TARTU UNIVERSITY
FACULTY OF SOCIAL SCIENCES

NARVA COLLEGE
STUDY PROGRAM “INFORMATION SYSTEMS DEVELOPMENT“

Daniel Neustus

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

Bachelor’s Thesis

Supervisor: Deniss Ruder

NARVA 2024

TARTU ÜLIKOOL
SOTSIAALTEADUSTE VALDKOND

NARVA KOLLEDŽ
“INFOTEHNOLOOGILISTE SÜSTEEMIDE ARENDUS“

Daniel Neustus

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

Lõputöö

Juhendaja: Deniss Ruder

NARVA 2024

**Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks**

Mina, Daniel Neustus,

1. Annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose:

   „Programm Minecrafti käskude ja käsuplokkide genereerimiseks kasutades C# Windows Forms ja Forge moddingut”,

   mille juhendaja on Deniss Ruder, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.
3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.
4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

*Daniel Neustus*

***17.05.2024***

# TABLE OF CONTENTS

# LIST OF TERMS AND ABBREVIATIONS

**C#** – The programming language used for developing the Windows Forms application.

**Windows Forms** – A graphical application framework in the Microsoft .NET framework for creating Windows desktop applications.

**Forge** – Modding framework for Minecraft, allowing the creation of custom mods using Java.

**Java** – The programming language used for Minecraft modding and potentially other components of the project.

**Minecraft** – A sandbox video game where players can build and explore virtual worlds made up of blocks.

**Command Blocks** – In Minecraft, a block that can execute commands in the game world.

**IDE** – Integrated Development Environment, such as Visual Studio, used for writing, debugging, and compiling code.

**API** – Application Programming Interface, a set of rules and protocols for building and interacting with software applications.

**GUI** – Graphical User Interface, the visual elements and interactions used for human-computer communication.

**TCP/IP** – Transmission Control Protocol/Internet Protocol, a suite of communication protocols used for network communication.

**JAR** – Java Archive, a file format used to package Java class files, associated metadata, and resources into a single file.

# 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 Minecraft commands and command blocks. As games evolve and new features are added, the need for efficient and dynamic methods becomes increasingly important (Fowler, M. 2002).

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

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

1) **Efficiency and automation:** Designing a system significantly reduces the time and effort required to generate Minecraft commands and command blocks.
2) **User-friendly interface:** Creating an intuitive and user-friendly interface suitable for all users.

Several smaller tasks must be completed throughout the development process to achieve the stated objectives. These tasks include both technical aspects and user experience:

1) **Technical background of Minecraft modification:** Explain the concept of Minecraft modification and why it is essential for the game's development.
2) **Programming language:** Develop strong integration with the Java and C# programming language, ensuring compatibility with Minecraft modification frameworks and programming languages.
3) **User interface design:** Come up with logical and understandable user interface using Windows Forms, facilitating easy navigation and interaction.
4) **Logic for command generation:** Implement algorithms and logic to generate Minecraft commands and command blocks based on user input and specifications, covering a wide range of in-game elements.

# 1. BACKGROUND / THEORETICAL FRAMEWORK

## 1.1 Command Blocks

### 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 commands.

Command blocks execute commands written in the Minecraft command language, enabling manipulation of the game world. (Persson, M., 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 using command blocks to solve repetitive tasks or create autonomous in-game systems. (Bergensten, J., 2012) 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 commands.

## 1.2 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, A., 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. Microsoft’s official documentation (Microsoft, 2023) will also be used to help develop the app.

### 1.2.2 Minecraft commands

Minecraft commands are a set of instructions that players can use to manipulate the game world, interact with entities, and perform various tasks within the game. These commands can be typed into the chat window or executed via command blocks.

## 1.3 Windows Forms

### 1.3.1 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, allowing developers to create rich and interactive user interfaces easily. Microsoft's official tutorials (Microsoft, 2024) will be used to develop the app's interface.

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

### 1.3.2 Integration with Other Technologies

Windows Forms seamlessly integrates with other technologies and frameworks, allowing developers to use the full power of the .NET ecosystem. Whether 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.

## 1.4 Java Modding

### 1.4.1 Forge

Forge, developed by the FML Team (FML Team 2012), is a popular modding framework for Minecraft that allows developers to create and customize game modifications (mods) easily. Developed in Java, Forge provides robust 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 effectively understand and utilize the framework.

### 1.4.2 IntelliJ IDEA

JetBrains developed IntelliJ IDEA, a robust integrated development environment (IDE) commonly used for Java development, including Minecraft modding. Its robust features, intelligent code assistance, and seamless integration with build systems make it a preferred choice among developers venturing into Minecraft modding. Oracle's official documentation (Oracle, 2023) on the Java language will be used to help develop the mod.

# 2. APPLICATION

## 2.1 Functional and Non-Functional Requirements

Before we get into the technical details of the app, it's important to list the things the application needs to do (functional requirements) and the qualities it needs to have (non-functional requirements).

### Functional Requirements

1. **Navigation:** The application's user interface should provide intuitive navigation between different forms and modules, allowing users to switch between potion generation, command generation and other generator functionalities seamlessly.
2. **Input Validation:** The app should validate user input to ensure that only valid values are accepted for potion properties, block attributes, and command parameters. Error messages should be displayed for invalid input.
3. **Command Generation:** The app should facilitate the generation of Minecraft commands and command blocks based on user input and selected parameters. Generated code should accurately reflect the user's specifications for potions, blocks, and commands.
4. **Integration with Minecraft Client:** The app should support integration with the Minecraft client or mod, allowing users to execute generated commands directly within the game environment for testing and validation purposes.
5. **Feedback Mechanisms:** The user interface should provide feedback to users on the status of their actions, such as successful code generation, errors in input validation, or completion of tasks.
6. **Customization Options:** The app should offer customization options for users to adjust settings, preferences, and appearance according to their preferences, such as theme selection, font size, and layout customization.

### Non-Functional Requirements

1. **Usability:** The app should be user-friendly, intuitive, and easy to navigate, catering to users with varying technical expertise and familiarity with Minecraft modding.
2. **Performance:** The app should be responsive and performant, with minimal latency and loading times, even when handling complex operations such as code generation for large-scale Minecraft projects.

3. **Reliability:** The app should be reliable and robust, capable of handling errors gracefully and recovering from unexpected failures without data loss or corruption.
4. **Scalability:** The app should be scalable and capable of accommodating future enhancements, updates, and expansions to accommodate evolving user requirements and technological advancements.
5. **Compatibility:** The app should be compatible with various Windows operating systems and hardware configurations, ensuring seamless performance across different devices and environments.

## 2.2 Design Considerations

Choosing Windows Forms as the GUI framework for the application is due to familiarity, integration capabilities, efficiency in development, and community support. By using Windows Forms in the .NET ecosystem, developers can make user interfaces that are easy to use and have many features, which improves how well the application works. We also need to think about how the program will work on different platforms and how we can involve the community. Windows Forms is a good choice for meeting the goals and needs of the project.

## 2.3 Components and Controls

Windows Forms provides abundant components and controls that facilitate the creation of rich, interactive interfaces. Some of the critical 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:** Users can 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, allowing users to customize their generated Minecraft code.
4. **Labels:** Provide descriptive text to convey information or instructions to users regarding the purpose of adjacent controls.

## 2.4 User Workflow

The Windows Forms UI is designed to improve the user workflow for generating Minecraft commands 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:** Users initiate the code generation process once configurations are set 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 can preview the output within the UI, allowing for validation and potential adjustments.
5. **Execution:** After satisfactory validation, users can execute the generated code directly within Minecraft or export it for further integration into their projects.

## 2.5 Visual Representation

Visual representation plays a crucial role in enhancing the appeal and usability of the Windows Forms UI. Using pleasing layouts, visually distinct controls, and appropriate use of icons and images enhances the overall user experience and reinforces brand identity. Careful attention to design principles such as alignment, spacing, and hierarchy ensures good visual composition that resonates with users as shown in (Figure 1).


***Figure 1.*** *The main application window.*

## 2.6 Utilization of Multiple Forms for Modular Functionality

The design of the application for generating Minecraft commands and command blocks incorporates a modular approach, using 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 of the program's functionality. The main form provides users with a centralized interface for navigating between generators, accessing settings, and initiating code generation processes. Users can seamlessly switch between various generator modules through the main form, each offering unique features and capabilities tailored to specific requirements.

**Individual Forms for Generator Modules**

The program consists of 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.

## 2.7 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, visually representing the resulting command blocks within the interface (Figure 2). Minecraft version 1.20.1 has been chosen as the command syntax basis. The original code is in Github. Here is the algorithm breakdown:

1) **Initialization**

   Declaring strings and integers.

2) **Parsing Input**

   Reading various values from text fields and boxes. Storing commands or ines in an array.

3) **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.

4) **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.

5) **Iterating and Resetting**

Incrementing variable to move through the lines array. Increments width. Reset variable to 0 if it reaches the end of the lines array.

6) **Finalization**

Final string is concatenated with various strings. The method returns the constructed string.

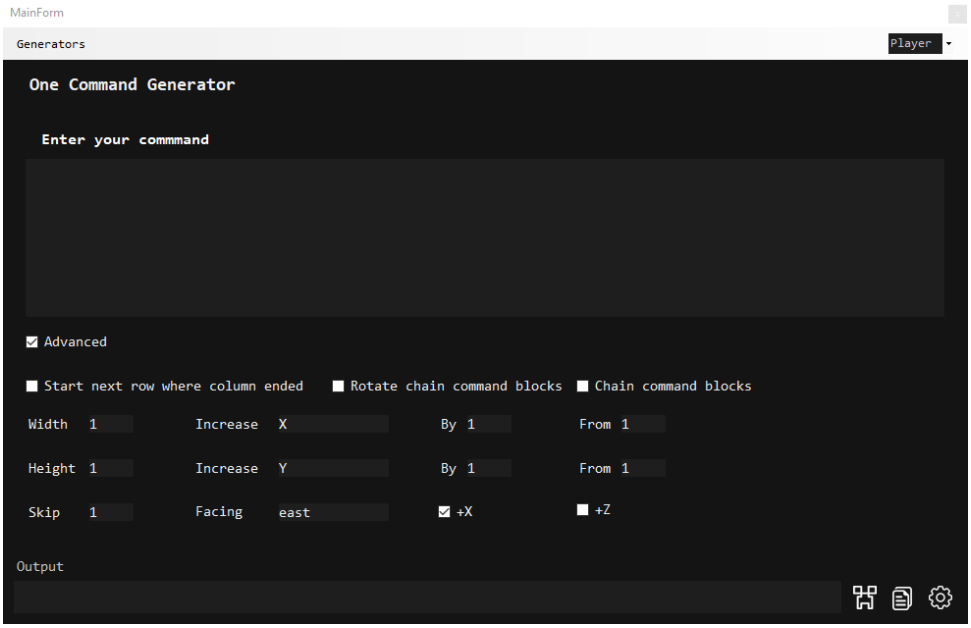


***Figure 2.*** *Command Block Generator Form.*

**Potion Generator Form:** Users can utilize this form to create custom Minecraft potions, including splash potions, lingering potions, and tipped arrows (Figure 3). The form facilitates the selection of item properties, color, and potion duration, generating corresponding item codes compatible with Minecraft's data format. The original code is in Github. Here is the algorithm breakdown:

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.

2) **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.

3) **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.

4) **Event Handlers**

**“AddToQueueButton_Click”:** Handles the click event of "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.

**“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.

**“RedBox_Leave”, “GreenBox_Leave”, “BlueBox_Leave”:** Handle focus leaving events for ensuring valid input for color properties

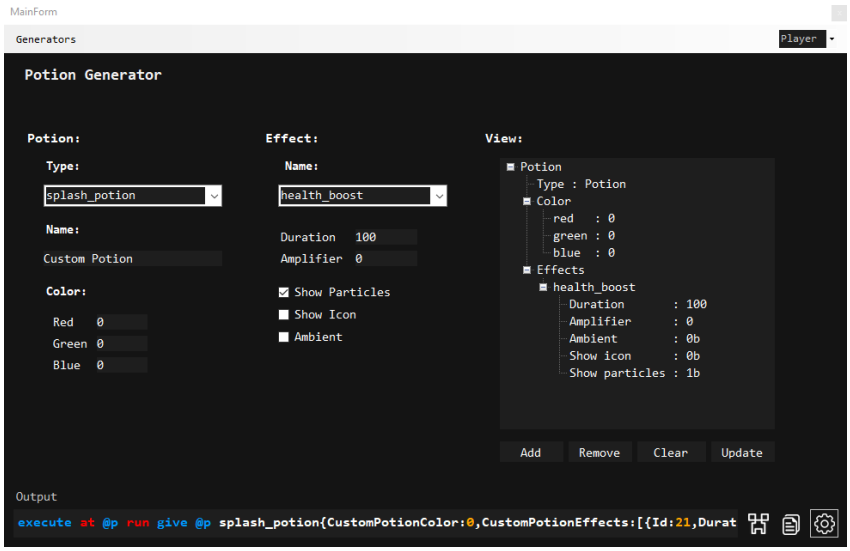


***Figure 3.*** *Potion Generator Form.*

## 2.8 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. We 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. Standard algorithms and utilities can be abstracted into reusable components, facilitating module consistency and efficiency.
3. **Maintenance:** The modular architecture simplifies maintenance and updates by isolating changes to specific components. We can troubleshoot issues, implement enhancements, and address bugs within individual forms without disrupting the program's overall operation.

## 2.9 Functionality of Action Buttons

The user interface of the application features a set of action buttons designed to improve the user experience and facilitate mandatory tasks related to command generation and execution. These action buttons serve as intuitive controls that enable users to interact with the program's functionality efficiently.

### Run Commands in the 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 communicates 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 allows users 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 improves 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, 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.

## 2.10 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 straightforward and intuitive controls for executing mandatory tasks, such as running commands, copying code, and generating output. The program improves the user's workflow by centralizing these functionalities within the interface and minimizes the cognitive load associated with manual operations.

2. **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 confidently navigate the interface, 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 Environment

### IntelliJ IDEA

IntelliJ IDEA is a robust integrated development environment (IDE) for Java, providing robust features and tools conducive to efficient and organized coding practices. IntelliJ has been chosen because of its familiarity, integration capabilities, efficiency in development, and community support. IntelliJ IDEA is an invaluable asset in the context of developing a Forge mod for Minecraft, offering a seamless development experience coupled with its intuitive user interface.

#### Setting Up the Environment

Upon launching IntelliJ IDEA, the first step entails creating a new project tailored for Minecraft Forge modding. This involves configuring the necessary dependencies and building settings to ensure compatibility with the Minecraft environment. Using the built-in Gradle support simplifies this process, as it automates the setup of project dependencies and facilitates seamless integration with Forge.

#### Creating Mod Components

With the project structure in place, we can create the mod components. IntelliJ IDEA's robust code editing features, including syntax highlighting, code completion, and intelligent refactoring tools, improve writing clean and concise code. Whether crafting custom blocks, items, or entities or implementing game mechanics, IntelliJ IDEA gives developers the power to iterate rapidly while maintaining code quality.

#### Debugging and Testing

Debugging constitutes a critical aspect of the development lifecycle, enabling developers to identify and rectify errors efficiently. IntelliJ IDEA offers comprehensive debugging capabilities, allowing developers to set breakpoints, inspect variables, and trace program execution seamlessly. This facilitates the identification of bugs and ensures the stability and reliability of the mod.

## 3.2 Mod Functionality

1. **Command Execution:**

   The mod should be able to execute Minecraft commands received from the C# Windows Forms application. It should interpret commands accurately and perform corresponding actions within the Minecraft server environment.

2. **Custom Command Support:**

   The mod should support custom commands defined within its codebase. It should allow for the registration of new commands to extend its functionality.

3. **Communication Protocol:**

   The mod should establish a reliable communication channel with the Windows Forms application. It should utilize a specified port for receiving commands and data from external sources.

4. **Compatibility:**

   The mod should be compatible with the Minecraft Forges modding framework. It should function seamlessly alongside other mods installed within the Minecraft server environment.

5. **Performance:**

   The mod should execute commands efficiently, minimizing any delays or lag within the Minecraft server environment. It should have low resource overhead to avoid impacting server performance negatively.

6. **Reliability:**

   The mod should be robust and reliable, capable of handling various scenarios without failure. It should gracefully handle errors and exceptions to prevent server crashes or instability.

7. **Scalability:**

   The mod should be scalable, capable of handling a large volume of incoming commands without degradation in performance. It should accommodate potential future expansions or updates without significant refactoring.

## 3.3 Community and Resources

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 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. Forge 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, containing developers of all skill levels and interests. From hobbyists creating mods in their spare time to professional developers producing polished experiences, there is something for everyone in 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.

However, modding has its challenges. The constant evolution of the Minecraft platform and 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 is 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 colorful and ever-evolving.

## 3.4 Framework and Architecture

Forge modding operates within the Java environment, using Minecraft's Java Edition as its base platform. The framework comprises various components such as libraries, hooks, and events facilitating 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 have functionalities for entity management, world generation, item creation, and more.

### Hooks

Hooks in Forge are 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 custom behaviors or features to be implemented.

### Events

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

## 3.5 Forge Mod Code Structure

### Mod Initialization

The mod begins by initializing itself within the Minecraft server environment. This initialization process includes setting up necessary event listeners, registering custom commands, and establishing communication channels to receive external instructions.

### Command Handling

The mod includes a command handling system for processing incoming commands through the communication channel. Upon receiving a command, the mod parses the command and executes the corresponding action within the Minecraft server environment.

### Communication with Windows Forms Application

The mod establishes communication with the C# Windows Forms application by listening on a predefined port. Upon receiving data from the application, the mod processes the incoming instructions and executes the corresponding actions within the Minecraft server environment.

# 4. INTERFACING MINECRAFT THROUGH MOD

At the heart of this integration lies a Forge mod crafted to connect the communication chasm between the Minecraft server and the application. This mod is endowed with the capability to interpret and execute commands received from external sources, thereby extending the traditional boundaries of Minecraft gameplay. Central to its functionality is establishing a dedicated server instance augmented with a listening port, where incoming commands are received and processed.

The Windows Forms application serves as the conduit through which users interact with the Minecraft environment. Through an intuitive graphical user interface (GUI), users can input commands and directives, which are then transmitted to the designated port of the Forge mod.

A crucial aspect of this integration is establishing a robust communication protocol between the Windows Forms application and the Forge mod. Using networking capabilities inherent to both platforms, commands are transmitted over TCP/IP or UDP protocols, ensuring reliable and efficient data exchange. Furthermore, measures are implemented to handle potential latency issues and mitigate communication errors, guaranteeing a smooth and uninterrupted user experience.

## 4.1 Port Configuration within the Forge Mod

A designated port is the entry point for external commands within the Forge mod. This port is configured to be receptive to incoming data packets, ensuring that any data sent to this port is received and processed by the mod. Through this configuration, the Forge mod establishes a communication channel with external application, enabling the seamless transmission of instructions to be executed within the Minecraft environment.

## 4.2 Execution of Commands within the Minecraft Server

Upon receiving data packets containing commands and command block code, the Forge mod processes the incoming instructions within the Minecraft server environment. Using the capabilities of the modding framework, the received commands are executed within the game world, effecting changes and interactions as dictated by the transmitted data.

## 4.3 Server side request handling

In 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 do various actions, including:

1. **Executing Commands:** Players and server administrators can send commands to the server using the application or mods. These commands 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 or application scripts.

## 4.4 Request code breakdown

The Java code used in the mod uses an asynchronous method named SendCommands that sends a list of strings as commands over a TCP connection to a server. Minecraft version 1.20.1 has been chosen for the mod. The end code will be compiled into a JAR file.

**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.

**2) 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.

3) **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.

4) **Construct Command String**

**„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.

5) **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.

6) **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.

# 5. TEST RESULTS

Screenshots were captured during the testing phase to visually demonstrate the effectiveness of the generated commands and command blocks. These screenshots are showcase of the program's capabilities and its impact on Minecraft.

The first screenshot (Figure 4) here depicts the process of making a custom potion. The potion is given in the output, which can be copied or, by pressing the run button, can be given to the player.

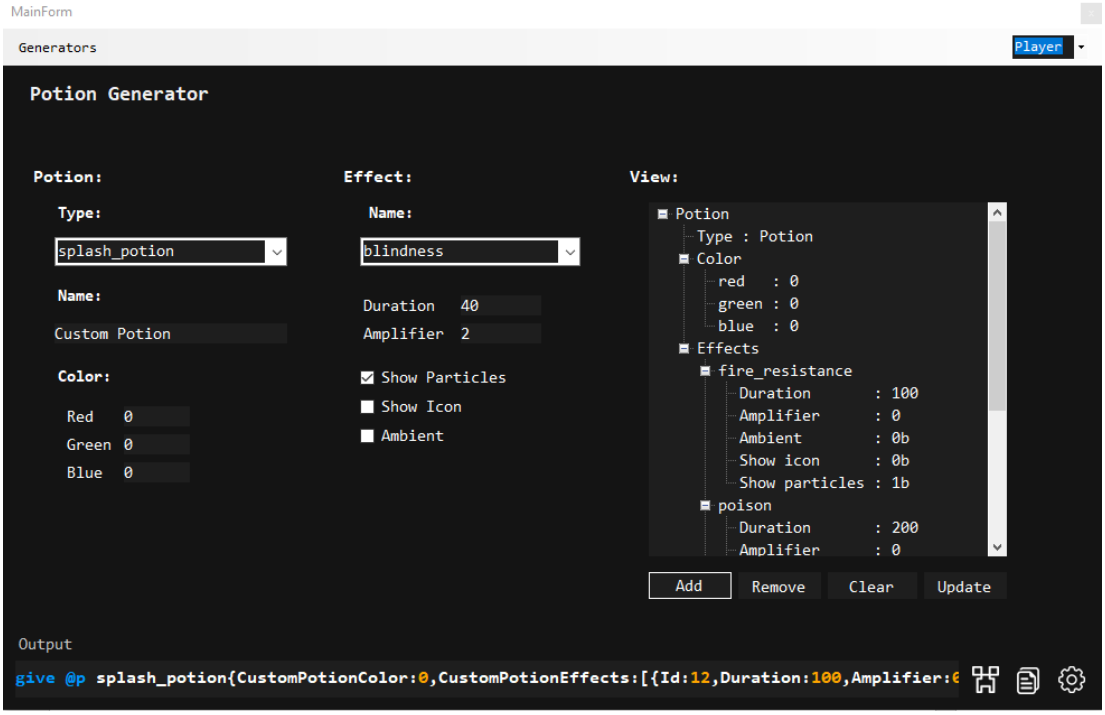


***Figure 4.*** *Potion Generator Form with various arguments.*

The next screenshot (Figure 5) showcases the Command Generator Form, in which multiple commands can be written in the first text box, and various arguments can be added to modify the output command. Let’s make the command block box 10 in width and 10 in height and add variable X, which will be incremented by 1.

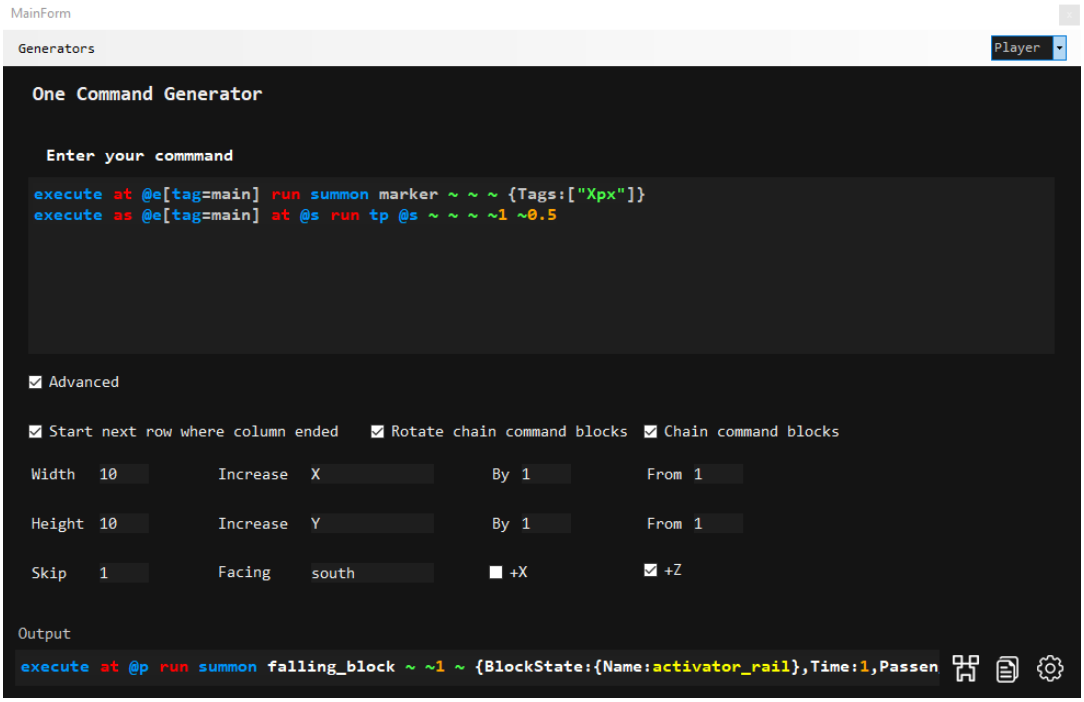


***Figure 5.*** *Command Block Generator Form with various arguments.*

By pressing the Run in Minecraft button, the command blocks appear like this (Figure 6).

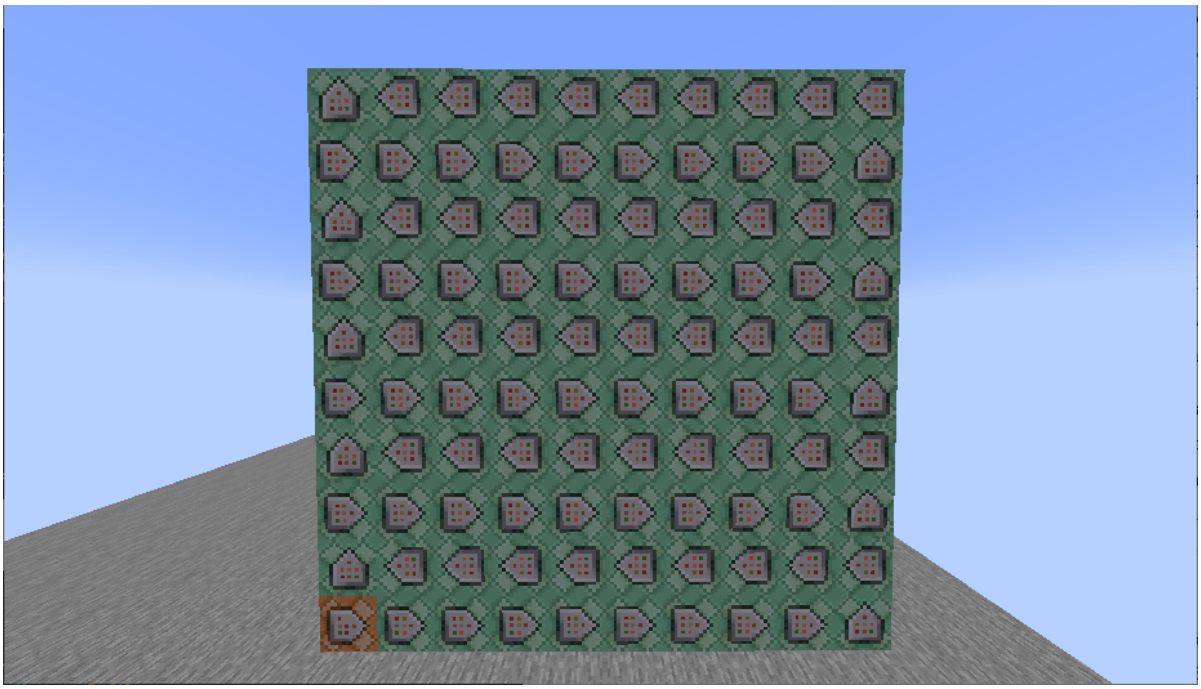

***Figure 6.*** *Screenshot taken in Minecraft showing the generated command blocks.*

The last screenshot (Figure 7) shows the commands inside the generated command blocks.


***Figure 7.*** *Screenshot taken in Minecraft showing the generated command blocks inside.*

# CONCLUSION

Through this project, we have demonstrated the potential of programming languages like C# and modding frameworks like Forge to support Minecraft players with tools to create complex gameplay mechanics, structures, and interactions within the game environment.

One of this work's key contributions is simplifying and improving the command making process. By providing a user-friendly interface through Windows Forms, we have lowered the barrier of entry for aspiring map makers and coders, allowing them to focus more on their creative ideas rather than getting bogged down by technical complexities. Furthermore, the integration with Forge Modding ensures compatibility and interoperability with the vast ecosystem of existing Minecraft mods.

Adding more generator forms to the program is worth considering as we look towards the future. These additional forms could cater to specific aspects of Minecraft gameplay, such as world generation, mob behavior, or even custom item creation. By diversifying the types of generators available, we can provide users with even greater flexibility and customization options, aiding them to realize a broader range of creative visions within the Minecraft universe.

Additionally, this program facilitates the generation of Minecraft commands and command blocks, enabling users to automate repetitive tasks and expedite the development process. This enhances productivity, encourages experimentation and innovation within the Minecraft community. By efficiently aiding users to translate their ideas into in-game experiences, we grow a culture of creativity and collaboration that enriches the Minecraft gameplay experience for players worldwide.

There are several avenues for future research and development in this field. Further refinement and expansion of the program's features could enhance its versatility and utility for a broader range of Minecraft map making or modding projects. Additionally, exploring ways to integrate advanced functionalities such as artificial intelligence or machine learning algorithms could open up new possibilities for interesting and adaptive gameplay experiences within Minecraft.

# RESÜMEE

Selle projekti kaudu oleme demonstreerinud programmeerimiskeelte nagu C# ja modifitseerimise raamistike, näiteks Forge'i, potentsiaali, et anda Minecraft'i mängijatele tööriistu keerukate mängumehaanikate, struktuuride ja interaktsioonide loomiseks mängukeskkonnas.

Üks selle töö peamisi panuseid on käskude tegemise protsessi lihtsustamine ja sujuvamaks muutmine. Pakkudes Windows'i vormide kaudu kasutajasõbralikku liidest, oleme alandanud ambitsioonikate kaarditegijate ja kodeerijate sisenemisbarjääri, võimaldades neil keskenduda rohkem oma loomingulistele ideedele, mitte jääda tehnilistesse keerukustesse. Lisaks tagab integreerimine Forge Moddingiga ühilduvuse ja koostalitlusvõime olemasolevate Minecraft'i modifikatsioonide tohutu ökosüsteemiga.

Lisaks tasub tulevikku vaadates kaaluda programmi generaatorivormide lisamist. Need lisavormid võivad rahuldada Minecraft'i mängu teatud aspekte, nagu maailma genereerimine, mobi käitumine või isegi kohandatud esemete loomine. Mitmekesistades saadaolevate generaatorite tüüpe, saame pakkuda kasutajatele veelgi suuremat paindlikkust ja kohandamisvõimalusi, võimaldades neil realiseerida laiemat valikut loomingulisi nägemusi Minecraft'i universumis.

Lisaks hõlbustab see programm Minecraft'i käskude ja käsuplokkide genereerimist, võimaldades kasutajatel korduvaid ülesandeid automatiseerida ja arendusprotsessi kiirendada. See suurendab tootlikkust, julgustab eksperimenteerimist ja innovatsiooni Minecraft'i kogukonnas. Võimaldades kasutajatel tõhusalt muuta oma ideid käegakatsutavaks mängusiseseks kogemuseks, kasvatame loovuse ja koostöö kultuuri, mis rikastab Minecraft'i mängukogemust kogu maailmas.

Selles valdkonnas on tulevaseks uurimis- ja arendustegevuseks mitu võimalust. Programmi funktsioonide edasine täiustamine ja laiendamine võib suurendada selle mitmekülgsust ja kasulikkust laiema valiku Minecraft'i kaartide koostamise või muutmise projektide jaoks. Lisaks võib täiustatud funktsioonide, näiteks tehisintellekti või masinõppe algoritmide integreerimise võimaluste uurimine avada uusi võimalusi dünaamilisteks ja adaptiivseteks mängukogemusteks Minecraft'is.

# SOURCE MATERIALS


Hejlsberg, A., Wiltamuth, S., Golde, P.(2003). The C# Programming Language. *Addison-Wesley*. ISBN-10: 9780321154910.

Fowler, M. (2002). Patterns of Enterprise Application Architecture. *Addison-Wesley*. ISBN-10: 0321127420.

Persson, M. (2011). Minecraft. Mojang Studios. https://www.minecraft.net/ (Accessed: 6 January 2024).

Microsoft. (2024). Create a Windows Forms app in Visual Studio with C#. https://learn.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022 (Accessed: 10 January 2024).

Bergensten, J. (2012). New In Minecraft: 1.5 Redstone Update. *Gamespot*. New In Minecraft: 1.5 Redstone Update - GameSpot (Accessed: 20 February 2024).

Microsoft. (2023). C# documentation. C# docs - get started, tutorials, reference. | Microsoft Learn (Accessed: 7 January 2024).

FML Team. (2012). Minecraft Forge. https://files.minecraftforge.net/ (Accessed: 15 February 2024).

JetBrains. (2024). Intellij official webiste. IntelliJ IDEA – the Leading Java and Kotlin IDE (jetbrains.com) (Accessed: 15 February 2024).

Github. (2024). Application and mod code. den1k0/Thesis (github.com) (Accessed: 19 May 2024).

Oracle. (2024). Java Documentation. https://docs.oracle.com/en/java/ (Accessed: 15 February 2024).