGreenHigh.Value;
            //UserSettings.Default.TBLowBlue = TBBlueLow.Value;
            //UserSettings.Default.TBHighBlue = TBGreenHigh.Value;
        }

        UserSettings.Default.Save();
        parentForm.UpdateTrackColorSample();
        parentForm.LoadTrackColor();
    }

    private void TBRedHigh_Scroll(object sender, EventArgs e)
    {
        if (UserSettings.Default.TrackRed)
        {
            UserSettings.Default.TRHighRed = TBRedHigh.Value;
        }
        else if (UserSettings.Default.TrackGreen)
        {
            UserSettings.Default.TGHighRed = TBRedHigh.Value;
        }
        else
        {
            UserSettings.Default.TBHighRed = TBRedHigh.Value;
        }

        UserSettings.Default.Save();
        parentForm.UpdateTrackColorSample();
        parentForm.LoadTrackColor();
    }

    private void TBGreenLow_Scroll(object sender, EventArgs e)
    {
        if (UserSettings.Default.TrackRed)
        {
            UserSettings.Default.TRLowGreen = TBGreenLow.Value;
        }
        else if (UserSettings.Default.TrackGreen)
        {
            UserSettings.Default.TGLowGreen = TBGreenLow.Value;
        }
        else
        {
            UserSettings.Default.TBLowGreen = TBGreenLow.Value;
        }

        UserSettings.Default.Save();
        parentForm.UpdateTrackColorSample();
        parentForm.LoadTrackColor();
    }

    private void TBGreenHigh_Scroll(object sender, EventArgs e)
    {
        if (UserSettings.Default.TrackRed)
        {
            UserSettings.Default.TRHighGreen = TBGreenHigh.Value;
        }
        else if (UserSettings.Default.TrackGreen)
```

```
            {
                UserSettings.Default.TGHighGreen = TBGreenHigh.Value;
            }
            else
            {
                UserSettings.Default.TBHighGreen = TBGreenHigh.Value;
            }

            UserSettings.Default.Save();
            parentForm.UpdateTrackColorSample();
            parentForm.LoadTrackColor();
        }

        private void TBBlueLow_Scroll(object sender, EventArgs e)
        {
            if (UserSettings.Default.TrackRed)
            {
                UserSettings.Default.TRLowBlue = TBBlueLow.Value;
            }
            else if (UserSettings.Default.TrackGreen)
            {
                UserSettings.Default.TGLowBlue = TBBlueLow.Value;
            }
            else
            {
                UserSettings.Default.TBLowBlue = TBBlueLow.Value;
            }

            UserSettings.Default.Save();
            parentForm.UpdateTrackColorSample();
            parentForm.LoadTrackColor();
        }

        private void TBBlueHigh_Scroll(object sender, EventArgs e)
        {
            if (UserSettings.Default.TrackRed)
            {
                UserSettings.Default.TRHighBlue = TBBlueHigh.Value;
            }
            else if (UserSettings.Default.TrackGreen)
            {
                UserSettings.Default.TGHighBlue = TBBlueHigh.Value;
            }
            else
            {
                UserSettings.Default.TBHighBlue = TBBlueHigh.Value;
            }

            UserSettings.Default.Save();
            parentForm.UpdateTrackColorSample();
            parentForm.LoadTrackColor();
        }

        private void ResetFineTune_MouseClick(object sender, MouseEventArgs
e)
        {
            if (UserSettings.Default.TrackRed)
```



```
            {
                FineTuningColor.Text = "RED RGB Settings";

                UserSettings.Default.TRLowRed = 140;
                UserSettings.Default.TRHighRed = 255;
                UserSettings.Default.TRLowGreen = 0;
                UserSettings.Default.TRHighGreen = 70;
                UserSettings.Default.TRLowBlue = 0;
                UserSettings.Default.TRHighBlue = 70;
            }
            else if (UserSettings.Default.TrackGreen)
            {
                FineTuningColor.Text = "GREEN RGB Settings";

                UserSettings.Default.TGLowRed = 0;
                UserSettings.Default.TGHighRed = 70;
                UserSettings.Default.TGLowGreen = 140;
                UserSettings.Default.TGHighGreen = 255;
                UserSettings.Default.TGLowBlue = 0;
                UserSettings.Default.TGHighBlue = 70;
            }
            else
            {
                FineTuningColor.Text = "BLUE RGB Settings";

                UserSettings.Default.TBLowRed = 0;
                UserSettings.Default.TBHighRed = 70;
                UserSettings.Default.TBLowGreen = 0;
                UserSettings.Default.TBHighGreen = 70;
                UserSettings.Default.TBLowBlue = 140;
                UserSettings.Default.TBHighBlue = 255;
            }

            UserSettings.Default.Save();
            parentForm.UpdateTrackColorSample();
            parentForm.LoadTrackColor();
            LoadSettings();
        }

        private void FineTuneTrackColorsForm_Deactivate(object sender,
EventArgs e)
        {
            if (GetForegroundWindow() != parentForm.Handle)
            {
                parentForm.MainForm_Deactivate(this, null);
            }
        }

        private void FineTuneTrackColorsForm_Activated(object sender,
EventArgs e)
        {
            parentForm.MainForm_Activated(this, null);
        }


        private void ResetBackFilterButton_MouseClick(object sender,
MouseEventArgs e)
```



```
        {
            UserSettings.Default.NoBackgroundFilter = false;
            UserSettings.Default.UseStaticBackFilter = false;
            UserSettings.Default.UseDynamicBackFilter = true;
            UserSettings.Default.BackgroundRefreshTime = 60;
            UserSettings.Default.BackgroundCalibrateTime = 2;
            UserSettings.Default.BackgroundDiffThreshold = 50;

            UserSettings.Default.Save();
            LoadSettings();
            UpdateBackFilter();
        }

        private void BackCalibrationTimeNB_ValueChanged(object sender,
EventArgs e)
        {
            UserSettings.Default.BackgroundRefreshTime =
(int)BackCalibrationTimeNB.Value;
            UserSettings.Default.Save();
            UpdateBackFilter();
        }

        private void BackDiffThreshNB_ValueChanged(object sender, EventArgs
e)
        {
            UserSettings.Default.BackgroundDiffThreshold =
(int)BackDiffThreshNB.Value;
            UserSettings.Default.Save();
            UpdateBackFilter();
        }

        private void BackRefreshTimeNB_ValueChanged(object sender, EventArgs
e)
        {
            UserSettings.Default.BackgroundRefreshTime =
(int)BackRefreshTimeNB.Value;
            UserSettings.Default.Save();
            UpdateBackFilter();
        }

        private void NoBackFilterRB_MouseClick(object sender, MouseEventArgs
e)
        {
            if (NoBackFilterRB.Checked == true)
            {
                UserSettings.Default.NoBackgroundFilter = true;
                UserSettings.Default.UseStaticBackFilter = false;
                UserSettings.Default.UseDynamicBackFilter = false;
                UseBackStaticFilterRB.Checked = false;
                UseBackDynamicFilterRB.Checked = false;
            }
            else if (NoBackFilterRB.Checked == false &&
UseBackStaticFilterRB.Checked == false && UseBackDynamicFilterRB.Checked ==
false)
            {
                UserSettings.Default.NoBackgroundFilter = true;
                UserSettings.Default.UseStaticBackFilter = false;
```

```
                UserSettings.Default.UseDynamicBackFilter = false;
                NoBackFilterRB.Checked = true;
                UseBackStaticFilterRB.Checked = false;
                UseBackDynamicFilterRB.Checked = false;
            }
            UserSettings.Default.Save();

            UpdateBackFilter();
        }

        private void UseBackStaticFilterRB_MouseClick(object sender,
MouseEventArgs e)
        {
            if (UseBackStaticFilterRB.Checked == true)
            {
                UserSettings.Default.NoBackgroundFilter = false;
                UserSettings.Default.UseStaticBackFilter = true;
                UserSettings.Default.UseDynamicBackFilter = false;
                NoBackFilterRB.Checked = false;
                UseBackDynamicFilterRB.Checked = false;
            }
            else if (NoBackFilterRB.Checked == false &&
UseBackStaticFilterRB.Checked == false && UseBackDynamicFilterRB.Checked ==
false)
            {
                UserSettings.Default.NoBackgroundFilter = true;
                UserSettings.Default.UseStaticBackFilter = false;
                UserSettings.Default.UseDynamicBackFilter = false;
                NoBackFilterRB.Checked = true;
                UseBackStaticFilterRB.Checked = false;
                UseBackDynamicFilterRB.Checked = false;
            }
            UserSettings.Default.Save();
            UpdateBackFilter();
        }

        private void UseBackDynamicFilterRB_MouseClick(object sender,
MouseEventArgs e)
        {
            if (UseBackDynamicFilterRB.Checked == true)
            {
                UserSettings.Default.NoBackgroundFilter = false;
                UserSettings.Default.UseStaticBackFilter = false;
                UserSettings.Default.UseDynamicBackFilter = true;
                NoBackFilterRB.Checked = false;
                UseBackStaticFilterRB.Checked = false;
            }
            else if (NoBackFilterRB.Checked == false &&
UseBackStaticFilterRB.Checked == false && UseBackDynamicFilterRB.Checked ==
false)
            {
                UserSettings.Default.NoBackgroundFilter = true;
                UserSettings.Default.UseStaticBackFilter = false;
                UserSettings.Default.UseDynamicBackFilter = false;
                NoBackFilterRB.Checked = true;
                UseBackStaticFilterRB.Checked = false;
                UseBackDynamicFilterRB.Checked = false;
```



```
            }
            UserSettings.Default.Save();
            UpdateBackFilter();
        }

        private void RefreshBackFilterButton_MouseClick(object sender,
MouseEventArgs e)
        {
            UpdateBackFilter();
        }


        private void CaptureBackgroundButton_MouseClick(object sender,
MouseEventArgs e)
        {
            SampleStaticBack = new Bitmap(parentForm.GetStaticBackBitmap());
            StaticBackFilterPB.Image = SampleStaticBack;
        }

        private void UpdateBackFilter()
        {
            parentForm.RefreshBackFilteringParams();
            timer1.Enabled = true;
            timer1.Start();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            RefreshBackFilterProgress.Value =
parentForm.RefreshingBackgroundFilterProgress();
            if (RefreshBackFilterProgress.Value >= 100)
            {
                timer1.Stop();
                timer1.Enabled = false;
                CaptureBackgroundButton_MouseClick(this, null);
            }
        }

        private void GrabAFrameButton_MouseClick(object sender,
MouseEventArgs e)
        {
            try
            {
                SelectColor = new Bitmap(parentForm.GetFullFrameBitmap());
                SelectColorPB.Image = SelectColor;
            }
            catch (Exception ee)
            {
                Console.WriteLine("Error. No webcam picture available");
            }
            GC.KeepAlive(parentForm);
        }


        private void SelectColorPB_MouseClick(object sender, MouseEventArgs
e)
        {
```



```
            Point PointofColor = e.Location;

            Bitmap tempBitmap = new Bitmap(SelectColor);
            UnsafeBitmap WebCamBitmap = new UnsafeBitmap(ref tempBitmap,
true);
            WebCamBitmap.LockBitmap();
            PixelFromPoint = new PixelData();
            PixelFromPoint = WebCamBitmap.GetPixel(PointofColor.X,
PointofColor.Y);
            PixelData tempPixel;

            PixelData transparentPixel = new PixelData();
            transparentPixel.red = (byte)0;
            transparentPixel.green = (byte)0;
            transparentPixel.blue = (byte)0;

            int TCRange = UserSettings.Default.TrackColorRangeThres;

            for (int i = 0; i < WebCamBitmap.Bitmap.Width; i++)
            {
                for (int j = 0; j < WebCamBitmap.Bitmap.Height; j++)
                {
                    tempPixel = WebCamBitmap.GetPixel(i, j);
                    if (Math.Abs((int)tempPixel.blue -
(int)PixelFromPoint.blue) > TCRange
                        || Math.Abs((int)tempPixel.red -
(int)PixelFromPoint.red) > TCRange
                        || Math.Abs((int)tempPixel.green -
(int)PixelFromPoint.green) > TCRange)
                    {
                        WebCamBitmap.SetPixel(i, j, transparentPixel);
                    }
                }
            }
            WebCamBitmap.UnlockBitmap();
            SelectColor = new Bitmap(tempBitmap);
            SelectColorPB.Image = SelectColor;

            SetTCwithTCRange();
        }

        private void ColorRangeThresNB_ValueChanged(object sender, EventArgs
e)
        {
            UserSettings.Default.TrackColorRangeThres =
(int)ColorRangeThresNB.Value;
            UserSettings.Default.Save();

            //SetTCwithTCRange();
        }

        private void SetTCwithTCRange()
        {
            int TCRange = UserSettings.Default.TrackColorRangeThres;
            if (UserSettings.Default.TrackRed)
            {
                FineTuningColor.Text = "RED RGB Settings";
```

```
                UserSettings.Default.TRLowRed = (int)PixelFromPoint.red -
TCRange;
                if (UserSettings.Default.TRLowRed < 0)
                    UserSettings.Default.TRLowRed = 0;

                UserSettings.Default.TRHighRed = (int)PixelFromPoint.red +
TCRange;
                if (UserSettings.Default.TRHighRed > 255)
                    UserSettings.Default.TRHighRed = 255;

                UserSettings.Default.TRLowGreen = (int)PixelFromPoint.green -
TCRange;
                if (UserSettings.Default.TRLowGreen < 0)
                    UserSettings.Default.TRLowGreen = 0;

                UserSettings.Default.TRHighGreen = (int)PixelFromPoint.green
+ TCRange;
                if (UserSettings.Default.TRHighGreen > 255)
                    UserSettings.Default.TRHighGreen = 255;

                UserSettings.Default.TRLowBlue = (int)PixelFromPoint.blue -
TCRange;
                if (UserSettings.Default.TRLowBlue < 0)
                    UserSettings.Default.TRLowBlue = 0;

                UserSettings.Default.TRHighBlue = (int)PixelFromPoint.blue +
TCRange;
                if (UserSettings.Default.TRHighBlue > 255)
                    UserSettings.Default.TRHighBlue = 255;
            }
            else if (UserSettings.Default.TrackGreen)
            {
                FineTuningColor.Text = "GREEN RGB Settings";

                UserSettings.Default.TGLowRed = (int)PixelFromPoint.red -
TCRange;
                if (UserSettings.Default.TGLowRed < 0)
                    UserSettings.Default.TGLowRed = 0;

                UserSettings.Default.TGHighRed = (int)PixelFromPoint.red +
TCRange;
                if (UserSettings.Default.TGHighRed > 255)
                    UserSettings.Default.TGHighRed = 255;

                UserSettings.Default.TGLowGreen = (int)PixelFromPoint.green -
TCRange;
                if (UserSettings.Default.TGLowGreen < 0)
                    UserSettings.Default.TGLowGreen = 0;

                UserSettings.Default.TGHighGreen = (int)PixelFromPoint.green
+ TCRange;
                if (UserSettings.Default.TGHighGreen > 255)
                    UserSettings.Default.TGHighGreen = 255;

                UserSettings.Default.TGLowBlue = (int)PixelFromPoint.blue -
TCRange;
```



```
                if (UserSettings.Default.TGLowBlue < 0)
                    UserSettings.Default.TGLowBlue = 0;

                UserSettings.Default.TGHighBlue = (int)PixelFromPoint.blue +
TCRange;
                if (UserSettings.Default.TGHighBlue > 255)
                    UserSettings.Default.TGHighBlue = 255;
            }
            else
            {
                FineTuningColor.Text = "BLUE RGB Settings";

                UserSettings.Default.TBLowRed = (int)PixelFromPoint.red -
TCRange;
                if (UserSettings.Default.TBLowRed < 0)
                    UserSettings.Default.TBLowRed = 0;

                UserSettings.Default.TBHighRed = (int)PixelFromPoint.red +
TCRange;
                if (UserSettings.Default.TBHighRed > 255)
                    UserSettings.Default.TBHighRed = 255;

                UserSettings.Default.TBLowGreen = (int)PixelFromPoint.green -
TCRange;
                if (UserSettings.Default.TBLowGreen < 0)
                    UserSettings.Default.TBLowGreen = 0;

                UserSettings.Default.TBHighGreen = (int)PixelFromPoint.green
+ TCRange;
                if (UserSettings.Default.TBHighGreen > 255)
                    UserSettings.Default.TBHighGreen = 255;

                UserSettings.Default.TBLowBlue = (int)PixelFromPoint.blue -
TCRange;
                if (UserSettings.Default.TBLowBlue < 0)
                    UserSettings.Default.TBLowBlue = 0;

                UserSettings.Default.TBHighBlue = (int)PixelFromPoint.blue +
TCRange;
                if (UserSettings.Default.TBHighBlue > 255)
                    UserSettings.Default.TBHighBlue = 255;
            }

            UserSettings.Default.Save();

            parentForm.UpdateTrackColorSample();
            parentForm.LoadTrackColor();


            if (UserSettings.Default.TrackRed)
            {
                FineTuningColor.Text = "RED RGB Settings";

                TBRedLow.Value = UserSettings.Default.TRLowRed;
                TBRedHigh.Value = UserSettings.Default.TRHighRed;
                TBGreenLow.Value = UserSettings.Default.TRLowGreen;
                TBGreenHigh.Value = UserSettings.Default.TRHighGreen;
```



```
                TBBlueLow.Value = UserSettings.Default.TRLowBlue;
                TBBlueHigh.Value = UserSettings.Default.TRHighBlue;
            }
            else if (UserSettings.Default.TrackGreen)
            {
                FineTuningColor.Text = "GREEN RGB Settings";

                TBRedLow.Value = UserSettings.Default.TGLowRed;
                TBRedHigh.Value = UserSettings.Default.TGHighRed;
                TBGreenLow.Value = UserSettings.Default.TGLowGreen;
                TBGreenHigh.Value = UserSettings.Default.TGHighGreen;
                TBBlueLow.Value = UserSettings.Default.TGLowBlue;
                TBBlueHigh.Value = UserSettings.Default.TGHighBlue;
            }
            else
            {
                FineTuningColor.Text = "BLUE RGB Settings";

                TBRedLow.Value = UserSettings.Default.TBLowRed;
                TBRedHigh.Value = UserSettings.Default.TBHighRed;
                TBGreenLow.Value = UserSettings.Default.TBLowGreen;
                TBGreenHigh.Value = UserSettings.Default.TBHighGreen;
                TBBlueLow.Value = UserSettings.Default.TBLowBlue;
                TBBlueHigh.Value = UserSettings.Default.TBHighBlue;
            }
        }

    }
}
```

SingleApplication.cs

```
//Class 7

using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Text;
using System.Diagnostics;
using System.Threading;
using System.Reflection;
using System.IO;

//In this class we get the first instance that is not this instance, has the
same process name and was started from the same file name
//and location. Also check that the process has a valid window handle in this
session to filter out other user's processes;
//As well as it controls the state of the current opened window such as
minimization, maximization...

namespace SingleInstance
{



      public class SingleApplication
      {
            public SingleApplication()
            {

            }

            // Imports


            [DllImport("user32.dll")]
            private static extern int ShowWindow(IntPtr hWnd, int nCmdShow);

            [DllImport("user32.dll")]
            private static extern int SetForegroundWindow(IntPtr hWnd);

            [DllImport("user32.dll")]
            private static extern int IsIconic(IntPtr hWnd);


            // GetCurrentInstanceWindowHandle


            private static IntPtr GetCurrentInstanceWindowHandle()
            {
                  IntPtr hWnd = IntPtr.Zero;
                  Process process = Process.GetCurrentProcess();
                  Process[] processes =
Process.GetProcessesByName(process.ProcessName);
                  foreach(Process _process in processes)
```

```
                {

                        if (_process.Id != process.Id &&
                              _process.MainModule.FileName ==
process.MainModule.FileName &&
                              _process.MainWindowHandle != IntPtr.Zero)
                        {
                              hWnd = _process.MainWindowHandle;
                              break;
                        }
                }
                return hWnd;
          }

          // SwitchToCurrentInstance

          private static void SwitchToCurrentInstance()
          {
                IntPtr hWnd = GetCurrentInstanceWindowHandle();
                if (hWnd != IntPtr.Zero)
                {
                      // Restore window if minimised. Do not restore if
already in
                      // normal or maximised window state, since we don't
want to
                      // change the current state of the window.
              if (IsIconic(hWnd) != 0)
              {
                  ShowWindow(hWnd, SW_RESTORE);
              }
              else
              {
                  ShowWindow(hWnd, 1);
              }

                      SetForegroundWindow(hWnd);
                }
          }


          // Execute a form base application if another instance already
running on
          // the system activate previous one
          /// <param name="frmMain">main form</param>
          /// <returns>true if no previous instance is running</returns>
          public static bool Run(System.Windows.Forms.Form frmMain)
          {
          //Console.WriteLine("Try running");
                if(IsAlreadyRunning())
                {
                      //set focus on previously running app
                      SwitchToCurrentInstance();
                      return false;
                }
          try
          {
              Application.Run(frmMain);
```

```
            }
            catch (Exception)
            {
                Console.WriteLine("Run exception");
            }

                  return true;
            }

            /// <summary>
            /// for console base application
            /// </summary>
            /// <returns></returns>
            public static bool Run()
            {
                  if(IsAlreadyRunning())
                  {
                        return false;
                  }
                  return true;
            }

            /// <summary>
            /// check if given exe alread running or not
            /// </summary>
            /// <returns>returns true if already running</returns>
            private static bool IsAlreadyRunning()
            {
                  string strLoc = Assembly.GetExecutingAssembly().Location;
                  FileSystemInfo fileInfo = new FileInfo(strLoc);
                  string sExeName = fileInfo.Name;
                  bool bCreatedNew;

                  mutex = new Mutex(true, "Global\\"+sExeName, out
bCreatedNew);
                  if (bCreatedNew)
                        mutex.ReleaseMutex();

                  return !bCreatedNew;
            }

            static Mutex mutex;
            const int SW_RESTORE = 9;
      }
}
```


UnsafeBitmap.cs

```
//Class 8

using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Drawing.Imaging;

//In this class, all the bitmap saving information will be initialized and
set once the program is launched, whether it was related to
//the way each pixel should be painted or related to another pixel it it had
an approximately related color

namespace SampleGrabberNET
{
    public unsafe class UnsafeBitmap
    {
        Bitmap bitmap;

        // three elements used for MakeGreyUnsafe
        int width;
        BitmapData bitmapData = null;
        Byte* pBase = null;

        public UnsafeBitmap(Bitmap bitmapRef)
        {
            this.bitmap = new Bitmap(bitmapRef);
            //this.bitmap = bitmapRef;
        }

        bool isARefrence = false;
        public UnsafeBitmap(ref Bitmap bitmapRef, bool isReferenced)
        {
            isARefrence = isReferenced;
            this.bitmap = bitmapRef;
        }

        //public UnsafeBitmap(int width, int height)
        //{
        //    this.bitmap = new Bitmap(width, height,
PixelFormat.Format24bppRgb);
        //}

        public void Dispose()
        {
            if (isARefrence == false)
            {
                bitmap.Dispose();
            }
            GC.Collect();
        }

        public Bitmap Bitmap
```

```
        {
            get
            {
                return (bitmap);
            }
        }

        private Point PixelSize
        {
            get
            {
                GraphicsUnit unit = GraphicsUnit.Pixel;
                RectangleF bounds = bitmap.GetBounds(ref unit);

                return new Point((int)bounds.Width, (int)bounds.Height);
            }
        }

        public void LockBitmap()
        {
            GraphicsUnit unit = GraphicsUnit.Pixel;
            RectangleF boundsF = bitmap.GetBounds(ref unit);
            Rectangle bounds = new Rectangle((int)boundsF.X,
          (int)boundsF.Y,
          (int)boundsF.Width,
          (int)boundsF.Height);

            // Figure out the number of bytes in a row
            // This is rounded up to be a multiple of 4
            // bytes, since a scan line in an image must always be a multiple
of 4 bytes
            // in length.
            width = (int)boundsF.Width * sizeof(PixelData);
            if (width % 4 != 0)
            {
                width = 4 * (width / 4 + 1);
            }
            bitmapData =
        bitmap.LockBits(bounds, ImageLockMode.ReadWrite,
PixelFormat.Format24bppRgb);

            pBase = (Byte*)bitmapData.Scan0.ToPointer();
        }

        public PixelData GetPixel(int x, int y)
        {
            PixelData returnValue = *PixelAt(x, y);
            //PixelData returnValue;
            return returnValue;
            //return *((PixelData*)(pBase + y * width + x *
sizeof(PixelData)));
        }

        public void SetPixel(int x, int y, PixelData colour)
        {
            PixelData* pixel = PixelAt(x, y);
            *pixel = colour;
```



```
        }

        public void UnlockBitmap()
        {
            bitmap.UnlockBits(bitmapData);
            bitmapData = null;
            pBase = null;
        }
        public PixelData* PixelAt(int x, int y)
        {
            return (PixelData*)(pBase + y * width + x * sizeof(PixelData));
        }
    }
    public struct PixelData
    {
        public byte blue;
        public byte green;
        public byte red;
    }
}
```

Références:

4) http://www.laltruiste.com/document.php?url=http://www.laltruiste.com/annexe/developpement_logiciel.html
5) http://fr.wikipedia.org/wiki/Cycle_en_V

