


ADVANCED RPG Maker

Tutorials, News and Reviews

Issue #7 Dec 2017



How to Make an MMORPG

Special Issue

Welcome

Hello, and welcome to what is an Advanced RPG Maker Special! It's been a long while since we've had an issue. This one is a sixteen page special feature entitled **How to Make an MMORPG**. Rather than create a plugin, which I do not think would be a good idea for various reasons (mainly that to create an MMORPG you really need to both know what you are doing and be willing to put a considerable amount of time and effort in), I am instead going to talk about the various systems I created for my MMORPG, *Afar*, and which I will be recreating and fine tuning for a future RPG Maker MV project.

ADVANCED **RPG Maker**



ARPGM is an online magazine, available at arpgmaker.com and produced for and by HBGames.org. It isn't created anywhere near as regularly as I'd like!

Thanks for reading, **Amy**

Featured graphics on this and the cover page: **Time Fantasy**, which is currently under offer as a multipack, and which I shall be using in a future MMORPG!

HOW TO MAKE AN MMORPG

The first thing I need to make clear before we start is that making an MMORPG is a very labour intensive process, and a continuous one with all of the updates and maintenance you'll have to provide to make it really work. Once we're past that reality we can start.

What is an MMORPG?

An ORPG is a role playing game played over the Internet. The second M tells us that it's multiplayer, and the first M tells us the scale - a Massively Multiplayer ORPG tells us that a theoretically unlimited number of people can join in at once. This is in reference to smaller games and MUDs of old where a set number of players join a dungeon.

To create such a system in RPG Maker then we need access to the Internet. Thankfully RPG Maker MV makes this very simple - merely hosting your game on the net is enough for this. Then we need a way of storing player accounts and interacting them with one another.

There are a few ways this can be done. In this guide I am going to be discussing my version, which is done via AJAX. I'll briefly discuss the benefits of this on the next page, but this may not (is not, in fact) the best way to go about it, it's just the lowest maintenance and thus the most realistic for a small team or sole developer. NodeJS is the alternative, and much preferred, but you need to dedicate a lot of time to managing this!

Parts List

Here's what you'll need as a bare minimum:

- RPG Maker MV
- A text editor such as Scite or Notepad 2
- A web server that supports:
 - A database software (I suggest MySQLi)
 - A programming language (I suggest PHP)
- Access to two devices simultaneously for testing

The latter isn't absolutely vital but you'll find much frustration if you don't. Luckily with RPG Maker MV we are able to run games on a phone or tablet so this is much easier.

My System

I have built up my system over the past thirteen years of MMORPG production. It's built on years of experience trying to get RPG Maker to spurn out online games and isn't necessarily the best way to go about anything. However, it works, is proven and tested. Over the next few pages I am going to describe the plugins I have used or created myself and how I have done so.

Caveats

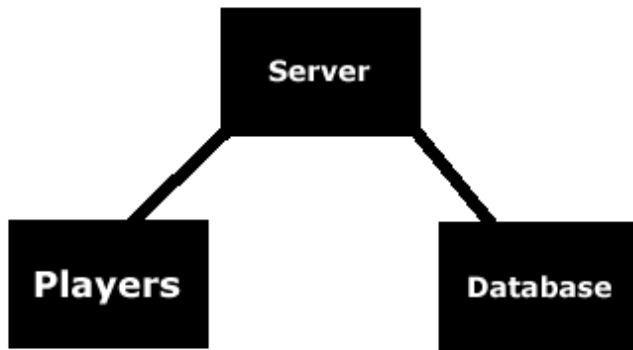
I am intentionally not providing the plugins I have made because I need you to have enough knowledge to produce your MMORPG. I don't want you to leap into this unprepared.

REQUEST/REPLY SYSTEM

The basic principle of my system is one of request and reply. This distinguishes it from an active system where the server sends and receives messages from players on the fly. Basically in my system the user sends a request to the game server and gets a reply back. The game then deals with this and sends the next request as and when needed.

This makes it not particularly prime for active Player versus Player combat and mass user events, although as I'll get to these can still be done! You just won't be able to create the kind of pinpoint accuracy you need for first person shooters or other such games.

The Basic System



Instead of a HTML file we have a PHP file which generates the required HTML. This is simpler than it sounds (see over). Our players can never contact the database itself for security reasons; instead the player contacts the server, and the server handles any database requests. Never, ever give your players direct access to any usernames or passwords for your system - this includes putting them in encrypted scripts. The player only ever contacts the server, not other players and not the database.

The database can be on a different system to your game server. For scalability you could also outsource your image and other file hosting and just have your game server isolated.

Open your index.html file in your text editor of choice. Resave it as **index.php** and remove game.html from your server.

My additions are shown in blue here.

```

<?php
    // PHP code goes here
?>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="apple-mobile-web-ap
        <meta name="apple-mobile-web-ap
content="black-translucent">
        <meta name="viewport" content="
        <link rel="icon" href="icon/ico
        <link rel="apple-touch-icon" hr
        <link rel="stylesheet" type="te
        <title>TEST PROJECT</title>
    </head>
    <body style="background-color: black"
        <script type="text/javascript"
        <script type="text/javascript"
src="js/libs/pixi-tilemap.js"></script>
        <script type="text/javascript"
src="js/libs/pixi-picture.js"></script>
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
src="js/libs/iphone-inline-video.browse
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
        <script type="text/javascript"
    </body>
</html>
  
```

Upload and test. There should be no difference to your original game; difference is your file is now actually generated by PHP and not a HTML file at all. If set up correctly, your player will not be able to see any PHP code you provide - ever. This is secure and you are now able to use this to connect to your database. Just make sure your server supports PHP and has it installed! You can use other programming languages if you wish.

DATABASE CONNECTION

In theory, your player will never see your PHP code. It's still bad practice for us to connect to the database in this public file however. For absolute security we'll create another PHP file and place this a level up on our server, above PUBLIC_HTML, where the player can't access (but the server can).

If you need to learn how to use databases I recommend w3schools who have an excellent tutorial.

We include this PHP file in our main PHP file.

AJAX

I have created a basic AJAX plugin which you can find on www.aRPGMaker.com under **Plugins**. This is a very basic plugin however and doubtless somebody with more experience could tone it and streamline it. Using my plugin we can talk to our server using the function:

```
ajaxRequest ( request , bool );
```

Our request takes the form of HTTP GET variables, for example:

```
a=1&b=2&c=3
```

These variables are interpreted by our server and used to generate a response.

Our boolean decides if the game should wait for a response or continue as if nothing happened and work later.

We create a third PHP file which I call the **action.php** file because this is where all the action happens! We need to get our GET variables and as I use PHPBB I have use of the **request_var** function.

Typically my action file looks something like this (simplified):

```
<?php
$return '';

switch ( request_var ( 'a', 'none' ) ) {
    case '1':
        // action 1
        break;
    case '2':
        // action 2
        break;
    case '3':
        // action 3
        break;
    default:
        // error clause
}

echo $return;

?>
```

Using the variable **a** I choose which action I want to perform with my data, and using the **\$return** variable I populate a reply to send back to the player. This reply could take the form:

```
1|2|167|4
```

Which I then split into an array of individual variables and play around with in the client.

STOP!

Any GET variables must be sanitised before using them! PHPBB's request_var does this mostly for me, but this is not always the case and you should never, ever trust user inputted variables. The server will perform using whatever the player has just injected, and GET injection is the simplest method of SQL injection and a **major security issue**. Please read up on SQL injection, and please sanitize your variables. I'll discuss this in the **security** section.

BASIC ACTIONS

Now that we have the means we can start creating some actions. How about getting the server's local time?

In Our Game

Create an event and make it perform the script command:

```
time = ajaxRequest('a=1');
```

I call my action variable "a" for two reasons: firstly it is almost security by obfuscation: the player doesn't need to be able to read what is going on. Secondly it saves bandwidth; we don't need a longer variable name!

So here I am calling action 1. I don't send any other variables at the moment.

In Our actions.php File

```
<?php
$return = '';
switch ( request_var ( 'a', 'none' ) ) {
    case '1':
        // Request Server Time

        $return = time();

    default:
        // error clause
}
echo $return;
?>
```

We now have a **time** variable in our game that we can play around with - in this case a TimeStamp - to tell us the time on the server!

What can we use this for?

This basic send-receive function is the basis of most of my online system.

We can use this to:

- Update a player variable
- Request another player's information
- Retrieve from a centrally stored database
- Let the server know we're still alive

Contacting to and from our database we can establish a list of players and work with this to create messaging, trading and other multiplayer systems. I'll go into more detail in a second.

At this point it's important to highlight some limitations.

No local playability

No modern browser or operating system will let a HTML file be ran locally with JavaScript as that would be a major security concern. That means our game can now only be played in a web browser and with a web connection of course.

Don't spam the server

You'll want to send as few requests as possible, otherwise you'll overload the server. Work out what you actually need and how little you can get away with.

QUICK SUMMARY

Set up a PHP file for your game index.

Remove your index.html file.

Set up an actions.php file.

Set up a MySQLi database.

Set up a database connection PHP file.

Use a switch statement to perform actions.

Don't Reinvent the Wheel...

It's cheating time.

What's the difference between an MMORPG and a web forum? Does that seem a daft question?

I put it to you that a web forum is just an MMORPG without the graphics. Think about the features that make an MMORPG tick. A list of players, the ability to communicate with one another, send private messages, acruce variables, poke each other, move around a world, even fight with each other.

All are present in a web forum. In fact, many forums have customisable sprites, and some even have RPG style battle systems!

Integrate a web forum with RPG Maker MV and we instantly get a secure and tested login and register system amongst other things.

Personally I use PHPBB but there is no reason Xenforo, vBulletin, Invision or other fora cannot work just as well.

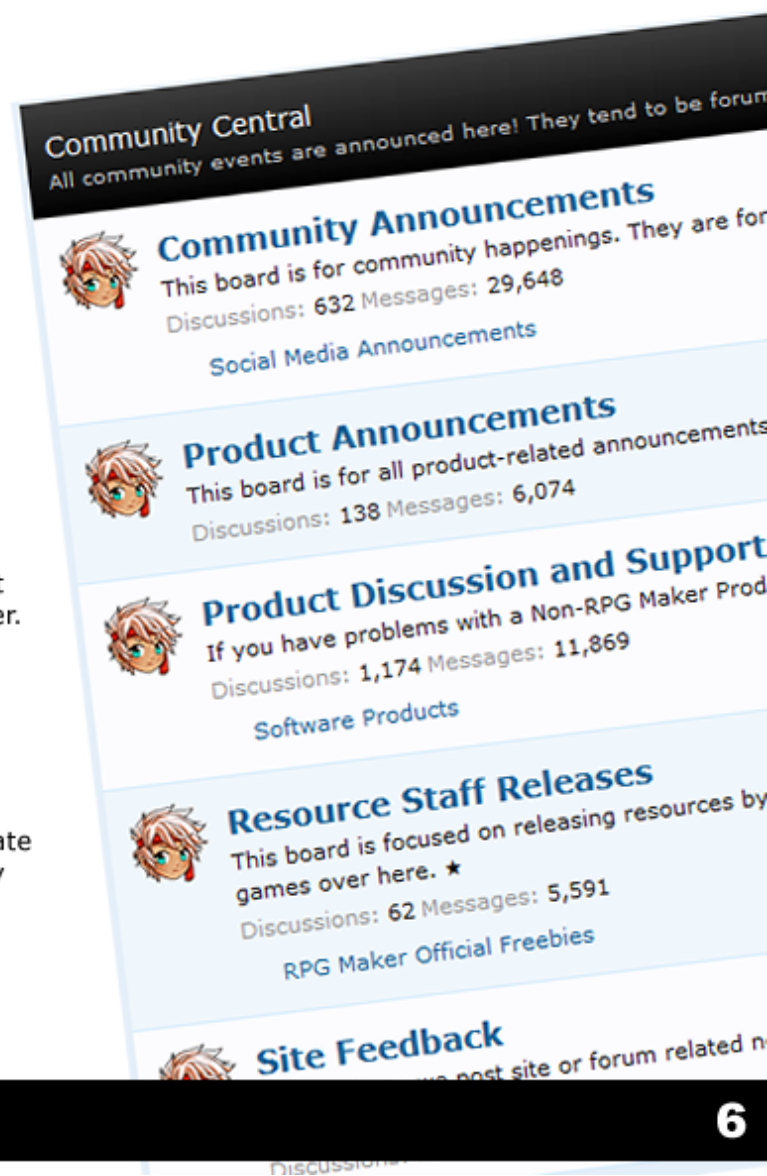
In order to set up such a system we need to understand that our RPG Maker game is just a web page like any other. The simplest way to integrate a forum is just to overlay it above our game. Using jquery we can pop up and hide the forum at will, and with great skinning knowledge various chunks of the forum can be shown to the player seperately for different aspects of our MMORPG game. For example, we could simplify the member list to create a list of online players! As long as our forum and PHP file are stored on the same website, under the same domain, we can use iFrames to create page-in-a-page aspects and with the right skinning nobody will be able to tell that anything magical is happening. The main key here is that iFrames can have a transparent background.

Remove Functionality

It may sound strange but the best way to ensure a coherent and logical system here is to remove any unnecessary clutter. Disabling right click menus, highlighting, post counts and other board features goes a long way!

Communication

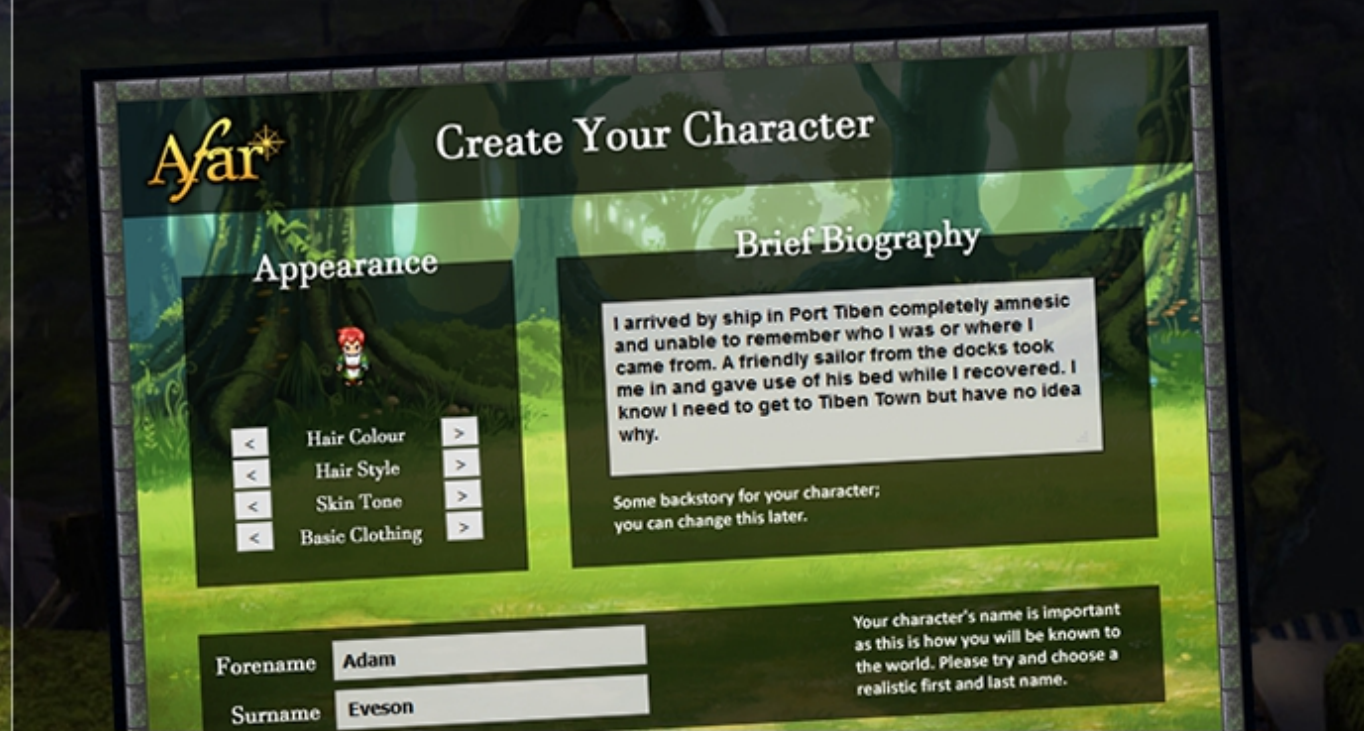
As long as your forum and PHP file are kept on the same domain most modern browsers will let you communicate between the two using the parent keyword. So, immediately you can use variables from one in the other to ensure a coherent experience.



CREATING MMORPG FEATURES

This next section of my guide is far more conceptual. The reason for this is that I am not going to make your game for you - because then it wouldn't be any different from any other out there! Instead I am going to discuss the systems I have used, how I have used them, and how you can apply them to your own game. Now that we have our basic MMORPG set up we can do a lot of powerful stuff with our online players, and integrate with various existing RPG Maker MV plugins.

I have used many of these in my game **Afar** (currently offline) and am refining them and recreating them for a future project as we speak. As I learn how to use things better and how my system can be optimised I will release some updates and further information, but not full plugins, as it's up to you to integrate this with your own game to work best with your own systems.



REAL WORLD EXAMPLE

Guide Window

This is an iFrame that pops up over the game window with a home-made wiki, but you could easily use MediaWiki to create an actual encyclopedia.

Online Players List

Showing who is logged in at the moment. Links show you their character profile and messaging.

In-Game Forums

An iFrame that presents a very slimmed down forum linked of course to in-game accounts.



IRC Chat

I successfully integrated KiwiIRC and will talk about this later.

Global Inventory

The player's inventory stored in the global database.

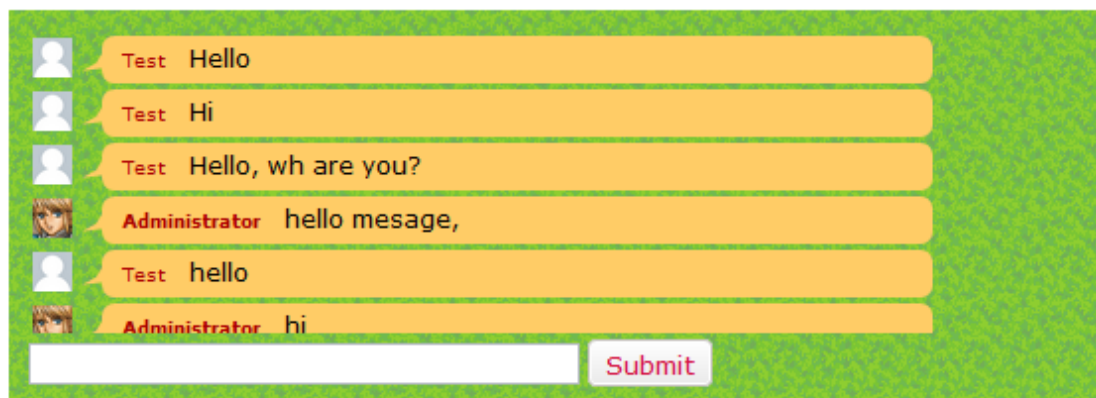
CHAT

Perhaps the most important system in an MMORPG is the ability to talk to other players in real time.

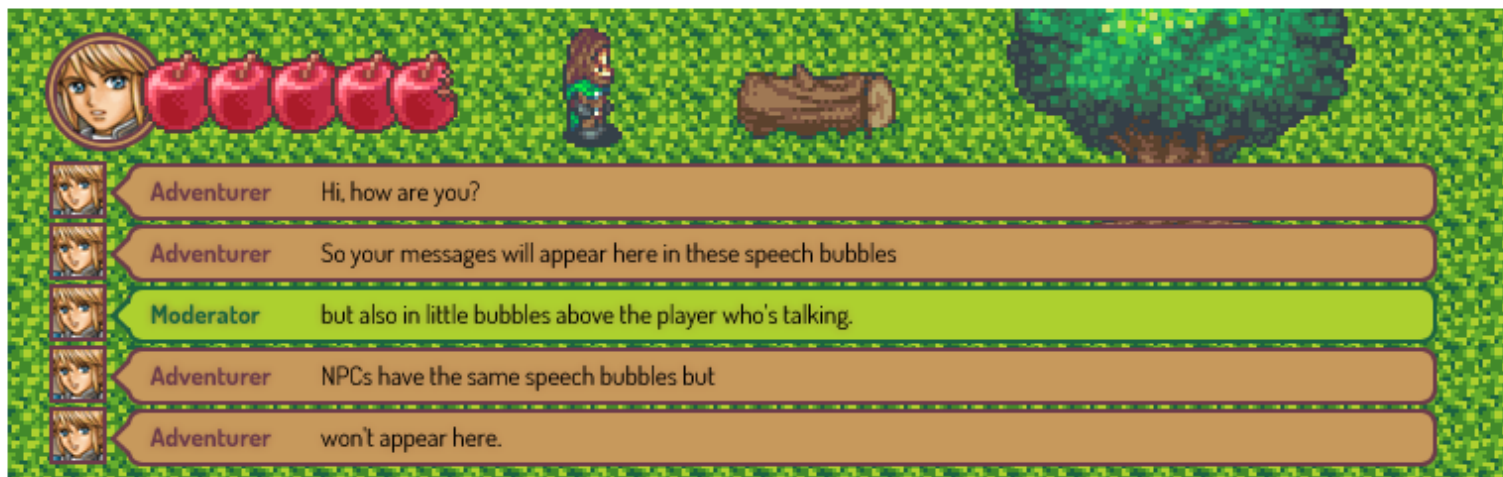
I successfully integrated KiwiIRC in Afar providing a web standard chat system. I did this in an iFrame. However, I have since moved on to other methods.

Currently I use Ajax Chat, a phpBB plugin, for my chat room. Ajax is slow for this however and not the best method for this quick, on the fly communication.

Ideally in my opinion chat should be layered over the game screen rather than rendered in-game. The reason for this is that we already have standard DOM elements that players are used to and know how they work. Text boxes and buttons in games such as Guildwars or RuneScape often have poor functionality and feel clunky and home made. For things such as chat where we want to be able to type unimpeded by bugs we really should stick to pre-made, standard web elements.



Above is an example of a working chat system, this one is Ajax IRC, a plugin for phpBB. It seems to work well! We are of course free to skin it however we like as it is open source. In the case above, clearly a lot of work is still needed! Here's my mockup and how it will look, honest.



LOGIN AND REGISTER

These are provided by our forum system, however integrating them with the look of our game is a key part of coherency and making it look like a proper system and not a cobbled together mess.

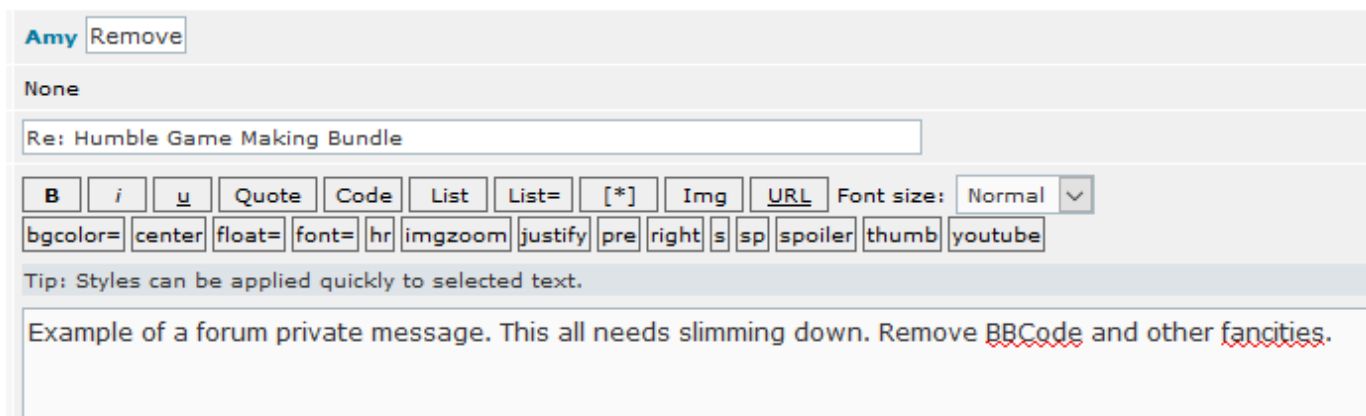
Keep fonts and graphics the same, and ideally make the login page look as if it's inside the game itself. Perhaps even load part of the game first, with some animation, while showing the login boxes.

ONLINE USER LIST

We need to establish what it means to be online. If being online is currently clicking something, then nobody is counted for very long. We need some kind of timer. Store the user's last action time by registering in action.php what time they did something. Store this in the database and reference this in your online users list. Select from the database a list of users where their time is less than a defined amount.

PRIVATE MESSAGING

Using an overlay we can make it appear that our private messages are in an in-game message box. This is all about theming and positioning and it's a lot of work to get right but worth it. PMs are handled by and already created by the forum system, but skinning them to slim them down to our purposes will take a lot of work.



SYSTEM MESSAGES

Most forums can globally send to all for this. Alternatively, use chat and have the chat messages themed differently if they are posted by a particular account.

CHAT ABOVE PLAYERS' HEADS

Plugin time! Using **Nelderson's** plugin **Text Pop Over Event** we are able to associate chat messages with sprites in game. The key is to work out what is updated in our Ajax Chat window and use that to call the text pop function. But how do we create the player sprites themselves?

LIVE PLAYERS

The best way I have found to show players live in game is this:

Store an X, Y, map and direction in the database. Also store their looks. Using our action.php file ask for these variables for each player online in the map and use these to draw a sprite.

For looks, look into using a paperdoll graphics system.

We need to use **Galv's Event Spawner** for this. We create an event that is a dummy for any player. We can give it some attributes and actions, such as what to do when the player interacts (show some messages, etc).

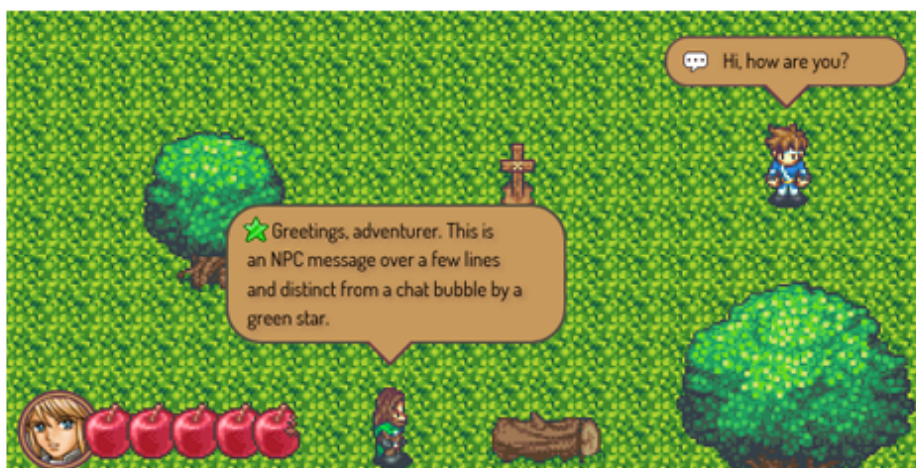
For each online player who is in our map we spawn our event. Then, by using:

\$gameMap._lastSpawnEventId

We can obtain the ID of the spawned event. Create an array that associates these events with players and their graphics.

Periodically, reobtain the list of players. Check what has changed, and if anything has changed use move routes and pathfinding to move the player's sprite to their new position.

In order to show what the player is doing we can store a variable for this and show a balloon or other graphic, even animate the player, based on their action. For example we could show them mining or fishing.



TRADING

In order to establish trading we first need globally stored items. We do this by having two database tables: **items** and **items_owned**.

These are associated with our players table; in my case this is my phpbbs **users** table.

Each **user** has many **items**. These are stored in **items_owned**.

For trading we then need to create a PHP system using our actions file. While I have yet to create this system, I know how to do it; I'll admit I am lazy here and am putting it off. What I *have* successfully implemented is the following:

GLOBAL ITEM EXCHANGE

This is key to a game economy.



But I'm not going to tell you how I did that!

Such a system is so big it needs to be unique and tailored to your game. That way you can form your own economy and have it work based on your players and what they are doing.

The basic principle is the same as any other, using actions.php and my database I associated some stored items with players and monetary values. The player sets their maximum buy price or minimum sale price and the game matches them up and performs the trade automatically. In short, it is a ripoff of the Grand Exchange in RuneScape (I did initially call mine the Tat Exchange!).



With a bit of work that same system could be recreated like for like (but it's more fun to create your own!)

METRICS

Once your gold, items and other variables are stored in your central database you can draw from them from all players to generate statistics and other information.

Use this on the fly and you can create a dynamic world where merchant and vendor prices are set by market trends. Create a real life economy in your game based on supply and demand, and create quests based on real items players need.

This all depends on the resource gathering implemented in your game itself.

Mining, smithing, and merchant skills are not something that is unique or limited to MMORPGs - that's all up to you.

PLAYER PROFILES

In its most basic form, this is just a forum user profile! Scale it down and only show key variables (I use Custom Profile Fields for a lot of variables as they are quite dynamic and easy to administrate).

SIGNATURES AND USER BARS

Using image generation tools for PHP we can create dynamic signatures. However, do **not** draw from the database for these or you will kill your game's performance!

What I do is generate the images on the server at key points.



GLOBAL EVENTS AND NODES

In order for this to work we need globally stored variables and switches (an important part of our game anyway and one I'll let you discover! Oops!)

We simply replace any switch calls and conditional branches with plugin calls that invoke our actions.php page again. By this method we can easily create world events as big as those seen in Guild Wars 2's living world!

Shops whose prices depend on what is happening elsewhere in the world. Bosses and dungeons that require multiplayer elements to complete. Shared enemies and nodes.



SECURITY CONCERNS

It's time to discuss the main limitations of our RPG maker system.

As most things are stored client side, **it is hackable**. While the player has no access to your server itself, any data sent to the server **can and will be manipulated**.

You need to decide what data is being sent, how well you can reasonably trust it, and the consequences of it being edited by the player. Because they will. They decided to the minute you said you'd make an MMORPG.

So you don't want to send, say, "gain 500,000 exp" to the server. But you could send "player attempted attack a". The server then decides if the player owned that attack, what they scored, performs the calculations and adds the exp. But guess what? Now the player is spamming attacks with a macro. So add a timer, and work out how to stop macros. Good luck, because RuneScape hasn't after 18 years.

Passwords

You have a duty under the Data Protection Act to store peoples information safely and securely. It's your prerogative to learn about these laws.

Never store passwords as plain text. **Never** allow players access to this database. **Never** allow unsanitised data to be used in SQL statements and **never** directly print from this database.

SQL Injection

All variables **have** to be sanitised. This means removing any special characters so that your system reads them as the correct type of variable. As a quick example, imagine your player named themselves:

Edgar Alan Poe'; DROP TABLE players;

What would happen? Surely a computer wouldn't be so daft as to delete every player in the game just because somebody typed some characters. Well, yes, it would, because computers are stupid. They will do whatever you tell them to. And if you don't sanitise your data, it is your players telling them to do anything. I can't stress this enough! It is vital that you research SQL injection.

Never let players store HTML

Some forum systems let you insert HTML into posts. Don't do this. There is never a need. Don't let players post iFrames, Flash, or any other media in the in-game forums.

Other Concerns

You would be a fool to take this or anything you read online as a comprehensive and final list. Web security is an ever growing issue and one that you **need** to keep researching regularly to keep up to date. Keep your systems updated and don't let assumed experience be your downfall.

PLAYER VERSUS PLAYER

The reason I left this until after our security talk is that I need you to understand the limitations and issues with letting players attack one another. Certainly you would not want to base your whole game on this in the way a first person shooter may. Hackers would run rampant. Here are a few options:

PvP Zones

RuneScape handled this well by having half of the world (at the time) a PvP enabled zone. This way even if it does go bottoms up you're only affecting a small portion of the map.



Non-Harmful PvP

You could have dud weapons equivalent to throwing a sheep at somebody on Facebook. The player sees that you're poking them, but relatively speaking they aren't too put out by it.

Sim PvP

To me this is the most interesting. Use your players database to generate enemies which use the look and name of the player. Go further and record players' play styles and what spells they regularly use, etc. Use this to create a realistic AI and have simulated player-versus-player combat.

Adventure Quest once put this system to good use.

Depending on the battle system used, some good tactical play could be put into effect. Players can then be notified when they've been defeated or won, and high scores still created, but it wouldn't matter *as much* if somebody hacked the game.



Amy's Kingdom

Very briefly I have mentioned that I am working on a new project. While *Afar* was successful, in that it worked, the scale and demands of the project are proof that an MMORPG is a huge undertaking. So, due to my time constraints, I am starting anew (sorry). Amy's Kingdom is going to be a much smaller affair, initially, and it's a concept I am still working on. Many systems have been created, however, including my live players including a paperdoll graphics system.

You can follow my progress at HBGames.org / aRPGMaker.com.

BACK ISSUES

The previous six issues of this magazine, and all 25 issues of HBGames.org the eZine which preceded it, are available at www.aRPGMaker.com.



Thank you for reading!

I was a bit unsure of how to make this guide. Because of how much progress has stalled on *Afar*, I wanted to try and pass my knowledge on, but I know my own work is unusable by anybody else as it's all bits and pieces here and there and based on years of dodgy code.

Thus I figured it would be far more useful to attempt to explain what I did and how I did it, so that somebody else can continue my work.

Please feel free to ask me anything, I will do my best to respond.

Thanks, **Amy**