UNIVERSITY OF CENTRAL FLORIDA

Measuring Collaboration in Minecraft

April 28, 2021

Group 21 (Minecraft Black Team)

Chloë Geller, Connor Austin,
Andrew Amado, and Anthony Yu

Supervisors

Dr. Gita Sukthankar
University Central Florida
Associate Professor

Contents

Executive Summary

The main goal of this research project starts with the simple task of studying collaboration in Minecraft through the use of visualization tools. We will be focusing on the development of these visualization tools. This document will thoroughly go over every detail of what we plan on doing. Here is a rough overview.

This project will be building off of a previous project titled HeapCraft. HeapCraft is a Minecraft server plugin that records player activity, such as movement, mining activity, deaths, etc. This is the data that we will be using for the visualizations.

We will be creating a web interface, which we will refer to as the hub. We give it this name because its purpose is to serve as a home for any and all visualization tools related to this project. This will pack everything we need to analyze the data given by HeapCraft into one area, where everything can be easily found.

The primary visualization tool that we will be creating is an interactive heatmap. This will plot all of the user events collected on to a canvas. They will be organized internally by time, and be displayed according to the location that the event occured. This heatmap will have both a 2D and a 3D version. 2D for a simple top-down view of the world if height information is not needed in the analysis. The 3D view is intended for a more detailed analysis, where height information would be needed, such as information for players building tall bases, or digging underground.

As the previous HeapCraft project did before us, we are leaving room for expansion with the creation of the hub, and starting the collection of the hopefully many visualization tools that will be added, with the 2D/3D heatmap. All of this aims to achieve the goal of helping researchers analyze Minecraft player activity to determine how players collaborate, communicate, and behave.

Chapter 1

Introduction

1.1 Project Narrative

Minecraft is a game played and enjoyed by millions of people all around the world. The rules are simple: build, craft, and survive. Though from these simplistic rules, complex behavior can emerge. As said by Dee Hock, ”Simple, clear purpose and principles give rise to complex intelligent behavior.”. This statement has been proven many times before with works such as John Conway’s game of life, where only four rules gave life to some of the most complex behavior ever seen from a computer program at the time.

There are many unwritten objectives in the game of Minecraft that players must achieve in order to have an enjoyable playing experience. For instance, one of these objectives would be to build a house, or some sort of shelter. At no point does the game direct you to build a house, or even take shelter, and there are no game-breaking disadvantages to not completing this objective, however most players do it anyway, why?

This project aims to analyze Minecraft player behavior, such as the previous example, and more. Given the framework that this project will be building off of, HeapCraft [1], we have the ability to both collect player data such as movement, actions, chat interactions, and much more. Given these data points, we plan to build tools to analyze and visualize certain aspects of this data where we hope to see complex emergent behavior that can then be studied and applied to real-world situations.

There is a field of psychology that this project belongs to called *Theory of Mind* (ToM). In a sense, Theory of Mind is the ability to attribute mental states to ourselves and others, serving as one of the foundational elements for social interaction. Having a theory of mind is important as it provides the ability to predict and interpret the behavior of others [2]. Using Minecraft as a medium, we will be taking advantage of the fact that our subjects are human, and the in-game behavior can be translated to real-life behavior in many cases, with the hopes of leading to new developments in the field of Theory of Mind.

1.2 Individual Motivations

Chloë Following my undergrad, I plan to pursue a PhD in the Computational Cognitive Sciences, using tools from Artificial Intelligence and Computer Science to computationally ground our understanding of decision making and social cognition. Particularly, I want to understand how we think about others and use that to inform our next steps – primarily in a language-oriented setting. This project is a great testbed to continue growing my computational skills, while also thinking of Theory of Mind, a crucial component in thinking about others. More immediately, this project will provide invaluable experience that will further prepare me for my summer research experience where I'll be working on social cognition and decision-making.

Connor Big data has been an interest of mine ever since I learned of its existence. This fondness for big data naturally led to affinities for other fields such as statistics, data visualization, and most of all, artificial intelligence. Being able to take a bunch of data, feed it into a machine, and have a comprehensible result come out on the other end is beautiful. I have decided to take advantage of and help further this cutting edge technology that we now have access to at our fingertips.

In this context, Minecraft is just a medium through which we are achieving our goals, but the underlying theme of big data with the possible implementations of artificial intelligence is still very much alive. Being able to combine many of my technical passions with a childhood game of mine just makes this project that much better.

Andrew Ever since I was introduced to Minecraft in my younger years, I always had a fascination about it. A world that is seemingly infinite, with no real rules to it. I have been a long-time player of Minecraft and it is what actually introduced me into the Computer Science field and programming. I started with the basics of the Bukkit API and worked my way up with countless hours of programming and debugging.

During the proposal period, I was not sure if I would be able to find a project that I felt I would really like to participate in until I heard this project about Minecraft. As soon as I saw the logo in the presentation, memories of my time coding server plugins came back arose, and my excitement grew ever so quickly. I would love to get back into coding Minecraft Plugins especially for a project such as this. My first thought I was that I needed to be apart of this project since it is something I had experience in, and now I am. I am willing to learn about the subjects that are required in order to succeed and make this project the best that I can contribute to. To me, it's crazy how the very first thing that introduced me to programming is now something I must revisit in order to complete my degree.

Anthony I am interested in this project because of my deep attachment to Minecraft. This has been my childhood game since I have played since the beginning. Whenever I was playing Minecraft, it was always in multiplayer, whether with friends and family or in an online server. So for me, Minecraft has always been a multiplayer game that should be enjoyed with others. While I have always played Minecraft with others, I never considered how much real interactions I have with the people around me. Learning that there is a research project that plans to explore that field caught my attention.

At the same time, as a Computer Science major, I have yet to find the field that I want to pursue. There were a few that have peak my interest, one of them being Artificial Intelligence and it's surrounding fields like Machine Learning and Deep Learning. Upon learning that there was a project that involves many of my interest, I couldn't pass it up.

1.3 Broader Impacts

The broader impacts for this project come from the project parent sponsor, The Defense Advanced Research Projects Agency (DARPA) [3]:

> Humans intuitively combine pre-existing knowledge with observations and contextual clues to construct rich mental models of the world around them and use these models to evaluate goals, perform thought experiments, make predictions, and update their situational understanding. When the environment contains other people, humans use a skill called theory of mind (ToM) to infer their mental states from observed actions and context, and predict future actions from those inferred states. When humans form teams, these models can become extremely complex. High-performing teams naturally align key aspects of their models to create shared mental models of their environment, equipment, team, and strategies. Theory of Mind and the ability to create shared mental models are key elements of human social intelligence. Together, these two skills form the basis for human collaboration at all scales, whether the setting is a playing field or a military mission.
>
> Artificial intelligence (AI) technologies have made little progress in understanding the most important component of the environments in which they operate: humans. This lack of understanding stymies efforts to create safe, efficient, and productive human-machine teams. The Artificial Social Intelligence for Successful Teams (ASIST) program seeks to develop foundational AI theory and systems that demonstrate the basic machine social skills needed to infer the goals and situational knowledge of human partners, predict what they will need, and offer context-aware actions in order to perform as adaptable and resilient AI teammates.

We hope our work will also help researchers answer the following questions:

- Do psychological theories of human team cognition apply to human-agent teams?
- Can we create socially intelligent agents capable of exhibiting theory of mind?

1.4 Legal, Ethical, and Privacy Issues

1.4.1 Ethical Issues

One of the more problematic Ethical issues our project faces is regarding the collection of human data. Our first step of many to this process is to complete the CITI Research, Ethics, and Compliance training, something that will be completed by the end of February. By completing this training, we would have been introduced to social-behavioral-educational research with a focus on the protection of human subjects. Throughout this course, we are presented case studies that are both historic and current that portray the key concepts of the course.

Another step we can take to move past this roadblock is getting our project approved by the Institutional Review Board (IRB). In order to officially collect our data and move on with this project, we must submit the project to the IRB Committee at the University of Central Florida (UCF).

1.4.2 Privacy Issues

Of course, there is a privacy concern when any sort of data collection occurs. To ease up on this, we will not collect any data that is self-identifying to our participants. Our collection efforts are mostly focused on player actions and interactions inside the world in which they are playing the game.

1.4.3 Legal Issues

A possible legal issue we need to consider is the Children's Online Privacy Protection Rule (COPPA). The summary of COPPA's rules [4] are stated below:

> COPPA imposes certain requirements on operators of websites or online services directed to children under 13 years of age, and on operators of other websites or online services that have actual knowledge that they are collecting personal information online from a child under 13 years of age.

To prevent being in violation of this rule we plan on asking those volunteering to participate to fill out a survey prior to joining the Minecraft server. The first question on the survey will ask participants if they are above the age of 13, if they are not, they will not have access to the server.

Since we are using Minecraft, we can also take advantage of the fact that they enforce these rules [5] as well:

> Mojang is a subsidiary of Microsoft Studios. As a US company, Microsoft implements and follows certain procedures to comply with the Children's Online Privacy Protection Act (COPPA).

However, we are aware that we must prepare for those who break the rules. As such, if it is brought to our attention that a participant was under the age of 13, we will immediately remove their data from our server, as well as banning them from the Minecraft server.

Chapter 2

Initial Requirements

2.1 Data Collection Plugin

This plugin is very important as it is considered to be the core of our project. Without its function, we would not be able to progress further in this project. This is because all of the data that our other plugins and creations will use will come from this particular plugin. This would be one of the first things our team would like to knock out first on the to-do list because of that aforementioned reason.

2.1.1 Minimum Requirements

- We would like to rework HeapCraft's Epilog plugin from the ground up so that it works for the latest version of Minecraft. As it stands now, the Epilog plugin seems to have been tested for up to Minecraft 1.12.1, which in itself was released on August 3, 2017. Having said this, our group thinks it would be best to re-create the plugin for the latest version of the game using some of HeapCraft's ideas while adding in our own as well.
- Our data needs to include everything we need to have the other components of our project fully functional. This is why it's important we decide which data we will collect and leave out and also how we format our data. Our initial plan is to use JSON and a Mongo database, but knowing what we will need would prevent unnecessary data from being collected.
- We would need to decide which server events we would be using for our data. Knowing this is important because all of our data will come from our chosen events. In our API of choice, there exists a vast variety of events, some of which are deemed to be unnecessary for our purposes and would be discarded. Our plan for deciding which events we would like to use would be to go over HeapCraft's Epilog plugin and see which events they used as a basis. After that, we would proceed to take a look at our API's documentation to further look at all of the events listed and then decide which ones we would be interested in using for our project.

2.1.2 Stretch Goals

- Include a command to help guide a new user on the process of using this plugin. This would allow players who would like to learn more about this plugin and what it does their opportunity to do so.
- Create a video tutorial of the use of this plugin. This could be especially useful to people who have an easier time learning visually. This would be an additional resource for players to learn more about the plugin.

2.2 Automated Friend's List

This project works along with our data collection plugin. In short, the idea is one of the metrics we will be collecting is time spent near a player. The more time you spend with another player, the more likely they are to be one of your friends, if not already.

2.2.1 Minimum Requirements

- Due to the fact that this plugin works in conjunction with the data collection plugin, and since that plugin sends all of its data to our database, it is important to have the automated friend list establish a connection to the database so that we may continue with adding functionality to this plugin.
- After we establish a connection, we will send a query to the database that will retrieve the data we need to begin adding players to another player's friends list. The data that we will be using for this would be time spent around another player.
- Once we retrieve the necessary data, we will start the process of formulating the friend list for a given player. Most likely, we would convert the data we received to some sort of data structure so that we may quickly pull the players with the most spent time together and add them to the friend's list.
- It is important that we create command-based functionality. This means that players would be able to mainly add, remove, and show the current players that are listed on their friend's list.

2.2.2 Stretch Goals

- Once we have the command functionality completed, our next goal would be to create a graphical user interface-based functionality. Without API, we are able to use chest inventories to give this effect. We would use this and add items to these inventories that players may not be able to take, but instead, clicking those items would essentially either add, remove, or show the players that are listed on their friend list.

2.3 DiviningRod Rework

Our group believes that this plugin is a wonderful idea in addition to their Epilog plugin. In short, this plugin allows users to find other users in the server with specific tags that may be read more about on their website or in the explanation that is located further down in this paper. Despite this plugin being a good addition, we believe that we may improve upon its initial functionality.

2.3.1 Minimum Requirements

- The main objective of this plugin is to locate other players, but in its current state, the plugin utilizes renaming a stick that is given to a player. Instead of being called 'Stick', it is instead renamed to the distance between the target player, and the player that is holding the stick. We think instead of using sticks as the previous creator did, we will instead utilize Minecraft's compass item as it proves to be more intuitive to use. This is because a compass is capable of pointing to some target location in the game. We may use the API to modify where this compass may point.
- One of the obstacles that the previous creator may have faced was an abundance of items being spawned in. This is because in order to use this plugin, a stick would be given to a player and that player would then locate another player. This would leave a lot of illegitimate sticks to be spawned in the game, and may have been a reason as to why the creator chose to stray away from compasses. Our goal is to find a way to prevent the given compasses to be exploited.

2.3.2 Stretch Goals

- As it stands now, we are giving the players the ability to find others with tags that are built inside of the current plugin. A good stretch goal would be to create new tags for players to use to find other players in certain criteria.

2.4 Heat Map

The Heat Map is our main visualization tool. Our goal for this part of the project is to give the server administrators or whoever may have access to the data the ability to see the results of the data that has been collected on a particular Minecraft world. Users would be able to do things such as sorting the data to include or omit certain events from the existing data.

2.4.1 Minimum Requirements

- Displays heat map of player positions over time
- Displays heat map of all other events over time
- Each cell on map can be interacted with and show extra details
- 3D heat map for an extra dimension of visualization
- Basic filters to single out certain data points

2.4.2 Stretch Goals

- Advanced filters for heat map for deeper analysis
- Room for extra visualization tools

2.5 General Minimum Requirements

- Reliable framework to collect player metrics from the server.
- Recording the position of players. Involve x, y, z, chunk, world, direction, pitch, and yaw.
- Recording the action of building within a designated plot of land.
- Visual interpretations of gathered metrics.
- Hosting an experimental server to ensure that the tools work.
- Hosting a server to collect data.
- Meeting with our sponsor, Professor Sukthankar, at least once a month.

2.6 Within Reach

- Using the data collected from our Minecraft server to train models.

2.7 Stretch Goals

- Creation of a mini-game in which players will have be formed into teams and will work together to reach some goal.
- Creation of multiple server using the “private worlds plugin” in HeapCraft.

Chapter 3

Projected Budgetary Requirements

Our team has been allocated a total of $500. Throughout Senior Design 1, we don't plan on touching this money very much as testing can be done from our computers, so we did not deem it necessary. The funding will more or less come into play during Senior Design 2. We plan to allocate some of the money towards servers/VPS to run our plugin and keep the server live 24/7. This will ensure that once we do have our plugin created, we will have ample opportunity to collect data. Afterward, as discussed with Professor Sukthankar, anything else we will have remaining will be put towards gathering research participants and perhaps advertising of the server to allow external participants to play on our server.

3.1 Additional Expenses

Though the main budget will be allocated towards server hosting, there are other expenses to keep in mind. These expenses include advertising and unexpected situations such as server crashes and high player count. Having an unexpected high player count means that we will need to upgrade to a server with more resources, costing us more money, putting more strain on our already tight $500 budget.

3.2 Actual Expenses

By the end of the project, we had only two areas of expenses, the MongoDB database, and the AWS Server. The AWS server cost at the time for an EC2 Instance with an M5 Large cost us roughly $84.31 a month. We ran the server for roughly two months, for an estimated total of $168.62. The reason we aren't exact with the cost is that the invoices for the server cost went to our sponsor. The other major cost is the MongoDB Atlas. This amounted to $99.49 dollars exactly. By the end of the project, out of the $500 we were given, we used $268.11, with $231.89 leftover.

Chapter 4

Projected Timeline

4.1 Spring 2021

4.1.1 February

February is composed of setting up infrastructure amongst the team. It will also be the month of brainstorming what idea to collect and to complete the CITI training module. Mostly during this phase of the project, we will be taking time to figure out the workflow and understand what is required to be successful during our time in Senior Design. This step will prove to be very important as these ideas will act as a framework and get the ball rolling.

- Feb. 16 - On this date, we are having our first TA Check-in. We will be going over what we have done so far in terms of the document and will bring along any questions we may have had leading up to that meeting.
- Feb. 18 - We are planning on having at least the Project Requirements/Initial for the design document completed. Meeting this goal should put us in a healthy place in terms of pace with this document.
- Feb. 21 - Feb. 27 - For our project, a big chunk of it involves collecting some sort of data. HeapCraft, the project that we were given as a basis, already collects data that is or would be similar to the type of data we would be planning on collecting. Since HeapCraft is more so out of date, deciding on the types of data and interactions to collect from players is another important step in the process.
- Feb. 28 - Since our project seems to be more of a research project, and due to the fact that we will be collecting data from players in our server, we must complete the CITI Social / Behavioral Research Investigators and Key Personnel training module. Doing this will give us the opportunity to learn the proper protocols for handling the collection and use of data.

4.1.2 March

At the beginning of the month, we should have already completed the CITI Research training, but if that has not been fully completed, we are allocating the first week of March to doing so since it was discussed during our sponsor meeting for the training to be

Task	Start - Finish	Status
TA Check-In I	2/16	
Project Requirements Completed	2/18	
Deciding what types of data we are collecting	2/21 - 2/27	
Complete CITI Program	2/28 - 3/7	

completed at the end around the end of February or beginning of March. Additionally, in order to begin the development of our data collection server plugin, we need to research the API that will allow the plugin to function. For this project, we are intending on using the PaperMC API. More information about this API can be found further down in the document, but essentially PaperMC is the most optimized, efficient API compared to our other options.

- Mar. 1 - 7 Complete CITI Social / Behavioral Research Investigators and Key Personnel training if not yet completed.
- Mar. 8 - 28 - We will research the PaperMC API and look at various tutorials on server-sided plugin development. This will allow us to get acquainted with the API so that we do not seem very lost when getting started.
- Mar. 11 - This meeting is our first major status meeting. We will construct a short presentation about what our project consists of, its goals, how we are planning on achieving those goals, and who in our group is leading which division of work. We will then explain everything we have so far with the professor and at the end show our table of contents at that point.
- Mar. 29 - This is our second TA check-in. We will again be showing essentially what we have so far and any general questions we may have or ask for any general advice.

Task	Start - Finish	Status
Research PaperMC API and plugin development	3/8 - 3/28	
Dr. Leinecker status meeting	3/11	
TA Check-In II	3/29	

4.1.3 April

This is more so our last month for Senior Design 1. Our main goals are to have the design document completed by our deadline of the 28th of April. During this time, we should have figured out a general idea of what we plan to make and have the technologies ready to be used. On top of that, ideally, we have begun working on a rough draft of the UI.

- April. 2 - This will be our first meeting out of many throughout this month. For our first meeting, we will focus on working on our document so that we may meet our deadline.
- April. 9 - Our second team meeting will be for deciding on what technologies we are planning to use and how we plan to use said technologies.

- April. 11 - April. 17 - Our third team meeting will focus on making rough drafts for the UI and UX. By the end of this week, we should have gone through a few iterations to make the UI look cleaner and the UX to be pleasant.
- April. 21 - Our fourth meeting will be another meeting for the Design Document. As we approach one week before the deadline, we must start to put the finishing touches into the document and get ready to submit it before then.
- April. 27 - Our fifth meeting will be a final check to make sure we do not have any errors such as spelling errors, grammar, syntax, and incorrect information.
- April. 28 - This is the deadline for the final design document for the class. At this point, we would have added any last ideas to the document and submit our paper.

Task	**Start - Finish**	**Status**
First Meeting about the Document	4/2	
Second Meeting about Technologies and Usage of Them	4/9	
Third Meeting about making UI drafts	4/11 - 4/17	
Fourth Meeting about the Document	4/21	
Fifth Meeting about Final Check of Document	4/27	
Final Design Document	4/28	

4.2 Summer 2021

4.2.1 May - August

While we are not technically in class since we are scheduled to take Senior Design 2 in the fall, we are planning on meeting throughout the summertime while some of us do not have a classwork load on our plates. At the beginning of this month, our goal is to begin working our IRB paperwork . Afterward, we will decide what we can do to get a head start and begin developing our data collection tool. This is what we will end up using the PaperMC API to code up.

- May. 1 - This will be our first team meeting in the summer. During this meeting, we will begin to address how to start and turn in the IRB paperwork.
- May. 2 - Aug. 31 - Begin working on IRB Paperwork.
- May. 28 - Aug. 31 - Since this project has a couple of components to it, it is important that we get a head start in the game so that we may ease ourselves throughout the fall as it is our last semester at UCF and we may be facing final difficult classes. Our main objective by the end of the summer break is to have at least most of the groundwork completed for the data collection tool.
- Aug. 31 As we mentioned before, but at the end of the summer, we are planning on having the groundwork completed, however, if it is possible, it would be wonderful to have a base working version of our data collection plugin.

Task	Start - Finish	Status
First Meeting about beginning IRB Paperwork	5/1	
Work on IRB Paperwork	5/2 - 8/31	
Begin Working on Data Collection Tools	5/28 - 8/31	

4.3 Fall 2021

4.3.1 September

We are now in our fully-fledged development phase of the project. We have a beta of the data collection tool and we should continue to add onto it as necessary. During this phase, we will be focusing on our decided player events to add to our data collection plugin. Most of those events should be listed in the Paper docs. Throughout the month we are to begin our visualization tool creation to have a better understanding of the data that we will have.

- Sept. 1 - Project Status Meeting.
- Sept. 1 - Sept. 30 - Continue working on IRB Paperwork.
- Sept. 1 - Sept. 30 - Continue working on finalizing the data collection tool.
- Sept. 1 - Sept. 30 - Begin creation of visual representation tools for collected data.
- Sept. 6 - Team meeting & progress check.
- Sept. 13 - Team meeting & progress check.
- Sept. 17 - Before we can turn in the IRB Paperwork to the IRB, we need to give it to our sponsor for review. We plan to give Professor Sukthankar and her Ph.D. students about two weeks to review the IRB documents and to give us feedback.
- Sept. 20 - Oct. 6 - Whenever we are given the go sign to submit the IRB Paperwork by Professor Sukthankar, we will submit the Paperwork. We plan to give about a month and a half for the IRB to be approved. This time also includes any documentation revision that the IRB might ask for.
- Sept. 20 - Team meeting & progress check.
- Sept. 23 - Sept. 30 - Project Status Team Meeting with Heinrich.
- Sept. 27 - Team meeting & progress check.

4.3.2 October

The beginning of October consist of us getting everything finalized for the server to be up and running. This includes the data plugin, IRB approval, and the actual server. Once the server was up and running, data collection, server management, and visual representation tools became priority.

- Sept. 27 - Oct. 4 - Preparing for the CDR Presentation
- Oct. 4 - CDR Presentation

Task	**Start - Finish**	**Status**
Project Status Meeting	9/1	
Continue working on IRB Paperwork	9/1 - 9/30	
Continue working on finalizing the data collection tool	9/1 - 9/30	
Begin Creating of Visual Representation Tools	9/1 - 9/30	
Team Meeting and Progress Check	9/6	
Team Meeting and Progress Check	9/13	
Getting approval of IRB Paperwork by Professor Sukthankar	9/17	
Submitting Paperwork	9/20 - 10/6	
Team Meeting and Progress Check	9/20	
Project Status Team Meeting with Heinrich	9/23 - 9/30	
Team Meeting and Progress Check	9/27	

- Oct. 4 - Team meeting & progress check.
- Oct. 5 - Meeting with UCF IT to set up server.
- Oct. 6 - Finish Data Collection Plugin
- Oct. 6 - Getting approval of IRB Paperwork by IRB commitee.
- Oct. 6 - Getting Minecraft server up.
- Oct. 6 - Setting up Minecraft server with plugins.
- Oct. 6 - Nov. 26 - Maintaining the server.
- Oct. 11 - Oct. 31 - Continue working on visual representation tools.
- Oct. 11 - Team meeting & progress check on visual analysis tool.
- Oct. 18 - Team meeting & progress check on visual analysis tool.
- Oct. 25 - Team meeting & progress check on visual analysis tool.

Task	**Start - Finish**	**Status**
Preparing for the CDR Presentation	9/27 - 10/4	
CDR Presentation	10/4	
Team meeting & progress check	10/4	
Meeting with UCF IT to set up server.	10/5	
Finish Data Collection Plugin	10/6	
Getting approval of IRB Paperwork by IRB commitee	10/6	
Getting Minecraft server up	10/6	
Setting up Minecraft server with plugins	10/6	
Maintaining the server	10/6 - 11/26	
Continue working on visual representation tools	10/11 - 10/31	
Team Meeting and Progress Check on Visual Analysis Tool	10/11	
Team Meeting and Progress Check on Visual Analysis Tool	10/18	
Team Meeting and Progress Check on Visual Analysis Tool	10/25	

4.3.3 November

As our visualization and collection are almost to fully complete, we must prepare our final presentations. Alongside this, we will be rehearsing our presentation to make sure that when it does come to presentation time, things will run smoothly.

- Nov. 1 - Nov. 5 - Heinrich Demo
- Nov. 1 - Team meeting & progress check.
- Nov. 8 - Team meeting & progress check.
- Nov. 15 - Team meeting & progress check.
- Nov. 22 - Team meeting & progress check.
- Nov. 22 - Dec. 1 - Final Presentation Preparation.
- Nov. 24 - Finish with Data Representation tools.
- Nov. 30 - Closing of Minecraft Server.

Task	Start - Finish	Status
Heinrich Demo	11/1 - 11/5	
Team meeting & progress check	11/1	
Team meeting & progress check	11/8	
Team meeting & progress check	11/15	
Team meeting & progress check	11/22	
Final Presentation Preparation	11/22 - 12/1	
Finish with Data Representation tools	11/24	
Closing of Minecraft Server	11/30	

4.3.4 December

This is the end of Senior Design.

- Dec. 1 - Final Presentation
- Dec. 3 - Fall 2021 Class ends.

Task	Start - Finish	Status
Final Presentation	12/1	
Class Ends	12/3	

Chapter 5

Team Operations

5.1 High Level Diagram

The following diagram displays the rough path that our data will take while the server is running, from collection to visualization, as well as everyone's role in the process.

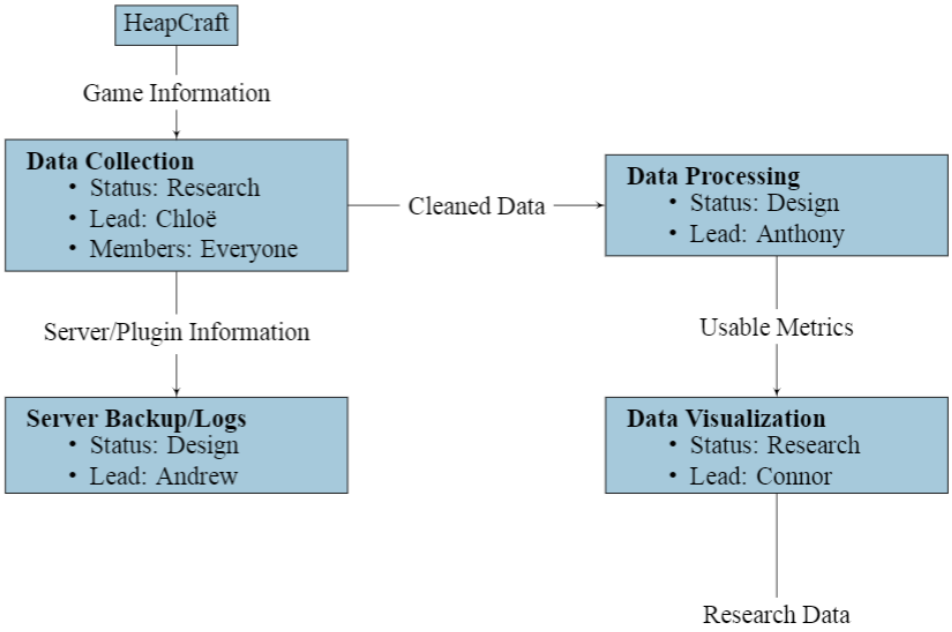

Figure 5.1: Role Diagram

5.1.1 Division of Tasks

In a team setting, it is important that we assign duties and roles for each team member. Doing this will not leave any ambiguity or confusion when it comes to what someone has to do at any given time. For our project, our plan is to divide our development phase into two teams.

The first team will be in charge of Minecraft plugin development. The second team will be in charge of web development. Throughout the development process, each team will be meeting separately every other week while both teams will be meeting weekly to go over progress and whether or not team members have met their goals.

Team 1 - Minecraft Plugin Development

Our first team will be focusing in Minecraft plugin development. This means that this team will be creating the Data Collection Core plugin.

The development of the Data Collection Core plugin will take the highest precedence because as it says in the title, this plugin is the core of our entire project. It is very important we get this plugin working and ready to go.

Members that will be assigned to this team are:

- Andrew
- Anthony
- Connor (support)
- Chloë (support)

For this team, the main team members are Andrew as the lead of this team and Anthony. We also will include the rest of the group as supporting members that are existing for if the team is stuck on a problem.

Team 2 - Web Development

Our second team will focus on creating the web applications that will go along with our project. The main project will consist of the HeatMap and the Web Hub that will host any other tools that we may like to add in the future.

Since there are multiple projects to be done, there has to be some order in which these projects will be created. The priority list is as follows:

The priority list for this team will be as follows:

- Web Hub
- Heat Map

The Web Hub will take priority over the Heat Map because the Web Hub is the basis and is meant to be holding all of our visual representations. This project is meant to modular, meaning adding and removing a new or existing project to the hub should be fairly easy.

Members that will be assigned to this team are:

- Chloë
- Connor
- Andrew (support)
- Anthony (support)

For this team, the main team members will consist of Chloë as the lead of this team and Connor. We decided to add Andrew and Anthony in the team, but only as supporting members where if the main team members are stuck on a problem which requires more time to solve. After the Minecraft Plugin Development Team finishes the Data Collection Core Plugin, the team will shift its focus on assisting the other team in any way they can.

Chapter 6

Project Decisions

6.1 Why Minecraft?

Released in late 2011, Minecraft has found major success over the course of its life. The game has sold over 200 million copies. Since launch, if we are taking the numbers, that is about 58,000 a day. Needless to say, Minecraft is a big game. With Minecraft being a massive multiplayer sandbox game, interactions between players are a given. For this project, we needed a big game for the purpose of measuring those interactions. There really is no other game like Minecraft that allows people to roam around with friends and have a good time together.

6.2 Client vs Server Mod

In Minecraft, there are two types of mods for the game. Server mods and Client mods. While the aim for both of these is to bring a bit of spice into the game, there are different situations where you would like to use one or the other.

6.2.1 Client-sided Mods

The aim for a client-sided mod is to create and add in unique features into the game such as blocks, monsters, or anything else that does not relate to the vanilla version of Minecraft. In order to create a client-mod, you will need to first decompose Minecraft's code and add anything you would like on top of that Minecraft code. These mods must be installed on the client before launching the game in order to function at all.

6.2.2 Server-sided Mods

Server-sided mods are more known as plugins. While plugins are aimed to also add unique features in the game, the way of approaching and reaching this goal is a bit different than our client-mod counterpart. Firstly, a plugin will run on an existing modded version of vanilla Minecraft. Plugins are dynamically loadable meaning if I would like to add a plugin while the server is running or maybe while players are online, I can do that without the players having to install the plugin on their clients. Overall, when creating a plugin, you are not creating new things in the game. Instead, you are using what vanilla Minecraft currently offers and manipulate it in such a way to bring something new to the table.

6.2.3 The Decision

After speaking about the two types of mods a developer may use for Minecraft modding, our group will be choosing the plugin route. This is because for the purpose of this project, we do not need to create anything super solid inside of the game. We are just going to be recording player interactions that happen inside of a server. We also do not want a player to have to go through the hassle of installing a client-sided mod. For newer users, this particularly can be quite troublesome and may steer potential players away from our server. Not only that, but if we did decide to use a client-sided mod, we would also have to use Forge for Minecraft which enables client mods to be used on the server. That in itself can be very tricky to set up and get everything in working order.

6.3 API for Development

Now that we know we will be developing a server plugin, we must choose what API to use to create it. Deciding which API to use for our plugin is no easy decision. We must take into account all our major options and weigh them against each other. Our three options for this project include Bukkit, Spigot, and Paper. Bukkit has been covered in an earlier section of this paper, so we will only cover what Spigot and Paper are about.

6.3.1 Spigot

According to Spigot's website, Spigot is "a modified Minecraft server based on CraftBukkit which provides additional performance optimizations, configuration options, and features, whilst still remaining compatible with all existing [Bukkit] plugins and consistent with Vanilla Minecraft game mechanics" [6].

In general, Spigot is a subsection of Bukkit but with its own twist. You may use Spigot to develop your Minecraft plugins as you would when using Bukkit because Spigot includes most of the Bukkit API inside of it. Along with this, Spigot contains performance improvements which are especially useful for larger servers since memory and CPU usage would be lower by using Spigot. Spigot also offers backwards compatibility with any other existing Bukkit plugins. So even if a developer solely used Bukkit for plugin development, they may use that plugin in a Spigot server with no issues.

6.3.2 Paper

Believe it or not, Paper is a subsection of Spigot. The goal of Paper is to outdo Spigot in terms of performance and add in their own features unique from Spigot and Bukkit. In short, there isn't much to be said about paper other than that it is better optimized than Bukkit and Spigot, can support all Bukkit and Spigot plugins, and it adds unique features to their API that Bukkit and Spigot don't have. Figure 6.1 displays the three different API's.

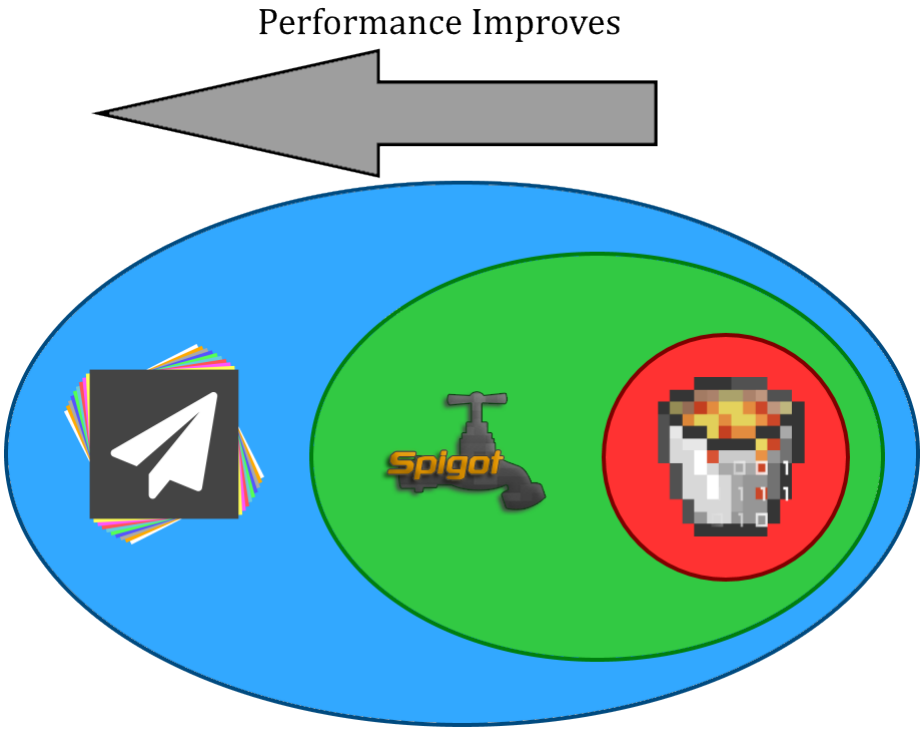

Figure 6.1: Comparison of our three API choices

For our project, we will be utilizing the Paper API for our plugin development and our source for hosting a modded version of vanilla Minecraft. This is because Paper is backwards compatible with Spigot and Bukkit Plugins while having extra features of its own. It's also worth noting that Paper is faster than Spigot which is also faster than Bukkit in terms of performance and optimization.

6.3.3 Data Storage

With any data collection plugin, there are usually a couple of methods to choose from when deciding where to send and store all of the data. Making this decision is important because you want to make sure that the application is reliable. In our group, There are two methods of choice we were considering when storing our data.

Locally

For data collection, an option we can take would be to store all of our data locally. Whenever there are extra files that are created for a plugin in a Minecraft server, those files will end up in the `/plugins` folder. The folder has the same name as the plugin name that is declared in the `plugin.yml` file. In that folder, it has all the files that are created during the plugin run time. Of course, these are updated in real-time.

There are some advantages to storing locally. Some of these include:

- All data is stored into the hard drive.
- Swiftness in storing
- Only accessed when someone accesses the machine.

One major disadvantage of storing our data locally is readability. In our folder, if we opted to store all of the data we needed inside one document, eventually, we would have mountains upon mountains of data inside one document. This will be a problem because using this data would be difficult for our applications.

The shortcomings of local data storage are:

- Devices have the potential to fail
- Not suitable for large projects

Cloud

Cloud data storage is essentially a response to the shortcomings of locally stored data. Cloud storage offers better flexibility, accessibility, and easier to scale upwards. Most of the time, this method also includes a way to easily sort and search for specific types of data that one may be looking for. This is something that cannot be easily done when storing data locally. And this will actually be the main reason why our group will be choosing to opt into cloud storage instead of local. Our group will need to be able to query our cloud database for our other plugins and applications we intend to build such as the heat map and automated friend's list.

6.4 Divining Rod

We decided not to use Divining Rod because we didn't like what it provided to the server and we didn't see a point in having the plugin. We thought that the ability to locate other players could be used maliciously. No base locations and no player would be safe if someone decided that they wanted to hunt others. Having to worry about these possibilities would lower the quality of the server, and potentially deter potential participants.

6.5 Automated Friends List as a Plugin

As more progress into the project was made, we realized that an automated friends list as a plugin didn't serve a real purpose. As a plugin, the only people that could see it's

information are the Minecraft players. We assumed that the player won't do much with that information. Instead, by making it part of the visualization tools on the website, the researchers that wanted to measure collaboration in Minecraft could make better use of this information. In the end, we opted not to develop the automated friends list because the automated friends list was proving to be a difficult challenge and that the heatmap had priority over everything else.

6.6 Hosting an AWS Server

Initially, we weren't sure on what exactly which server provider we wanted to use. We were debating between AWS and the UCF IT Data Center. The advantage of AWS is that our members had familiarity with the technology, while coming at a higher budget cost. UCF IT Data Center's advantage was the lower cost, but we weren't sure on how exactly it operated. We made our decision after our sponsor introduced us to someone that worked for UCF IT. They introduced us to the UCF AWS service that we could use. On top of that, they gave us a crash course on which server to choose, how to set up that server, and was available for email whenever we had any questions. After the feedback, we decided to use the EC2 service, and in particular, a m5 large server.

Chapter 7

Initial Ideation

- Measuring the average distance traveled from initial spawn point to settling area. (Can be plotted on a graph).
- Measuring collaboration between players and classifying the type of relationship they have with each other.
- Measuring the changes in player activity when incentives are introduced. (i.e. Reward 5 diamonds to the first player that collects 100 wood).
- Measure the change in player activity when people are given "roles" such as farming and building.
- Measuring where players choose to settle in.
- Measuring the location in the world where people like to spend their time such as being underground and the surface.
- Measuring player activity in teams when quests are introduced.
- Measuring player activity in teams when put into random parties.
- Measuring the number of resources mined around another player compared to the amount gathered while playing alone.
- Expand data collection to different types of Minecraft activities such as mini-games and creative mode.
- Introduction of a high score board for different player metrics such as distance traveled, diamonds collected and monsters killed.
- Using Natural Language Processing, measure the Sentiment of player to player conversation.

Chapter 8

Proposed Creations

8.1 Automated Friends List

Playing Minecraft is most enjoyable with your best of friends. You and your friends may have a plan to build a house together, going mining together, or just taking a break and exploring. The main point is when you play with your friends, no matter how big the server may be, a player will usually spend most of their time with their friends. We will use this idea to create our automatic friends list.

8.1.1 How will players make friends?

As stated previously, we are going to use the idea that players will usually be spending most of their time with the people they know. All a player has to do to have their friend pop up into their friends list is play with their friends.

8.1.2 How will the plugin know who to automatically add?

As of right now, the idea is to use the PlayerMoveEvent to trigger a timer that will check if there are players nearby within some radius. The timer will repeat every one second and will continue until there are no other players around. To do this, we will make use of the `getNearbyEntities()` method. This method takes three `doubles` named x, y, and z for the respective coordinates in the game. Using this will create an invisible box around a particular player with that radius, as shown in figure 8.1.

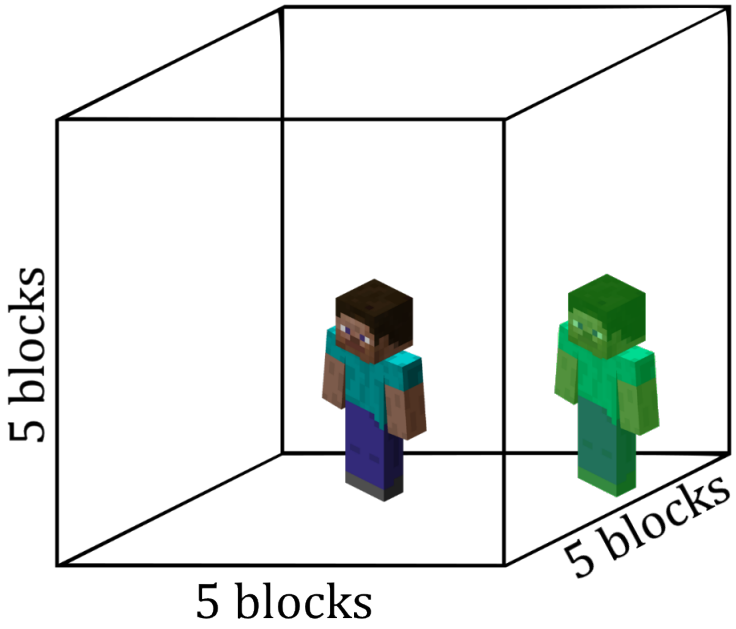

Figure 8.1: Using getNearbyEntities(5, 5, 5) and returning a list of one player

As seen above, we used the `getNearbyEntities(5, 5, 5)` method that is apart of the Player class in the PaperAPI. `getNearbyEntities()` returns a list of players that are within this box. The player that is in green that is shown above resembles the returned List of type Player with one entry inside of it.

Before we retrieve this list, we will put it inside of a scheduled task. For the scheduled task, we will make use of the PaperAPI's `BukkitScheduler` class. This class gives developers the ability to create delayed or repeating tasks. For this case, we will use the repeating task since we want to know the duration of time a player is around another. To create a repeating task, we call `scheduleSyncRepeatingTask()`.

When the player moves around, the server will call this repeating task and will track how long a particular player is around another one since our repeating task will occur every second, we will track how many seconds players are near each other. When we see that a player has left the zone, we will cancel the scheduler and add in the total time a player is near another one in the database.

8.1.3 How will players view the friends list?

The plugin will know who to automatically add by taking the top 5 players with the most time spent nearby. There will be two approaches to showcasing the friends list.

- Command based representation
- Visual GUI using chests.

Using the PaperAPI, we can actually create inventories for visual representation. Using the Inventory class, we can create a chest-like inventory and add items to it for display, as shown in figure 8.2.

Figure 8.2: Mock up of friends list

The above figure is a mock-up on how the friends list should look like, however instead of there being colors, there will be player heads of the players that are suggested to be a friend. Number 1 represents the other player that has spent the most time with the example player. The number two is the player who has spent the second-longest time together, and so on. If a player is on that list that is undesirable, a player can right-click the head and it will remove.

Note that all of these heads are suggestions for friends that a player may choose to add. Note that in order to be given the friend status for a pair of players, both of the players must left-click on each other's head. Instructions on how this plugin will function will be present in a command format and as well as a description when the player's mouse icon is hovering over a head.

8.1.4 How can players add each other on the Friends List?

The way that most friends list that we see today work is you must manually type in the username of the person you would like to add somewhere and then they must accept your friend request. However for the purposes of this plugin, we are taking a different approach. Instead of having players manually add each other, instead we will use the data from our data collection plugin to match friends together. Once again, if a person sees a player they do not wish to be friends with, they have the option to right-click the head of the player and the friend suggestion will be removed.

Overall we feel like this plugin will be an important part of our project because we will be able to tell who is spending time with who the most. This can be a very high indication of some sort of collaboration since friends usually perform activities together.

8.2 Rework of Divining Rod

We think that the Divining Rod plugin from HeapCraft is wonderful but still needs to be worked on. For example, their choice in using a stick as opposed to a compass seems questionable. This is because with the help of the PaperAPI, we can actually have the stick point towards any location on the map. This is possible by using the `setCompassTarget(Location)` that is included in the Player class. To properly use this method, we must first get the location of the target (whether it be a player or just an actual set of coordinates in a map) and then call the method when you are already referencing a player. So for example, we can use this whenever a player calls the `PlayerInteractEvent` which triggers when a player right or left clicks while they are holding some item.

Overall, Using a compass just seems like a better fit because in the current version of Divining Rod, when a player is trying to find someone in specific, they have to guess the direction they are supposed to head after at first since there is no indication of which direction they should be going specifically. The framework is illustrated in figure 8.3.

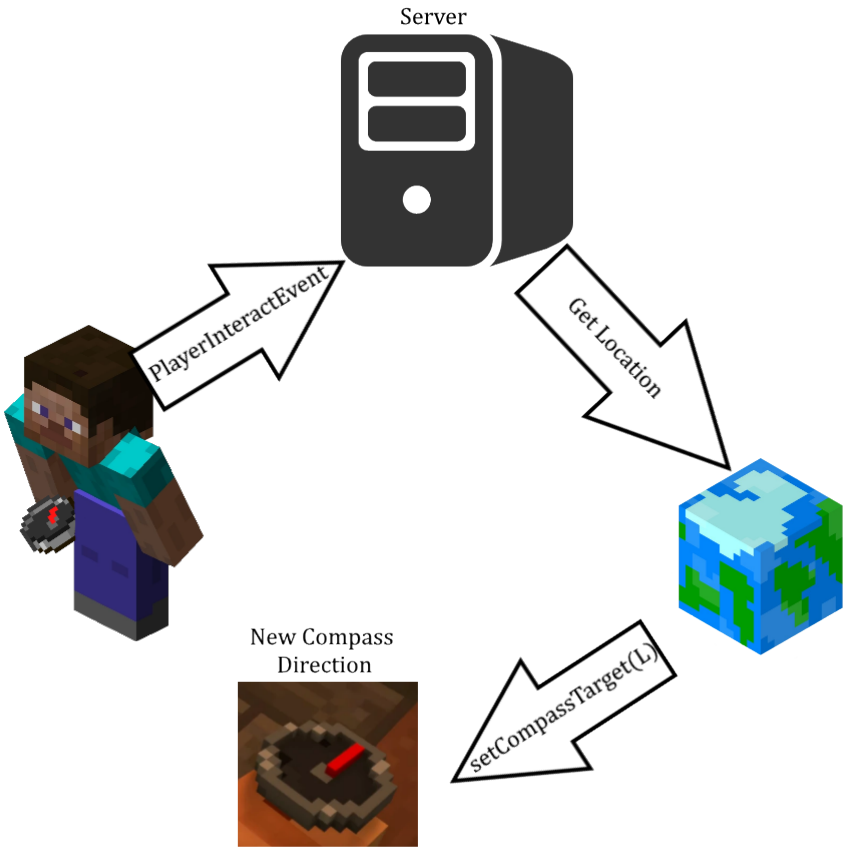

Figure 8.3: How `setCompassTarget(L)` works

8.2.1 Plugins Working in Conjunction

Another flaw would be how there could be a bit of ambiguity to players on how to actually use this plugin as there does not seem to be a help section on the website. Our goal for this rework is to address these flaws and try to better the plugin as a whole.

We feel like Divining Rod can be used in conjunction with our Automated Friends List. As it stands right now, Divining Rod has tags which will point you to specific players. As of now, the `friend` tag will point you to someone to be friends with. This makes it seem like it is picking someone at random, which doesn't really seem too useful. Instead, our goal is to instead have the compass point in the location of an existing friend in your friends list. If you have multiple friends, then you may indicate who it should point to with a command that may look like `/find friend <friendName>`.

8.2.2 Graphical User Interface

An important goal for this rework is to create a GUI for the plugin. We can create a chest-like inventory similar to how we would in the Automated Friends List plugin and instead fill it with functions for finding certain tags. Having some sort of visual with instructions will allow players to learn and use this plugin with ease. By entering the command `/find` will bring up the following menu (Figure 8.4).

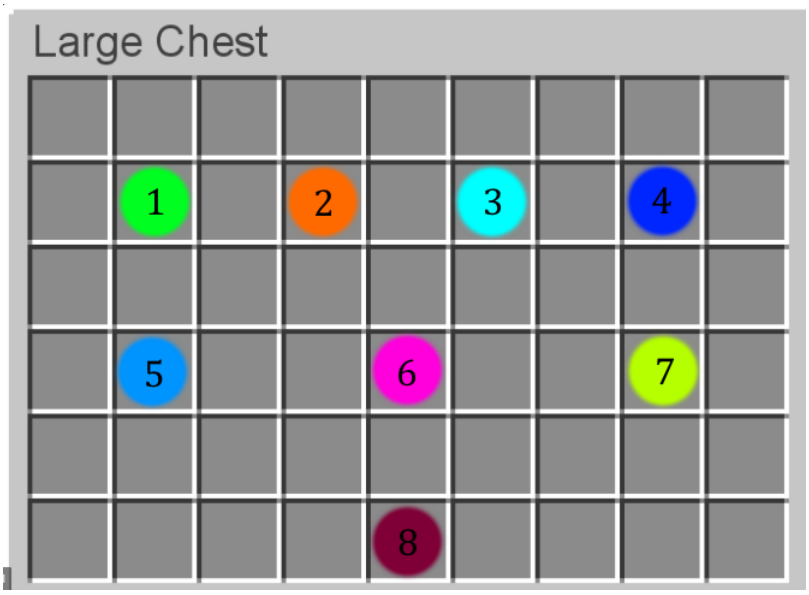

Figure 8.4: Mock up of GUI for Divining Rod rework.

The dot labeled 1 would be to find `spawn`, the one labeled 2 would be to find the `nearest` player, third for a specific player (`@<user>`), fourth for a `friend`, fifth for `miner`, sixth for `noob`, seventh for `builder`, and eighth for leaving the GUI. Each of these items will be given a name and will be some sort of head item instead of these colored dots. By giving the items a name, it will make it easier on the player to navigate using the GUI. Overall, the GUI will be a good feature for beginner players all up until they become accustomed to switching over to typing in the commands themselves.

8.3 Intelligent AI Agents

Using data from the events collected from players, we could create intelligent AI players to simulate the behavior of a real player. This would be a real test to see if we can teach a computer how to collaborate with others and to play the game efficiently.

8.3.1 Training Data

The data used to train the AI agent would be a chronologically ordered sequence of player events, gathered from our server. The agent would then learn what a player does, and when they do it, and potentially answering the *why* of player actions.

8.3.2 Data Organization

Each chronologically ordered sequence of events would have to be organized by player, this is to ensure that the sequence of events that is fed into the neural network is coherent and makes sense, rather than just throwing all of the events that happened at a certain time, across all players. This essentially aims to follow the common rule of machine learning: *good data in, good data out.*

8.3.3 Model Validation

To validate the AI models, we will need to construct some sort of scoring heuristic of how "human" the agent behaves. This heuristic will have to take in a sequence of events as input, and output whether the sequence is human-like or not. This can then be used in reinforcement learning to help train and validate the AI agents. The structure of this creation is displayed in figure 8.5.

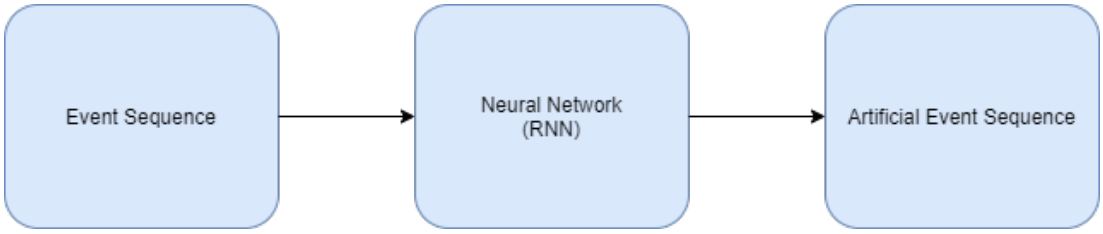

Figure 8.5: Visual representation of rough Neural Network I/O

8.4 Player Activity Forecasting

At any given moment, we will be able to see how many players are online. Every minute we can collect how many players are on the server, and we can then use that time series data to predict when the most players will be online.

8.4.1 Time Series Analysis

There are many ways to forecast what a time series will do, however it is much easier to do with cyclical, or seasonal data. Figure 8.6 below shows an example of what cyclical data can look like.

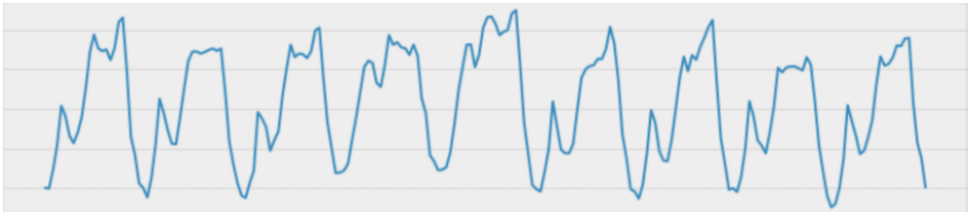

Figure 8.6: Seasonal Time Series Example

As shown, we can see that the data has peaks and valleys separated at relatively regular intervals, knowing this, we can apply certain techniques to predict when these peaks and valleys will occur.

8.4.2 Analysis Technique

One of the most common ways to forecast a time series, is by using a fast Fourier transform, formula shown below.

$$\sum_{n=0}^{N/2-1} a_n^{even} e^{-2\pi i(2n+1)k/N} + \sum_{n=0}^{N/2-1} a_n^{odd} e^{-2\pi ink/(N/2)}$$

Essentially, the Fast Fourier Transform (FFT) is an algorithm that computes the discrete Fourier transform of a sequence or series, a time series in this case. This converts the signal which is in time space, to a frequency domain. Frequencies are essentially a combination of sinusoidal functions to create a repeating time series. So the FFT is making use of Euler's identity to create a sinusoidal composition function. Then theoretically, we can input some time in the future into this function, and get the number of expected players online as an output.

8.5 Settlement Prediction

Another use of the data that we collect, could be predicting where newly spawned players will settle and build their base. This can be useful when we want to try to understand why players make certain decisions. We can use the heatmap to display this visually by filtering the data by player, and looking at their location data, and where they sleep.

By using an event such as the `PlayerBedEnter` event combined with location data to determine where a player has settled. We would then observe and record the path that the player took from their spawn point, to the area of settlement and see if there is any correlation among all of the players, is it a random distribution? Did they generally favor one direction over another?

8.5.1 Additional Details

We can also collect details about the surrounding area of the settlement, such as biome, distance from other players, concentration of high value ores, etc. If we find that there is a correlation with any or all of these factors, we can then use that data to train a predictive model that attempts to determine where a new player is likely to settle.

This helps us understand more what goes through the mind of a new player when looking for a base to build, whether their reasoning be conscious or subconscious, we can quantify that using a method similar to what was just described.

Chapter 9

Data

For the purposes of our project we will be using a combination of data previously collected for the HeapCraft project, data that was given to us by Professor Sukthankar, and data we will be collecting on our Minecraft server.

9.1 Events Collected by HeapCraft

Since we will be using the Epilog plugin by HeapCraft, we can build on top of the following events their plugin collected:

- `PlayerMoveEvent`
- `PlayerInteractEvent`
- `InventoryContent`
- `BlockDamageEvent`
- `BlockBreakEvent`
- `PlayerToggleSprintEvent`
- `PlayerItemHeldEvent`
- `PlayerPickupItemEvent`
- `PlayerItemInHandEvent`
- `EntityDamageByPlayerEvent`
- `InventoryCloseEvent`
- `PlayerRegisterChannelEvent`
- `EntityPortalEnterEvent`
- `BlockPlaceEvent`
- `InventoryOpenEvent`
- `PlayerTeleportEvent`
- `PlayerVelocityEvent`
- `FoodLevelChangeEvent`
- `PlayerInteractEntityEvent`
- `PlayerDamageByEntityEvent`
- `PlayerRegainHealthEvent`
- `PlayerToggleSneakEvent`
- `PlayerExpChangeEvent`
- `ProjectileLaunchEvent`
- `PlayerDamageEvent`
- `PlayerDropItemEvent`
- `PlayerCommandPreprocessEvent`
- `AsyncPlayerChatEvent`
- `ChestContent`
- `PlayerToggleFlightEvent`
- `FakeBlockDamageEvent`
- `FakeBlockBreakEvent`

- PlayerDamageByPlayerEvent
- PlayerItemConsumeEvent
- CraftItemEvent
- PlayerLevelChangeEvent
- PlayerChangedWorldEvent
- ArmorContent
- PlayerDeathEvent
- PlayerLoginEvent
- EnderChestContent
- EntityDamageByBlockEvent
- PlayerRespawnEvent
- EntityCombustByEntityEvent
- EntityShootBowEvent
- PlayerJoinEvent
- FurnaceExtractEvent
- PlayerBucketEmptyEvent
- PlayerBucketFillEvent
- PlayerQuitEvent
- EntityCombustByBlockEvent
- PlayerGameModeChangeEvent
- PlayerDamageByBlockEvent
- EntityPortalExitEvent
- PlayerItemBreakEvent
- PlayerPortalEvent
- EntityCombustEvent
- BlockIgniteEvent
- PlayerFishEvent
- PlayerKickEvent
- PlayerShearEntityEvent
- HangingPlaceEvent
- PlayerBedLeaveEvent
- PlayerBedEnterEvent
- SignChangeEvent
- PlayerChatTabCompleteEvent
- PlayerLeashEntityEvent
- StructureGrowEvent
- PlayerEggThrowEvent
- PlayerUnleashEntityEvent
- PlayerEditBookEvent

9.2 Data Given

The data we were given from Professor Sukthankar included a 2.6 GB JSON file of previously collected player events. Some of the events in JSON format can be seen below in Listing 1.


```
{
    "time":1408468966479,
    "event":"EntityShootBowEvent"
}
{
    "time":1408468966429,
    "cause":"ENTITY_ATTACK",
    "player":621,
    "event":"EntityDamageByPlayerEvent",
    "entity":4231,
    "damage":7,
    "entityClass":"CraftSkeleton",
    "health":0.5,
    "worldUUID":2
}
{
    "time":1408468966679,
    "player":505,
    "event":"FoodLevelChangeEvent",
    "var":15,
    "worldUUID":2
}
{
    "time":1408468966279,
    "cause":"PROJECTILE",
    "player":992,
    "event":"PlayerDamageByEntityEvent",
    "entity":4234,
    "damage":1,
    "entityClass":"CraftArrow",
    "health":0.46500000953674314,
    "worldUUID":2
}
{
    "time":1408468965978,
    "event":"EntityCombustEvent"
}
{
    "time":1408468967229,
    "event":"EntityDeathEvent"
}
{
    "time":1408468966279,
    "player":621,"event":"PlayerVelocityEvent",
    "worldUUID":2
}
```

```
{
    "time":1408468967880,
    "player":5996,
    "event":"PlayerPickupItemEvent",
    "var":0,
    "worldUUID":2,
    "material":"ARROW"
}
{
    "time":1408468967880,
    "event":"PlayerItemInHandEvent",
    "material":"IRON_SWORD"
}
{
    "time":1408468968230,
    "player":621,
    "event":"PlayerExpChangeEvent",
    "var":106,
    "worldUUID":2
}
{
    "time":1408468969282,
    "player":505,
    "event":"PlayerToggleSprintEvent",
    "var":1,
    "worldUUID":2
}
{
    "time":1408468986979,
    "player":2001,
    "event":"InventoryCloseEvent",
    "worldUUID":2
}
```


Listing 1: Example of data provided

We included the snippet above so you can better visualize the data being collected. As you can see, each player has a number associated with them instead of their name for anonymity purposes.

For the purposes of this project, we plan on only using the data given to us while setting up the visualization tools, or any kind of testing we might need to do prior to obtaining our own data. We also plan on combining the data set provided to us if we are not able to collect enough data from our Minecraft server.

9.3 Data Analysis

In order for us to properly estimate which data we need to keep from the data set given to us, as well as the events we will need to add, we conducted the data analysis described below.

9.3.1 Understanding the Data

Since we were given a 2.6 GB JSON file, and we already somewhat knew which aspects we wanted to keep, we focused on those to begin with. To avoid using Python's JSON libraries, we decided it would be best to load the data into a Pandas DataFrame given the abundant built-in libraries available for data manipulation. Once we had our DataFrame we were able to properly inspect everything that HeapCraft collected.

There were a total of 14,850,417 entries collected which were all a combination of the 70 events listed in Section 9.1. There were also 22 game attributes collected:

- `yaw`
- `time`
- `player`
- `event`
- `z`
- `pitch`
- `worldUUID`
- `y`
- `x`
- `var`
- `material`
- `blockX`
- `enum`
- `blockFace`
- `blockY`
- `blockZ`
- `cause`
- `entity`
- `damage`
- `health`
- `entityClass`
- `worldName`

HeapCraft's server collected data for 42 days, during which a total of 44 people played.

9.3.2 Data Used

The purpose of this data analysis was to estimate which portions we would need to keep to help build the visualization tools. We ended up only using 5 of the 22 attributes collected: time, player, x, y, and z. As for the events, only those which were performed by a player, or affected a player, were kept.

Once we had only the data we determined as necessary, we started parsing the DataFrame.

9.3.3 Data Manipulation

We focused on separating the information in a way that would be more legible and practical to manipulate. We started off by separating the play instances by each player,

rather than having each entry listed in chronological order, then chunking out each time a player played the game.

For their analysis, HeapCraft removed the `PlayerMoveEvents`, events that occurred less than 100 times, events that would normally come one after the other, but most importantly, `PlayerLoginEvents` and `PlayerQuitEvents`. However, for this analysis, we needed a way of verifying when players were playing at the same time. So, we first located each player's index, then used each of their respective `Login` and `Quit` events to get a list of Tuples with the index that matches when a player logged in as the first element, and the time they logged out as the second element. For instance, Player 1 logged in for the first time on index 81154, then logged out for the first time on index 111384, which tells us that the first instance they played occurred between indices (81154, 111384).

After this was done, we needed to re-combine this list with each player's times, and coordinates, and were now able to make a list of DataFrames which each contained an individual player's information. However, we were now facing a different issue where we needed to remember which index in the list corresponded to which player, and then which particular instance they played. Another example here would be that Player 1 had 36 individual instances during which they played, each one resembling the DataFrame in Figure 9.1 below.

	time	x	y	z
81154	1408583827586	143	97	-2
81158	1408583828163	143	97	-2
81159	1408583828415	143	97	-2
81160	1408583828861	143	97	-2
81161	1408583829012	143	97	-2
...	...	...	...	...
111380	1408586498510	143	10	26
111381	1408586498659	143	10	26
111382	1408586498711	143	10	26
111383	1408586498810	143	10	26
111384	1408586498960	143	10	26

Figure 9.1: First playtime instance of Player 1

We needed another way to locate each instance played by each player. For practicality, and to avoid remembering which player corresponds with which index, we made a Dictionary composed of a list of DataFrames and used their player number as their corresponding key. This solution gave us a quick and easy way to locate the information we needed to possibly start visualizing the data, see Figure 9.2 below.

```
{1: [                time    x   y   z
  81154   1408583827586  143  97  -2
  81158   1408583828163  143  97  -2
  81159   1408583828415  143  97  -2
  81160   1408583828861  143  97  -2
  81161   1408583829012  143  97  -2
  ...               ...  ...  ..  ..
  111380  1408586498510  143  10  26
  111381  1408586498659  143  10  26
  111382  1408586498711  143  10  26
  111383  1408586498810  143  10  26
  111384  1408586498960  143  10  26

  [24681 rows x 4 columns],
```

Figure 9.2: Dictionary with player numbers as keys for their respective instances

9.3.4 Data Visualization

Our original goals were to possibly conduct a Time Series Analysis, but due to time constrictions, we were unable to do so this semester. However, we were still happy with finally being able to see what a player's path would look like. We had recently heard about Plotly and saw how interactive the maps were, so we quickly learned how to use the libraries.

With this tool we made a 3D scatter plot, connecting each point with a line. We were finally able to visualize what a singular path would look like, see Figure 9.3 and Figure 9.4 below.

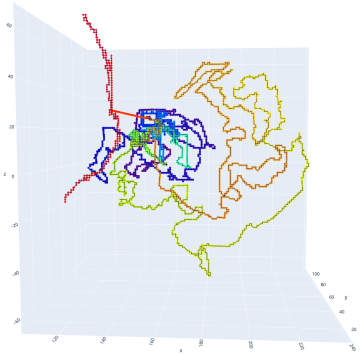

Figure 9.3: Player 1, inst. 6, front

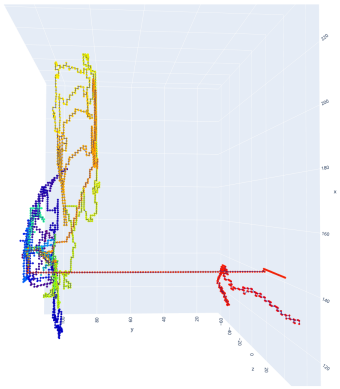

Figure 9.4: Player 1, inst. 6, side

The last feasible goal we had with our limited time and compute resources, was to locate times during which players played together. This proved to be more difficult than we had originally anticipated given the size of the data set. After many hours spent trying to use `itertools.product()`, then making lists of sets, and finding the intersection of those sets, we remembered that Pandas has an amazing function called `isin()`. This solved all the problems we were having with our computers running out of RAM while attempting to simply find player overlapping instances.

However, the results did not resemble what we had imagined and there were many inconsistencies. Player instances that we knew did not overlap, were showing up as True in the DataFrame. Upon realizing that this was due to players logging in, and then logging out without barely any activity, we needed to find a way to simply remove those

instances without removing the player entirely. By locating which players had instances with a standard deviation that were Not a Time (NaT) (see Figure 9.5), we were able to remove those particular ones from the DataFrame.

742	1	0 days 00:04:08.580000	NaT	0 days 00:04:08.580000	0 days 00:04:08.580000	0 days 00:04:08.580000	0 days 00:04:08.580000	0 days 00:04:08.580000

Figure 9.5: Standard Deviation NaT due to lack of activity

In some cases, this was not possible as they had only played during one instance, which left us with 33 viable players out of the original 44.

Once that issue was resolved we were able to see which players played at the same time as other players, as well as when their coordinates overlapped. We have included Figure 9.6, Figure 9.7, and Figure 9.8 to demonstrate this below. We also made a video demonstrating each plot with descriptions, which can be found **here**.

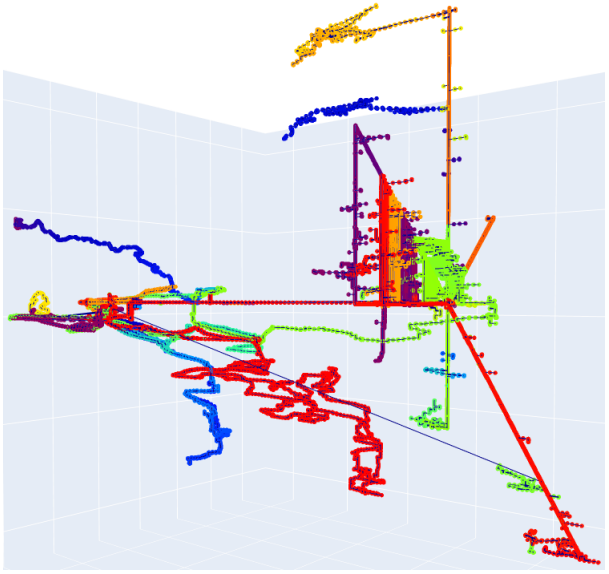

Figure 9.6: All instances Player 1 overlapped with Player 3

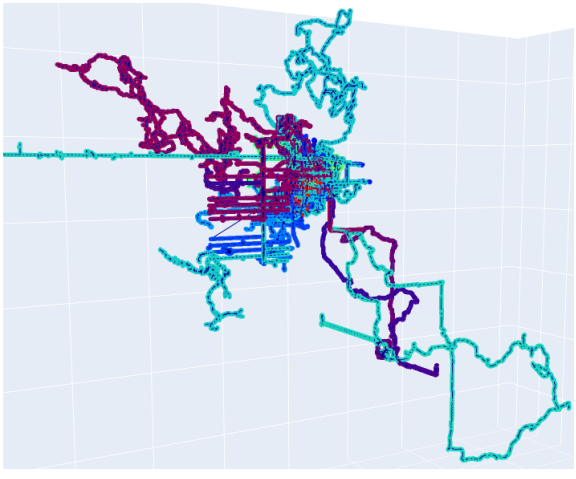

Figure 9.7: All instances Player 3 overlapped with Player 1

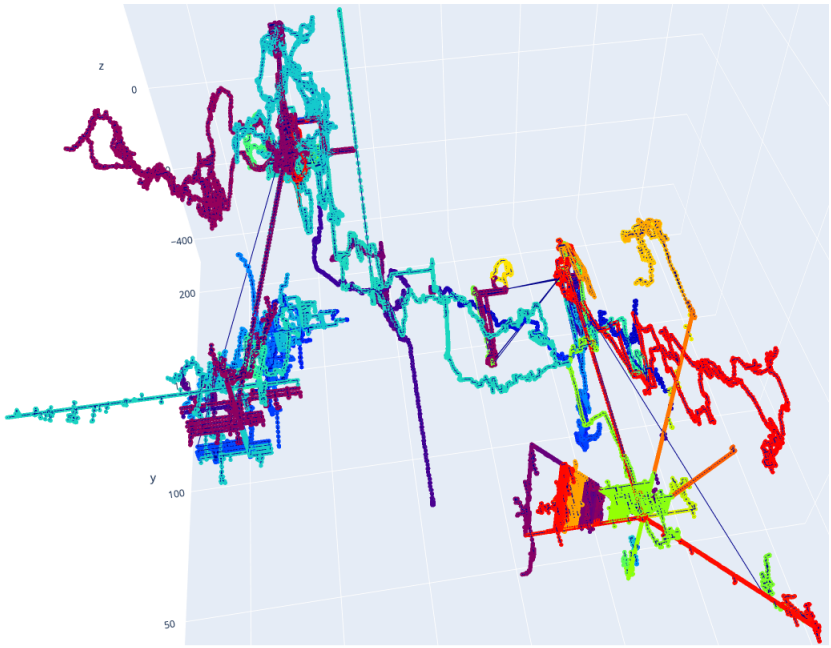

Figure 9.8: All instances with both Players 1 and 3, together

By including these plots we hope to show that simply measuring when players overlapped does not suffice to show the possibility of these players playing together. However, in our **video demonstration** we show that by using Plotly we can enable and disable which instances we would like to see. Doing this showed us that Player 1 and Player 3 were not playing together during their first few instances, however once they crossed paths during their last few instances, they seemed to remain in areas near each other.

9.3.5 Complications

Doing this data analysis gave us a few things to consider. We ran into a few obstacles that we hope to be able to prevent when we start collecting our own data.

First Complication

We originally planed to conduct a Time Series Analysis, but were unable to due to time constraints, as well as lack of computational resources.

Second Complication

We did not expect the data manipulation portion of the analysis to take as much time as it did. The data set we were given needed to be re-formatted so the information could be properly transferred to the DataFrame.

Third Complication

Multiple players would login and forget to logoff for extended periods of time. This meant that we had to also find the instances where it occurred and removed them from the data set. However this was not as easy as simply removing the instances where

players were inactive. The most active players were also the ones who forgot to logout (See figure 9.9). Spending time attempting to separate them was ineffective since we would not be using those players data in our final product.

We were also considering the possibility of doing Path Prediction using the data set given to us, but players were allowed to do certain events such as teleport, which makes path prediction very hard to do.

60	61	13 days 19:00:04.374377049	11 days 09:01:40.198974501	0 days 00:04:11.561000	0 days 18:41:02.233000	12 days 18:47:37.564000	24 days 22:21:32.292000	31 days 22:37:44.521000

Figure 9.9: Most active player

9.3.6 Solutions

Most of the complications we ran into will be easily fixed once we will be able to collect our own data. Being able to format the JSON file the way we need it to be right from the beginning will be more effective and less time consuming. As for the issues with computational resources, we plan on using a part of our budget for hosting a server, which should resolve these.

We also plan on analyzing player collaboration based on a player's friends list (see Section 8.1), which is not something we would be able to do if players can teleport.

9.4 Our Data Collection

We intend to use the data we will be collecting from our Minecraft server for training models and the final version of our visualization tools.

Here is an overview of the data we will be primarily collecting:

- Player location and movement: `PlayerMoveEvent`.
- Location of block breaking: `BlockBreakEvent`.
- Location of block placement: `BlockPlaceEvent`.
- Entity damaging any other entity: `EntityDamageByEntityEvent`.
- Player deaths: `PlayerDeathEvent`.
- Re-spawn location: `PlayerRespawnEvent`.
- Entering and leaving of a Nether or End portal: `PlayerChangedWorldEvent`.
- When a player uses an enchantment table: `EnchantItemEvent`.
- Player purposely dropping items: `PlayerDropItemEvent`.
- Entering and leaving a bed: `PlayerBedEnterEvent` and `PlayerBedLeaveEvent`.
- When a player leashes and unleashes other entities: `PlayerLeashEntityEvent` and `PlayerUnleashEntityEvent`.
- When a player hatches a chicken from an egg throw: `PlayerEggThrowEvent`.

- When a player shears a sheep: `PlayerShearEntityEvent`.
- When a player joins and leaves the server: `PlayerJoinEvent` and `PlayerQuitEvent`.
- When a bucket is filled or emptied with liquids: `PlayerFillBucketEvent` and `PlayerEmptyBucketEvent`.
- Whenever a player interacts with any block: `PlayerInteractEvent`.
- Player Chat: `AsyncChatEvent`.

9.4.1 Keeping Data

Data that is kept will only be used for research and visualization, none of the player information collected is personally identifiable, as that would violate research ethics on many levels. However, if a player would like their information removed at any time, for any reason, we will have the ability to do that.

If anything, we can simply remove the player's name from the data set, but keep all events triggered by them, as well as location data. This would still be valid data, but just simply not have any player name attached to it, giving the human subject absolute anonymity.

9.4.2 Removing Data

This project will not be using all events that the Epilog plugin collected.

We will have certain features disabled in Minecraft so we can get proper measurements from players. For this project we plan on measuring player collaboration, therefore we see no use in collecting the following:

- `InventoryContent`
- `BlockDamageEvent`
- `PlayerToggleSprintEvent`
- `PlayerItemHeldEvent`
- `PlayerPickupItemEvent`
- `PlayerItemInHandEvent`
- `EntityDamageByPlayerEvent`
- `InventoryCloseEvent`
- `PlayerRegisterChannelEvent`
- `EntityPortalEnterEvent`
- `InventoryOpenEvent`
- `PlayerTeleportEvent`
- `PlayerVelocityEvent`
- `FoodLevelChangeEvent`
- `PlayerInteractEntityEvent`
- `PlayerDamageByEntityEvent`
- `PlayerRegainHealthEvent`
- `PlayerToggleSneakEvent`
- `PlayerExpChangeEvent`
- `ProjectileLaunchEvent`
- `PlayerDamageEvent`
- `PlayerComandPreprocessEvent`
- `AsyncPlayerChatEvent`
- `ChestContent`
- `PlayerToggleFlightEvent`
- `FakeBlockDamageEvent`

- FakeBlockBreakEvent
- PlayerDamageByPlayerEvent
- PlayerItemConsumeEvent
- CraftItemEvent
- PlayerLevelChangeEvent
- ArmorContent
- PlayerLoginEvent
- EnderChestContent
- EntityDamageByBlockEvent
- EntityCombustByEntityEvent
- EntityShootBowEvent
- FurnaceExtractEvent
- EntityCombustByBlockEvent
- PlayerGameModeChangeEvent
- PlayerDamageByBlockEvent
- EntityPortalExitEvent
- PlayerItemBreakEvent
- PlayerPortalEvent
- EntityCombustEvent
- BlockIgniteEvent
- PlayerFishEvent
- PlayerKickEvent
- HangingPlaceEvent
- SignChangeEvent
- PlayerChatTabCompleteEvent
- StructureGrowEvent
- PlayerEditBookEvent

Since we intend to store usernames, we will be able to remove a player's data upon request.

9.4.3 Adding Data

Although we were provided with a large amount of data, we still need more in order to properly train models and accomplish our goals.

Since we are removing data that we deemed useless for this project, we need to replace it with events we believe will be useful. The data that will be remove is based on if it will contribute to measuring collaboration and if it provides redundant data. For example, all the entity damage can be grouped under EntityDamageByEntityEvent. The data that will be added will be from making modification to the Epilog plugin to incorporate items listed in Chapter 8.

Below we list the events we intend to add to the Epilog plugin:

- EntityDamageByEntityEvent
- ProjectileHitEvent
- EnchantItemEvent
- AsyncChatEvent

9.4.4 Optimizing the Data

The most triggered event in HeapCraft by far would be the PlayerMoveEvent by more than twice than the second most triggered event in HeapCraft. Since expanding storage

space was a budget concern, we tried to find ways to decrease the amount of data we are gathering. In particular, we found that PlayerMoveEvent triggered whenever the player moved their character and moved the camera in the slightest. For example, if the player's location is (0,0,0), if the player moves to (0.1,0,0.1), that would trigger the event. By changing the Data Collection Plugin to ignore when the player moves their camera and to check if the player actually moved to a completely new block, we estimate that it would cut the data by at least half.

9.4.5 Data Formatting and Requirements During Collection

Requirements on how the data is structured is important because it will affect what our other plugins will be able to pull from our data. Our initial decision was to use JSON for the type of data we will store, but we saw that a CSV file takes up less space in comparison. Because the hosting of the actual data outside of the database is an issue, we decided to export the data as a CSV even though the data is in JSON.


```
_id: ObjectId("6166f64890040f372e2e61a5")
location: Object
    x: -80
    y: 67
    z: 2
player: "7ae1ce09e166b11edb08be2c58017837"
worldName: "world"
worldTime: 196990
event: "PlayerJoinEvent"
```


Figure 9.10: Example of a single document in the database.

Activity Data

The Activity data feature was created to aid in the development of the Automated Friends List. This feature captures an the activity that most closely matches their what theyre doing in a UI that is opened when the player performs the /act command.

Figure 9.11: Activity Menu.

As shown, there are four activities one can choose from by clicking on one of the corresponding items. The netherite pickaxe represents "Mining", the map represents "Exploring", the brick block represents "Building", and the cake represents "Idle".

As an example, If I'm a player and am currently mining, I would click on the pickaxe and a new UI will open that shows up to ten nearby players to your current location represented by that player's head. Hovering your mouse over it shows the distance

between you and the player in meters. It should be noted that the names of the nearby players are not recorded but rather the hashed username.

Figure 9.12: After clicking the mining item in the Activity menu.

Clicking the emerald block will send the data off to the cache to then be sent to the database. An example of an activity record is as follows.


```
_id: ObjectId("61a44d2135d4f33a2fd343ba")
› location: Object
  player: "cd1d675519ad2c32d140974f1a6f21a1"
  worldName: "world"
  worldTime: 81124670
  activity: "Exploring"
⌄ nearbyPlayers: Object
  ⌄ 0: Object
      player: "4ec2a2c48b7707339c2e090ed592f58c"
      distance: 2.14
```


Figure 9.13: Example of activity document in the database.

For records that have 10 other player online, the nearbyPlayers object would have objects 0 through 9.

Exporting Data to CSV

A Python script in conjunction with the MongoDB tools for Ubuntu are used to export the data. We chose to organize the csv files by having a folder for each world and then putting a csv file for each event type we collected into it. Each csv file would only have data belonging to that specific world and event. So for example, if we wanted to find the player move data related to the overworld only, we would go to the world/PlayerMoveEvent.csv file.

The Python script that is executed on Ubuntu in a directory that already contains three folders titled world, world_nether, and world_the_end. It would then pull the data with a query via the mongoexport command.

```
1
2  mongoexport --db=Data --collection=worldData --type=csv
     --fields=event,location.x,location.y,location.z,player
    ,worldName,worldTime,block,damaged,cause,weapon,target,
    oldWorld,itemDrop,entity,hatched,contents,itemHeld,
    action --query='{"event": "PlayerMoveEvent", "worldName
    ": "world"}' --out="PlayerMoveEvent.csv" --uri=
    mongo_uri_here
3
```

It's worth mentioning that the fields that are in each file is uniform. Meaning each file has the same fields, even if they are empty. All of the fields can be located in the mongoexport command under the –fields flag.

Sorting the CSV

An issue that we ran into with our database was gathering all the data when it was under a single CSV or JSON file. We solved this by separating the data into the three dimensions, the Overworld, Nether, and End. Afterwards, all the events were separated into their own folders, such as Movement, BlockPlace, and BlockBreak.

General Data

For our general data, meaning data that we are collecting from the events from the API, our requirements are as follows:

- World time (tick)
- World time to seconds
- Location (x, y, z)
- Event
- World
- Player

As specified in Section C.1, Minecraft world time differs from our real world time. In Minecraft, 20 ticks is equivalent to 1 second in real time. Often, the phrase "ticks per second" or TPS can directly represent whether or not a server may be having issues with lag. Optimally, a server would run at 20 TPS, however that isn't always the case.

For our first record, we will record the current Minecraft world time in ticks. To pull this value, we will use either of the following methods:

In the `World` class, there are two methods we could use for retrieving the time. `getTime()` or `getFullTime()`. At first, we weren't sure which option was better to use, so a quick plugin was created to test these methods. For this test, we wanted to see which method gave us the ongoing world time and which one would give us the clock time that resets to 0 with each day. The code for this plugin is as follows:

```
1
2  @Override
```

```
3  public boolean onCommand(CommandSender sender, Command
    cmd, String label, String [] args)
4  {
5    if  (label.equalsIgnoreCase("ct"))
6    {
7      sender.sendMessage("getTime() : " + Bukkit.getWorld
    ("world").getTime());
8      sender.sendMessage("getFullTime() : " + Bukkit.
    getWorld("world").getFullTime());
9      sender.sendMessage("--------------------");
10     sender.sendMessage("getTime() / 20: " + (Bukkit.
    getWorld("world").getTime() / 20));
11     sender.sendMessage("getFullTime() / 20: " + (Bukkit
    .getWorld("world").getFullTime() / 20));
12     return true;
13   }
14
15   return false;
16 }
17
```

Listing 9.1: Code snippet with BukkitAPI of above scenario

When a either the console or the player performs the `ct` command in game, the server will send the player will send the time to the player in ticks in the top half with the two methods. On the bottom half, the server divides this value by 20 to convert the ticks into seconds. The results of this are as follows.

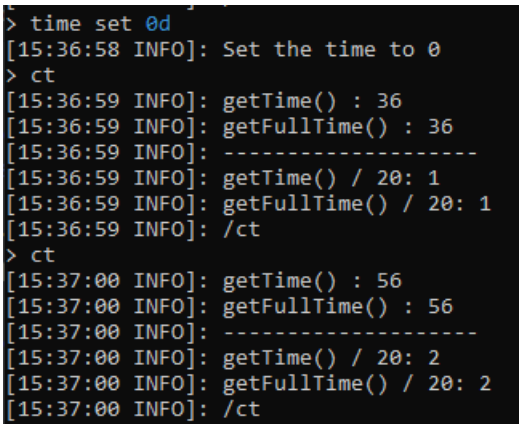

Figure 9.14: Setting the world time to zero days.

First we set the time to 0 days to compare the output of both of the functions. It was expected for both of these functions to output the same number as we set the world time to 0 days.

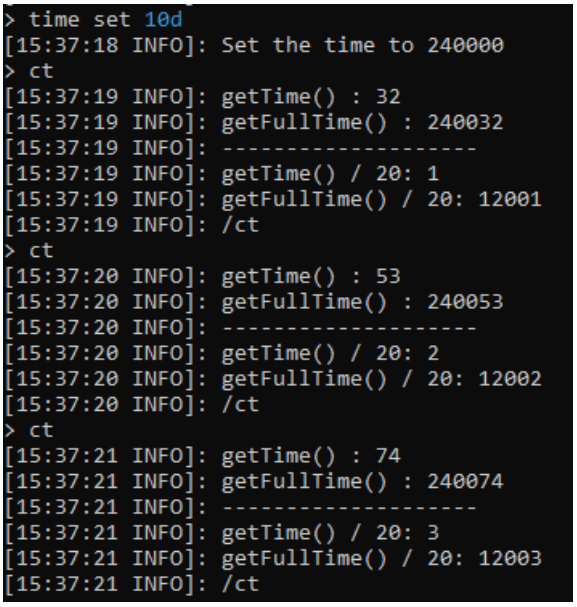

Figure 9.15: Setting the world time to 10 days.

Afterwards, we set the world time to exactly 10 days. Our functions were now outputting different values, which was a good thing. As it turns out the `getTime()` method works similarly in terms of a clock where there is a day period and once it hits that period, the time will reset to zero. On the contrary, `getFullTime()` returned the running time of the world, which was exactly what we were looking for.

```

// Getting world time directly from a player.
player.getWorld().getFullTime();

// Getting world time from a string, worldName.
Bukkit.getWorld(worldName).getFullTime();
```

Listing 9.2: Check time command abbreviated as ct

These methods both perform the same function, but they have different methods of doing so. We would use the first method to retrieve the world time directly from the Player that may have performed an action. This is because in the Player class, we have access to the current world that a player is existing in.

The second method for retrieving the time of a world would be useful for retrieving the time when a player may not have requested it. If for some reason, another entity or the console wanted to retrieve the time as we saw in the above scenario.

Now we know that all we would have to do to retrieve the running world time for our first slot of our data record is by using the `getFullTime()` method. The next slot would be storing the current tick to seconds. To do this, all that has to be done is to take the value we get from `getFullTime()` and divide it by 20.

The `Location` will come from the `getLocation()` mehthod from the `Player` class that is inherited from the `Entity` class. This method will return a `Location` object in which we can take a player's X, Y, and Z coordinates. Most likely, the `Location`

record will be an object that contains a section for the X, Y, and Z coordinates to be housed.

```
1
2 // Getting the Location from a player that triggered an
     event.
3 Player player = event.getPlayer();
4
5 Location location = player.getLocation();
6
7 int x = location.getBlockX();
8 int y = location.getBlockY();
9 int z = location.getBlockZ();
```

Listing 9.3: Example of the getLocation() method

For the `Event` record, we will utilize the `getEventName()` method from the Event class. All events have this method, so when a player triggers this event, we can get the name for later use by using that method.

```
1
2 // Getting the name of PlayerInteractEvent
3 @Override
4 public void onInteract(PlayerInteractEvent event)
5 {
6     String eventName = event.getEventName();
7
8     return;
9 }
```

Listing 9.4: Example of retrieving the name of a triggered event.

The World record will be retrieved using the `getWorld()` from the Player class. This in itself returns a World object, which we can use to get the name of the world.

```
1
2 // Getting the name of the world a player is currently
     residing.
3 Player p = event.getPlayer();
4
5 World world = p.getWorld();
6 String worldName = world.getName();
```

Listing 9.5: Example of retrieving the world name

And finally, for the `Player` record, we will use the `getUniqueId()` method in the Player class. This method returns a UUID object, but we can use the `toString()` method to have it return a string.

```
1
2 // Getting the name of the world a player is currently
     residing.
3 Player p = event.getPlayer();
4
5 String uuid = p.getUniqueId();
```

Listing 9.6: Example of retrieving a player's UUID

9.4.6 Data Processing and Manipulation

Player Mapping

Once the CSVs are sorted as mentioned in Section 9.4.5, we load the CSV for the Overworld `PlayerMoveEvent` into a Dask DataFrame, see Listing 9.7. Using this DataFrame, we gather every hash, and map them to a player index $0 - n$ where n is the number of unique players. We then apply this player mapping to every CSV, keeping the player index as the player identifier, and removing the hashes previously used. This process is also repeated for the chat log events, as well as the activity events.

```
1 # Using Dask's DataFrames to glob all the event CSVs in `
     datapath`
2 ddf = dd.read_csv(f"{DATAPATH}/overworld/*.csv")["player"
     ].unique().compute()
3
4 # Swapping the index/data values so that when `dict(...)`
      is run we get entries like:
5 #   "<hash>": <index>,
6 ddf = pd.DataFrame({"player": ddf.values, "pindex": ddf.
     index.values})
7
8 playerindex = {player: pindex for (player, pindex) in zip
     (ddf["player"], ddf["pindex"])}
9
10 def preprocess_df(file):
11     df = pd.read_csv(file)
12
13     # Use the Player Index created by Dask (above) to do
     the mappings
14     df["pindex"] = df["player"].replace(playerindex)
15
16     return df
```

Listing 9.7: Code for player mapping.

Optimizations

While we already have measures in place for reducing data redundancy, we were still left with very large files for `PlayerMoveEvent` and `EntityDamageByEntityEvent`. Due to this we selected only every 60^{th} point per player for these events to render the 2D maps, and every 30^{th} point per player for the 3D maps, see Listing 9.8.

```
# Getting every 30th point per player
def get_every_30(df):
    return df.loc[df.index % 30 == 0]

# Getting every 60th point per player
def get_every_60(df):
    return df.loc[df.index % 60 == 0]
```

Listing 9.8: Reducing data.

Chapter 10

Web Interface

We will be creating a web interface to host all sorts of visualizations, one of which we will be creating ourselves. This will serve as a hub for all visualizations of data collected on the HeapCraft server. There is plenty of room for expansion, making it ideal to hand off to other teams in the future, or possibly even collaborate with the other team currently working on HeapCraft.

10.1 Development Stack

Two of the most popular stack development are MEAN and MERN. While both use MongoDB, Express.js, and Node.js, MEAN uses Angular versus the React in MERN.

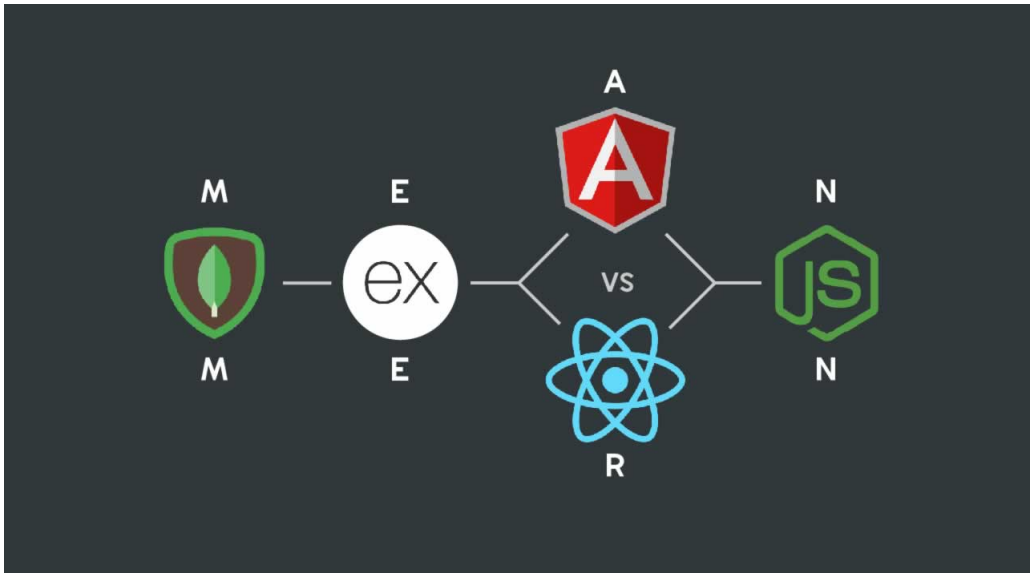

Figure 10.1: Difference between the Development Stacks [7]

10.1.1 Relational Database vs Non-Relational Database

Relational Database

Relational Databases stores data in tables and rows and referred to them as records. Examples of such are Microsoft SQL Server and MySQL. The main way relational database operates is through the usage of "keys". Keys are a unique identifier assigned to a row of data within a table. This unique identifier is referred to as "primary key" when the key is used to reference a different table or record. When this key is added to a record in another table, it is referred to as the "foreign key" in the associated table. This makes a connection between the primary and foreign key, allowing for a relationship between the tables [8].

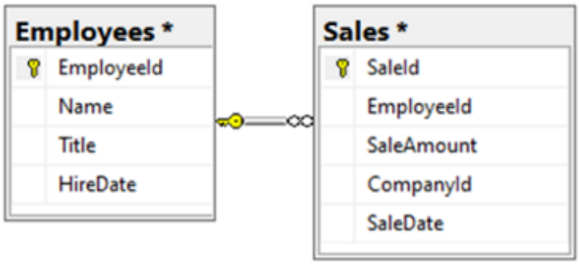

Figure 10.2: Example of Relational Database [8]

An example of this can be seen in Figure 14.1, with two tables named Employee and Sales. The Employee table will contain information such as the employee's ID, name, title. The Sales table will contain information such as the sale's ID, employee's ID, and the amount of money made from the sale. Using keys, the employee's ID in the Employee table will be the primary key. The employee's ID in the Sales table will be referred to as the foreign key in this case because an employee can make multiple sales. This is particularly useful because every sale that the employee makes will be connected to his employee ID, making querying easier [8].

The biggest benefit to Relational Databases is its Referential integrity. Referential integrity refers to the accuracy and consistency of data that is achieved through the usages of primary and foreign keys. This is possible because of the three rule constraints put into place to help preserve data integrity [8].

1. No Orphan rule
2. Cascade Delete
3. Cascade Update

The first rule is the "no orphans" rule, which states that every foreign key must have a corresponding primary key. The second rule is "Cascade Delete", which means that for a record in the primary table to be deleted, all related records referencing the primary key, aka records containing the foreign keys, must be deleted. The third and last rule is "Cascade Update", which means that if the primary key for a record change, all corresponding records using the primary key as a foreign key must also be modified [8].

Non-Relational Database

Non-Relational Databases stores data, but in a completely different way to Relational Databases. Non-Relational does not use tables, rows, primary keys, or foreign keys. Instead, they use a storage model for specific requirements on the types of data that needs to be stored. There are four main non-relational types, document data store, column-oriented database, key-value store, and graph database [8].

Document Data Store manages a set of named string fields and object data referred to as documents, which are typically stored as JSON documents, which can be encoded in many ways such as XML and YAML. The advantage of document store is that it doesn't require all documents to maintain identical data structures, giving it a large amount of flexibility.

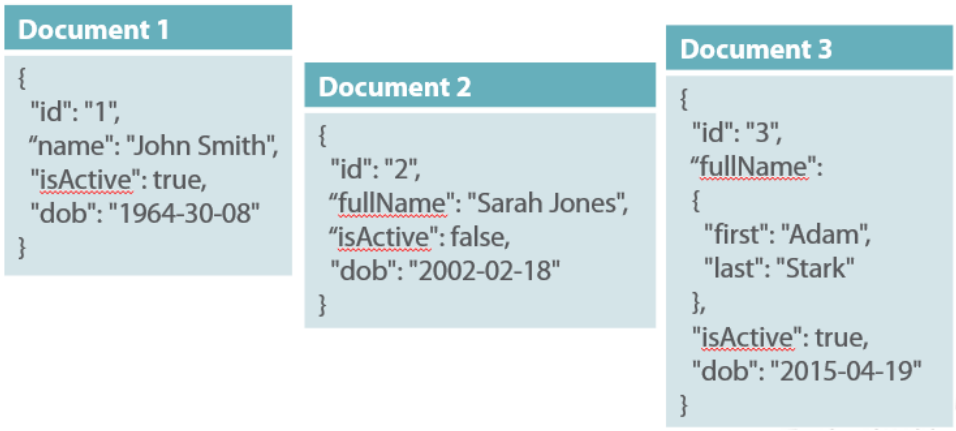

Figure 10.3: Example of Document Data Store [9]

Column-Oriented Database organizes data similar to a relational database. However, it differentiates itself by organizing data into columns. This allows for an advantage in structuring sparse data.

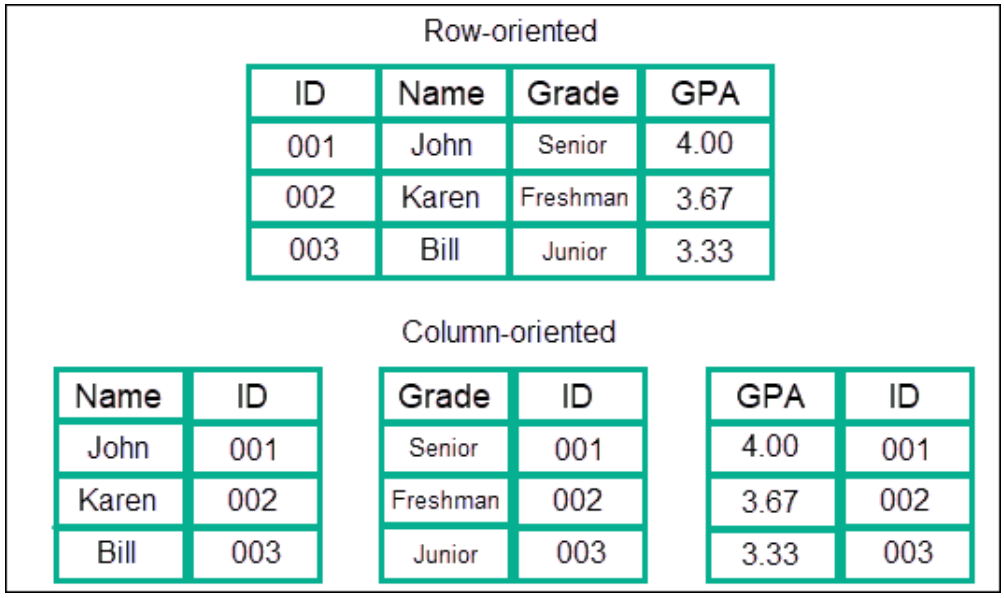

Figure 10.4: Example of Column Database [10]

Key-Value Store is the simplest of the four. It is simply a collection of key-value pairs.

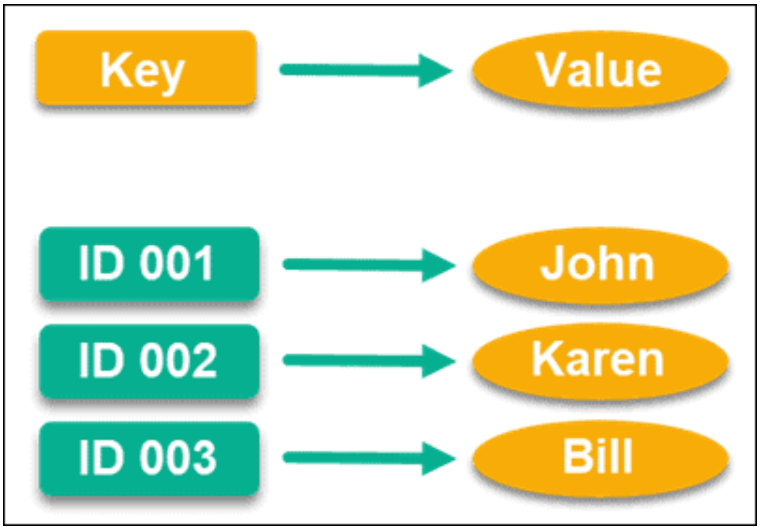

Figure 10.5: Example of Key-Value Store [10]

Graph database is the most complex of the four. This is design to efficiently store relations between entities. This is aimed for when data is greatly interconnected with one another such as purchasing and manufacturing systems or referencing catalogs.

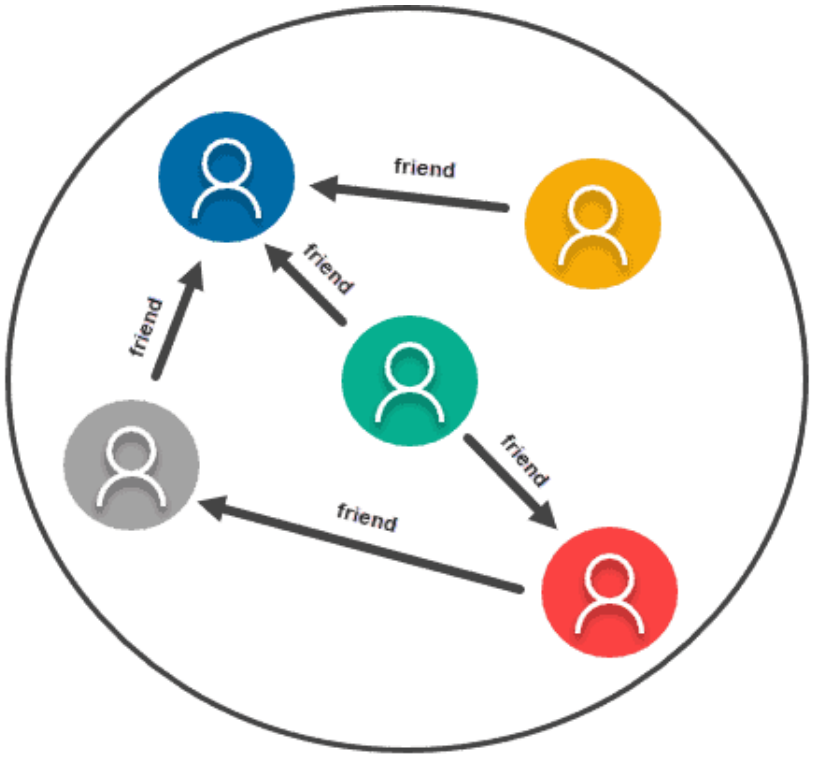

Figure 10.6: Example of Graph Database [10]

Conclusion

Looking at the difference between Relational and Non-Relational Database, makes Non-Relational Databases an appealing option, particularly document data stores. Relational Database is very good when we know how the information is going to be structured. However, because we plan to collect a large amount of information at different times, we are going to have a lot of different data structures. Non-Relational and document data stores are perfect for this because of its flexibility.

10.1.2 MongoDB

"MongoDB is a document-oriented NoSQL database used for high volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs which are the basic unit of data in MongoDB" [11].

MongoDB will be used as the database framework. Since MongoDB is a document-oriented database, this makes MongoDB very flexible in what data it can store. Other features MongoDB support are queries, indexing, and replication. Queries allow for searching based on field and regular expression searches. Indexing can be used to help improve the searches within MongoDB by allowing any MongoDB document to be indexed. Replication ensures that there are replica sets, which means that there will always be a backup.

10.1.3 Mongoose vs. MongoDB Native

Mongoose

Mongoose is a ODM library for MongoDB and Node.js. In Mongoose, there are Mongoose Schema and Mongoose Model. Mongoose Scheme defines things such as the structure of the document, default values, and validator. Mongoose Model is a wrapper on the Mongoose scheme and provides an interface to the database for creating, querying, updating, and deleting records. [12]

When using Mongoose, a user can define the schema for the documents in a particular collection. This provides convenience in the making and management of data in MongoDB. This is due to the fact that Mongoose requires less repetitive code to write and therefore the risk of error is lower. On the other hand, learning mongoose takes some time and has limitation on handling complex schemas [13].

MongoDB Native

MongoDB Native is the native driver when interacting with a MongoDB instance. The benefits of using MongoDB Native is that it is simpler to pick up and performs operations faster, making it better at handling complex queries. The downsides to native is that there is more code to write for validating data and that the risk of error is higher [13].

Conclusion

We plan to use MongoDB Native because of MongoDB's benefits of being simpler to pick up and that it has a better time with dealing with unpredictable and complex collection schema. The main benefit will be the ability to deal with complex schema because the data we will be collecting will largely vary from one another.

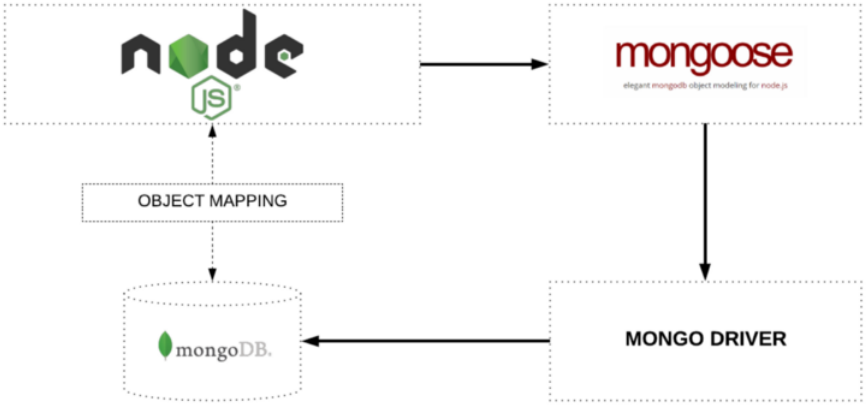

Figure 10.7: Object Mapping between Node and MongoDB managed via Mongoose [12]

10.2 Heatmap

The first visualization that will be developed to add to the Web Interface is an interactive heat map, where we will be plotting all player events that are relevant to analyzing player behavior and collaboration. All of the events collected can be found in section 9.1. A portion of these events that we find relevant will be color-coded and plotted on a top-down view of the Minecraft world. This will tell us where each event is happening and we can study which events tend to happen on the map (More movement events along player-built paths, more deaths in a PVP arena or near monster spawners, etc).

10.2.1 Plotly

Plotly is a data visualization framework that has the unique ability to make a 3D heat map, this is what we will primarily use it for. It is easy to use and does a lot of the heavy lifting for us, saving us time to focus on other aspects of the project. The 3D heat map will be explained further in the next section.

10.2.2 3D Heat Map

One major thing that the standard heat map is lacking, is a third dimension. It would be visualizing data from a 3D game on a 2D map. A 3D heat map would give us the freedom to show data for all coordinates fully: x, y, and z coordinates. This is opposed to the standard heat map, which would only be plotting data on two axis, x and z. Adding the third axis will allow us to better visualize what players are doing both underground and in the air. This includes creations like tall bases or mining activity in deep caves. Figure 10.8 below shows an example of a 3D heat map generated using Plotly.

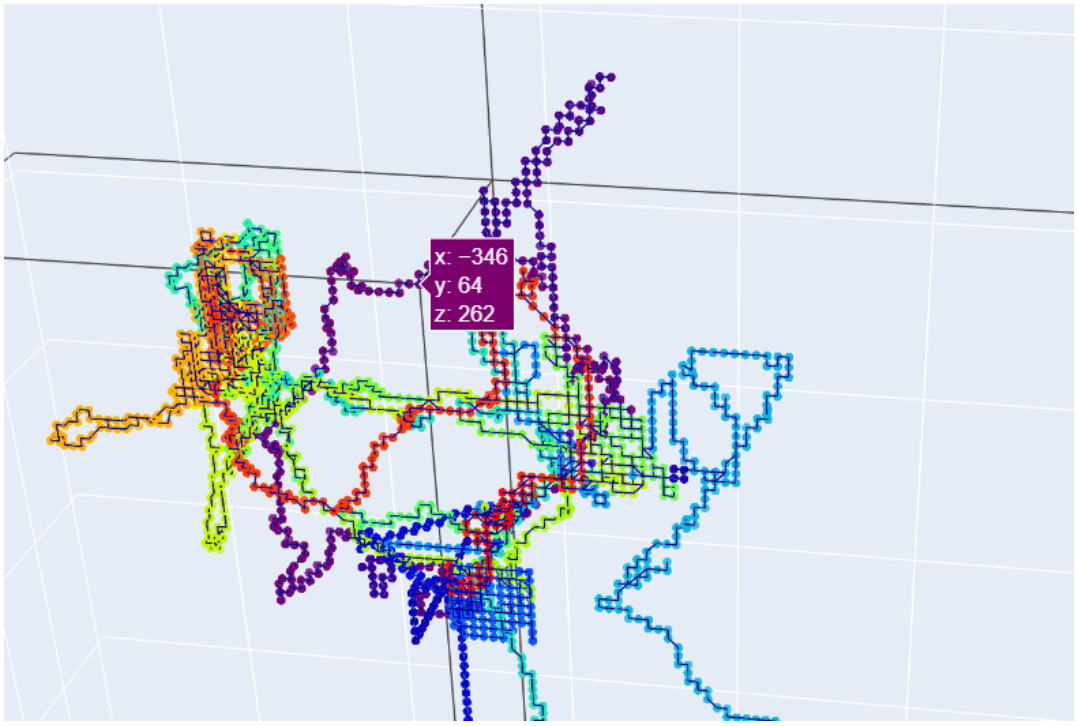

Figure 10.8: 3D Heatmap Example

This entire extra dimension of visualization can help us further analyze player behavior to serve the main objective of this project. Further analysis of the data also helps in all use cases of this project, such as creating more human-like autonomous agents. If we are able to see what they are doing in all three dimensions, we can better understand what they are doing right, and what they are doing wrong.

10.2.3 Events included

- `PlayerMoveEvent`
- `BlockBreakEvent`
- `BlockPlaceEvent`
- `PlayerDamageByEntityEvent`
- `ProjectileHitEvent`
- `PlayerDeathEvent`
- `PlayerRespawnEvent`
- `PlayerChangedWorldEvent`
- `EnchantItemEvent`
- `PlayerDropItemEvent`
- `PlayerBedEnterEvent`
- `PlayerBedLeaveEvent`
- `PlayerLeashEntityEvent`
- `PlayerUnleashEntityEvent`
- `PlayerEggThrowEvent`
- `PlayerJoinEvent`
- `PlayerQuitEvent`
- `PlayerBucketEmptyEvent`
- `PlayerBucketFillEvent`
- `PlayerInteractEvent`
- `AsyncChatEvent`

10.2.4 Using data from HeapCraft database

The event data will be collected from the HeapCraft database in JSON (JavaScript Object Notation) format. This makes it trivial to process it through our web interface and use the data for whatever visualization we choose, in this case, the heat map.

10.2.5 Each Square

One pixel on the heat map will translate to one block in the Minecraft world. The color of the squares will be adjusted depending on which events happened there. The legend for the color of each event has yet to be decided. When clicked, the selected square will display which events happened as well as who triggered the events.

10.2.6 Searchable Data

The visualizations on the heat map will be able to be searched and filtered by certain parameters. For instance, there will be an option to only show the events of a single player, or just the movement events of a certain player. This will help us study what specific players do, and determine which players are collaborating with each other.

Being able to search and filter through the data can also help us pinpoint certain anomalies, such as regions where players are more likely to take damage, or where the majority of players die, etc.

10.2.7 Chapter Conclusion

For the development of the web interface, we initially planned to use a classic MERN (MongoDB, Express.js, React.js, Node.js) stack. After getting introduced to Express, react, and node, our team decided that is was best if we avoid using these technologies. Instead, we opted to use Vanilla JavaScript as the replacement for all three. This was a decision made after careful consideration of what these technologies provided. The comfortability that our team would lose using these technologies wasn't worth the trade-off in our opinion. In the end, MongoDB is the database framework that will be used and vanilla JavaScript will be used for the back end, front end, and how we will encapsulate all the other packages.

Chapter 11

UX/UI Importance and Iterations

11.1 Why is UX/UI Important in this project?

Even though this project is research based, the layout of the user interface and the quality of the user experience is still an essential part that should not be overlooked.

11.1.1 UI Importance

The main content of a research project is data, visualizations of that data, and interpretations of those visualizations. A clean user interface (UI) has the primary purpose of organizing the data on the screen to present it to the users, which in this case, are researchers.

In our case, the UI must clearly display the heatmap and all of the information that it holds. As mentioned before, each square on the heatmap will represent one square block in the Minecraft world from a top-down view. Each square will then be able to be clicked on to show which events happened at that location. Each event will also have their own color associated with it, with the contrast of that color increasing with the frequency of that event at that location.

Having the server event information organized in this manner will make it easy for us to pinpoint where and when certain events happen, allowing us to easily and visually analyze the data to draw conclusions. But without a clean and understandable UI, the difficulty of drawing conclusions from the data would significantly higher.

11.1.2 UX Importance

Under the scope of a research project, the user experience (UX) is another essential aspect of how the data is visualized and navigated. If the user experience is pleasant, and friction less, productivity will go up, and in our research case, productivity is imperative to reach quick, yet still accurate research conclusions.

11.2 Design Iterations

11.2.1 First Iteration

The first iteration of the web-based heat map is shown below in figure 11.1. The main aspects of the design that we will be tweaking is the placement of the main heat map canvas, and the filters menu. We can think of these iterations as prototypes, we are aiming to see which one best fits our needs.

Some key features to points out, the heat map canvas is centered horizontally and vertically on the page, and the filters menu is one long ribbon along the bottom of the page, centered horizontally. This will change with the next iteration.

It is worth noting that the heat map shown throughout these design iterations is purely conceptual and is not generated using any real data.

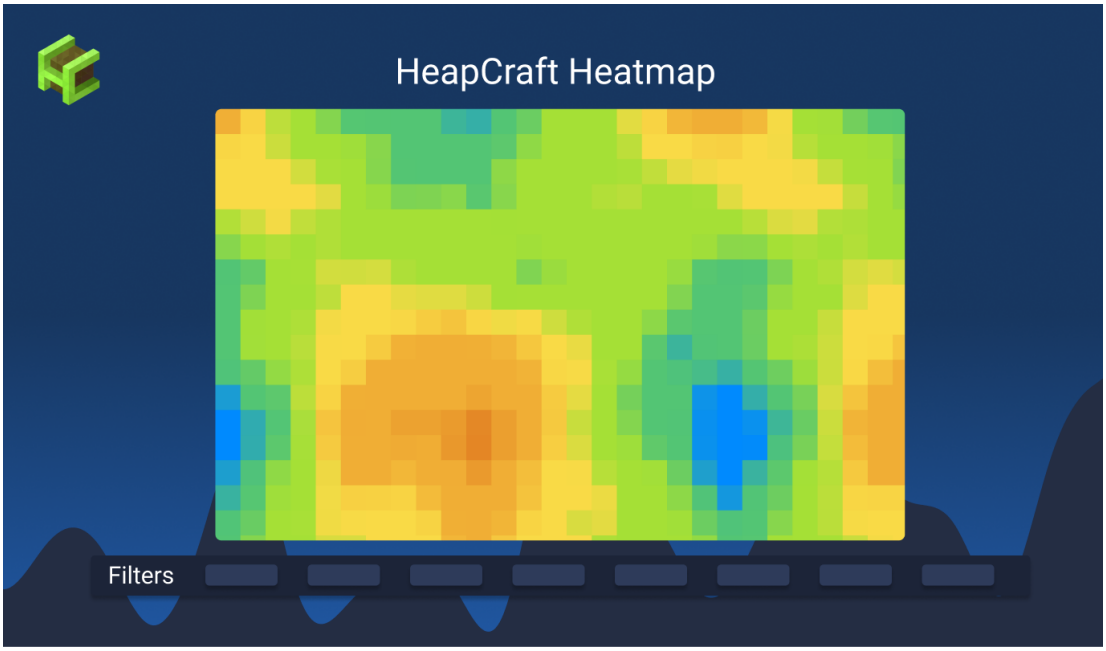

Figure 11.1: First UI Iteration

11.2.2 Second Iteration

One of the most jarring differences between this iteration and the last, is the filters bar. Instead of a ribbon along the bottom of the page, it has instead been moved to the left side, and changed to a vertical format, instead of horizontal. This takes a likeness to the filters bar on shopping websites like amazon.

The next large change is the position and size of the heat map canvas. The canvas has been moved off-center, and the size has been increased slightly. This new layout makes a more efficient use of the available screen space and increases the size of the main visual component, making the visualizations easier to see.

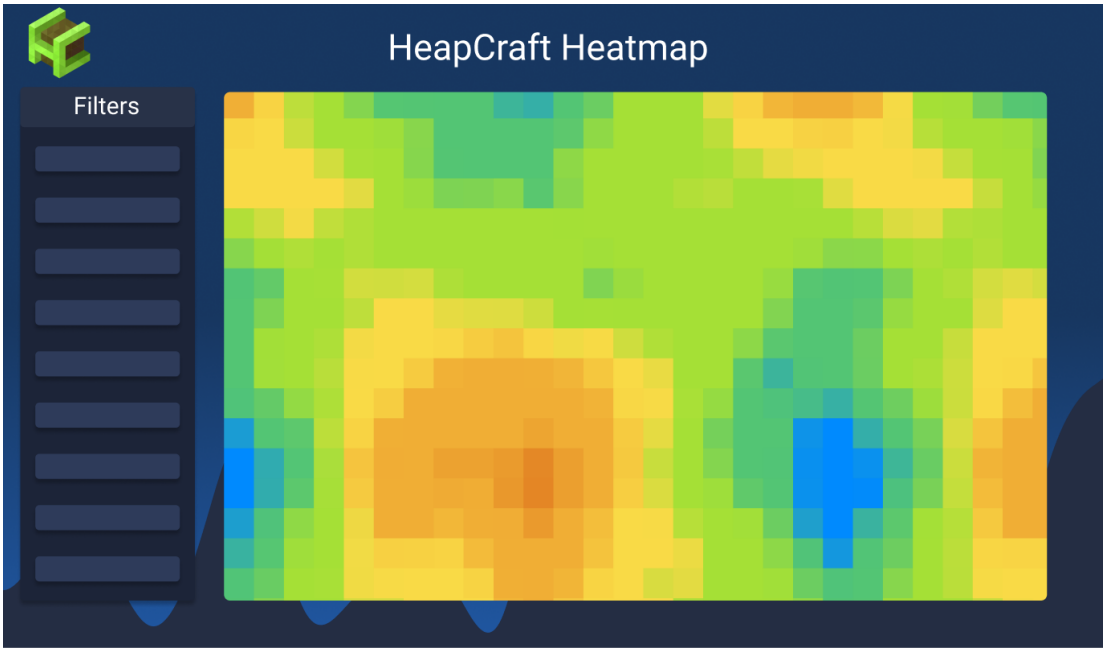

Figure 11.2: Second UI Iteration

11.2.3 Third Iteration

Each iteration will likely show smaller and smaller changes, while slowly settling on a final look. This iteration includes some minor changes, which include an extended filters bar, which now takes up more of the left side of the screen. There was also some re-organizing of the logo and title. The most visual change is quite simple, just trying out the look on the 3D heatmap, but ideally you will be able to switch between 2D and 3D views, this is likely to show in future iterations. The mockup of this UI iteration is shown below in Figure 11.3.

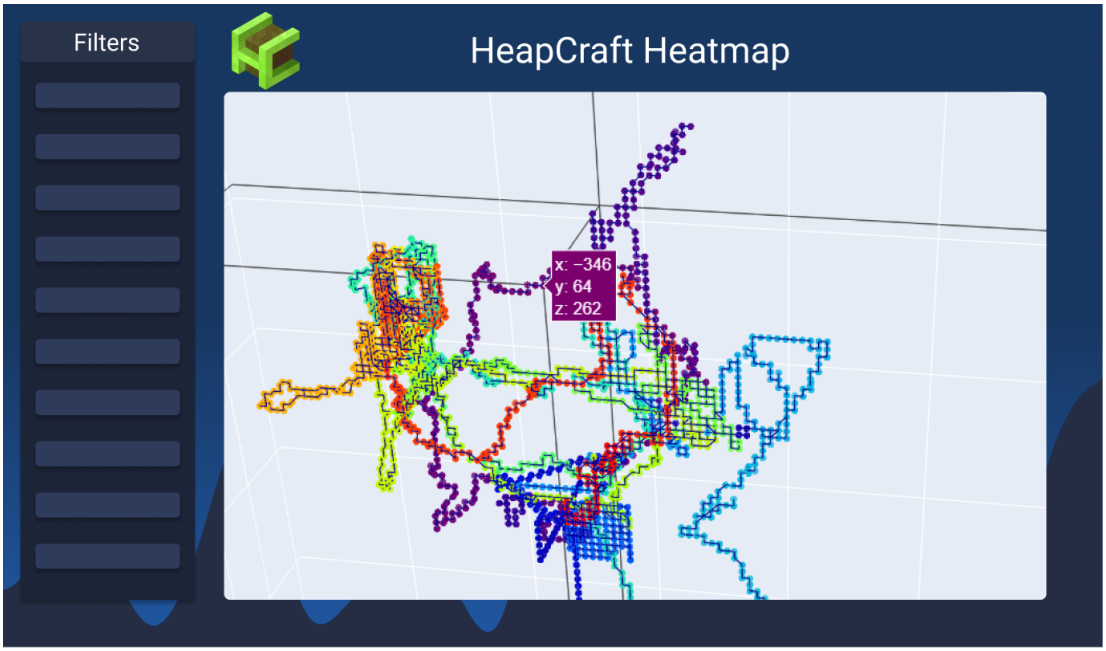

Figure 11.3: Third UI Iteration

11.2.4 Fourth Iteration

This iteration focuses more on making good use of screen real estate. The left filters bar now takes up the entire left side of the screen, instead of floating to the left as a panel, this makes more efficient use of screen space and allows us to display more filtering options. The logo has also been moved to the bottom right hand of the canvas, it is still there and noticeable, yet it is more discrete and out of the way.

One of the largest design changes here is the introduction of tabs. These tabs are intended to show options for other types of visualizations or other ways to analyze the HeapCraft data. Using tabs is a very intuitive design choice because most users will already be familiar with the concept, as tabs are implemented in most if not all web browsers. The size of the main canvas has also been increased, making better use of the space given.

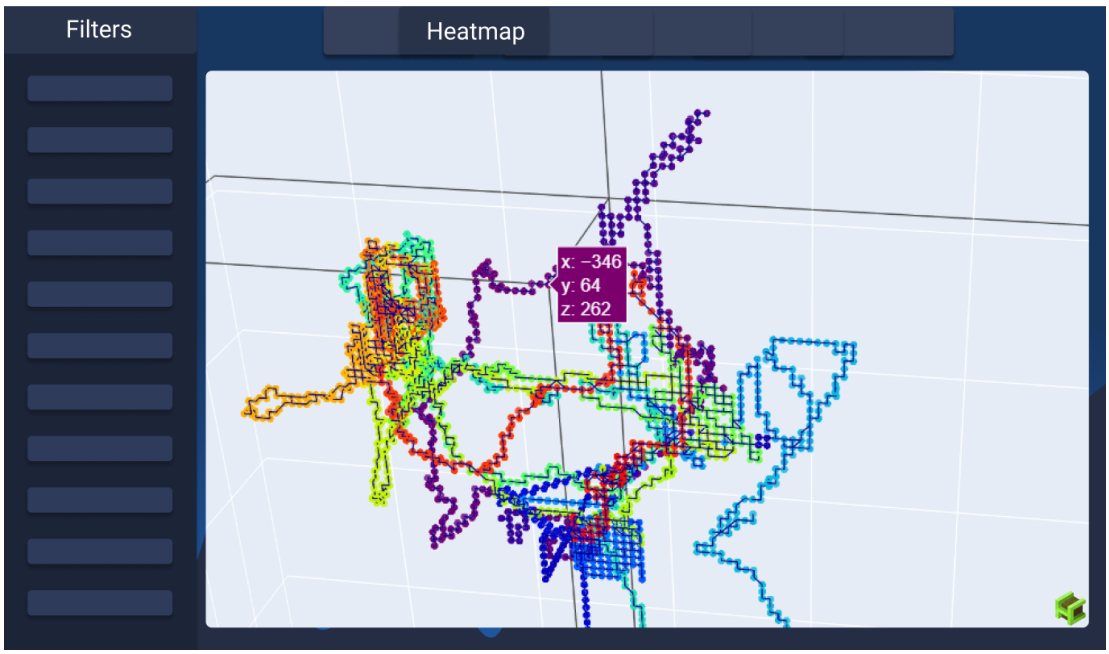

Figure 11.4: Fourth UI Iteration

11.2.5 Fifth Iteration

This iteration is mainly adding smaller details, such as a fullscreen button, a scrollbar to the filters window, and a 2D/3D selector for the heatmap. The 2D/3D selector will either disappear or be greyed out when visualization options are selected that don't use this feature. That space could then be used for additional options that only apply to that particular visualization. There were also some minor color changes, though the color palette may change as we test out more color schemes. Figure 11.5 below shows this design iteration.

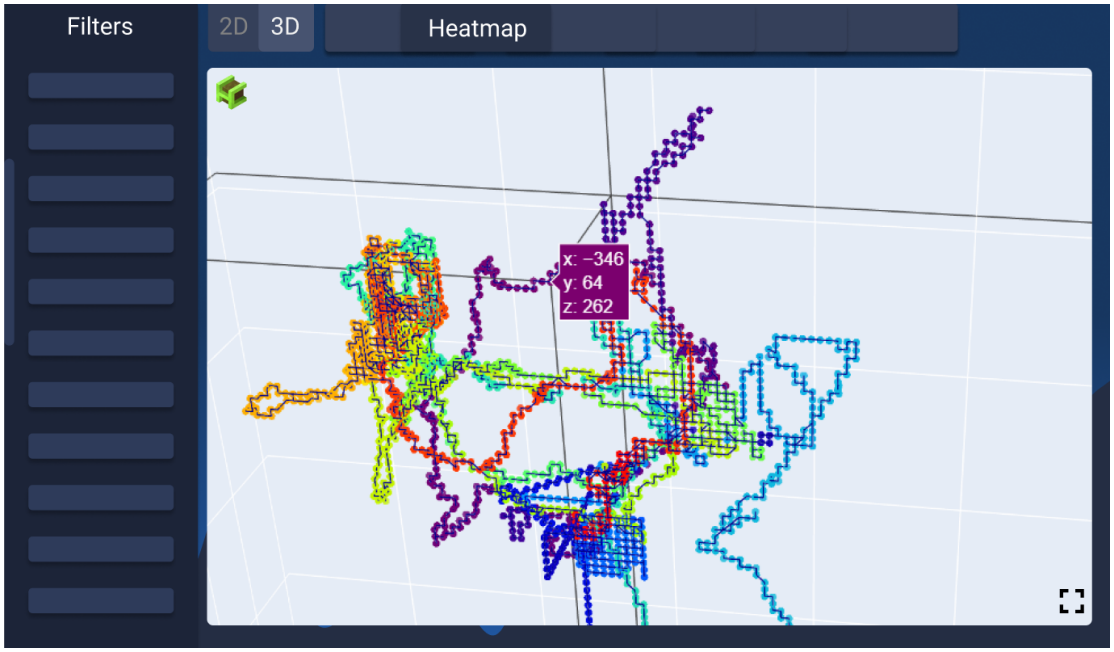

Figure 11.5: Fifth UI Iteration

11.2.6 Final Iteration

Unfortunately, we didn't keep track of the iterations from the fifth to the final. However, as it is shown, we kept the major component that allowed us to show the website to its best of its ability. To get a further look into the website, go to minecraftcollab.com.

In the first image is our home page. Here, there is a summary of the project, links to all important information about the project, and finally the members that worked on the project.

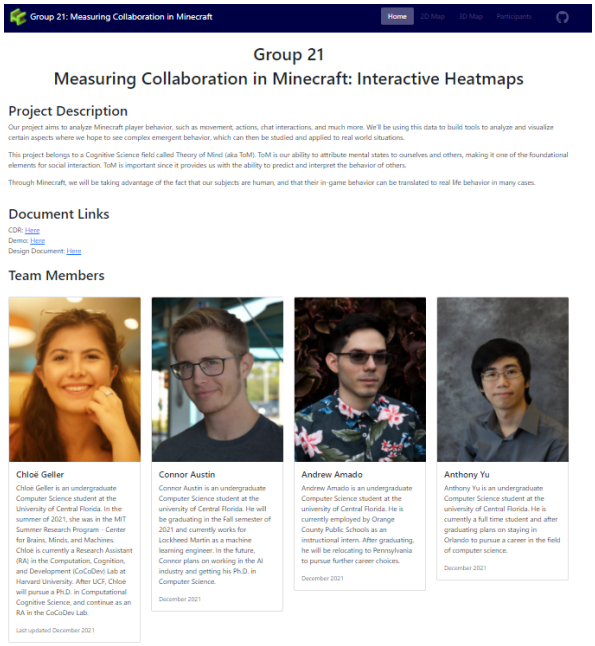

Figure 11.6: Final UI Iteration

In the 2D tab, we have all the events that we thought was useful to have in a 2D map. At the very top, there are two drop downs that allow you to control what you are seeing. The first one determines which world you are looking at, the Overworld, Nether, and the End. The second one determines which event that you want to look at. These include Movement, Block Place, and Block Break. On the right side of the map you can selectively choose which players you want to observe by clicking on certain players. On top of that, you can zoom in the map by dragging a box to the area that you want to expand on.

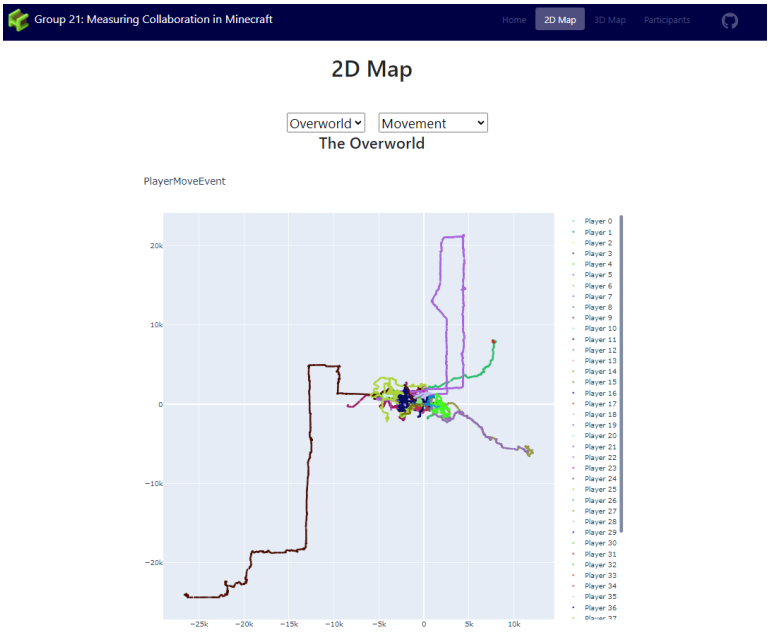

Figure 11.7: Final UI Iteration

In the 3D tab, like in the 2D tab, you can select what information you are seeing and select certain players. Unlike the 2D map, we can't select a window to zoom into an area, rather we can now rotate the camera angle to get a better look on the data in 3D.

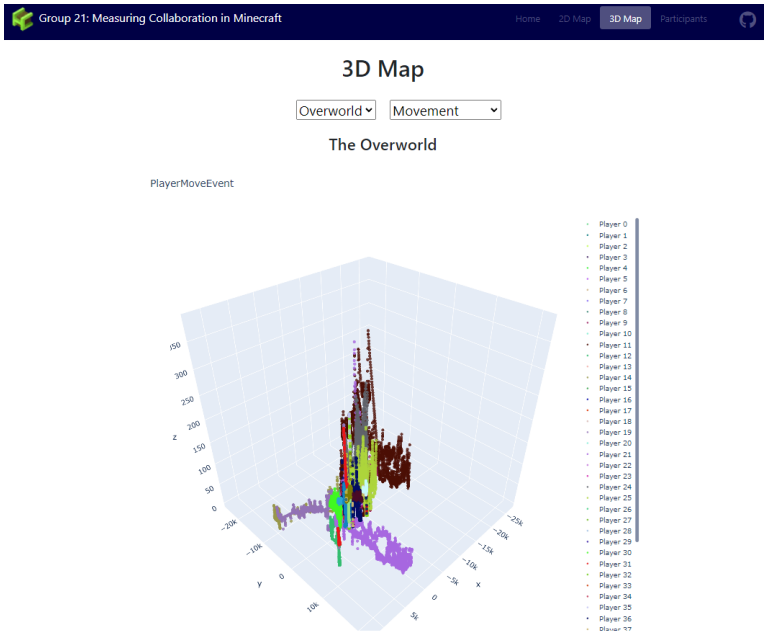

Figure 11.8: Final UI Iteration

In the participants tab, we have the information about the participants from the survey ordered in a visual way to understand the data better. At the end of all the Pre-Survey Data are several screenshots from the server that the participants built together.

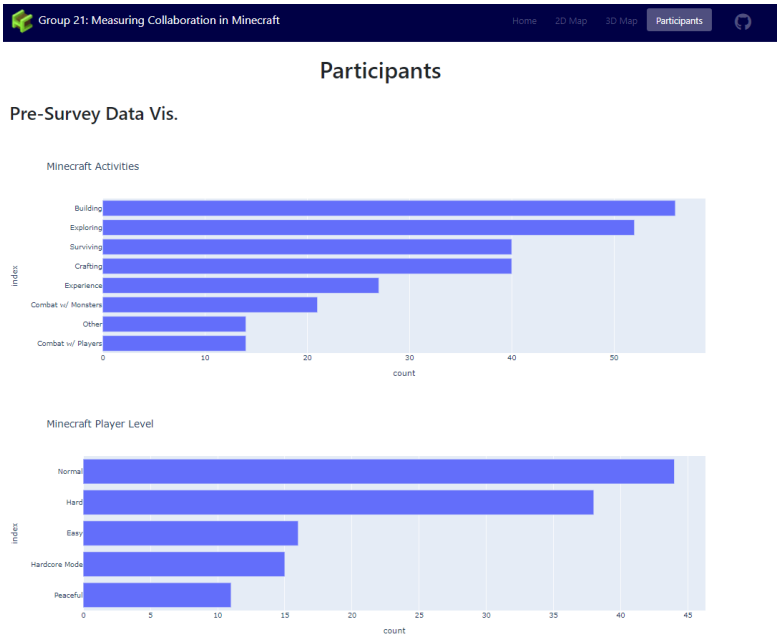

Figure 11.9: Final UI Iteration

Chapter 12

Communication

12.1 Team Communication

Discord is a VoIP platform that specializes in allowing its users to use text, images, video, and audio to communicate with one another. Discord operates by making communities called servers. Servers are a collection of persistent chat rooms and voice chat channels. While having multiple voice chat isn't useful for our purpose, having multiple chat rooms are. This allows us to separate our communications into different sections such as general, resources for coding, and Minecraft-related information.

The advantage of Discord is that it can be accessed through the mobile app, desktop app, and web browsers. Its ease of access and the fact that everyone in the group already had a Discord account made it the most appealing option for team communication [19].

Figure 12.1 shows the current discord setup for our group.

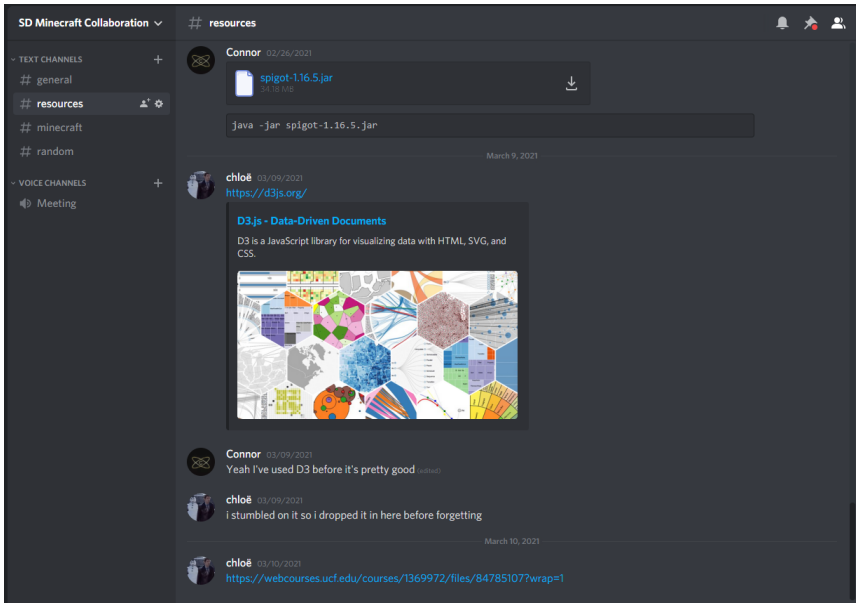

Figure 12.1: Discord Screenshot

12.2 Server Communication

The Minecraft server will be open to anyone that is willing to follow the rules and is willing to participate in the research project. Since the goal of the server is to track the communication between players within the server, communication between people that don't know each other will occur. To help facilitate this, we will be using Discord to make a Discord server dedicated to the Minecraft server. Within the Discord server will be multiple voice chat rooms to allow for multiple groups of players to communicate with one another without disrupting other groups. At the same time, we will make multiple chat rooms for our users to communicate through text and images if they choose to.

Connecting to a Discord server is simple. By making a permanent Discord invite link, people can copy and paste it into a browser or the discord app to connect to the Discord server. To spread the link to anyone that wants to join the Discord server, the link will be posted at the spawn location. After the players have read all the rules and voluntarily agree to join the research, will the link be made available to them.

12.3 Ideas of Collaboration using Communication

Discord allows for the creation of "Discord Bots". Discord Bots are automated programs that can automatically respond to certain events and commands on Discord.

12.3.1 Potential Ideas

A potential Discord Bot that could aid in the measurement of collaboration between players is a bot that tracks the voice chat rooms. The bot could track many metrics such as what voice chat rooms contain what users and for how long have they been in that room. This coupled with our automated friend's list could provide us with more interesting data about collaboration.

Another potential thing that the bot could track is the text chat between users. Things that are most notable that could be tracked is tracking user messages between one another within a set time frame, such as 5 minutes and the frequency of pings one user makes to another. A ping is essentially a built-in notification system in Discord. It is done by typing "@" followed by the recipient's Discord name in the server. This ping does two differentiating things from a normal message. First, it shows up with a yellow background and secondly provides a "ding" sound to the user being pinged.

12.3.2 Downsides

The biggest problem that comes with connecting the data from the Discord Bot with information from the server, is that the identity of the participant is no longer completely anonymous. This is because getting the interaction of a Discord user will provide no context to how they are interacting within the Minecraft server. To obtain meaningful information from the data, the identity of the Discord user must be matched with the identity of their Minecraft account. This in turn will cause anonymous problems because this forces one of the Minecraft data that must be collected to be their Minecraft identity. On top of that, since the players will be interacting with one another to collaborate, the anonymous problem is extended not only to us, the researchers, but everyone that they interact with.

12.3.3 Out of Reach Idea

A potential metric for the voice chat is for the Discord Bot to record the audio within the voice chat. A potential usage of that information is to provide it as data to an AI. This AI should be able to use both automated speech recognition and natural language processing to extract useful information from the data. Expanding on this idea, the AI could also be made to analyze the messages sent within the chat room. Using this data alongside other Minecraft tracked data, it can create a more in-depth analysis of the collaboration between users.

While an interesting idea, this provides a few major issues. Most notable is the act of recording audio and the large scale of this addition. First, the data collected from the audio can be identifiable and will influence many people's decision to participate in either or both the Discord and Minecraft server. The other downside is the large-scale implementation of such a project. Such an addition will make this a project that cannot be done within a couple of months.

Chapter 13

CITI Program

This chapter will discuss the Collaborative Institutional Training Initiative (CITI) Program and how we plan to abide by the three principles presented in "The Belmont Report". The three principles are respect of persons, beneficence, and justice.

Respect of Persons is the concept that everyone deserves the right to be treated as autonomous agents and anyone with diminished autonomy is entitled to protection. We plan to follow this principle by only collecting information and data from the Minecraft Character and its actions within the server. To further follow this principle and to ensure that everyone participating in the research is voluntarily participating, we will put a board at the spawn location. This board will contain information about the research, rules for the server, a link to a survey, and a notification that by leaving the spawn location, they are agreeing to their participation in the research [20].

Beneficence is the concept that the research project will minimize possible harm for the research subjects and to ensure maximum possible benefit. We plan to follow this by providing the server with common rules such as no racial slurs, (others). In addition to that, the participates are allowed to exit the research at any point by not logging back into the server. We will aim for the maximum possible benefit by making the server as enjoyable of an experience as it can be. This will be done by allowing the players to play the game as they feel fit that is within the set rules of the server [20].

Justice is the concept that the procedures are administered fairly in a reasonable and non-exploitative way. Our procedures will not allow anyone to be treated unequally and will strive to treat everyone equally. This will be monitored by our group. Our group will ensure that the procedures are followed properly by being the administers of the server. In addition to that, we will be monitoring the server's status to ensure that no bugs are interrupting the procedures [20].

Chapter 14

IRB Paperwork

Since our project involves doing research that requires human participants, we need to get approval from the UCF Institutional Review Board (IRB).

14.1 Institutional Review Board

The IRB is a committee that was established to protect the rights and welfare of human participants in research.

All research that involves human participants must be reviewed and approved by the UCF IRB. Prior to including human participants in investigations, researchers must obtain approval.

More details about the IRB Policy can be found on their website [21].

14.2 Paperwork

For the purposes of our project, we are aiming to be approved as Not Human Subjects Research (NHSR). To do so, we will be filling the following paperwork:

1. HRP-250-FORM-Request for NHSR
2. HRP-254-FORM-Explanation of Research
3. HRP-255-FORM-Request for Exemption
4. HRP-503-TEMPLATE-Protocol

If we were sure that our project would be approved as NHSR we would only be filing Form (1) and Form (2). However, we are unsure if we will qualify since we are using data gathered from people playing on our Minecraft server. Which is why we will also we filling out Form (3) in case we need to ask for an exemption instead. We may or may not need to fill out Form (4) depending on if the IRB committee requests it, and if the PhD students in Professor Sukthankar's lab. believe we need to as well.

Chapter 15

Minecraft Server

15.1 Minecraft Server Rules

To ensure that the Minecraft Server has some structure, we are implementing certain rules that should hopefully not even influence how the Player will play the game. These rules include:

- Be peaceful, and friendly with other players.
- No cheating.
- Be respectful of other players builds.
- No spamming in chat.
- No racism, sexism, or other forms of discrimination.
- No derogatory terms/hate speech.
- Be a decent human being.
- No harmful lavacasting. (In spawn or in other player's homes)

One Rule that we were on the fence about is the "No Griefing" rule. Griefing is the act of intentionally disrupting the immersion of another player in their game play with the goal to cause anxiety and distress. This can be done through many ways such as destroying buildings and constantly killing of others. Looking at griefing as a whole, griefing can be seen as a form of anti collaboration. While it could be related to collaboration, we decide to not allow it on the server. This is because being grief may cause a loss of player retention and will lower the quality of the server.

15.2 Survey

In order to join the server, we have configured the process so that a survey is required to join the server. This begins by having participants see the flyer through any form of advertisement. This will include our methods of recruitment and through word of mouth to friends. Within the flyer will be a link that leads to a UCF qualtrics survey. On completion of the survey, the participant will be directed to a Google drive PDF that

will provide them the information to log into the server to begin playing. Within this survey will be the following questions/information that we are looking for:

- Age (Complies with COPPA)
- Consent to Participating
- Following Rules Acknowledgement
- Minecraft Name
- Have you ever played Minecraft before this study?
- When you play Minecraft, do you play by yourself or with others?
- How would you rank your expertise in Minecraft?
- Which dimension(s) do you spend most of your time in when you're playing Minecraft?
- Which game mode(s) do you normally play Minecraft in?
- Which level(s) of difficulty do you normally play Minecraft in?
- When you play Minecraft, which activities do you focus on?
- Did you ask any friends, family, and/or co-workers to participate in the server with you?
- How did you hear about this study?

15.3 Plugins Used

Plugins on the server can be separated into two sections. They are data collection plugins, quality of life, and improvement plugins.

Data Collection Plugins

- Data Collection Core. This will be collecting all the information that we are looking for so that we can make our 2D and 3D Heat map and Automated Friends List.

Quality of Life and Improvement Plugins

- World Guard is a plugin that lets you and players guard areas of land against griefers and undesirable effects such as explosions and mob spawning. This is to be used for authorized server administrators.
- World Edit that essentially gives a player new tools to assist in the building process of a server. This plugin is also a dependency for WorldGuard. This is to be used for authorized server administrators.
- Essentials is a plugin that adds essential features for any new server such as setting a custom spawn point, setting warps, custom kits for new players, nicknames, jail, and permissions.

- EssentialsSpawn is an additional plugin that goes together with Essentials. It is needed specifically to add custom spawn points.
- GroupManager is one of the many permission plugins that will restrict and allow players to perform certain actions. Think of it as setting a policy as a system administrator to group on some OS.

15.4 Advertisement

Advertising for the server will come from two primary sources, friends and family that want to participate, and UCF Students. Our main way of recruiting UCF Students involved using the UCF CS, ECE and IT mailing list to email our flyers to all the UCF students within those majors. Our secondary way of recruiting more participates was through word of mouth. Participates gained through this method were typically given by the friends and family of the researchers and participates already playing within the server.

15.5 Challenges

15.5.1 Initial Challenges

Player Moderation

As our first challenge, we want to make sure that all players are following the set rules. Player moderation will be necessary in order to obtain reliable data from the server. It is yet to be determined whether moderation will be done by actual players, or have an automated moderation system. An automated system would be ideal, though its feasibility is questionable.

Our solution was to make the four of us, aka the researchers, the admins of the server. We had a dedicated Discord server for the server allowing for the participates to notify us about any possible problems such as server issues and any player issues.

Throughout the server's life, we had a couple of players that were causing mischief. These players created lava casts nearby spawn to obstruct spawn, exploded the builds of other players with tools or TNT. To combat this behavior, updates were made to the data plugin to track who places lava/water and in what locations this is being done. Using what we previously had made finding the culprit difficult and time consuming.

Player Count

This was tough to gauge. We initially did not know how many players we would have playing on the server, however it is necessary to ballpark this value, at the very least. Getting an idea of how many players there will be will help greatly when deciding what hardware we want in our hosting machine.

Increased player counts will require the host machine to have more dedicated RAM (Random Access Memory). This is to ensure a smooth experience for all players, without any lag or stuttering. However, increasing the amount of RAM on the host ma-

chine will increase the cost. So this problem bleeds into our expenses and budgeting as well.

We may also run into issues getting players with the new version of Minecraft coming out during our proposed time of running the server. If our server is running on an older version of Minecraft, less people will want to play. Then again, we do not know if the update will break our server plugins so this is a very tentative concern.

During the running of the server, the active player base never hit a point of needing to expand the machine that we decided to use, allowing us to not have to spend extra budgeting. Throughout the entire length of running the server, approximately fifty unique Minecraft accounts logged into the server. At the busiest time of the server, there would be approximately ten Minecraft accounts logged in at the same time, which our server was able to handle.

Data Organization

With all of the data that is being collected on the players, we need to ensure that all of the data is properly organized for ease of use when the data is processed. We need to ask ourselves how we want the data to be organized before we design the data processing step. When coming up with a methodology on how to do things, we need to keep all of the other steps in mind to make sure they all complement each other and do not cause any unnecessary bumps in the road.

Updating Epilog

The Epilog plugin was written for Minecraft version 1.12, which is severely outdated. We want to update Epilog to be compatible with Minecraft version 1.17.1, the current version as of the writing of this document. New features, items, and entities have been added since version 1,12, and these new features come with new events that we can collect and visualize, which makes it a top priority to make sure we add support for these new additions.

Epilog is currently using the Bukkit API, which is fine and does most of what we need it to do, however we will be using the Spigot API because Paper is a complete upgrade over Spigot and Bukkit. Initally, we decided on modifying Epilog's existing code to support the features of the current version of Minecraft. However, after looking through Epilog's existing code, we realized that Epilog collects a lot of unnecessary data, which in turn will raise the cost of storage. Instead, we decided to rebuild Epilog from the ground up to suit our needs.

Minecraft 1.17

At the time of beginning this project, Minecraft version 1.17 was being worked on and was scheduled for partial release mid 2021, so in short during the Summer time. The next date that Minecraft 1.17 is scheduled for full release would be during the holidays towards the end of the year. Our goal is to be working on our data collection plugin a bit during the summer in order to ease up the workload on ourselves so it is a little less worrying to do later on. With that being said, when the new major update comes out for Minecraft, there is no guarantee that our current build of the project may be compatible

with the latest. Along with that, we will have to wait for Minecraft 1.17 to receive API support which means we would be delayed if we wanted to have the plugin functional for Minecraft 1.17. Inevitably, our options are:

- Work on the data collection tool early, and then updating once Paper is released for 1.17. This option could take a while as we do not know if PaperMC is already in the works to shorten the time required to have a working version of their API put out for public use.
- Disregard Minecraft version 1.17 and focus on working in Minecraft 1.16.5 for the entirety of the project. While this is shorter, there is no guarantee that our plugin would work for future versions of Minecraft.

In the end, we chose option one. Updating the Data Collection Tool to 1.17.1 didn't cause any issues to occur. In addition to that, PaperMC was already updated within a few days of the launch of Minecraft 1.17.1

15.5.2 Challenges

Griefing

The launch of the server was not smooth. One of the biggest problems we encountered during the server's lifespan was griefing. Due to the fact that the world's spawn location will naturally be the most travelled area, it became the target of griefing. This didn't coincide nicely with the fact that spawn was the only central hub in the server, so many decided to invest time into building a base and community near the spawn. At the time, while we didn't have the explict rule that banned lavacasting, the act of putting water on top of lava to create large amounts of cobblestone, which is generally viewed as unpleasant, we had rules in place to help ensure that building weren't destroyed. Despite this, a user still decided to lavacast spawn, ruining many builds and wasting player investment. This event caused a variety of reactions ranging from quitting the server to being indifferent to the whole event.

As stated in the server rules, griefing wasn't allowed. While the rules addressed griefing in a vague way, the main grief that kept occurring was the act of lavacasting an area. To combat this specific problem, we decided to add the last rule of explictly stating that harmful lavacasting was not permitted. In addition to the adding of a new rule, we banned the user that caused the lavacast at spawn and tried to fix the damages to the best of our abilities.

Server Crashing

Another problem during the early stages of the server was the constant server crashing. In particular, the server would crash roughly from 2AM - 4AM, causing all current players to be forced off the server and unable to connect back to the server until one of the researchers restarted the server manually.

Identifying the reason for why the server was crashing was critical in maintaining player engagement. The cause can be traced back to the way we designed the Data Collection Plugin. Originally, the Data Collection Plugin would send data in batches every

minute, creating a new MongoDB connection every minute, causing the server to eventually crash. To fix this issue, we changed the way we created connections. Instead of creating a new connection every minute to send data, we created one at the start of the server/plugin, which we kept open the entire duration of the server/plugin.

15.6 Setting up a Minecraft Server

In order for our group to begin collecting data, we must have our Minecraft server up and ready to run. When we install this Minecraft Server, we do not have to use the traditional route of double clicking a `.jar` file and having it run that way. When installing the PaperMC API, there are different steps to successfully setting the server up.

15.6.1 Downloading Paper

This step is simple as we are only downloading the API. Once we head over to the official **PaperMC website** and then on the top right of the website, we click the 'DOWNLOADS' button, we are greeted with a web with a list of all the previous versions of the API. We download the latest version and save it to a new folder.

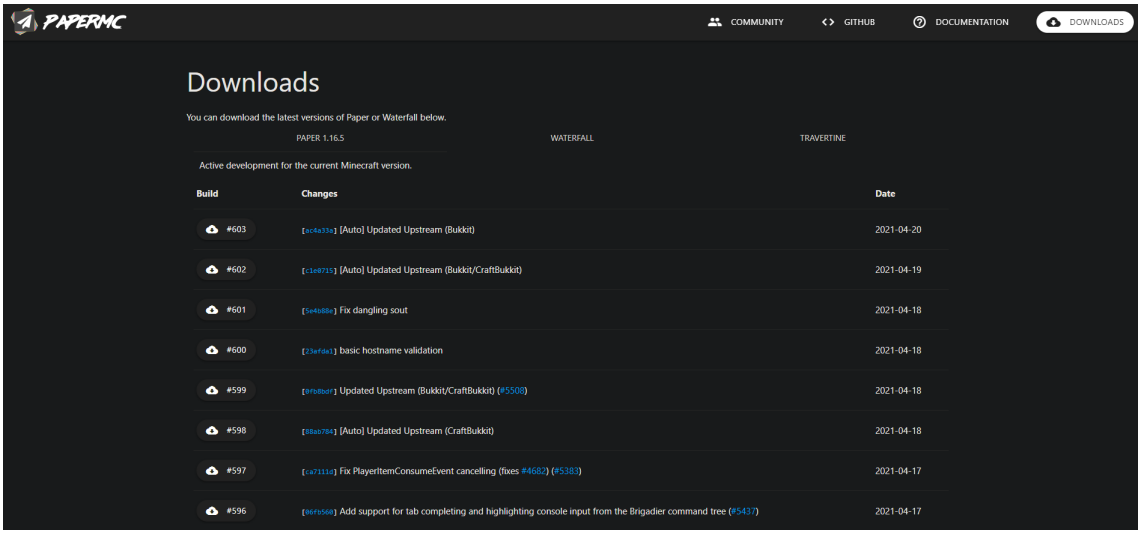

Figure 15.1: PaperMC Downloads Page

Afterwards, we create a new text file inside of the folder and paste the following code inside of it:

```
java -Xmx1024M -Xms1024M -jar paper.jar nogui
PAUSE
```

Listing 15.1: Shell code for running a Minecraft server with Paper.

Essentially what this code does is execute the jar file named `paper.jar` with 1024 megabytes of memory. Running this file will not bring up the Minecraft Server GUI either. Adding `PAUSE` to the end of the file will keep the command prompt open before the server closes and will close if we press any key. This is useful for if there is an unknown crash that occurs so we may find the cause quickly. We may also adjust how

much RAM we would like to allocate to the server at any time by editing the file again, but for now we can keep it at 1 GB.

Afterwards, we save and then rename the file to run.bat and double click to run. After doing so, the file will download some files and then crash. This is because we need to accept the EULA in the `eula.txt` file. To do this, we just set `eula=true`.

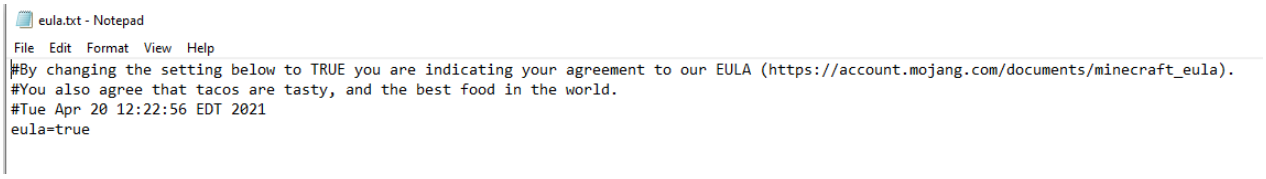

Figure 15.2: Minecraft Eula file

After this step, we are free to boot up the server once again by running the `run.bat` file once again.

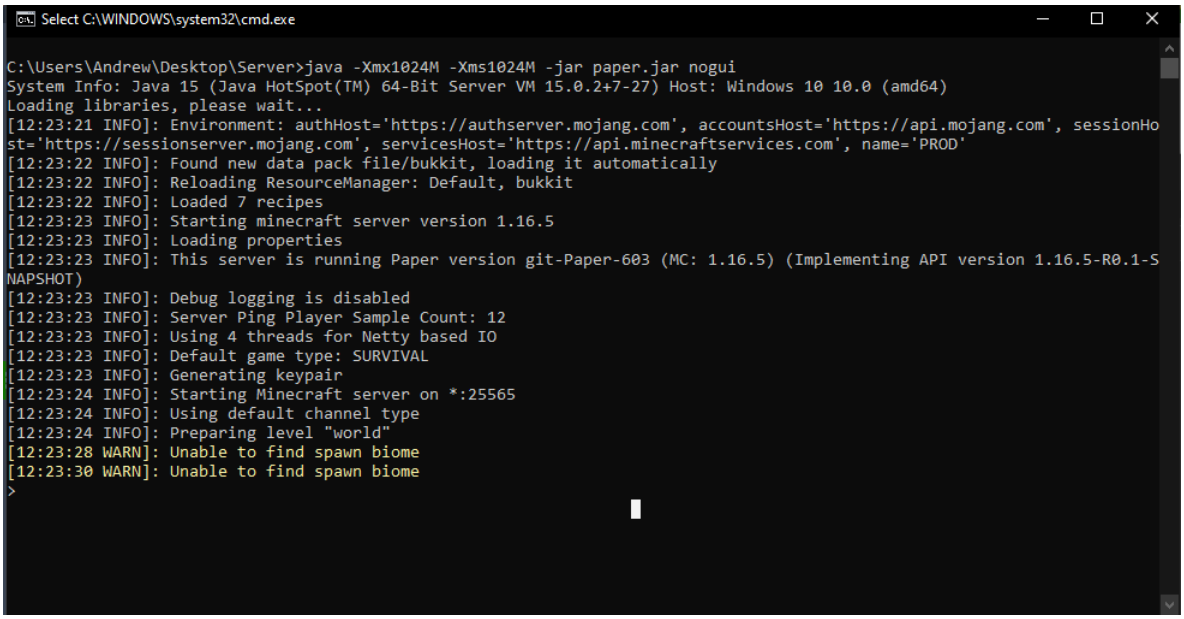

Figure 15.3: Successful run of batch file

Progression to this screen signals that we successfully accepted the EULA. Once we progress even further, additional files will be automatically downloaded into the root server folder. These files include:

- World files. These consist of the files for the overworld, nether, and the end.
- Server properties file. This file can be edited through notepad and offers the ability to slightly customize opetions such as spawn protection radius, world names and enabling flying.
- Whitelist. A list of users to allow in a server. If this list has one user inside, only that user would be able to join and no one else.
- Banlist. A list of users that are prohibited from joining the specific server.

- Operator list. A list of users that have been granted administrative powers.

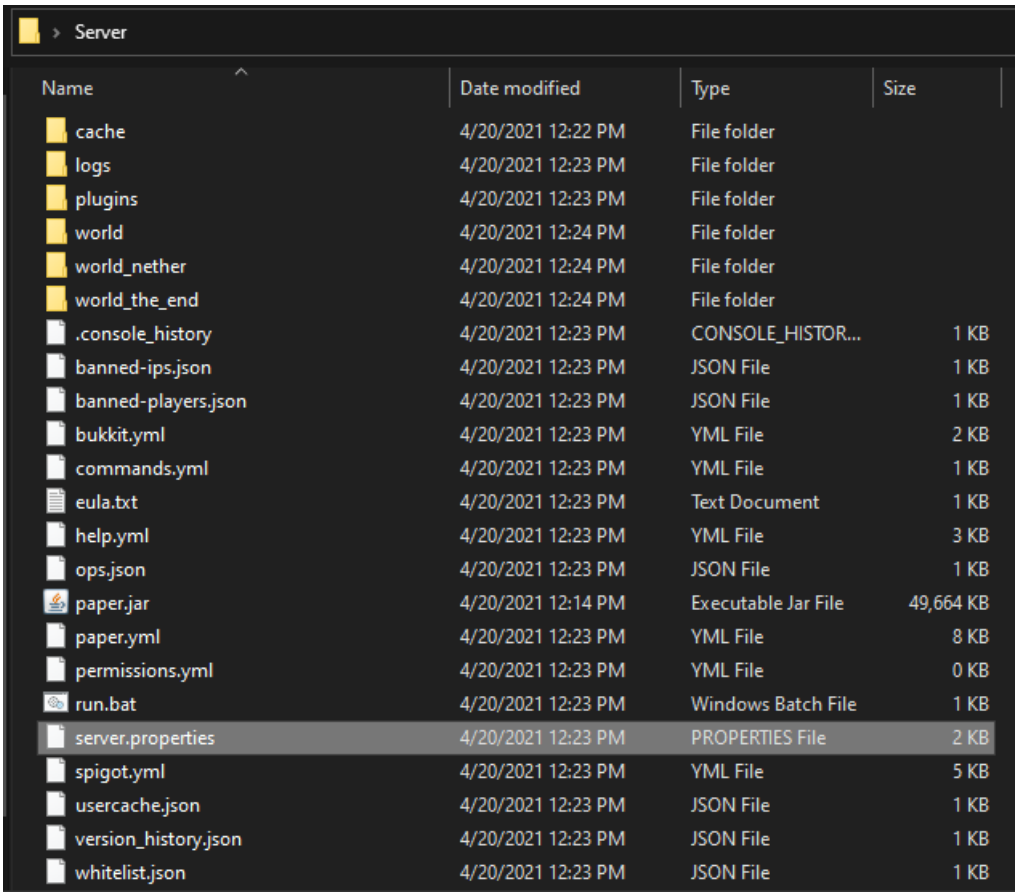

Figure 15.4: All Minecraft server files

15.6.2 Adding Plugins to a Minecraft Server

In order to add plugins into our server, all we have to do is download the plugin we would like to install. Plugins have a `.jar` file extension. After we download our desired plugin, we just have to drag the plugin into the `/plugins` directory and run the server or use the reload command on a server that is already running. Note that once you install a plugin on a server that is already running, you will have to stop the server in order to remove it as you cannot move files that are in use by another program in Windows.

15.6.3 Linux Setup

The process of setting a Minecraft server up on Linux mostly the same, but has its differences. Firstly we create a directory to store all of our server files inside.

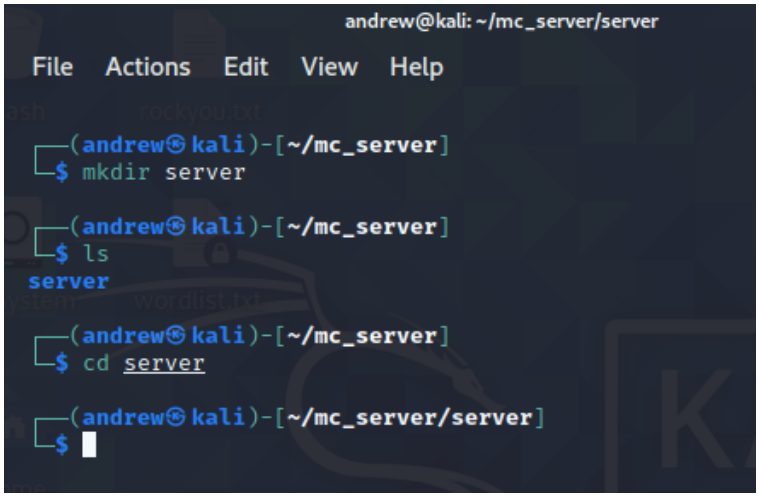

Figure 15.5: Creating a directory for our files

Afterwards we go back to the downloads page on the PaperMC website and get the link in which the file is located. We essentially will download the file at the given link and automatically place it inside the current directory that you are in.

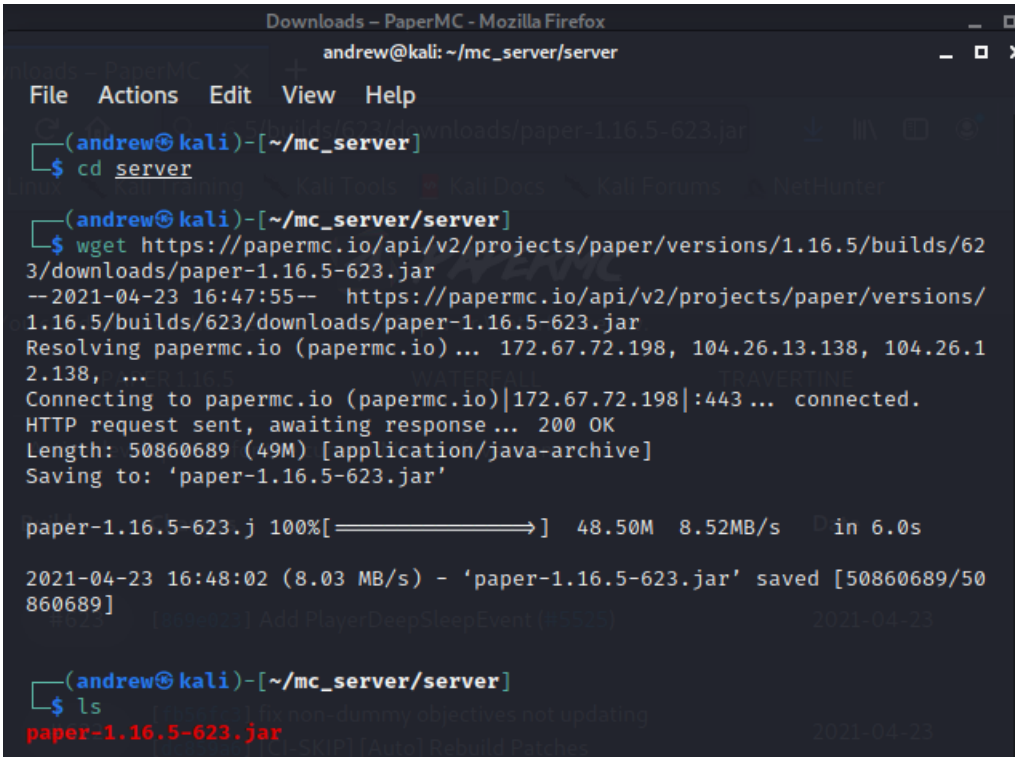

Figure 15.6: Downloading the API using wget

Once this file is installed, it is important we install Java or update it to its latest version. This is done with one command `sudo apt install default-jre`.

The next step is to create our `run` file. In Windows, we named it `run.bat`, but in Linux, we will name this file `run.sh`. We use `sudo nano run.sh` to create the file and begin to edit it. We will then enter in the same code as previously shown before and save the file.

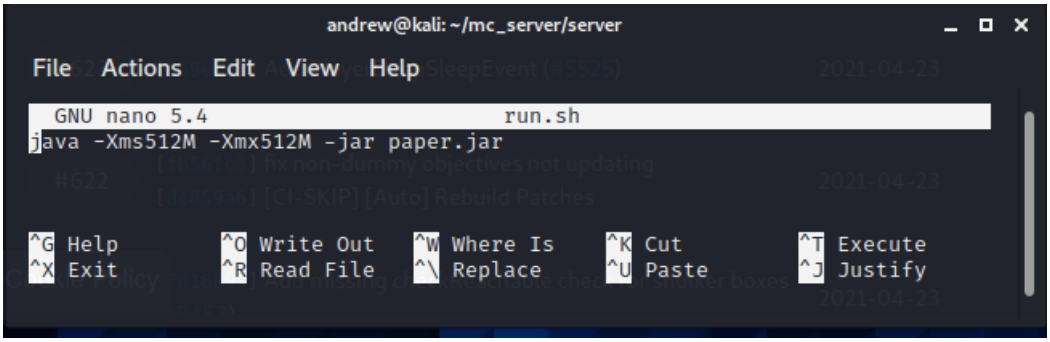

Figure 15.7: Creating run.sh and entering the necessary script code

All that is left to do now is run our `run.sh` file and then follow the same steps with changing `eula.txt` to true and rerunning `run.sh`. The final result is a fully functioning Minecraft server from Linux.

15.6.4 Connecting to a Local Server for Testing

During our testing phase of the project, we will not have purchased a server for our group members to use for testing. Our solution for this is to locally host a server for testing purposes. Once we have completed the testing phase, we will move on to then purchase a server and add our plugins in there. The only issue with this is that we cannot join the same world for testing purposes since that would require port forwarding which is another process in its own.

To join a server that is hosted on your machine, you will first need to run the batch file. After the initialization process is completed, we can open up our Minecraft client. After Minecraft has loaded, we select the Multiplayer option and then we select ‘Add Server.’ Afterwards, we would just type in ‘localhost’ in the ’Server Address’ box.

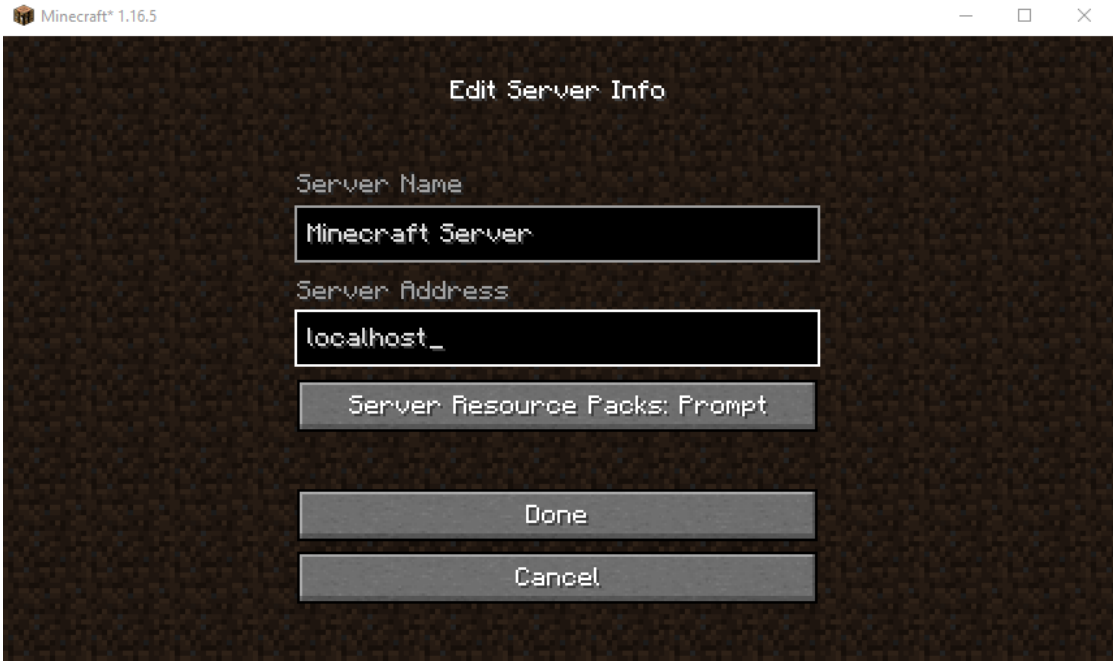

Figure 15.8: Adding our local server to the Minecraft client

Once you click done, you would see a server with 20 slots in it with the message ‘A Minecraft Server.’ After that, we can join and begin playing. Once you join, the server will log this. Usually any activity that happens in the console is logged into a file nearing the end of stopping a server. If for some reason your Minecraft server were to unexpectedly crash, looking into the logs for the specific date you ran the server may become useful for figuring out how to fix the issue.

Figure 15.9: Joining our local server in the Minecraft client with the server prompt above it.

15.6.5 Executing Commands on a Minecraft Server via Console

The server command prompt, or the console, is where all server activity is showed to a user. Typically, this user is the server administrator. It is important we do not give any person access to the console because the console has administrative powers. From the console, we can execute commands that would:

- Stop a server
- Restart a server
- Give players items
- Give players blocks
- Give players operator (administrative powers)
- Change game rules
- Teleport players
- Change the game mode of a player (survival, creative, adventure, spectator)
- Change time of the day
- Change the difficulty of the world
- Give potion effects to players

And this is only the beginning of it, but this just shows how much power the console has. Another way to think of the console is as the root user in a Linux system. With just these in our list, it is important that not any player can have access to the console.

These are all possible because the console can execute commands. In order to execute a command with the console, you must type in which command you would like to execute and the necessary arguments. For example, if we wanted to stop the server, we would just type `stop` following with the enter key. There are many more commands you may perform in the console, and others that require a player to send the command. For the full list of them, one may utilize the `help` command for a full list of commands that are available to them.

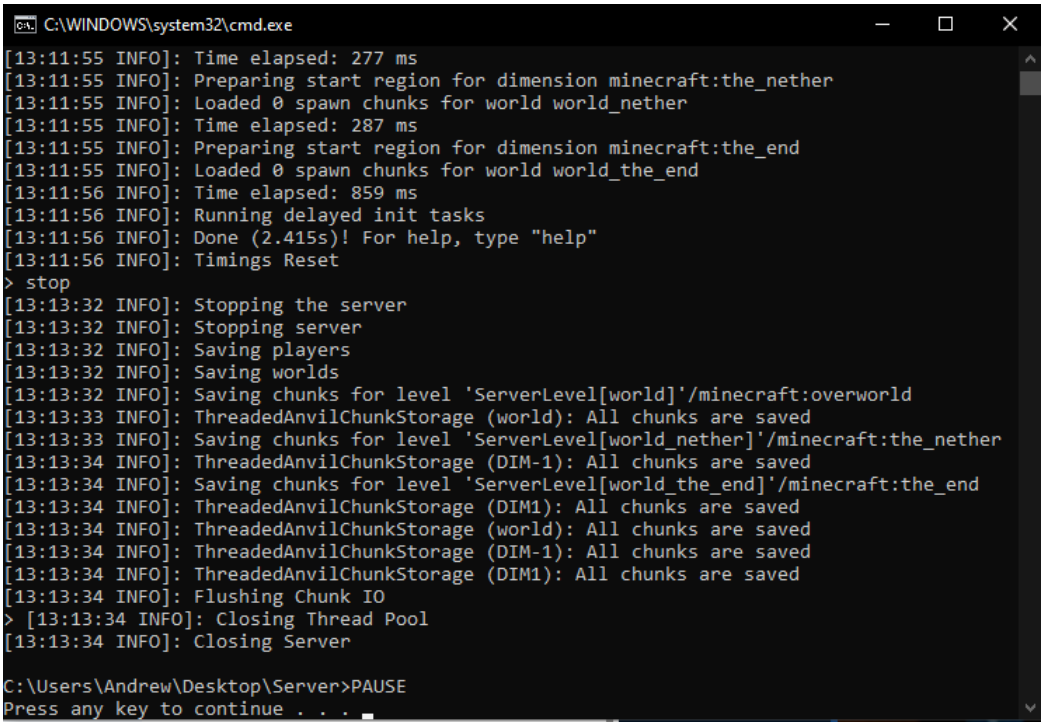

Figure 15.10: Stopping a server via the client

Executing commands in the console versus executing commands in-game as a player is slightly different. In the console, the only thing one must do to is type the command that they would like to execute. In game, you also have to include a forward slash before your command to designate that you would like to send a command.

15.7 Running the Minecraft Server

To be able to run our Minecraft Server, we needed many things to be finished. This included setting up the MongoDB to store all the data that we collect, having the data collection plugin fully operational, having the AWS server up and running, and obtaining IRB approval to start recruiting participants.

15.7.1 Setting up the Server

After deciding which EC2 instance we wanted, in this case, M5 Large, setting up the server was easy. Since the instance was a command line console that was running with Linux, installation of the server involved the same steps as the Linux setup outlined above. The only difference involved having to download the Paper server with the wget Linux command. Next step involved uploading all the plugins that we plan to use on the server onto our group Github. At the end, we used wget on the Github repository to get all the plugins and most importantly, our data collection plugin.

15.7.2 Setting up the Plugins

Using our plugins, we set up how the server would operate. WorldEdit was used to make an area suitable for spawn and simplified the process of building for the researchers. WorldGuard was used to ensure that the spawn was grief-proof and safe for players to login. Essentials, EssentialsSpawn, and GroupManager were all used in combination to help make the server run smoothly. Most importantly, this allowed us to change the global spawn location to our desired location and allowed us to constantly remind our participants of the rules of the server.

15.7.3 Maintenance of the Server

To ensure the best Minecraft playing experience for our players, we had a server Discord. The Discord allowed our participants to chat with one another. At the same time, it allowed our participants to notify any of the researchers about server problems or issues. The most common issue that occurred during the beginning of the Minecraft server was the crashing of the server. The band aid solution at the time was to manually restart the server by going into the AWS terminal. Following the fix that caused the occasional server crash, there has not been an issue in maintaining the server.

15.7.4 Closing of the Server

All participants will be told of the last day of the server on the server Discord. On the closing day, we will close the Minecraft server and perform our last pull of the data from the MongoDB Database. Some of our participants wishes for the continuation

of the server, however continuous running of the server will cost money. To form a compromise with our participants, we will be posting the Minecraft server save so that they can continue the world if they want.

Chapter 16

Project Summary

In this project, we have to dig into the development side of Minecraft. We have had to learn about the Bukkit, Spigot, and Paper API in order to develop our core plugins for the data collection and also use that collected data to draw conclusions about collaboration in Minecraft.

Our main goal is to create visualization tools to assist with quantifying and analyzing the collaboration between, and behavior of Minecraft players.

Since we are planning to make a modular hub, any other developer that would like to contribute to this project will have the ability to easily create their tool, and then add it onto our hub.

The code and data used for this project can be found **here**, and our project website can be found **here**.

16.1 Conclusions

Being able to collect out own data made data manipulations easier because we did not have to reformat Epilog's data set. We also had a better idea on which parts of data to sift out. After taking a look at the Epilog data set, we noticed their method of collecting data was very poor in terms of optimization. This allowed us to think of ways to decrease the amount of wasted space to our database.

Bibliography

[1] Stephan Mueller, Barbara Solenthaler, Mubbasir Kapadia, Seth Frey, Severin Klingler, Richard Mann, Robert W. Sumner, Dr. Markus Gross. *HeapCraft*. http://heapcraft.net/

[2] Charlotte Ruhl. *Theory of Mind*. Published Aug 07, 2020. https://www.simplypsychology.org/theory-of-mind.html

[3] Defense Advanced Research Projects Agency. *Artificial Social Intelligence for Successful Teams (ASIST)*. As of Feb 18, 2021. https://www.darpa.mil/program/artificial-social-intelligence-for-successful-teams

[4] Federal Trade Commission *Children's Online Privacy Protection Rule*. As of Apr 28, 2021. https://www.ftc.gov/enforcement/rules/rulemaking-regulatory-reform-proceedings/childrens-online-privacy-protection-rule

[5] Minecraft *Mojang account users under 13*. As of Apr 28, 2021. https://help.minecraft.net/hc/en-us/articles/360034753912-Mojang-account-users-under-13

[6] SpigotMC. *About Spigot*. As of Mar 8, 2021. https://bit.ly/2Oc9R96

[7] Development Stack. *Image*. As of Mar 22, 2021. https://morioh.com/p/c9c01dcad602

[8] Relational Databases vs Non-Relational Databases. *About Databases*. As of Mar 15, 2021. https://www.pluralsight.com/blog/software-development/relational-vs-non-relational-databases

[9] Example of Document Store. *Image*. As of Mar 22, 2021. https://lennilobel.wordpress.com/2015/06/01/relational-databases-vs-nosql-document-databases/

[10] Example of Column Database, Key-Value Store, and Graph Database. *Image*. As of Mar 22, 2021. https://phoenixnap.com/kb/nosql-database-types

[11] MongoDB. *About MongoDB*. As of Mar 15, 2021. https://www.guru99.com/what-is-mongodb.html

[12] About Mongoose *Mongoose*. As of Mar 25, 2021.
https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/

[13] Mongoose vs MongoDB *Mongoose vs MongoDB*. As of Mar 25, 2021.
https://stackoverflow.com/questions/28712248/difference-between-mongodb-and-mongoose

[14] Expressjs. *About Expressjs*. As of Mar 15, 2021.
https://www.guru99.com/node-js-express.html

[15] Expressjs. *About Expressjs*. As of Mar 15, 2021.
https://expressjs.com/

[16] Nodejs. *About Nodejs*. As of Mar 15, 2021.
https://nodejs.org/en/about/

[17] Comparing Angular and React. *Comparing Angular and React*. As of Mar 15, 2021.
https://www.guru99.com/react-vs-angular-key-difference.html

[18] Google Trend of Angular and React. *Image*. As of Mar 15, 2021.
https://trends.google.com/trends/explore?date=today%205-y&geo=US&q=React,Angular

[19] Wikipedia. *About Discord*. As of Mar 9, 2021.
https://en.wikipedia.org/wiki/Discord_(software)

[20] Belmont Report. *About The Belmont Report*. As of Mar 9, 2021.
https://www.hhs.gov/ohrp/regulations-and-policy/belmont-report/read-the-belmont-report/index.html

[21] UCF Office of Research & Commercialization *Institutional Review Board*. As of Mar 15, 2021.
https://www.research.ucf.edu/compliance/IRB/About/index.html

[22] Fandom. *Minecraft Wiki - Player*. As of Feb 18, 2021.
https://minecraft.gamepedia.com/Player

[23] Fandom. *Minecraft Wiki - World*. As of Feb 18, 2021.
https://minecraft.gamepedia.com/World

[24] Fandom. *Minecraft Wiki - The Nether*. As of Feb 18, 2021.
https://minecraft.gamepedia.com/The_End

[25] Fandom. *Minecraft Wiki - The End*. As of Feb 18, 2021.
https://minecraft.gamepedia.com/The_Nether

[26] Fandom. *Minecraft Wiki - Overworld*. As of Feb 18, 2021.
https://minecraft.gamepedia.com/Overworld

[27] Fandom. *Minecraft Wiki - Chunk*. As of Feb 18, 2021.
https://minecraft.gamepedia.com/Chunk

[28] Bukkit. *Official BukkitWiki*. As of Feb 18, 2021.
https://bukkit.gamepedia.com/Main_Page

[29] Java Documentation. *Bukkit API Docs. Version 1.16.5*. As of Feb 18, 2021. https://hub.spigotmc.org/javadocs/bukkit/

[30] Blair Felter. *About Cloud Computing*. As of Mar 8, 2021. https://www.vxchnge.com/blog/different-types-of-cloud-computing

[31] Microsoft. *About Cloud Computing*. As of Mar 8, 2021. https://azure.microsoft.com/en-us/overview/what-are-private-public-hybrid-clouds/

[32] Cloud Pyramid. *Image*. As of Mar 22, 2021. https://www.lucidchart.com/blog/cloud-computing-basics

[33] Cloud Competitors. *Image*. As of Mar 22, 2021. https://www.zdnet.com/article/top-cloud-providers-2019-aws-microsoft-azure-google-cloud-ibm-makes-hybrid-move-salesforce-dominates-saas/

[34] Microsoft Azure. *About Azure*. As of Mar 9, 2021. https://azure.microsoft.com/en-us/overview/what-is-azure/

[35] Amazon Web Services. *About AWS*. As of Mar 9, 2021. https://aws.amazon.com/

[36] Google Cloud. *About Google*. As of Mar 9, 2021. https://cloud.google.com/

[37] Wiki on Visual Studio Code. *Visual Studio Code*. As of Mar 25, 2021. https://en.wikipedia.org/wiki/Visual_Studio_Code

[38] Wiki on Eclipse. *Eclipse*. As of Mar 25, 2021. https://en.wikipedia.org/wiki/Eclipse_(software)

[39] PaperMC *GitHub - PaperMC*. As of Apr 27, 2021. https://github.com/PaperMC/Paper

[40] Agile vs Waterfall. *Agile vs Waterfall*. As of Mar 25, 2021. https://project-management.com/agile-vs-waterfall/

[41] Agile Diagram. *Image*. As of Mar 25, 2021. https://www.wearemarketing.com/blog/what-is-the-agile-methodology-and-what-benefits-does-it-have-for-your-company.html

[42] Waterfall Diagram. *Image*. As of Mar 25, 2021. https://searchsoftwarequality.techtarget.com/definition/waterfall-model

[43] PaperMC. *Frequently Asked Questions - PaperDocs*. As of Mar 8, 2021. https://paper.readthedocs.io/en/latest/about/faq.html#what-is-paper

[44] Fandom. *Minecraft Wiki*. As of Feb 18, 2021. https://minecraft.gamepedia.com/Minecraft_Wiki

[45] Fandom. *Minecraft Wiki - Block*. As of Feb 18, 2021. https://minecraft.gamepedia.com/Block

[46] Fandom. *Minecraft Wiki - Dimension*. As of Feb 18, 2021. `https://minecraft.gamepedia.com/Dimension`

[47] Microsoft. *Playing Minecraft*. As of Feb 18, 2021. `https://help.minecraft.net/hc/en-us/sections/360004960811-Playing-Minecraft`

Appendix A

Nomenclature

A.1 Technologies

- AWS - Amazon Web Services
- GCP - Google Cloud Platform
- GUI - Graphical User Interface
- VOIP - Voice Over Internet Protocol
- AI - Artificial Intelligence
- NN - Neural Network
- RNN - Recurrent Neural Network
- CNN - Convolutional Neural Network
- API - Application Programming Interface
- JSON - JavaScript Object Notation
- CSS - Cascading Style Sheet
- HTML - Hypertext Markup Language
- UX - User Experience
- UI - User Interface
- Bukkit - Software that enables the use of plugins in Minecraft Servers
- IDE - Integrated Development Environment
- SDLC - Software Development Life Cycle
- ODM - Object Data Modeling

A.2 Minecraft

- TPS - Ticks per second
- NPC - Non-Player Character
- Mob - Monster, enemy
- Skin - Texture of Minecraft Player Model
- Entity - All dynamic, moving objects throughout the Minecraft world. Such as, mobs, players, boats and minecarts.
- Steve - Default Male Minecraft skin
- Alex - Default Female Minecraft skin
- Redstone - Minecraft's form of power, like electricity

A.3 General/Administrative

- IRB - Institutional Review
- CITI - Collaborative Institutional Training Initiative
- DARPA - Defense Advanced Research Projects Agency
- ASIST - Artificial Social Intelligence for Successful Teams
- NHSR - Not Human Subjects Research

Appendix B

General Background Information

B.1 Characters

Users primarily interact with Minecraft as the **Player**, however, they may control additional NPCs in the game.

B.1.1 Appearance

When building their Player, users may select one of two skins, **Steve** (Figure B.1) or **Alex** (Figure B.2). These skins are intended to be generic representations of humans, though skins may be further customized and tailored to most of the User's desires. Players are generally 0.6 blocks wide and 1.8 blocks high. While sneaking, they shorten to 1.5 blocks high. Further, when gliding, swimming, or crawling they shorten to 0.6 blocks high. Entertainingly, they further shrink to a width and height of 0.2 blocks when sleeping [22].

B.1.2 Steve

Steve was the original Minecraft Player, introduced with the game's release in 2009. His skin is considered the "classic" skin and is considered the rendition of a "male" Player. His character model is also used for Human-like Mobs such as Zombies.

B.1.3 Alex

Alex was introduced in 2014 (Minecraft v1.8). her skin is considered the "slim" skin and is considered the rendition of a "female" Player. Notably, the "slim" characters have smaller arms.

B.2 Movement

Walking Players can walk approximately 4.317 meters per second (this is a 1:1 correspondence with blocks, as blocks are approximately 1 cubic meter). In the span of a Minecraft day (approximately 20 human minutes), Players can walk up to 5,181 blocks

Figure B.1: **Steve** [22]

Figure B.2: **Alex** [22]

(5.2 kilometers) – to achieve this, Players must walk in a straight line, ignore hunger, and must not sprint nor sneak [22].

Sprinting When sprinting, Players cover approximately 5.612 meters per second, though they will drain their hunger considerably. Notably, at 60% hunger (or less), Players will not be able to sprint. Sprinting also enables horizontal jumps of up to four blocks, compared to one block. As with typical Minecraft mechanics, if timed correctly, Players may just five blocks after at least 30 consecutive sprint-jump combinations [22].

Sneaking Sneaking prevents players from falling more than half a block. This effect is massively useful when building horizontally into an open space (e.g. to cross a canyon-like structure). If one uses sneak to pass a block's edge, but then disengages sneak, they will remain on that block – contrary to typical behaviors which would result in falling down to the nearest block beneath them. Additionally, sneaking does not prevent Players from dismounting blocks – they simply need to jump over the block's edge [22].

B.3 Worlds and Chunks

B.3.1 Worlds

A **World** in Minecraft is the playable area in which a Player can navigate. Within a World are three dimensions: the **Overworld** (Figure B.3), the **Nether** (Figure B.4), and the **End** (Figure B.5) [23].

- The Overworld is the dimension in which Players spawn, and begin their Minecraft journey [26].
- The Nether is a dangerous hellscape filled with fire, lava, fungal vegetation, and more hostile mobs (at all times, compared to the Overworld's darkness mechanics) [25].
- The End is a dark, space-like dimension made up of separate islands in the void made out of end stone. Most notably, the Ender Dragon resides in the End, which is considered Minecraft's "final boss" [24].

The entire World is programmatically generated based on a seed. While the World is divided into Dimensions, each Dimension is further subdivided into Biomes or Structures.

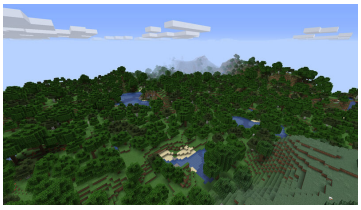

Figure B.3: Overworld [26]

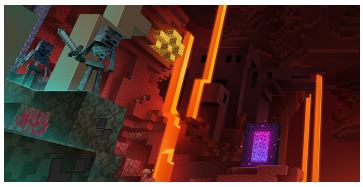

Figure B.4: Nether [25]

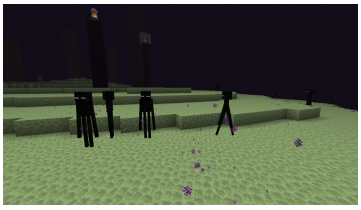

Figure B.5: End [24]

B.3.2 Chunks

The World is generated on a Chunk-by-Chunk basis as Players move about the World. Each Chunk is $16 \times 16 \times 256$ blocks, order $65,536$ blocks total. Chunks are generated using the map seed, which allows for deterministic World generation by reusing the same Minecraft version and seed value [27].

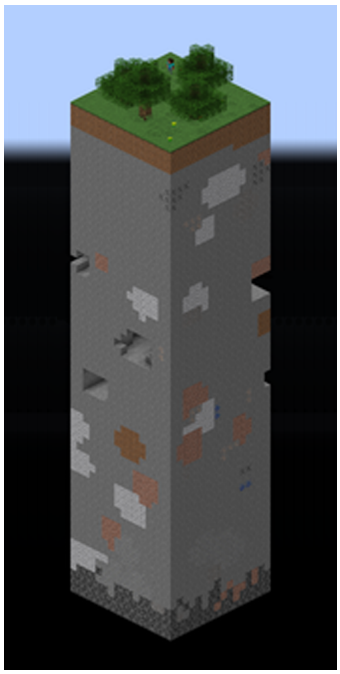

Figure B.6: Minecraft Chunk [27]

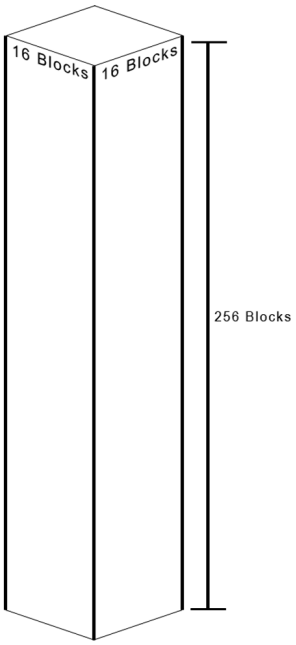

Figure B.7: Chunk Dimensions

Appendix C

Technical Background Information

C.1 Time in Minecraft

Similar to most video games, time works differently in Minecraft than in the real world. This is important for us to keep in mind when measuring player events such as the amount of time a player spent in a location or the amount of time it took for a player to find another player nearby.

Time in Minecraft is exactly 72 times faster than it is in the real world. Since it's a constant, it will be easier for us to make the conversions. Time is also measured in terms of **ticks**. A tick occurs every 0.05 seconds since Minecraft's game loop is around 20 ticks per second. Below we have included Table C.1 to help you visualize the conversions.

Time will be an essential aspect in our visualization, as data is organized chronologically. There are multiple functions within the API that we will be using to get the time of events, each returning different numbers with different purposes. This will be explained in further detail later in the document.

Minecraft time	**Minecraft ticks**	**Real time**
1 second	0.27	0.0138 seconds
1 minute	16.6	0.83 seconds
1 hour	1,000	50 seconds
1 day	24,000	20 minutes
1 week (7 days)	168,000	2.3 hours (2h 20min)

Table C.1: Minecraft time to real time

C.1.1 Time Cycles

Daytime Daytime cycles are the longest sections in Minecraft. They last for 10 minutes.

Sunset/dusk Sunset occurs between daytime and nighttime, and lasts $\frac{5}{6}$ minutes.

Nighttime Nighttime in Minecraft lasts for $8\frac{1}{3}$ minutes.

Sunrise/dawn Sunrise occurs between nighttime and daytime, and lasts for $\frac{5}{6}$ minutes.

We have also included Table C.2, as well as Table C.3 to help visualize the cycles in terms of ticks, and Minecraft time.

Period	Start	Mid	End
Daytime	0	6000	12000
Sunset/dusk	12000		13000
Nighttime	13000	18000	23000
Sunrise/dawn	23000		24000 (0)

Table C.2: Minecraft daylight cycles in ticks

Period	Start	Mid	End
Daytime	06:00:00.0	12:00:00.0	18:00:00.0
Sunset/dusk	18:00:00.0		19:00:00.0
Nighttime	19:00:00.0	00:00:00.0	05:00:00.0
Sunrise/dawn	05:00:00.0		06:00:00.0

Table C.3: Daylight cycles in Minecraft time

C.2 Bukkit API

Back in June 2009, Multiplayer for the Minecraft Java Edition was introduced. This opened up a world of possibilities of collaboration and cooperation with friends and strangers alike. You could build relationships or reinforce existing ones with friends across the globe. Minecraft servers quickly started to populate at every corner on the internet so players could join up and enjoy Minecraft's classic game mode of survival. However, things in the multiplayer scene started to become more interesting towards the end of 2010.

Figure C.1: Bukkit Logo [28]

In December 2010, Bukkit was first introduced to the Minecraft community. According to the Bukkit Wiki "Bukkit is a free, open-source, software project that provides the means to extend the popular Minecraft multiplayer server." The purpose of Bukkit was to essentially put the power of creativity, programming, and control in the hands of the Minecraft Community. This drastically changed the way Minecraft servers as a whole functioned. Throughout Minecraft's timeline, we would see an exponential rise of servers who would be using this software to create a vast amount of different server types.

Many of the top Minecraft servers would come to use this piece of software to run their servers and create their plugins. Servers like Hypixel or Mineplex to this day depend on this software to create their mini-games for millions to enjoy. Today, we see Minecraft servers categorized into which type of game mode they are providing such as hunger games, creative, towny, mini-games, survival multiplayer, etc.

Bukkit is a very versatile piece of software. It is used not only as the basis of which a modded Minecraft server is hosted, but is also used as a library for developers to realize almost any of their blocky ideas. The Bukkit API is well documented with every method and class that it provides. These can be found in the Bukkit API documentation for version `1.16.5` [29]. In terms of where to download it, which can be found **here**.

C.2.1 Bukkit Installation

For installation, a user would download and place the Bukkit JAR file inside of an empty folder in which they would like for their server to reside in their computer. Afterward, you have the option to create a script to run the program or you may simply double click it. When Bukkit runs for the first time, it will generate the necessary files and folders that are used for any Minecraft server with the addition of a folder called "plugins." To install a plugin to your Minecraft server, the `plugins` folder is where you will drag and drop all of the plugin JAR files you desire to have for your server. After your desired plugin is inside, all that is needed next is to run the Bukkit JAR file and you are good to go.

C.2.2 What is a plugin?

For the sake of definition, a "*plugin*" is essentially a server mod. This server mod does not inherently modify a game client directly, but instead, it communicates with the server and the server could perhaps do something special to a player if the plugin is written to do so. In short, a player does not have to install any client mods in order to join a Bukkit Minecraft server. The only thing the player would technically need is the vanilla (default) version of Minecraft that the server is currently running.

In Minecraft, the height in which you will die of fall damage in Minecraft is 24 blocks. However, if we choose that we do not want to die due to falling, we can choose to create a plugin to do that task. We would have to write a method that is related to the `EntityDamageEvent` and ensure that the entity that is taking the damage is in fact a player. Afterward, we would get the `DamageCause` that is built into the `EntityDamageEvent` via `getCause()` and check if it is `DamageCause.FALL`. Afterward, we can get the damage taken and check if that damage is greater than or equal to 20 (each heart is 2 damage). If everything checks out, we may call the `setCancelled(true)` method, which will cancel the `DamageEvent` and successfully negate fatal fall damage.

```
@EventHandler
  public void negateFatalFallDamage(EntityDamageEvent e)
  {
    // Return if damaged entity is not a player
    if (!(e.getEntity() instanceof Player))
      return;

    // Return if the damage cause is not due to falling.
    if (!(e.getCause() == DamageCause.FALL))
      return;

    // Return if damage is less than 20 (10 hearts)
    if (e.getDamage() < 20)
      return;

    // Cancel the event. Player doesn't take damage.
    e.setCancelled(true);

    return;
  }
```

Listing C.1: Code snippet with BukkitAPI of above scenario

Events are crucial in Bukkit. Without the existence of them built into the API, Bukkit developers would be very limited in terms of what they are able to manipulate in a Minecraft server. As explained above, doing something as simple as negating fatal fall damage takes one event, and each event has their own properties as we saw with how the `EntityDamageEvent` has a `getCause()` method. Events are invoked when the specific action occurs in the world. As an example, `EntityDamageEvent` would fire when any entity (not exclusive to players) has taken some sort of damage.

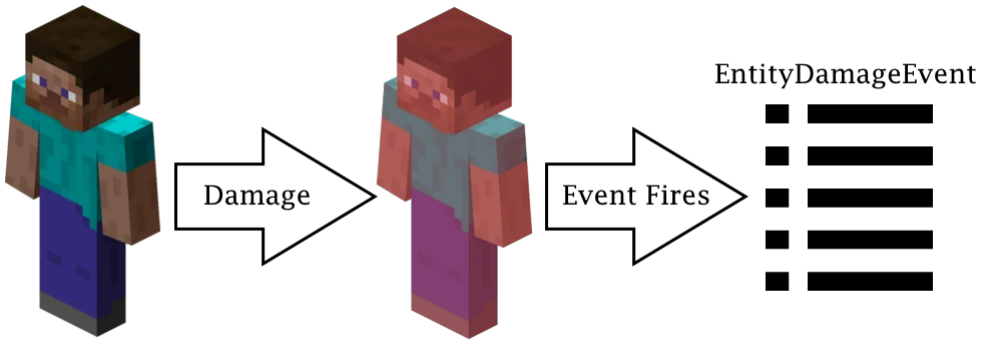

Figure C.2: Visual on how events fire in Bukkit

C.2.3 Project Application

Our group plans to use the Bukkit API to create our main data collection tool with the help of the events that Bukkit provides us. With access to the numerous events, we can collect and find out many things about player interactions in the world. As an example, a useful metric to collect could be to see which form of damage players take the most. Damage is not the only event though, there exists a wide range from things such as `PlayerFishEvent`, `PlayerLoginEvent`, `PlayerMoveEvent` to `BlockPlaceEvent` or `PlayerTeleportEvent`. In short, utilizing events that are important to us will be very useful for this project.

Overall, Bukkit is a very powerful tool that certainly changed the way Minecraft servers are today. Over the years, players have seen a vast variety of servers, each with their own unique twist to Minecraft.

Even though we will be developing our plugin using the PaperAPI, it is good to note that we can still develop our plugins the same as if we were using Bukkit. This is because Paper is backward compatible with Bukkit and Spigot (another API).

C.3 HeapCraft

Initially, when we were assigned this project, we were given a resource in case we found it helpful. This resource was HeapCraft [1], a Minecraft plugin that acts similarly to how we are planning to create our Minecraft plugin.

HeapCraft is essentially a set of tools that according to the HeapCraft website "..help Minecraft server admins to build strong, friendly, and collaborative communities." On the HeapCraft website, under the Tools tab, you can see the tools that have been created. These tools are split into three sections. The first section is the Web Interface. This section is mostly the web UI where you can see live statistics based on the players that are online.

Figure C.3: HeapCraft logo [1]

The second section of the Tools tab are the Plugins. In total there are three plugins, each of them having their own purpose. Epilog is the first plugin listen. According to the website "Epilog is the heart of our collaboration tools. Some of the other tools use the analysis done by Epilog to improve their functionality." Epilog's main function is to send behavioral data that it collects to HeapCraft's server.

The second plugin that is listed is the **DiviningRod** plugin. This plugin uses the data generated by Epilog for its application. In the chat, you type the command `/find <tag>`. DiviningRod has many tags built in such as:

- `socializer`
- `noob`
- `friend`
- `loner`
- `pro`

`socializer` will point the stick you get to the most social player, `noob` will point you in the direction of the newest player, `friend` points to a player to become friend with, `loner` points to the least social player, and `pro` to the most experienced player in the server.

The third plugin is called **Classify**. This plugin will automatically detect which actions a player may be doing and display those actions on the player list. There are several categories that this plugin will categorize you.

- Building
- Mining
- Exploring
- Fighting
- Idle

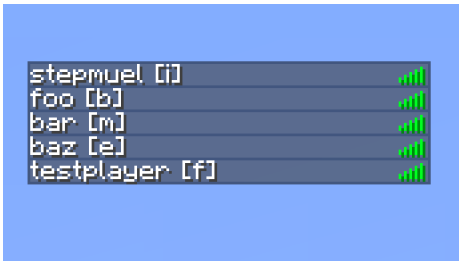

Figure C.4: Example of an application of Classify [1]

In short, you are assigned the first letter of the above categories.

The last category on the Tools page is Services. Here we have the **EpilogAPI**. This works like any other web API where you may send requests with your specific server key and the API will retrieve some sort of visualization. The visualizations that are offered seem to be a map of where people have mined in the world and a weighted graph representation of player relationships. These are each named MapMiner and GraphMiner respectively.

C.3.1 Framework

Figure C.5 below illustrates the functionality of the current HeapCraft framework. Server data will be taken from the main source, **Epilog**, and then transferred to a logging server which then organizes the data for its respective use.

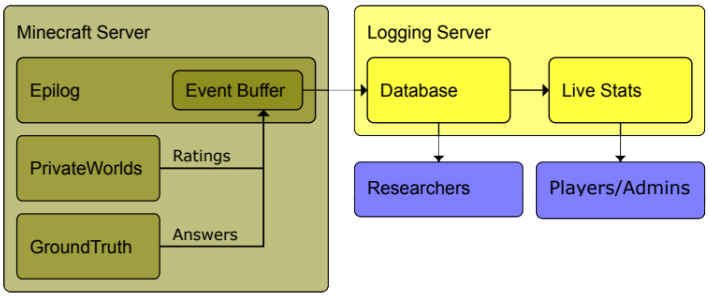

Figure C.5: HeapCraft Framework [1]

C.3.2 Framework with Web Interface

Figure C.6 below is an illustration of the overall framework, including the web interface that will help us visualize the necessary data:

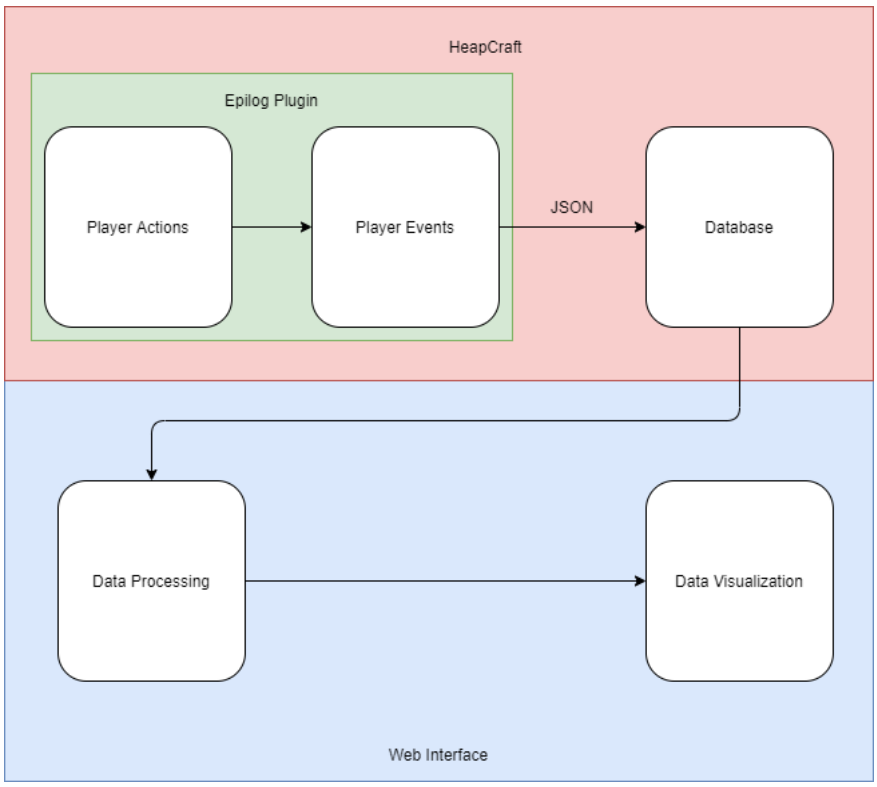

Figure C.6: HeapCraft Framework with Web Interface

Appendix D

Cloud Computing

Using Cloud Computing to store data allows developers lower cost, more flexibility, and better reliability than using a local server. In addition to that, this provides an easy option for the server to be on 24/7. There are three main types of cloud computing.

D.1 Public Cloud

Public Cloud most popular providers are Google, Amazon, and Microsoft. When using this service, the company provides the client with both service and infrastructure that is shared by all customers.

D.1.1 Pros

Public Cloud advantages comes in the fact that it has large amounts of available space, which allows for easy scalability. Due to this, its versatility and its "pay as you go" system allows for customers to buy more capacity as needed.

Another benefit is the provider's security and disaster recovery plan. With this benefit, this ensures that the system will be industry-standard and guarantees that the data stored will be secure. In addition to that, the disaster recovery plan ensures that if anything does go wrong, there are built-in redundancies. While this is good insurance, it comes at a high cost, which is a consideration with our $500 budget.

D.1.2 Cons

The downside to public cloud is that the essential infrastructure and operating system of the cloud are under the cloud provider. This means that changing providers will be difficult because there may be difficulties in repatriating assets. On top of that, if the provider does go out of business or make significant changes to the platform, customers will be forced to make major infrastructure changes on short notice. With the cloud architecture under the cloud provider, there is a risk of an unpatched security vulnerability [30, 31].

D.1.3 Structure

Within Public Cloud exist three different types of cloud service that build off one another. This means that these three services make a pyramid with infrastructure as a server being the foundation.

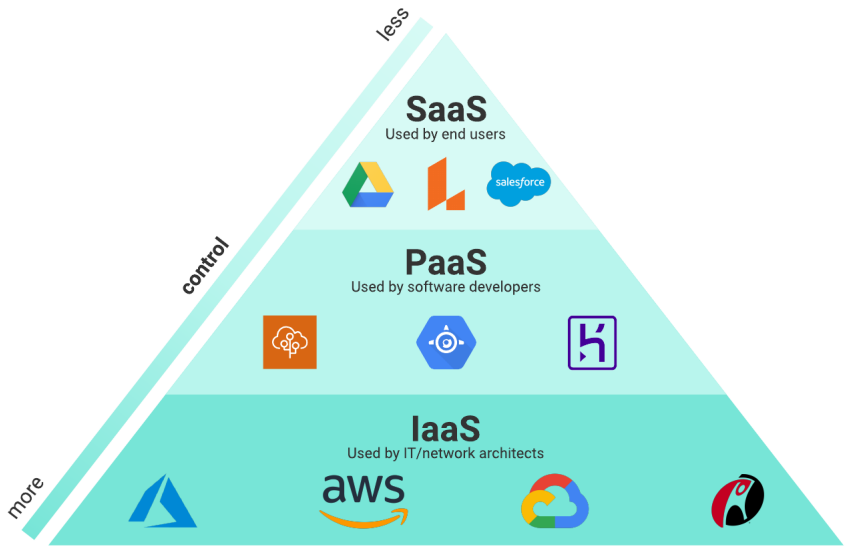

Figure D.1: Cloud Services [32]

D.1.4 Infrastructure as a server (IaaS)

IaaS is the foundation, meaning it contains the least features among the three. Due to this, it gives the highest level of flexibility. IaaS typically provides access to networking features, hardware, and data storage space. This gives the advantage of reducing cost because of the provided hardware and the latest security and patches because it is managed by the IaaS providers. Since the foundation already gives us everything we are looking for in a public cloud, it becomes the best option for us without overpaying [30].

D.1.5 Platform as a service (PaaS)

PaaS is one level higher than IaaS, giving everything IaaS and more. PaaS provides the framework needed to build, deploy, manage, and update software products. It does this by providing development tools, operating systems, and much more software needed to create software applications [30].

D.1.6 Software as a Service (SaaS)

On top of everything provided up to PaaS, at the top of the pyramid, SaaS is a fully-developed software solution. SaaS applications are useful when the customer doesn't want to deal with deploying their own hardware and software. Examples are Microsoft Office and Google Apps [30].

D.2 Private Cloud

Private Cloud are dedicated to one organization and usually behind a firewall. Private Cloud offers both security and control. This extra control allows for restricted access to valuable data and ensures that the data is only in trusted hands. Since the organization is the one controlling the cloud, there is no risk of sudden changes that will disrupt the infrastructure.

The downside to private cloud is the higher level of maintenance because the company owns the cloud. This means that the company is responsible for both the software and infrastructure, which makes it a less economical model than the public cloud. In addition to that, private cloud lacks versatility because expanding the cloud involves adding more hardware and storage capacity [30, 31].

D.3 Hybrid Cloud

Hybrid Cloud is a combination of both public and private clouds. Hybrid cloud uses multiple clouds that are designed to interact with one another seamlessly. Due to hybrid cloud being a mix of both, it gains the advantages of public cloud such as scaleability, and the advantages of private cloud's security and control. There are two commonly used types of hybrid cloud architecture. The first one is called cloud bursting, where the private cloud is used as the main cloud, storing data in a secure environment. Eventually, when the private cloud capacity can't keep up with the demands, the public cloud's computing resources are used to help supplement the private cloud. The second one uses the private cloud to store more confidential and restricted access information, while storing general data in the public cloud.

The downside to hybrid cloud is due to the nature of hybrid cloud using two different clouds to communicate with one another. There could be potential performance and security risks when the two clouds communicate and share data with each other [30, 31].

D.4 Chapter Conclusion

Since we do not plan to store any highly confidential data, private clouds and hybrid clouds are excessive for what we are looking for. Public cloud's advantage of being cheap and its ability to expand the storage on demand makes it a great option. Looking at its disadvantages, it does not affect us greatly. Since this server is planned to only be hosted for a few months, the changing of providers is highly unlikely. On top of that, public cloud's providers are popular and trusted, therefore making it unlikely for the disadvantages to affect us.

D.5 Public Cloud Computing Options

There are three main providers for public cloud computing that we will be taking a look at: Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP).

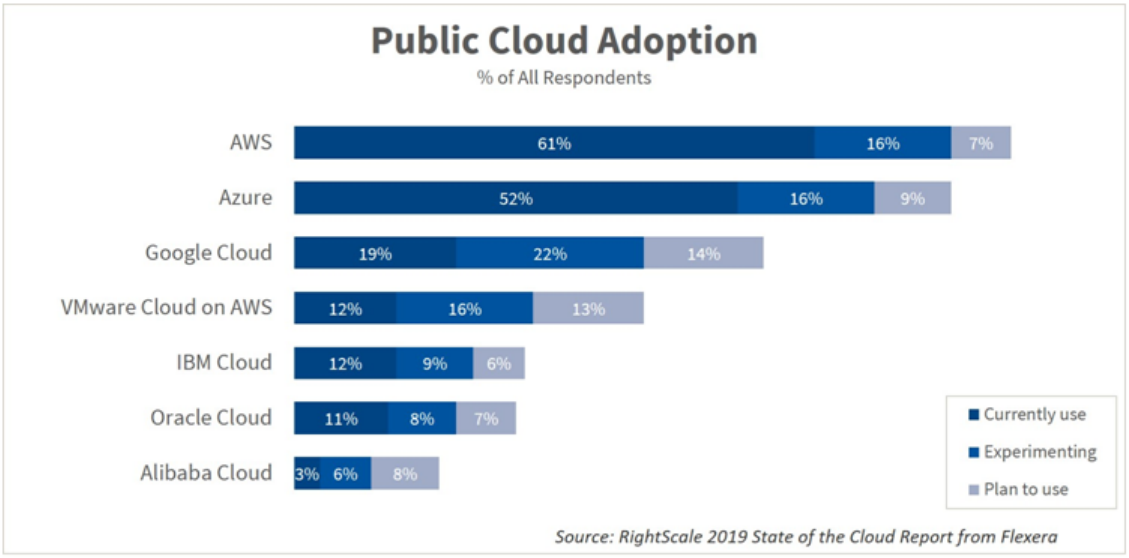

Figure D.2: Cloud Competitors [33]

D.5.1 Microsoft Azure

"The Azure cloud platform is more than 200 products and cloud services designed to help you bring new solutions to life—to solve today's challenges and create the future. Build, run, and manage applications across multiple clouds, on-premises, and at the edge, with the tools and frameworks of your choice."

Microsoft Azure supports server hosting through "Azure Dedicated Host" and supports database storage through "Azure Database for MySQL", "Azure SQL Database", and many more. [34]

D.5.2 Amazon Web Services

"Whether you are looking for compute power, database storage, content delivery, or other functionality, AWS has the services to help you build sophisticated applications with increased flexibility, scalability, and reliability."

AWS supports server hosting through "Amazon Lightsail". Lightsail is a virtual private server that allows for easy hosting from Minecraft Servers to web applications. Lightsail uses Amazon EC2 and is intended for small developers who don't need the additional complex functionalities. AWS also supports database storage for popular options such as MySQL, SQL, and many more [35].

Another feature that AWS has is their EC2 (Elastic Computing) instances. These are great to get a fast virtual machine up and running that will scale with your computing demands. The payment plan is based on how much computing time you use, so you're never paying for server downtime, making this another great choice for hosting a Minecraft server.

D.5.3 Google Cloud Platform

"Whether your business is early in its journey or well on its way to digital transformation, Google Cloud's solutions and technologies help solve your toughest challenges."

Google Cloud supports server hosting through "Google Compute Engine" and also supports database through "Cloud SQL". While Google does support what we are looking for in a cloud provider, we can not choose it. This is due to the sponsor's budget of not supporting Google Cloud [36].

D.5.4 Conclusion

The cloud service we will be using to store data and host the HeapCraft server will be Amazon Web Services (AWS). One of the primary reasons for choosing AWS over other cloud services such as Microsoft Azure, or Google Cloud Platform, is familiarity. As a group, we have a greater understanding of AWS and how to use it for our needs. Using a cloud service provider that is already familiar will ensure we do not waste time learning an entirely different platform.

In general, the pricing models between the three main cloud providers (AWS, Microsoft Azure, Google Cloud Platform) are all similar, so pricing had little to no bearing on our choice of a cloud computing provider.

We will also be taking advantage of Amazon Web Service's 'Free Tier' systems, which allows 750 hours per month of free computing time. This will be the biggest cost saver since our main expense in the first place is server hosting.

D.6 Cloud Computing Framework

Figure D.3 illustrates the structure of cloud computing for our use case:

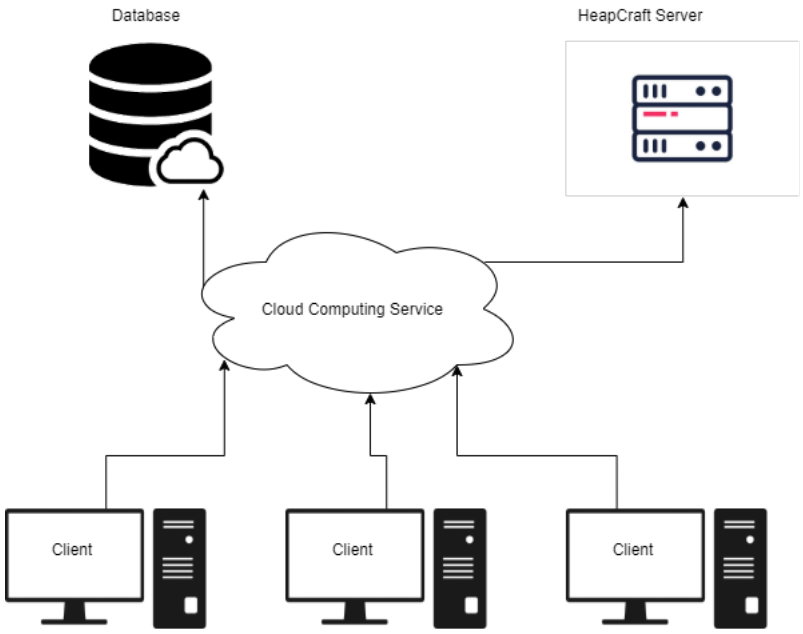

Figure D.3: Cloud Computing Framework

Appendix E

Development Environment

It is important we have an environment to help us along the development process. Whether it be for the plugins, or the web interface for our HeatMap, we may have different options in mind that we would be considering when choosing a specific IDE for the task. Mainly, the IDE that our group will be using would either be Visual Studio or Eclipse. The IDE that each member of the team uses will be based on preferences and familiarity with the environment.

E.1 Visual Studio Code

Visual Studio Code is a source-code editor developed by Microsoft. Visual Studio Code can be used to develop a wide range of applications from websites to mobile apps. Importantly, Visual Studio supports many different programming languages, which include JavaScript, TypeScript, HTML, and CSS. On top of that, it supports Python, Ruby, and Node.js. This makes it an ideal environment to code because of its support for the languages we will be primarily using, which are JavaScript and Node.js [37].

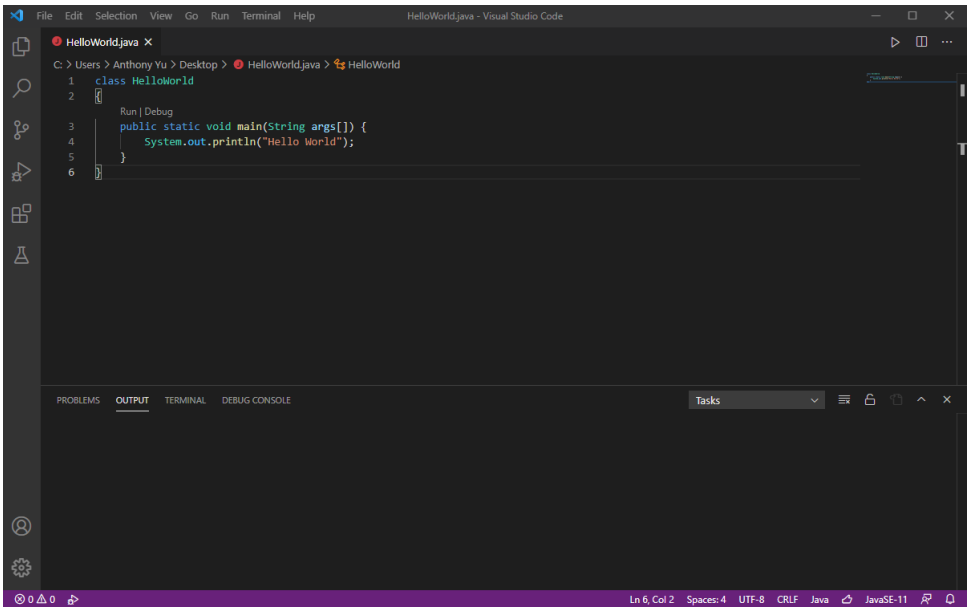

Figure E.1: Visual Studio Code

E.2 Eclipse

Eclipse is an IDE that primarily focuses on Java applications. However, through the usage of its extensible plug-in system, it can be used to support JavaScript, Node.js, and many others [38]. More specifically, this IDE will be in handy when developing our Minecraft plugins.

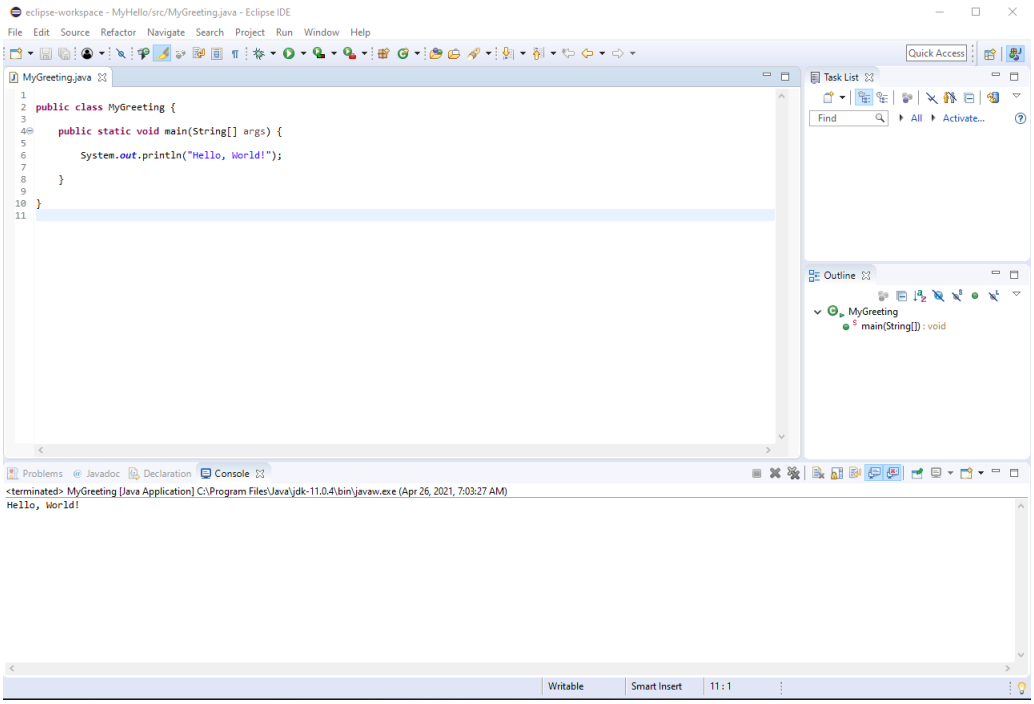

Figure E.2: Eclipse

E.2.1 Creating a plugin in Eclipse

The process for creating plugins in Eclipse is very simple. Before you start any coding, create a new Project and select 'Maven Project' and click next. Check the box that says "Create a simple project" and then fill out the next prompts.

Once the project has been created, open it up on the Package Explorer and then open up the `pom.xml` file. Here is where we will add in our dependencies and repositories. On the PaperMC GitHub [39], there is code for developers to copy and paste to their `pom.xml` files to import the API. The code is as follows:

```
<repositories>
    <repository>
      <id>papermc</id>
      <url>https://papermc.io/repo/repository/maven-
   public/</url>
    </repository>
    </repositories>

  <dependencies>
    <dependency>
      <groupId>com.destroystokyo.paper</groupId>
      <artifactId>paper-api</artifactId>
      <version>1.16.5-R0.1-SNAPSHOT</version>
      <scope>provided</scope>
  </dependency>
    </dependencies>
```

Listing E.1: Code for installing PaperMC API in Maven.

After saving and closing out of this file, if you take a look at the package explorer and look at your maven project, you will see a new tab called ”Maven Dependencies.” If you have done the aforementioned steps correctly, you should see the following:

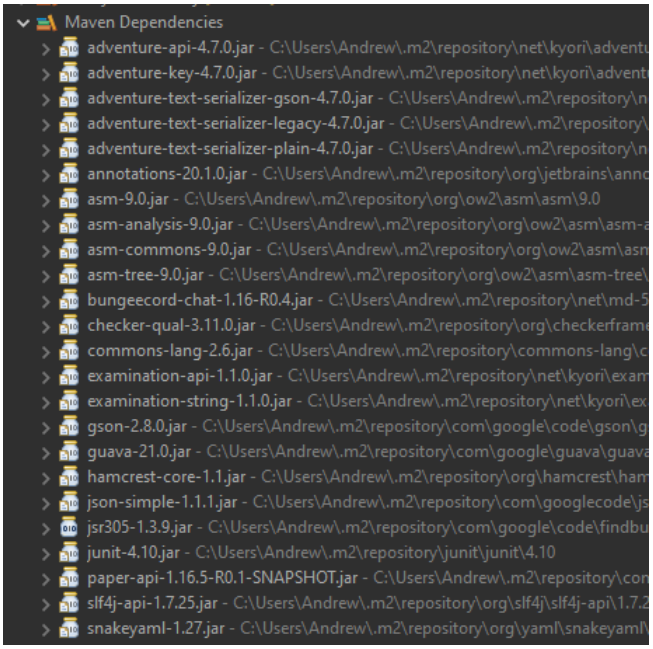

Figure E.3: Correct Maven dependencies

E.2.2 Exporting a plugin in Eclipse

Exporting our finished code is our last step in the plugin development process. Before we begin exporting however, we must first include a new file, `plugin.yml`. Without this file, the plugin will not function at all. You must include all commands that have been created in development. Take for example the TimeCheck plugin. In TimeCheck, we added a new command named `ct` which would check the time. As an example, here is an example of a `plugin.yml` file from our TimeCheck plugin.

Plugin.yml

```
main: me.storm.Main
name: TimeCheck
version: '1.0'
description: Time checker
author: Stoworm
commands:
  ct:
   description: Checks the time
```

Listing E.2: plugin.yml file

There are a couple of sections that are required for a plugin.yml file to work.

- `Main` refers to path of the main class. In plugin development, the Main class is the class that extends the JavaPlugin, a class in the Paper API.
- `Name` refers to the name of the plugin. This name will appear in the plugin list when using the `plugins` command and as well the folder for any generated files the plugin may create.
- `Version` refers to the version of the plugin. This number is determined by the author and is at their discretion.
- `Description` refers to a description for the plugin that is created.
- And as mentioned before, `commands` refer to all the commands that the plugins will include.

Exporting to a JAR file

Once we have all of our necessary files, right click the project and click on "Export." Navigate to the Java folder and select the "JAR File" option and then click Next. Then we select the destination path and select "Finish." If we were to open up that directory, we should find the file at that location.

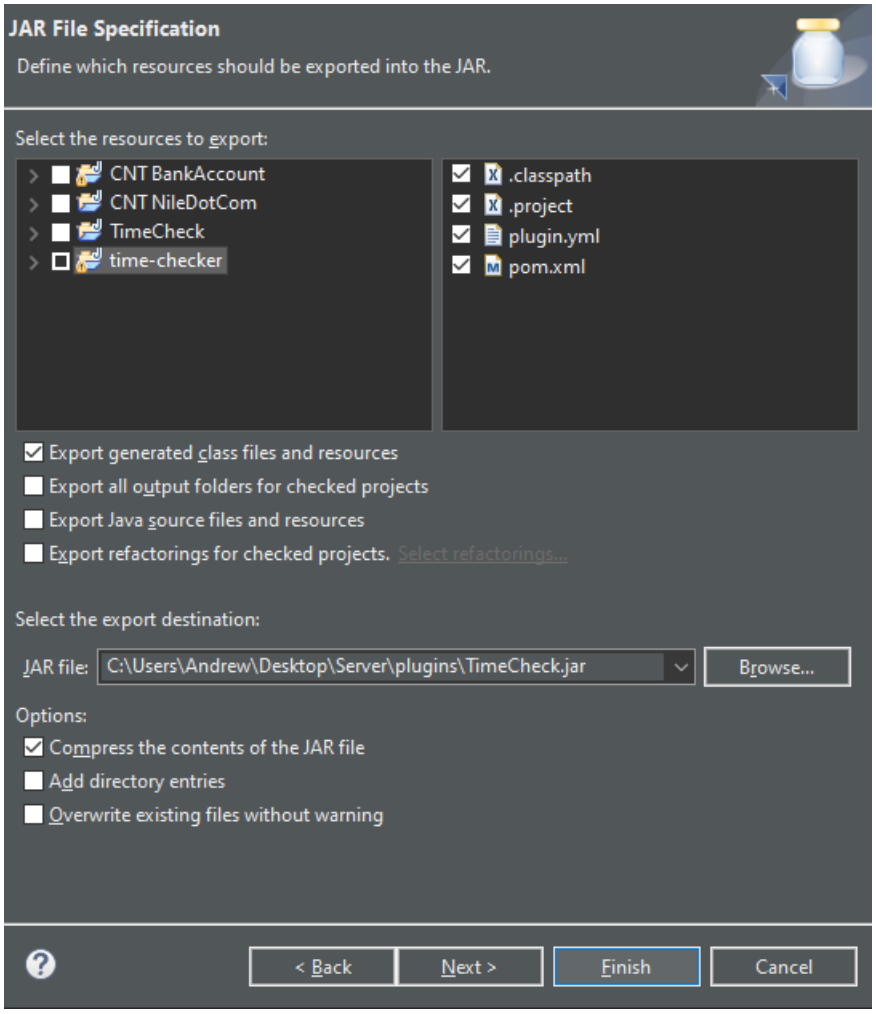

Figure E.4: Final screen for exporting a JAR file.

E.3 MongoDB Compass

MongoDB Compass provides a GUI for users to easily manage and maintain their database. The GUI allows its users to visualize, modify, debug, and optimize their database. This is particularly useful during development to test whether the Node.js server is properly interacting with the MongoDB documents.

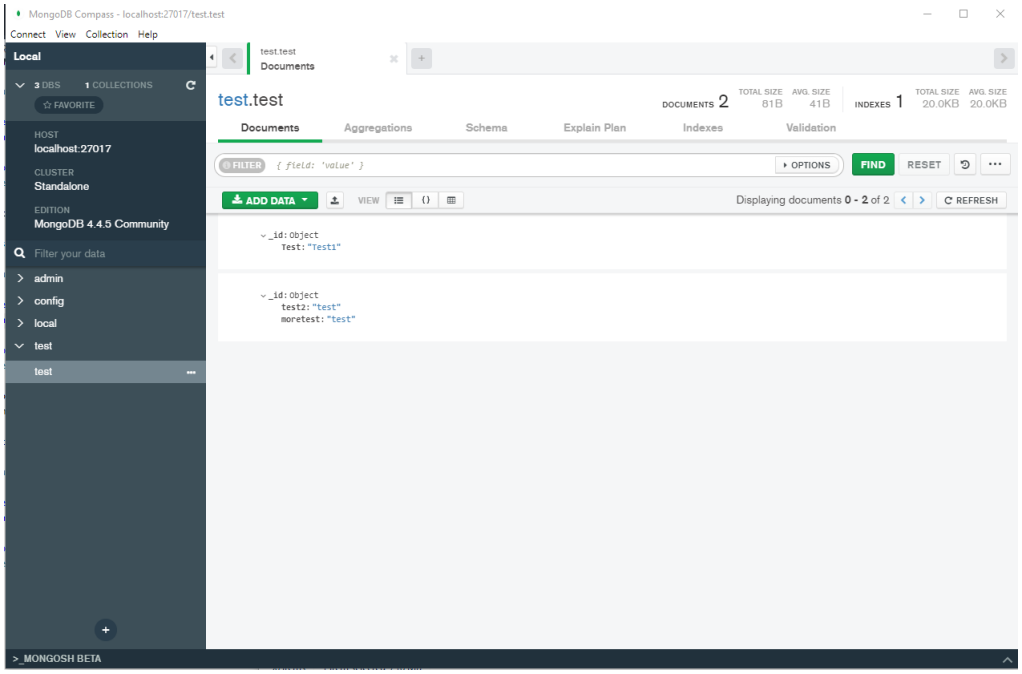

Figure E.5: MongoDB Compass

MongoDB Compass is not the only way to use MongoDB, but it is the most user friendly. The alternative would be to use a terminal to connect to the database and do all of the database management via the command line, which some users can find daunting. Using a GUI makes it much more welcoming and easy to use right out of the box. An example of the MongoDB Compass web GUI is shown above in figure E.5.

Appendix F

Other Tools Used

F.1 Version Control

Version control is a must have in any collaborative software project. Version control is done by tracking each individual change when a contributor makes changes to the project. Since all changes are committed to one location, everyone will always have access to the latest version of the project, allowing for everyone on the team to work at the same time, even on the same file. On top of that, version control has built-in system that helps its users sort out any conflicts that may occur from two users trying to edit the file. The other useful feature of version control is its ability to save versions of the project. This allows for easy access to previous versions of the project. The major advantage of this is that it enables the ability to revert a change back to a previous iteration of the project if a mistake is made.

There are many version controls such as Github, Concurrent Versions System (CVS), and Mercurial. The one that our group settled on is Github. The alternatives doesn't provide any major additional features that Github doesn't already give us.

F.1.1 Github

Our Github repository will likely be similar to that of the current HeapCraft repository. The real major differences will be when we add the web interface, as that is adding an entirely new layer of abstraction to the application, thus will add much more code to the project and expand the repository, both in size and language variation.

F.2 Task Management

Organization is a critical aspect in the development of this project for it to succeed. Staying organized will allow for our team to work more efficiently and gives a picture of what needs to be started, worked on, and whats already finished.

The two main options were Trello and Notion. We decided to use Notion because it did everything Trello did, but with more features.

F.2.1 Notion

Like Trello, Notion allows for the making of lists, cards, and the making of due dates to said cards. Additionally, Notion allows for the uploading of important documents and files so that it is in a central location that everyone will always have access to. A screenshot of a Notion page is displayed in figure F.1.

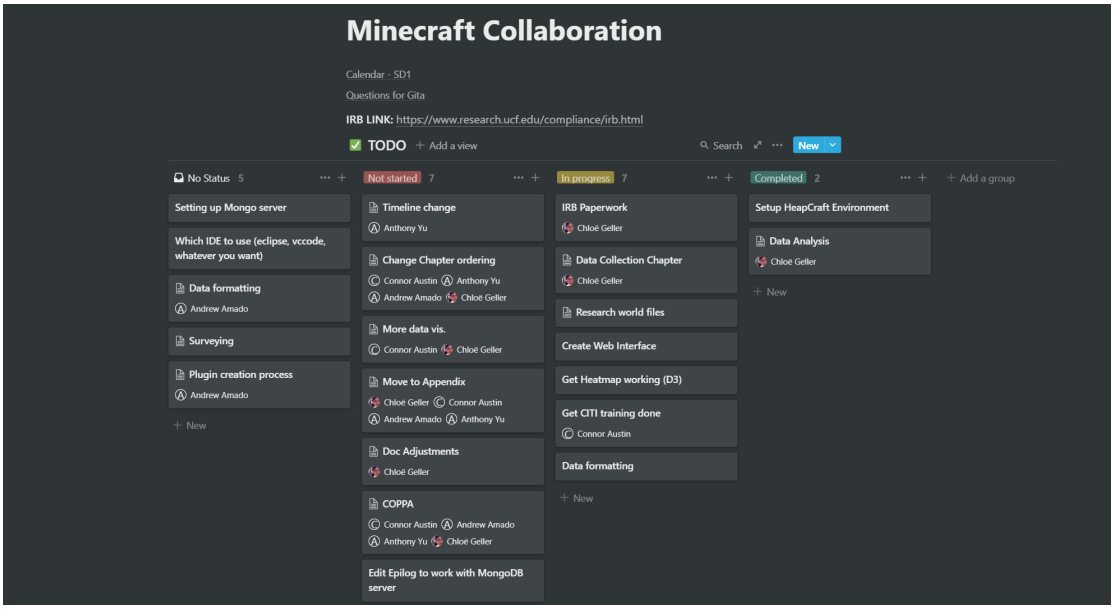

Figure F.1: Notion Screenshot

F.2.2 Google Drive

Our group will be using google drive to share any large non-code files that are pertinent to the project. Google Drive makes uploading, storing, and sharing files easy. With each file, we can either create a clickable link to download, or we can create a shared folder among the entire group where we can all edit the contents, which is what we have done.

Ideally, we would use Google Drive in association with Notion. One example of this association is the sharing of our recorded meetings. Whenever we have a meeting with Professor Sukthankar, we record it and upload the video to Google Drive. Once the video has been uploaded and processed, we then embed the video on one of our Notion pages, so that the video is easily accessible to all team members and we can refer to those meeting videos to retrieve any information we might have missed.

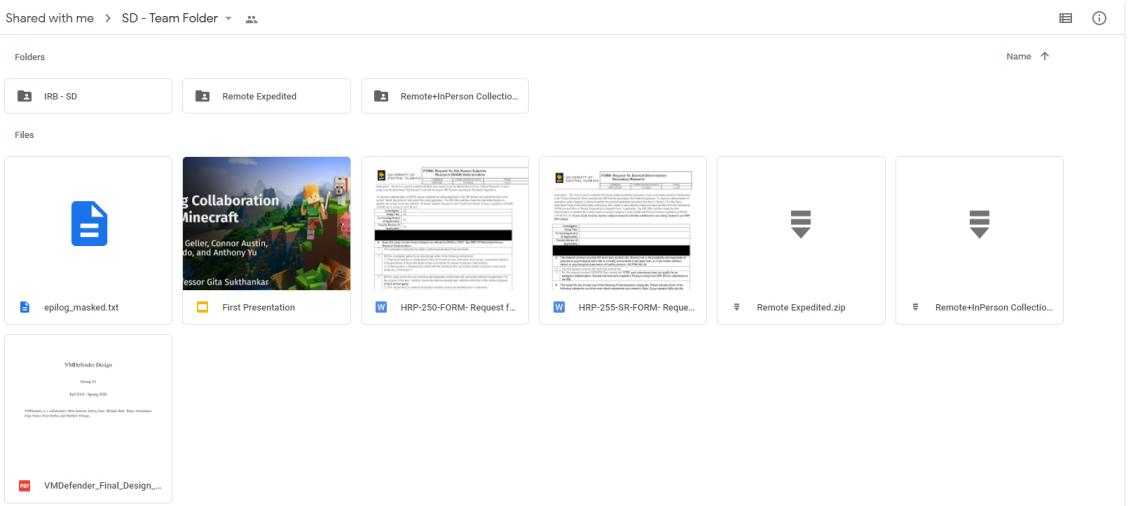

Figure F.2: Google Drive Screenshot

F.2.3 Overleaf

Overleaf is one of our most important tools, in fact, it is the very tool that we are using to write this paper. Overleaf is a collaborative LaTeX editor, where all members of a team can work on the same document simultaneously. Overleaf is an in-browser editor so no additional software needs to be installed and documents are shareable via hyperlink.

LaTeX is a markup language that not all of us were familiar with at first, but it has proven to be quite simple, and is making the process of writing this document much easier compared to other popular word processors such as Microsoft Word or Google Docs. Using LaTeX allows us to auto-generate our table of contents, add sections with one line of code, and generate appendices and a bibliography as shown at the end of this document, making the writing process much easier and efficient.

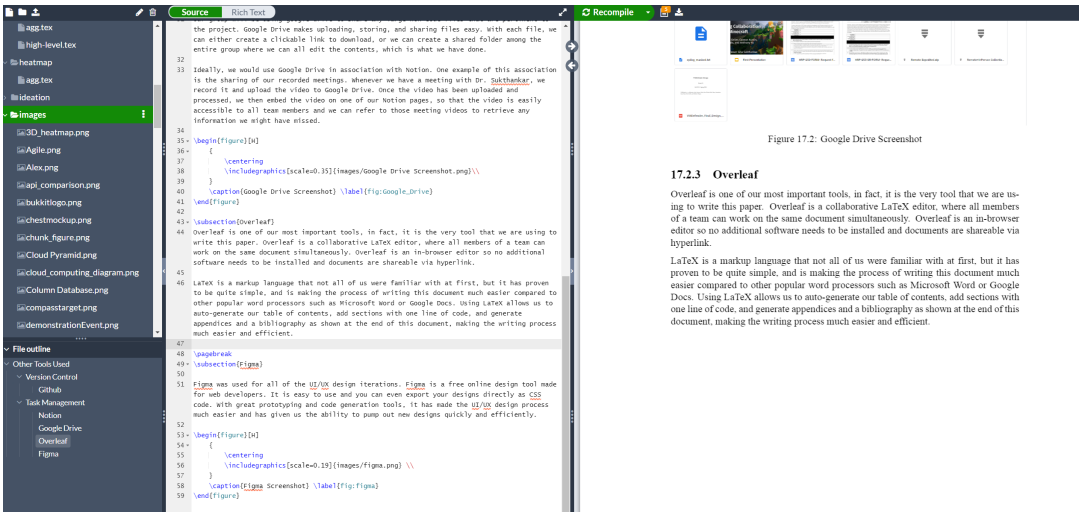

Figure F.3: Overleaf Screenshot

F.2.4 Figma

Figma was used for all of the UI/UX design iterations. Figma is a free online design tool made for web developers. It is easy to use and you can even export your designs directly as CSS code. With great prototyping and code generation tools, it has made the UI/UX design process much easier and has given us the ability to pump out new designs quickly and efficiently.

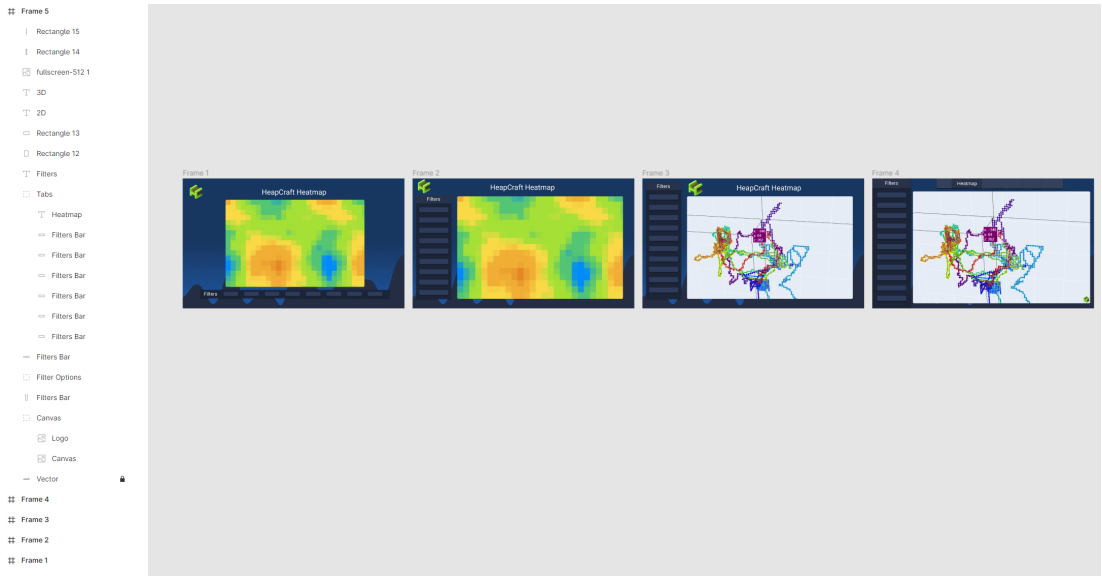

Figure F.4: Figma Screenshot

As you can see in figure F.4 above, you can easily have multiple iterations of a UI design side by side, being able to compare one iteration to another with ease. Each page design is stored in a Frame, which acts as a container for everything within the page. When you are ready to start a new iteration of the design, you can simply copy the entire frame and make any changes you'd like. However, Frames do not have to be used just for design iterations, but they are also commonly used to layout different pages of an entire website and how they interact with each other.

Appendix G

Software Development Methodology

G.1 Agile Methodology

The Agile Methodology is a methodology that takes a more team-based approach that emphasizes on rapid deployment of functional applications that focus on customer satisfaction. Agile does this by having a defined period of time, typically one to two weeks, called a sprint. At the beginning of each sprint, a list of deliverables are prioritized by the customer's input. From that list, the team chooses based on priority and specialty of each member. At the end of each sprint, the team and the customer review and evaluate the work for future sprints. Due to this reviewing of work, members of the team can readjust the amount of work they take from the list, meeting goals. [40]

The main benefit of Agile is its ability to deliver functional solutions to the customer. This reduces the risk of failing to produce a project. Due to sprints, Agile also has more flexibility to changes and helps promote teamwork and communication.

The downside of Agile is that Agile requires a large amount of customer involvement, which some customers may be comfortable with. On top of that, Agile assumes that each team member is dedicated to the success of the project. This is because sometimes timed iterations may not be enough to accommodate all deliverables.

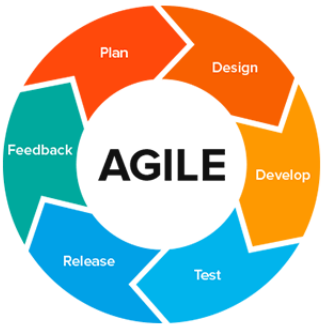

Figure G.1: Agile Diagram [41]

G.2 Waterfall Methodology

The Waterfall Methodology is a software methodology that takes a sequential approach to the software development life cycle. The sequential approach means that each phase in SDLC must be completed before the next phase can begin and that there can be no overlapping of phases. The SDLC distinct phases are requirements gathering, analysis and design, coding and unit testing, system and user acceptance testing, and deployment. [40]

The benefit of operating like this is that all planning and designing is done before the start of the project. This leads to a better system design as a whole because it is all planned beforehand, defined team roles, defined scope of work, and clear measurements of progress.

For the same reason that the Waterfall Methodology has its advantages, it leads to a few disadvantages. One of the biggest things is that test is only done during the later phases of the project, resulting in high risk and uncertainty to the success of the project. In addition to that, due to its rigid structure, Waterfall Methodology has a hard time allowing for any type of change.

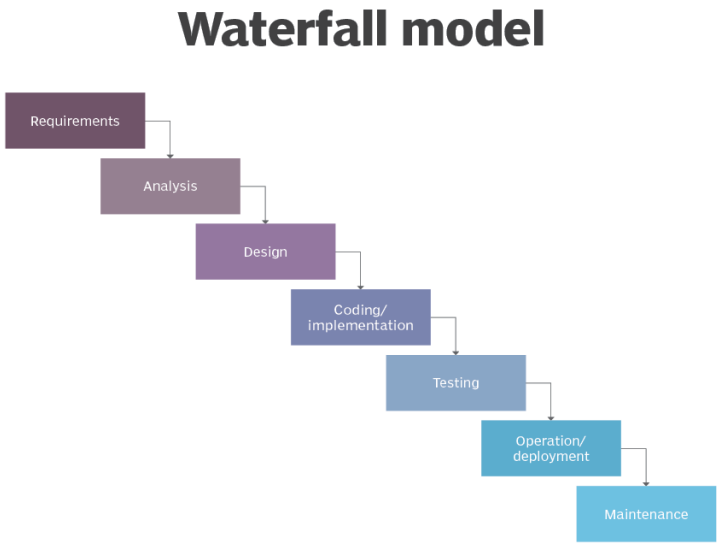

Figure G.2: Waterfall Model Diagram [42]

G.3 Conclusion

Due to our project being a research project, following a Waterfall Methodology is difficult because of its requirement of having all the planning done at the beginning. We will be following an Agile Methodology because it works well with our project. The biggest benefit of Agile is its ability to deliver functional solutions at the end of each sprint. As we are creating tools to visualize collaboration, it is important that we make sure our tools work before we continue to improve and expand upon them.