M^3 Specification

Version 1

General

M³ or Mass Multi-player Minesweeper is a multi-player minesweeper game developed in order to make a presentation.

Rule

At first, one player create a server instance and join as the master of the game.

Master have permission to adjust the game parameters.

Master may waiting for others to join or just start the 1 player game.

The first click of every user should never be a MINE.

The server stat the total opened block in the map for each player.

The client will display the field with color of others.

Game can has time limit and field size limit or just speed race.

The 3BV and other important variables should be calculated in the minefield.

Game ends when time-up or field clear or one user win or all users died.

Winner rule is defined in game parameters.

Basic minesweeper rules.

Reference

http://www.minesweeper.info/wiki/3BV

CHAPTER 1

Environment & Techs

1

This section discuss the build environment and technologies using in the project. This chapter also give a brief architecture of the whole system.

Environment

OS

Development will mostly be done under a **Unix-liked** or **Linux** based Operating System.

Also Windows is allowed for development of Qt, but not promoted.

Development Tools

Qt4 Development Suite is used for the current project.

The environment should include a full Qt4 support.

Building

GCC of course...

Image Processing and UI Design

GIMP is used for image processing

A UI draft tool called **Pencil** available as a Firefox plugin or standalone version is used to make a static User Interface design.

= =# No suitable tool found for make a full dynamic UX modeling.

Technologies

HTTP Based stateless communication

The connection between connectors will be implemented in an HTTP way and some of comet skills such as long polling.

Qt4 Animation Framework

Using the Qt4 Animation Framework will make the effect more effective (IMPORTANT in the presentation)

UDP based Game Host Auto-detect

Use the UDP broadcast to test for hosts in a local area network.

JSON Remote Procedure Call

HTTP Method with a post and a JSON data format embedded to get a peer protocol between client and server. The basic idea is that the client RPC the server and server RPC the client in a same way.

Deploy Architecture

Basic system is divided into the part of **client**, **server** and **wrapper**.

Server and client are wrapped in the wrapper and deployed to the end user.

User start-up the wrapper and then choose the working mode. The server mode will start a server thread and a client thread with local connection parameter; client mode will start a client and ask for the user for a server address.

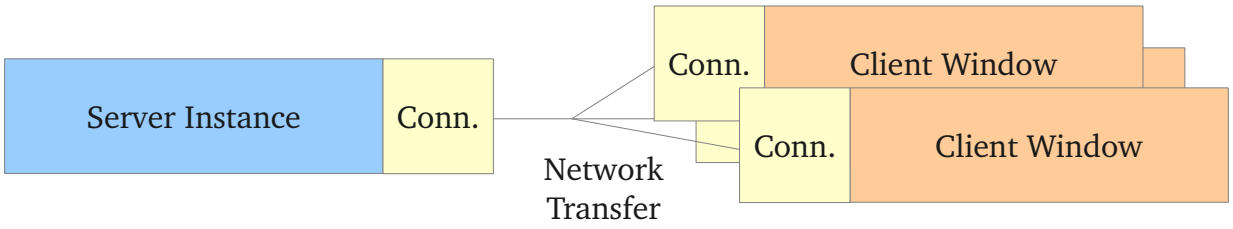
The host detecting function using UDP broadcast to search for hosts in the local network may also be implemented on the wrapper.

CHAPTER 2

Connector

2

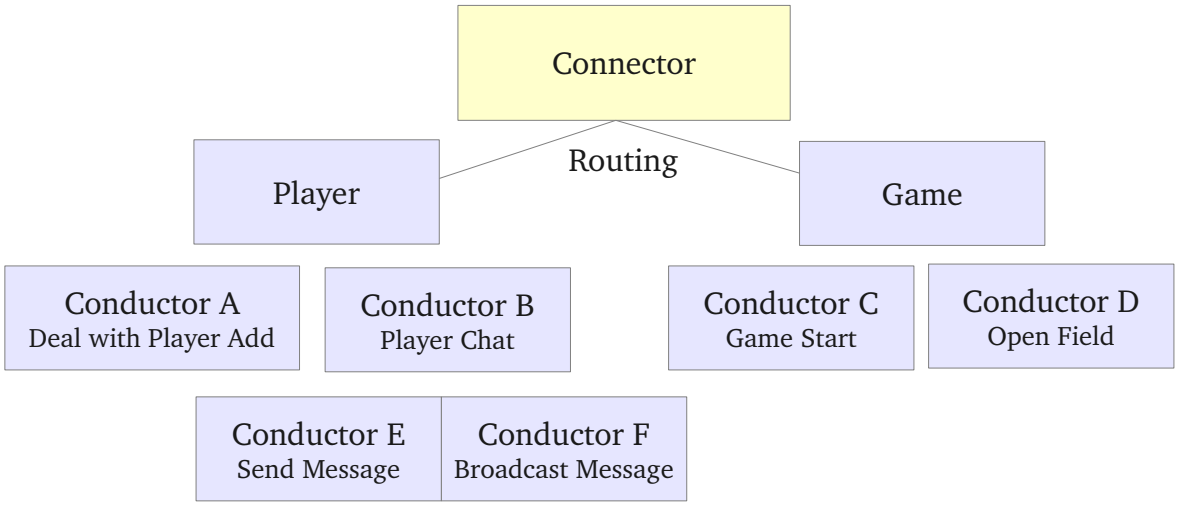
Connector is a program component for both the client and server which keep a protocol and provide primitives for client and server.



The Connector encapsulate the network transfer details and provide service to both server and client window in two different set of primitives.

Server Side Interface

Server side interface provides service to start a network listen, accept players, and dispatch operations to the Processors(Conductor in the diagram)



So, a server side connector provides register functions for Processors to be registered and provides an event based programming style interface to the server.

- Connector maintain the session and the connection.
- The sessions will be managed in group, the user session will have access to the sessions in the same group to perform actions. The interface of Session will be described later.
- Connector will recognize what the request is and use a factory method to get an instance of corresponding Processor.
- Connector will decide who is the user (User in the layer of session, not view) and pass information to Processor
- Connector keep the context (state) in a Memento Pattern (Means that connector doesn't care about what is in memory)

The interface of Processors will be described later.

Client Side Interface

Client side interface is simpler, provide register interface to register handlers (we call Processor as Listener on the client side), the diagram is exactly the same as the server side.

- A set of parameter needed to initialize the connector(Address, Port...)
- Connector will maintain only one context and one connection
- All I/O actions performed by the Client Processor will be asynchronized by and RPC request form server to client (defined by objects in Context)

CHAPTER 3

Server Logic Programming

3

This section describe the logic programming part of the server. Concept of Processor will be described in this section. Processor make the logic event driven and more flexible, reusable.

This chapter also describes the Context object interface and how the server should make use of it to get state and other things.

Server logic mainly implement the action function of the processor interface. Server response the requested client with a return phase. Processor procedure can also send message to other sessions by accessing the Session objects in the context and perform the raw send method.

Processor

Processor is an unified interface to perform operations defined as:

```
Interface {
     Response action(Local Context, Global Context); // Core logic
     static Interface instance(); // Factory method to inst processor
     static char* id(); // The identifier
}
```

And by invoking action(ctx1, ctx2) the processor will be run by the connector

- **Local Context** contains the user sessions.
- **Global Context** contains global settings, such as database handler, Read-Only!
- **Return** value will be recognized as response and have a copy stored in local context as:
  - I@`id()`=”Response Copy”
- **Return a null value** will ignore the response action.

The two context can be directly accessed from Connector on client side, but hidden on server side.

>>> NOTE!!!

The Request-and-Response pattern may be deprecated!!!

That is to say: No response! But an S-C RPC instead.

Local Context

Local Context is a structure indicating the session group. Basically contains the context id, state flag, session number, current request session id, session objects and an extra data field storing the user defined context state.

Local Context can perform actions of add_session, remove_session, etc.

Local context will has two special fields to perform action to the request initial and all sessions in Local Context.

Basic Context Elements

Session

Session Object specified actions and store the basic information of users in the current Context.

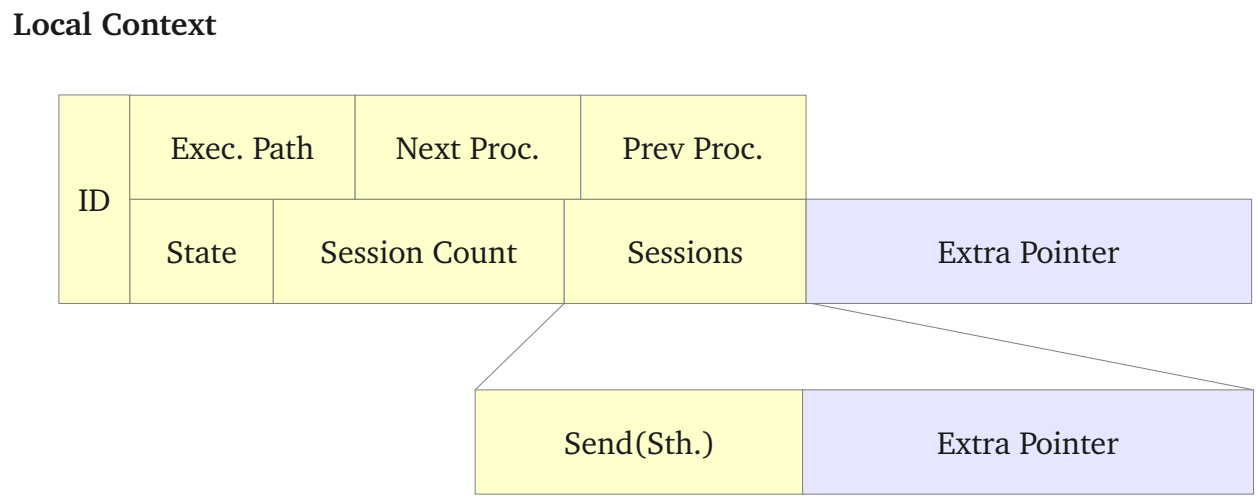
A session object typically contains a field to store the states of the session, this field is user defined Object used to store the extra user data(Store the nickname or other user specified information not relate to the connection/connector).

Session object provide interface for the processors to send data on the channel of the session and establish a mapping between extra user information and the connection.

Extra Data Field

Store the data object that not related to the connector. Because the connector need not to know the detail of the Local Context concretely stand for.

Extra Data Field is designed to be used as a pointer to the storage of data such as mine field parameters, 3BV information, Mine field information and other room based information produced/calculated by the processor.



Yellow Part Will be accessed by connector

Purple Part is dedicated for processors

Working Flow

1. Server process started
2. Find processors in some way
3. Instantiate a connector
4. Register the processors to the connector
5. Waiting for something to happen
6. Requested, connector find correct processor path PA/PB/PC
7. Mapping to the User Context in memory (Context Pool)
8. Get the instance of PA PB PC, store path to user context
9. Invoking PA.action(ctx1, ctx2), PB.action(ctx1, ctx2), PC.action(ctx1, ctx2) sequentially.
    1. PA.action decorates the ctx1
    2. PB.action decorates the ctx1
        1. PB.action invoking PD.action(ctx1, ctx2) for some reason
    3. PC.action decorates the ctx1
10. Action completed, the returned response of the PC is used for end-up response.

CHAPTER 4

Client Logic Programming

4

Logic programming on client side differs from the logic on server side in some implementation part. But with the same Connector mechanism.

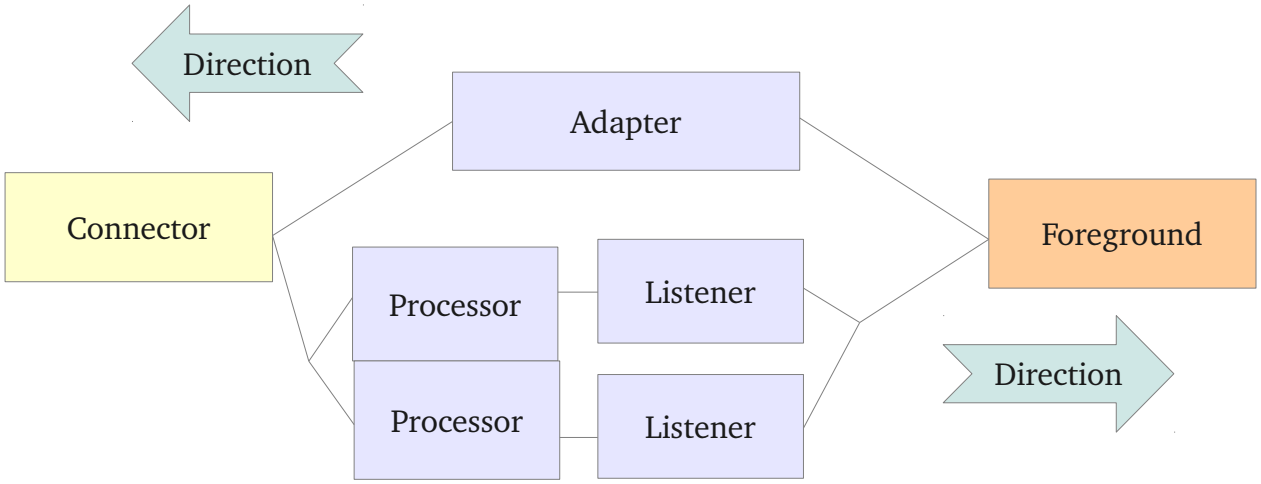
Basic Model

The basic client background model is the same as the server side. Provides RPC interface for the server to invoke and connector dispatch RPC requests from server to specific processor.

Client background logic provide the listener interface (A set of interfaces to be implement by foreground to handle formatted information from Processor Group) to handle RPC from server.

Another component only on the client side is the Adapter, which implement and encapsulate the logic for the protocol to the foreground.

The foreground of the client access data from adapters directly and register events to the listeners.



Connector

Connector provides Context and Session just like the server side. But the Session on the client side will not have the send function, instead, send function is placed at Context object and Session object of the client only store the state of player(In extra state pointer manipulated by processor).

Processor

Almost the same definition as the server side Processor to parse the specific S-C RPC.

```
Interface {
     void action(Local Context, Global Context); // Core logic
     void register(Specific Listener); //Register a listener
     static Interface instance(); // Factory method to inst processor
     static char* id(); // The identifier
}
```

A new register function is added for the UI to connect a proper Listener to handle the result of the Processor.

Adapter and Listener

Adapter provide interface to foreground for direct use. The send method of the connector should be invoked by Adapter to send something to the server.

Listener provide the interface for the foreground to receive the proceed message from server. Implement the client side RPC service.

The Protocol Layer of the client side is implemented by the Adapter and Listener.

Next section will discuss the detailed interface.

Adapter Interface

Adapter provide RPC service in direction of C-S.

Adapter Interface		
Function	**Parameters**	**Comment**
join	ROOMID	Join a Room
user	String/USER	Set player information
new	Setting	Create a Room with settings
ready		Set the ready signal of the game, when all players ready, game can start
start		Invoke by the game master
step	POINT(X,Y)	Left click on a point
test	POINT(X,Y)	Test a cell(left click on a opened cell or middle key in M$ Minesweeper)
flag	POINT(X,Y)	Flag a cell
pause		Pause the game for all, pause time can be up to 3
resume		Resume the game paused by self
leave		Leave the game Room
chat	String/MESSAGE	Chat

The foreground use an Adapter's interface to send requests, the request is judged by server and returned in a S-C RPC channel. So the result will be on the Listener, and foreground must register for these event to make the game run.

Listener Interface

Processor interface provides event based UI refresh interface to the foreground. Foreground use specific Listener implementation to handle these events.

First, the basic shape of a Listener Interface:

```
Interface {
     void some_action(Parameters);
}
```

The following table lists the designed Listener interfaces.

Listener Types

Name	Parameters	Comment
rooms	[ROOMID/ROOMINFO, ...]	The list of the rooms hosted on server
room	ROOMINFO	The room change event
users	[String/USER, ...]	A list of user information in the room
user	String/USER	New user to the room
chat	USER, String/MESSAGE	Chat message
ready	USER	Specific user is ready
config	CONFIGURATION	Room configuration change
start		Set the ready signal of the game, when all players ready, game can start
time	TIME	Game time change broadcast, countdown
gamedata	[(POINT(X,Y),State), ...]	Game data transfer
paused	USER, MY_REMAIN_PAUSE	Game has been paused by USER, I have MY_REMAIN_PAUSE pause chances.
resume		Game resumed
game_over	GAME_RESULT	Game over with result
time_up	GAME_RESULT	A timed compete is over
leave		A user leave the game room

The foreground implements Interfaces above and just register them to specified processors. Then, logic works!

CHAPTER 5

RPC Protocol Format

5

The JSON RPC data format protocol between Client and Server. This protocol will be applied by connector.

It's Simple and Stupid

RPC Protocol format is defined as an extend to the JSON (JavaScript Object Notation).

This base protocol will be applied by connector component of both the server and client. The main idea of this mechanism is to make the auto routing of function modules and provide a more clear function map.

The basic data format describes as following:

Module/Module.../Function	Payload Field

On both the server side and the client side, the Module/Module.../Function, Or called function path, is passed from Processors or Adapters to the Connector's send/request function and packed into the package above. Then, transferred in a channel defined by the connector.

CHAPTER 6

User Interface Guide

6

Here introduce the Instructions on how to build a user interface for MMM. Also some rough idea about the user interface design.

CHAPTER 7

Errors

7

This chapter introduce the error code and definition of the errors.

CHAPTER OVERFLOW

Rexycle

O

This is the recycle bin for old document content

Interface Specification

A Server is the core logic unit in the project.

Server takes the duty of managing the mine field, arbitrate the user actions and send the result to the client.

So, a server has two main layer: Logic Layer and Communication Layer.

Also there is two mapping to be maintained on server: Logic-Communication Mapping and Communication-Client mapping.

And now, one server instance only holds one game on it.

Logic-communication Interface

Specify how the logic layer work with communication layer.

L-C Interface			
Function	**Parameters**	**Return**	**Comment**
initialize	SIZE		Start the server
field		FIELD	Get the Mine Field
size		SIZE	Get the field size
open	POINT(X,Y)	FIELD	Open a mine
check	POINT(X,Y)	CHECKFIELD	Check a block and return the field
C-L Interface			
Event	**Parameters**	**Return**	**Comment**
complete			Successfully complete

Server-Client Interface

S-C Interface			
Function	**Parameters**	**Return**	**Comment**
start	SIZE		Start the server
user	USER		Send player info to server
size		SIZE	Get the field size
field		FIELD	Get the mine field
open	POINT(X,Y)	FIELDDIFF	Open a mine

S-C Interface			
check	POINT(X,Y)	CHECKFIELDDIFF	Check a block and return the field
exit			Exit a server

C-S Interface			
Event	**Parameters**	**Return**	**Comment**
lose	player		Declare a player's lost
players	players		Push new player list to client