Generative AI Agents for Minecraft: Automating Creative World Tasks with Amazon Bedrock

Team Members

Taeesh Azal Assadi
Sanjna Kumari
Faraz Ulhaq Shah

February 13, 2025

1 Project Charter

1.1 Project Name

Generative AI Agents for Minecraft: Automating Creative World Builds with Amazon Bedrock

1.2 Problem Statement

Creating a fully developed and engaging Minecraft server is time-consuming and requires advanced building skills, posing a challenge for new and casual players. Our Generative AI Agent, powered by AWS Bedrock, automates world-building and gameplay setup through natural language commands, enabling users to quickly create detailed environments and interactive elements, making server creation faster and more accessible.

1.3 Project Description

Our project focuses on developing a Generative AI Agent for Minecraft, designed to automate world-building and gameplay setup tasks within the game. Leveraging AWS Bedrock, the AI agent allows players to create complex environments, structures, and interactive game elements using natural language commands. This system addresses the significant challenge faced by new and experienced server owners who often spend weeks or months manually building detailed worlds and setting up game mechanics. By integrating AI into Minecraft's in-game chat system, the agent enables users to quickly generate custom landscapes, buildings, and minigames in real-time, enhancing the server creation process. The primary stakeholders include new server owners, experienced builders, content creators, gaming communities, educators, and AI enthusiasts, all of whom benefit from reduced development time, increased creativity, and more accessible server management. The system's cloud-based, real-time integration ensures scalability and seamless operation without requiring external applications, making server ownership and world development faster and more efficient.

1.4 Project Outcome

Functional Minecraft Agent: A working AI-powered agent integrated into Minecraft, capable of executing natural language commands to build custom environments, structures, and interactive game elements in Creative Mode.

1.5 Project Team Members and Roles

Team Member	**Key Responsibilities**
Taeesh Azal Assadi	Develop and fine-tune the generative AI model that interprets natural language commands to automate world-building and gameplay tasks. Train and optimize the AI model to generate Minecraft structures, terrains, and interactive elements based on user input. Work with AWS Bedrock to leverage pre-trained language models and integrate generative capabilities into the system. Ensure AI-generated content aligns with user expectations and gameplay requirements.
Sanjna Kumari	Design and implement the backend infrastructure to bridge the AI model with the Minecraft server environment. Develop APIs and data pipelines to handle requests from the AI model and execute corresponding world-building and gameplay actions in Minecraft. Implement real-time communication between the AI processing system and the Minecraft server. Ensure server performance, request handling, and data flow efficiency for seamless execution.
Faraz Ulhaq Shah	Integrate the generative AI system with Minecraft servers, ensuring compatibility with server-side plugins and in-game command execution. Deploy and manage cloud infrastructure using AWS Bedrock and other AWS services to support scalable and low-latency AI processing. Implement server-side automation for executing AI-generated builds and gameplay events within Minecraft in real-time. Monitor system performance, troubleshoot cloud-based issues, and optimize resource usage to ensure system reliability during peak server loads.

1.6 Timeline for Completion

Week	Phase/Sprint	Key Tasks	Deliverables
2-3	Phase 1	Research Minecraft APIs, AWS Bedrock, Generative AI models. Draft initial problem statement and project scope. Identify key functional and non-functional requirements. Prepare high-level project plan.	Phase 1 Deliverable
3-4	Phase 2	Refine system design based on team discussions. Prepare detailed project requirements, architecture, tools.	Phase 2 Deliverable
5	Sprint 1	AI Model Setup & Backend Prep. Begin developing natural language processing (NLP) model for simple structure commands. Set up Minecraft server environment, API framework. Set up AWS Bedrock environment.	Basic Development Environment Setup
6	Sprint 1	AI Command Execution. Implement basic structure commands. Connect backend to Minecraft server. Integration Testing: Simple commands executed in-game.	Basic AI Commands Executed in Minecraft
7	Sprint 1	Testing & Refinement. Test simple AI-generated builds in Minecraft. Backend optimization for command processing.	Testing for Sprint 1
8	Sprint 1	Sprint Review & Planning. Conduct Sprint 1 review and retrospective. Document progress and plan Sprint 2 tasks focusing on complex structures and performance improvements.	Sprint 1 Deliverable
9	Sprint 2	Complex Structure Generation. Enhance the model to generate more complex structures. Implement support for multi-command requests.	AI Generating Structures
10	Sprint 2	Real-Time Execution Testing. Implement real-time API request handling. Utilize AWS Bedrock performance.	Build Execution in Minecraft
11	Sprint 2	Review & Improvements. Test full structure and terrain generation in-game. Optimize AI accuracy and reduce command execution time. Conduct Sprint 2 review and plan Sprint 3.	Sprint 2 Deliverable (Builds & Real-Time Testing)
12	Sprint 3	Interactive Elements & Gameplay. Extend AI to support interactive elements. Develop multi-command processing.	Basic Interactive Features.
13	Sprint 3	Minigames & Server Events. Implement sample minigames (e.g., hide-and-seek). Automate event-based builds. Ensure stability with multiple AI commands.	Minigames and Automated Events Implemented
14	Sprint 3	Final Testing & Review. Test full integration: world generation, interactive elements, and minigames. Final bug fixes and performance tuning. Conduct Sprint 3 review and retrospective.	Sprint 3 Deliverable
15	Final Product & Presentation	Prepare final system demonstration video. Complete technical documentation and user manual. Final team review and polish deliverables. Submit final product.	Final Product Submission

1.7 Communication Plan for the Team

We will conduct two team meetings per week in person or remote to maintain alignment and address any blockers promptly.

Frequency	**Purpose**	**Duration**
Twice a Week (Every Monday & Thursday)	Sprint Progress Updates & Task Synchronization: Review ongoing tasks, discuss roadblocks, and reassign work as needed to maintain velocity.	30 Minutes
Sprint Planning (Beginning of Each Sprint)	Sprint Planning: Define sprint goals, prioritize backlog items, and break down user stories into actionable tasks.	60 Minutes
Sprint Review (End of Each Sprint)	Sprint Review: Demonstrate completed work, review deliverables, gather team feedback, and refine the backlog based on outcomes.	45-60 Minutes
Sprint Retrospective (End of Each Sprint)	Sprint Retrospective: Reflect on the sprint process, discuss what went well, what could be improved, and agree on changes for the next sprint.	30 Minutes

Our team will communicate via Discord/Whatsapp for quick updates, Email for formal deliverables, and Trello for task tracking. We will post daily check-ins on Discord, sharing what we did yesterday, today's tasks, and any blockers, following Agile Stand-Up practices to ensure alignment and progress.

2 The Case for the System (Product)

2.1 Need for the System (Product)

Why do we need this product? what purpose does it serve?
Minecraft is a globally popular sandbox game that allows players to explore, build, and survive in an open-world environment made entirely of blocks. Players can gather resources, craft tools, construct buildings, and interact with an infinite landscape limited only by their imagination. The game offers two primary modes: Survival Mode, where players must gather materials and manage health while facing threats like hostile mobs, and Creative Mode, where players have unlimited resources and can freely build without restrictions. Multiplayer servers allow multiple players to connect and play together in shared worlds. Some of the most popular servers have massive, intricately designed worlds with custom landscapes, themed cities, minigames, and automated gameplay mechanics that attract thousands of players. Examples include **"Winter Wonderland"**, a snowy landscape with interactive events, or **"Zombieland"**,where players fight off hordes of undead in a post-apocalyptic setting. These servers often take months or even years to build manually, requiring teams of experienced builders and developers to maintain them.
However, when a new player or group wants to start their own server, they face a major challenge—an empty, lifeless world that requires extensive effort to develop into something engaging. Manually constructing detailed environments, setting up interactive game mechanics, and building structures can be overwhelming, especially for casual players or those with limited building skills. This is where our Generative AI Agent for Minecraft comes in.
By leveraging AWS Bedrock, our AI-powered agent acts as a Minecraft assistant that automates creative

world-building tasks within Creative Mode. Instead of spending weeks or months manually constructing a world, players can simply issue natural language commands, such as:

- "Build a medieval castle with walls and towers."
- "Create a forest with rivers and hidden caves."
- "Generate a large stadium with stands and a scoreboard."
- "Set up a hide-and-seek game with obstacles and hiding spots."

The AI will process these requests and immediately execute them within the game, allowing users to rapidly populate their servers with complex and visually appealing structures.
Beyond world-building, the AI assistant enhances gameplay experiences by automating game mechanics. For example, it can:

- **Facilitate Minigames** – Set up games like Squid Game, obstacle courses, or treasure hunts.
- **Create NPCs and Interactive Elements** – Generate AI-controlled characters that guide players, act as enemies, or play cooperative roles.
- **Manage Server Events** – Automate seasonal events like Halloween haunted houses or holiday-themed builds to keep the server fresh and engaging.

Ultimately, our AI-powered Minecraft helper solves a key problem in the multiplayer gaming space—the difficulty of creating a fully developed and engaging Minecraft world from scratch. By reducing the time, skill, and effort required, this system makes Minecraft server ownership more accessible and enjoyable for everyone, from casual players to professional server operators.

Who are the main stakeholders and end users?
The system serves a wide spectrum of stakeholders, each deriving something different from the AI-based Minecraft agent:

- **New Minecraft server owners** – People or groups that need to create their own servers but do not have time or capability to hand design elaborate worlds. They can generate custom worlds, buildings, and game modes using the AI agent in a short period of time.
- **Experienced Minecraft Builders & Server Admins** – Even skilled players take time to build large builds and interactive game design. AI assistance hastens development, allowing them to work on details and optimization of game mechanics.
- **Game Modders & Creators of Content** – Modders of Minecraft and YouTube creators of custom in-game experiences can leverage AI to generate worlds and interactive behaviour of NPCs to facilitate engaging content.
- **Gaming Communities & Multiplayer Networks** – Large Minecraft communities, like Hypixel-style servers, can leverage AI automation to add new content, season events, and new game modes without having to put in a great deal of work manually.
- **Educators & Learning Platforms** – In schools, Minecraft is used to teach computer programming, engineering, and creativity. AI-infused Minecraft worlds can be utilized to facilitate automated lesson plans, construction of landmarks in history, or modelling of experiments in physics.
- **Cloud Computing & AI Enthusiasts** – AI and cloud services enthusiasts can discover how a game environment is interacted with using generative AI, making this system a wonderful resource for AI automation studies and innovations.

If your system is an app, why does it have to be a mobile app (not a desktop or a web application)?
This system is not a distinct mobile or web application because it is deeply integrated in the chat system of Minecraft and is server side based. In place of players having to interact with a different application, the AI agent is native to the game, providing smooth and natural interaction.

- **No External UI is required** – The AI is controlled using in-game chat, making everything intuitive and accessible to players.
- **Real-Time Game Integration** – AI agents execute orders in real-time in the game environment, enabling instant adjustment of buildings, landscapes, and game mechanics.
- **Cloud-Based Scalability** – The application is hosted on AWS Bedrock, providing on-demand processing without bogging down the player's machine.
- **Persistent Server Automation** – AI agents remain active in the Minecraft world to assist in uninterrupted gameplay without having to be toggled between applications as a separate utility.

Because of this, a traditional application-based method would be unnecessary and wasteful, since the AI agent is of most use when it is integrated straight into Minecraft's existing UI.

2.2 Current Market

What are the other systems that have goals similar to your system (mention some examples)?
There are a few systems and tools that already try to help with automation in Minecraft and world generation, yet none of them deliver the same extent of integration, flexibility, and AI-based automation that our Generative AI Agent for Minecraft is capable of. Some of the existing alternatives, and their shortcomings, are outlined below:

1. Minecraft World-Editing Tools (e.g., WorldEdit, FAWE)
Description:

- WorldEdit and its more efficient variant, FAWE (Fast Async WorldEdit), are highly used plugins that assist you in editing Minecraft worlds more efficiently. Both allow players to paste, terraform, and edit terrain using a GUI or via commands.
- These tools allow for speedy construction manually yet require the user to input accurate commands and be familiar with the tool's syntax.

Criticism:

- **Steep learning curve** – Users must learn a set of commands and parameters to use these tools effectively.
- **Lack of AI decision-making** – The tool does not generate creative designs on its own; the user still needs to manually plan and specify what to build.
- **No interactivity or automation** – Cannot be used for setting up minigames or interactive game mechanics beyond static world editing.

2. AI-Generated Minecraft Structures (e.g., OpenAI's GPT-Generated Blueprints, Litematica, Schematica)
Description:

- Some AI models, such as GPT-based systems, have been used to generate text-based blueprints for Minecraft builds.

- Litematica and Schematica allow players to import existing structures in their worlds using schematic files.

Criticism:

- **Prebuilt structures are not flexible** – Users cannot easily personalize AI-made blueprints.
- **Still requires human input** – The players must manually import and place buildings rather than having AI generate buildings dynamically in real-time.
- **No gameplay interaction** – Such tools apply to static builds without involving AI in game mechanics, challenges, or automation.

How Our System Is Superior to Other Solutions
Our Generative AI Agent for Minecraft is distinct from such existing solutions in that:

- **Utilizing AI to dynamically generate builds and game scenarios in real-time** – As opposed to WorldEdit, our bot can generate personalized builds without players having to manually define dimensions or commands.
- **Leveraging Natural Language Processing (NLP)** – In a departure from script or syntax learning like Mineflayer and WorldEdit, players can input in plain language via chat (for example, "Build a moated fortress", "Make a PVP arena", "Create a hideout underground").
- **Seamless multiplayer server support** – In contrast to tools such as Baritone that are better adapted to single-player automation, our AI agent is developed to function in shared server settings to support cooperative or competitive AI play.
- **Expanding beyond mere world-building** – The system would also generate game creation automatically in that it would generate minigames, challenges, and interactive NPCs, something that programs such as Litematica or Schematica cannot provide.

By addressing these limitations, our system offers a total AI-based system that not only simplifies world construction but also enhances gameplay, making it easier to build Minecraft servers, more interactive, and more enjoyable to play.

2.3 Competitive Analysis

What is "new" about the system? Is it the idea, or is it the way it approaches a solution that already exists?
Our Generative AI Agent introduces a new approach, not a new concept. There already is automation and AI tools in Minecraft (pathfinding bots or world-editing plugins, for instance), yet none of these deliver a whole integrated AI-based system of game automation and world generation using natural language input.
The novelty of our system is that it combines generative AI, real-time interactivity, and automation in the cloud using AWS Bedrock. Unlike existing tools that require one to program using pre-defined scripts, advanced commands, or even human input, our AI allows users to generate and customize their Minecraft world easily just by instructing it in chat on what to do.
Additionally, while current implementations support one-player automation (such as moving or mining bots), our system is designed to be used in multiplayer servers, making it useful to large community-run Minecraft servers in addition to casual players.

How do you think your system will be different or better than existing products?

Feature	**Existing Tools (WorldEdit, Baritone, Mineflayer, etc.)**	**Our AI Agent**
Natural Language Processing	No NLP; requires commands/scripts	AI understands chat-based requests
Automated World-Building	Limited to copying/pasting structures	AI generates custom builds in real time
Game Automation (Minigames, AI NPCs, etc.)	Mostly manual setup required	AI can set up challenges and interactive elements
Multiplayer Server Integration	Works mostly in single-player or admin-controlled mode	Designed for multiplayer servers
Creativity & Adaptability	Users must plan builds manually	AI adapts and generates dynamic structures
Scalability	Limited by local server resources	Cloud-based AWS Bedrock processing
Customization & Flexibility	Requires players to modify configs or scripts	AI adapts to different requests with ease

Table 1: Comparison Between Existing Tools and Our AI Agent

Why Our System Will Be Better

- **Ease of use** – Say goodbye to scripting, goodbye to hard-coded commands—just use natural language commands, and let AI handle it.
- **AI Creativity & Flexibility** – The AI constructs distinct settings dynamically, not mere replication of existing frameworks, depending on user intent.
- **Multiplayer-Ready** – As opposed to most AI tools that focus on automation in a single-player setup, our system is multiplayer-ready, making it a fantastic utility for server admins, creators, and ordinary players.
- **Seamless Gameplay Integration** – Aside from world creation, our AI introduces in-game characters, interactive challenges, and mini-games, making it not just a utility, but a dynamic game participant.
- **Cloud-Powered for Efficiency** – By processing workloads in AWS Bedrock, high performance is ensured without bogging down the local Minecraft server.

By combining AI-assisted world generation, game automation, and multiplayer support, our system beats existing alternatives in making Minecraft servers more interactive, easier to set up, and infinitely configurable.

3 System Description

3.1 Technical, Business, or Administrative Problem Addressed

Technical Problem
Developing interactive and dynamic environments in Minecraft manually is a tedious and time-intensive process. Additionally, the lack of built-in AI-driven interactivity limits how non-player characters (NPCs) can engage with players. The integration of AI-powered agents using Amazon Bedrock enables a solution where bots can build structures, play games, assist users, and engage in interactive activities, making the gaming experience richer and more autonomous.

Business Problem
For Minecraft server administrators, content creators, and game developers, creating engaging and immersive worlds is crucial for retaining players. Manual world-building and gameplay enhancements require significant human effort, leading to increased costs and longer development times. By leveraging AI-driven bots, businesses can automate world generation, create dynamic NPC interactions, and offer AI-driven companions, significantly reducing operational overhead while enhancing the player experience.

Administrative Problem
Managing a multiplayer server with dynamic environments and interactive features often requires continuous supervision. Players expect engaging NPCs, automated construction, and interactive in-game assistance. This AI-powered bot helps automate world modifications, interact with players, and provide guidance or companionship, reducing the administrative burden on server moderators and improving server efficiency.

3.2 Dataflow Diagrams (DFD)

Context Diagram

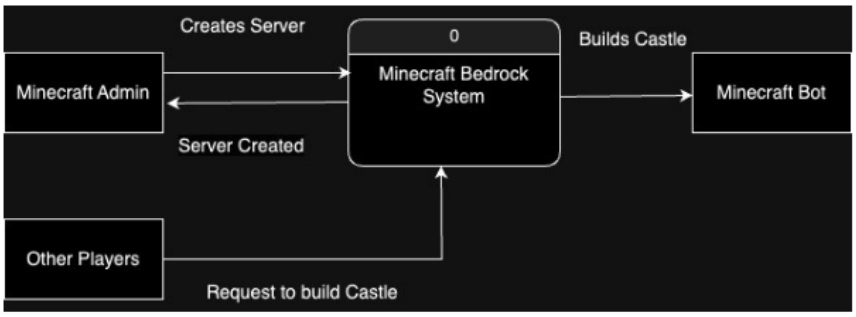

Figure 1: Context Diagram of Minecraft Bedrock System

This Context Diagram represents the interaction between different entities in the Minecraft Bedrock System for automating the construction of a castle (here the action could be anything) using an AI-powered bot. The key components include:

- **Minecraft Admin:** Creates the Minecraft server and manages its operation.
- **Other Players:** Can request the system to perform actions (here - build a castle).
- **Minecraft Bedrock System:**
  - The central system that processes requests.
  - Receives server creation instructions from the Minecraft Admin.
  - Accepts requests to build a castle from other players.
  - Sends a "Build Castle" command to the Minecraft Bot.

- **Minecraft Bot:** Executes the building task within the Minecraft environment.

Level 0 Diagram

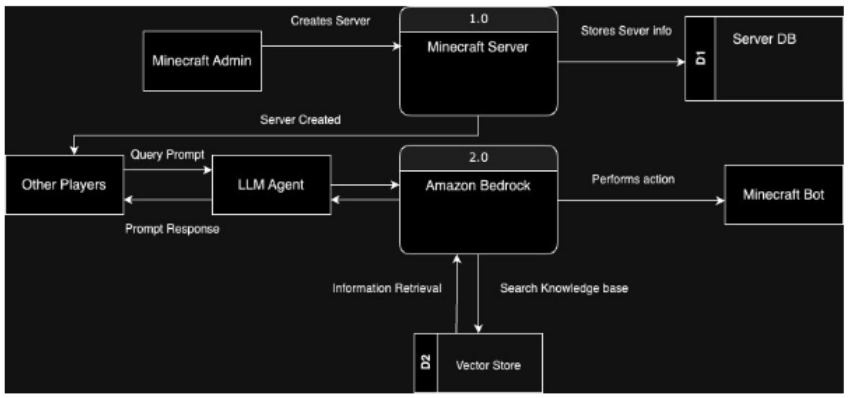

Figure 2: Level 0 Diagram of Minecraft Bedrock System

This Level-0 Data Flow Diagram (DFD) represents the detailed process of how the Minecraft Server, Amazon Bedrock, and various components interact to handle player requests and automate tasks. Key components and their interactions include:

- **Minecraft Admin**
  - Creates the Minecraft Server.
  - The Minecraft Server stores server-related information in the Server Database (D1).
- **Other Players**
  - Submit query prompts (e.g., "Build me a castle" to the LLM Agent.
  - Receive responses from the LLM Agent based on query processing.
- **LLM Agent**
  - Processes player queries and sends them to Amazon Bedrock for execution.
  - Returns the prompt response to the player.
- **Amazon Bedrock**
  - Receives the processed prompt from the LLM Agent.
  - Performs actions by instructing the Minecraft Bot to build structures or execute tasks.
  - Retrieves information from the Vector Store (D2) for knowledge-based actions.
  - Searches the knowledge base in the Vector Store to enhance response generation.
- **Minecraft Bot**
  - Executes actions based on the instructions from Amazon Bedrock.
  - Performs in-game tasks such as building structures.

3.3 Assumptions and Risks

Assumptions

- **Server and API Integration** – The Minecraft server supports bot integration via API or plugin (e.g., Mineflayer).
- **AI Model Understanding** – The AI agent can correctly interpret user commands and adapt to various interactions.
- **Scalability** – The system can handle multiple bot interactions simultaneously without performance degradation.
- **Real-Time Processing** – AI responses and bot actions occur with minimal latency for smooth gameplay.
- **User Engagement** – Players will interact meaningfully with the bot for building, playing, and assistance purposes.

Risks

- **Security Threats** – If not properly secured, malicious users could exploit the bot to alter game worlds or misuse interactive features.
- **AI Misinterpretation** – The bot may misunderstand vague commands and perform unintended actions.
- **Server Load Impact** – Running multiple AI-powered bots simultaneously may overload the Minecraft server.
- **AI Ethical Concerns** – Automating world-building and NPC interactions might reduce human creativity and engagement.
- **Dependency on Amazon Bedrock** – The system relies on Amazon Bedrock's AI capabilities, meaning any service disruption could affect bot functionality.

4 Team Dynamic

What skills do your team members bring to this project?
Our team brings a solid foundation in AI development and backend systems. We have experience in Natural Language Processing (NLP), machine learning, and generative AI models, enabling us to develop the AI component that interprets player commands and generates Minecraft content. Additionally, our backend expertise includes API development, server-side programming, and real-time system integration, allowing us to build a robust connection between the AI model and the Minecraft server.

What skills are missing and you'll need to learn to deliver the system?
We lack specific knowledge in AWS Bedrock and cloud based advanced services. Since our system relies heavily on AWS Bedrock for deploying and scaling the generative AI model, we will need to learn how to set up, configure, and integrate AWS Bedrock services. This includes understanding cloud-based AI processing, API interactions with Bedrock, and optimizing performance for real-time execution in Minecraft. Acquiring these skills will be essential for ensuring the scalability and efficiency of our system.

How are you planning to obtain such missing skills?
We plan to obtain the missing AWS Bedrock skills through a combination of self-directed learning, online resources, and hands-on experimentation. We will refer to AWS documentation, tutorials, and official guides specific to AWS Bedrock and cloud-based AI deployments. Platforms like AWS Training, Udemy, and YouTube offer practical courses on AWS services that can help us understand Bedrock's capabilities, including

setting up models, integrating APIs, and managing cloud resources. Additionally, we will collaborate as a team to share knowledge and overcome any learning gaps. This approach will ensure we develop the necessary cloud and deployment expertise alongside our AI and backend work.