A Visualization Tool for Minecraft Actor Properties and Permutations

Cal Poly Computer Science Senior Project

Prepared by: Jillian Polsin

Advised by: Dr. Michael Van De Vanter

Industry Sponsors: Mojang Studios (John Seghers, Adrian Orszulak, Halishia Chugani, John Thornton, Michael Weilbacher)
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minecraft Background</td>
<td>3</td>
</tr>
<tr>
<td>Problem</td>
<td>3</td>
</tr>
<tr>
<td>Opportunity</td>
<td>5</td>
</tr>
<tr>
<td>Actor Properties and Permutation Visualizer</td>
<td>5</td>
</tr>
<tr>
<td>Debug Mob Selector</td>
<td>6</td>
</tr>
<tr>
<td>Impact</td>
<td>8</td>
</tr>
<tr>
<td>Challenges and Solutions</td>
<td>8</td>
</tr>
<tr>
<td>Deliverables</td>
<td>10</td>
</tr>
<tr>
<td>Final Thoughts</td>
<td>10</td>
</tr>
<tr>
<td>Next Steps</td>
<td>11</td>
</tr>
</tbody>
</table>
Minecraft Background

Minecraft is an open world video game with over 120 million players worldwide on over ten different platforms that recently just rose to the top of the best selling games list of all times, just beating Tetris. Minecraft has multiple different gameplay modes ranging from survival mode, where players have to gather items and craft tools to stay alive, to creative mode, where players have all the materials needed to build anything the players want. Minecraft aims to inspire creativity to both players who want to explore the worlds and content creators who want to add their own personal spin on the game.

Problem

One of Minecraft’s biggest draws has been that ability to add new components to the game, also known as modding. Mojang Studios has always done a great job ensuring that content creators have the correct access to the game to ensure that they can add the entities or items into the game that they wanted, but debugging the entities that a creator creates can be difficult, especially when it comes to custom entities (also known as “mobs” or “actors”). When a creator decides that they want to make their own custom mob, a basic layout for the entity is provided to the creator. However, there is very limited in-game debugging for the new entities. Currently, a mob can be selected as a “debug entity” when testing their content in game, meaning the developer now has access to all of the information about that entity and can perform specific debugging actions on the entity that they were not able to before. A picture of a mob being selected as a debug entity is shown below.
Before a mob is selected as the debug entity

After the mob is selected as the debug entity. Note the red box drawn around the entity

However, the current debugging system is mostly textual debugging, except for a few select systems that have a few more debugging features. If an entity performs unexpected behavior while in game, the content creators are expected to debug their entity by hand using only the JSON file they created and the small snippets of textual debugging that are currently in the engine. Additionally, Minecraft is currently transitioning into a new way of writing new actors using something called Actor Properties and Permutations. Because of this, the current debugging tools do not have support for the new Actor Properties and Permutations system. A debugging tool for Actor Properties and Permutations would make the transition to the new system much easier.

In addition to this, the creator currently cannot debug a mob if they cannot see it on screen and interact with it. This can be problematic for mobs like Creepers, who may explode and cause massive damage to the world when interacting with the player in the game.
Opportunity

Actor Properties and Permutation Visualizer

An in-game debugger can be made to help assist content creators with their entities. The debugger can be used to display properties, permutations and the components that connect to those permutations of a specific mob. This can be done with a 3rd party library called ImGui that allows us to display the entity’s information to the developer with tables and node graphs. A node graph is used to visualize which components belong to which permutation and what the value of the components are. This allows for the developer to see exactly which components are active alongside the values that each component has. For example, if a mob were to have two different loot tables depending on what their age is (for example, baby or adult), then the creator will be able to see the values of each loot table in the node graph. The node graph also updates whenever the conditions change. For example, if a wolf grows up and is no longer a baby, the node graph will automatically update to show that transition.

For example, the picture below is an example of what the node graph looks like for mobs that are written using the actor properties and permutations architecture, this picture specifically is representing the wolf written with properties and permutations. The permutation is labeled as active or inactive (in this case looking at the tamed permutation and seeing that the wolf is not tamed), and that there are 3 components that are active when this specific permutations is active: the tameable component, the avoid mob type component and the rideable component. Also note that the connection between the permutation node and the component nodes are color coded to show that that permutation is active. When that permutation becomes inactive, the connections will turn red to show that the permutation is currently inactive.
Debug Mob Selector

The other tool that can be developed to help creators debug actors is a tool that allows the user to select any debug entity without having to physically interact with it. For example of what this might look like, see the first picture of the squid below. The squid is contained in a small closed “aquarium” with water so it doesn’t die. However, there are blocks in between the squid and the player, so the player cannot use normal methods to select it as the debug entity. The second picture is what the tool looks like after the squid is selected. Note that the squid is correctly selected as the debug entity.
Jillian Polsin
Minecraft Actor Permutation Tool Final Report
The debug mob tool will show a menu with all the mobs in a current user defined radius. This will allow the user to easily select debug entities that are enclosed like the squid above. Also, the user can select mobs that could be physically far away from them. For example, if the user needs to debug a zombie in a cave 200 blocks below them, they can pull up the debug window and select that zombie as the debug mob without having to see the zombie or move towards it. In addition to this, the player will be able to switch debug entities faster and with a lot more ease since they don't have to go find the mob and approach it to select it.

**Impact**

The new tools can impact the time it takes to develop new tools for both the content creators that are creating new content packs and the developers who are making new mobs for the game. The content creators and the developers will no longer have to aimlessly search through their JSON code to find what’s wrong with the new mob that they developed if it has unexpected behaviour, they can instead pull up the node editor to see if all the permutations line up the components they thought it would and ensure that the components have the correct values attached to them. This in turn decreases the time it takes to debug mobs when they have undefined behavior. In addition to this, the visual debugging tool can allow for a smoother transition from the existing component system to actor properties and permutations. The creator can build their mob permutation by permutation and see how the mob is being built in the node editor.

The debug mob selector can greatly increase the quality of life for the creators. They no longer have to physically interact with the mobs to debug select them. This decreases debugging time because the creator no longer has to walk over to the mob, find a way to contain it and debug select it to be able to debug the mob. The creator can also switch between multiple debug mobs with ease with this new tool. In addition to this, if a creator has to debug a dangerous mob, like a creeper that would explode on contact with the creator, they don’t have to worry about the mob creating unintended consequences to the world by trying to debug select it. Also, since a creator can now choose a mob to debug without physically interacting with it, they can debug a mob no matter where it is in the world. For example, the creator can now debug select a mob that is in a cave 200 blocks below them, again leading to ease when switching between mobs to debug.

**Challenges and Solutions**

Throughout the development of the project, a few major challenges were encountered:

1. Getting Minecraft itself to build under the current circumstances is difficult as the current build environment is missing a critical submodule -- this proved to be one of the most
difficult challenges in the early stages of the project, but was eventually overcome by carefully cherrypicking the submodule out of build.

2. All of the entity’s data is stored on the server, while ImGui requires all of the information to be on the client. This was solved by intercepting the data as soon as it was parsed to be sent to the client. Luckily, the framework needed to send the data down to the client already exists.

3. Ensuring that ImGui has the capability to render the information we want to show the way we want it to show and no other library is available. This did not end up being as big an issue as it could have been, as ImGui was able to represent the data we wanted perfectly without any major additions to the ImGui code.

4. How do we determine a unique name for each component?
   - The node editor requires that all the names of the inputs and outputs are unique. However, Minecraft allows for a component to exist at multiple different stages in an entity’s life span.
   - While the same component can exist at multiple places in the entity's life span, the data in the component can be different. For example, both a baby sheep and an adult sheep have the loot table component, but the loot for a baby sheep is extremely different from an adult sheep. How do we represent that the component itself is the same, but the data is different?
   - The issue of identity also needs to be considered at this time. Minecraft views the component’s identity as the whole component and its data while ImGui views the components identity as just the name of the component. If the name needs to be unique, how do we make it unique in a way that the information can correctly be transferred from the server to the client and vice versa without mass confusion between the two systems?
   - This sort of identity issue was fixed by manipulating the data on the client side to contain a unique ID that matches the specific component with the permutation so ImGui knows which permutation belongs to which component. Even though this was fixed, this is something we’d like to see ImGui handle to replace the string manipulation that happens on the client.

5. How will future tool developers be able to get the data from the components in string format? The data’s string version is discarded very early in the entity parsing process, and it’s unclear if we can send that data to the client before it is discarded. Luckily, a class already exists to keep the data from the JSON throughout the lifespan of the game and contains a method to display the JSON in the desired format.

6. Determining a unique name for each actor. There may be thousands of squids in the user selected radius. How do we represent them as unique mobs? They all have the same identifier, minecraft:squid, and they need to be told apart so the debug entity can be selected appropriately. This was fixed by mapping all Minecraft identifiers to a specific ID and suffix combo so the mob can be kept and rendered uniquely.
Deliverables

The main goal of this project is to not only to help users debug actor properties and permutations, but to create a foundation of the Minecraft tool set that could enhance the usability of all tools. The main goal of the permutation tool is to transfer the information about the actor permutations into the node editor. The node editor shows which components are related to each permutation, as well as the information about components themselves (for example, which loot table the minecraft:loot component has). Note that the node editor is not currently a live editor for actor properties and permutations, but just a visualizer for them.

After all of the data is transferred to the server correctly and is displayed correctly, a feature for selecting mobs will be added to the entire tool architecture. This feature will allow users to filter mobs by type and range so they can be more easily selected for debugging. For example, some entities, like a creeper or the example of the squid from up above, may need to be kept in enclosures so they don’t wander off and destroy some aspects of the world. However, currently, you can’t select a mob that is kept in an enclosure because the blocks in front of the entity keep you from physically selecting the entity. This new feature will fix that issue by allowing you to not be directly next the entity to select it for debugging.

Final Thoughts

ImGui is a great tool to display the actor properties and permutations in a way that would benefit the creator when they’re making new content. It allows for easy visual debugging of the new architecture with lists and node editors and gives the creators the ability to switch debug entities quickly and with ease.

However, there is a problem with the idea of identity in ImGui and the Minecraft engine that makes working with the two problematic. ImGui doesn’t have a great way to make a tool like the actor properties and permutations node editor that allows for components that have the same name. Every node input and output needs a completely separate and unique name that requires string manipulation in the ImGui code. If the tool wanted to be used as a live editor, all of the actor’s original information would have to be mapped to the unique name and we would have to ensure that the right version of the name was sent to the client and the server. However, a lot of the pipelines were already built to transfer the information about which mob is the debug entity, so we didn’t have to deal with the identity issue too much in this regard. But with that being said, the identity issue should be addressed before ImGui can be seamlessly used as a live editing tool.
Next Steps

Since the Minecraft code was forked to develop the debugging tool, a lot has changed in regard to the actual architecture of the actor properties and permutations. The code will need to be updated to ensure that the tool can correctly render all of the new information and intercept it at the correct point in time on the server side. In addition to this, some of the code that could be applied to more than one tool could be abstracted to ensure that all tools can access it correctly with the correct parameters and information.