MineBike Game Documentation

Andrew Weller & [your name here]

July 2019

Contents

0 This Manual

Hello anonymous developer! Welcome to the MineBike project. If this is not your desired destination you may leave this document now.

If you are not already aware of what the project is, mineBike is a minecraft mod designed as an alternative to traditional physical therapy for quarantined hospital patients who are unable to go outside to excercise.

Please refer to the table of contents if you wish to find what you are looking for quickly. Otherwise reading this manual in chronological order should get you up to speed with the game with zero prior knowledge.

0.1 What is this manual?

This manual is intended to inform the user about the mineBike GAME portion of the code. This manual does not relate to the middleware or the database portion. If you are looking for documentation on that, you should search elsewhere.

HOWEVER, there will be a section explaining (not as well as the rest of the manual) where to begin searching to find out about how the game receives information from the bike's raspberry pi.

0.2 What will this manual teach you?

If you want to add **CONTENT** to the MineBike game, this is the right place to be. This manual will go over all the tools you need to add your own minigames/quests to this game.

If you have other goals in mind this manual will still be valuable as it will show you the important parts of the code so that you can search for pieces relevant to your goal.

1 Setup

Setting the game up on your computer is *relatively* painless. If you have set up minecraft forge before this is a bonus, because it is pretty much the exact same process with a slight twist for this mod's specific quirks.

Step 1: Install Java JDK 8.

Minecraft was written over 10 years ago and still depends on Java 8 to build. Go to this download page and select the latest Java 8 JDK for your operating system.

Step 2: Install Minecraft Forge

Go to the minecraft forge downloads page for minecraft 1.7.10 and download the Installer if you are on mac/linux or the windows installer if you are on windows.

Step 3: Clone Repository

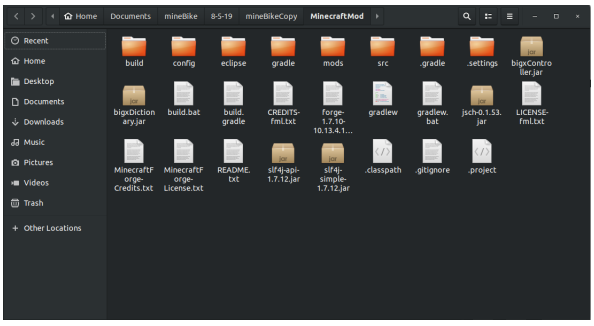
```
$ git clone https://github.com/andrewwellercs/mineBikeCopy
```

Step 4: Unzip Forge Source

Unzip "MinecraftMod/Forge Source/forge-1.7.10-10.13.4.1492-1.7.10-src.zip" into "MinecraftMod/".

Be sure to overwrite any files.

Your MinecraftMod/ directory should look like this now:



Step 5: Run gradle scripts

After unzipping all that stuff there should be a script called gradlew inside of the MinecraftMod/ directory. Run the script using these commands in either your terminal or command prompt.

Mac/Linux (terminal):

```
$ ./gradlew setupDecompWorkspace --refresh-dependencies
```

```
$ ./gradlew eclipse
```

Windows (command prompt):

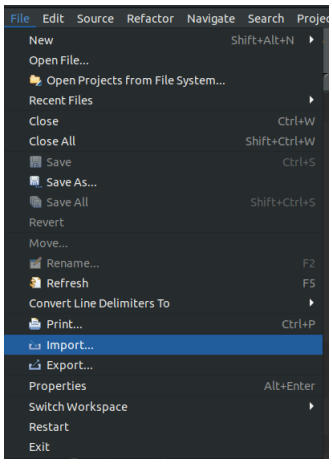
```
$ gradlew.bat setupDecompWorkspace --refresh-dependencies
```

```
$ gradlew.bat eclipse
```

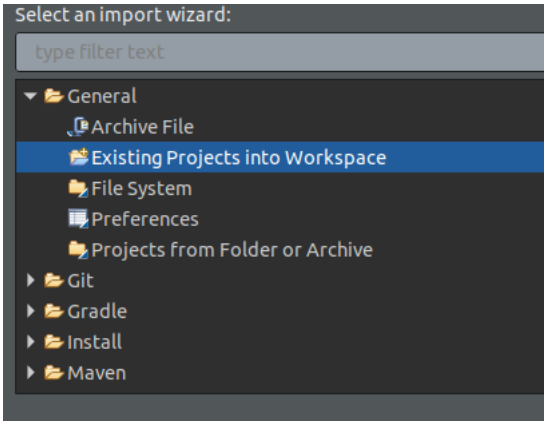
This will create an eclipse workspace in the MinecraftMod/ directory.

Step 6: Import into Eclipse

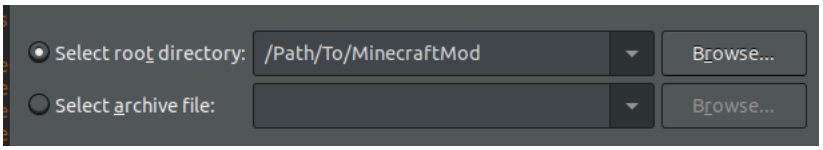
Go to File → Import in eclipse.



Then go to General → Existing Projects into Workspace (remember the gradlew scripts created the workspace).



Next for the root directory point eclipse to the location that the Minecraft-Mod/ folder in your copy of the repository.

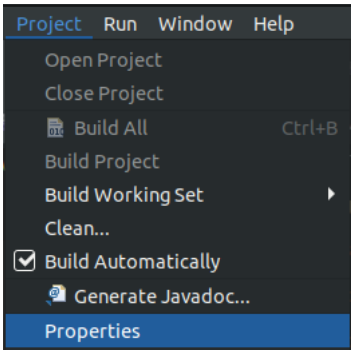


Step 7: Final setup instructions for eclipse.

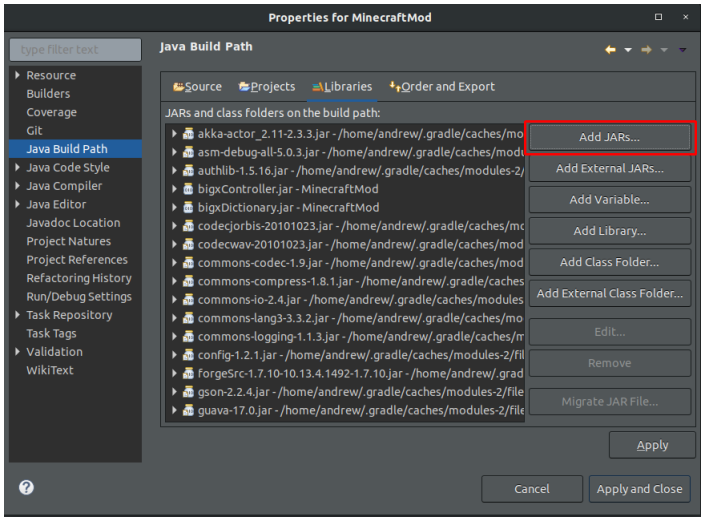
Now there are a couple things you need to do before eclipse will be able to launch the project.

First, you have to link some jars that are included with the project.

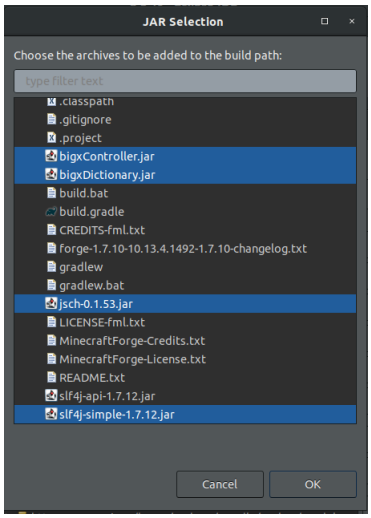
Go to Project → Properties in eclipse.



Then go to Java Build Path → Libraries tab → Add Jars.



Select bigxController.jar, bigxDictionary.jar, jsch-0.1.53.jar, and slf4j-simple-1.7.12.jar. (slf4j-api-1.7.12.jar is not required, I don't know why, please edit this document if you figure out why)



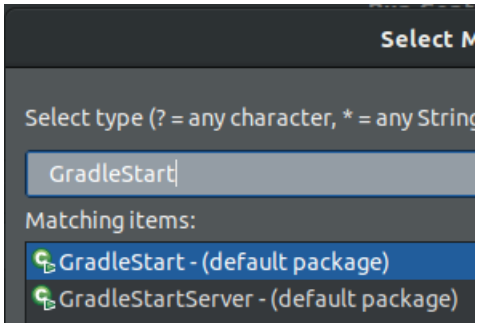
Step 8: Launching the Game.

You're almost done. You might have noticed that if you try to launch the project that it doesn't work. You need to point eclipse towards the main class so that it can start the game properly.

You can do this by going to Run → Run Configurations, then double click Java Application.

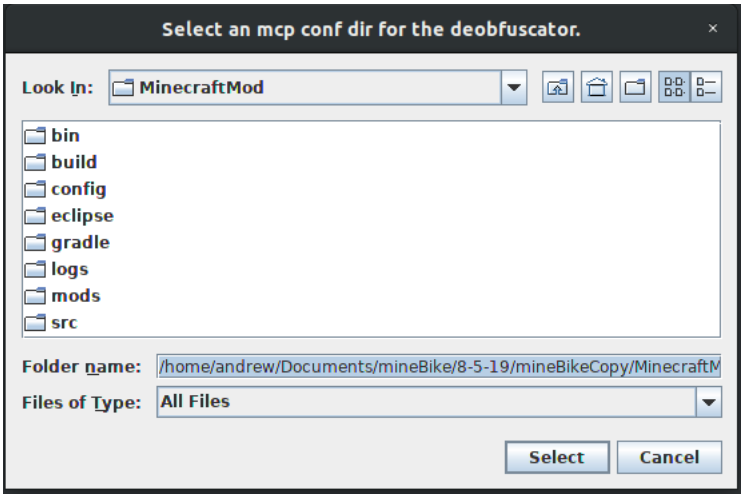
This will create a new run configuration.

Now click search in the main class box, and then find GradleStart.



Now when you run it, it should work.

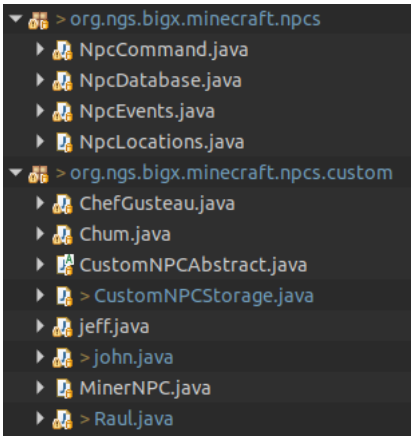
The first time you run the application, you will receive a pop-up that looks like this.



2 Codebase Overview

The Codebase looks very big, but there are only a few very important parts to it. Most of the code is either not necessary for manipulating the game or is deprecated. Let's take a look at some important packages.

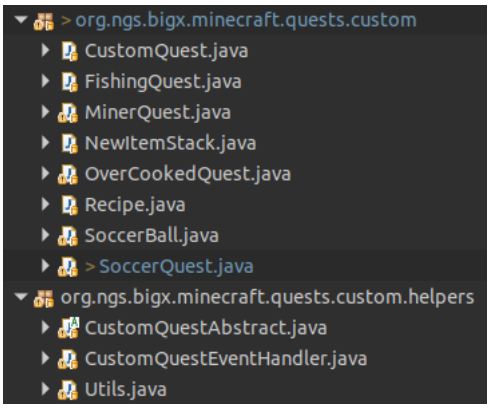
2.1 org.ngs.bigx.minecraft.npcs



This package is largely responsible for the creation of NPCs in the overworld map. Creating npcs in quests is another story and you can learn about it in the npcs section.

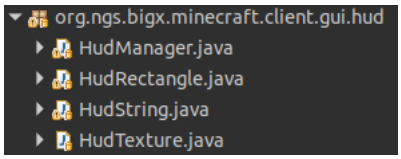
Noteable classes in here include **CustomNpcAbstract** which combined with a simple addition to **CustomNpcStorage** is a one-stop shop to adding simple npcs into the game. More on that in the npcs section.

2.2 org.ngs.bigx.minecraft.quests(.custom)



This package is responsible for the "quests" or "minigames" that the game has, excluding the original ChasingQuest which was in the game. The questing system is a modular way to build minigames.

2.3 org.ngs.bigx.minecraft.client.gui.hud



This package is very short and specialized. The tools in here can be used to add HUD elements to the game with no knowledge of OpenGL.

More on the usage of this package can be found in the Hud section.

3 Npcs

You might wonder why npcs are first in this guide and not quests. The answer is simple. Npcs are easier to add to the game. So to get your feet wet it is recommended to do this first. Also every quest usually has an npc attached to it. That is the pattern for this game. You start quests by talking to npcs. So once you have created your own npc you can then attach a quest to them and start testing that in the next section.

3.1 Simple Npcs (overworld)

Adding Npcs to the overworld is easy.

Figure 1: A custom NPC made using the CustomNpcAbstract class

You can add your own npc by creating a new class the extends the class **Cus-**

tomNpcAbstract. Located in the package **org.ngs.bigx.minecraft.npcs.custom**.

```
1  package org.ngs.bigx.minecraft.npcs.custom;
2
3  import org.ngs.bigx.minecraft.npcs.NpcDatabase;
8
9  public abstract class CustomNPCAbstract
10 {
11     //Three properties of an NPC
12     protected String name;
13     protected Vec3 location;
14     protected String texture;
15
16     //important that this method is called after the instantiation of the object
17     public void register()
18     {
19         NpcDatabase.registerNPC(name, location);
20     }
21
22     public abstract void onInteraction(EntityPlayer player, EntityInteractEvent event);
23
24     public Vec3 getLocation() {return location;}
25     public String getName() {return name;}
26     public String getTexture() {return texture;}
27 }
```

Figure 2: The CustomNpcAbstract class

There are three protected variables inside this class that should be set in the constructor in order for this class to work as intended.

`protected String name;`

This variable will change the display name of the npc in game. It is also used as an identifier in other pieces of the code so it should be **unique** and not overlap with any other npcs that exist either in the **CustomNPCStorage** class or the **NpcDatabase** class, located in org.ngs.bigx.minecraft.npcs.custom and org.ngs.bigx.minecraft.npcs packages respectively.

`protected Vec3 location;`

This variable will change the location that the npc spawns at in the overworld. It should be created with the Vec3.createVectorHelper method.

`protected String texture;`

This variable will change the texture that the npc has in the game. It should be a file location of an image. It technically shouldn't matter where the image is located but sometimes it doesn't work if the images is in a weird folder. In order to guarantee that it works it should be located somewhere in the src/main/resources/assets/customnpcs/ folders, by convention all the skins for customnpcs have been placed inside of the .../customnpcs/textures/entity/ folders

The variable must be set in a certain way. In order to set the texture to a texture in customnpcs/textures/entity/humanmale/ the texture variable should be set as follows:

```
customnpcs:textures/entity/humanmale/Example_Texture.png
```

You can find new textures online by searching for "minecraft skins" online. Note that you will have to find one that is in the right format for the version of the game this mod is using - 1.7.10.

Here is an example of an Npc made using this formula. You can also view this in the project as Raul.java in org.ngs.bigx.minecraft.npcs.custom

```java
package org.ngs.bigx.minecraft.npcs.custom;

import org.ngs.bigx.minecraft.quests.custom.SoccerQuest;

public class Raul extends CustomNPCAbstract
{
    public static final String NAME = "Raul";
    public static final Vec3 LOCATION = Vec3.createVectorHelper(121, 70, 239);
    public static final String TEXTURE = "customnpcs:textures/entity/humanmale/SoccerSteve.png";

    private CustomQuestAbstract quest;

    //Constructor for Raul
    public Raul()
    {
        name = NAME;
        location = LOCATION;
        texture = TEXTURE;

        quest = new SoccerQuest();

        this.register();
    }

    @Override
    public void onInteraction(EntityPlayer player, EntityInteractEvent event)
    {
        System.out.println("Interacting with raul!");
        quest.start();
    }

}
```

Figure 3: An example of an npc made with the CustomNpcAbstract

3.2 Advanced Npcs (quests/moving)

If you wish to add an Npc that *d*oes *s*omething into the game, there are tools available for that.

There is a mod in the codebase which has been slightly modified in house called CustomNpcs (in the noppes package).

It is hard to outline specifically what to do to create something using these tools since I (the author of this manual) was not in contact with the creators and it is a relatively niche mod with not much help for it online.

The best way to learn how it works would be to look through other sections of code that use it like **SoccerQuest or TRONQuest** from **org.ngs.bigx.minecraft.quests.custom**.

4 Quests

4.1 What is a quest?

A quest in the mineBike game is basically just a class that has access to the Minecraft forge event bus. So if a quest is active you can run certain code when different events happen in the game.

4.2 Making a quest

To make a quest, much like an npc you must create a new class which extends the class **CustomQuestAbstract** (org.ngs.bigx.minecraft.quests.custom.helpers)

Inside this class there are several methods which correspond to different minecraft forge events. All you have to do is override those methods to make your class do stuff. If you look at the code inside of the class **CustomQuestEventHandler** it will become apparent what is going on here.

Let's take a look at the class and what each of the methods does.

```
1  package org.ngs.bigx.minecraft.quests.custom.helpers;
2
3  import org.ngs.bigx.minecraft.gamestate.CustomQuestJson;
17
18 public abstract class CustomQuestAbstract
19 {
20     public static EntityPlayer player = Minecraft.getMinecraft().thePlayer;
21
22     protected int progress;
23     protected String name;
24     protected boolean completed;
25     protected boolean started;
26
27     public enum Difficulty
28     {
29         EASY,
30         MEDIUM,
31         HARD;
32     }
33
34     //
35     public CustomQuestAbstract()
36     {
37
38     }
39
40     public void loadFromJson(CustomQuestJson json)
41     {
42         progress = json.getProgress();
43         name = json.getName();
44         completed = json.getCompleted();
45         started = json.getStarted();
46     }
47
48     //this must be called in constructor or your quest will not work
49     public void register()
```

Figure 4: The CustomQuestAbstract class

Foreword: Most of this information is available online, I would start searching here

4.2.1 onPlayerTickEvent & onWorldTickEvent

These two methods are very similar so they will both be talked about in this section. These methods are fired continuously by the game's event bus. If you've played minecraft before, you might know that minecraft's game engine runs at 20 ticks per second. This does not apply here. Yes the game runs at 20 ticks per second, but that means that *o*ne single object in the game receives an update 20 times a second. This method is a catch all, such that every time a single object is updated, this method is called. There is a a science to the madness but basically you can think of these methods as something that gets executed every frame or just on a loop, however they cannot be reliably used to count time.

The only reason you would use one over the other is that the onWorldTickEvent's parameter (TickEvent.WorldTickEvent) allows you to access the *w*orld object from the game and the onPlayerTickEvent's parameter (TickEvent.PlayerTickEvent) allows you to access the *p*layer object which is being ticked.

Note: When acessing a world from a world tick event, it is useful to make sure that you are accessing one that has both been loaded and that you are accessing the right dimension.

4.2.2 onItemUse

This method is called whenever the player right clicks with an item in their hand that does something on right click. This may come in handy for you.

4.2.3 onItemPickUp

This method is called whenever the player picks up an item off the ground or is given one from a call of the /give command inside the game. You might find this useful to check when a player is picking up gold etc.

Personally a lot of previous developers have used this as a debugging mechanism to end the quest.

4.2.4 onAttackEntityEvent

This method is called whenever an entity is attacked. It grants you access to the entities involved in the transaction.

4.2.5 onEntityInteractEvent

Called whenever the player right clicks on an entity who is interactable (a villager, etc.) Grants access to the parties of the transaction.

4.2.6 onWorldLoadEvent

This one is important. It is called whenever the player loads into a new dimension. So if the player was transported into a new dimension for a quest to start, this method should be used to verify that the dimension has been loaded before trying to modify the world of that dimension. You can see this pattern in a couple of the quests that already exist where the quest sets a worldLoaded flag in this event.

4.2.7 onEntityJoinWorld

Called whenever an entity changes dimensions. Similar to the previous method, but don't use this for that same purpose.

4.2.8 onQuit

Called when the game is exited.

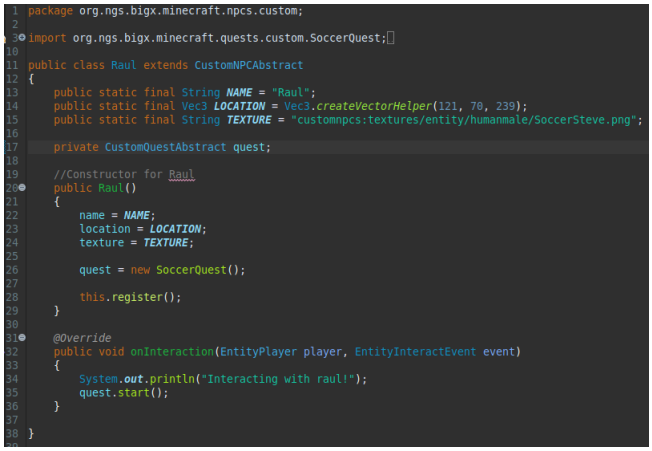
4.3 Making it work

In order for the quest to actually do stuff it must be registered with the CustomQuestEventHandler. You can do this by calling **CustomQuestEventHandler.registerQuest(CustomQuestAbstract quest)** and remove it with **unregisterQuest**.

The event methods of the CustomQuestAbstract class sit dormant until the quest has been registered with the CustomQuestEventHandler. A look at the **CustomQuestEventHandler** class will illuminate how this works. Then they will start being called appropriately. This effectively keeps track of the "active" quest, and you could perhaps have multiple quests going on at the same time. This has not yet been done though in the game.

The CustomQuestEventHandler is just a regular minecraft forge event handler. You can learn more about it here https://mcforge.readthedocs.io/en/latest/events/intro/

The easiest way to go about getting your quest to start normally is to attach your quest to an Npc made with the **CustomNPCAbstract** class. You can see examples of this by looking at npcs that already exist like **Raul**.



```java
package org.ngs.bigx.minecraft.npcs.custom;

import org.ngs.bigx.minecraft.quests.custom.SoccerQuest;

public class Raul extends CustomNPCAbstract
{
    public static final String NAME = "Raul";
    public static final Vec3 LOCATION = Vec3.createVectorHelper(121, 70, 239);
    public static final String TEXTURE = "customnpcs:textures/entity/humanmale/SoccerSteve.png";

    private CustomQuestAbstract quest;

    //Constructor for Raul
    public Raul()
    {
        name = NAME;
        location = LOCATION;
        texture = TEXTURE;

        quest = new SoccerQuest();

        this.register();
    }

    @Override
    public void onInteraction(EntityPlayer player, EntityInteractEvent event)
    {
        System.out.println("Interacting with raul!");
        quest.start();
    }

}
```

Figure 5: Raul, who has a SoccerQuest as a private data member.

When the npc is interacted with, it calls the **start()** method in the CustomQuestAbstract class. By default this method registers the class with the event handler, so that after the quest is started, all of its methods will start being called from the forge event bus.

5 Hud

The Minigames all feature various Hud elements like rectangles, text, and textures. There are different ways to accomplish this and if you want to do it your own way, this is a good starting point.

There are tools made by yours truly that you can use to add Hud to your game without knowledge of OpenGL or the minecraft rendering pipeline.

Enter the Hud Manager.

5.1 The Hud Manager

The Hud Manager, seen in **HudManager.java** in **(org.ngs.bigx.minecraft.client.gui.hud)** is a simple GuiScreen class which has built in tools to draw **three** different hud elements. There are rectangles, text, and textures.

Each of these different elements in the HudManager have three classes that correspond to them, and two methods to register and unregister them respectively.

All of the classes talked about in this section are in **org.ngs.bigx.minecraft.client.gui.hud**.

5.2 HudRectangle

The HudRectangle class (in **org.ngs.bigx.minecraft.client.gui.hud**) can be used to fill a rectangle of a solid color on the screen. Noteable uses of this are in the Fishing minigame (for the gui on the fishing rod) and in the Soccer Quest (for the color behind the scoreboard elements).

The class only has

5.3 CenterX and CenterY

This has its own section because it is kind of confusing.

All of the HudManager helper classes (HudRectangle, HudString, HudTexture) have two data members called CenterX and CenterY. When these variables are true, it will turn the x and y coordinates of the class into *r*elative coordinates with respect to the center of the screen. *Such that the x and y values are now x + (screenwidth / 2) and y + (screenwidth / 2) respectively.

*There is a special case for the HudString. If you have CenterX enabled the text will be centered automatically based off of its width on the screen. So if you want centered text, just leave x = 0 and flip centerX on.

For example, if the screen resolution is 1920x1080, and I wish to place a HudRectangle right below the player's crosshair, I would create a HudRectangle and set x and y to be 0. Then the top left corner of the rectangle would be on the crosshair in the center of the screen.

Or if you want to have a box that is at the top of the screen in the center. You would just enable centerX and have centerY disabled, example x and y coordinates could be like -50, 100.

It is good practice to use these variables so that a changing screen resolution will not screw with your hud.

6 Making the Game Multiplayer

The game currently only works via single player. I (The original writer of this manual) do not have the extensive knowledge of minecraft required to outline this in full, however I can lead the implementer in the right direction.

6.1 Minecraft Server Client Model

Minecraft can technically be a single player game, however it does not work the same way that a traditional single player game works. Minecraft runs an internal server which the player connects to just the same as if they were connecting to an external server.

The implications of this are that when writing a minecraft mod, the author of this mod has access to both the internal server that minecraft runs and the client that it is running.

6.2 In Practice

In practice this means that all game objects accessed within the mod are either on the server or on the client.

Many objects have a way to detect if they exist on the client or on the server.

This is the isRemote() method. It should only be found on objects that are relevant such as Worlds or Entities that have to be updated by the server.

For example a world object which holds a single dimension has this method. In all of the quests in **org.ngs.bigx.minecraft.quests.custom** the onWorldTickEvent method has this line:

```
if (event.world.provider.worldObj.isRemote)
return;
```

If an object is remote, that means it is remote from the server, therefore it exitst on the client.

So objects that are remote should not be accessed unless it is for client specific things like hud if the game is to become multiplayer.

Other than that I would look at some resources for developing multiplayer minecraft forge mods and developing mods for a minecraft forge server.

That's about as far as my knowledge goes for this. Good luck if you decide to tak this path.