Computer Science Department
Prairie View A&M University

Senior Design Project I (Fall 2019 – Spring 2020)

Report

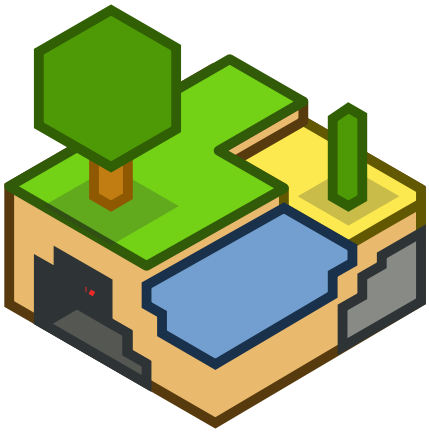
Minetest Mods: Change Player Model + Building Maker + Auto World Explorer

Solo projects by Richard Qian, for Minetest

Instructor: Dr. Lei Huang

This project has no affiliation with any of the Minetest developers.

10/21/19



Abstract

This project set consists of three different mods for the free/open-source voxel game Minetest that aim to enhance the gameplay of the default game that ships with all installations of that game. Two of these projects will involve machine learning (closely related to artificial intelligence) in order to enable gameplay without requiring human supervision. Like most other Minetest mods, these mods will be open-source, to be released under either the GNU Lesser General Public License, version 3.0 or later, or the GNU General Public License, version 3.0 or later.

Change Player Model (*chchar*) is a mod for Minetest that aims to allow players to change the default character model for another one, optionally with default animations.

Building Maker (*building-maker*) is a mod for Minetest that enables a player or non-playable entity to create multiple buildings or pieces of pixel art on their own, optionally without the need for human involvement. The outputs produced can have their appearance, strength, and size influenced.

Auto World Explorer (*auto-explorer*) is a mod for Minetest that enables a player or non-playable entity to automatically navigate, by walking, flying, or swimming, terrain without the need to human input. If a piece of terrain cannot be navigated, the entity will attempt to jump or dig nodes where necessary in order to navigate it. Useful resources may be acquired on the way, and they could be used to aid navigation.

Table of Contents

List of Figures

Chapter 1: Problem Statement

1.1 Background and Statement

Minetest is a free/open-source voxel game engine with easy modding and game creation, plus support for online servers. It has potential to showcase diverse kinds of gameplay due to its Lua-based modding API, such as becoming an RPG (role-playing game) or FPS (first-person shooter).

However, the vast majority of gameplay showcased in practice is mainly like the ones found in Minecraft, InfiniMiner, and similar games, with considerably less content out-of-the-box. To a newcomer looking for a free alternative to Minecraft, he/she may become disappointed to find out that Minetest is not entirely meant to be one, and thus may turn away from it. To some FOSS (free/open-source software) critics, they may point out about the inconvenience of having to download and install 3rd-party games and/or mods just to get a descent experience, with many having varying levels of quality.

In hopes of trying to solve at least a part of these problems, I aim to write a set of three mods that can become stepping stones in creating games with more diverse kinds of gameplay or just making the existing gameplay more interesting. Upon the first public release, their impact will:

- Enable users to load and use arbitrary 3D model files, as long as those formats are supported by Irrlicht, the game engine used in Minetest. The 3D models do not need to be voxel-styled, so users will have more choice in visual styles shown during gameplay.
- Allow for CPU-based gameplay, where a human is not involved in most points of gameplay. The gameplay can be artificial-intelligence based, or based off of pre-made instructions. It might make worlds more lively, especially when there are few or no players playing in it at a time.
- Possibly allow for the creation of more dynamic and helpful non-playable entities who can complete tasks on behalf of players, even if they are not active.

1.2 Challenges Assessment

Normally a project like mine would be entirely volunteer-based. That means it gets worked on whenever I find the time to do so, and releases are made whenever they are done. This project set doesn't cost me any money to complete (except the electricity required to power any computers I use for development), as it is entirely software-based. However, each project will require at least two months to reach completion, for a total of six months for three mods. I have some coding experience, but I will need to learn a lot more and spend more time working if I hope to realize a working version of these mods.

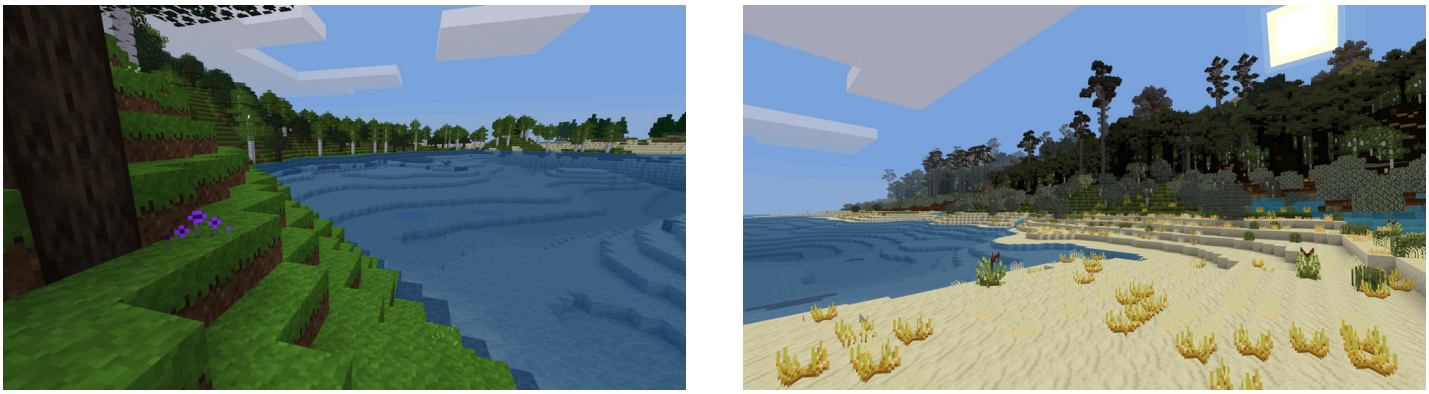


Figure 1: Screenshots from Minetest Game with a Minecraft-like terrain and another not like

Chapter 2: Introduction and Existing Work Survey

2.1 Introduction

The sole developer of the Change Player Model mod, Richard Qian, is currently a 5th-year senior undergraduate student a Prairie View A&M University, majoring in Computer Science. FOSS are not mainstream in most consumer markets, even if some are used as building blocks of consumer-facing proprietary software, so getting assistance has become too difficult for me. The code name for one of my mods, "*chchar*", stands for "Change Characters", and follows a UNIX tradition of shortened executable names. This isn't the case for the other two mods, which instead have the code names "*building-maker*" and "*auto-explorer*". All three of the mod will be open-source, to be released under the GNU (Lesser) General Public License, version 3.0 or later, so that people can integrate them into their own packages or games.

Most open-world and similar-genre video games allow customization of the player character to varying degrees, from changing the character's clothing and skin color to swapping out different models for different kinds of characters, such as from a human to animal. They sometimes also contain some amount of code mainly used for CPU players (if available) enemies, and other NPCs. Additionally, some of these games also allow further customization through scripting languages that extend the functionality of these games. For my project, Minetest is written in C++, built on top of the Irrlicht engine, which is also written in C++. Minetest uses Lua for all things scripting, the most significant usage of it being for writing mods that extend this game's functionality. My project will focus more on the scripting part, so that users won't have to recompile Minetest just to gain the functionality that this mod will provide.

2.2 Existing Work Survey

Minetest already ships with a default game, Minetest Game, that itself is composed of several mods. Below will be described the works (both 1st-party and 3rd-party) most similar to what I will do.

- *mods/player_api*: This mod is what enables the loading of the default player model, and it provides the facilities that make my mod possible to create.
- TenPlus1/mobs_redo: This mod provides an API for adding monsters, animals, and other entities into worlds.
- stujones11/minetest-3d_armor: This provides wearable armor and makes wielded items visible in the camera view.
- minetest-mods/skinsdb: This mod provides support for player skins, supporting three popular inventory mods, including unified_inventory, sfinv and smart_inventory.
- MirceaKitsune/minetest_mods_creatures: This mod allows playing as different kinds of mobs, each of whom can have different properties from the default player characters (humans). Players start out as ghosts that have to be possessed by a mob in order to play them, but can become them upon death.
- AiTechEye/aliveai: This mod is another implementation of mobs, consisting of the base code plus several example mobs.
- jordan4ibanez/open_ai: This mod is another implementation of mobs, but it contains additional code for mobs to target and pathfind, as well as rudimentary building from schematic files (*.mts). I have deemed this mod to contain the most amount of implemented code useful for my three projects.

These works are incompatible with Minetest, but some code found in them may be useful as part of writing my mods:

- facebookresearch/craftassist: “The goal of this project is to build an intelligent, collaborative assistant bot in the game of Minecraft that can perform a wide variety of tasks specified by human players. Its primary purpose is to be a tool for artifical intelligence researchers interested in grounded dialogue and interactive learning.” Written primarily in Python 3, certain files seem to be useful to reference when implementing equivalent code.

- alejo/mario-ai: “This project contains code to train a model that automatically plays the first level of Super Mario World using only raw pixels as the input (no hand-engineered features). The used technique is deep Q-learning.” Written primarily in Lua, the same language that Minetest uses for mods.

Figure 2: The default player model, with the default skin

2.3 Development Environment

The set of software that I have to use is very rigid. However, I can use any text editor to write the code, including those with integrated development environments. Exclusively for *chchar*, the models that that mod can load can be made with any 3D modeling program, but only one of them will be described in more detail.

- Lua – A powerful, efficient, lightweight, embeddable scripting language. The main scripting language that all mods for Minetest have to be written in. My mod will be no exception.

- Irrlicht - A cross-platform high performance realtime 3D engine written in C++. The game engine that Minetest uses for lower-level

functionality. I will have to take this into consideration when writing my mod what file formats are supported by this engine. I can use any format supported by this engine, but only those formats; other files will have to be converted to those supported formats first before they can be used.

- Blender - An integrated 3d suite for modelling, animation, rendering, post-production, interactive creation and playback (games).

  This is the primary program I will use to create compatible 3D models for use with my mod, but they don't have to be created with it.

Chapter 3: Project Time Table --- Gantt Chart

3.1 Project Task List

Some parts of my project have already been done. I've actually started this project back in May 2019, but suspended it due to not having free time to work on it. The following things have already been completed (only for chchar):

- Created the project, and published it to a GitLab repository at https://gitlab.com/Worldblender/chchar.
- Wrote a readme and a few of the files required for all Minetest mods.
- Selected the project license.

The following tasks will have to be completed:

1. Additional project repositories have to be made for the other two mods.
2. Directory layout of the mods and structure for loading models and audiovisual resources. Additionally determine where to place the Lua code files.
3. Research similar works to see how they deal with the player model and implement AI code.
4. Decide on the file formats for describing information about the models (for *chchar* only).
5. Interface design, including the GUI aspects and chat interaction.
6. Implementation of the Lua code, reusing any relevant code if licenses permit it.
7. Testing of the mods in Minetest, making sure that whatever I write actually works.
8. Marking the first stable release, and making the three mods public for everyone to use.

3.2 – Gantt Chart

	Name	Duration	Start	Finish	Predecessors
1	Create Git repositories	1 day	12/10/19, 8:00 AM	12/10/19, 5:00 PM	
2	Add README and other files	3 days	12/11/19, 8:00 AM	12/13/19, 5:00 PM	1
3	Analyze similar mods	1.375 days	10/22/19, 8:00 AM	10/23/19, 11:00 AM	
4	**Implement Lua code (chchar)**	**136 days?**	**10/22/19, 3:00 AM**	**4/28/20, 5:00 PM**	**1**
5	Decide on directory layout	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
6	Write inital Lua scripts	6.8 days?	12/11/19, 8:00 AM	12/19/19, 3:24 PM	
7	Write code dealing with 3D models	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
8	Write code dealing with UI	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
9	Test mod within Minetest	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
10	Produce first stable release	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
11	**Implement Lua code (building-mak...**	**136 days?**	**10/22/19, 3:00 AM**	**4/28/20, 5:00 PM**	
12	Decide on directory layout	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	1
13	Write inital Lua scripts	6.8 days?	12/11/19, 8:00 AM	12/19/19, 3:24 PM	
14	Write code dealing with files and AI	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
15	Write code dealing with UI	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
16	Test mod within Minetest	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
17	Produce first stable release	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
18	**Implement Lua code (auto-explorer)**	**136 days?**	**10/22/19, 3:00 AM**	**4/28/20, 5:00 PM**	
19	Decide on directory layout	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	1
20	Write inital Lua scripts	6.8 days?	12/11/19, 8:00 AM	12/19/19, 3:24 PM	
21	Write code dealing with nodes and...	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
22	Write code dealing with UI	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
23	Test mod within Minetest	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	
24	Produce first stable release	136 days?	10/22/19, 3:00 AM	4/28/20, 5:00 PM	

Figure 3: Time line (spreadsheet form)

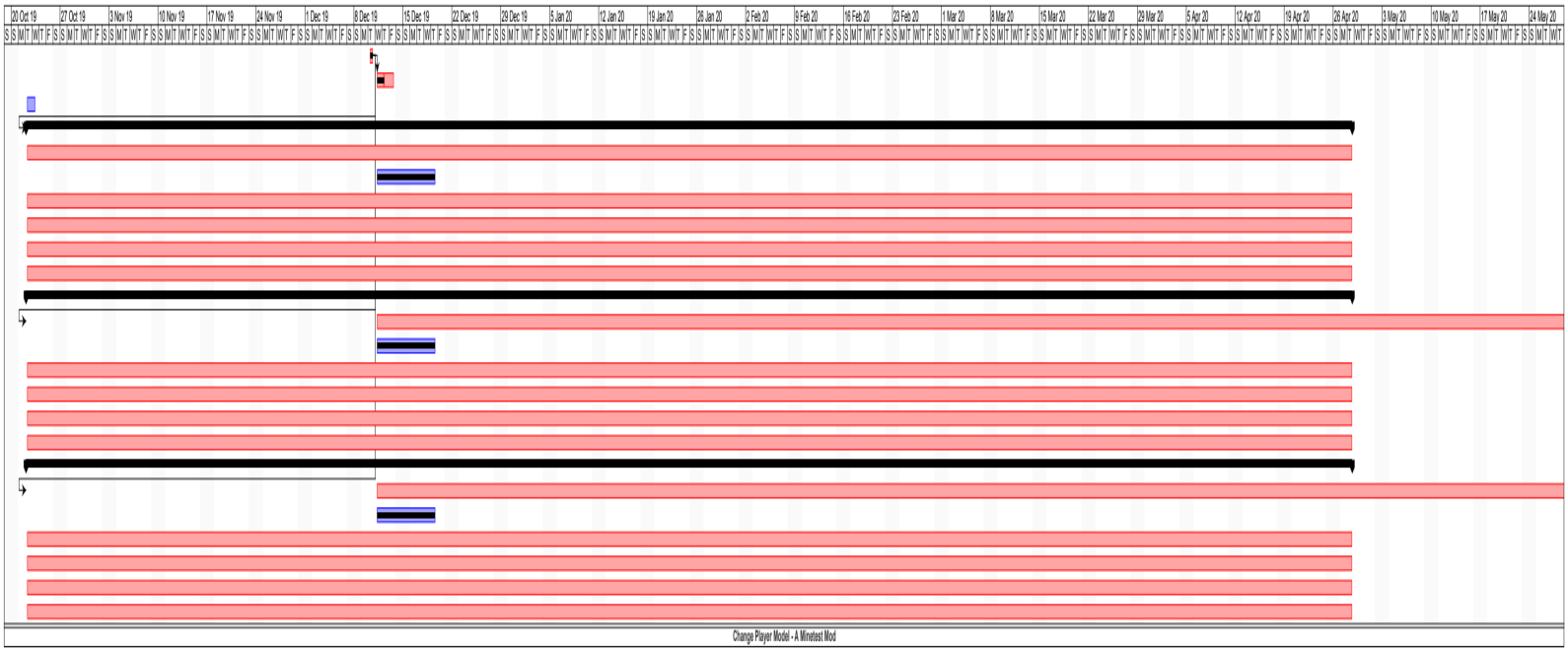


Figure 4: Timeline (Gantt chart form)

Chapter 4: Requirements and Usage Scenario

As these mods run on top of Minetest, they share a few requirements:

- Require a computer and compatible operating system with access to the Minetest client; Windows, macOS, GNU/Linux, FreeBSD, OpenBSD, DragonFly BSD, and Android explicitly supported. (non-functional)
- The mods must be enabled on a per-world basis, and runs for as long a selected world is active. (non-functional)

4.1 *chchar* Requirements

4.1.1 Functional Requirements

- Provides users the ability to load arbitrary 3D models and let them replace the player model.
- Allows to reset to the default player model for compatibility with mods that alter it.
- User interface (both graphical and textual) shall list available models, information about them, and enable users to switch between them and the default player model.
- 3D model settings are saved on a per-player basis. That means new players will start with the default model, while existing ones will have their settings loaded.
- The mod needs to aware of other mods that alter the default player model, and handle cases where it and those happen to be loaded simultaneously; either this or the other mods may be disabled to prevent conflicts.

4.1.2 Non-functional Requirements

- Can load only the 3D model formats that the Irrlicht engine supports.
- 3D models and other audiovisual resources can only be loaded from mod-specific directories; they cannot be loaded from just anywhere.
- GUI has to be called from either the chat prompt or an inventory button (supported mods only).
- Animations are supported, but only with certain 3D model formats, and most formats the Irrlicht engine supports lack animation support by the engine.

4.2 *building-maker* Requirements

4.2.1 Functional Requirements

- A player or NPC can create a building or pixel art by itself, either with pre-made instructions or by itself.
- For pixel art, it can either be made procedurally, or from an image file loaded as a texture.
- There has to be a way for the AI to determine when to stop building, or it could build on forever without end.
- There is a toggle switch that allows stopping the automatic building at anytime, in case a human player wants to take over building or inspect the outputs.

4.2.2 Nonfunctional Requirements

- Buildings and pixel art can be made only with materials seen in the item slot. Up to eight can be loaded at once.
- Only texture files with formats supported by the Irrlicht engine will be loaded.
- It is possible for buildings and pixel art to be made with undesired blocks. This is more likely to happen if survival mode is enabled instead of creative mode.
- The materials used depend on which mods are installed and enabled at run time. More mods can provide a richer palette of blocks, while playing only with the default game mods provides only limited palette of blocks.
- It is best to have creative mode enabled and both survival mode and damage disabled for optimal operation of this mod. The creative mode allows infinite block stacks so that an entity will not run out of blocks to use.

4.3 *auto-explorer* Requirements

4.3.1 Functional Requirements

- A player or NPC can navigate at least one chunk of a world map by itself, without human input. Ideally, it is trying to locate useful resources, with a tree or bush being the first target (both provide wood for making tools out of). If navigation is successful, it may be rewarded with rare items such as mese crystals and diamonds.

- The entity can try to jump and dig nodes if it cannot navigate a piece of terrain. If available, it can also try building instead.
- There is a toggle switch that allows stopping the travelling at anytime, in case a human player wants to take over navigation.
- The entity can decide to explore the world surface, dig underground and mine for resources, or build upwards into the sky.

4.3.2 Nonfunctional Requirements

- World maps are not infinite, so the entity needs to either stop or turnaround when hitting one of the six boundaries.
- It is possible for the entity to die if damage is enabled. It may warp back to a certain point and have to restart navigation.
- The rewards given out depend on which mods are installed and enabled at run time. More mods can provide other rewards, while playing only with the default game mods will offer limited rewards.
- It is best to have survival mode enabled and creative mode disabled for optimal operation of this mod. Enabling damage is optional, but having damage can act as a punishment that an entity can try to avoid.

Figure 5: Screenshots from Minetest showing a built castle and default heads-up display

4.3 Example Use Cases For Each Mod

4.3.1 For *chchar*

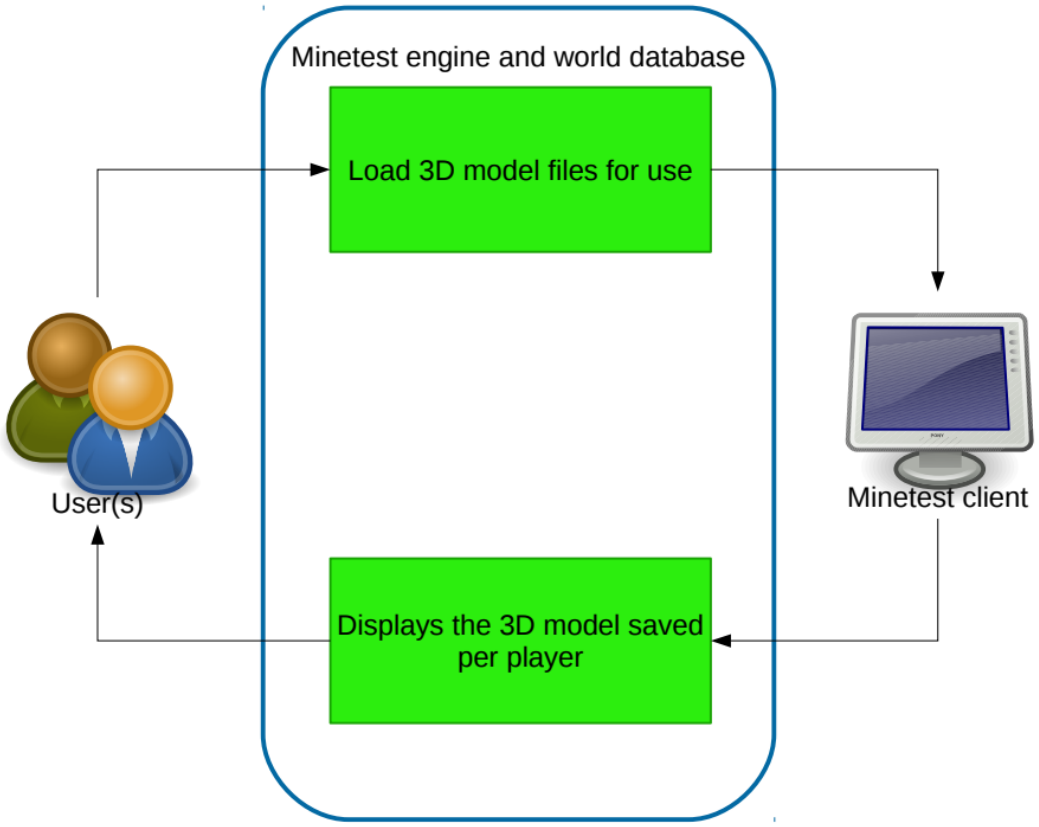


Figure 6: Use case for *chchar*

4.3.2 For *building-maker*

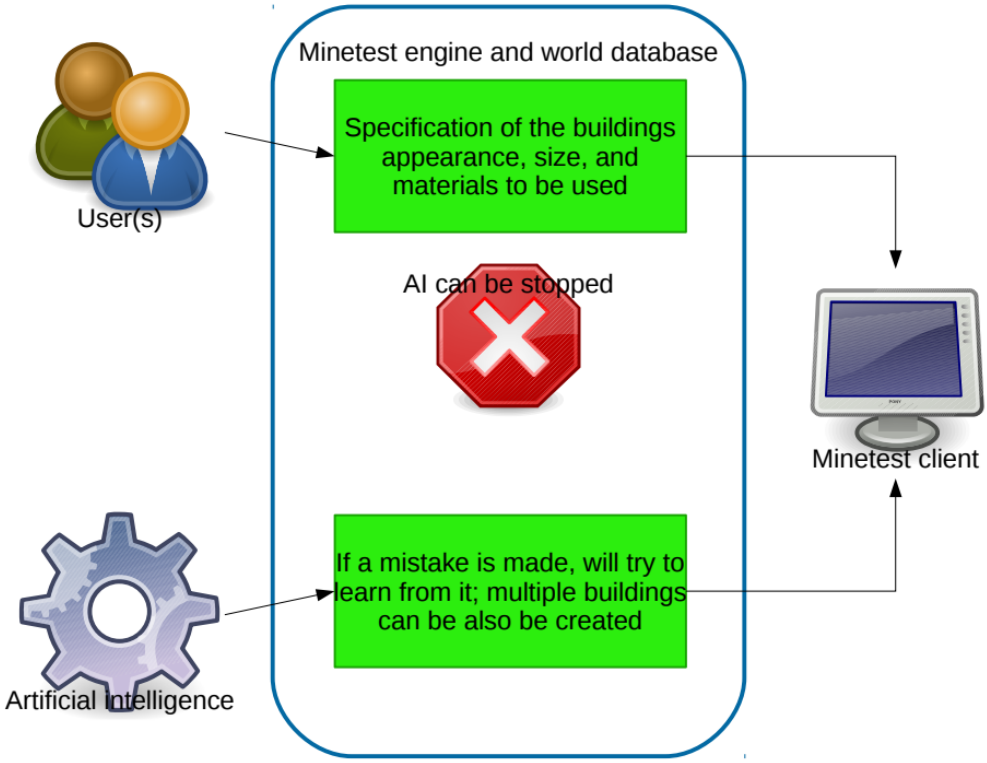


Figure 7: Use case for *building-maker*

4.3.3 For *auto-explorer*

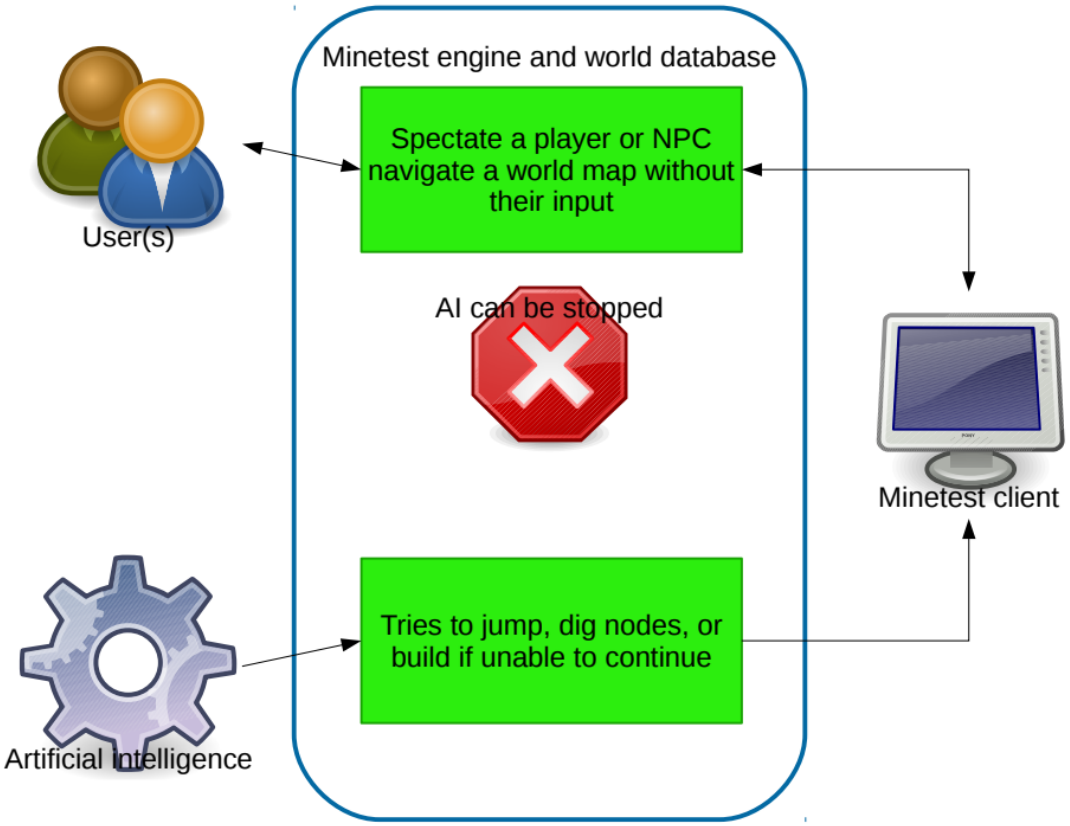


Figure 8: Use case for *auto-explorer*

Chapter 5: System Architecture Description and Interface Design

'*chchar*', '*building-maker*', and '*auto-explorer*' are Minetest mods, so their architectures work more like plugins or add-ons rather than as standalone applications. The content below will be described with that view in mind.

5.1 System Architecture

Minetest is designed as a client-server architecture. The server part is what runs a game world and mods, while the client is the visualization of this world, letting users interact with it.

Server:

- Native C++ that manages a world. It can be run headlessly for remote play.
- The world map consists of a database file (SQLite by default).
- Runs the Lua scripts that alter the functionality of worlds. Loads mods that contain Lua scripts if they have been enabled on a per-world basis.

Client:

- Displays a visualization of the world map onto an output device in 3D.
- Uses the Irrlicht rendering engine to display its contents. This engine is written in C++ for better performance.
- Takes input from users, by default with a keyboard and mouse combo.

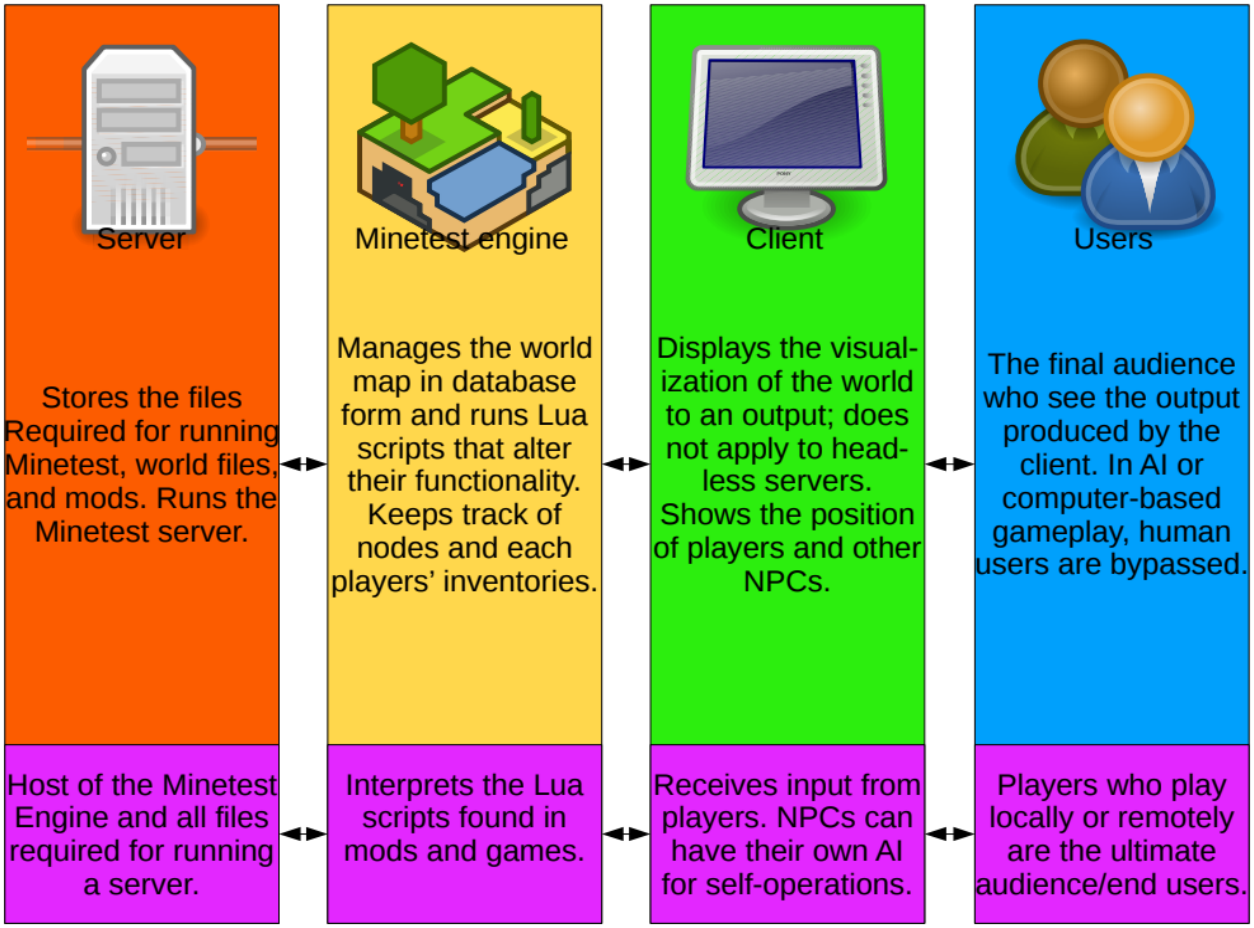


Figure 9: System architecture of a standard Minetest Installation

5.2 Interface Design: Graphical and Textual

The AI nature of the mods *building-maker* and *auto-explorer* means that a large chunk of functionality will involve running by itself, without human input. Thus an interface for these two mods is not feasible, unless this interface is one that can control the running of the AIs themselves (for starting and stopping) or their parameters.

However, an interface for *chchar* can be created. This interface must be called from one of various entry points already provided by Minetest itself or appropriate mods, both 1st-party and 3rd-party ones. The options available are to call one from the chat prompt, adding a button to a mod that provides a different inventory interface, or adding a tab to the built-in *sfinv* mod.

- Graphical – To be launched from an inventory mod, where the inventory can be launched with a hotkey.

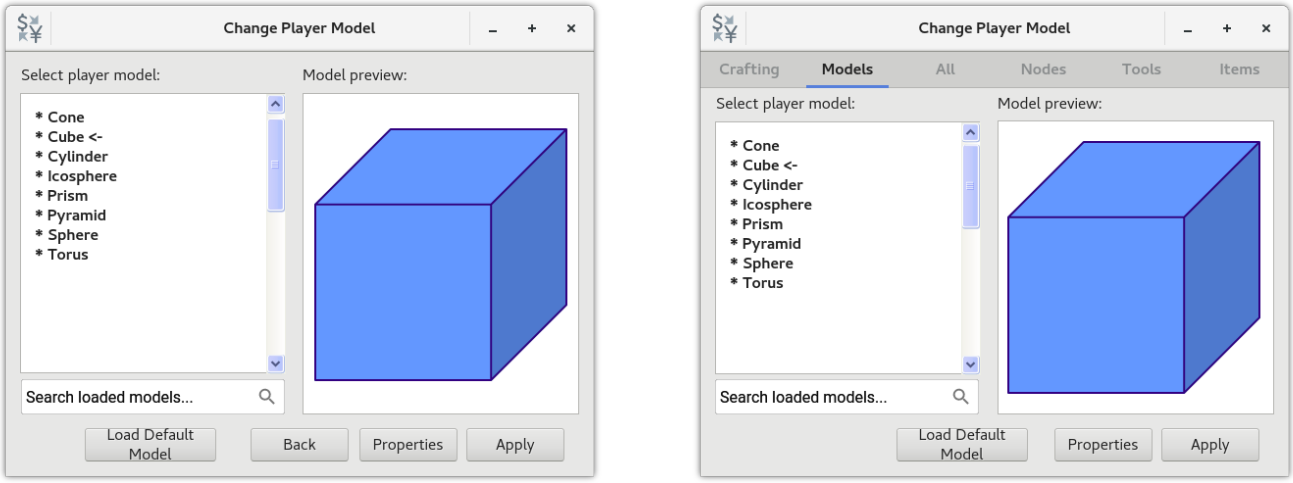


Figure 10: Rough visualization of the GUI (left: all other 3rd-party) | right: for sfinv)

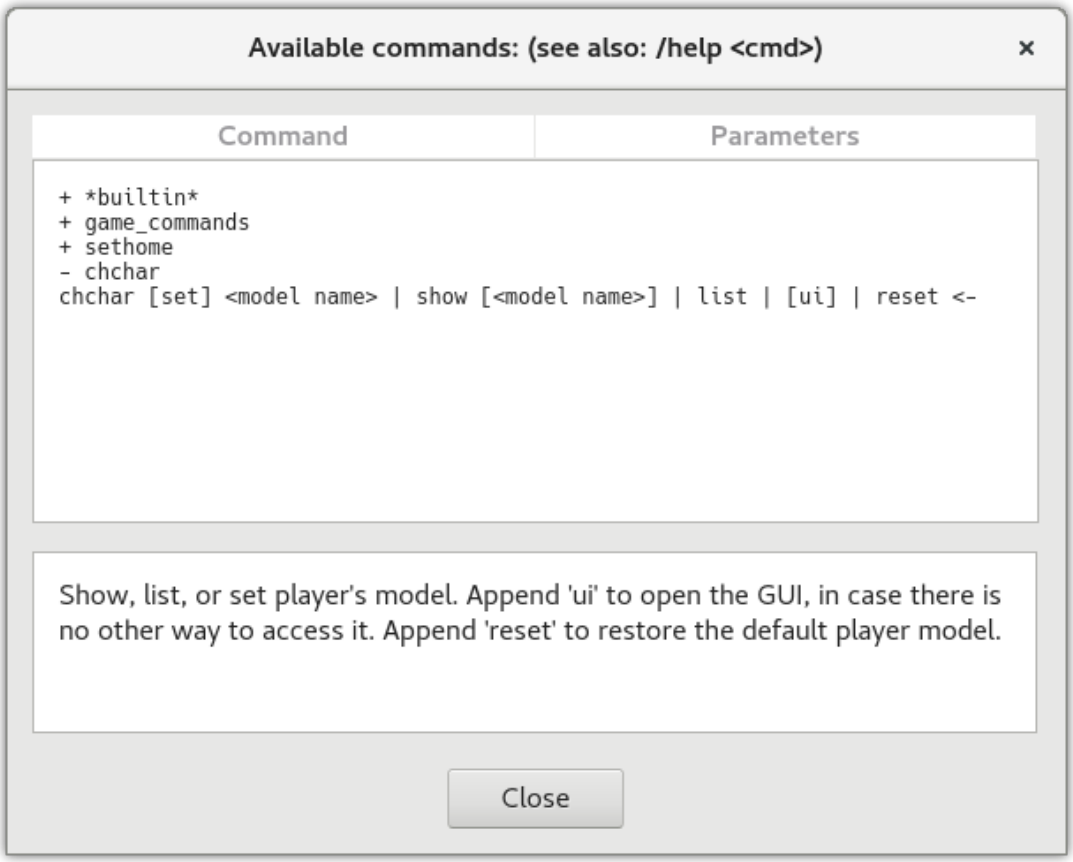


Figure 11: Visualization of the TUI (displaying the command help screen)

- Textual – To be accessed from the chat prompt, and it can either call the GUI or perform a quick action, depending on what arguments are passed after '/chchar'.

Chapter 6: Module and sub-module design and Function Description

6.1 Flowchart

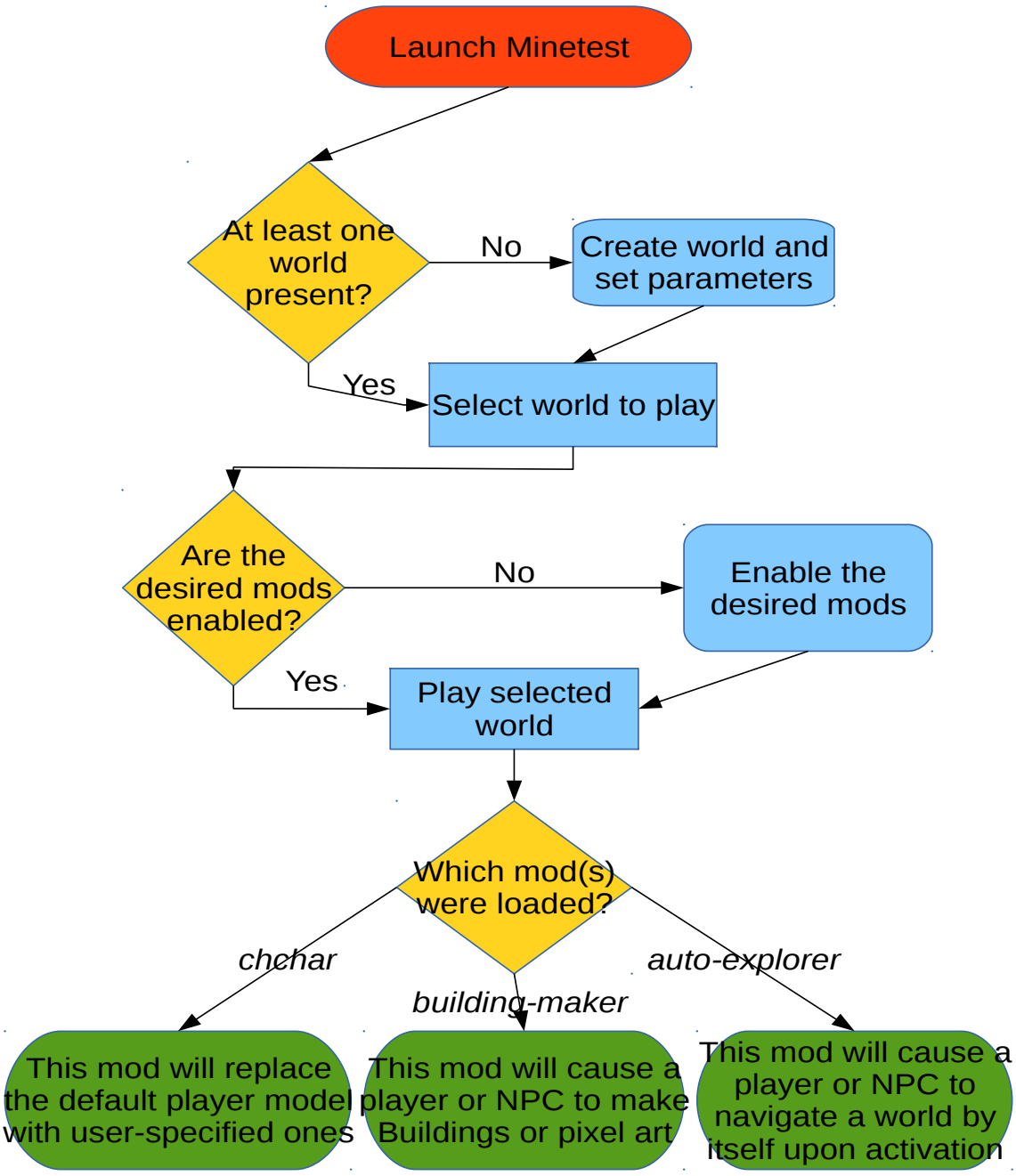


Figure 12: Flowchart from Minetest startup

6.2 Database Design

Minetest uses a database (default format is SQLite) to store each world map and their nodes. *chchar* does not need to perform any operations on it, but *building-maker* and *auto-explorer* will and may, respectively, perform read/write operations indirectly through an entity. Neither of these three mods perform direct operations on the world maps themselves.

6.3 Data Flow Descriptions

Below is a data flow diagram for the entire Minetest program, as of version 0.3. It's not made by me, but I will tell where my mods come into play.

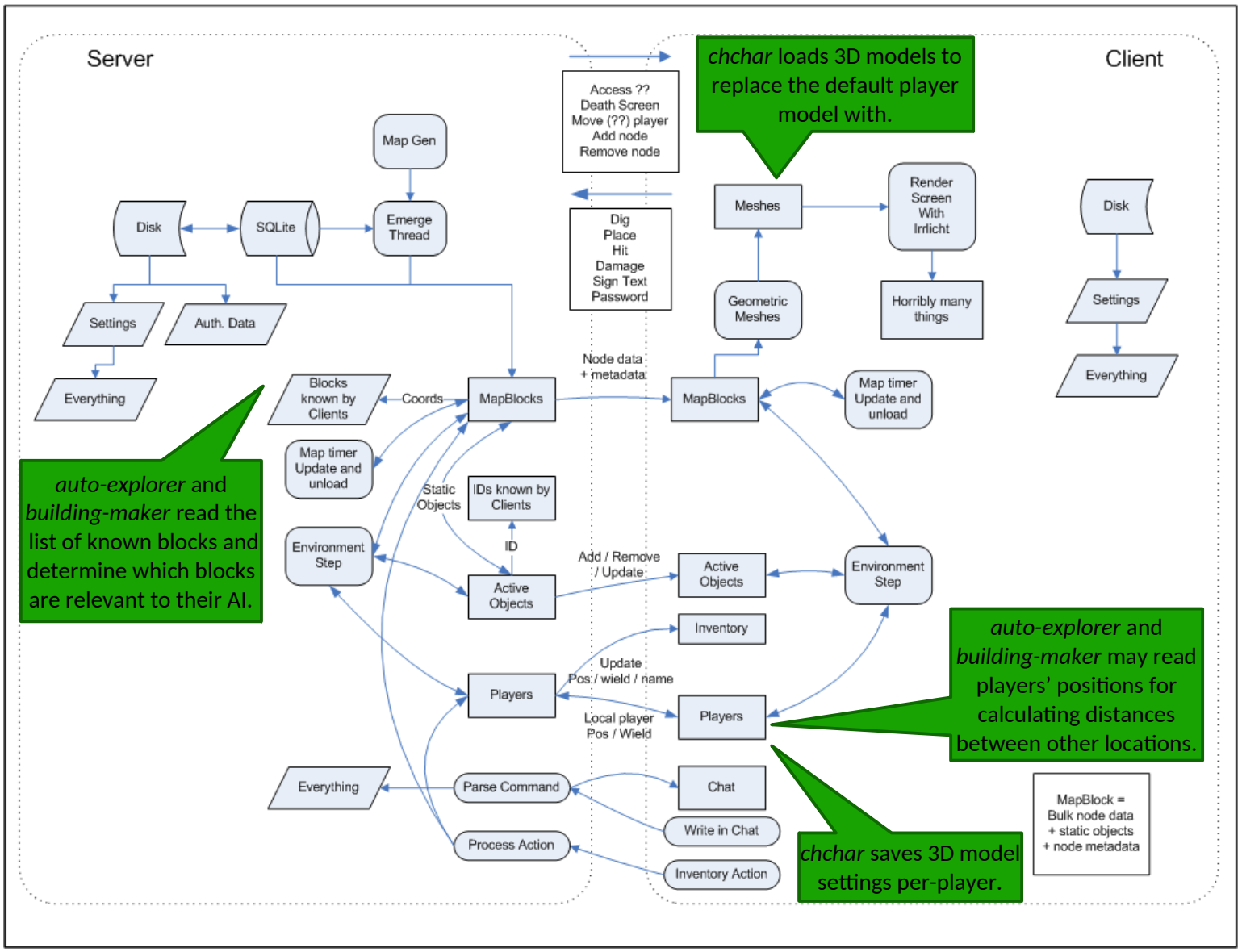


Figure 13: Data flow diagram for Minetest 0.3

Chapter 7: Requirement Validation

As these mods run on top of Minetest, they share a few requirements that they absolutely have to pass:

- None of the mods end in an error during loading of a world. If an error and stack trace is displayed, this requirement will fail for the mod(s) ended in an error. Fixing the code pointed out may solve this problem.
- While a world is running, at least one of these mods has to cause visible change in a player or NPC, whether their appearance, location, or inventory contents. If this isn't the case, this requirement will fail for the mod(s) that did not cause any change.
- In the chat prompt, the command "*/mods*" can be used to determine what mods are loaded in the running world. If the names of any of my three mods are printed, that means that they have successfully loaded.

7.1 Minimum Requirements

7.1.1 For *chchar*

- All 3D models to be loaded are those only in formats that the Irrlicht engine supports.
- Information files for the 3D models are loaded only if they are not malformed. Any malformed ones are skipped and such models associated with those files won't be loaded.
- The mod will save settings per-player. If two players are switched between, they should have their own settings saved.
- There must a way to cleanly switch back to the default player model, and there must not be any traces left behind after this operation.

7.1.2 For *building-maker*

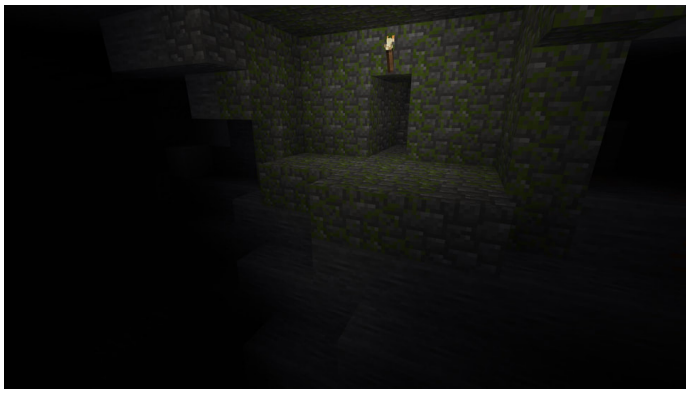
- An AI-controlled player or NPC can successfully create a building or pixel art without having to interrupt its work (final output to be evaluated by a human).
- There must be a way for the entity to stop building. This can be done by checking for closed loops or using a special block or item after hitting a set limit.
- If building pixel art from an image, the blocks' colors closely match individual pixels from the original image (but they don't have to 100% match).

7.1.3 For *auto-explorer*

- An AI-controlled player or NPC can successfully navigate between chunk to chunk without human input or having to be manually interrupted.
- The entity starts moving on its own, without human input (but after the AI is started). It ideally can locate useful resources (such as wood, stone, and iron ores) by itself.
- The entity can avoid actions that harm itself (such as falling down more than five blocks, staying in water for too long, touching lava, or eating red mushrooms). If that does happen, it can use items that restore health (such as apples, brown mushrooms, and blueberries).
- The entity can find workarounds to terrain that can't be readily navigated, such as by finding alternate routes, digging nodes, or building.

7.2 Optional Requirements

- Detection of other mods that alter the default player model needs to be written in order to reduce the likelihood of crashes or conflicts. Those mods will have to be disabled, or my mods will have to accommodate them somehow.
- Optional usage of special privileges if they are enabled, including *fast* (fast movement speeds), *fly* (move without being subject to gravity), and *noclip* (pass through solid blocks).
- Possible integration between two or more of my mods (*building-maker* and *auto-explorer* more likely) in the form of sharing code between them or advertising compatibility.



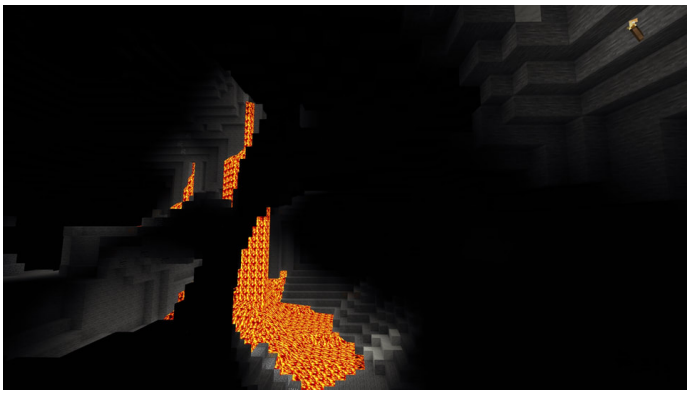


Figure 14: Screenshots from Minetest showing underground (lava is a hazard)

Chapter 8: Risk Assessment and Planning

8.1 Known Risks

As with all other pieces of software, my mods are not expected to be perfect. It is possible for one or more of them to end in failure (while running in Minetest). However, I know that none of these risks but the last one can cause significant damage, as the operations tend to be read-only on data. Here are some that can happen:

- An error in the code itself – Minetest will abort loading or running the world and display the error and where the error happened in the code. I have to step in and fix the errors that arise if they happen.
- The computer Minetest runs on does not have a sufficient enough GPU to run Minetest – Minetest will be unable to run its engine. Software rendering can be used as a workaround, but performance will likely suffer.
- A 3D model or texture with a format unknown to the Irrlicht engine or is corrupted – the model will be skipped, and errors will be printed in the chat window. In the case of a texture, a dummy texture will be created instead, and errors will be printed in the chat window.
- A 3D model that lacks the required information files – the model will not be loaded, and become unusable until such files are created, and the running world is shut down. (only for *chchar*)
- Other mods that alter the default player model or take control of entities with incompatible methods are loaded and not detected – conflicts may arise or the alterations will look out-of-place if no crashes occur.
- A change in the APIs dealing with the player and entities – may require that I rewrite my mods to support them. Not doing this means that it won't support future Minetest versions that change those APIs.
- The APIs dealing with the player and entities happen to be too difficult to understand or cannot be adapted to my use cases – I will be unable to complete any or all of my mods, and they could be left in a potentially perpetual alpha state. The worst case scenario that should not happen, as I have the code of other related mods that I can reference and use to help bootstrap some of my code.

Chapter 9: Interface Implementation

9.1 Programming Environment

All operating systems are fair game, as long as a port of Minetest exists for them. Windows, macOS, GNU/Linux, FreeBSD, OpenBSD, DragonFly BSD, and Android are explicitly supported. The bare minimum needed for coding is a basic text editor. Optionally, an IDE that supports code completion and similar enhancements for Lua can be used. Most testing must be done inside Minetest, as it has the Lua interpreter that also supports Minetest-specific functions. It is possible to test code outside of it if only standard Lua code is being tested. However, two 3rd-party mods allow for running (almost) arbitrary Lua code within Minetest:

- minetest-mods/qa_block: A developer helper mod that allows running any Lua code for testing purposes. It can list and run Lua scripts placed in "checks" subfolder. with some check scripts. The second part is being able to display the global Lua tables tree.
- prestidigitator/minetest-mod-luacmd: "This mod adds the useful debugging command /lua which executes its parameter string as a Lua chunk, as if it were being run in the body of a function inside some mod file. Errors are reported to the player via in-game chat, as is anything sent to the print() function."

9.2 System Development Environment

No special hardware is required to run Minetest, as my projects are entirely software-based. Any desktop, laptop, tablet, or smartphone can run Minetest as long as there exists a port for those devices. It doesn't matter whether a system's CPU architecture is x86 or ARM, all will work, as long as both a suitable operating system and Minetest have been ported to those architectures. System requirements are low, but at least a GPU (graphics processing unit) capable of OpenGL or DirectX is required to optimally display the 3D graphics. It is also possible to complete parts of the development process offline (no internet connection required), although I may be unable to access online resources if I happen to need them during this time.

9.3 UI Screenshots

9.3.1 Screenshots for *chchar*

Below is the chat command help for *chchar*, displayed in-game. It is always accessible no matter what inventory mod is being used.

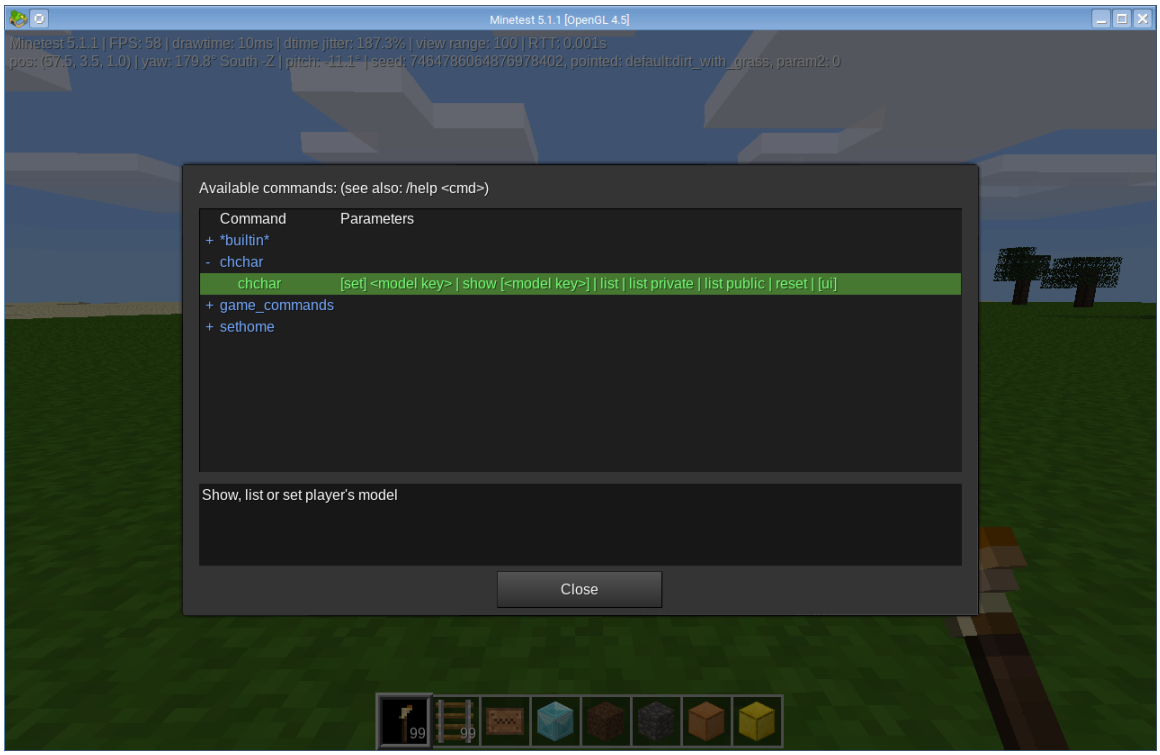


Figure 15: Help information for *chchar* chat command

It is possible to get a GUI out-of-the-box with the *sfinv* mod from the default Minetest Game, assuming that no other 3rd-party inventory mods are installed and enabled. One 3rd-party inventory mod is supported, that being *unified-inventory*.

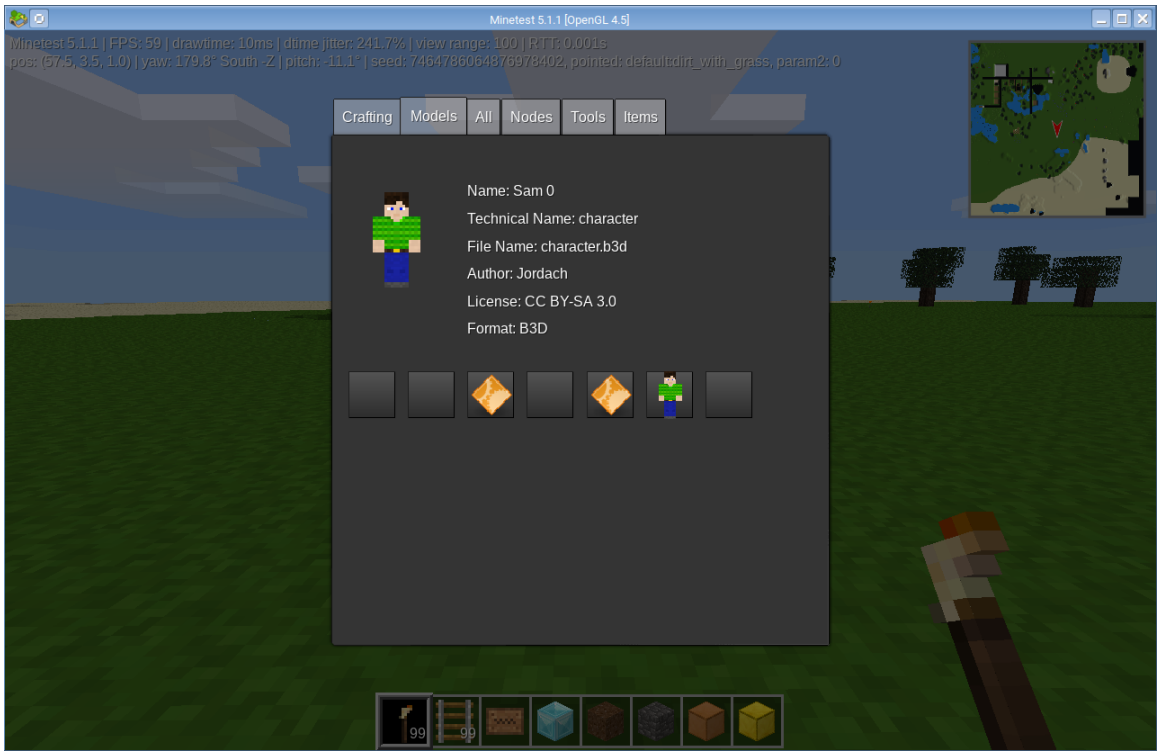


Figure 16: *chchar* GUI using the built-in *sfinv* inventory mod

Figure 17: *chchar* GUI using the 3rd-party *unified-inventory* inventory mod

Chapter 10: Implementation Technical Details

10.1 Implementation of *chchar*

The mod *chchar* is now available at https://gitlab.com/Worldblender/chchar. I have based this mod on top of an existing one from https://github.com/minetest-mods/skinsdb. For the most part, I simply reused the majority of the already-existing code, saving me a considerable amount of time in the process. Afterwards, I thought about how to adapt the code so that it works with 3D model files instead. I did all of the following to adapt it:

- Alter the format of the metadata files to include more fields (specs can be seen on Figure 18).
- Simplified actually changing player models to revolve around using `player_api.register_model()` and `player_api.set_model()`, essentially making this mod into a frontend for *player_api*.
- Adapt the API and its documentation to work with 3D models, mainly in the form of search-and-replace.
- Disable the code that enables downloading of skins in-game, as it is irrelevant to this purpose of this mod.

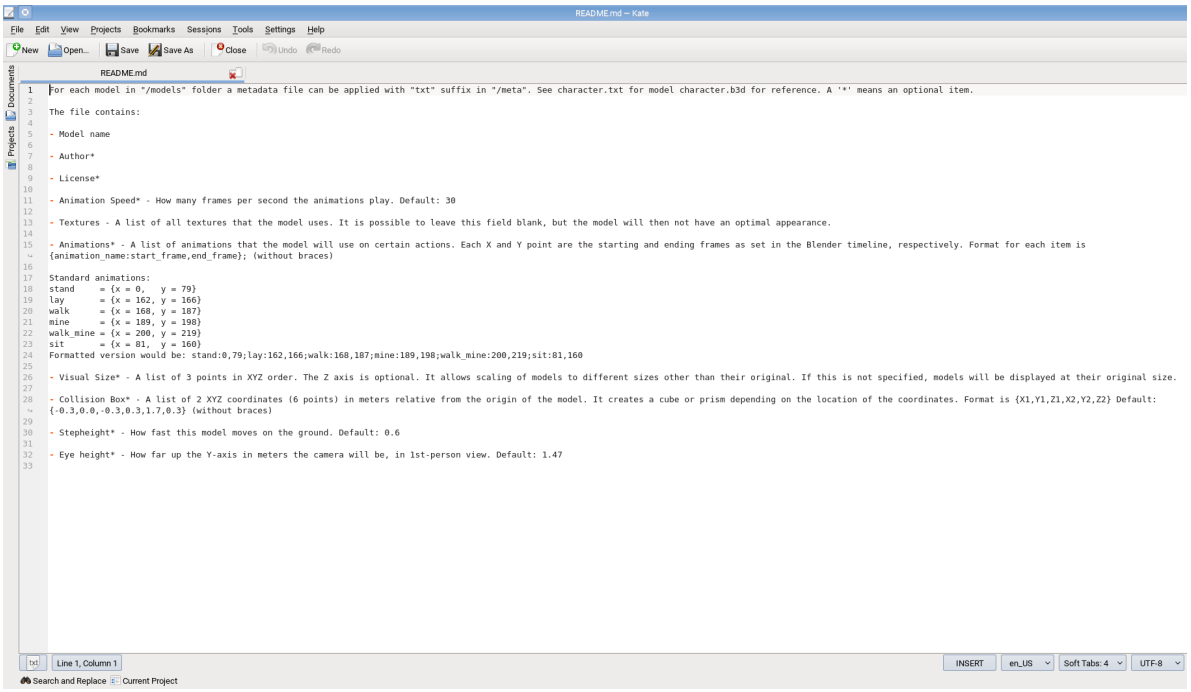


Figure 18: The readme detailing the specifications for metadata files

In order to better support 3D models and not image files used as player skins, I wrote some additional code to parse the additional fields I specified (written in Lua like with all other mods):

```
-- How to parse config files from https://stackoverflow.com/q/55523750
	local file = io.open(chchar.modpath.."/meta/"..name..".txt", "r")
	if file then
	  print("Found meta file for "..fn)
	  for line in file:lines() do
	    if line:find("name = ") ~= nil then
	      model_obj:set_meta("name", line:split(" = ")[2])
	    elseif line:find("author = ") ~= nil then
	      model_obj:set_meta("author", line:split(" = ")[2])
	    elseif line:find("license = ") ~= nil then
	      model_obj:set_meta("license", line:split(" = ")[2])
	    elseif line:find("animation_speed = ") ~= nil then
	      model_obj:set_meta("animation_speed", line:split(" = ")[2])
	    elseif line:find("stepheight = ") ~= nil then
	      model_obj:set_meta("stepheight", line:split(" = ")[2])
	    elseif line:find("eye_height = ") ~= nil then
	      model_obj:set_meta("eye_height", line:split(" = ")[2])
```

```
    elseif line:find("collisionbox = ") ~= nil then
        model_obj:set_meta("collisionbox", line:split(" = ")[2]:split())
    elseif line:find("visual_size = ") ~= nil then
        local data = line:split(" = ")[2]:split()
        if #data == 3 then
            model_obj:set_meta("visual_size", {x = data[1], y = data[2], z = data[3]})
        elseif #data == 2 then
            model_obj:set_meta("visual_size", {x = data[1], y = data[2], z = data[2]})
        elseif #data == 1 then
            model_obj:set_meta("visual_size", {x = data[1], y = data[1], z = data[1]})
        end
    elseif line:find("textures = ") ~= nil then
        local data = line:split(" = ")[2]:split()
        if #data < 2 then
            model_obj:set_meta("textures", {data})
        else
            model_obj:set_meta("textures", data)
        end
    elseif line:find("animations => ") ~= nil then
        local data = line:split(" => ")[2]:split(";")
        -- Inserting nested tables from https://stackoverflow.com/a/22598242
        local animtable = {}
        for item = 1, #data, 1 do
            local subdata = data[item]:split(":")
            local subnum = subdata[2]:split()
            animname = subdata[1]
            animtable[animname] = subnum
        end
        model_obj:set_meta("animations", animtable)
    end
end
file:close()
```

10.2 Implementation for *auto-explorer* and *building-maker*

As of this writing on 03/04/20, the code for *building-maker* and *auto-explorer* have not yet been implemented. Depending on the way that any existing code I find is implemented, they may build on top of them, or they may become standalone mods. However, I saw that the mod *open_ai*, available at https://github.com/jordan4ibanez/open_ai, has the rudimentary concepts of both of these proposed mods implemented (although mostly undocumented). I knew that trying to redo this functionality would take a while to complete for a solo project, so I just decided to observe the behavior from the mod itself, editing any code where necessary. The following actions led me to discover the similar behavior:

- The mobs could already walk around randomly, but looking at the code, I saw that they can generate a path to players who hold certain items, node-by-node (see Figure 19 for action).
- Editing the AI behavior to pick a random number from 0 to 2, not just from 0 to 1, in order to enable mobs to access building functionality. This causes them to build structures based from schematic files, node-by-node, bottom-to-top (the code can be seen in Figure 20).
- "**Schematics** are pre-defined node patterns to be placed somewhere in the world. They allow to create some complex figures or structures and repeat them with little random alterations. A schematic tells in an area what nodes should be created, with a given probability for each node to appear." From https://dev.minetest.net/schematic. By default, the only schematic shipping with *open_ai* is a jungle tree. After I added some more schematic files (purely consisting of various kinds of trees and bushes, no man-made structures), and adding additional code to randomly pick from that list of files, the mobs seem to build them just fine (see Figure 21 for action).

Below follows more Lua code that reads a random schematic file that the mobs will then use:

```
--inject a schematic for what it will build, else don't execute
function ai_library.build:load_schematic(self)
    if self.schematic == nil then
        local schem_dir = minetest.get_modpath("open_ai").."/schematics/"
        local schem_list = minetest.get_dir_list(schem_dir)
        local building = schem_list[math.random(1, #schem_list)]
        print("getting schematic "..building)
```

```
            local str = minetest.serialize_schematic(schem_dir..building, "lua",
{lua_use_comments = false, lua_num_indent_spaces = 0})..." return(schematic)"
            self.schematic = loadstring(str)()
            self.schematic_map = {x=1,y=1,z=1}
            self.schematic_origin = table.copy(self.mpos)
            self.schematic_size = schematic.size
            self.index = 1
        end
end
```

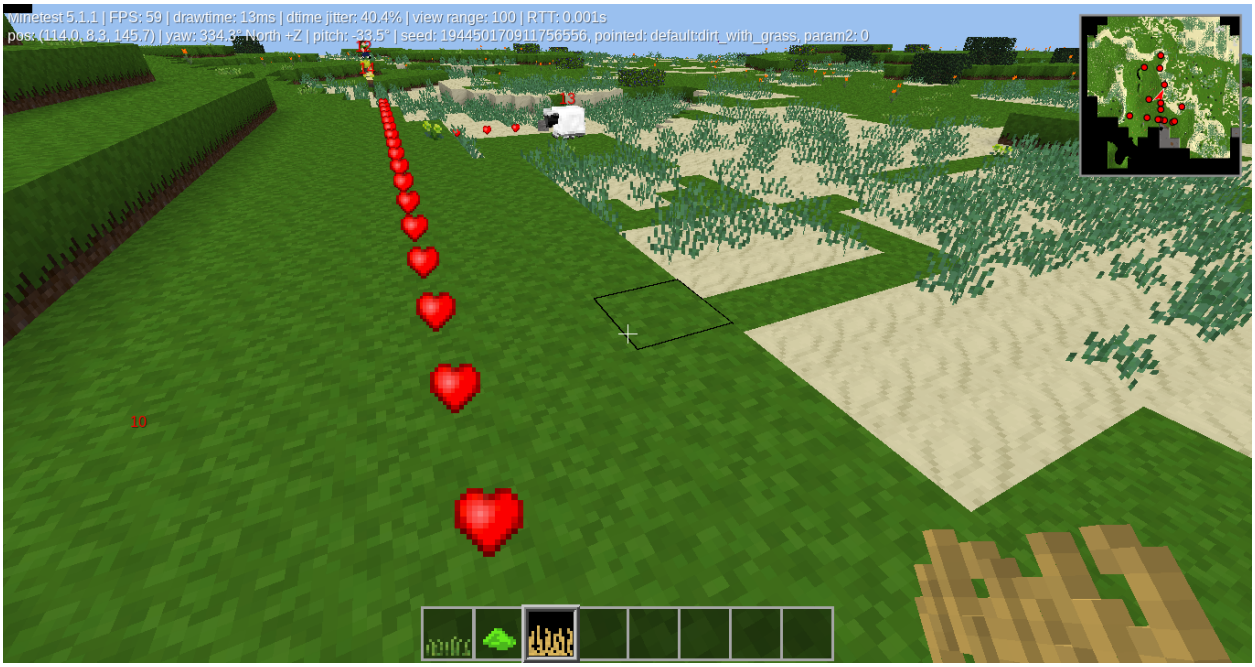


Figure 19: A nearby sheep mob walking towards the player holding dry grass

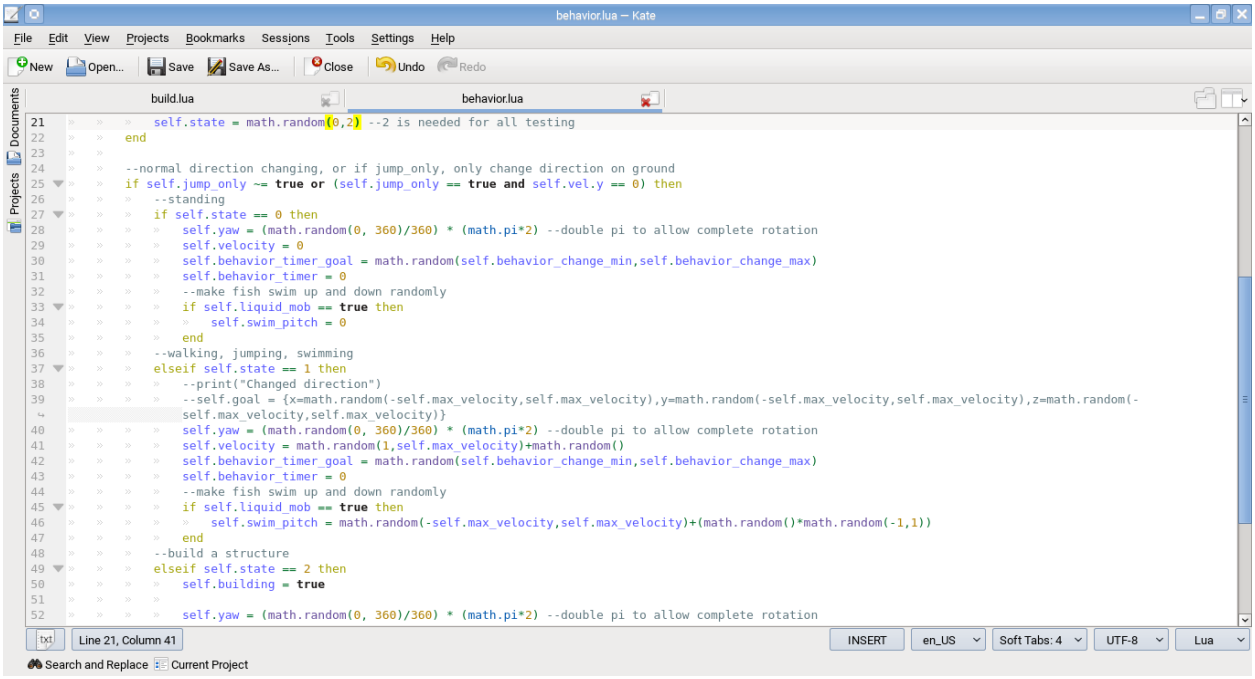


Figure 20: Changing the behavior to enable mobs to access building functionality

Figure 21: Various mobs building different kinds of vegetation, although randomly

Chapter 11: Test Plan and Test Results

Chapter 12: User Operation Manual

Requirement on system running environment; operation manual; snapshots, etc.

Chapter 13: Conclusions (SD II)

References

Appendices