BirchBot is an automated controller for directing an agent to repetitively harvest birch tree farms in the popular computer game Minecraft. The controller handles tree harvesting, replanting, axe crafting, and inventory management. We present a robust status system for handling errors during execution, and a remote command-and-control system accessible within the game by other accounts. Finally, the script supports arbitrary spatial layout of birch farms by our novel "block-based command system", which allows users to specify bot actions by embedding certain blocks into the floor of a farm.

Implementation is achieved through a scripting language built into to the Macro/Keybind mod. BirchBot is a member of a family of automated Minecraft controllers, all which share a common library of utility scripts and code style.

COMPATABILITY

BirchBot was found to be compatible with both Windows (7, 10), and Linux (Ubuntu). macOS was not tested. The popular mod framework Forge was also found to be compatible. The scripts themselves are compatible with Minecraft 1.12.2.

INSTALLATION 1.12.2

The prerequisites of this tool are:

1. Liteloader 1.12.2
2. Macro/Keybind Mod 1.12.2
3. CML (Common Macro Library)

The first two may be found online, and links are not provided here to prevent link rot.

After installation of the above, the Windows macro folder is "%appdata%/.minecraft/liteconfig/common/macros/". The provided .txt files beginning with "birchBot" must be placed in the macro folder.

CML is a common collection of scripts that build a useful framework for developing robust automated controllers. The .txt files in CML are directly placed in the macro folder. A copy of CML is provided with every distribution of automated controllers.

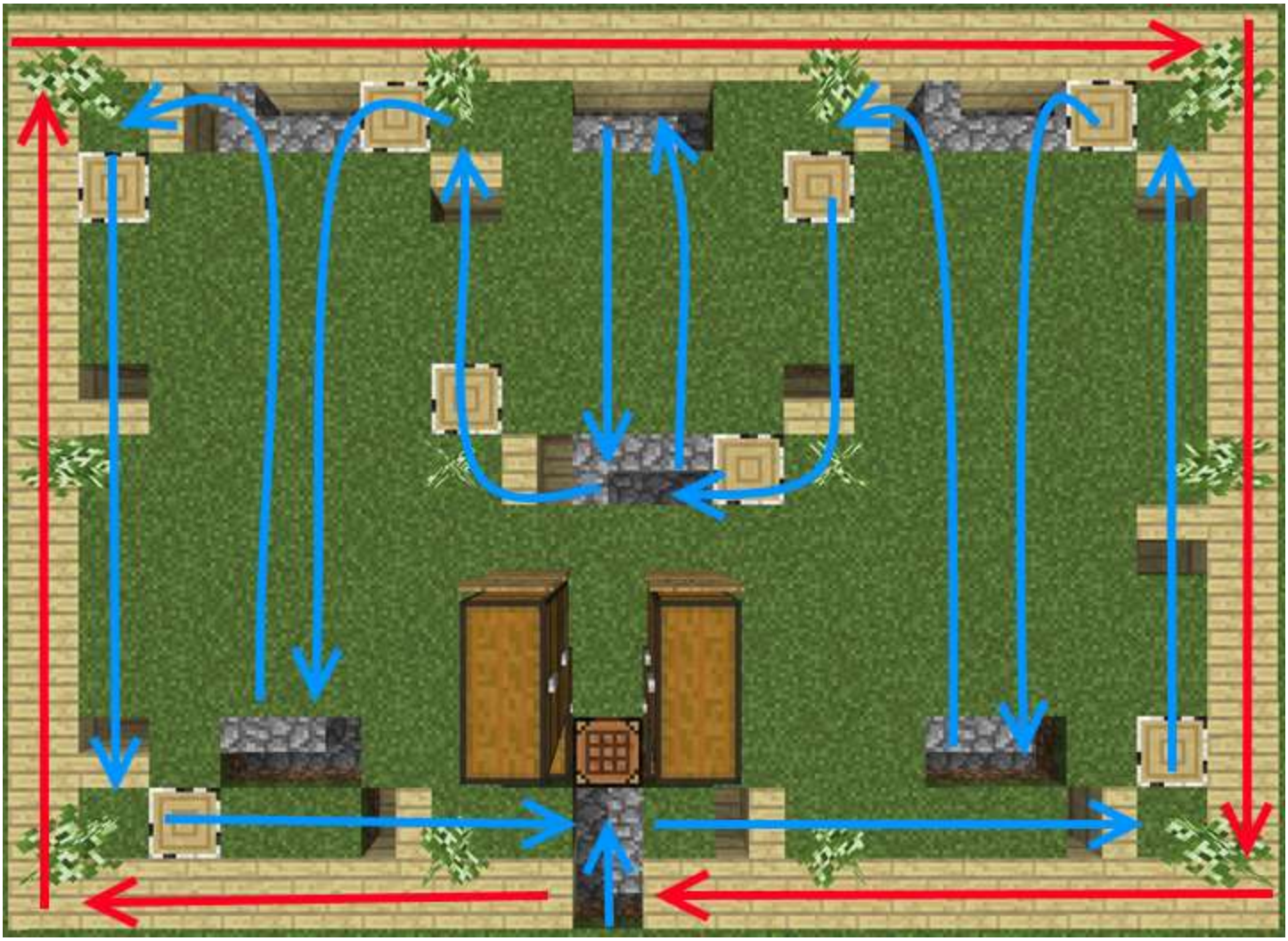
After installation proper, two triggers must be manually placed inside the in-game Macro/Keybind menu. Assign the following two triggers:

1. To any key, assign "$${$$<birchBot.txt>}$$
2. To the event "onJoinGame", assign "$${$$<handleCommands.txt>}$$

For detailed help assigning these two triggers, refer to Macro/Keybind usage documentation found online.

FARM LAYOUT

We support arbitrary farm layouts by a novel block-based command system. The user must embed certain control blocks into the floor of the farm, that the bot will read in order to decide the next action. What follows is a visual standard implementation. Note, that cobblestone stair elements may be replaced by their stone brick stair counterparts for aesthetic considerations.



The control blocks are as follows:

1. **Log**: Marks the direction bot should turn to after a tree was just processed. Applies to bots that are adjacent to it and just finished processing a tree.

2. **Cobblestone Stairs / Stone Brick Stairs:** Turns bot towards direction pointed, as if to "walk up" the stair.

3. **Birch Stairs:** Instructs bot the pointed direction is a tree/sapling place and it should be processed as thus.

4. **Crafting Table:** When adjacent to a cobblestone stair / stone brick stair, instructs the bot that this is a "home position" with corresponding chests. This is the position that inventory management and axe crafting is done.

5. **Planks:** Directs bot to follow a circuit of planks until a non-plank block is encountered. This is used to enforce a boundary, so even in the case of glitches the bot will reset itself and get back on the path. **(Red path)**

6. **Crafting Table:** Directs the bot to turn towards a adjacent stone stair block, if one exists. This is useful in alternative chest orientations where the chests are pointed towards the outside of the farm.

Note that the above farm may be arbitrarily customized, as long as the rules for control blocks are considered. Using this structure, the farm layout can be extended to arbitrary scales and shapes. For large farms, an arbitrary number of "home positions" may be specified.

As well, note the extraneous stone stair blocks. These serve to direct the bot to patrol a circuit that includes the center grass regions, so that saplings which drop there may be picked up.

CHEST LAYOUT

The bot expects 6 double chests located around the crafting table in the following way. The contents of the chests are specified below as well, with the "input chest" highlighted in blue.

Note that before initially using the bot, either an initial axe should be placed in the input chest, or sufficient cobblestone and logs should be placed in their respective chests as materials for axe generation. The bot will look in Logs 1 first to find materials, and if empty will assume the other two are empty.

Since the axe crafting is performed in batches of four (i.e. 1 log of sticks), a minimum of 12 cobblestone is required for axe generation. In production, many stacks of cobblestone may be provided and the bot will continue running until it runs out. or some other end-condition occurs.

USAGE

On triggering the BirchBot script, the bot will walk forward until a control block is found. It will then follow the layout specified. Customarily, it is usual to begin at the "home position", i.e. the stair block adjacent to the crafting table, but not required.

After started, commands may be sent to the bot to inquire about status, to pause/unpause the bot, and more. See "Messaged Commands" for stipulations. The bot will continue running until an "end condition" or "error" is encountered.

END CONDITIONS / ERROR HANDLING

Even working nominally, the bot cannot run forever without some external materials. Namely, these two materials are food and cobblestone. Because food is optional, we don't define a true end-condition for running out of food (on normal-difficulty servers, the bot will not die). Even if the bot does not run out of materials, it may run out of chest space for products harvested.

So, we define the following end-conditions for BirchBot:

1. **Out of logs:** Occurs when there are no axes and no wood to build axes.
2. **Out of cobblestone:** Occurs when there is no cobblestone to build axes.
3. **Logs chest full:** Occurs when all three birch log chests are full.
4. **Panic:** Occurs on various non-optimal unrecoverable situations, such as being underwater, etc.

The first two conditions will only trigger if there are currently no axes to use. The first condition will (likely) only trigger if the user forgets to supply the bot with seed materials or axes. The third will trigger under normal operation and is the most likely end result in the bot is used correctly.

When an end condition is encountered, the bot becomes **paused**. The bot halts all activity and will spin in a circle (to visually notify the effect) until it is manually resumed using the command /resume. If the option "logtochat" is set true (as is default), then an end condition will be notified in chat publicly.

Errors not defined here, such as being attacked, falling in water, chests not being present, lava, etc. result in undefined behavior. It is not really possible to exhaustively enumerate and respond to all of the crazy fail-mode situations possible in Minecraft, so we define "Panic" as a catch-all for things that should never occur. Here are (some) of those cases:

1. Health drops below a certain threshold
2. Our y-coordinate does not match the y-coordinate we started with (e.g. fell into a hole)
3. Our x/z positions are suddenly changed, as if we were teleported

By default, a panic condition becoming true results in an immediate disconnect. The "disconnectonpanic" option may be set to false in order to merely stop the script.

MESSAGED COMMANDS

The bot supports the following commands, whispered to it via the /msg command:

1. /help — List of commands
2. /status — Whether the bot is currently paused, unpaused, or idle (not running a script)
3. /identify — Identity of bot, e.g. returns the name of the script
4. /pause — Will pause the bot, halting all action (after the current task is performed).
5. /resume — Will resume the bot, causing it to take the yaw-orientation it has when it was paused.

Depending on server type, the default command handling regular expressions specified may not be compatible with that server's method of private messages. If that is the case, you may modify "handleCommands.txt" if you are inclined to customize it to your server. The default whispering mode is /msg as in vanilla Minecraft.

VIRTUAL MACHINES / MULTIPLEXING

It is a straightforward extension to run multiple instances of this on multiple Minecraft accounts or use your computer while the bot is running. To do this, the best way I've found is to run instances of Ubuntu on a virtualization software, under different accounts.

Doing this in general allows one to have many different bots performing different actions and will guarantee that actions taken on the machine (e.g. web browsing) will not interfere with the bot, and vice-versa. To do this without lag, it is best to resize the in-VM Minecraft screen until it is very small. This will reduce the rendering requirement and improve performance. As well, Optifine is recommended in this case.