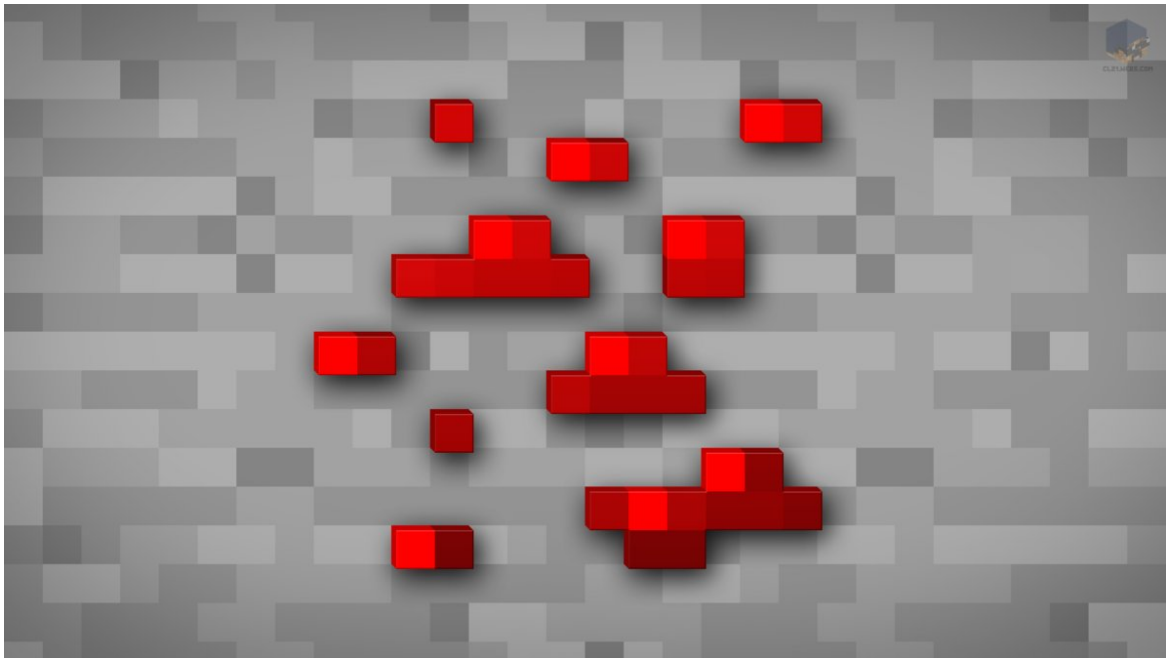


Minecraft U Sequence 2: Redstone and Programming

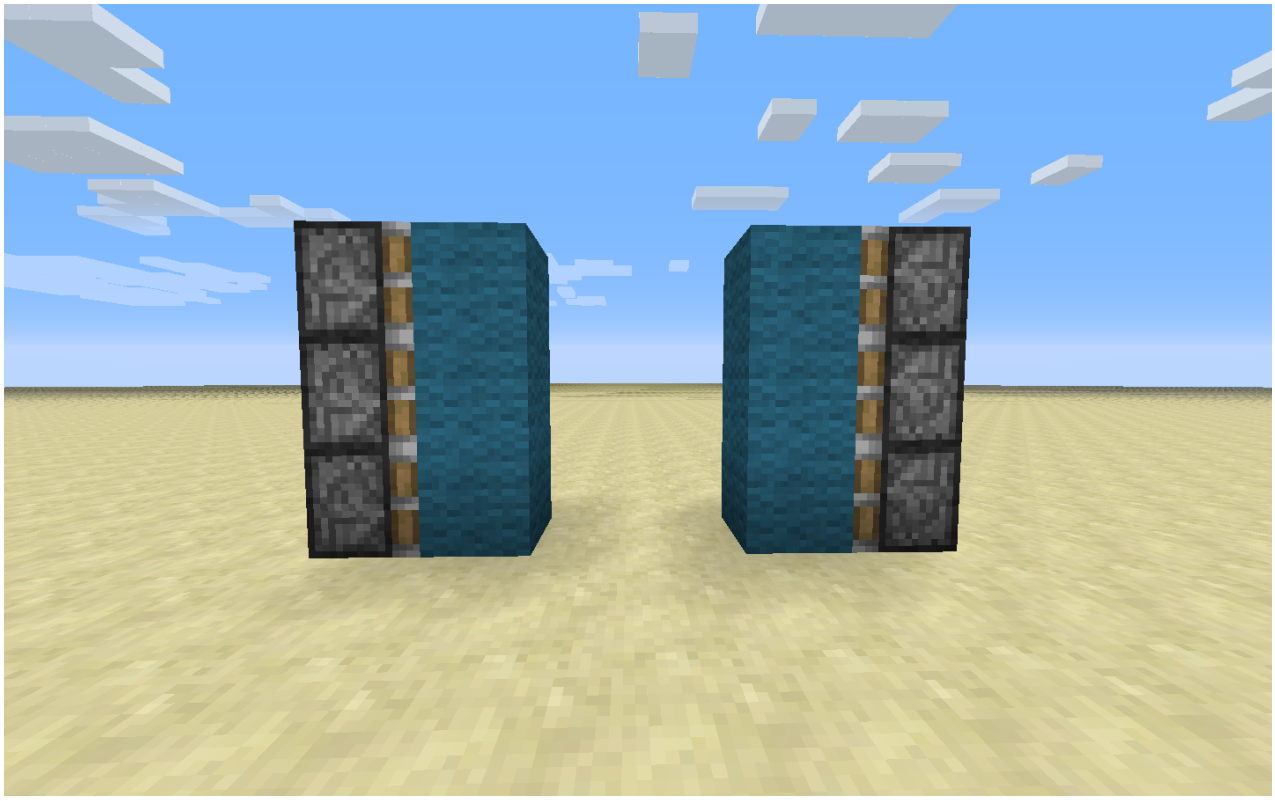
Redstone minecrafters are the electrical engineers of Minecraft, and their creations often amaze and confound more basic players. Command blocks are special blocks in Minecraft that can be programmed to accomplish specific tasks. ComputerCraft is a modification for Minecraft that's all about computer programming. Used individually and together these elements of Minecraft open up elements of programming within the game's unique and engaging environment. Also: cannons!

Advanced Redstone

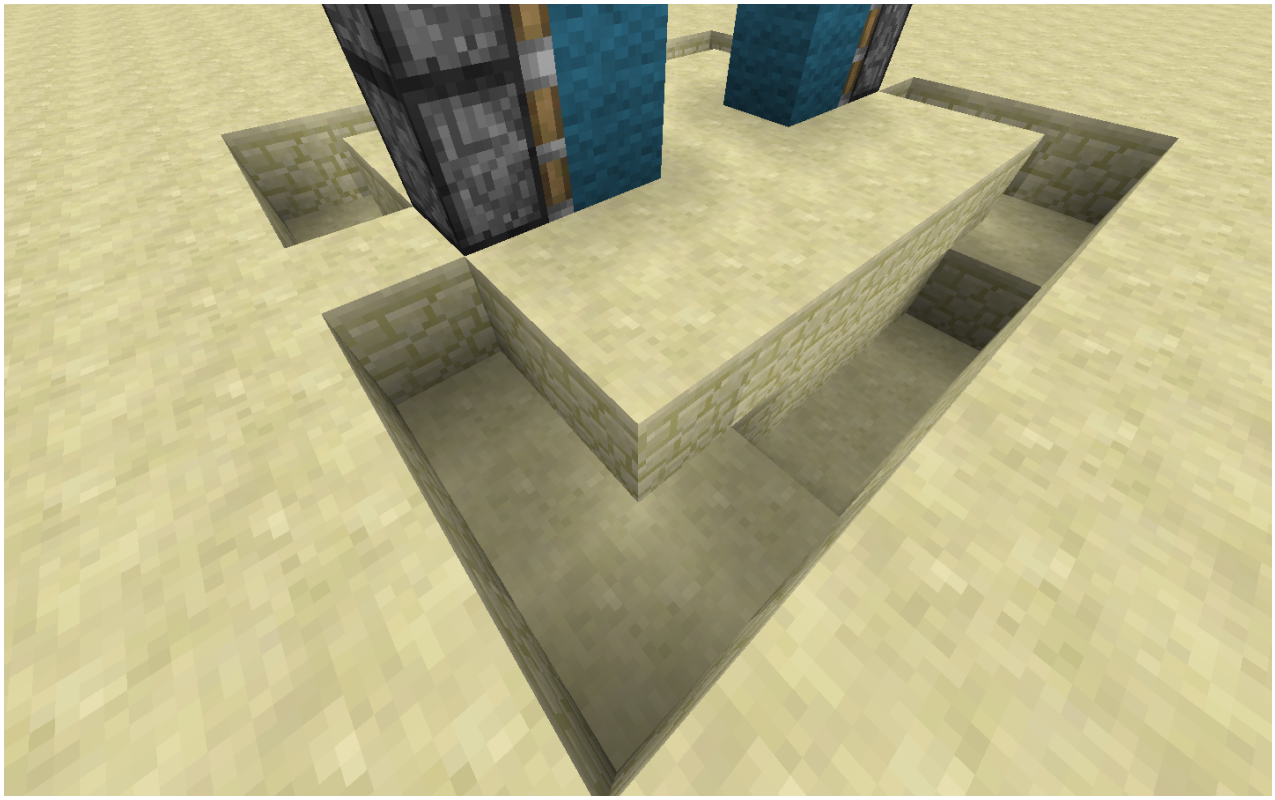
Piston Doors

This section concerns making doors powered by pistons and redstone. Rather than using wood or iron doors, we'll be using pistons to make large doors using blocks. For this you will need redstone, redstone torches, and sticky pistons. Since sticky pistons can both push and pull blocks, they're ideal for making something like a door. The redstone circuit used is simple but still requires some space, which can be hidden inside a wall (especially since doors normally appear in walls!).

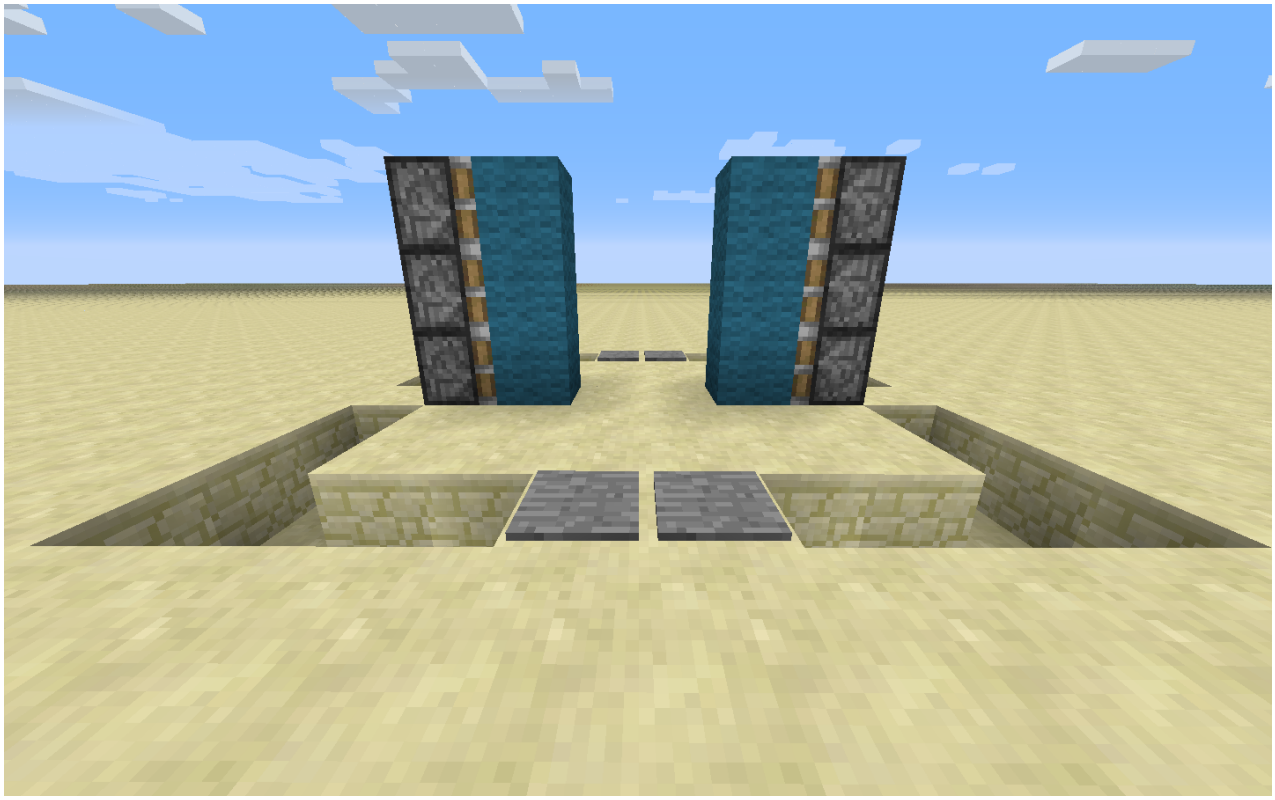
1. The door will be two blocks wide and three blocks tall. Place two columns of three sticky pistons facing each other with four empty air blocks between them. Then place your door blocks (lapiz, gold, etc) onto each of the sticky piston faces. When unpowered, there will be a two-block gap between the pieces of the door.



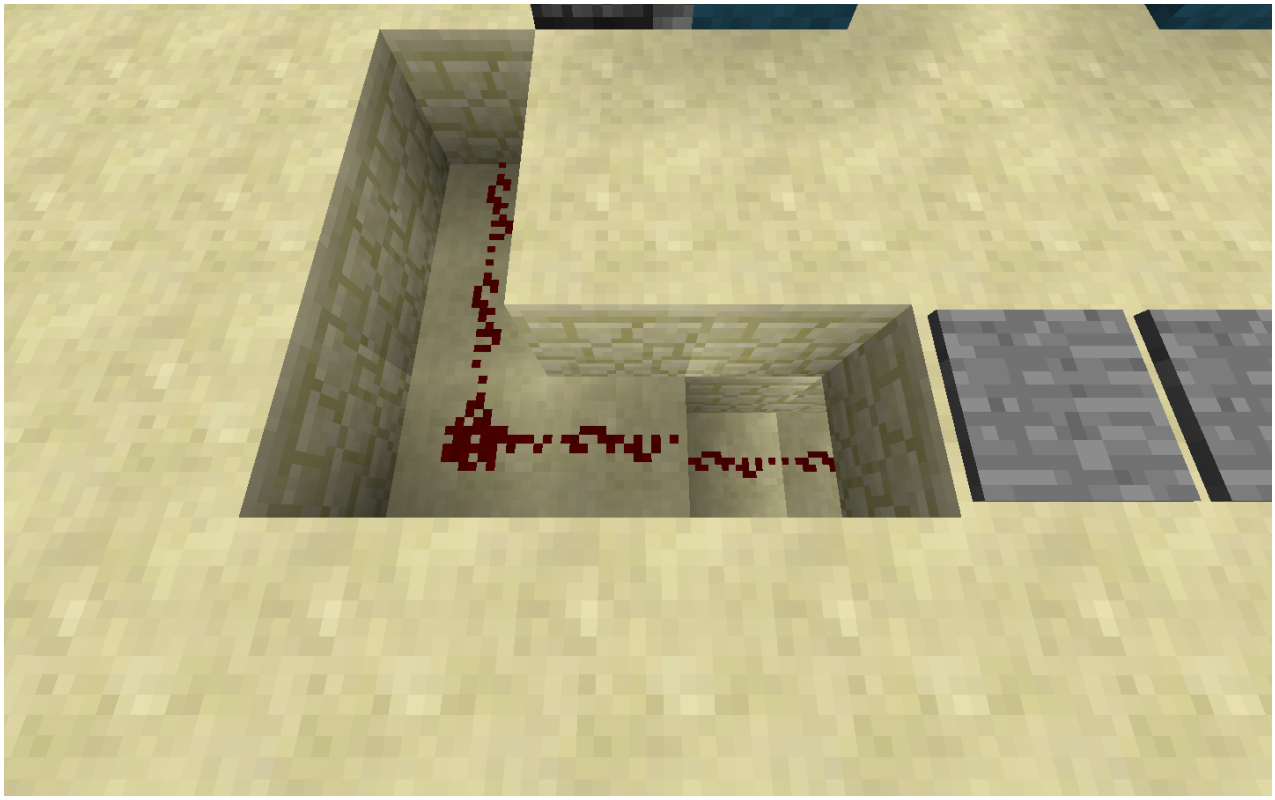
2. Dig a one block deep channel around the doors as shown in this picture. We will hide the redstone we're using to power the doors within this channel. Dig out two more blocks from the front of the door (consult the screenshot if you are confused).



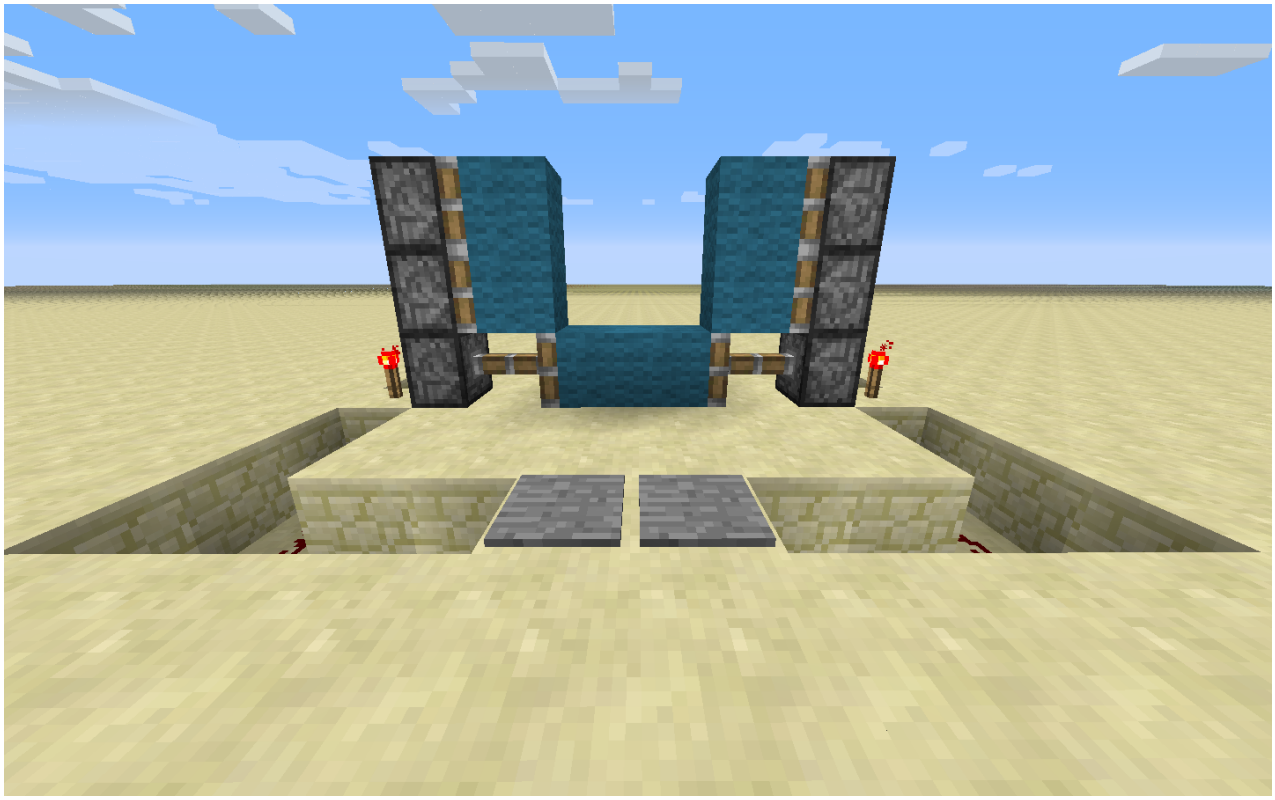
Then put two ground blocks back in the middle (keeping the just-mined block underneath empty) and put two pressure plates on top of them.



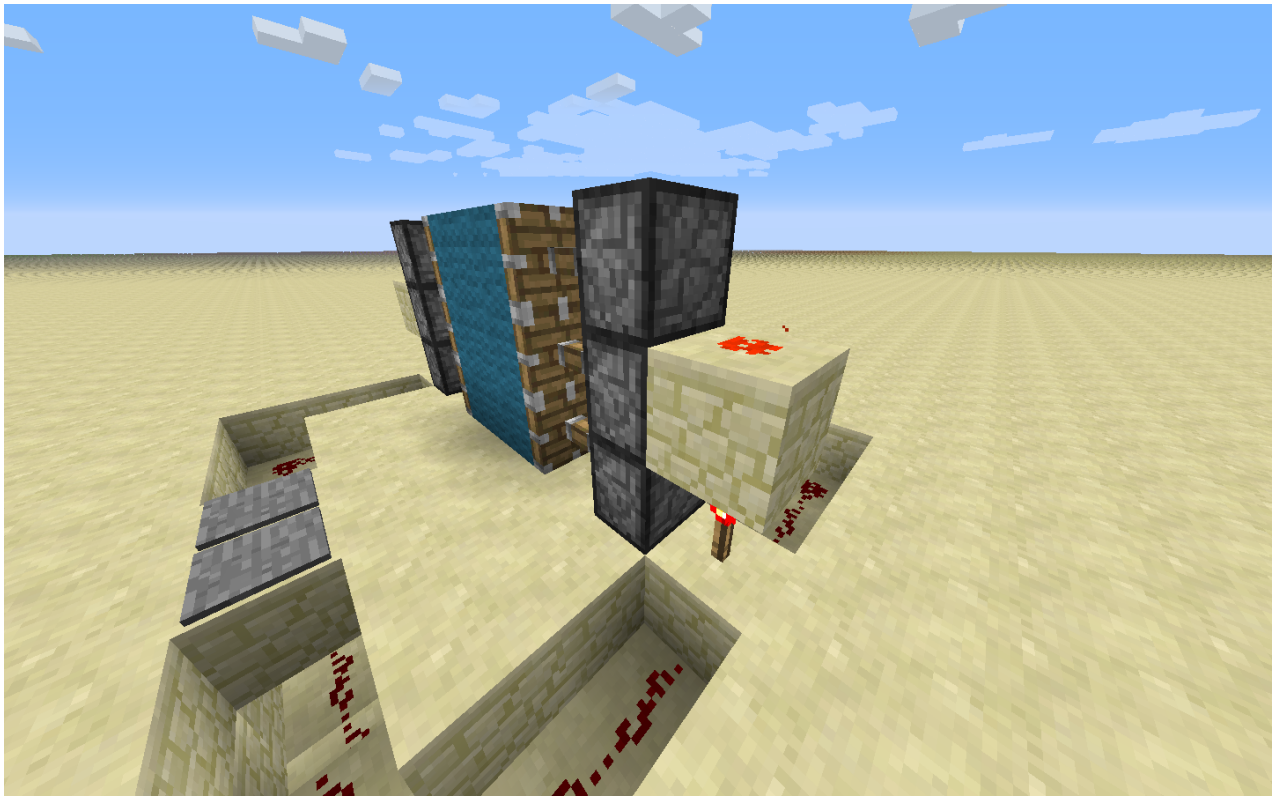
3. Now run redstone from the very bottom space all the way around the bottom of the channel, with the ends pointing into the remaining sandstone block on either side.



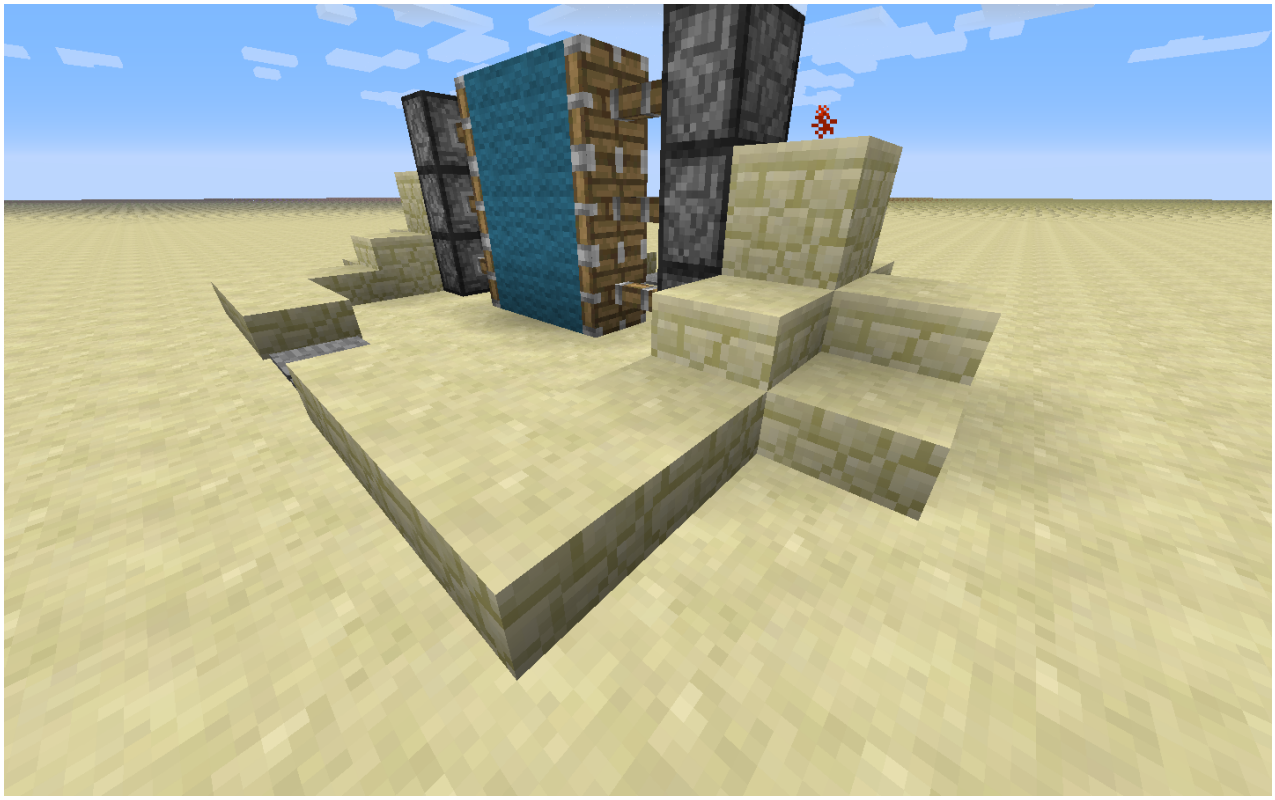
4. Place a redstone torch on top of that remaining sandstone (the torch should be next to the bottom-most sticky piston).



Place a sandstone block on top of that torch and redstone dust on top of that sandstone block. Repeat this for the other side of the door.



5. Cover up the redstone using slabs, which give the best aesthetics. You can also cover up the side towers (pistons and all) with sandstone, wood, or other attractive patterns.



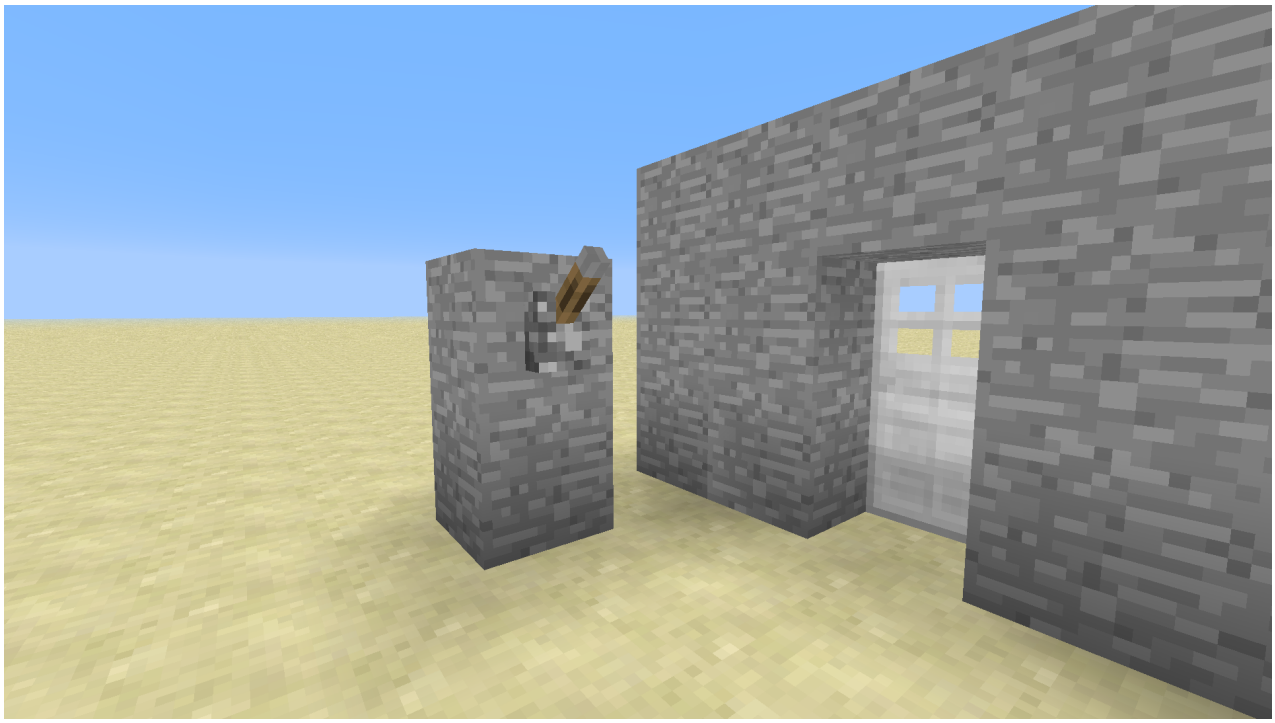
Ultimately, only the two-block space in the middle of the door is what shouldn't be covered. Our suggestion is to build this door as part of a wall where it looks best.

Locked Doors

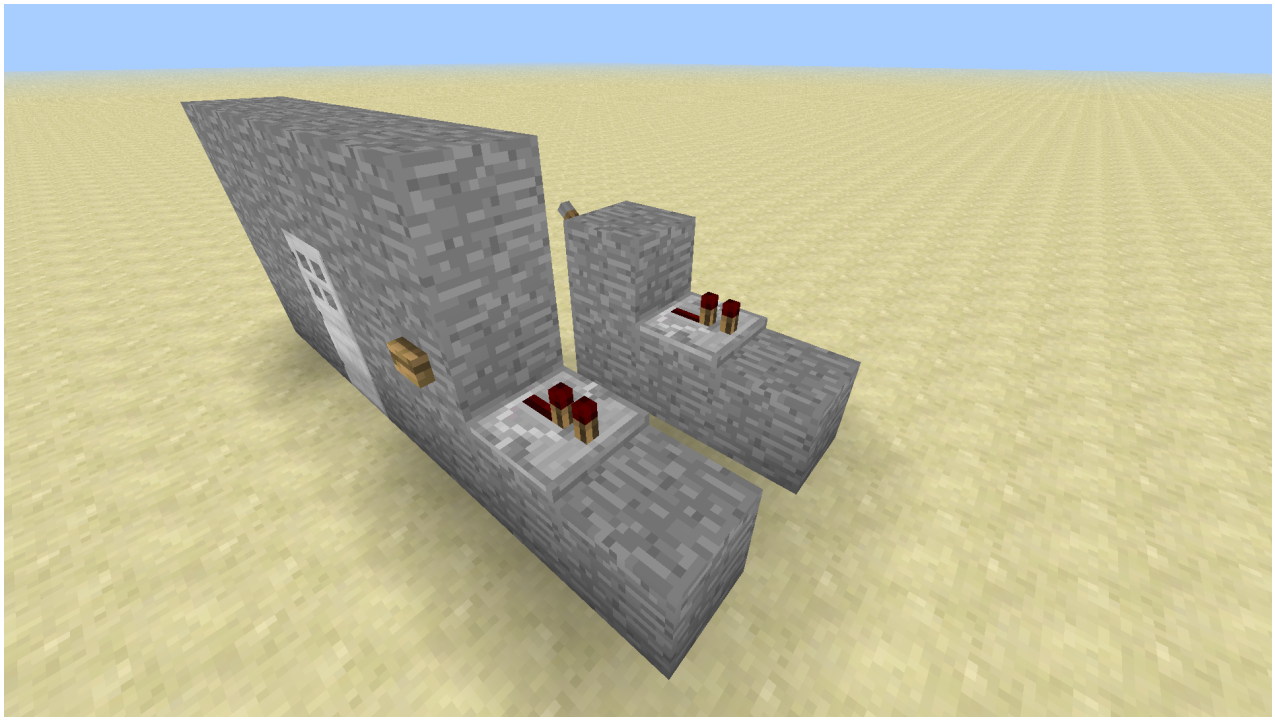
Sometimes you may want to prevent people from entering a door using a lock. By having a lever on the inside, you can prevent the door from being opened unless the lever is in a specific state.

1. Place a door and build a wall around the door. A button should be placed one block away from the door; its block cannot be touching the door or it will open the door every time.

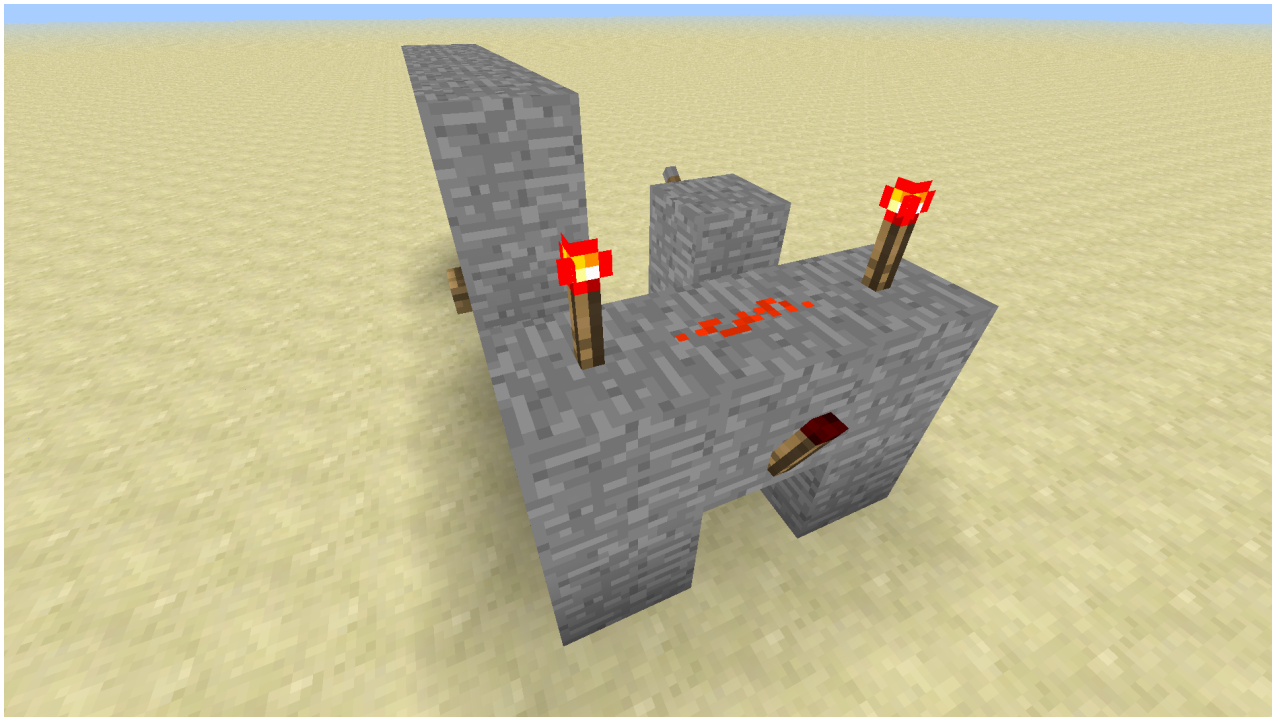
2. The location of the lock lever inside the door.



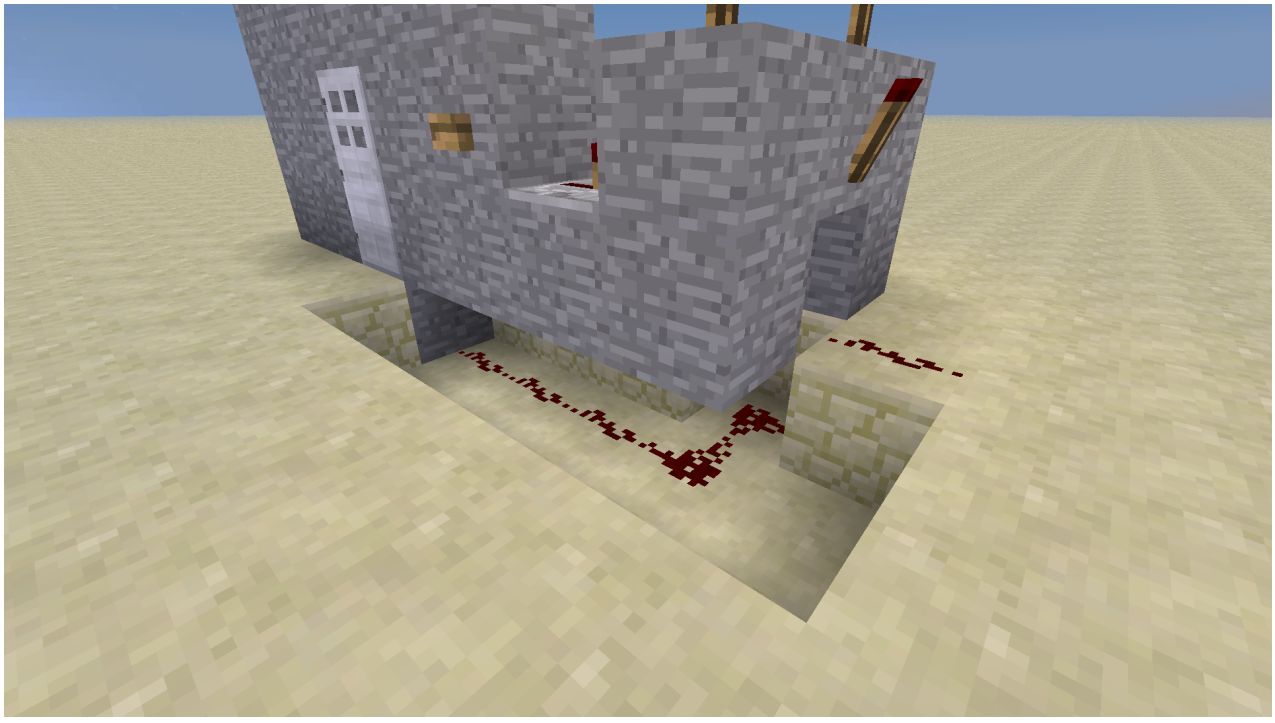
3. Repeaters that carry the redstone signal from both the button and the lever. Be sure to place the layer of stone underneath them so they are powered by the button and lever.



4. An AND gate. The two inputs are the lever and the button, and the output goes to the door. The output will only be ON when both the button and lever are ON as well.



5. The redstone that carries the output of the AND gate to the block directly underneath the door. Now, the door will only open when the lever is set and the button is pressed.



Traps: Design and Execution

There are a few key features that every good trap must have. They must avoid suspicion and lure in victims with chests or rare items. They should have an entertaining or unique mechanism, since in the end all Mincraft traps are purely for entertainment. A trap has three main parts: the trap mechanism, the trigger, and the bait.

1. The first thing to consider when making a trap is the trap mechanism itself. How will you ensnare the player who has been snooping around your house? Obsidian can make for very strong traps as it requires diamond (and lots of time) to break; if the trapped player doesn't have a way to break obsidian, they're basically stuck! Water and lava traps can slow and even kill a player, but are significantly harder to conceal. For our trap we will be using stone and water. Stone is easier to obtain than obsidian but is still somewhat difficult to break, especially when the player is under the water we will be dropping on their head!

2. Triggers can take many forms. Tripwires and pressure plates are the most obvious, and you can place them in various ways to avoid detection (for example, only putting wood pressure plates on wood of matching color). However, buttons and levers can also be used. You can make it appear as if a button opens a specific door, when in reality it activates the trap. Levers can be used for dual purposes. For example, you may have 3 levers that act as a combination lock for a door. One combination will open the door, while incorrect combinations will trigger a trap for the trespasser. Our trap will use a trap chest as the trigger.

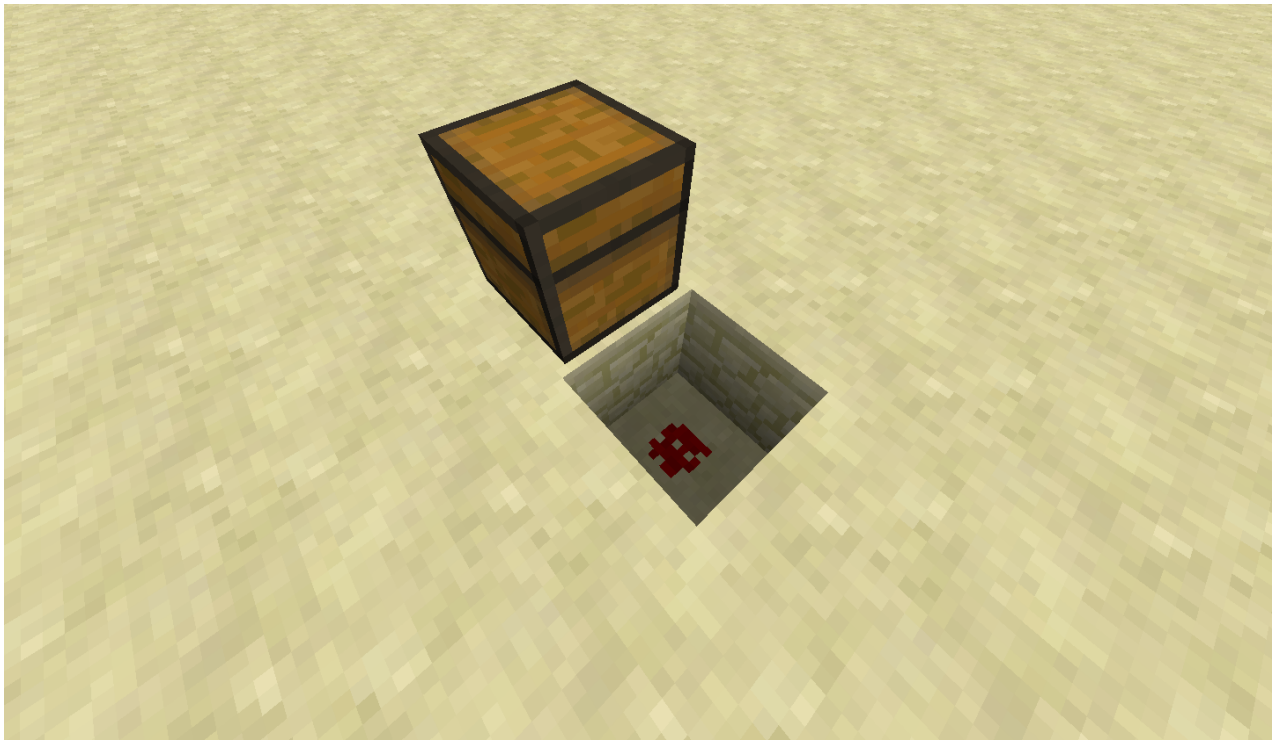
3. A trap chest filled with valuables is serving as the bait for this trap. However, interesting rooms, rare blocks, and even mine entrances can be used as the bait. As long as the object has something that a player would desire, it can serve as bait for a trap. Protective traps don't really have true "bait" as the trap is not supposed to be luring players in; rather, the item they are protecting is the bait in some sense. Our bait is the valuable items we will be placing inside of our trap chest.

Now that you've learned about the basics of traps, we're going to build an example trap using a trap chest, water, and pistons.

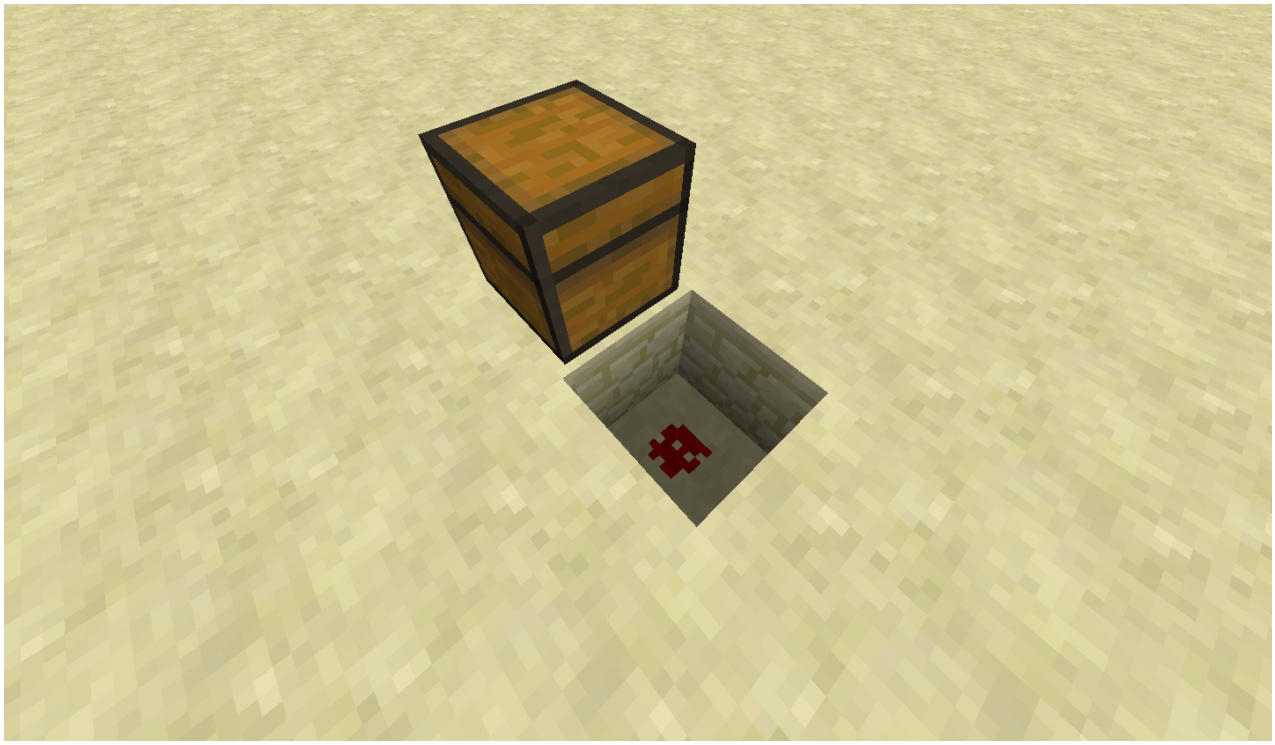
Trap Chests

Trap chests emit a redstone signal whose power is based on the number of people who are opening the chest. We can use this redstone signal to spring a trap designed for people opening the chest. It will open a flow of water when they open the chest, and it will not stop until a reset button has been pressed.

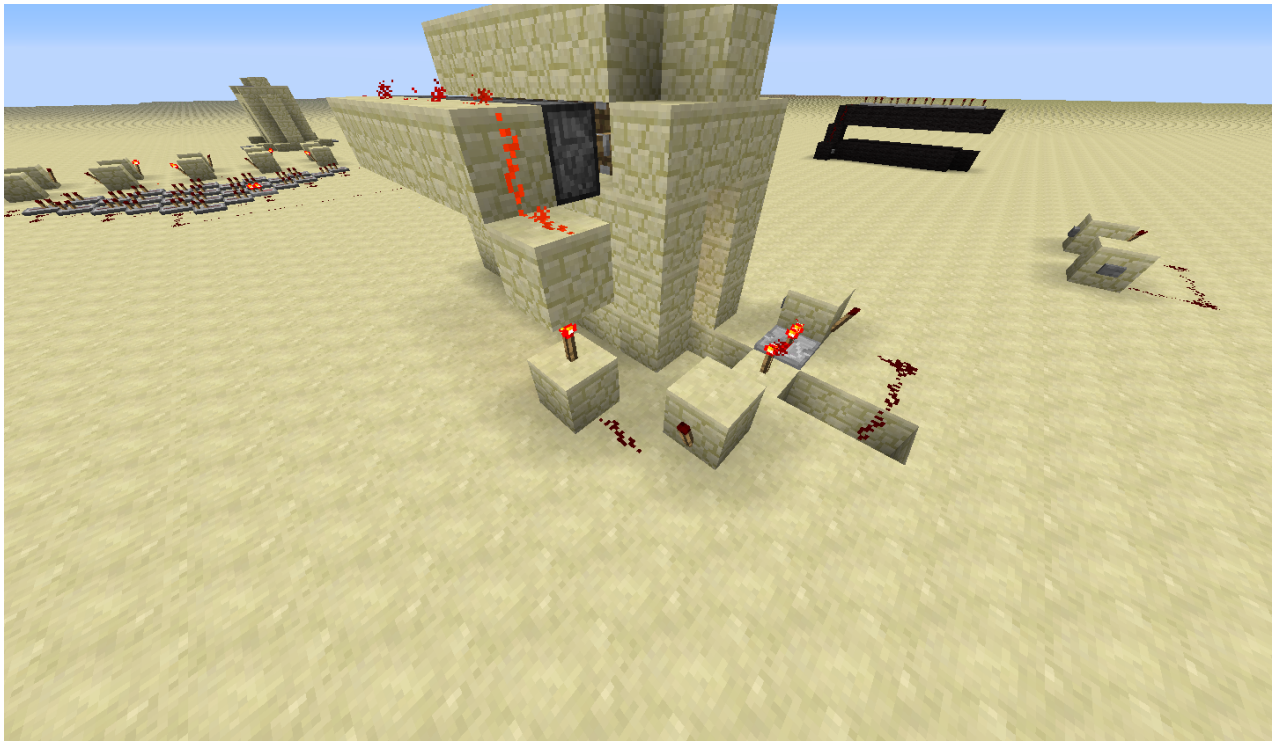
1. Place a trap chest somewhere. They're made out of a regular chest and a tripwire. Dig a whole behind the chest and place one patch of redstone dust.

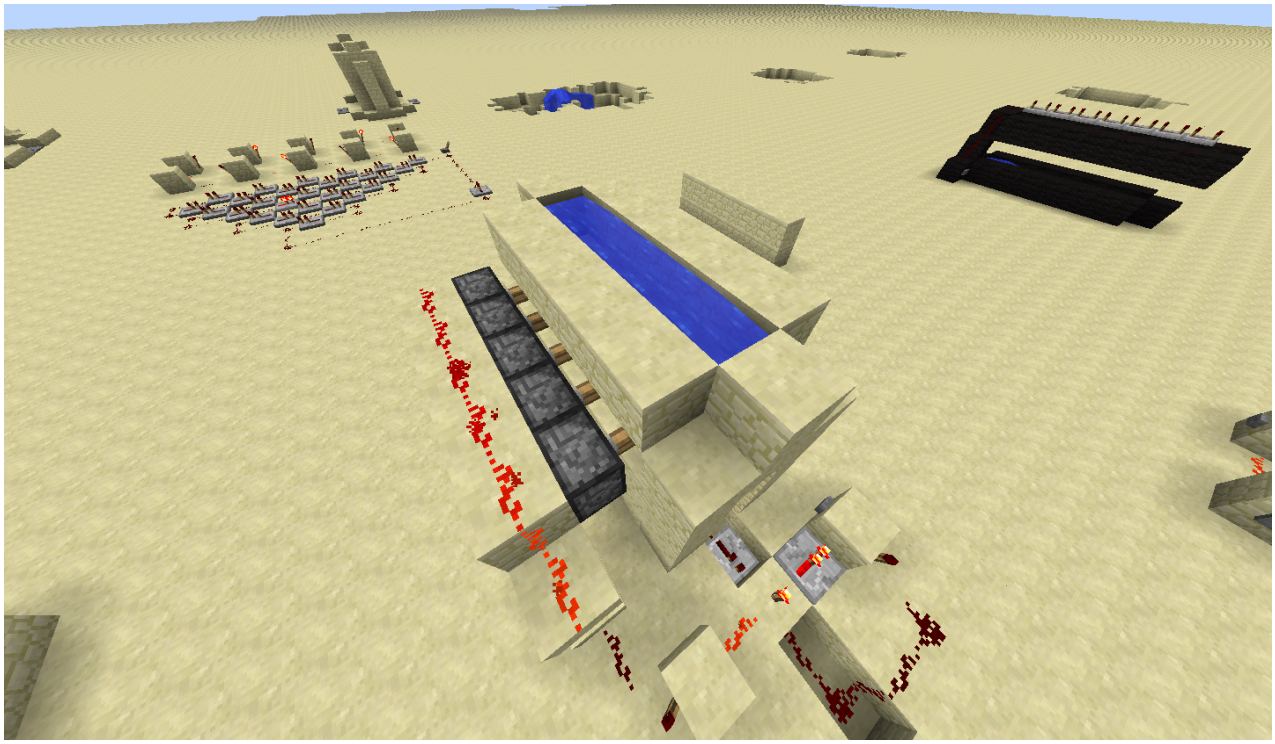


2. Dig a second hole and place a redstone repeater. Since the redstone power is proportional to the number of people opening the chest, this trap will trigger if just one person opens the chest.

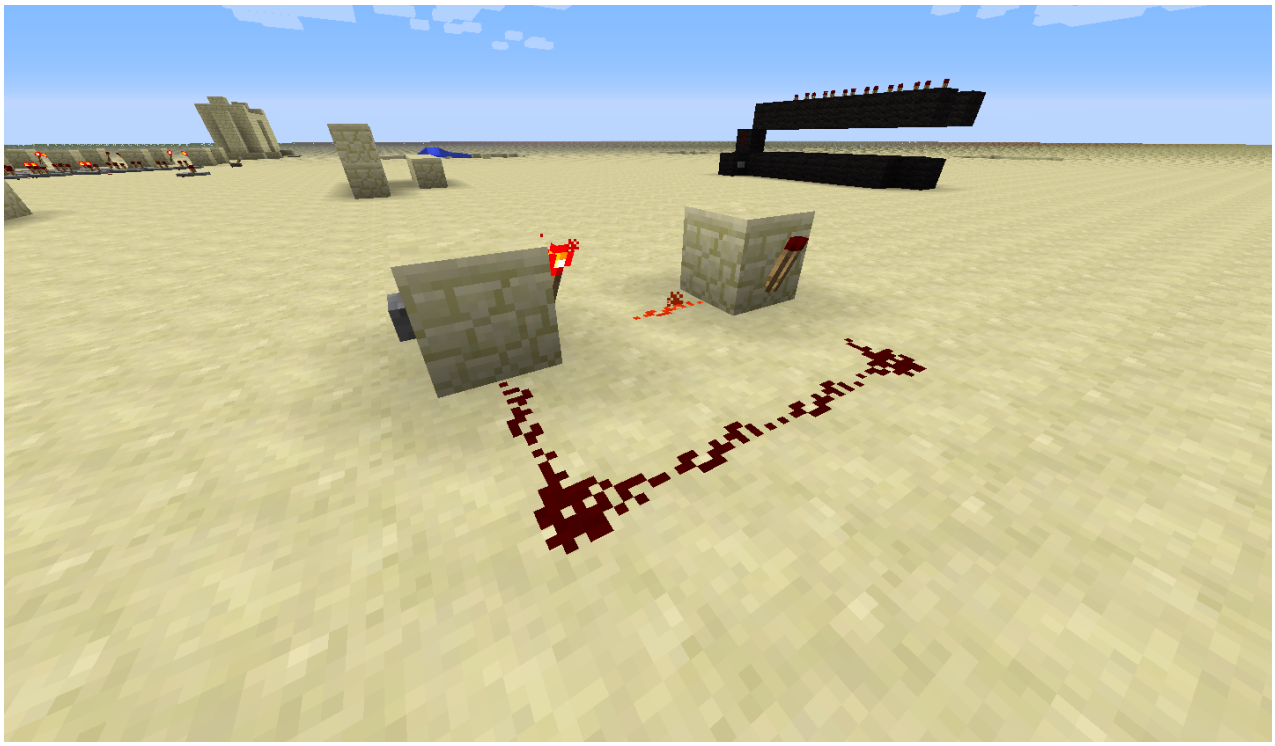


3. The wiring that will make the water gate open when the chest is opened. It’s a bit tricky but further down there are additional pictures for you to follow.





4. This is the “one-way gate” that prevents the water from turning off even when the trap is closed. When the button on the left is pressed, the first torch is turned off and the second one turns on (since the first one is no longer disabling it). Now the second torch is disabling the first torch as its redstone leads into the block of the first one. Toggling the button (or chest in our case) will not alter the state of the switch.

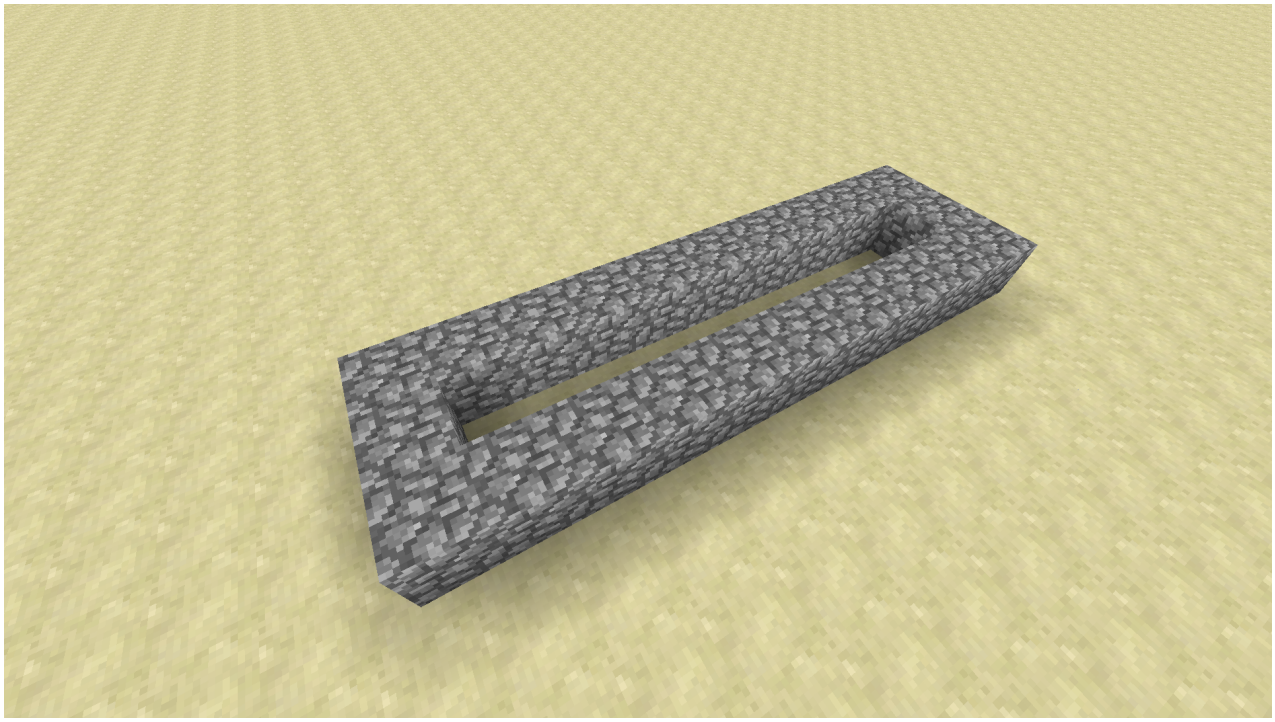


Try to think of some other ideas for traps that could be fun or useful. If we have time left over, feel free to experiment and share ideas with each other.

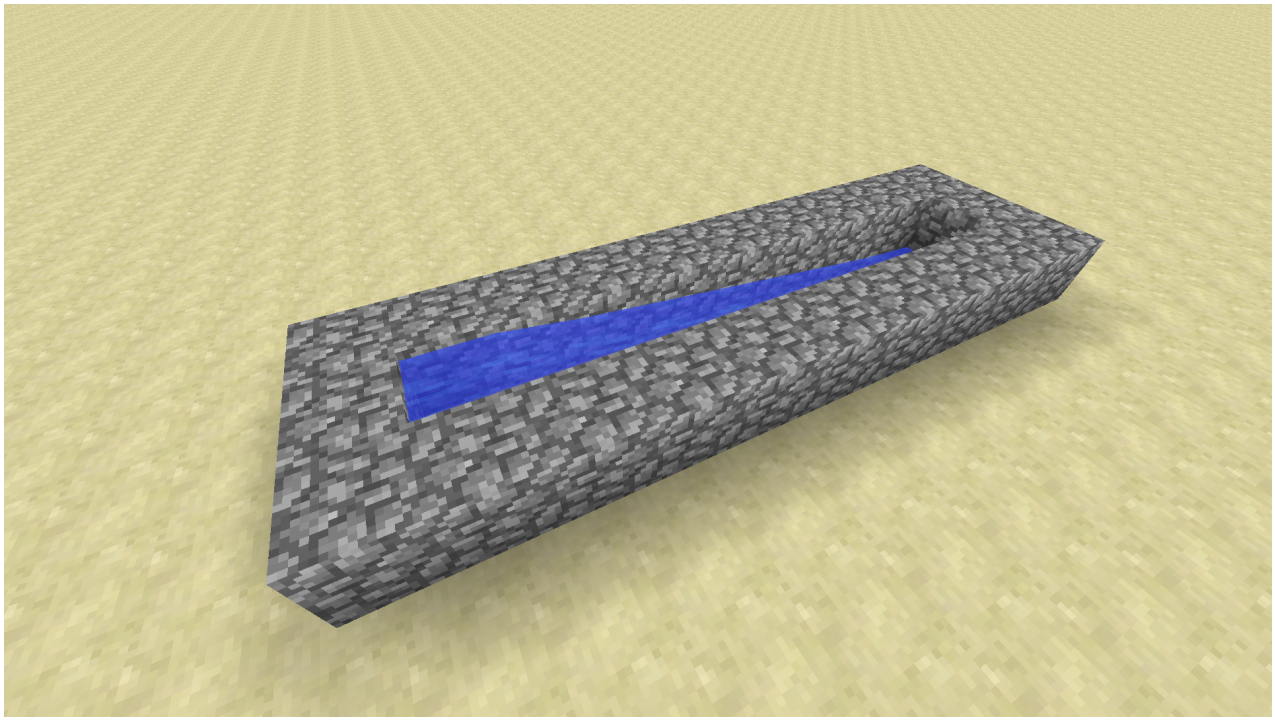
Cannons

By exploiting some of the physics of Minecraft, we can make cannons that fire TNT! As you've probably seen, explosions normally destroy blocks and throw items. Explosions in water, however, will not destroy blocks but will still affect items. Lit TNT is one item that can be thrown, and so we can use TNT and water to shoot a lit TNT block out of our cannon. We will be using redstone to ignite our TNT.

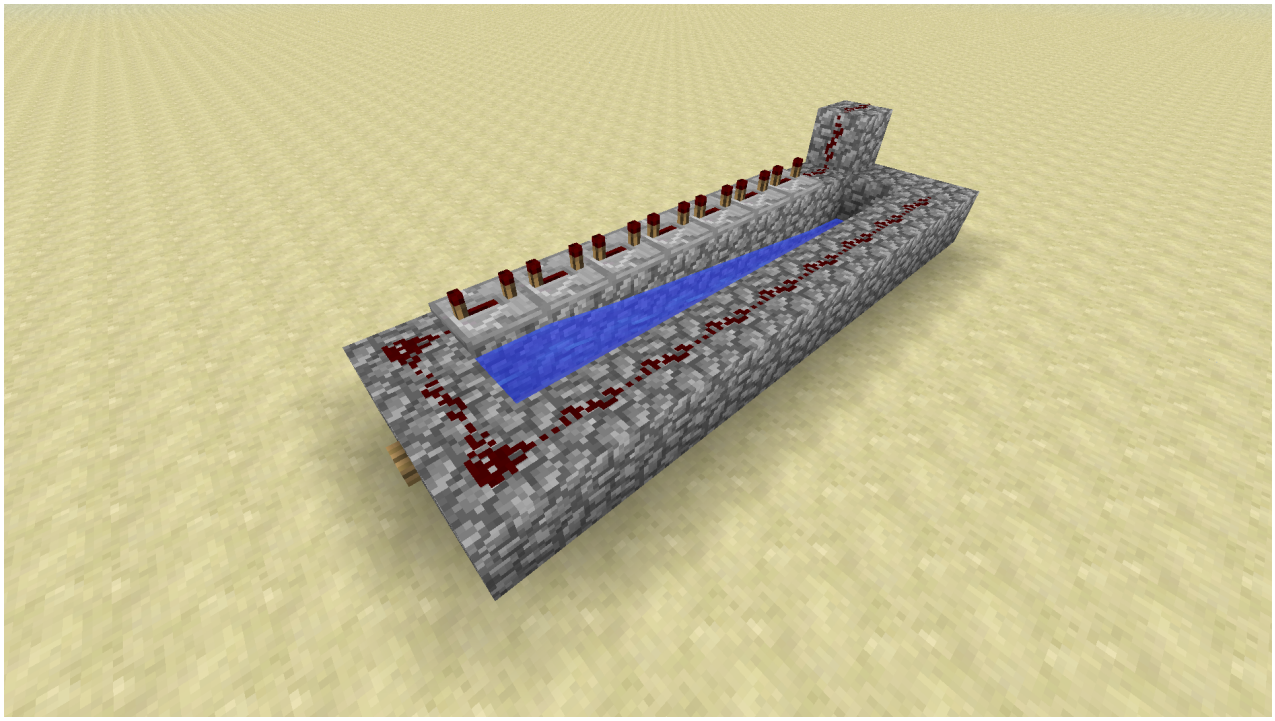
1. Set up the body of the cannon. Cobblestone is what we will be using, though most normal building blocks will work.

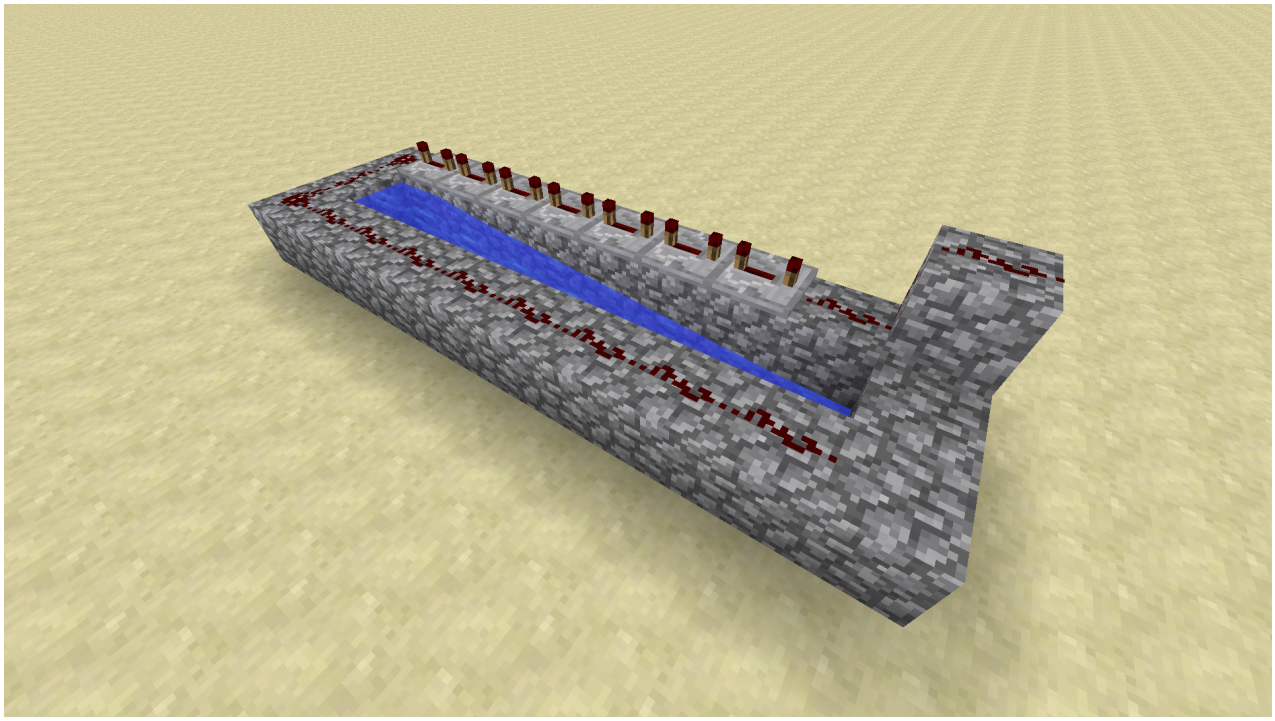


2. Fill the body with water, which will protect the body of the cannon from the blast.

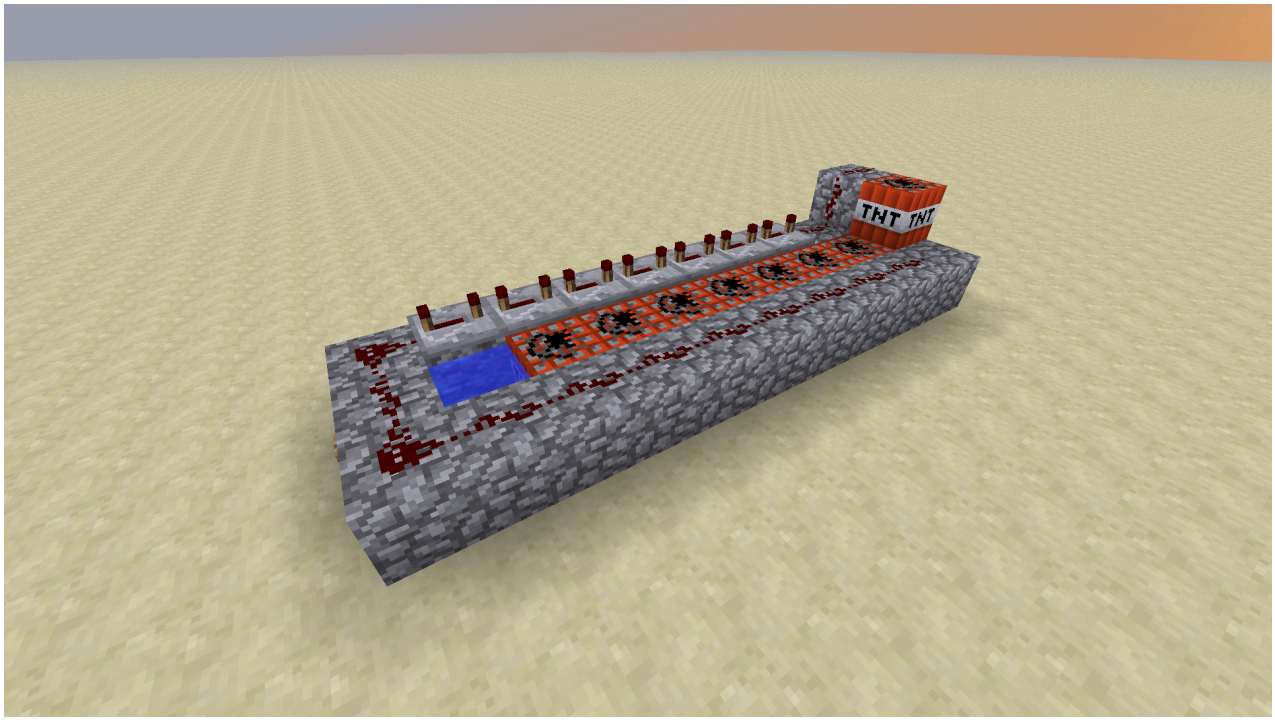


3. Place our redstone and repeaters as well as a button to trigger them. The trail of redstone will ignite our propellant TNT. The repeaters will ensure that the projectile TNT is lit right before it fires.





4. To fire the cannon, we will fill the body with TNT. Be sure not to accidentally place TNT over the source block of water. Then, place the projectile on the very end.



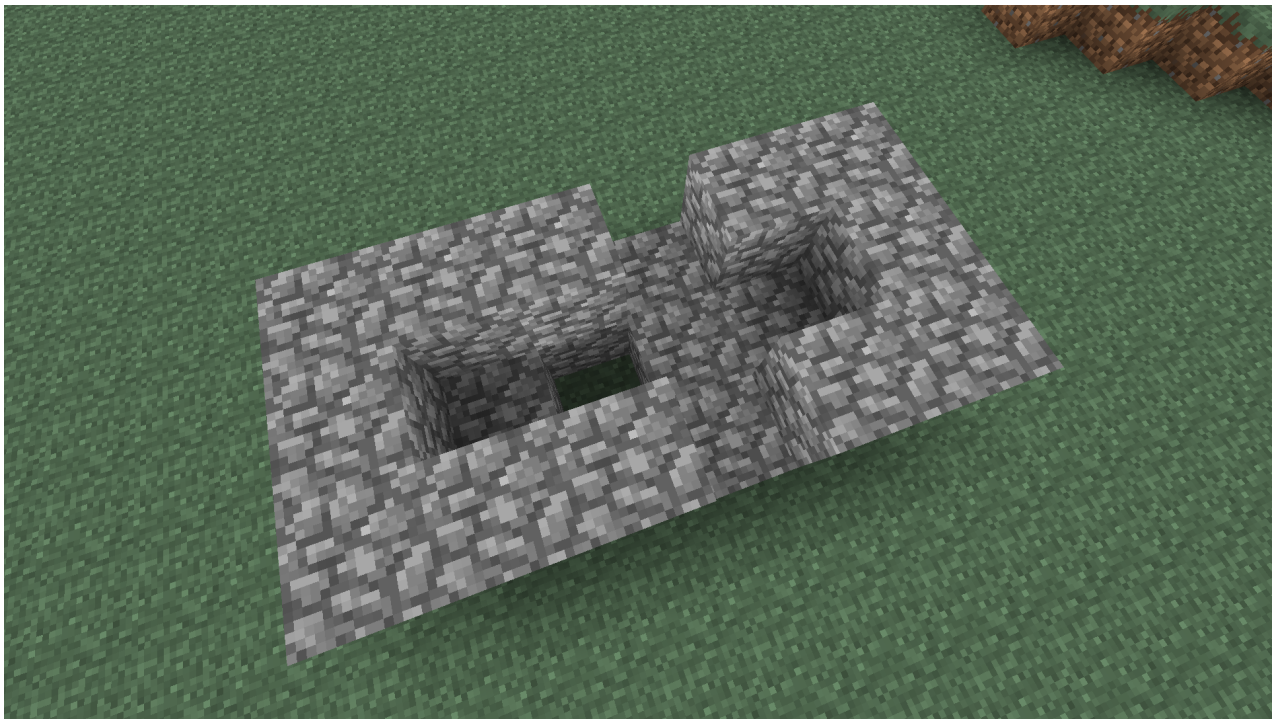
5. Press the button to fire!

We know that we're limited to using TNT seven blocks away from whatever we want to fire. How could we increase our firing power with this limitation? Come up with some ideas and designs and test them out. Also, experiment with the height of the projectile TNT. How does placing it onto a slab change the distance shot?

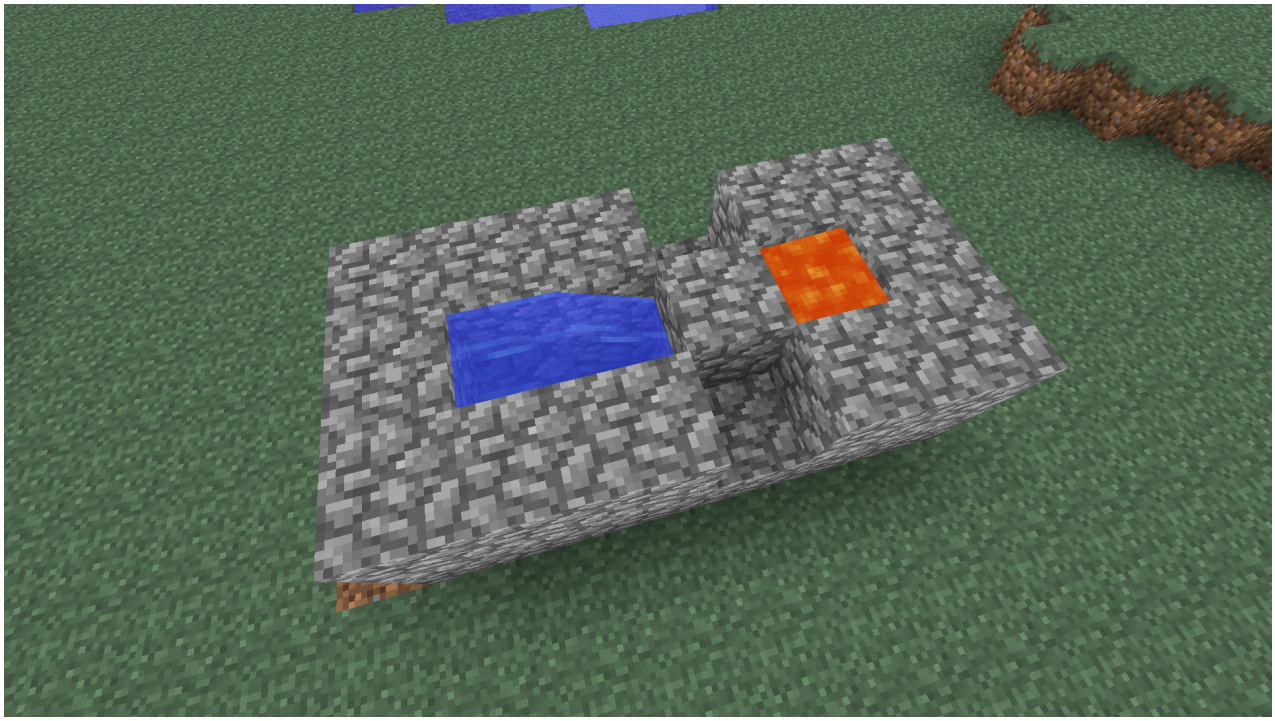
Cobblestone Generator

A cobblestone generator is both less costly and more useful than a TNT cannon. Cobblestone is created when flowing water and lava combine and is a valuable resource, especially on resource-light worlds. By controlling the flow of water and lava and utilizing a piston circuit, we can create a contraption that automatically builds cobblestone for us to harvest.

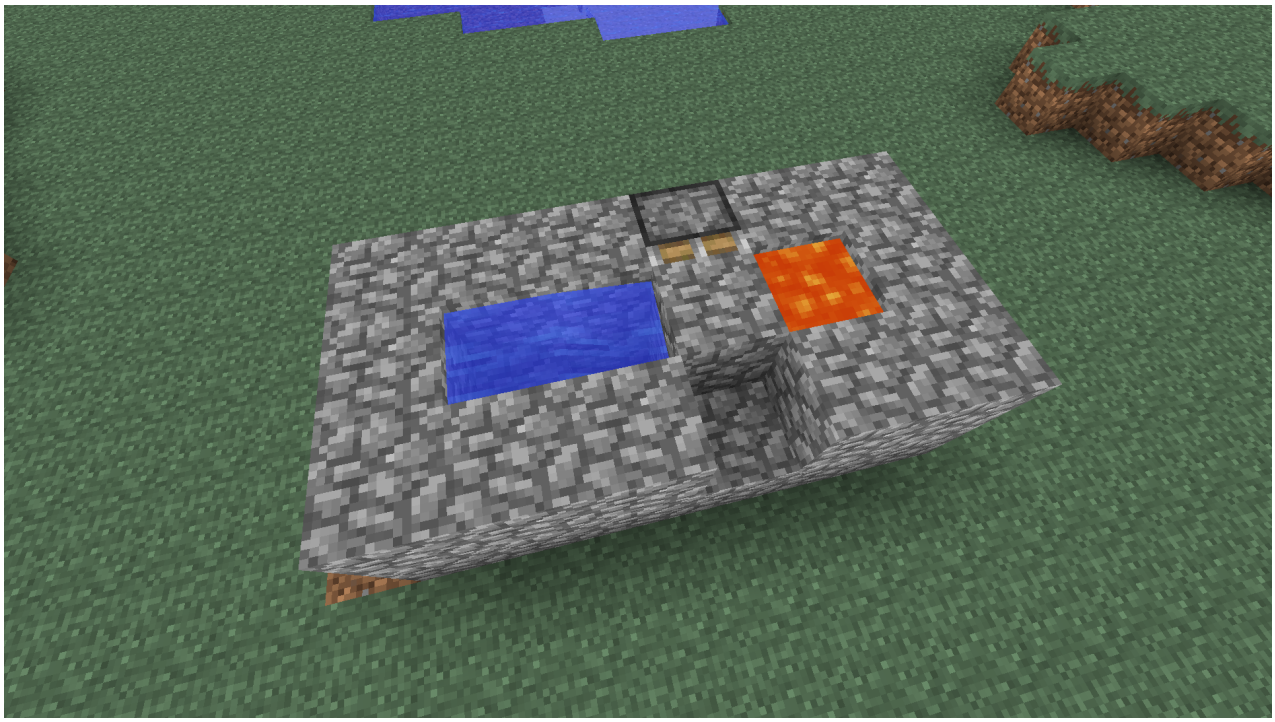
1. Build the body of the generator. Be sure to include the hole in the middle, as without the hole water would flow over the lava source block and turn it into obsidian.



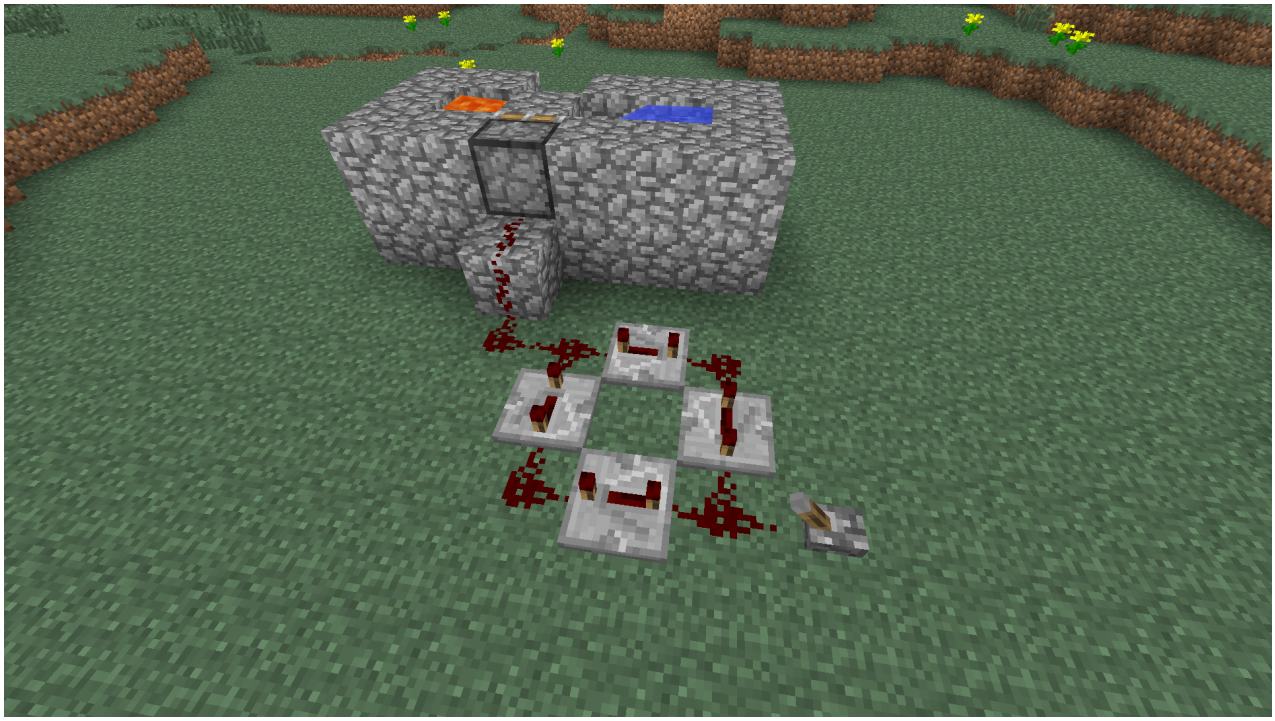
2. Place the water and lava source blocks from a bucket. Notice how cobblestone is formed in the middle!



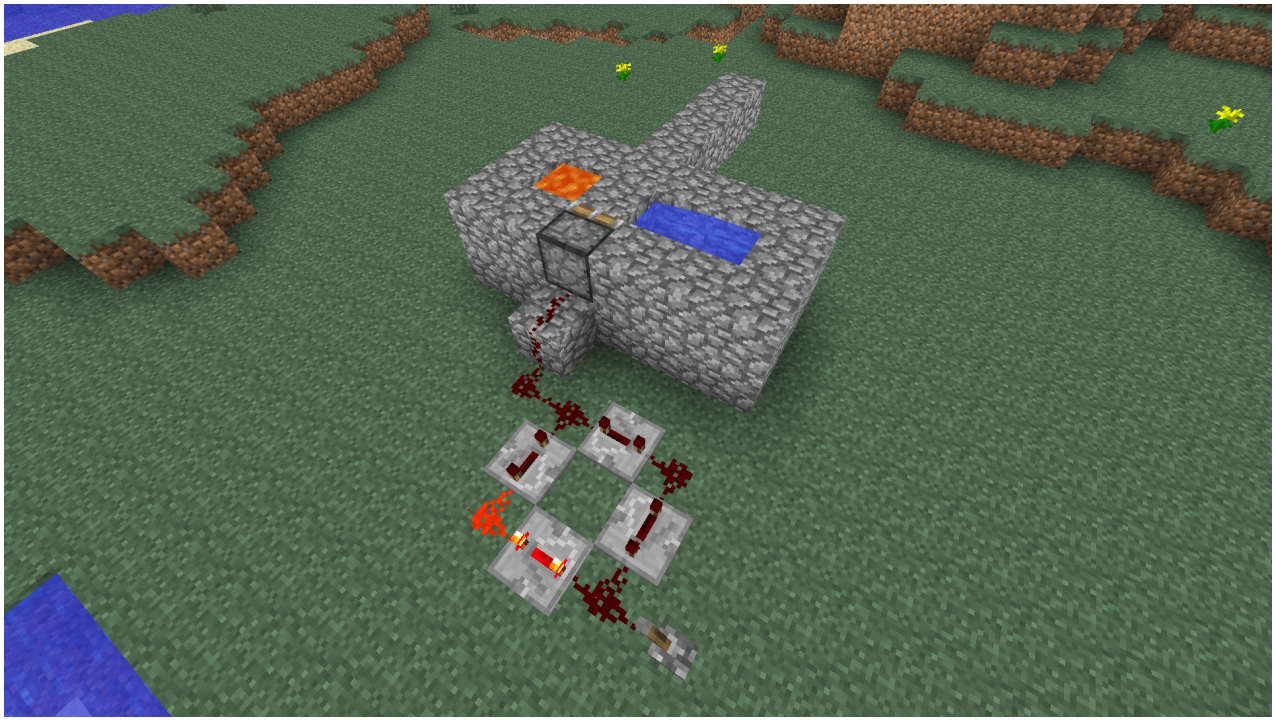
3. Place the piston that will push out the cobblestone blocks.



4. Create the redstone circuit that will cause the piston to automatically extend and then retract. We'll be talking about redstone clocks later in the lesson, but this one is simple enough that we can make use of it right now.

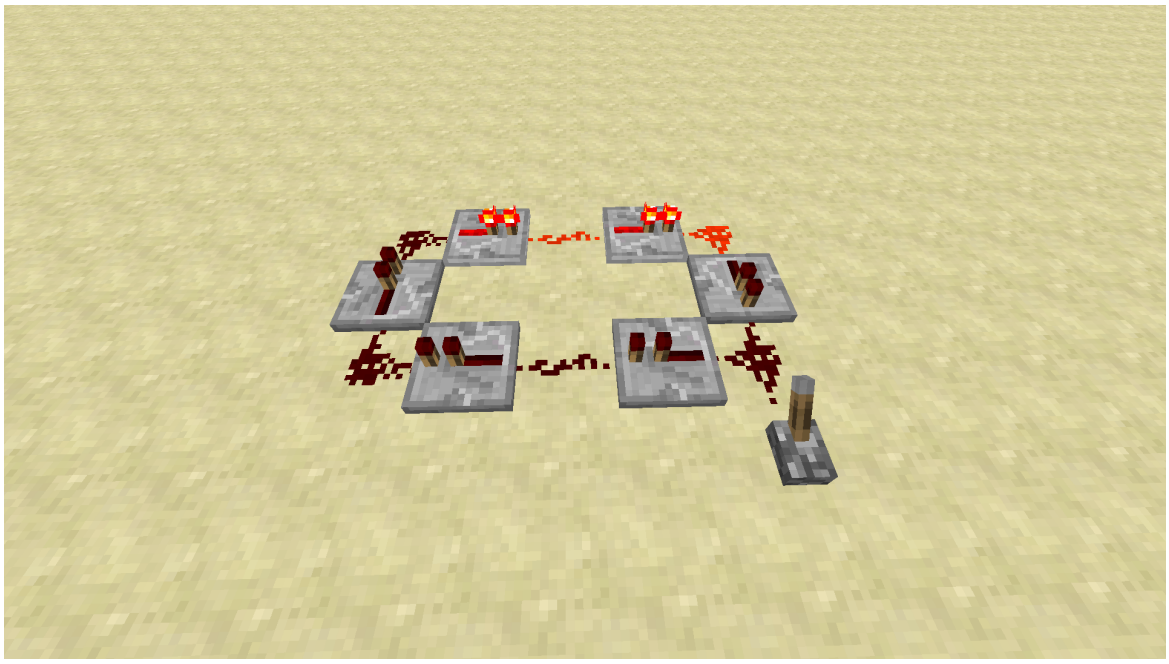


5. Quickly flip the lever on and off to start the generator. Pistons will push up to 12 blocks before stopping, but if you keep mining out the cobblestone column this generator will run forever.



Redstone Clocks

Interestingly enough, you can use redstone to make simple clocks in Minecraft. Redstone signals do not travel instantaneously. Instead, they update based on the timing of “ticks”. A “tick” is one update in the world of Minecraft; for redstone, ten ticks occur per second. Generally torches, repeaters, and other blocks take one tick to update. Repeaters are unique in that they can be set to a specific number of ticks; the “setting” on a repeater (moving the torch back and forth along it) changes it to 1, 2, 3, or 4 ticks. So repeaters can be used to make clocks that pulse at a regular rate.

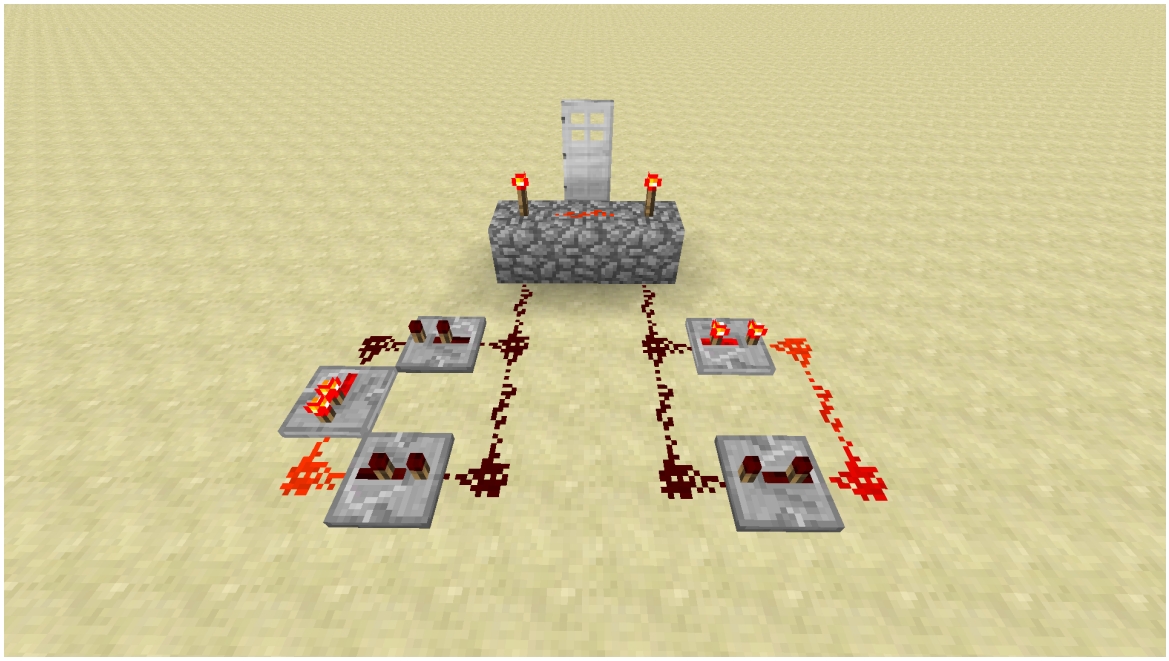


As simple redstone clock made using repeaters.

You can make more complicated clocks by using a simple mathematical principle and something called a logic gate. Let’s say we wanted to make a 30-tick clock, which would pulse every 3 seconds. To use our above design we’d need 8 total repeaters (7 of 4 ticks and 1 of 2 ticks). However, we can actually get away with using just 4 repeaters. To do this, we’re going to make two separate blocks of length 5 and 6. Individually, these clocks only require two repeaters each.

Now, think of the tick numbers that these clocks will pulse on. The first one will pulse at 5, 10, 15, 20, 25, and 30. The second one will pulse at 6, 12, 18, 24, and 30. Both of them will pulse at 30! Since 5 and 6 are both factors of 30, the two clocks will always pulse on multiples of 30. Thirty is the least common multiple (LCM) of the two numbers, and we can use the LCM to make clocks out of far fewer repeaters than with our simple design.

Next, we will use a logic gate called an AND gate. Basically, it’s a series of blocks that takes two redstone inputs. The output will only be ON if the two inputs are both ON. Otherwise, they’ll be off. If we AND the two outputs of our 5 and 6 clocks, the output of that AND will pulse only every 30 ticks, or 3 seconds.



An advanced redstone clock.

Redstone clocks have any number of uses. They can be used to make dispensers output something every set period of time, timers for playing games, or activate pistons for making cobblestone generators. Try to think of some productive uses for redstone clocks and implement them yourself!

Command Blocks

> A command block is a redstone component that can execute commands when activated. It cannot be obtained legitimately in survival mode, and is primarily used on multiplayer servers and in custom maps.

Practicing the basic commands available to command blocks

Create a new creative world.

To obtain a command block type the command `/give <playername> command_block`.

Place the command block and right-click it. Now do the /give command again. This will give the nearest player an iron pickaxe, for example:

```
/give @p 257
```

Click “done” and then put a button on the command block by holding the button and shift clicking on the command block. Then activate the command block by right-clicking the button.

Command block reference

Some of the other more useful commands for use in command blocks are:

- `say`, `tell`, `msg` and `w` (whisper) to communicate messages to a player(s). `tell <player> <message>`
- `clear <player> [item] [data] [maxCount] [dataTag]` clears a players inventory, or just the items specified in the arguments
- `effect <player> <effect> [seconds] [amplifier] [hideParticles]` gives the targeted player or entity the specified effect for the specified time (default is 30 seconds). There is also `effect <player> clear`
- `gamemode <mode> [player]`
- `playsound <sound> <player> [x] [y] [z] [volume] [pitch] [minimumVolume]`
- `setblock <x> <y> <z> <TileName> [dataValue]`
- `summon <EntityName> [x] [y] [z]`
- `testfor <player> [dataTag]`
- `time set <value>`
- `toggledownfall`
- `tp [target player] <destination player>` or `tp [target player] <x> <y> <z>`
- `weather <clear|rain|thunder> [duration in seconds]`

- `xp <amount> [player]`

There are some additional new commands coming in the 1.8 release. We will update the electronic version of this book when that version is released.

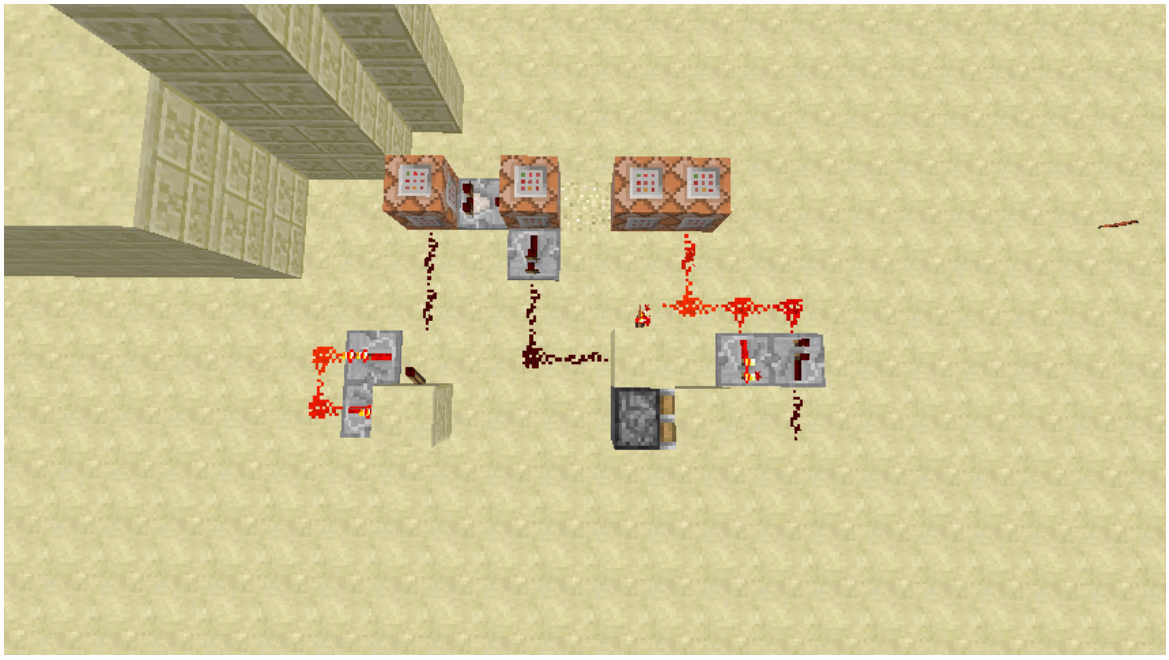
Communicating to a player with command blocks

This exercise is going to use command blocks to warn a player of impending doom. We've provided a map with the trap already set. Try it out, then observe how it works.

The first circuit involves a clock, a `testfor` block which then powers a `say` block when a player enters a specific radius:

Clock/Testfor/Say circuit

The second circuit involves the same configuration as above, with the addition of a switched clock and two `setblock` blocks:



Drop stuff on the player's head when they enter the chamber

Change the messages the `say` command blocks output. Change the type of block that is dropped on the player (hint: there are only a few types of blocks that will fall, and two of them won't require the clock circuit). If you have time, recreate the trap. (If you don't, don't worry, we're going to create another trap later.)

The map for this exercise is called "It's a trap!".

Moving a player around a map with command blocks

Let's take a look at how to build a simple security system for your house, using a command block and a pressure plate.

- First, place a pressure plate in front of your door.
- Make sure that the player will step on it when walking to the door.
- Now, place a command block under the pressure plate somewhere so that it will be powered when the plate is stepped on.
- Now go to the command block and enter
  `/tp @p[r=<radius>,name=!<yourname>] <x> <y> <z>`

Breaking this command down we have `/tp`, which is the command for teleporting players. Then we have the `@p` specifier, which says that this command block acts on players. The `@p` command takes arguments, `r=` for radius and `name=` for which players to teleport. Setting the radius is straightforward. Setting the name, however is a little interesting. Here, we use the `!` operation, which means `NOT`. Just like in redstone, this inverts the output of a command. Right now, we're using it to make sure any player that is `NOT` you gets teleported, while you remain safe. The final part of the command is the location to teleport to, which you put in place of the `<x> <y> <z>` in the

command.

Giving a player items with command blocks

The `/give` command is one of the more versatile commands available for use in Command Blocks. In this section, we'll list a few good uses.

Public Lottery

```
/give @r
```

Add the `@r` specifier to the `/give` command to make the block give the player a random item. Be careful, since this can give items that are not otherwise available.

Starter Kits

```
/give @p[r=2 m=2] <item>
/gamemode 0 @p[r=2]
```

These two commands need to be activated in sequence. The first command only activates if the player is in gamemode 2. The second command changes the players mode to survival so that they can't activate the block a second time. These commands use the same `@p` specifier as the pervious commands.

Command Block “Spawner”

In this section we will use two clock circuits to control two command blocks, creating a custom mob spawner whenever a player enters a certain distance from the contraption.

The area needed for this spawner is approximately 8x5. It could be done more efficiently so if you finish early try to make a more compact contraption with the same functionality.

The first step is to create a simple clock circuit to power the command block that will detect when a player is within a certain radius.

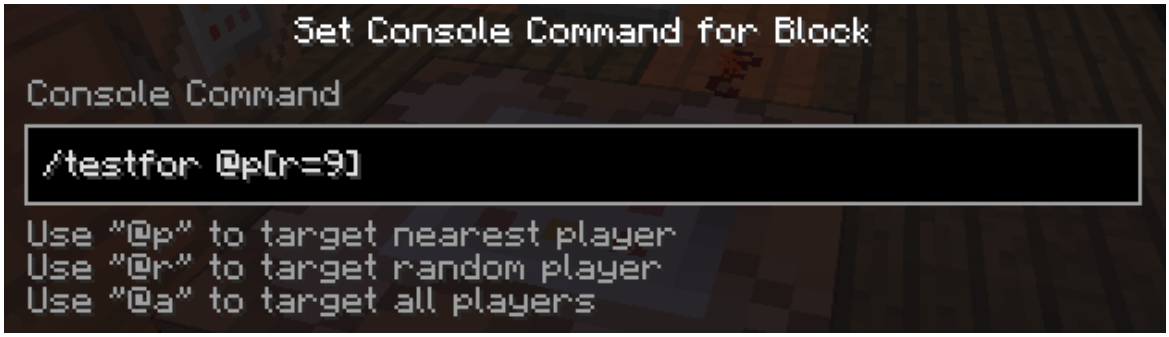
Our basic clock circuit for the monster spawner.

The reason we need the clock circuit is to continuously fire the command block command. The block executes the command once for each redstone pulse, so a continuous redstone signal will only cause the command to execute once. We need to continuously check for players within the radius of the “spawner” so we want to pulse the signal with the clock.

This is the basic design of the spawner, with the initial clock in the upper right, triggering the detector block in the bottom right, setting off the second (slower) clock in the bottom left, which in turn pulses the command block in the center, which summons a mob.

The design of the monster spawner.

The first command block uses the `testfor` command to test for any player within *N* blocks (in this case, nine)



The test for a player within a certain radius.

Connected to this block is a redstone comparator. Comparator blocks have a number of uses but in this case we are using its properties specific to command blocks. The comparator will output a signal equal to the number of times the command block's command succeeded. Since we're connecting the comparator directly to a NOT gate it won't matter as long as the block's command succeeds once (i.e. there is one player within the given radius).

The comparator block.

The redstone torch forming that NOT gate stops the second clock circuit from executing by keeping

the circuit powered continuously, unless it's switched off by the signal from the comparator.

That circuit powers the second command block, which uses the `summon` command to create zombies in a certain spot, in this case relative to the command block, just outside the walls surrounding the contraption. Normally the contraption would be hidden underground or some other location that would be difficult to grief, especially while zombies are spawning every few seconds!

The summon command.

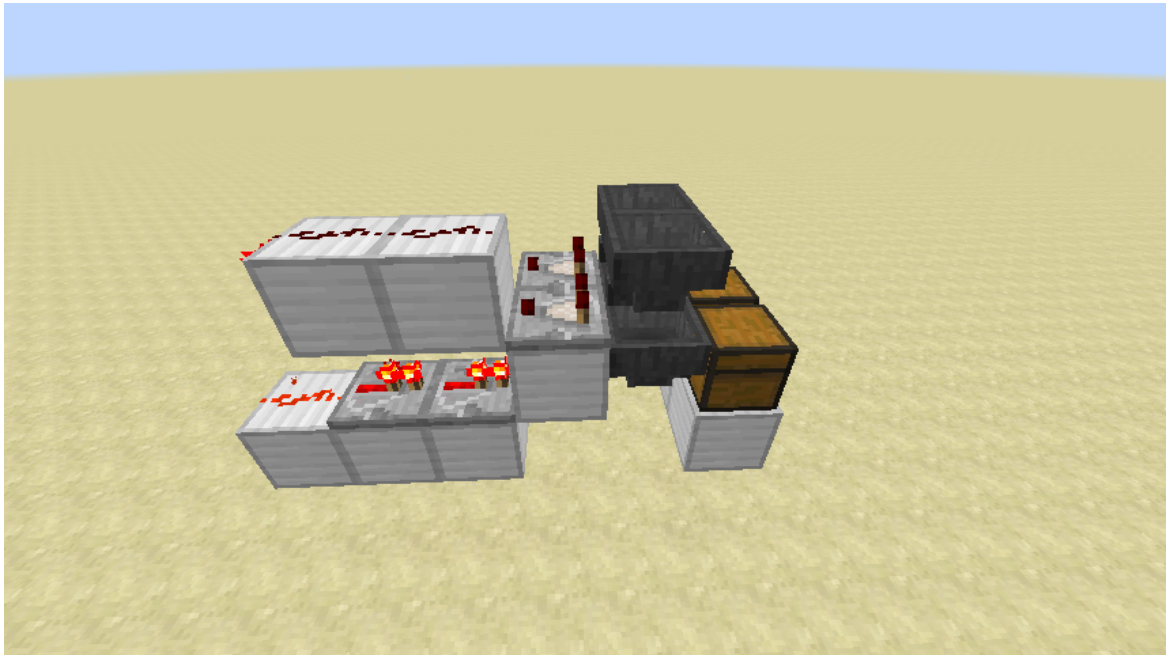
Spawner Upgrades Challenge

Once you have a working spawner, try to figure out these upgrades:

1. Make the contraption more compact without decreasing the delay between zombies.
2. Spawn more than one mob at a time, in different areas around the spawner.
3. Add a switch to turn on and off the spawner, by making the first clock circuit a switchable loop.

Sorting

Sorting contraptions involve NOT gates, comparators and exploiting the special characteristics of hoppers. This one is complex so we'll be supplying a sample map with a completed sorter for you to reference.



Sorting circuit.

Here is a screenshot of a basic sorting circuit. The hopper configuration is important here: the bottom row of hoppers must be attached to the chests (shift-click to attach a hopper to a chest) and the top row of hoppers must be attached to the comparators. This is where we exploit a special feature of comparators:

> If a comparator is placed next to a container, it will provide an output based on the percentage of used space in the container. link

There is some specific math at the above link to determine how to get the comparator to output, but with the hopper it is easy enough to just fill the hopper until you see the output occur. In this case we're going to fill the hopper so that any additional item creates the redstone signal, which will in turn power the hopper below. This also exploits a special feature of hoppers:

> When powered by redstone, the hopper won't take items from the inventories of blocks directly above it, put items into an attached inventory or suck up items from the environment. However, a hopper below it can still take items from its inventory and a hopper above or beside it can still put items into it. link

Build enough of the circuit to test the output signal from the top hopper. Put a single item of the type you want to sort in the first slot in the hopper. Put a single item that won't be coming through the

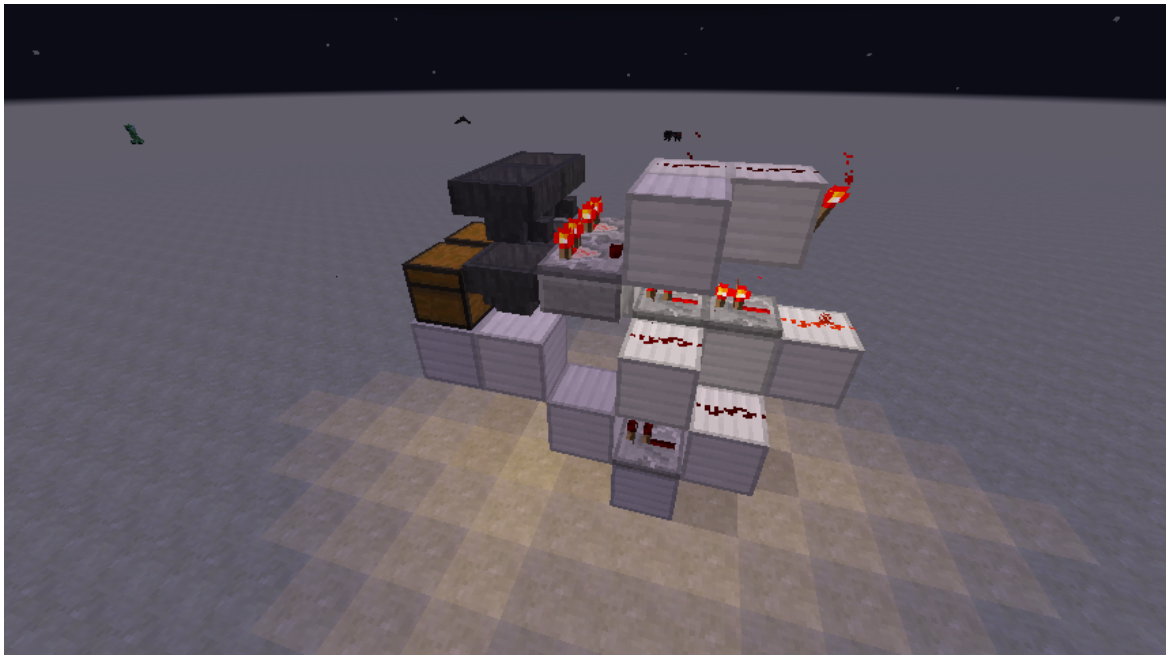
sorting machine in the rest of slots. Fill the *last* slot until you get a signal, then take one item out. Now only those two items can enter the hopper, and the hopper below will take from the first slot (the item type you want to sort) whenever one more item lands in the topmost hopper.

Now refer to the screenshot above again and finish the circuit. The idea is the send the signal from the top hopper to the bottom hopper. This could be done any number of ways (other designs power sticky pistons with redstone blocks), but we're trying to keep our sorter size to a minimum. The configuration is comparator->two blocks with redstone dust->redstone torch–this creates a NOT gate, the torch will stay lit as long as there is not a signal from the comparator. Then back towards the bottom hopper with a redstone dust and two repeaters. As long as the bottom part of the circuit stays powered (there aren't enough items in the top hopper to power the top part of the circuit), the bottom hopper won't take items from the hopper above it.

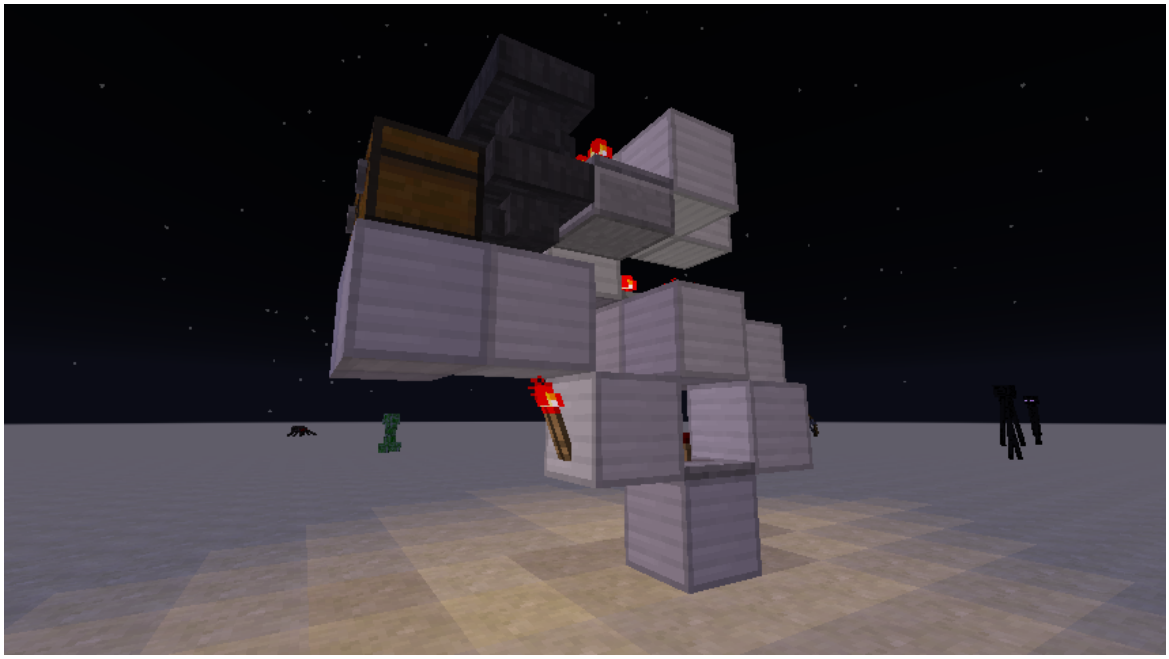
You can now test this part of the sorter but placing the sorted item into the hopper. They should immediately fall into the bottom hopper, then the chest. (Except for one–the one downside of this design is is leaves one item in the bottom hopper.)

Sorting circuit.

Now we want to create a similar circuit for the next item we want to sort. The trouble is, if we build the same circuit, it will interfere with the one next to it. So this one has to be changed up a little bit. It goes down one more block, brings the signal back towards the front and puts the NOT gate below the hopper. Refer to the screenshots below and use the completed example as a reference if you get stuck.



Second sorting circuit.



Another view of the second sorting circuit.

Finish it off

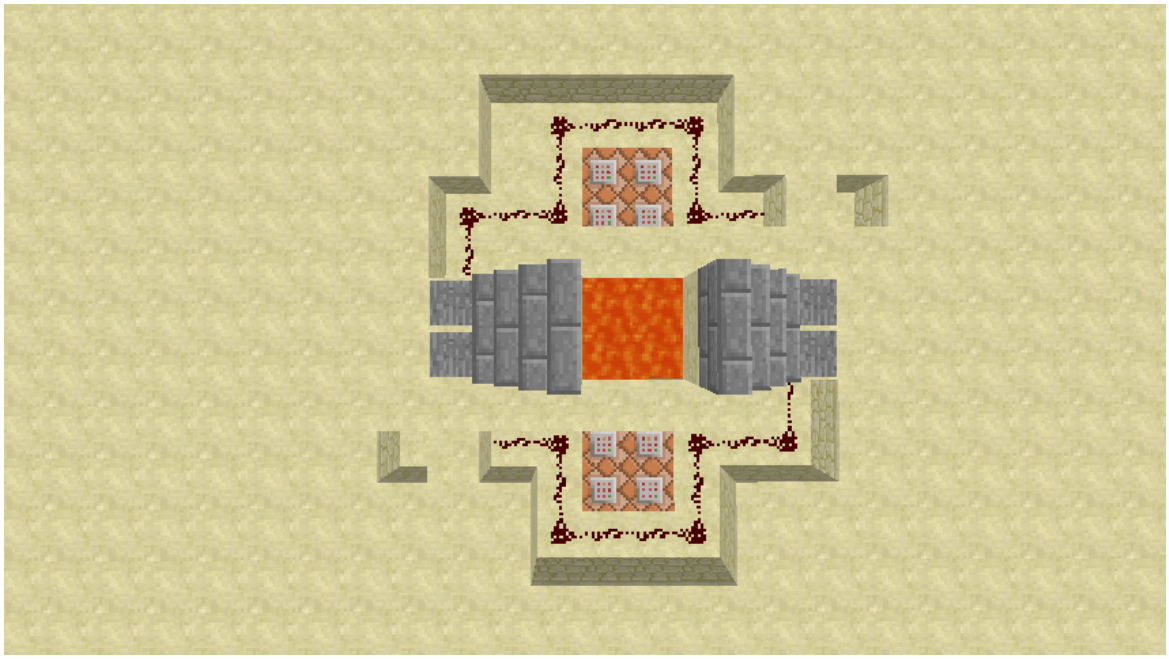
Now complete your sorting machine with two essential finishing touches:

1. A way to load the machine (refer to the example map if you're having trouble thinking of a good way to accomplish this).
2. A "catch all" chest for items that don't have a specified chest in the machine.

If you have enough time, you can make your machine bigger, as well (add alternating rows of each sorting circuit type).

Automatic Bridge

In this section we are going to build a small prototype of an automatic one-way bridge.



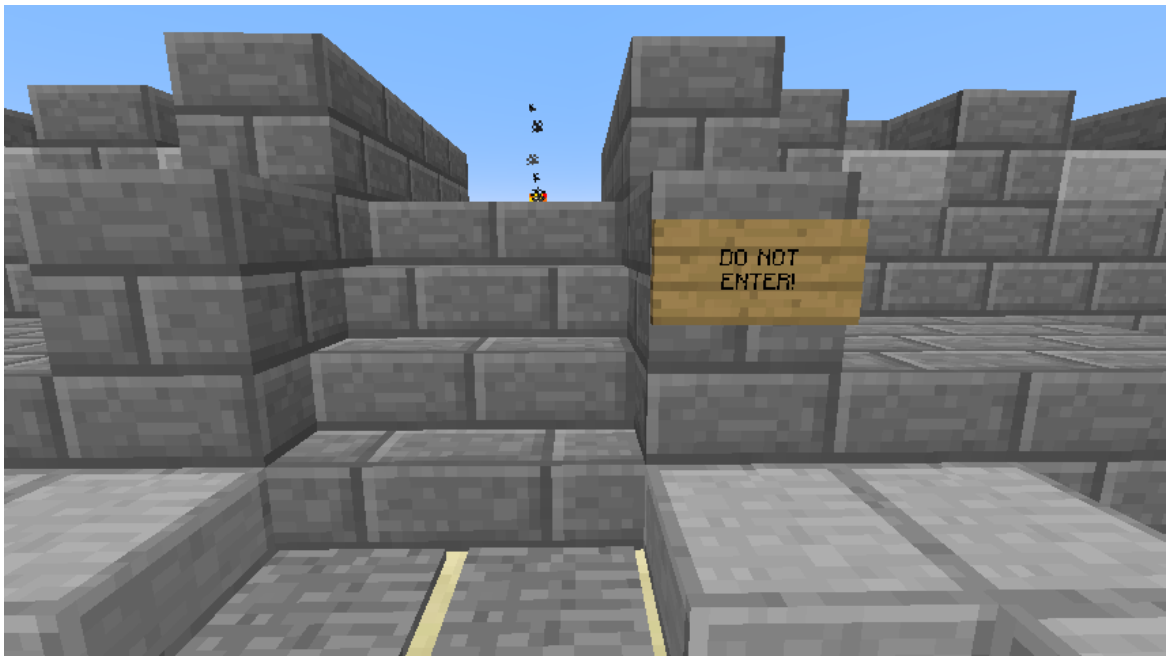
The build of our automatic bridge concept.

The construction consists of four command blocks to create the bridge and four command blocks to return those blocks to air once you've crossed. We'll use the `setblock` command with relative positions, so the following examples rely on putting your command blocks in exactly the same relative position compared to the bridge. You don't have to do that, though–you can simply adjust the relative coordinates based on the placement of your command blocks.

Setblock command to place the bridge stones.

Setblock command to replace the bridge stones with air.

You can test your command blocks by placing a button on them (shift-click to place a button on a command block). Once you have the `setblock` commands all set and working correctly, connect your command blocks to pressure plates via redstone dust, as shown in the first screenshot above. Now hide your command blocks and redstone!

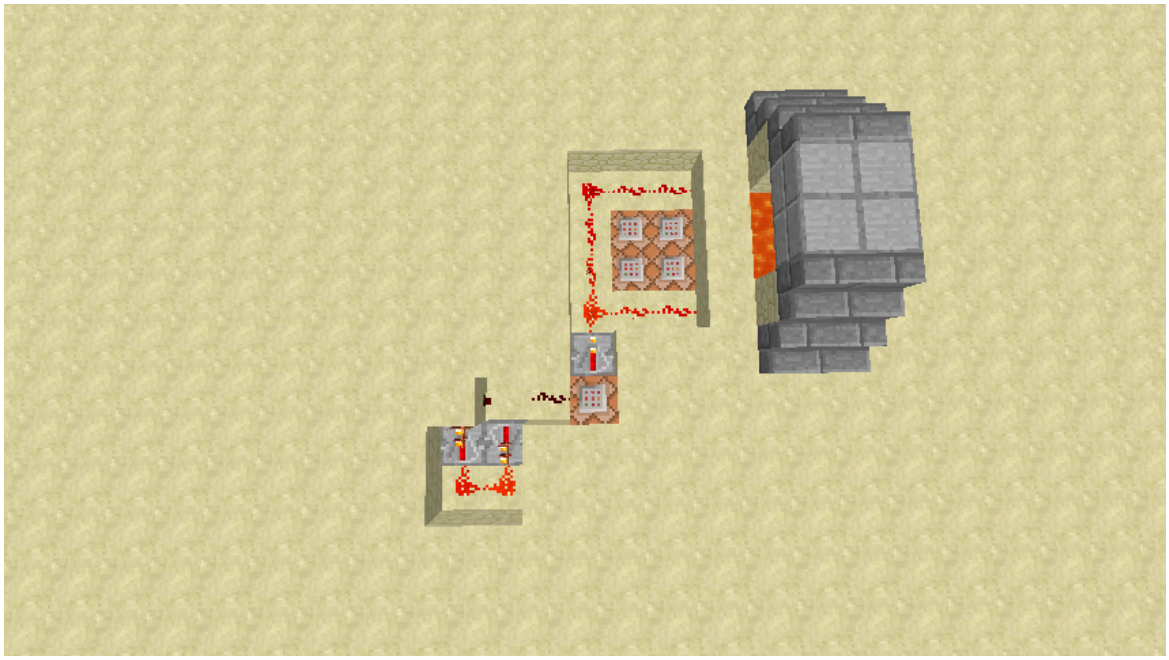


Do not enter!

Make it better

The above build is more of a proof-of-concept. It's pretty easy to see that the pressure plates cause the bridge action, and it's not much of a deterrent, besides. One improvement we could make would

be to use a detector block to determine when a player approaches from one side and place the bridge blocks or air appropriately. Use the techniques from our “spawner” contraption to create a clock powering a `testfor` command block.



Bridge redstone with detector block instead of pressure plates.

Testfor command. Your coordinates will be different! (Or you could just do a large radius).

Now make the `testfor` command only test for your player. You could even make it a trap bridge, only letting you cross and turning the blocks to air when any other player tries to cross.

ComputerCraft

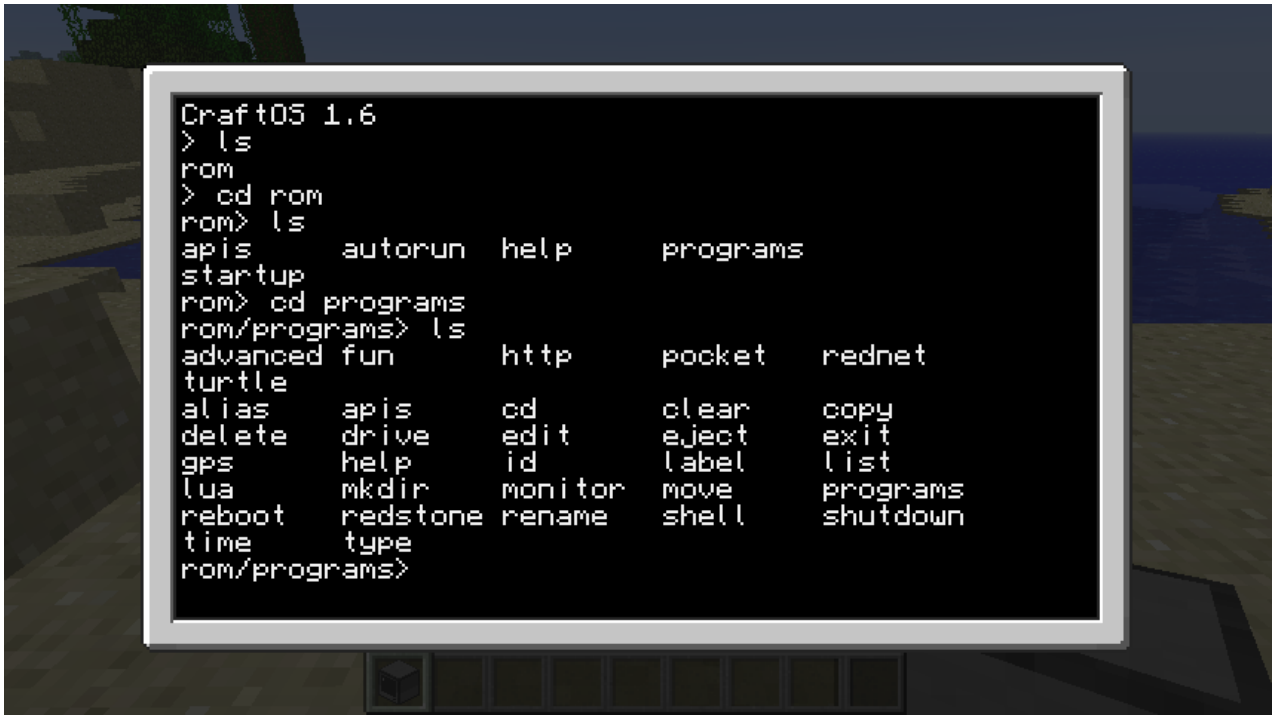
Introduction

> ComputerCraft is a modification for Minecraft that's all about computer programming. It allows you to build in-game Computers and Turtles, and write programs for them using the Lua programming language. The addition of programming to Minecraft opens up a wide variety of new possibilities for automation and creativity. If you've never programmed before, it also serves as excellent way to learn a real world skill in a fun, familiar environment.

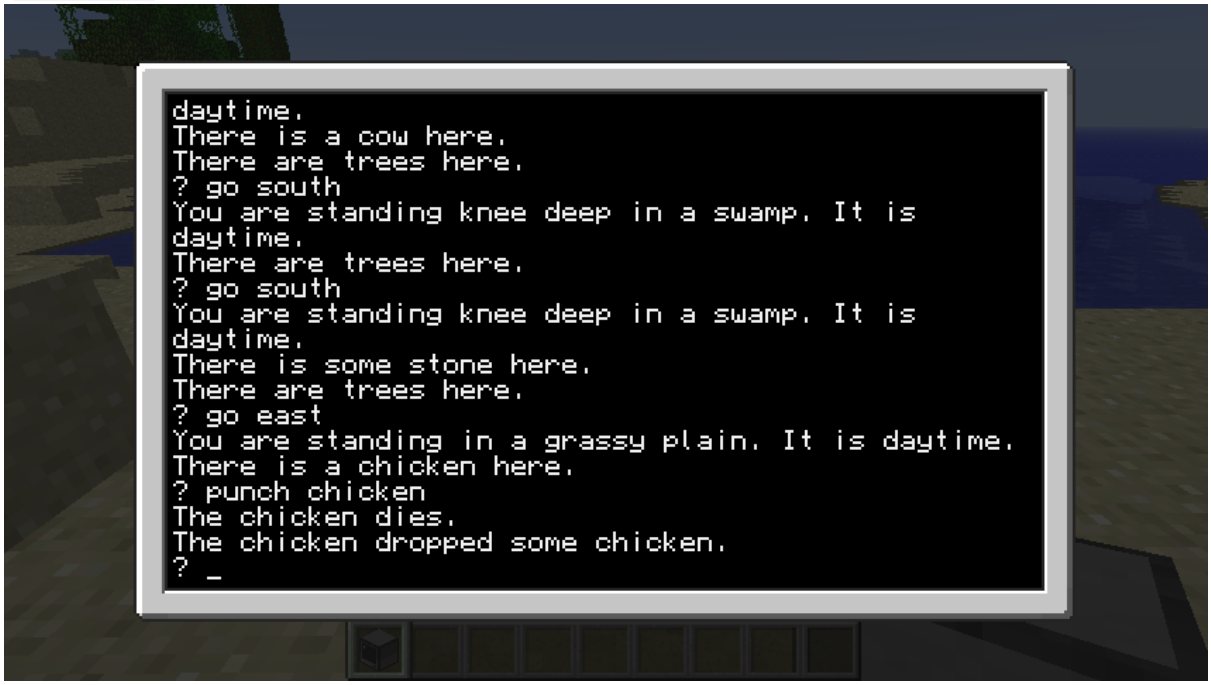
Getting familiar with ComputerCraft

1. Open the ComputerCraft world
2. Open your inventory and search for `computer`
3. Place a computer on the ground and right-click on it

4. Navigate to programs
5. the `ls` command lists the contents of the current directory
6. the `cd` command changes directories, e.g. `cd rom`



7. Play text adventure Minecraft inside a ComputerCraft computer
8. type `adventure`
9. some of the commands available in the Adventure program:
    1. `punch`
    2. `take` or `grab`
    3. `craft` or `make`
    4. `go`
    5. `eat`
    6. `inventory`

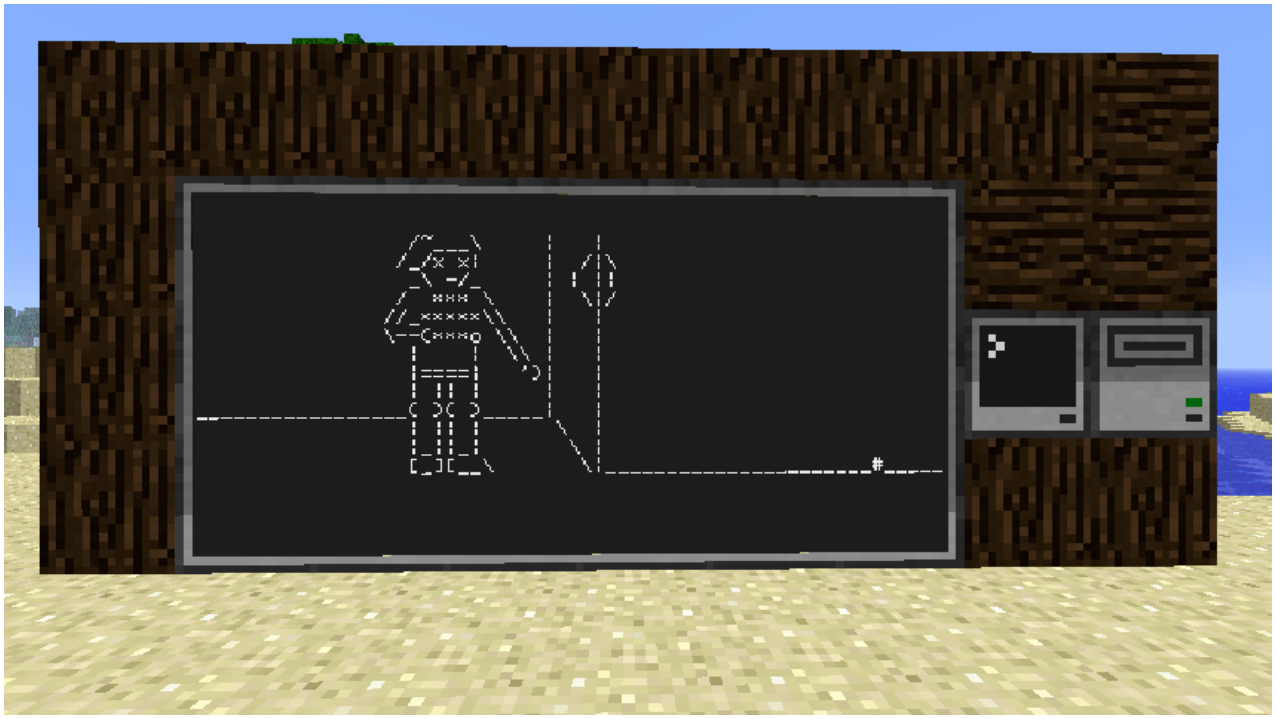


In case you haven't noticed by now, Adventure is really just text-based Minecraft. You're playing Minecraft on a computer inside Minecraft.

10. Add disk drive
11. Open your inventory and search for `disk drive`
12. Place the disk drive next to the computer

13. Add a music disk to the disk drive
14. Play the music with the `dj` program
15. Create a monitor
16. Open your inventory and search for `monitor`
17. Place 12 monitors in a 6 wide by 3 high pattern to create a giant widescreen monitor
18. Place a computer next to the monitor
19. Place a disk drive next to the computer
20. Watch a movie on the monitor
21. Open your inventory and search for `disk`
22. Find a disk labeled `alongtimeago` and place it into the disk drive

23. Right click on the computer and run the `alongtimeago` program
    1. To run a program on the disk, specify the full path to the program like this: `disk/alongtimeago`
    2. To run the program on the monitor, specify the monitor first, so this: `monitor [top|bottom|left|right|front|back] disk/alongtimeago` The syntax `[top|bottom|left|right|front|back]` means pick which side your monitor is on and only type that direction, of those six shown. So your command would be something like `monitor left disk/alongtimeago`.
24. Sit back and enjoy the show



25. To quit any running program, hold down `ctrl + r`

Turtles

Intro

Turtles are programmable robots that you can use to collect resources, clear terrain, and other such tasks. They run an OS called turtleOS and the programs they run can be stored on internal memory or floppy disks. There are farming, mining, crafting, and melee turtles. They are categorized based on the Diamond tool* you equip them with.

*Note: Tools equipped to turtles will not wear out and turtles themselves are indestructible (unless you break them yourself). This makes them one of the safest ways to utilize diamond tools, not to mention the time they will save you.

Like any robot, turtles require fuel. They can get energy from anything that would work in a furnace as well as other more advanced options we'll get to later. Different types of fuel will yield different *fuel counts* which is the number of blocks the turtle can move with that amount of fuel. For example, coal will give the turtle 80 fuel, so the turtle can now move 80 blocks.

1. Add some dancing turtles
2. Open your inventory and search for `turtle`
3. Place a turtle or two on the ground
4. Right-click on the turtle
5. Run the `dance` program

Make it move!

Turtles have several default programs including the "go" program.

1. Select a turtle and put a *coal* in its inventory.
2. type `refuel`
3. Notice it says *Fuel level is 80*
4. type `go forward 10` and watch it go!
5. type `refuel` and notice that the fuel level is now 70.
6. Whenever there is no fuel source in the turtle's inventory, you can type `refuel` to check its fuel level.

The "go" program has the following format: `go <direction> <distance>`

Note: For fast/mass refueling, type `refuel all`

Appendix 1: MCU Servers

We have a lot of cool stuff on the servers and have more things planned, like build projects, adventures and online lessons. Right now there is a Minecraft U building with all the redstone logic gates we built in our redstone camps; there are two large mazes built with ComputerCraft turtles and maze making algorithms; an obstacle race course; a distant and mysterious pagoda that whispers to you as you approach; a special ComputerCraft island and an undersea village.

And the main point of maintaining the server—you know who is on it. Only campers and staff, plus a very few vetted outside players, are on the MCU servers.

This is all included as a part of your week at Minecraft U, and you may continue to play on the server indefinitely.

Access to the camp server is contingent on behavior guidelines just like those in real life. Anything you wouldn't do to someone else in real life, you should not do on the server. PVP is turned off. Do not ask for it to be turned on.

Violation of any of these rules will result in bans on a graduated scale described below.

1. No griefers. A griefer is a player in a multiplayer video game who deliberately irritates and harasses other players within the game. This includes causing damage to other players, damaging existing structures/contraptions and stealing.
2. It bears repeating: No stealing. You wouldn't just walk into someone else's house and take things out of their closet. Similarly, you may not remove things from other player's chests or houses without consequences. The only exceptions to this are chests marked with a sign as community chests, and food in the case of emergency.
3. No bad language.
4. No raging or rage quitting. Even if not intended as raging, all caps chatting may be interpreted as such.
5. Fill creeper holes and repair any other damage done by creeper blasts. (We have disabled creeper block damange on the servers for now.)
6. Replant community crops, re-breed community animals and leave as much behind in community chests as you take out.

Punishment for breaking any of the server rules are as follows:

1. Verbal warning through chat.
2. If immediately reachable, warning to a parent.
3. Kick.
4. Indefinite ban. Length determined after discussion with a parent.
5. Permanent ban. Reversible only through appeal.