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: Deniss Ruder

NARVA 2024

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: Deniss Ruder

NARVA 2024

**TABLE OF CONTENTS**

[LIST OF TERMS AND ABBREVIATIONS 5](#_Toc164807688)

[INTRODUCTION 6](#_Toc164807689)

[RESEARCH GOALS 7](#_Toc164807690)

[1. BACKGROUND / THEORETICAL FRAMEWORK 7](#_Toc164807691)

[1.1 Command Blocks 7](#_Toc164807692)

[1.1.1 Overview of Minecraft Command Blocks 7](#_Toc164807693)

[1.1.2 Command Block Automation 7](#_Toc164807694)

[1.2 C# Programming Language 8](#_Toc164807695)

[1.2.1 Introduction to C# 8](#_Toc164807696)

[1.2.2 Minecraft code 8](#_Toc164807697)

[1.3 Code Generation Techniques 8](#_Toc164807698)

[1.3.1 Code Generation in Software Development 8](#_Toc164807699)

[1.3.2 Dynamic Code Generation in C# 8](#_Toc164807700)

[1.4 Windows Forms 8](#_Toc164807701)

[1.4.1 Windows Forms in Software Development 8](#_Toc164807702)

[1.4.2 Integration with Other Technologies 9](#_Toc164807703)

[1.5 FORGE 9](#_Toc164807704)

[2. USER INTERFACE 10](#_Toc164807705)

[2.1 Design Considerations 10](#_Toc164807706)

[2.2 Functional and Non-Functional Requirements of UI 10](#_Toc164807707)

[2.3 Components and Controls 11](#_Toc164807708)

[2.4 User Workflow 12](#_Toc164807709)

[2.5 Visual Representation 12](#_Toc164807710)

[2.6 Utilization of Multiple Forms for Modular Functionality 13](#_Toc164807711)

[2.7 Generator Forms 13](#_Toc164807712)

[2.8 Benefits of Modular Design 16](#_Toc164807713)

[2.9 Functionality of Action Buttons 17](#_Toc164807714)

[2.10 Benefits of Action Buttons 18](#_Toc164807715)

[3. MINECRAFT MODIFICATION 19](#_Toc164807716)

[3.1 Environment 19](#_Toc164807717)

[3.2 Functional and Non-Functional Requriements of the Mod 20](#_Toc164807718)

[3.2 Community and Resources 21](#_Toc164807719)

[3.3 Framework and Architecture 22](#_Toc164807720)

[3.4 Forge Mod Code Structure 23](#_Toc164807721)

[4. INTERFACING MINECRAFT THROUGH MODS 24](#_Toc164807722)

[4.1 Port Configuration within the Forge Mod 24](#_Toc164807723)

[4.2 Execution of Commands within the Minecraft Server 24](#_Toc164807724)

[5. REQUEST TO MOD SERVER PORT 25](#_Toc164807725)

[5.1 Instructions from a client to the server 25](#_Toc164807726)

[5.2 Request code breakdown 26](#_Toc164807727)

[6. TEST RESULTS 27](#_Toc164807728)

[CONCLUSION 30](#_Toc164807729)

[SOURCE MATERIALS 31](#_Toc164807730)

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 your 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.

JSON – JavaScript Object Notation, a lightweight data-interchange format commonly used for data serialization.

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.

API – Application Programming Interface, a set of tools and protocols for building software applications.

SDK – Software Development Kit, a collection of tools and libraries used for developing software applications.

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 the underlying codes and command blocks. As games evolve and new features are added, the need for efficient and dynamic code generation methods 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 creating complex structures, functions, games, and applications in Minecraft, serving as an educational resource and 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 with a tool that can quickly transform creative ideas into in-game elements. Other objectives include:

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

Several tasks must be completed throughout the development process to achieve the stated objectives. 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 essential 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 GOALS

In the development of the program, specific 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 critical security and safety aspects to consider when developing the Minecraft code generation system using C#?

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 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 using 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, allowing developers to create rich and interactive user interfaces easily.

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 code 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.

1.4.2 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 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 the 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.

2. USER INTERFACE

2.1 Design Considerations

Choosing Windows Forms as the GUI framework for our proposed program to generate Minecraft code and command blocks is based on its 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.

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

2.2 Functional and Non-Functional Requirements of UI

**Functional Requirements**

1. **User Interface Navigation:** The user interface should provide intuitive navigation between different forms and modules, allowing users to switch between potion generation, block generation, and command generation functionalities seamlessly.
2. **Input Validation:** The user interface 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. **Code Generation:** The user interface should facilitate the generation of Minecraft code 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 user interface should support integration with the Minecraft client, 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 user interface 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 user interface should be user-friendly, intuitive, and easy to navigate, catering to users with varying technical expertise and familiarity with Minecraft modding.
2. **Performance:** The user interface 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 user interface should be reliable and robust, capable of handling errors gracefully and recovering from unexpected failures without data loss or corruption.
4. **Accessibility:** The user interface should be accessible to users with disabilities, adhering to accessibility standards such as keyboard navigation support, screen reader compatibility, and high contrast mode.
5. **Scalability:** The user interface should be scalable and capable of accommodating future enhancements, updates, and expansions to accommodate evolving user requirements and technological advancements.
6. **Security:** The user interface should prioritize security by implementing data encryption, authentication mechanisms, and secure communication protocols to protect user data and prevent unauthorized access or tampering.
7. **Compatibility:** The user interface should be compatible with various Windows operating systems and hardware configurations, ensuring seamless performance across different devices and environments.

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.

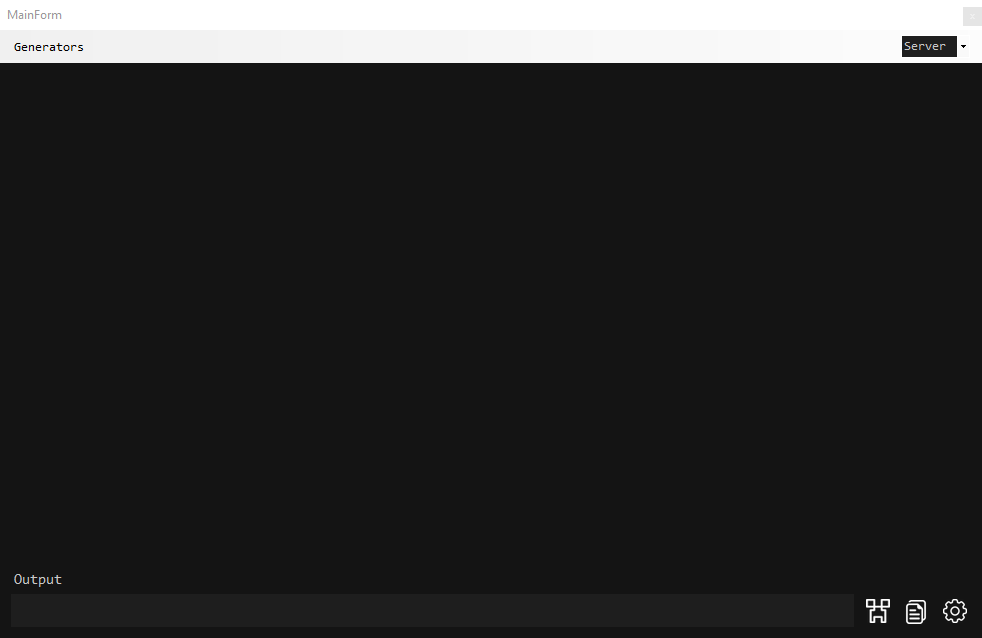
2.4 User Workflow

The Windows Forms UI is designed to 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:** 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. Leveraging aesthetically 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 a harmonious visual composition that resonates with users.



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

2.6 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 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 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.

2.7 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. The original code is in Github. 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.

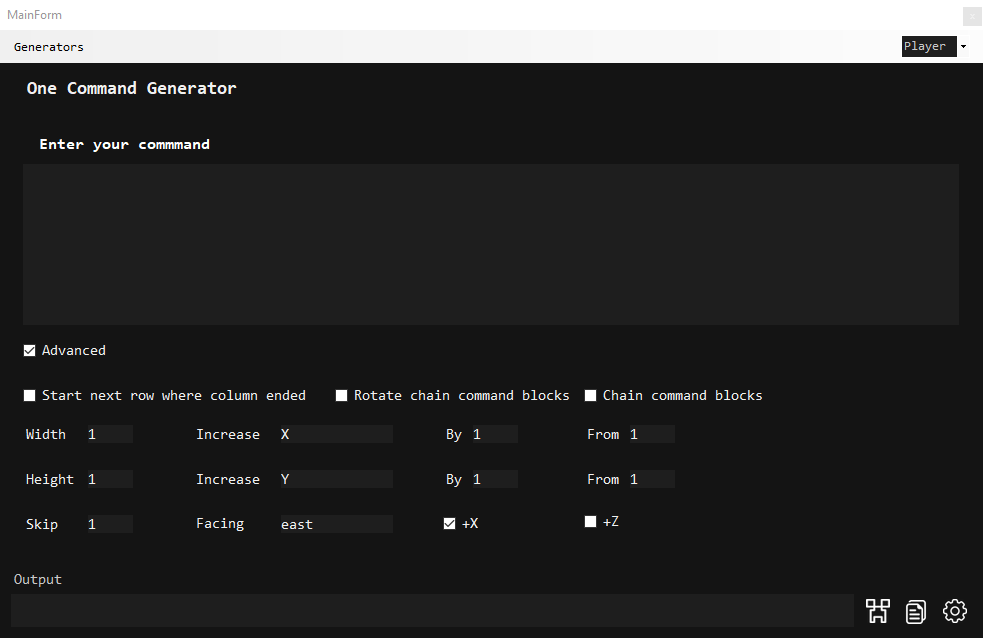
Increments width.

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.



***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. 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:

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.

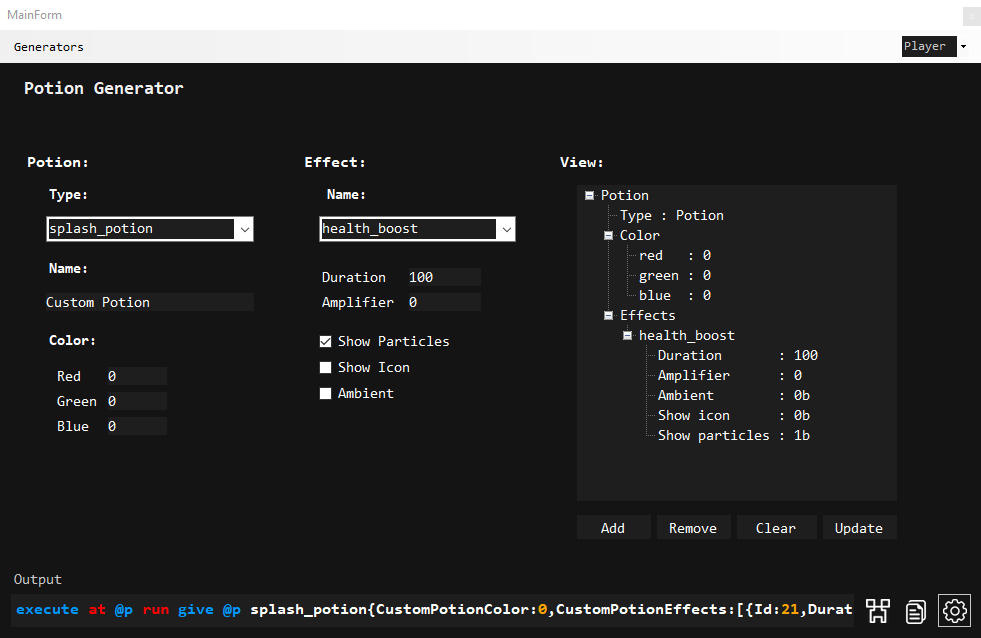
RemoveButton\_Click: Handles the click event of the "Remove" button. It removes the selected effect from the potion's effects list and updates the potion tree display accordingly.

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.



***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. 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. 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. Developers 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 proposed program for generating Minecraft code and command blocks features a set of action buttons designed to streamline the user experience and facilitate everyday 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 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 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.

2.10 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 everyday tasks, such as running commands, copying code, and generating output. The program streamlines the user's workflow by centralizing these functionalities within the interface and minimizes the cognitive load associated with manual operations.
2. **Enhanced Productivity:** Action buttons enhance user productivity and efficiency by automating repetitive tasks and reducing manual intervention. Users can perform tasks such as code execution and copying with minimal effort, freeing 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 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 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. Leveraging 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, streamline writing clean and concise code. Whether crafting custom blocks, items, or entities or implementing game mechanics, IntelliJ IDEA empowers developers 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.

Furthermore, IntelliJ IDEA facilitates unit testing through its seamless integration with popular testing frameworks such as JUnit. This enables developers to write and execute test cases to validate the functionality of individual components, thereby enhancing the overall robustness of the mod.

3.2 Functional and Non-Functional Requriements of the Mod

**Functional Requirements:**

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.

1. **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.

1. **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.

1. **Event Handling:**

The mod should handle Minecraft server events appropriately, ensuring that it responds to in-game events as needed.

1. **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.

**Non-Functional Requirements:**

1. **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.

1. **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.

1. **Security:**

The mod should implement secure communication practices to prevent unauthorized access or exploitation.

It should validate incoming commands to ensure they originate from trusted sources.

1. **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.

1. **Documentation:**

The mod should be well-documented, with clear instructions on installation, configuration, and usage.

It should provide comprehensive documentation for developers interested in extending or modifying its functionality.

1. **User Interface (UI):**

While not directly applicable to the mod itself, the Windows Forms application should have an intuitive user interface for sending commands to the mod.

It should provide feedback to users on the status of commands sent and received from the mod.

3.2 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, encompassing 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.

Modding also plays a significant role in the educational sphere, with many educators leveraging Minecraft to teach programming, game design, and other STEM (Science et al.) subjects. By engaging with modding, students learn valuable technical skills and foster creativity and problem-solving abilities in a fun and engaging environment.

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 vibrant and ever-evolving.

3.3 Framework and Architecture

Forge modding operates within the Java environment, leveraging 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 encompass 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.

3.4 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 MODS

At the heart of this integration lies a Forge mod meticulously crafted to bridge the communication chasm between the Minecraft server and external programs. 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. This seamless interaction between the Windows Forms application and the mod empowers users with unprecedented control over the Minecraft universe, enabling them to manipulate game elements and orchestrate complex scenarios dynamically.

A pivotal aspect of this integration is establishing a robust communication protocol between the Windows Forms application and the Forge mod. Leveraging 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 applications, 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. Leveraging 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.

5. REQUEST TO MOD SERVER PORT

5.1 Instructions from a client to the server

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 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 range from simple actions like teleportation or item manipulation to more advanced operations involving mod-specific features.

1. **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.

1. **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.

1. **Server Administration:** Server administrators may request 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.

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.

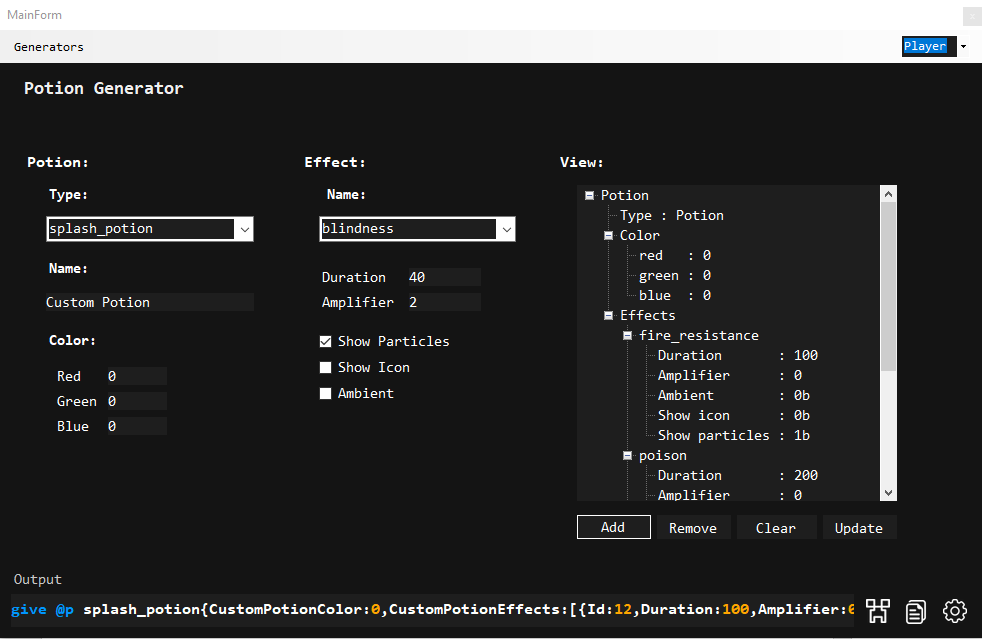
1. **Cleanup**

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. TEST RESULTS

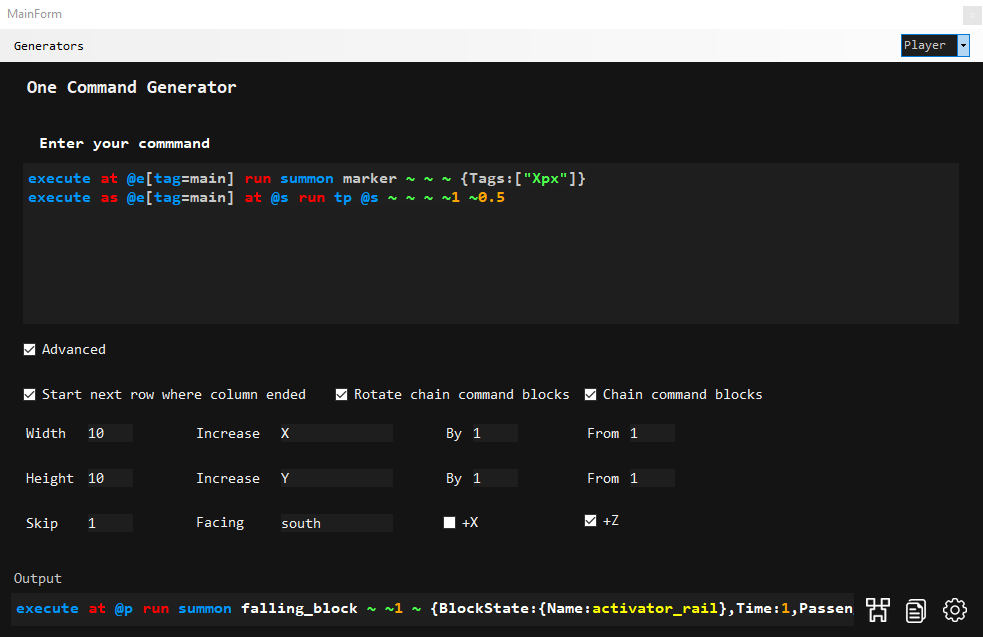
Screenshots were captured during the testing phase to visually demonstrate the effectiveness of the generated commands and command blocks. These screenshots are tangible evidence of the program's capabilities and showcase its impact on Minecraft.

The first screenshot 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.



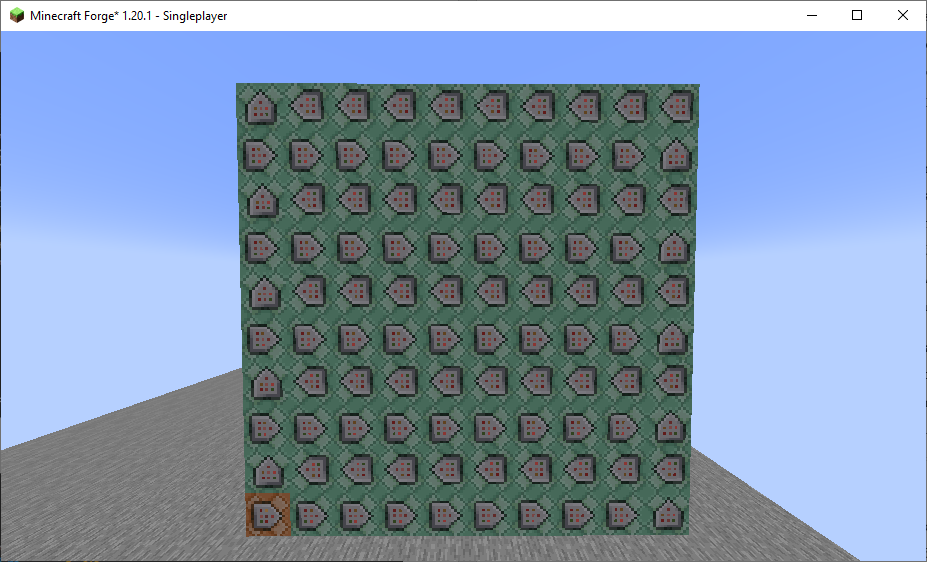
***Figure 4.*** *Potion Generator Form with various arguments.*

The second screenshot 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 block appear like this.



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

The fourth screenshot 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 leveraging programming languages like C# and modding frameworks like Forge to empower 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 streamlining the modding process. By providing a user-friendly interface through Windows Forms, we have lowered the barrier of entry for aspiring modders, 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, enabling users to combine and expand upon existing functionalities seamlessly.

Moreover, adding more generator forms to the program is worth considering as we look toward 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, empowering them to realize a broader range of creative visions within the Minecraft universe.

Additionally, this program facilitates the generation of Minecraft code and command blocks, enabling users to automate repetitive tasks and expedite the development process. This enhances productivity and encourages experimentation and innovation within the Minecraft community. By efficiently empowering users to translate their ideas into tangible in-game experiences, we foster 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 modding projects. Additionally, exploring ways to integrate advanced functionalities such as artificial intelligence or machine learning algorithms could open up new possibilities for dynamic and adaptive gameplay experiences within Minecraft.

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). 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). 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.

JetBrains. (2024). Intellij official webiste. [IntelliJ IDEA – the Leading Java and Kotlin IDE (jetbrains.com)](https://www.jetbrains.com/idea/)

Github. (2024). Application and mod code. [den1k0/Thesis (github.com)](https://github.com/den1k0/Thesis)