4/25/2016

Samuel Morin, Frank Torchia, Peter McCloskey, Dustin Chylstek, Jacob Schriener

CS 1538 Course Project

Simulating Resource Processing in Minecraft

**Introduction:**

Minecraft, an open-world game officially released in 2011, gives players the freedom to do or build almost anything they can imagine. Building structures and crafting equipment in the game requires the player to explore the procedurally generated worlds and find materials. Many of these raw materials, however, must first be processed before they are useful to the player. Minecraft offers many ways of automating both the mining and processing of these raw materials. By crafting and assembling various configurations of power generators and machines, the player can fully automate the resource acquisition portion of the game. This automation can have several benefits to the player, and can greatly expand the scope of what they are able to accomplish. It can allow them to focus all of their time on other aspects of the game, and crafting materials can be gathered and processed at a much greater rate than by hand.

Automation can, however, be a daunting task for players, as it requires a complex setup of devices in order to function properly. There are several different possible ways of processing the gathered ore into usable bars, with a varying number of steps. There are different machines that accomplish this, with varying output yields. The automation process also requires power, and each machine has different power requirements. This power can be provided by generator machines, which require resources of their own to run. There are also different types of power, and multiple generator models for each type. On top of all this, most machines can be upgraded (for a resource cost) to improve their performance (this may also have negative effects, like increasing power usage). Thus, players are faced with choosing from a large set of possible configurations for automating their resource gathering.

Our goal was to simulate and analyze the possible configurations to determine which ones would be ideal for certain situations. We focused on three areas in which the configurations could differ: the amount of materials output, the amount of power used, and the total time to process all of the raw materials. To accurately simulate the automation process, we recorded the rates at which each type of resource appeared when mining, and used that data to model the rates at which a player would expect to receive each type when playing. We then divided the machines into a set of three basic “paths” through which we could put the raw resources, and modeled the paths in our simulation. After modeling each generator type, we ran the simulation multiple times on a wide variety of potentially optimal paths that were constructed by combining different basic paths and generator types.

Once we had finished running our simulation, we were able to analyze the output to determine which one would be best for a player given their initial goals. Comparing the results of four different furnace types, three machine paths, and four furnace types, left us with two distinct options that were clearly ideal based on possible player preferences. For efficiency, the machine type three, generator type three, and furnace type three yielded the highest output per coal unit used. For raw output, we discovered that machine type two with furnace type three yielded the most.

**Background:**

For this study, we focused on the resource gathering aspect of Minecraft. Acquiring and processing the resources by hand is tedious, so many players automate this process using in-game machines. There are many possible configurations of these machines, and manually testing each would require setting up each configuration in-game and then waiting for it to mine and process the materials. This process would be highly time consuming itself, and would require each configuration to be tested many times to account for discrepancies in the resources acquired by an automated mine (if placed in a location with a large cavern or underground river, for example, the mine may output far fewer resources or stop functioning entirely). Thus, creating a simulation that accurately models the resources a player could expect to find in an average area allows us to determine which automation configuration is ideal for the player’s goals.

The world of Minecraft is made up of procedurally generated blocks, each of which has a type. Mining the appropriate type of block gives the player a randomly generated small number of the resource (or, more often, a single unit of that resource). These resources must then be processed, either by hand or through the use of various machines. Each machine requires power, which is provided by a generator. Each generator requires coal as an input, and outputs one of two possible power types (RF and EU). The general procedure for processing resources is as follows: First, the raw materials are put into a machine which outputs an enhanced version of the material at a ratio determined by the machine and upgrade type, with a small chance of outputting an extra unit of the enhanced material. Once the materials have gone through all of the enhancing steps, they are put into a furnace machine, which outputs the materials as a bar at a 1:1 ratio. The bars are the form of material that the player desires, as they are the form of the material that is most useful in-game.

Our goal was to determine the optimal path for processing time, bar output, and power usage, so we broke the process into two steps: machine paths (which can be thought of as the “pre-processing” of the materials) and the furnace. There were three possible machine paths, detailed in the attached “MineCraftDataSheet.xlsx” file and four possible furnaces. Each machine path and furnace required power, of which there are two possible types, and a total of four different possible generators (also detailed in the attached file, under the tabs marked “generator”). Our simulation allowed us to test the potential configurations to determine which ones output the most bars, used the least amount of power, and finished in the shortest amount of time.

**Approach/Method:**

Our approach started with collecting data about the resources a player can expect to find when they play Minecraft. The resources that we were interested in were primarily the ones that require processing into bars, which were iron, tin, gold, and silver. To collect the data, we created a script that recorded each block mined by an automated quarry, and ran two quarries simultaneously. The script also noted the layer (or how many blocks under the surface) each block was mined at, as the depth affects the quantity and type of resources that can be found. In addition to the resources which required processing, the script also recorded instances of the diamond and “lapiz” resource types, which are useful for players but do not require processing, and the coal resource type which is used by the generators to produce power. The script was run on mining areas that measured 64 x 64 x 42 blocks. The results of these scripts were what we used to model our input data.

For input modeling, we first divided the results into 5 layers, as the depth of a block (how far it is from the surface) affects the types of resources that it can be. Our results (located in the “MineCraftDataSheet.xlsx” file, in the first 3 tabs) showed that the overwhelming majority (Roughly 98%) of blocks that were encountered were regular stone blocks. These blocks were of no further interest to us, as we were focusing on the blocks that contained resources. The results of the non-stone blocks are detailed in the attached “Block Usability.xlsx” file (note, the “TF” category refers to the tin resource in layers 1,2, and 3, and silver in layers 4 and 5). The graph below gives an overview of the breakdown over the non-stone blocks in layers 1-5 (Note: Redstone and Diamond have been combined, as they required no further processing).



We then performed analysis of this data to determine the best way to model our simulation’s input. We theorized that the blocks we were interested in (non-stone blocks) would be geometrically distributed throughout our results. Of the special blocks, we theorized that a uniform distribution based on the layer of the block would be the best way to model which type of block each non-stone block was. We therefore decided that the best way to represent this input model in our simulation would be to essentially perform Bernoulli trials with a probability of success based on the probability of finding a non-stone block on the current layer, and then sample from a uniform distribution based on the chances of finding a particular resource type in that layer. Comparisons of the distributions of block types produced by our simulation and the distributions of block types in our raw data indicated that the two were not significantly different.

For the simulation itself, we decided on an event-based model implemented using a priority queue. Each time the simulation is run, all of the non-stone blocks that the automated quarry will mine and time that the quarry will reach them are calculated and added to the queue. Next, the particular resource path (generator type, machine path type, and furnace type) are added to the simulation, as well as the resource types that we are interested in for this particular run. This enables us to examine the performance of the system when processing only certain types of resources (for example, a player may only be interested in maximizing their iron bar output and may not care about gold or tin). The machine types and resource types are located in a “paths.txt” file, which can be easily modified for different paths and resources. Machines are broken down into two categories, “machines” which are the pre-processing steps in a path, and “furnaces”, which represent the final step in the path and convert any resources into bars. Four types of generators were included in the simulation, three of which were RF and one of which was EU. As there was only one EU generator, this was always available in the simulation and was switched on if a machine required it. The paths.txt file included which of the RF generator types to use. Once all of the machines and the generator types have been initialized, the simulation begins.

As the non-stone blocks leave the queue, they are added to the appropriate processing path. This causes the first machine in the path to be marked as running (and using power). It also adds a “machine finished” event to the priority queue, which is inserted at the time that the machine would finish. The block and machine events make up the majority of the simulation, as they represent the actual workings of the automated mining process. The power usage is updated constantly, and all of the output is displayed in an output frame as the simulation is running. The simulation tracks all resources mined, all resources produced as output, any currently unprocessed resources, the amount of power that is both generated and used, and the amount of coal that was used by the generators. The individual events that make up the simulation are detailed below.

The most basic event type is the “ping” event, which simply updates the current power usage and determines if the quarry and/or the resource path is running. The “block arrived” event represents the quarry simulation, it indicates that at this time the quarry encountered its next relevant block. When this occurs, the simulation checks to see if the block is one that we are including in this simulation, and updates the machine path for the block accordingly. If there is not enough power for the machine to run, another generator will be turned on at the next ping event. The “Generator Finished” represents a generator that has finished generating power from its last deposit of coal. Upon exiting the queue, this event causes the simulation to check to see if the generator is still needed. If it is, the simulation refills its coal supply, otherwise it is switched off. A “Machine Finished” event indicates that a machine has completed processing a unit of materials. It causes the simulation to update the path that it was on, as well as update the current output totals (if applicable). The “Quarry End” event indicates that the quarry has completed mining the designated area, and that there will be no more blocks and that the quarry will no longer consume power. The final event type is a simple refresh, to update the display and to check to see that the simulation still has work to do (either the quarry or machine paths are still working). Below is a rough sketch of the basic simulation process:



It should be noted that, for this simulation, we made some simplifying assumptions about the Minecraft quarry model. The first of these concerned the way the generators operate. In real life, the generators require a supply of coal to produce power, and the quarry that mines the coal must have power to run. Furthermore, if the generators ever run out of coal they will stop running, so the quarry must be able to acquire enough coal to ensure that it has enough power to operate. In our simulation, all of the generators have access to an unlimited “reserve coal” supply, which they can take from if the actual coal stocks are depleted. The amount of this reserve coal is tracked by the simulation, and so this simplification may actually provide a benefit. By tracking the usage of reserve coal, our simulation can give players an estimate of how much coal they may want to have on hand before they begin an automated mine if they do not want it to stop functioning.

The next simplification was of the input buffers for the machines. In real Minecraft, the machines have a limited amount of space with which to store unprocessed materials, as well as materials that it has completed processing. We simplified this by giving the machines an unlimited buffer size of materials, which we believe to be a reasonable simplification. In Minecraft, it would not be particularly difficult for the player to construct additional infrastructure to support something like this, where resources that were waiting to be processed could go to a different storage mechanism if the machines were full. More importantly, the goal of our simulation was to find the optimal configuration for resource processing when considering power consumption, resource output, and total time taken to process. Determining which machines may require additional storage could possibly be a functionality to be added to the simulation at a later date, but was not the focus of this project.

The final simplification concerned our input, the raw material blocks encountered by the quarry. In Minecraft, as in real life, it is likely that resources are found in clusters. This means that a real player, were they to mine an area in Minecraft, would likely encounter several iron blocks in close proximity to each other, rather than uniformly distributed throughout the area. For our model, we treated the likelihood of finding a particular resource as a probability that was constant regardless of recently discovered resources. We believe that this is a reasonable simplification for two reasons. The first is that, for the actual resource processing, simplification number two (the input buffer size) largely negated the effects of this. The likelihood of finding resources clustered rather than evenly distributed would be of concern primarily because it could cause the input buffers of certain machines to overfill. As we had an unlimited buffer size, this would not have an effect on our simulation. The second reason is that, were resources to be clustered, the likelihood of finding enough coal early on to fuel the system might be affected. If enough coal were not found, the user would have to begin with a larger supply to ensure the quarry and machines continued to run and to prevent the system from shutting down. As we used an unlimited coal reserve, our system could not shut down, and can give users an estimate of starting coal necessary for each configuration.

**Experimental Setup:**

Each experiment that we ran followed the same general form. We configured the simulation to run all resources through one path, and ran it five separate times for each. Each simulation was run on a quarry of size 64 x 64 x 45, and a new quarry was generated for each run. We ran a total of 36 different configurations, covering all of the different machines, furnaces, and RF generators that were included in our simulation. We then examined the paths according to two separate breakdowns. First, we looked at the outputs from the 12 different possible machine/furnace paths (across all three generator types) to determine which would be optimal for maximizing outputs. We then examined the coal consumption for each generator type, averaged across all machine path types. We focused on these two areas, as we realized that actual power consumption was basically a constant function of the machine type used, and as such would not benefit greatly from simulation. We theorized originally that generator type three, with machine type one and furnace type three, would use the least amount of coal (as it had the highest amount of output per unit of coal), however we were unsure as this generator also ran for the shortest amount of time. As for outputs, we initially theorized that machine path three would produce the highest amount of output, as this path was the most complex and had the highest chances for receiving bonus materials. Given our initial predictions, some of the results were surprising. The results of our experiments are detailed in the three graphs below, and are somewhat surprising.

As we can see, Generator type three was the most efficient generator when it came to coal usage. This would indicate that, for this sort of automation, the ideal generator type is the one that has the highest ratio of power output per coal unit used. This was in line with our expectations, and indicates that the shorter length of the coal burning was not a major factor. More interestingly, the results indicate that machine type three and furnace type three used the least amount of coal. This was surprising, given that we had predicted the opposite (generator type one and machine type three). This would seem to indicate that the EU power type is more efficient in general than the RF power type. As far as total bar output, the highest was machine two with furnace three, just edging machine two with furnace one. Most surprising about the bar outputs was the low number of bars output by paths that used machine path three. This went against our initial predictions, and would indicate that the bonuses afforded by the extra steps of machine type three were not advantageous in maximizing output. Lastly, we examined the average coal used by each machine/furnace path compared to the output of that path. Here, we see that machine type three with furnace type three was the most efficient. This would indicate, again, the relative efficiency advantages of using the EU type.

**Conclusions:**

From the results of our experiments, we were able to determine that the ideal RF generator type to use is generator type three. For maximizing bar output (without consideration of coal usage), machine two and furnace three would be ideal (although machine two and furnace one also performed well). For maximizing efficiency, however, machine path three and furnace type three were clearly the best option. Thus, depending on the goals of our Minecraft player (raw output or energy efficiency) we have results that clearly indicate the advantages of certain choices. Based on these

One of the main shortcomings that our group felt was present in our project concerned the way that machine paths were implemented. We ended up with only three possible paths, two of which were simply one pre-processing machine. If we had the time, we would have preferred to have modeled each individual machine within a path separately. This would have allowed us greater flexibility in designing paths to test, as well as allowed us to better examine the usefulness of individual machines within a path.

We think that this project, while offering some useful insights into the Minecraft automation process, offers many possibilities for further testing. Possible future work could include removing some of the simplifications that we implemented. For example, it may be useful to try to implement resource clustering (something that we were unable to do within the time frame of this project). Another area that we feel would be particularly useful is the implementation of machine upgrades in the simulation. Most of the machines, in real Minecraft, have several upgrade possibilities. While we were not able to implement these, they may have a drastic effect on the efficiency of the system as a whole. Determining which upgrades provide the most cost-effective advantages could be very useful to players, as the upgrades represent a significant initial investment of resources.