Grand Quest: IF3230 Distributed and Parallel Systems
Distributed Marketplace Application

Release Date : Monday, April 6, 2015
Deadline Date : Sunday, April 26, 2015 11.59 p.m.

I. Background

"Thou shan't falter, as the time hast come for thee to arise. Ye hast to forestall the darkest side of Terra from resurrection. Gaia hast called upon thee, all of the bravest priest and priestess of Labtek V. Combine thine wit to gather the Philosopher Stone. Mankind's best hopes rest all with thee!"

Welcome to the expansion pack of Elder Tale. The archbishops have decided to create an Unison Market in the astral realm. In this marketplace, all adventurers can engage in trading freely with other people from all over the world. Each adventurer can gather resources from their own regions and bring them to the astral realm in order to optimize their resources. Fortunately, the cardinals have reunited all prominent alchemists for the aforementioned goal: the construction of Philosopher Stone.

All members of the distributed system class have been summoned to participate in this task. We believe that all of you are highly capable in combining networking with distributed system skills. You are asked to create a **well-designed distributed system**. For example, you are able to **minimize the number of broadcasts between servers** by **considering the minimum trade-off** (inconsistent data for a short timespan, etc). It's also expected that your application can handle **general failures**. For simplicity, you don't need to consider the security aspect of this system.

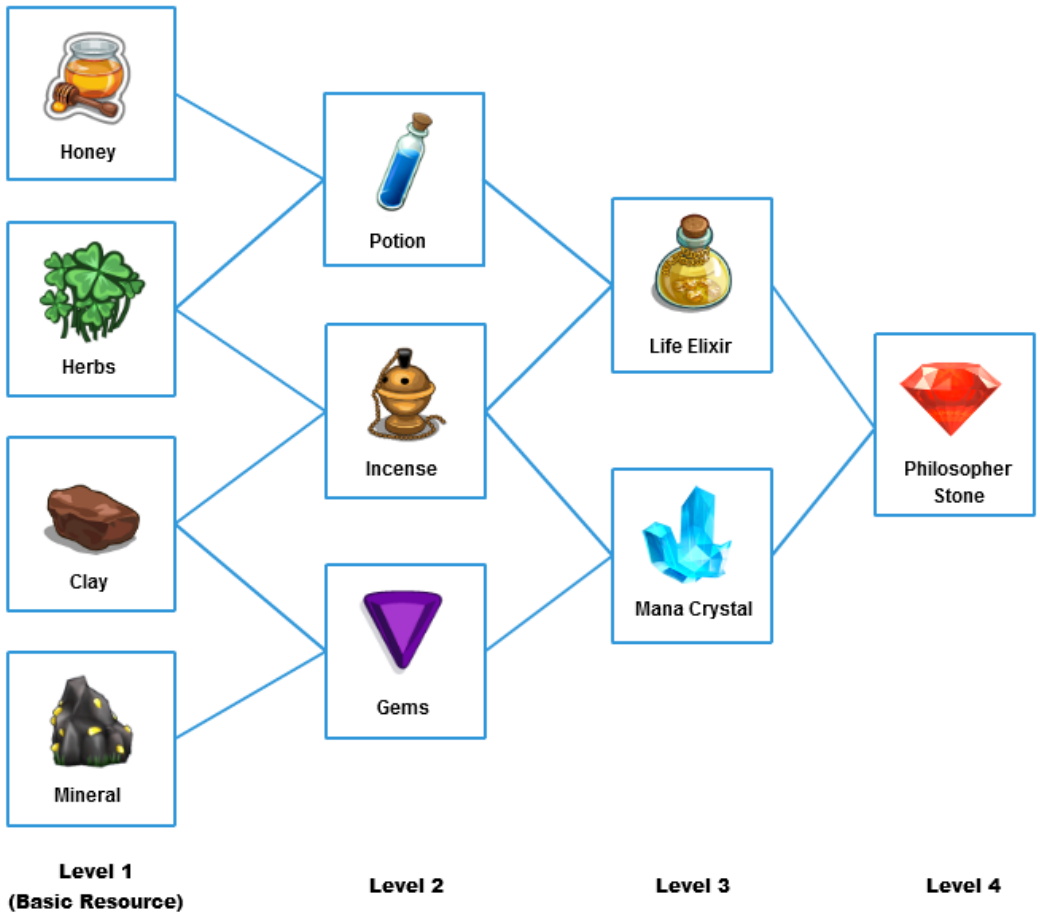
Rise up, once more!

II. Application Specification

II.1 Gameplay Description

In this task, you will build a simple online multiplayer game. In this game, each player starts in an arbitrary field in a map. The player will receive a random basic resource when player enter a field. The player can move to another neighboring field to find other resource, but it takes time (in minutes). After collecting resources, the player can combine them to make a new resource. Hence, the player task in this game is to get all possible resource and produce the ultimate resource, the philosopher stone.

Here is the resource level and recipe diagram. To make a higher level resource, **3 items** are needed for **every** ingredient resources. For example, a potion needs 3 honey and 3 herbs, a mana crystal needs 3 incense and 3 gems, etc.



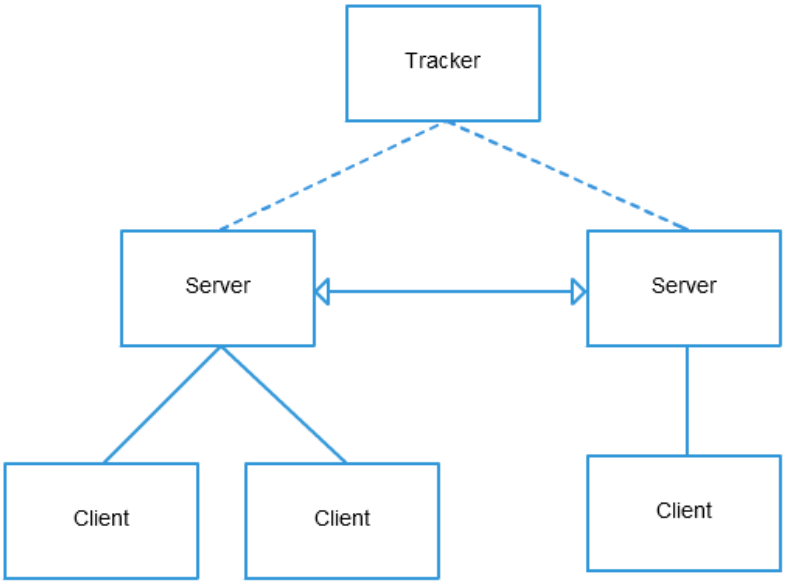
For simplicity, **Honey** will be represented as "R11" or 0 (index-based), **Herbs** will be represented as "R12" or 1 (index-based), **Potion** will be represented as "R21" or 4 (index-based), etc.

Since it takes time to collect new resources, players are encouraged to get resources by trading with each other. The trading is done as below:

1. You want to trade one of your potion for two clay. You make an offer and take one of your potion from inventory to the trade box.
2. Player X is searching for potion offerings. He gets your offer information and is interested with it. He sends two clays from his inventory to your trade box, and receives potion from your trade box to his inventory.
3. When you check your offer, you see that your offer is already responded. You finalize the offer and get two clays from your trade box to your inventory.

II.2 Architecture

This game is expected to have many players from a wide range of geographical location. Hence, multiple servers are needed so that player can connect to nearest server. It can also helps to distribute connection loads in the server. The overall architecture is shown below.



A player plays the game from a client application. Using the client application, the player may sign up to any server, but **can only** login to the server where he/she register to. All the game logic is handled in the server, so the client application only send player input to server and display server output to the player. Since a player may trade with other players from different servers, the servers can communicate to each other using **peer-to-peer** communication schemes. Server membership is maintained by a tracker application.

II.3 Protocol

All network communication is done using TCP socket. Every packet (both reply and response) sent during communication is formatted as a JSON string. TCP connection is opened when a program need to send a request and immediately closed after receiving response. Please refer to **Appendix A** for the detailed packet format, data types, endpoints and other protocol-related behaviors. Your application has to comply with all of the protocol specification, so that

every student application can communicate with each other and with assistant's provided tracker.

III. Grading Criteria

III.1 Mandatory

You need to make two programs for this assignment:

1. Server program. The server:
   - should have all server endpoints described in the **Appendix A**, except the one that marked with "(bonus)".
   - should only open one listening TCP socket, with its port number can be assigned from program parameter
   - can load the map from a JSON file. Use the JSON map format defined in the **Appendix B**.
   - handles all game logics and trading, including validation. (i.e. resource number validation when making or responding to an offer)
2. Client program. The client:
   - should have a GUI
   - can be assigned with targeted server address (using parameter or user input)
   - should not handle any game logic and trading except for interface (e.g. countdown for displaying travel time).
   - may, but not required to, handle validation especially for helping user interaction.

You can use **any programming language** that has TCP socket library, JSON library and GUI library. You may also implement the server and client program with different languages on different platforms (e.g. server uses Python on Windows, client uses Java on Android). All networking part should only be handled using standard TCP socket library, **without** any other helper library. If you want to use libraries other than mentioned above, please ask the assistant whether it is allowed. As for the tracker program, it is already provided by assistant and can be accessed at **167.205.32.46** port **8000** using the described endpoints in the **Appendix A**. It is **required** to use **openVPN** if you are accessing it from outside ITB. The implementation can be accessed at http://gitlab.informatika.org/freedomofkeima/IF3230-Tugas-Besar-Sister-2015.

III.2 Bonus

Robust Service: Additional score will be rewarded if your application can handle a backup server (one user / client may access more than one server's endpoint). For reference, you may read MongoDB Primary-Secondary model.

Battle Royale: Assistants will reward an additional 10 points for one of the best application. Your application will be graded based on server handling, user experience, and so on.

Go Mobile: Assistants will reward an additional score for client application which is developed as a mobile apps (Android / Windows Phone / iOS)

IV. Submissions and Deliverables

1. Register your team at: https://docs.google.com/spreadsheets/d/1x12MPqKUJQT8neJySDXeZe-Qj7ngMC_sE7wxzNOZc6E
2. Create a fork from http://gitlab.informatika.org/freedomofkeima/IF3230-Tugas-Besar-Sister-2015. Please commit your work regularly so that we can assess each person's contribution in this project. You can submit your work by using pull request. The deliverables should contain:
   - *bin* folder : Contains two folders, *server* and *client*. (optional)
   - *doc* folder : Contains a .doc(x) / .pdf file which contains the explanation of the designed system (database, system architecture, distributed system optimization techniques)
   - *src* folder : Contains two folders, *server* and *client*.

   As an additional note, it is required to include README.md (or/and Makefile), which contains the instructions to use your application.
3. Demonstration: The demonstration session for this task will be announced later. We are planning to arrange a joint-session for every x groups ($x \geq 2$). It is expected for each team to give a brief presentation about their work (**~ 7 minutes**).
4. We are trying to make this specification as clear as possible. Please submit bug reports or questions regarding ambiguous statements to Mailing List IF3230 (if3230@students.if.itb.ac.id).

Appendix A. Data types and Protocol Specification

All network communication is done using TCP socket. Every packet sent during communication is in JSON format. Request and response packet generally have the following format.

Request format	`{`
	`  "method": "<TYPE_NAME>",`
	`  ... // other elements, depend on type`
	`}`

OK response format	`{`
	`  "status": "ok",`
	`  ... // other elements, depend on request`
	`}`

Error response format	`{`
	`  "status": "error",`
	`  "description" : "depend on error type"`
	`}`
	
	`// responses for invalid or not complete request`

Fail response format	`{`
	`  "status": "fail",`
	`  ... // other elements depend on request`
	`}`

1. Tracker

The tracker application provides a simple server membership. It has the list of all active servers. The tracker is already provided by assistants and can be accessed at **167.205.32.46** port **8000**. Available endpoints code and its description:

1. **Join**: Join server cluster

   This endpoint is called to register new server to the server cluster. The tracker receive the request, call `NEWSERVER` endpoint in every active server in the list to announce the new server information. If a `NEWSERVER` request is not replied in **3 seconds**, the tracker deemed the corresponding server is down and drop it from the active server list. After all active server replied the tracker or timed out, the tracker send the current active server list to the new server. The new server may start its activity (as it is already registered in every active server). It should be noted that there is a probability of false failure detection.

Caller	**New server**
Request example	`{ "method": "join", "ip": "167.205.32.46", "port": 8000 }`
Response example (correct parameters)	`{ "status": "ok", "value": [ {"ip" : "167.205.32.46", "port": 8000}, {"ip" : "167.205.32.47", "port": 8000}, {"ip" : "167.205.32.48", "port": 8000} ] }`
Error Response example	`{ "status": "error", "description": "ERROR DESCRIPTION HERE" }`

2. Server-side (Application & Database Layer)

Server application **must be implemented** by all teams. You may access our servers at **167.205.32.46** port **8025** and port **8026** (both servers have different database).

It's important to note that our tracker may send **empty request** (empty data) for **ping** purposes. For example:

```
if data != "": // Ensuring data is not empty before json.loads
    request = json.loads(data)
```

1. **ServerStatus**: Receive all (current) active servers from tracker

   This endpoint should be provided by the server to retrieve all newest server list information from our tracker. For every `NEWSERVER` connections, our tracker will broadcast this request to all available servers.

Caller	**Tracker**
Request example	`{ "method": "serverStatus", "server": [ {"ip" : "167.205.32.46", "port": 8000}, {"ip" : "167.205.32.47", "port": 8000},`

	{“ip” : “167.205.32.48”, “port”: 8000}] }

OK Response example	{
	“status”: “ok”
	}

Error Response example	{
	“status”: “error”
	}

2. **Signup**: Create new user

This endpoint should be provided by the server to create new user.

Caller	**Client**

Request example	{
	“method”: “signup”,
	“username”: “nozirohilol”,
	“password”: “a8f064bbec9819f488e6402a7c570da9”
	}

OK Response example	{
	“status”: “ok”
	}

Fail Response example	{
	“status”: “fail”,
	“description”: “username exists”
	}

Error Response example	{
	“status”: “error”
	}

3. **Login**: Authentication

This endpoint should be provided by the server to authenticate the user. User must be authenticated to play. This endpoint will provide the token required to call other gameplay-related endpoints. Parameter “*time*” is epoch time in UTC (0 is default).

Caller	**Client**
Request example	{ “method”: “login”, “username”: “nozirohilol”,

	<pre> "password": "a8f064bbec9819f488e6402a7c570da9" }</pre>

OK Response example	<pre>{
	  "status": "ok",
	  "token": "47af9eb308ce2fc95bcc088add67b79e",
	  "x": 0,
	  "y": 0,
	  "time": 1427730710
	}</pre>

Fail Response example	<pre>{
	  "status": "fail",
	  "description": "username/password combination is not found"
	}</pre>

Error Response example	<pre>{
	  "status": "error"
	}</pre>

4. **Inventory**: Inquire user's inventory

This endpoint should be provided by the server to inquire user's current inventory. The OK Response will contain "inventory" list. It represents Honey, Herbs, Clay, Mineral, Potion, Incense, Gems, Elixir, Crystal, and Stone respectively.

Caller	**Client**

Request example	<pre>{
	  "method": "inventory",
	  "token": "47af9eb308ce2fc95bcc088add67b79e"
	}</pre>

OK Response example	<pre>{
	  "status": "ok",
	  "inventory": [1,2,3,4,5,1,2,3,4,5]
	}</pre>

Error Response example	<pre>{
	  "status": "error"
	}</pre>

5. **MixItem**: Combining two items

This endpoint should be provided by the server to combine two items. In the following example, we want to combine **Honey** (item1 with id = 0) with **Herbs** (item2 with id = 1). Three of each items will be combined to one **Potion** (item with id = 4).

Caller	**Client**
Request example	`{ "method": "mixitem", "token": "47af9eb308ce2fc95bcc088add67b79e", "item1": 0, "item2": 1 }`
OK Response example	`{ "status": "ok", "item": 4 }`
Fail Response example (insufficient / wrong mixture)	`{ "status": "fail", "description": "FAILURE MESSAGE HERE" }`
Error Response example	`{ "status": "error" }`

6. **Map**: Requesting gameplay map

This endpoint should be provided by the server to retrieve map's information.

Caller	**Client**
Request example	`{ "method": "map", "token": "47af9eb308ce2fc95bcc088add67b79e" }`
OK Response example	`{ "status": "ok", "name": "Bandar Behari", "width": 4, "height": 4 }`
Error Response example	`{ "status": "error" }`

7. **Move**: Move to position (x, y)

This endpoint should be provided by the server to move from current position to position (x, y), where current position is not equal to (x, y) and (x, y) is defined within the map's boundary.

Caller	**Client**

Request example	<pre>{
	  "method": "move",
	  "token": "47af9eb308ce2fc95bcc088add67b79e",
	  "x": 0,
	  "y": 1
	}</pre>

OK Response example	<pre>{
	  "status": "ok",
	  "time": 1427730710
	}</pre>

Fail Response example (otw / not move / out of boundary)	<pre>{
	  "status": "fail",
	  "description": "FAILURE MESSAGE HERE"
	}</pre>

Error Response example	<pre>{
	  "status": "error"
	}</pre>

8. **Field**: Collect item from current position

This endpoint should be provided by the server to pick item from current (newly) position. For example, position (0, 1) in our map is defined as "R13". "R13" is equal to **Clay** (item with id = 2).

Caller	**Client**

Request example	<pre>{
	  "method": "field",
	  "token": "47af9eb308ce2fc95bcc088add67b79e"
	}</pre>

OK Response example	<pre>{
	  "status": "ok",
	  "item": 2</pre>

	}

Fail Response example (otw / already collected)	{
	  “status”: “fail”,
	  "description": "FAILURE MESSAGE HERE"
	}

Error Response example	{
	  “status”: “error”
	}

9. **Offer**: Put an offer to the marketplace

This endpoint should be provided by the server to put clients’ offerings in the marketplace. Each offer will be stored in exactly one corresponding server (the server where the respective client resides). See **Tradebox** (no. 10) for better understanding.

Caller	**Client**

Request example	{
	  “method”: “offer”,
	  “token”: “47af9eb308ce2fc95bcc088add67b79e”,
	  “offered_item”: 0,
	  “n1”: 1,
	  “demanded_item”: 1,
	  “n2”: 1
	}

OK Response example	{
	  “status”: “ok”
	}

Fail Response example	{
	  “status”: “fail”,
	  "description": "insufficient"
	}

Error Response example	{
	  “status”: “error”
	}

10. **Tradebox**: Retrieve all of own offerings

This endpoint should be provided by the server to retrieve all of the clients’ offerings in the marketplace. An offer can be cancelled (see **CancelOffer**) if it’s still available (no buyer). On the other hand, the client can fetch (see **FetchItem**) the demanded item if the offer has

been completed. Each offers will be represented in an array of [**offered item id**, **number of offered item**, **demanded item id**, **number of demanded item**, **availability**, **offer token**].

For example:
`[2, 10, 3, 8, false, "0f5bb63592f716c05511cec90e171a89"]`
Offered Item : Clay
Number of Offered Item : 10
Demanded Item : Mineral
Number of Demanded Item : 8
Availability : false (**ready to fetch**)
Offer Token : "0f5bb63592f716c05511cec90e171a89" (**random md5 string, assuming uniqueness among all servers**)

Caller	**Client**

Request example	`{`
	`  "method": "tradebox",`
	`  "token": "47af9eb308ce2fc95bcc088add67b79e"`
	`}`

OK Response example	`{`
	`  "status": "ok",`
	`  "offers":`
	`  [`
	`  [2, 10, 3, 8, false, "0f5bb63592f716c05511cec90e171a89"],`
	`  [2, 1, 3, 1, true, "a8179cde12f80a011874bce9d8a01483"]`
	`  ]`
	`}`

Error Response example	`{`
	`  "status": "error"`
	`}`

11. **SendFind**: Retrieve all offers with the specified (searched) item
This endpoint should be provided by the server to support a search function for the specified item. The final result should **exclude own offerings** from the list. For example, the following query is used to search **Clay** (item with id = 2).

Caller	**Client**

Request example	`{`
	`  "method": "sendfind",`
	`  "token": "47af9eb308ce2fc95bcc088add67b79e",`

	<pre> "item": 2 }</pre>

OK Response example	<pre>{
	  "status": "ok",
	  "offers":
	  [
	  [2, 1, 3, 1, true, "a8179cde12f80a011874bce9d8a01483"]
	  ]
	}</pre>

Error Response example	<pre>{
	  "status": "error"
	}</pre>

12. **FindOffer**: Receive a search request for a specified item from other servers

This endpoint should be provided by the server to support communications between server in retrieving **SendFind** results.

Caller	**Server**

Request example	<pre>{
	  "method": "findoffer",
	  "item": 2
	}</pre>

OK Response example	<pre>{
	  "status": "ok",
	  "offers":
	  [
	  [2, 1, 3, 1, true, "a8179cde12f80a011874bce9d8a01483"]
	  ]
	}</pre>

Error Response example	<pre>{
	  "status": "error"
	}</pre>

13. **SendAccept**: Accept other people's offer

This endpoint should be provided by the server to receive an offering acceptance from client. The player who accepts an offer will automatically receive the offered items from the accepted offering.

Caller	**Client**

Request example	<pre>{
	  “method”: “sendaccept”,
	  “token”: “47af9eb308ce2fc95bcc088add67b79e”,
	  “offer_token”: “a8179cde12f80a011874bce9d8a01483”
	}</pre>

OK Response example	<pre>{
	  “status”: “ok”
	}</pre>

Fail Response example (insufficient / not exist)	<pre>{
	  “status”: “fail”,
	  "description": "FAILURE MESSAGE HERE"
	}</pre>

Error Response example	<pre>{
	  “status”: “error”
	}</pre>

14. **Accept**: Receive an acceptance request for the specified offer_token from other servers
This endpoint should be provided by the server to support communications between server in accepting an offer (from **SendAccept**).

Caller	**Server**

Request example	<pre>{
	  “method”: “accept”,
	  “offer_token”: “a8179cde12f80a011874bce9d8a01483”
	}</pre>

OK Response example	<pre>{
	  “status”: “ok”
	}</pre>

Fail Response example	<pre>{
	  “status”: “fail”,
	  "description": "offer is not available"
	}</pre>

Error Response example	<pre>{
	  “status”: “error”
	}</pre>

15. **FetchItem**: Fetch demanded items from the finished offer

This endpoint should be provided by the server to fetch the demanded items from the finished offer. It should be noted that the finished offer doesn't automatically add the demanded items to the inventory.

Caller	**Client**
Request example	`{ "method": "fetchitem", "token": "47af9eb308ce2fc95bcc088add67b79e", "offer_token": "0f5bb63592f716c05511cec90e171a89" }`
OK Response example	`{ "status": "ok" }`
Fail Response example (still available / already collected / not exist)	`{ "status": "fail", "description": "FAILURE MESSAGE HERE" }`
Error Response example	`{ "status": "error" }`

16. **CancelOffer**: Cancel own offer from the marketplace

This endpoint should be provided by the server to cancel an offer from the marketplace. When the respective offer is cancelled, all of the offered items will be returned to the inventory.

Caller	**Client**
Request example	`{ "method": "canceloffer", "token": "47af9eb308ce2fc95bcc088add67b79e", "offer_token": "0f5bb63592f716c05511cec90e171a89" }`
OK Response example	`{ "status": "ok" }`

Fail Response example (already finished / not exist)	<pre>{
	  “status”: “fail”,
	  "description": "FAILURE MESSAGE HERE"
	}</pre>

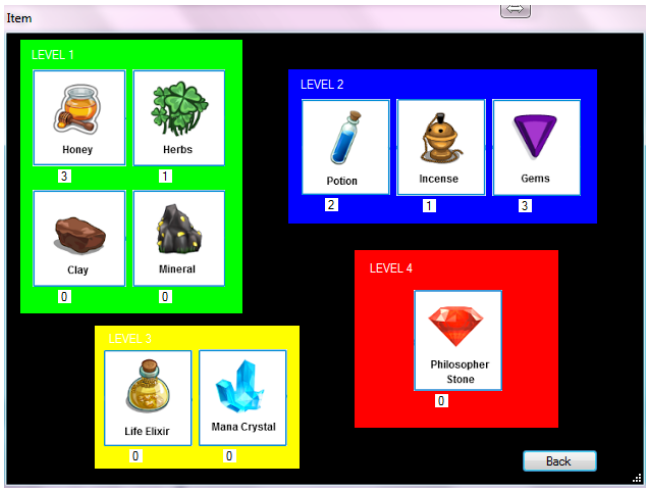
Error Response example	<pre>{
	  “status”: “error”
	}</pre>

3. Client-side

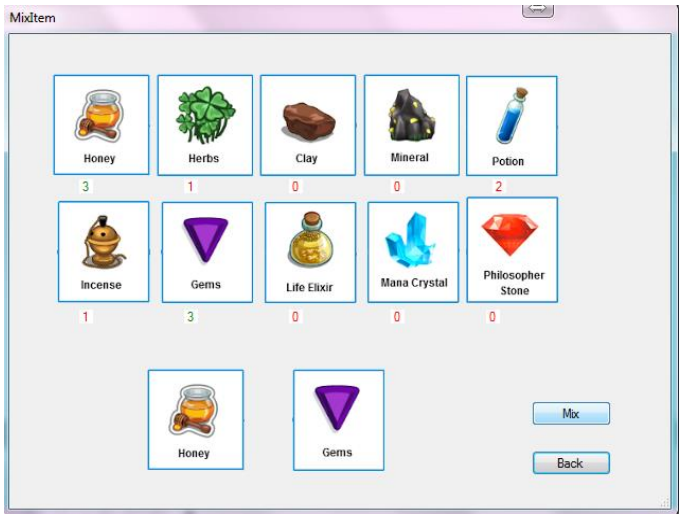
Client application **must be implemented** by all teams. The sample client (for Windows) can be downloaded at https://dl.dropboxusercontent.com/u/58181220/LoliHorizon.rar. For clarity, we have also provided several screenshots in this document.

Description: This screenshot is taken from **Signup** & **Login** page. Users can specify server's IP address (top left), username, and password within this form.

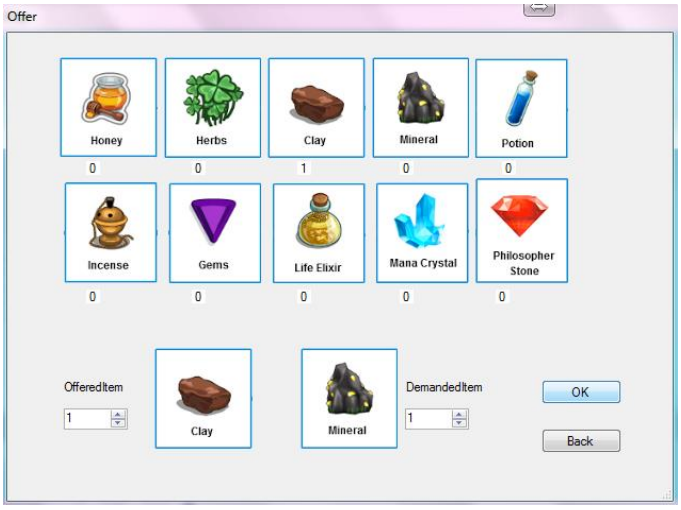
Description: This screenshot is taken from **Map** page. Users may call **Move** and **Field** function within this page. *Time* > *Current Time* if and only if the character moves from one point to the other point. **Field** function could be called once in the specified position if *Time* <= *Current Time*.



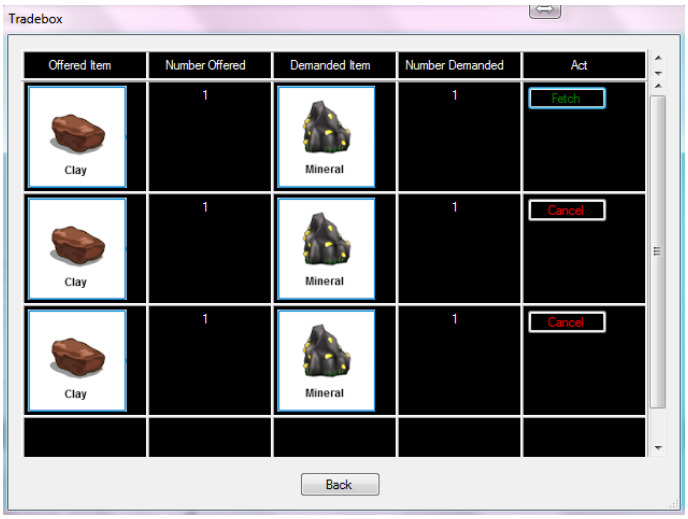
Description: This screenshot is taken from **Inventory** page.



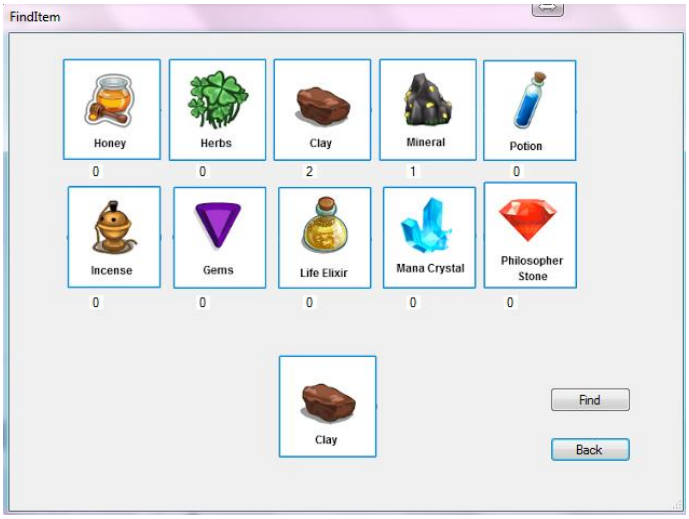
Description: This screenshot is taken from **MixItem** page.



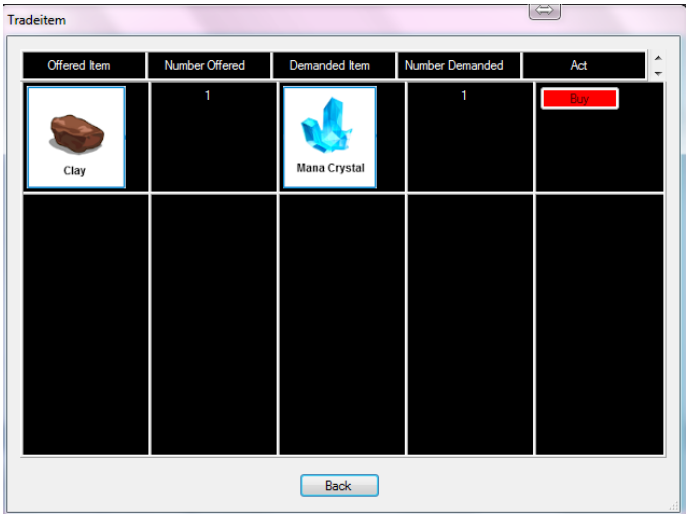
Description: This screenshot is taken from **Offer** page.



Description: This screenshot is taken from **Tradebox** page. Users may call **FetchItem** and **CancelOffer** function within this page.



Description: This screenshot is taken from **SendFind** page.



Description: This screenshot is taken from **SendAccept** page.

Appendix B. Additional Attachments

map.json

```
{
     "name" : "Bandar Behari",
     "width" : 4,
     "height" : 4,
     "map" :
     [
          ["R13", "R13", "R13", "R13"],
          ["R13", "R13", "R14", "R14"],
          ["R13", "R14", "R14", "R14"],
          ["R13", "R14", "R14", "R14"]
     ]
}
```