The Frankenstein of Hip Gaming

Technical University of Denmark
BSc thesis

Thomas Feld (s103466)

August 1, 2013

Abstract

This report regards the development of a piece of software connecting two popular indie games: Dwarf Fortress and Minecraft. During the report, multiple solutions are explored for some of the challenges involved with this. The overall project is not successful in implementing a fully working program to connect these games fully, but offers some alternative methods to solving some of the problems not fully addressed. A few tests are conducted on the working parts of the software as well.

Contents

1 Introduction

Two of the hottest indie games today are Minecraft[1] and Dwarf Fortress[2].

Minecraft is an open world sandbox style game, played in first person perspective. The world is represented by voxels[3] which are generally agreed to be 1 meter in each dimension. This 3D world is procedurally generated and can therefore in theory extend forever, though a maximum size has been defined, outside which the player cannot venture. This size is so big that it wouldn't be feasible to reach the edge of the world during normal gameplay.

Dwarf Fortress is both roguelike and a city-building type game, with a few different gamemodes. In 'Fortress Mode' the player takes control of a group of seven dwarves with a few starting supplies, and sets out to start a fortress. There are several challenges during this, including fighting off monsters, conquering the terrain, and managing the resources in the fortress.
Another game mode called 'Adventure Mode' is a roguelike adventure game, where the player only has one dwarf. The world itself is represented similarly to Minecraft in a three-dimensional voxel-type setting, though the interface itself displays the world in only a single slice at a time, using ASCII characters as sprites to represent each block or creature.

The goal of this project is to put these two projects together, rendering the world of Dwarf Fortress in 3D within Minecraft, thereby bringing the two communities together. This synthesis will be accomplished by turning a sense-compute-control architecture into a screenscraping-compute-render loop, but doing so in a flexible fashion that naturally evolves with the evolution of both products.

2 Analysis

To combine Minecraft and Dwarf Fortress, two pieces of software are needed: one, which can read the data from Dwarf Fortress, and another, which will be a mod for Minecraft. These two pieces of software will have to be able to communicate. In this project, the connection will be network based, which allows the two games to run on separate machines if so desired.

2.1 MC part

The basics of the Minecraft mod are displaying the current map received from Dwarf Fortress, and registering keystrokes that are sent to Dwarf Fortress as input.

The first thing that should be done will be to create a simple mod, which implement the basic features needed. These features are placing and removing blocks, and registering keystrokes. The easiest way to implement both features will most likely be to use a keystroke as a trigger to display or remove a block.

The next step will be to display whole slices of data in the world of minecraft. This data represents a slice of the world from Dwarf Fortress.

[1] `https://minecraft.net/`
[2] Full title: *Slaves to Armok: God of Blood Chapter II: Dwarf Fortress*, `http://www.bay12games.com/dwarves/`
[3] Three-dimensional cubes

2.2 DF part

The Dwarf Fortress side of the software will be different, as it does not directly modify the game like with Minecraft, but instead must simulate input and screenscrape from the game, without interacting with the games code directly.

Firstly the program should send keystrokes to the running Dwarf Fortress application, which could be done by simulating keystrokes directly on the computer, or aimed at the application itself.

Secondly the program needs to read the data currently displayed by the application. This could possibly be done by actually 'scraping' the screen, by reading what the game prints on the window. This may be quite hard though, since the graphics use openGL, and is therefore not text but and image. The way to do it would then be by capturing an image of the screen and then analyzing the image pixel by pixel to find the data of each square.

Another method could be reading from the memory of the running application, if the memory addresses of the data are obtained, and the data can be decoded.

2.3 Network connection

Finally the two parts of the project must be connected. Since a network connection should be used, the immediate solution would be to use sockets.

The data sent back and forth would also need to be encoded to be sent via a socket connection. If both ends of the software are written in Java, this could easily be achieved since Java can natively encode whole Objects at a time.

If the two ends of the software are written in different languages, the encoding would probably have to be done manually.

2.4 Requirements

These are the requirements specified in the project description.

2.4.1 Mandatory

- Create an error free mod for Minecraft (i.e. no crashing)
- Implement a playable Fortress mode.
- The connection between Minecraft and Dwarf Fortress should be network-based.

2.4.2 Optional

- Running Dwarf Fortress on a separate machine should only allow one connection per game.
- Having both Adventure and Fortress modes from Dwarf Fortress.
- All of Dwarf Fortress in Minecraft (every functionality, objects, etc.)
- Starting a new game should be done from within Minecraft (menus and options etc.)
- Accessing all Dwarf Fortress menus/options through Minecraft in corresponding settings menu.

- Custom meshes for various Dwarf Fortress creatures etc.
- Easily installable modpack compatible with launchers such as MagicLauncher[4] which can install mods and run the Minecraft client.
- Mod-compatibility with mods that do not alter the gameplay of Minecraft, such as graphic enhancements etc.

[4]http://www.minecraftforum.net/topic/939149-launcher-magic-launcher-100-mods-options-news/

3 Design

3.1 Program structure

The overall program structure can be represented by a simple diagram[5], showing the relations between the different components. The two 'mods' are connected via sockets, the Minecraft part is integrated into Minecraft itself, while the Dwarf Fortress part is separate from the DF executable.

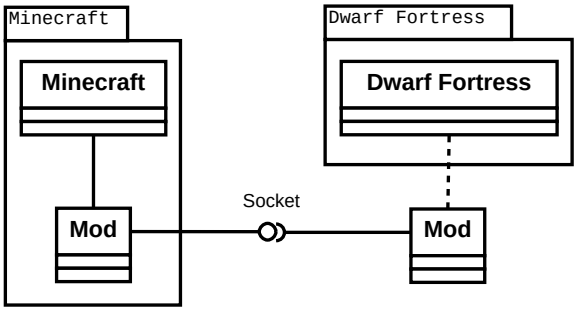


Figure 1: Simple component diagram

3.2 Simple Minecraft mod

The creation of a basic Minecraft mod should be relatively simple. Using the MCP toolkit[6] Minecraft can be decompiled, allowing the source code to be modified.

A mod called ModLoader[7] provides a basic API for modding Minecraft, and allows for mods that do not directly alter the Minecraft source code, but instead the mods are written in separate classes. This helps with mod compability since the original Minecraft source is kept intact.

Using ModLoader's API a simple mod to place a block can be implemented.

3.2.1 Slice

With the API provided by ModLoader the rendering of a slice of map, is simply accomplished by looping across an array, and placing each block. If the array is large, a 'smart' rendering system may be needed to reduce the number of blocks being updated in Minecraft.

3.3 Screenscraping

Screenscraping from Dwarf Fortress is slightly more complicated. The most likely way to do so, is reading directly from the memory used by Dwarf Fortress.

DFHack is a third party application which reads and writes to the memory used by Dwarf Fortress to manipulate a game, and can be used to 'cheat' or do things that are not necessarily in the normal game.

[5] Shown in figure 1

[6] *Mod Coder Pack*, `http://http://mcp.ocean-labs.de/`

[7] `http://www.minecraftforum.net/topic/75440-v162-risugamis-mods-updated/`

It seems that the memory addresses of data in Dwarf Fortress changes in different versions of the game[8], and that makes it harder to ensure compability in future updates.

Because DFHack reads from the memory it could be used to retrieve the map from Dwarf Fortress.

DFHack has support for Lua and ruby scripts. This can be useful because they should be useable in different version of DFHack. This would mean that when Dwarf Fortress updates, and a new version of DFHack is released, the script would still work, meaning that the software created will require minimum maintenance.

3.4 Commands to DF

Because Dwarf Fortress primarily runs on Windows (though linux and mac versions are available) the software running on the Dwarf Fortress end can be Windows specific, such as a .NET language. The .NET framework has support for simulating keystrokes of various types.

Starting Dwarf Fortress from within the program, gets a handle that can be used to send keystrokes to that application even if it is not in focus.

3.5 Network connection

TCP sockets are a simple way to create a network connection, and can easily be used to connect programs written in different languages. This makes sockets ideal for this application.

Because Minecraft is used as the 3D visualizer for Dwarf Fortress, the Dwarf Fortress end is used as the server, while the Minecraft mod is used as a client. This also makes sense in the way that Minecraft takes all the user input. Minecraft gets a user command, sends it to the Dwarf Fortress part, which sends back a response with the current map.

3.5.1 Data between DF and MC

Because the Minecraft end and Dwarf Fortress end will be implemented in different languages, an encoding system is needed to transmit data more complex than simple strings or numbers. If only the map itself is sent as a two-dimensional array, it can easily be flattened and sent as a stream of bytes. If more data, like creatures and dwarves, needs to be transmitted as well, a more complex encoding system may be needed.

To flatten the two-dimensional array to a one-dimensional array is reasonably simple. The array is just printed one line at a time, making a long one-dimensional array as shown in figure 2. In order to decode this array the original dimensions are needed, so they are transmitted before the array of data. This also tells the receiving socket how much data to read.

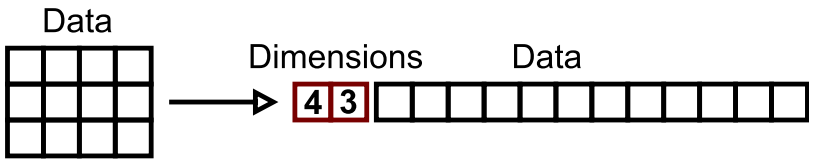


Figure 2: Simple flattening of array, with dimensions appended to front.

[8] http://dwarffortresswiki.org/index.php/DF2012:Memory_hacking

4 Implementation

4.1 Simple Minecraft mod

The simplest way to mod Minecraft is to decompile it, and modify the source. The MCP toolkit makes this trivial.

ModLoader provides an API for easy modding of Minecraft. To use this API, the ModLoader files are included in the Minecraft .jar file, which is then decompiled using MCP.

ModLoader automatically loads all files prefixed with '`mod_`' and extending the BaseMod class, which allows for a mod to be placed in separate files, which allows for greater mod compability.

4.1.1 Placing a single block

The first mod to be implemented simply places and removes a single block at a specific location in the world. This is triggered by a keypress from within Minecraft.

The method used to place a block is '`setBlock`' which was found by looking through the Minecraft source. The main `Minecraft` object is accessed through ModLoader, and is used to call various methods:

```
Minecraft mc = ModLoader.getMinecraftInstance();
mc.theWorld.setBlock(0, 20, 0, Block.blockDiamond.blockID);
```

The keystrokes are defined by a `KeyBinding` object, which must be registered in the system. This is done in the method '`load`', which is called by `ModLoader`. When a key is pressed the method '`keyboardEvent`' is called.

```
KeyBinding key = new KeyBinding("key", 36); // J

@Override
public void load() {
    ModLoader.registerKey(this, this.key, false);
    ModLoader.addLocalization("key_test", "Test");
}
```

The source for this mod is in the file '`mod_PlaceBlock.java`', but will not be included in the final mod. This simple mod was tested in Minecraft. Two screenshots from the game can be found in appendix A.1, which shows the block being placed.

4.1.2 Displaying a slice

A similar approach is used to implement the rendering of a whole slice of blocks, the difference being that instead of placing a single block, a method is called, which loops over an array, placing a block for each field.

The source is in '`mod_DisplaySlice.java`', and will not be in the final mod. Similar tests to before can be seen in two screenshots in appendix A.2.

4.2 Commands to DF

In order to send commands to Dwarf Fortress, the software needs to be able to simulate keystrokes. Using the .NET framework this can be done with the PostMessage function. The specific software used to write the Dwarf Fortress end of the software is C#, since it is the .NET language closet to Java.

One of the parameters in the PostMessage function is a handle to the window to which the message will be sent. The easiest way to obtain this handle is to start an instance of Dwarf Fortress from the C# application itself, since that creates a direct reference to the process. This handle can be retrieved using `Process.MainWindowHandle`.

This works fine as long as Dwarf Fortress is executed normally by the program. But when DFHack is installed another window for DFHack is opened, which is set at the main window. In order to find the correct handle, all windows associated with the process are looked at, and the one with 'Dwarf Fortress' as the title is chosen:

```
foreach (var hWnd in EnumerateProcessWindowHandles(df.Id)) // Gets
    all windows for the current df process.
{
    int capacity = GetWindowTextLength(hWnd) * 2;
    StringBuilder sb = new StringBuilder(capacity);
    GetWindowText(hWnd, sb, sb.Capacity);
    if (sb.ToString().Equals("Dwarf Fortress")) // The window title
        should be Dwarf Fortress
    {
        df_handle = hWnd;
        break;
    }
}
```

By using this handle a keystroke can be sent to Dwarf Fortress. This can also be used to send keys to DFHack, which runs next to Dwarf Fortress as a console window.

If an application uses a console window, the standard input and output (the console) can be redirected to streams in the C# application with the handles.

This means that DFHack could be used directly to get the map data from Dwarf Fortress. The problem is that DFHack does not actually use a console window, it merely looks like one. The input is registered by a keyboard listener, and the output is drawn on the screen, which means that the standard in/output cannot be redirected.

A potential solution to this problem would be to use the same PostMessage function as with Dwarf Fortress, which works fine for single characters, but does not work as when trying to send a series of keys to simulate the user writing a command.

Another problem discovered is that because the application uses a keyboard listener, a modifier key, such as shift, is registered even if the application is not in focus. After discovering this, the same was found to be true of Dwarf Fortress itself, meaning that the PostMessage cannot be used to simulate a modifier key, since the physical key I registered instead.

In conclusion, the software can send single keystrokes to both Dwarf Fortress and DFHack, but cannot control the modifier keys.

4.3 Screenscraping

4.4 Network

The network connection is implemented using sockets. The Software controlling the Dwarf Fortress part of the system will act as the server, while the Minecraft mod will be the client. This makes sense as a typical client-server setup where the client sends a request to the server, and the server responds. In this case the client sends a keystroke to the server, which returns the current map of the world.

Since both the client and server run on the same computer while programming, the IP-address of the server is just 'localhost'. An arbitrary port is also chosen, in this case the port number is 1234.

4.4.1 Sending keys

The Minecraft end of the connection consists of sending a key command to the server, and receiving a message in return. The message will often be the map, but in some cases the game map will not be updated, so it is not necessary to return the map.

When a key in Minecraft is pressed the 'keyboardEvent' method is called. In this method the key is identified, and a new thread is created. This thread connects to the server and sends a string representing the key command. The thread then waits for a response from the server.
The message is sent via the socket using a PrintWriter:

```
Socket socket = new Socket(host, port);

PrintWriter out = new PrintWriter(new BufferedWriter(new
    OutputStreamWriter(socket.getOutputStream())), true);
BufferedReader in = new BufferedReader(new
    InputStreamReader(socket.getInputStream()));

// Send key to C#
out.println(sendString);
```

Running each connection in a separate thread means that if an error occurs, only that thread will terminate, allowing the rest of the program to continue.

To allow for modifier keys to be used, the string sent consists of two parts: a modifier key and a key. These are separated by an underscore character. An example could be '`Shift_e`' or '`Ctrl_a`' or anything of that nature. If no modifier is used, the string would simply be '`_e`'.

4.4.2 Returning map

On the C# end of the program the socket Is managed in a thread as well. A single thread is started at the start of the program execution. This thread consists of an infinite loop, which utilizes a TcpListener object to listen for clients connecting to the server. When a client connects, a different method is called, which reads the incoming stream of bytes. These bytes are converted to a string,

which can be processed and an appropriate keystroke can be sent to Dwarf Fortress. In order to accommodate for the different modifier keys, the received string is split at the '_' character and put in an arrays of length 2.

Depending on the received message a keystroke is now sent to the Dwarf Fortress executable as described above.

Because the map will not always need to be returned to the client, a single byte is sent back to the client informing the client to expect an array of map information, or simply a return message. After this byte either the map data or a simple message is returned to the client, and the connection is closed.

Because the socket only sends a stream of data the two-dimensional map needs to be flattened as described in the Design part of the report. Firstly the dimensions of the array are sent as bytes, and then the map array is sent, one line at a time:

```
static byte[] Encode(byte[,] input)
{
    using (var stream = new MemoryStream())
    using (var writer = new BinaryWriter(stream, Encoding.UTF8))
    {
        var rows = input.GetLength(0);
        var cols = input.GetLength(1);
        writer.Write(rows);
        writer.Write(cols);
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                writer.Write(input[i, j]);
            }
        }
        return stream.ToArray();
    }
}
```

The map is stored as a two-dimensional byte array, because it is assumed that there are not more than 256 different block types in Dwarf Fortress. This could of course be changed to more bytes per block, but it will probably be safe to assume that 2 bytes will be enough since that allows for more than 65 thousand different values.

Because the dimensions of the array could be more than 255 in either direction, they are represented as integers, and therefore 4 bytes each. This means that 8 bytes are sent before the map itself.

4.5 Showing the map

After receiving the map from the C# server, the Java part has to decode and display it in Minecraft. Firstly the dimensions of the map array must be decoded, which is quite simple (b.length is 4):

```
for (int i = 0; i < b.length; i++) {
    inWidth += a[i]* Math.pow(256, i);
    inHeight += b[i]* Math.pow(256, i);
}
```

Now that the dimensions of the array are known, the stream of bytes are converted to a twodimensional array of integers, which can then be displayed in Minecraft:

```
int[][] result = new int[inHeight][inWidth];

for (int z = 0; z < inHeight; z++) {
    for (int x = 0; x < inWidth; x++) {
        result[z][x] = inArray[z*inWidth+x];
    }
}
```

This map is displayed in much the same way as the slice in the simple mod implemented at the beginning. The mayor difference is that because the map could potentially be quite big, a 'smarter' algorithm is implemented.

4.5.1 'Smart' rendering

The basics of the smarter rendering algorithm is that any block in the new map which is the same type as the corresponding block in the current map is not displayed using Minecraft's displayBlock method. This should cut down on time during the rendering of a new slice of the map.

Another thing to consider is that the new map might be smaller or larger than the old one. If the new one is larger, the drawing of the new block simply expand the map, but if the new map is smaller than the old one, a piece of the old map could be left around the new map.

This is considered in the algorithm by looking at the sizes of the maps, and if the old one is bigger, the non-overlapping parts are cleared by setting the block as an air-block. The right edge, the bottom edge, and the bottom right corner are iterated over seperately, to minimise the number of blocks looked at.

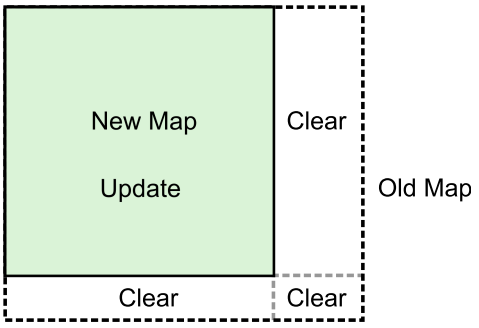


Figure 3: Clearing of excess map when the new map is smaller.

5 Discussion

5.1 Minecraft

The Minecraft part of the software contains the two most critical parts: reading user input, and displaying the slice of map sent from the server. There are several things which need to be expanded, one being the number of hotkeys implemented, and another being a translation from Dwarf Fortress block types to Minecraft block types. Unfortunately the Dwarf Fortress map export has not been completed, so the second part is not immediately possible.

5.1.1 'Smart rendering' tests

The rendering of a slice of map is done by iterating through every block in the slice, and displaying it in the Minecraft world using a function. In the simple 'smart' rendering the previous state of the map Is remembered, so only new blocks will have to be changed.

To see what impact this smart rendering has on performance, a simple test is conducted[9]. An array of 500 by 200 blocks is displayed in the world, and the time of each update is recorded. The first test uses Minecraft's blocks 1 and 2 (Stone and Grass). Each block is randomly assigned one of these two block types.

Both 'smart' and 'dumb' rendering are tested with 10 updates, with an initial run where everything is displayed first.

The results show that the two methods are very similar in regards to speed, but there is a big spike at the beginning, where a blank world is replaced by 500x200 blocks. This seems to indicate that replacing the air blocks takes the longest.

To test this, another test is conducted with the same two materials as the first test, only this time a third block (Air) is added. In this second test both algorithms take a long time to display the blocks. The smart algorithm is slightly quicker, but only by 7%, and that is too small a variation over only ten tests to says that conclusively that it is faster. The obvious conclusion is that air-blocks or transparent blocks take a very long time to place or remove.

[9]The results of these tests can be seen in appendix C.1

A third test with five different block types is run, and shows the same results as the first test.

The results of these tests show that placing or removing transparent blocks takes a lot longer than replacing vast amounts of solid blocks. This may be caused by the fact that transparent blocks show more faces of the other voxels, which means that a lot more blocks have to be calculated and rendered, instead of just a single face per block.

The difference in update time between the solid slice and the one with transparent blocks is massive. With air blocks each update takes around 7 seconds to complete, while the solid slice only take a few milliseconds.

One final test is done, containing six block types, including air. This time only one in twenty blocks change, and still have a 1/6 chance o staying the same. This test would be closer to the real game, where only a few blocks change every step.

This test shows that the smart rendering has an effect on performance, with an average time of 0.153 seconds, while the dumb algorithm has an average of 0.786 seconds. This is around five times faster.

5.1.2 Keystrokes

Keystrokes in Minecraft are reasonably easy to set up, but a lot of keys are already used in the game. An example would be the keys W, A, S, D, Space, and Shift, which are used to move the avatar around.

This means that there might not be enough keys left to represent the keys in Dwarf Fortress where almost the entire keyboard is used.

One solution to this might be to use the chat function of Minecraft, where a user can type anything without interfering with the rest of the game. This would mean that all the keys are free for whatever else they might be used for, and a number of extra commands would be available in the chat.

One advantage to using hotkeys over commands in the chat is that they can be remapped from within Minecraft, but the sheer number of them might quickly make it hard to remember everything. In the chat the game itself can write back, and would function like a primitive command line interface within the game.

5.2 Dwarf Fortress

The Dwarf Fortress part of the software is far less developed, and mostly serves as a placeholder or a proof of concept for a final program, which can properly send simulate keystrokes and extract data from Dwarf Fortress

5.2.1 Keystrokes

As mentioned in the implementation part there are several problems with the registration of keystrokes in Dwarf Fortress. The most significant is probably the lack of modifier keys, since a lot of hotkeys use Ctrl or Shift.

This problem seems to be caused be the fact that Dwarf Fortress uses a keyboard listener, and might be solvable by running DF on a different machine from Minecraft, but that seems to be too much to require from the user.

Another option might be to use the Lua API provided by DFHack. From looking at the scripts provided in DFHack, it seems that simulating keystrokes with the API might be possible, but it's not something that has been looked into.

5.2.2 Reading map data

Since reading the memory might be the only viable method to extract the map data from Dwarf Fortress, the Lua scripts look like a good solution here as well. The only problem then is to get the C# program to run the Lua scripts every time a connection is sent from Minecraft.

It might be possible to install some sort of plugin for Lua which provides support for sockets and using that to make the Lua script in DFHack a server communicating directly with Minecraft.

5.3 Network

The use of sockets as a method of connecting the two parts of the project seems to be a good solution, especially if the game needs to run over a network. The good thing about sockets is that they can be used across multiple languages as long as each end knows how to encode and decode the data.

The basic structure of the project also lends itself nicely to the server/client structure, where Dwarf Fortress is the server, and Minecraft is the client.

The sending of keystrokes and map objects (and potentially more data from Dwarf Fortress) could be quite easily achieved if both the server and client ends were written in Java. The 'encoding' would take place when the software reads directly from Dwarf Fortress, and once an object is created, the data would be easily accessible and organized.

5.4 Further development

As mentioned one potential further development would be to read all commands from the Minecraft chat instead of as keystrokes. This allows for the keys normally used by the game to be free, and would also go a long way towards making the mod compatible with other mods, since all the keys would be free to use.

Another thing to do would be to list all available commands from Dwarf Fortress, to make it easier to implement.

Similarly a list of all block types, creatures, dwarf types etc. should be compiled to give an overview of the things the Minecraft mod should be able to view. Potentially this would also make it easier to add new things if this list was structured in a good way.

The implementation of Lua scripts with some sort of sockets would also be a very obvious extension, and possible the first that should be implemented as it would give the biggest leap in functionality.

After that, almost everything from the list of optional goals from the projcect description could be added as well.

6 Conclusion

Overall this project was not successful in meeting all the requirements, specifically the requirement that Dwarf Fortress should be playable from within Minecraft. The requirements that were fulfilled were:

- Creating an error free Minecraft mod

The mod has had no crashes in any of the development or testings, but on one hand it also does not have a lot of functionality or complexity, which helps to reduce instability.

- Connection the two games via a network based connection.

The two pieces of software attached to each game are connected via TCP sockets.

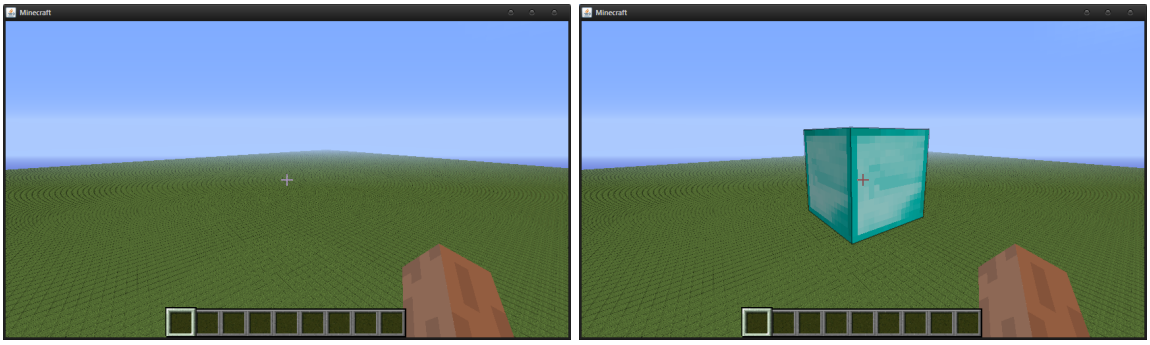
- The optional requirement: Easily installable mudpack..

This was more or less automatically achieved by using ModLoader as an API for modding, which means that all code is in separate files, and none of the original source is modified.

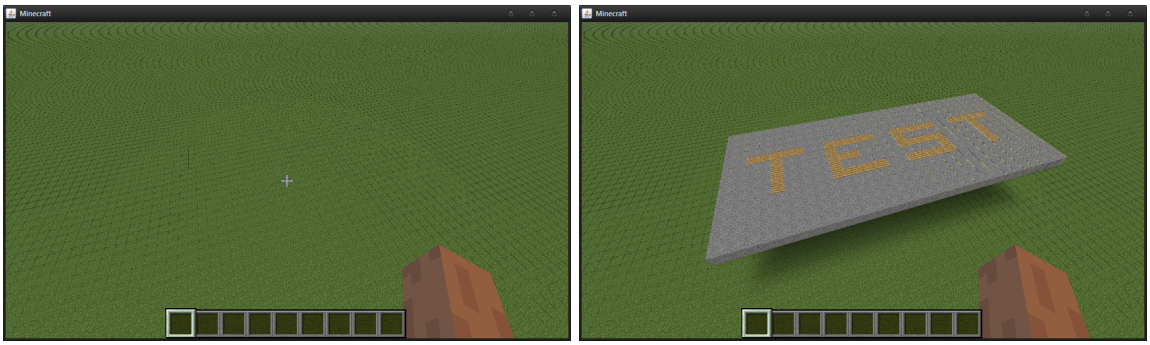
All in all this project can be seen more as a proof of concept of most of the things discussed in the project (Minecraft modding, simulating keystrokes, screenscraping, etc.) even though it is not a whole working project by any means.

A Screenshots

A.1 PlaceBlock

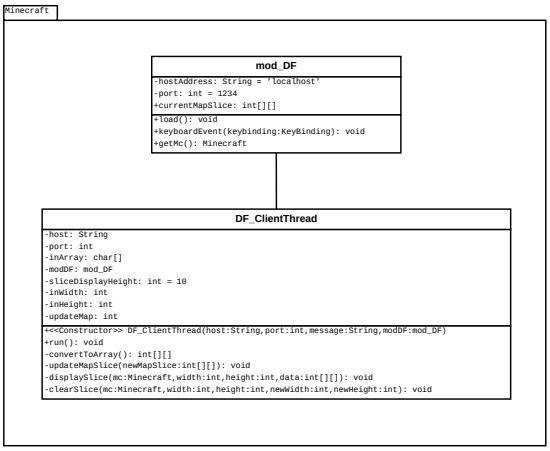


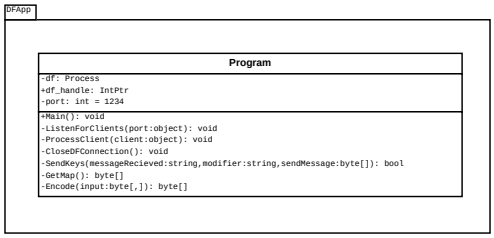
A.2 DisplaySlice



B Diagrams

B.1 Class diagram





C Tests

C.1 Smart rendering

All times are in milliseconds.

	Simple test 2 blocks		Test w. air 3 blocks		Simple test 5 blocks		Fancy test 6 blocks	
	dumb	smart	dumb	smart	dumb	smart	dumb	smart
init	17720	16497	3373	4246	17476	17002	5879	5238
	11	8	7571	8280	10	11	328	195
	10	6	6788	7447	11	9	819	118
	10	5	8538	6426	10	8	540	147
	10	5	6809	8443	10	9	1092	119
	10	6	9571	7397	10	8	656	133
	10	6	7712	8284	11	8	1021	99
	10	6	7956	5335	11	8	542	217
	10	6	7420	6445	11	9	543	47
	11	5	9587	7271	11	9	1015	213
	10	5	6598	7833	10	9	1306	243
average	10.2	5.8	7855	7316,1	10,5	8,8	786,2	153,1