|  |
| --- |
| Ecole des Mines ParisTech |
| Rapport d’informatique |
| Projet de Toile Virtuelle |
|  |
| **Evann Courdier** |
| **01/04/2013** |

|  |
| --- |
| [Tapez le résumé du document ici. Il s'agit généralement d'une courte synthèse du document. Tapez le résumé du document ici. Il s'agit généralement d'une courte synthèse du document.] |

# I – Description du projet :

## Nom :

Programme de dessin en mode client-serveur : « Toile Virtuelle »

## Principe :

Ce programme permet à plusieurs utilisateurs connectés en réseau de dessiner sur une « toile virtuelle ». Les utilisateurs ont en leur possession les outils traditionnels simples de logiciels de dessin tels que Paint pour dessiner et verront en temps réel l’évolution du dessin. Ils pourront en outre dialoguer via un mini-chat.

## 1 - Evolution par rapport à l’avant projet

## Fonctionnalité de l’interface utilisateur :

### Les outils de dessin

Le programme développé permet de faire du dessin grâce aux outils usuels :

Un crayon permettant de dessiner un *trait*

Des formes prédéfinies : *Cercle*, *Rectangle*, *Ligne*

Un outil *texte* permettant l’insertion de texte

Le programme permet de modifier certaines propriétés élémentaires du pinceau :

La couleur

L’épaisseur

La *police* lorsqu’il s’agit d’un texte

En outre, il est apparu intéressant de pouvoir sélectionner à la main les différents motifs dessinés afin de pouvoir gérer leur affichage et leurs propriétés, un outil de *sélection* est donc également disponible ainsi qu’un outil de *déplacement* pour modifier la position d’un groupe de motifs.

### La gestion de l’historique

Pour pouvoir gérer les motifs déjà dessinés, un historique des dessins faits est disponible dans un onglet à droite de la feuille de dessin. Celui-ci permet de sélectionner des motifs dessinés, motifs dont les contours se mettent alors à clignoter pour permettre de les visualiser. Il permet un tri par dessinateur, par type d’objet, par ordre de création, par couleur et par visibilité. D’autre part, un filtre est disponible pour filtrer un type ou un nom d’utilisateur.  
L’historique permet de voir la contribution de chacun, de modifier ou supprimer les motifs après leur création et de les masquer à l’écran.

### La gestion des utilisateurs

Chaque utilisateur s’étant connecté au dessin est enregistré et son état de connexion peut être consulté dans un onglet à gauche de la feuille de dessin. Un utilisateur peut aussi dans cet onglet choisir de masquer la contribution d’un utilisateur particulier pour mieux travailler ou si ses dessins ne lui plaisent pas.

### La communication

Pour pouvoir s’entendre sur quoi et comment dessiner, un mini-chat permet aux utilisateurs de s’envoyer des messages de texte.

### Enregistrer

Le dessin terminé peut être enregistré sous la forme d’un .png pour être réutilisé par l’utilisateur.

## Partie réseau

L’aspect réseau du projet en fait tout son intérêt. Cette partie d’une part permet l’échange des motifs et des messages de chat des utilisateurs. D’autre part, elle permet de transmettre les informations de connexion des utilisateurs pour permettre de savoir qui est connecté.

## Analyse et comparaison avec l’avant projet :

Les fonctions envisagées dans l’avant projet ont toutes été programmées. De plus, grâce aux idées apportées en entretien (avec vous), j’ai pu ajouter des fonctionnalités.  
Les fonctionnalités en plus par rapport l’avant-projet sont :

* Possibilité de supprimer plusieurs motifs à la fois
* Possibilité d’afficher/masquer un motif particulier de l’historique
* Possibilité de modifier un motif particulier de l’historique
* Possibilité de filtrer/classer l’historique
* Outils de sélection de motifs
* Outil de déplacement de motifs
* Clignotement des motifs sélectionnés
* Possibilité de dessiner des formes pleines
* Certains raccourcis clavier (ctrl + z, ctrl + sélection, échap)

Le nombre de classes prévues a été largement dépassé, mais cela s’explique en grande partie par le grand nombre (9) de classes implémentant les différentes boites de dialogue. Pour ce qui est des classes nécessaires dans le projet initial et qui n’avaient pas été prévues :

* Classe Dessinateur : Cette classe enregistre chez chaque client les dessinateurs du dessin, leur état de connexion et leur visibilité chez le client. L’utilisation de la classe DataClient destinée initialement à cet usage n’était ni judicieuse car elle contenait (en attribut) aussi des sockets et des flux d’entrée/sortie, ni possible car elle n’était pas serializable donc impossible à faire transiter via le réseau.
* Classe EcouteClavier : Cette classe écoute tout ce qui passe par le clavier car elle a été créé par « Toolkit.getDefaultToolkit().addAWTEventListener ». Elle n’était pas prévue initialement car l’écoute était censée être faite à l’aide de différents KeyListeners, qui ne recevaient malheureusement rien lors de leur implémentation du fait du grand nombre de panel imbriqués pouvant chacun avoir le focus clavier. Elle permet en autre l’utilisation de la touche suppr, échap, ctrl, …
* Classe OngletDessinateurs et OngletHistorique : elles héritent de JScrollPane/JPanel et ont la particularité de toute deux contenir un tableau JTable pour lequel il faut définir de nombreuses méthodes particulières et des modèles de tableau. La taille de ces classes justifie leur implémentation séparée (non prévue)
* Classe ModeleTableauDessinateur et Classe ModeleTableauHistorique : Ces modèles sont nécessaires pour personnaliser l’affichage dans les Jtables et permettre l’interaction avec l’utilisateur. La non connaissance du fonctionnement des JTables a empêché leur anticipation.

Pour ce qui est des classes rajoutées suite aux entretiens et en plus par rapport à l’avant projet :

* L’ensemble des Boites de dialogues : pour changer la couleur, l’épaisseur, la police ; pour modifier rectangle, cercle, ligne, texte, trait (outil feutre) ; pour se connecter.
* La classe clignotementPointille : Cette classe est très courte et est runnable. Elle permet de lancer une fonction (en tant que thread séparé) qui change la valeur d’un booléen toutes les 0.5 sec.

D’autre part, de nombreuses méthodes ont été rajoutées dans différentes classes par rapport à ce qui avait été prévu pour implémenter les fonctionnalités proposées en entretien, notamment dans le package forme qui regroupe la classe Motif et toutes ses filles.  
On y a rajouté les méthodes :

* clone : renvoie une copie du motif (et non une référence)
* estIntersecteParRect : renvoie un booléen qui indique si un rectangle donné en paramètre coupe le motif en question. Utile pour la sélection (rectangle).
* deplacer : déplace le motif.

… ainsi que d’autres fonctions de détermination d’intersections en fonction du motif qui servent à la méthode « estIntersecteParRect ».  
Des méthodes ont été modifié dans la classe DataClient (appelé « Client » dans l’avant-projet) et qui surcharge la méthode « envoyer » pour pouvoir l’utiliser simplement dans tout les besoins d’envois.

Il est à noter que le programme Serveur a autant de classe que prévu initialement, on aura simplement complété la classe ServeurReception si besoin.

## 2 – Analyse de quelques classes particulières

A – Classe Image dessin

Cette classe créé une image (de type BufferedImage) qu’elle enregistre en attribut sous le nom d’*imageActuelle,* image qui est ensuite utilisée par la classe Canevas qui n’a qu’à la coller pour avoir le dessin actuel.

Pour cela, la classe dispose d’une méthode créerImageDepuis(int idMotif) qui prend comme paramètre le numéro du motif à partir duquel on doit dessiner et qui créé l’image actuelle à utiliser. Lors de la création de l’image actuelle, cette classe enregistre des images intermédiaires à intervalle de motifs réguliers qui peuvent être réutilisées lors des suppressions de motifs par exemple.

Elle dispose aussi d’une méthode mettreAJourImage qui rajoute un motif sur l’image actuelle et d’une méthode undo qui supprime les motifs et appelle la fonction creerImage.

B – Classe Canevas

Cette classe récupère les dessins de l’utilisateur, les envoie au serveur, et peint le canevas. Sa méthode paintComponent est intéressante par sa simplicité (si on omet la gestion du clignotement) car elle ne fait que peindre l’image et si besoin peindre le motif que l’on est en train de dessiner.

L’autre partie intéressante est la gestion du clignotement qui utilise un thread séparé qui change la valeur d’un booléen de true à false toute les 500 ms. Dans canevas, on réveille si besoin ce thread et on dessine les composants sélectionnés en pointillé si le booléen est à true.

C – Classe EcouteClavier

On remarquera simplement ici le fait que l’utilisation d’une telle classe héritant de AWTEventListener était nécessaire pour écouter les événements du clavier.

D – Classe OngletHistorique

Cette classe affiche l’onglet de gestion de l’historique et a demandé un gros travail de gestion de tableau notamment sur la gestion des index.

On pourra remarquer la méthode *synchroniser(int [] num)* qui, pour garder en mémoire les éléments sélectionnés lors d’une synchronisation entre la liste des motifs et la liste du tableau, se révèle être plus complexe qu’on ne pourrait l’imaginer. On se reportera aux commentaires du code pour plus d’explications sur son fonctionnement.

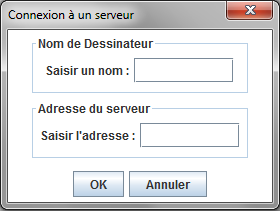
E – Classe DataClient

Cette classe enregistre les données des clients, à savoir leur nom et les sockets ouverts avec eux. Il est important de n’ouvrir qu’un flux de sortie pour éviter les conflits et c’est ce que permet d’une part leur mise en mémoire comme attribut dans la classe client et d’autre part la centralisation des méthodes envoyer qui ne sont présentes que dans la classe DataClient. Cela empêche les conflits et autres problèmes d’envoi.

F – Classes du package forme

Les classes définissant des formes pouvant être dessinées possèdent toutes une méthode clone qui renvoie une copie de l’objet, nécessaire à certains endroit, notamment pour le clignotement. Elles ont aussi une méthode qui gère l’intersection avec un rectangle (de sélection), méthode pouvant être légèrement complexe. L’intersection entre un rectangle de sélection et une ligne a été la plus complexe à programmer par le fait que si n’importe quelle portion de la ligne est comprise dans le rectangle, il faut la compter comme sélectionnée. Un peu de mathématique élémentaire résout cependant vite le problème.

# II - Notice d’utilisation

  
*Fonctionnement :* Pour fonctionner, le programme client a besoin d’être connecté à un serveur. Une fois connecté, il reçoit les différentes données du serveur et on peut commencer à dessiner en utilisant les outils disponibles sur la barre de gauche.

I) Connexion : Cliquer sur le bouton « Se Connecter ». Rentrer un nom et l’adresse IP du serveur de dessin (127.0.0.1 en cas de serveur local)

Figure - Fenêtre de connexion

II) Utilisation :

* Il suffit ensuite d’utiliser les boutons pour choisir l’outil avec lequel dessiner. On maintient le clic appuyé pour garder « la mine » appuyée.
* Parcourir les onglets pour voir les possibilités
* On peut utiliser certaines touches du clavier :
  + Echap : Annule la sélection
  + Suppr : Supprime les motifs sélectionnés
  + Ctrl + sélection : complète la sélection
  + Ctrl + z : Annule le dernier motif dessiné par le client
* Enregistrer enregistre une image au format .png (aucune boite de dialogue ne s’affiche : pas assez de temps pour tout faire) dans le répertoire du projet.

III) Déconnexion : appuyer sur le bouton « Se déconnecter ».

# III – Sources

Je ne donne ici que les sources des classes principales entrant en jeu. Les classes correspondant aux boites de dialogues, simples sur le principe, ne sont pas présentées ici.

### Coté Client

Classe MainClient : classe qui lance l’affichage graphique.

**package** ProgrammeClient;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.List;

**import** ProgrammeServeur.DataClient;

**public** **class** MainClient {

**public** **static** DataClient *client*;

**public** **static** List<Dessinateur> *listeDessinateur* = Collections.*synchronizedList*(**new** ArrayList<Dessinateur>());

**public** **static** Fenetre *fenetre*;

**public** **static** **void** main(String[] args) {

System.*out*.print("Initialisation de l'interface graphique...");

*client* = **new** DataClient();

*client*.setConnecte(**false**);

*fenetre* = **new** Fenetre();

System.*out*.print(" ok !");

}

}

Classe Fenetre : classe qui gère toute l’interface graphique

**package** ProgrammeClient;

**import** java.awt.\*;

**import** java.util.\*;

**import** javax.swing.\*;

**import** forme.\*;

@SuppressWarnings("serial")

**public** **class** Fenetre **extends** JFrame {

**private** JPanel contenant = **new** JPanel();

JTabbedPane menuDroite = **new** JTabbedPane();

JPanel menuHaut = **new** JPanel();

JPanel menuGauche = **new** JPanel();

JPanel panneauBas = **new** JPanel(**new** BorderLayout());

JPanel panneauCentre = **new** JPanel(**new** BorderLayout());

JPanel propriete = **new** JPanel(**new** GridLayout(1,2));

JPanel option = **new** JPanel(**new** GridLayout(1,4));

JPanel barreEtat = **new** JPanel(**new** BorderLayout());

Canevas canevas = **new** Canevas(**this**);

BufferedImage imageIcone = **new** BufferedImage(15, 15, BufferedImage.*TYPE\_INT\_ARGB*);

ImageIcon iconeCouleur = **new** ImageIcon(imageIcone);

JButton bDeplacer = **new** JButton("Deplacer");

JButton bSelection = **new** JButton("Selection");

JButton bFeutre = **new** JButton("Feutre");

JButton bLigneDroite = **new** JButton("Ligne");

JButton bRectangle = **new** JButton("Rectangle");

JButton bCercle = **new** JButton("Cercle");

JButton bTexte = **new** JButton("Texte");

JButton bConnexion = **new** JButton("Se connecter à un serveur");

JButton bSynchroniser = **new** JButton("Synchroniser");

JButton bEnregistrer = **new** JButton("Enregistrer");

JButton bAnnuler = **new** JButton("Annuler");

JButton bCouleur = **new** JButton("Couleur");

JButton bTaille = **new** JButton("Epaisseur");

JButton bPolice = **new** JButton("Police");

JLabel lBarreEtat1 = **new** JLabel(" FEUTRE", SwingConstants.*LEFT*);

JLabel lBarreEtat2 = **new** JLabel("(0;0) ", SwingConstants.*RIGHT*);

OngletDessinateur ongletDessinateurs = **new** OngletDessinateur(**new** JTable());

OngletHistorique ongletHistorique = **new** OngletHistorique(**new** JTable());

**private** JTextArea historiqueChat = **new** JTextArea("Chat");

**private** JScrollPane historiqueChatScroll = **new** JScrollPane(historiqueChat);

**private** JTextArea entreeChat = **new** JTextArea();

**private** JScrollPane entreeChatScroll = **new** JScrollPane(entreeChat);

**private** JSplitPane splitChat = **new** JSplitPane(JSplitPane.*VERTICAL\_SPLIT*, **true**, historiqueChatScroll, entreeChatScroll);

JPopupMenu jpm = **new** JPopupMenu();

**private** JMenuItem remplirForme = **new** JMenuItem("Forme pleine");

**private** JMenuItem contourForme = **new** JMenuItem("Contour uniquement");

**private** **boolean** plein = **false**;

**private** **float** epaisseur = 2.0f;

**private** Font police = **new** Font("Arial", Font.*PLAIN*, 12);

**private** TypeMotif motifSelectionne = TypeMotif.*FEUTRE*;

**final** Dimension dimFenetreInit = **new** Dimension(1070, 700);

EcouteClavier ecouteur;

**public** Fenetre(){

addWindowListener(**new** EcouterFermetureFenetre());

ecouteur = **new** EcouteClavier(**this**);

Toolkit.*getDefaultToolkit*().addAWTEventListener(ecouteur, AWTEvent.*KEY\_EVENT\_MASK*);

**this**.setTitle("Toile Virtuelle - V1");

**this**.setPreferredSize(dimFenetreInit);

**this**.setMinimumSize(**new** Dimension(650,500));

**this**.setLocationRelativeTo(**null**);

**this**.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

contenant.setBackground(Color.*white*);

contenant.setLayout(**new** BorderLayout());

remplirForme.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent arg0) {

setPlein(**true**);

}

});

contourForme.addActionListener(**new** ActionListener(){

**public** **void** actionPerformed(ActionEvent arg0) {

setPlein(**false**);

}

});

jpm.add(remplirForme);

jpm.add(contourForme);

canevas.setPreferredSize(Canevas.*getDimensionCanevas*());

panneauCentre.add(canevas);

contenant.add(panneauCentre, BorderLayout.*CENTER*);

menuGauche.setLayout(**new** GridLayout(7, 1));

menuGauche.add(bDeplacer);

menuGauche.add(bSelection);

menuGauche.add(bFeutre);

menuGauche.add(bLigneDroite);

menuGauche.add(bRectangle);

menuGauche.add(bCercle);

menuGauche.add(bTexte);

contenant.add(menuGauche, BorderLayout.*WEST*);

option.add(bConnexion);

option.add(bSynchroniser);

option.add(bEnregistrer);

option.add(bAnnuler);

propriete.add(bCouleur);

propriete.add(bTaille);

//propriete.setPreferredSize(new Dimension(this.getWidth()/7\*2, 30));

menuHaut.setPreferredSize(**new** Dimension(**this**.getWidth(), 30));

menuHaut.setLayout(**new** GridBagLayout());

GridBagConstraints gbc = **new** GridBagConstraints();

//On positionne la case de départ du composant

gbc.gridx = 0;

gbc.gridy = 0;

//La taille en hauteur et en largeur

gbc.gridheight = 1;

gbc.gridwidth = 4;

gbc.fill = GridBagConstraints.*HORIZONTAL*;;

gbc.weightx = 0.5;

gbc.insets = **new** Insets(0,3,0,3);

menuHaut.add(option, gbc);

//---------------------------------------------

gbc.fill = GridBagConstraints.*HORIZONTAL*;

gbc.gridwidth = 2;

gbc.gridx = 4;

gbc.weightx = 0.5;

gbc.insets = **new** Insets(0,0,0,3);

menuHaut.add(propriete, gbc);

//---------------------------------------------

gbc.gridwidth = GridBagConstraints.*REMAINDER*;

gbc.gridx = 6;

gbc.weightx = 0.5;

gbc.insets = **new** Insets(0,0,0,3);

menuHaut.add(bPolice, gbc);

//---------------------------------------------

contenant.add(menuHaut, BorderLayout.*NORTH*);

menuDroite.add("Dessinateurs", ongletDessinateurs);

menuDroite.add("Historique", ongletHistorique);

menuDroite.setPreferredSize(**new** Dimension(270, 400));

menuDroite.setBorder(BorderFactory.*createLineBorder*(Color.*black*));

contenant.add(menuDroite, BorderLayout.*EAST*);

Border raisedbevel = BorderFactory.*createRaisedBevelBorder*();

Border loweredbevel = BorderFactory.*createLoweredBevelBorder*();

Border compound = BorderFactory.*createCompoundBorder*(

raisedbevel, loweredbevel);

barreEtat.add(lBarreEtat1, BorderLayout.*WEST*);

barreEtat.add(lBarreEtat2, BorderLayout.*EAST*);

barreEtat.setBorder(compound);

panneauCentre.add(barreEtat, BorderLayout.*SOUTH*);

historiqueChat.setEditable(**false**);

entreeChat.addKeyListener(**new** entreeChatListener(entreeChat));

splitChat.setPreferredSize(**new** Dimension(900, 150));

splitChat.setDividerLocation(150\*3/5);

panneauBas.add(splitChat, BorderLayout.*CENTER*);

contenant.add(panneauBas, BorderLayout.*SOUTH*);

bCercle.addActionListener(**new** BoutonMenuGaucheListener(**this**));

bRectangle.addActionListener(**new** BoutonMenuGaucheListener(**this**));

bLigneDroite.addActionListener(**new** BoutonMenuGaucheListener(**this**));

bTexte.addActionListener(**new** BoutonMenuGaucheListener(**this**));

bFeutre.addActionListener(**new** BoutonMenuGaucheListener(**this**));

bSelection.addActionListener(**new** BoutonMenuGaucheListener(**this**));

bDeplacer.addActionListener(**new** BoutonMenuGaucheListener(**this**));

bFeutre.addMouseListener(**new** SourisMenuGaucheListener(**this**));

bRectangle.addMouseListener(**new** SourisMenuGaucheListener(**this**));

bCercle.addMouseListener(**new** SourisMenuGaucheListener(**this**));

bConnexion.addActionListener(**new** BoutonMenuHautListener(**this**));

bSynchroniser.addActionListener(**new** BoutonMenuHautListener(**this**));

bAnnuler.addActionListener(**new** BoutonMenuHautListener(**this**));

bEnregistrer.addActionListener(**new** BoutonMenuHautListener(**this**));

bCouleur.addActionListener(**new** BoutonMenuHautListener(**this**));

bTaille.addActionListener(**new** BoutonMenuHautListener(**this**));

bPolice.addActionListener(**new** BoutonMenuHautListener(**this**));

actualiserIconeCouleur();

bCouleur.setIcon(iconeCouleur);

modeDeconnecte();

**this**.setContentPane(contenant);

**this**.pack();

**this**.setVisible(**true**);

}

**public** TypeMotif getMotifSelectionne() {

**return** motifSelectionne;

}

**public** **void** setMotifSelectionne(TypeMotif motifSelectionne) {

**this**.motifSelectionne = motifSelectionne;

}

**public** Canevas getCanevas() {

**return** canevas;

}

**public** **void** afficherCanevas() {

canevas.setVisible(**true**);

}

**public** **void** masquerCanevas() {

canevas.setVisible(**false**);

}

**public** **void** actualiserIconeCouleur(){

Graphics g = imageIcone.createGraphics();

Graphics2D g2d = (Graphics2D) g;

g2d.setRenderingHint(RenderingHints.*KEY\_ANTIALIASING*,RenderingHints.*VALUE\_ANTIALIAS\_ON*);

g2d.setColor(canevas.getCouleur());

g2d.fillOval(0, 0, 15, 15);

}

**public** **void** ajouterMessageChat(String message){

historiqueChat.setText(historiqueChat.getText() + "\n" + message);

historiqueChat.setCaretPosition(historiqueChat.getText().length());

}

**public** **void** modeConnecte(){ // Gere uniquement l'AFFICHAGE graphique lors de la connexion

afficherCanevas();

bSynchroniser.setEnabled(**true**);

bEnregistrer.setEnabled(**true**);

bAnnuler.setEnabled(**true**);

bCouleur.setEnabled(**true**);

bTaille.setEnabled(**true**);

bPolice.setEnabled(**true**);

historiqueChat.setText("Chat");

entreeChat.setEnabled(**true**);

menuDroite.setEnabled(**true**);

bConnexion.setText("Se d\351connecter");

}

**public** **void** modeDeconnecte(){ // Gere uniquement l'AFFICHAGE graphique lors de la deconnexion

masquerCanevas();

bSynchroniser.setEnabled(**false**);

bEnregistrer.setEnabled(**false**);

bAnnuler.setEnabled(**false**);

bCouleur.setEnabled(**false**);

bTaille.setEnabled(**false**);

bPolice.setEnabled(**false**);

historiqueChat.setText(" ");

entreeChat.setEnabled(**false**);

menuDroite.setEnabled(**false**);

bConnexion.setText("Se connecter à un serveur");

}

/\*\*

\* **@return** the plein

\*/

**public** **boolean** isPlein() {

**return** plein;

}

/\*\*

\* **@param** plein the plein to set

\*/

**public** **void** setPlein(**boolean** plein) {

**this**.plein = plein;

lBarreEtat1.setText(" " + **this**.getMotifSelectionne().toString() + ((**this**.getMotifSelectionne() == TypeMotif.*RECTANGLE* || **this**.getMotifSelectionne() == TypeMotif.*CERCLE* || **this**.getMotifSelectionne() == TypeMotif.*FEUTRE*)? (**this**.isPlein() ? " (Plein)" : " (Contour)") :""));

}

/\*\*

\* **@return** the epaisseur

\*/

**public** **float** getEpaisseur() {

**return** epaisseur;

}

/\*\*

\* **@param** epaisseur the epaisseur to set

\*/

**public** **void** setEpaisseur(**float** epaisseur) {

**this**.epaisseur = epaisseur;

}

/\*\*

\* **@return** the police

\*/

**public** Font getPolice() {

**return** police;

}

/\*\*

\* **@param** police the police to set

\*/

**public** **void** setPolice(Font police) {

**this**.police = police;

}

**public** JTextArea getEntreeChat() {

**return** entreeChat;

}

**public** JTextArea getHistoriqueChat() {

**return** historiqueChat;

}

}

**class** BoutonMenuGaucheListener **implements** ActionListener{

Fenetre fenetre;

**public** BoutonMenuGaucheListener(Fenetre f){

**this**.fenetre = f;

}

**public** **void** actionPerformed(ActionEvent a) {

**if**(a.getSource() == fenetre.bCercle){

fenetre.setMotifSelectionne(TypeMotif.*CERCLE*);

}

**else** **if**(a.getSource() == fenetre.bLigneDroite){

fenetre.setMotifSelectionne(TypeMotif.*LIGNEDROITE*);

}

**else** **if**(a.getSource() == fenetre.bRectangle){

fenetre.setMotifSelectionne(TypeMotif.*RECTANGLE*);

}

**else** **if**(a.getSource() == fenetre.bTexte){

fenetre.setMotifSelectionne(TypeMotif.*TEXTE*);

}

**else** **if**(a.getSource() == fenetre.bFeutre){

fenetre.setMotifSelectionne(TypeMotif.*FEUTRE*);

}

**else** **if**(a.getSource() == fenetre.bSelection){

fenetre.setMotifSelectionne(TypeMotif.*SELECTION*);

}

**else** **if**(a.getSource() == fenetre.bDeplacer){

fenetre.setMotifSelectionne(TypeMotif.*DEPLACER*);

}

fenetre.setPlein(**false**);

fenetre.lBarreEtat1.setText(" " + fenetre.getMotifSelectionne().toString() + ((fenetre.getMotifSelectionne() == TypeMotif.*RECTANGLE* || fenetre.getMotifSelectionne() == TypeMotif.*CERCLE* || fenetre.getMotifSelectionne() == TypeMotif.*FEUTRE*)? " (Contour)": ""));

}

}

**class** BoutonMenuHautListener **implements** ActionListener{

Fenetre fenetre;

**public** BoutonMenuHautListener(Fenetre f){

**this**.fenetre = f;

}

**public** **void** actionPerformed(ActionEvent a) {

**if**(a.getSource() == fenetre.bConnexion){

/\*JOptionPane jop1 = new JOptionPane();

jop1.showMessageDialog(null, InitClient.client.isConnecte(), "Information", JOptionPane.INFORMATION\_MESSAGE);//\*/

**if**(!MainClient.*client*.isConnecte()){

//\*

BdDialogueConnexion jop = **new** BdDialogueConnexion(**null**, "Connexion à un serveur", **true**);

String [] donnees = jop.afficher();

//\*/String [] donnees = {"127.0.0.1", "Toto"};

**if**(donnees != **null**){

//System.out.println("adresse (" + donnees[0] + ")");

(**new** Thread(**new** Connexion(donnees[0], donnees[1]))).start();

}

}

**else** **if**(MainClient.*client*.isConnecte()){

JOptionPane jop = **new** JOptionPane();

**int** option = jop.*showConfirmDialog*(**null**, "Voulez-vous vous déconnecter du serveur ?", "Deconnexion du serveur", JOptionPane.*YES\_NO\_OPTION*, JOptionPane.*QUESTION\_MESSAGE*);

**if**(option == JOptionPane.*YES\_OPTION*){

MainClient.*client*.envoyer("Bye");

}

}

}

**else** **if**(a.getSource() == fenetre.bSynchroniser){

MainClient.*client*.envoyer("synchro");

MainClient.*client*.envoyer("synchroClient");

}

**else** **if**(a.getSource() == fenetre.bEnregistrer){

ImageDessin.*enregisterImageActuelle*();

}

**else** **if**(a.getSource() == fenetre.bCouleur){

bdDialogueCouleur jop = **new** bdDialogueCouleur(fenetre, "Choisissez la couleur", **true**);

Color couleur = jop.afficher();

**if**(couleur != **null**)

fenetre.getCanevas().setCouleur(couleur);

fenetre.actualiserIconeCouleur();

}

**else** **if**(a.getSource() == fenetre.bTaille){

BdDialogueEpaisseur jop = **new** BdDialogueEpaisseur(fenetre, "Choisissez la taille", **true**);

fenetre.setEpaisseur((**float**)jop.afficher());

}

**else** **if**(a.getSource() == fenetre.bPolice){

BdDialoguePolice jop = **new** BdDialoguePolice(fenetre, "Choisissez la taille", **true**);

fenetre.setPolice(jop.afficher());

}

**else** **if**(a.getSource() == fenetre.bAnnuler){

**synchronized**(ImageDessin.*listeMotif*){

Motif motifASupprimer;

ListIterator<Motif> it = ImageDessin.*listeMotif*.listIterator(ImageDessin.*listeMotif*.size());

**while**(it.hasPrevious()){

motifASupprimer = it.previous();

**if**(motifASupprimer.getNumUser() == MainClient.*client*.getNumuser()){

MainClient.*client*.envoyer("u" + it.nextIndex());

**break**;

}

}

}

}

}

}

**class** entreeChatListener **implements** KeyListener{

**private** JTextArea zoneTexte;

**public** entreeChatListener(JTextArea zoneTexte) {

**this**.zoneTexte = zoneTexte;

}

@Override

**public** **void** keyPressed(KeyEvent event) {

}

@Override

**public** **void** keyReleased(KeyEvent event) {

**if**(event.getKeyCode() == 10 && zoneTexte.getText().length() > 0){ //Si on appuie sur Entree et qu'on a ecrit qqchose

**if**(zoneTexte.getText().substring(0, zoneTexte.getText().length()-1).contains("\n"))

zoneTexte.setText(zoneTexte.getText().replace("\n".charAt(0), ' '));//On enlève les '\n' qui posent problème à l'envoi

MainClient.*client*.envoyer("c" + zoneTexte.getText().substring(0, zoneTexte.getText().length()-1)); // On envoie

zoneTexte.setText("");

}

}

@Override

**public** **void** keyTyped(KeyEvent event) {

}

}

**class** SourisMenuGaucheListener **extends** MouseAdapter{

Fenetre fenetre;

**public** SourisMenuGaucheListener(Fenetre f) {

**this**.fenetre = f;

}

@Override

**public** **void** mouseClicked(MouseEvent event) {

**if**(event.getButton() == MouseEvent.*BUTTON3*){

((AbstractButton) event.getSource()).doClick();

fenetre.jpm.show((Component) event.getSource(), event.getX(), event.getY());

}

}

}

**class** EcouterFermetureFenetre **extends** WindowAdapter{

**public** **void** windowClosing(WindowEvent w) {

**if** (MainClient.*client*.isConnecte())

MainClient.*client*.envoyer("Bye");

**long** t = (**long**) System.*currentTimeMillis*();

**while** (MainClient.*client*.isConnecte()

&& System.*currentTimeMillis*() - t < 1500) {

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

MainClient.*fenetre*.dispose(); // libération des ressources utilisées par

// la frame

System.*exit*(0);

}

}

Classe Canevas : Gère les dessins que l’on peut faire sur le panneau central

**package** ProgrammeClient;

**import** java.awt.\*;

@SuppressWarnings("serial")

**public** **class** Canevas **extends** JPanel{

**private** **static** Dimension *dimensionCanevas* = **new** Dimension(500, 400);

**public** **static** Fenetre *fenetre*;

**public** **boolean** clicEnCours = **false**, clignotement = **true**;

**private** Point point1;

**private** Motif motifEnCours;

**private** Color Couleur = Color.*black*;

**private** ClignotementPointille threadClignotement = **new** ClignotementPointille();

**public** Canevas(Fenetre f){

**super**();

*fenetre* = f;

addMouseListener(**new** EcouteurSourisCanevas(**this**));

addMouseMotionListener(**new** EcouteurMouvementSourisCanevas(**this**));

**this**.setVisible(**false**);

}

**public** **void** paintComponent(Graphics g) {

g.setColor(Color.*gray*);

g.fillRect(0, 0, **this**.getWidth(), **this**.getHeight());

g.drawImage(ImageDessin.*imageActuelle*.img, 0, 0, **null**);

**if**(clicEnCours){

**if**(*fenetre*.getMotifSelectionne() == TypeMotif.*SELECTION*){

motifEnCours.setCouleur(Color.*red*);// couleur du rectangle de selection

motifEnCours.setTaille(4f); // taille du trait du rectangle de selection

motifEnCours.setSelection(**true**);

}

**else**

motifEnCours.setCouleur(Couleur);

motifEnCours.dessinerSur(g);

}

/\*\*

\* Gestion du clignotement des éléments séléctionnés

\*/

//Si des colonnes sont selectionnés et que la variable clignotement (mis à jour par le thread) est à true, on va dessiner des pointillés

**if**(OngletHistorique.*ongletActuel*.getTableau().getSelectedRowCount() > 0 && clignotement){

//Si le thread n'est pas en marche, on le lance.

**if**(threadClignotement.getState() != java.lang.Thread.State.*RUNNABLE* && threadClignotement.getState() != java.lang.Thread.State.*TIMED\_WAITING* ){

threadClignotement = **new** ClignotementPointille();

threadClignotement.start();

}

// On récupère le tableau Jtable

JTable tableau = OngletHistorique.*ongletActuel*.getTableau();

// On prend dans le Jtable le numero des motifs à redessiner en pointillés -> on enregistre ces numéros dans un tableu d'int

**int** [] tableauADessine = **new** **int** [tableau.getSelectedRowCount()];

**for**(**int** i = 0 ; i < tableau.getSelectedRowCount() ; i++)

tableauADessine[i] = tableau.convertRowIndexToModel(tableau.getSelectedRows()[i]);

//On le trie, ici pas très utile

Arrays.*sort*(tableauADessine);

//Pour chaque motif dont le rang dans la liste listeMotif est dans le tableau d'int (cad pour chaque motif sélectionné), on demande à le redessiner en "mode selectionné", cad en pratique en pointillés

**for**(**int** i : tableauADessine){

Motif motifSelectionne = (Motif) ImageDessin.*listeMotif*.get(i).clone();

motifSelectionne.setCouleur(**new** Color((**int**)((255-motifSelectionne.getCouleur().getRed())\*0.8),(**int**)((255-motifSelectionne.getCouleur().getGreen())\*0.8),(**int**)((255-motifSelectionne.getCouleur().getBlue())\*0.8)));

motifSelectionne.setSelection(**true**);

motifSelectionne.dessinerSur(g);

}

}

//Fin de la gestion du clignotement

}

**public** **void** masquerCanevas(){

**this**.setVisible(**false**);

}

**public** Point getPoint1() {

**return** point1;

}

**public** **void** setPoint1(Point point1) {

**this**.point1 = point1;

}

**public** Motif getMotifEnCours() {

**return** motifEnCours;

}

**public** **void** setMotifEnCours(Motif motifEnCours) {

**this**.motifEnCours = motifEnCours;

}

/\*\*

\* **@return** the couleur

\*/

**public** Color getCouleur() {

**return** Couleur;

}

/\*\*

\* **@param** couleur the couleur to set

\*/

**public** **void** setCouleur(Color couleur) {

Couleur = couleur;

}

/\*\*

\* **@return** the dimensionCanevas

\*/

**public** **static** Dimension getDimensionCanevas() {

**return** *dimensionCanevas*;

}

/\*\*

\* **@param** dimensionCanevas the dimensionCanevas to set

\*/

**public** **static** **void** setDimensionCanevas(Dimension dimensionCanevas) {

**int** w = 0, h = 0;

w = MainClient.*fenetre*.getCanevas().getWidth() - dimensionCanevas.width;

h = MainClient.*fenetre*.getCanevas().getHeight() - (dimensionCanevas.height);

*fenetre*.setSize((**int**) (*fenetre*.getWidth() - w), (**int**) (*fenetre*.getHeight()-h));

*fenetre*.validate();

Canevas.*dimensionCanevas* = dimensionCanevas;

ImageDessin.*setDimension*(dimensionCanevas);

}

}

**class** EcouteurMouvementSourisCanevas **implements** MouseMotionListener/\*Adapter\*/{

**private** Canevas canevas;

**public** EcouteurMouvementSourisCanevas(Canevas c){

**this**.canevas = c;

}

@Override

**public** **void** mouseDragged(MouseEvent e) {

**if**(canevas.clicEnCours && MainClient.*client*.isConnecte()){

**if**(Canevas.*fenetre*.getMotifSelectionne() == TypeMotif.*CERCLE*){

canevas.setMotifEnCours(**new** Cercle(canevas.getPoint1(), (**int**)Math.*sqrt*((e.getX() - canevas.getPoint1().x)\*(e.getX() - canevas.getPoint1().x) + (e.getY() - canevas.getPoint1().y)\*(e.getY() - canevas.getPoint1().y)), Canevas.*fenetre*.isPlein()));

}

**else** **if**(Canevas.*fenetre*.getMotifSelectionne() == TypeMotif.*LIGNEDROITE*){

canevas.setMotifEnCours(**new** LigneDroite(canevas.getPoint1(), e.getPoint()));

}

**else** **if**(Canevas.*fenetre*.getMotifSelectionne() == TypeMotif.*RECTANGLE*){

Point point1 = **new** Point(Math.*min*(canevas.getPoint1().x, e.getPoint().x), Math.*min*(canevas.getPoint1().y, e.getPoint().y));

Point point2 = **new** Point(Math.*max*(canevas.getPoint1().x, e.getPoint().x), Math.*max*(canevas.getPoint1().y, e.getPoint().y));

canevas.setMotifEnCours(**new** Rectangle(point1, point2, Canevas.*fenetre*.isPlein()));

}

**else** **if**(Canevas.*fenetre*.getMotifSelectionne() == TypeMotif.*FEUTRE*){

**if**(!(canevas.getMotifEnCours() **instanceof** Feutre)){

canevas.setMotifEnCours(**new** Feutre());

}

((forme.Feutre) (canevas.getMotifEnCours())).ajouterPoint(e.getPoint());

}

**else** **if**(Canevas.*fenetre*.getMotifSelectionne() == TypeMotif.*SELECTION*){

Point point1 = **new** Point(Math.*min*(canevas.getPoint1().x, e.getPoint().x), Math.*min*(canevas.getPoint1().y, e.getPoint().y));

Point point2 = **new** Point(Math.*max*(canevas.getPoint1().x, e.getPoint().x), Math.*max*(canevas.getPoint1().y, e.getPoint().y));

canevas.setMotifEnCours(**new** Rectangle(point1, point2, **false**));

}

**else** **if**(Canevas.*fenetre*.getMotifSelectionne() == TypeMotif.*DEPLACER*){

canevas.setMotifEnCours(**new** LigneDroite(canevas.getPoint1(), e.getPoint()));

}

canevas.getMotifEnCours().setTaille(MainClient.*fenetre*.getEpaisseur());

canevas.repaint();

}

MainClient.*fenetre*.lBarreEtat2.setText("(" + e.getX() + " ; " + e.getY() + ") ");

}

@Override

**public** **void** mouseMoved(MouseEvent arg0) {

MainClient.*fenetre*.lBarreEtat2.setText("(" + arg0.getX() + " ; " + arg0.getY() + ") ");

}

}

**class** EcouteurSourisCanevas **extends** MouseAdapter{

**private** Canevas canevas;

**public** EcouteurSourisCanevas(Canevas c){

**this**.canevas = c;

}

@Override

**public** **void** mouseClicked(MouseEvent e) {

**if**(MainClient.*fenetre*.getMotifSelectionne() == TypeMotif.*TEXTE*){

JOptionPane jop = **new** JOptionPane();

String texte = jop.*showInputDialog*(**null**, "Texte à insérer : ", "Ecrire du texte", JOptionPane.*QUESTION\_MESSAGE*);

**if**(texte != **null** && texte != ""){

Texte txt = **new** Texte(e.getPoint(), texte, Canevas.*fenetre*.getPolice());

txt.setCouleur(canevas.getCouleur());

MainClient.*client*.envoyerMotif(txt);

}

}

}

@Override

**public** **void** mouseEntered(MouseEvent e) {

**if**(MainClient.*fenetre*.getMotifSelectionne() == TypeMotif.*TEXTE*){

MainClient.*fenetre*.getCanevas().setCursor(Cursor.*getPredefinedCursor*(Cursor.*TEXT\_CURSOR*));

}

**else** **if**(MainClient.*fenetre*.getMotifSelectionne() == TypeMotif.*DEPLACER*){

MainClient.*fenetre*.getCanevas().setCursor(Cursor.*getPredefinedCursor*(Cursor.*MOVE\_CURSOR*));

}

**else** **if**(MainClient.*fenetre*.getMotifSelectionne() == TypeMotif.*SELECTION*){

MainClient.*fenetre*.getCanevas().setCursor(Cursor.*getPredefinedCursor*(Cursor.*HAND\_CURSOR*));

}

**else**

MainClient.*fenetre*.getCanevas().setCursor(Cursor.*getPredefinedCursor*(Cursor.*DEFAULT\_CURSOR*));

}

@Override

**public** **void** mouseExited(MouseEvent e) {

canevas.clicEnCours = **false**;

canevas.repaint();

}

@Override

**public** **void** mousePressed(MouseEvent e) {

canevas.clicEnCours = **true**;

canevas.setPoint1(e.getPoint());

canevas.setMotifEnCours(**new** Motif());

}

@Override

**public** **void** mouseReleased(MouseEvent e) {

**if**(canevas.clicEnCours && MainClient.*client*.isConnecte() && canevas.getMotifEnCours().getType() != **null** && MainClient.*fenetre*.getMotifSelectionne() != TypeMotif.*SELECTION* && MainClient.*fenetre*.getMotifSelectionne() != TypeMotif.*DEPLACER*){

//Si c'est un feutre, on met plein que quand on envoie, sinon c'est incompréhensible quand on dessine

**if**((canevas.getMotifEnCours() **instanceof** Feutre))

((forme.Feutre) canevas.getMotifEnCours()).setPlein(Canevas.*fenetre*.isPlein());

canevas.getMotifEnCours().setCouleur(canevas.getCouleur());

canevas.getMotifEnCours().setNumUser(MainClient.*client*.getNumuser());

MainClient.*client*.envoyerMotif(canevas.getMotifEnCours());

}

//

**else** **if**(MainClient.*fenetre*.getMotifSelectionne() == TypeMotif.*SELECTION* && canevas.getMotifEnCours().getType() != **null**){

**int** i = 0;

**if**(!MainClient.*fenetre*.ecouteur.isCtrl())

OngletHistorique.*ongletActuel*.deselectionner();

**for**(Motif m : ImageDessin.*listeMotif*){

**if**(!m.isMasque() && m.estIntersecteParRect((Rectangle) canevas.getMotifEnCours()) && MainClient.*listeDessinateur*.get(m.getNumUser()).getVoir())

OngletHistorique.*ongletActuel*.selectionnerLigne(i);

i++;

}

canevas.repaint();

}

**else** **if**(MainClient.*fenetre*.getMotifSelectionne() == TypeMotif.*DEPLACER* && canevas.getMotifEnCours().getType() != **null**){

Object [] aEnvoyer = **new** Object [2];

JTable tableau = OngletHistorique.*ongletActuel*.getTableau();

**int** [] tableauADeplacer = **new** **int** [tableau.getSelectedRowCount()];

**for**(**int** i = 0 ; i < tableau.getSelectedRowCount() ; i++)

tableauADeplacer[i] = tableau.convertRowIndexToModel(tableau.getSelectedRows()[i]);

aEnvoyer[0] = tableauADeplacer;

aEnvoyer[1] = **new** Dimension(e.getX() - canevas.getPoint1().x,e.getY() - canevas.getPoint1().y);

**if**(tableauADeplacer.length > 0)

MainClient.*client*.envoyer("b",aEnvoyer);

canevas.repaint();

}

**else** **if**(canevas.clicEnCours && MainClient.*client*.isConnecte())

OngletHistorique.*ongletActuel*.deselectionner();

canevas.clicEnCours = **false**;

}

}

Classe ServeurReceptionClient : classe qui interprète les données reçues du serveur et appelle les méthodes nécessaires pour les traiter.

**package** ProgrammeClient;

**import** ProgrammeServeur.DataClient;

**import** forme.\*;

**public** **class** ServeurReceptionClient **implements** Runnable {

**private** DataClient client;

**private** **boolean** connect = **true**;

**private** BufferedReader in = **null**;

**private** ObjectInputStream in2 = **null**;

**public** ServeurReceptionClient(DataClient client){

**super**();

**this**.client = client;

}

**public** **void** run() {

String message = **null**;

System.*out*.println("R\351ception lanc\351e");

**try** {

in = MainClient.*client*.getInTexte();

in2 = MainClient.*client*.getInMotif();

**while**(connect){

message = in.readLine();

**if**(message == **null**)

**continue**;

System.*out*.println("\n----------\nReception (serveur) : " + message);

**if**(message.substring(0,1).equals("c")){

System.*out*.println("Message de Chat: " + message.substring(1));

MainClient.*fenetre*.ajouterMessageChat(message.substring(1));

}

**else** **if**(message.substring(0,1).equals("u") && message.substring(1,2).equals("l")){

System.*out*.print("Suppression des motifs ");

**int** [] num = (**int**[]) (in2.readObject());

**for**(**int** i = 0 ; i < num.length ; i++)

System.*out*.print("(" + num[i] + ")");

System.*out*.println();

ImageDessin.*undo*(num);

OngletHistorique.*ongletActuel*.synchroniser(num);

}

**else** **if**(message.substring(0,1).equals("u")){ // u pour undo

System.*out*.print("Suppression du motif ");

**int** num = Integer.*parseInt*(message.substring(1));

System.*out*.println("(" + num + ")");

ImageDessin.*undo*(num);

**int** [] num2 = **new** **int**[1];

num2[0] = num;

OngletHistorique.*ongletActuel*.synchroniser(num2);

}

**else** **if**(message.substring(0,1).equals("m")){

System.*out*.println("Motif");

Motif motifRecupere = (forme.Motif) (in2.readObject());

OngletHistorique.*ongletActuel*.ajouterMotif(motifRecupere);

ImageDessin.*miseAJourImage*(motifRecupere);

}

**else** **if**(message.substring(0,1).equals("b")){ // b pour bouger

System.*out*.println("D\351placement de Motifs");

**int** [] indexMotifs = (**int**[]) (in2.readObject());

Dimension dim = (Dimension) (in2.readObject());

ImageDessin.*deplacer*(dim, indexMotifs);

}

**else** **if**(message.substring(0,1).equals("k")){ // k parce que j'ai vraiment plus d'inspiration

System.*out*.println("Modification de Motifs");

List<Motif> listeMotifTemp = (**new** ArrayList<Motif>());

**int** [] indexMotifs = (**int**[]) (in2.readObject());

listeMotifTemp = (List<Motif>) (in2.readObject());

ImageDessin.*modifierMotifs*(indexMotifs,listeMotifTemp);

}

**else** **if**(message.equals("synchro")){

System.*out*.println("Synchronisation...");

synchronisation();

}

**else** **if**(message.equals("synchroClient")){

System.*out*.println("Synchronisation des clients...");

synchronisationClient();

}

**else** **if**(message.substring(0,1).equals("a")){

Dessinateur dessinateur = (Dessinateur) (in2.readObject());

**boolean** existe = **false**;

**synchronized** (MainClient.*listeDessinateur*) {

Iterator<Dessinateur> itc = MainClient.*listeDessinateur*.iterator();

**while**(itc.hasNext() && !existe){

Dessinateur dessinateur2 = itc.next();

**if**(dessinateur2.getNom().equals(dessinateur.getNom())){

dessinateur.setConnecte(**true**);

OngletDessinateur.*ongletActuel*.connecterDessinateur(dessinateur);

existe = **true**;

}

}

**if**(!existe){

MainClient.*listeDessinateur*.add(dessinateur);

OngletDessinateur.*ongletActuel*.ajouterDessinateur(dessinateur);

}

}

MainClient.*fenetre*.ajouterMessageChat(" - " + dessinateur.getNom() + " s'est connect\351 - ");

System.*out*.println(dessinateur.getNom() + " s'est connect\351");

}

**else** **if**(message.substring(0,1).equals("d")){

Dessinateur dessinateur = (Dessinateur) (in2.readObject());

**synchronized** (MainClient.*listeDessinateur*) {

Iterator<Dessinateur> itc = MainClient.*listeDessinateur*.iterator();

**while**(itc.hasNext()){

Dessinateur dessinateur2 = itc.next();

**if**(dessinateur2.getNom().equals(dessinateur.getNom())){

dessinateur.setConnecte(**false**);

MainClient.*fenetre*.ajouterMessageChat(" - " + dessinateur.getNom() + " s'est deconnect\351 - ");

OngletDessinateur.*ongletActuel*.deconnecterDessinateur(dessinateur);

}

}

}

System.*out*.println(dessinateur.getNom() + " est d\351connect\351");

}

**else** **if**(message.equals("Bye")){

System.*out*.println("Deconnexion...");

client.deconnecter();

MainClient.*fenetre*.modeDeconnecte();

connect = **false**;

}

MainClient.*fenetre*.getCanevas().repaint();

} // Fin de la boucle de reception

} **catch** (IOException e) {

System.*err*.println("Serveur deconnect\351.");

e.printStackTrace();

connect = **false**;

} **catch** (ClassNotFoundException e) {

System.*err*.println("Erreur de conversion de donn\351es");

e.printStackTrace();

connect = **false**;

}

}

**private** **void** synchronisation() **throws** ClassNotFoundException, IOException {

List<Motif> listeMotifTemp = (**new** ArrayList<Motif>());

Dimension dimensionCanevasTemp = **new** Dimension();

listeMotifTemp = (List<Motif>) (in2.readObject());

**synchronized** (ImageDessin.*listeMotif*) {

ImageDessin.*listeMotif* = listeMotifTemp;

}

dimensionCanevasTemp = (Dimension) (in2.readObject());

**synchronized** (Canevas.*getDimensionCanevas*()) {

Canevas.*setDimensionCanevas*(dimensionCanevasTemp);

}

System.*out*.println("Synchronisation Termin\351e !");

OngletHistorique.*ongletActuel*.synchroniser();

ImageDessin.*creerImage*();

}

**private** **void** synchronisationClient() **throws** ClassNotFoundException, IOException {

List<Dessinateur> listeDessinateurs = Collections.*synchronizedList*(**new** ArrayList<Dessinateur>());

listeDessinateurs =(List<Dessinateur>) (in2.readObject());

MainClient.*listeDessinateur* = listeDessinateurs;

**for**(**int** i = 0 ; i < listeDessinateurs.size() ; i++){

**if**(listeDessinateurs.get(i).getNom().equals(MainClient.*client*.getNom()))

MainClient.*client*.setNumuser(i);

}

OngletDessinateur.*ongletActuel*.synchroniser();

System.*out*.println(listeDessinateurs);

System.*out*.println("Synchronisation des Clients Termin\351e !");

}

}

Classe ImageDessin : Elle crée l’image qui est affichée à l’écran

**package** ProgrammeClient;

**import** java.awt.\*;

**import** java.awt.image.BufferedImage;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collections;

**import** java.util.Iterator;

**import** java.util.List;

**import** javax.imageio.ImageIO;

**import** javax.imageio.ImageWriter;

**import** javax.imageio.stream.ImageOutputStream;

**import** forme.\*;

**public** **class** ImageDessin {

**static** **int** *width* = MainClient.*fenetre*.getCanevas().getWidth(),

*height* = MainClient.*fenetre*.getCanevas().getHeight();

**static** **int** *freq* = 10;// freq = frequenceEnregistrementImage (ici la periode

// en fait...)

**static** ImageDessin *imageActuelle*;

**static** List<ImageDessin> *listeImage*;

**static** List<Motif> *listeMotif* = Collections

.*synchronizedList*(**new** ArrayList<Motif>());

**private** **static** **int** *nb* = 0;

BufferedImage img;

**int** id = -1; // id = numero du dernier motif dessiné dans la liste des

// motifs

**static** {

*listeImage* = **new** ArrayList<ImageDessin>();

BufferedImage imgInit = **new** BufferedImage(*width*, *height*,

BufferedImage.*TYPE\_INT\_BGR*);

Graphics g = imgInit.createGraphics();

g.setColor(Color.*white*);

g.fillRect(0, 0, *width*, *height*);

*imageActuelle* = **new** ImageDessin(imgInit, 0);

*listeImage*.add(0, **new** ImageDessin(imgInit, 0));

}

**public** ImageDessin(BufferedImage image) {

**this**.img = image;

}

**public** ImageDessin(BufferedImage image, **int** id) {

**this**.img = image;

**this**.id = id;

}

**public** **static** **void** setDimension(Dimension dimension) {

*width* = dimension.width;

*height* = dimension.height;

BufferedImage imgInit = **new** BufferedImage(*width*, *height*,

BufferedImage.*TYPE\_INT\_BGR*);

Graphics g = imgInit.createGraphics();

g.setColor(Color.*white*);

g.fillRect(0, 0, *width*, *height*);

*listeImage*.remove(0);

*listeImage*.add(0, **new** ImageDessin(imgInit, 0));

}

**public** **static** **void** miseAJourImage(Motif motif) {

**synchronized** (*listeMotif*) {

*listeMotif*.add(motif);

}

Dessinateur dessinateur = MainClient.*listeDessinateur*.get(motif

.getNumUser());

System.*out*.println(" - Reception d'un motif : " + motif.getType());

BufferedImage image = **new** BufferedImage(*width*, *height*,

BufferedImage.*TYPE\_INT\_RGB*);

Graphics g = image.createGraphics();

g.drawImage(*imageActuelle*.img, 0, 0, **null**);

**if** (dessinateur.getVoir())

motif.dessinerSur(g);

*imageActuelle* = **new** ImageDessin(image, *listeMotif*.size() - 1);

**if** (*listeMotif*.size() % *freq* == 0) {

*listeImage*.add((**int**) ((*listeMotif*.size()) / *freq*), **new** ImageDessin(

image, *listeMotif*.size() - 1));

}

*nb*++;

}

**public** **static** BufferedImage creerImageDepuis(**int** idMotif) { // idMotif = num

// du motif dans

// la liste

**int** i = 0, imgAUtilise; // i correspond au numero du motif dans la liste

BufferedImage image;

imgAUtilise = (idMotif + 1) - ((idMotif + 1) % *freq*); // Ici, on choisit

// le rang du

// motif à

// partir duquel

// on va

// redessiner

// pour utiliser

// une image

// déjà créée.

System.*out*.println("Image utilisée : " + imgAUtilise / *freq* + " ("

+ imgAUtilise + ")");

image = **new** BufferedImage(*width*, *height*, BufferedImage.*TYPE\_INT\_RGB*);

Graphics g = image.createGraphics();

g.drawImage(*listeImage*.get(imgAUtilise / *freq*).img, 0, 0, **null**);

**synchronized** (*listeMotif*) {

Iterator<Motif> iter = *listeMotif*.iterator();

**while** (iter.hasNext() && i < imgAUtilise) {

iter.next();

i++;

}

**while** (iter.hasNext()) {

Motif motif = iter.next();

Dessinateur dessinateur = MainClient.*listeDessinateur*.get(motif

.getNumUser());

**if** (dessinateur.getVoir() && !motif.isMasque())

motif.dessinerSur(g);

**if** ((i + 1) % *freq* == 0) { // Si on est multiple de la période,

// on enregistre l'image

BufferedImage imageTemp = **new** BufferedImage(*width*, *height*,

BufferedImage.*TYPE\_INT\_RGB*);

Graphics g2 = imageTemp.createGraphics();

g2.drawImage(image, 0, 0, **null**);

*listeImage*.add((**int**) ((i + 1) / *freq*), **new** ImageDessin(

imageTemp, i));

}

i++;

}

}

*imageActuelle* = **new** ImageDessin(image, i - 1);

**return** image;

}

**public** **static** BufferedImage creerImage() {

**return** *creerImageDepuis*(0);

}

**public** **static** **void** undo(**int**[] idASupprimer) {

**synchronized** (*listeMotif*) {

**for** (**int** i = 0; i < idASupprimer.length; i++) {

*listeMotif*.remove(idASupprimer[i] - i);

}

}

*creerImageDepuis*(idASupprimer[0] - 1);

*nb*++;

}

**public** **static** **void** deplacer(Dimension dim, **int**[] indexMotifs) {

Arrays.*sort*(indexMotifs);

**for** (**int** i : indexMotifs)

*listeMotif*.get(i).deplacer(dim);

*creerImageDepuis*(indexMotifs[0] - 1);

}

**public** **static** **void** undo(**int** idASupprimer) {

**synchronized** (*listeMotif*) {

*listeMotif*.remove(idASupprimer);

}

*creerImageDepuis*(idASupprimer - 1);

*nb*++;

}

**public** **static** **void** masquerMotif(**int**[] idAMasquer) {

Arrays.*sort*(idAMasquer);

**synchronized** (*listeMotif*) {

**for** (**int** i = 0; i < idAMasquer.length; i++) {

*listeMotif*.get(idAMasquer[i]).setMasque(**true**);

}

}

*creerImageDepuis*(idAMasquer[0] - 1);

}

**public** **static** **void** afficherMotif(**int**[] idAAfficher) {

Arrays.*sort*(idAAfficher);

**synchronized** (*listeMotif*) {

**for** (**int** i = 0; i < idAAfficher.length; i++) {

*listeMotif*.get(idAAfficher[i]).setMasque(**false**);

}

}

*creerImageDepuis*(idAAfficher[0] - 1);

}

**public** **static** **void** enregister(BufferedImage image) {

*enregister*(image, *nb* + "");

}

**public** **static** **void** enregister(BufferedImage image, String toto) {

Iterator writers = ImageIO.*getImageWritersByFormatName*("png");

ImageWriter writer = (ImageWriter) writers.next();

**if** (writer == **null**) {

**throw** **new** RuntimeException("PNG not supported?!");

}

ImageOutputStream out;

**try** {

out = ImageIO.*createImageOutputStream*(**new** File("Image " + toto

+ ".png"));

writer.setOutput(out);

writer.write(image);

writer.dispose();

out.close();

*nb*++;

} **catch** (IOException e) {

e.printStackTrace();

}

}

**public** **static** **void** enregisterImageActuelle() {

*enregister*(*imageActuelle*.img);

}

**public** **static** **void** modifierMotifs(**int**[] indexMotifs,

List<Motif> listeMotifTemp) {

**int** j = 0;

**synchronized** (*listeMotif*) {

**for** (**int** i : indexMotifs) {

*listeMotif*.remove(i);

*listeMotif*.add(i, listeMotifTemp.get(j));

OngletHistorique.*ongletActuel*

.modifier(i, listeMotifTemp.get(j));

j++;

}

}

*creerImageDepuis*(indexMotifs[0] - 1);

}

}

**Il y a plusieurs autres classes non présentes ici mais cela n’est pas nécessaire et alourdirait encore le rapport.**

### Coté Serveur

Classe MainServeur : Ecoute les clients potentiels

**package** ProgrammeServeur;

**import** java.awt.Dimension;

**import** java.io.IOException;

**import** java.net.InetAddress;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**import** java.util.\*;

**import** java.util.concurrent.CopyOnWriteArrayList;

**import** forme.Motif;

**public** **class** MainServer {

**static** ServerSocket *socketServeur1*;

**static** ServerSocket *socketServeur2*;

**static** List<Motif> *listeMotif* = Collections.*synchronizedList*(**new** ArrayList<Motif>());

**static** List<DataClient> *listeClient* = Collections.*synchronizedList*(**new** ArrayList<DataClient>());

**static** Dimension *dimensionCanevas* = **new** Dimension(700, 500);

**public** **static** **void** main(String[] args) {

**try** {

*socketServeur1* = **new** ServerSocket(2013);

*socketServeur2* = **new** ServerSocket(2014);

InetAddress thisIp = InetAddress.*getLocalHost*();

System.*out*.println("Adresse du serveur : " + thisIp.getHostAddress());

System.*out*.println("Serveur Texte lanc\351\nA l'\351coute du port " + *socketServeur1*.getLocalPort());

System.*out*.println("Serveur Dessin lanc\351\nA l'\351coute du port " + *socketServeur2*.getLocalPort());

**while**(**true**){

Socket socketClient = *socketServeur1*.accept();

System.*out*.println("\nConnexion d'un nouveau participant : " + socketClient.getLocalAddress());

(**new** Thread(**new** ServeurReception(socketClient))).start();

}

} **catch** (IOException e) {

System.*err*.println("Un des ports de connexion est d\351jà utilis\351 !");

}

}

}

Classe ServeurReception : Ecoute un client et retransmet ses motifs et autres changements

**package** ProgrammeClient;

**import** java.awt.Dimension;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Iterator;

**import** java.util.List;

**import** javax.swing.JOptionPane;

**import** ProgrammeServeur.DataClient;

**import** forme.\*;

**public** **class** ServeurReceptionClient **implements** Runnable {

**private** DataClient client;

**private** **boolean** connect = **true**;

**private** BufferedReader in = **null**;

**private** ObjectInputStream in2 = **null**;

**public** ServeurReceptionClient(DataClient client){

**super**();

**this**.client = client;

}

**public** **void** run() {

String message = **null**;

System.*out*.println("R\351ception lanc\351e");

**try** {

in = MainClient.*client*.getInTexte();

in2 = MainClient.*client*.getInMotif();

**while**(connect){

message = in.readLine();

**if**(message == **null**)

**continue**;

System.*out*.println("\n----------\nReception (serveur) : " + message);

**if**(message.substring(0,1).equals("c")){

System.*out*.println("Message de Chat: " + message.substring(1));

MainClient.*fenetre*.ajouterMessageChat(message.substring(1));

}

**else** **if**(message.substring(0,1).equals("u") && message.substring(1,2).equals("l")){

System.*out*.print("Suppression des motifs ");

**int** [] num = (**int**[]) (in2.readObject());

**for**(**int** i = 0 ; i < num.length ; i++)

System.*out*.print("(" + num[i] + ")");

System.*out*.println();

ImageDessin.*undo*(num);

OngletHistorique.*ongletActuel*.synchroniser(num);

}

**else** **if**(message.substring(0,1).equals("u")){ // u pour undo

System.*out*.print("Suppression du motif ");

**int** num = Integer.*parseInt*(message.substring(1));

System.*out*.println("(" + num + ")");

ImageDessin.*undo*(num);

**int** [] num2 = **new** **int**[1];

num2[0] = num;

OngletHistorique.*ongletActuel*.synchroniser(num2);

}

**else** **if**(message.substring(0,1).equals("m")){

System.*out*.println("Motif");

Motif motifRecupere = (forme.Motif) (in2.readObject());

OngletHistorique.*ongletActuel*.ajouterMotif(motifRecupere);

ImageDessin.*miseAJourImage*(motifRecupere);

}

**else** **if**(message.substring(0,1).equals("b")){ // b pour bouger

System.*out*.println("D\351placement de Motifs");

**int** [] indexMotifs = (**int**[]) (in2.readObject());

Dimension dim = (Dimension) (in2.readObject());

ImageDessin.*deplacer*(dim, indexMotifs);

}

**else** **if**(message.substring(0,1).equals("k")){ // k parce que j'ai vraiment plus d'inspiration

System.*out*.println("Modification de Motifs");

List<Motif> listeMotifTemp = (**new** ArrayList<Motif>());

**int** [] indexMotifs = (**int**[]) (in2.readObject());

listeMotifTemp = (List<Motif>) (in2.readObject());

ImageDessin.*modifierMotifs*(indexMotifs,listeMotifTemp);

}

**else** **if**(message.equals("synchro")){

System.*out*.println("Synchronisation...");

synchronisation();

}

**else** **if**(message.equals("synchroClient")){

System.*out*.println("Synchronisation des clients...");

synchronisationClient();

}

**else** **if**(message.substring(0,1).equals("a")){

Dessinateur dessinateur = (Dessinateur) (in2.readObject());

**boolean** existe = **false**;

**synchronized** (MainClient.*listeDessinateur*) {

Iterator<Dessinateur> itc = MainClient.*listeDessinateur*.iterator();

**while**(itc.hasNext() && !existe){

Dessinateur dessinateur2 = itc.next();

**if**(dessinateur2.getNom().equals(dessinateur.getNom())){

dessinateur.setConnecte(**true**);

OngletDessinateur.*ongletActuel*.connecterDessinateur(dessinateur);

existe = **true**;

}

}

**if**(!existe){

MainClient.*listeDessinateur*.add(dessinateur);

OngletDessinateur.*ongletActuel*.ajouterDessinateur(dessinateur);

}

}

MainClient.*fenetre*.ajouterMessageChat(" - " + dessinateur.getNom() + " s'est connect\351 - ");

System.*out*.println(dessinateur.getNom() + " s'est connect\351");

}

**else** **if**(message.substring(0,1).equals("d")){

Dessinateur dessinateur = (Dessinateur) (in2.readObject());

**synchronized** (MainClient.*listeDessinateur*) {

Iterator<Dessinateur> itc = MainClient.*listeDessinateur*.iterator();

**while**(itc.hasNext()){

Dessinateur dessinateur2 = itc.next();

**if**(dessinateur2.getNom().equals(dessinateur.getNom())){

dessinateur.setConnecte(**false**);

MainClient.*fenetre*.ajouterMessageChat(" - " + dessinateur.getNom() + " s'est deconnect\351 - ");

OngletDessinateur.*ongletActuel*.deconnecterDessinateur(dessinateur);

}

}

}

System.*out*.println(dessinateur.getNom() + " est d\351connect\351");

}

**else** **if**(message.equals("Bye")){

System.*out*.println("Deconnexion...");

client.deconnecter();

MainClient.*fenetre*.modeDeconnecte();

connect = **false**;

}

MainClient.*fenetre*.getCanevas().repaint();

} // Fin de la boucle de reception

} **catch** (IOException e) {

System.*err*.println("Serveur deconnect\351.");

connect = **false**;

client.deconnecter();

MainClient.*fenetre*.modeDeconnecte();

JOptionPane jop = **new** JOptionPane();

jop.*showMessageDialog*(MainClient.*fenetre*, "Vous avez \351t\351 d\351connect\351 du serveur", "Le serveur ne r\351pond plus", JOptionPane.*WARNING\_MESSAGE*);

} **catch** (ClassNotFoundException e) {

System.*err*.println("Erreur de conversion de donn\351es");

e.printStackTrace();

connect = **false**;

}

}

**private** **void** synchronisation() **throws** ClassNotFoundException, IOException {

List<Motif> listeMotifTemp = (**new** ArrayList<Motif>());

Dimension dimensionCanevasTemp = **new** Dimension();

listeMotifTemp = (List<Motif>) (in2.readObject());

**synchronized** (ImageDessin.*listeMotif*) {

ImageDessin.*listeMotif* = listeMotifTemp;

}

dimensionCanevasTemp = (Dimension) (in2.readObject());

**synchronized** (Canevas.*getDimensionCanevas*()) {

Canevas.*setDimensionCanevas*(dimensionCanevasTemp);

}

System.*out*.println("Synchronisation Termin\351e !");

OngletHistorique.*ongletActuel*.synchroniser();

ImageDessin.*creerImage*();

}

**private** **void** synchronisationClient() **throws** ClassNotFoundException, IOException {

List<Dessinateur> listeDessinateurs = Collections.*synchronizedList*(**new** ArrayList<Dessinateur>());

listeDessinateurs =(List<Dessinateur>) (in2.readObject());

MainClient.*listeDessinateur* = listeDessinateurs;

**for**(**int** i = 0 ; i < listeDessinateurs.size() ; i++){

**if**(listeDessinateurs.get(i).getNom().equals(MainClient.*client*.getNom()))

MainClient.*client*.setNumuser(i);

}

OngletDessinateur.*ongletActuel*.synchroniser();

System.*out*.println(listeDessinateurs);

System.*out*.println("Synchronisation des Clients Termin\351e !");

}

}

Classe DataClient : Très importante, cette classe est à la fois utilisée sur le serveur et le client et stocke les données des différents clients. Elle comporte les méthodes d’envoi de données aux clients.

**package** ProgrammeServeur;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.io.PrintWriter;

**import** java.io.Serializable;

**import** java.net.Socket;

**public** **class** DataClient **implements** Serializable {

**private** Socket socket1, socket2;

**private** String nom = **null**;

**private** BufferedReader inTexte = **null**;

**private** ObjectInputStream inMotif = **null**;

**private** PrintWriter outTexte = **null**;

**private** ObjectOutputStream outMotif = **null**;

**private** **boolean** connecte = **true**;

**private** **int** numUser;

**public** DataClient() {

}

**public** String getNom() {

**return** nom;

}

**public** **void** setNom(String nom) {

**this**.nom = nom;

}

**public** BufferedReader getInTexte() {

**return** inTexte;

}

**public** ObjectInputStream getInMotif() {

**return** inMotif;

}

**public** PrintWriter getOutTexte() {

**return** outTexte;

}

**public** ObjectOutputStream getOutMotif() {

**return** outMotif;

}

**public** **boolean** isConnecte() {

**return** connecte;

}

**public** **void** setConnecte(**boolean** connecte) {

**this**.connecte = connecte;

}

**public** **void** closeSocket1() {

**try** {

socket1.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**public** **void** closeSocket2() {

**try** {

socket2.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

**public** **void** setSocket1(Socket socket1) {

**this**.socket1 = socket1;

**if** (outTexte == **null**) {

**try** {

outTexte = **new** PrintWriter(**this**.socket1.getOutputStream());

} **catch** (IOException e) {

e.printStackTrace();

}

}

**if** (inTexte == **null**) {

**try** {

inTexte = **new** BufferedReader(**new** InputStreamReader(socket1.getInputStream()));

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**public** **void** setSocket2(Socket socket2) {

**this**.socket2 = socket2;

**if** (outMotif == **null**) {

**try** {

outMotif = **new** ObjectOutputStream(socket2.getOutputStream());

} **catch** (IOException e) {

e.printStackTrace();

}

}

**if** (inMotif == **null**) {

**try** {

inMotif = **new** ObjectInputStream(socket2.getInputStream());

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**public** **void** envoyerMotif(Object motif) {

envoyer("m", motif);

}

**public** **void** envoyer(String chaine) {

envoyer(chaine, **null**);

}

**public** **void** envoyer(Object obj) {

envoyer(**null**, obj);

}

**public** **synchronized** **void** envoyer(String chaine, Object obj) {

**synchronized** (outMotif) {

**synchronized** (outTexte) {

**if** (chaine != **null**) {

outTexte.print(chaine + "\r\n");

outTexte.flush();

}

**if** (obj != **null**) {

**try** {

outMotif.writeObject(obj);

outMotif.flush();

} **catch** (IOException e) {

System.*err*.println("Pas de r\351ponse du serveur distant (" + nom + ") : " + e.getMessage());

deconnecter();

}

}

}

}

}

**public** **synchronized** **void** envoyer(Object[] obj) {

envoyer(**null**, obj);

}

**public** **synchronized** **void** envoyer(String chaine, Object[] obj) {

**synchronized** (outMotif) {

**synchronized** (outTexte) {

**if** (chaine != **null**) {

outTexte.print(chaine + "\r\n");

outTexte.flush();

}

**if** (obj != **null**) {

**try** {

**for** (**int** i = 0; i < obj.length; i++) {

outMotif.writeObject(obj[i]);

outMotif.flush();

}

} **catch** (IOException e) {

System.*err*.println("Pas de r\351ponse du serveur distant (" + nom + ") : " + e.getMessage());

deconnecter();

}

}

}

}

}

**public** Socket getSocket2() {

**return** socket2;

}

**public** Socket getSocket1() {

**return** socket1;

}

**public** String toString() {

**return** nom;

}

**public** **void** deconnecter() {

**try** {

inTexte.close();

inMotif.close();

outMotif.close();

outTexte.close();

socket1.close();

socket2.close();

} **catch** (IOException e) {

// e.printStackTrace();

}

inMotif = **null**;

inTexte = **null**;

outMotif = **null**;

outTexte = **null**;

connecte = **false**;

}

**public** **int** getNumuser() {

**return** numUser;

}

**public** **void** setNumuser(**int** num) {

numUser = num;

}

}

# VI – Résultat d’exécution : Capture d’écran de l’IHM

Figure 2 - IHM déconnectée

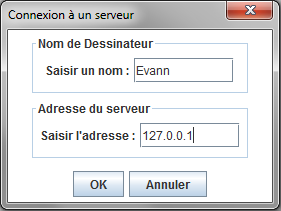


Figure 3 - Fenêtre de connexion

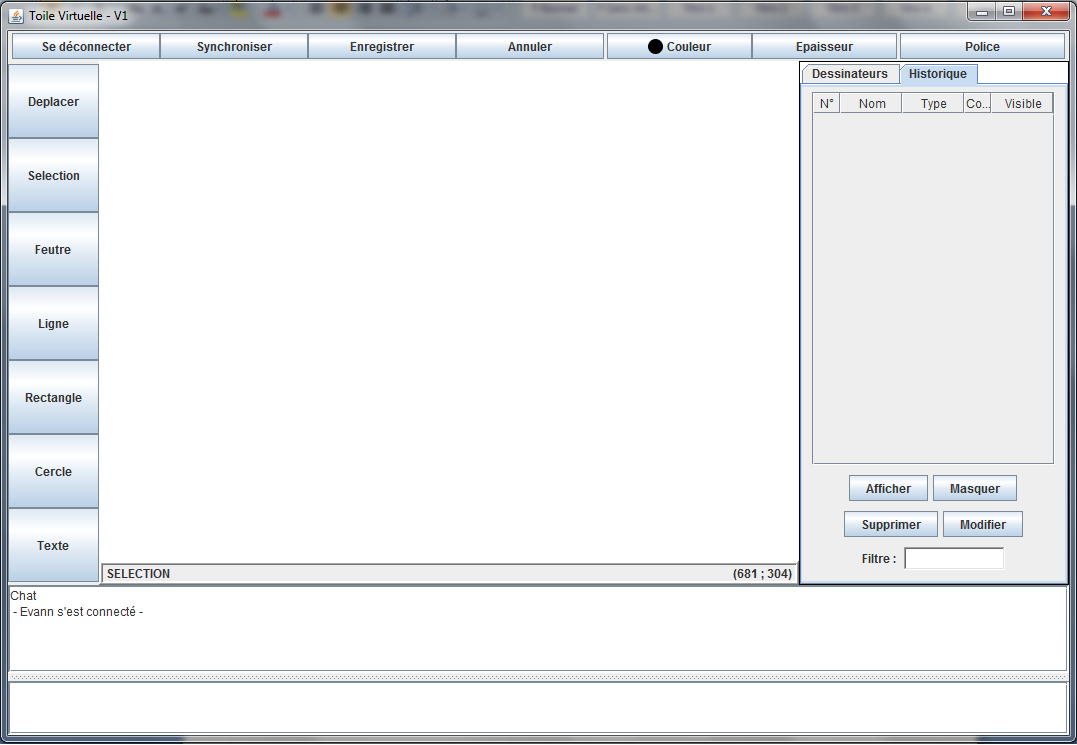


Figure 4 - IHM connectée

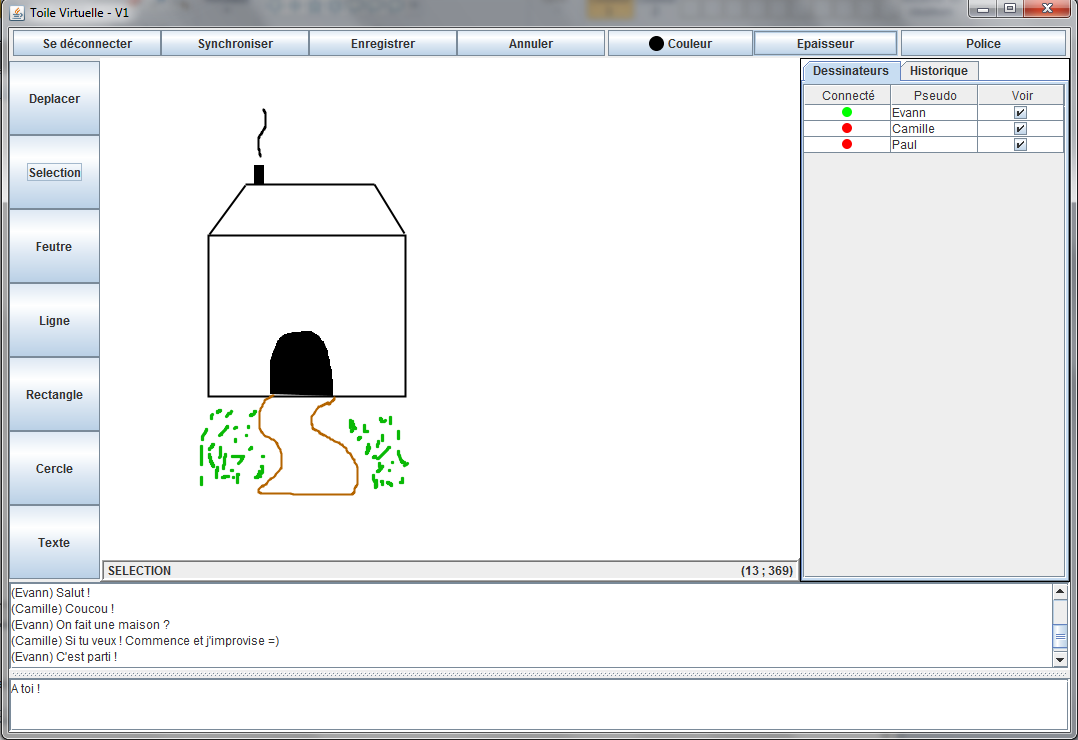


Figure 5 - Dessin en cours

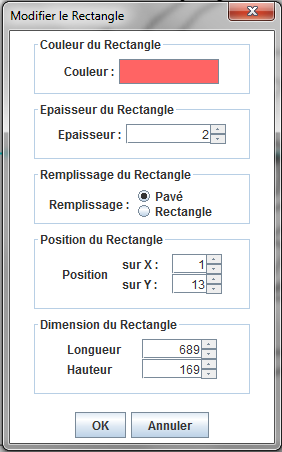
*Caractéristiques du projet :   
Le projet comporte 32 classes dont 28 côté client et 4 côté serveur.  
Les classes DataClient (côté serveur) et Dessinateur (côté client) sont utilisées par les deux programmes.  
Le programme client comporte environ 4600 lignes de code et le programme serveur comporte environ 570 lignes pour un total d’environ 5200 lignes.*

Figure 6 - Boite de dialogue de modification d'un rectangle