STEAM Education, Spring 2026

Project Members

Christopher Tuckman
Joanna Britto
Kimari Guthre
McElroy Ruman
Yimer Duggan

Project Sponsor

Mike Conroy, PM
Florida Space Institute
Partnership I, Research Pkwy,
Orlando, FL 32826

Executive Summary

The Steam Education Project teaches middle schoolers to code in Python and prepares them for high school computer science and robotics. It takes the form of a nine-week after-school program which features: A VSCode extension, a lesson plan, a teacher manual, teacher tools, a pilot program, and potentially a simulation.

Most of the project is embedded in a VSCode extension. It has a menu that lets you view the course content, a code view that shows the current code you're working on, and a lesson view. The lessons are rendered in Jupyter notebooks and notify students of their progress as they go along. It even allows more lessons and courses to be added. By default, this extension will include the content of the nine-week after-school program. The default content will consist of a Unit on the fundamentals of Python, a Unit on the necessary JSON and POST request skills, and a Unit for those who want to keep learning after the course.

The team will get the opportunity to test the program at Discovery Key Elementary School. This will allow us to make changes and identify the most valuable teacher tools. The team will then create these tools and package them together with the teacher manual. The teacher manual includes instructions for using these tools and an annotated PDF of the lessons.

Ultimately, the team aims to create an approachable and human after-school program. So we’ve abstained from using artificial intelligence in the work. All the work here will also be open source; that way, it’ll be accessible to as many people as possible, as education should be.

Table of Contents

I. INTRODUCTION

A. Purpose of Project

This project intends to create the tools and curriculum for an after-school program, intended for middle school students to learn Python in the context of programming a RE-RASSOR rover. The tools are going to be extensible, allowing the lesson plans to be swapped. This allows the end user to learn more than what is initially provided in the deliverables of this project, and allows later teams to develop on top of what this team has in order to facilitate higher-level learning for more advanced students. Additionally, the intention of these tools is to work with professional-level software, allowing students to transition from simplified models of programming (such as the lesson software provided by Code, MIT's Scratch, or Google's Blockly) to more concrete facets of software engineering.

B. Definitions

The project is split into two essential halves: one focused on the tools used for teaching the students (the "Program") and the lesson plan that the students will be following to achieve their end goal (the "Resources"). Additionally, a stretch goal of the project is to create a simple rover simulation to be packaged with the Resources, meant to allow students to test their code throughout the course (the "Simulation"). Furthermore, the term "VS Code", sometimes written "VSCode", refers to the program known as Visual Studio Code, an IDE created and maintained by Microsoft that is available for multiple operating systems, including Windows and Linux.

C. Ownership of the project and ongoing Interest

The sponsor of this project is Mike Conroy, Project Manager at the Florida Space Institute (FSI). Mike Conroy coordinates a wide array of capstone projects, many of which revolve around the RE-RASSOR rovers, on which the lesson plan created by this project centers. This project, in particular, is aimed at middle schools looking to create a STEAM club for students between Grade 6 and Grade 8, and will be made available freely to schools that want to use the technology as a teaching tool exclusively. The rovers are available through FSI's open source repositories, and so long as the schools are able to create their rovers using their own resources, they will be able to keep and use the rover that they create with the lesson plan. Additionally, if the school is near enough to FSI's resources, coordination with FSI would allow rovers to be lent for use in the final steps of the lesson plan, which is focused on a competition for dynamic pathing. There is some interest from members of this team to continue working with FSI to facilitate further development of this project beyond the initial steps taken here.

D. Motivations

Yimer Duggan

My name is Yimer C. Duggan. I'm really excited to be working with FSI and having the opportunity to help improve education in programming. I started learning code in middle school, and over the years, I've become very grateful that I had the opportunity to learn early on. So to help other kids also get that head start is an honor for me. Another big interest of mine has always been space. I still fondly remember the miniature solar system my dad hung from my ceiling and all the cool sci-fi ideas I wanted to bring into reality.

Many of the folks I've taught Python to or have just worked with in general say I'm a great teacher, and I do have a great time giving presentations and sharing what I know. Python is one of my favorite coding languages. Most of my projects use some amount of Python, and I've used it to automate repetitive computer tasks, serve as an API for SQL databases, and even program some of my own games from scratch. So I'm sure I'll have a blast teaching it and showing kids what Python is capable of.

Joanna Britto

My name is Joanna Britto, and the primary reason I chose STEAM Education as my top choice for Senior Design is my desire to make computer education accessible to students in all age groups. I am deeply passionate about mentorship and STEAM education, especially for students who may be exploring these fields for the first time. My personal journey in STEAM beg6th-grade6th grade coding class. At first, the mandatory class did not capture my interest. That changed when a local Women in STEM club visited and presented career opportunities in the computing field. For the first time, I saw women like me represented in these careers, and I connected with their thoughts, experiences, and the impact they were making in the world. I was inspired to follow a similar path, focusing my skills on coding and web development. Since then, I have worked extensively with K-12 students in STEAM camps with the Museum of Science and History, and more recently, with the Initiatives in STEAM program at the University of Central Florida. Through these experiences, my career goals shifted from pursuing corporate computer design to STEAM education. I am currently applying for a Master’s of Education degree in Learning, Design, and Technology, with the goal of becoming a technology and computer teacher who can inspire young learners. I want to help make STEAM education exciting, meaningful, and accessible. This project aligns perfectly with both my passions and long-term aspirations, and I am eager to contribute my skills and experiences to bring this project to life. Overall, I am excited to see how this project can enhance student learning outcomes while supporting teachers in the classroom as they introduce coding.

Chris Tuckman

My name is Chris Tuckman, and the primary reason I chose this project is that I wish I had started programming at a younger age. I have always enjoyed using computers, whether it is just messing around with various software or playing basic computer games. In middle school, I would be assigned to work on coding software that utilized block coding. I found it interesting, but when I thought about moving on to normal coding, I would find myself overwhelmed with learning all of the syntax required. The block coding was definitely interesting to give me an idea of what I could do with code, but it was entirely ineffective at giving me the tools I needed to continue in my programming career. Python is a great language to work in, as there are lots of helpful libraries to assist with working on the more technical and mathematical programming challenges. However, it is still a text-based language, with lots of specific syntax to learn. So, a heavily guided lesson plan that teaches most of the core coding syntax (variable, if, for) in the context of a very visual RASSOR system reacting to their work is crucial. This ensures those who find coding interesting do not get bored due to the challenge of getting a visual response from early programming capabilities. This was also a personal problem that I found myself having while learning how to program. I liked the initial process of learning basic coding, but it was a challenge to continue to get the visual response that I wanted. Overall, I look forward to continuing to work on this project and creating software that teaches programming better than before.

McElroy Ruman

My name is McElroy Ruman, and the primary draw that I had to this project was how beneficial STEAM projects were to me during my time in K-12. I was exposed to computer science through a personal computer in my family’s household, but I truly started getting into programming through a game design course and club offered at my high school. From there, part of my volunteer hours that I used for my Bright Futures scholarship when I was entering college were for assisting my younger siblings’ elementary school STEM Club with programming simple robots that were pre-assembled with some minor customization. Through that process, I noticed a few problems with how the students were learning about programming, often focusing on producing a static solution that would solve the problem at hand without allowing for any variation. During that time, I spent a considerable portion of my hours explaining programming from a broader standpoint that would allow the students to build more robust, complex solutions.

In my personal experience, most of my programming education was self-led, primarily relying on open platforms and online videos to learn while doing. Over time, I picked up a lot of the foundations of what would become my career in computer science by myself, and bolstered it with formal education and training once I had access to those means. Much of my early experience with programming was work I did in the Unity game engine, focused on 3D rendering and basic game design, which then led to three years of teaching programming in the context of game development as part of the game development team in high school. Unity as a platform uses .NET C# for most of its in-engine scripting, which coincides with my current ecosystem of choice to this day, and what we will be using for the senior design project.

Kimari Guthre

My name is Kimari Guthre, and the primary reason why I chose STEAM Education for Senior Design was my passion for programming and my desire to see another avenue for younger people to learn programming. While of course, platforms like Scratch and code.org exist, when I was younger, I was somewhat skeptical of the block-based approach of both (though code.org does let you code without blocks), and I was never exposed to the latter (or at the very least, I do not remember my time with it). In recent years, though, I eventually came around to the idea of block-based programming being a great way to introduce the concepts, though I feel as though there should be some program made to bridge the gap. Regardless, I feel as though not only will a program designed to teach Python to those in the 8th grade be a great way to introduce younger people to programming, but the RE-RASSOR program will also give them a good source of motivation: powering a real NASA rover.

The Team

As a team, the primary motivations are to provide young students with a curiosity for code, a reliable and accessible solution to computer science education. The team’s personal experiences reveal many of the gaps in the current educational landscape and provide us with ideas to improve the technology available in classrooms today. The team aims to create a solution easily integrated into the current school system and capable of providing students with interactive feedback that supports continued learning and fosters a love of coding. Computer Science can be challenging, especially for those who are just beginning their journey. With the use of robotics and the software available at FSI, the Team aims to create a project that showcases to students the capabilities of code and how they can make an impact.

E. Legal, Ethical and Privacy Concerns

From the beginning of the planning process, the team has aimed to make the project as open-source and locally run as possible to keep the cost of implementation down, allow future groups to extend the project easily, reduce reliance on external resources, and keep users' data in as few hands as possible. Since everything involved in the extension is purely locally run, students' data will only stay on their computers and their teacher's (barring any security intrusions). However, since the project has been primarily implemented as a Visual Studio Code extension, you will have to agree to the license terms set by Microsoft [6] to use Visual Studio Code, and thus the extension.

When using the project, students' progress and only their progress will be synchronized to the teacher's computer, and stored there in the form of a TinyDB database. TinyDB itself is free and open-source, and does not have a terms of service to agree to. TinyDB, by default, stores data as a plain JSON file, and it will be the teacher's responsibility to follow good computer security practices to prevent this data from being stolen. However, this data is relatively insensitive:

the only thing stored is each student's progress and their names, and no other data is collected or stored. As development progresses though, work will be done to try and increase the extension's security to make it harder to steal this data.

Because the lesson plans are included as a set of markdown files and Jupyter notebooks, the only resource necessary to write a new lesson plan is a word processor that supports Markdown and a way to create Jupyter notebooks. While multiple programs that do either exist, a free and locally run solution (though one that is not open-source) is the note-taking app Obsidian and its extension JupyMD. However, should you take this path you will be bound by Obsidian's terms of service [5] (though it allows free use for non-profit purposes). Should that avenue be unavailable though (or should the writer prefer another solution), the extension does not prevent the use of any other Markdown word processors or Jupyter notebook managers.

To further allow for the extension to be modified and extended for future use, the team also intends to publish the source code on a public Github repository when the project is complete and allow the general public to freely adapt the project for their own use. While the team does not plan to take any royalties or impose any restrictions on how the project can be extended (aside from preventing commercial use of the extension), beyond basic documentation of the code there are no plans to provide any assistance with the modification of the extension. While this section is representative of the team's intentions for the extension at the time of writing, all of the above is subject to change should the Florida Space Institute require this course of action to be changed.

Steam Education

F. Broader Societal Impacts

Though far from the first programming teaching platform to be created, the extension does represent a small step forward for the adoption of free and open-source software in the classroom. Though numerous studies have been made on the college level attempting to increase the level of adoption for open-source software for the purpose of learning, such as Chenggui Duan's article [7], very little work has been done on the middle school level. With the Florida Space Institute's goal being to teach middle school students Python through the use of the RE-RASSOR rovers and the team's dedication to keeping the project free, open-source, and locally run, the hope between the two teams is to make learning Python far more accessible and fun at the middle school level.

G.Commitment to Refuse Generative Content.

This project, and all of its members, are committed to the active learning of software engineering principles and discourage the use of generative technology. AI Tools, such as ChatGPT, Gemini, or Copilot, will not be used for the duration of the project, including but not limited to: the development of learning tools, VSCode extensions, and the lesson plan developed for the first iteration of this project. **At no point during this project will the Team be using any of these tools, as doing so puts us at risk of misleading those who seek to use this project to learn or teach programming.** The Team is committed to doing their part in educating the next generation of software engineers, and while these tools are valuable to some, on a fundamental level, the team does not believe them to be necessary to the foundations of computer science.

II. BACKGROUND AND APPROACH

A.Background

Computer Science is a growing field, especially with the rise of Artificial Intelligence and Robotic Systems. It is becoming more important than ever to have an in-depth understanding of programming and computational thinking to contribute to these fields and be the creators behind technological change. Students with exposure to software engineering concepts and computer science-related thinking skills have a better chance of navigating these fields and future STEAM careers.

Current computer science courses and exposure in elementary and middle school education have fundamental gaps in the curriculum, teacher expertise, and student engagement and participation. Roadblocks arise from the lack of coding-specific curriculum or classes, insufficient or uneven resource allocation of materials, software, and infrastructure, and the variability in teacher preparedness and professional development [1]. It is unreasonable to expect average middle school teachers to be programming experts. As such, student foundations in coding remain weak, with possible misunderstandings of concepts creating a difficult transitionary period for students. This issue reveals an unmet need in STEAM education that can be addressed through an after-school program.

A key consideration when building out the program is the need to develop students' computational thinking skills that allow them to reason through difficult engineering problems [2]. In this regard, the focus is less on the 'correct' code and more on student reasoning to accomplish a certain task. VELA exercises,

Steam Education

as described by Shuchi Grover in “Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school”, build on this concept by relating logic problems to real-life situations [3][4]. The lesson plans of this program aim to emulate this approach by providing students with a relatable frame of reference for coding concepts.

Although students have excitement and curiosity regarding coding and robotics, a lack of background in coding creates struggle points for students in the coding curriculum centered around variables, expressions, and loops [3]. Without a strong understanding of these basic concepts, more complicated robot and rover manipulations will prove challenging, and without support, discourage students from continuing. To rectify this concern and support student learning, the program will strongly emphasize the building blocks of programming and require students, even ones with coding experience, to submit the corresponding activities before continuing to rover manipulation [4].

One of the biggest hurdles faced in schools regarding learning coding is the lack of expertise, especially when the applications of the industry are constantly evolving. This leaves teachers desiring to include computer concepts into their curriculum but not having the support, knowledge, or time to fully implement these changes in their classroom [1]. The after-school program not only acts as a student learning tool, but also aims to teach educators best practices in integrating programming and computational thinking into everyday learning [2]. The ability to reference student work in the program during class time is a key component of involving teachers in the implementation of the program. The standalone nature of the extension increases accessibility to students not in a traditional classroom. However, the full impact of the program is best seen in the collaboration between the program, the student, and the educator. This sets the teachers up for success in highlighting modern reasoning and logic processing.

B. Approach

Before each lesson truly starts, a description of the learning goals of that lesson will be shown to the student. While far from a proper implementation of the VELA exercises described in Grover's article [3], this will aid in introducing each concept and their place in writing a program before the student begins writing code, setting them up to learn the concepts first before possibly building their programming skills on a weak foundation. As such, the students should be prepared not only to learn programming during the after-school program, but also in the greater world after the program concludes.

To emphasize the basic building blocks of programming, the very first unit of the lesson plan is dedicated almost entirely to introducing these concepts, one by one. The lesson plan starts with introducing data types and variables and how they are used to store information and their role in computation through expressions and assignments, before switching to if-statements, loops, and their role in general program flow. After this, the students will be introduced to functions, their role in encapsulating and abstracting logic within programs, how they are called, and how to define them.

Due to the program's goal to teach the students not only to program, but to program a RE-RASSOR rover, the next unit will focus entirely on communicating with a RE-RASSOR rover and understanding its valid requests. To aid in this, an extension will be provided to seek out the IP address of any RE-RASSOR rovers on the local network to prevent students from having to find them manually.

Furthermore, to encourage students to program Courage to the best of their ability, a one-week unit immediately after will feature a competition.

The main content of the 9-week after-school program will be covered in the first three units. However, due to student interest and engagement in the program and the extensibility of a computer science course such as this, a ‘Unit Extra’ has been created. Unit Extra includes chapters on relevant coding concepts that, due to time constraints, were excluded from the original program. However, by including them in Unit Extra, future teams will be able to easily extend these chapters into full units and design activities around them, similar to a part two to the after-school program for students who have completed the first 9-week course.

Currently, the first chapter in Unit Extra is accessing files and file management. This chapter teaches students how to read and write from files in Python. Lesson one discusses file paths, open() pointers, different access modes for files, and provides a series of examples on file modifications. Lesson two provides specifics on navigating files and inserting specific lines without overwriting existing file data. Finally, lesson three briefly overviews retrieving objects from files.

Another chapter in Unit Extra is object-oriented programming and classes. While initially it was decided to introduce object-oriented programming and its basic principles earlier, due to the short time the program has to run, the object-oriented programming unit was moved to the very end. This choice was made in the case that, some of the time, the program has to run would be lost in some way, either due to scheduling errors or the school day the program runs on being canceled. During the object-oriented programming unit, the role of objects in further abstracting and encapsulating logic will be put front and center, encouraging students to place custom groupings of data and their related functions in a class and extending them through inheritance. In case students are unable to complete this unit within the duration of the after-school program, the database system will allow students to continue the program at their own location and pace. This provides extendability of the program past this senior-design project to add to the lesson plan and integrate more competitions in future units.

All lessons for the program were originally written in markdown files on Obsidian using a Quillcode theme. Lessons follow a content and examples, student practice, and common errors structure, with some variance depending on the topic of the lesson. The use of Obsidian to create the markdown files allowed a clear organization pattern in the lessons, with distinct headers, explanations, and example code. The organization and readability of the markdown files were emphasized as these will become printable lessons for teachers to use within the program or in school coding lessons.

The teams approach to lesson planning is to provide multiple pathways for access and extension. Markdown files act as the basis for conversion to PDFs or application to the extension. They also provide a simple way for educators to create their own lessons or courses to be integrated with the extension. Due to the open-source nature of the project, it will contain the rover program for a middle school after-school program, and also provide the framework for anyone to add their own courses. In this sense, the project is creating an overall infrastructure solution to open-source computer science education, as well as creating a specific program to fulfill an unmet need in the community with that infrastructure.

Steam Education

For implementation within the extension, the markdown files will be converted to Jupyter Notebooks, with the exception of the description markdown file for each chapter, which will remain as a markdown file in the extension. The goal of using Jupyter Notebooks is to encapsulate the coding experience within one file, where students can design, write, and run code. It also allows students to run the examples and tester code within the lesson in the appropriate sections. The use of Jupyter Notebooks is the primary mode of learning for students, and the manner in which completion checkpoints will be calculated. When students complete the content and related practice code in a cell, the checkpoint will be marked as complete, updating the progression bar.

The majority of the code and student progress will be stored locally on the student's computer during the program's active run time. The extension is designed this way to prevent complications with database cost, hardware, or setup and maintenance for future school programs and beyond. While trying to preserve the open source nature of the program, the team opted to avoid tying the extension to 3rd-party database systems such as MongoDB. Instead, student work is stored locally until manually chosen to be saved. The save process uses the .NET backend within the extension to save student progress and login credentials to a LiteDB setup. Since the database is already a part of the extension structure, no outward calls or software are needed for the program to run. This preserves the structural goal of the project while still allowing student progress to be saved, monitored, and continued individually.

For testing purposes, and the specific need the project is designed to meet, more teacher tool features will be built into the program. These features will be specific to assisting teachers or program coordinators during the 1-hour program sessions. The goal of the teacher tools is to monitor student progress and ensure students remain on task and engaged with the material. In case students are struggling with specific content, the developed tools will allow teachers to identify points of error and correct discrepancies in student learning. Additional teacher manuals will be created to facilitate this process. The manual will be similar to the lesson plans, while also including the correct answer or steps the checkpoint is looking for. With this information, the teacher can assist the student without needing coding-specific debugging knowledge. The goal is that teachers will be able to learn alongside the students in this way, increasing student and educator retention of computational and computer science-related thinking skills. These skills can then be reflected back in the classroom, further strengthening the knowledge foundation of students and equipping teachers to be empowered computer science educators.

In Spring 2026, a pilot program will be implemented in Discovery Middle School to test the effectiveness of the lesson plans and usability of the extension. The approach to the pilot program is to remain highly student-focused, learning from the methods students retain information best and editing the program to fit these needs. It also allows the team to understand what teacher tools would be best suited for the program and how scalable, designed solutions would be in the actual classroom.

The overall goal of the project is to be an integrated, accessible solution to early computer science education. As such, the team's approach to designing and building the program and extension reflects student and educator needs, with thoughtful inclusions to assist in the learning process.

Steam Education

III. THE MINIMUM VIABLE PRODUCT

The minimum viable product, as discussed with the project sponsor, Mike Conroy.

A. Create an after-school program for students to learn how to program Rovers with Python.
    a. Utilizes a Visual Studio Code extension to assist with student learning so that they can learn both with teacher assistance and on their own. This extension houses multiple features:
        i. A lesson plan that consists of units, chapters, and lessons. Units and chapters are file structures to help categorize and separate the lessons. Lessons consist of the files that the students need, which include:
            1. Jupyter Notebook files, which allow for markdown instructions within the file, as well as a clear separation between blocks of code, make it easier for students to track their progress.
            2. PDF files showing the instructions, separate from the Jupyter Notebook files.
        ii. A visual feedback system is used to help students receive a visual indication of their progress. This will allow for quicker response times for valid coding responses. As a student works through a specific cell in a Jupyter notebook file, the extension will check if the code they have created is valid. This will help the teacher help students who are struggling, instead of having

to verify students' correct work as often. Additionally, due to the limited time available after school for students to work through these lessons, it allows those who need extra time to verify if the work they complete at home is correct. The system also assists the students if they get stuck.

b. Tools that simplify the process of using rovers for teaching purposes.
   i. Tools may include…
      1. A system for managing rover access in the classroom or a simulation environment.
      2. A simplified robot interface for teacher and student use, including a system to easily find the IP of any locally connected RE-RASSOR rovers to aid in programming.

c. A schedule to provide teachers with advice on how to run the program.
   i. The account creation process for both teachers and students will be explained first, including the introduction of “teacher codes”, which will be elaborated on later.
   ii. The menu will then be introduced with all of its functions, namely the lesson tree, the ability to add and remove courses, and the linking and unlinking process of a course to a teacher code on the student’s side.
   iii. Afterwards, the teacher dashboard will be explained, including the ability to search for specific students, monitor a student’s progress through a class, group students together, and remotely unlink a student’s class from a teacher code.

Steam Education

iv. Next, the lesson view will be introduced, including an explainer of the lesson view’s anatomy and a walkthrough of an example lesson.

v. Finally, the RE-RASSOR-specific extensions will be introduced, including how to access them and incorporate the use of them into a student’s code. Furthermore, instructions should be provided to instruct the teachers on connecting to the RE-RASSOR rover.

IV. USER STORIES

A. Lesson Plan

As a teacher, **I want** to have premade lessons for myself **so that** I am not required to be as educated on RE-RASSOR software as I would otherwise be.

As a student, **I want** to cover the basics of Python first **so that** I will be better prepared to tackle more challenging topics.

As a student, **I want** lessons to be engaging and not just reading **so that** I am encouraged to try to learn to program and able to have fun while doing so.

As a student or teacher, **I want** the entire course to fit in a 9-week, 9-hour after-school program **so that** this course is quick, efficient, and able to run twice a year if necessary.

As a student or teacher, **I want** lessons to include detailed instructions that are intended to be adequate for learning without additional instruction, **so that** the large ratio between students and teachers does not inhibit the short timeframe.

As a student, **I want** the lesson plan to end with a competition between my fellow students **so that** I can be encouraged to really progress and learn the entire course, while also being able to have fun using a RE-RASSOR rover.

As a teacher, **I want** to have the ability to add new lessons, chapters, or units to the extension **so that** I can teach any additional topics or pieces of information that I see fit.

B. The Extension

As a teacher, **I want** a VS Code extension that houses and handles displaying the units and chapters containing the lesson materials **so that** I do not have to install an entirely new piece of software, which can be a challenge to navigate in a school setting for security reasons.

As a student, **I want** the extension to have an easy-to-navigate, node-based file display system **so that** I can easily locate and utilize the lesson materials.

As a student, **I want** a visible progress bar for each lesson **so that** I can keep track of my progress and feel encouraged to continue working on a lesson.

As a teacher, **I want** a visible progress bar for each student's lesson **so that** I can ensure no one is falling behind and, if they are, provide assistance to keep them on pace.

As a student, **I want** all of the lesson material to be displayed at a single click of the lesson node **so that** I do not have to manually open each component and navigate between them, saving me time and reducing frustration.

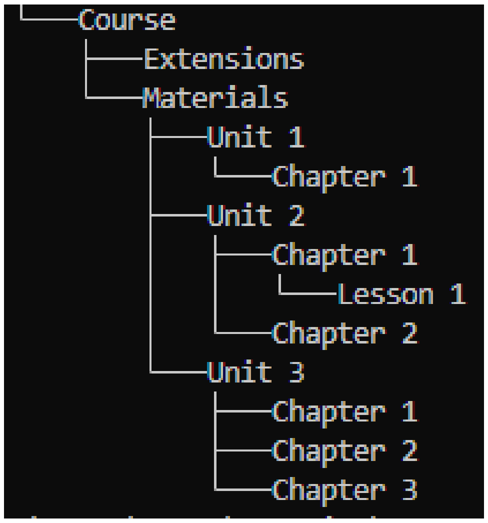
As a teacher, **I want** the extension to be able to guide the students to work through the lesson material largely on their own **so that** I can focus on helping those who are not understanding the material, instead of guiding the students to the next lesson each time they finish one.

V. THE PROGRAM

The program is, primarily, a VSCode Extension, bringing into the existing IDE a new menu, accessing the resources provided in order to display the entire lesson plan in an indented list. When a lesson is selected, it displays in VSCode using a Jupyter Notebook and pop-ups to show completion as the student makes their way through the lesson.

A.Menu

The menu is dynamically rendered, pulling assets from a folder created by the extension upon installation. VSCode will attach itself to the selected folder, and the resources will be dragged into the folder. From there, the menu will be populated based on the structure present below:



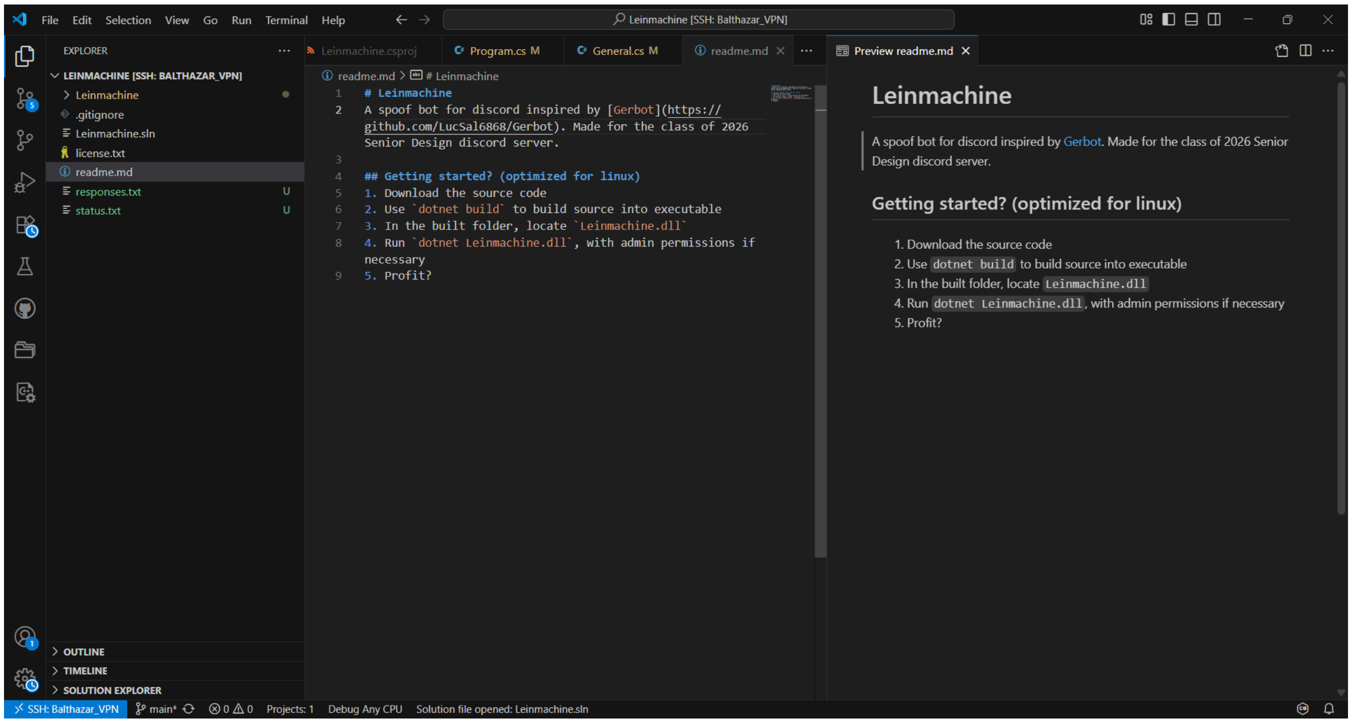
```
└───Course
    ├───Extensions
    └───Materials
        ├───Unit 1
        │   └───Chapter 1
        ├───Unit 2
        │   ├───Chapter 1
        │   │   └───Lesson 1
        │   └───Chapter 2
        └───Unit 3
            ├───Chapter 1
            ├───Chapter 2
            └───Chapter 3
```

(1): A figure showing an example of the folder structure.

The folder structure behind this menu is examined further in **V. THE RESOURCES**. Each item in the menu is clickable, and each will have a short markdown file that acts as a brief overview of the item. The Course should be described with wide-reaching learning objectives, establishing the base theme of the entire course for the nine-week club. Units are smaller subgroups of activities, focusing on a few of the base learning objectives at a time. An example of a unit would be “Object Oriented Programming,” in which the fundamentals behind OOP would be examined in an age-appropriate, simple way. Chapters examine individual concepts, such as Inheritance or Abstraction in the case of OOP, and lessons focus on the individual properties of those concepts. It is intended for a unit to be completed in 1-2 sessions, with some room for students to take work home and learn independently.

Additionally, because this menu renders items using the resources folder, it will be extensible, allowing changes to be made to the lesson plan if corrections are needed throughout the course.

The right side of the menu should display a simple markdown render, with the associated markdown file for each Unit, Chapter, and Lesson populating there to display the overarching plan, learning objectives, and progress towards completion. VSCode has support for markdown rendering through various other plugins, which may be added as dependencies to the extension. Thankfully, this is simply done when publishing the extension. A simple markdown render is displayed below, using a markdown file for a project under McElroy’s GitHub.



```
# Leinmachine
A spoof bot for discord inspired by [Gerbot](https://
github.com/LucSal6868/Gerbot). Made for the class of 2026
Senior Design discord server.

## Getting started? (optimized for linux)
1. Download the source code
2. Use `dotnet build` to build source into executable
3. In the built folder, locate `Leinmachine.dll`
4. Run `dotnet Leinmachine.dll`, with admin permissions if
necessary
5. Profit?
```

Leinmachine

A spoof bot for discord inspired by Gerbot. Made for the class of 2026 Senior Design discord server.

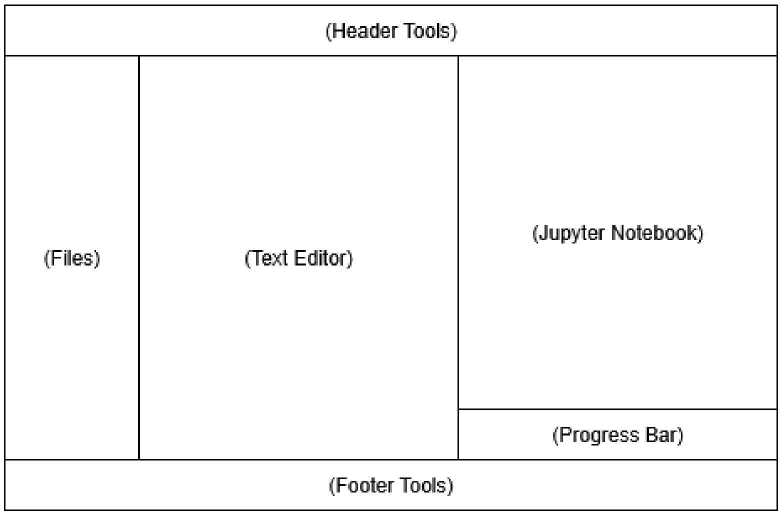
Getting started? (optimized for linux)

1. Download the source code
2. Use `dotnet build` to build source into executable
3. In the built folder, locate `Leinmachine.dll`
4. Run `dotnet Leinmachine.dll`, with admin permissions if necessary
5. Profit?

(2): A figure showing an example of a markdown file being displayed in Visual Studio Code.

B. Lesson View

The lesson view is simple, with the VSCode interface split evenly in half down the middle of the screen. On the left side is the text editor, displaying the current code file that the student is working on. On the right side is the Jupyter Notebook, essentially giving the step-by-step instructions for the lesson. The right side will also display a simple progress bar, which will indicate student progress through the lesson. Jupyter is the system that the team has chosen because it supports a wide variety of languages that can run in block sections within the notebook, which will allow the lesson documents to demonstrate compilation and function of certain aspects of programming firsthand for the students to then emulate. The team does not intend to make the Jupyter notebooks editable by students, they are meant to demonstrate the concept in an isolated manner. Below is a simplified rendering of a lesson view:



(3): A figure showing an example of how a normal lesson's components will be arranged.

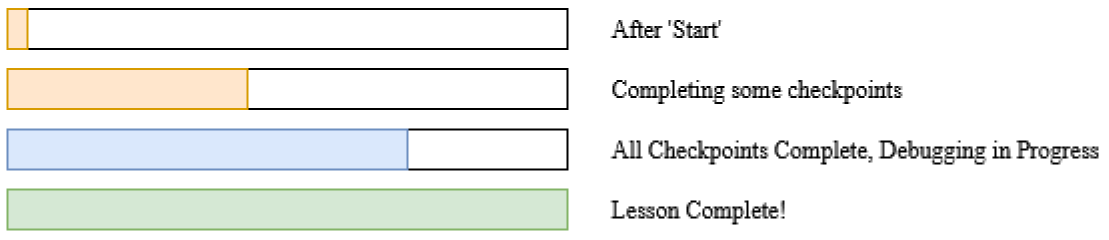
The left side of the screen will function as usual, providing access to VSCodes' wide breadth of tools and debugging equipment. Ideally, one of the early lessons in the program would be a short introduction to this interface, explaining concepts for debugging and running code that has been written by students, while offering the ability to skip to the more complex material if the user is already familiar with the IDE. The flexibility of this environment is meant to be a major feature, allowing students to move windows around as best fits their needs.

The right side is intended to be more locked down, providing the staple features of the Jupyter notebook, such as running code snippets from within the notebook. Additionally, this means that the extension requires the extensions necessary to render Jupyter files.

C. Checkpoints & Lesson Completion

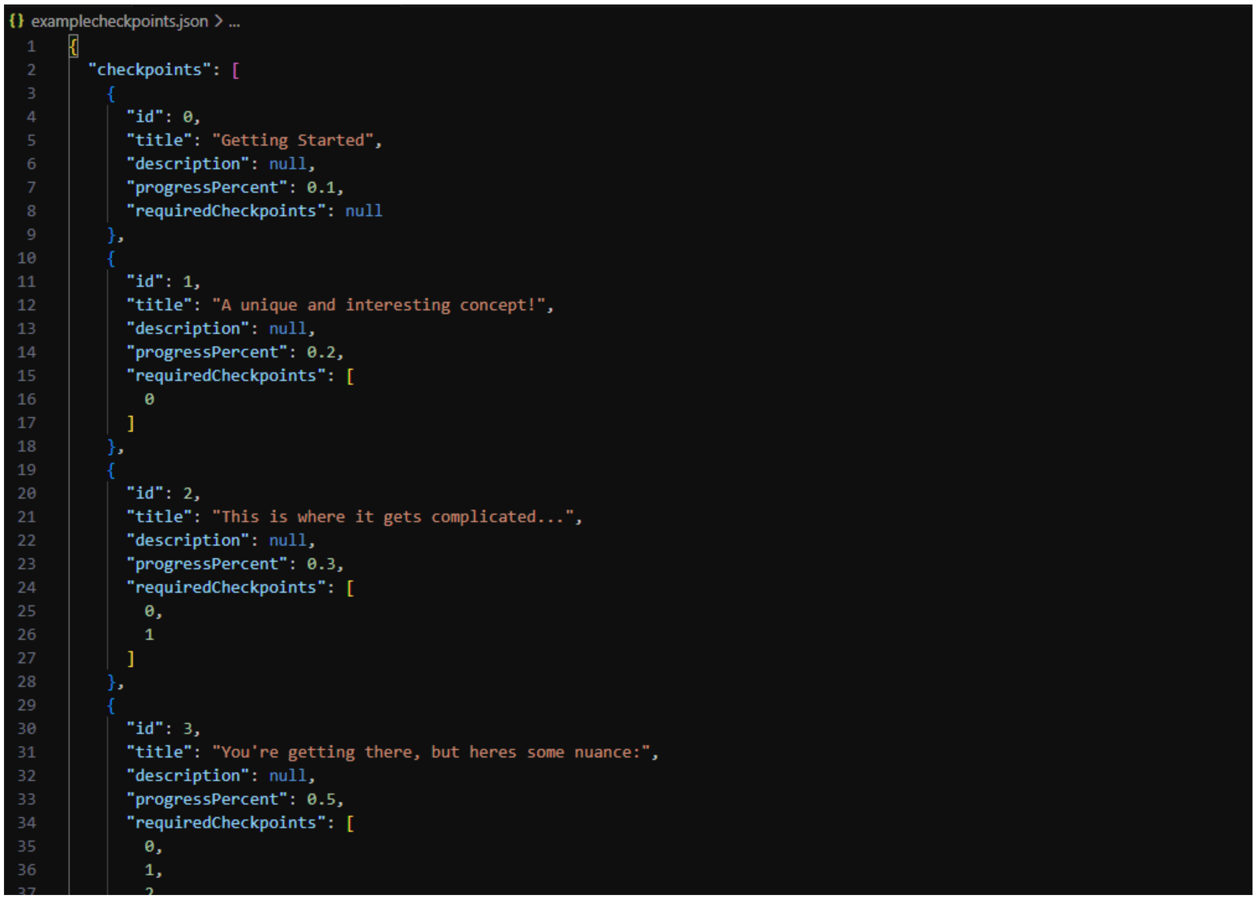
Pop-ups will show periodically throughout the lesson, highlighting the completion of key aspects of the lesson. These functions use ‘Checkpoints’, JSON tags in the lesson folder that look for a specific string to be present in the text editor. The Checkpoints contribute to the overall completion rating for the lesson. Ideally, when a checkpoint is reached, the relevant information is checked off in the Jupyter notebook, giving the student a clear idea of where they should be looking in order to move on to the next step.

Lesson completion is calculated based on a few factors. The first and primary contributor is the aforementioned checkpoints, but completion also requires no errors in syntax or compilation, and in cases where there is an associated unit test, completion of said unit test. Unit tests for lessons should always provide feedback on what went wrong and give the student an idea of where they should be checking the code that they’ve written. A progress bar for lesson completion will also be provided, color-coded based on what is missing for the final steps. The bar will display as Orange while the students are working on the code, Blue once they’ve completed the code and have moved onto debugging and compilation, and Green once all objectives for the lesson have been completed.


After 'Start'
Completing some checkpoints
All Checkpoints Complete, Debugging in Progress
Lesson Complete!

(4): An example of how the progress bar will change as a user progresses through a lesson.

The checkpoints themselves are structured in a json format, taking in a few fields- including the popup content, title and buttons, the progress marker (e.g. what the student has to have on the page in order for progress to be confirmed), and, if needed, references to previous objectives that need to be completed prior to progress being advanced to the currently selected checkpoint.



```
{} examplecheckpoints.json > ...
{
  "checkpoints": [
    {
      "id": 0,
      "title": "Getting Started",
      "description": null,
      "progressPercent": 0.1,
      "requiredCheckpoints": null
    },
    {
      "id": 1,
      "title": "A unique and interesting concept!",
      "description": null,
      "progressPercent": 0.2,
      "requiredCheckpoints": [
        0
      ]
    },
    {
      "id": 2,
      "title": "This is where it gets complicated...",
      "description": null,
      "progressPercent": 0.3,
      "requiredCheckpoints": [
        0,
        1
      ]
    },
    {
      "id": 3,
      "title": "You're getting there, but heres some nuance:",
      "description": null,
      "progressPercent": 0.5,
      "requiredCheckpoints": [
        0,
        1,
```

(5): An example of the JSON file for checkpoint instructions.

The example above is truncated from a typical file for conciseness – usually, a brief version of the next step would be included in the description field, coinciding with the lesson’s next steps in further detail on the left to locate the student in the notebook.

D.Debugging and Unit Tests

Unit tests are packaged with the lessons in the resource structure. Additionally, the debugging tools available in VSCode are entirely available to students. At the time of writing, it is uncertain whether there will be a chapter on the tools present in VSCode that are available to students. Some languages may require additional extensions in order to support Unit Testing functionality, such as allowing students to run tests on an individual basis. The Simulation, presented as a stretch goal in the introduction, would be an injectable debugging and unit testing executable that would simulate a live rover in a game engine. Running the program would compile the code for the rover to use, and from there display a simulation of the rover navigating through whichever task was required of it. Tools such as this would be packaged in the top-level directory of the Resources structure.

E. Frontend and The Visual Experience

A key concern highlighted by the sponsor, Mike Conroy, was the transition of students from block code to professional text code. Initial discussions centered around building a block code interface that would slowly transition to Python by elaborating on details of each block, turning the block into an outline, until eventually the student is fully coding in Python. However, correspondence with the teacher in Discovery Middle School, where the pilot program will be run, revealed that not all students have coding experience, block coding, or otherwise.

Steam Education

This shifted the front-end aspect of the project from a transitional model to one centered around teaching Python in a text-based, example-driven format. The introduction of a block coding interface, quickly changing to text in less than a 9-week timeframe, has the possibility of being more confusing for students than helpful. However, to accommodate students with block-coding experience, there will be a “Block-coding Equivalent” section at the end of each lesson in Unit 1 with an image of what the block code would look like.

Instead, the project integrates professional keyword highlighting, similar to highlighting traditionally present in VSCode, into the lesson plan itself. For students familiar with block coding, the coloring is helpful to relate coding keywords to concepts they might already be family with. For students new to coding, the visual experience is a simplified version of professional development in an integrated development environment (IDE) such as VSCode.

The overall extension visuals are meant to compartmentalize the various sections of the program to increase access to resources and reduce distractions. Students will not have to balance multiple tabs, learning materials, or page changes because all resources needed are integrated into a single view meant to highlight instructions, activities, and help.

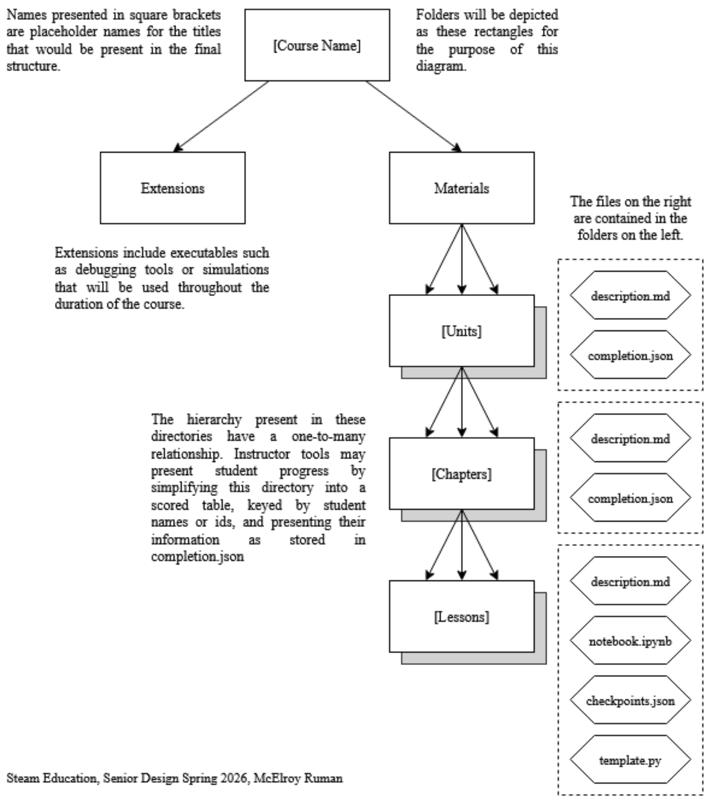
The effectiveness of the visual interface will be evaluated during the pilot program in Discovery Middle School. Students will provide feedback on which aspects of the program were helpful, engaging, overwhelming, or unneeded. Based on this feedback, edits to the visual experience will be made to enhance student retention. Although the extension uses a professional development tool and is meant to emulate a professional experience, it is still important that the visual experience is one that is captivating and interactive with students within an 11-13 age range.

VI. THE RESOURCES

Lesson plans are packaged in such a manner that would be best to distribute in a compressed archive, with the headers of Units, Chapters, and Lessons existing as folders within a directory tree.

A. Structure

The figure, shown below, describes the file structure of a lesson plan.


Names presented in square brackets are placeholder names for the titles that would be present in the final structure.
[Course Name]
Folders will be depicted as these rectangles for the purpose of this diagram.
Extensions
Materials
The files on the right are contained in the folders on the left.
Extensions include executables such as debugging tools or simulations that will be used throughout the duration of the course.
[Units]
description.md
completion.json
The hierarchy present in these directories have a one-to-many relationship. Instructor tools may present student progress by simplifying this directory into a scored table, keyed by student names or ids, and presenting their information as stored in completion.json
[Chapters]
description.md
completion.json
[Lessons]
description.md
notebook.ipynb
checkpoints.json
template.py
Steam Education, Senior Design Spring 2026, McElroy Ruman

Steam Education

(6): A figure showing the folder structure of a course.

When a student first enters a lesson, the main screen will be populated with the template code from template.py, and the right side will show the Jupyter Notebook stored in notebook.ipynb, which contains the instructions and test cases for the lesson. As progress is made and checkpoints are reached, checkpoints.json will be updated to align with the student's progress. When the student finishes all the checkpoints in a lesson, the lesson is considered "complete," and the completion.json file for both the chapter and unit will be updated to reflect this. description.md will contain a general overview of the lesson, chapter, or unit that it is assigned to, but will not contain any general instructions. Finally, extensions can be called upon by any lesson as long as the extension exists within the course, and the data on which extensions to use in a lesson will also be stored in checkpoints.json. Furthermore, all of these files are local: that is to say, the completion.json on one student's computer will only contain the completion.json for that student, and will have to be synced with the MongoDB server separately by the backend.

B. Importing A Course

When opening the extension for the first time, it will ask the user to import a course and open up a window in the operating system's file explorer. Upon selecting a .zip file containing a course, the course will be unzipped and stored within a "courses" folder of the extension's install folder. Furthermore, additional courses can be installed through an "Install A Course" option in the main menu, and unnecessary courses can be removed through a "Remove A Course" option. Upon selecting a course to remove or install, a dialog box will be created to ask the user to confirm if they really want to install or remove that course. Removing a course will not delete the user data for that course, which will already be uploaded to the MongoDB server as described in VI. Backend.

Steam Education

VII. BACKEND

The program runs as a VSCode extension; however, it will have a comprehensive database meant to store student login credentials, lesson plans and teacher tools, student learning progress, and student code. The backend is vital to increasing the portability of the program and extending the reach of early programming education.

VSCode was selected as the delivery software for the program and lesson plan, as it is a professional-level software development tool. For students who continue in computer science, they will spend much of their higher education and professional careers working with VSCode. Thus, an early introduction to professional tools reduces the transition period for students and instead highlights programming learning outcomes. This method was selected to address teacher concerns regarding the applicability of block coding through organizations such as Code, MIT's Scratch, or Google's Blockly to professional development.

A.Teacher Tools

The teacher tools encompass a suite of extensions to the program, allowing an instructor to monitor the progress of the students in real time and help with difficult lessons. These tools are meant to be field-deployable, running exclusively as necessary when the class or club is in session, and are not necessary for the successful function of the software in the long term.

Included with the teacher tools is an instructor guide, providing information and insights on how to create personal lesson plans, modify existing plans, and the process for developing the lesson structure. In this case, this is one way that the team is avoiding future technical debt for any teams that come along later down the line, as this also acts as a simpler breakdown of the lesson resources and structures involved. The team recommends the use of the application Obsidian, which is what was used in the process of the lesson development for this project, in order to create the pages of the textbook, though you can also write markdown in any other text editor or markdown visualizer to a similar effect. Jupyter Notebooks were chosen as the in-app demonstration tool because they allow us to combine the existing printable markdown files with interactive segments that demonstrate specific functions using an easy-to-learn, fun-to-interact-with format.

Lessons are generally comprised of a specific goal and several steps with which to accomplish that goal. The checkpoints tool allows the instructor to indicate to the extension which specific objectives should be completed by the student and when, in order to adequately make progress in the lesson. For more information regarding how checkpoints are processed and interacted with, visit V.C (page 23) for the technical explanation. The checkpoint tool asks the instructor for a line of code that must be present in the student's workspace, a title and description for the trigger, and the increment of progress that is indicated by completion of the associated step. This can also be set in an iterative mode, where instead of asking the teacher to input a completion percentage in the form of a decimal, it calculates the completion rate in equal steps, dividing the total (1) by the number of steps present, and dynamically adding that information as more steps are added. The downside to this approach is that it equivalently weights all steps of the process as equal, though some steps may need more effort than others.

Steam Education

B. Serialization and “Grading”

When a user is independently using the extension and lesson tools without a connection to a club or class, all work is saved directly to their native device. As such, any progress made is local and must be saved or synchronized voluntarily through platforms such as GitHub, Dropbox, OneDrive, or other similar competing services. As long as the top-level folder of the course is saved and can be reopened by a Visual Studio Code editor with the extension installed, the data should be able to be read, and thus a local copy can be backed up.

This, however, presents an issue in the modern classroom. In many cases, students may not possess their own personal devices, but instead share a classroom set and are not guaranteed the same system every time that they attend the class or club program. As a result, an additional teacher tool is necessary. The synchronization tool has several functions, but the primary purpose is to automatically save student progress and commit them to a local SQL-based database, running LiteDB, a tool built for such a purpose. Running it in this manner allows for the solution to work in many environments that are not necessarily the most technically up-to-date, using a variety of network and device configurations, as well as keeping the data structures as compact as possible. This approach also allows any data held by the program officials to be kept secure by their own standards, as it may contain personal information pertaining to student success within the program.

Another benefit of maintaining a consistent connection with student devices is that it allows the educator or any assistants to maintain a clear view of which students need assistance during the program. The students are given the ability to raise their hand within the extension, which indicates to the instructor's view in which order students need help and what they may need help with based on their progress and completion. For example, if the student is stuck on a specific checkpoint completion step, the application may bring up the lesson text for the instructor to review and refresh on before approaching the student with an adequate solution. This accomplishes two major goals: the preservation of progress held by the student while they are completing the lesson that they are on, and making sure that the teacher is up to date and armed with the resources they need in order to best help the student requesting help.

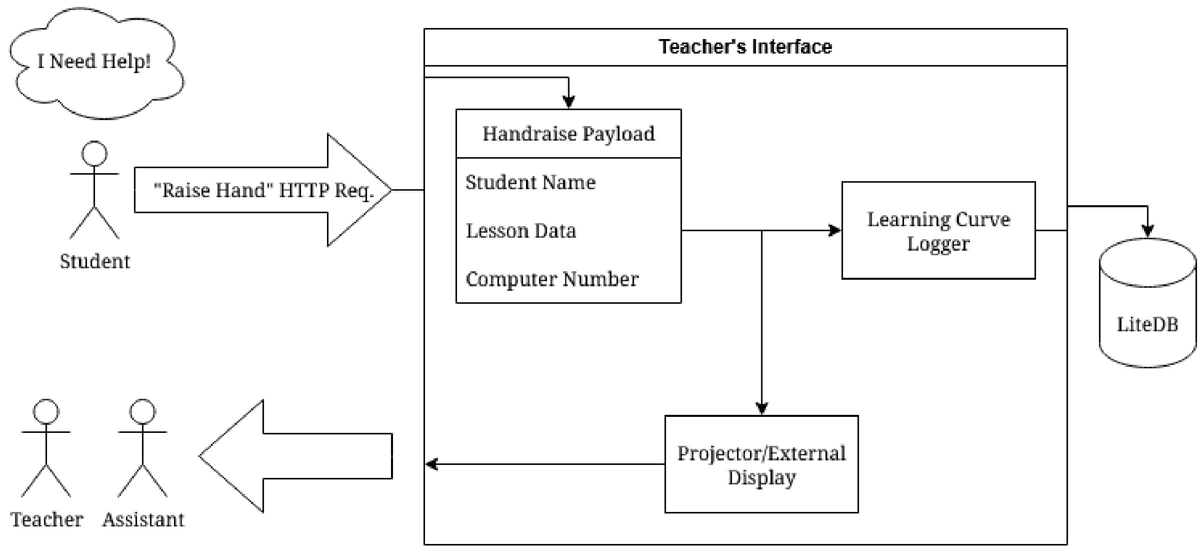
The synchronization tool also allows the teacher to print out a report of each session that is held, elaborating on student progress in a visually detailed manner, showing where they began at the start of the day versus where they were by the end of the class. A chart of student progress may look like a set of bar graphs depicting student progress ranked against their previous work. This allows the instructor to have a bird’s eye view of class participation and effort, as well as provides a deliverable proof of success for a student to demonstrate to their parents in order to not only provide incentive to continue the program, but also to facilitate student success and motivation via standard reward mechanisms. Additionally, if the program is being held in a class environment, this allows the teacher to generate a grading scale based on student progress, as opposed to comparing classmates to one another on a linear basis. Computer science as a field can be more difficult for some than others, and as a result, it is generally more fair in the long term to compare a student to themselves as opposed to the best of the students in the class.

Steam Education

A small problem with this design is that it requires a simple authentication system that sends requests to and from student computers within the school network. Depending on the network configuration, this may create issues that need to be solved before the program is run for the first time. It also makes it so that the instructor must become the administrator of their program, solving simpler issues such as students forgetting usernames or passwords, progress being lost due to a connection issue, or other unprecedented problems. As much as the team would like to claim otherwise, errors are inevitable in the scale of such a program, and the team looks forward to mitigating any of these issues before they arise in a real-world environment.

C. Hand-raising and Student Success

A feature of the teacher tools is hand-raising. This allows the instructors and volunteers to get a clear view of where students are struggling most in lessons, and make adjustments over the course of the program. When a student presses the button to ‘raise their hand,’ it indicates to an instructor which student needs help, what lesson and checkpoint they're on, and what information they may need in order to facilitate that student’s understanding.



(7): A figure showing a use-case diagram, focusing on the feature of hand-raising.

Additionally, the hand raising process marks in the synchronization system that a hand was raised at a specific lesson, at a specific checkpoint, giving diagnostic information to find areas of difficulty within the lesson structure, identifying problem areas that may need more elaboration or understanding over the course of units, chapters, and lessons. It may indicate that a new lesson may need to be created and inserted in order to introduce a foundational concept that was missed during the plan creation, or give insight into the understanding of a particular section relative to the wording or examples shown in the Jupyter notebook.

D.Technologies

The framework selected for the backend tools is .NET, for its cross-platform compatibility and versatility between frontend interfaces, backend api development, and familiarity. It is compatible with most major operating systems, with new developments facilitating widespread adoption. This allows the broadest audience for the extension and tools present as part of this program, and facilitates this as one of the broader goals of the program as it is developing. The majority of the tools above rely on the ASP.NET toolkit for accepting HTTP requests from connected clients.

For database services, the team selected the PseudoSQL/NoSQL platform LiteDB. This service accepts SQL syntax for queries and data entry, but stores data as an embedded database file that can be streamed or dumped to local data solutions. It's extremely fast and lightweight, and is entirely written in .NET C#. Additionally, LiteDB is completely free and open source, with long-term support being provided by a wide-reaching community. More information about LiteDB can be accessed at their Github page.

VIII. LESSON PLAN

The lesson plan has been developed as a textbook resource for the program. It features 3 Units, 2 of which feature the basics of Python and then the all-important rover chapter.

A.Program

1. Unit One: Building Blocks of Programming
    a. Chapter 0: Welcome to Python
        i. Introduction
        ii. print("Hello World!")

b. Chapter 1: Data Types and Variables
    i. Lesson 1.1: Variables
    ii. Lesson 1.2: Datatypes
        1. Int, float, string, bool
    iii. Lesson 1.3: Operators
    iv. Lesson 1.4: Print Formatting
    v. Lesson 1.5: Lists
    vi. Lesson 1.6: Dictionaries
c. Chapter 2: Conditionals and Loops
    i. Lesson 2.1: So what if?
    ii. Lesson 2.2: For loops
    iii. Lesson 2.3: While loops
d. Chapter 3: Functions
    i. Lesson 3.1: Declaration
    ii. Lesson 3.2: Content
    iii. Lesson 3.3: Calling Functions and Return

2. Unit Two: HTTP requests
    a. Chapter 4: How To Communicate With The Rover
        i. Lesson 4.1: So where’s my rover?
            1. A brief discussion on how communications to the rover are processed.
        ii. Lesson 4.2: Preparing a message
            1. The steps to post a request
    b. Chapter 5: JSON
        i. Lesson 5.1: What is JSON?
        ii. Lesson 5.2: Dictionary to JSON

Steam Education

iii. Lesson 5.3: JSON to Dictionary

c. Chapter 6: POST time
   i. Lesson 6.1: The POST Request
   ii. Lesson 6.2: The Rover’s Requests
   iii. Lesson 6.3: POST Errors
   iv. Lesson 6.4: The Secret Sauce
      1. ArUco markers

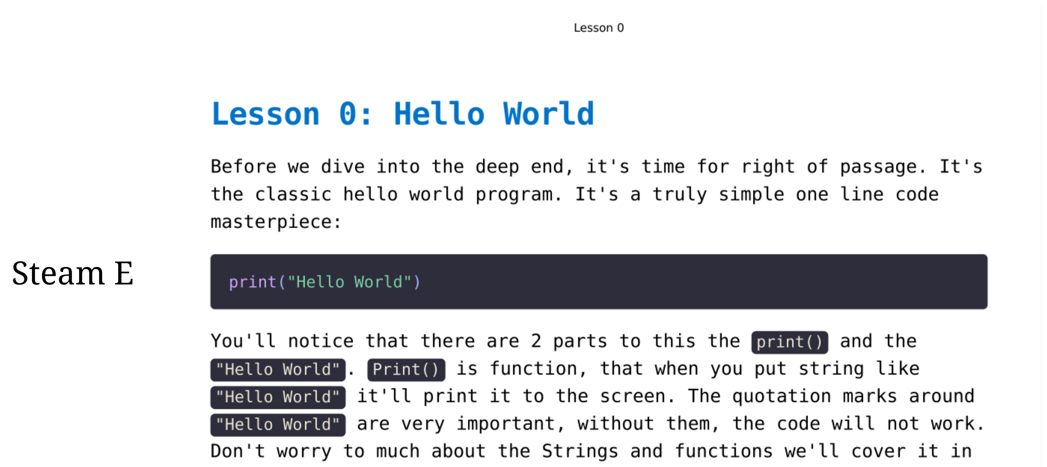
d. Chapter 7: Challenges >:)
   i. There will be a collection of challenges and lessons on logic.

3. Unit Three: The Competition
4. Unit Extra: Files and Classes
   a. Chapter 8: File Access and Management
      i. Lesson 8.1: The open() Function
      ii. Lesson 8.2:
   b. Chapter 9: Basic Classes
      i. Lesson 9.1: Your very own data type!
         1. The use case of classes
      ii. Lesson 9.2: __init__
         1. Constructors
      iii. Lesson 9.3: your own methods too!
      iv. Lesson 9.4: __str__

Each of these lessons features a breakdown of the topic and examples, if applicable. The team starts by introducing the topic and then diving into a ‘Running the Code Section’ to let the kids try it out for themselves. It’s followed by a ‘Bugs and Issues’ section to help those who are struggling with errors. This flow may repeat itself in a lesson if there are multiple topics. This structure loosely follows an I-do, we-do, you-do approach. The I-do aspect of the lesson plan is primarily content explaining coding concepts. The we-do portion is the examples and corresponding explanation to showcase to students what the expected outcome should look like. Finally, the you-do section is the ‘Running the Code’ section that emphasizes student testing.

A key aspect of coding is the ability to write the same piece of code in multiple different ways, essentially meaning there is no ‘wrong answer’ in terms of specific commands used. Instead, the goal of the lesson plans is to build computational thinking skills in students while providing guidance on possible combinations of code to try. As a result of this approach, many of the activities integrated into the lessons will not have a ‘right’ answer. Instead, students are encouraged to utilize their own logic to write code. Completion bars will be dependent on whether the results outputted to the console match the expectations of the lesson. For example, if the assignment expects a variable assignment to a string, the processor will look for an appropriate variable name, the equals operator, and some text enclosed by quotes.

This method of grading allows flexibility and creativity in student learning, a beneficial aspect for the competition and real-life application of code, where there are no strict guidelines or step-by-step plans on what to code. Instead, students are provided an end goal they think through to complete.



Lesson 0: Hello World

Before we dive into the deep end, it's time for right of passage. It's the classic hello world program. It's a truly simple one line code masterpiece:

```
print("Hello World")
```

You'll notice that there are 2 parts to this the `print()` and the `"Hello World"`. `Print()` is function, that when you put string like `"Hello World"` it'll print it to the screen. The quotation marks around `"Hello World"` are very important, without them, the code will not work. Don't worry to much about the Strings and functions we'll cover it in

(8): An example of how a lesson’s markdown file will appear.

The lesson plan is built on the assumption that students can complete one chapter in one hour, approximately the length of a weekly after-school club meeting. Each chapter has approximately three to five lessons, with the initial introductory chapters being more extensive. However, the lesson plan is built with flexibility and differing student backgrounds in mind, allowing students to skip introductory chapters to focus on rover-specific capabilities.

Each chapter in a unit will have a corresponding ‘Description’ file that outlines the agenda for that particular session. It will outline the overall objective, general concepts, and connect the concepts to the end goal of the program. At the request of the teacher correspondent, Ms. Robinson, the chapter objectives will become a part of a journal kept by each student, primarily acting as a note-taking and physical progress log of coding concepts. This will become a key takeaway for students that can also be referenced in the day-to-day operations of the classroom.

The lesson plans must remain personable and engaging for students with a clear voice and path to success. As such, lessons are written in an approachable tone, interweaving appropriate humor and connection to students. The after-school program, while supported by a teacher, will primarily be operated through the VSCode extension. Teachers and facilitators do not need to be coding specialists; this requires the lesson plan and program to not only be a teaching tool but also an interactive feedback module aimed at teaching, listening, and guiding students through coding challenges.

Student progress in the lesson plan, as indicated by the completion bar, will be saved to a database and can be retrieved with the proper authentication and the program's reboot for the next session. Students will be able to pick up from where they left off, allowing a self-paced module that can be continued and referenced in a home setting as well. This leverages the portable nature of VSCode, furthering the impact on students to learn on their own and showcase results during school meetings.

B. Unit 1

Unit One covers the building blocks of programming. Students with previous coding experience, either in block coding or Python, Java, and other text-based languages, will find this unit will be a refresher of common coding concepts. Students will be able to skip text-based lessons, but will be required to submit the activities from the ‘Running the Code’ sections to gauge chapter completion and student understanding. This allows the program to provide real-time assistance on knowledge gaps and misunderstandings that the students may have. The goal of the lesson plan structure is to take into consideration varying backgrounds and provide adequate challenges and support in all aspects of basic coding. This will standardize the student population and allow for common bugs to be resolved before coding the rover.

Unit One comprises three chapters. Chapter One is a review of data types and variables, the starting point to understanding more complex data storage and retrieval methods. Lesson one details the definition of a variable, when they should be used, and conventions for naming variables. Unit One defines the primary structure that the remaining course will follow. So, although lesson one is small, it still includes a brief Running the Code section to test student understanding. Here, students are tasked with creating an appropriate variable name and assigning it a value. An example completion checkpoint would be the proper creation of a variable and assignment to a value. Student code will be checked to ensure there are no errors, and if students are meeting the standard for an appropriate variable name as laid out in the lesson. The Bugs and Issues section outlines common errors, directing students to additional lessons as needed.

Lesson 1.2 expands on the values to be placed in variables by explaining data types. Each datatype is clearly separated with headers and subheadings and provides a relatable example of each. This lesson is primarily text-based but includes some thought questions and answers for students struggling to distinguish between the datatypes and understand their conventions.

Next up is math class for students, discussing how operators and concatenation allow students to do basic computations. A new organizational pattern introduced in Lesson 1.3 is Side Note - Topic. The side note topic introduced in this lesson is comments. Comments are a key feature in programming as they increase readability and understanding, assist in debugging, and aid in documentation. In this lesson, students are encouraged to comment on example code with their understanding of what should happen. This exercise is important as it encourages students to interact with the content before individual practice, similar to a we-do approach to learning. The Running the Code section tests students' application of math concepts, such as the order of operations, to code. The Bugs and Issues section teaches struggling students how to decompose larger problems into smaller, more manageable chunks, a key component in computer science and software engineering.

Lesson 1.4 is an introduction to print statements and formatted printing with a corresponding student activity. Lessons 1.5 and 1.6 are the students' introduction to data structures, lists, and dictionaries, respectively. For both of these lessons, the identifying factors of the data structure are described with examples, and then students are directed to practice by coding the appropriate data type. Students will have completed the lesson with the compilation and running of the code with no errors.

Unit One, Chapter Two is all about conditionals and loops in Python. Lesson 2.1 covers if statements, lesson 2.2 for loops, and lesson 2.3 while loops. In the original planning of the lessons, switch statements and nested loops were specific topics. However, due to time constraints and the feasibility of exposing students to multiple different topics with the expectations of quick understanding and application, switch statements are excluded, and nested loops are based on the students' understanding and testing of coding concepts. As the lessons begin to delve into more complex concepts that build the logical foundation for rover movement, the lessons begin to relate the content to rover and space terminology and take on a more approachable tone. The thought behind designing the lessons this way is to maintain student excitement towards the robotics aspect while learning more challenging concepts.

The final chapter in Unit One is Chapter Three on functions. Functions are the first major introduction students have to encapsulating logic and the single responsibility principle. Lesson 3.1 covers how to define functions with parameters as needed. Lesson 3.2 focuses on the content and emphasizes indentation. Students are encouraged to use the coding concepts and structures from previous lessons to build functions with creative tasks. Finally, Lesson 3.3 is a brief overview of return statements and the overall goal of a function. Throughout Chapter 3, students are building one function part-by-part. This is to emphasize the structure and importance of each piece, as well as allow students to test different logic schemes and use repetition to gain confidence in function creation and calls. Functions were decided to be the last chapter in Unit One and the last section of basic coding concepts, as students will begin making rover-specific function calls in Unit Two.

Chapters are designed to be completed in one session, approximately one hour. However, Unit One covers many basic principles that can be overwhelming for students. Lessons have been edited down to the most important concepts needed for rover logic and manipulation. Students unable to complete all the activities in a chapter during the session are encouraged to complete the remaining information at their own time and place, to be prepared for the next session the following week. Activities in Unit One are primarily to test student' understanding of declaring and initializing their own variables, data types, data structures, loops, and functions with the appropriate values. It aims to show students the importance of details in code, exposing them to common errors in the basics before reaching more complex rover-specific programming.

C. Unit 2

Unit Two is where students start building the special skills required for the competition. This is where students get to building JSONs, making POST requests, specific POST requests on the rover, and the introduction of the ArUco markers.

These are the crucial skills needed to interact with the RE-RASSOR rovers, so ensuring that these are explained well is paramount to giving the students the best chance of success.

The unit opens with a chapter on how communication to the rover works. The lessons explain the motivations behind coding with the rover rather than just using the controller: The impracticality of operating the rover 24/7, a shakier connection over long distances, and the rover's limited hardware. This is important so that the students understand why rovers are coded this way instead of just using the given controller. After all, it’s very easy to fall into the mindset of “why bother”, which can be detrimental to one's learning. Then the students are given a preview of the full post request before moving along to the next chapter.

Before the fun parts, the team needs to hit JSONs so students can send data over POST requests. Fortunately, there is a JSON library in Python. So with that and a quick review of Dictionaries. Students should be able to turn dictionaries into JSONs and vice versa. This is the last major step before being able to make a post request, so keeping it as simple as possible was a priority, and thanks to the library, it is.

Now, finally, the team gets to actual post requests. This will be the final week of teaching before the competition and thus the most crucial. The team starts by using the public API learning site https://api.restful-api.dev to practice the basics of making post requests. Then the team shows all the post requests that the RE-RASSOR rovers have at their disposal to make them move. The team will also give them an exercise to practice rover movement. Finally, the team introduced one last POST: the ArUco markers. It’ll be kept to a brief explanation of what the markers stand for and a demonstration of what can be achieved with them.

D.Unit 3

To measure student understanding of the lesson plan, the 9-week after-school program culminates in a friendly rover competition. This unit consists of challenges that the teacher can use for the competition. Ranging from really basic to complicated. The team expects the classes to do 1-3 of these challenges, depending on whether they pick the harder or easier ones.

The first challenge the team came up with is called the 2 Meter Dash. In the 2 Meter Dash, a distance of 2 meters is marked out on the floor, and students must get the rover as close as possible to the 2 meter mark without going over. They get one practice run before the main event. This event is fairly simple and serves as the easiest to set up. It's well-suited for a class of students who may be struggling, or if the program is running short on time.

Then there is a full suite of mazes. The first of which is the Follow the Signs Maze. Signs with ArUco markers are placed in the maze, and students must use the ArUco markers to navigate the maze as quickly as possible. Here, the students must program the rover to find the markers and act upon them. This is more challenging and may take students a few attempts, but the goal here is to find the general solution instead of going for a specific one.

The next maze is the Guided Maze. Where the students have 5 ARUCO markers, they can hold them in front of the rover. They'll need to program the rover to identify the ARUCO markers as certain commands to reach the end. The goal is to reach the end as quickly as possible, but students who use fewer ArUco markers get a multiplier to their final score. This challenge has students code the rover in a way that allows them to manipulate its movement using the signs. The bonus challenge of using fewer ArUco markers is meant to encourage students to attempt some creative solutions and encourage advanced problem-solving.

Up next is the Guided Maze's twin, the backwards guided maze. Here, the students must complete the guide maze challenge, but they aren't allowed to move the rover forward. It has the same challenge as the guided maze, but with some inverted turns to overcome. This makes a great sequel to the guided maze that can be run swiftly after completing the other.

This last maze is the most difficult maze of them all, because it requires the use of Python's underused feature: interpretation. In the Terminal Maze, students aren't coding into files this time; they are coding the post commands to the Rover in real time. The student with the quickest time wins the challenge. This is a true test of typing and post request knowledge.

Now, for a different type of challenge, Value Locator has students program the rover to find and approach a box labeled with a certain value. The Value in question isn't known until the time is up, so students must program the rover to find a generic value. The boxes will be sorted in a certain way by the teacher, and if students can figure out how to leverage the sorting to optimize their search, then victory will be theirs.

Finally, but not least, have the marathon event: Rover's Odyssey. It's a combination of Guided Maze, followed by Value Locator, and lastly a 2 Meter Dash towards the value. Students will need to code the rover to do each of these tasks back-to-back. If they've done all of these before, then it would be an excellent opportunity to reuse code and link it all together. Ultimately, this is a challenge of breaking it down into small tasks and then putting it all together. A classic engineering principle.

E. Unit Extra

Containing information that isn't strictly necessary for learning how to program the rovers, and being able to be taught either before or after the competition, Unit Extra mainly goes over accessing files through Python and defining classes. While initially intended to contain only information about classes and inheritance, the team eventually had to decide whether to teach file access and, due to the very limited time the program ran, what to cut if it were to be included. The team ultimately decided to cut inheritance for the sake of teaching file access, though the lesson plan for the chapter on classes will mention it.

The first chapter of Unit Extra, Chapter 8, is about file access and management. Its first lesson starts by teaching the absolute basics on what a file path is, how access modes work, and how to return a file object using open(). From there, it introduces the f.read() and f.write() functions, using them in their most basic form to read a whole file and write information to the end of a file. Finally, the first lesson closes out by introducing the f.readline() and f.readlines() functions to read a file line-by-line.

The next lesson opens by delivering a warning about how writes in Python work: instead of inserting data into a file and leaving pre-existing data untouched, f.write() will overwrite existing data without warning if there is any in front of the file object’s current location. Afterwards, it teaches how to easily insert lines into an existing file by opening a file with the “r+” access mode using f.readlines() to split a file into lines. Then, the new line (or lines) gets inserted into the returned list using list.insert(), the file is zeroed out by using f.seek(0, 0) and f.truncate(), and all the lines are written back into the file. Meanwhile, the final lesson teaches students how to serialize and deserialize object data in Python to and from the CSV format to easily store and retrieve information and carry it between runs of a program.

The second chapter of Unit Extra, Chapter 9 (and the final chapter overall), focuses on how to create a custom class, give it variables and functions, and define both a constructor and a string representation. While understanding classes and objects does give programmers a great advantage and helps to further teach the concept of encapsulation, it requires a decent understanding of variables, functions, and control flow. Because of this and the need to get students working with Courage sooner rather than later, for the sake of proper pacing and keeping their interest, the proper introduction of classes and objects had to go near the end.

Chapter 9 opens with introducing the concept of a class and showing how to define variables without a constructor, setting students up to be able to use simple classes to store data in a fashion similar to structs in C. From there, students are taught how to create a constructor for their new class, what the “self” parameter is and how it works, and how to set and define variables within the constructor using the self parameter. Afterwards, students are introduced to class-specific functions (both ones that do and do not require a reference) and how to create them. Finally, Chapter 9 closes by explaining how to make a class be printed as a proper representation of its data through the definition of the __str__() function.

F. Block Coding

Despite the lack of coding classes or teaching within schools, many students have been exposed to coding principles through game-focused learning sites such as Scratch or Code.org. With sites such as these, students are able to use drag-and-drop blocks to direct a character to do a certain task. Block coding is a visual approach to coding lessons, focused on teaching students how logic structures translate to action. While block coding lessons teach students computational thinking skills, they do not translate to professional coding development that students are able to use in the workforce. Many students will begin the transition to text-based coding languages in high school or in college classes. The project sponsor expressed a need in schools regarding students who struggle in text-based coding classes, even after exposure to coding principles through block code. Students struggled to make the connection between the visual coding to action experience and the syntax and nuances of Python or Java.

Initial discussions and the proposal from the sponsor centered around a transitionary teaching model from block coding to text coding in Python. The plan was to design a block coding interface similar to Scratch or Code.org and slowly take away block coding aspects in favor of text. The block coloring would emulate the text highlighting of Python in VSCode, creating visual similarities between the block and text. The first step of this process would be to slowly expand the terminology within the blocks to reflect Python keywords. Then, while still employing the block interface, inside the blocks, Python syntax would be integrated to show students how block coding translates to Python. As students begin to understand the logical structure of Python, the blocks will slowly become transparent, requiring students to type in text instead of using drag and drop blocks. By the final stages of the curriculum, students will be fully coding in text, and the final block coding support of a colored box outline over the different structures of code will disappear.

This model of the project centered around teaching students Python, with lessons that went in-depth on basic coding principles, each with multiple activities to showcase the nuances of code. However, as the sponsor is a representative of the Florida Space Institute, a need was established to integrate the FSI RE-RASSOR program.

FSI's current educational outreach program includes the 'Build Your Own Mars Rover', where an open-source repository is provided for schools to 3D print and build their own rovers. The project sponsor, Mike Conroy, expressed interest in building a part two to this educational program where students are able to code the rover built in their or other classrooms. This senior design project's teacher contact was part of this initial program and built a rover named Courage in her 5th grade classroom. Due to her close connection, understanding, and interest in FSI's educational outreach programs she has offered her classroom as the tester for the coding lesson plans.

With the new motivation to include the rover program and RE-RASSOR software, the extension shifted from a transitional model to one focused on teaching students how to code rovers. Further discussion with the teacher contact Ms. Robinson revealed that while some students have block coding experience, many students have not been exposed to coding to any significant degree. The after-school program would be their first experience learning to code. As such, the team agreed beginning with block coding will increase the mental load on students and shift the focus away from the rovers. Instead the program is designed to teach students without coding experience the basic principles in Python so that they are able to code their rover.

For students with block coding experience, the lesson plans still work to bridge this gap with a section titled ‘Block Coding Equivalent’. Block Coding Equivalent sections will primarily be present in Unit One, and provide diagrams of code structure with more colloquial terms. The goal with these sections is to provide a relatable frame of reference for students who have used block coding sites. Instead of providing a visual character-based run time result, the block code will be a visual aid to the output provided by the example Python code. To emphasize the connection between block coding and text code, the blocks will reflect the text highlighting for Python in VSCode and be side-by-side with the text equivalent. Arrows will be drawn from the text sections to the blocks in the diagram.

A diagram based approach to discussing block coding is the current solution to improving students’ computer science skills while utilizing the incoming experience and background of students. There is room for extension here with possibilities of set drag and drop blocks in a fill-in-the-blank model. This current senior design project focuses on the planning and execution of a rover-based after-school program so further implementation of block coding will not be emphasized here. However, future integration of a transitional block coding experience will improve the accessibility of the program and highlight the take-home and practice availability of the extension.

G. The Teacher Manual

The team recognizes that teachers may not be experts in computer science or robotics, so to assist, we'll be building a teacher manual and providing example Python files. The Manual itself will be a PDF and will be bundled together into a zip file with all the example Python files. This manual features a guide to operating RE-RASSOR rovers, a list of available tools, a guide to adding or removing courses, as well as all the lessons with annotations and advice.

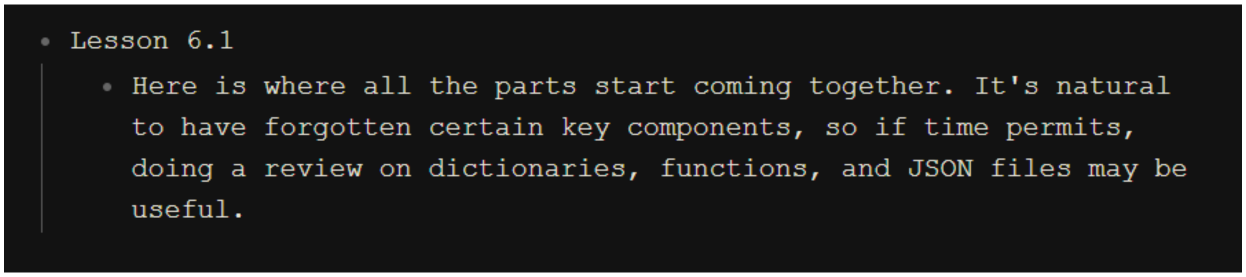
The guide to operating RE-RASSOR rovers will include links to FSI's documentation and as well as a guide to using their rover controller, but it will give a basic rundown of how the rover operates in plain language while also introducing and explaining some terminology. The goal here is to get the teachers acquainted with the parts of the RE-RASSOR rovers they'll be operating with, so we'll be mostly focusing on the post requests, the rover's vision, and its physical limitations; however, the team will have a section for reimaging the Rover's memory because it's the easiest fix to most software bugs.

The list of available tools will not only list out what is available to the teachers, but also give a rundown of how to use them. This will include the simulation, interfacing tools, and progress tracking tools. It'll be a comprehensive guide and listing of all the tools they may need to run the school program.

The guide to adding or removing courses will also serve as the installation guide. There will be a walk-through of how to install the extension, how to add or remove a course from the extension, and even how to make personal courses. The team hopes this will allow for future project groups and teachers to continue to add and create content for the extension even after we're gone.

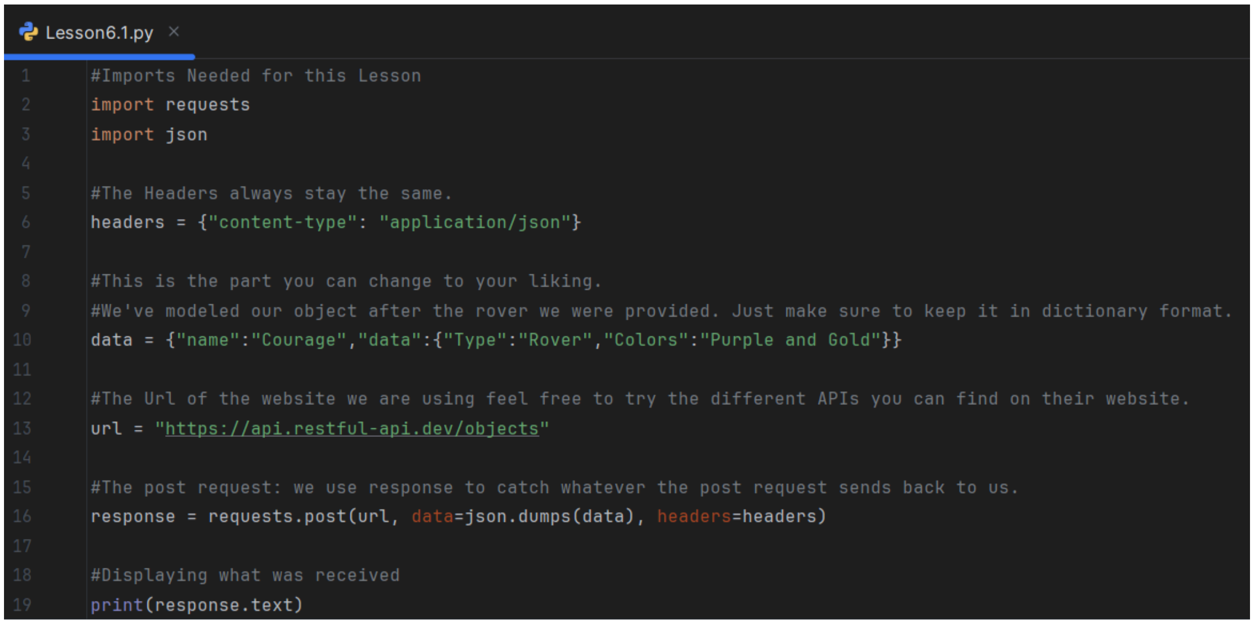
Steam Education

The main meat and potatoes will be the annotated lesson plan. It’ll be very similar to the lesson plan, but with annotations and advice that may help the teachers in teaching the content. This, along with Python files, should provide teachers with the information they need to teach the content. Here is an example Python script and annotation from the POST request lesson.



- Lesson 6.1
  - Here is where all the parts start coming together. It's natural to have forgotten certain key components, so if time permits, doing a review on dictionaries, functions, and JSON files may be useful.

(9): Example Teacher Manual Annotation



```python
#Imports Needed for this Lesson
import requests
import json

#The Headers always stay the same.
headers = {"content-type": "application/json"}

#This is the part you can change to your liking.
#We've modeled our object after the rover we were provided. Just make sure to keep it in dictionary format.
data = {"name":"Courage","data":{"Type":"Rover","Colors":"Purple and Gold"}}

#The Url of the website we are using feel free to try the different APIs you can find on their website.
url = "https://api.restful-api.dev/objects"

#The post request: we use response to catch whatever the post request sends back to us.
response = requests.post(url, data=json.dumps(data), headers=headers)

#Displaying what was received
print(response.text)
```

(10): Example Python File for Teachers

As you can see, the annotations are brief, and the Python files are also fully annotated to explain what each part of the code does, so teachers can help students debug by looking at the example. Although most of the lessons are already in the extension, having this in the Manual essentially turns it into a printable version of the lesson plan.

With all this, the team believes that the teacher manual will provide teachers with the guidance they need for teaching and running this after-school program.

H. The Practice Run - Pilot Program

Thanks to Ms. Robinson of Discovery Key Middle School, the team has been given the opportunity to test the lesson plan at Discovery Key Middle School. This pilot program of lesson plans and extension will serve as proof of concept and will hopefully unravel any key discrepancies in the program. Thanks to the project sponsor, we've been given access to a rover named Courage, which will be used for the final lessons of the course and the final competition. The pilot program will run as a 9-week course, with one hour sessions every week during the typical 9-week quarter. The current plans is for the team to facilitate the program and gather data every Tuesday from 4:00 PM - 5:00 PM, immediately after the school day is over in Ms. Robinson's classroom.

Ms. Robinson will distribute an interest form at the beginning of the Spring semester to gauge student interest in the program. Depending on the feedback received, the number of students will be decided. Currently, Ms. Robinson has a list of 8-12 students who she believes will be a good fit for the program and who have expressed verbal interest. Participating students will be from Ms. Robinson's 8th grade engineering classes. Many of these students work on computational thinking skills in the classroom every day so by running the program with these students, the goal is to gauge how well new information is processed and if there are any clear areas in need of improvement. To be able to work with students, the team has completed the volunteer application for Discovery Middle School and been approved.

The lesson plans are built to be self-paced. Students will read through the material and complete the activities in each lesson. The goal is for students to have completed one chapter every session. Understandably, the initial excitement of the program and general student behavior will determine the feasibility of the timing and structure of the program. Any material not completed during the session, can become take home activities or be the starting point for that student the next session. The initial chapters are lengthier and cover more principles in a shorter amount of time, while later chapters focus more on implementation and application to the rover. Due to these differences, it is expected that students who need more time on initial chapters will be able to catch up before the competition.

As students work through the examples and activities in the extension, the team will be circulating, answering any questions, and noting areas of contention in terms of lessons and the visual user experience. Any activities students struggle with or reach a blocking point on, will be noted. In this case, the struggling students will be assisted through the activity to determine if there is a specific explanation or approach that works best to improve their understanding. Based on this feedback, edits to the lesson plans will be made before the next session.

For a more formal evaluation of the lesson plans and extension experience, students will be given a quick questionnaire at the end of each session with four questions. Question one would be their favorite part of the lesson, question two their least favorite part of the lesson, question three their favorite part of using the extension, question four their least favorite part of using the extension, with a fifth optional comments/questions/suggestions section. Question three and four will be removed or demeaned optional after the first few sessions as the visual structure of the extension itself should not change drastically.

The public release of the program relies on the teacher or program coordinator's ability to run the program individually, answering student questions, and dealing with any technical difficulties. To determine what these needs might be and what details to include in a teacher manual for the program, each after-school club session will be followed with a debriefing with Ms. Robinson. This will be a quick discussion on what Ms. Robinson deems as progressing well, what attributes of the overall program need improvement, and the resources she would need to run the program individually.

Using the information gained from the pilot program from both the student and teacher level, the team will work to make the Visual Studio Code extension more intuitive, the lesson plan more sound, and correct any flaws in the teaching approach. Furthermore, the team will also note what parts of the lessons capture the students' attention, taking notes on the parts that interest the students more. Depending on this, and how well students worked with the time constraint, lower-priority sections or those that don't interest the students and are not vital to rover manipulation will be redone or removed, and any higher-priority, yet uninteresting sections will be edited to be more palatable.

This will continue until students reach the challenge chapter. Here students will attempt the challenges using Courage to gauge their comfort and understanding of basic rover manipulation and requests. Then, the final challenge will be revealed before the last week of the program to provide students with time to brainstorm and discuss ideas. The final challenge will be run during the last week. Issues or weak points specific to the final challenge will be noted as to implement optimizations for future iterations of the challenge week and the after-school program.

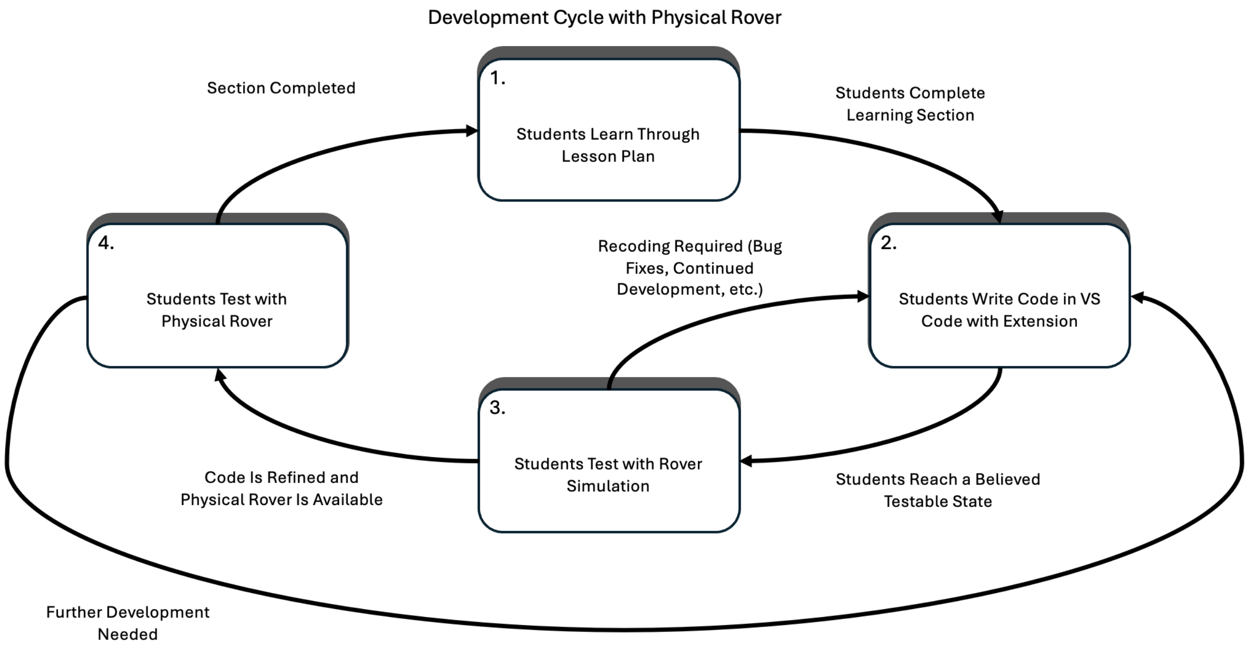
STRETCH GOAL: THE SIMULATION

A.Introduction

The simulation is a 3D, Godot-based system that allows students to visualize the code they have developed without the need for a physical rover. Students can use the simulation during the development of their programming skills to visualize the behavior of the rover based on the commands they are inputting. Without the simulation, the students would likely be limited to one or a small number of rovers to use for testing, which would bottleneck the progress being made by the group. With the simulation, the development process can be highly interactive and lead to increased engagement.

Another effect of having the simulation is that it allows schools that cannot provide a rover to be able to work through the program as well. It is incredibly important that this program is accessible to as many young students as possible to learn how to program using cutting-edge technology. The rover could limit that outreach due to many factors: material cost, 3D printing technology accessibility, and the general time to create and maintain a physical rover. This alternative looks to not only allow students more testing time with their code. It is also designed to allow students who would otherwise not be able to complete this program the ability to work through the same exact lessons.

When utilizing the simulation, the lesson plans will be the same as if the students were to use the physical rover. This is not designed to be an easier testing system. It is meant to be supplemental to the course or, as needed, a replacement for using a physical rover for testing. Another major benefit of having the 1-1 simulation compared to the physical rover is that it allows students to work through their bugs or issues in a much lower-stakes environment. While the physical rover is incredibly safe to work with, working in a simulation removes even the smallest risks.


Development Cycle with Physical Rover
1.
Students Learn Through Lesson Plan
Students Complete Learning Section
2.
Students Write Code in VS Code with Extension
Students Reach a Believed Testable State
3.
Students Test with Rover Simulation
Recoding Required (Bug Fixes, Continued Development, etc.)
Code Is Refined and Physical Rover Is Available
4.
Students Test with Physical Rover
Section Completed
Further Development Needed

(11): A diagram showing how students will learn as they use the extension.

B. Technical Requirements

There are many important benchmarks that the simulation must meet. It must be able to function as a viable replacement for having a physical rover. Therefore, it needs to be a 3D simulation with the ability to reset and modify the code being run as needed. If the simulation does not allow for easy modifications, the benefit of saving time for testing could be lost. There must also be accurate and comparable physics as compared to the physical rover. Rover movement, such as driving and turning, must be an identical match to the movement otherwise. If they do not match up, half of the purpose of the simulation (to allow earlier testing without needing access to the physical rover) is lost.

Additionally, handling collisions for obstacles that are placed must be the same as how it would be handled otherwise. Different objects have different weights, shapes, sizes, etc. These must match nearly 1-1 in the simulation compared to in real life. If the rover is programmed to push an object, that object in the simulation must react the same as how the object would normally. Otherwise, completion cases that would not pass for the physical rover could end up working in the simulation, or vice versa.

Terrain handling is another important technical requirement to consider. The floor that the rover drives on contains multiple characteristics to consider when making a simulation. For example, a rover driving on carpet will have plenty of traction. However, if that same rover were to drive on a slick ramp, it would likely have much more trouble navigating the terrain. Being able to implement these challenges is crucial for giving students accurate feedback on their programs, and failing to implement these distinct problems will almost certainly hurt the effectiveness of the simulation.

Lastly, the simulation needs to be able to simulate the ArUco marker reading capabilities of the physical rovers, such as Courage. This task is intended to allow the physical rovers to read specific markers that are placed around the intended space and have the rover react according to what it reads (based on the programming of the students). Testing a system that makes a decision as the rover is running is incredibly important, so ensuring this system is effective and matches how it would work during regular rover operation is crucial.

Steam Education

C. The Translator

In order for the simulation to act as previously stated, there must be a translation system to allow the exact same code to run correctly on two fundamentally different systems. This would come as a custom Python library that would handle the different needs without student input (beyond pushing a “Run in Simulation” button). Therefore, for any simple command that would normally be passed through the standard process, there must be a system that takes that command and creates a JSON message to be handled by the simulation. This backend system would never need to be seen by the students and will keep the process of learning how to program a rover using a simulation as simple as possible.

D. Feedback

Another way that the simulation will assist students with learning how to program is by providing detailed, helpful feedback. Students are going to want to know why their code is not doing what they want it to do. A good way to reduce frustration and help the teacher who needs to keep up with a large number of students is to give descriptive error messages. This would be handled in many different ways. One of the biggest ways is by implementing fail cases.

When a student is tasked with moving through a maze of objects without hitting any of them, it will be important to actually tell them when they have failed. This helps keep the students honest and also allows quicker feedback to be given on the program they have worked on. When a student is prompted to make the rover move a specific distance in a certain direction, the code needs to first execute and be allowed to run its course. Then, the student should be informed if their rover in the simulation did not complete the provided task. On top of feedback for fail cases, it is also important to give the students detailed information about what their rover did.

The simulation is designed to mimic a real-world rover and provide quick feedback for the students to learn from. However, if the simulation does not give the students all of the information they need to learn from their mistakes, it fails at the second goal. It is then important to give an overview of the important data about the rover's crucial metrics (position, speed, etc.). It is also crucial to consider that part of the lesson the students need to learn is how to analyze how the rover responds to their input visually. If the data provided to the students is too detailed, it can remove a crucial skill that should be instilled in the young developers. There is a middle ground between giving too little data, leaving the students unprepared to continue their development, and too much data, where the students do not even need to look at what the rover is doing and can instead just analyze the data alone.

E. Level Design

For the physical rover testing, there would normally be specific scenarios made in the general working area. The students would then ensure the rover is in the starting position, and would run their program to test if the rover is performing the correct actions. When it comes to the simulation, the levels can be made in advance to the benefit of the students. This allows for consistent testing between students and ensures that, if there is an issue with the placement of the objects, it is likely that the students and teacher will be able to tell. Some of the levels would include:

- The Introductory Level
    - Students are given a simple task: make the rover move in a specific direction and a set distance, resting it on a set target point. The level itself is empty.
- Turning
    - Students would build upon the previous level and add turning to their program, with the target point being placed forward and to the right of the start. The level itself remains empty.
- Terrain: Inaccessible
    - Students need to work to navigate around a terrain that is being deemed inaccessible (a hole, for example). The target point will be placed on the opposite side of this inaccessible terrain, forcing the rover to navigate around the inaccessible terrain to reach the target point.
- Terrain: Friction
    - Students will learn how the specific terrains have different textures, as well as how these textures affect the rover at different slopes. Students will also be made to turn the rover on lower friction surfaces, observing how the rover slips far more on a surface with less friction.
- Obstacles: Avoidance
    - Students must avoid the obstacle placed on the terrain. If the students hit the obstacle, the level is failed, and they must return to their program. There will also be a specific location that the students must reach, but flexibility will be allowed in how exactly they reach it, with multiple viable routes to the target point being present.

- Obstacles: Moving
    - Students must push an obstacle (different from the obstacle used for avoidance) to a specific location. If they do not push the obstacle to the correct place, the level is failed. This is a crucial component of the physical rover, and the level must be designed carefully. Since real-world, physical objects have different weights, textures, and strengths, the simulation must use the physics of an object that is easily accessible to a school and will not break down after use.
- ArUco Markers
    - Students will need to navigate to an ArUco marker that they will need to translate into a specific action. This will test the students' capabilities to create a program that is able to think on the fly, navigating based on the placement of each ArUco marker.
- The Challenge
    - This will match the design of the final challenge . It will contain all of the previous skills that they have had to learn in order to test their total retention from the course. This will mainly serve as a practice run for the real-world final challenge, allowing students to bounce around ideas and try different approaches in preparation for the actual challenge.

IX. DOCUMENTATION

A.Research

RE-RASSOR rovers, including the one the team was provided with, named Courage are mainly controlled by sending HTML POST requests containing JSON files. As of right now, there are six JSON objects that can be sent either individually or simultaneously in the same JSON POST request:

- {wheel_action: {linear_x: float, angular_z: float}} - Moves the rover's wheels. linear_x controls forward and backward, with -1.0 being the fastest backward speed and 1.0 being the fastest forward speed. angular_z controls the rover's turning, with 1.0 being the maximum leftward turn and -1.0 being the maximum rightward turn.
- {target_coordinate: {x: int, y: int}} - Sends the rover to automatically navigate to the specified coordinates. How the coordinates are defined seems to be unclear, and this command is not used in the existing frontend.

These other actions are either irrelevant to Courage or are not fully implemented.

- {shoulder_action: {linear_y: float, angular_y: float, linear_x: float, angular_z: float}} - Moves the rover's shoulder to a specific position and angle. While this is a part of the RE-RASSOR rovers' valid commands, Courage does not possess a shoulder, and thus this command is useless for Courage.
- {routine_action: bytes} - Causes the rover to perform some pre-defined action. A few routine actions are described in code, but none of them have been implemented as of the writing of this document.

- {front_arm_action: float} and {back_arm_action: float} - Raises (1.0), lowers (-1.0) or stops an arm. These commands are valid, but are not fully implemented.
- {front_drum_action: float} and {back_drum_action: float} - Digs with (1.0), dumps out (-1.0) or stops a drum. These commands are valid, but are not fully implemented.

Additionally, there seem to be a set of commands related to operating a version of the RE-RASSOR rover that has a paver. However, this is only implemented on the frontend side, with no known backend endpoint that receives them. As such, these will not be documented.

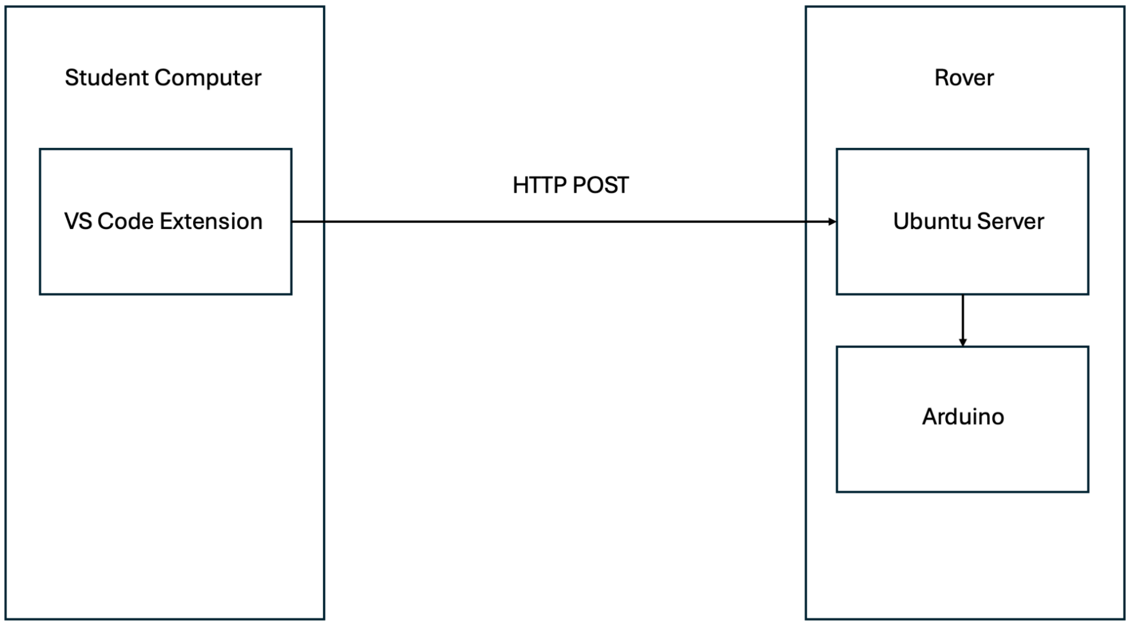
B. Design Summary

The project goal is to create a software solution to teach students how to code in an after-school club setting. The software designed is a VSCode extension meant to emulate a professional development environment, with a .NET backend to improve accessibility and save progress. Lesson plans are integrated into the extension as files. Each lesson contains examples and activities, where activities include competition flags that mark a lesson as complete and allow a student to progress forward in the program. Primary users are students and teachers, each with different accessibility points and power controls.

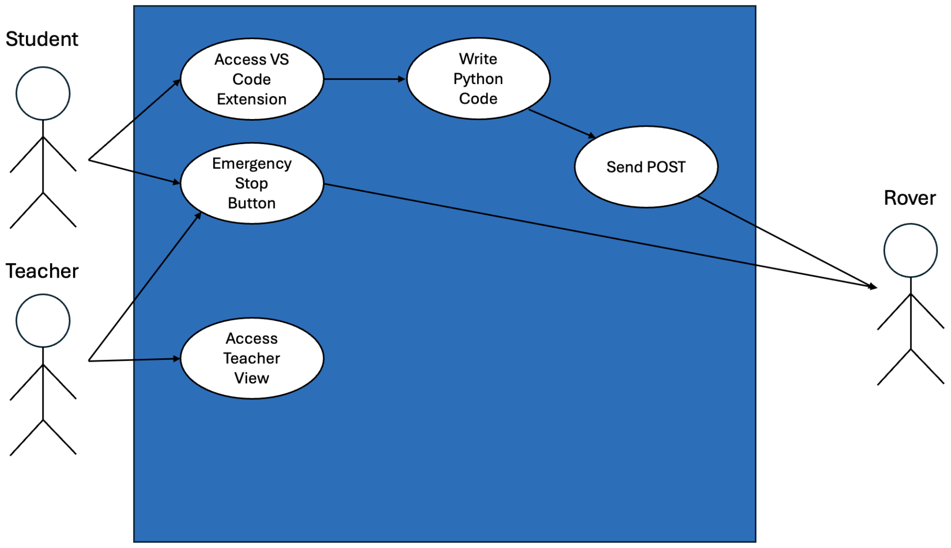
This program uses the RE-RASSOR software available through the Florida Space Institute, and the rover controls completed by previous senior design teams. Students will be testing their knowledge of code on rovers, as such the design of the program includes integration with existing FSI software and the hardware components of Courage the Rover.

Steam Education

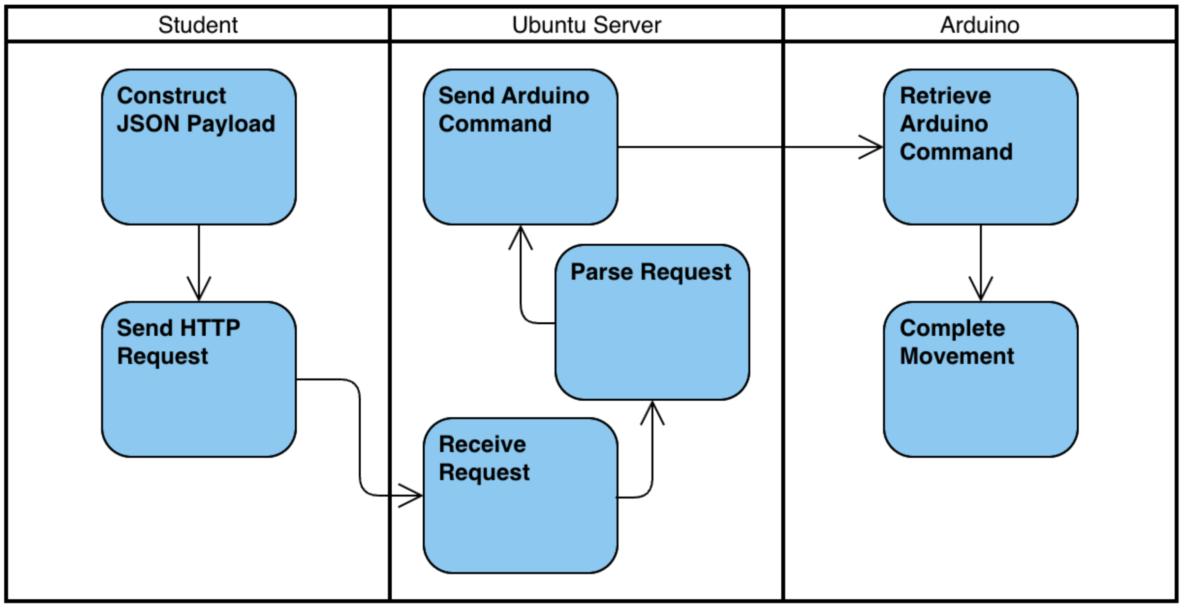
C. Design Diagrams



(12): A block diagram showing how communication works between the extension and the rover.

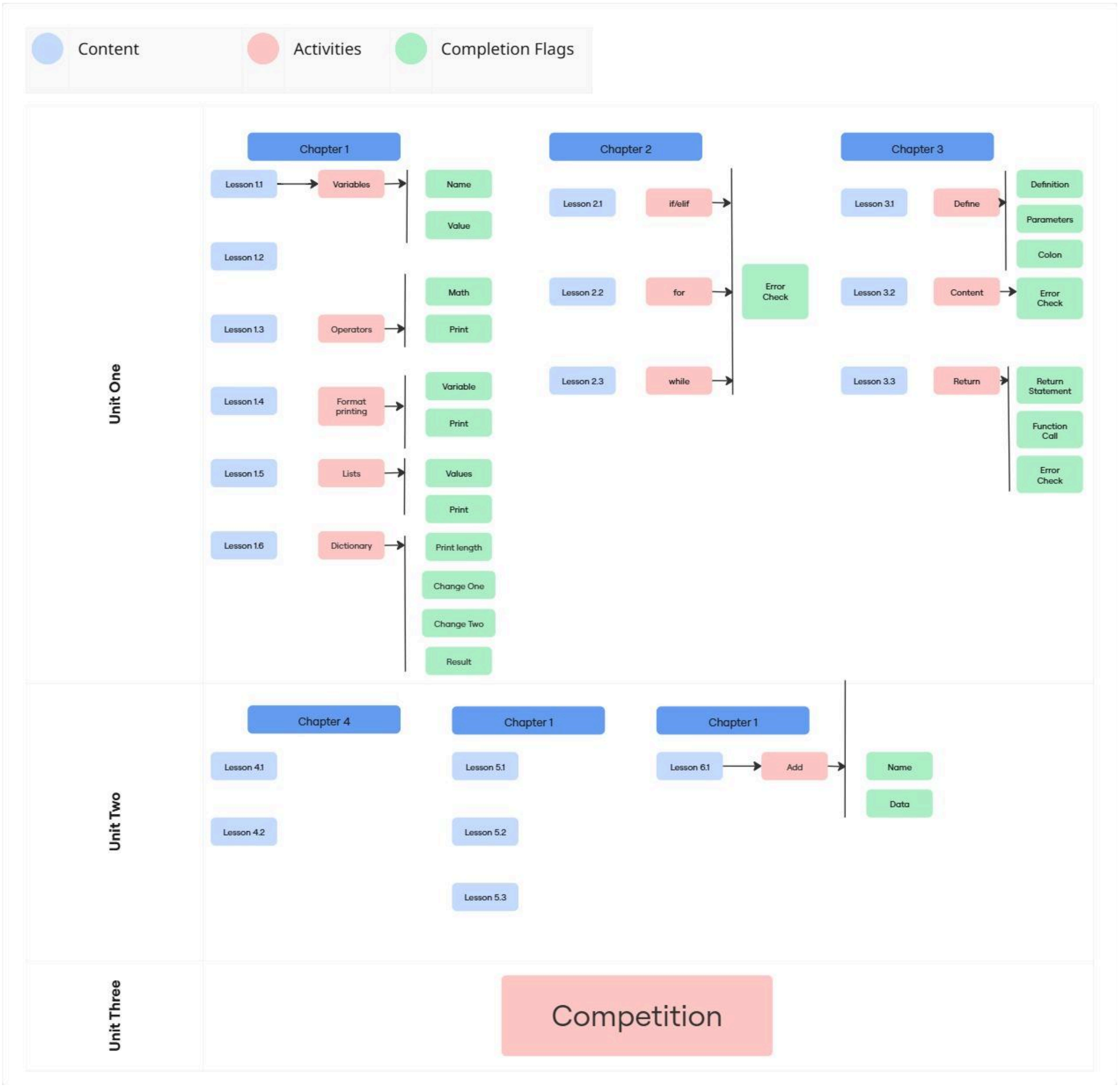


(13): A use-case diagram showing how communication will work between the student, teacher and rover.



(14): An activity diagram showing how JSON payloads are constructed, sent to the rover, interpreted and acted upon.

Steam Education

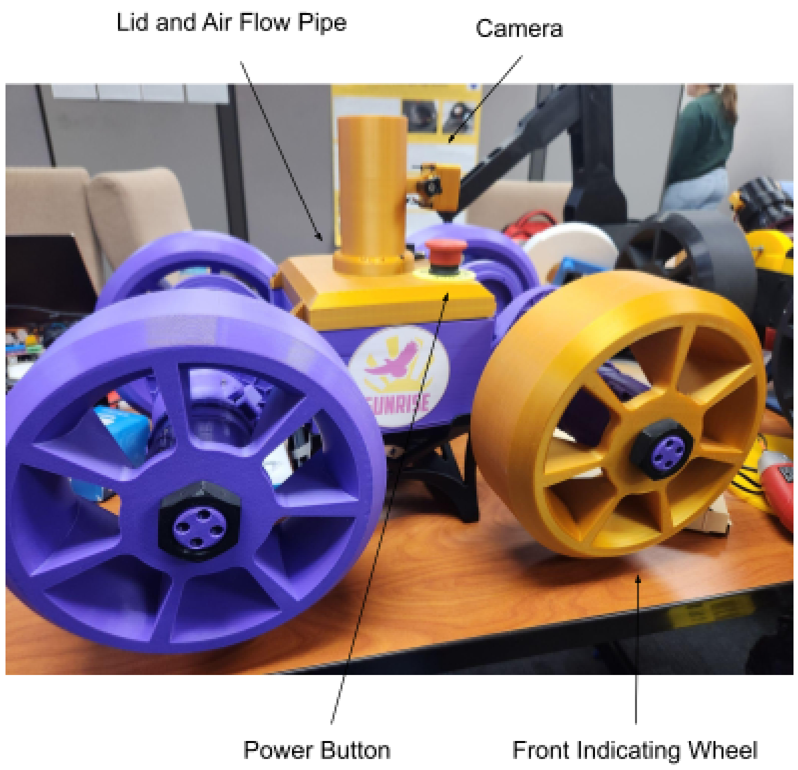


(15): A lesson planning flowchart outlining activities and completion flags within the extension.

X. RE-RASSOR Rovers

Because the school program is about rovers, FSI has provided the team with a rover. His name is Courage and was originally 3D printed by Sunrise Elementary. Rovers come with 2 colors usually, one being the base color (in Courage's case that's Purple) and another is used for the lid and front wheel (for Courage that's gold). The 2nd color is used to indicate the forward-facing direction of the rover. The lid has a pipe to act as both a way to pick up the rover, but also help with air flow for cooling.

The RE-RASSOR rovers run on a single-board computer in similar structure to a Raspberry PI. Courage specifically uses Libre Computer's AML-S905X-CC (which the contacts at the FSI lab call The Potato). The Potato features a Quad Core ARM® Cortex-A53 CPU, Penti-Core ARM® Mali-450 GPU, 4K AVE10 VPU, and 2 GB RAM. The Potato runs vision logic and an Ubuntu server with a Python Flask proxy to receive the POST requests. It then tells an Arduino which arduino code to run based on those post requests. While the students could theoretically be taught to program the Potato, this approach would be limited by the short time the after-school program runs for, would likely take far longer for the students to see results, and could be hazardous to Courage should the Potato be programmed incorrectly. Because of this, the students will not be taught to program for the Potato, with the focus being entirely on deciding what HTTP requests to send to Courage externally through Python.



(16): Courage and a few key parts

RE-RASSOR rovers also have a camera, which currently identifies ArUco Markers and their value. ArUco markers are 2D binary-encoded visual markers that look a lot like smaller QR codes. The camera and Aruco markers will be used as the primary focus of the rover lessons, as it will allow students to use them as triggers for certain behaviours. Unfortunately, the current way the ArUco marker detection is set up just ports the value directly to the video feed for the controller app. To make it useful, we'll need to develop a new API that can be used to return this value.

The new ArUco marker API will be added to the Python Flask proxy. It will be a fairly simple POST request that doesn't require much to be sent in JSON, but returns a JSON with the value of the visible ArUco marker. The actual difficult part will be to get the value over to the API as the ArUco detection occurs in separate python file and directory. Despite the lack of documentation, the team is confident in discovering a way to obtain the ArUco marker value for use.

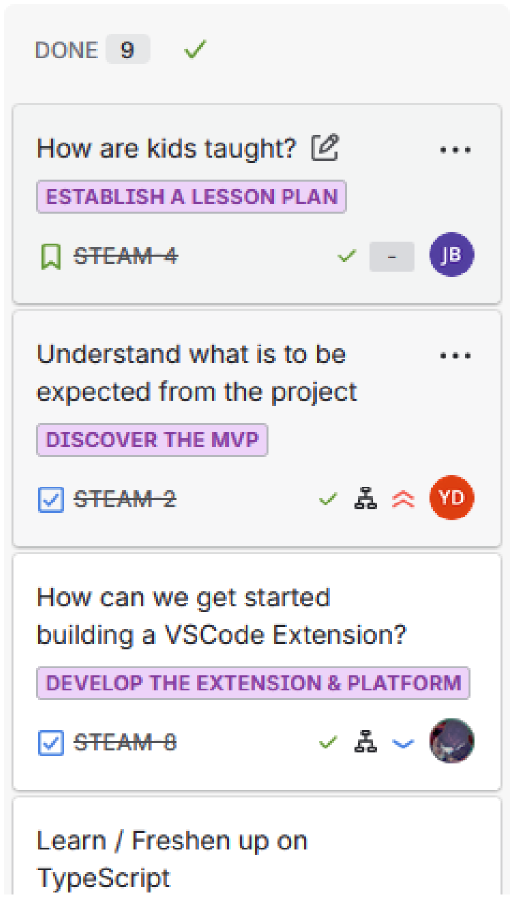
For this project, the team was provided with a controller app that not only simplifies the process of connecting with the rover, but also provides a GUI that acts like a remote control and allows you to see the video stream. Pressing forward or backward causes the rover to begin moving in that direction, and pressing it again causes it to speed up. You can only speed the rover up up to 3 times, and the speed-ups aren't very drastic. A person can easily outwalk the rover even at its max speed, so it'll be safe for the children to use it even at max speed.

For demonstration purposes, the controller is great, but beyond that, it won't be used much because students will be tasked with coding the robot through challenges, not controlling it by hand. However, analyzing the code did allow us to gain a handle on Courage's valid HTTP commands much faster. Furthermore, the code that simplifies connecting with the rover could be adapted for use by the Visual Studio Code extension, should the team get permission to use that code. Otherwise, extra time will be spent on creating a personal solution, as instructing the teachers on manually finding the IP address of the RE-RASSOR rover would be a potentially error-prone and lengthy process. Another way to achieve the same effect is by using the rover domain names to access the rover. Each Rover has a domain name (Courage's is ubuntu@Courage), so this may be able to be used as an advantage, as it's much easier for teachers to remember a domain name than an IP address.

When connecting to rovers over the school's Wifi, the school usually tags the post and video streams a security threat and begins blocking them. To avoid this, the current solution is to use an internetless router to connect from device to rover. This might be unavoidable, but if there is a possibility of eliminating this factor the connection process to the rovers will be made simpler and cut the requirement of an extra router. A potential way to do this is to have the school's IT department to manually whitelist the rover to prevent it from being blocked, though finding a more elegant solution to prevent the rover from being considered dangerous would further decrease the amount of setup the rovers have to undergo, allowing them to be deployed far faster and with less hassle.

XI. TIMELINE

Originally, for the project the team was using a Jira board with Scrum in order to get the project rolling before direct contact with the sponsor could be made due to a family emergency on his end. Most of the progress made while working with Jira was focussed on external factors, relearning or brushing up on programming languages that will be needed in order to work on the project, researching specific topics that were known would be related to program goals, and meta goals such as reaching the sponsor for a meeting.


DONE 9 ✓

How are kids taught?
ESTABLISH A LESSON PLAN
~~STEAM-4~~ ✓ - JB

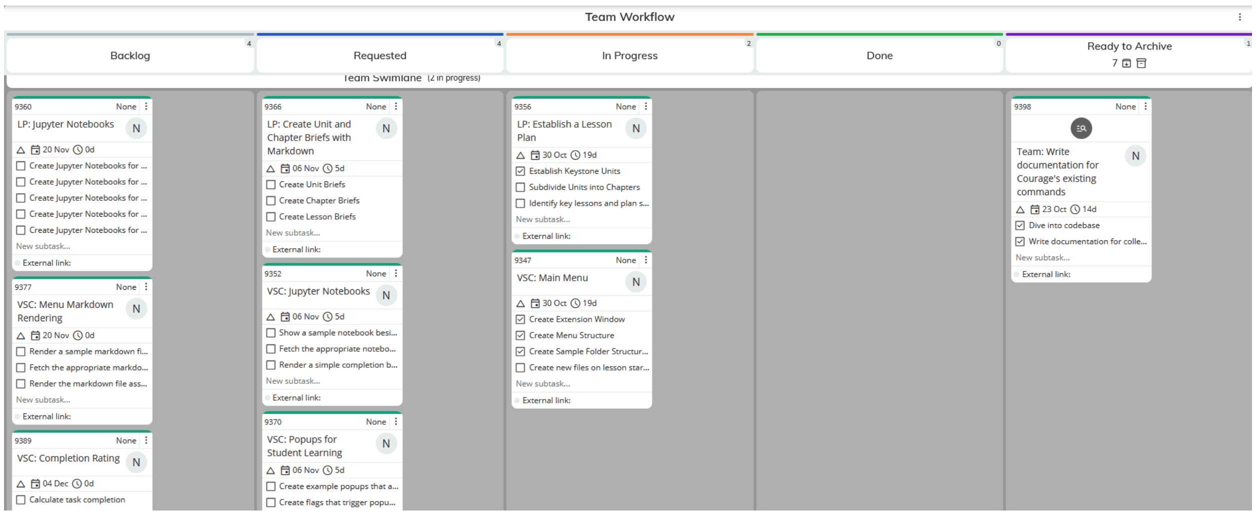
Understand what is to be expected from the project
DISCOVER THE MVP
~~STEAM-2~~ ✓ YD

How can we get started building a VSCode Extension?
DEVELOP THE EXTENSION & PLATFORM
~~STEAM-8~~ ✓

Learn / Freshen up on TypeScript

(17): An image of the Jira board the project used.

After meeting with Mr. Conroy, the team switched over to a kanban board run on a private kanbanize session hosted by FSI. The switch to kanban has been relatively simple, with only some small organizational hiccups creating discrepancies that have since been resolved.


Team Workflow

Backlog 4 | Requested 4 | In Progress 2 | Done 0 | Ready to Archive 7 1

Team Swimlane (2 in progress)

Backlog

9360 None
LP: Jupyter Notebooks N
20 Nov 0d
- [ ] Create Jupyter Notebooks for ...
- [ ] Create Jupyter Notebooks for ...
- [ ] Create Jupyter Notebooks for ...
- [ ] Create Jupyter Notebooks for ...
- [ ] Create Jupyter Notebooks for ...

New subtask...
External link:

9377 None
VSC: Menu Markdown Rendering N
20 Nov 0d
- [ ] Render a sample markdown fi...
- [ ] Fetch the appropriate markdo...
- [ ] Render the markdown file ass...

New subtask...
External link:

9389 None
VSC: Completion Rating N
04 Dec 0d
- [ ] Calculate task completion

Requested

9366 None
LP: Create Unit and Chapter Briefs with Markdown N
06 Nov 5d
- [ ] Create Unit Briefs
- [ ] Create Chapter Briefs
- [ ] Create Lesson Briefs

New subtask...
External link:

9352 None
VSC: Jupyter Notebooks N
06 Nov 5d
- [ ] Show a sample notebook besi...
- [ ] Fetch the appropriate notebo...
- [ ] Render a simple completion b...

New subtask...
External link:

9370 None
VSC: Popups for Student Learning N
06 Nov 5d
- [ ] Create example popups that a...
- [ ] Create flags that trigger popu...

In Progress

9356 None
LP: Establish a Lesson Plan N
30 Oct 19d
- [x] Establish Keystone Units
- [ ] Subdivide Units into Chapters
- [ ] Identify key lessons and plan s...

New subtask...
External link:

9347 None
VSC: Main Menu N
30 Oct 19d
- [x] Create Extension Window
- [x] Create Menu Structure
- [x] Create Sample Folder Structur...
- [ ] Create new files on lesson star...

New subtask...
External link:

Done

Ready to Archive

9398 None
Team: Write documentation for Courage's existing commands N
23 Oct 14d
- [x] Dive into codebase
- [x] Write documentation for colle...

New subtask...
External link:

(18): An image of the Kanban board the project currently uses.

The backlog for the entire rest of the semester is up to date on the kanban board, with the remaining time for Senior Design 2 being dedicated towards iteration and improvement, with what is essentially an open beta test of the program being run at a local middle school campus. The team has gotten involved with a teacher at Discovery Middle School, Ms. Kristina Robinson, who has worked with students on FSI projects in the past, in order to organize an after-school program that will put the program in direct contact with the end users for any observation and improvement needs.

A.Timeline

- Sept 25th through October 8th

- Primarily working on brushing up on necessary skills and identifying key materials for the project ahead
- Programming Languages
    - Typescript is the primary language necessary to create Visual Studio Code extensions, the focal point of the project. Some of us had previously touched on Typescript, but the majority had never used it. Thankfully, by this point, every one of us has touched on JavaScript, and that leads very well into TypeScript
    - .NET/C# is a language that the team selected for backend tools and teacher resources. Typescript easily integrates with these features, and VSCode natively supports C# and .NET tools.
    - Python is, of course, the primary language that the team will be teaching, so each one of us needs to understand it on a level at which the team can then turn around and teach the concepts well.
- Research
    - Much of this work period was dedicated to research, including the capabilities of a VSCode extension, Lesson structure, and topics that the team should cover over the course of the lesson plan, and how the team should go about teaching those lessons in a comprehensive, compelling way.

● October 9th through October 23rd

- Extension basics and File Structure
    - Over the course of this section, the team built out the foundational elements of thebird's-eyenu, providing a bird's eye view of the course structure. In the process, the team

created a sample folder structure that would be used to tweak the rendering of the menu and dropdowns for the lessons.

- Building the rover Courage for demonstrations and teaching purposes.
    - The end goal is to get the students to program for the rover at some capacity, so Courage will be essential for demonstrations. The FSI Team had the components of Courage, the RE-RASSOR Rover, ready and waiting to be taken apart and discarded, but when the team was finally able to make contact and reach out, the team was able to secure Courage for the project's use.
- Documenting HTTP requests
    - The team acquired some technical debt because the HTTP requests designed to operate Courage are not documented. These HTTP pathways are required components in order to control the rover, so the team went through the process of writing new documentation to control Courage. This will be essential to the lesson design process for the later lessons, in which the students will have direct access to the rover in the classroom.
- Building the lesson plan overview.
    - During this period, the team worked through the main lesson plan, establishing the key units and chapters, and began to work through the writings for the early lessons.

● October 24th through November 5th

- Turning all the lessons into markdown files for the extension.

        - The lessons are being written out in Obsidian, which thankfully uses markdown natively; however, the team will need to transfer those files over to the primary file structure of the extension.
    - Integrate Jupyter Notebooks into the extension.
        - Jupyter notebooks will be the specific containers for the lessons as the students make progress through the chapters. They will contain runnable segments of code that demonstrate the concepts that the team is attempting to teach.
    - Creating pop-ups for task completion
        - Additionally, as student complete tasks in the lesson plan, popups will congratulate them on their progress and prompt an update to the progress bar displayed as part of the Jupyter notebook windows.
- November 6th through November 19th
    - Integrate the Markdown files into the Jupyter Notebooks
        - The markdown files and Jupyter notebooks should reference one another as the student makes their way through the lesson, with the markdown files showing completion just the same as the Jupyter notebooks. Key sections and areas of completion will be indicated with a static progress bar or sidebar highlighting completed areas.
    - Make the menus fetch the corresponding markdowns
        - When a student selects an element of the course, whether that is the unit, chapter, or lesson, the menu should pull up the associated markdown overview and display it to the left of the

tree overview. For lessons, this will be accompanied by a button to either start, continue, or reset the lesson progress.

- November 20th through December 4th
    - Make Completion Ratings
        - Lessons will contain a completion json that highlights key moments in the lesson progress that will prompt a popup, and change the progress bar. The team will have the framework for these events to occur at this point of the project, but the checkpoints.json files queue the events based on specific text being present in the students' text editor.
    - Document all progress made in Senior Design 1
        - This final step for the first half of senior design would be an extension of this document, explaining in detail exactly how far along the team is and how much the team has accomplished to this point. This is the Final design document as described by the syllabus.
- January 15th through January 28th
    - Start the practice run of the school program with Ms. Robinson.
        - This will enable us to test the effectiveness of the resources and tools. It'll also allow us to see if the team missed anything or included too much content. This task will take 9 weeks.
    - Brainstorm professor tools.
        - With feedback from professors and sponsors, we'll look to see what tools teachers may need to run the school program more efficiently. We'll rank them by importance.
- January 29th through February 11th

  - Continue the practice run
  - Develop high-priority professor tools
    - The tools the team marked as high priority will get worked on. This may rollover depending on how difficult these tools are.
- February 12th through February 25
  - Continue the practice run
  - Instructor Tools Completion
    - Ensure that any tools that the team started are completed.
- February 26th through March 11th
  - Continue the practice run
  - Final competition preparations
    - We’ll work on making sure everything is in order for the competition at the end of the practice run.
- March 12th through March 25th
  - Finish the practice run.
  - Evaluate how the practice run went
    - We’ll look over all notes and feedback to see if there is anything the team can improve upon or add within the last month.
- March 26th through April 8th
  - Make lesson plan adjustments as needed.
    - Based on the evaluation, the team will adjust the lesson plan and transfer those changes to the extension.
  - Work on the simulation
    - This would be a perfect time for the simulation if everything goes as planned.

Steam Education

- April 9th through April 22nd
    - Make final adjustments to the project and wrap up loose ends.
        - This includes everything from lesson plans to professor tools, as the team begins preparing the presentation soon. There will also be a crucial transition of handing this off to FSI for future use.
    - Complete simulation
        - Assuming there is time to work on the simulation, it will be completed during this time before the presentation.
- April 23rd through May 6th
    - Prepare Presentation
        - Gather all data findings and demonstrations into a presentation. This will include the extension, professor tools, and anything else made for the project.

XII. FUTURE POSSIBILITIES

A.Live Block Coding Visualization

Many students who partake in this program will already have some level of coding experience. While some may have even worked in text-based programming, many will likely have just worked in block coding. While it is believed that doing entire lessons in block coding would be more of a detriment than a help to the students (given the short 9-week timeframe), there is another way to help the students connect between block coding and text-based programming: the block code visualizer.

The block code visualizer would be a system that, as the student types, would show what the block coding equivalent would be while the student works, instead of simply displaying the full block coding representation at the end of the lesson itself. This tool would not only bridge the mental gap between coding using blocks and regular text-based programming, but also would seek to show students how much more powerful text-based programming is. Creating a text-to-block system is highly challenging, however, since text-based coding is highly versatile. Therefore, in order to keep the system lightweight, the lessons would likely need to be highly structured, and each section that would represent a block would likely be individual cells in the Jupyter notebook file that a student works in.

This system would be highly effective for students to visualize what their code actually represents, especially for those who have already worked in block coding software. For students who do not already know about block coding, it would still be helpful to have a visual representation of what they are programming for the earlier lessons.

B. Teacher Management System

There is almost always going to be a large group of students being taught by a single teacher. A management system that the teacher controls would be highly effective at assisting students on a larger scale. If a system could be in place that allowed a teacher to assist and maintain progress tracking at one location, students would be able to receive assistance at a faster pace. Therefore, a system that allows the teacher to track student progress, give permission to further lessons on an individual student basis, and additionally house all of the other teacher materials in one singular location would be highly beneficial.

The teacher management system would have many features, but likely the most important one is that it allows teachers to track the progress of students. Every student is different, especially at younger ages. Their starting points are likely to be different for those taking this course. Many will have almost no programming experience, while others will have worked in block coding or even some text-based programming. Therefore, it is incredibly important that a teacher is able to easily track the progress of the students at a glance. This is not only for the students who are falling behind, but also for those who are working ahead of the rest of the group.

For a topic as complex as programming, some students are going to have trouble grasping some of the more complex topics. This is expected and is why the lesson planning component of this project has been researched heavily. However, there is also a teacher who is actively with the students while this course is being taught. They are there to assist the students, but there could be more than one student who is struggling. Additionally, many times a student may feel nervous about saying they are falling behind. A system that calmly warns a teacher of a student who might be struggling would be an effective way of mitigating the lost time working through a problem such as this. It would check in on the progress bar of the students and would warn the teacher on their own computer if there has not been the expected amount of progress on a specific section. Then, the teacher would be able to check in and see if they could help.

The progress checking system would not only be useful for assisting students falling behind. It would also allow teachers to visually see that a student is moving at an accelerated pace. This would allow a teacher to potentially allow this student to continue past the lesson that is expected of the students that day, or provide a reward to encourage the quick work. A student working ahead of pace may also be able to reach bonus lessons that are outside of the expected scope of the 9-week program, as long as the teacher allows them to work ahead. Having systems that allow the teacher to see the students working at an accelerated pace, as well as allowing only these students to work ahead in the material, would be beneficial for those who are able to work faster than others.

A major goal of this program was to allow as many schools as possible to be able to effectively work through a course teaching young students how to program. Some schools may not have the means to have a teacher who focuses specifically on technology, and as such, the teacher may not have prior knowledge of programming or robotics to be able to effectively help the students on their own. This is why there is a prepared Teacher Manual to guide teachers in being able to help their students as needed. With the teacher management

Steam Education

system, a teacher would be able to have these guides pulled up automatically based on which student they want to assist. This would continue to allow students to work closer to their own pace without needing to wait or accelerate to meet the needs of the group. When a teacher needs to help a student with a section they do not understand, the information they need would automatically be pulled up based on which student they are helping. This component seeks to reduce potential frustration on the side of the teacher, especially if they are not well-versed in programming in Python, as well as on the side of the students, who are hoping to receive help as quickly as possible from their teacher.

Overall, the teacher management system could be a highly beneficial tool. Students are highly different from each other, and will want to be able to work at their own pace with assistance as needed. If the teacher has a system that is entirely designed around making their life easier, it is going to additionally make the learning process of the students better as well.

C. Lesson Changes

One of the major goals of this program was to ensure it was both easily editable and extensible. As a field, Computer Science is always changing and growing. It is likely that some of the things that are taught in this program, especially the higher-level topics, will either become incorrect or no longer best practice. Therefore, in the future, it may become necessary to review some of the material that is taught in this course. However, it is even more crucial that more lessons or even entire units can be added to the program.

As a 9-week after-school course, there is only so much material that can be covered. If more time were able to be had in the future, or if students wanted to be able to go home and continue to learn about programming, more lessons could be added to the end of the current program. These could be directly related to the material that has already been covered, such as more lessons or challenges with the Re-RASSOR rover. However, they could be entirely separate pieces of material from the main selection. The intent is that new programming topics, potentially in entirely new languages, could be taught using the already developed framework.

A common struggle for those getting into programming is the growing number of languages that are available to use. This course largely focuses on Python, but there are countless others that are potentially viable depending on the scenario. Therefore, if someone believes they have a good potential course that could be taught to students separate from the currently developed one, they could make an entirely new course using the system in place. This could allow more freedom on the part of the students in what topics they would like to learn about. One common approach for teaching students, especially young programmers who are beginning to learn through block coding, is to have them develop games. An entirely new course that teaches students how to develop a game using text-based programming could be developed. This is a highly interactive field in programming, and would likely be interesting to many young students looking to learn more about the development field.

Steam Education

XIII. TECHNICAL DEBT

When work on the project began, the team was warned very early on that the previous RE-RASSOR teams had never written any documentation for either the software of the rovers or the existing user-friendly controller app. As such, a priority of ours has been to write documentation as necessary to make the development of the extension and integrating the work with those of past teams much easier. As of right now, this has mainly manifested in the form of a list of valid commands that the rover's controller server will accept.

While it was known that the rovers communicated over HTTP, more details were not shared on the format the program expected the commands to be sent. In order to write this list of commands and discover what format the rover expected, it was imperative to dive directly into the RE-RASSOR's code in the Github provided to us by FSI. Due to the contact at FSI, Mike Conroy being unavailable for the first portion of the semester, the team was unable to gain access to the FSI Github for some time. When access was eventually granted, however, due to the lack of documentation there was a lot to search through on the FSI Github for the current iteration of Courage's code (though thankfully, it did not take terribly long to find it).

Once Courage's code was found, thanks to a few tips picked up during a session speaking with FSI, the team was able to search a small portion of the code to find the controller server section. There, the team searched for any apparent endpoints to learn the expected format of Courage's commands and what commands were considered valid. Eventually, it was discovered that the rovers expected commands to be sent via a POST request containing a JSON file, and were able to put together a list of valid commands. More details on the valid commands and what elements are expected in the JSON file can be found in **IX. DOCUMENTATION**.

In addition to this, after searching through the code some more, an odd quirk was found with how the rovers handle ArUco markers. The RE-RASSOR rovers are fitted with a camera that naturally detects ArUco markers and logs where they are in the camera's vision. Furthermore, a view of the camera stream, complete with the ArUco markers being marked. However, the raw data of these ArUco markers never leave the rover, instead being baked directly into the camera stream. Because of this, the team will have to develop a new API to grab ArUco marker data from the rover and process it externally.

Due to the lack of documentation causing issues time and time again, the group will likely discuss how to handle documentation for not only the current work, but the pre-existing work done by the other teams. Should it be chosen to start writing more documentation than absolutely necessary to finish the project, though, it will likely take a lesser role in the existing responsibilities. Whether or not it was chosen to write more documentation on others' work, the experience of getting the project off the ground has more than convinced us to make documenting the current work a priority, especially considering the team's commitment to making the extension easy to extend and implement.

Steam Education

XIV. CONCLUSION

The STEAM Education senior design project aims to create impactful educational software geared towards early computer science education. The program interface, lesson plan integration, and final competition are centered on improving student retention of coding concepts, acting as a building block in today’s technology-focused world. It seeks to give students a head start on learning some of the most important programming concepts while also allowing them to have fun. The open-source nature of the program allows a broader reach beyond schools, allowing for continued development in a field that is constantly changing and building upon itself. This seeks to increase the availability and accessibility of programming and robotics-related work.

The Pilot program in Discovery Middle School will provide real-time feedback on the effectiveness of the program, identifying gaps in student learning outcomes, and allowing the team to tailor the program to bridge any deficiencies before general release to the public. The aim is to make the program as self-sufficient as possible without any need for intervention from the development team or anyone beyond the teacher who is in the classroom with the students for the program. Thus, instructor tools are vital for problem-solving and addressing common student questions and difficulties. All resources regarding the project will be available through FSI once the program is ready for public release, similar to the RE-RASSOR rover build documentation.

Depending on community feedback to the program, there are multiple facets for extension. One of the largest ways of building upon this project would be by developing a simulation environment, as outlined in this project as a stretch goal.

Further challenges and competitions and an extended lesson plan to expose students to more robotics concepts would also be beneficial for students who want to learn more about programming. Additionally, developing a system that gives students the ability to visually see the block coding alternatives to the text-based programming they are working through would help bridge the gap between what many of them have already seen. Lastly, giving teachers a system that allows them to assist students at a faster pace while also not requiring high technical knowledge on the part of the teachers would be a beneficial way of building on this project. These are some of the additional components that could be implemented to continue to develop the best possible teaching experience for young programmers.

Student engagement and interest are paramount to the successful deployment of the after-school program, so the software will be built with the end users in mind to create an engaging and interactive robot coding process. This program is intended not only to be helpful for teaching students how to program with RE-RASSOR rovers, but also to allow students to have fun at the same time. This is not a required topic in a class; it is a course that students will be voluntarily signing up to take in their free time, and as such, they should be having fun while learning. Ultimately, the goal of this project is to give young students who aspire to get into a field that can be incredibly daunting the means to learn about programming.

Steam Education

XV. ACKNOWLEDGEMENT

We’d like to give special thanks to the teachers, students, and FSI personnel who helped us with this project. Their support, feedback, and insights have facilitated the production of this school program.

Special thanks to Ms. Robinson of Discovery Key Middle School for her teaching expertise and for the opportunity to test our school program with middle school students. These insights and opportunities served us to great benefit in ensuring audience satisfaction with the developed resources. Her assistance with lesson planning, gauging interest in the program, and facilitating the setup of the pilot program are integral to the development of the teaching software.

A big thank you to Discovery Key Middle School and it’s students for participating in our pilot program. Their willingness to learn has given us ample opportunities to fine-tune this project to the needs of both students and teachers alike.

A big shout-out to all the Senior Design groups who came before us, because it’s thanks to their hard work that RE-RASSOR rovers function in the first place. Some of them even went as far as to help us get oriented when we first started this project. It really makes it feel like we’re standing on the shoulders of giants. We can only hope that our work also becomes a great foundation for future team projects. Thank you.

We are very grateful for the knowledge and support of Christian Vincent, whose profound knowledge of RE-RASSOR rovers enabled us to quickly grasp how they work and what we needed to teach. He also rebuilt Courage for us and gave us the idea to use ArUco markers as part of the program.

We’d also like to thank our professors at UCF for passing down their knowledge, taking the time to guide us, and serving as a connection point between us and these opportunities. They are the ones who keep the school running with so many opportunities to learn. The notable ones are our Senior Design Professors: Dr. Gerber, Dr. Amoruso, and Dr. Leinecker.

We are also grateful to the Teaching Assistants who guided us through the first half of this project: Julia De Macedo Soares Da Silva and Alex Reyes.

FSI’s Lab and Rovers were a huge help in testing, learning, and teaching. Without Courage, a lot of the later units and the competition would have been difficult to test, and building our own Rover would have taken a lot of extra time and money. Thank you.

Finally, we’d like to thank Mike Conroy for providing us with this opportunity to prove our skills as Computer Scientists. This opportunity has opened our eyes to the education side of Computer Science, and we are very grateful for it.

XVI. REFERENCES

[1] Q. Ou et al. “Investigation and analysis of the current situation of programming education in primary and secondary schools,” *Science Direct*, vol. 9, no. 4, April 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405844023027378. [Accessed October 9, 2025]

[2] L. Suters, and H. Suters. “Coding for the Core: Computational Thinking and Middle Grades Mathematics,” *CITE Journal*, vol. 20, no. 3, 2020. [Online]. Available: https://citejournal.org/volume-20/issue-3-20/mathematics/coding-for-the-core-computational-thinking-and-middle-grades-mathematics/. [Accessed October 9, 2025]

[3] S. Grover, N. Jackiw, and P. Lundh. “Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school,” *Taylor & Francis*, vol. 29, no.2-3, 2019. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/08993408.2019.1568955. [Accessed October 9, 2025]

[4] S. Grover, R. Pea, and S. Cooper. “Designing for deeper learning in a blended computer science course for middle school students,” *Taylor & Francis*, vol. 25, no. 2, 2015. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/08993408.2015.1033142. [Accessed October 9, 2025]

[5] Dynalist Inc. “Terms of service,” Obsidian. https://obsidian.md/terms (accessed Nov. 24, 2025)

[6] Microsoft, “Microsoft software license terms,” RSS. https://code.visualstudio.com/license (accessed Nov. 24, 2025).

[7] C. Duan, “Advancing Open Education through open-source software: Examining UTAUT 2 factors in adoption and implementation,” *Asian Association of Open Universities Journal*, vol. 19, no. 3, pp. 313–326, Oct. 2024. [Online]. Available: doi:10.1108/aaouj-09-2024-0119 [Accessed Nov. 24, 2025]