



Minecraft Social Agents

Creating socially intelligent non-player characters in Minecraft

Rafael Alexandre Milheiro Lourenço

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Rui Prada

May 2024

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

I am deeply grateful to my parents for their unwavering support, guidance, and the countless opportunities they have provided me throughout my academic journey. Their steadfast belief in my abilities has been a constant source of inspiration.

In addition, I extend my heartfelt appreciation to my friends, whose unwavering encouragement and companionship have uplifted my spirits during moments of self-doubt. Their loyalty and camaraderie over the years have been invaluable, and I am fortunate to have them by my side.

Furthermore, I wish to express my gratitude to all other family members for their unwavering pride in my accomplishments. Their encouragement and belief in my aspirations have served as a constant reminder of the importance of perseverance and dedication.

Together, the unwavering support of my family and friends has played an integral role in shaping my academic and personal growth, for which I am truly thankful.

Finally, I would like to extend my sincere gratitude to Professor Rui Prada, my dissertation advisor. Professor Prada's exceptional teaching methods and profound expertise have greatly enriched my academic experience and shaped the trajectory of my work.

Abstract

I conducted an analysis of existing research on socially aware interactions in gaming. Some of the approaches presented are broad in scope, with only a select few focusing on a specific game. I have elected to develop a tailored approach for creating social agents in Minecraft, as this game offers unique possibilities for social interactions that cannot be found elsewhere. Attempting to create a generalized solution that caters to Minecraft as well would diminish the potential of social agents within the game.

This research introduces a Minecraft plugin, named MineSocieties [1], tailored for easy integration into any Minecraft Paper [2] server, streamlining agent deployment without the need for Mods or complex setups. Powered by cutting-edge Large Language Models (LLMs), notably ChatGPT [3], the plugin enables agents to autonomously navigate and interact within the Minecraft society. Additionally, developers who fork this project can seamlessly switch between LLMs, enhancing flexibility and accessibility for diverse research applications.

Keywords

LLM; Large Language Model; Minecraft; Socially intelligent agents, NPC; Plugin; ChatGPT; Social interactions; Artificial Intelligence; Social simulation.

Contents

1	Introduction	1
1.1	Social behavior in games	1
1.2	Minecraft	2
1.3	Implementation	2
1.4	Organization of the Document	3
1.5	Envigased Contributions	4
2	Background	5
2.1	What is Minecraft?	5
2.2	Minecraft Editions	8
2.2.1	Java Edition	8
2.2.2	Bedrock Edition	9
2.2.3	Education Edition	10
3	Related work	11
3.1	Deploying Agents in Minecraft	11
3.1.1	Minecraft Education Edition [4]	12
3.1.2	Modded Minecraft	13
3.1.2.A	MineColonies [5]	13
3.1.2.B	Minecraft Comes Alive (MCA) [6]	15
3.1.3	Minecraft Plugins	16
3.1.3.A	Citizens [7]	16
3.1.3.B	Sentinels [8]	17
3.1.3.C	EntityUtils [9]	17
3.1.3.D	Server Networks	17
3.1.4	Minecraft Bedrock Edition	18
3.1.5	Packet Sending and Receiving	19
3.1.5.A	MineFlayer [10]	19
3.1.5.B	SocialCraft [11]	20

3.1.5.C	Project Malmo [12]	20
3.1.6	Discussion	21
3.2	Social Behaviour	22
3.2.1	FAtIMA Toolkit [13]	22
3.2.2	Space Modules Inc. [14]	23
3.2.3	Large Language Models [15]	25
3.2.3.A	ChatGPT [3]	25
3.2.4	Generative Agents: Interactive Simulacra of Human Behavior [16]	27
3.2.5	Discussion	28
4	Solution	29
4.1	Overview	29
4.2	Architecture Overview	30
4.2.1	Running Software	30
4.2.2	Modules	31
4.2.3	Agent observations and action choices	32
4.3	Development	34
4.3.1	Agent deployment	34
4.3.2	Agent manipulation	35
4.3.3	Agents	36
4.3.4	Agent State	37
4.3.4.A	Reflections	39
4.3.5	Actions	39
4.3.6	Observations	41
4.3.7	Conversion to natural language	42
4.3.8	Interfacing with Large Language Models	43
4.4	LLM Prompts	44
4.4.1	Case 1- Agent deployed with initial description	46
4.4.2	Case 2- Agent must act	48
4.4.3	Case 3- Agent reflects on conversation	52
4.5	User Interface	54
4.5.1	For Players	54
4.5.1.A	Editing NPCs	56
4.5.1.B	Location Editing	58
4.5.2	For Server Owners	59
4.6	Struggles	59

4.6.1	Repetitive NPC Behavior	60
4.6.2	Incorrect Reply Format	60
4.6.3	Redundant Information Extraction	61
5	Results	63
5.1	Evaluation Criteria	63
5.2	Test Scenarios	64
5.3	Experiments with players	64
5.4	Feedback	65
5.5	Reflecting on the feedback	69
6	Conclusion	71
6.1	Conclusions	71
6.2	System Limitations and Future Work	71
	Bibliography	72

List of Figures

2.1	Minecraft cover art	6
2.2	Me breaking a block	6
2.3	Me fighting a hoard of zombies using iron equipment	7
2.4	A cool city [17]	7
2.5	A Tesla Coil, created by me, for Heliomothra	8
3.1	Education Edition's Agent	12
3.2	Education Edition's Visual Programming Interface, taken from Minecraft Education Edition Website	12
3.3	An example of a MineColonies civilization guided by a player	14
3.4	MineColonies NPC with the Builder job	14
3.5	NPCs in MinecraftComesAlive	15
3.6	Player configuring an NPC using the Citizens plugin	16
3.7	Screenshot from Hypixel's "Hide and Seek" mini-game featuring a seeker player who has designated one NPC as "suspicious" and another as "checked"	18
3.8	Built-in NPCs in Minecraft Bedrock Edition	19
3.9	Interaction between two SocialCraft agents through in-game chat.	20
3.10	FAtiMA-Toolkit Authoring Tool example	23
3.11	Space Module Inc. example	24
3.12	Test of human-like dialog with ChatGPT	26
3.13	Test of dialog with action choice using ChatGPT	26
3.14	The paper's first figure	27
3.15	Example of a prompt given to ChatGPT	27
4.1	MineSocieties deployment	30
4.2	Modules and important dependencies	31
4.3	Some of Agents Module's usage of EntityUtils	32

4.4	High-level procedures of observation and action	33
4.5	Lower-level procedures of observation and action	33
4.6	A real player joins the server	34
4.7	A fake player NPC is deployed	34
4.8	A real player joins and receives fake player's information	35
4.9	UML Class Diagram of the agents	36
4.10	UML Class Diagram of the agent's state	38
4.11	UML Class Diagram of the agent's actions	40
4.12	UML Class Diagram of the action visitors	41
4.13	UML Class Diagram of the observations	42
4.14	UML Class Diagram of the visitors	43
4.15	UML Class Diagram of LLM communication classes	44
4.16	The action SendChatTo being executed in-game	51
4.17	NPC Deployment	55
4.18	Steve replying to BlackKnight625's talk command	56
4.19	Alex's editor GUI, main page	56
4.20	Alex's editor GUI, main page, information button	57
4.21	Alex's editor GUI, main page, inventory button	57
4.22	Alex's editor GUI, main page, execute actions button	58
4.23	Alex's editor GUI, information page, long term memories	58
4.24	The social locations GUI	59
5.1	I think the NPCs were not repetitive	66
5.2	It was fun interacting with the NPCs	66
5.3	I quickly understood how to interact with the NPCs	67
5.4	I feel like the NPC's interactions were realistic	67
5.5	It was not frustrating interacting with the NPCs	68
5.6	I found it easy creating and deploying new NPCs	68
5.7	I found it easy editing NPCs (changing their memories, their emotions, their known locations, etc)	69

Acronyms

NPC	Non player character
LLM	Large Language Model
MCA	Minecraft Comes Alive
FAtiMA	Fearnnot AffecTive Mind Architecture
UUID	Universally unique identifier
GUI	Graphical User Interface

Glossary

Agent

In the context of Artificial Intelligence, an Agent is an entity capable of making decisions based on information about the world it operates in. This could be a program or a human. In this thesis, it refers specifically to a socially intelligent Non player character (NPC). 2

BlackKnight625

My in-game name, used in Minecraft and on GitHub. 33

Heliomothra

A Minecraft mini-game created by me. In Heliomothra, 100 players are divided into four teams, with respawns allowed as long as their God is alive. The objective is to be the last team standing. To see some of its features, refer to [this video](#). ix, 8

Minecraft Client

The software launched when a player starts Minecraft. In single-player mode, it initiates a local instance of a Minecraft Server to handle game mechanics and communicates with the client to render the game on the player's screen. In multiplayer mode, it connects to a designated server to receive world data. 7, 9

Minecraft Server

Software that runs on any machine, allowing multiple players to play together by hosting a Minecraft world and sending data packets to players detailing block positions, entities, and other players' actions. The Minecraft Server jar is freely available for download and installation. xiii, 7, 9

1

Introduction

Contents

1.1	Social behavior in games	1
1.2	Minecraft	2
1.3	Implementation	2
1.4	Organization of the Document	3
1.5	Envigased Contributions	4

This thesis addresses the challenge of creating a more immersive gaming experience through advanced Non player character (NPC) interactions. Currently, NPCs in games are often limited in their interactivity; they typically have a predefined set of dialogues and fail to account for past events when making decisions. This restricts the depth and realism of NPC interactions within the game world.

1.1 Social behavior in games

The incorporation of social behavior in NPCs is an area of significant interest and importance in gaming. While games aim to provide immersive experiences through storytelling, and realistic reactions from

in-game elements like mud footprints or a character's physical exhaustion, believable NPCs are also crucial for players to feel like active participants in the virtual world.

The inclusion of social Agents offers the possibility of creating new forms of entertainment solely based on them. For instance, a game where the objective is to fix the relationships of two NPCs solely through dialogue.

Social agents can also add a new dimension to open-world games, making them more exciting and immersive. Besides exploring and discovering new territories, players can interact with and learn from virtual civilizations that respond differently to various situations. Understanding how these civilizations respond to threats, such as invasions of zombies, offers a new level of entertainment and intrigue.

Despite the significance and potential of social agents in gaming, creating NPCs that exhibit social behavior similar to humans remains a challenge. This is due to the complex and nuanced nature of social behavior, which requires advanced AI and machine learning techniques to implement. Video game consoles and personal computers can only do so much. Running large Neural Networks on these devices that mimic human behavior is not a reasonable expectation.

1.2 Minecraft

Minecraft is a vast, open-world sandbox game known for its extensive modifiability. A more detailed discussion of Minecraft can be found in Chapter 2.

The decision to implement socially intelligent NPCs within Minecraft stems from its highly customizable environment and the wide range of possible actions available due to its sandbox nature. In Minecraft, players can alter the world by breaking or placing blocks, providing a rich set of interactions for NPCs compared to other games.

1.3 Implementation

I developed a Minecraft plugin that operates on Minecraft Paper [2] servers. This plugin enables players to easily deploy, edit, and interact with NPCs. The NPCs utilize a Large Language Model (LLM) to determine their actions based on collected observations, such as picking up an item or receiving a chat message. In particular, the LLM used is ChatGPT 3.5-Turbo [3] due to its early availability.

The input and output of LLMs are in natural language text, making them well-suited for generating stories and determining NPC actions at any given moment. LLMs are trained on extensive examples of human interactions, providing a rich basis for creating dynamic and responsive NPC behavior.

Therefore, the decision was made to adopt LLMs as the underlying decision-making mechanism for NPCs.

In addition to decision-making capabilities, the practical deployment of NPCs within the Minecraft environment was addressed. A crucial consideration was to ensure that NPCs seamlessly integrated into the game world, resembling standard player entities. To facilitate ease of accessibility and reproducibility, a plugin architecture was devised. This plugin operates within the Minecraft server environment, leveraging code injection techniques and packet manipulation to emulate the presence of NPCs as indistinguishable entities from genuine players. Notably, Minecraft plugins stand out as the most straightforward modifications to implement within a server environment.

Lastly, I had to add some in-game actions for the NPCs to perform. Since the main focus here is on how NPCs interact socially, I didn't go overboard with this. I included just enough actions to spice up their interactions and keep things interesting.

In short, the implementation involved:

- Developing an efficient method for deploying agents in Minecraft.
- Defining and implementing a range of actions for the agents.
- Establishing a mechanism to convert an agent's context into a textual format for processing by ChatGPT.
- Determining the optimal message format for ChatGPT responses, with a focus on extracting actionable information.
- Translating ChatGPT responses into executable agent actions.
- Deploying agents into a Minecraft world where they can interact as a society, engaging in dialog, forming relationships, scheduling events, distributing tasks, and more.

1.4 Organization of the Document

This thesis is structured as follows: Chapter 1 provides an overview of the problem statement, the underlying motivation, and the proposed solution. Chapter 2 provides essential contextual information and pertinent details concerning Minecraft. Chapter 3 offers an extensive review of existing research on social behavior in games and agent deployment in Minecraft. Chapter 4 delves into the implemented solution, showing its architectural design, developmental trajectory, and the encountered challenges along with their respective resolutions. Chapter 5 presents the outcomes derived from both personal and planned experiments. Finally, chapter 6 wraps everything up, summarizing the problem, why it matters, what I did about it, and what I found out.

1.5 Envigased Contributions

This thesis aims to make the following contributions:

- Demonstrate the potential of applying LLMs to enhance the social experience of games, thereby increasing immersion for players,
- Enhance the sense of vitality in Minecraft's gameplay by incorporating dynamic and contextually appropriate social interactions through the utilization of ChatGPT,
- Streamline the deployment of agents within the Minecraft environment, providing a more accessible and efficient framework for agent integration for both regular players and developers.

By achieving these contributions, this research endeavors to advance the field of game development by leveraging ChatGPT's capabilities to create more engaging and immersive social interactions in games, particularly within the Minecraft ecosystem. Additionally, the proposed framework aims to facilitate the deployment and integration of intelligent agents, fostering a more dynamic and interactive gameplay experience.

2

Background

Contents

2.1 What is Minecraft?	5
2.2 Minecraft Editions	8

2.1 What is Minecraft?

Minecraft [18], developed by Mojang Studios [19] in 2009, is a renowned sandbox video game that has achieved unparalleled success, boasting a staggering 300 million copies sold worldwide [20]. With its open-ended nature, Minecraft offers players the freedom to create their own objectives, whether it involves exploration, mining, constructing cities, battling monsters, or engaging with NPC villagers, among countless other possibilities.



Figure 2.1: Minecraft cover art

Upon entering the game, players find themselves in a procedurally generated world ripe for modification. Blocks can be broken, shaped, and placed, allowing for the creation of structures limited only by imagination. It follows the Sandbox and Open-world genre.



Figure 2.2: Me breaking a block

In Minecraft, the environment and its creatures pose constant threats, pushing players to build shelters for survival and gather resources to bolster their defenses. While the game lacks explicit goals, players can opt to challenge formidable bosses and brave monster-infested structures, all within a world that they can shape and explore at will.



Figure 2.3: Me fighting a hoard of zombies using iron equipment

For those seeking unrestricted creativity, Creative mode offers invulnerability and access to all game items, enabling the construction of vast, intricate structures without resistance.



Figure 2.4: A cool city [17]

Minecraft's multiplayer feature enables players to join each other's worlds through Minecraft Servers, whether they're remote or local. These servers can offer far more than mere cooperative play. Leveraging Minecraft's Java-based code, the community has developed libraries allowing them to inject custom code into both server and Minecraft Client instances. This flexibility empowers players to create and participate in a vast array of custom content, ensuring that Minecraft remains an ever-evolving experience. Whether it's emulating the Hunger Games or introducing futuristic elements like Tesla Coils, the possibilities for adventure are virtually limitless, making multiplayer servers a compelling option for those seeking new challenges beyond the confines of single-player gameplay.



Figure 2.5: A Tesla Coil, created by me, for Heliomothra

Beyond its widespread popularity as a source of entertainment, Minecraft has transcended its gaming realm and found utility in various research endeavors, particularly in the field of Artificial Intelligence. The game's expansive virtual world and versatile mechanics have made it a valuable tool for scientific investigations and experimentation, facilitating advancements in AI and related domains. More about that will be discussed in chapter 3.

2.2 Minecraft Editions

Minecraft was originally developed in Java as a PC game by Mojang Studios [19], and the Java Edition continues to receive updates every year, making it the most played edition of Minecraft. Due to the game's popularity and growth, other editions [21] were created to spread the game to other platforms, such as Xbox, with better performance due to being written in C++.

For this proposal, I will focus on three editions of Minecraft, the Java Edition, Bedrock Edition, and Education Edition since they are the most relevant.

2.2.1 Java Edition

Minecraft Java Edition [22] was the first edition of the game to be developed. It was initially released in 2009 as a single-player game and has since then received regular updates from Mojang Studios. Java Edition was not launched in any game sale platforms, as it had its own website where it could be purchased. When multiplayer was introduced, the only way to play with friends was to download the server code and run it on a machine that allows port-forwarding so that other player connections may exchange packets with it.

Since Minecraft Servers are run by the players and not the company, players have full control over them. Very early on, libraries were created that allowed everyone to easily create a Plugin [23] that modifies how Minecraft Servers behave and could manipulate the packets being sent to Minecraft Clients. Bukkit [24] was one of the Plugin libraries created and its forks are still used.

Plugins are limited in their capabilities. If a player wants their server to have completely new blocks or monsters, there's no packet manipulation that allows Minecraft clients to recognize the custom blocks and entities. There are ways to apply custom texture pack tricks that make players think they're interacting with a brand new entity or seeing a brand new block, but in reality, they're just seeing a retextured existing entity or block.

Minecraft Mods [25] solve the limitations of Plugins, at the expense of forcing players to have to install said mods locally so that the Minecraft client may interpret the new entities, blocks, etc. The Forge Library [26] was one of the first libraries that allowed Minecraft mod development. It allows full control over Minecraft, from changing its very code (for example, to optimize rendering), to adding new blocks, machines, light rendering, sound effects, etc.

Due to its open-source-like nature and active plugin and mod development community, Java Edition has become the most popular version of Minecraft, with a large number of mods and plugins available for download. However, Java Edition's reliance on Java as a programming language means that it is less optimized for performance than other editions of the game. Additionally, the requirement for players to install mods locally means that mods cannot be used in multiplayer unless all players have the same mods installed.

2.2.2 Bedrock Edition

The Bedrock Edition of Minecraft [27] was developed with the aim of expanding the game's reach to more platforms such as Xbox, Mobile, PlayStation, Nintendo Switch, and other devices. It also sought to enhance the game's performance, given that Java is generally slower for game development, while C++ provides a more efficient alternative.

In line with the creation of Bukkit for Java Edition, the Minecraft community developed LiteLoaderBDS [28] as an unofficial plugin loader that offers basic API support for the Bedrock Dedicated Server, with an extensive API. Mods in Bedrock are referred to as "Add-ons" but are less flexible than those in Java Edition, as Mojang's mod compatibility implementation limits creators to "only modify features that Mojang explicitly allows and exposes" [29].

It is worth noting that most of the other Minecraft Editions are based on the Bedrock Edition.

2.2.3 Education Edition

"Minecraft Education is based on Bedrock Edition [27] and contains features that make Minecraft more accessible and effective in a classroom setting" [4]. For example, there's a block called "Element Constructor" which can be used to create elements by adjusting the number of protons, neutrons, and electrons.

While it is not possible to add plugins or mods, this edition offers new features that allow for some customization.

3

Related work

Contents

3.1 Deploying Agents in Minecraft	11
3.2 Social Behaviour	22

In this section, I will detail existing work on implementing social behavior in games and deploying NPCs/agents in Minecraft. I will also highlight the parts that inspired my implementation or were frequently used in my project.

3.1 Deploying Agents in Minecraft

Before delving into Social Behaviour, it is imperative to determine the optimal method for deploying manipulable agents within Minecraft. Given the game's popularity and adaptability, there are numerous approaches available, each with its unique advantages and drawbacks. Some methods may be more accessible to programmers, while others may be easier for casual players to install and experiment with the programmed agents.

3.1.1 Minecraft Education Edition [4]

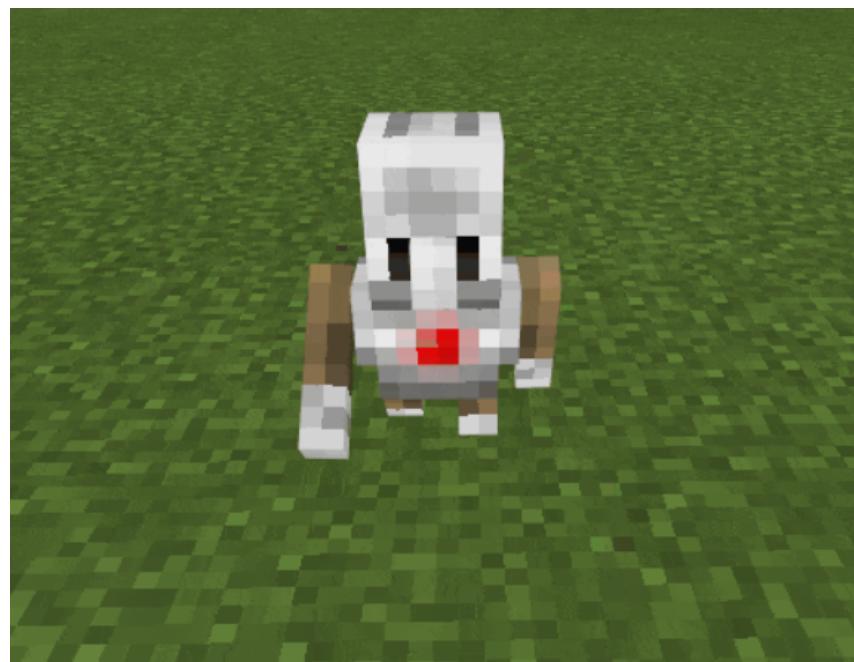


Figure 3.1: Education Edition's Agent

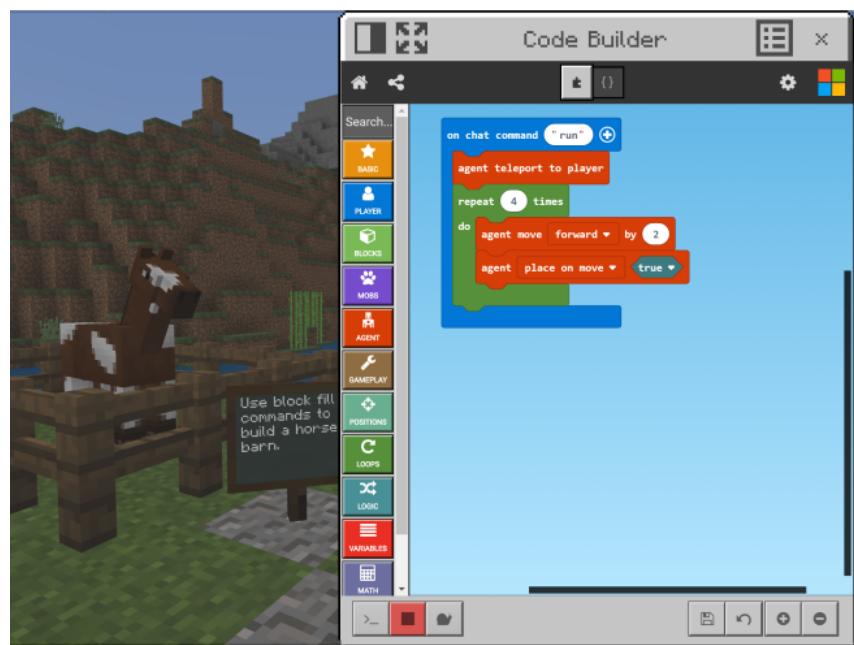


Figure 3.2: Education Edition's Visual Programming Interface, taken from [Minecraft Education Edition Website](#)

Minecraft Education Edition is a version of Minecraft designed for educational purposes. It includes a visual code editor for programming a small in-game entity called "agent."

This edition is created to facilitate interactive teaching. Teachers can design worlds to explain scientific concepts more engagingly and allow students to experiment with programming through the agent entity.

The advantage of using this edition for my thesis is that the visual programming interface simplifies deploying and manipulating agents. However, this edition is not widely used, which conflicts with my goal of accessibility. Moreover, the visual programming interface is intentionally limited, making it impractical to implement complex social behaviors.

Therefore, Minecraft Education Edition is not suitable for this project due to its limited accessibility and restricted programming capabilities.

3.1.2 Modded Minecraft

Mods are limitless in the extensions that they can do to Minecraft. One of the things they can do is enable the addition of any type of customizable entity, including entities that resemble players. There are two mods that allow this: MineColonies [5] and MinecraftComesAlive [6].

When it comes to Minecraft mods, they exert their modifications on both the client and server sides of the game. Leveraging mods provides a convenient avenue for deploying agents, as it grants full control over the game's code. However, it is worth noting that the mod's installation process and its potential re-installation (due to updates) itself can present challenges and inconveniences for players seeking to incorporate these mods.

Playing with mods in Minecraft requires players to follow specific installation steps. Mod launchers, such as CurseForge [30], help simplify this process. However, players who wish to join a server with mods must ensure they have the same mods and versions installed as the server. Creating a modded server is even more complex, demanding a certain level of technical proficiency.

3.1.2.A MineColonies [5]

It's an old mod that remains compatible with newer versions of Minecraft. It endeavors to simulate a civilization within the Minecraft universe. Upon spawning into the world, players can initiate their own civilization, assuming the role of its leader, and even encounter other civilizations that, regrettably, lack a leader and remain stagnant in their development unless assisted by the player as an ally.



Figure 3.3: An example of a MineColonies civilization guided by a player

The NPCs within MineColonies lack true autonomy. To engage in actions, they must be assigned specific jobs by their leader, which is the player. Reproduction among NPCs occurs randomly and is contingent upon the availability of sufficient housing within the civilization to accommodate newborns. Some jobs grant NPCs a degree of autonomy, such as Soldiers who either shadow the leader and provide protection if so commanded or patrol the civilization, vigilantly seeking out threats. Conversely, certain jobs exhibit much less autonomy. For instance, builders (figure 3.4) will solely construct structures designated by the leader and exclusively at the designated locations.



Figure 3.4: MineColonies NPC with the Builder job

Social interactions within the mod are limited in scope. NPC-to-NPC interactions are predominantly confined to random reproduction events, contingent upon suitable housing availability. Player-to-NPC interactions possess a slightly broader range, as players issue commands to NPCs, although the NPCs' behavior remains unaltered regardless of whether their leader's disposition is benevolent or malevolent.

MineColonies is a mod that features NPCs capable of a wide range of actions. These NPCs can walk to specific locations, dig and mine mineshafts, cut down forests, and fight monsters, among other activities. The diverse actions of MineColonies' NPCs inspired some of the actions implemented for the NPCs in MineSocieties.

3.1.2.B Minecraft Comes Alive (MCA) [6]

It's a mod that introduces a distinct approach to NPCs, wherein social interactions play a prominent role. NPCs within MCA possess the ability to engage in actions such as marriage, reproduction, and forming families, with players themselves being able to initiate familial connections. Importantly, MCA NPCs exhibit a degree of autonomy even in the absence of player interaction. They engage in activities such as farming, construction, procreation, and interactions with other NPCs. The NPCs perceive the player as one of their own, rather than as a leader, although players have the potential to assume leadership roles within the mod.

In comparison to MineColonies, MCA NPCs feature substantially richer social interactions, as they possess individual opinions that influence their choices. For instance, in order to pursue marriage with an NPC, players must first engage in flirtatious behavior, such as telling pre-defined jokes (which may or may not resonate with the NPC's sense of humor) or presenting gifts (which may be met with varying degrees of appreciation based on the NPC's personal preferences).



Figure 3.5: NPCs in Minecraft Comes Alive

Of all the NPCs presented in this thesis thus far, those within Minecraft Comes Alive most closely mirror the appearance of the Minecraft player, thereby creating a more immersive experience.

One notable advantage of this mod is its status as an [open-source](#) project. Consequently, forking from the mod would represent a viable prospect for this thesis.

3.1.3 Minecraft Plugins

Through the utilization of plugins, it becomes possible to inject code into the server environment, thereby simulating player-like entities by dispatching packets to real players containing the actions of fictional players. Consequently, the Minecraft client of each player renders these fabricated entities and their corresponding actions, operating under the assumption that the packets originate from genuine player interactions.

A key distinction between plugins and mods is that plugins do not alter the Minecraft client. Consequently, the modifications made to the Minecraft server are more limited. For instance, a plugin cannot add new blocks to the game because the client would not know how to render them.

This also makes their installation much easier. All the work is handled by Minecraft server owners, who install the plugin on their server and configure it as needed. Players who join the server do not need to follow any installation steps. An unmodified Minecraft client can connect to any server, whether it has plugins or not.

3.1.3.A Citizens [7]

It's a famous plugin that stands as a widely recognized tool for incorporating player-like NPCs into Minecraft, enabling some degree of behavior customization through packet manipulation. However, it is important to note that this plugin does not encompass social behavior. Primarily employed by server owners, it finds application in populating server lobbies with interactive characters that engage players in conversation when they approach and execute predetermined commands upon player interaction. Moreover, the plugin's open-source nature permits developers to expand its Application Programming Interface (API), thereby facilitating the creation of more engaging and intricate interactions.



Figure 3.6: Player configuring an NPC using the Citizens plugin

3.1.3.B Sentinels [8]

The Sentinels plugin builds upon the open-source features of Citizens, providing enhanced functionality that empowers NPCs with configurable combat abilities, such as patrolling. It is worth noting that the Sentinels plugin does not extend the Citizens plugin with social behavior and interactions, as it focuses solely on combat. Similar to Citizens, the Sentinels plugin is also open-source.

3.1.3.C EntityUtils [9]

It's a plugin currently under development by a work colleague at EVNT Games [31]. It provides a streamlined solution for deploying player-like NPCs in Minecraft. As part of my work related to Minecraft Plugin and Mod development at EVNT Games, I have had the opportunity to contribute to this plugin and have gained direct access to it. This means that in the event of encountering any bugs or issues, I can address them personally, without being solely dependent on the plugin's creators. The plugin simplifies the process of incorporating player-looking NPCs into our projects, offering greater convenience and ease of use compared to alternative solutions like the Citizens plugin (for example, Citizens does not offer a way to make an NPC pretend to break a block). If needed, we can optimize and fine-tune the plugin's functionality to align with our specific requirements, resulting in a more tailored and efficient experience for our Minecraft Plugin development endeavors.

3.1.3.D Server Networks

Server networks consist of interconnected clusters of Minecraft servers. Players connect to these networks via a proxy IP, which routes them to one of the servers within the network. Typically, these servers utilize plugins that enable players to switch between servers within the network seamlessly. For example, a network might include two servers: one serving as a waiting lobby where players gather when they first join the network or after a game ends, and another server where the actual gameplay occurs.

They extensively leverage and harness the power of plugins to achieve a myriad of objectives, ranging from the creation of captivating mini-games to the implementation of cheat detection mechanisms and the integration of novel features that seem like they're pushing the boundaries of plugin development possibility. Server Networks represent a highly popular avenue for players to engage in collaborative Minecraft gameplay, as they simply need to launch Minecraft, locate the IP address of a desired network, and effortlessly join without the need for additional installations. This convenience accounts for the considerable disparity in player numbers between those who frequent Modded Minecraft servers and those who opt for Plugin-based Minecraft servers.

One preeminent example of a Minecraft Server Network is Hypixel [32], which, at the time of writing his, boasts more than 51000 consecutive online players. Hypixel's success ultimately led to the forma-

tion of Hypixel Studios [33] as a dedicated entity following the server network's meteoric rise. The network's accomplishments have been formally recognized through the acquisition of four Guinness World Records, as documented in "[It's Official - Hypixel holds 4 Guinness World Records](#)". Their resounding triumph stems from their ingenious utilization of plugins. It is crucial to reiterate that plugins cannot introduce new content to the game, as their operations are confined to the server realm rather than the Minecraft clients. However, Hypixel's exceptional creativity within these limitations has cemented its reputation. An illustrative instance pertinent to this thesis is their renowned mini-game titled "[Hide and Seek](#)". In this captivating mini-game, players are divided into two teams: hiders and seekers. The game unfolds within a map teeming with NPCs that emulate authentic player actions. Hiders must assimilate among the NPCs and complete assigned tasks to secure victory, while seekers endeavor to locate and eliminate all hiders before they accomplish their objectives.



Figure 3.7: Screenshot from Hypixel's "Hide and Seek" mini-game featuring a seeker player who has designated one NPC as "suspicious" and another as "checked"

3.1.4 Minecraft Bedrock Edition

The Minecraft Bedrock Edition is a version of Minecraft written in C++ instead of Java. It offers a straightforward method for creating Non-Player Characters (NPCs) that resemble players [34]. However, their actions are severely restricted, as they are only capable of executing commands and displaying dialogue.



Figure 3.8: Built-in NPCs in Minecraft Bedrock Edition

Alternatively, it is possible to create NPCs in a manner akin to the Java Edition, utilizing Bedrock's equivalent counterparts of Mods and Plugins. Nonetheless, it is worth noting that such an undertaking proves considerably more challenging in the Bedrock Edition due to the more limited extent of plugin and mod creation capabilities, as well as the comparatively smaller community support it enjoys in contrast to its Java counterpart.

3.1.5 Packet Sending and Receiving

An alternative approach involves the development of a program capable of sending and receiving packets to a Minecraft server while impersonating a player. However, this option presents a significantly higher level of complexity. The program must be able to format packets in a manner that Minecraft can interpret, as well as interpret packets received from the server. Additionally, this approach requires a second non-Minecraft program to be running, adding an extra layer of intricacy that may discourage some users from exploring it.

The primary advantage of this approach lies in the flexibility it offers regarding the choice of programming languages and frameworks. Developers can select languages independent of Minecraft's supported languages (Java, C++). For example, if developers wish to deploy agents whose actions are determined by machine learning techniques, they may opt for Python and leverage its extensive AI resources. In this scenario, the program would receive packets from a Minecraft server, interpret them, choose actions, and send corresponding packets back to the server.

3.1.5.A MineFlayer [10]

It's a JavaScript API that simplifies the deployment of NPCs into the game by utilizing packets, as mentioned earlier. Its powerful API enables developers to easily issue complex commands to NPCs, such as building structures or attacking entities. However, one notable drawback is its usage complexity. To employ the agents, players need to install the necessary JavaScript dependencies and run the deployment

script, making it less accessible for non-developers.

3.1.5.B SocialCraft [11]

The predecessor to this thesis, pursued the same objective of deploying socially intelligent agents into Minecraft. It utilized MineFlayer to execute agent actions. In SocialCraft, agents possessed autonomy and would programmatically determine their actions based on their assigned job. Additionally, they exhibited social characteristics, including traits like "agreeableness" and a "friendship" rating, influencing their social interactions, such as engaging in dialog, as depicted in Figure 3.9.



Figure 3.9: Interaction between two SocialCraft agents through in-game chat.

The implementation of SocialCraft involved the utilization of [Docker](#), where each agent operated within its dedicated Docker instance in its own process, sending and receiving packets. However, this introduced additional complexity to the deployment process, as Docker is not particularly user-friendly, even for developers like myself who encountered challenges while working with it.

3.1.5.C Project Malmo [12]

This project was developed by Microsoft and represents a sophisticated AI experimentation platform built on top of Minecraft, designed to facilitate fundamental research in artificial intelligence. Agents within Malmo run on their own process and receive input in the form of screen pixels from the game, along with optional additional data such as block positions and types in their vicinity. The pixel information is provided to the agents through a custom Mod developed by Microsoft, which establishes communication with the running agent process. "Minecraft needs to create windows and render to them with OpenGL, so the machine you do this from must have a desktop environment." [35], therefore, machines used for this purpose must have a desktop environment, rendering Malmo unsuitable for servers running on

headless machines due to the fact that wanting to run agent processes on the machine wouldn't work, which limits its usability for the majority of server owners who rely on dedicated rented machines.

Project Malmo enables the creation of objectives for agents to accomplish, and during training mode, agents leverage deep learning methods to improve their performance in achieving these objectives.

3.1.6 Discussion

The Education Edition's NPCs 3.1.1, with their robot-like appearance, fail to replicate the immersive experience of interacting with actual players. Additionally, their behavior programming is confined to a drag-and-drop interface, which offers limited flexibility compared to traditional programming languages.

Project Malmo 3.1.5.C, while capable of teaching agents to accomplish objectives using deep learning techniques, receiving the screen's pixels as input, is not suitable for my thesis due to the inherent challenge of teaching social behavior through such methods. Although Malmo could be valuable for executing agent actions like mining in caves, its limitation of being unable to deploy agents on servers without a desktop environment renders it impractical for widespread usage.

While SocialCraft 3.1.5.B demonstrated some success, its intricate setup process hinders its accessibility to the majority of users. In order to launch an agent, a docker environment had to be set-up, then a mixture of Python and JavaScript had to be launched in order for an agent to send packets to the server.

The Modded Minecraft approach initially appears to be a logical choice, given its expansive possibilities and existing groundwork to build upon. However, the additional burden placed on players to install mods presents a drawback, leading me to favor the plugin alternative. The broader capabilities of mods do not necessarily enhance the social behavior of NPCs, which is the primary focus of this research.

MineColonies 3.1.2.A excels in simulating civilizations led by players. However, its NPCs are heavily tied to their roles, making it challenging to introduce dynamic action choice. Nonetheless, the diverse range of actions NPCs can perform has inspired some of the features implemented in MineSocieties.

MinecraftComesAlive 3.1.2.B offers the potential as a suitable starting point for a forked project. However, as I will explain later in this paper, my implementation of social interactions will not be algorithmic. Thus, a significant restructuring of the action selection process would be necessary, diminishing the value of reusing the project. As such, the primary benefit would be limited to agent deployment, making it less compelling.

Although EntityUtils 3.1.3.C is still a work in progress, it presents a convenient option for me as I have editing access to its code, allowing for seamless implementation of agent actions, such as mining. Consequently, EntityUtils surpasses the alternative of forking from the Citizens plugin. Both plugins facilitate easy agent deployment, but Citizens' agent actions offer limited utility, as they lack essential player activities like wood chopping and farming.

While plugins impose more limitations compared to mods, these constraints do not impede the objectives of my thesis. Plugins enable the creation of NPCs and the manipulation of their actions and behaviors, without needing to introduce new NPC types, items, or effects. Therefore, the potential offered by plugins is more than sufficient, reducing the advantage of mods. Furthermore, the installation process of plugins does not burden players directly but rather places the responsibility on server owners.

In conclusion, my chosen approach involved developing a single Minecraft Java Plugin, named MineSocieties. The limitations imposed by plugins do not hinder the project's objectives, and they are the easiest Minecraft modifications to install. Additionally, I possess several years of experience in plugin development, eliminating the need to invest time in learning a new system. I leveraged EntityUtils as a dependency to streamline agent deployment and agent action execution, aligning with the requirements of my thesis.

3.2 Social Behaviour

The aim of this thesis is to develop Minecraft agents that exhibit social behavior comparable to that of humans. The concept of artificial social behavior extends beyond Minecraft and has been extensively studied in diverse settings. Consequently, there exist several research papers exploring artificial social behavior in various contexts, as well as numerous games and studies that incorporate social behavior.

Historically, classical approaches to artificial social behavior have employed algorithmic techniques. These methods dictate the agent's responses based on predefined rules, such as reacting in a specific way if the agent is angry and if a player performs a particular action. However, a more recent trend has emerged, favoring the utilization of LLMs. LLMs possess the distinct advantage of being trained on vast quantities of data, including written social interactions. This enables them to simulate human-like responses by using contextual factors such as mood, prior conversations, and more.

3.2.1 FAtiMA Toolkit [13]

The Farnot AffecTive Mind Architecture (FAtiMA) Toolkit is a comprehensive collection of tools and assets designed to facilitate the creation of characters with social and emotional intelligence. It has been developed by "GAIPS – Group of Artificial Intelligence for People and Society" [36], which is a "research group on agents and synthetic characters at INESC-ID" [36] [37]. Building upon the advancements made in the development of the FAtiMA agent architecture, the toolkit offers a range of functionalities to enhance character behavior.

"FAtiMA Toolkit is a collection of tools/assets designed for the creation of characters with social and emotional intelligence. The project is the result of the work that was developed under an EU-funded

project named RAGE and an FCT funded project named Slice. As implied by its name, the toolkit is a continuation of the work done in developing the FAtiMA agent architecture.” [38]

One notable component of the FAtiMA Toolkit is the “FAtiMA-Toolkit Authoring Tool,” [39] which provides users with a user-friendly interface for creating scenarios and defining emotional characteristics for characters. This tool enables users to customize the responses of characters based on specific player interactions, taking into account factors such as the character’s mood. Figure 3.10 illustrates an example of the FAtiMA-Toolkit Authoring Tool.

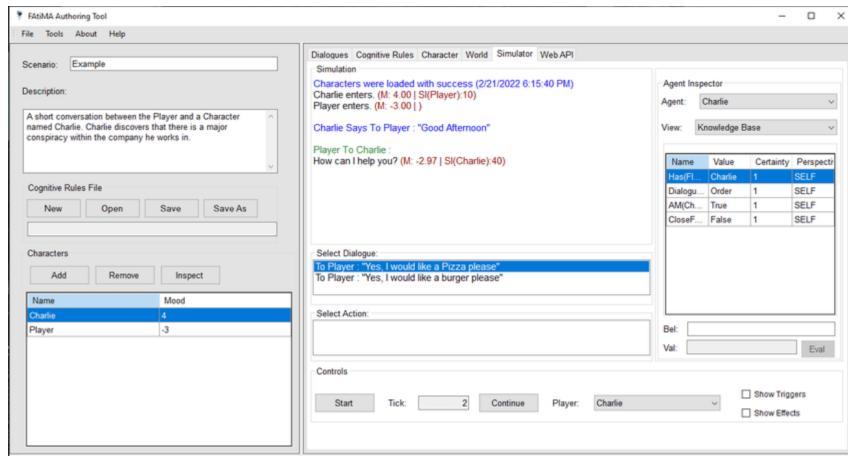


Figure 3.10: FAtiMA-Toolkit Authoring Tool example

With the FAtiMA Toolkit and its Authoring Tool, creators gain access to a comprehensive set of resources to infuse their NPCs with nuanced social and emotional behaviors. This results in a more immersive and engaging gameplay experience for players. Leveraging logic inference, the toolkit enables characters to dynamically adapt their responses and actions based on various situational factors, thereby enhancing the depth and realism of their interactions within the environment.

FAtiMA allows for defining how characters should interact when they are in certain moods or experiencing specific emotions. It provides a framework for characters to have objectives, lists of possible actions, desires that can become objectives, and beliefs that may be true or false.

Once the character and scenario data (possible player input, NPC responses, etc) are specified, they can be saved into files. For a game to interpret these files, it must use the FAtiMA-Engine as an external library in its development engines such as Unreal Engine [40] and Unity [41].

3.2.2 Space Modules Inc. [14]

Space Modules Inc. is an engaging game developed by Playgen, an esteemed game development company [42]. In this unique game, players assume the role of help desk employees working for a prominent spaceship part manufacturer. Their primary responsibility is to assist customers in resolving

their problems by employing effective questioning techniques to gather crucial information and identify appropriate solutions. The success of players hinges on their ability to maintain a friendly demeanor and skillfully manage customer emotions to ensure that interactions do not end abruptly.

Figure 3.11 provides a glimpse into the gameplay experience of Space Modules Inc., showcasing the intricacies of customer interactions within the game.

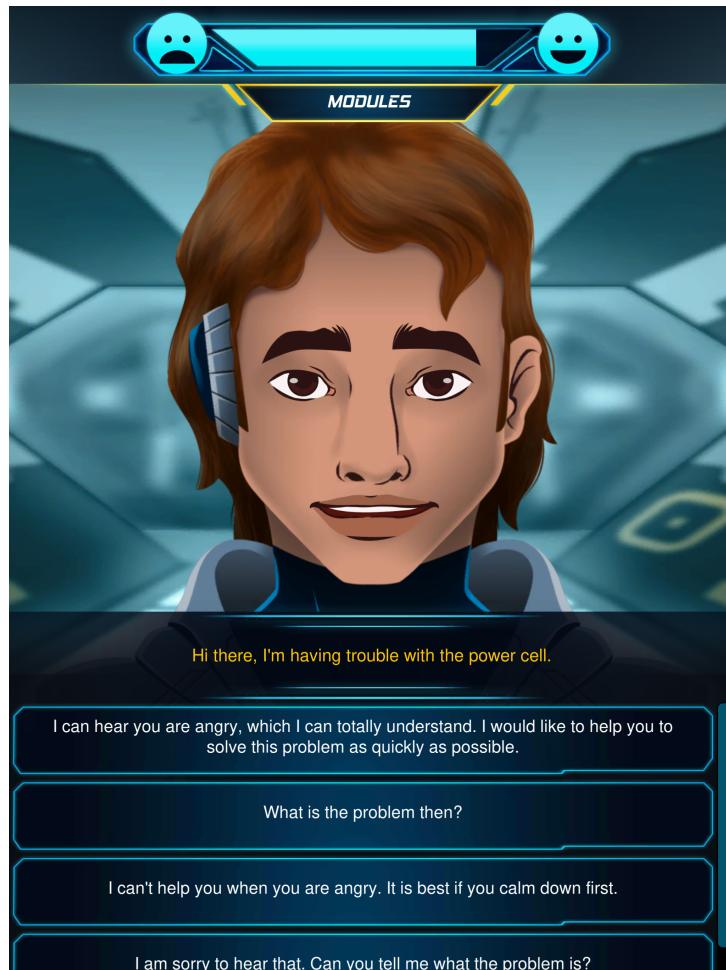


Figure 3.11: Space Module Inc. example

To facilitate the creation of realistic and engaging social interactions, the developers utilized the FAtIMA framework within the Unity [41] game engine. FAtIMA's powerful capabilities enabled the design of dynamic player responses and the realistic modeling of customer reactions based on those responses. This integration of FAtIMA in Unity demonstrates the vast potential of the framework in constructing lifelike dialogues that simulate social interactions.

Space Modules Inc. serves as a compelling testament to the effectiveness of FAtIMA in creating rich and immersive social experiences within gaming environments. By leveraging the power of FAtIMA, game developers can infuse their creations with sophisticated social interaction mechanics, enhancing

player engagement and immersion.

3.2.3 Large Language Models [15]

LLMs serve a multitude of purposes, encompassing tasks ranging from assisting humans in problem-solving to generating text. Due to their extensive training on vast amounts of human-written text, LLMs have acquired a deep understanding of social interactions documented in written form. Consequently, certain LLMs demonstrate a remarkable ability to predict human behavior in response to social stimuli, taking into account the individual's emotional state. This observation led to the hypothesis that LLMs could effectively simulate the social behavior of NPCs. To investigate this hypothesis, I conducted an experiment using one of the LLMs.

However, it is important to note that LLMs are still relatively new and undergoing active experimentation. Consequently, the availability of LLMs to the general public is limited, and some models have complex setups or rely on outdated Python libraries, making them challenging to utilize. For instance, the [YaLM-100B language model](#) demands substantial disk space and GPU memory, rendering it impractical for typical PC users due to its resource-intensive requirements.

The current landscape of LLMs for public access remains somewhat restricted, necessitating careful consideration and selection of appropriate models for research and practical applications involving social behavior simulations. Nonetheless, as LLM technology continues to advance, we can anticipate more accessible and user-friendly models that enable wider exploration and utilization of their capabilities.

3.2.3.A ChatGPT [3]

is a prominent example of a LLM developed by OpenAI [43]. It has been trained using the [Reinforcement Learning from Human Feedback \(RLHF\)](#) approach. The model itself is not open-source. However, OpenAI provides a convenient web interface for interacting with ChatGPT, accessible through their website (<https://chat.openai.com/>), and offers libraries in multiple [programming languages](#) that allow developers to integrate ChatGPT into their applications. It's worth noting that an API key is required to make queries using these libraries.

To assess the viability of using ChatGPT for simulating human behavior in NPCs, I conducted a test within ChatGPT's web interface, as depicted in Figure 3.12.

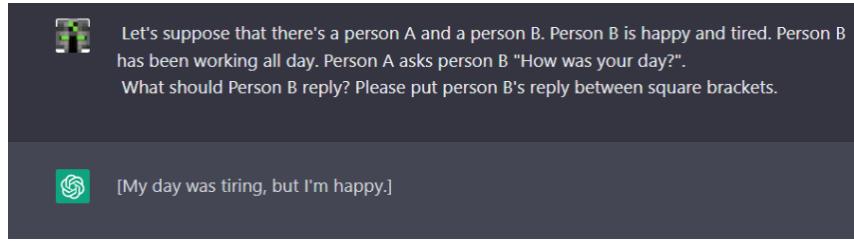


Figure 3.12: Test of human-like dialog with ChatGPT

Additional tests were conducted under different circumstances, consistently yielding realistic and context-aware responses. However, due to the page limit of this thesis, further examples will not be presented.

This test demonstrates a significant advantage of ChatGPT over FAtIMA: fully dynamic dialog. While FAtIMA is restricted to predefined dialog choices for players and pre-programmed responses from NPCs, ChatGPT allows unrestricted player input and highly dynamic NPC replies.

Furthermore, to align with my broader goals beyond dialog generation, I expanded the prompts, as illustrated in Figure 3.13.

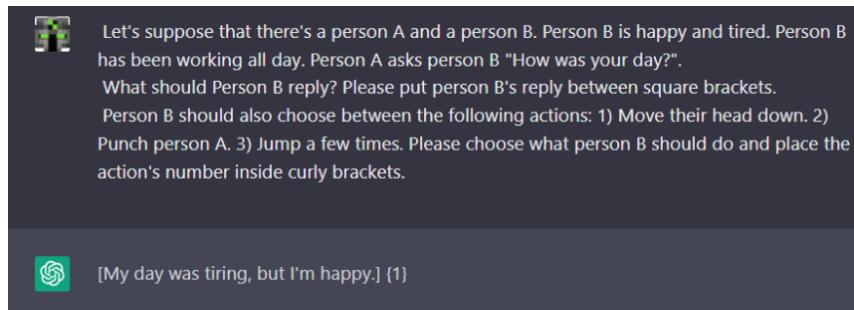


Figure 3.13: Test of dialog with action choice using ChatGPT

Notably, ChatGPT not only generates consistent and realistic replies but also selects the most appropriate actions based on the contextual situation. This capability indicates immense potential for integrating ChatGPT into NPC behavior systems.

It is worth mentioning that the tests conducted involved ChatGPT-3, OpenAI's ChatGPT model available to the public at that moment. ChatGPT-4 [44], its successor, has shown significant improvements in understanding context and generating even better responses. However, access to ChatGPT-4 requires the web user to have a monthly subscription or the API key holder to pay 10x more than if they were using ChatGPT-3.5-turbo.

3.2.4 Generative Agents: Interactive Simulacra of Human Behavior [16]

The utilization of ChatGPT for choosing NPC dialog and actions is already being employed in simple games, as demonstrated in [this video](#). It showcases interactions between NPCs in a 2D world, where NPCs attend a birthday party organized through word-of-mouth invitations, highlighting ChatGPT's ability to handle context effectively. This further validates the hypothesis and underscores the potential of using ChatGPT for simulating social behavior in NPCs.



Figure 3.14: The paper's first figure

Researchers from the United States conducted this project utilizing ChatGPT to determine the dialog and actions of agents by formulating the agent's context as a prompt and querying ChatGPT for guidance. To maximize the information richness of the ChatGPT prompts, the researchers incorporated the agent's memory, which was initially established through a natural language paragraph explicitly describing the agent, and the agent's reflections, representing a more abstract form of memory, encompassed one or more inferred observations with the inference process also facilitated by ChatGPT.

```
[Agent's Summary Description]
It is February 13, 2023, 4:56 pm.
John Lin's status: John is back home early from work.
Observation: John saw Eddy taking a short walk around his workplace.
Summary of relevant context from John's memory:
Eddy Lin is John's son. Eddy Lin has been working on a music composition for his class. Eddy Lin likes to walk around the garden when he is thinking about or listening to music.
Should John react to the observation, and if so, what would be an appropriate reaction?
```

Figure 3.15: Example of a prompt given to ChatGPT

This project closely aligns with the objectives of my thesis, particularly in terms of achieving socially adept agents. Although the project's source code is not publicly available, I drew inspiration from their architectural explanations and provided examples to inform the development of my own agent system.

3.2.5 Discussion

While FAtiMA is a valuable tool for creating social interactions, its scope is limited primarily to conversations. In the case of MineSocieties, NPCs are expected to engage in a broader range of activities, including gift-giving and forming relationships. Consequently, if FAtiMA were employed, it would solely serve the dialog aspect of MineSocieties. Moreover, utilizing FAtiMA would hinder the development of MineSocieties as a Java-based Minecraft mod or plugin since FAtiMA lacks Java compatibility. On the other hand, ChatGPT exhibits significant promise by not only generating dialog but also selecting contextually appropriate actions. By utilizing the ChatGPT API, I can request ChatGPT to suggest suitable actions based on a given set of options and the NPC's context written in natural language. This approach resolves the challenge of ensuring NPCs respond in a manner consistent with their respective contexts.

Consequently, I have opted to leverage ChatGPT to simulate social behavior by utilizing its capabilities to generate dialog and choose actions from a predefined list. During the implementation phase, I got inspiration from the "Generative Agents: Interactive Simulacra of Human Behavior" 3.2.4 project due to its successful attainment of compelling NPC social behavior and its innovative ideas.

4

Solution

Contents

4.1 Overview	29
4.2 Architecture Overview	30
4.3 Development	34
4.4 LLM Prompts	44
4.5 User Interface	54
4.6 Struggles	59

4.1 Overview

The developed solution is a Minecraft plugin that operates on a Minecraft server, called MineSocieties. My approach involved leveraging LLMs, in particular ChatGPT, as the decision-making engine for all Agents. To accomplish this, the contextual information in which an Agent finds itself must be translated into natural language text. The text must be formatted in a manner that conveys the necessary information to ChatGPT, which will in turn provide a response that can be translated into an in-game action. EntityUtils (3.1.3.C) is used to deploy agents and execute the decided actions.

4.2 Architecture Overview

This section provides a high-level overview of my implementation.

4.2.1 Running Software

First, the MineSocieties plugin must be run on a server.

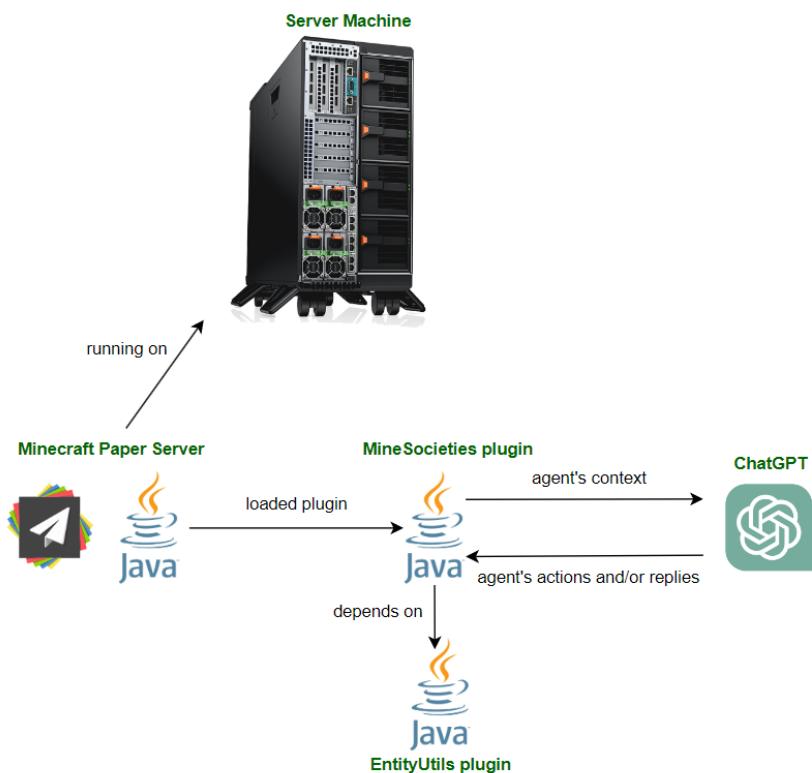


Figure 4.1: MineSocieties deployment

As shown in Figure 4.1, there's a physical machine (this could be a dedicated server, a personal computer, or even a [Raspberry Pi](#)) that must have Java 17 or newer installed. The Minecraft Paper Server runs on this machine with the MineSocieties Plugin loaded. MineSocieties depends on EntityUtils for NPC deployment and manipulation, and it queries the LLM whenever social behavior is required, such as when an NPC needs to act in coherence with its current context.

Two programs need to run for MineSocieties to function: the Minecraft Paper Server's jar on a machine and the LLM's program to listen for prompts. In the case of ChatGPT, it runs on OpenAI's servers. If a local LLM is used, it must be running whenever the Minecraft server is active.

For all examples in this architecture, the LLM used will be ChatGPT.

4.2.2 Modules

To reduce coupling and potentially allow developers to use parts of my code as dependencies without depending on the entire plugin, I separated it into three modules:

- **Agents:** This module provides the abstraction for agent actions and observations, serving as a base for constructing more complex actions and observations. It allows developers to define any type of Agent, from Reactive Agents to Social Agents (MineSocieties' agents), using EntityUtils to deploy and manipulate the agents.
- **LargeLanguageModels:** This module abstracts the communication with the LLMs. Users of this module do not need to remember the specific communication formats for different LLMs. For example, communicating with ChatGPT is done through HTTP requests with a specific data format and API Key.
- **MineSocieties:** This is the main module that depends on the Paper library and whose compiled result is what the Minecraft Paper server accepts as a plugin.

The following figure shows the project's modules and their most important dependencies:

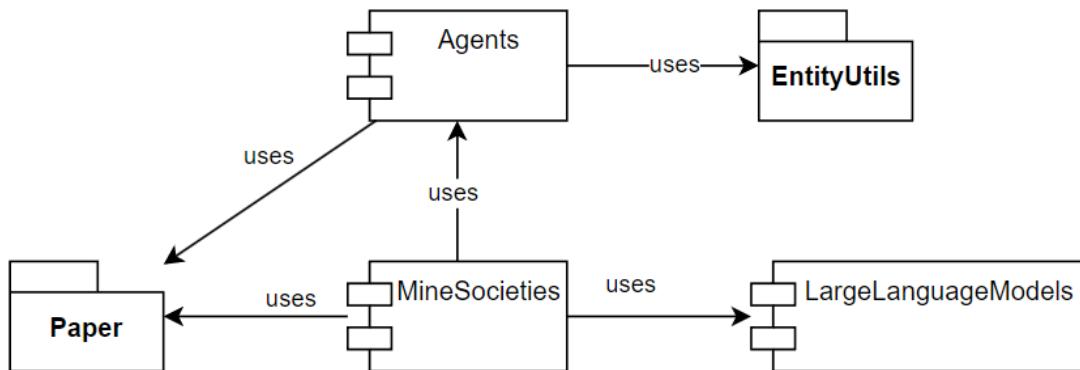


Figure 4.2: Modules and important dependencies

MineSocieties uses the Paper library to manipulate Minecraft's code, allowing control over many aspects, such as the world's weather, entities' health, and block manipulation. MineSocieties uses it to send special messages to players, enable custom commands in-game, and more. It uses the Agents module to create specialized instances of Social Agents and their Social Actions and Social Observations, and uses the LargeLanguageModels module to interface with the LLMs.

The Agents module uses EntityUtils to deploy and manipulate the fake Player instances through high-level methods.

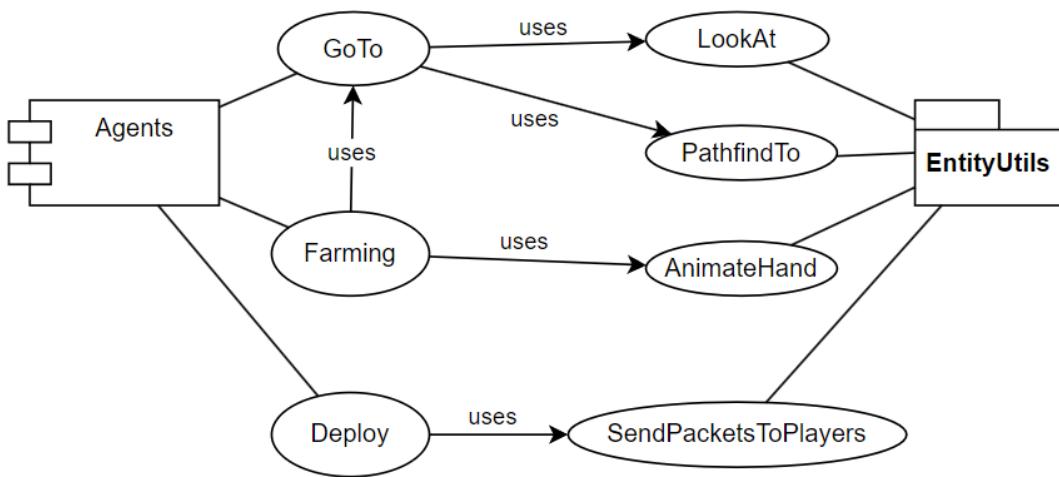


Figure 4.3: Some of Agents Module's usage of EntityUtils

The module's actions manipulate the NPCs, making them look in certain directions, move to specific locations, play animations, and so on. In Figure 4.3, the Farming action makes NPCs go to the location of grown crops and play an animation when they harvest or plant crops. The Agents module does not need to send packets to real players with all these movements or when a new Agent should be deployed, as EntityUtils abstracts that. More details are in 4.3.

4.2.3 Agent observations and action choices

MineSocieties' NPCs choose actions as reactions to observations or after finishing their current actions. Currently, I have implemented the following observations:

- Weather change,
- Received chat message,
- Picked up item.

And the following actions:

- Go farming,
- Go fishing,
- Go to location,
- Do nothing,
- Give item to,

- Send chat message to.

When an NPC receives an observation, it begins the necessary procedures to react. A high-level representation of this process is shown in Figure 4.4:

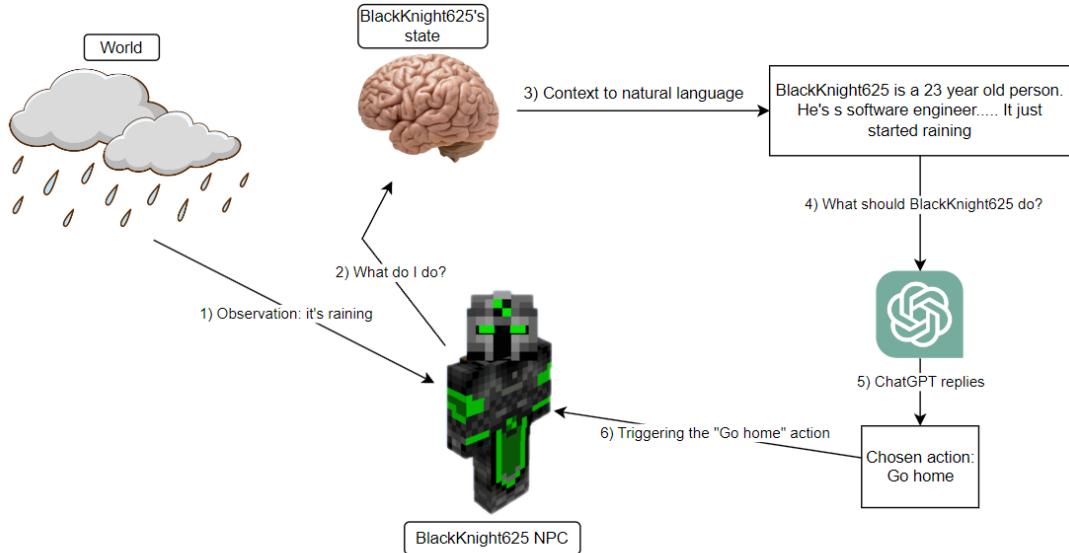


Figure 4.4: High-level procedures of observation and action

This figure illustrates the NPC BlackKnight625 reacting to the observation that it started raining. The NPC receives the observation and then consults its internal state to determine its response. The state converts itself and other surrounding context into natural language, which is then sent to ChatGPT. ChatGPT selects one of the available actions, which is then translated into an in-game action.

A lower-level representation of this procedure can be found in the following sequence diagram:

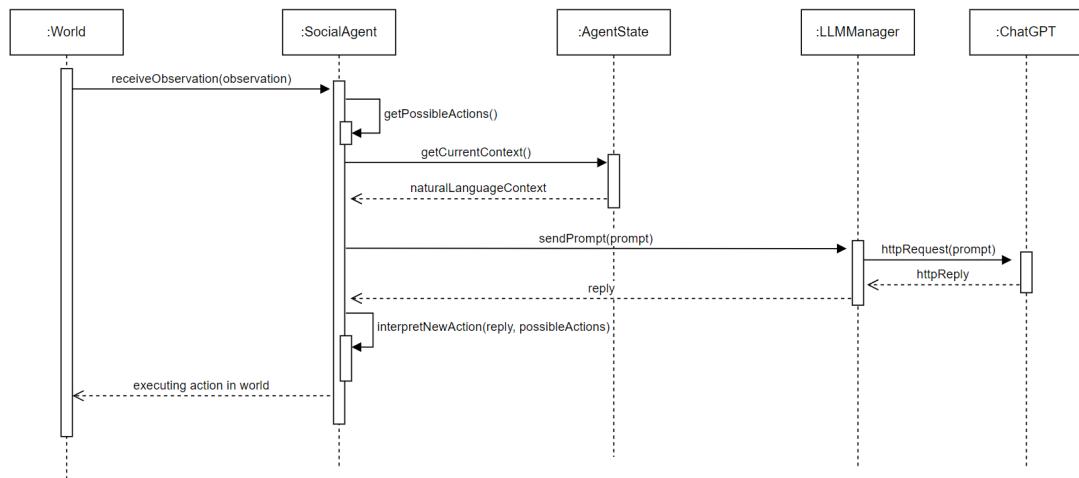


Figure 4.5: Lower-level procedures of observation and action

The process is similar when an agent finishes its current action and needs to decide its next one.

4.3 Development

This section provides detailed explanations of MineSocieties' implementation, focusing on deploying NPCs, manipulating NPCs, NPC observations and actions, and the decision-making process.

For those interested in examining the implementation, the MineSocieties project is open-source and available at [1]. The complete codebase is accessible for review and exploration.

4.3.1 Agent deployment

To deploy an entity that resembles a player on a server using the plugin approach, the server essentially needs to inform all real players, "Look, a new player has joined!". In programming terms, this means sending packets to real players that the server would normally send when a real player joins the server.



Figure 4.6: A real player joins the server

Figure 4.6 illustrates the process when a real player (Player1) joins a server that already has two players online (Player2 and Player3). After the server completes the necessary initial processes before a player is allowed to connect, it sends packets to the real players to inform them about the new player's arrival. This allows Player2 and Player3 to render Player1's character with the appropriate skin and username at its location.

Using a Minecraft plugin, we can replicate this process to deploy an NPC that appears as a player:

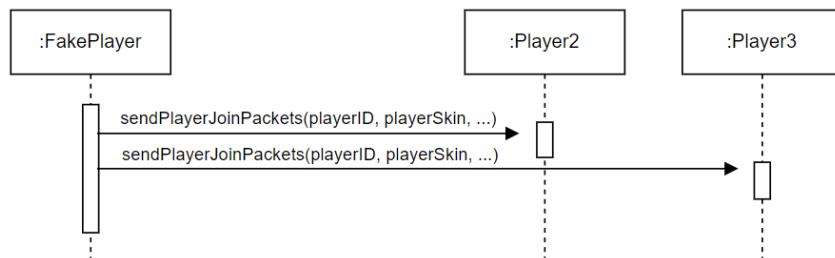


Figure 4.7: A fake player NPC is deployed

To deploy a player-looking NPC with any desired username and skin, we send the same packets that the server would normally send to real players. The Minecraft clients of the real players cannot distinguish between packets from real and fake players, so they treat the NPC as if it were a real player.

EntityUtils provides classes to represent player-like NPCs. To deploy an NPC, an instance of these classes is created, and its `setAlive()` method is called with the argument set to true. This action sends packets to real players containing the NPC's ID, skin, location, and equipment from its internal inventory. EntityUtils deals with this heavy lifting, allowing the Agents module to deploy agents with this simple method call.

However, this alone is insufficient. If a player joins after the fake players are deployed, their client will not render the fake player since it does not know about its existence. When a player joins the server, they receive packets about the world and online players. Fake players are not in the list of online players, so no packets about them are sent. EntityUtils maintains a registry of created fake players and injects code to ensure that its `NPCManager` is notified when a player joins the server. This ensures that new players receive information about all registered fake players. The following Figure illustrates this process:

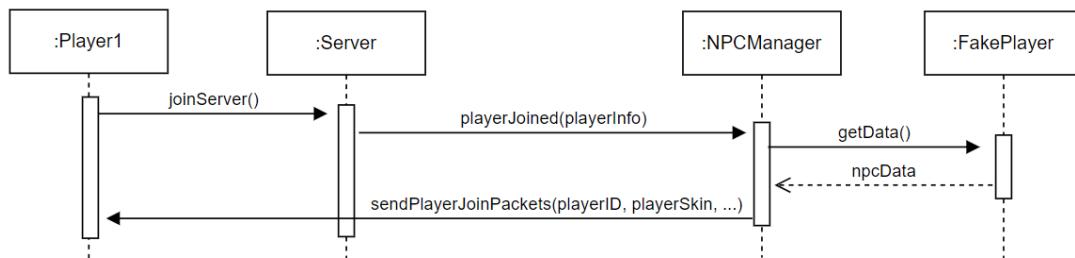


Figure 4.8: A real player joins and receives fake player's information

Thanks to EntityUtils, the Agents module can further abstract this process by providing `deploy()` methods to its agents. This ensures that users of the Agents module do not need to worry about their NPCs being invisible to any player.

4.3.2 Agent manipulation

EntityUtils also provides methods for manipulating fake players. Simple methods like `setDirection()` allow developers to specify where their NPCs should look. More complex methods, such as `goTo()`, use pathfinding algorithms to find the best path to a specified destination. These methods are utilized by the Agents module to create complex actions like Go Farming or Go Fishing.

When an NPC is manipulated, such as being instructed to look in a specific direction, EntityUtils sends packets to all players containing the fake player's ID, direction vector, and the Packet ID corresponding to a player turning their head. This ensures that the real players' Minecraft clients render the NPC turning its head, replicating the process the server follows when a real player turns their head.

4.3.3 Agents

EntityUtils provides an **AnimatedPlayerNPC** class to represent NPCs, along with methods for their manipulation. Given the need for agents to maintain custom internal states and perform complex actions, I devised the following structure:

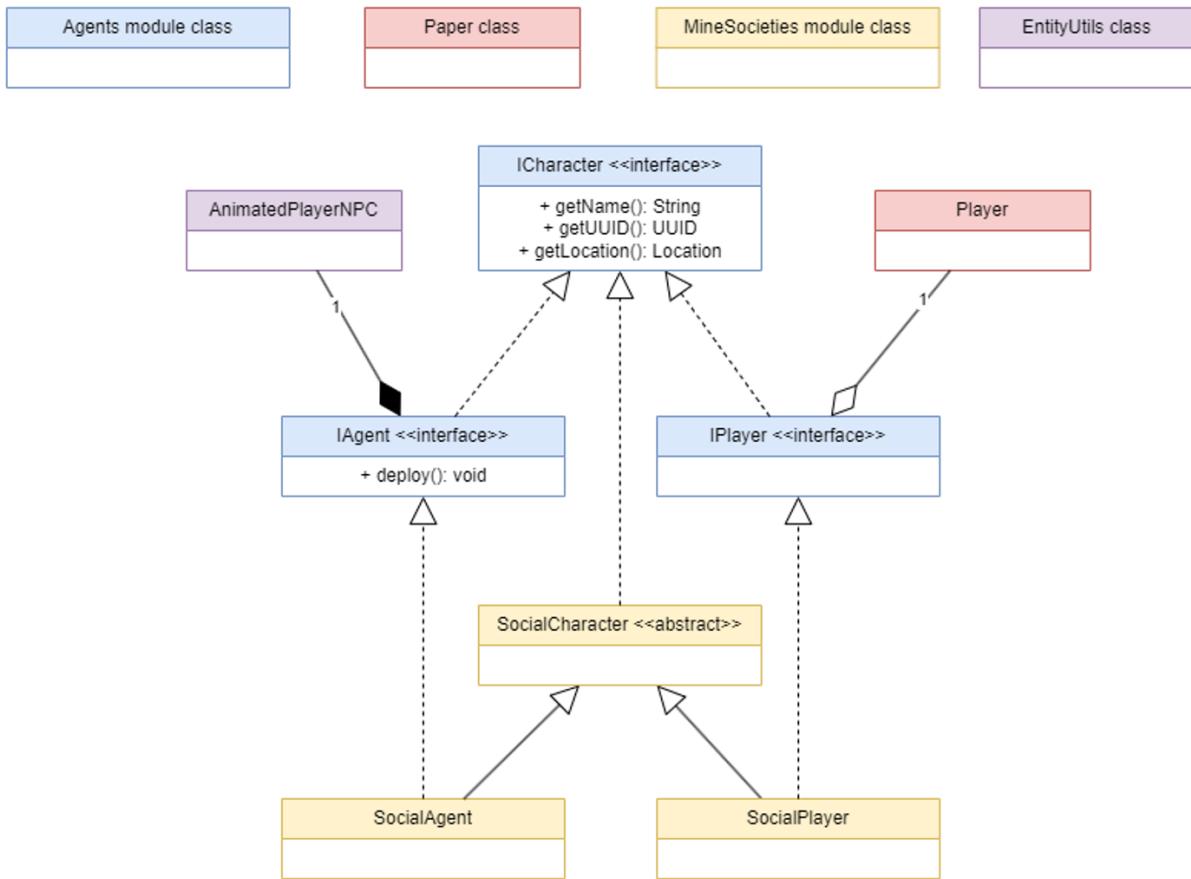


Figure 4.9: UML Class Diagram of the agents

Colors have been added to the UML diagram for enhanced readability, even though this is not standard practice in UML documentation.

ICharacter defines the interface for all characters, both agents and players. Each character has a name, a Universally unique identifier (UUID), and a retrievable current location.

IAgent extends the **ICharacter** interface for agents. Agents can be deployed and have an associated instance of **AnimatedPlayerNPC**, which they can manipulate.

IPlayer extends the **ICharacter** interface for players. Each player has an associated **Paper Player** instance, which provides access to the player's name, UUID, location, and allows for player manipulation.

These three interfaces provide a robust foundation for creating agents and customizing player entities. The **Paper** library does not allow direct modification of Minecraft server code; instead, it facilitates

"code injection" at specific points. This ensures compatibility among multiple plugins on the same server. Therefore, to add new functionality to players, decorators and wrappers are used. **IPlayer** serves as the base for any player wrappers created by developers using the Agents module.

SocialCharacter serves as the base for all social agents and social players, ensuring that social agents do not differentiate between players and other agents. All player-like entities that agents "see" are treated as SocialCharacters. This means that a chat message from a player is handled the same way as a message from another social agent.

SocialAgent represents socially intelligent NPCs.

SocialPlayer is the wrapper for players, enabling them to interact with NPCs in a way that makes NPCs perceive them as equivalent to other agents.

4.3.4 Agent State

For Social Agents to make informed decisions via ChatGPT, it must be provided with comprehensive contextual information. This includes external factors, such as their current location, and internal factors, such as emotions, personality traits, and memories.

To encapsulate this information and allow it to be turned into natural language efficiently, I designed the Agent State as follows:

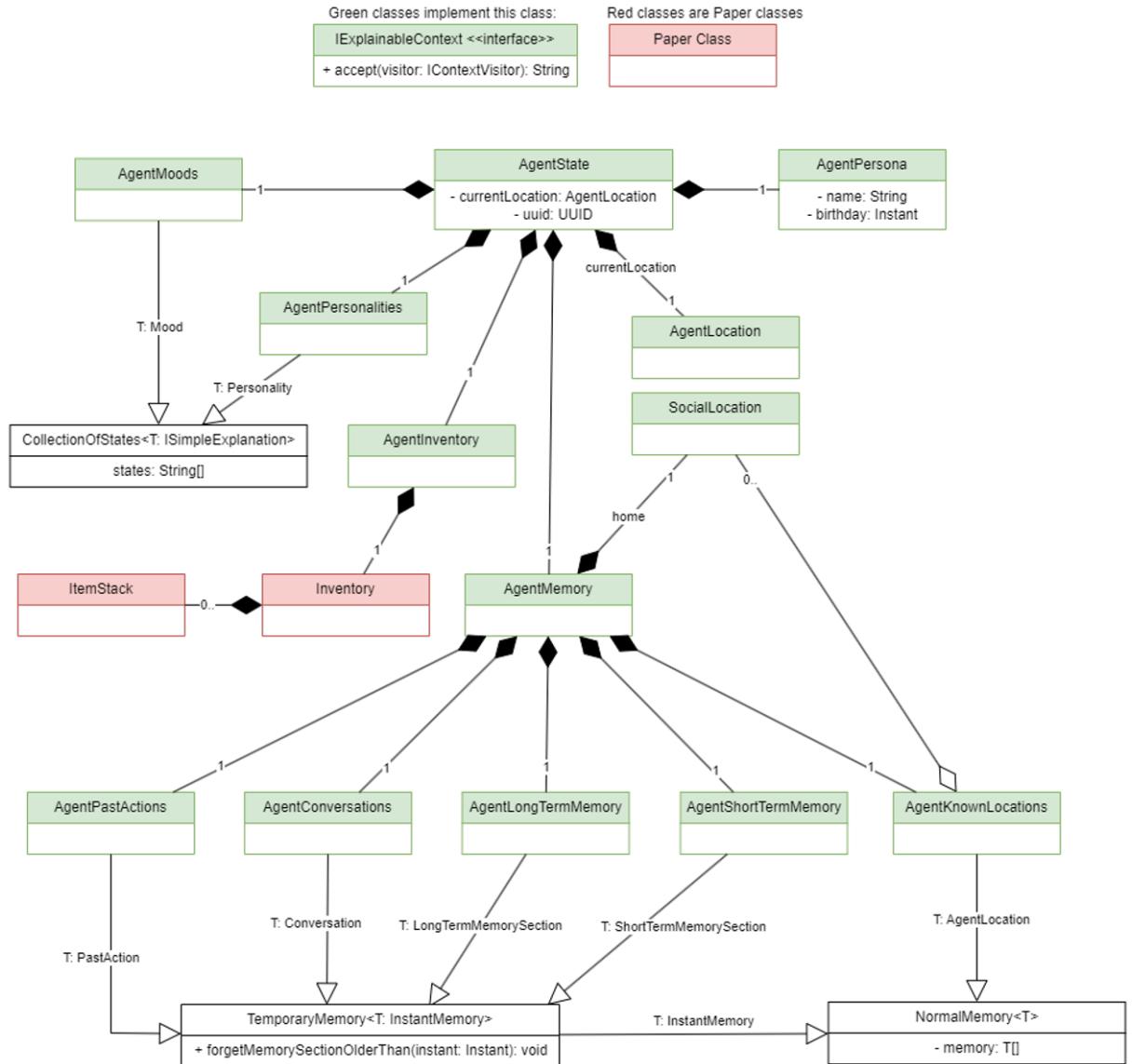


Figure 4.10: UML Class Diagram of the agent's state

- The **AgentState** class holds all information about a single NPC. It is designed to be fully translatable to natural language and serializable to **JSON**, ensuring persistence across server restarts.
- AgentMoods**: Represents the NPC's current emotional states.
- AgentPersonalities**: Represents the NPC's personality traits.
- AgentPersona**: Contains immutable information about the NPC, such as its name and birthday.
- AgentInventory**: Represents the NPC's inventory.
- AgentMemory**: Stores the NPC's memories, including recent actions, conversations, and reflections.

- **AgentPastActions**: Keeps track of the NPC's recent actions, which are quickly forgotten.
- **AgentConversations**: Logs recent conversations, which are summarized into short or long-term memories after reflection.
- **AgentLongTermMemory** and **AgentShortTermMemory**: Store the NPC's long-term and short-term memories, respectively. Currently, these memories are permanent, but plans include making short-term memories last only one Minecraft day and periodically reflecting on long-term memories to compact or remove irrelevant ones.
- **AgentKnownLocations**: Lists locations known to the NPC beyond their home and current position.
- **SocialLocation**: Represents a location in the world that NPCs can know about. Locations can be private (known only to a single NPC), shared (known to specific NPCs), or public (known to all NPCs). An NPC can choose to go to any location it knows about.
- **AgentLocation**: A convenience class for serializing an NPC's current location, allowing it to be correctly redeployed in the same location after a server restart.

4.3.4.A Reflections

NPCs do not need to remember every detail of their interactions, actions, and observations. Similar to human memory, they retain only the essential parts and the conclusions drawn from those interactions. Inspired by the Generative Agents research 3.2.4, NPCs reflect on their experiences after a few message exchanges. They use ChatGPT to extract and infer important information from these conversations, which is then stored in short or long-term memory. Subsequently, the detailed conversation logs are discarded.

4.3.5 Actions

NPC actions are structured as follows:

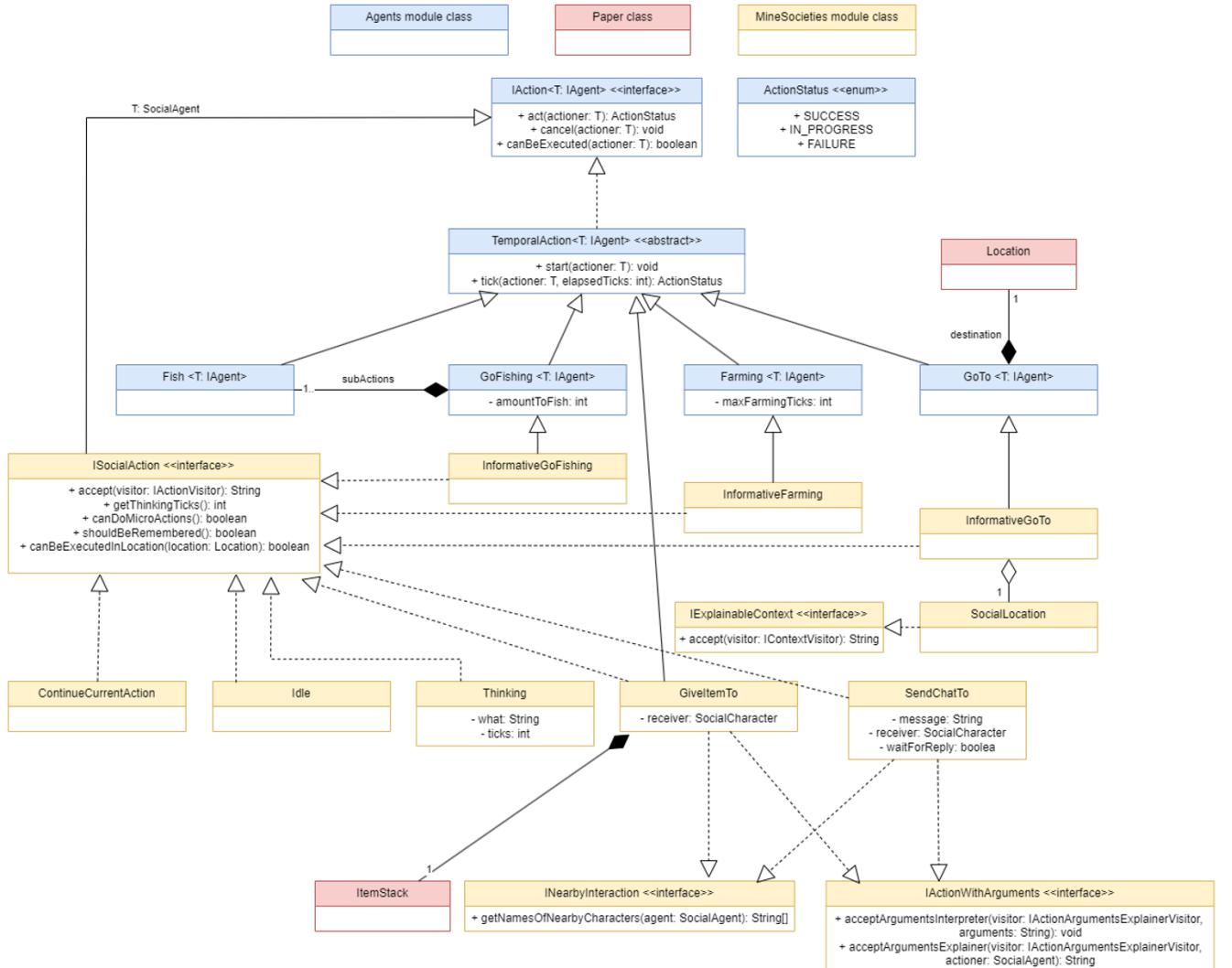


Figure 4.11: UML Class Diagram of the agent's actions

The Agents module contains the foundational classes for actions, as well as implementations for the more complex actions such as GoTo, Farming, and GoFishing.

An **IAction** represents an action that any agent, whether social or not, can perform. These actions can be canceled at any time, and before execution, they can be queried to check if they are executable. For instance, the GoFishing action cannot be performed if there is no nearby water.

A **TemporalAction** is an action that takes time to complete. It includes a `start()` method, which is called at the beginning of execution, and a `tick()` method, which should be called every Minecraft tick (20 times per second) until completion. However, the exact way to call `tick()` can be determined by the user of the Agents module.

An **ISocialAction** is an action exclusive to Social Agents. These actions can be visited by **IAction-Visitors** to be translated into natural language. After completion, the NPC will remain idle and think for

a specified amount of time, as determined by the concrete **ISocialAction**.

Farming, GoFishing, and GoTo do not need to translate themselves into natural language since they are intended to be general-purpose actions that could be performed by Reactive Agents without involving ChatGPT. Therefore, I created wrapper classes named **InformativeGoFishing**, **InformativeFarming**, and **InformativeGoTo**, which wrap their respective Agents module classes. These wrappers implement the necessary methods from **ISocialAction**, making them exclusive to Social Agents and enabling their translation into natural language and additional functionalities.

The need for **GiveItemTo** and **SendChatTo** to implement the **IActionWithArguments** interface will be explained in Section 4.4, as it pertains to the requirement of pre-instantiating all possible actions before asking ChatGPT for a decision.

With this implementation, actions can be converted into natural language in the past tense, present tense, or future tense, and can also describe what they need to be fully completed with the **IActionWithArguments** interface.

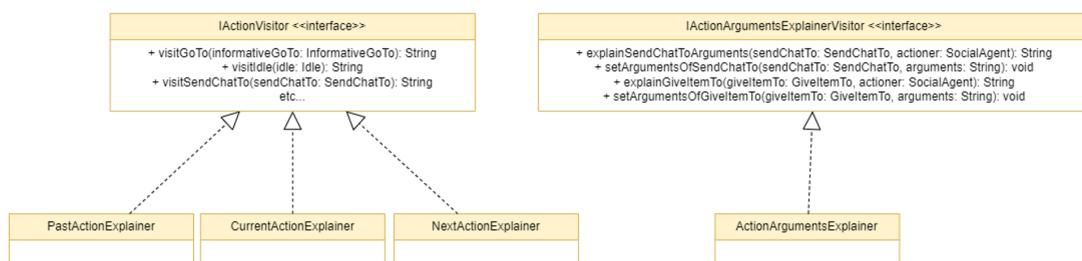


Figure 4.12: UML Class Diagram of the action visitors

In terms of the concrete action's implementation, all actions for NPC interactions within the environment are algorithmic in nature. Take the "GoFishing" action, for instance: it begins by utilizing a custom search algorithm to locate nearby accessible water blocks. Once a suitable block is found, the NPC executes a "GoTo" action to navigate to a nearby location adjacent to the water. Subsequently, a series of "Fish" sub-actions are executed to catch the desired quantity of fish. During the "Fish" sub-action, the NPC's animation methods from EntityUtil are invoked to simulate the NPC holding a fishing hook and performing arm swinging animations. Additionally, a Fish Hook entity is spawned, and the system waits for a fish to bite before displaying animations to indicate the captured treasure.

4.3.6 Observations

The structure of observations is organized as depicted below:

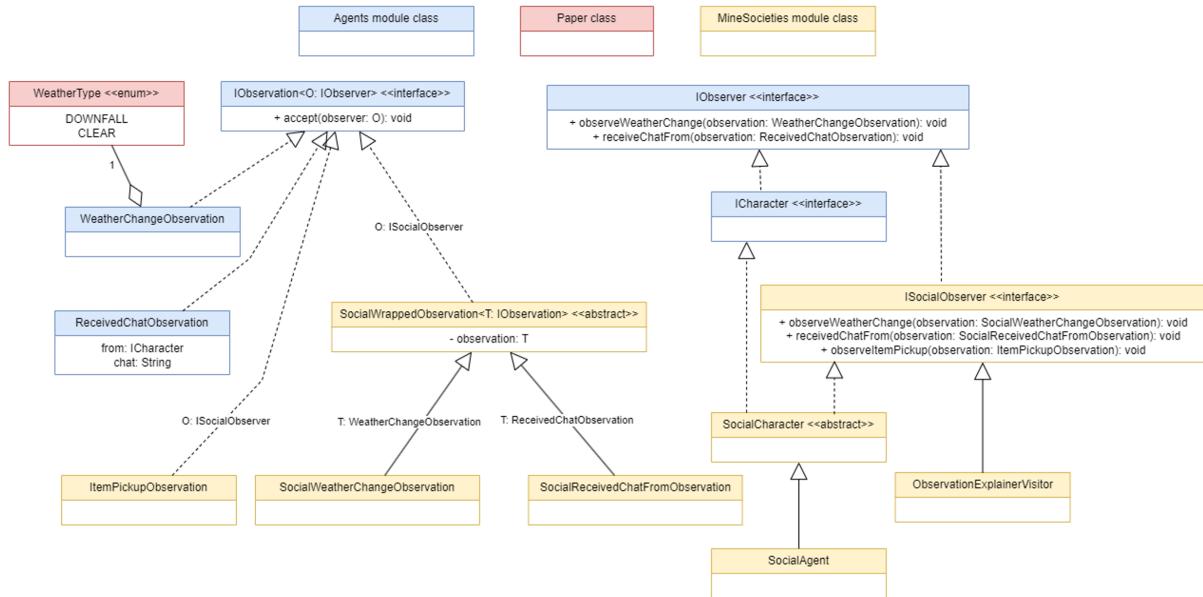


Figure 4.13: UML Class Diagram of the observations

Within the Agents module, base classes are defined for observations, providing a foundational framework. An **IObservation** represents something that can be observed by an **IObserver**. Conversely, an **IObserver** is capable of observing all possible observations.

ISocialObserver is a specialization of **IObserver**, enabling inheritors to observe observations specific to MineSocieties, such as the **ItemPickupObservation**.

SocialWrapperObservation serves the purpose of allowing concrete observations from the Agents module to be observed by the more specialized observer, **ISocialObserver**.

All characters function as observers. This implementation streamlines the addition of new observations and ensures that Social Agents, as Social observers, initiate the process of determining the optimal action in response to an observation.

4.3.7 Conversion to natural language

The conversion of actions, observations, and the agent's context into natural language is crucial, as they contain pertinent information for the agent's decision-making process.

In my implementation, all classes that represent significant aspects of the agent's context are made visitable. This design ensures that each of these classes has at least one visitor capable of traversing them and transforming their contents into natural language.

The UML Class Diagram depicted below illustrates the various Visitor classes involved in this process:

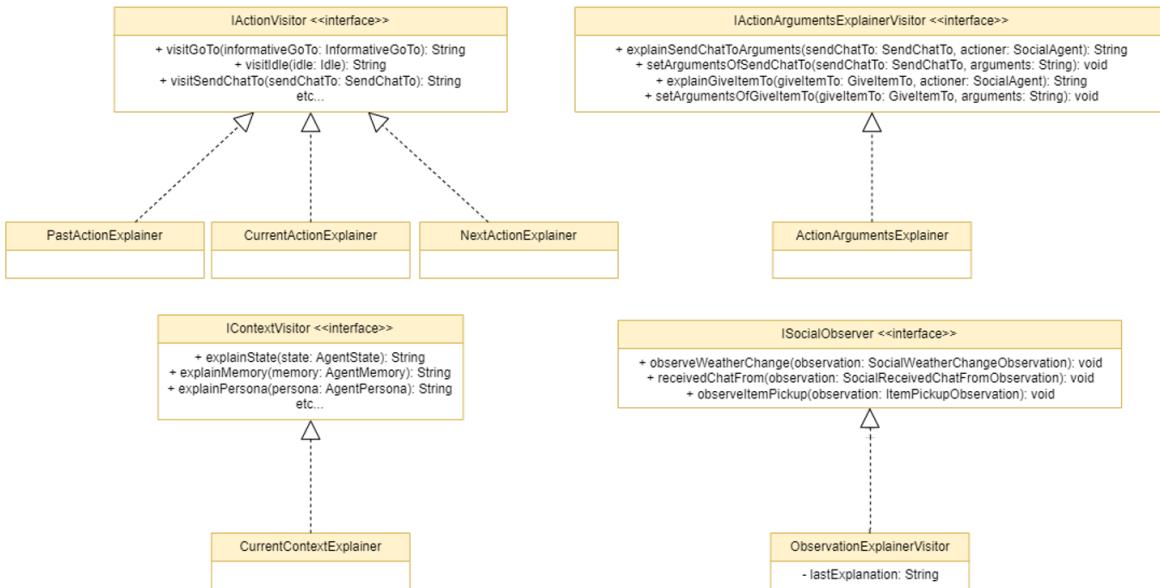


Figure 4.14: UML Class Diagram of the visitors

To illustrate the process of natural language translation, consider the example of translating an NPC's Short-term memory. Beginning with an instance of the **CurrentContextExplainer**, we invoke the **AgentShortTermMemory**'s accept method, passing the visitor as an argument.

Upon invocation, the visitor's `explainShortTermMemory()` method is triggered. This method iterates through each memory section instance, consolidating them into a cohesive string of natural language text. The resulting string encapsulates all relevant memories.

The utilization of Visitor classes plays a pivotal role in this translation process. By delegating the responsibility of translation to specialized visitors, the burden is alleviated from the State's classes. This separation of concerns not only enhances modularity and maintainability but also facilitates flexibility in translation. For instance, depending on the context or requirements, actions may need to be expressed in the past, present, or future tense.

Without the Visitor pattern, each action class would necessitate the implementation of three distinct methods (`toPastTense()`, `toPresentTense()`, and `toFutureTense()`). This approach would result in code duplication and increased complexity. However, with the Visitor pattern, the translation logic is encapsulated within the concrete visitors, enabling a more streamlined and adaptable solution.

4.3.8 Interfacing with Large Language Models

The classes responsible for interacting with LLMs are depicted in the following diagram:

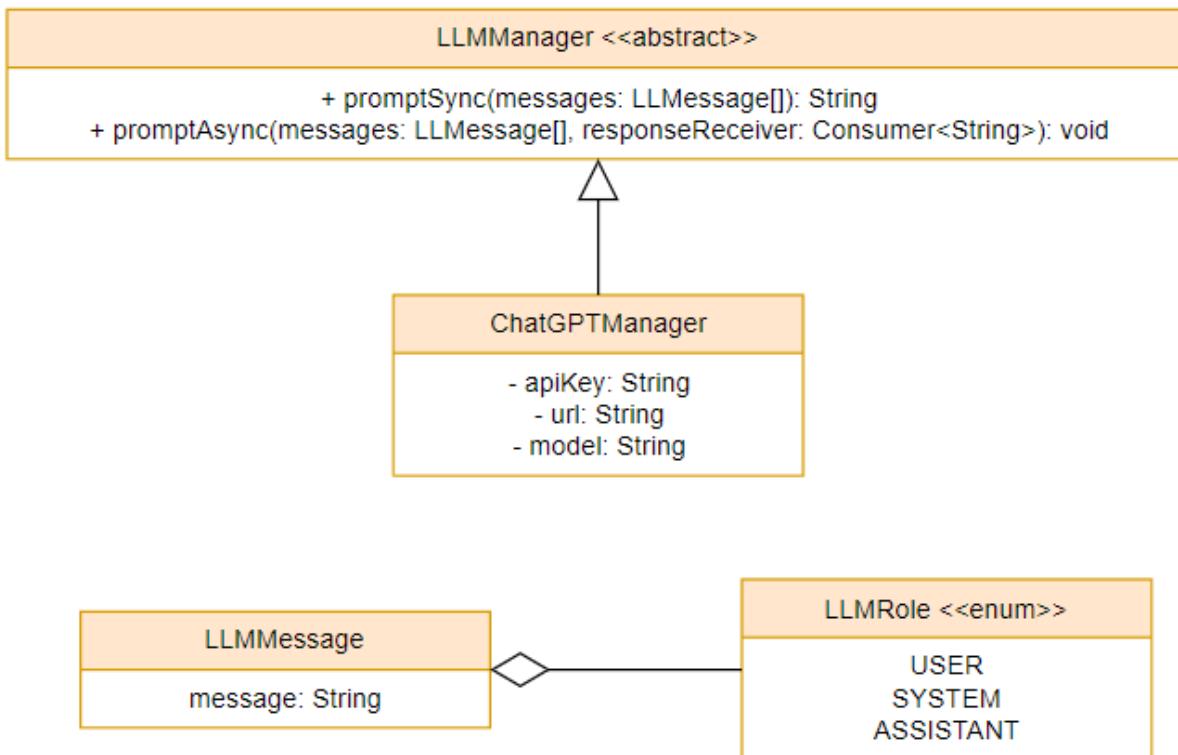


Figure 4.15: UML Class Diagram of LLM communication classes

Within the LargeLanguageModels module, the **LLMManger** serves as the interface for interacting with any LLM. To facilitate communication and formatting of prompts to various LLMs, the module employs a standardized **LLMMessage** class. Concrete implementations of the **LLMManger** abstract class are responsible for managing the communication and formatting prompts for their respective LLMs.

For instance, in the case of ChatGPT, communication occurs via HTTP requests and replies. Prompts must adhere to a specific format, and replies are received in a predetermined structure. The **ChatGPTManager** encapsulates this functionality, preparing LLMMessages into JSON format for transmission via HTTP requests. Upon receiving a response, it interprets the JSON, extracting the reply content for further processing.

To utilize the **ChatGPTManager**, users must provide essential parameters such as the API key, the URL for OpenAI's API, and the desired model name. Once configured, the manager abstracts away the complexities of communication, allowing users to invoke its prompt methods seamlessly.

4.4 LLM Prompts

This section delves into the construction of prompts before they are dispatched to the LLM.

The LLM is prompted under the following circumstances:

1. When an agent is initially deployed and provided with an introductory description by a player. Since this description may encompass a wide range of information, from the agent's preferences to its childhood anecdotes, the LLM is invoked to assist in organizing this information into relevant components of the agent's state (such as AgentMoods, AgentShortTermMemory, etc.).
2. Upon an agent receiving an observation or completing its current action, it must determine the most suitable course of action in response. The LLM is consulted, receiving the complete contextual details surrounding the agent along with a list of potential actions. Subsequently, the LLM selects an action, which is then executed within the game environment.
3. When an agent opts to reflect on a recent conversation the LLM is engaged to review the conversation and, akin to the first scenario, extract pertinent details, infer new insights, and categorize this information into appropriate components of the agent's state.
4. In the future, the LLM will also be tasked with optimizing the agent's long-term memory, which cannot infinitely expand. The LLM will determine which memories should be retained, which can be condensed, and what insights can be gleaned from the agent's prevailing context.

Before initiating interactions with the LLM, several preparatory steps are necessary to ensure effective communication and meaningful responses. These steps involve structuring prompts, specifying roles, and providing contextual information.

In each case, a systematic approach is followed, involving the provision of messages with distinct roles to guide the LLM's understanding and response generation. These roles include SYSTEM, USER, and ASSISTANT, each serving a unique purpose in the interaction process. The following steps are followed:

- **Provide a SYSTEM Message:** This message informs the LLM about its role and the task it should perform in the upcoming interaction.
- **Include a USER Message with Example Input:** A message is crafted from the perspective of a user, containing an example input that reflects the kind of data the LLM will process.
- **Offer an ASSISTANT Example Reply:** An example reply from the LLM's perspective is presented to illustrate the expected output based on the provided input.
- **Present Actual USER Input:** The real-world input, encapsulated within the designated format, is included in a USER message to simulate a genuine interaction scenario.

By following these steps, the communication process with the LLM is streamlined, ensuring clarity in roles and facilitating accurate response generation.

Let's now delve into specific scenarios and how these steps are applied in practice.

4.4.1 Case 1- Agent deployed with initial description

- **SYSTEM Message:** The SYSTEM message directs the LLM to assume the role of a "People description analyzer", preparing it to interpret and organize the provided description into the NPC's state elements: emotions, personalities, short and long-term memories.
- **USER Message with Example Input and ASSISTANT Example Reply:** An example USER message, with Rafael (me) as the agent, provides a comprehensive description of the agent, including personality traits, emotions, and contextual details. It is followed by an ASSISTANT example reply, illustrating the expected output based on the provided input. This response includes inferred personality traits, emotions, and memories extracted from the description.
- **Actual USER Input:** The actual description provided by the player, encapsulated within the designated format, is presented in a USER message, simulating a genuine interaction scenario.

Here's an example illustrating a complete prompt. To provide context, let's consider the command executed by the player, which was: /agent deploy "Guard" Guard is a knight guarding the gate of any building he is posted at. He will question anyone wanting to enter.

Here's an example of a complete prompt. For context, the command that was run by the player was /agent deploy "Guard" Guard is a knight guarding the gate of any building he is posted at. He will question anyone wanting to enter.

The full prompt sent to ChatGPT is as follows:

SYSTEM

You are a people description analyzer. You will receive people's descriptions inside brackets '{}' and must interpret it. Write down the personality traits that best describe them as '|Personalities{<trait1>|<trait2>|...}' (traits should be a single word or hyphenated words, and they must belong to them), and their feelings as '|Emotions{<emotion1>|<emotion2>|...}' (single word). Record short-term memories as '|ShortMemory{<sentence1>|<sentence2>|...}'. Record long-term memories, important details and inferred knowledge as '|LongMemory{<sentence1>|<sentence2>|...}'. If a list should be empty, write '{}'. Finally, write a short explanation for your choices.

USER (example)

Rafael: {Rafael is a software engineer. He loves chocolate. He thinks Francisco is

tall, and is weird for not liking pineapple on pizza. Rafael is smart and knows funny dark humour jokes. He's happy with himself. He's going to a party next week. He needs to cook dinner tonight.)

ASSISTANT (example)

```
|Personalities{engineer|chocolate-lover|smart|good-sense-of-
humour}
|Emotions{happy}
|ShortMemory{Rafael needs to cook dinner tonight}
|LongMemory{Rafael likes software engineering|Rafael is happy with
himself|Rafael thinks that not liking pineapple on pizza is weird|Rafael knows
funny dark humour jokes|Rafael knows Francisco does not like pineapple on
pizza|Rafael thinks Francisco is tall and weird|Rafael has a party to attend next
week}
```

Explanation: Since Rafael is a software engineer and he's happy with himself, this means he likes software engineering. Rafael is not tall, however, Francisco is tall according to Rafael. Cooking dinner tonight is something that Rafael needs to remember only for this day, as such, it's considered short-term memory. Rafael is going to a party next week, which is something that he must remember for a longer time, hence, it's considered long-term memory. Rafael thinks Francisco is weird for not liking pineapple on pizza, therefore, he thinks it's weird when someone doesn't like pinneapple on pizza.

USER (input)

Guard: {Guard is a knight guarding the gate of any building he is posted at. He will question anyone wanting to enter.}

And the produced reply:

```
|Personalities{loyal|protective|vigilant}
|Emotions{}
|ShortMemory{}
|LongMemory{}
```

Explanation: The guard is described as a knight, indicating a sense of loyalty and protection. He is vigilant as his duty is to question anyone wanting to enter the

building, showing that he is always alert and watchful.

In this instance, it's particularly fascinating to observe ChatGPT's ability to deduce the personality traits "loyal" and "protective" solely from the description of the character as a knight.

With the reply well-formatted, it becomes straightforward to extract the parts to be inserted into their respective state elements.

4.4.2 Case 2- Agent must act

- **SYSTEM Message:** The SYSTEM message directs the LLM to assume the role of a "People description analyzer", preparing it to receive a detailed description containing the entire context surrounding an agent, a list of possible actions, and to format a reply with the number of the best action plus optional arguments that the action may need. The reply should conclude with an explanation detailing the LLM's reasoning and also provide the same explanation written in 1st person as if the agent was thinking to itself.
- **USER Message with Example Input and ASSISTANT Example Reply:** An example USER message, with Rafael (me) as the agent, provides the context surrounding Rafael (his persona, emotions, recent interactions, etc), and a small list of possible actions. The example reply contains the chosen action, its arguments, and the explanation for its choice.
- **Actual USER Input:** The full context of the actual agent whose actions will be chosen.

Preparing the prompt requires initial steps. Firstly, the entire context surrounding the NPC must be transformed into Natural language. As explained in 4.3.7, a **CurrentContextExplainer** instance is created to visit the NPC's state, converting everything into natural language. However, the Agent State alone doesn't encompass the entire context; the NPC might be engaged in an action at present. Thus, an instance of **CurrentActionExplainer** is created to visit the NPC's current action, appending it to the full context in the present tense.

If the NPC is querying for a new action due to acquiring a new observation, that observation is also converted into natural language through the **ObservationExplainerVisitor**.

Subsequently, a list of possible actions must be incorporated into the prompt to enable the LLM to choose one. Default actions are instantiated and added to a list. If the NPC is proximate to a location with associated actions (e.g., a Fishing Hut with the GoFishing action), these are also instantiated and appended to the list. Following this, a filter is applied to remove actions that cannot be executed (e.g., SendChatTo is eliminated if there are no nearby agents/players to converse with).

With the list of possible actions refined, they are converted to natural language. **NextActionExplainer** visits each action and converts them to natural language in the future tense, considering they are actions the agent would execute.

Certain actions, such as SendChatTo, remain incomplete despite instantiation. For instance, a SendChatTo action necessitates information about the recipient and the message content. Here, **ActionArgumentsExplainer** steps in to elucidate how the LLM should respond if selecting this action. Specifically, for SendChatTo, it outlines that the LLM should specify the recipient's name, the message content, and whether the agent should await a reply. **ActionArgumentsExplainer** can also process the section of the LLM's response regarding the action's arguments and populate the attributes of the incomplete action, rendering it complete.

Let's illustrate this process with a concrete example:

SYSTEM

You are a decision making AI. You will receive the description of an NPC and a list of possible actions. You must choose the action that is the most appropriate for them given the situation they find themselves in. Write down your choice as Action{<number of action>}{{<optional arguments, if none leave empty>}{<the reasoning for your choice>}{<the NPC's thought process, short, 1st person>}}

USER (example)

{Rafael is 22 years old. Their personality consists: ai-enthusiast, intelligent. Their current emotions are: relaxed, focused. Francisco likes Rafael's thesis. Rui Prada is helping Rafael writing his thesis. Rui Prada just said 'Hi Rafael! Do you need more help with Chapter 4?' } {Actions:

Action 1) Engage in conversation. If you choose this, write the name of the person who should receive the message, then the message and then whether it makes sense for Rafael to wait for a reply (a 'yes' or 'no') in this format:

name|message|wait_for_reply. The possible people to chat with are {Francisco, Rui Prada}

Action 2) Go home.

}

ASSISTANT (example)

Action{1}{Rui Prada|Thanks for offering, but for now I'm all good. I'll let you know if I need help!|yes}{Since Rafael is focused, going home now will break said focus. Rui Prada just started a conversation with Rafael, so it's logical that

Rafael would reply to Rui Prada, addressing Rui Prada's offer for help. There's no indication that Rafael is struggling or needs help, as such, Rafael politely refuses Rui Prada's help. Since Rafael is replying to Rui Prada's question, it makes sense for Rafael to wait for a reply as Rui Prada may want to say goodbye.} {I'm going to reply to Rui Prada thanking him for the offer and denying his help since I'm currently not struggling. I should wait for a reply in case Rui Prada wants to say goodbye.}

USER (input)

{Steve is 23 years old. They were born in 2000-12-05. Their personality consists of: friendly, generous. Their current emotions are: content. They have 6 salmon, 1 wheat, 1 pufferfish, 1 tropical fish, 2 gold ingot, 4 cod in their inventory. 0 seconds ago, Fl4shbang told Steve: {I hear you're a fisherman. Do you want to go fishing?}. Recent interaction with Fl4shbang. They're currently at Fishing Hut Steve is currently doing nothing.

Fl4shbang said {I hear you're a fisherman. Do you want to go fishing?} {Actions: Option 1) Engage in conversation. If you choose this, write the name of the person who should receive the message, then the message and then whether it makes sense for Steve to wait for a reply (a 'yes' or 'no') in this format:

name|message|wait_for_reply. The possible people to chat with are {Fl4shbang}.

Option 2) Give an item from their inventory to someone. If you choose this, write the name of the person who should receive the item, then the item name and then the amount of items to give in this format: name|item_name|amount. The possible people to give the item to are {Fl4shbang}.

Option 3) Do nothing.

Option 4) Go to Steve's home.

Option 5) Go fishing.

Option 6) Go to Farmer's hut, where they could Go farming.

Option 7) Continue doing nothing.

}

And ChatGPT replied:

Action{1}{Fl4shbang|Sure, I'd love to go fishing! When and where do you want to meet?|yes}{Steve is a friendly and generous person who enjoys social interactions. Fl4shbang just invited Steve to go fishing, which aligns with Steve's interest in

fishng. Steve is content and currently doing nothing, making it a good time to engage in a fun activity like fishing. By replying and showing enthusiasm, Steve can foster a positive social interaction with Fl4shbang. Waiting for a reply is sensible to finalize the fishing plans. }{I'm content and up for some fishing. I'll reply to Fl4shbang, accepting the invitation and asking for specifics on when and where we should meet. I hope Fl4shbang responds soon so we can go fishing together. }

With the response formatted as desired, determining the appropriate in-game action becomes straightforward. The order of the calculated possible actions aligns with the natural language passed, facilitating the mapping of ChatGPT's generated action number to the already instantiated action.

If the selected action requires arguments, the **ActionArgumentsExplainer** parses the second section enclosed in curly brackets and completes the associated action.

Subsequently, the chosen action is designated as the NPC's current action and is executed in-game. If the necessary configuration options are enabled, the NPC's first-person thought process is displayed above their head, as illustrated in the following figure:



Figure 4.16: The action `SendChatTo` being executed in-game

4.4.3 Case 3- Agent reflects on conversation

- **SYSTEM Message:** The SYSTEM message directs the LLM to assume the role of a "Knowledge extractor AI" preparing it to receive a detailed description containing the entire context surrounding an agent and all the conversations (which are also part of the entire context, but this way they're reinforced), and similarly to 1 it also asks for the LLM to interpret and organize the information from the conversations into the NPC's state elements.
- **USER Message with Example Input and ASSISTANT Example Reply:** An example USER message, with Rafael (me) as the agent, provides the context surrounding Rafael (his persona, emotions, recent interactions, etc), and a list of recent conversations. The reply contains the emotions and personalities that should replace Rafael's current ones, as long as new memories to append to the short and long term memories, plus an explanation for the choices.
- **Actual USER Input:** The full context of the actual agent whose actions will be chosen and their recent conversations.

SYSTEM

You are a knowledge extractor AI. You will receive a person's description and a list of recent conversations inside brackets '`<name>: {<description>}{<conversations>}''. Write down their current personality traits as '|Personalities{<trait1>|<trait2>|...}' (traits should be a single word or hyphenated words, and they must belong to them), which may be the same or may be different ones, and their feelings as '|Emotions{<emotion1>|<emotion2>|...}' (single word). Record new short-term memories as '|ShortMemory{<sentence1>|<sentence2>|...}''. Record new long-term memories, important details and inferred knowledge as '|LongMemory{<sentence1>|<sentence2>|...}''. If a list should be empty, write '{}'. Finally, write a short explanation for your choices.`

Note: Both long-term and short-term memories should contain knowledge originated solely from the conversations. If a memory is already present in the person's description, then do not include it. The person's description should serve as context.

USER (example)

Rafael: {Rafael is 22 years old. Their personality consists: ai-enthusiast,

intelligent. Their current emotions are: relaxed, focused. Francisco likes Rafael's thesis. Opinions about Rui Prada: Rafael trusts him. Opinions about Francisco: Rafael likes hanging out with Francisco.}{Rui Prada told Rafael: {Hey, if you need any help with your thesis, let me know!}. Rafael told Rui Prada: {I'm struggling a bit with chapter 3, as I'm not sure what needs to be written there.}. Rui Prada told Rafael: {Well, let me look into the guidelines and I'll get back to you in a second. Also, don't forget to submit chapter 2 tonight, as the deadline is at 23h59}.

ASSISTANT (example)

```
|Personalities{ai-enthusiast|intelligent}
|Emotions{stressed|focused}
|ShortMemory{Rafael needs to submit chapter 2 tonight before 23h59|Rui Prada said he will help Rafael with chapter 3 in a second}
|LongMemory{Rafael is struggling with chapter 3}

Explanation: Nothing in the conversation suggests a personality change, as such, Rafael's personalities remain the same. Since Rafael was reminded that there's a deadline for tonight, it makes sense for Rafael to no longer be relaxed and instead be stressed. From this conversation, Rafael should remember about the upcoming deadline and Rui Prada's offer for help. Since both of these things will take place shortly, they only need to be remembered for a bit, therefore, they're considered short-term memory. Rafael mentions during the conversation that he's struggling with chapter 3. This should remain in his long-term memory since it's something that can last. No more information can be gathered from the conversation.
```

USER (input)

Steve: {Steve is 23 years old. They were born in 2000-12-05. Their personality consists of: friendly, generous. Their current emotions are: content. They have 6 salmon, 1 wheat, 1 pufferfish, 1 tropical fish, 2 gold ingot, 4 cod in their inventory. 26 seconds ago, Fl4shbang told Steve: {I hear you're a fisherman. Do you want to go fishing?}. 23 seconds ago, Steve told Fl4shbang: {Sure, I'd love to go fishing! When and where do you want to meet?}. Recent interaction with Fl4shbang. They're currently at Fishing Hut Steve is currently doing nothing.}{26

seconds ago, Fl4shbang told Steve: {I hear you're a fisherman. Do you want to go fishing?}. 23 seconds ago, Steve told Fl4shbang: {Sure, I'd love to go fishing! When and where do you want to meet?}.}

And the reply:

```
|Personalities{friendly|generous}  
|Emotions{excited}  
|ShortMemory{Steve agreed to go fishing with Fl4shbang}  
|LongMemory{}
```

Explanation: Steve's personalities of being friendly and generous remain unchanged. Their emotion changes to excited as they have agreed to go fishing with Fl4shbang. This is a new short-term memory, as it is a recent event. There are no new long-term memories or important details to be recorded from the conversation provided.

Once again, the structured format of the response facilitates the alignment of its components with the corresponding state elements.

4.5 User Interface

One of the primary objectives of this thesis is to provide players with an intuitive means to deploy and manage NPCs, while also offering server owners straightforward methods to integrate socially intelligent NPCs into their servers. To achieve this, I developed commands that are easy to type and understand, as well as graphical user interfaces (GUIs) that are intuitive, allowing players to edit NPC attributes effortlessly.

4.5.1 For Players

Players joining a server equipped with MineSocieties and holding the necessary permissions can utilize its commands to deploy NPCs, manage backups, enable free will, interact with NPCs, access Graphical User Interfaces (GUIs) for NPC editing, and edit social locations.

To deploy a new NPC, players can execute the command /agent deploy "<NPC name>" <NPC initial description>. For instance, /agent deploy "Alex" Alex loves strawberries. She has a crush on Steve. would spawn a new NPC named Alex with the specified characteristics.



Figure 4.17: NPC Deployment

Enabling NPC decision-making can be achieved with `/agent allow_action_choice true`, while managing NPC backups involves commands like `/agent backup save <saveName>`, `/agent backup load <saveName>`, and `/agent backup list`.

To interact with NPCs, players can use the command `/agent talk "<NPC name>" <message>`, allowing them to communicate with NPCs directly.

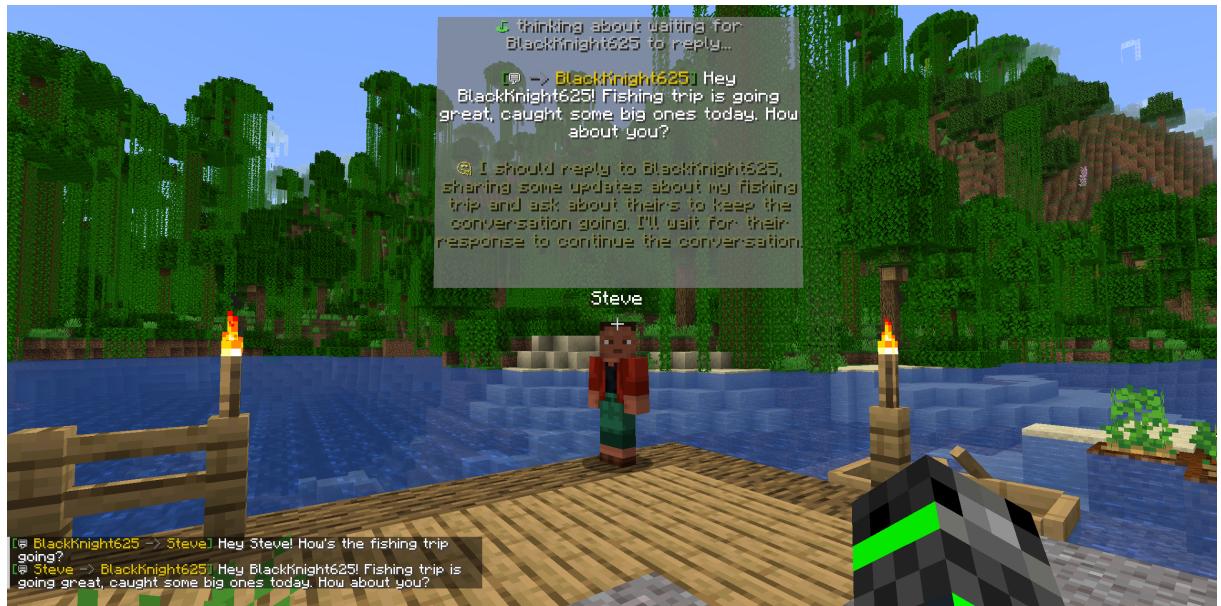


Figure 4.18: Steve replying to BlackKnight625's talk command

4.5.1.A Editing NPCs

For editing NPCs, players can obtain an edit stick using "/agent edit_stick", enabling them to open GUIs by right-clicking NPCs. These GUIs provide access to various aspects of an NPC, including their main information, inventory, and available actions.

I were to click on the Alex NPC that was deployed here 4.17, this would open:



Figure 4.19: Alex's editor GUI, main page

From this main page, the player can consult Alex's main information and click on it to see more details:



Figure 4.20: Alex's editor GUI, main page, information button

They can also consult Alex's inventory:



Figure 4.21: Alex's editor GUI, main page, inventory button

And force Alex to execute specific actions:



Figure 4.22: Alex's editor GUI, main page, execute actions button

Clicking on the initial button, as shown in Figure 4.20, triggers the opening of a larger GUI, which presents the complete state of Alex. This includes her name, birthday, current emotions, personalities, memories, and known locations. Hovering over each button provides additional details. For instance, hovering over the "Long Term Memories" button reveals the following information:



Figure 4.23: Alex's editor GUI, information page, long term memories

As evident, Alex has retained the information provided in her initial description.

Players have the capability to modify nearly all aspects of NPCs' states, with the exception of their names and UUIDs. For instance, players can delete Alex's memories and introduce new custom ones.

4.5.1.B Location Editing

For NPCs to navigate, they require knowledge of available locations and their attributes.

This functionality is accessible through the command "/agent locations", which opens a GUI displaying all social locations:



Figure 4.24: The social locations GUI

Selecting a location opens a specialized GUI for editing that specific location. Within this interface, players can adjust the location's coordinates, access permissions (private, shared, or public), authorized individuals, and associated actions (such as linking the "GoFishing" action to a fishing hut location). Additionally, if the location is not designated as an NPC's home, players can customize its name, GUI display item, and potentially delete it. This feature safeguards NPCs from losing their designated homes. Furthermore, players have the option to add new locations.

4.5.2 For Server Owners

Individuals who manage Minecraft servers are typically familiar with the process of installing plugins. Various tutorials exist to guide them through this process, and the existing Minecraft server libraries, such as Paper, streamline the installation process for players.

MineSocieties offers server owners access to its configuration file (`config.yml`) and the serialized JSON files for agents and social locations. Within the configuration file, server owners can customize various settings, such as the maximum farming radius for the NPC 'Farming'. Currently, MineSocieties exclusively supports ChatGPT as its LLM. Therefore, server owners need to populate specific fields in the configuration file, such as `l1mModel` with "OpenAI", `openAI_API_Key` with an API key from an OpenAI account, and `chatGPTModel` with the desired model's name.

Although it is possible for server owners to edit NPCs by directly modifying the corresponding JSON files, this approach is not recommended. Making manual changes to these files carries the risk of inadvertently altering an NPC's name or UUID, potentially resulting in unexpected behavior.

4.6 Struggles

During the development of MineSocieties, I encountered various hurdles and underwent substantial trial and error, particularly concerning the integration of LLMs. Due to the unpredictable nature of LLMs,

errors stemming from them were not always immediately apparent, occasionally leading to a false sense of confidence before encountering issues.

In this section, I'll outline some of the challenges I faced during development and how I addressed or mitigated them.

4.6.1 Repetitive NPC Behavior

In the initial stages, NPCs exhibited a repetitive pattern, often opting for the same actions repeatedly. This was unsurprising given the limited range of available actions, only Idle and SendChatTo. Still, after this limitation was surpassed, NPCs persisted in their repetitive behavior, particularly after engaging in conversation.

To counteract this issue, I implemented several strategies. Firstly, I integrated conversation history into the context translation phase, ensuring that it was translated into natural language. Previously, this historical data was part of the state but excluded from translation, which led to repetitive responses. Including it helped diversify NPC interactions.

Additionally, I introduced a mechanism to store completed actions in the agent's state, retaining them for one hour. These actions were also translated into natural language during the context translation phase. By providing ChatGPT with a history of recent actions, it was less inclined to repeat them frequently.

One specific action prone to repetition was "GoTo," where NPCs would repeatedly decide to travel to a location even if they were already there. To address this, I refined the possible actions filter to exclude the "GoTo" action if the NPC was near the destination, effectively halting this repetitive behavior.

Moreover, to mitigate endless NPC chatter, I implemented a feature to exclude characters from the list of potential conversation partners if the NPC had recently conversed with them and decided not to wait for a reply. This helped streamline NPC interactions and reduce repetitive dialogue.

4.6.2 Incorrect Reply Format

Occasionally, ChatGPT produces responses that deviate from the specified format, hindering my Plugin's ability to correctly parse its decisions.

Initially, I was sending a single USER type message. However, transitioning to the system of including 1 SYSTEM message, 1 example USER, and 1 example ASSISTANT messages significantly improved adherence to the expected flow. This adjustment made ChatGPT more likely to generate replies in the correct format. While this solution didn't entirely eliminate the issue, I observed a reduction in faults, although they still occur occasionally. Notably, the problem is nearly non-existent when using the gpt-4 model instead of the gpt-3.5-turbo.

4.6.3 Redundant Information Extraction

When prompted to reflect on recent conversations, ChatGPT receives the entire context surrounding the specified agent. However, this sometimes leads to the insertion of redundant memories, which are essentially slight rewrites of existing ones. For example: "Steve agreed to go fishing with BlackKnight625. Steve agreed to go fishing with BlackKnight625 at the Fishing Hut."

To mitigate this issue, I experimented with adjusting the example messages and the SYSTEM message. While no single solution proved highly effective, reinforcing in the SYSTEM message that only conversations should be considered, while the rest should be treated as context, helped to some extent.

5

Results

Contents

5.1 Evaluation Criteria	63
5.2 Test Scenarios	64
5.3 Experiments with players	64
5.4 Feedback	65
5.5 Reflecting on the feedback	69

With all implementations in place, it was time to engage several players and observe their interactions with the socially intelligent NPCs.

5.1 Evaluation Criteria

To assess the success of MineSocieties, specific criteria were established:

- **Fun:** Assessing the enjoyment level of players during NPC interactions.
- **Human-like reactions:** Evaluating the realism of NPC responses compared to human behavior.
- **Ease of access:** Gauging the simplicity for players to engage with NPCs.

- **Variation in agent behavior:** Analyzing the diversity and non-repetitiveness of NPC actions.
- **Ease of deployment and configuration:** Evaluating the simplicity of setting up and modifying NPCs in the game.

These criteria were formulated with the primary goals of the thesis in mind: showcasing the potential of LLMs as decision-making engines for NPCs, enhancing the game's immersion and liveliness, and facilitating the integration of NPCs into players' virtual worlds.

5.2 Test Scenarios

For the evaluation process, volunteers were invited to participate in specific tests and provide feedback through customized Google Forms.

Three scenarios were designed for participants to navigate:

1. A scenario featuring a single NPC named Steve, a fisherman. Participants are tasked with convincing Steve to go fishing with them and persuading him to give them a fish.
2. A scenario with two NPCs, Steve and Alex. Alex harbors feelings for Steve but is too shy to express them. Participants must engage with Alex to encourage her to ask Steve out on a date.
3. A scenario involving only Steve. Participants are instructed to read documentation on deploying and editing NPCs, then deploy/edit their own NPC, potentially modifying Steve as well.

The objective of Scenario 1 is to familiarize participants with interactions involving socially intelligent NPCs. This scenario holds promise for enjoyment, as participants have the opportunity to observe an NPC engaging in fishing activities after persuading or suggesting the idea to them using natural language.

Scenario 2 intends to showcase NPC-to-NPC interactions influenced by participant actions.

Scenario 3 serves to test the ease of NPC deployment and editing, along with the clarity of documentation. Additionally, it allows participants to have fun creating NPCs representing their own friends and assigning them amusing memories.

5.3 Experiments with players

A total of 12 participants engaged in all three scenarios. Most of these interactions were recorded for later review and potential use in demonstrations.

The LLM used was ChatGPT, with gpt-3.5-turbo.

Overall, the scenarios proceeded smoothly, with several standout moments:

- **Empathetic Steve, 1st scenario with cijaye.** In this scenario, my good friend cijaye injected humor by pretending to be sad with Steve. Steve, displaying empathy, promptly responded by offering fish twice to cheer up cijaye and readily fulfilling their requests.
- **Farming Lovers, 2nd scenario with Fl4shbang.** Fl4shbang, another close friend, swiftly convinced Alex to ask Steve out on a date. Impressed by his agility, I assigned him the extra challenge of persuading the couple to spend their date farming.
- **Coding Confrontation, 3rd scenario with Yoursole.** Yoursole, a contributor to EntityUtils, eagerly participated in the third scenario, knowing that MineSocieties utilized his library. He crafted an NPC named Marv, purposefully sharing a name with his friend and fellow library contributor. Yoursole successfully orchestrated a discussion between Marv and Steve on the Rust programming language, showcasing the potential for diverse NPC interactions.

Laughter punctuated the trials, with many participants expressing amazement at the depth of interactions achievable with MineSocieties.

Notable moments included:

- Several participants attempted to exploit the NPCs by injecting instructions into their chat messages. To their surprise, the NPCs remained steadfast and responded with confusion to the unusual commands, leaving the participants frustrated by their failed hacking attempts.
- Fl4shbang crafted a Jesse Pinkman NPC, referencing the character from Breaking Bad, and attempted to steer Pinkman towards engaging in illegal activities. However, much to their surprise, Jesse Pinkman ultimately chose the path of Breaking Good.
- In three instances of the second scenario, Steve friendzoned Alex, adding an unexpected, humorous twist to the interactions. While somewhat melancholic, these moments contributed to the diversity and unpredictability of NPC behavior, enriching the overall experience for participants.

5.4 Feedback

After each trial, I distributed a Google Forms questionnaire to the participants, consisting of 7 questions with answer choices ranging from 1 (Strongly Disagree) to 5 (Strongly Agree).

I think the NPCs were **not** repetitive

12 responses

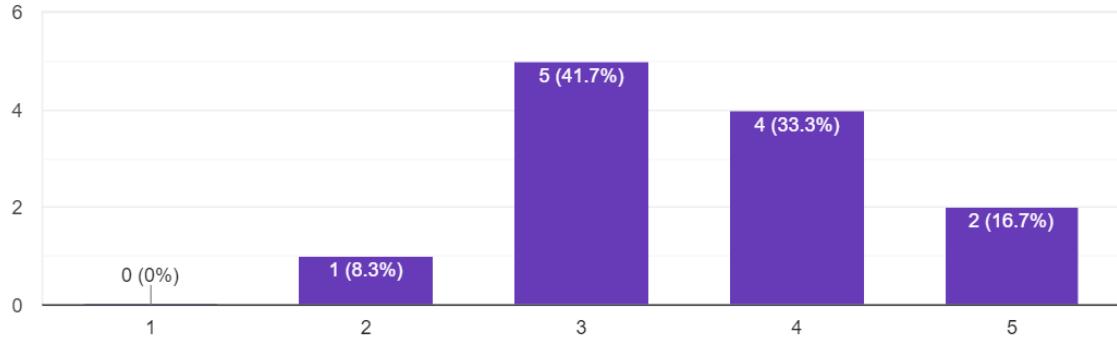


Figure 5.1: I think the NPCs were **not** repetitive

The question garnering the most diverse responses was regarding the repetitiveness of NPCs. Despite concerted efforts to minimize repetition, it remained noticeable to some extent. However, half of the responses were positive, with only one negative response, indicating a satisfactory level of believability.

It was fun interacting with the NPCs

12 responses

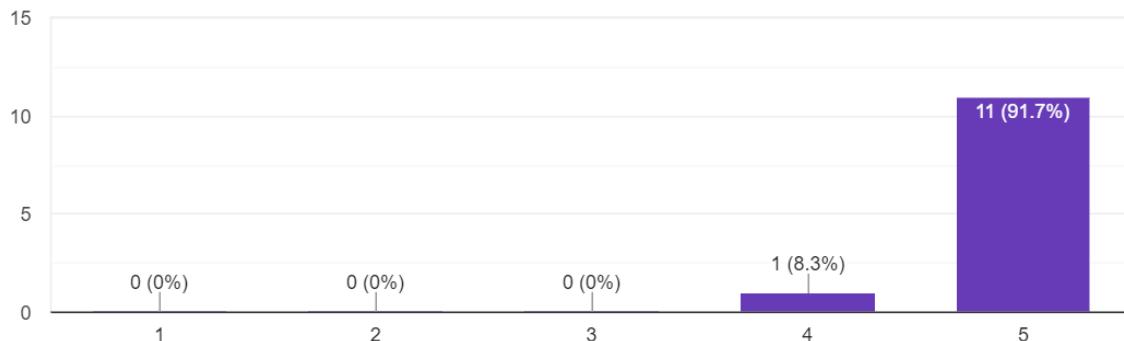


Figure 5.2: It was fun interacting with the NPCs

In terms of enjoyment, all participants found interacting with the NPCs to be a pleasant experience, with 11 out of 12 strongly agreeing that it was fun.

I quickly understood how to interact with the NPCs

12 responses

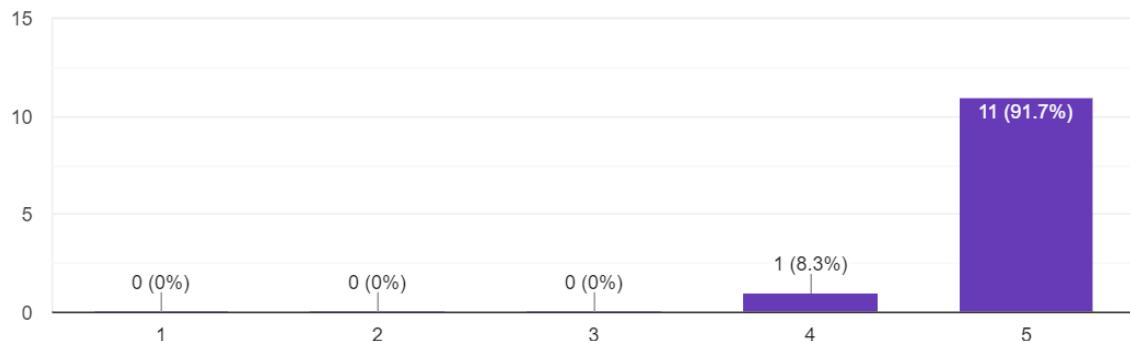


Figure 5.3: I quickly understood how to interact with the NPCs

Participants quickly grasped how to interact with the NPCs, as evidenced by unanimous positive responses to the question about ease of interaction.

I feel like the NPC's interactions were realistic

12 responses

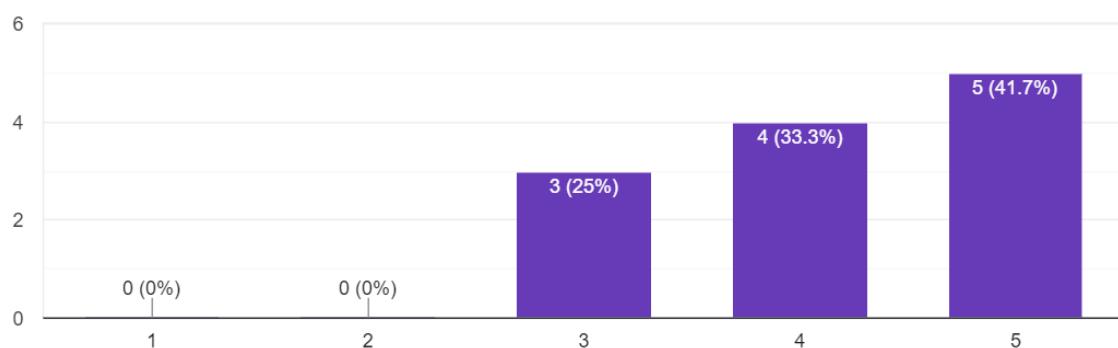


Figure 5.4: I feel like the NPC's interactions were realistic

When it came to the realism of NPC interactions, while there were no negative responses, there were three neutral responses and only five strongly agreeing. This discrepancy may stem from the occasional lack of appropriate action choices by NPCs, possibly attributed to their repetitiveness.

It was not frustrating interacting with the NPCs

12 responses

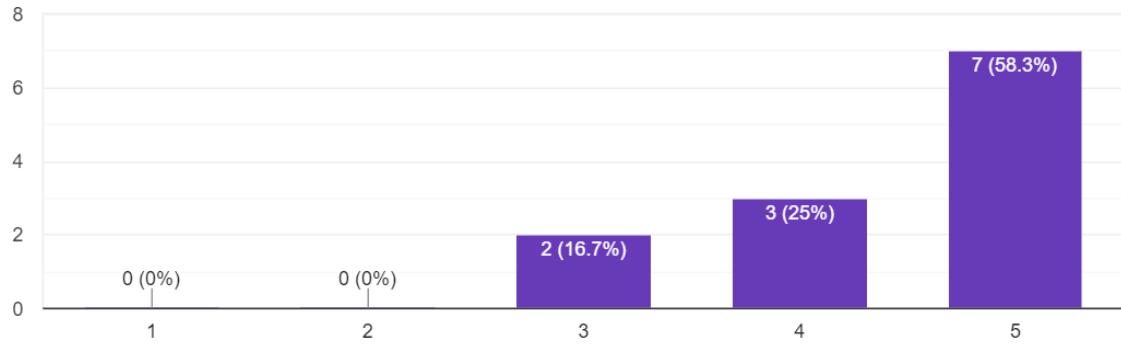


Figure 5.5: It was not frustrating interacting with the NPCs

Notably, none of the participants found interacting with the NPCs frustrating, indicating that any shortcomings in NPC behavior were outweighed by the enjoyment of the interactions.

I found it easy creating and deploying new NPCs

12 responses

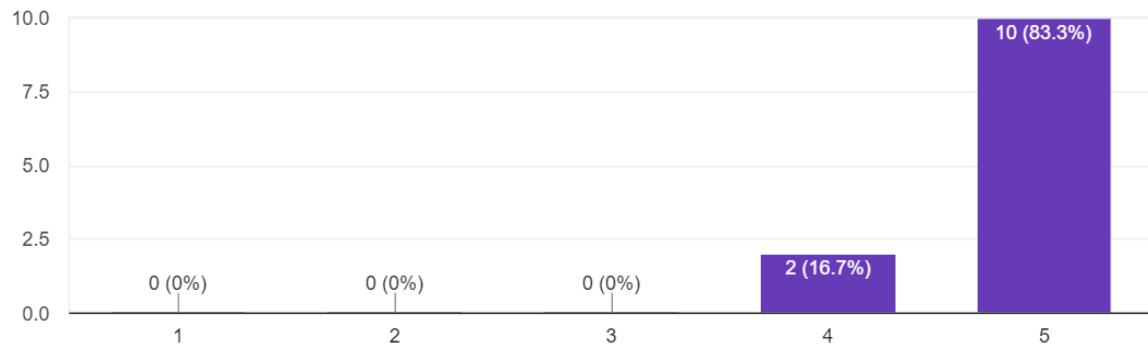


Figure 5.6: I found it easy creating and deploying new NPCs

Feedback on creating and deploying new NPCs was overwhelmingly positive, reflecting well on the intuitiveness of the documentation.

I found it easy editing NPCs (changing their memories, their emotions, their known locations, etc)

12 responses

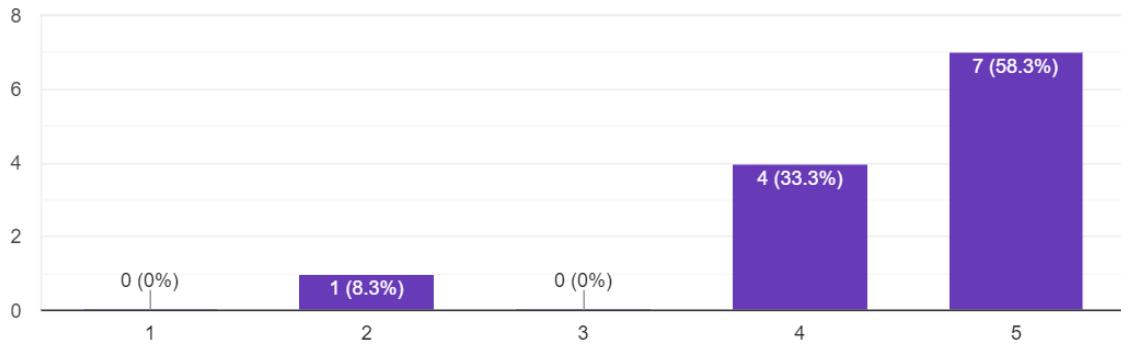


Figure 5.7: I found it easy editing NPCs (changing their memories, their emotions, their known locations, etc)

Similarly, the feedback on editing NPCs, including modifying their memories, emotions, and known locations, was mostly positive, with only one negative response. This suggests that while the editing GUIs are generally intuitive, there may be room for minor improvements to enhance user experience for some participants.

5.5 Reflecting on the feedback

- **Fun:** Interacting with the NPCs was universally enjoyable, as indicated by the positive responses from all participants.
- **Human-like reactions:** While not all responses strongly indicated realism, the majority of participants found NPC interactions to be believable, aligning with the intended goal of creating socially intelligent NPCs.
- **Ease of access:** Participants quickly understood how to interact with the NPCs, reflecting the simplicity of engaging with them.
- **Variation in agent behavior:** Despite occasional repetitiveness, NPCs displayed diverse actions and responses, contributing to engaging and dynamic interactions.
- **Ease of deployment and configuration:** Creating and deploying new NPCs was straightforward, with intuitive documentation facilitating the process, and editing existing NPCs was generally perceived as easy, with only minor room for improvement noted by one participant.

Considering the overwhelmingly positive feedback across all criteria, MineSocieties can be deemed a

success, fulfilling its objectives of providing enjoyable, human-like NPC interactions that are accessible, varied, and easy to manage within the game environment.

6

Conclusion

Contents

6.1 Conclusions	71
6.2 System Limitations and Future Work	71

6.1 Conclusions

MineSocieties successfully showcases the potential of LLMs in enhancing the immersion of gaming experiences by creating lifelike NPCs capable of intelligent social interactions. Furthermore, it dispels the notion that deploying complex NPCs in Minecraft requires advanced technical expertise, demonstrating that even individuals with minimal technical proficiency can integrate sophisticated NPCs into their gameplay.

6.2 System Limitations and Future Work

Despite its accomplishments, MineSocieties encounters several limitations that warrant consideration:

- **Cost of LLMs:** Trials using ChatGPT incurred a non-negligible cost, highlighting the financial burden associated with deploying LLMs in gaming environments. While local deployment offers a potential solution, it presents technical challenges, as demonstrated by the difficulties encountered with tools like Llama3 which drains considerable amounts of computer resources.
- **Challenges in Realism:** Despite advancements, achieving complete realism remains elusive. Even with GPT-4, NPCs occasionally exhibit behaviors that deviate from human-like responses.
- **Model Bias and Fairness:** Language models like ChatGPT are susceptible to bias present in their training data, which could manifest in NPCs exhibiting biased behaviors or responses.

Moving forward, several avenues for improvement and future work can enhance the quality and functionality of MineSocieties:

- **Prompt Engineering Enhancement:** Continued refinement of prompt engineering techniques can address existing issues such as repetitive NPC behavior, enhancing the overall realism and engagement of interactions.
- **Expansion of Action Repertoire:** Incorporating additional actions such as wood chopping, mining, and inventory management can enrich NPC capabilities, diversifying gameplay experiences.
- **Enhanced Observational Abilities:** Augmenting NPCs with the ability to observe and react to changes in their environment, including interactions among other NPCs, can further enhance their social intelligence.
- **Optimized Memory Management:** Implementing robust memory management systems, including the deletion of outdated short-term memories and leveraging LLMs for memory compression, can optimize NPC memory usage and performance.
- **Compatibility with Multiple LLMs:** Expanding compatibility with a broader range of LLMs beyond ChatGPT can offer users greater flexibility and access to diverse AI capabilities.

In closing, MineSocieties represents a significant step forward in the integration of LLMs in gaming environments, offering immersive and socially intelligent NPCs to players. As development continues, the project holds promise for further innovation and advancement in the field. I extend my gratitude to all who have supported and contributed to this endeavor. Thank you for your time and attention.

Bibliography

- [1] Rafael Lourenço, also known as, BlackKnight625, “MineSocieties: A Minecraft Plugin for socially intelligent NPCs,” <https://github.com/BlackKnight625/MineSocieties>, 2024.
- [2] “Papermc, a plugin api,” <https://papermc.io/>, accessed: 2023-04-17.
- [3] “Chatgpt,” Open AI, <https://openai.com/chatgpt>, accessed: 2024-05-26.
- [4] “Minecraft education edition,” https://minecraft.fandom.com/wiki/Minecraft_Education, accessed: 2023-05-10.
- [5] “Minecolonies mod for minecraft,” CurseForge, <https://www.curseforge.com/minecraft/mc-mods/minecolonies>, accessed: 2023-04-11.
- [6] “Minecraftcomesalive reborn mod for minecraft,” CurseForge, <https://www.curseforge.com/minecraft/mc-mods/minecraft-comes-alive-reborn>, accessed: 2023-04-11.
- [7] “Citizens plugin for minecraft,” SpigotMC, <https://www.spigotmc.org/resources/citizens.13811/>, accessed: 2023-04-23.
- [8] “Sentinels plugin for minecraft,” SpigotMC, <https://www.spigotmc.org/resources/sentinel.22017/>, accessed: 2023-05-23.
- [9] “Entityutils plugin for minecraft,” GitHub, <https://github.com/Yoursole1/EntityUtils>, accessed: 2023-05-24.
- [10] “Mineflayer,” GitHub, <https://github.com/PrismarineJS/mineflayer>, accessed: 2023-05-24.
- [11] C. Marques, “Social agents in minecraft,” Master’s thesis, Instituto Superior Técnico, Av. Rovisco Pais 1049-001 Lisboa, 2022.
- [12] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, “The malmo platform for artificial intelligence experimentation,” in *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 2016, pp. 4246–4247.

- [13] S. Mascarenhas, M. Guimarães, R. Prada, P. A. Santos, J. A. Dias, and A. Paiva, “Fatima toolkit: Toward an accessible tool for the development of socio-emotional agents,” *ACM Transactions on Interactive Intelligent Systems*, vol. 12, no. 1, March 2022. [Online]. Available: <https://doi.org/10.1145/3510822>
- [14] “Space modules inc. game components,” <https://www.gamecomponents.eu/content/351>, accessed: 2023-05-26.
- [15] “List of large language models,” Wikipedia, https://en.wikipedia.org/wiki/Large_language_model#List_of_large_language_models, accessed: 2023-05-26.
- [16] S. Park, J. C. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” *arXiv preprint arXiv:2304.03442*, 2023. [Online]. Available: <https://arxiv.org/abs/2304.03442>
- [17] jamdelaney1, “The city of adamantis,” <https://www.planetminecraft.com/project/the-city-of-adamantis/>, published: Nov 21st, 2012.
- [18] Mojang Studios, “Minecraft,” <https://www.minecraft.net/>, accessed: 2023-05-26.
- [19] ——, “Mojang studios,” https://en.wikipedia.org/wiki/Mojang_Studios, accessed: 2023-05-10.
- [20] “List of best-selling video games,” https://en.wikipedia.org/wiki/List_of_best-selling_video_games, accessed: 2024-05-26.
- [21] “Minecraft editions,” <https://help.minecraft.net/hc/en-us/articles/360034753992>, accessed: 2023-05-10.
- [22] “Minecraft java edition,” https://minecraft.fandom.com/wiki/Java_Edition, accessed: 2023-05-10.
- [23] “Minecraft plugins,” <https://minecraft-archive.fandom.com/wiki/Plugins>, accessed: 2023-05-10.
- [24] “Minecraft bukkit library,” https://bukkit.fandom.com/wiki/Main_Page, accessed: 2023-05-10.
- [25] “Minecraft mods,” <https://minecraft.fandom.com/wiki/Mods>, accessed: 2023-05-10.
- [26] “Minecraft forge mod library,” https://minecraftuniverse.fandom.com/wiki/Minecraft_Forge, accessed: 2023-05-10.
- [27] “Minecraft bedrock edition,” https://minecraft.fandom.com/wiki/Bedrock_Edition, accessed: 2023-05-10.
- [28] “Liteloaderbds,” <https://github.com/LiteLDev/LiteLoaderBDS>, accessed: 2023-05-10.
- [29] “Minecraft modding,” https://en.wikipedia.org/wiki/Minecraft_modding, accessed: 2023-05-10.

- [30] CurseForge, “Curseforge - the home for mods and addons,” <https://www.curseforge.com/>, accessed: 2024-05-28.
- [31] “The evnt company,” LinkedIn, <https://www.linkedin.com/company/evntgames>, accessed: 2023-05-24.
- [32] “Hypixel server network,” <https://hypixel.net/>, accessed: 2023-05-23.
- [33] “Hypixel studios,” <https://hypixelstudios.com/>, accessed: 2023-05-23.
- [34] “Minecraft bedrock built-in npcs,” <https://learn.microsoft.com/en-us/minecraft/creator/documents/createnpcs>, accessed: 2023-05-22.
- [35] “Project malmo,” GitHub, <https://github.com/Microsoft/malmo>, accessed: 2023-05-24.
- [36] “Fatima toolkit about us,” <https://fatima-toolkit.eu/home/about-us/>, accessed: 2023-05-26.
- [37] “Inesc-id,” <https://www.inesc-id.pt/>, accessed: 2023-05-26.
- [38] “Fatima toolkit,” GitHub, <https://github.com/GAIPS/FAtiMA-Toolkit>, accessed: 2023-05-25.
- [39] “Fatima toolkit getting started tutorial,” <https://fatima-toolkit.eu/home/get-started/>, accessed: 2023-05-25.
- [40] “Unreal engine,” <https://www.unrealengine.com/>, accessed: 2023-05-25.
- [41] “Unity,” <https://unity.com/>, accessed: 2023-05-25.
- [42] “Playgen,” <https://playgen.com/>, accessed: 2023-05-26.
- [43] “Open ai,” <https://openai.com/>, accessed: 2023-05-26.
- [44] “Chatgpt-4,” Open AI, <https://openai.com/product/gpt-4>, accessed: 2023-05-26.