1.2 Problem Description

Write a chat client using the distributed [big-bang](http://docs.racket-lang.org/teachpack/2htdpuniverse.html#%28form._world._%28%28lib._2htdp%2Funiverse..rkt%29._big-bang%29%29) from 2htdp/universe.

1.3 General

A chat client window is divided into three sections.

1. On the left is a column displaying the user names of all chat participants. This area should be 100*x*400*pixels*. The names should be listed in lexicographic order, top to bottom. This section should be updated whenever users enter or leave the chat. If there are more users than fit in this part of the window, then the user list should be truncated, starting with the usernames that are lexicographically last.

UPDATE 2015-03-10: Sort the names without regard to case. This means that the usernames Stephen and stephen may appear in any order. You may need to give a custom comparator to [sort](http://docs.racket-lang.org/htdp-langs/intermediate-lam.html#%28def._htdp-intermediate-lambda._%28%28lib._lang%2Fhtdp-intermediate-lambda..rkt%29._sort%29%29) and you may find the [char-downcase](http://docs.racket-lang.org/htdp-langs/intermediate-lam.html#%28def._htdp-intermediate-lambda._%28%28lib._lang%2Fhtdp-intermediate-lambda..rkt%29._char-downcase%29%29) function useful.

1. On the bottom is an area for the user to enter a message. This area should resemble the one-line editor from previous problem sets and should be 300*x*20*pixels* in size. Specifically:
   * The left and right keys move the cursor.
   * Pressing \b (backspace) erases the character to the left of the cursor.
   * Pressing \t (tab) should autocomplete a user name. Specifically, if the cursor is all the way to the right in the editor and the text entered so far is a prefix of a current chat participant, then pressing tab should change the editor’s contents to that chat participant’s user name, with the cursor still at the right. If there is more than one match, choose the match that would appear first lexicographically. If there are no matches, then the tab key is ignored.

UPDATE 2015-03-10: Case-sensitivity matters when auto-completing. Only autocomplete when there is an exact prefix match. So ste may autocomplete tostephen but not Stephen.

* + Pressing enter should send the currently entered message. Sending message is discussed below.
  + This area has no size limit. If the entered message is too large to display, only the most recently entered characters are displayed. However, the entire message should still be sent to the server.
  + UPDATE 2015-03-06: When the entered message is too large to fit entirely within the editor, pressing "left" and "right" at the edges should scroll what is displayed, as in a typical text editor. Specifically:
    - When the cursor is all the way at the right edge, and "right" is pressed, if there are more characters available on the right, then one additional character should appear at the right, and some characters should disappear at the left to compensate. Shift out enough characters on the left such that no characters get cut off when rendering the editor. The cursor should still be rendered all the way to the right,
    - and vice versa when the cursor is all the way to the left edge and "left" is pressed.
    - If the cursor is somewhere in the middle of the editor, then only the position of the cursor changes when "left" or "right" is pressed.

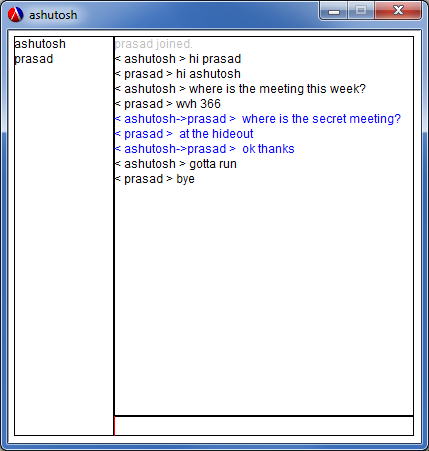
1. The remaining area should display received messages, sent messages, and other events, with the most recent messages at the bottom.
   * No more than 25 events should be displayed in this area.
   * If there’s not enough room to render 25 events, then render as many events as possible, giving preference to more recently received events.
   * When the history exceeds 25 events, the oldest message should be dropped from the chat state.
   * A message should be displayed with the originating user name in angle brackets, with a single space inside each bracket, followed by a space outside the right bracket, followed by the message content.
   * Private messages should be formatted slightly differently, as explained below.
   * If an event cannot be cannot be rendered within the width of the display area, it should be rendered on multiple lines, fitting as many words as possible on the first line, then fitting as many words as possible on the second line, etc.

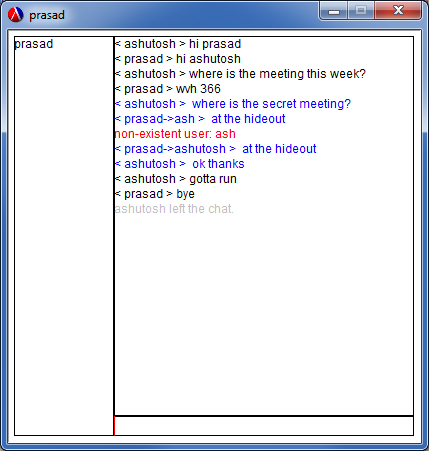
UPDATE 2015-03-10: If one word exceeds the width of the display area, then fit as many characters as possible on each line.

* + Join and leave events should be rendered in gray.
  + Error messages from the server should be rendered in red.

Use black 12 point font to display all text, except where otherwise mentioned.

Here are some screenshots.





1.4 Private Messages

By default, a message is *broadcast* to all chat participants. A *private* message may be sent by prefixing a message in the editor input area with a user name, followed with a :(followed by the message) in the input part of the window. For example, if Ashutosh wishes to send Prasad a private message, he would enter into the editor:

prasad: secret message

Display private messages in blue.

For private message senders, both the user’s name and the message target’s name should be displayed, separated by a ->. For private message receivers, only the sender’s name should be displayed.

UPDATE 2015-03-10: The chat client should only attempt to send a private message if the text before the : is a valid UserName. Otherwise, it should send a broadcast message with the entire entered message, including the text before the : and the : itself.

1.5 Server Protocol

1.5.1 Client to Server

If a client wishes to send a message to the server, here is the format that the messages should have:

|  |
| --- |
| ; A UserName is a String, consisting of only letters and numbers, |
| ; and is between 1 and 12 characters long. |
| ; Represents a chat room participant. |
|  |
| ; A Message is a String |
|  |
| ; MsgToServer is a: |
| ; - (list 'broadcast Message) |
| ; - (list 'private UserName Message) |
| ; The user in the private message is the intended recipient of the message. |

1.5.2 Server to Client

A chat client must also handle messages from the server. Here are all possible messages that a client may receive from the server.

|  |
| --- |
| ; A MsgFromServer is a: |
| ; - (list 'userlist ListOf<UserName>) ; all the current chat participants |
| ; - (list 'join UserName) ; the user that just joined |
| ; - (list 'leave UserName) ; the user that just left |
| ; - (list 'error Message) ; an error message from the server |
| ; - (list 'private UserName Message) ; a priv msg from the specified user |
| ; - (list 'broadcast UserName Message) ; a broadcast msg from the specified user |

A chat client can rely on the server to relay actions from other chat participants. However, a server will not tell a client about it’s own behavior, so it’s up the client itself to save and render events like sent messages.

The user list in the left part of the chat window should be updated on receiving auserlist, join, or leave message.

The other received messages should be rendered as described above.

1.6 Required Functions

Use the following run function to start your chat client.

|  |
| --- |
| ; run : UserName IPAddress -> World |
| ; Connect to the given chat server with user name nam. |
| ([define](http://docs.racket-lang.org/htdp-langs/intermediate-lam.html#%28form._%28%28lib._lang%2Fhtdp-intermediate-lambda..rkt%29._define%29%29) (run nam server) |
| ([big-bang](http://docs.racket-lang.org/teachpack/2htdpuniverse.html#%28form._world._%28%28lib._2htdp%2Funiverse..rkt%29._big-bang%29%29) (mk-world nam) |
| ([on-receive](http://docs.racket-lang.org/teachpack/2htdpuniverse.html#%28form._universe._%28%28lib._2htdp%2Funiverse..rkt%29._on-receive%29%29) receive) |
| ([to-draw](http://docs.racket-lang.org/teachpack/2htdpuniverse.html#%28form._world._%28%28lib._2htdp%2Funiverse..rkt%29._to-draw%29%29) render) |
| ([on-key](http://docs.racket-lang.org/teachpack/2htdpuniverse.html#%28form._world._%28%28lib._2htdp%2Funiverse..rkt%29._on-key%29%29) key-handler) |
| ([name](http://docs.racket-lang.org/teachpack/2htdpuniverse.html#%28form._world._%28%28lib._2htdp%2Funiverse..rkt%29._name%29%29) nam) |
| ([register](http://docs.racket-lang.org/teachpack/2htdpuniverse.html#%28form._universe._%28%28lib._2htdp%2Funiverse..rkt%29._register%29%29) server) |
| ([port](http://docs.racket-lang.org/teachpack/2htdpuniverse.html#%28form._universe._%28%28lib._2htdp%2Funiverse..rkt%29._port%29%29) 5010))) |

The server will reject connections with an invalid UserName, or if the given UserName is already taken.

In addition, you must define and provide the following functions.

|  |
| --- |
| ; A HandlerResult is a: |
| ; - World |
| ; - (make-package World MsgToServer) |
|  |
| ; mk-world : UserName -> World |
| ; Returns the initial world state for user name. |
| ([define](http://docs.racket-lang.org/htdp-langs/intermediate-lam.html#%28form._%28%28lib._lang%2Fhtdp-intermediate-lambda..rkt%29._define%29%29) (mk-world nam) [...](http://docs.racket-lang.org/htdp-langs/intermediate-lam.html#%28form._%28%28lib._lang%2Fhtdp-intermediate-lambda..rkt%29._......%29%29)) |
|  |
| ; receive : World MsgFromServer -> HandlerResult |
| ; Handles messages received from the server. |
|  |
| ; key-handler : World KeyEvent -> HandlerResult |
| ; Handles keyboard user input. |
|  |
| ; get-users : World -> ListOf<UserName> |
| ; Returns a list of current chat participants, in lexicographic order. |
|  |
| ; get-editor : World -> Editor |
| ; Returns a representation of the chat client's input area. |
|  |
| ; get-editor-pre : Editor -> String |
| ; get-editor-post : Editor -> String |
| ; Returns an editor's content before and after the cursor. |
|  |
| ; get-chat-history : World -> ListOf<String> |
| ; Returns a list of chat events, rendered to string, |
| ; where each string format is the same as when the event is |
| ; rendered to the chat window, except the string should not be broken into |
| ; multiple lines. |

1.7 Available servers

To test your chat client, you may connect to any of these servers (port 5010):

* primary: cs5010-chat.ccs.neu.edu
* backup: 108.26.212.136