Vrije Universiteit Amsterdam

Bachelor Thesis

Analysing Player Behaviour in Minecraft-Like Games

Author: Kevin Denneman (S2558785)

1st supervisor: ir. Jesse Donkervliet
2nd reader: Prof. dr. ir. Alexandru Iosup

August 21, 2020

Abstract

There are 2.5 billion people worldwide playing video games, together generating $152.1 billion in revenue for the gaming industry. Some games allow people to play with one another through the Internet. One of the most popular games is Minecraft, a game with over 126 million monthly players. In Minecraft, players can play together in online Modifiable Virtual Environments (MVE) and interact with each other and the world. Although players are able to connect and play together, when trying to connect a large number of players, there are scalability issues for the servers these MVEs are hosted on. In order to tackle the scalability issues that arise when trying to connect a large amount of players, we require more testing and measuring tools. To develop a good benchmarking tool, intensive workloads are required. Workloads can be made by having a lot of players connect and play on the same server, which is every expensive and hard to organise. Another way of creating a workload is by using realistically emulated players, which is much cheaper, however, this requires a behavioural model. This thesis will show a way to acquire a data on player behaviour, and presents a player behaviour model based on analysis of this data suitable for emulation in MVEs.

Contents

1 Introduction

The video game industry is large. 2.5 billion people worldwide play video games and the industry generated a revenue of $152.1 billion in 2019 [9]. Games have a positive impact on society, with, for example, its uses in education [12]. One of the most popular games is Minecraft, as show in their sales numbers and player count: Minecraft has sold over 200 million copies worldwide, and has over 126 million monthly players [13]. Minecraft is a game that has a positive impact on society. For example, Minecraft's Education Edition [1] has uses for teaching, and during the COVID-19 outbreak of 2020 it allowed people to attend the graduation ceremony in an online Minecraft world [5]. This is possible because of Minecraft's modifiable worlds and online servers allowing multiple players to play together.

There are games that are build around endless procedurally generated worlds, Minecraft is one of these games. The term used to describe these Minecraft-like games is Modifiable Virtual Environment (MVE). These MVEs can be played by a single player, but they can also be played with others on online servers. When playing together MVEs are typically self-hosted or deployed as a game as a service (GaaS). When hosted on online servers, it allows multiple players to connect and interact with each other in the same world. These online worlds, however, do not support large numbers of players, each server instance does not scale past 200 to 300 players, even in favorable conditions [4]. With a player base as large as Minecraft, limitations on the scalability of these online systems is an interesting and large important challenge.

1.1 Problem Statement

To improve the scalability of these systems, we first need to be able to measure their scalability. To properly measure and test the scalability of MVEs through benchmarking, we require intensive workloads. In order to obtain a suitable workload we need large amounts of players to simultaneously connect and play on the servers while monitoring the server outputs. To organise having many players test is a very difficult and an expensive task. A proposed solution to this challenge is the use of *emulated players*. An emulated player is an artificial player which is able to be deployed on servers, using multiples of these players allows us simulate a large workload. However to gain confidence in the results, these emulated players should behave as similarly to real players, unfortunately, research player behaviour in Minecraft-like games is limited.

1.2 Research Questions

This thesis aims to improve the state-of-the-art in benchmarking Minecraft-like games and to generate new knowledge about player behaviour in these games. It focuses on learning more about player behaviour in Minecraft-like games. This is done by addressing the following research questions:

RQ1 How to effectively collect Minecraft player behaviour data?

Collecting data to create a dataset will provide a solid base for further experimentation. To effectively collect the data, we have to observe players while they are playing the game. It is challenging to collect data on player behaviour in a non-intrusive manner, without any server-side modifications. As we have to observe a player up close, while they are playing the game, without impacting their behaviour and without hosting our own server. Furthermore, we make data collection more effective, because this simplifies deployment and data-gathering in general, since it requires no actions from the server host.

RQ2 How to model player behaviour in Minecraft?

From our dataset we can extract important metrics. The challenge is in identifying and selecting the right metrics. These metrics can allow us to understand what drives and motivates players in a sandbox environment, and perhaps draw conclusions about their intentions. Furthermore, game states or player behavioural models can be observed and perhaps recreated. Possible observations we could look out for are: time spent by players of the game, central positions on the server and preferred activities. Since Minecraft is an unguided, open-world, open-ended experience, analysing player behaviour can be difficult. Therefore, observing and imitating players can be useful.

We hypothesise that by answering these questions, it will allow us to improve the state-of-the-art in benchmarking Minecraft-like games.

1.3 Main Contribution

The main contributions of this thesis are:

1. The design of a system for efficient data collection in MVEs. The data is collected by observing and logging the actions of players in online servers doing various activities.

2. A Minecraft player behaviour model, based on an analysis of collected player data.

Figure 1: Minecraft.

2 Background

In this section, we provide additional background information on topics relevant to this thesis. First, we describe Minecraft and Yardstick in Section 2.1 and Section 2.2 respectively. Second, we discuss related work in Section 2.3.

2.1 Minecraft

Minecraft is an incredibly popular game. Because the game is open-ended and provides a lot of activities the players can choose from, it allows the players to be creative in choosing their goals. The world of Minecraft is *voxel based*, meaning it consists of a three-dimensional grid of blocks. An important feature of the game is that the player can place and break these blocks, and interact with the different entities in the Minecraft world. Important activities in Minecraft include:

1. Mining, gathering and crafting. Resources can be mined, gathered or combined to create new items or blocks.
2. Building, from simple houses to complex logic circuits which could be used to for example automate some resource gathering.
3. Exploring a procedural generated three-dimensional world without borders.
4. A combat system where you can fight different enemies and even boss monsters.
5. Playing custom game modes, created and shared by players from the community. These games are highly popular, and come in many varieties. From completing mazes, to a single-player story-driven adventure, to a game where you can race other players.

Although these are important activities, the game offers many more ways to engage with the world. Minecraft offers different game modes that accommodate these activities: creative, survival, adventure or hardcore. In Minecraft, players have the opportunity to play individually, or share an instance between multiple players. These instances are of varying sizes and focus on different activities. For example, worlds vary from shared between a handful of friends, focused on mining and building, to hub-worlds, which connect hundreds of online players to a variety of custom game modes. These smaller instances might

be hosted by an individual player on a desktop computer, whereas the larger servers are typically hosted by a community, using dedicated infra structures. In small and large instances, the players are almost always interacting with the world in some way. Interactions require communication between the server and the Minecraft clients. This communication protocol is documented and publicly available on the Internet [2].

2.2 Yardstick

The Yardstick benchmark is designed for evaluating and comparing the performance of Minecraft-like games. It consists of a workload of user defined behaviour scripts that connect to Minecraft-like game servers. When connected, Yardstick can consequently monitor and log the results.

2.3 Related work

Research on player behaviour in Minecraft-like games is limited, although there have been studies in player behaviour for Massively Multiplayer Online Games (MMOGs) like World of Warcraft [10] and Second Life [11]. But these games have a static game environment. Minecraft, however, has a dynamic and ever growing world that the players can interact with and modify. This dynamic world brings with it a vast amount of options a player has and thus creating more detailed and different player behaviour profiles. This dynamic behaviour means regular methods of player profile creation such as SAMOVAR [11] do not apply. This means a tailored approach to create a player behaviour model is required.

Minecraft player behaviour has been studied, using the HeapCraft framework [7]. HeapCraft observes which network packets a server sends and receives in order to track the activities of players. They acquire data through a server side plugin. The authors of this project categorised player behaviour a priori in exploring, mining, building and fighting, and use data analysis to determine the frequencies of the behaviours [8].

MineRL [6] is an initiative that aims to solve Minecraft using Machine Learning. They provide a large dataset that consists of observations on four activities within Minecraft: navigation through the world, collecting as much wood as possible, collecting some more complex items and having players survive for as long as possible. To collect their data, players are required to modify their Minecraft client such that it can send collected data to the dataset. This dataset is aimed towards developing a Minecraft AI, similar to the goal of this thesis. The main difference is that MineRL is focused on creating a dataset for the development of AI through reinforced learning. They have the AI study frames to decide on the best course of action. In this thesis, however, we are more interested in the server scalability and trying to obtain a dataset such that the goal is not to solve the game, but have our AI behave similar to human players.

3 Design of a Player-Behaviour Collection System for MVEs

This section describes the design of the player-behaviour collection system, which aims to gather data by monitoring players in MVEs. Section 3.1 describes the functional requirements of such a system. Section 3.2 presents our design and gives a general design overview.

3.1 Requirements

The following requirements allow the system to collect data on player behaviour on a larger scale.

1. The system is to monitor players and collect data from the players' activities. Therefore, the system is required to track a player throughout the world.
2. To collect data not influenced by the actions of the system, the system must not be intrusive to players' behaviour.
3. Data collection must be handled by the system, with minimal effort needed from server operators or players.
4. The system must uphold player privacy.

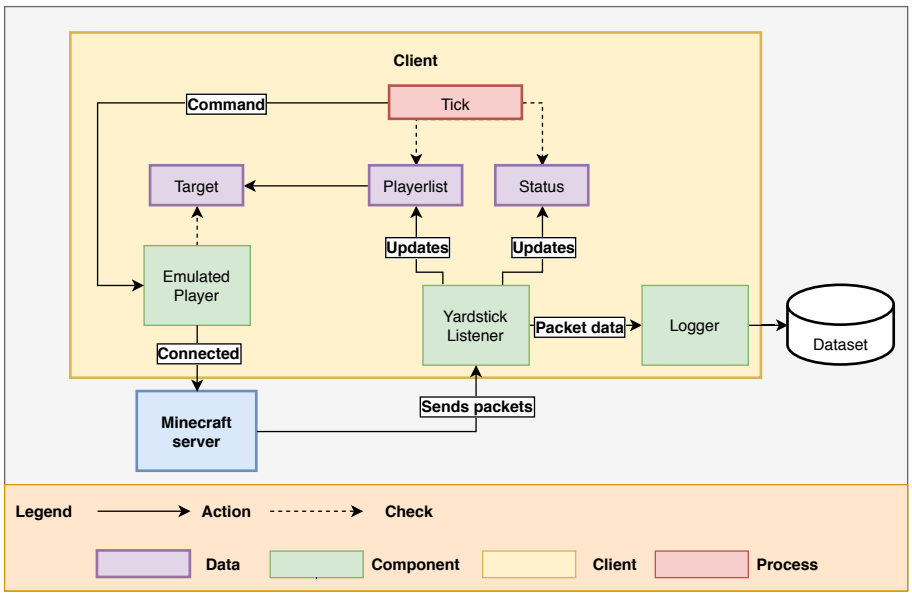

Figure 2: System overview of the data gathering using an emulated player connected to a server.

3.2 Design Overview

This section presents the design. Our system extends Yardstick [3]. In Figure 2 presents an overview of the system. The client operates using three different components.

1. An emulated player which is connected to a Minecraft server to perform the player tracking.

2. The Yardstick listener is where the data gathering takes place, it intercepts the packets received from the server.

3. A logger, to provide output.

An emulated player is connected to a Minecraft server, the server will send packets to communicate with our emulated player. The packets will be received by the listener and are passed on to the Logger, and where appropriate, it uses the packets received to update internal data such as a playerlist and the status of the emulated player. The tick process is the heart of our player tracking model. Every fifty milliseconds it triggers a check for updates on the emulated player's status and surroundings, and periodically sends commands our emulated player can execute, such as telling the emulated player to teleport to the target.

3.2.1 Player Tracking

The first and second requirements involve player tracking. To track player movement, the system connects an emulated player (bot) to a server. This bot uses Minecraft's built-in teleport command. The bot follows the player by teleporting to a monitored player in frequent intervals. Having the bot track players by foot seemed inefficient and intrusive for the monitored players, which would conflict with the second requirement. If the bot was trying to find players on the other side of the world it would take a a long time to get there just walking while simultaneously avoiding monsters and possibly having to do it all over again in the case of a player death. The bot would also not hinder the monitored players by teleporting a little distance away from the monitored player. The bot also attempts to be in "spectator" mode, where the body is completely invisible to other players and it is not able to interact with other entities in the world. In order to not forget about the multiplayer aspect of Minecraft the bot is also periodically teleporting and checking on other players currently active on the server, to obtain some information about their whereabouts. However, choosing to use teleportation as the main way of tracking players introduces multiple limitations: the server on which the bot is deployed needs to have a teleport or spectator command enabled, or the bot needs to be given operator powers for the server. With these operator powers the bot can bypass the server's player restrictions to use teleportation and enable spectator mode.

3.2.2 Data Collection

The data the system collects are the client-bound packets the system receives from the server. There are a total of 93 different client-bound Minecraft packets that can be received. An overview of these packets can be found in the Minecraft documentation [2].

The system logs almost all of the incoming packets. The few exceptions are listed below.

1. `ServerChatPacket`. To meet requirement 5, and not intrude player privacy, the system does not track players' messages.

2. `ServerEntityHeadLookPacket`. Because this packet is sent whenever a nearby entity changes its head position. After testing, it quickly became apparent that this packet was received very frequently and would significantly increase the size of the dataset.

3. `ServerUpdateLightPacket`. This packet would update the light levels in the area. The size of this packet is large compared to the other packets. Similar to the `ServerEntityHeadLookPacket`, it would significantly increase the size of the dataset.

4. `ServerEntityPositionPacket`. This is the main packet that allows us to track a player's movement, however the format of this packet is such that it only contains info on what direction a step was being made. To make this packet easier to analyse the players new coordinates are added to this packet when it is being logged.

3.2.3 Analysis of System Overhead

Currently the system's tracking adds one player to a server. Having an extra player, our emulated player, in a server would add some overhead to the server load. This overhead is smaller than the amount of data a real player would contribute. Our system does not engage in many activities and therefore will require a minimal amount of data from the server.

3.3 Protocol-Specific Efficiency Improvements

This section will highlight the features of the system that were implemented to improve the efficiency of data collection.

To prevent long periods of inactivity that occur after an in-game death of the bot, the system monitors the bot's status. An in-game death causes the bot to be prompted with a game-over screen, making the bot unable to collect data. When an in-game death is detected, the bot will send a `ClientRequestPacket` to the server, requesting a respawn. After being respawned the bot is able to continue collecting data.

To prevent Away From Keyboard (AFK) related disconnects from the server when there are currently no players to monitor, the bot would periodically send a `ClientPlayerSwingArmPacket`. This makes the bot play an arm-swinging animation and causes the server to detect activity from the bot.

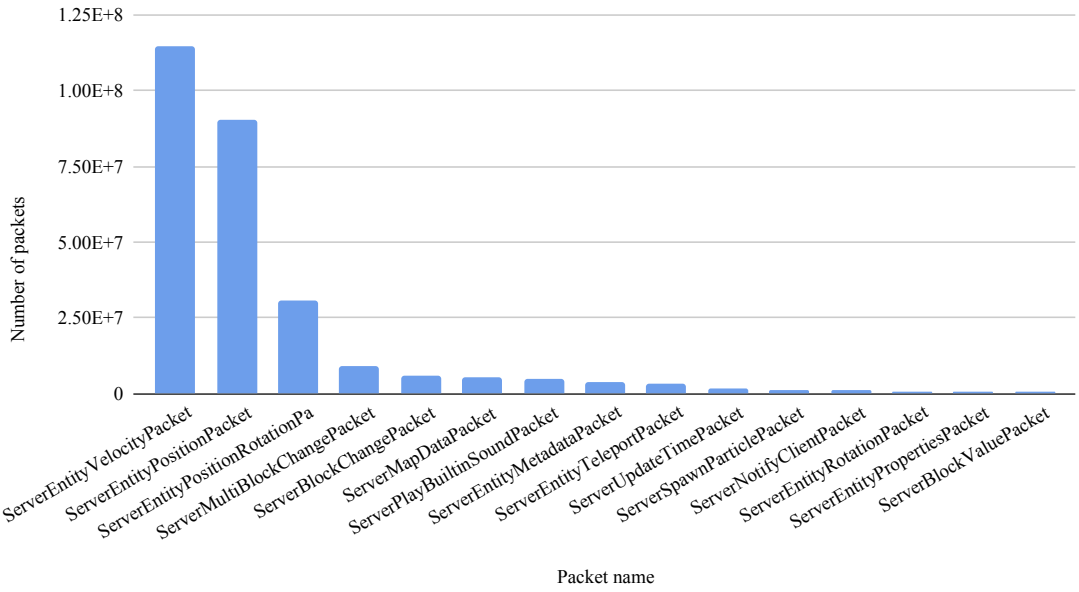

Figure 3: Number of recorded packets by packet name.

4 Results and Analysis

In this section, we give an overview of the gathered data and analyse the data to create player behaviour models suitable for MVEs in 3D space.

4.1 Dataset

Our experiment collected a dataset of packets that were obtained by tracking players on a small private community server. The server's game mode is set to *survival mode*. Survival mode means that players have to gather resources, engage in combat to defend themselves and stay nourished by regularly consuming food items. In the data collection period of 2 weeks, a total of 127 sessions were logged, with 66 hours of active gameplay. Together, these sessions amounted to a total of over 278 million packets. The logged packets were stored in `txt` format.

Figure 3 shows an overview of the top 16 packet types that were received. The gathered packets mostly consisted of `ServerEntityVelocityPacket`, with over 100 million packets and `ServerEntityPositionPacket` which were received more than 90 million times. A `ServerEntityVelocityPacket` is received whenever an entity receives a burst of speed, through for example when being hit by another player, jumping during spring, or using movement abilities such as an *Elytra*, which is an equipment item that allows for flight.

Packet	Description	Assumption
`ServerBlockChangePacket`	Update a block in render distance.	We can observe block changes surrounding the player.
`ServerEntityPositionPacket`	Updates an entity's position.	We can track movement of entities.
`ServerEntityMetadataPacket`	Updates an entity's metadata.	We can see what happens to an entity. Examples of metadata properties are: pose, health.
`ServerEntityCollectItemPacket`	Triggered on item pick up.	We can see the resources gathered by entities.

Table 1: Important packet overview.

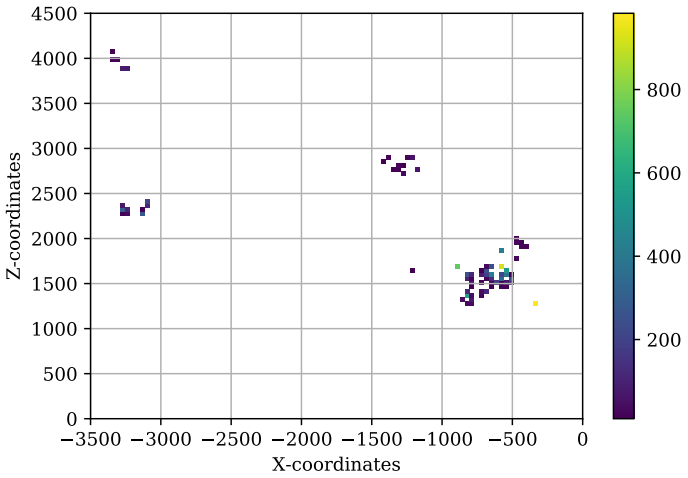

Figure 4: Heat map of `ServerBlockChangePacket`.

4.2 Analysis of the Dataset

We further investigate the obtained information and look at certain metrics, similar to the player activities discussed in Section 2.1, to analyse real player behaviour. An overview with assumptions of the important packets we will discuss is shown in Table 1.

4.2.1 Materials

In total there were 5.5 million block update packets received while there were players active on the server. A heatmap of all gathered `ServerBlockChangePacket` is shown in Figure 4. We can see that there are a few small clusters around the world with a larger cluster in the bottom right. This suggests that this server had player hubs, where most of the block updates were received.

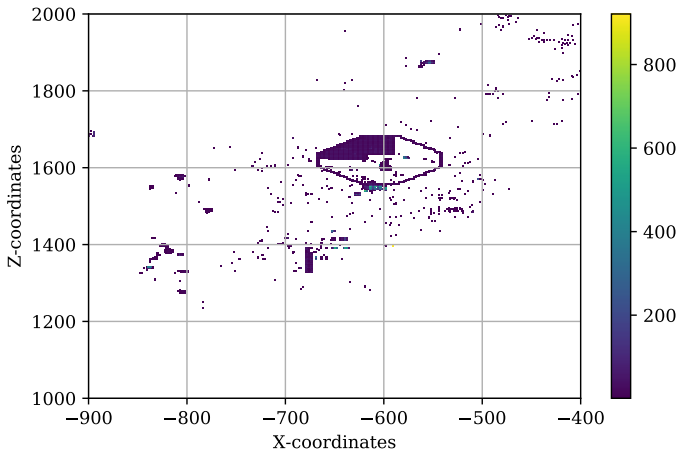

Figure 5: Heat map of cluster.

In Figure 5 a close up of the bottom right cluster is shown. We see a clear circular pattern appear. This pattern has a regular shape, this regularity suggests it is man made. This heat map also shows us that the high amount of packet updates mostly come from small block clusters.

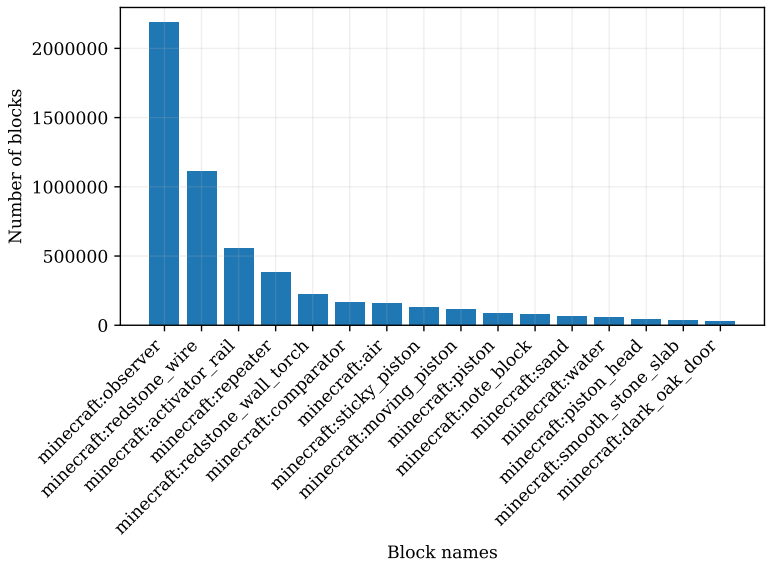

Figure 6: 16 Most commonly observed `ServerBlockChangePacket`.

Figure 6 shows the 16 most common `ServerBlockChangePacket`s that were received. The most common block is `minecraft:observer`, this block listens to state changes of the block in front of it and sends out a *redstone pulse* when a change is detected. In Minecraft redstone functions as an electrical current, allowing players to create redstone circuits. From the observed blocks we see a trend, most of the blocks interact with

`minecraft:redstone_wire`, the second most common observation. Redstone is a resource mainly used in the later stages of the game where players are already further progressed and is used to automate the gathering of resources, one of the player activities discussed in Section 2.1.

The most observed farm-able resource is `minecraft:wheat` at 6103 observations. Wheat is a resource that has to be planted and will grow over time. Wheat is a crafting component of bread and the growth of wheat is easily automated, making it a popular food item of choice which likely has an automated farm set up on this server.

We now look at the distribution of the `ServerEntityCollectPacket`, which is the packet that is purely for the server to communicate that an entity triggers the picking-up-an-item-from-the-ground animation. However, after filtering out all the packets that were received by non-player entities, we can use it to determine how likely a player is to engage in the gathering of resources as all activities that involve a resource trigger a `ServerEntityCollectPacket` from monster drops to mined blocks. Figure 8 shows a curve similar to an exponential distribution. However, it does not closely match any of the distributions. The mean number of items collected in a session is 882, and the standard deviation is 1029.

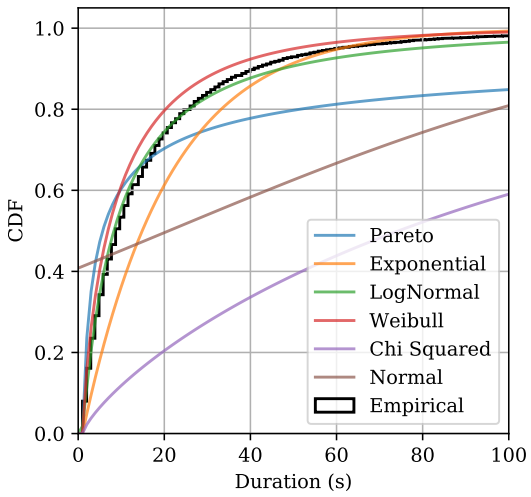

Figure 7: Cumulative distribution of the duration of a walk.

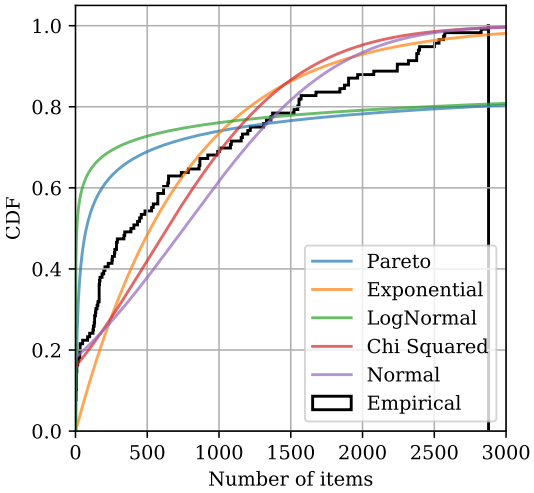

Figure 8: Cumulative distribution of `ServerEntityCollectPacket`

4.2.2 Movement

The gathered data is split up in 'walks'. A walk consists of all observerd `ServerEntityPositionPacket`s of a player after an initial position packet is received, which indicates that the player started moving, up until the player halts. From this we can discern an initial starting location and then we will track for how long the player keeps sending position updates. In between these walks another metric can be found, namely a pause duration.

To estimate the movement characteristics of our model, we fitted probability distributions to our data. Figure 7 shows these cumulative distributions of the duration of a player's walk. The average duration of these walks was 21 seconds, the standard deviation was 90 seconds. The best fit for this distribution is a log-normal distribution.

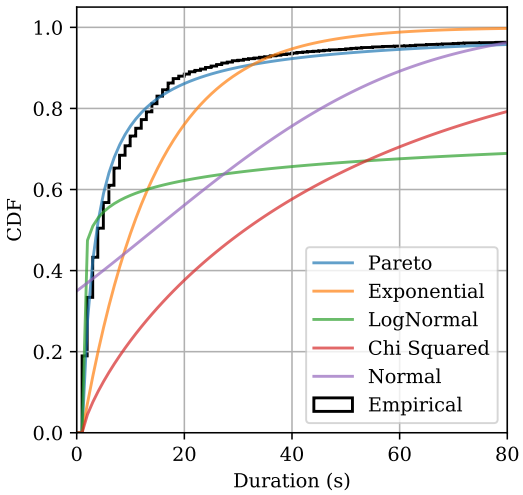

Figure 9: Cumulative distribution of the distance of a walk.

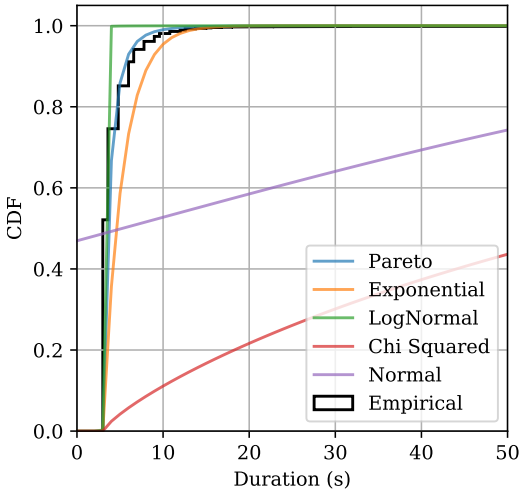

Figure 10: Cumulative distribution of the pause duration between walks.

The distance was measured from starting coordinates to end coordinates, calculating the Euclidean distance between the two coordinates. The average distance a player walked on these runs was 13 blocks, the standard deviation was 35 blocks. Since Minecraft has a 3D environment, we take into account vertical movement. When adding the vertical axis, the average changes to 13.9 blocks and the standard deviation becomes 36 blocks. Figure 9 shows these distributions, of which the pareto distribution best fits the data.

As shown in Figure 10, a pareto distribution also fits the pause duration distribution. The average pause duration was 5 seconds and has a standard deviation of 68 seconds. The steep CDF curve suggests that the probability of a longer pause between walks is low, indicating that human players are often move around.

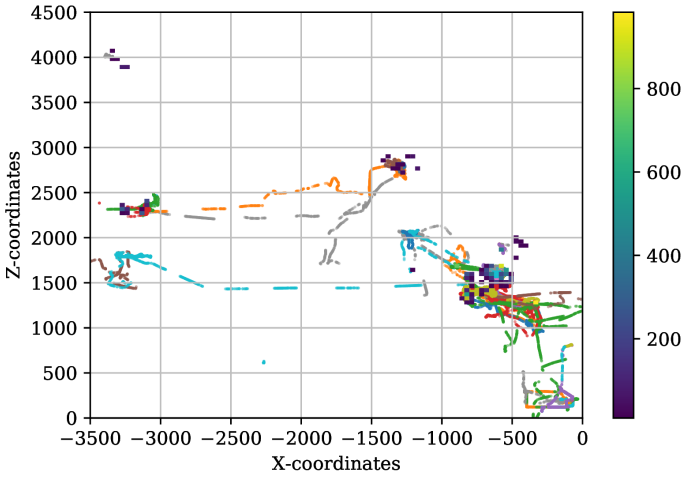

Figure 11: Figure 4 overlapped with `ServerEntityPositionPacket`.

Figure 11 shows the received `ServerEntityPositionPacket` combined with Figure 4. From this we can indeed conclude that most activity took place around the [-500, 1500] coordinates. With a large amount of activity gathered in this one central hub and the lower bursts of movement that we could see from the distributions, it suggests that in this server the players have most of their needs nearby.

The line breaks we can see in the central part of Figure 11 together with the large number of `ServerEntityVelocityPacket` we saw in Figure 3, suggests that the players use some sort of movement abilities, such as the *Elytra*, when traveling larger distances.

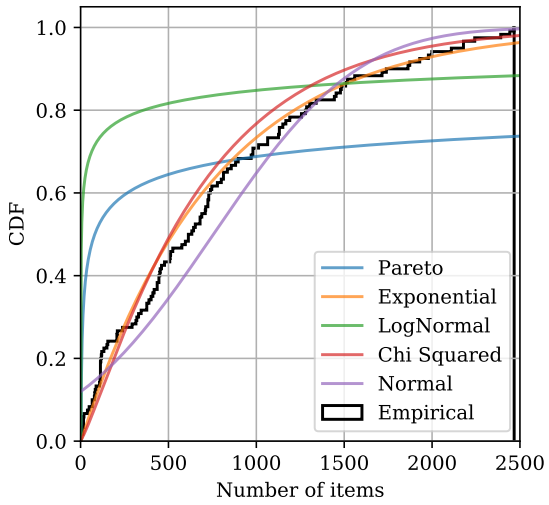

Figure 12: Cumulative distribution of `ServerEntityMetadataPacket`.

4.2.3 Combat

Combat is an activity that requires specific analysis. A `ServerEntityMetadataPacket` is triggered by combat events. However, this packet is also triggered by updates to various status effects from players which do not necessarily require combat, such as being in a crouched state and the packet does not discern who inflicted the change in status effect to the entity in the case of an attack. We did, however, fit a distribution, as seen in Figure 12 to to give an intuition of how likely it is for players to go through different status changing effects in an average session. With an average number of packets of 887, and standard deviation of 834, an exponential distribution seems to fit this data. Other metrics for combat would require us to deeply connect different packets, such as an `ServerEntityAnimationPacket`, while making sure the same entity is holding a weapon in a `ServerEntityEquipmentPacket` and then looking for `ServerEntityMetadataPacket` changes on creatures nearby the player. Such analysis proved complex, and is further discussed in Section 5.

4.3 Player Behaviour Model

From the observed dataset we were able to identify and analyse certain characteristics of players on the server. We will now model these characteristics.

First we will have to identify and model an environment for our emulated players. In the dataset, we saw multiple player hubs, one hub in particular seemed very popular among the players. Therefore, we add a likelihood of a player to want to be, and stay, in a certain player hub based on observations of the number of people that visited a hub.

Since we want our emulated player to behave like real players, we also have to add some incentives for social interactions apart from intrinsic motivation. For example, periodically,

events are created that incentivise player gatherings. In accordance with our observed data, we will create one central player hub that incentivises players to stay in that area. Other hubs will be given a baseline likelihood so players will be able to escape the popular hub. In Minecraft a simple central popular area to pick would be the spawning point of the server, as all new players will start off here.

In player hubs, we observed a high level of automation of resource gathering. To facilitate this in our model, we will recreate this by placing redstone contraptions in these areas to simulate this automation and mark locations within a select amount of hubs where players can collect food items.

To model movement within this environment, players are given the option to move using earlier discussed 'walks' to get to, and move to, different hubs around the world. This process will use the following steps:

1. Players will be assigned a subset of hubs they are able to visit, to prevent all players from following the exact same routes. All players will be assigned the central hub, the smaller hubs will be assigned randomly among to population. To increase the likelihood of a player moving out of an incredibly popular hub, players will also be given an *explorer* trait. This trait makes it possible for a player to escape a hub despite it being highly disincentivised by the hub.

2. Each player will select a target they will move to. This target will be either a hub or a food gathering location. The target hub is selected according to hub popularity, and the player's explorer trait. It is possible for players to select their current hub, mimicking constant movement in a hub. A nearby food gathering location is chosen with priority, if the player moved for a certain duration and gets hungry.

3. Players will be given a task to execute a walk. The distance of these walks are determined according to the distribution of distance. A location of this distance away and in the general direction of the player's target is selected. The player will then proceed to move to this location for a duration decided by the duration distribution.

4. After executing their walk, players will pause for a duration. This duration is taken from the pause distribution.

5. Tasks will be repeated until players have reached their target, after which players will select a new target.

To execute their movement, the emulated players will require a path finding algorithm compatible with a 3D environment. The world has many obstacles such an algorithm has to take into account.

With current analysis, it is hard to model a realistic combat system. However, we can add an approximation to the model by forcing players to trigger status updates. This is done by adding behaviour types to players, based on the distribution of `ServerEntityMetaDataPacket`s. Players that receive a behaviour with a high likelihood of engaging combat would then generate more status updates by alternating between sprinting and sneaking during their movement tasks.

5 Discussion

Our results give new insights to player behaviour in Minecraft and allows the development of a more realistic player behaviour model. However, there are still some limitations with the current approach. This section aims to discuss these limitations.

The question of whether our analysis can create actual realistic player behaviour models is still unanswered, as we did not compare to real players, or performed any testing to support this end. The discovered characteristics also specifically relate to the behaviour we observed in a small private community server, therefore, they might not represent players from other servers. The current analysis also uses generalised data obtained from different players. To create a more detailed player image, it is beneficial to also differentiate between every individual player that plays on a server as we conjecture that players each have unique play styles.

There still exists a lot more information hidden in the details of certain packets that have not been fully explored in this project. Examples include: the types of blocks held and exchanged by players, and types of mobs fought. This required additional extraction and modification of the gathered packets before the start of data collection, which, unfortunately, was not possible within the time constraints of this project. The data in the packets we observed in this project was untouched, except for a few packets, to more efficiently do in-depth analysis during logging certain interactions could already be further processed. For example, when a packet is received for a certain entity it only contains an `EntityID` field. From this `EntityID` field, it is difficult to, in hindsight, figure out to what creature or player this ID belonged to. This issue became increasingly apparent when parsing the observed data due to the chosen storage format.

As a continuation of the previous paragraph, our data made it hard to quantify the block changes done by individual players. Because this was hard to quantify, currently our model does not posses an activity involving block placement and block breaking.

A technical limitation that we encountered was Yardstick's logging speed. As mentioned in Section 3.2.2, intermittent packet loss was encountered, packets were omitted because they would cause the bot to fall behind, and in some cases even cause the emulated player to time out and drop connection. This happened when there were a lot of entities around, which send constant updates through `EntityHeadLookPacket`, or when the bot went to check on other players in quick succession, causing it to receive multiple `LightUpdatePacket` which contain around 90,000 characters, it caused the output writer to fall behind and skip other packets.

6 Conclusion

The individual server instances in Minecraft-like games do not scale up to large numbers of players. To aid the improvement of scalability issues, we aimed to improve the state-of-the-art in benchmarking by creating a dataset and to generate more knowledge about player behaviour in Minecraft.

As demonstrated, we have shown a way to collect a dataset, have provided analysis of this data and presented a model for player behaviour in Minecraft. Data-collection was done by connecting an emulated player to a Minecraft server and observing player actions by logging the packets sent by the server.

From our created dataset we were able to identify how some of the main activities in Minecraft-like games are related to the packets. Movement data can be processed, and some conclusions about player behaviour can be inferred through mapping their movement on a grid and comparing it with block updates. Additional insights were discovered by looking at distributions regarding a players movement, resource usage and combat activities. Using these insights we were able to present a model of player behaviour, based on our observations of the players from a small private community server.

7 Future Work

This project's aim is to improve the state-of-the-art benchmarking of Minecraft-like games by providing a dataset and the analysis thereof. Although rudimentary data analysis is performed on the gathered data, more advanced data collection and processing techniques could be applied. There are still improvements to be made.

Firstly, more improvements could be made to the data collection system. The option to have multiple systems track individual players would allow for more accurate data as the system does not periodically have to check on other players during multi-player sessions.

As a second improvement, the data collection system can be improved by a blue-green deployment system. A blue-green deployment system invokes two mainframes of the bot, only one of which is live. This means you can push updates on the offline version and switch it out after testing. This method will effectively make it so that if you notice something about your data collection that could be improved there are no complications with the current live version. During this project it became evident it was hard to make changes as they would result in downtime of the bot, causing us to miss out on potential important data.

Furthermore, the resulting analysis and the player models to be constructed from player data are not directly compared to real players, which is the next step in a future project.

Finally, previously mentioned as limitations in section 3.2.2 and further discussed in Section 5 there are still optimizations to be made in the amount of data that is able to be collected and in the format of the data collected. Whether it be through improvements and modifications to Yardstick or a custom interface capable of faster reading and writing. Improved data formatting would lead to an easier to parse dataset, as the amount and

structure of the packets received made it so that parsing proved to be a time consuming obstacle for this project. An improvement would be the creation of links between packets and Minecraft environment data. For example, a `ServerBlockChangePacket` has an internal block state ID. This makes it difficult to process and identify afterwards, as it is hard to distinguish the material type and it's implications from this numerical value. However, if you match the material type with the ID when the packets are received, the dataset becomes easier to analyse for researchers.

References

[1] Minecraft: Education Edition. https://education.minecraft.net/.

[2] Minecraft protocol. https://wiki.vg/Protocol. Accessed: 2020-06-01.

[3] J. Donkervliet. Design and experimental evaluation of a system based on dynamic conits for scaling minecraft-like environments. Master's thesis, Delft University of Technology, 2018.

[4] J. Donkervliet, A. Trivedi, and A. Iosup. Towards supporting millions of users in modifiable virtual environments by redesigning minecraft-like games as serverless systems. In *HotCloud*, 2020.

[5] Eurogamer. University students graduate in official minecraft ceremony. https://www.eurogamer.net/articles/2020-05-18-university-students-graduate-in-official-minecraft-ceremony. Accessed: 2020-07-25.

[6] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov. MineRL: A large-scale dataset of Minecraft demonstrations. *IJCAI*, 2019.

[7] S. Muller, M. Kapadia, S. Frey, S. Klinger, R. Mann, B. Solenthaler, R. W. Sumner, and M. Groß. Heapcraft social tools: Understanding and improving player collaboration in minecraft. 2015.

[8] S. Müller, M. Kapadia, S. Frey, S. Klingler, R. P. Mann, B. Solenthaler, R. W. Sumner, and M. H. Gross. Statistical analysis of player behavior in minecraft. In J. P. Zagal, E. MacCallum-Stewart, and J. Togelius, editors, *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015, Pacific Grove, CA, USA, June 22-25, 2015*. Society for the Advancement of the Science of Digital Games, 2015.

[9] Newzoo Game Industry. The Global Games Market Will Generate $152.1 Billion in 2019 as the U.S. Overtakes China as the Biggest Market. https://newzoo.com/insights/articles/the-global-games-market-will-generate-152-1-billion-in-2019-as-the-u-s-overtakes-china-as-the-biggest-market/. Accessed: 2020-07-25.

[10] S. Shen, N. Brouwers, A. Iosup, and D. Epema. Characterization of human mobility in networked virtual environments. *Proceedings of the 24th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV 2014*, 03 2014.

[11] S. Shen and A. Iosup. Modeling avatar mobility of networked virtual environments. *Proceedings of the 6th ACM International Workshop on Massively Multiuser Virtual Environments, MMVE 2014*, 2014.

[12] K. Squire. Video games in education. *Int. J. Intell. Games & Simulation*, 2(1):49–62, 2003.

[13] The Verge. Minecraft still incredibly popular as sales top 200 million and 126 million play monthly. https://www.theverge.com/2020/5/18/21262045/minecraft-sales-monthly-players-statistics-youtube. Accessed: 2020-07-25.