Tutor Application

Comp 380 Spring 2010

Matthew Hoggan
Andrew Hamilton
Alex David
Vahe Margoussian
Daniel Sery

Contents

Comp 380 Spring 2010

Assignment #2

Matthew Hoggan, Andrew Hamilton, Alex David, Vahe Margoussian, Daniel Sery

Requirements and Specification

1. PROJECT BACKGROUND

Most students on campus do not take advantage of tutoring resources available to them. Students either do not know these resources exist or they do not know how to access them. Additionally, due to employment or long daily commutes to and from campus, CSUN students spend less time at the university relative to other academic institutions. These restrictions determine whether students have access to an on campus tutoring lab. Under the present system, students with restricting schedules do not have fair access to the tutoring lab services provided by their corresponding departments.

If CSUN were to setup a system for remote tutoring, students and tutors alike would be limited to using the English language and would be unable to use other academic languages such as those encountered in mathematics and science. This project will create a medium in which students can get access to such advanced tutoring resources from anywhere, whether on or off campus. Such a dynamic system would free the students and tutors to work where it is most convenient for them.

2. MISSION (GOAL) STATEMENT

The project is designed to be an aid to students who may need academic help with course material, but are for whatever reason, unable to work with a tutor in person. The project allows students who require assistance with their course work to quickly connect to a tutor, from any computer, and receive help from a tutor proficient at the topic in question. The tutor will have an easy to use UI at her disposal; she will be able to send text, and even draw out more complex elements such as diagrams and graphs. By using the resources provided by our program, students can receive a broad range of assistance from any tutor logged into the system at that time in a comprehensive manner.

3. APPLICATION DESCRIPTION

The application will consist of both a client side and server side program which will facilitate the communication between students and tutors across campus. The client side will be run through any web browser available to students. This

information will then be placed in a queue and made available to tutors logged in on the server side. Tutors will be able to respond to any question they feel fit to answer through a custom built program which will facilitate communication back to the student who is logged in at their personal computer.

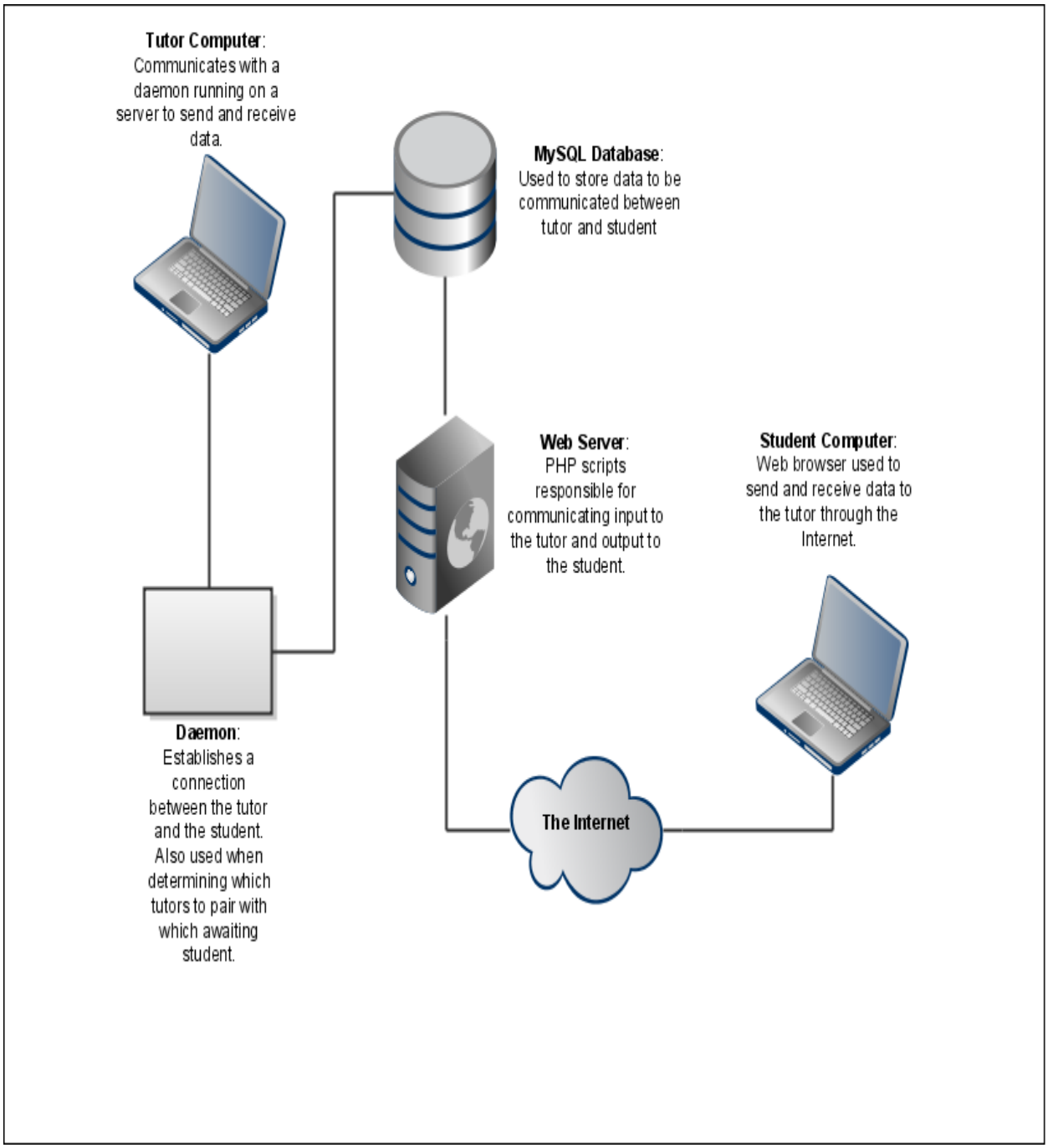


Diagram 1: Main Reference Diagram for Project 1

4. THE SCOPE OF THE WORK (IN/OUT of SCOPE LIST)

1. IN: The project shall consist of six essential pieces of software. Four of the proposed pieces will be developed by our team, and the remaining two pieces will consist of an Apache Web Server and a MySQL Database. Only the tutor app will be developed using an object oriented approach.

2. IN: The deliverables shall include a requirement document, design document, source code, and executable binaries.

3. IN: The project shall include the implementation of a GUI using GTK, GTKMM, and GLIB libraries for the tutor. Additionally, it will include a TCP/IP interface for the tutor application to retrieve information posted to a database by the students.

4. IN: The project shall also include a web based application made available to students which will interact with an Apache based Web Server. The web server in turn will communicate with PHP scripts on the server to GET and POST to the MySQL Database.

5. OUT: This project does not require the development of hardware.

6. OUT: A remote tutoring system which can be used by universities across the United States, which will better serve a more mobile generation.

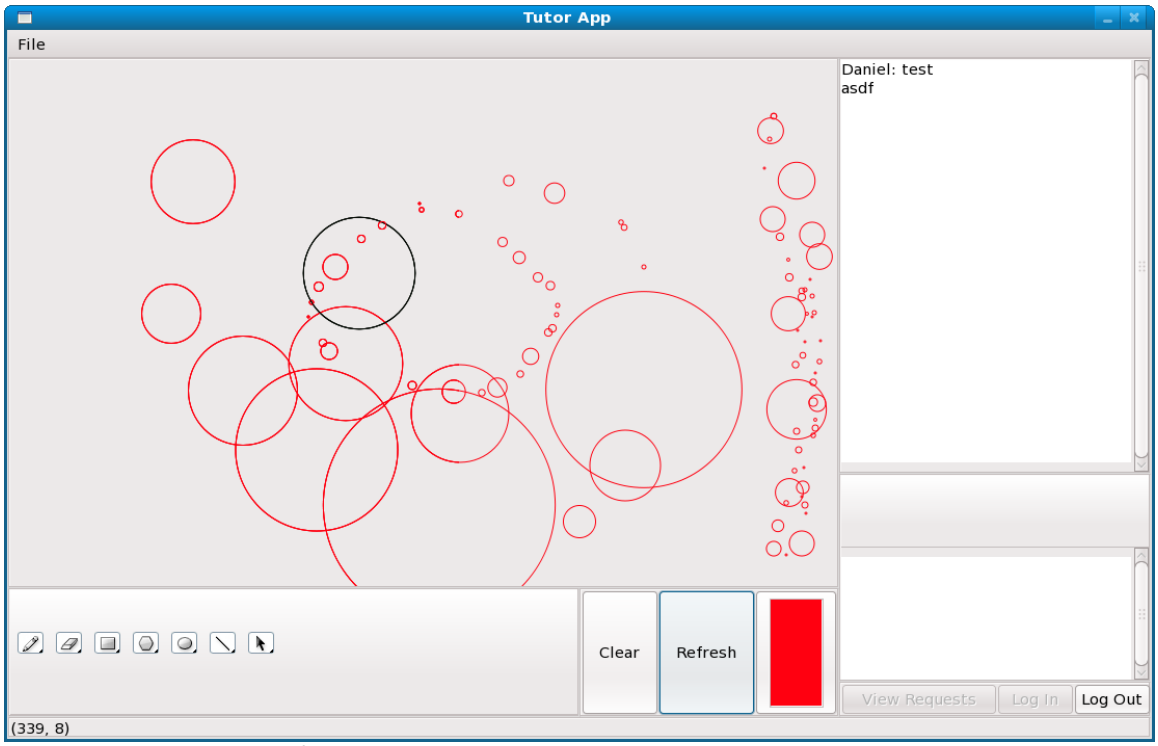
5. FUNCTIONAL REQUIREMENTS

By and large, the functional requirements for the tutoring system consist of three groups. The first group contains the tutoring application. This will be a stand alone application used by the tutors and will communicate directly with the daemon (discussed below) on the server. The second group contains the components on the server and is responsible for allowing the student and the tutor to communicate. The components of the second group can be broken down into two subgroups. The first subgroup contains a daemon responsible for establishing a connection with *n* tutors, a pairing process responsible for pairing available tutors with awaiting students, and a MySQL database to store user data. The second subgroup contains a set of PHP scripts to interact with incoming and outgoing requests to the Apache based Web Server. The final group contains the students' Web interface. This interface will be implemented using CSS, XHTML, and Javascript and accessible to *m* students.

$5_{a)}$ *The Tutoring Application*

The tutoring application will allow the tutor to interact with the student by means of different forms of digital information. The tutor should be able to draw

pictures, and send plain text. The GUI part for this application shall appear as follows:



Picture 1: Tutoring Application Interface

5_2 *The Server Side Programs*

5_{2a} *Daemon for Connections*

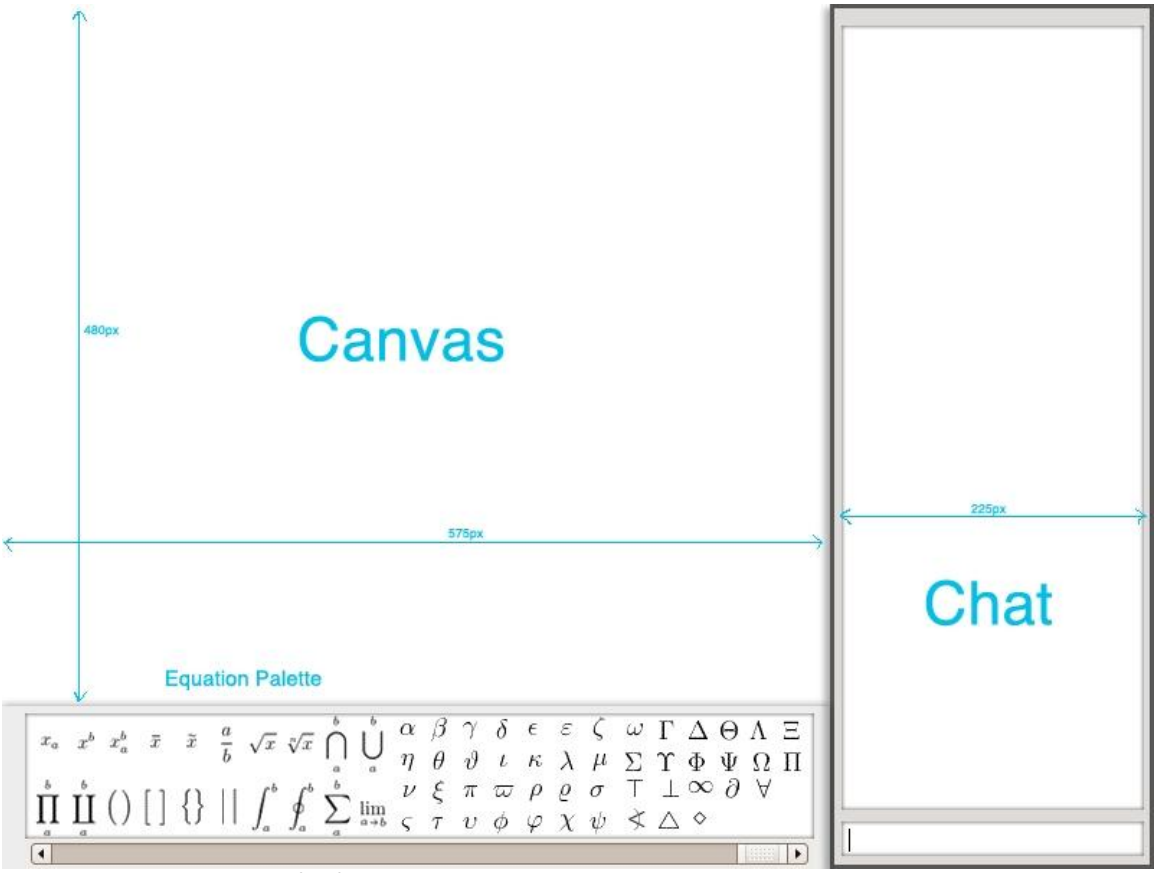
The daemon's primary function will be to establish connections with the tutors. After the daemon establishes a connection with the tutoring application it will fork off a child process. The child process will be in control of accessing data from the table created for the student currently being tutored. The child process terminates when the tutor disconnects by ending the chat with the student or logs out of the system.

5_{2b} *PHP Scripts*

The primary responsibility of these scripts will be to write information to and from a specific table in the database, created for the student upon connection. These scripts will also remove that table when the session between the tutor and student is terminated.

5_3 *Web Interface for Students*

The web interface for students (WIFS) will be responsible for allowing the student to interact with her tutor. The information being produced by the student will by asynchronously uploaded to the server using JQuery AJAX libraries. Additionally, the WIFS will download information posted by the tutor to a separate viewing pane using JQuery AJAX libraries. The information rendered to the screen plain text, or vector based graphics. The WIFS will make it seem as if student and tutor were meeting in person. The WIFS shall appear in general as seen below:

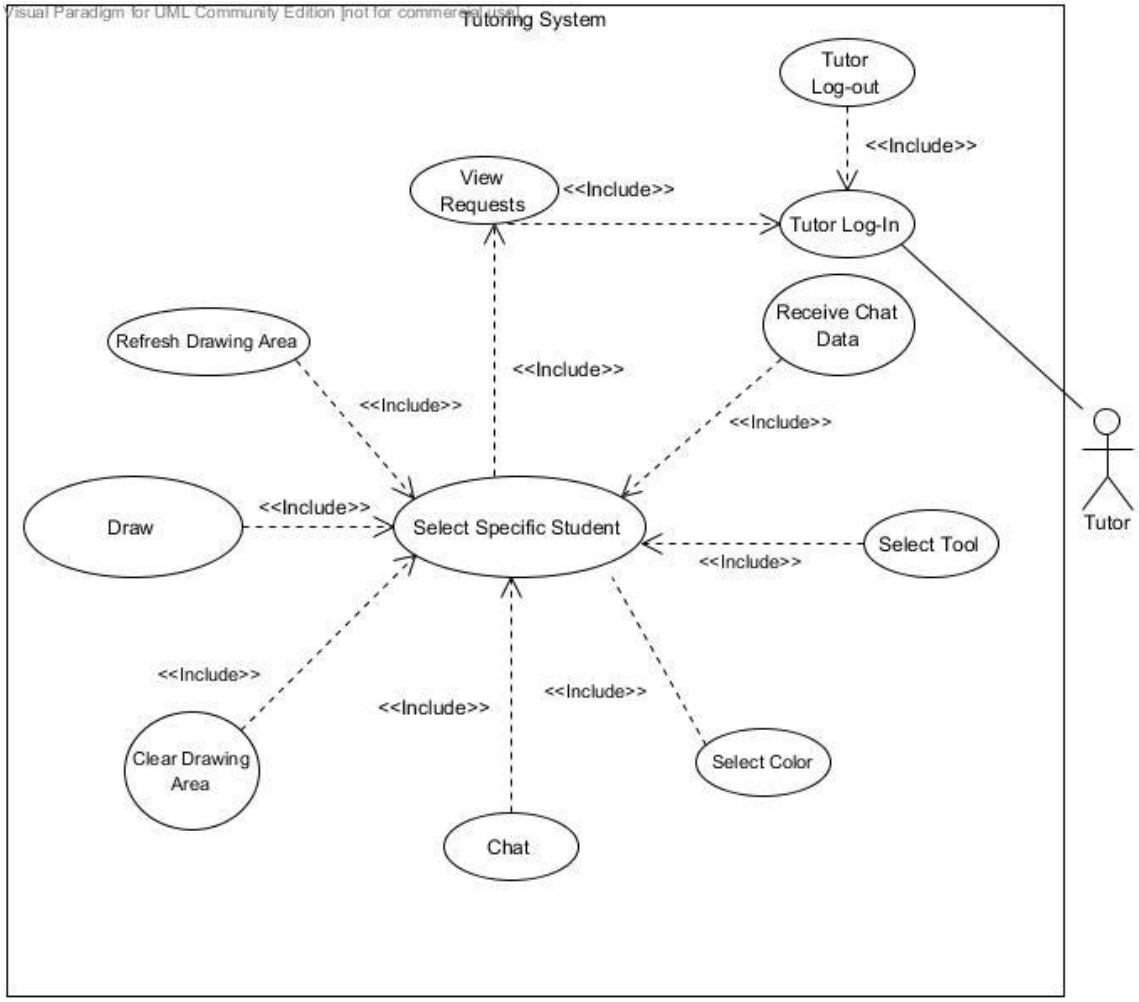


Picture 2: The Web Interface for Students

6. Use Case

6_a. Use Case Diagram

(SEE NEXT PAGE)



6_b Use Case Description

Characteristic Information	
UC1: Tutor Log-In	
Goal	To log into the system
Pre-Condition	The tutor has tutor application open and installed on computer
Success End Condition	The tutor can perform any of the included actions
Primary Actor	Tutor

Characteristic Information	
UC2: View Requests	
Goal	Tutor is able to see all requests from students on server
Pre-Condition	Tutor is logged in
Success End Condition	Tutor can see if there are students in queue and pick one
Primary Actor	Tutor

Characteristic Information	
UC3: Select Specific Student	
Goal	Tutor can pick from the queue a student
Pre-Condition	The tutor is logged in
Success End Condition	Tutor opens session with student
Primary Actor	Tutor

Characteristic Information	
UC4: Refresh Drawing Area	
Goal	If drawing area is damaged tutor can refresh it
Pre-Condition	Tutor is logged in, he has selected a student from queue
Success End Condition	The students question is answered
Primary Actor	Tutor

Characteristic Information	
UC5: Draw	
Goal	The tutor can draw to student using selected tool
Pre-Condition	Tutor is logged in, he has selected a student from queue
Success End Condition	The students question is answered
Primary Actor	Tutor

Characteristic Information	
UC6: Clear Drawing Area	
Goal	The tutor has filled up drawing area or wants to start over
Pre-Condition	Tutor is logged in, he has selected a student from queue
Success End Condition	The students question is answered
Primary Actor	Tutor

Characteristic Information	
UC7: Chat	
Goal	The tutor can chat with client through text
Pre-Condition	Tutor is logged in, he has selected a student from queue
Success End Condition	The students question is answered
Primary Actor	Tutor

Characteristic Information	
UC8: Select Color	
Goal	The tutor can change the color of drawing area
Pre-Condition	Tutor is logged in, he has selected a student from queue
Success End Condition	The students question is answered
Primary Actor	Tutor

Characteristic Information	
UC9: Select Tool	
Goal	The tool applied to drawing area is changed
Pre-Condition	Tutor is logged in, he has selected a student from queue
Success End Condition	The students question is answered
Primary Actor	Tutor

Characteristic Information	
UC10: Tutor Log-out	
Goal	Tutor quits and table instace deleted as well as connection closed
Pre-Condition	The tutor is logged in
Success End Condition	The tutor can leave
Primary Actor	Tutor

Characteristic Information	
UC11: Receive Chat Data	
Goal	Tutor is able to see chat information sent by student
Pre-Condition	Tutor is logged in
Success End Condition	Tutor can respond to text based question
Primary Actor	Tutor

6_c. Detailed Use Case Description

UC1: Tutor Log-In		
Primary Scenario		
Step	Actor/System	Action Description
1	Tutor	Open the tutor application
2	System	load the drivers and componments
3	System	display login screen
Alternative Scenario		
Step	Condition	Action Description
2a	unable to load drivers and components	system display message error
3a	login screen fails to appear	system display message error

UC2: View Requests		
Primary Scenario		
Step	Actor/System	Action Description
1	Tutor	Tutor presses the view requests button / Or Tutor Logs in
2	Application	Displays all available requests from users awaiting the tutor's attention. Also displays name of the user and the requested subject
4	Tutor	Tutor selects a user to assist and clicks select
Alternative Scenario		
Step	Condition	Action Description
2a	No Users available	Informs tutor that no users are currently seeking assistance.

UC3: Select Specific Student		
Primary Scenario		
Step	Actor/System	Action Description
1.	Application	Tutor is presented with a queue of questions that pertain to the subject they are proficient at.
2.	Tutor	Tutor Select Student
3.	Tutor	Tutor Clicks Cancel
Alternative Scenario		
Step	Condition	Action Description
2a.	Tutor Clicks Select	GUI Activated Session Started
3a.	Tutor Clicks Cancel	GUI Deactivate, tutor can only select Log Out

UC4: Refresh Drawing Area		
Primary Scenario		
Step	Actor / System	Action Description
1	Tutor	Tutor clicks on the the "Logout" button
2	Application	Iterates Over Data Structure Redrawing Background
Alternative Scenario		
Step	Actor / System	Action Description

UC5: Draw		
Primary Scenario		
Step	Actor / System	Action Description
1	Tutor	Tutor clicks in drawing area and drags mouse producing shape
2	Application	On mouse release shape information sent over server
3	Application	If pencil is selected, constant data sent on mouse motion
4	Application	Data writen to data structure
Alternative Scenario		
Step	Actor / System	Action Description

UC6: Clear Drawing Area		
Primary Scenario		
Step	Actor / System	Action Description
1	Tutor	Tutor Clicks On Clear button
2	Application	Program deletes the data structure and builds new one
3	Application	Drawing area is invalidated
Alternative Scenario		

Step	Actor / System	Action Description

UC7: Chat		
Primary Scenario		
Step	Actor / System	Action Description
1	Tutor	Tutor types message and hits enter
2	Application	Text is extracted from input and sent to server
3	Application	Text is pasted to output
4	Application	Text is removed from input
Alternative Scenario		
Step	Actor / System	Action Description

UC8: Select Color		
Primary Scenario		
Step	Actor / System	Action Description
1	Tutor	Tutor clicks on color button
2	Application	Color Dialog Opens
3	Application	Tutor Select color or clicks color
4	Application	Color is set based on color of button
5	Application	Portion of screen invalidated
Alternative Scenario		
Step	Actor / System	Action Description

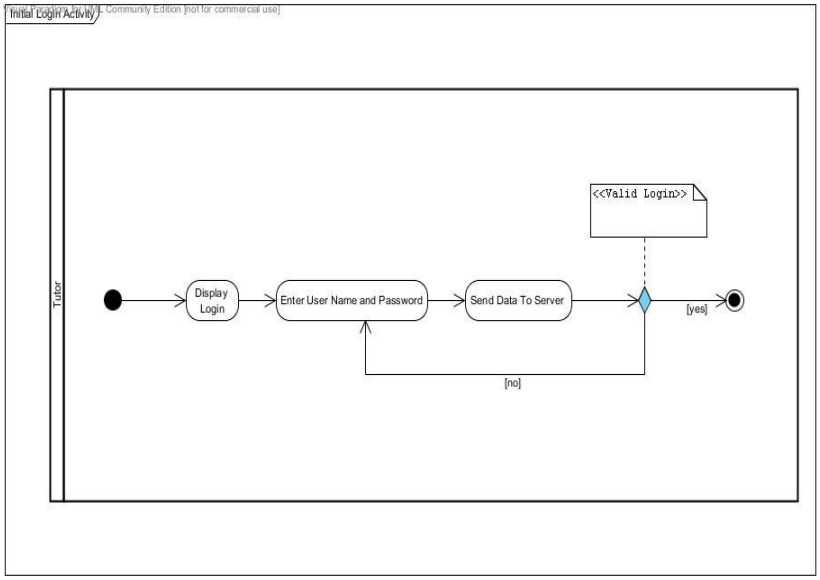
UC9: Select Tool		
Primary Scenario		
Step	Actor / System	Action Description
1	Tutor	Tutor Selects Tool
2	Application	State of Drawing portion of program changes
Alternative Scenario		
Step	Actor / System	Action Description

UC9: Tutor Logout		
Primary Scenario		
Step	Actor / System	Action Description
1	Tutor	Tutor clicks on the the "Logout" button
2	Application	Program Terminates
Alternative Scenario		
Step	Actor / System	Action Description

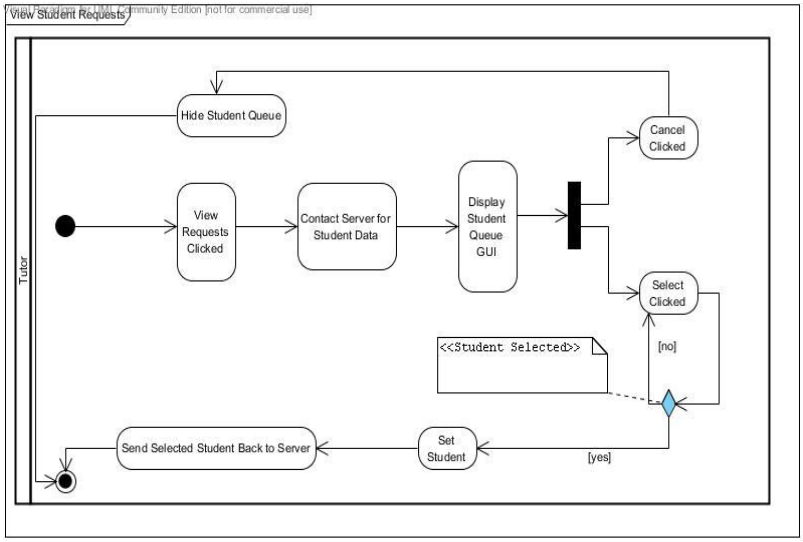
UC11: Receive Chat Data		
Primary Scenario		
Step	Actor / System	Action Description
1	Application	Socket recieves chat data
2	Application	Socket displays chat data in output box
Alternative Scenario		
Step	Actor / System	Action Description

7. Activity Diagrams

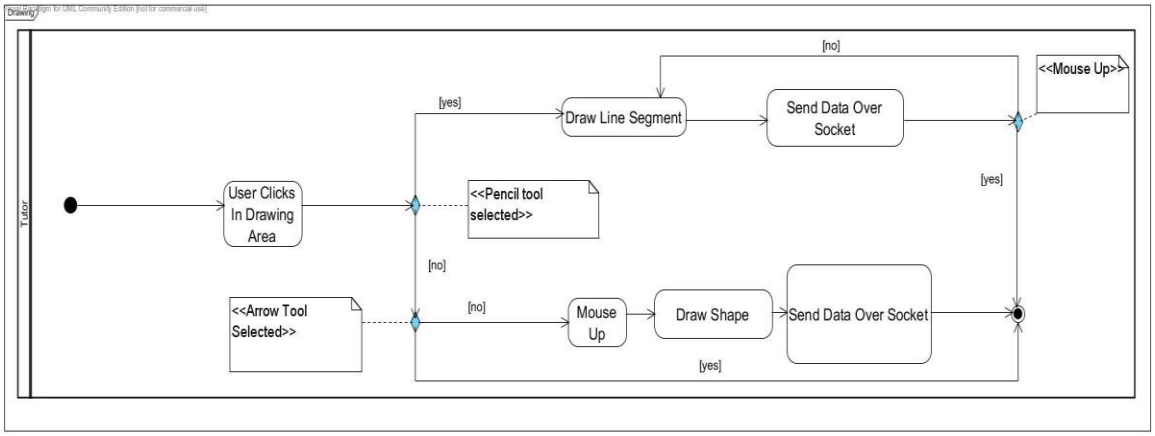
7_a. Initial Login Activity Diagram



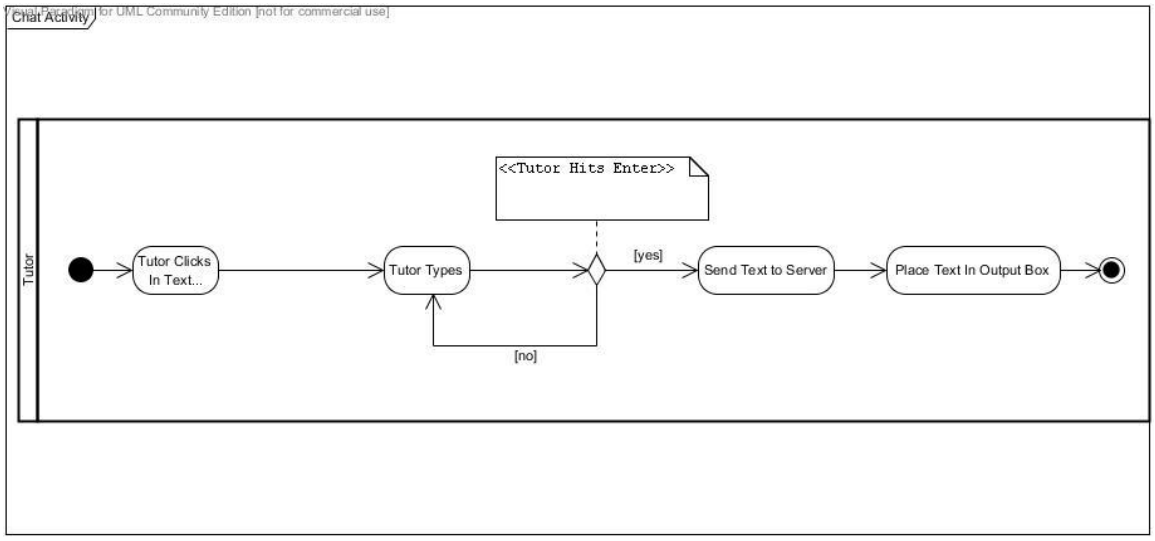
7_b. View Student Requests Activity Diagram



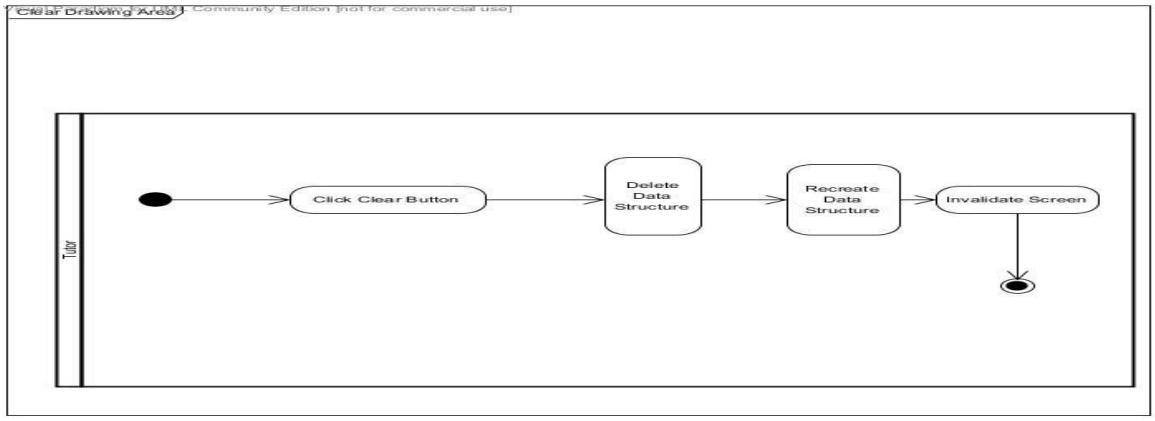
7_c. Drawing Activity Diagram



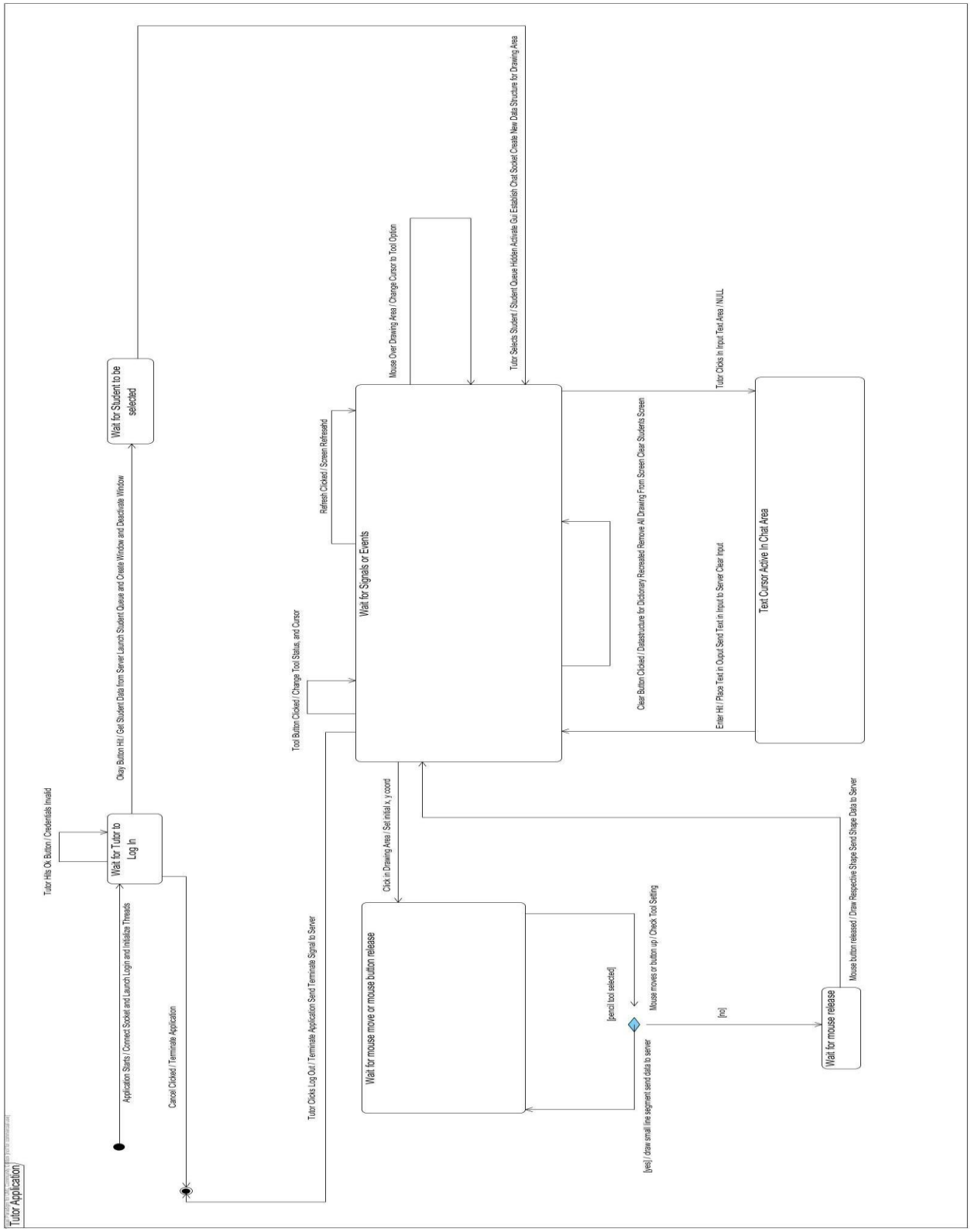
7_d. Chat Activity Diagram



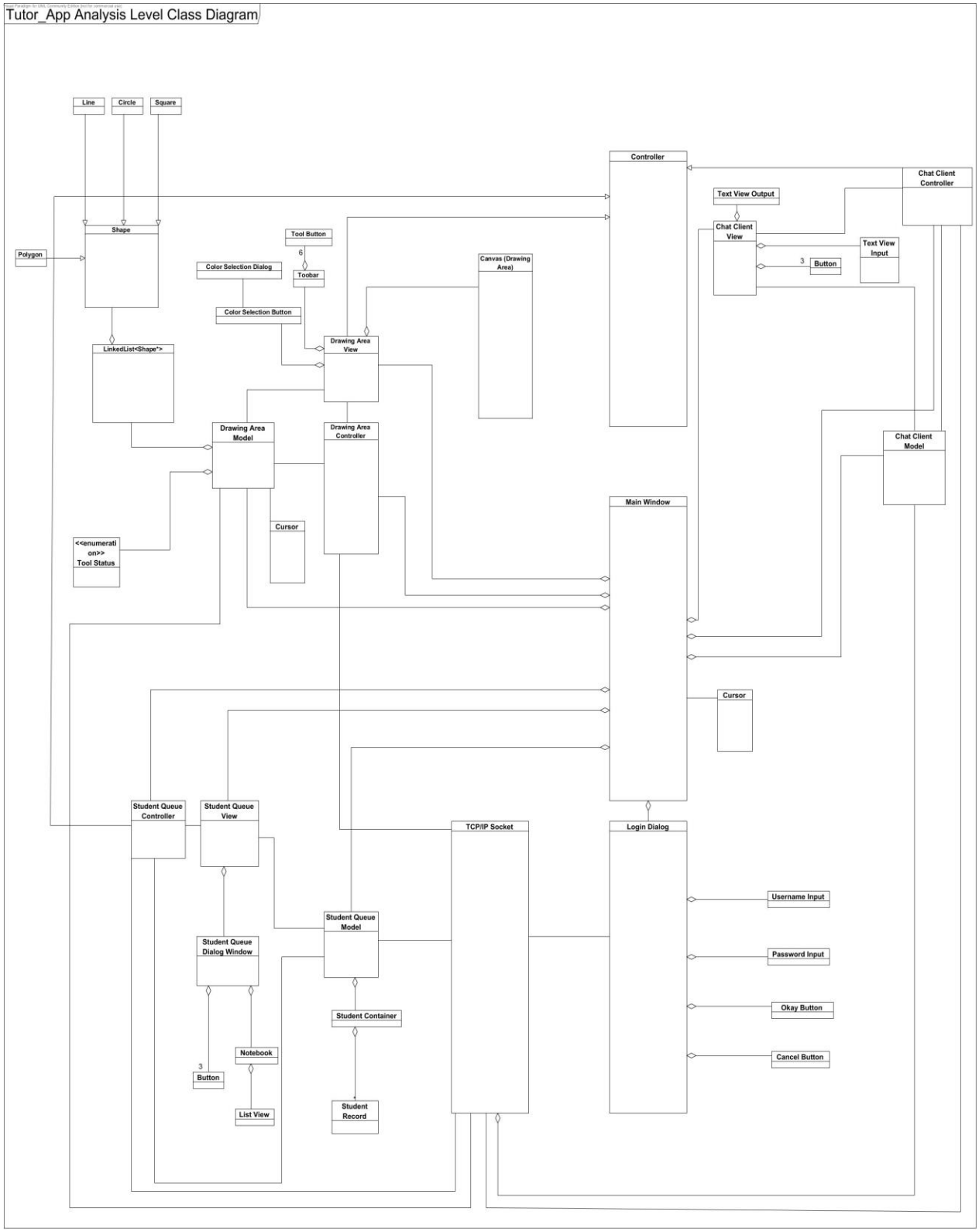
7_e. Erase Drawing Area



8. State Chart Diagram



9. Analysis Level Class Diagram



10. CRC Cards

Circle	
Responsibilities 1) Encapsulates data needed to draw a specific circle to the sketch board in Model Drawing	Colaborators LinkedList<Shape*> ControlerDrawing ModelDrawing

Controler	
Responsibilities 1) Abstract class for ControlerDrawing, ControlerQueue, and ControlerChat 2) Defines the basic input mechanisims for a Controler in MVC Architecture	Colaborators ControlerDrawing ControlerQueue ControlerChat

Controler Chat	
Responsibilities 1) Set sensitivity on chat widgets 2) Set editable state on chat widgets 3) Clears the input boxes 4) Establishes a socket for chat data	Colaborators ModelChat ViewChat MainWindow

Controler Drawing	
Responsibilities 1) Set sensitivity on drawing widgets 2) Set and change curosr for drawing area 3) Set the state for the drawing area based on which tool is selected 4) Refresh drawing area	Colaborators ModelDrawing ViewDrawing MainWindow

5) Clear drawing area 6) Draw shapes to drawing area 7) Set color for drawing area	

Controler Queue	
Responsibilities 1) Gets student data from server 2) Sets student for session 3) Builds and delete student vector	Colaborators ModelQueue ViewQueue MainWindow LinkedList<Student*>

Line	
Responsibilities 1) Encapsulates data needed to draw a specific line to the sketch board in Model Drawing	Colaborators LinkedList<Shape*> ControlerDrawing ModelDrawing

Linked List	
Responsibilities To house generic data for multiple parts of the application	Colaborators Node<T> ModelDrawing ModelQueue MainWindow

Login_Dialog	
Responsibilities 1) Sets tutor name for main application 2) Reads in user name and password for server authentication	Colaborators MainWindow

3) Hashes the password across the network	

MainWindow

Responsibilities	Colaborators
1) Sets the environment and acts as the bridge and controler across all parts of the program. All objects at some point communicate through the main window to some other object 2) Responisble for threading to handle input from server 3) Creates the login screen, and communicates to the server through its socket both for authentication and sending drawing data 4) Creates the student queue and obtains the student from it	ModelDrawing ViewDrawing ControlerDrawing ModelChat ViewChat ControlerChat ModelQueue ViewQueue ControlerQueue Login_Dialog Tutor Student TCP_IP_Socket

ModelChat

Responsibilities	Colaborators
1) House the socket to transfer and receive data designed for the chat area	TCP_IP_Socket MainWindow

ModelColumns

Responsibilities	Colaborators
1) Responsible for building the list view seen in ViewQueue	ModelQueue

ModelDrawing

Responsibilities	Colaborators
1) Houses the data structure for all drawing data 2) Houses the color set for the drawing area 3) Houses the cairo device context responsible for drawing to the drawing area	MainWindow LinkedList<Shape*>

ModelQueue	
Responsibilities 1) To allow the tutor to select a student to work with	Colaborators LinkedList<Student*> MainWindow Student

Polygon	
Responsibilities 1) Encapsulates data needed to draw a specific polygon to the sketch board in Model Drawing	Colaborators LinkedList<Shape*> ControlerDrawing ModelDrawing

Shape	
Responsibilities 1) Acts as and abstract class for Polygon, Line, Circle, and Square	Colaborators Polygon Line Circle Square

Square	
Responsibilities 1) Encapsulates data needed to draw a specific square to the sketch board in Model Drawing	Colaborators LinkedList<Shape*> ControlerDrawing ModelDrawing

Student	
Responsibilities 1) The student selecte from the student Queue	Colaborators LinkedList<Student*> MainWindow ModelQueue ControlerQueue

TCP_IP_Socket	
Responsibilities 1) Responsible for all communication with server 2) Send Login data 3) Sends and receives chat data 4) Sends drawing data	Colaborators MainWindow ModelChat Student Tutor Shape Polygon Line Square Circle

Tokenizer	
Responsibilities 1) Breaks strings of data into pieces for various aspects of the program	Colaborators MainWindow ControlerQueue LinkedList<std::string>

Tutor	
Responsibilities 1) Holds the information for the tutor who loged in	Colaborators MainWindow

ViewChat	
Responsibilities 1) Interacts with the ControlerChat and ModelChat to display the data from model and respond to the actions from controller	Colaborators ModelChat ControlerChat

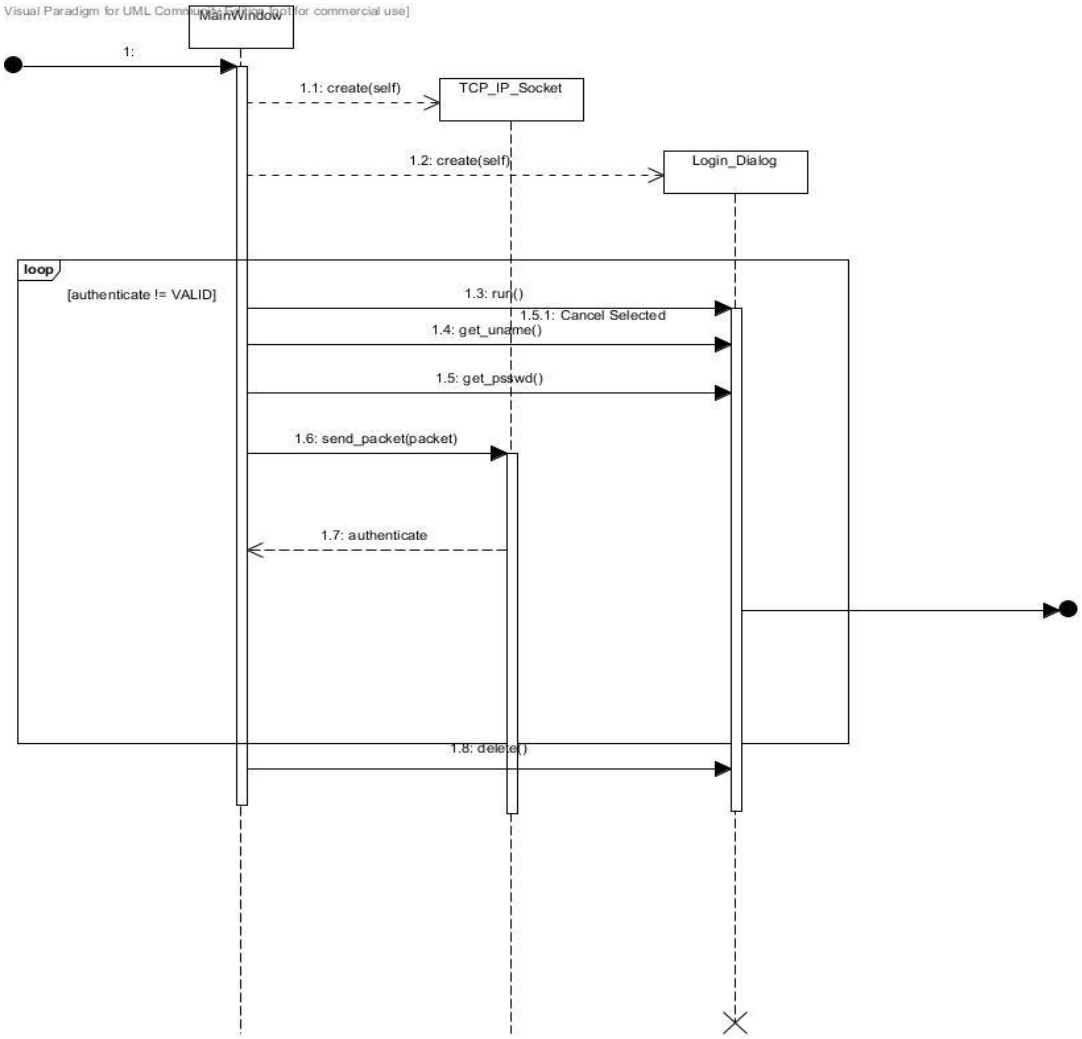
ViewDrawing	
Responsibilities 1) Interacts with the ControlerDrawing and Model Drawing to display the data from model and respond to the actions from	Colaborators ModelDrawing ControlerDrawing

controller	

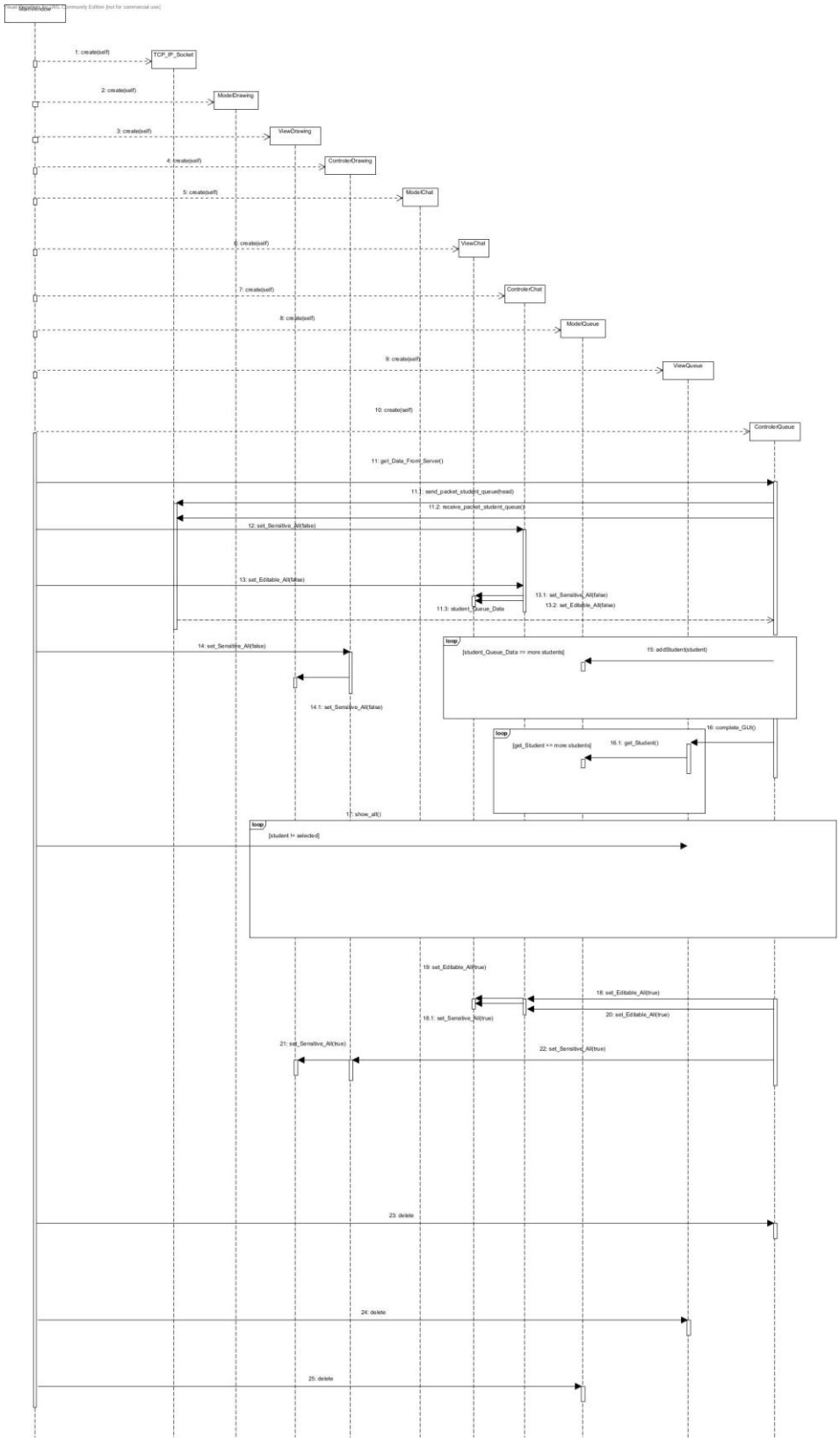
ViewQueue	
Responsibilities 1) Interacts with the ControlerQueue and ModelQueue to display the data from model and respond to the actions from controller	Colaborators ModelQueue ControlerQueue

11. Sequence Diagrams

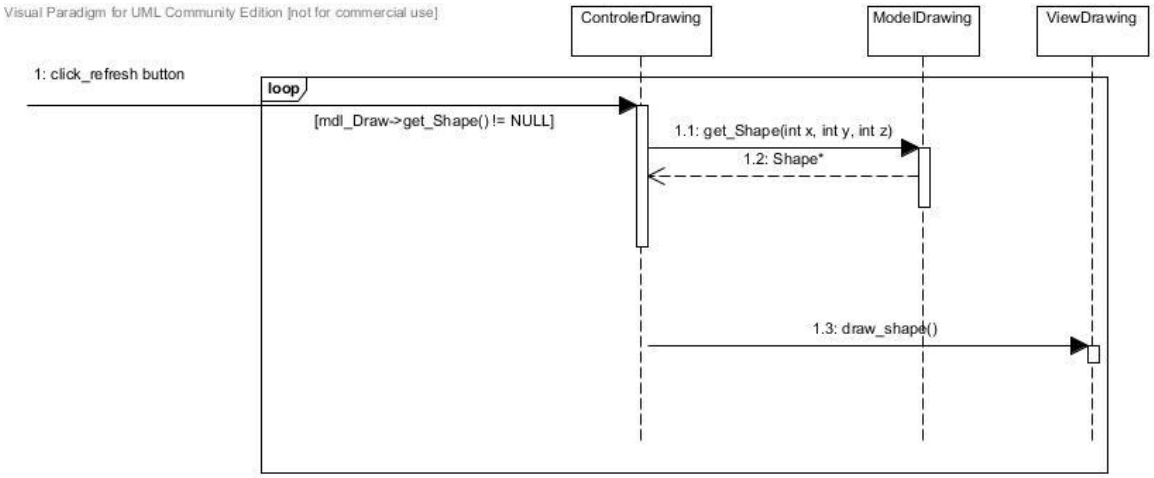
11_a. Tutor Log-In Sequence Diagram



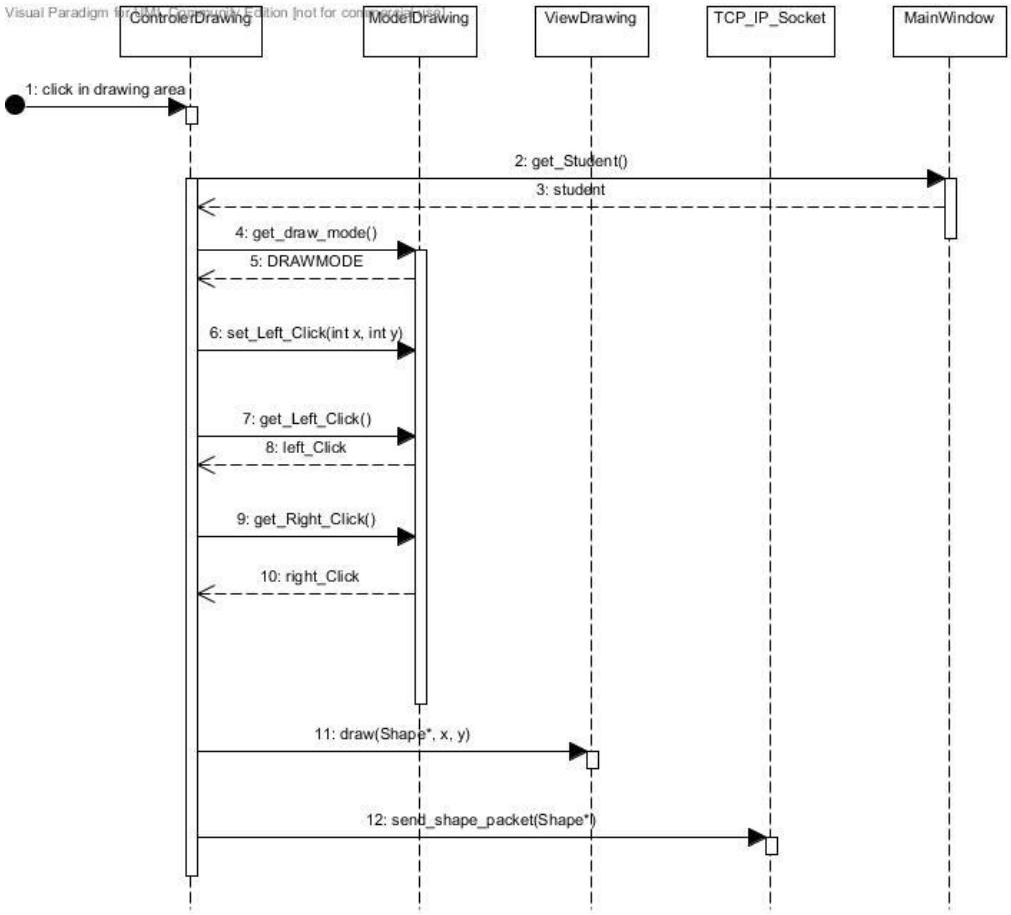
11_b. View Requests and Select Specific Student Sequence Diagram



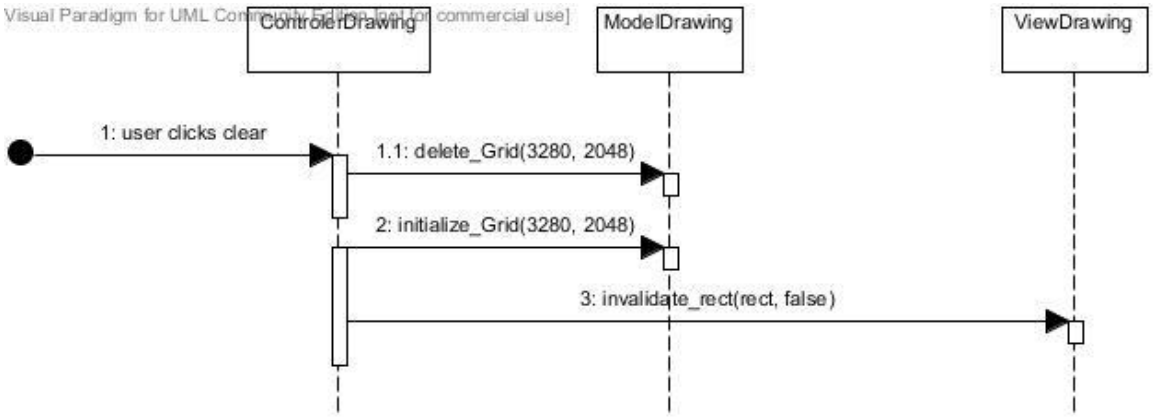
11_c. Refresh Drawing Area Sequence Diagram



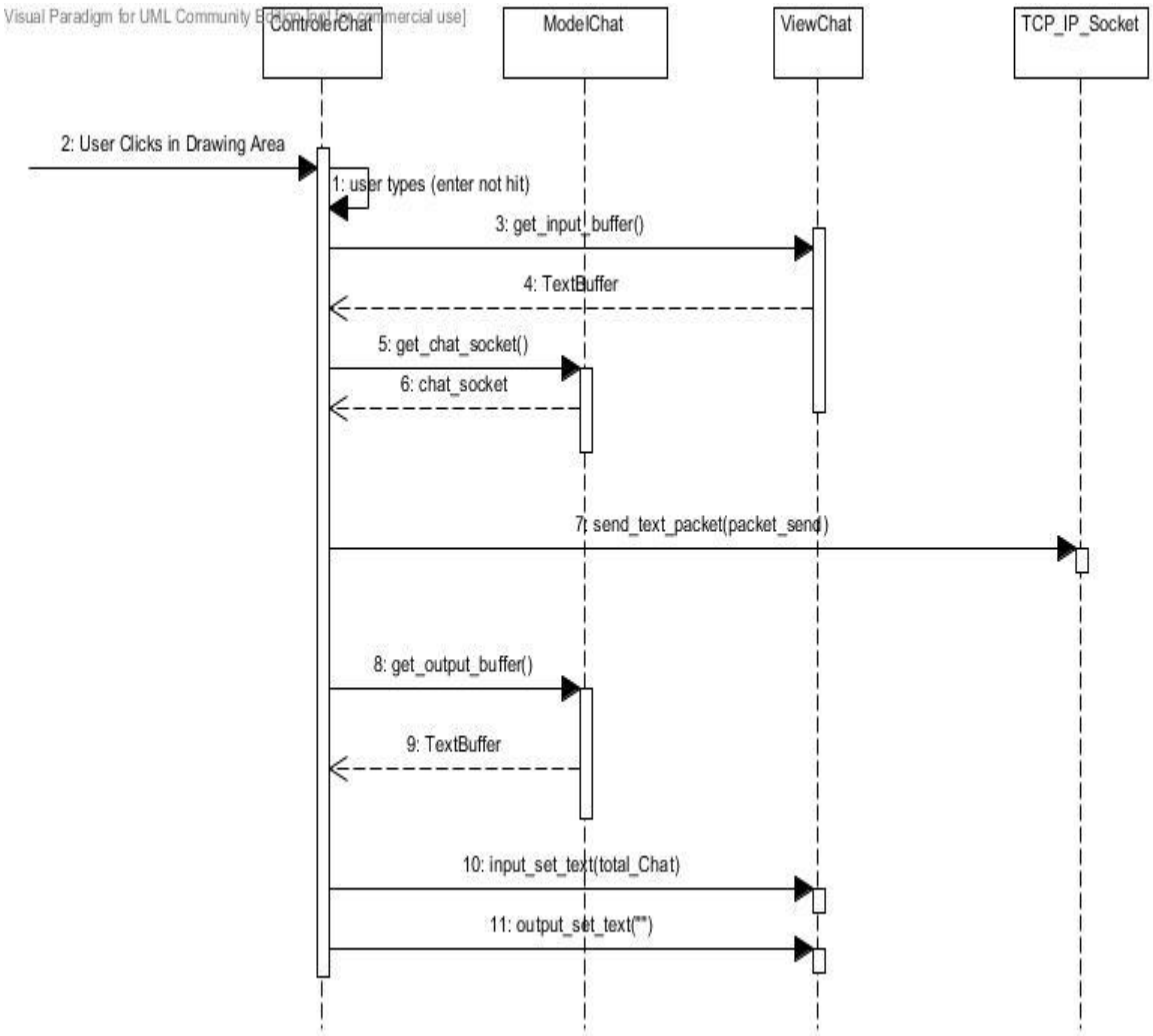
11_d Draw Sequence Diagram



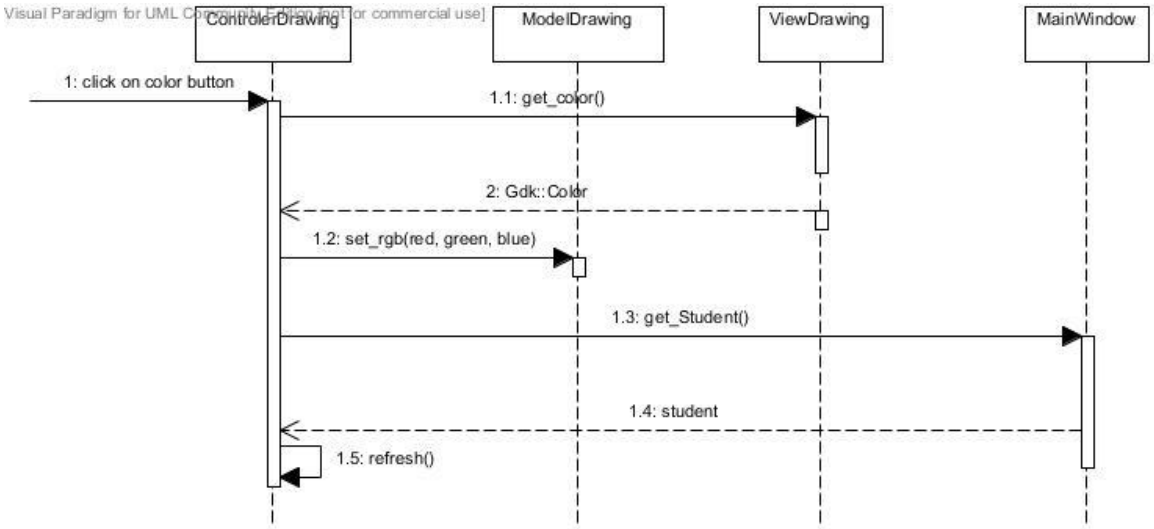
11_e Clear Drawing Area Sequence Diagram



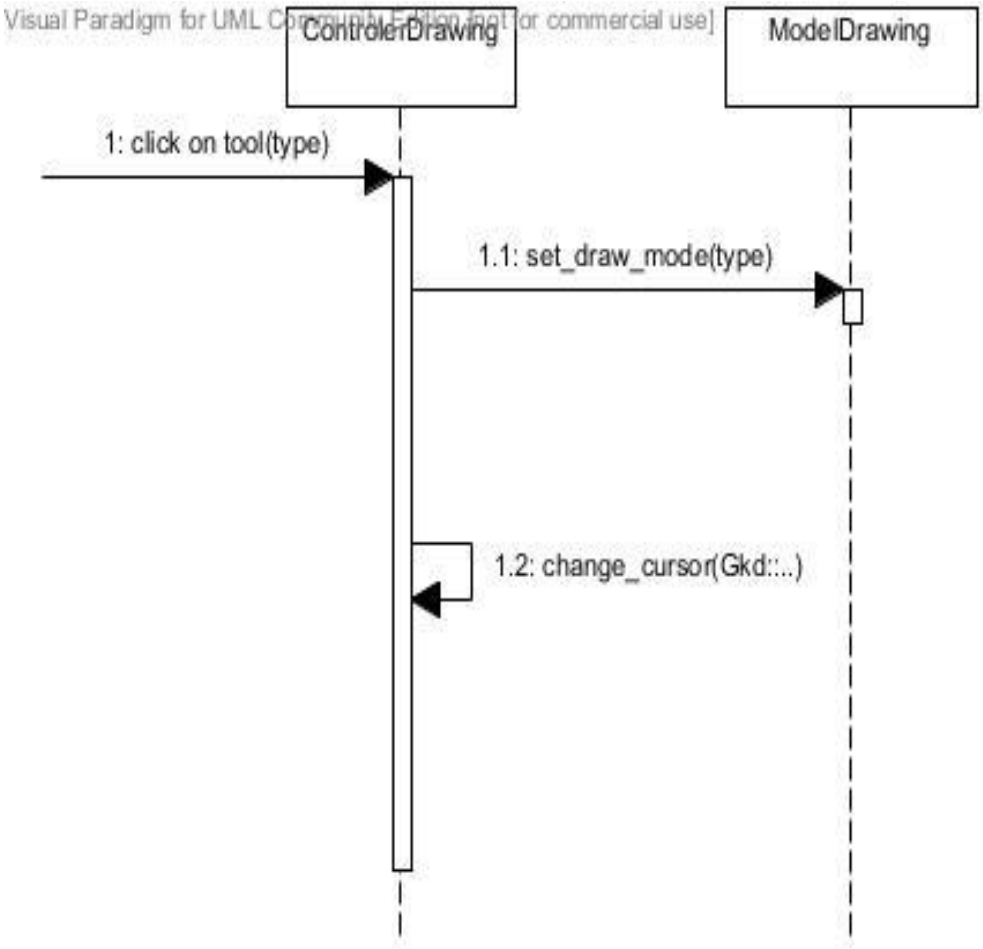
11_f Chat Sequence Diagram



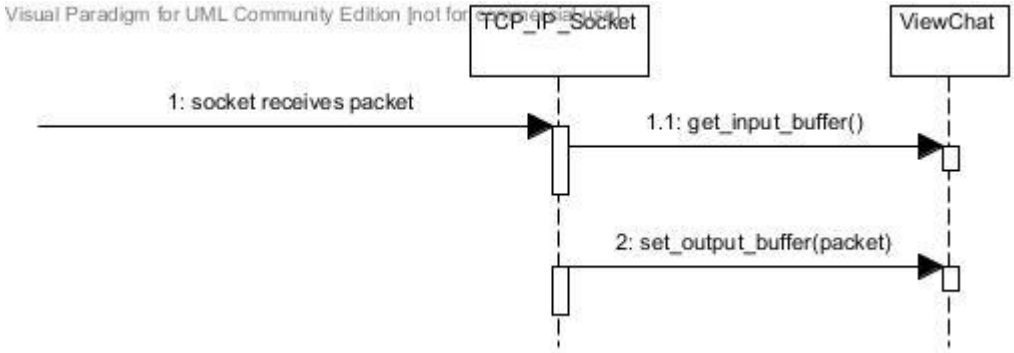
11_g Select Color Sequence Diagram



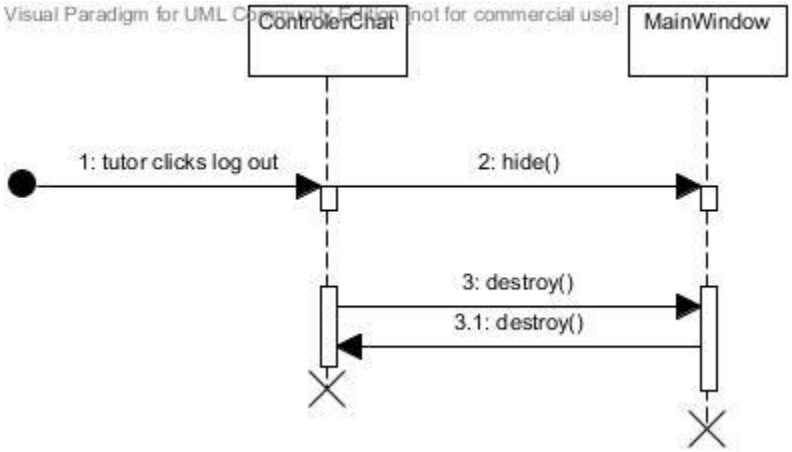
11_h Select Tool Sequence Diagram



11_i Receive Chat Sequence Diagram



11_j Tutor Log-out Sequence Diagram



12. Detailed Class Diagrams

Circle
-type : int -radius : double -theta : double -x_0 : double -y_0 : double -x : double -y : double -sides : double -red : double -green : double -blue : double
+Circle(type : int, radius : double, theta : double, x_0 : double, y_0 : double, x : double, y : double, sides : double, red : double, green : double, blue : double); +~Circle()

+get_type() : int
+get_radius() : double
+get_theta() : double
+get_x_0() : double
+get y_0() : double
+get_x() : double
+get_y() : double
+get_sides() : double
+get_red() : double
+get_green() : double
+get_blue() : double

Controler

+Controler()
+~Controler()
+virtual on_expose_event(event : GdkEventExpose) : bool*
+virtual button_press_event(event : GdkEventButton) : bool*
+virtual button_release_event(event : GdkEventButton) : bool*
+virtual button_notify_event(event : GdkEventMotion) : bool*

ControlerChat

-mdl_Chat : ModelChat*
-view_Chat : ViewChat*
-hwnd : MainWindow*

+ControlerChat()
+ControlerChat(hwnd : MainWindow*, mdl_Chat : ModelChat*, view_Chat : ViewChat*)
+~ControlerChat()
+Set_Sensitive_All(active : bool) : void
+Set_Sensitive_Queue(active : bool) : void
+Set_Sensitive_log_On(active : bool) : void
+Set_Sensitive_log_Off(active : bool) : void
+Set_Sensitive_fontclr_Option(active : bool) : void
+Set_Sensitive_emo_Option(active : bool) : void
+Set_Editable_Output(edit : bool) : void
+Set_Editable_Input(edit : bool) : void
+on_expose_event(event : GdkEventExpose*) : bool
+button_press_event(event : GdkEventButton*) : bool
+button_release_event(event : GdkEventButton*) : bool
+motion_notify_event(event : GdkEventMotion*) : bool
+on_release_create() : void
+key_press_even(event : GdkEvenKey*) : bool
+clear_Intput() : void
+clear_Output() : void
+establish_Socket() : void
+delete_Socket() : void
-open_StudentQueue() : void
-terminate_Program() : void
-open_Login() : void

-check_Student_Queue() : void

ControlerDrawing

-mdl_Draw : ModelDrawing*
-view_Draw ViewDrawing*
-hwnd MainWindow*
-dc : Cairo::RefPtr<Cairo::Context>
-event_state : enum Event_State

+ControlerDrawing()
+ControlerDrawing(hwnd : MainWindow*, mdl_Draw : ModelDrawing*, view_Draw : ViewDrawing)
+~ConstolerDrawing()
+Set_Sensitive_All(active : bool) : void
+Set_Sensitive_sketchBoard(active : bool) : void
+Set_Sensitive_elipse_ToolButton(active : bool) : void
+Set_Sensitive_eraser_ToolButton(active : bool) : void
+Set_Sensitive_text_ToolButton(active : bool) : void
+Set_Sensitive_line_ToolButton(active : bool) : void
+Set_Sensitive_pencil_ToolButton(active : bool) : void
+Set_Sensitive_polygon_ToolButton(active : bool) : void
+Set_Sensitive_rectangle_ToolButton(active : bool) : void
+Set_Sensitive_arrow_ToolButton(active : bool) : void
+Set_Senstive_color_Button(active : bool) : void
+Set_Sensitve_clear_Button(active : bool) : void
+Set_Senstive_refresh_Button(active : bool) : void
+change_cursor() : void
+change_cursor(cursor_Type : Gdk::CursorType) : void
+on_expose_event(event : GdkEventExpose*) : bool
+button_press_event(event : GdkEventButton*) : bool
+button_release_event(event : GdkEventButton*) : bool
+motion_notify_event(event : GdkEventMotion*) : bool
+on_damage(GdkEventExpose* event) : bool
+get_event_state() : bool
-set_Drawing_Mode(type : DRAWINGMODE) : void
-on_realize_create() : void
-on_color_change() : void
-set_dc_color(red : double, green : double, blue : double) : void
-set_dc_color(red : double, green : double, blue : double, trans : double) : void
-set_dc_line_width(width : int) : void
-refresh() : void
-clear() : void
-draw_polygon(x_0 : double, y_0 : double, sides : int, radius : double, angle : double, red : double, green : double, blue : double) : void
-draw_circle(double x_0, double y_0, double radius, double red, double green, double blue) : void
-draw_line(x_0 : double, y_0 : double, y : double, x : double, red : double, green : double, blue : double) : void
-calculate_theta(x_0 : int, y_0 : int, x : int, y : int) : double
-calculate_radius(x_0 : int, y_0 : int, x : int, y : int) : double
-refresh_Grid(width : int, height : int) : void

ControlerQueue

-mdl_Queue : ModelQueue*
-view_Queue : ViewQueue*
-hwnd : MainWindow*

+ControlerQueue()
+ControlerQueue(hwnd : MainWindow*, mdl_Que : ModelQueue*, view_Que : ViewQueue*)
+~ControlerQueue()
+get_Data_From_Server() : void
+get_Selected_Student() : void
+cancel_Selected_Student() : void
+select_Selected_Student() : void
+void Show() : void
+on_expose_event(event : GdkEventExpose*) : bool
+button_press_event(event : GdkEventButton*) : bool
+button_release_event(event : GdkEventButton*) : bool
+motion_notify_event(event : GdkEventMotion*) : bool
-build_Student_Vector(data : char*) : void

Line

-type : int
-radius : double
-theta : double
-x_0 : double
-y_0 : double
-x : double
-y : double
-sides : double
-red : double
-green : double
-blue : double

+Line(type : int, radius : double, theta : double, x_0 : double, y_0 : double, x : double, y : double, sides : double, red : double, green : double, blue : double);
+~Line()
+get_type() : int
+get_radius() : double
+get_theta() : double
+get_x_0() : double
+get y_0() : double
+get_x() : double
+get_y() : double
+get_sides() : double
+get_red() : double
+get_green() : double
+get_blue() : double

Node<T>

-next : Node<T>*

-data : T

+Node()
+Node(data : T)
+~Node()
+getNext() : Node<T>*&
+setNext(node : Node<T>*) : void
+getData() : T

LinkedList<T>

-head : Node<T>*
-size : int

+LinkedList()
+~LinkedList()
+InsertTail(data : T) : void
+InsertHead(data : T) : void
+Delete(data : T) : void
+DeleteHead() : void
+getHead() : Node<T>*
+Display() : void
+Search(data : T) : bool
+getItem(index : int) : T
+getSize() : int

Login_Dialog

-hwnd : MainWindow*
-label_value : Glib::ustring*
#user_layout : Gtk::HBox*
#pswd_layout : Gtk::HBox*
#user_name : Gtk::Label*
#user_name : Gtk::Entry*
#pswd_user : Gtk::Label*
#pswd_entry : Gtk::Entry*
#Ok : Gtk::Button*
#Cancel : Gtk::Button*
#status_label : Gtk::Label*

+Login_Dialog(type : int, hwnd : MainWindow*)
+~Login_Dialog()
+set_status_label(label : Glib::ustring) : void
+get_uname() : Glib::ustring*
+get_psswd() : Glib::ustring*
+set_uname(uname : Glib::ustring*) : void
+set_psswd(psswd : Glib::ustring*) : void
+hash_psswd(psswd : char*) : void
-hash_char(c : char) : char

MainWindow

-socket : TCP_IP_Socket*

-menu : Gtk::Widget* -m_refActionGroup : RefPtr<Gtk::ActionGroup> -m_refUIManager : RefPtr<Gtk::UIManager> -mdl_Draw : ModelDrawing* -view_Draw : ViewDrawing* -cntrl_Draw : ControlerDrawing* -mdl_Chat : ModelChat* -view_Chat : ViewChat* -cntrl_Chat : ControlerChat* -mdl_Que : ModelQueue* -view_Que : ViewQueue* -cntrl_Que : ControlerQueue* -label_frame : Frame* -status_label : Label* -login : Login_Dialog* -tutor : Tutor* -student : Student* -chat_thread : Glib::Thread* -draw_thread : Glib::Thread* #main_HBox : Gtk::HBox* #base_VBox : Gtk::VBox*
+MainWindow(type : int) +~MainWindow() +on_menu_file_new_generic() : void +on_menu_file_quit() : void +get_mdl_Draw() : ModelDrawing* +get_view_Draw() : ViewDrawing* +get_cntrl_Draw() : ControlerDrawing* +get_mdl_Chat() : ModelChat* +get_cntrl_Chat() : ControlerChat* +get_mdl_Que() : ModelQueue* +get_view_Que() : ViewQueue* +get_cntrl_Que() : ControlerQueue* +get_chat_thread() : Glib::Thread* +get_draw_thread() : Glib::Thread* +create_chat_thread() : void +create_draw_thread() : void +join_chat_thread() : void +join_draw_thread() : void +get_status_label() : Label* +set_status_label(label : Glib::ustring) : void +get_socket() : TCP_IP_Socket* +Login_Screen() : int +set_Student(set_std : Student*) : void +get_Student() : Student* +set_Tutor(set_tutor : Tutor*) : void +get_Tutor() : Tutor* +create_Student_Queue() : void +create_Login_Screen() : void +on_realize_create() : void

ModelChat
-hwnd : MainWindow* -chat_socket : TCP_IP_Socket*
+ModelChat() +ModelChat(hwnd : MainWindow*) +~ModelChat() +get_chat_socket() : TCP_IP_Socket* +establish_Socket() : void +delete_Socket() : void

ModelColumns
+m_col_id : Gtk::TreeModelColumn<unsigned int> +m_col_name : Gtk::TreeModelColumn<Glib::ustring> +m_col_subject : Gtk::TreeModelColumn<Glib::ustring> +m_col_time : Gtk::TreeModelColumn<int>
+ModelColumns() +~ModelColumns()

ModelDrawing
-hwnd : MainWindow* -draw_mode : enum DrawingMode -left_Click_x : int -left_Click_y : int -trans : double -red : double -green : double -blue : double -grid : LinkedList<Shape*>*** -grid_Row : int -grid_Col : int +shape : enum Shape_Type
+ModelDrawing() +ModelDrawing(hwnd : MainWindow*) +~ModelDrawing() +get_draw_mode() : int +set_draw_mode(draw_mode : DRAWINGMODE) : void +get_left_Click_x() : int +get_left_Click_y() : int +set_left_Click_x_y(x : int, y : int) : void +get_trans() : double +get_red() : double +get_green() : double +get_blue() : double +set_red(red : double) : void +set_green(green : double) : void +set_blue(blue : double) : void

+set_rgb(red : double, green : double, blue : double) : void
+insert_shape(shape : Shape*) : void
+initilize_Grid(width : int, height : int) : void
+delete_Grid(width : int, height : int) : void
+get_Shape(x : int, y : int, z : int) : Shape*
+print_grid() : void

ModelQueue

-hwnd : MainWindow*
-num_O_Students : int
-student_Queue : LinkedList<Student*>*

+ModelQue()
+ModelQue(hwnd : MainWindow*)
+~ModelQue()
+get_Student(full_Name : Glib::ustring) : Student*
+get_Student_List() : LinkedList<Student*>*
+add_Student(student : Student*) : void
+get_Student(index : int) : Student*

Polygon

-type : int
-radius : double
-theta : double
-x_0 : double
-y_0 : double
-x : double
-y : double
-sides : double
-red : double
-green : double
-blue : double

+Polygon(type : int, radius : double, theta : double, x_0 : double, y_0 : double, x : double, y : double, sides : double, red : double, green : double, blue : double);
+~Polygon()
+get_type() : int
+get_radius() : double
+get_theta() : double
+get_x_0() : double
+get y_0() : double
+get_x() : double
+get_y() : double
+get_sides() : double
+get_red() : double
+get_green() : double
+get_blue() : double

Shape

-type : int *-radius : double* *-theta : double* *-x_0 : double* *-y_0 : double* *-x : double* *-y : double* *-sides : double* *-red : double* *-green : double* *-blue : double*
+Shape(); *+~Shape()* *+get_type() : int* *+get_radius() : double* *+get_theta() : double* *+get_x_0() : double* *+get y_0() : double* *+get_x() : double* *+get_y() : double* *+get_sides() : double* *+get_red() : double* *+get_green() : double* *+get_blue() : double*

Square
-type : int -radius : double -theta : double -x_0 : double -y_0 : double -x : double -y : double -sides : double -red : double -green : double -blue : double
+Square(type : int, radius : double, theta : double, x_0 : double, y_0 : double, x : double, y : double, sides : double, red : double, green : double, blue : double); +~Square() +get_type() : int +get_radius() : double +get_theta() : double +get_x_0() : double +get y_0() : double +get_x() : double +get_y() : double +get_sides() : double +get_red() : double

+get_green() : double
+get_blue() : double

Student

-first_Name : Glib::ustring
-last_Name : Glib::ustring
-subject : Glib::ustring

-time_in_queue : Glib::ustring

+Student(first_Name : Glib::ustring, last_Name : Glib::ustring, subject : Glib::ustring, time_in_queue : Glib::ustring)
+~Student()
+~ModelQue()
+get_first_Name() : Glib::ustring
+get_last_Name() : Glib::ustring
+get_subject_Name() : Glib::ustring
+get_time_in_queue() : Glib::ustring

TCP_IP_Socket

hwnd : MainWindow*
server_name : const char*
socketfd : int
connectfd : int
server_domainname : Glib::ustring*
server_ip : Glib::ustring*
specs : struct addrinfo
results : struct addrinfo*

+TCP_IP_Socket(hwnd : MainWindow*, port : char*t)
+~TCP_IP_Socket()
+set_server_ustring(const char* server_domainname) : void
+set_serverip_ustring(const char* ip) : void
+get_server_domainname_string() : Glib::ustring*
+get_server_ip_string() : Glib::ustring*
+send_packet(char* packet) : int
+receive_packet_login() : char
+receive_packet_student_queue() : const char*
+send_shape_packet(shape : Shape*) : int
+send_text_packet(text_packet : const char*) : int
+recieve_Text_Data(student : Glib::ustring, output : TextView*) : void
+recieve_Draw_Data(x : int) : void
+set_session(SESSION : bool) : void
-establish_server_domainname_string(domain_name : const char*) : void
-establish_server_ip_string() : void

Tokenizer

-chainOtOkens : LinkedList<string>*
-toBeTokenized : string
-word : int

+Tokenizer()
+Tokenizer(tokenize : string)
+Tokenizer(tokenize : string, tokenStop : char)
+~Tokenizer()
+getWord() : int
+nextToken() : string
+hasMoreTokens() : bool
+getToken(token : int) : string

Tutor

-uname : const char*
-psswd : const char*

+Tutor(uname : Glib::ustring*, psswd Glib::ustring*)
+~Tutor()
+get_uname() : const char*
+get_psswd() : const char*
+set_uname(uname : char*)
+set_psswd(psswsd : char*)

ViewChat

-hwnd : MainWindow*
-mdl_Chat : ModelChat*
#right_Vbox : Gtk::Vbox*
#output_Frame : Gtk::Frame*
#rbutton_Frame : Gtk::Frame*
#rbutton_HBox : Gtk::HBox*
#input_Frame : Gtk::Frame*
#text_Options : Gtk::Toolbar*
#scroll_Output : Gtk::ScrolledWindow*
#Output : Gtk::TextView*
#student_Queue : Gtk::Button*
#log_On : Gtk::Button*
#log_Off : Gtk::Button*
#font_Color : Gtk::Image*
#fontclr_Option : Gtk::ToolButton*
#smily_Face : Gtk::Image*
#emo_Option : Gtk::ToolButton*
#scroll_Input : Gtk::ScrolledWindow*
#Input : Gtk::TextView*

+ViewChat()
+ViewChat(hwnd : MainWindow*)
+~ViewChat()
+set_Model(mdl_Chat : ModelChat*) : void
+get_right_VBox() : Gtk::VBox*
+get_student_Queue_Button() : Gtk::Button*
+get_log_On_Button() : Gtk::Button*
+get_log_Off_Button() : Gtk::Button*
+get_fontclr_Option() : Gtk::ToolButton*

+get_emo_Option() : Gtk::ToolButton*
+get_Input() : Gtk::TextView*
+get_Output() : Gtk::TextView*

ViewDrawing

-hwnd : MainWindow*
-mdl_Draw : ModelDrawing*
-cursor : Gtk::Cursor*
#left_Vbox : Gtk::Vbox*
#drawing_Frame : Gtk::Frame*
#tool_DivideBox : Gtk::HBox*
#tool_Vbox : Gtk::VBox*
#ltopbutton_Frame : Gtk::Frame*
#clrbtn_Box : Gtk::HBox*
#sketchBoard : Gtk::DrawingArea*
#ellipse_Image : Gtk::Image*
#ellipse_Image : Gtk::Image*
#ellipse_ToolButton : Gtk::ToolButton*
#eraser_Image : Gtk::Image*
#eraser_ToolButton : Gtk::ToolButton*
#line_Image : Gtk::Image*
#line_ToolButton : Gtk::ToolButton*
#pencil_Image : Gtk::Image*
#pencil_ToolButton : Gtk::ToolButton*
#polygon_Image : Gtk::Image*
#polygon_ToolButton : Gtk::ToolButton*
#rectangle_Image : Gtk::Image*
#rectangle_ToolButton : Gtk::ToolButton*
#arrow_Image : Gtk::Image*
#arrow_ToolButton : Gtk::ToolButton*
#clearSketchBoard : Gtk::Button*
#refresh : Gtk::Button*
#color_Button : Gtk::ColorButton*

+ViewDrawing()
+ViewDrawing(hwnd : MainWindow*)
+~ViewDrawing()
+set_Model(mdl_Draw : ModelDrawing*) : void
+get_left_VBox() : Gtk::VBox*
+get_sketchBoard() : Gtk::DrawingArea*
+get_ellipse_ToolButton() : Gtk::ToolButton*
+get_eraser_ToolButton() : Gtk::ToolButton*
+get_line_ToolButton() : Gtk::ToolButton*
+get_pencil_ToolButton() : Gtk::ToolButton*
+get_polygon_ToolButton() : Gtk::ToolButton*
+get_rectangle_ToolButton() : Gtk::ToolButton*
+get_arrow_ToolButton() : Gtk::ToolButton*
+get_color_Button() : Gtk::ColorButton*
+get_clearSketchBoard() : Gtk::Button*
+get_refresh() : Gtk::Button*

+get_cursor() : Gdk::Cursor*
+create_cursor() : void
+create_cursor(Gdk::CursorType cursor_Type) : void

ViewDrawing
-hwnd : MainWindow* -dialog : Gtk::Window* -main_layout : Gtk::VBox* -mdl_Que : ModelQueue* #notebook_Frame : Gtk::Frame* #student_Window : Gtk::ScrolledWindow* #m_ButtonBox : Gtk::HButtonBox* #m_ButtonSelect : Gtk::Button* #m_Columns : Gtk::ModelColumns* #m_refTreeModel : Glib::RefPtr<Gtk::ListStore> #m_TreeView : Gtk::TreeView* #row : Gtk::TreeModel::Row
+ViewQueue() +ViewQueue(hwnd : MainWindow*) +~ViewQueue() +set_Model(mdl_Que : ModelQueue*) void +get_window() : Gtk::Window* +get_notebook_Frame() : Gtk::Fram* +get_student_Window() : Gtk::ScrolledWindow* +get_Quit_Button() : Gtk::Button* +get_Select_Button() : Gtk::Button* +get_Tree_View() : Gtk::TreeView* +get_Tree_Model_Row() : Gtk::TreeMode::Row& +get_List_Store() : Glib::RefPtr<Gtk::ListStore>& +get_Model_Columns() : ModelColumns* +complete_GUI() : void +get_Selected_Student() : Student*

13. Source Code

```
#ifndef CIRCLE_H_INCLUDED
#define CIRCLE_H_INCLUDED

#include "Shape.h"

class Circle : public Shape
{
public:
    Circle(int type, double radius, double theta, double x_0, double
y_0, double x, double y, double sides, double red, double green, double
blue);
    ~Circle();
    double get_x_0();
    double get_y_0();
```

```
    double get_radius();
    double get_sides();
    int get_type();
    double get_theta();
    double get_x();
    double get_y();
      double get_red();
      double get_green();
      double get_blue();
private:
    double x_0;
    double y_0;
    double x;
    double y;
    double radius;
    double sides;
    int type;
    double theta;
      double red;
      double green;
      double blue;
};

#endif // CIRCLE_H_INCLUDED

#include "Circle.h"

Circle::Circle(int type, double radius, double theta, double x_0,
double y_0, double x, double y, double sides, double red, double green,
double blue)
{
    this->type = type;
    this->radius = radius;
    this->x_0 = x_0;
    this->y_0 = y_0;
    this->sides = sides;
    this->theta = theta;
    this->x;
    this->y;
      this->red = red;
      this->green = green;
      this->blue = blue;
}

Circle::~Circle()
{

}

double Circle::get_radius()
{
    return this->radius;
}

double Circle::get_x_0()
{
```

```
    return this->x_0;
}

double Circle::get_y_0()
{
    return this->y_0;
}

double Circle::get_sides()
{
    return this->sides;
}

int Circle::get_type()
{
    return this->type;
}

double Circle::get_theta()
{
    return this->theta;
}

double Circle::get_x()
{
    return this->x;
}

double Circle::get_y()
{
    return this->y;
}

double Circle::get_red()
{
      return this->red;
}

double Circle::get_green()
{
      return this->green;
}

double Circle::get_blue()
{
      return this->blue;
}



/*
 * Controler.h
 *
 *  Created on: Feb 7, 2010
 *      Author: Matthew
 */
```

```
#ifndef CONTROLER_H_
#define CONTROLER_H_

#include "includes.h"

class Controler
{
public:
      Controler();
      virtual ~Controler();

    virtual bool on_expose_event(GdkEventExpose* event) = 0;
      virtual bool button_press_event(GdkEventButton* event) = 0;
      virtual bool button_release_event(GdkEventButton* event) = 0;
      virtual bool motion_notify_event(GdkEventMotion* event) = 0;
};

#endif

/*
 * Controler.cpp
 *
 *  Created on: Feb 7, 2010
 *      Author: Matthew
 */

#include "Controler.h"

Controler::Controler()
{

}

Controler::~Controler()
{
}
```

```
/*
 * ControlerChat.h
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */
```

```
#ifndef CONTROLERCHAT_H_
#define CONTROLERCHAT_H_

#include "includes.h"
#include "Controler.h"
#include "ModelChat.h"
#include "ViewChat.h"

class MainWindow;

class ControlerChat : public Controler
{
public:
      ControlerChat();
      ControlerChat(MainWindow* hwnd, ModelChat* mdl_Chat, ViewChat*
view_Chat);
      ~ControlerChat();

      void Set_Sensitive_ALL(bool active);
      void Set_Sensitive_Queue_Button(bool active);
      void Set_Sensitive_log_On(bool active);
      void Set_Sensitive_log_Off(bool active);
    void Set_Sensitive_fontclr_Option(bool active);
    void Set_Sensitive_emo_Option(bool active);

    void Set_Editable_ALL(bool edit);
      void Set_Editable_Output(bool edit);
      void Set_Editable_Input(bool edit);

      bool on_expose_event(GdkEventExpose* event);
      bool button_press_event(GdkEventButton* event);
      bool button_release_event(GdkEventButton* event);
      bool motion_notify_event(GdkEventMotion* event);

      void on_realize_create();
      bool key_press_event(GdkEventKey* event);

      void clear_Input();
      void clear_Output();

      void establish_Socket();
      void delete_Socket();

private:
      ModelChat* mdl_Chat;
      ViewChat* view_Chat;
      MainWindow* hwnd;
      void open_StudentQueue();
      void terminate_Program();
      void open_Login();
      void check_Student_Queue();
};

#endif /* CONTROLERCHAT_H_ */
```

```
/*
 * ControlerChat.cpp
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#include "ControlerChat.h"
#include "MainWindow.h"

ControlerChat::ControlerChat()
{

}

ControlerChat::ControlerChat(MainWindow* hwnd, ModelChat* mdl_Chat,
ViewChat* view_Chat)
{
	this->hwnd = hwnd;
	this->mdl_Chat = mdl_Chat;
	this->view_Chat = view_Chat;
	this->view_Chat->set_Model(this->mdl_Chat);
	this->Set_Sensitive_ALL(false);
	this->Set_Editable_ALL(false);
	this->view_Chat->get_student_Queue_Button()-
>signal_clicked().connect(sigc::mem_fun(*this,
&ControlerChat::open_StudentQueue));
	this->view_Chat->get_log_Off_Button()-
>signal_clicked().connect(sigc::mem_fun(*this,
&ControlerChat::terminate_Program));
	this->view_Chat->get_log_On_Button()-
>signal_clicked().connect(sigc::mem_fun(*this,
&ControlerChat::open_Login));
	this->view_Chat->get_Input()-
>signal_realize().connect(sigc::mem_fun(*this,
&ControlerChat::on_realize_create));
}

ControlerChat::~ControlerChat()
{

}

void ControlerChat::Set_Sensitive_ALL(bool active)
{
    //this->Set_Sensitive_Queue_Button(active);
      //this->Set_Sensitive_log_On(active);
      this->Set_Sensitive_log_Off(active);
    this->Set_Sensitive_fontclr_Option(active);
    this->Set_Sensitive_emo_Option(active);
}

void ControlerChat::Set_Editable_ALL(bool edit)
{
    this->Set_Editable_Output(edit);
    this->Set_Editable_Input(edit);
}
```

```
void ControlerChat::Set_Sensitive_Queue_Button(bool active)
{
    this->view_Chat->get_student_Queue_Button()->set_sensitive(active);
}

void ControlerChat::Set_Sensitive_log_On(bool active)
{
    this->view_Chat->get_log_On_Button()->set_sensitive(active);
}

void ControlerChat::Set_Sensitive_log_Off(bool active)
{
    this->view_Chat->get_log_Off_Button()->set_sensitive(active);
}

void ControlerChat::Set_Sensitive_fontclr_Option(bool active)
{
    this->view_Chat->get_fontclr_Option()->set_sensitive(active);
}

void ControlerChat::Set_Sensitive_emo_Option(bool active)
{
    this->view_Chat->get_emo_Option()->set_sensitive(active);
}

void ControlerChat::Set_Editable_Output(bool edit)
{
    this->view_Chat->get_Input()->set_editable(edit);
}

void ControlerChat::Set_Editable_Input(bool edit)
{
    this->view_Chat->get_Output()->set_editable(edit);
}

void ControlerChat::open_StudentQueue()
{
    this->hwnd->get_cntrl_Chat()->Set_Sensitive_ALL(false);
    this->hwnd->get_cntrl_Chat()->Set_Editable_ALL(false);
    this->hwnd->get_cntrl_Draw()->Set_Sensitive_ALL(false);
    this->check_Student_Queue();
    if(this->hwnd->get_mdl_Chat()->get_chat_socket() != NULL)
    {
        this->hwnd->get_mdl_Chat()->get_chat_socket()-
>set_session(false);
    }
    this->hwnd->get_socket()->set_session(false);
    this->hwnd->create_Student_Queue();
}

void ControlerChat::terminate_Program()
{
    cout << "Terminating Program" << endl;
    this->hwnd->hide();
}
```

```
void ControlerChat::open_Login()
{
    if(this->hwnd->get_mdl_Chat()->get_chat_socket() != NULL)
    {
        this->hwnd->get_mdl_Chat()->get_chat_socket()-
>set_session(false);
    }
    this->hwnd->get_socket()->set_session(false);
    this->hwnd->get_cntrl_Chat()->Set_Sensitive_ALL(false);
    this->hwnd->get_cntrl_Chat()->Set_Editable_ALL(false);
    this->hwnd->get_cntrl_Draw()->Set_Sensitive_ALL(false);
    this->hwnd->create_Login_Screen();
    this->check_Student_Queue();
    this->hwnd->create_Student_Queue();
}

void ControlerChat::check_Student_Queue()
{
    if(this->hwnd->get_mdl_Que() == NULL)
    {
    }
    else
    {
        delete this->hwnd->get_mdl_Que();
    }

    if(this->hwnd->get_view_Que() == NULL)
    {
    }
    else
    {
        delete this->hwnd->get_view_Que();
    }

    if(this->hwnd->get_cntrl_Que() == NULL)
    {
    }
    else
    {
        delete this->hwnd->get_cntrl_Que();
    }
}

bool ControlerChat::on_expose_event(GdkEventExpose* event)
{
}

bool ControlerChat::button_press_event(GdkEventButton* event)
{
}

bool ControlerChat::button_release_event(GdkEventButton* event)
{
}

bool ControlerChat::motion_notify_event(GdkEventMotion* event)
{
```

```
}

void ControlerChat::on_realize_create()
{
    this->view_Chat->get_Input()-
>signal_key_press_event().connect(sigc::mem_fun(*this,
&ControlerChat::key_press_event), false);
}

bool ControlerChat::key_press_event(GdkEventKey* event)
{
    bool ret;
    if(event->keyval ==  GDK_Return)
    {
        if(event->state == GDK_SHIFT_MASK)
        {
            ret = false;
        }
        else
        {
            Glib::RefPtr<TextBuffer> output = this->view_Chat-
>get_Input()->get_buffer();
            Glib::ustring tutor_Name = this->hwnd->get_Tutor()-
>get_uname();
            tutor_Name.append(": ");
            const Glib::ustring output_String = output-
>get_text(false);
            tutor_Name.append(output_String);
            tutor_Name.append("\n");
            string packet = output_String;
            const char* packet_send = packet.c_str();
            this->mdl_Chat->get_chat_socket()-
>send_text_packet(packet_send);
            const Glib::ustring tutor_Input = tutor_Name;
            Glib::RefPtr<TextBuffer> input = this->view_Chat-
>get_Output()->get_buffer();
            Glib::ustring input_String = input->get_text(false);
            input_String.append(tutor_Input);
            const Glib::ustring total_Chat = input_String;
            input->set_text(total_Chat);
            output->set_text("");
            ret = true;
        }
    }
    else
    {
        ret = false;
    }
    return ret;
}

void ControlerChat::clear_Input()
{
    Glib::RefPtr<TextBuffer> input = this->view_Chat->get_Output()-
>get_buffer();
    input->set_text("");
}
```

```
void ControlerChat::clear_Output()
{
    Glib::RefPtr<TextBuffer> output = this->view_Chat->get_Input()-
>get_buffer();
    output->set_text("");
}

void ControlerChat::establish_Socket()
{
    this->mdl_Chat->establish_Socket();
}

void ControlerChat::delete_Socket()
{
    this->mdl_Chat->delete_Socket();
}
```

```
/*
 * ControlerDrawing.h
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#ifndef CONTROLERDRAWING_H_
#define CONTROLERDRAWING_H_

#include "includes.h"
#include "Controler.h"
#include "ModelDrawing.h"
#include "ViewDrawing.h"
typedef int DRAWINGMODE;

class MainWindow;

class ControlerDrawing : public Controler
{
public:
	ControlerDrawing();
	ControlerDrawing(MainWindow* hwnd, ModelDrawing* mdl_Draw,
ViewDrawing* view_Draw);
	~ControlerDrawing();

	void Set_Sensitive_ALL(bool active);
	void Set_Sensitive_sketchBoard(bool active);
	void Set_Sensitive_ellipse_ToolButton(bool active);
	void Set_Sensitive_eraser_ToolButton(bool active);
	void Set_Sensitive_text_ToolButton(bool active);
	void Set_Sensitive_line_ToolButton(bool active);
	void Set_Sensitive_pencil_ToolButton(bool active);
	void Set_Sensitive_polygon_ToolButton(bool active);
	void Set_Sensitive_rectangle_ToolButton(bool active);
	void Set_Sensitive_arrow_ToolButton(bool active);
	void Set_Sensitive_color_Button(bool active);
	void Set_Sensitive_clear_Button(bool active);
	void Set_Sensitive_refresh_Button(bool active);

	void change_cursor();
	void change_cursor(Gdk::CursorType cursor_Type);

	bool on_expose_event(GdkEventExpose* event);
	bool button_press_event(GdkEventButton* event);
	bool button_release_event(GdkEventButton* event);
	bool motion_notify_event(GdkEventMotion* event);
	bool on_damage(GdkEventExpose* event);

	int get_event_state();

private:
	ModelDrawing* mdl_Draw;
	ViewDrawing* view_Draw;
	MainWindow* hwnd;

	Cairo::RefPtr<Cairo::Context> dc;
```

```
    void Set_Drawing_Mode(DRAWINGMODE type);
    void on_realize_create();
    void on_color_change();

    void set_dc_color(double red, double green, double blue);
    void set_dc_color(double red, double green, double blue, double
trans);
    void set_dc_line_width(int width);
    void refresh();
    void clear();

    enum Event_State
    {
        NORMAL, MOUSE_DOWN, MOUSE_UP, MOUSE_MOVE, COLOR_SELECTION,
RESIZE
    } event_state;

    void draw_polygon(double x_0, double y_0, int sides, double radius,
double angle, double red, double green, double blue);
    void draw_circle(double x_0, double y_0, double radius, double red,
double green, double blue);
    void draw_line(double x_0, double y_0, double y, double x, double
red, double green, double blue);
    double calculate_theta(int x_0, int y_0, int x, int y);
    double calculate_radius(int x_0, int y_0, int x, int y);
    void refresh_Grid(int width, int height);
};
#endif /* CONTROLERDRAWING_H_ */


/*
 * ControllerDrawing.cpp
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#include "ControlerDrawing.h"
#include "MainWindow.h"
#include "Shape.h"
#include "Circle.h"
#include "Polygon.h"
#include "Line.h"
#include "Square.h"

ControlerDrawing::ControlerDrawing()
{

}

ControlerDrawing::ControlerDrawing(MainWindow* hwnd, ModelDrawing*
mdl_Draw, ViewDrawing* view_Draw)
{
      this->hwnd = hwnd;
      this->mdl_Draw = mdl_Draw;
```

```
      this->view_Draw = view_Draw;
      this->view_Draw->set_Model(this->mdl_Draw);
      this->mdl_Draw->set_draw_mode(6);
      this->view_Draw->get_pencil_ToolButton()-
>signal_clicked().connect(sigc::bind<int>(sigc::mem_fun(*this,
&ControlerDrawing::Set_Drawing_Mode), 0));
    this->view_Draw->get_eraser_ToolButton()-
>signal_clicked().connect(sigc::bind<int>(sigc::mem_fun(*this,
&ControlerDrawing::Set_Drawing_Mode), 1));
    this->view_Draw->get_rectangle_ToolButton()-
>signal_clicked().connect(sigc::bind<int>(sigc::mem_fun(*this,
&ControlerDrawing::Set_Drawing_Mode), 2));
    this->view_Draw->get_polygon_ToolButton()-
>signal_clicked().connect(sigc::bind<int>(sigc::mem_fun(*this,
&ControlerDrawing::Set_Drawing_Mode), 3));
    this->view_Draw->get_ellipse_ToolButton()-
>signal_clicked().connect(sigc::bind<int>(sigc::mem_fun(*this,
&ControlerDrawing::Set_Drawing_Mode), 4));
    this->view_Draw->get_line_ToolButton()-
>signal_clicked().connect(sigc::bind<int>(sigc::mem_fun(*this,
&ControlerDrawing::Set_Drawing_Mode), 5));
    this->view_Draw->get_arrow_ToolButton()-
>signal_clicked().connect(sigc::bind<int>(sigc::mem_fun(*this,
&ControlerDrawing::Set_Drawing_Mode), 6));
    this->view_Draw->get_refresh()-
>signal_clicked().connect(sigc::mem_fun(*this,
&ControlerDrawing::refresh));
    this->view_Draw->get_clearSketchBoard()-
>signal_clicked().connect(sigc::mem_fun(*this,
&ControlerDrawing::clear));
    this->view_Draw->get_sketchBoard()-
>signal_realize().connect(sigc::mem_fun(*this,
&ControlerDrawing::on_realize_create));
    this->event_state = NORMAL;
    this->view_Draw->get_color_Button()-
>signal_damage_event().connect(sigc::mem_fun(*this,
&ControlerDrawing::on_damage), false);
    this->view_Draw->get_color_Button()-
>signal_color_set().connect(sigc::mem_fun(*this,
&ControlerDrawing::on_color_change));
}

ControlerDrawing::~ControlerDrawing()
{

}

void ControlerDrawing::Set_Sensitive_ALL(bool active)
{
      this->Set_Sensitive_sketchBoard(active);
      this->Set_Sensitive_ellipse_ToolButton(active);
      this->Set_Sensitive_eraser_ToolButton(active);
      this->Set_Sensitive_line_ToolButton(active);
      this->Set_Sensitive_pencil_ToolButton(active);
      this->Set_Sensitive_polygon_ToolButton(active);
      this->Set_Sensitive_rectangle_ToolButton(active);
    this->view_Draw->get_arrow_ToolButton()->set_sensitive(active);
```

```
        this->Set_Sensitive_color_Button(active);
        this->Set_Sensitive_refresh_Button(active);
        this->Set_Sensitive_clear_Button(active);
}

void ControlerDrawing::Set_Sensitive_sketchBoard(bool active)
{
    this->view_Draw->get_sketchBoard()->set_sensitive(active);
}

void ControlerDrawing::Set_Sensitive_ellipse_ToolButton(bool active)
{
    this->view_Draw->get_ellipse_ToolButton()->set_sensitive(active);
}

void ControlerDrawing::Set_Sensitive_eraser_ToolButton(bool active)
{
    this->view_Draw->get_eraser_ToolButton()->set_sensitive(active);
}

void ControlerDrawing::Set_Sensitive_line_ToolButton(bool active)
{
    this->view_Draw->get_line_ToolButton()->set_sensitive(active);
}

void ControlerDrawing::Set_Sensitive_pencil_ToolButton(bool active)
{
    this->view_Draw->get_pencil_ToolButton()->set_sensitive(active);
}

void ControlerDrawing::Set_Sensitive_polygon_ToolButton(bool active)
{
    this->view_Draw->get_polygon_ToolButton()->set_sensitive(active);
}

void ControlerDrawing::Set_Sensitive_rectangle_ToolButton(bool active)
{
    this->view_Draw->get_rectangle_ToolButton()->set_sensitive(active);
}

void ControlerDrawing::Set_Sensitive_arrow_ToolButton(bool active)
{
    this->view_Draw->get_arrow_ToolButton()->set_sensitive(active);
}

void ControlerDrawing::Set_Sensitive_color_Button(bool active)
{
    this->view_Draw->get_color_Button()->set_sensitive(active);
}

void ControlerDrawing::Set_Sensitive_clear_Button(bool active)
{
    this->view_Draw->get_clearSketchBoard()->set_sensitive(active);
}

void ControlerDrawing::Set_Sensitive_refresh_Button(bool active)
{
```

```
    this->view_Draw->get_refresh()->set_sensitive(active);
}

void ControlerDrawing::Set_Drawing_Mode(DRAWINGMODE type)
{
    this->mdl_Draw->set_draw_mode(type);
    switch(type)
    {
        case 0:
        {
            this->change_cursor(Gdk::PENCIL);
            break;
        }
        case 1:
        {
            this->change_cursor(Gdk::CIRCLE);
            break;
        }
        case 2:
        {
            this->change_cursor(Gdk::CROSSHAIR);
            break;
        }
        case 3:
        {
            this->change_cursor(Gdk::CROSSHAIR);
            break;
        }
        case 4:
        {
            this->change_cursor(Gdk::CROSSHAIR);
            break;
        }
        case 5:
        {
            this->change_cursor(Gdk::CROSSHAIR);
            break;
        }
        case 6:
        {
            this->change_cursor(Gdk::CROSSHAIR);
            break;
        }
        case 7:
        {
            this->change_cursor();
            break;
        }
        default:
        {
            exit(0);
        }
    }
}

int ControlerDrawing::get_event_state()
{
```

```
    return this->event_state;
}

void ControlerDrawing::set_dc_color(double red, double green, double
blue)
{
    double dRed = red/65535;
    double dGreen = green/65535;
    double dBlue = blue/65535;
    this->dc->set_source_rgb(dRed, dGreen, dBlue);
}

void ControlerDrawing::set_dc_color(double red, double green, double
blue, double trans)
{
    this->dc->set_source_rgba(red, green, blue, trans);
}

void ControlerDrawing::set_dc_line_width(int width)
{
    this->dc->set_line_width(width);
}

void ControlerDrawing::on_realize_create()
{
    cout << "*++++++++++++++++++++++++++++++++++++++++++++Canvas
Created*++++++++++++++++++++++++++++++++++++++++++++" << endl;
    this->view_Draw->get_sketchBoard()-
>signal_expose_event().connect(sigc::mem_fun(*this,
&ControlerDrawing::on_expose_event), false);
      this->view_Draw->get_sketchBoard()-
>signal_button_press_event().connect(sigc::mem_fun(*this,
&ControlerDrawing::button_press_event), false);
      this->view_Draw->get_sketchBoard()-
>signal_button_release_event().connect(sigc::mem_fun(*this,
&ControlerDrawing::button_release_event), false);
      this->view_Draw->get_sketchBoard()-
>signal_motion_notify_event().connect(sigc::mem_fun(*this,
&ControlerDrawing::motion_notify_event), false);
      this->view_Draw->get_sketchBoard()-
>add_events(Gdk::ALL_EVENTS_MASK);
    int draw_mode = this->mdl_Draw->get_draw_mode();
    this->change_cursor();
    this->dc = this->view_Draw->get_sketchBoard()->get_window()-
>create_cairo_context();
    this->on_color_change();
    this->set_dc_line_width(1.0);
    this->set_dc_color(this->mdl_Draw->get_red(), this->mdl_Draw-
>get_green(), this->mdl_Draw->get_blue(), 1);
    Gtk::Allocation allocation = this->view_Draw->get_sketchBoard()-
>get_allocation();
    int width = allocation.get_width();
    int height = allocation.get_height();
    this->mdl_Draw->initilize_Grid(3280, 2048);
}

void ControlerDrawing::change_cursor()
```

```
{
    this->view_Draw->create_cursor();
    Glib::RefPtr<Gdk::Window> window = this->view_Draw-
>get_sketchBoard()->get_window();
    window->set_cursor(*(this->view_Draw->get_cursor()));
}

void ControlerDrawing::change_cursor(Gdk::CursorType cursor_Type)
{
    this->view_Draw->create_cursor(cursor_Type);
    Glib::RefPtr<Gdk::Window> window = this->view_Draw-
>get_sketchBoard()->get_window();
    window->set_cursor(*(this->view_Draw->get_cursor()));
}

void ControlerDrawing::on_color_change()
{
    this->event_state = COLOR_SELECTION;
    Gdk::Color color = this->view_Draw->get_color_Button()-
>get_color();
    gushort red = color.get_red();
    gushort green = color.get_green();
    gushort blue = color.get_blue();
    double iRed = (double)red;
    double iGreen = (double)green;
    double iBlue = (double)blue;
    this->mdl_Draw->set_rgb(iRed/65535, iGreen/65535, iBlue/65535);
    this->set_dc_color(iRed, iGreen, iBlue);
    if(this->hwnd->get_Student() != NULL)
    {
        this->refresh();
    }
}

bool ControlerDrawing::on_damage(GdkEventExpose* event)
{
}

bool ControlerDrawing::on_expose_event(GdkEventExpose* event)
{
    Gtk::Allocation allocation = this->view_Draw->get_sketchBoard()-
>get_allocation();
    int width = allocation.get_width();
    int height = allocation.get_height();
    this->refresh_Grid(width, height);
}

bool ControlerDrawing::button_press_event(GdkEventButton* event)
{
    int x;
    int y;
    if((event->button) == 1)
    {
        this->event_state = MOUSE_DOWN;
        if(this->hwnd->get_Student() != NULL)
        {
            if(this->mdl_Draw->get_draw_mode() == 0)
```

```
            {
                view_Draw->get_sketchBoard()->get_pointer(x, y);
                this->dc->move_to(x, y);
                this->mdl_Draw->set_left_Click_x_y(x, y);
            }
            if(this->mdl_Draw->get_draw_mode() == 1)
            {
            }
            if(this->mdl_Draw->get_draw_mode() == 2)
            {
                view_Draw->get_sketchBoard()->get_pointer(x, y);
                this->mdl_Draw->set_left_Click_x_y(x, y);
            }
            if(this->mdl_Draw->get_draw_mode() == 3)
            {
                view_Draw->get_sketchBoard()->get_pointer(x, y);
                this->mdl_Draw->set_left_Click_x_y(x, y);
            }
            if(this->mdl_Draw->get_draw_mode() == 4)
            {
                view_Draw->get_sketchBoard()->get_pointer(x, y);
                this->mdl_Draw->set_left_Click_x_y(x, y);
            }
            if(this->mdl_Draw->get_draw_mode() == 5)
            {
                view_Draw->get_sketchBoard()->get_pointer(x, y);
                this->dc->move_to(x, y);
                this->mdl_Draw->set_left_Click_x_y(x, y);
            }
        }
    }
}

bool ControlerDrawing::button_release_event(GdkEventButton* event)
{
    int x;
    int y;
    view_Draw->get_sketchBoard()->get_pointer(x, y);
    cout << " " << endl;
    if((event->button) == 1)
    {
        this->event_state = MOUSE_UP;
        if(this->hwnd->get_Student() != NULL)
        {
            if(this->mdl_Draw->get_draw_mode() == 0)
            {
                double x_0 = (double)this->mdl_Draw-
>get_left_Click_x();
                double y_0 = (double)this->mdl_Draw-
>get_left_Click_y();
                double theta = this->calculate_theta(x_0, y_0, x, y);
                double radius = this->calculate_radius(x_0, y_0, x, y);
                this->dc->line_to(x, y);
                this->dc->stroke();
                this->mdl_Draw->set_left_Click_x_y(x, y);
```

```
                Line* line = new Line(0, radius, theta, x_0, y_0, x, y,
1, this->mdl_Draw->get_red(), this->mdl_Draw->get_green(), this-
>mdl_Draw->get_blue());
                this->hwnd->get_socket()->send_shape_packet(line);
                this->mdl_Draw->insert_shape(line);
            }
            if(this->mdl_Draw->get_draw_mode() == 1)
            {

            }
            if(this->mdl_Draw->get_draw_mode() == 2)
            {
                double x_0 = (double)this->mdl_Draw-
>get_left_Click_x();
                double y_0 = (double)this->mdl_Draw-
>get_left_Click_y();
                double theta = this->calculate_theta(x_0, y_0, x, y);
                double radius = this->calculate_radius(x_0, y_0, x, y);
                this->draw_polygon(x_0, y_0, 4, radius, theta, this-
>mdl_Draw->get_red(), this->mdl_Draw->get_green(), this->mdl_Draw-
>get_blue());
                Square* square = new Square(1, radius, theta, x_0, y_0,
x, y, 4, this->mdl_Draw->get_red(), this->mdl_Draw->get_green(), this-
>mdl_Draw->get_blue());
                this->hwnd->get_socket()->send_shape_packet(square);
                this->mdl_Draw->insert_shape(square);
            }
            if(this->mdl_Draw->get_draw_mode() == 3)
            {
                double x_0 = (double)this->mdl_Draw-
>get_left_Click_x();
                double y_0 = (double)this->mdl_Draw-
>get_left_Click_y();
                double theta = this->calculate_theta(x_0, y_0, x, y);
                double radius = this->calculate_radius(x_0, y_0, x, y);
                int sides = 5;
                this->draw_polygon(x_0, y_0, sides, radius, theta,
this->mdl_Draw->get_red(), this->mdl_Draw->get_green(), this->mdl_Draw-
>get_blue());
                Polygon* polygon = new Polygon(2, radius, theta, x_0,
y_0, x, y, sides, this->mdl_Draw->get_red(), this->mdl_Draw-
>get_green(), this->mdl_Draw->get_blue());
                this->hwnd->get_socket()->send_shape_packet(polygon);
                this->mdl_Draw->insert_shape(polygon);
            }
            if(this->mdl_Draw->get_draw_mode() == 4)
            {
                double x_0 = (double)this->mdl_Draw-
>get_left_Click_x();
                double y_0 = (double)this->mdl_Draw-
>get_left_Click_y();
                double theta = this->calculate_theta(x_0, y_0, x, y);
                double radius = this->calculate_radius(x_0, y_0, x, y);
                this->draw_circle(x_0, y_0, radius, this->mdl_Draw-
>get_red(), this->mdl_Draw->get_green(), this->mdl_Draw->get_blue());
```

```
                Circle* circle = new Circle(3, radius, theta, x_0, y_0,
x, y, -1, this->mdl_Draw->get_red(), this->mdl_Draw->get_green(), this-
>mdl_Draw->get_blue());
                this->hwnd->get_socket()->send_shape_packet(circle);
                this->mdl_Draw->insert_shape(circle);
            }
            if(this->mdl_Draw->get_draw_mode() == 5)
            {
                double x_0 = (double)this->mdl_Draw-
>get_left_Click_x();
                double y_0 = (double)this->mdl_Draw-
>get_left_Click_y();
                double theta = this->calculate_theta(x_0, y_0, x, y);
                double radius = this->calculate_radius(x_0, y_0, x, y);
                this->dc->line_to(x, y);
                this->dc->stroke();
                this->mdl_Draw->set_left_Click_x_y(x, y);
                Line* line = new Line(0, radius, theta, x_0, y_0, x, y,
1, this->mdl_Draw->get_red(), this->mdl_Draw->get_green(), this-
>mdl_Draw->get_blue());
                this->hwnd->get_socket()->send_shape_packet(line);
                this->mdl_Draw->insert_shape(line);
            }
        }
    }
}

bool ControlerDrawing::motion_notify_event(GdkEventMotion* event)
{
    int x;
      int y;
      view_Draw->get_sketchBoard()->get_pointer(x, y);

      if(this->hwnd->get_Student() != NULL)
      {
          if(this->event_state == MOUSE_DOWN)
          {
              if(this->mdl_Draw->get_draw_mode() == 0)
              {
                  double x_0 = (double)this->mdl_Draw-
>get_left_Click_x();
                double y_0 = (double)this->mdl_Draw-
>get_left_Click_y();
                double theta = this->calculate_theta(x_0, y_0, x, y);
                double radius = this->calculate_radius(x_0, y_0, x, y);
                this->dc->line_to(x, y);
                this->dc->stroke();
                this->dc->move_to(x, y);
                this->mdl_Draw->set_left_Click_x_y(x, y);
                Line* line = new Line(0, radius, theta, x_0, y_0, x, y,
1, this->mdl_Draw->get_red(), this->mdl_Draw->get_green(), this-
>mdl_Draw->get_blue());
                cout << "Passing Socket a Line" << endl;
                //this->hwnd->get_socket()->send_shape_packet(line);
                this->mdl_Draw->insert_shape(line);
              }
              if(this->mdl_Draw->get_draw_mode() == 1)
```

```
            {
                Glib::RefPtr<Gdk::Window> window = this->view_Draw-
>get_sketchBoard()->get_window();
                Gdk::Rectangle rect(x, y, 20, 20);
                window->invalidate_rect(rect, false);
            }
          }
      }
    char* status = (char*)malloc(256);
      sprintf(status, "(%d, %d)", x, y);
      Glib::ustring stat(status);
    this->hwnd->set_status_label(stat);
      free(status);
}

void ControlerDrawing::draw_polygon(double x_0, double y_0, int sides,
double radius, double angle, double red, double green, double blue)
{
    double y_i = y_0 + radius * sin(angle);
    double x_i = x_0 + radius * cos(angle);
    this->dc->move_to(x_i, y_i);
    double delta_angle = ((2 * M_PI)/sides);
    for(int i = 2; i <= sides + 1; i++)
    {
        angle = angle + delta_angle;
        while(angle > 2 * M_PI)
        {
            angle = angle -2 * M_PI;
        }
        y_i = y_i + radius * sin(angle);
        x_i = x_i + radius * cos(angle);
        this->dc->line_to(x_i, y_i);
        this->dc->move_to(x_i, y_i);
    }
    this->dc->stroke();
}

void ControlerDrawing::draw_line(double x_0, double y_0, double x,
double y, double red, double green, double blue)
{
    this->dc->move_to((int)x_0, (int)y_0);
    this->dc->line_to((int)x, (int)y);
    this->dc->stroke();
}

void ControlerDrawing::draw_circle(double x_0, double y_0, double
radius, double red, double green, double blue)
{
    this->dc->arc(x_0, y_0, radius, 0.0, 2.0 * M_PI);
    this->dc->stroke();
}

double ControlerDrawing::calculate_radius(int x_0, int y_0, int x, int
y)
{
    double delta_x_0 = (double)x - x_0;
    double delta_y_0 = (double)y - y_0;
```

```
    double radius = sqrt(pow(delta_x_0, 2.0) + pow(delta_y_0, 2.0));
    return radius;
}

void ControlerDrawing::refresh_Grid(int width, int height)
{
    int z = 0;
    for(int x = 0; x < width; x++)
    {
        for(int y = 0; y < height; y++)
        {
            Shape* temp;
            while((temp = this->mdl_Draw->get_Shape(x, y, z)) != NULL)
            {
                        this->dc->save();
                        this->dc->set_source_rgb(temp->get_red(), temp-
>get_green(), temp->get_blue());
                if((int)temp->get_type() == 0)
                {
                    this->draw_line(temp->get_x_0(), temp->get_y_0(),
temp->get_x(), temp->get_y(), temp->get_red(), temp->get_green(), temp-
>get_blue());
                }
                else if((int)temp->get_type() == 1)
                {
                    this->draw_polygon(temp->get_x_0(), temp-
>get_y_0(), temp->get_sides(), temp->get_radius(), temp->get_theta(),
temp->get_red(), temp->get_green(), temp->get_blue());
                }
                else if((int)temp->get_type() == 2)
                {
                    this->draw_polygon(temp->get_x_0(), temp-
>get_y_0(), temp->get_sides(), temp->get_radius(), temp->get_theta(),
temp->get_red(), temp->get_green(), temp->get_blue());
                }
                else if((int)temp->get_type() == 3)
                {
                                this->draw_circle(temp->get_x_0(), temp-
>get_y_0(), temp->get_radius(), temp->get_red(), temp->get_green(),
temp->get_blue());
                }
                z++;
                        this->dc->restore();
            }
            z = 0;
        }
    }
}

double ControlerDrawing::calculate_theta(int x_0, int y_0, int x, int
y)
{
    double delta_x_0 = (double)x - x_0;
    double delta_y_0 = (double)y - y_0;
    double theta = atan(delta_y_0 / delta_x_0);
    if(delta_x_0 < 0 && delta_y_0 < 0)
    {
```

```
        theta = theta + M_PI;
    }
    else if(delta_x_0 < 0)
    {
        theta = theta + (M_PI);
    }
    else if(delta_y_0 < 0)
    {
        theta = theta + 2 * M_PI;
    }
    else if(delta_x_0 == 0)
    {
        theta = 0;
    }
    return theta;
}

void ControlerDrawing::refresh()
{
    Gtk::Allocation allocation = this->view_Draw->get_sketchBoard()-
>get_allocation();
    int width = allocation.get_width();
    int height = allocation.get_height();
    this->refresh_Grid(width, height);
}

void ControlerDrawing::clear()
{
    this->mdl_Draw->delete_Grid(3280, 2048);
    this->mdl_Draw->initilize_Grid(3280, 2048);
    Gtk::Allocation allocation = this->view_Draw->get_sketchBoard()-
>get_allocation();
    int width = allocation.get_width();
    int height = allocation.get_height();
    Glib::RefPtr<Gdk::Window> window = this->view_Draw-
>get_sketchBoard()->get_window();
    Gdk::Rectangle rect(0, 0, width, height);
    window->invalidate_rect(rect, false);
}
```

```
/*
 * ControlerQueue.h
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#ifndef CONTROLERQUEUE_H_
#define CONTROLERQUEUE_H_

#include "includes.h"
#include "Controler.h"
#include "ModelQueue.h"
#include "ViewQueue.h"
#include "Tokenizer.h"

class MainWindow;

class ControlerQueue : public Controler
{
public:
	ControlerQueue();
	ControlerQueue(MainWindow* hwnd, ModelQueue* mdl_Que, ViewQueue*
view_Que);
	~ControlerQueue();
	void get_Data_From_Server();
	void get_Selected_Student();
	void cancel_Selected_Student();
	void select_Selected_Student();
	void Show();

	bool on_expose_event(GdkEventExpose* event);
	bool button_press_event(GdkEventButton* event);
	bool button_release_event(GdkEventButton* event);
	bool motion_notify_event(GdkEventMotion* event);


private:
	ModelQueue* mdl_Que;
	ViewQueue* view_Que;
	MainWindow* hwnd;
	void build_Student_Vector(char* data);
};

#endif /* CONTROLERQUEUE_H_ */

/*
 * ControlerQueue.cpp
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#include "ControlerQueue.h"
#include "MainWindow.h"
```

```
ControlerQueue::ControlerQueue()
{

}

ControlerQueue::ControlerQueue(MainWindow* hwnd, ModelQueue* mdl_Queue,
ViewQueue* view_Queue)
{
      this->hwnd = hwnd;
      this->mdl_Que = mdl_Queue;
      this->view_Que = view_Queue;
      this->view_Que->set_Model(this->mdl_Que);
}

ControlerQueue::~ControlerQueue()
{
}

void ControlerQueue::get_Data_From_Server()
{
    char* header = (char*)malloc(4);
      sprintf(header, "%d", 2, 4);
      this->hwnd->get_socket()->send_packet(header);
      const char* student_Queue_Data = (this->hwnd->get_socket()-
>receive_packet_student_queue());
      char* parse = const_cast<char*>(student_Queue_Data);
      char* student_Que = "Matthew Hoggan Physicis 2:7:30 Andrew
Hamilton Math 24:0:0 Alex David Selebesy 0:0:1 Vahe Margoussian
Automata 2:10:23 Daniel Serry Math 2:10:23";
      //this->build_Student_Vector(parse);
      this->build_Student_Vector(student_Que);
}

void ControlerQueue::build_Student_Vector(char* data)
{
    string tokens(data);
    Tokenizer* t = new Tokenizer(tokens);
    while(t->hasMoreTokens())
    {
        Glib::ustring* firstName = NULL;
        Glib::ustring* lastName = NULL;
        Glib::ustring* subject = NULL;
        Glib::ustring* time_in_Queue = NULL;
        if(t->hasMoreTokens())
        {
            firstName = new Glib::ustring(t->nextToken());
        }
        if(t->hasMoreTokens())
        {
            lastName = new Glib::ustring(t->nextToken());
        }
        if(t->hasMoreTokens())
        {
            subject = new Glib::ustring(t->nextToken());
        }
        if(t->hasMoreTokens())
        {
```

```
            time_in_Queue = new Glib::ustring(t->nextToken());
        }
        if(firstName != NULL && lastName != NULL && subject != NULL &&
time_in_Queue != NULL)
        {
            Student* std = new Student(*(firstName), *(lastName),
*(subject), *(time_in_Queue));
            this->mdl_Que->add_Student(std);
            delete std;
        }
    }
    this->view_Que->complete_GUI();
    this->view_Que->get_Quit_Button()-
>signal_clicked().connect(sigc::mem_fun(*this,
&ControlerQueue::cancel_Selected_Student));
    this->view_Que->get_Select_Button()-
>signal_clicked().connect(sigc::mem_fun(*this,
&ControlerQueue::select_Selected_Student));
    delete t;
}

void ControlerQueue::cancel_Selected_Student()
{
    this->view_Que->get_window()->hide();
    //this->hwnd->get_cntrl_Chat()->Set_Sensitive_Queue_Button(true);
    this->hwnd->get_cntrl_Chat()->Set_Sensitive_log_Off(true);
    //this->hwnd->get_cntrl_Chat()->Set_Sensitive_log_On(true);
    if(this->hwnd->get_mdl_Chat()->get_chat_socket() != NULL)
    {
        this->hwnd->get_mdl_Chat()->get_chat_socket()-
>set_session(false);
        if(this->hwnd->get_chat_thread() != NULL)
        {
            //this->hwnd->join_chat_thread();
        }
    }
    this->hwnd->get_socket()->set_session(false);
    if(this->hwnd->get_draw_thread() != NULL)
    {
        //this->hwnd->join_draw_thread();
    }
    cout << "Threads removed" << endl;
}

void ControlerQueue::select_Selected_Student()
{
    Glib::RefPtr<Gtk::TreeSelection> refTreeSelection = this->view_Que-
>get_Tree_View()->get_selection();
    TreeModel::iterator iter = refTreeSelection->get_selected();
    if(iter)
    {
        TreeModel::Row row = *(iter);
        Glib::ustring uname = row[this->view_Que->get_Model_Columns()-
>m_col_name];
        std::string name(uname);
        Tokenizer* t = new Tokenizer(name);
        Glib::ustring firstName = t->getToken(1);
```

```
        Glib::ustring lastName = t->getToken(2);
        Glib::ustring usubject = row[this->view_Que-
>get_Model_Columns()->m_col_subject];
        Glib::ustring time("0");
        Student selected(firstName, lastName, usubject, time);
        this->hwnd->set_Student(&selected);
        this->view_Que->get_window()->hide();
        string Name = t->getToken(1).append(" ");
        Name = Name.append(t->getToken(2));
        char* send = &Name[0];
        char packet[] = "3";
        this->hwnd->get_socket()->send_packet(packet);
        this->hwnd->get_socket()->send_packet(send);
        this->hwnd->get_socket()->send_packet("\n");
        delete t;
        this->view_Que->get_window()->hide();
        this->hwnd->get_cntrl_Chat()->Set_Sensitive_ALL(true);
        this->hwnd->get_cntrl_Chat()->Set_Editable_Output(true);
        this->hwnd->get_cntrl_Draw()->Set_Sensitive_ALL(true);
            this->hwnd->get_mdl_Draw()->delete_Grid(3280, 2048);
          this->hwnd->get_mdl_Draw()->initilize_Grid(3280, 2048);
            Gtk::Allocation allocation = this->hwnd->get_view_Draw()-
>get_sketchBoard()->get_allocation();
            int width = allocation.get_width();
            int height = allocation.get_height();
         Glib::RefPtr<Gdk::Window> window = this->hwnd-
>get_view_Draw()->get_sketchBoard()->get_window();
            Gdk::Rectangle rect(0, 0, width, height);
            window->invalidate_rect(rect, false);
            this->hwnd->get_cntrl_Chat()->clear_Input();
            this->hwnd->get_cntrl_Chat()->clear_Output();
        this->hwnd->get_mdl_Chat()->get_chat_socket()-
>set_session(true);
        this->hwnd->create_chat_thread();
        this->hwnd->get_socket()->set_session(true);
        this->hwnd->create_draw_thread();
    }
}

bool ControlerQueue::on_expose_event(GdkEventExpose* event)
{
}

bool ControlerQueue::button_press_event(GdkEventButton* event)
{
}

bool ControlerQueue::button_release_event(GdkEventButton* event)
{
}

bool ControlerQueue::motion_notify_event(GdkEventMotion* event)
{
}
```

```
/*
 * includes.h
 *
 *  Created on: Feb 16, 2010
 *      Author: Matthew
 */

#ifndef INCLUDES_H_
#define INCLUDES_H_

#include <stdio.h>
#include <stdlib.h>
#include <iomanip>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <vector>
#include <math.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <gtk-2.0/gtk/gtk.h>
#include <gtkmm-2.4/gtkmm.h>
#include <glibmm-2.4/glibmm.h>
#include <gdkmm-2.4/gdkmm.h>
#include <gdk/gdk.h>
#include <gtk/gtk.h>
#include <gdk/gdkkeysyms.h>
#include <cairo/cairo.h>

using namespace std;
using namespace Gtk;
using namespace Glib;

#endif /* INCLUDES_H_ */
```

```
#ifndef LINE_H_INCLUDED
#define LINE_H_INCLUDED

#include "Shape.h"

class Line : public Shape
{
public:
    Line(int type, double radius, double theta, double x_0, double y_0,
double x, double y, double sides, double red, double green, double
blue);
    ~Line();
    double get_x_0();
    double get_y_0();
    double get_radius();
    double get_sides();
    int get_type();
    double get_theta();
    double get_x();
    double get_y();
      double get_red();
      double get_green();
      double get_blue();
private:
    double x_0;
    double y_0;
    double x;
    double y;
    double radius;
    double sides;
    int type;
    double theta;
      double red;
      double green;
      double blue;
};


#endif // LINE_H_INCLUDED

#include "Line.h"

Line::Line(int type, double radius, double theta, double x_0, double
y_0, double x, double y, double sides, double red, double green, double
blue)
{
    this->type = type;
    this->radius = radius;
    this->x_0 = x_0;
    this->y_0 = y_0;
    this->sides = sides;
    this->theta = theta;
    this->x = x;
    this->y = y;
      this->red = red;
```

```
      this->green = green;
      this->blue = blue;
}

Line::~Line()
{

}

double Line::get_radius()
{
    return this->radius;
}

double Line::get_x_0()
{
    return this->x_0;
}

double Line::get_y_0()
{
    return this->y_0;
}

double Line::get_sides()
{
    return this->sides;
}

int Line::get_type()
{
    return this->type;
}

double Line::get_theta()
{
    return this->theta;
}

double Line::get_x()
{
    return this->x;
}

double Line::get_y()
{
    return this->y;
}

double Line::get_red()
{
      return this->red;
}

double Line::get_green()
{
      return this->green;
```

```
}

double Line::get_blue()
{
	return this->blue;
}
```

```
#include <cstdio>
#include <iostream>
#include <fstream>

using namespace std;

#ifndef LINKED
#define LINKED
template <typename T> class Node
{
      public:
            Node();
            Node(T);
            ~Node();
            Node<T>*& getNext();
            void setNext(Node<T>*);
            T getData();
      private:
            T data;
            Node<T>* next;
};

template <typename T> Node<T>::Node()
{
}

template <typename T> Node<T>::Node(T info)
{
      data = info;
      next = NULL;
}

template <typename T> Node<T>::~Node()
{
}

template <typename T> Node<T>*& Node<T>::getNext()
{
      return next;
}

template <typename T> void Node<T>::setNext(Node<T>* nextNode)
{
      next = nextNode;
}

template <typename T> T Node<T>::getData()
{
      return data;
}

/*-------------------------------------------------------------------------
----------*/

template <typename T> class LinkedList
{
      public:
```

```
            LinkedList();
            ~LinkedList();
            void InsertTail(T);
            void InsertHead(T);
            void Delete(T);
            void DeleteHead();
            Node<T>* getHead();
            void Display();
            bool Search(T);
            T getItem(int index);
            int getSize();
      private:
            Node<T>* head;
            int size;
};

template <typename T> LinkedList<T>::LinkedList()
{
      head = NULL;
      size = 0;
}

template <typename T> LinkedList<T>::~LinkedList()
{
      if(head == NULL)
      {
      }
      else
      {
            Node<T>* slider = head;
            Node<T>* trail = slider;
            while(slider != NULL)
            {
                  trail = slider;
                  slider = slider->getNext();
                  delete trail;
            }
      }
}

template <typename T> void LinkedList<T>::InsertTail(T info) {
      if(head == NULL)
      {
            Node<T>* data = new Node<T>(info);
            head = data;
            size++;
      }
      else
      {
            Node<T>* data = new Node<T>(info);
            Node<T>* slider = head;
            while(slider->getNext() != NULL)
            {
                  slider = slider->getNext();
            }
            slider->setNext(data);
            size++;
```

```
	}
}

template <typename T> void LinkedList<T>::InsertHead(T info)
{
	if(head == NULL)
	{
		Node<T>* data = new Node<T>(info);
		head = data;
		size++;
	}
	else
	{
		Node<T>* data = new Node<T>(info);
		data->setNext(head);
		head = data;
		size++;
	}
}

template <typename T> void LinkedList<T>::Delete(T info)
{
	if(head == NULL)
	{
	}
	else
	{
		Node<T>* slider = head;
		while(slider != NULL && slider->getNext()->getData() !=
info)
		{
			slider = slider->getNext();
		}
		if(slider != NULL) slider->setNext(slider->getNext()-
>getNext());
		size--;
	}
}

template <typename T> void LinkedList<T>::DeleteHead()
{
	if(head == NULL)
	{
	}
	else
	{
		head = head->getNext();
		size--;
	}
}

template <typename T> Node<T>* LinkedList<T>::getHead()
{
	return head;
}

template <typename T> void LinkedList<T>::Display()
```

```
{
	if(head == NULL)
	{
	}
	else
	{
		fstream file_Out("X:\\Matthew\\CSUN\\Final
Project\\EditMemberList\\Output\\Output1.txt", ios::out);
		Node<T>* slider = head;
		while(slider != NULL)
		{
			file_Out << slider->getData() << endl;
			slider = slider->getNext();
		}
		file_Out.close();
	cout << endl << endl;
	}
}

template <typename T> bool LinkedList<T>::Search(T item)
{
	bool ret = false;
	if(head == NULL)
	{
	}
	else
	{
		Node<T>* slider = head;
		while(slider != NULL && slider->getData() != item)
		{
			slider = slider->getNext();
		}
		if(slider != NULL) ret = true;
	}
	return ret;
}

template <typename T> T LinkedList<T>::getItem(int index)
{
    T item;
    if(index > size - 1 || index < 0)
    {
        item = NULL;
    }
    else
    {
        int count = 0;
        Node<T>* slider = head;
        while(count < index)
        {
            slider = slider->getNext();
            count++;
        }
        item = slider->getData();
    }
    return item;
}
```

```
template <typename T> int LinkedList<T>::getSize()
{
	return size;
}
#endif
```

```
/*
 * Login_Dialog.h
 *
 *  Created on: Mar 2, 2010
 *      Author: Matthew
 */

#ifndef LOGIN_DIALOG_H_
#define LOGIN_DIALOG_H_

#include "includes.h"
class MainWindow;

class Login_Dialog : public Gtk::Dialog
{
public:
	Login_Dialog(int type, MainWindow* hwnd);
	~Login_Dialog();
	void set_status_label(Glib::ustring label);
	Glib::ustring get_uname();
	Glib::ustring get_psswd();
	void set_uname(Glib::ustring uname);
	void set_psswd(Glib::ustring psswd);
	void hash_password(char* psswd);

private:
	MainWindow* hwnd;
	Glib::ustring* label_value;
	char hash_char(char c);

protected:
	HBox* user_layout;
	HBox* pswd_layout;
	Label* user_name;
	Entry* user_entry;
	Label* pswd_user;
	Entry* pswd_entry;
	Button* Ok;
	Button* Cancel;
	Label* status_label;
};

#endif /* LOGIN_DIALOG_H_ */

/*
 * Login_Dialog.cpp
 *
 *  Created on: Mar 2, 2010
 *      Author: Matthew
 */

#include "Login_Dialog.h"
#include "MainWindow.h" // <-- This is a temport fix, I hope it does
not have side effects

Login_Dialog::Login_Dialog(int type, MainWindow* hwnd)
```

```
{
	this->hwnd = hwnd;
	this->set_default_size(100, 150);

	this->user_layout = new HBox();
	this->pswd_layout = new HBox();


	this->user_name = new Label("Username");
	this->user_entry = new Entry();
	this->pswd_user = new Label("Password");
	this->pswd_entry = new Entry();
	this->Ok = add_button("Ok", 1);
	this->Cancel = add_button("Cancel", 0);

    /*
     *  This is purely for testing purposes
     */
	Glib::ustring* text0 = new Glib::ustring("You are connected to:
");
	Glib::ustring* text1 = new Glib::ustring(text0->append(*hwnd-
>get_socket()->get_server_domainname_string()));
	this->label_value = text1;
	this->status_label = new Label(*(this->label_value), ALIGN_LEFT,
ALIGN_LEFT, false);
	/*
	 *    End of testing
	 */


	this->Ok->set_size_request(74, -1);
	this->Cancel->set_size_request(74, -1);
	this->pswd_entry->property_visibility() = false;

	this->user_layout->pack_start(*(this->user_name), true, true);
	this->user_layout->pack_end(*(this->user_entry), true, true);
	this->pswd_layout->pack_start(*(this->pswd_user), true, true);
	this->pswd_layout->pack_end(*(this->pswd_entry), true, true);
	this->get_vbox()->pack_start(*(this->user_layout));
	this->get_vbox()->pack_end(*(this->status_label), true, true);
	this->get_vbox()->pack_end(*(this->pswd_layout));

	this->show_all();
}

Login_Dialog::~Login_Dialog()
{
}

void Login_Dialog::set_status_label(Glib::ustring label)
{
	this->status_label->set_label(label);
}

Glib::ustring Login_Dialog::get_uname()
{
	return this->user_entry->get_text();
```

```
}

Glib::ustring Login_Dialog::get_psswd()
{
	return this->pswd_entry->get_text();
}

void Login_Dialog::set_uname(Glib::ustring uname)
{
	this->user_entry->set_text(uname);
}

void Login_Dialog::set_psswd(Glib::ustring psswd)
{
	this->pswd_entry->set_text(psswd);
}

void Login_Dialog::hash_password(char* psswd)
{
	printf("Hashing Password\n");
	int x = 0;
	char s;
	while((s = *(psswd + x)) != '\0')
	{
		*(psswd + x) = hash_char(*(psswd + x++));
	}
}

char Login_Dialog::hash_char(char c)
{
	c = (char)(c + 12);
	c = (char)(c % 12);
	return c;
}
```

```
//============================================================================
=======
// Name        : main.cpp
// Author      : Matthew Hoggan
// Version     :
// Copyright   : This is free software
// Description : Tutor_Application Fall 2009
//============================================================================
=======

#include "MainWindow.h"

int main(int argc, char* argv[])
{
      if(!Glib::thread_supported())
      {
            Glib::thread_init();
            cout << "Threads initialized " << endl;
      }
      Main kit(argc, argv);
      MainWindow* window = new MainWindow(WINDOW_TOPLEVEL);
      kit.run(*(window));
      return 0;
}
```

```
/*
 * MainWindow.h
 *
 *  Created on: Feb 7, 2010
 *      Author: Matthew
 */

#ifndef MAINWINDOW_H_
#define MAINWINDOW_H_

#include "includes.h"

#include "ModelDrawing.h"
#include "ViewDrawing.h"
#include "ControlerDrawing.h"
#include "ModelChat.h"
#include "ViewChat.h"
#include "ControlerChat.h"
#include "ModelQueue.h"
#include "ViewQueue.h"
#include "ControlerQueue.h"
#include "Login_Dialog.h"
#include "TCP_IP_Socket.h"
#include "Tutor.h"
#include "Student.h"

#define VALID true

class MainWindow : public Window
{
public:
      MainWindow(int type);
      ~MainWindow();

      void on_menu_file_new_generic();
      void on_menu_file_quit();

      ModelDrawing* get_mdl_Draw();
      ViewDrawing* get_view_Draw();
      ControlerDrawing* get_cntrl_Draw();

      ModelChat* get_mdl_Chat();
      ViewChat* get_view_Chat();
      ControlerChat* get_cntrl_Chat();

      ModelQueue* get_mdl_Que();
      ViewQueue* get_view_Que();
      ControlerQueue* get_cntrl_Que();

      Glib::Thread* get_chat_thread();
      Glib::Thread* get_draw_thread();

      void create_chat_thread();
    void create_draw_thread();
```

```
    void join_chat_thread();
    void join_draw_thread();

      Label* get_status_label();
      void set_status_label(Glib::ustring label);

      TCP_IP_Socket* get_socket();
      int Login_Screen();

      void set_Student(Student* set_std);
      Student* get_Student();

      void set_Tutor(Tutor* set_tutor);
      Tutor* get_Tutor();

      void create_Student_Queue();
      void create_Login_Screen();

      void on_realize_create();

private:
      TCP_IP_Socket* socket;

      Widget* menu;
      RefPtr<Gtk::ActionGroup> m_refActionGroup;
      RefPtr<Gtk::UIManager> m_refUIManager;

      ModelDrawing* mdl_Draw;
      ViewDrawing* view_Draw;
      ControlerDrawing* cntrl_Draw;

      ModelChat* mdl_Chat;
      ViewChat* view_Chat;
      ControlerChat* cntrl_Chat;

      ModelQueue* mdl_Que;
      ViewQueue* view_Que;
      ControlerQueue* cntrl_Que;

      Frame* label_frame;
      Label* status_label;

      Login_Dialog* login;
      Tutor* tutor;
      Student* student;

    Glib::Thread* chat_thread;
    Glib::Thread* draw_thread;

protected:
      //Containers
      HBox* main_HBox;
      VBox* base_VBox;
};

#endif /* MAINWINDOW_H_ */
```

```
/*
 * MainWindow.cpp
 *
 *  Created on: Feb 7, 2010
 *      Author: Matthew Hoggan
 */

#include "MainWindow.h"

MainWindow::MainWindow(int type)
{
    this->tutor = NULL;
    this->student = NULL;
    this->draw_thread = NULL;
    this->chat_thread = NULL;

      char* start_port = "3490";
      this->socket = new TCP_IP_Socket(this, start_port);

      //Login Section
    this->create_Login_Screen();
    cout << "CONNECTED" << endl;

    this->signal_realize().connect(sigc::mem_fun(*this,
&MainWindow::on_realize_create));
    this->set_default_size(1024, 600);
      this->set_border_width(1);
      this->set_title("Tutor App");

      this->base_VBox = new VBox();
      this->main_HBox = new HBox();
      this->label_frame = new Frame();

      m_refActionGroup = Gtk::ActionGroup::create();
      m_refUIManager = Gtk::UIManager::create();
      m_refActionGroup->add(Gtk::Action::create("FileMenu", "File"));
      this->add_accel_group(m_refUIManager->get_accel_group());
      Glib::ustring ui_info =
      "<ui>"
      "<menubar name='MenuBar'>"
      "     <menu action='FileMenu'>"
      "     </menu>"
      "</menubar>"
      "</ui>";
      m_refUIManager->insert_action_group(m_refActionGroup);
      m_refUIManager->add_ui_from_string(ui_info);
      this->menu = m_refUIManager->get_widget("/MenuBar");

      this->mdl_Draw = new ModelDrawing(this);
      this->view_Draw = new ViewDrawing(this);
      this->cntrl_Draw = new ControlerDrawing(this, (this->mdl_Draw),
(this->view_Draw));

      this->mdl_Chat = new ModelChat(this);
      this->view_Chat = new ViewChat(this);
```

```
      this->cntrl_Chat = new ControlerChat(this, (this->mdl_Chat),
(this->view_Chat));

      this->status_label = manage(new Label("Welcome to The Tutor App",
ALIGN_LEFT, ALIGN_LEFT, false));

      //Put it all together
      this->main_HBox->pack_start(*(this->view_Draw->get_left_VBox()));
      this->label_frame->add(*(this->status_label));
      this->base_VBox->pack_end(*(this->label_frame));
      this->main_HBox->pack_end(*(this->view_Chat->get_right_VBox()));
      this->base_VBox->pack_start(*(this->menu), Gtk::PACK_SHRINK);
      this->base_VBox->pack_end(*(this->main_HBox), true, true);

      this->label_frame->set_size_request(-1, 20);

      this->add(*(this->base_VBox));
      this->set_resizable(false);
      this->show_all();

    this->cntrl_Chat->delete_Socket();
    this->cntrl_Chat->establish_Socket();
    this->cntrl_Chat->Set_Sensitive_Queue_Button(false);
      this->cntrl_Chat->Set_Sensitive_log_On(false);

      this->create_Student_Queue();
}

MainWindow::~MainWindow()
{
}

ControlerChat* MainWindow::get_cntrl_Chat()
{
    return this->cntrl_Chat;
}

ModelChat* MainWindow::get_mdl_Chat()
{
    return this->mdl_Chat;
}

ViewChat* MainWindow::get_view_Chat()
{
    return this->view_Chat;
}

ControlerDrawing* MainWindow::get_cntrl_Draw()
{
    return this->cntrl_Draw;
}

ControlerQueue* MainWindow::get_cntrl_Que()
{
    return this->cntrl_Que;
}
```

```
ModelQueue* MainWindow::get_mdl_Que()
{
    return this->mdl_Que;
}

ViewDrawing* MainWindow::get_view_Draw()
{
      return this->view_Draw;
}

ModelDrawing* MainWindow::get_mdl_Draw()
{
      return this->mdl_Draw;
}

ViewQueue* MainWindow::get_view_Que()
{
    return this->view_Que;
}

Label* MainWindow::get_status_label()
{
      return this->status_label;
}

void MainWindow::set_status_label(Glib::ustring label)
{
      this->status_label->set_label(label);
}

TCP_IP_Socket* MainWindow::get_socket()
{
    return this->socket;
}

void MainWindow::set_Student(Student* set_std)
{
    this->student = set_std;
}

Student* MainWindow::get_Student()
{
    return this->student;
}

void MainWindow::set_Tutor(Tutor* set_tutor)
{
    this->tutor = set_tutor;
}

Tutor* MainWindow::get_Tutor()
{
    return this->tutor;
}

Glib::Thread* MainWindow::get_chat_thread()
{
```

```
    return this->chat_thread;
}

Glib::Thread* MainWindow::get_draw_thread()
{
    return this->draw_thread;
}


void MainWindow::on_menu_file_quit()
{
      hide(); //Closes the main window to stop the Gtk::Main::run().
}

void MainWindow::on_menu_file_new_generic()
{
   std::cout << "A File|New menu item was selected." << std::endl;
}

void MainWindow::create_Login_Screen()
{
    int authenticate;
      this->login = new Login_Dialog(0, this);
      while((authenticate = (this->Login_Screen())) != VALID)
      {
                  printf("Invalid Login: \n");
      }
      this->login->hide();
      delete this->login;
}

int MainWindow::Login_Screen()
{
      int valid_credentials = 0;
      int status;
      status = this->login->run();
      if(status == 0)
      {
            exit(1);
      }
      else
      {
            Glib::ustring* uname = new Glib::ustring(this->login-
>get_uname());
            Glib::ustring* passwd = new Glib::ustring(this->login-
>get_psswd());
            if(this->tutor != NULL)
            {
                delete this->tutor;
            }
            this->tutor = new Tutor(uname, passwd);
            this->login->set_uname("");
            this->login->set_psswd("");
      }
      char* header = (char*)malloc(4);
      sprintf(header, "%d", 1, 4);
    this->socket->send_packet(header);
```

```
      const char* uname_packet = this->tutor->get_uname();
      const char* psswd_packet = this->tutor->get_psswd();
      char* packet = (char*)malloc(512);
      sprintf(packet, "%s %s", uname_packet, psswd_packet, 512);
      this->socket->send_packet(packet);
      free(packet);
      char server_status;
      cout << endl;
      server_status = (this->socket->receive_packet_login());
      valid_credentials = server_status - 48;
      return valid_credentials;
}

void MainWindow::create_Student_Queue()
{
    //By Default Create and Open Up Student Queue
    this->mdl_Que = NULL;
    this->view_Que = NULL;
    this->cntrl_Que = NULL;
      this->mdl_Que = new ModelQueue(this);
      this->view_Que = new ViewQueue(this);
      this->cntrl_Que = new ControlerQueue(this, (this->mdl_Que),
(this->view_Que));
      this->cntrl_Que->get_Data_From_Server();
      this->view_Que->get_window()->show_all();
      this->cntrl_Chat->Set_Sensitive_ALL(false);
    this->cntrl_Chat->Set_Editable_ALL(false);
    this->cntrl_Draw->Set_Sensitive_ALL(false);
}

void MainWindow::on_realize_create()
{
}

void MainWindow::create_chat_thread()
{
    this->chat_thread = Glib::Thread::create(sigc::bind<Glib::ustring,
TextView*>(sigc::mem_fun(*(this->mdl_Chat->get_chat_socket()),
&TCP_IP_Socket::recieve_Text_Data), this->student->get_first_Name(),
this->view_Chat->get_Output()), false);
    //this->chat_thread =
Glib::Thread::create(sigc::bind<Glib::ustring>(sigc::mem_fun(*(this-
>mdl_Chat->get_chat_socket()), &TCP_IP_Socket::recieve_Text_Data),
this->student->get_first_Name()), false);
}

void MainWindow::create_draw_thread()
{
    this->draw_thread =
Glib::Thread::create(sigc::bind<int>(sigc::mem_fun(*(this->socket),
&TCP_IP_Socket::recieve_Draw_Data), 10), false);
}

void MainWindow::join_chat_thread()
{
    cout << "Joining Chat Thread" << endl;
    this->chat_thread->join();
```

```
}

void MainWindow::join_draw_thread()
{
    cout << "Joining Draw Thread" << endl;
    this->draw_thread->join();
}
```

```
/*
 * ModelChat.h
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#ifndef MODELCHAT_H_
#define MODELCHAT_H_

#include "includes.h"

class TCP_IP_Socket;
class MainWindow;

class ModelChat
{
public:
      ModelChat();
      ModelChat(MainWindow* hwnd);
      ~ModelChat();
    TCP_IP_Socket* get_chat_socket();
    void establish_Socket();
    void delete_Socket();
private:
      MainWindow* hwnd;
      TCP_IP_Socket* chat_socket;
};

#endif /* MODELCHAT_H_ */

/*
 * ModelChat.cpp
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#include "ModelChat.h"
#include "TCP_IP_Socket.h"

ModelChat::ModelChat()
{

}

ModelChat::ModelChat(MainWindow* hwnd)
{
      this->hwnd = hwnd;
    this->chat_socket = NULL;
}

ModelChat::~ModelChat()
{

}
```

```
TCP_IP_Socket* ModelChat::get_chat_socket()
{
    return this->chat_socket;
}

void ModelChat::establish_Socket()
{
    //THIS SHOULD BE 3491 Waiting for andrew to finish this part
      char* chat_port = "3491";
    this->chat_socket = new TCP_IP_Socket(this->hwnd, chat_port);
    cout << "CONNECTED" << endl;
}

void ModelChat::delete_Socket()
{
    if(this->chat_socket != NULL)
    {
        cout << "Deleting chat socket " << endl;
        delete this->chat_socket;
    }
}
```

```
#ifndef MODELCOLUMNS_H_
#define MODELCOLUMNS_H_

class ModelColumns : public TreeModel::ColumnRecord
{
public:
	ModelColumns()
	{
		this->add(m_col_id);
		this->add(m_col_name);
		this->add(m_col_subject);
		this->add(m_col_time);
	}
	~ModelColumns()
	{
	}
	Gtk::TreeModelColumn<unsigned int> m_col_id;
	Gtk::TreeModelColumn<Glib::ustring> m_col_name;
	Gtk::TreeModelColumn<Glib::ustring> m_col_subject;
	Gtk::TreeModelColumn<int> m_col_time;
};

#endif // MODELCOLUMNS_H_
```

```
/*
 * ModelDrawing.h
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#ifndef MODELDRAWING_H_
#define MODELDRAWING_H_

#include "includes.h"
#include "Shape.h"
#include "LinkedList.h"
typedef int DRAWINGMODE;

class MainWindow;

class ModelDrawing
{

private:
      MainWindow* hwnd;

      enum DrawingMode
      {
        PENCIL, ERASER, RECTANGLE, POLYGON, ELLIPSE, LINE, ARROW
      } draw_mode;

      int left_Click_x;
      int left_Click_y;

      double trans;

      double red;
      double blue;
    double green;

    LinkedList<Shape*>*** grid;
    int grid_Rows;
    int grid_Col;
public:
      ModelDrawing();
      ModelDrawing(MainWindow* hwnd);
      ~ModelDrawing();

      int get_draw_mode();
      void set_draw_mode(DRAWINGMODE draw_mode);

      int get_left_Click_x();
      int get_left_Click_y();
    void set_left_Click_x_y(int x, int y);
    double get_trans();
    void set_trans(double trans);

    double get_red();
```

```
    double get_green();
    double get_blue();

    void set_red(double red);
    void set_green(double green);
    void set_blue(double blue);
    void set_rgb(double red, double green, double blue);
    void insert_shape(Shape* shape);

    enum Shape_Type
    {
        T_LINE, T_SQUARE, T_POLYGON, T_CIRCLE
    } shape;

    void initilize_Grid(int width, int height);
    void delete_Grid(int width, int height);
    Shape* get_Shape(int x, int y, int z);
    void print_grid();
};

#endif /* MODELDRAWING_H_ */

/*
 * ModelDrawing.cpp
 *
 *      Author: Matthew
 */

#include "ModelDrawing.h"

ModelDrawing::ModelDrawing()
{
    this->draw_mode = ARROW;
    this->set_trans(0.0);
    this->grid = NULL;
    this->grid_Rows = 0;
    this->grid_Col = 0;
}

ModelDrawing::ModelDrawing(MainWindow* hwnd)
{
      this->hwnd = hwnd;
}

ModelDrawing::~ModelDrawing()
{

}

int ModelDrawing::get_draw_mode()
{
    return draw_mode;
}

int ModelDrawing::get_left_Click_x()
{
```

```
    return this->left_Click_x;
}

int ModelDrawing::get_left_Click_y()
{
    return this->left_Click_y;
}

void ModelDrawing::set_left_Click_x_y(int x, int y)
{
    this->left_Click_x = x;
    this->left_Click_y = y;
}

double ModelDrawing::get_trans()
{
    return this->trans;
}

void ModelDrawing::set_trans(double trans)
{
    this->trans = trans;
}

double ModelDrawing::get_red()
{
    return this->red;
}

double ModelDrawing::get_green()
{
    return this->green;
}

double ModelDrawing::get_blue()
{
    return this->blue;
}

void ModelDrawing::set_red(double red)
{
    this->red = red;
}

void ModelDrawing::set_green(double green)
{
    this->green = green;
}

void ModelDrawing::set_blue(double blue)
{
    this->blue = blue;
}

void ModelDrawing::set_rgb(double red, double green, double blue)
{
    this->red = red;
```

```
    this->green = green;
    this->blue = blue;
}

void ModelDrawing::set_draw_mode(DRAWINGMODE draw_mode)
{
    switch(draw_mode)
    {
        case 0:
        {
            this->draw_mode = PENCIL;
            break;
        }
        case 1:
        {
            this->draw_mode = ERASER;
            break;
        }
        case 2:
        {
            this->draw_mode = RECTANGLE;
            break;
        }
        case 3:
        {
            this->draw_mode = POLYGON;
            break;
        }
        case 4:
        {
            this->draw_mode = ELLIPSE;
            break;
        }
        case 5:
        {
            this->draw_mode = LINE;
            break;
        }
        case 6:
        {
            this->draw_mode = ARROW;
            break;
        }
        default:
        {
            cout << "An error has occured" << endl;
            exit(0);
        }
    }
}

void ModelDrawing::initilize_Grid(int width, int height)
{
    this->grid_Col = width;
    this->grid_Rows = height;
    grid = new LinkedList<Shape*>**[width];
    for(int x = 0; x < width; x++)
```

```
    {
        grid[x] = new LinkedList<Shape*>*[height];
        for(int y = 0; y < height; y++)
        {
            grid[x][y] = new LinkedList<Shape*>();
        }
    }
}

void ModelDrawing::delete_Grid(int width, int height)
{
    for(int x = 0; x < width; x++)
    {
        for(int y = 0; y < height; y++)
        {
            delete(grid[x][y]);
        }
        delete(grid[x]);
    }
    delete(grid);
}

Shape* ModelDrawing::get_Shape(int x, int y, int z)
{
    Shape* shape = grid[x][y]->getItem(z);
    if(shape != NULL)
    {
    }
    return shape;
}

void ModelDrawing::insert_shape(Shape* shape)
{
    int x = shape->get_x_0();
    int y = shape->get_y_0();
    if(x >= 0 && x < this->grid_Col && y >= 0 && y < this->grid_Rows)
    {
        this->grid[x][y]->InsertHead(shape);
        this->shape = (Shape_Type)shape->get_type();
    }
}

void ModelDrawing::print_grid()
{
    int x;
    int y;
    int z = 0;
    for(x = 0; x < this->grid_Col; x++)
    {
        for(y = 0; y < this->grid_Rows; y++)
        {
            while(grid[x][y]->getItem(z) != NULL)
            {
                z++;
            }
            z = 0;
        }
```

```
        }
    }
```

```
/*
 * ModelQueue.h
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#ifndef MODELQUEUE_H_
#define MODELQUEUE_H_

#include "includes.h"
#include "Student.h"
#include "LinkedList.h"

class MainWindow;

class ModelQueue
{
public:
	ModelQueue();
	ModelQueue(MainWindow* hwnd);
	~ModelQueue();
	Student* get_Student(Glib::ustring full_Name);
	LinkedList<Student*>* get_Student_List();
	void add_Student(Student* stdnt);
	Student* get_Student(int index);
private:
	MainWindow* hwnd;
	int num_O_Students;
	LinkedList<Student*>* student_Queue;
};

#endif /* MODELQUEUE_H_ */

/*
 * ModelQueue.cpp
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#include "ModelQueue.h"

ModelQueue::ModelQueue()
{
}

ModelQueue::ModelQueue(MainWindow* hwnd)
{
	this->hwnd = hwnd;
	student_Queue = new LinkedList<Student*>();
}

ModelQueue::~ModelQueue()
{
    if(student_Queue == NULL)
    {
```

```
    }
    else
    {
        delete student_Queue;
    }
}

Student* ModelQueue::get_Student(Glib::ustring full_Name)
{
    Student* retn;
    return retn;
}

void ModelQueue::add_Student(Student* stdnt)
{
    Student* add = new Student(stdnt->get_first_Name(), stdnt-
>get_last_Name(), stdnt->get_subject_Name(), stdnt-
>get_time_in_queue());
    student_Queue->InsertTail(add);
    num_O_Students++;
}

Student* ModelQueue::get_Student(int index)
{
    Student* present_Student = student_Queue->getItem(index);
    return present_Student;
}

LinkedList<Student*>* ModelQueue::get_Student_List()
{
    return this->student_Queue;
}
```

```
#ifndef POLYGON_H_INCLUDED
#define POLYGON_H_INCLUDED

#include "Shape.h"

class Polygon : public Shape
{
public:
    Polygon(int type, double radius, double theta, double x_0, double
y_0, double x, double y, double sides, double red, double green, double
blue);
    ~Polygon();
    double get_x_0();
    double get_y_0();
    double get_radius();
    double get_sides();
    int get_type();
    double get_theta();
    double get_x();
    double get_y();
      double get_red();
      double get_green();
      double get_blue();
private:
    double x_0;
    double y_0;
    double x;
    double y;
    double radius;
    double sides;
    int type;
    double theta;
      double red;
      double green;
      double blue;
};


#endif // POLYGON_H_INCLUDED

#include "Polygon.h"

Polygon::Polygon(int type, double radius, double theta, double x_0,
double y_0, double x, double y, double sides, double red, double green,
double blue)
{
    this->type = type;
    this->radius = radius;
    this->x_0 = x_0;
    this->y_0 = y_0;
    this->sides = sides;
    this->theta = theta;
    this->x = x;
    this->y = y;
      this->red = red;
      this->green = green;
      this->blue = blue;
```

```
}

Polygon::~Polygon()
{

}

double Polygon::get_radius()
{
    return this->radius;
}

double Polygon::get_x_0()
{
    return this->x_0;
}

double Polygon::get_y_0()
{
    return this->y_0;
}

double Polygon::get_sides()
{
    return this->sides;
}

int Polygon::get_type()
{
    return this->type;
}

double Polygon::get_theta()
{
    return this->theta;
}

double Polygon::get_x()
{
    return this->x;
}

double Polygon::get_y()
{
    return this->y;
}

double Polygon::get_red()
{
      return this->red;
}

double Polygon::get_green()
{
      return this->green;
}
```

```
double Polygon::get_blue()
{
	return this->blue;
}
```

```
#ifndef SHAPE_H_INCLUDED
#define SHAPE_H_INCLUDED

class Shape
{
public:
    Shape();
    virtual ~Shape() = 0;
    virtual double get_radius() = 0;
    virtual double get_x_0() = 0;
    virtual double get_y_0() = 0;
    virtual double get_sides() = 0;
    virtual int get_type() = 0;
    virtual double get_theta() = 0;
    virtual double get_x() = 0;
    virtual double get_y() = 0;
      virtual double get_red() = 0;
      virtual double get_green() = 0;
      virtual double get_blue() = 0;
private:
    double x_0;
    double y_0;
    double x;
    double y;
    double radius;
    double sides;
    int type;
    double theta;
      double red;
      double green;
      double blue;
};

#endif // SHAPE_H_INCLUDED

#include "Shape.h"

Shape::Shape()
{
}

Shape::~Shape()
{
}
```

```
#ifndef SQUARE_H_INCLUDED
#define SQUARE_H_INCLUDED

#include "Shape.h"

class Square : public Shape
{
public:
    Square(int type, double radius, double theta, double x_0, double
y_0, double x, double y, double sides, double red, double green, double
blue);
    ~Square();
    double get_x_0();
    double get_y_0();
    double get_radius();
    double get_sides();
    int get_type();
    double get_theta();
    double get_x();
    double get_y();
      double get_red();
      double get_green();
      double get_blue();
private:
    double x_0;
    double y_0;
    double x;
    double y;
    double radius;
    double sides;
    int type;
    double theta;
      double red;
      double green;
      double blue;
};


#endif // SQUARE_H_INCLUDED

#include "Square.h"

Square::Square(int type, double radius, double theta, double x_0,
double y_0, double x, double y, double sides, double red, double green,
double blue)
{
    this->type = type;
    this->radius = radius;
    this->x_0 = x_0;
    this->y_0 = y_0;
    this->sides = sides;
    this->theta = theta;
    this->x = x;
    this->y = y;
      this->red = red;
      this->green = green;
      this->blue = blue;
```

```
}

Square::~Square()
{

}

double Square::get_radius()
{
    return this->radius;
}

double Square::get_x_0()
{
    return this->x_0;
}

double Square::get_y_0()
{
    return this->y_0;
}

double Square::get_sides()
{
    return this->sides;
}

int Square::get_type()
{
    return this->type;
}

double Square::get_theta()
{
    return this->theta;
}

double Square::get_x()
{
    return this->x;
}

double Square::get_y()
{
    return this->y;
}

double Square::get_red()
{
      return this->red;
}

double Square::get_green()
{
      return this->green;
}
```

```
double Square::get_blue()
{
	return this->blue;
}
```

```
#ifndef STUDENT_H_
#define STUDENT_H_

#include "includes.h"

class Student
{
public:
    Student(Glib::ustring first_Name, Glib::ustring last_Name,
Glib::ustring subject, Glib::ustring time_in_queue);
    ~Student();
    Glib::ustring get_first_Name();
    Glib::ustring get_last_Name();
    Glib::ustring get_subject_Name();
    Glib::ustring get_time_in_queue();
private:
    Glib::ustring first_Name;
    Glib::ustring last_Name;
    Glib::ustring subject;
    Glib::ustring time_in_queue;
};

#endif //STUDENT_H_

#include "Student.h"

Student::Student(Glib::ustring first_Name, Glib::ustring last_Name,
Glib::ustring subject, Glib::ustring time_in_queue)
{
    this->first_Name = first_Name;
    this->last_Name = last_Name;
    this->subject = subject;
    this->time_in_queue = time_in_queue;
}

Student::~Student()
{

}

Glib::ustring Student::get_first_Name()
{
    return this->first_Name;
}

Glib::ustring Student::get_last_Name()
{
    return this->last_Name;
}

Glib::ustring Student::get_subject_Name()
{
    return this->subject;
}

Glib::ustring Student::get_time_in_queue()
{
```

```
    return this->time_in_queue;
}
```

```
/*
 * TCP_IP_Socket.h
 *
 *  Created on: March 5, 2010
 *      Author: Matthew
 */

#ifndef TCP_IP_SOCKET_H_
#define TCP_IP_SOCKET_H_

#include "includes.h"

class Shape;
class MainWindow;
class Student;

//#define MYPORT "3490"                    // the port server sends data
to
//#define IPADDRESS "www.geoginfo.com"    // ip address of server
//#define IPADDRESS "www.sandbox.csun.edu"      // ip address of server
#define IPADDRESS "pisco.grid.csun.edu"
//#define IPADDRESS "127.0.0.1"
//#define IPADDRESS "192.168.1.1"
#define NUMOFCONNECTIONS 1          // how many pending connections
queue will hold

class TCP_IP_Socket
{
public:
      TCP_IP_Socket(MainWindow* hwnd, char* port);
      ~TCP_IP_Socket();
      void set_server_ustring(const char* server_domainname);
      void set_serverip_ustring(const char* ip);

      Glib::ustring* get_server_domainname_string();
      Glib::ustring* get_server_ip_string();

      int send_packet(char* packet);
      char receive_packet_login();
      const char* receive_packet_student_queue();

      int send_shape_packet(Shape* shape);
      int send_text_packet(const char* text_packet);

      void recieve_Text_Data(Glib::ustring student, TextView* output);
      void recieve_Draw_Data(int x);

      void set_session(bool SESSION);
private:
      MainWindow* hwnd;
      const char* server_name;
      int socketfd;
      int connectfd;
      Glib::ustring* server_domainname;
      Glib::ustring* server_ip;
      struct addrinfo specs;
      struct addrinfo* results;
```

```
	void establish_server_domainname_string(const char* domain_name);
	void establish_server_ip_string();

	bool SESSION;
};

#endif /* TCP_IP_SOCKET_H_ */

/*
 *  TCP_IP_Socket.cpp
 *
 *  Created on: March 5, 2010
 *      Author: Matthew Hoggan
 */

#include "TCP_IP_Socket.h"
#include "Shape.h"
#include "MainWindow.h"
#include "Student.h"
#include <string.h>

TCP_IP_Socket::TCP_IP_Socket(MainWindow* hwnd, char* port)
{
	this->hwnd = hwnd;
	this->establish_server_domainname_string(IPADDRESS);

	memset(&(this->specs), 0, sizeof (this->specs));
    this->specs.ai_family = AF_UNSPEC;  // use IPv4 or IPv6, whichever
	this->specs.ai_socktype = SOCK_STREAM;
    this->specs.ai_flags = AI_PASSIVE;     // fill in my IP for me

	int status0 = 0;
    if((status0 = getaddrinfo(IPADDRESS, port, &(this->specs), &(this-
>results))) == -1)
	{
		fprintf(stderr, "Failed to getinfo: %s\n",
gai_strerror(status0));
		exit(0);
	}

	this->establish_server_ip_string();

    if((this->socketfd = socket((this->results)->ai_family, (this-
>results)->ai_socktype, (this->results)->ai_protocol)) == -1)
    {
		fprintf(stderr, "Failed to get socket: %s\n",
gai_strerror(this->socketfd));
		exit(0);
	}

	if((this->connectfd = connect((this->socketfd), (this->results)-
>ai_addr, (this->results)->ai_addrlen)) == -1)
	{
		fprintf(stderr, "Failed to connect: %s\n",
gai_strerror(this->connectfd));
		exit(0);
```

```
      }
}


TCP_IP_Socket::~TCP_IP_Socket()
{
    int eof = EOF;
    int* p_eof = &eof;
    send((this->socketfd), p_eof, 1, 0);
    close(this->socketfd);
      freeaddrinfo(this->results);
}

void TCP_IP_Socket::set_server_ustring(const char* server_name)
{
    this->server_domainname = new ustring(server_name);
}

void TCP_IP_Socket::set_serverip_ustring(const char* ip)
{
    this->server_ip = new ustring(ip);
}

Glib::ustring* TCP_IP_Socket::get_server_domainname_string()
{
    return this->server_domainname;
}

Glib::ustring* TCP_IP_Socket::get_server_ip_string()
{
    return this->server_ip;
}

void TCP_IP_Socket::establish_server_domainname_string(const char*
domain_name)
{
      this->server_name = domain_name;
      this->set_server_ustring(this->server_name);
    printf("%s", this->server_name);
}

void TCP_IP_Socket::establish_server_ip_string()
{
      char ipstr[INET6_ADDRSTRLEN];
      void* addr;
    if ((this->results)->ai_family == AF_INET)
      { // IPv4
            struct sockaddr_in* ipv4 = (struct sockaddr_in*)(this-
>results)->ai_addr;
            addr = &(ipv4->sin_addr);
      }
      else
      { // IPv6
            struct sockaddr_in6* ipv6 = (struct sockaddr_in6 *)(this-
>results)->ai_addr;
            addr = &(ipv6->sin6_addr);
      }
```

```
      inet_ntop((this->results)->ai_family, addr, ipstr, sizeof ipstr);
      this->set_serverip_ustring(ipstr);
      printf(" = %s\n", ipstr);
}

char TCP_IP_Socket::receive_packet_login()
{
      void* pckt;
      int pckt_size = 0;
      pckt = (char*)malloc(4);

    if((pckt_size = recv((this->socketfd), pckt, 4, 0)) == 0)
      {
            fprintf(stderr, "Failed to connect: %s\n",
gai_strerror(pckt_size));
      }
      return *((char*)(pckt));
      //return '1';
}

int TCP_IP_Socket::send_packet(char* packet)
{
      int status;
      size_t size = strlen(packet);
      if((status = send((this->socketfd), packet, size, 0)) == 0)
      {
            fprintf(stderr, "Failed to send: %s\n", gai_strerror);
      }
      return 0;
}

const char* TCP_IP_Socket::receive_packet_student_queue()
{
    void* pckt = NULL;
    void* pckt2 = NULL;
      if(pckt != NULL)
      {
          memset(&(pckt), 0, sizeof (pckt));
        free(pckt);
      }
      if(pckt2 != NULL)
      {
          memset(&(pckt), 0, sizeof (pckt2));
        free(pckt2);
      }
    string test("Matthew Hoggan Physicis 2:7:30 Andrew Hamilton Math
24:0:0 Alex David Selebesy 0:0:1 Vahe Margoussian Automata 2:10:23
Daniel Serry Math 2:10:23");
      const char* pckt_test = test.c_str();
      return pckt_test;
}

int TCP_IP_Socket::send_shape_packet(Shape* shape)
{
    int rtn = 0;
    char packet[2048];
```

```
    sprintf(packet, "%d %f %f %f %f %f %f %f %f %f %f\n", shape-
>get_type(), shape->get_radius(), shape->get_x(), shape->get_y(),
shape->get_x_0(), shape->get_y_0(), shape->get_theta(), shape-
>get_red(), shape->get_green(), shape->get_blue(), shape->get_sides());
    size_t size = strlen(packet);
      if((rtn = send((this->socketfd), packet, size, 0)) == 0)
      {
            fprintf(stderr, "Failed to send: %s\n", gai_strerror);
      }
    return rtn;
}

int TCP_IP_Socket::send_text_packet(const char* text_packet)
{
    int rtn = 0;
    char packet[2048];
    sprintf(packet, "%s", text_packet);
    strcat(packet, "\n");
    size_t size = strlen(packet);
      if((rtn = send((this->socketfd), packet, size, 0)) == 0)
      {
            fprintf(stderr, "Failed to send: %s\n", gai_strerror);
      }
    return rtn;
}

void TCP_IP_Socket::set_session(bool SESSION)
{
    this->SESSION = SESSION;
}

void TCP_IP_Socket::recieve_Text_Data(Glib::ustring student, TextView*
output)
{
    int* pckt;
    while(SESSION)
    {
        char pckt[4096];
        int pckt_size = 0;
        if((pckt_size = recv((this->socketfd), pckt, 4096, 0)) == 0)
        {
            fprintf(stderr, "Failed to connect: %s\n",
gai_strerror(pckt_size));
        }

        cout << "Just recieved " << pckt << endl;

        Glib::RefPtr<TextBuffer> input = output->get_buffer();
        Glib::ustring input_String = input->get_text(false);
        Glib::ustring post(input_String);
        post.append(student);
        post.append(": ");
        post.append(pckt);
        input->set_text(post);
        memset(&(pckt), 0, 4096);
    }
}
```

```
void TCP_IP_Socket::recieve_Draw_Data(int x)
{
    cout << "The value passed to draw thread function was " << x <<
endl;
    int count = 0;
    int* pckt;
    while(SESSION)
    {
    }
}
```

```
#include <iostream>
#include <string>
#include <iomanip>
#include "LinkedList.h"

using namespace std;

#ifndef TOKENIZER
#define TOKENIZER
class Tokenizer
{
public:
	Tokenizer();
	Tokenizer(string tokenize);
	Tokenizer(string tokenize, char tokenStop);
	~Tokenizer();
	int getWord();
	string nextToken();
	bool hasMoreTokens();
	string getToken(int);
private:
	LinkedList<string>* chainOtOkens;
	string toBeTokenized;
	int word;
};
#endif

#include <iostream>
#include <string>
#include <iomanip>
#include "Tokenizer.h"
#include "LinkedList.h"

Tokenizer::Tokenizer() {
}

Tokenizer::Tokenizer(string tokenize)
{
	chainOtOkens = new LinkedList<string>();
	this->toBeTokenized = tokenize;
	word = 0;

	int count1 = 0;
	int num = 1;
	while(count1 < this->toBeTokenized.size())
	{
		string token = "";
		int count2 = count1;
		while(tokenize[count2] != ' ' && count2 < this-
>toBeTokenized.size())
		{
			token = token + tokenize[count2];
			count1++;
			count2++;
		}
		count1++;
		chainOtOkens->InsertTail(token);
```

```
	}
	word = 0;
}

Tokenizer::Tokenizer(string tokenize, char tokenStop)
{
	chainOtOkens = new LinkedList<string>();
	this->toBeTokenized = tokenize;
	word = 0;

	int count1 = 0;
	int num = 1;
	while(count1 < this->toBeTokenized.size())
	{
		string token = "";
		int count2 = count1;
		while(tokenize[count2] != tokenStop && count2 < this-
>toBeTokenized.size())
		{
			token = token + tokenize[count2];
			count1++;
			count2++;
		}
		if(count2 < this->toBeTokenized.size()) token = token +
this->toBeTokenized[count2];
		count1++;
		while(count1 < this->toBeTokenized.size() &&
tokenize[count1] == ' ')
		{
			count1++;
		}
		chainOtOkens->InsertTail(token);
	}
	word = 0;
}

Tokenizer::~Tokenizer()
{
	delete chainOtOkens;
}

int Tokenizer::getWord()
{
	return word;
}

string Tokenizer::nextToken()
{
	string s = "";
	if(chainOtOkens == NULL || chainOtOkens->getHead() == NULL)
	{
	}
	else
	{
		s = chainOtOkens->getHead()->getData();
		chainOtOkens->DeleteHead();
	}
```

```
	return s;
}

bool Tokenizer::hasMoreTokens()
{
	bool ret = false;
	if(chainOtOkens->getHead() == NULL)
	{
	}
	else
	{
		ret = true;
	}
	return ret;
}

string Tokenizer::getToken(int tokenNumber)
{
	string ret = "";
	if(chainOtOkens->getHead() == NULL)
	{
	}
	else if(tokenNumber > 0 && tokenNumber <= chainOtOkens-
>getSize())
	{
		Node<string>* slider = chainOtOkens->getHead();
		int getToken = 1;
		while(getToken != tokenNumber)
		{
			slider = slider->getNext();
			getToken++;
		}
		ret = slider->getData();
	}
	return ret;
}
```

```
/*
 * Tutor.h
 *
 *  Created on: Feb 7, 2010
 *      Author: Matthew
 */

#include "includes.h"

class Tutor
{
public:
	Tutor(Glib::ustring* uname, Glib::ustring* psswd);
	~Tutor();
	const char* get_uname();
	const char* get_psswd();
	void set_uname(char* uname);
	void set_psswd(char* psswd);
private:
	const char* uname;
	const char* psswd;
};

/*
 * Tutor.cpp
 *
 *  Created on: Feb 7, 2010
 *      Author: Matthew Hoggan
 */

#include "Tutor.h"
Tutor::Tutor(Glib::ustring* uname, Glib::ustring* psswd)
{
	locale_from_utf8(*(uname));
	locale_from_utf8(*(psswd));

	this->uname = uname->c_str();
	this->psswd = psswd->c_str();
}

Tutor::~Tutor()
{
}

const char* Tutor::get_uname()
{
	return this->uname;
}

const char* Tutor::get_psswd()
{
	return this->psswd;
}

void Tutor::set_uname(char* uname)
{
	this->uname = uname;
```

```
}

void Tutor::set_psswd(char* pswd)
{
      this->psswd = psswd;
}
```

```
/*
 * ViewChat.h
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#ifndef VIEWCHAT_H_
#define VIEWCHAT_H_

#include "includes.h"

class MainWindow;
class ModelChat;

class ViewChat
{
public:
	ViewChat();
	ViewChat(MainWindow* hwnd);
	~ViewChat();
	void set_Model(ModelChat* mdl_Chat);
	VBox* get_right_VBox();

	Button* get_student_Queue_Button();
	Button* get_log_On_Button();
	Button* get_log_Off_Button();
	ToolButton* get_fontclr_Option();
	ToolButton* get_emo_Option();
	TextView* get_Input();
	TextView* get_Output();

private:
	MainWindow* hwnd;
	ModelChat* mdl_Chat;

protected:
	VBox* right_VBox;
	Frame* output_Frame;
	Frame* rbutton_Frame;
	HBox* rbutton_HBox;
	Frame* input_Frame;
	VBox* text_VBox;
	Toolbar* text_Options;

	ScrolledWindow* scroll_Output;
	TextView* Output;
	Button* student_Queue;
	Button* log_On;
	Button* log_Off;
	Image* font_Color;
	ToolButton* fontclr_Option;
	Image* smily_Face;
	ToolButton* emo_Option;
	ScrolledWindow* scroll_Input;
	TextView* Input;
};
```

```
#endif /* VIEWCHAT_H_ */

/*
 * ViewChat.cpp
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#include "ViewChat.h"

ViewChat::ViewChat()
{

}

ViewChat::ViewChat(MainWindow* hwnd)
{
	//	Create All Containers
	this->right_VBox = manage(new VBox());
	this->output_Frame = manage(new Frame());
	this->rbutton_Frame = manage(new Frame());
	this->rbutton_HBox = manage(new HBox());
	this->input_Frame = manage(new Frame());
	this->text_VBox = manage(new VBox());
	this->text_Options = manage(new Toolbar());

	//	Add Child Containers to Parent Container
	this->right_VBox->pack_end(*(this->rbutton_Frame), false, false);
	this->right_VBox->pack_end(*(this->input_Frame), false, false);
	this->right_VBox->pack_end(*(this->output_Frame), true, true);
	this->text_VBox->pack_start(*(this->text_Options), true, true);

	//	Place Containers into Right Frames
	this->rbutton_Frame->add(*(this->rbutton_HBox));
	this->input_Frame->add(*(this->text_VBox));

	//	Build All Widgets
	this->scroll_Output = manage(new ScrolledWindow());
	this->Output = manage(new TextView());
	this->Output->set_border_window_size(TEXT_WINDOW_BOTTOM, 9);
	this->Output->set_wrap_mode (Gtk::WRAP_WORD_CHAR);
	this->scroll_Output->add(*(this->Output));
	this->scroll_Output->set_policy(POLICY_NEVER, POLICY_ALWAYS);
	this->Output->set_editable(false);
	this->student_Queue = manage(new Button("View Requests"));
	this->log_On = manage(new Button("Log In"));
	this->log_Off = manage(new Button("Log Out"));
	this->scroll_Input = manage(new ScrolledWindow());
	this->scroll_Input->set_policy(POLICY_NEVER, POLICY_ALWAYS);
	this->Input = manage(new TextView());
	this->Input->set_border_window_size(TEXT_WINDOW_TOP, 9);
	this->Input->set_wrap_mode (Gtk::WRAP_WORD_CHAR);
	this->scroll_Input->add(*(this->Input));
	this->font_Color = manage(new
Image("./Tools_Images/fontcolor.png"));
```

```
	this->fontclr_Option = manage(new ToolButton(*(this->font_Color)));
	this->smily_Face = manage(new Image("./Tools_Images/smilyface.png"));
	this->emo_Option = manage(new ToolButton(*(this->smily_Face)));

	//	Set Size Request
	this->right_VBox->set_size_request(300, -1);
	this->output_Frame->set_size_request(300, 400);
	this->input_Frame->set_size_request(300, 200);
	this->text_Options->set_size_request(300, 10);
	this->Input->set_size_request(300, 30);
	this->rbutton_HBox->set_size_request(300,30);
	this->student_Queue->set_size_request(150, -1);
	this->log_On->set_size_request(74, -1);
	this->log_Off->set_size_request(74, -1);

	//	Add Right Widgets to Containers
	this->output_Frame->add(*(this->scroll_Output));
	this->rbutton_HBox->pack_start(*(this->student_Queue), false, false);
	this->rbutton_HBox->pack_start(*(this->log_On), false, false);
	this->rbutton_HBox->pack_start(*(this->log_Off), false, false);
	//this->text_Options->add(*(this->fontclr_Option));
	//this->text_Options->add(*(this->emo_Option));
	this->text_VBox->pack_end(*(this->scroll_Input), true, true);
}

ViewChat::~ViewChat()
{

}

void ViewChat::set_Model(ModelChat* mdl_Chat)
{
	this->mdl_Chat = mdl_Chat;
}

VBox* ViewChat::get_right_VBox()
{
	return (this->right_VBox);
}

Button* ViewChat::get_student_Queue_Button()
{
    return this->student_Queue;
}

Button* ViewChat::get_log_On_Button()
{
    return this->log_On;
}

Button* ViewChat::get_log_Off_Button()
{
    return this->log_Off;
}
```

```
ToolButton* ViewChat::get_fontclr_Option()
{
    return this->fontclr_Option;
}

ToolButton* ViewChat::get_emo_Option()
{
    return this->emo_Option;
}

TextView* ViewChat::get_Input()
{
    return this->Input;
}

TextView* ViewChat::get_Output()
{
    return this->Output;
}
```

```
/*
 * DrawingView.h
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#ifndef DRAWINGVIEW_H_
#define DRAWINGVIEW_H_

#include "includes.h"

class MainWindow;
class ModelDrawing;

class ViewDrawing
{
public:
      ViewDrawing();
      ViewDrawing(MainWindow* hwnd);
      ~ViewDrawing();
      void set_Model(ModelDrawing* mdl_Draw);
      VBox* get_left_VBox();

    DrawingArea* get_sketchBoard();
      ToolButton* get_ellipse_ToolButton();
      ToolButton* get_eraser_ToolButton();
      ToolButton* get_line_ToolButton();
      ToolButton* get_pencil_ToolButton();
      ToolButton* get_polygon_ToolButton();
      ToolButton* get_rectangle_ToolButton();
      ToolButton* get_arrow_ToolButton();
      ColorButton* get_color_Button();
      Button* get_clearSketchBoard();
      Button* get_refresh();

      Gdk::Cursor* get_cursor();
      void create_cursor();
      void create_cursor(Gdk::CursorType cursor_Type);

private:
      MainWindow* hwnd;
      ModelDrawing* mdl_Draw;
      Gdk::Cursor* cursor;

protected:
      VBox* left_VBox;
      Frame* drawing_Frame;
      HBox* tool_DivideBox;
      VBox* tool_VBox;
      Frame* ltopbutton_Frame;
      Toolbar* toptool_Bar;
      Frame* clrbtn_Frame;
      HBox* clrbtn_Box;

      DrawingArea* sketchBoard;
      Image* ellipse_Image;
```

```
	ToolButton* ellipse_ToolButton;
	Image* eraser_Image;
	ToolButton* eraser_ToolButton;
	Image* line_Image;
	ToolButton* line_ToolButton;
	Image* pencil_Image;
	ToolButton* pencil_ToolButton;
	Image* polygon_Image;
	ToolButton* polygon_ToolButton;
	Image* rectangle_Image;
	ToolButton* rectangle_ToolButton;
	Image* arrow_Image;
	ToolButton* arrow_ToolButton;
	Button* clearSketchBoard;
	Button* refresh;
	ColorButton* color_Button;
};

#endif /* DRAWINGVIEW_H_ */

/*
 * ViewDrawing.cpp
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#include "ViewDrawing.h"

ViewDrawing::ViewDrawing()
{

}

ViewDrawing::ViewDrawing(MainWindow* hwnd)
{
	this->hwnd = hwnd;

	this->cursor = NULL;

	//	Create All Containers
	this->left_VBox = manage(new VBox());
	this->drawing_Frame = manage(new Frame());
	this->tool_DivideBox = manage(new HBox());
	this->tool_VBox = manage(new VBox());
	this->ltopbutton_Frame = manage(new Frame());
	this->toptool_Bar = manage(new Toolbar());
	this->clrbtn_Frame = manage(new Frame());
	this->clrbtn_Box = manage(new HBox());

	//	Add Child Containers to Parent Containers
	this->left_VBox->pack_start(*(this->drawing_Frame), true, true);
	this->left_VBox->pack_start(*(this->tool_DivideBox), false,
false);
	this->tool_DivideBox->pack_start(*(this->tool_VBox), true, true);
	this->tool_DivideBox->pack_end(*(this->clrbtn_Frame), false,
false);
```

```
	this->clrbtn_Frame->add(*(this->clrbtn_Box));
	this->tool_VBox->pack_start(*(this->ltopbutton_Frame), false,
false);

	//	Place Tool Containers into Left Frames
	this->ltopbutton_Frame->add(*(this->toptool_Bar));

	//	Build All Widgets
	this->sketchBoard = manage(new DrawingArea());
	this->ellipse_Image = manage(new
Image("./Tools_Images/ellipse.jpg"));
	this->ellipse_ToolButton = manage(new ToolButton(*(this-
>ellipse_Image), ""));
	this->line_Image = manage(new Image("./Tools_Images/line.jpg"));
	this->line_ToolButton = manage(new ToolButton(*(this-
>line_Image), ""));
	this->polygon_Image = manage(new
Image("./Tools_Images/polygon.jpg"));
	this->polygon_ToolButton = manage(new ToolButton(*(this-
>polygon_Image), ""));
	this->rectangle_Image = manage(new
Image("./Tools_Images/rectangle.jpg"));
	this->rectangle_ToolButton = manage(new ToolButton(*(this-
>rectangle_Image), ""));
	this->eraser_Image = manage(new
Image("./Tools_Images/eraser.jpg"));
	this->eraser_ToolButton = manage(new ToolButton(*(this-
>eraser_Image), ""));
	this->pencil_Image = manage(new
Image("./Tools_Images/pencil.jpg"));
	this->pencil_ToolButton = manage(new ToolButton(*(this-
>pencil_Image), ""));
	this->arrow_Image = manage(new
Image("./Tools_Images/arrow.jpg"));
	this->arrow_ToolButton = manage(new ToolButton(*(this-
>arrow_Image), ""));
	this->color_Button = manage(new ColorButton());
	this->clearSketchBoard = manage(new Button("Clear"));
	this->refresh = manage(new Button("Refresh"));

	//	Add Left Widgets to Containers
	this->drawing_Frame->add(*(this->sketchBoard));
	this->toptool_Bar->add(*(this->pencil_ToolButton));
	this->toptool_Bar->add(*(this->eraser_ToolButton));
	this->toptool_Bar->add(*(this->rectangle_ToolButton));
	this->toptool_Bar->add(*(this->polygon_ToolButton));
	this->toptool_Bar->add(*(this->ellipse_ToolButton));
	this->toptool_Bar->add(*(this->line_ToolButton));
	this->toptool_Bar->add(*(this->arrow_ToolButton));
	this->clrbtn_Box->pack_start(*(this->clearSketchBoard), true,
true);
	this->clrbtn_Box->pack_start(*(this->refresh), true, true);
	this->clrbtn_Box->pack_start(*(this->color_Button), true, true);

	//	Set Default Size
	this->left_VBox->set_size_request(800, 600);
	this->clrbtn_Frame->set_size_request(250, 120);
```

```
	this->toptool_Bar->set_size_request(-1, 120);
}

ViewDrawing::~ViewDrawing()
{

}

void ViewDrawing::set_Model(ModelDrawing* mdl_Draw)
{
	this->mdl_Draw = mdl_Draw;
}

DrawingArea* ViewDrawing::get_sketchBoard()
{
    return this->sketchBoard;
}

VBox* ViewDrawing::get_left_VBox()
{
	return this->left_VBox;
}

ToolButton* ViewDrawing::get_ellipse_ToolButton()
{
    return this->ellipse_ToolButton;
}

ToolButton* ViewDrawing::get_eraser_ToolButton()
{
    return this->eraser_ToolButton;
}

ToolButton* ViewDrawing::get_line_ToolButton()
{
    return this->line_ToolButton;
}

ToolButton* ViewDrawing::get_pencil_ToolButton()
{
    return this->pencil_ToolButton;
}

ToolButton* ViewDrawing::get_polygon_ToolButton()
{
    return this->polygon_ToolButton;
}

ToolButton* ViewDrawing::get_rectangle_ToolButton()
{
    return this->rectangle_ToolButton;
}

ToolButton* ViewDrawing::get_arrow_ToolButton()
{
    return this->arrow_ToolButton;
}
```

```
ColorButton* ViewDrawing::get_color_Button()
{
    return this->color_Button;
}

Button* ViewDrawing::get_refresh()
{
    return this->refresh;
}

Button* ViewDrawing::get_clearSketchBoard()
{
    return this->clearSketchBoard;
}

Gdk::Cursor* ViewDrawing::get_cursor()
{
    return this->cursor;
}

void ViewDrawing::create_cursor()
{
    this->cursor = new Gdk::Cursor();
}

void ViewDrawing::create_cursor(Gdk::CursorType cursor_Type)
{
    delete this->cursor;
    this->cursor = new Gdk::Cursor(cursor_Type);
}
```

```
/*
 * ViewQueue.h
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#ifndef VIEWQUEUE_H_
#define VIEWQUEUE_H_

#include "includes.h"
#include "Student.h"
#include "ModelQueue.h"
#include "ModelColumns.h"

class MainWindow;

class ViewQueue
{
protected:
      Frame* notebook_Frame;
      Notebook* notebook;
      ScrolledWindow* student_Window;
      HButtonBox* m_ButtonBox;
      Button* m_Button_Quit;
      Button* m_Button_Select;
    ModelColumns* m_Columns;
      Glib::RefPtr<Gtk::ListStore> m_refTreeModel;
      TreeView* m_TreeView;
      TreeModel::Row row;
public:
      ViewQueue();
      ViewQueue(MainWindow* hwnd);
      ~ViewQueue();
      void set_Model(ModelQueue* mdl_Que);
      Window* get_window();
      Frame* get_notebook_Frame();
      Notebook* get_notebook();
      ScrolledWindow* get_student_Window();
      Button* get_Quit_Button();
      Button* get_Select_Button();
      TreeView* get_Tree_View();
      TreeModel::Row& get_Tree_Model_Row();
      Glib::RefPtr<Gtk::ListStore>& get_List_Store();
      ModelColumns* get_Model_Columns();
    void complete_GUI();
    Student* get_Selected_Student();
private:
      MainWindow* hwnd;
      Window* dialog;
      VBox* main_layout;
      ModelQueue* mdl_Que;
};
#endif /* VIEWQUEUE_H_ */

/*
 * ViewQueue.cpp
```

```
 *
 *  Created on: Feb 14, 2010
 *      Author: Matthew
 */

#include "ViewQueue.h"
#include "Student.h"

ViewQueue::ViewQueue()
{

}

ViewQueue::ViewQueue(MainWindow* hwnd)
{
	this->hwnd = hwnd;
	this->dialog = new Window();
	this->dialog->set_default_size(400, 300);
	this->dialog->set_border_width(1);
	this->dialog->set_title("Student Queue");

    //  Main Layout Widget
	this->main_layout = manage(new VBox());

	//    Create All Containers
	this->notebook_Frame = manage(new Frame());

	//    Build All Widgets
	this->notebook = manage(new Notebook());
	this->student_Window = manage(new ScrolledWindow);
	this->student_Window->set_policy(POLICY_NEVER, POLICY_ALWAYS);

	ustring tab1_label = "Student Queue";
	this->notebook->prepend_page(*(this->student_Window), tab1_label,
false);

	this->m_ButtonBox = manage(new HButtonBox(BUTTONBOX_END, 1));
    this->m_Button_Quit = manage(new Button("Quit"));
	this->m_Button_Select = manage(new Button("Select"));

	//    Set Size Request
	this->notebook_Frame->set_size_request(-1, 300);

	//    Add Right Widgets to Containers
	this->m_ButtonBox->pack_end(*(this->m_Button_Select), false,
false);
	this->m_ButtonBox->pack_end(*(this->m_Button_Quit), false,
false);
	this->notebook_Frame->add(*(this->notebook));
	this->main_layout->pack_start(*(this->notebook_Frame), true,
true);
	this->main_layout->pack_end(*(this->m_ButtonBox), true, true);
	this->dialog->add(*(this->main_layout));
}

ViewQueue::~ViewQueue()
{
```

```
}

void ViewQueue::set_Model(ModelQueue* mdl_Que)
{
	this->mdl_Que = mdl_Que;
}

Frame* ViewQueue::get_notebook_Frame()
{
	return this->notebook_Frame;
}

Notebook* ViewQueue::get_notebook()
{
	return this->notebook;
}

ScrolledWindow* ViewQueue::get_student_Window()
{
	return this->student_Window;
}

Window* ViewQueue::get_window()
{
	return this->dialog;
}

Button* ViewQueue::get_Quit_Button()
{
    return this->m_Button_Quit;
}

Button* ViewQueue::get_Select_Button()
{
    return this->m_Button_Select;
}

TreeView* ViewQueue::get_Tree_View()
{
    return this->m_TreeView;
}

TreeModel::Row& ViewQueue::get_Tree_Model_Row()
{
    return this->row;
}

Glib::RefPtr<Gtk::ListStore>& ViewQueue::get_List_Store()
{
    return this->m_refTreeModel;
}

ModelColumns* ViewQueue::get_Model_Columns()
{
    return this->m_Columns;
}
```

```
void ViewQueue::complete_GUI()
{
    this->m_Columns = new ModelColumns();
    this->m_TreeView = manage(new TreeView());
    this->student_Window->add(*(this->m_TreeView));


    this->m_refTreeModel = ListStore::create(*(this->m_Columns));
    this->m_TreeView->set_model(this->m_refTreeModel);


      Student* student;
      int place = 0;
      while((student = this->mdl_Que->get_Student(place)) != NULL)
      {
        row = *(m_refTreeModel->append());

        row[m_Columns->m_col_id] = place + 1;

          Glib::ustring name = student->get_first_Name().append(" ");
          name.append(student->get_last_Name());
          row[m_Columns->m_col_name] = name;

          row[m_Columns->m_col_subject] = student->get_subject_Name();

          place++;
      }

      //Add the TreeView's view columns:
      //This number will be shown with the default numeric formatting.
      m_TreeView->append_column("ID", m_Columns->m_col_id);
      m_TreeView->append_column("Name", m_Columns->m_col_name);
      m_TreeView->append_column("Subject", m_Columns->m_col_subject);

    //Display a progress bar instead of a decimal number:
      CellRendererProgress* cell = manage(new
Gtk::CellRendererProgress);
      int cols_count = m_TreeView->append_column("Time Waited", *cell);
      //row[m_Columns.m_col_time] = 15;
      TreeViewColumn* pColumn = m_TreeView->get_column(cols_count - 1);
      if(pColumn)
      {
            pColumn->add_attribute(*cell, "value", m_Columns-
>m_col_time);
      }

      this->dialog->show_all_children();
}
```

```
/*
 *
=====================================================================
==============
 *
 *       Filename:  tutord.c
 *
 *    Description:  This is the source of the Tutor Application daemon.
This
 *                  server handles all communications between tutors
and students.
 *
 *        Version:  1.0
 *        Created:  04/12/2010 02:09:55 PM
 *       Revision:  none
 *       Compiler:  gcc
 *
 *         Author:  Andrew Hamilton
 *        Company:
 *
 *
=====================================================================
==============
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <errno.h>
#include <fcntl.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <string.h>
#include <strings.h>
#include <sys/stat.h>
#include <poll.h>

#define BUF_SIZE    1024

int get_socket();
int auth_users(char *msg);
int init_server(int type, const struct sockaddr *addr,
        socklen_t alen, int qlen);
void serve(int draw_fd, int chat_fd);

int main(int argc, char *argv[]) {
    int                 draw_sockfd, chat_sockfd;
    struct addrinfo     *drawing_list, *aip_1, *chat_list, *aip_2;
    struct addrinfo     hint;
    char                *host;

    host = malloc(256);
    if(host == NULL) exit(3);
    // if(gethostname(host, 256) < 0) exit(4);
```

```
    host = "130.166.39.188";

    /* socket = get_socket();*/
    memset(&hint, 0, sizeof(struct addrinfo));
    hint.ai_flags =     AI_PASSIVE;
    hint.ai_family =    AF_INET;
    hint.ai_socktype =  SOCK_STREAM;
    hint.ai_protocol =  0;
    hint.ai_addrlen =   0;
    hint.ai_canonname = NULL;
    hint.ai_addr =      NULL;
    hint.ai_next =      NULL;

    /* Get a list of addresses for the host machine */
    if(getaddrinfo(host, "3490", &hint, &drawing_list) != 0) {
        fprintf(stderr, "tutord: Error getting address information \ton
port 3490\n");
        exit(5);
    }
    if(getaddrinfo(host, "3491", &hint, &chat_list) != 0) {
        fprintf(stderr, "tutord: Error getting address information \ton
port 3490\n");
        exit(5);
    }

    for(aip_1 = drawing_list; aip_1 != NULL; aip_1->ai_next) {
        if((draw_sockfd = initserv(SOCK_STREAM, aip_1->ai_addr,
                        aip_1->ai_addrlen, 100)) >= 0) {
            break;
        }
    }

    for(aip_2 = chat_list; aip_2 != NULL; aip_2->ai_next) {
        if((chat_sockfd = initserv(SOCK_STREAM, aip_2->ai_addr,
                       aip_2->ai_addrlen, 100)) >= 0) {
            break;
        }
    }

    serve(draw_sockfd, chat_sockfd);
    exit(0);
    return 1;
}

int initserv(int type, const struct sockaddr *addr, socklen_t alen,
        int qlen) {

    int fd, err;
    int reuse = 1;

    if((fd = socket(addr->sa_family, type, 0)) < 0) return -1;

    /* if(setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(int))
< 0) {
        err = errno;
        close(fd);
        errno = err;
```

```
        return(-1);
    }*/

    if(bind(fd, addr, alen) < 0) {
        err=errno;
        close(fd);
        errno = err;
        return -1;
    }

    if(type == SOCK_STREAM || type == SOCK_SEQPACKET) {
        if(listen(fd, qlen) < 0) {
            err = errno;
            close(fd);
            errno = err;
            return -1;
        }
    }

    return(fd);
}

void serve(int draw_sockfd, int chat_sockfd) {
    int     draw_fd, chat_fd;
    int     len, auth, tutor_chat, tutor_draw, student_chat;
    int     tutor_file, chat_len, str_len, timeout_msecs;
    size_t  read_len;
    pid_t   pid, chat_pid;
    char    msg[BUF_SIZE], chat_msg_inbound[BUF_SIZE];
    char    chat_msg_outbound[BUF_SIZE], header[BUF_SIZE];
    size_t  nbytes;
    ssize_t bytes_read;
    struct pollfd fds[2];

    for(;;) {
        if((draw_fd = accept(draw_sockfd, NULL, NULL)) < 0) {
            fprintf(stderr, "tutord: Error accepting connection for
drawing\n");
            exit(6);
        }

        /* ************************************************
         * After accepting the connection we check to see whether the
username
         * and password combination given are valid before splitting
off a child
         * process.
         * ************************************************ */
      if(read(draw_fd, header, BUF_SIZE) == -1) {
            fprintf(stderr, "tutord: Error receiving message\n");
            break;
        }

        if((len = strlen(header)) > BUF_SIZE) {
            fprintf(stderr, "tutord: More information than the allotted
buffer was received\n");
            break;
```

```
        }

      if(read(draw_fd, msg, BUF_SIZE) == -1) {
            fprintf(stderr, "tutord: Error receiving message\n");
            break;
        }

        auth = auth_users(msg);
        write(draw_fd, (void *)&auth, 1);

        if(auth != '1') {
            fprintf(stderr, "tutord: Authorization failed\n");
            close(draw_fd);
            break;
        }

        /* ************************************************
         * Since the user has been authorized, we will now accept a
connection
         * for the chat side of the messages.
         * ************************************************ */
      if((chat_fd = accept(chat_sockfd, NULL, NULL)) < 0) {
            fprintf(stderr, "tutord: Error accepting connection for
chat\n");
            exit(6);
        }

        if((pid = fork()) < 0) {
            exit(7);
        } else if (pid == 0) { /* Child process */
            /* This is where the magic happens...(cue music) */


            /* ************************************************
             * For another child to be able to work on the drawing side
and on
             * the chat side.  The drawing side will only take in
information
             * and the chat side will both send a receive information.
             * ************************************************ */
            if((chat_pid = fork()) < 0) {
                fprintf(stderr, "tutord: Error forking the chat
process\n");
                break;
            } else if(chat_pid == 0) {
                /* ************************************************
                 * This is the process in charge of handling the chat
side of
                 * the transfers.
                 *
                 * The file descriptor for the chatting side is
chat_fd.
                 * ************************************************
*/
```

```
                if((tutor_chat = open("/tmp/ta_000999/tutor_chat",
O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) <
0) {
                    fprintf(stderr, "tutord: Error opening
tutor_file\n");
                    break;
                }

                /*  if((tutor_draw = open("/tmp/ta_000999/tutor_draw",
O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) <
0) {
                    fprintf(stderr, "tutor: Error opening
tutor_draw\n");
                    break;
                }*/

                if((student_chat = open("/tmp/ta_000999/student_chat",
O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR | S_IRGRP |S_IRGRP |
S_IROTH | S_IWOTH)) < 0) {
                    fprintf(stderr, "tutor: Error opening
student_draw\n");
                    break;
                }
            if((fchmod(student_chat,
S_IRWXU|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH)) == -1) {
                fprintf(stderr, "tutor: could not change permission on
student_chat file\n");
                break;
            }
            read_len = 1024;
                fds[0].fd = chat_fd;
                fds[0].events = POLLIN;

                timeout_msecs = 3000;

                do {
                    poll(fds, 1, timeout_msecs);

                    if(fds[0].revents & POLLIN) {
                        bzero(chat_msg_inbound, BUF_SIZE);

                        if((read_len = read(chat_fd, chat_msg_inbound,
BUF_SIZE)) == -1) {
                            fprintf(stderr, "tutord: Error receiving
tutor chat\n");
                            break;
                        }

                  if((chat_len = strlen(chat_msg_inbound)) > BUF_SIZE)
{
                            break;
                        }

                  if(write(tutor_chat, chat_msg_inbound, chat_len) == -
1) {
                            fprintf(stderr, "tutor: Write to tutor_file
failed\n");
```

```
                            break;
                        }
                    }

                    do {
                        bzero(chat_msg_outbound, BUF_SIZE);

                        if(read(student_chat, chat_msg_outbound,
BUF_SIZE) == -1) {
                            fprintf(stderr, "tutord: Error reading
student chat\n");
                            break;
                        }

                        if((chat_len = strlen(chat_msg_outbound)) >
BUF_SIZE) {
                            break;
                        }

                        if(write(chat_fd, chat_msg_outbound, chat_len)
== -1) {
                            fprintf(stderr, "tutord: Error writing to
tutor\n");
                            break;
                        }
                    } while(chat_len != 0);
                } while(!(fds[0].revents & POLLIN) || read_len > 0);

            close(chat_fd);
                close(student_chat);
                close(tutor_chat);
                close(tutor_draw);
            exit(0);
            } else {
                /* ************************************************
                 * This is the process in charge of handling the
drawing side of
                 * the transfers.
                 *
                 * The file descriptor for the drawing side is draw_fd.
                 * ************************************************
*/
                if((tutor_file = open("/tmp/ta_000999/tutor_draw",
O_RDWR | O_CREAT | O_TRUNC, S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH)) <
0) {
                    fprintf(stderr, "tutor: Error opening
tutor_draw\n");
                    break;
                }

                do {

                    bzero(msg, sizeof(msg));

                    if((read_len = read(draw_fd, msg, BUF_SIZE)) == -1)
{
```

```
                        fprintf(stderr, "tutord: Error receiving
header\n");
                        break;
                    }

                    if((len = strlen((const char *)msg)) > BUF_SIZE) {
                        fprintf(stderr, "tutord: Too much information
read.\n");
                        break;
                    }

                    if((write(tutor_file, msg, len)) == -1) {
                    fprintf(stderr, "tutord: Error writing
infomation.\n");
                        break;
                    }
                } while(read_len > 0);
            }
            close(tutor_file);
            close(draw_fd);
         exit(0);
        }
    }
}

int auth_users(char *msg) {
    int auth_file, authz;

    authz = '1';
    // auth_file = open("/var/tutorapp/auth.txt", O_RDONLY);


    return authz;
}

char *student_queue() {

    return NULL;
}
```