Cacti Council’s Minetest: Coding Education via Modding

Abstract

Cacti Council’s revision of Minetest, an open-source recreation of the popular sandbox game Minecraft, is to encourage programming skill development through ease of access to modding techniques in various languages. Currently, Minetest builds game data is written in Lua, an unpopular programming language according to TIOBE index and Cacti Council’s mission is to write out the dependencies on this language and write in a more compilable language such as C or C++, which is ranked 2nd and 3rd in the most popular languages [6]. In Design 2, we break down the current functions of the game into their Lua dependent C++ code and strictly C++ code ensuring that when the game data is rewritten as much code is salvaged from the project as possible.

Introduction

In the 21st-century educational landscape, the prominence of digital technologies cannot be understated. These tools, which include everything from online platforms to interactive simulations, reshape the way knowledge is shared and consumed. Modern students, adept at navigating the digital realm, seek learning experiences that mirror their everyday interactions: dynamic, engaging, and interactive. Responding to this need, the educational sector has increasingly embraced gamification, a strategy that infuses game-like elements into learning and turns passive absorption into active engagement.

Minetest stands out in this gamified world as an open-source counterpart to the globally recognized Minecraft, Minetest provides a sandbox environment in which creativity and learning converge. It is not just a game; it is a versatile and cost-effective tool that promises to revolutionize classroom experience. Given its open-source nature, Minetest allows for the free access to its game engine and encourages players to create and innovate.

However, every tool, no matter how powerful it is, has room for enhancement. Minetest's reliance on Lua scripts presents a steep learning curve for those unfamiliar with this programming language even with all the included documentation. This is where our team stepped in, with a mission to make Minetest even more accessible and versatile. Our team’s goal is to recreate the Lua game data in C++ and our project revolves around rewriting and developing C++ functions that can interface with the existing Lua functions in Minetest. By leveraging C++, a language with wider recognition and acceptance, we aim to lower entry barriers for students and educators.

The ultimate goal of this endeavor is to remodel the game engine in C++, laying the groundwork for multilanguage compilation. Such a shift will not only simplify interactions within Minetest, but will also broaden its appeal, making it a truly global educational platform. Through this project, we endeavored to make remote coding education more accessible and efficient, seamlessly merging traditional teaching methods with modern digital tools to offer learners a comprehensive, intuitive, and enriching experience.

Background

Since the release of modding capabilities in the 90s with games such as Doom and Quake, gamers have been actively customizing their experiences with modding. A seminal study underscoring the educational capabilities of games like Minecraft is pivotal in this context, demonstrating its utility as a coding instruction medium [1][2]. Minecraft offers an engaging platform in which learners can actively shape, modify, and navigate virtual spaces. This research accentuates the potency of such games in engaging learners and making intricate coding paradigms accessible and intuitive.

In Minetest's educational page and CAS TV's youtube seminar present Minetest is currently used as a learning tool for growing developers and coders [2][3]. Not only is it a game that many people enjoy, but also a place where modding is encouraged. With Minetest's server side modding, it allows for all clients to receive the server's mods without the installation difficulties that would normally arise with Minetest's counterparts. This creates an environment where gamers can quickly and efficiently try out each other's work and could potentially encourage collaboration techniques.

Figure 1: Before we began our mission, we created mods in the Lua modding environment to understand the Minetest documentation

Minetest allows the student to quickly and intuitively interact with a sandbox world, creating blocks, changing game physics, adding items, etc [F.1]. Simplified coding techniques have been found to increase student's critical thinking, debugging techniques, evaluation, complexity management, and engage students in high-order thinking [4]. By immersing students in interactive realms where abstract concepts translate into tangible experiences, these platforms captivate attention and foster profound cognitive engagement. This experiential learning not only sustains student interest, but also bolsters understanding and long-term retention.This is why our team decided to recreate Minetest's games data in C++. This would allow for multiple languages that have the ability to compile in C code to generate mods in more popular languages, such as Python, through Cython, and Java, natively built in [5].

Data reveals a pronounced preference for C++ functions against Lua scripts in terms of user friendliness and compilability [6]. Lua, which is intricate yet versatile, exhibits a steep learning trajectory for new coders as shown by its 26th position ranking. Conversely, C++'s widespread educational acclaim makes it an accessible starting point for both educators and learners where it and C and place 2nd and 3rd of most popular languages.

Design

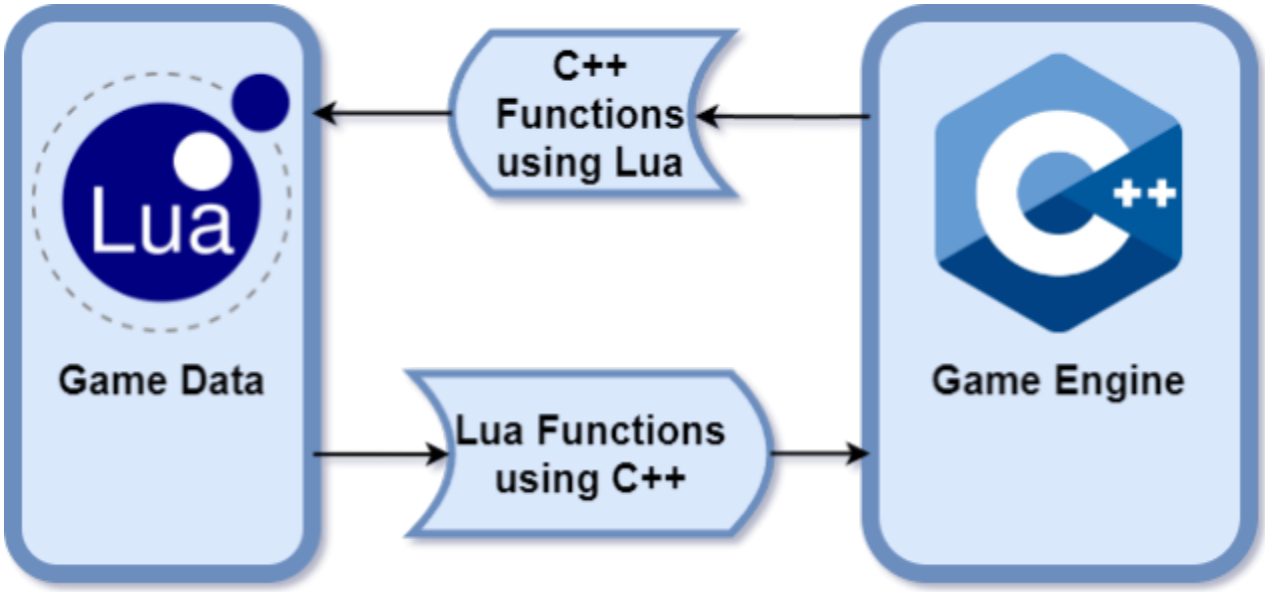


Figure 2: Current Implementation of the Minetest Game Data and Game Engine

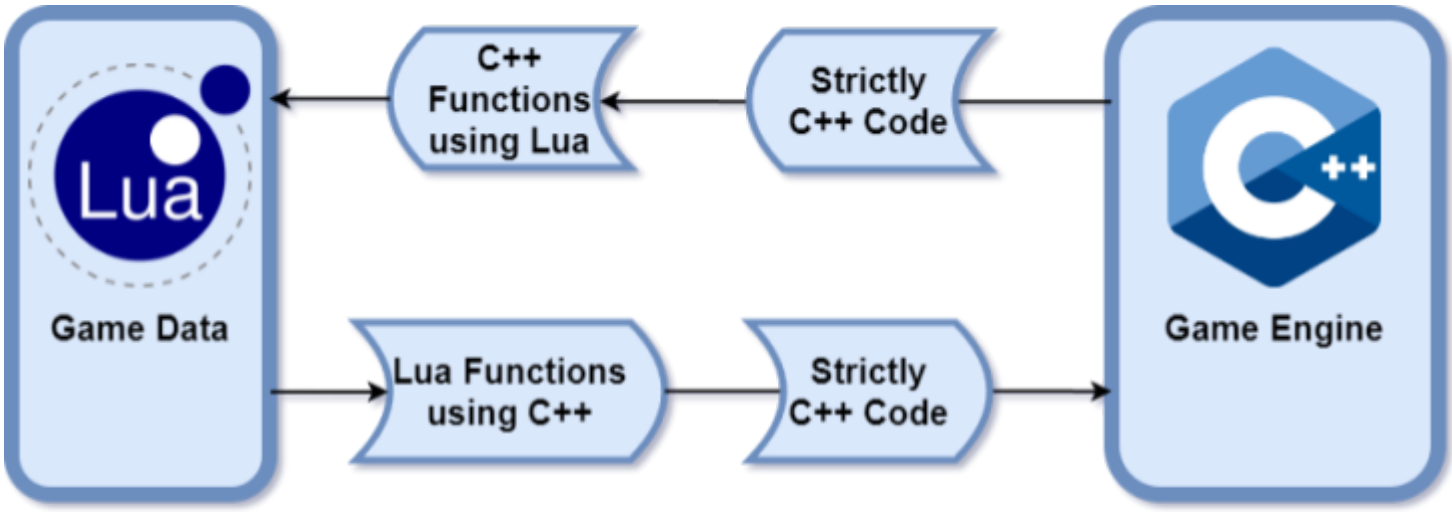


Figure 3:The separation stage of Cacti Council and the focus of the Design 2 team are working towards

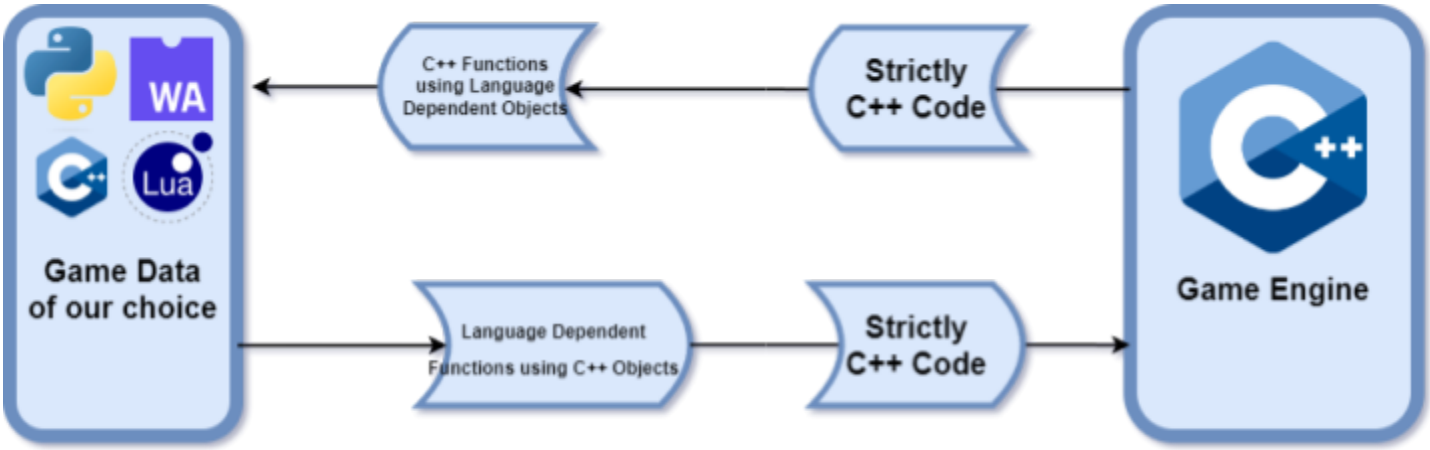


Figure 4: Future plans for the Cacti Council and the end goal for the Minetest project

Current implementation of Minetest suggests a program interaction such as a diagram from above [F.2]. Lua game data holds tick information of players in Lua tables and C++ uses Lua dependent code to scrape said data from the tables. The Cacti Council's first stage of rewriting game data is to salvage as much C++ code from the game's function as possible to reduce full function rewriting [F.3]. In future stages of development the game data can be rewritten in any language of our choosing, preferably an accessible(Python, C++, or C) or easily compilable language (C, C++, or ASM). Currently the team is leaning towards C++ but that's for future development plans [F.4].

The current initiative aims to increase accessibility for future coders to develop mods and plugins, with the team having already converted 171 out of 480 functions before joining the Minetest team and around 65% remaining functions for both teams to separate before reworking the game data. Through analyzing the codebase and identifying areas where Lua was used, the team worked on converting Lua functions to C++ and modifying the game's engine to work with the new code, with the ultimate goal of making the Minetest API more user-friendly, bringing it closer to its objective of being utilized in classrooms.

Following in the footsteps of the previous teams we found that in order to optimize the process of rewriting Minetest functions, our team will partition the work into distinct classes, each comprising one or multiple functions that interact with the Lua script functions. To guarantee the effectiveness and accuracy of the rewritten functions, our team has incorporated a three-stage process for each member to follow. The initial stage, "written," entails creating and registering Native API functions in the corresponding Lua bound C files. In the subsequent stage, "compile," team members draft chat commands for each Native API function, invoking them and ensuring they can be compiled and executed without errors or yielding nil values. The final stage, "test," concentrates on verifying the accuracy of the rewritten functions by crafting chat commands for each Native API function and comparing the outcomes with the corresponding Lua API function. Should any discrepancies emerge, the Native API function is adjusted to maintain consistency with the Lua function. This system, based on Cacti Council's approach, enables our team to effectively rewrite and validate the new C++ functions while preserving the integrity of the original Minetest functions.

Finally, the importance of fluid communication in such a vast collaborative endeavor cannot be overstated. Platforms such as Slack and Microsoft Teams have been our communication lifelines, whereas tools such as Google Docs and Sheets have allowed real-time editing and task assignment capabilities. Together, these elements culminate in a design that promises to redefine the Minetest experience.

Tools and Standards

In an age characterized by rapid technological advancement and an increasing emphasis on digital tools for education, projects like ours are not only timely, but also essential. The transition of Minetest from Lua to C++ is not merely a shift in programming language; it signifies a broader commitment to making high-quality immersive educational tools universally accessible.

Our study strongly underlines the tenets of inclusivity. With Minetest's open-source nature, the tool has become readily available to a vast audience, transcending geographical and cultural boundaries. Every culture, regardless of its economic prowess or technological advancement, deserves access to top-notch educational tools. By offering a universally adaptable platform, we are enabling educators from around the world to tailor Minetest to resonate with their unique curricula and cultural nuances.

On the global front, the project addresses a pressing need. Quality education should never be a privilege confined to a specific region or economy. Our endeavor with Minetest ensures that irrespective of where a student is located, they have an equal shot in a dynamic learning experience. The economic implications of

these findings are profound. Commercial alternatives often come with exorbitant price tags, making them inaccessible to institutions operating with tighter budgets. Minetest, with its open-source foundation, eliminates these financial barriers and levels the playing field.

The environmental impact is noteworthy beyond the cultural and economic realms. By championing a digital platform, we inadvertently contribute to reducing reliance on physical resources and ensuring a more sustainable ecological footprint. This aligns with the global call for more eco-friendly educational solutions, emphasizing the importance of conservation in every sector, including education.

Moreover, with the increasing adoption of digital platforms, concerns regarding data privacy and security have inevitably arisen. As we chart new territories in digital education, ensuring the safety and privacy of every user becomes paramount. Although our platform prioritizes user experience, we are equally committed to safeguarding the data entrusted to us, emphasizing a transparent, secure, and ethically sound operational framework.

Needs and Impacts

In an age characterized by rapid technological advancement and an increasing emphasis on digital tools for education, projects like ours are not only timely, but also essential. The transition of Minetest from Lua to C++ is not merely a shift in programming language; it signifies a broader commitment to making high-quality immersive educational tools universally accessible. With Minetest's open-source nature, the tool has already become readily available to a vast audience and encourages a new age of programmers through the gamification of programming. By offering a universally adaptable platform, we are enabling educators from around the world to tailor Minetest to resonate with their unique teaching and programming needs. Being not only able to write mods in multiple languages but play other mods server side could create a modding community diverse not only in location but also programming language expertise.

On the global front, the project addresses a pressing need. Quality education should never be a privilege confined to a specific region or economy. Our endeavor with Minetest ensures that irrespective of where a student is located, they have an equal shot in a dynamic learning experience. The economic implications of these findings are profound. Commercial alternatives often come with exorbitant price tags, making them inaccessible to institutions operating with tighter budgets. Minetest, with its open-source foundation, eliminates these financial barriers.

Results

Minetest contains 31 classes that interact with Lua within C++ functions. Of those classes,19 have been successfully separated into its Lua dependent and C++ counterparts within the Lua and Native directories [F.5]. Currently the Design 2 portion of the Minetest team has completed milestones such as Rollback and Storage classes. These functions, while quite small, still took us 3 weeks to complete as translating variables made within Lua to C++ is not simple. In total we converted 3 functions, which all code can be found within our GitHub.

Over the summer and during Design 2, we wanted to conquer the largest class by number of functions in the game, object. This class consisted of 95 unique functions that were integral to the game's mechanics. Separating the code into two functions is not the only necessary step, our team tests each code and creates comparison tests for each function to ensure Cacti Council’s native version of the functions return the same as the Lua function.With around 65% of the total function in object being rewritten and 63% of testing the original functions, the native and test comparisons have yet to take a priority but with our shift into documenting the modding process for new members, we plan to tackle this for the final release [F.6]. We’ve been making steady

progress towards the separation of the Lua and C++ code and by the end of our project we hope to have the object class completed.

Class	Num Funcs	Type	Assignee	Written	Compiled	Tests Written	Tested
l_areastore.cpp	11	Server	Lisa	Done	Done	Done	Done
l_auth.cpp	6	Server	Nathan	Done	Done	Done	Done
l_base.cpp	1	Server	Unassigned				
l_camera.cpp	9	Client	Nathan	Done	Done	Done	Done
l_client.cpp	23	Client	Nathan	Done	Done	Done	Done
l_craft.cpp	5	Server	Matthew	In Progress			
l_env.cpp	45	Both	Nathan	In Progress	In Progress	In Progress	In Progress
l_http.cpp	3	Server	Chris	In Progress			
l_inventory.cpp	19	Server	Lisa	Done	Done	Done	Done
l_item.cpp	33	Both	Colin	Done	Done	Done	Done
l_itemstackmeta.cpp	1	Both	Erik	Done	Done	Done	Done
l_localplayer.cpp	26	Client	Ilsa	Done	Done	Done	Done
l_mainmenu.cpp	39	?	Unassigned				
l_mapgen.cpp	33	Server	Lisa	In Progress	In Progress		
l_metadata.cpp	11	Server	Erik				
l_minimap.cpp	10	Client	Lisa	Done	Done	Done	Done
l_modchannels.cpp	4	Both	Erik	Done	Done	In Progress	In Progress
l_nodemeta.cpp	2	Both	Unassigned				
l_nodetimer.cpp	6	Server	Erik	Done	Done	Done	Done
l_noise.cpp	13	Server	Lisa	Done	Done	Done	Done
l_object.cpp	95	Server	Sean	In Progress			
l_particles_local.cpp	3	Client	Anthony	Done	Done	Done	Done
l_particles.cpp	3	Server	Anthony	Done	Done	Done	Done
l_playermeta.cpp	0	Server	Unassigned				
l_rollback.cpp	2	Server	Sean	Done	Done	Done	In Progress
l_server.cpp	27	Server	Unassigned				
l_settings.cpp	10	Server	Lisa	Done	Done	Done	Done
l_sound.cpp	2	Mainmenu	Unassigned	Done	Done	Done	Done
l_storage.cpp	1	Both	Chris	Done	In Progress	In Progress	
l_util.cpp	21	Both	Unassigned				
l_vmanip.cpp	16	Server	Unassigned				
TOTAL NUM FUNCS	480			COMPLETION			36.46%

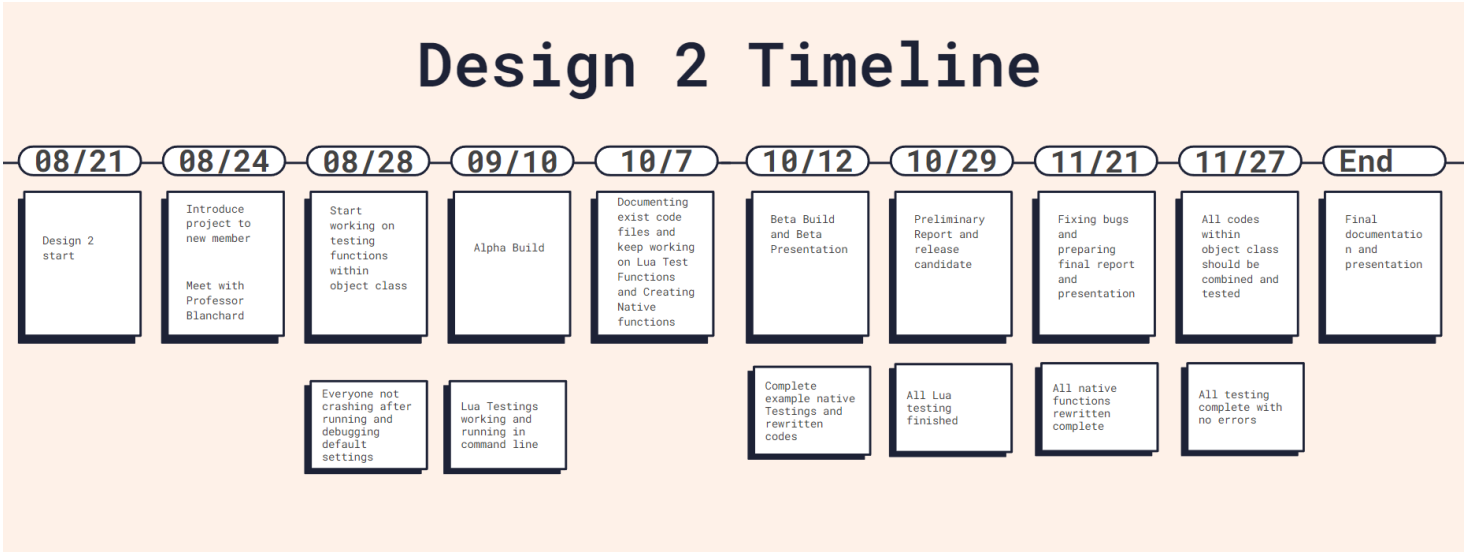
Figure 5: The Cacti Council's Class Assignment Sheet

During the Beta build and revision in the Release Candidate, our team developed a new automated logging method which called all methods with a predetermined prefix. In the log file, the function printed whether the function call was successful or failed and what line(s) the code failed on. We have also been able to print the debug file to the output as well as the time Minetest takes in running each function, which proves useful for comparing if our functions change the game performance significantly. This tool has quickly become adopted by the rest of the team and is now integral to the development stages.

C Func	Lua	Native	Comparison
65.00%	63.00%	6.00%	0.00%

Figure 6: The Design 2 Portion of the Minetest team's completion of Object class

Status



Since the initiation of our Design 2 project on 08/21, we have remained dedicated to this mission. With the guidance of Professor Blanchard, the integration of a new team member, and a focus on refining our testing functions, we have made steady progress. This culminated in Alpha Build by 09/10, after which we navigated through challenges with Lua testing in a command-line setting.

However, after the unveiling of our Beta Build on 10/12, it became evident that the project needed more robust documentation. This realization was primarily driven by our instructor to ensure that the project was comprehensible and accessible to a broader audience. Therefore, instead of strictly adhering to our original timeline, we chose to invest more time in crafting detailed documents for our preliminary report on 11/3. This shift became even more pronounced as we approached the release candidate phase.

We openly acknowledge that this deliberate and strategic pivot towards enhanced documentation has placed us slightly behind our initial schedule. However, we believe that this will serve as an invaluable resource for future users and collaborators, making it easier for them to understand and engage with our project. We hope to integrate all codes within the object class by 11/27 and prepare demo tools for our final presentation. Throughout this journey, our commitment to transparency and communication remained steady. By regularly updating our team and stakeholders regarding our progress and decisions, we are confident that this project will be fruitful and impactful.

Conclusion

Our journey with the Minetest Project has been a blend of innovation and collaboration. We transitioned from Lua to C++ to improve teaching viability, and our selection of development tools and communication platforms has streamlined our efforts. Our Google Sheets checklist has been instrumental in managing tasks without overlap, and our distinct testing approach, particularly the in-game chat command, highlights our dedication to quality. Reflecting on our progress, we remain committed to refining our processes and delivering a seamless gaming experience. Our aim is clear: continuous improvement for our team, Cacti Council, and the Minetest community.

References

[1] M. S. El-Nasr and B. K. Smith, “Learning through game *modding*,” *Computers in Entertainment*, vol. 4, no. 1, p. 7, 2006. doi:10.1145/1111293.1111301

[2] “Minetest for Education,” Minetest, https://www.minetest.net/education/.

[3] J. Lamb, “Minetest as a teaching resource,” YouTube, https://www.youtube.com/watch?v=kH6DhUYvZc4&ab_channel=CASTV.

[4] M. Grizioti and C. Kynigos, “Game modding for computational thinking,” *Proceedings of the 17th ACM Conference on Interaction Design and Children*, 2018. doi:10.1145/3202185.3210800

[5] Dbohdan, “Dbohdan/compilers-targeting-C: A list of compilers that can generate C code,” GitHub, https://github.com/dbohdan/compilers-targeting-c.

[6] “Tiobe index,” TIOBE, https://www.tiobe.com/tiobe-index/.

[7] “CMake: The Standard Build System,” CMake, https://cmake.org/features/.