# CMPT 371 group project report

## Game Description and design

This game is a Speed-Drawing game, has a maximum of 4 players and each player should draw as fast as possible to paint most of the cells on the map. Once a cell in the map was full-filled and it would be no longer painted with another color. After all the cells are painted and the game would be over. The system will show the winner in this game.

For the game design. Please see the graph below.

In Game:

Thread 1 - Thread 4: listen to each client’s message

Thread 5: Process the message send from Thread1-4, send the proceeded message out

End Game

Close socket

Negotiate Player

Server start

ID, Color and Game start signal

Request Color and ID

User is drawing Which cell with which color.

Mouse up signal

Other player’s drawings.

Which cell is drawable.

If the game has end.

Confirm Color and Start Game signal

Client start

End Game  
Show winner

Close socket

In Game:

Thread 1: Receive the message from the server, update the game status on GUI

Thread 2: Send drawing message(color & draw location) to the server

At first, when the server starts, it would wait for clients connected to it. When a new client comes in, it would negotiate with the client and distribute the ID and color to the client. In the server inside, every client has its unique thread which is used to listen to the message from that client. After all the clients are ready to start, the server would send a game start signal to all players, and the game start.

During the game, in the server, every thread which receives the message from the client would process the message to a class data structure and put the data into the queue. Every data structure includes the information of who painted which pixel on the map. Thread 5 will drag the data from the queue and process them. For example, if there is a user full filled a cell over 50%, or which cell should be cleaned out as one of the clients just pulls up his pen. The server would broadcast the processed data to all clients to let other clients updates their GUI.

For the client during the game, there are 2 threads working for the game. One is used to receive the data from the server and update its client surface to show how much other players painted in the game. The other one is used to record the area it’s drawn and send the corresponding data to the server.

After every cell is filled out, the server would send the close command to all clients. Once the clients receive the close command, it would close the socket to the server and show who is the winner in the game. The whole process of the game is over.

## Open Socket Design in the Game

### **Open sockets design (client side):**

* When the client connections to the server: Text

  Description automatically generated

Use the received message to confirm if the connection work.

* When negotiating with the server to get the player ID:   
  (id info socket from is **“;;{“PID”: [User ID], “MAX”: [max player]};;”**, “;;” is used to determine the start and end of the message)  
  Text

  Description automatically generated  
  Set the expiration time as 100 seconds, during this time keep the client listening to the server to get ID information from the server. Once it gets info that meets the match conditions, it will update the Client ID and determine what color this user would use by the ID
* When listening to the message sent from the server, use match to let client identify different message:  
  (Standard info of this section:  
  **“;;{“UID”:[Player ID], “loc”: [x,y]};;”** -> Show how the other player drawing  
  **“;;{“Lock”:[Player ID], “loc”: [x,y]};;”**-> Show which cell is not drawable **“;;{“UID\_cell”:[Player ID], “loc”: [x,y]};;”**->Upate the whole cell in the map when someone full-fill a cell or system need to clean up a cell

**“;;CLOSEGAME;;” ->** Indicate the client the game is over

**“;;{“UID”:[Player ID], “percentage”: \*};;”** -> indicate who is the winner in this game  
)  
Text

Description automatically generated  
Receive the data and pre-process the data, split them into the matchable pieces  
Text

Description automatically generated  
Match the corresponding message, and update the client GUI according to the message

* When sending the drawing message to the server:  
  (Standard info of this section:  
  **“;{“UID”: [Player ID], “draw\_record”: [x,y], “more”: True/False};”** |  
  [UID] indicates which player is drawing,   
  [draw\_record] indicates the coordinate of drawing pixels in the map,   
  [more] indicates if the server needs to clean up the unfull-filled block to this user)  
  Text

  Description automatically generated  
  A screenshot of a computer

  Description automatically generated with low confidenceText

  Description automatically generated  
  A line consists of many pixels. When drawing a line, push the drawing pixel into the buffer. And use the JSON to transform the drawing record into a string, and add the start/termination mark “;” to the string. And send them to the server.  
    
  When a user pulls up the mouse button, the client program would use the “False” [more] flag to indicate the server to clean up the unfilled cell and release the cell locked by the user. As follows:  
  Text

  Description automatically generated  
  Repeat the False message 3 times to avoid possible packet loss
* When the game is over, the client will receive a close message to indicate that the game is over.

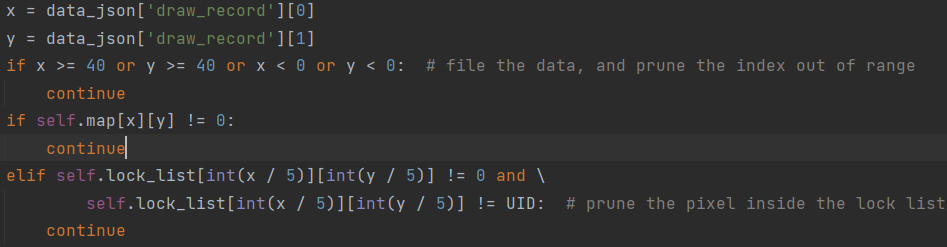
  
After this message, the client will close the panel and show who is the winner

### Open sockets design (Server side):

* When negotiating with clients and distributing ID:   
  Text

  Description automatically generated   
  Once a new user comes in, give that user a new ID  
  Text

  Description automatically generated  
  Send the distributed ID to the client, and sent it twice to avoid possible packet loss
* When receiving the message from the Client:  
  (Receive format: **“;{“UID”: [Player ID], “draw\_record”: [x,y], “more”: True/False};”,** see the sending section in client socket design)  
  Text

  Description automatically generated  
  For every message received from the client, pre-process them by the start/termination signal, and use JSON to transform the string into a python readable object. Put the object into a queue to let every client enjoys “first in first proceed” in the server
* In the process section of the client message:  
    
  Firstly pull data from the queue to process  
    
  Filter the message from the data, only calculate the data draw the pixel which is white and unlocked by another user. And throw the date out of the drawing panel.  
    
  Text

  Description automatically generated  
  Once a “good” pixel(draw in a white place, and not be locked by other users) information comes in, broadcast the pixel to all clients to let them update their GUIs. Update the map and lock list inside of the server for future calculate  
    
  Text

  Description automatically generatedText

  Description automatically generated  
  Check in the user filled over 50% of a cell by adding this pixel, if yes, then broadcast all clients, this cell has been full filled to let them update GUI, by using the string “;;{“UID\_cell”: [player id], “loc”: [x,y]};;” as the protocol. Repeat broadcast it 5 times to avoid possible packet loss.  
    
  Graphical user interface, text, application

  Description automatically generated  
  When a user pulls up his mouse button, the server would look for the cells locked by this user

Text

Description automatically generated

For every cell locked by this user but not full-filled, the server will release those user-locked cells and flush them into white. Then broadcast those messages to all clients. Repeated broadcast 5 times to avoid possible packet loss

* In the end of the game  
  Text

  Description automatically generated

Sever will check out if there is any white cell left on the map. If all the cell has been painted, the server will send the close game message and indicate who is the winner to the client and stop its socket then close itself.

The close signal is just “;;CLOSEGAME;;”, the winner is in JSON format as well. Similarly, repeat the message for few times to avoid packet loss

## Handle Shared Object Design

### Handle Shared Object Design(Client side):

On the client side, there are 2 threads. Thread1 is used to keep on receiving the message and update the GUI of the Client program. Thread2 is used to keep record the user’s drawing, calculate the pixel in the drawing line, and send them to the server. Both threads shared the same variable: map and lock list.

Graphical user interface, text

Description automatically generated  
The map shows the whole map, using 40\*40 size (8\*8 cells, each cell has 5\*5 pixels)  
lock list indicates which cell is drawable to the user, and shows the cell is controlled by who

The sending thread never modifies the map and lock list data, only reads the data inside of it to indicate which pixel or cell is drawable and which one is not. If a cell is not drawable, the client will not send the drawing record of in that cell to the server.

Text

Description automatically generated

When organizing sending record, it will not send the pixel in the locked list

Only the receiving thread can modify the map and lock list. And update the client UI. This design can ease the possible lock issue in multi-thread and let both threads run without congestion.

Text

Description automatically generated

Muti-thread running in the client, Painter is the GUI and sending thread

### Handle Shared Object (server side)

The server side has 2 kinds of threads, client listening threads, and data processing threads. The data processing thread also includes the data sending message to the clients. And all threads share the data queue of the received message. The client listener threads push the received data into the queue, and the data processing thread pulls data and processes it, sending them to all clients.

Once a new Client comes in, the server will create a new thread to receive the message from that client:



Text

Description automatically generated

When the client listening thread is running, a new message would make the listener thread lock the other thread, in order to push the data into the queue. After the pushing operation is finished, the lock will be released to let other threads push date into the queue

Text

Description automatically generated

In the data processing thread, it only pulls the data from the queue and never adds data into the queue. The detailed process algorithm is mentioned in the open socket design section, click here for more information: [Click me to see how the server handles game data](#_Open_sockets_design).

This design helps to divide the read operation and write operation in the server’s data queue. It can help to lessen the data processing congestion and thread read-and-write issues. And promises all the client can update their GUI as fast as possible.

## Contribute to the teamwork

Shenyu Gu 301385392 [shenyug@sfu.ca](mailto:shenyug@sfu.ca) 50%

Bruce Wang

This project is completed by 2 people. The other 2 students are not contributing to it.

The presentation record under this link:

<https://drive.google.com/file/d/1V4mEKv3ip6lCvIL6lh0u2XPiAgF1_xm6/view?usp=sharing>

The source code is under this link:

<https://github.com/gushenyu1998/CMPT371-Drawing-Game>