Teach Coding Using Scratch 3.0 in Minecraft: Education Edition

With the Guidance of Microsoft Lead, Jeff McKune
Kyle Burch, Michael Crews, Trent Douglas, Alexandria Leto, Shania Jo RunningRabbit

Introduction

The purpose of this project was to implement Scratch 3.0 in Minecraft: Education Edition (MEE). MEE is an extension of the Mojang video game Minecraft, in which players explore a 3D block-world with the option to extract raw materials, craft tools, or build structures. In the Education Edition, educators use the controlled game environment to construct lessons for their students in a variety of subjects (e.g. art and design, history and culture, science, math, language arts, and computer science).

Before our project, computer science students using MEE could write programs that build structures or allow them to teleport in game through the use of the code editors MakeCode and Tynker. Over the course of our project, we were tasked with adding MIT's latest block-based coding language, Scratch 3.0, to the list of code editors within MEE. Because the consumer base for Scratch 3.0 and Minecraft largely overlap, by implementing Scratch 3.0 functionality into MEE, our goal was to provide users with a new and exciting way to use a familiar coding interface.

The implementation of Scratch 3.0 into MEE required a landing page that would be linked in the game beneath links for Tynker and MakeCode (*see Appendix A*), for there to be tabs on the homepage for saved projects, tutorials, and achievements (*see Appendix B*), and to preserve a consistent "Minecrafty" feel throughout the deliverable. Functionally, the implementation of Scratch 3.0 into MEE required the editor to be accessible from the landing page of the website, new projects to automatically save under the "My Projects" tab of the website, and, importantly, users to be able wipe any collected cookies quickly and easily in order to protect the user's confidentiality and right to be forgotten.

Requirements

Functional Requirements

Ultimately our job is to create a user-friendly website interface that includes a code editor powered by Scratch 3.0 that can communicate with MEE.

Scratch 3.0 Website

- Home Page (see Figure 1 for example)

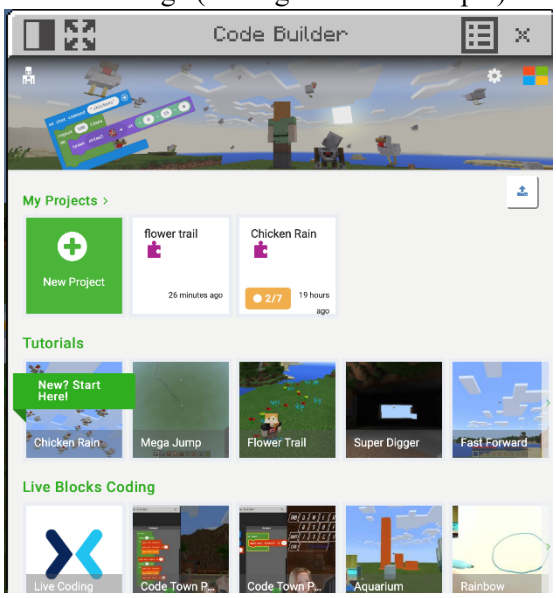

Figure 1: Home Page Reference Image from MakeCode

    - My Projects
        - Create new project
        - Open/Update/Delete projects
        - List of already existing projects
    - Tutorials
        - Project Tutorials
    - Settings Dropdown
        - Reset Cookies (clear user data)
        - About page
- Code Builder (see Figure 2 for example)

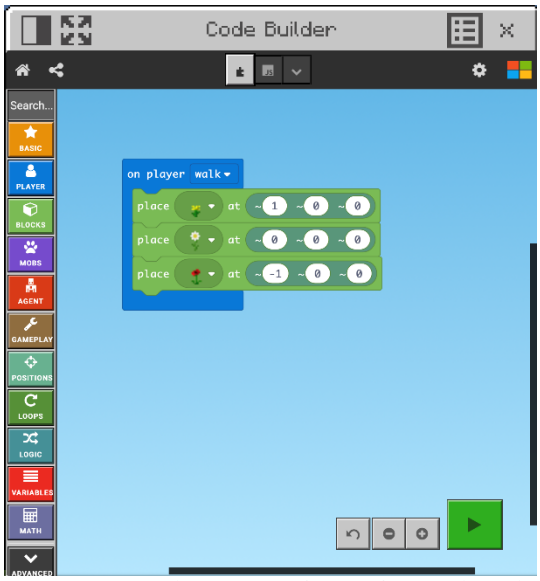

Figure 2: Editor Page Reference from MakeCode

- Task Bar
  - Home Link
  - Share Project (Possibly send to Scratch)
    - Publish Project
- Scratch Workspace
  - Scratch Block Toolbox (List of all code blocks)
  - Scratch Building Space (Drag and Drop Space)
    - Undo Change/Redo Change
    - Zoom Out / Zoom In
    - Run/Stop

Non-Functional Requirements

Curriculum Development

- How to introduce coding to kids
- Progression catering to different skill levels
  - Lower levels not only associated with younger age groups; higher levels not only associated with older students

Licenses

- Follow Scratch's licensing requirements
- Add any other license that Microsoft wants

Security

- Must maintain user privacy (right to be forgotten)
- Code window should not be able to be used as a web browser

Design (see Figure 3 for example)

Figure 3: In Game Experience Reference

- Keep integration and webpage "Minecrafty", similar to Azure Notebooks page
- Basic functionalities but not too simple to prevent nested webpages

Definition of Done

List of Minimal Features (Priority I)

- Internet accessible website that integrates Scratch 3.0
- Integrate Minecraft blocks into Scratch 3.0 deployment
- Users should be able to run or stop code execution at any time
- Have a queue for commands waiting to be run
- Every slash command represented as a block of code

List of Minimal Features (Priority II)

- Combine sets of commands to do something "interesting", functions, high-level semantics (I.e. create cube)
- Tutorials – directing users through steps to create something before "moving on"
- Persistence of user's code, progress
- Project semantics (create project, load project, etc.)

Stretch Goals (Priority III)

- We create our own, CSM Branded, tutorials and project schematics

Tests that must pass before software is accepted

- All implemented slash commands function as expected

Method of Product Delivery

- GitHub – possibly open source
- Link to the globally available Website

System Architecture

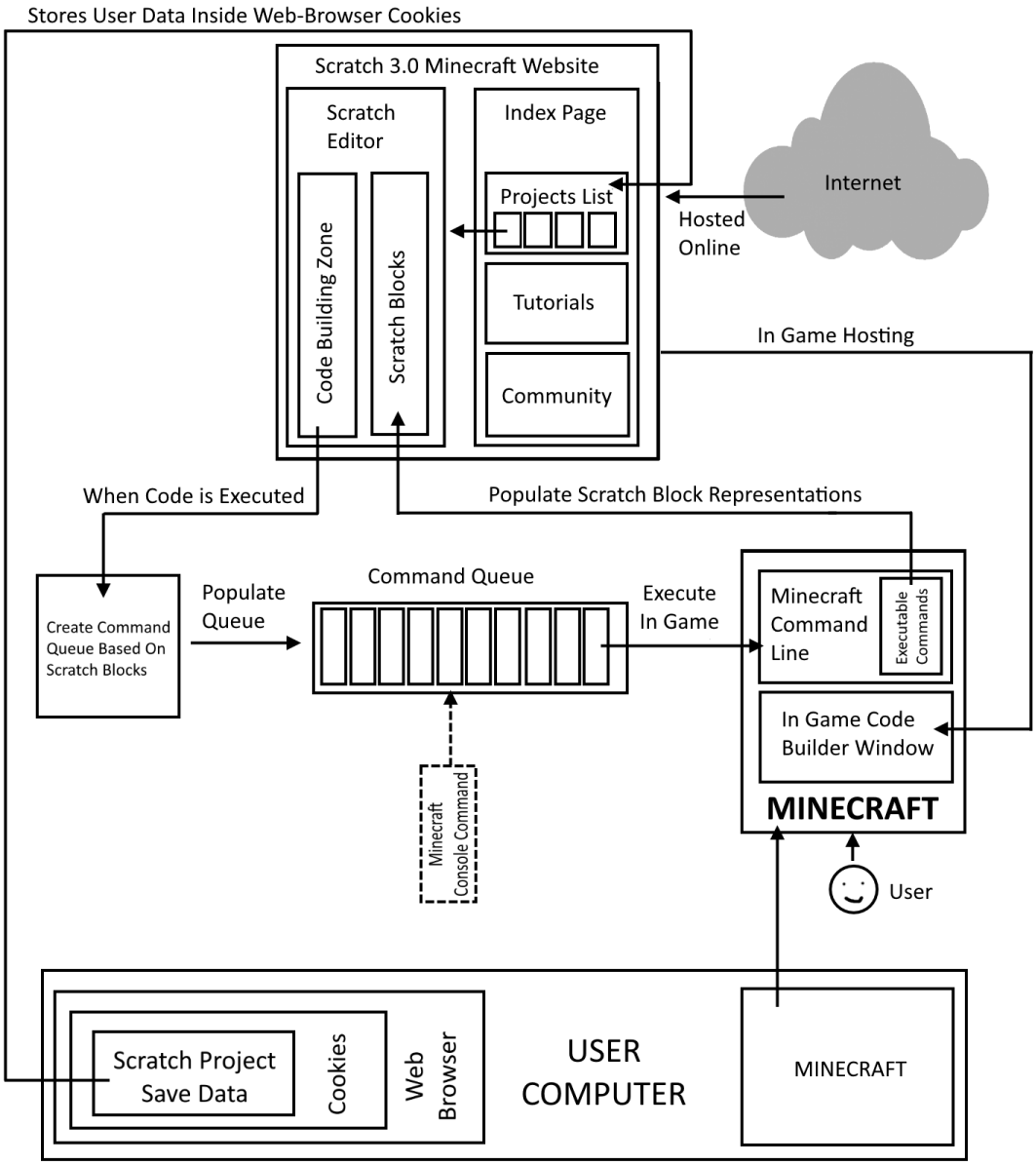

Figure 4: System Architecture Diagram

Our code implements a Scratch 3.0 code editor inside of Minecraft: Education Edition (MEE). The Scratch editor is hosted by Microsoft on an online server, can be accessed by the user from inside the MEE application, and executes Scratch programs in-game via the MEE command line. The project architecture that permits our project's web-hosted editing and in-game execution is comprised of four primary subsystems: The Webportal, The MEE Command Line / Command Queue, The User Device, and Minecraft: Education Edition.

The MEE Scratch 3.0 Webportal is hosted on an online web server which is maintained by Microsoft and can be accessed through any mainstream internet browser. The Webportal is comprised of two primary subpages: the Index and the Editor. The Index acts as the website's landing page. The Index displays a list of all the user's projects, allows the user to create new projects, tracks user statistics, showcases user achievement progress, and hosts tutorials that the user can follow. The most

important part of the Index is project creation and project listing. Previously created projects are stored on the User's Computer inside of the webcache. The project list is generated by pulling previous projects from the webcache. When a user opens a previous project, the old project data is pulled from the webcache. The user is then brought to the Scratch Editor page where they can resume work on their old project. When a new project is created, the user is brought to the website's Scratch editor page which allows users to build an executable Scratch program using prebuilt Scratch blocks. Each available Scratch block represents either an In-Game command such as "Spawn Mob" or a logical operator such as "If-Else". The Editor page also executes code by communicating with the Minecraft Command Queue. When the user's Scratch code is executed, the Editor converts the Scratch Blocks back into their respective commands and populates the MEE Command Queue based on how the Scratch code was written.

The MEE Command Queue is a queue that stores valid Minecraft Commands for In-Game execution. In Minecraft, time is tracked in units called ticks. The game's world state is updated after every tick. Because of this game behavior, only one command can be executed every tick. Unless the Command Queue is empty, every tick the game retrieves the next command from the command queue and executes it. When code is executed in the Scratch Editor, the command queue is populated with the Minecraft command representation of each Scratch block. Logical operators such as "If-Else" and "For-Loops" do not have valid Minecraft command representations. As such, the Scratch Editor is responsible for pushing commands to the Command Queue in the correct order. The commands in the Command Queue can change the game world upon execution which allows the user to experience their code execution block by block / tick by tick / command by command.

MEE is the intended means by which the User should interact with the Scratch 3.0 Editor. Through MEE, users can display a minimalistic web browser on top of their game. This overlaid web browser, known as the Code Connection window, allows users to access the web portals that host various MEE code editors. When the users access the Scratch 3.0 Editor via the Code Connection window, they are brought to the Scratch 3.0 Index Page. Once the user has reached the index page, they can use the web portal as described above but from within the Minecraft game. The Scratch 3.0 Editor's Scratch Blocks each represent a valid Minecraft command. Whenever a Scratch program is executed, the Scratch blocks are converted into their Minecraft command representations on the server side where they are then pipelined into Minecraft via the Code Connection window where they are then executed on the client-side. These commands can be executed from within the game without the Scratch 3.0 editor by utilizing the in-game command line. Whenever a command is queued through either the command line or the Scratch 3.0 editor, the game will wait until the next valid tick before executing it. Every command has a different effect on the Minecraft game; some commands place blocks in the game world, some commands spawn game entities, and others change in-game variables. Whenever a command is executed successfully, Minecraft returns a success code. Unless given special permission to ignore success codes, the Scratch Editor waits for the success code after executing a command before continuing its code execution.

The final component of our Scratch 3.0 System is the User's device, which is used primarily for storage purposes. In order to respect privacy laws, our system stores user data inside the Internet Browser's data cache as a cookie. Because all of the user's data is stored locally and they have free access to viewing/editing their data, no privacy laws are violated. Whenever a Scratch project is saved, the project data is stored as a cookie on the User's system. Whenever the projects list is populated, the list is created by pulling from the same cookies. Achievements and user statistics are stored in an identical manner. (see *Figure 4* for a graphical representation)

Technical Design

Tutorials

Two members of our group implemented tutorial functionality in MEE and created tutorials with topics we thought were interesting, such as the Colorado School of Mines' mascot Blaster. The language the tutorials are written in as well as some of their topics are inspired by Microsoft's MakeCode. The tutorials are written in markdown, a high-level markup language with basic formatting syntax, so future teams or educators can easily add more tutorials without much of a learning curve.

Our tutorials (reference *Figure 5*) were created to engage users beyond just freeform block coding. Instead, these tutorials were created to help develop the users' understanding regarding usable block types and their various applications.

Figure 5: Tutorials

We also created the system that displays the tutorials on the homepage and the individual tutorials within the editor once they are selected (reference *Figure 6*). The tutorial navigation interface can help the user navigate through the different steps of the tutorial, see what blocks they need to use, and see how many steps are in the tutorial.

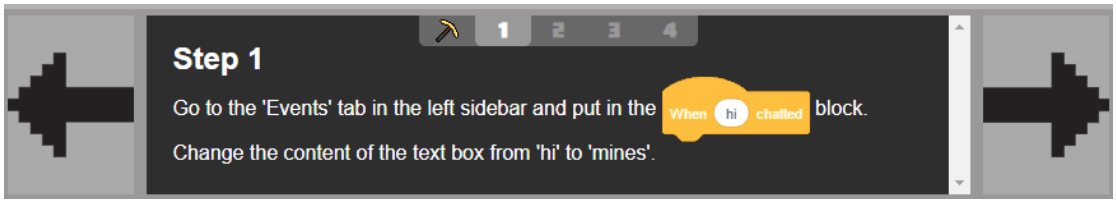

Figure 6: Tutorial navigator

Each tutorial presents users with a unique opportunity to engage with their Minecraft world; the only requirement on the user's end is to have any Minecraft world open on their device.

Achievements

Our team also designed and implemented basic Achievements in MEE. MEE did not originally have an Achievement system before we built and implemented our own. The Achievements are located on the Achievements tab on our web portal. Users can gain these achievements through two different

methods: milestones or special conditions. Milestone Achievements are given to the user after they have reached a certain statistical milestone such as reaching 100 Scratch blocks placed, 10 projects created, etc. Special Condition Achievements are given to the user when they complete specific tasks such as finishing the Hello World tutorial, changing the in-game weather, etc. The Achievement experience is split between three subsections: My Achievements, Statistics, and the Achievement Gallery.

The My Achievements tab (reference *Figure 7*) displays all of the MEE Scratch Achievements and current user progress towards each achievement. Each Achievement has an initially silhouetted icon, a title, and a description outlining the Achievement's requirements. For Milestone Achievements, Achievement progress is based on the user's statistics. When the user's progress matches the Achievement requirement or the required task is completed, such as in the event-based Achievements, the Achievement is given to the user. When a user gets the Achievement, the Achievement icon is displayed, the Achievement is highlighted on the My Achievement page, and the Achievement's icon appears in the Achievement gallery.

Figure 7: Achievements Page

The Statistics tab (reference *Figure 8*) displays all of the user's tracked statistics such as blocks placed, programs executed, etc. The user Statistics are stored on the user's device as a browser cookie. As the user increases their statistics, the changes are displayed on the Statistics page. Tracked user statistics and Achievements can be deleted – along with all other stored user data – from the settings tab.

Figure 8: Achievements Statistics Page

The Achievement Gallery (reference *Figure 9*) displays all of the user's completed Achievements as a representative icon. When the user selects an Achievement in the Achievement Gallery, the Achievement is highlighted, and a description of the Achievement is displayed. Future functionality will allow users to unlock different features such as new coding backgrounds or player cosmetics through this page.

Figure 9: Achievements Gallery Page

Quality Assurance

User Interface Testing

As a team, we were able to divide out tasks evenly when new tasks arose, which meant multiple people working on different areas of the UI concurrently. This also means we each spent a decent amount of time testing our local changes and ensuring they blended with already committed code. This individual testing differed depending on which part of the website was being worked on (homepage, editor, tutorials, etc.), but it was mainly a cycle of making changes, making note of those changes, testing their functionality along with aesthetics, and committing them.

There were also some tests that were consistent across our website. These tests included:

- Making the website elements size dynamically so it looks good on any screen size
- Opening site in Code Connection
    - Emulating how changes will look in Minecraft
    - Keeping the same usability as the browser version of the website
- Each member testing the entirety of the site, watching out for obvious glitches or issues
    - Maintaining the user friendliness of the site, pointing out features that are too obscure or that may not be obvious enough to the user

Most features were created, tested, and implemented with a group of either 2 or 3 team members to guarantee that certain components of the UI would be ideal in our final product.

Integration Testing

Since our project was to integrate Scratch 3.0 into Minecraft, we had to consistently ensure our integration was functional and sensible. Considering we were using both demo code, which already had basic functionality in MEE, and Code Connection, which handles the backend integration, we focused our efforts on the actual interface. To make sure our interface changes worked without issue, we had to test our changes both on a browser and in Code Connection to get an understanding of what it will look like in the Minecraft code builder.

Project Demo and Feedback

Our team intended on allocating time for developers from MEE to read over our code, but we were only able to allocate time for a general project review. We demonstrated our project to a group of around 30 Education Edition developers who were able to give us positive feedback on our integration of Scratch 3.0. The feedback mainly consisted of compliments rather than true constructive criticism, but the main goal of the presentation was reached: to portray what a Scratch 3.0 integration could look like and how much potential it has once it is passed onto a future team.

Future Testing

Due to our project being hosted by Microsoft, our work was more of a proof of concept to get future developers excited about a full-fledged Scratch 3.0 integration. Though our code is usable and works well, it is not the clean, high-quality, and easily scalable implementation that Microsoft would release. Once the code is handed off, more rigorous testing will be done by Microsoft.

Results

All of the original functional requirements to consider the project "done" were met. However, much of the time we spent working on the project was spent on our stretch goals, as priority requirements were implemented relatively quickly. There were several additional features which we wanted yet were unable to implement given time constraints. Below is a list of some of our stretch goal features we did not have time to implement.

List of features we did not have time to implement:

- Community Tab (the ability to share projects with other people)
- Properly working backpack inside the editor
- Properly resizing of the editor so elements of the editor do not look too small or too big
- Creating source code to replace a file that is thousands of lines long
- A tutorial that has a specific Minecraft world linked to it

Performance Testing Results (Rough estimate to load pages):

	Chrome	**Code Connection**
Landing Page	Within 5 seconds	Within 5 seconds
Editor	Within 5 seconds	May take a few minutes to load
Spawn Blaster Tutorial	Within 5 seconds	May take a few minutes to load
My Achievements	Within 5 seconds	Within 5 seconds
Statistics	Within 5 seconds	Bug, can't open statistics
Gallery	Within 5 seconds	Bug, can't open gallery
Settings	Within 5 seconds	Within 5 seconds

Figure 10: Testing Chart

Summary of Testing (e.g., browsers):

- Tested our website by itself in Chrome
    - Easier to test because we don't need MEE open and it is a faster and more reliable browser than Code Connection with inspection features
- Tested our website in Code Connection that links to MEE
    - A good way to ensure Mines Minecraft communicates with MEE and it is a more accurate representation of how the website will look when it is integrated into MEE

Results of Usability Tests (e.g., testing educational software with real students):
So far, we have not done a lot of usability tests outside of our team. Ideally, we would have a test group in which teachers would let their students use MEE and fill out forms that asked about the user-friendly nature of the website, about what parts of the website students liked and did not like, and about bugs the students may have encountered while using the website. We would then make the necessary changes to the website depending on the feedback.

Future Work (e.g., any ideas you may have for extensions):

- Adding the ability to share projects with one other
- Seeing the most popular projects from the Mines Minecraft community at any given time
- Adding more tutorials and adding a tutorial linked to a specific Minecraft world
- Adding the ability for the user to easily add new blocks to the editor
- Adding new settings so the user can customize the look of Mines Minecraft

Lessons Learned:
Here is a list of languages/software we have learned more about throughout this project:

- HTML
- JavaScript
- CSS
- Scratch 3.0
- Blockly (Used to build Scratch 3.0)
- WebStorm
- Minecraft: Education Edition
- Code Connection
- Git
- Microsoft Teams
- Slack

We have learned a lot regarding how to run and test a website locally; for example, we found out running our website locally using Web Storm was the best option. Many of the features of our website such as the tutorials required an understanding of HTML, JavaScript, CSS, and Scratch 3.0 to be implemented correctly. Other features such as the Blocks Menu Resizer required a deep understanding of Blockly to modify Scratch 3.0. The code we were given from MIT presented a big challenge as much of the source code wasn't provided; instead, we received a single file thousands of lines long. In retrospect, we should have spent more time rebuilding the source files instead of editing this vast file.

Documentation for Future Developers

This project was built using Scratch 3.0 developed by MIT. As a starting point we used a demo created by MIT/Playful Invention that can be found at https://www.playfulinvention.com/temp/tynker8/. Screenshots of the MIT demo can be seen in *Appendix C*.

This demo from MIT/Playful Invention can be added as an editor to Code Connection. When the MIT/Playful Invention demo is running in Code Connection it can successfully communicate with MEE. For example, when a user runs a project inside of the MIT demo, the proper commands will be sent to MEE execute the project.

But there are some major drawbacks of using the MIT demo. The biggest one is that the demo doesn't include all of the source code. Usually when developing Scratch 3.0, you start out with source code from the Scratch 3.0 GitHub which would be done by cloning the scratch-blocks repo. Cloning the scratch-blocks repo would give you all the necessary source code that could easily be edited. Usually after you are done editing, you run a npm command to compile all the source code into one file called Vertical.js. The problem with the MIT demo is that we only have the Vertical.js file and not the rest of the modified scratch-blocks source code. In our case Vertical.js was the length of a 500-page long word document which made editing the file extremely difficult. Because Vertical.js is difficult to edit and understand, our recommendation is to take all the useful bits from our project but clone the scratch-blocks repo and start that part of the project from scratch. In the end, good code quality can only be achieved through re-creating Vertical.js from the scratch-blocks source code. The reason we didn't just clone scratch-blocks was because there were numerous differences in the Vertical.js file from the MIT demo that enabled it to work with the rest of the project.

The scratch-blocks repo can be found at https://github.com/LLK/scratch-blocks. To understand how to compile/run scratch-blocks look at these documents: https://github.com/LLK/scratch-blocks/wiki and https://github.com/LLK/scratch-gui/wiki/Getting-Started. As a note, the Veritcal.js file can be found inside of scratch-blocks/dist/web after compiling the scratch-blocks clone with this command: npm install.

Throughout our project, we have had difficulties updating the sizes of elements inside the editor. For example, just to update the size of the toolbox would require modifying many different lines of code in numerous files. For us, it become too much of a hassle every time we wanted to change the size of the toolbox, toolbox icons, flyout, the flyout resizer, and the backpack. To fix this problem we created one file called editorConfig.js that is located in the js folder. The editorConfig.js file sets the sizes of the toolbox, toolbox icons, flyout, the flyout resizer, and the backpack in pixels. For example, you could change the getToolboxWidth() function inside editorConfig.js to return 20 which would set the toolbox width to 20 pixels. We also removed the backpack by changing the getBackpackWidth() function to return 0.

List of Known Issues/Bugs:

- Home page issues/bugs
    - The settings icon should be updated
    - When using Code Connection, the achievements drop down does not display all the available options
    - When the page is made smaller, the achievements drop down sometimes doesn't format correctly
- Editor issues/bugs
    - The toolbox (Categories box) doesn't size correctly and some of the categories may be cut off
    - Issues/bugs when the backpack is enabled
        - Sometimes the backpack icon cuts off one or two category icons
        - When you drag a block out of the backpack, the block isn't placed correctly

Appendix A: Implementation into Minecraft

When a user types “c” in MEE, the window shown in *Figure A.1* pops up. Currently, a user can choose to write programs with MakeCode or Tynker. Our implementation of Scratch 3.0 will be linked below these two options. This area is circled in red on the *Figure A.1*..This link will lead to the landing page of the website that we designed.

Figure A.1: The link to the Mines Minecraft home page will be linked in the red circle.

Appendix B: Webpage Visuals

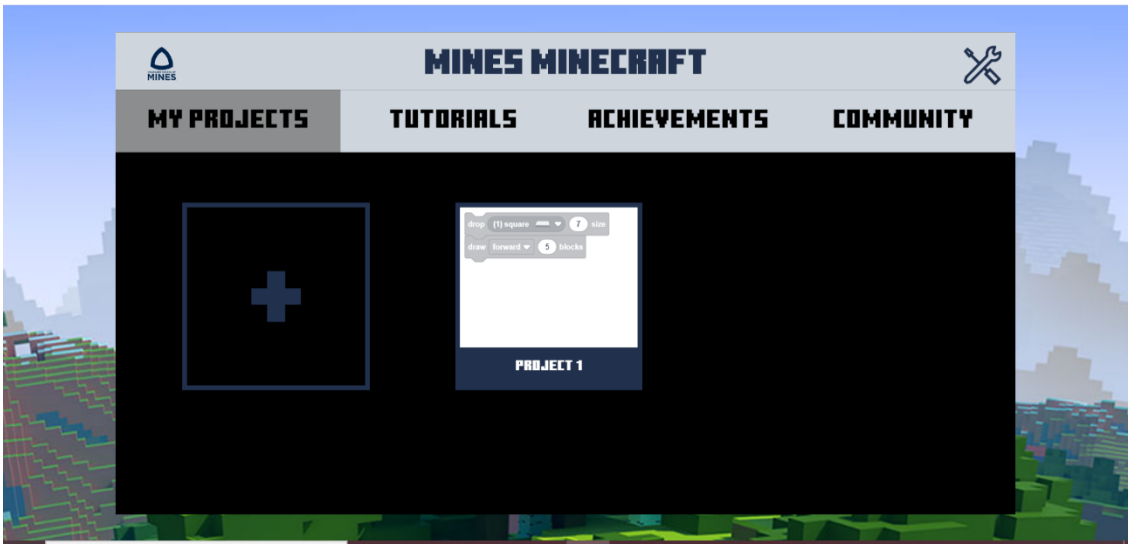

Figure B.1: Landing page with add project button and saved projects

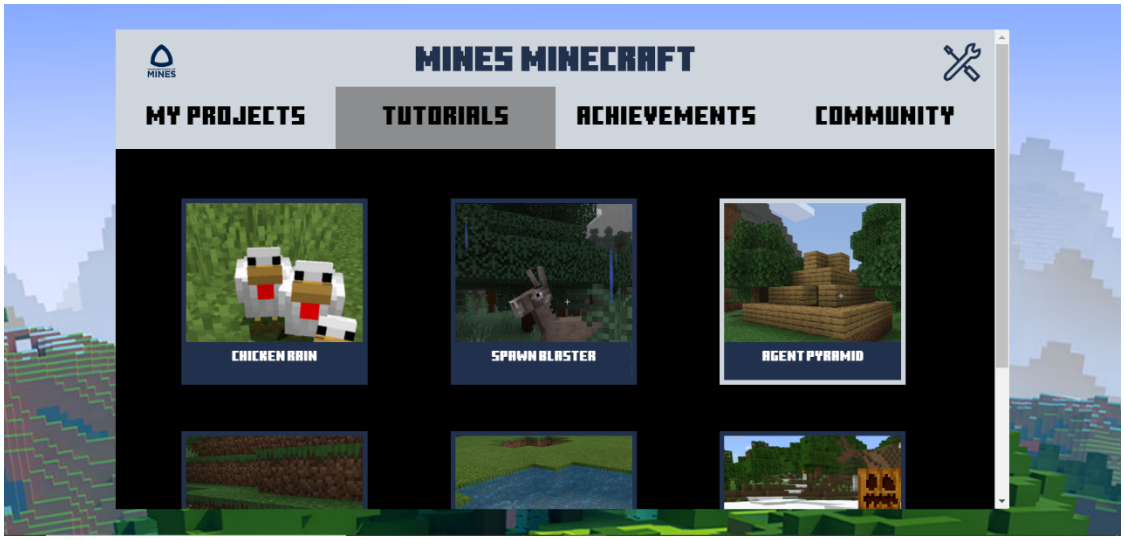

Figure B.2: Tutorial page

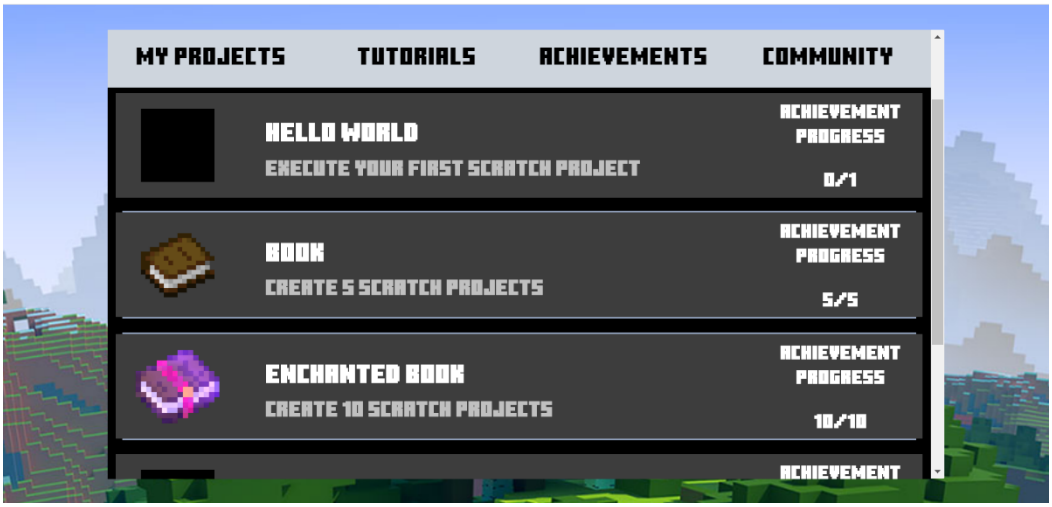

Figure B.3: Achievement page

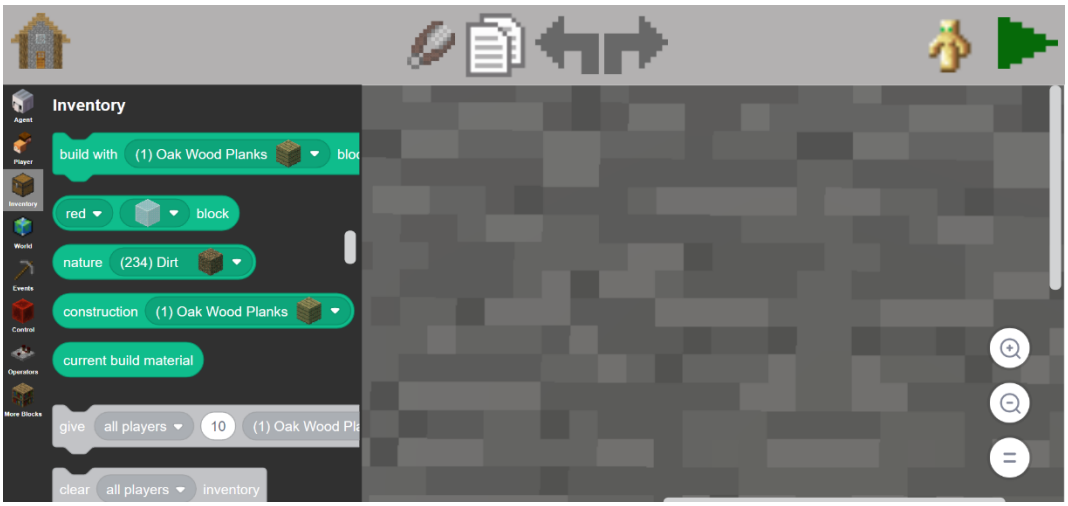

Figure B.4: Code editor linked from landing page

Appendix C: MIT Prototype Visuals

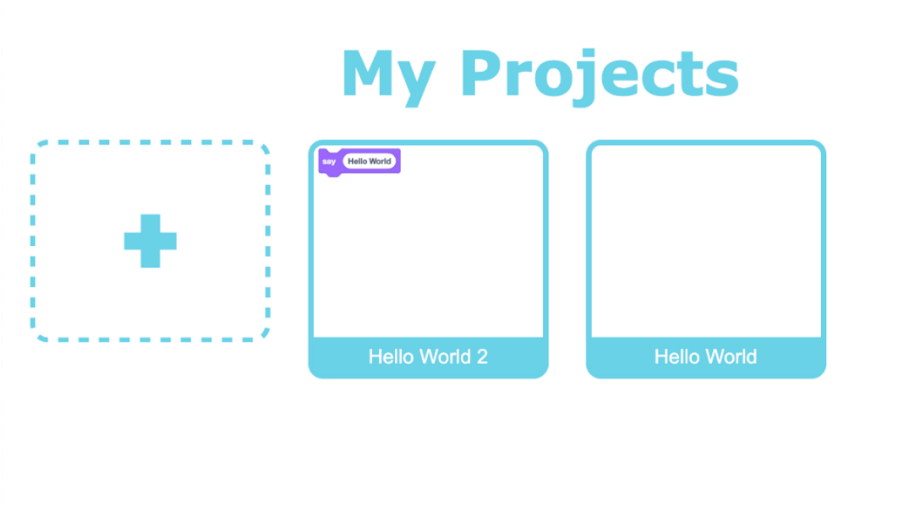

Figure C.1: MIT Prototype landing page with add project button and saved projects

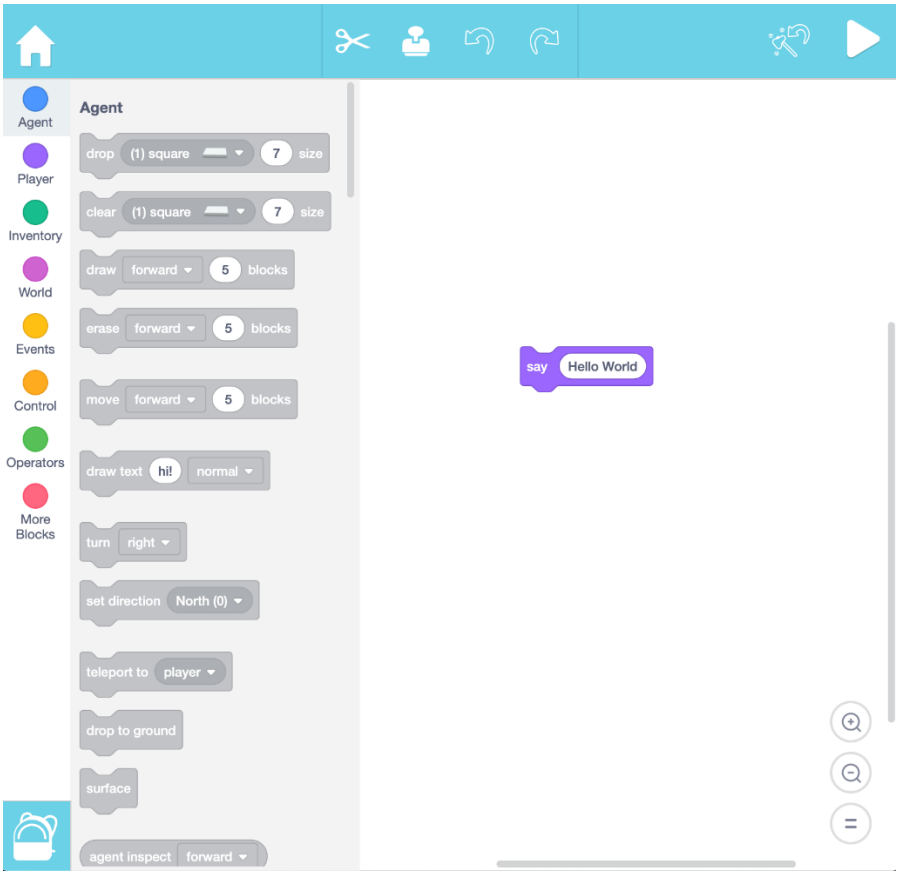

Figure C.2: MIT Code editor linked from landing page