TARTU UNIVERSITY   
FACULTY OF SOCIAL SCIENCES

NARVA COLLEGE  
STUDY PROGRAM “INFORMATION SYSTEMS DEVELOPMENT“

Daniel Neustus

**A Program for Generating Minecraft Code and Command Blocks Using C# Windows Forms and Forge Modding**

Bachelor’s Thesis

Supervisor: Dennis Ruder

NARVA 2023

TARTU ÜLIKOOL   
SOTSIAALTEADUSTE VALDKOND

NARVA KOLLEDŽ  
“INFOTEHNOLOOGILISTE SÜSTEEMIDE ARENDUS“

Daniel Neustus

**Programm Minecrafti koodi ja käsuplokkide genereerimiseks kasutades C# Windows Forms ja Forge moddingut**

Lõputöö

Juhendaja: Dennis Ruder

NARVA 2023

**TABLE OF CONTENTS**

[INTRODUCTION 4](#_Toc163759468)

[RESEARCH QUESTIONS 6](#_Toc163759469)

[1. BACKGROUND / THEORETICAL FRAMEWORK 6](#_Toc163759470)

[1.1.1 Overview of Minecraft Command Blocks 6](#_Toc163759471)

[1.1.2 Command Block Automation 6](#_Toc163759472)

[1.2 C# Programming Language 7](#_Toc163759473)

[1.2.1 Introduction to C# 7](#_Toc163759474)

[1.2.2 Minecraft code 7](#_Toc163759475)

[1.3 Code Generation Techniques 7](#_Toc163759476)

[1.3.1 Code Generation in Software Development 7](#_Toc163759477)

[1.3.2 Dynamic Code Generation in C# 7](#_Toc163759478)

[1.4 WINDOWS FORMS 8](#_Toc163759479)

[1.4.1 Windows Forms in Software Development 8](#_Toc163759480)

[1.4.2 Integration with Other Technologies 8](#_Toc163759481)

[1.5 FORGE 8](#_Toc163759482)

[2. USER INTERFACE 9](#_Toc163759483)

[2.1 Design Considerations 9](#_Toc163759484)

[2.2 Components and Controls 9](#_Toc163759485)

[2.3 User Workflow 10](#_Toc163759486)

[2.4 Visual Representation 10](#_Toc163759487)

[2.5 Utilization of Multiple Forms for Modular Functionality 11](#_Toc163759488)

[2.6 Generator Forms 11](#_Toc163759489)

[2.7 Benefits of Modular Design 14](#_Toc163759490)

[2.8 Functionality of Action Buttons 14](#_Toc163759491)

[2.9 Benefits of Action Buttons 15](#_Toc163759492)

[3. MINECRAFT MODIFICATION 16](#_Toc163759493)

[3.1 Forge Modding 16](#_Toc163759494)

[3.2 Community and Resources 17](#_Toc163759495)

[3.3 Framework and Architecture 17](#_Toc163759496)

[3.4 Development Workflow 18](#_Toc163759497)

[4. CONFIGURING A SERVER 19](#_Toc163759498)

[5. REQUEST TO SERVER 20](#_Toc163759499)

[5.1 Instructions from a client to the server 20](#_Toc163759500)

[6. END RESULT 23](#_Toc163759501)

[CONCLUSION 24](#_Toc163759502)

[SOURCE MATERIALS 25](#_Toc163759503)

INTRODUCTION

Minecraft is a sandbox game where players can develop their imagination. The endless world-building possibilities of Minecraft often require a thorough understanding of the underlying codes and command blocks. As games evolve and new features are added, the need for efficient and dynamic methods of code generation becomes increasingly important (Fowler, M. 2003).

This thesis discusses the field of Minecraft modification, focusing on the use of the C# Windows Forms. Drawing on Hejlsberg's (2003) work on C# design principles and features, the aim is to create a convenient program that simplifies the process of generating Minecraft code and command blocks. This program assists in the creation of complex structures, functions, games, and applications in Minecraft, serving as an educational resource as well, enabling users to develop their programming skills.

The main objective of this research is to develop a convenient program that utilizes the C# programming language providing Minecraft players and programmers a tool that can easily transform creative ideas into in-game elements. Other objectives include:

1. Efficiency and automation: Designing a system that significantly reduces the time and effort required for generating Minecraft code and command blocks.
2. User-friendly interface: Creating an intuitive and user-friendly interface suitable for all users.

To achieve the stated objectives, several tasks must be completed throughout the development process. These tasks encompass both technical aspects and user experience:

1. Technical background of Minecraft modification: Explain the concept of Minecraft modification and why it is important for the game's development.
2. C# programming language: Develop strong integration with the C# programming language, ensuring compatibility with Minecraft modification frameworks and programming languages.
3. User interface design: Create an intuitive and visually appealing user interface using Windows Forms, facilitating easy navigation and interaction.
4. Logic for code generation: Implement algorithms and logic to generate Minecraft code and command blocks based on user input and specifications, covering a wide range of in-game elements.

RESEARCH QUESTIONS

In the development of the program, certain challenges and potential problems may arise, requiring careful consideration and problem-solving strategies.

1. How can we ensure that the Minecraft code generation system developed using C# is well-maintainable, readable, and follows best practices?
2. How does the generation system impact players' user experience, enabling various scenarios and worlds?
3. What are the key security and safety aspects to consider when developing the Minecraft code generation system using C#?

1. BACKGROUND / THEORETICAL FRAMEWORK

1.1.1 Overview of Minecraft Command Blocks

Minecraft, developed by Mojang Studios (Persson, M., 2011), is a popular computer game that allows players to construct and explore virtual worlds made of blocks. Command blocks allow players to automate actions and create complex systems within the Minecraft environment. Understanding the structure and functionality of command blocks is essential for comprehending the importance of generating Minecraft code.

Command blocks execute commands written in the Minecraft command language, enabling manipulation of the game world. Persson's (2011) seminal work and subsequent updates from Mojang Studios serve as the basis for understanding the syntax and capabilities of command blocks.

1.1.2 Command Block Automation

Minecraft automation involves the use of command blocks to streamline repetitive tasks or create dynamic in-game systems. Bergensten's (2012) study explores the potential of command blocks in automating various aspects of the game, from teleportation mechanisms to Redstone contraptions. Understanding the scope of automation lays the foundation for developing an efficient and customizable program for generating Minecraft code.

1.2 C# Programming Language

1.2.1 Introduction to C#

Developed by Microsoft, C# is a versatile and object-oriented programming language designed for creating robust and scalable applications. Hejlsberg's (2003) work serves as the basis for understanding the design principles and features of C#. This section explores the language's core elements, including syntax, data types, and object-oriented concepts.

1.2.2 Minecraft code

A thorough understanding of C# and the Minecraft interface is crucial for developing the Minecraft code generation program. Minecraft Forge API, as discussed by the FML Team (2012), provides a framework for modding Minecraft using Java. Adapting these principles to C# requires a nuanced approach, considering language differences and compatibility issues.

1.3 Code Generation Techniques

1.3.1 Code Generation in Software Development

Code generation is the process by which a program produces source code or other forms of executable code based on predefined templates or rules. Fowler (2003) provides an overview of code generation techniques and their applications in software development. Understanding these techniques is essential for designing a program capable of dynamically generating Minecraft code.

1.3.2 Dynamic Code Generation in C#

Dynamic code generation in C# involves creating and executing code at runtime. Sussman (2012) delves into the intricacies of dynamic code generation, highlighting its advantages and potential challenges. This knowledge is crucial for developing a program that dynamically generates Minecraft code tailored to specific in-game scenarios.

1.4 WINDOWS FORMS

1.4.1 Windows Forms in Software Development

Windows Forms (WinForms) is a graphical user interface (GUI) framework for designing and developing desktop applications on the Microsoft Windows platform. It provides a comprehensive set of tools and controls that allow developers to create rich and interactive user interfaces with ease.

Windows Forms remains a popular choice for developing desktop applications due to its simplicity, flexibility, and extensive feature set. With its intuitive visual designer, robust event-driven programming model, and seamless integration with other .NET technologies, WinForms provides a powerful framework for creating sophisticated GUI applications. In the context of the proposed program for generating Minecraft code and command blocks, Windows Forms offers a familiar and efficient environment for designing the user interface, enabling developers to focus on implementing the core functionality of the application.

1.4.2 Integration with Other Technologies

Windows Forms seamlessly integrates with other technologies and frameworks, allowing developers to leverage the full power of the .NET ecosystem. Whether it's integrating with databases using ADO.NET, accessing web services via WCF, or incorporating advanced functionality through third-party libraries, WinForms provides a versatile platform for building feature-rich desktop applications.

1.5 FORGE

Forge developed by FML team (2012) is a popular modding framework for Minecraft that allows developers to create and customize game modifications (mods) with ease. Developed in Java, Forge provides a robust set of APIs and tools for extending and modifying the Minecraft game world. Forge offers a comprehensive API (Application Programming Interface) that exposes various hooks, events, and functionalities for mod developers. The Forge documentation provides detailed guides, tutorials, and examples to help developers understand and utilize the framework effectively.

2. USER INTERFACE

2.1 Design Considerations

The selection of Windows Forms as the GUI framework for the proposed program for generating Minecraft code and command blocks is rooted in its familiarity, integration capabilities, development efficiency, and community support. By leveraging the strengths of Windows Forms within the context of the .NET ecosystem, developers can create intuitive, feature-rich user interfaces that enhance the overall usability and functionality of the application. With careful consideration of cross-platform considerations and a focus on community engagement, Windows Forms serves as a strategic choice for realizing the objectives and requirements of the project.

Before going into the technical aspects, it's crucial to outline the design considerations that guide the development of the Windows Forms UI. These considerations include:

1. Intuitiveness: The interface should be intuitive, allowing users to navigate through different functionalities effortlessly. This involves arranging controls logically and providing descriptive labels and tooltips for clarity.
2. Consistency: Maintaining consistency in design elements such as color schemes, font styles, and control placement fosters familiarity and reduces cognitive load for users.
3. Flexibility: The UI should be flexible enough to accommodate various screen resolutions and form factors, ensuring a consistent experience across different devices.
4. Feedback Mechanisms: Incorporating feedback mechanisms like progress indicators, status messages, and error dialogs enhances user confidence and provides valuable insights into program operations.

2.2 Components and Controls

Windows Forms provides an abundance of components and controls that facilitate the creation of rich, interactive interfaces. Some of the key components utilized in our code generation program include:

1. Buttons: Used for triggering actions such as code generation, saving configurations, or initiating specific functionalities within the program.
2. Text Boxes: Allow users to input textual information such as file paths, variable names, or command parameters.
3. List Boxes and Combo Boxes: Enable selection from predefined options or lists, providing users with choices for customizing their generated Minecraft code.
4. Labels: Provide descriptive text to convey information or instructions to users regarding the purpose of adjacent controls.

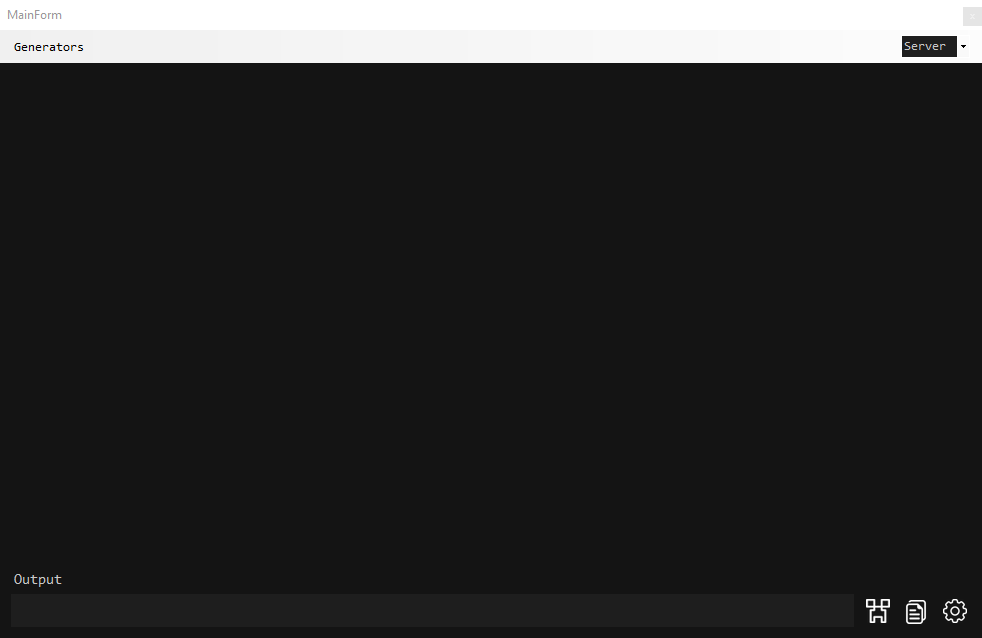
2.3 User Workflow

The Windows Forms UI is designed to streamline the user workflow for generating Minecraft code and command blocks. The typical workflow involves the following steps:

1. Initialization: Upon launching the application, the UI initializes with default settings and displays relevant options for customization
2. Configuration: Users can configure various parameters such as block types, command properties, and mod preferences through intuitive controls provided within the interface.
3. Generation: Once configurations are set, users initiate the code generation process by interacting with a designated button or menu option. The UI provides real-time feedback on the progress and completion of code generation tasks.
4. Validation and Preview: Before finalizing the generated code, users have the option to preview the output within the UI, allowing for validation and potential adjustments.
5. Execution: After satisfactory validation, users can proceed to execute the generated code directly within Minecraft or export it for further integration into their projects.

2.4 Visual Representation

Visual representation plays a crucial role in enhancing the appeal and usability of the Windows Forms UI. Leveraging aesthetically pleasing layouts, visually distinct controls, and appropriate use of icons and images not only enhances the overall user experience but also reinforces brand identity. Careful attention to design principles such as alignment, spacing, and hierarchy ensures a harmonious visual composition that resonates with users.



2.5 Utilization of Multiple Forms for Modular Functionality

The design of the proposed program for generating Minecraft code and command blocks incorporates a modular approach, leveraging multiple forms to encapsulate distinct generators, algorithms, and functions. This architectural decision enables the program to maintain a clear separation of concerns, enhance code organization, and facilitate the development of reusable components.

**Main Form as the Entry Point**

At the core of the application lies the main form, serving as the primary entry point and orchestrator of the program's functionality. The main form provides users with a centralized interface for navigating between different generators, accessing settings, and initiating code generation processes. Through the main form, users can seamlessly switch between various generator modules, each offering unique features and capabilities tailored to specific requirements.

**Individual Forms for Generator Modules**

The program comprises multiple individual forms, each dedicated to a specific generator module responsible for generating Minecraft code or command blocks based on predefined algorithms and functions. These generator forms encapsulate the logic, user interface elements, and processing capabilities associated with their respective functionalities, promoting modularity and maintainability.

2.6 Generator Forms

**Command Block Generator Form:** This form allows users to generate complex Command Blocks for executing custom actions, events, or scripts within the game world by specifying parameters such as command block parameters, targets, command block type, rotation, dimensions, and behavior. The form encapsulates the algorithms for generating block code based on user input, providing a visual representation of the resulting command blocks within the interface. Here is the algorithm:

1. **Initialization**

Declaring strings and integers.

1. **Parsing Input**

Reading various values from text fields and boxes.

Storing commands or ines in an array.

1. **Looping Through Grid**

Two nested „for“ loops iterate over a grid by rows and columns.

Within the loop, there are various conditions checking the state of checkboxes and altering the behavior of the loop accordingly.

1. **Building Commands**

Inside the loop, commands are constructed based on the current iteration.

Methods replace placeholders in the lines array with incremented values.

Constructs midpart string based on various conditions and concatenates it with the modified lines.

1. **Iterating and Resetting**

Increments k to move through the lines array.

Increments width.

Resets k to 0 if it reaches the end of the lines array.

1. **Finalization**

Final string is concatenated with various strings.

The method returns the constructed string.

**Potion Generator Form:** Users can utilize this form to create custom Minecraft potions, including splash potions, lingering potions, and tipped arrows. The form facilitates the selection of item properties, color, and potion duration, generating corresponding item code compatible with Minecraft's data format. Here is the algorithm:

1. **Namespace and Class Declaration**

The code is contained within the Command\_Block\_Generator.UI.Forms namespace.

The PotionGeneratorForm class inherits from the Form class and implements the IControlContainer and ICommandGenerator interfaces.

1. **Private Fields**

\_onAddToQueue: A delegate representing an action to be performed when items are added to a queue.

\_potion: An instance of the Potion class representing the potion being generated.

\_potionParser: An instance of an object implementing the IPotionParser interface for parsing potion data.

Several dictionaries for mapping between various potion properties and their corresponding identifiers or names.

1. **Constructor**

Initializes the form and assigns the provided action to the \_onAddToQueue field.

Instantiates a potion parser object and updates the potion tree display with the initial potion data.

1. **Event Handlers**

AddToQueueButton\_Click: Handles the click event of the "Add to Queue" button. It creates a new Effect object based on user input, adds it to the potion's effects list, generates commands based on the updated potion data, and invokes the action specified by \_onAddToQueue.

ClearButton\_Click: Handles the click event of the "Clear" button. It resets the potion data and updates the potion tree display.

UpdateButton\_Click: Handles the click event of the "Update" button. It updates the potion tree display with the current potion data.

RemoveButton\_Click: Handles the click event of the "Remove" button. It removes the selected effect from the potion's effects list and updates the potion tree display accordingly.

PotionTypeBox\_SelectedIndexChanged,RedBox\_TextChanged, GreenBox\_TextChanged, BlueBox\_TextChanged: Handle user input events for updating potion properties such as type and color.

RedBox\_Leave, GreenBox\_Leave, BlueBox\_Leave: Handle focus leaving events for ensuring valid input for color properties.

1. **Private Methods**

Various helper methods for updating the potion tree display, expanding nodes, parsing commands, and handling input validation.

2.7 Benefits of Modular Design

The modular design of the program offers several advantages in terms of flexibility, extensibility, and maintainability:

1. Scalability: New generator modules can be easily integrated into the program without affecting existing functionality. Developers can introduce additional forms for generating custom content, expanding the program's capabilities to accommodate evolving requirements and user preferences.
2. Reusability: Each generator form encapsulates self-contained logic and functionality, promoting code reuse and minimizing redundancy. Common algorithms and utilities can be abstracted into reusable components, facilitating consistency and efficiency across different modules.

3) Maintenance: The modular architecture simplifies maintenance and updates by isolating changes to specific components. Developers can troubleshoot issues, implement enhancements, and address bugs within individual forms without disrupting the overall operation of the program.

2.8 Functionality of Action Buttons

The user interface of the proposed program for generating Minecraft code and command blocks features a set of action buttons designed to streamline the user experience and facilitate common tasks related to code generation and execution. These action buttons serve as intuitive controls that enable users to interact with the program's functionality efficiently.

**Run Commands in Minecraft Button**

The "Run Commands in Minecraft" button triggers the execution of generated Minecraft commands directly within the game environment. Upon clicking the button, the program initiates communication with the Minecraft client, transmitting the generated commands for immediate execution within the player's game session. This seamless integration empowers users to test and validate their custom commands in real-time, facilitating rapid iteration and refinement of gameplay mechanics and features.

**Copy Command to Clipboard Button**

The "Copy Command to Clipboard" button provides users with the ability to copy the generated Minecraft command to the system clipboard with a single click. This functionality enhances accessibility and convenience, enabling users to quickly transfer the generated code to external applications, text editors, or command consoles for further customization, sharing, or storage. By simplifying the process of copying commands, the button streamlines the workflow and eliminates the need for manual selection and copying of text.

**Generate Command into Output Button**

The "Generate Command into Output" button triggers the generation of Minecraft commands based on user input and selected parameters, populating the output area of the program's interface with the generated code. Upon clicking the button, the program executes the underlying algorithms and logic associated with the selected generator module, dynamically generating command syntax tailored to the user's specifications. This instantaneous feedback empowers users to visualize and review the generated code in real-time, facilitating iterative refinement and adjustment of parameters to achieve desired outcomes.

2.9 Benefits of Action Buttons

The inclusion of action buttons within the user interface offers several benefits in terms of usability, efficiency, and productivity:

1. **Streamlined Workflow:** Action buttons provide users with clear and intuitive controls for executing common tasks, such as running commands, copying code, and generating output. By centralizing these functionalities within the interface, the program streamlines the user's workflow and minimizes the cognitive load associated with manual operations.
2. **Enhanced Productivity:** By automating repetitive tasks and reducing manual intervention, action buttons enhance user productivity and efficiency. Users can perform tasks such as code execution and copying with minimal effort, freeing up time and cognitive resources for more creative and strategic activities related to Minecraft modding and gameplay customization.
3. **Improved User Experience:** The availability of action buttons enhances the overall user experience by offering convenient and accessible controls for interacting with the program's functionality. Users can navigate the interface with confidence, knowing that essential actions are just a click away, resulting in a more satisfying and enjoyable interaction with the program.

3. MINECRAFT MODIFICATION

3.1 Forge Modding

Modding in Minecraft is made possible through various tools and platforms, the most prominent of which is the Minecraft Forge modding framework. Forge provides a robust and flexible platform for mod developers to create, distribute, and manage their modifications. It offers a comprehensive set of APIs (Application Programming Interfaces) that allow developers to hook into various aspects of the game, including blocks, items, entities, and more. This enables the creation of diverse mods ranging from simple tweaks and additions to complex overhauls that introduce entirely new gameplay mechanics and features.

The modding community for Minecraft is incredibly diverse, encompassing developers of all skill levels and interests. From hobbyists creating mods in their spare time to professional developers producing polished experiences, there's something for everyone in the world of Minecraft modding. This diversity is reflected in the wide range of mods available, covering everything from new biomes and dimensions to advanced automation systems and magic spells.

Modding also plays a significant role in the educational sphere, with many educators leveraging Minecraft as a platform for teaching programming, game design, and other STEM (Science, Technology, Engineering, and Mathematics) subjects. By engaging with modding, students not only learn valuable technical skills but also foster creativity and problem-solving abilities in a fun and engaging environment.

However, modding is not without its challenges. The constant evolution of the Minecraft platform, coupled with the complexity of the game's codebase, can make mod development a daunting task for newcomers. Additionally, ensuring compatibility between mods can be a delicate balancing act, requiring careful coordination and communication within the modding community.

Despite these challenges, modding continues to thrive in the Minecraft ecosystem, fueled by the passion and creativity of its dedicated community. Whether it's creating custom maps, crafting new items, or designing intricate redstone contraptions, modding empowers players to shape their Minecraft experience according to their imagination, ensuring that the world of Minecraft remains vibrant and ever-evolving.

3.2 Community and Resources

Forge modding benefits from a vibrant and supportive community of developers, enthusiasts, and contributors. Various online forums, communities, and resources provide valuable assistance, tutorials, and documentation for modders at all skill levels. Additionally, Forge itself is open-source, enabling collaboration and contributions from the community to continuously improve and expand the modding platform.

3.3 Framework and Architecture

Forge modding operates within the Java environment, leveraging Minecraft's Java Edition as its base platform. The framework consists of various components such as libraries, hooks, and events that facilitate the interaction between custom modifications and the core game logic.

**Libraries:**

Forge provides a set of libraries that extend Minecraft's capabilities, enabling modders to access and manipulate various game elements programmatically. These libraries encompass functionalities for entity management, world generation, item creation, and more.

**Hooks:**

Hooks in Forge serve as entry points for injecting custom code into the game's execution flow. Modders can utilize hooks to intercept specific events or actions within Minecraft, allowing for the implementation of custom behaviors or features.

**Events:**

Forge employs an event-driven architecture, where specific actions or occurrences trigger corresponding events that can be intercepted and handled by modders. This event system enables seamless integration of custom logic with the game's existing mechanics

3.4 Development Workflow

Forge modding typically follows a structured development workflow, involving the following steps:

**Setup and Configuration:**

Modders begin by setting up their development environment, which involves installing the Forge Mod Development Kit (MDK) and configuring their Integrated Development Environment (IDE) for Java development.

**Mod Initialization:**

The creation of a new mod entails initializing the mod project within the Forge MDK, which generates the necessary project structure and boilerplate code.

**Code Implementation:**

Modders implement their desired features and functionalities by writing custom Java code within designated mod classes. This involves utilizing Forge's APIs and event system to interact with the game's mechanics and assets.

**Testing and Debugging:**

Throughout the development process, modders conduct rigorous testing and debugging to ensure the stability and compatibility of their modifications with different Minecraft environments and configurations.

**Deployment:**

Once the mod is deemed stable and functional, it can be packaged into a distributable format and released to the Minecraft modding community through platforms such as CurseForge or the Minecraft Forums.

4. CONFIGURING A SERVER

5. REQUEST TO SERVER

5.1 Instructions from a client to the server

In the context of Minecraft modding and server administration, making a request typically refers to sending a command or instruction from a client (e.g., a player's game client) to the server, triggering a specific action or response within the game environment. Requests can encompass various actions, including:

1. **Executing Commands:** Players and server administrators can send commands to the server using the in-game chat interface or server console. These commands can range from simple actions like teleportation or item manipulation to more advanced operations involving mod-specific features.
2. **Interacting with Mods:** Modded servers may support custom commands and interactions unique to the installed modifications. Players can make requests to trigger mod-specific functionalities, such as spawning custom entities, modifying game mechanics, or executing scripted events.
3. **Modifying World State:** Requests can also involve modifying the state of the game world, including terrain generation, block placement, entity behavior, and player interactions. These modifications can be initiated through player actions, server-side scripts, or modded gameplay mechanics.
4. **Server Administration:** Server administrators may make requests to manage server settings, monitor player activity, enforce rules and regulations, and troubleshoot technical issues. These requests typically involve administrative commands and tools provided by the server software.

**5.2 Request code breakdown**

This C# code defines an asynchronous method named SendCommands that sends a list of strings as commands over a TCP connection to a server.



Let's break down the code step by step:

1. **Method Signature**

private async void SendCommands(List<string> commands): This method takes a list of strings (commands) as input and returns void. It's marked as async, indicating that it's an asynchronous method.

1. **Task.Run**

await Task.Run(() => { ... });: This line uses Task.Run to offload the execution of the code inside the lambda expression to a background thread. This is done to prevent blocking the UI thread, as the method involves network I/O, which can be time-consuming.

1. **Network Communication**

TcpClient client = new TcpClient(serverIp, port);: Creates a new TCP client and establishes a connection to the server specified by the serverIp and port variables. NetworkStream stream = client.GetStream();: Retrieves the network stream associated with the TCP client for sending and receiving data.

1. **Construct Command String**

string result = "";: Initializes an empty string to store the concatenated commands. for (int i = 0; i < commands.Count; i++) { result += commands[i] + "\n"; }: Iterates through the list of commands and concatenates them into the result string, separated by newline characters.

1. **Convert Data to Bytes and Send**

byte[] data = Encoding.UTF8.GetBytes(result);: Converts the string result to a byte array using UTF-8 encoding, as network communication typically deals with bytes. stream.Write(data, 0, data.Length);: Writes the byte array to the network stream to send the commands to the server.

1. **Cleanup**

stream.Socket.Close();: Closes the underlying socket associated with the network stream. client.Dispose(); stream.Dispose();: Disposes of the TCP client and network stream objects to release associated resources.

6. END RESULT

CONCLUSION

SOURCE MATERIALS

Hejlsberg, A. (2003). The C# Programming Language. Microsoft Press.

Fowler, M. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley.

Persson, M. (2011). Minecraft. Mojang Studios. <https://www.minecraft.net/>

Player Counter. (2023). Minecraft Live Player Count. [Minecraft Player Count - How Many People Are Playing Now? (playercounter.com)](https://playercounter.com/minecraft/)

Microsoft. (2023, 26. jaanuar). Create a Windows Forms app in Visual Studio with C#. [Create a Windows Forms app with C# - Visual Studio (Windows) | Microsoft Learn](https://learn.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022)

Bergensten, J. (2012). Minecraft Redstone Update. Mojang Studios.

Sussman, G. J. (2012). Clojure Programming. O'Reilly Media.

Microsoft. (2023, 9. jaanuar). C# documentation. [C# docs - get started, tutorials, reference. | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/csharp/)

FML Team. (2012). Minecraft Forge. <https://files.minecraftforge.net/>

Richards M. (2015). Software Architecture Patterns. Goodreads.

Nygard M. T. (2017). Design and Deploy Production-Ready Software. O'Reilly Media.