Project Paper: Computational Thinking: An Educator Tool

Samuel Adegoke
sadegoke3@gatech.edu

ABSTRACT

Computational thinking is an essential skill, widely advocated within educational curricula. Despite its recognized importance, educators struggle to identify and assess computational thinking effectively, especially in non coding contexts like popular video games. This paper presents a novel web based tool designed to help educators recognize and annotate computational thinking skills exhibited by students in mainstream, non coding games such as Minecraft, Teamfight Tactics, and Super Smash Bros. Leveraging intuitive interfaces and clear visualizations, the tool simplifies the annotation process, requiring no coding experience from educators. Initial usability tests demonstrate the tool's accessibility, effectiveness, and potential for integration into classroom practices. This paper discusses the design rationale, implementation details, user feedback, identified limitations, and future directions for research and development.

INTRODUCTION

The integration of computational thinking into K-12 education has gained substantial attention in recent years. Despite this growing emphasis, there remains a significant challenge in identifying and assessing computational thinking within non-coding contexts, such as video games. Games such as *Minecraft*, *Teamfight Tactics (TFT)*, and *Super Smash Bros* inherently demand strategic thinking, problem solving, pattern recognition, debugging, and abstraction. These skills correspond closely to the core principles of computational thinking, yet traditional assessment tools predominantly focus on explicit coding tasks or specialized software setups. This gap represents a missed opportunity for educators to harness the widespread engagement of students with popular games to promote and evaluate critical computational skills.

This paper introduces a lightweight, accessible, and educator friendly web based tagging tool designed explicitly for identifying and annotating computational thinking skills within gameplay scenarios. The primary aim is to facilitate the recognition of computational thinking through games students regularly play,

thereby bridging the gap between entertainment and education.

BACKGROUND AND MOTIVATION

Computational thinking is increasingly recognized as a critical skill for the 21st century. With its emphasis on decomposition, abstraction, pattern recognition, and algorithmic thinking, computational thinking serves as a foundation for many disciplines beyond computer science. However, despite its growing importance, implementing computational thinking in classrooms remains a challenge, particularly for educators with limited technical backgrounds or those outside of STEM disciplines.

Video games, widely adopted by students, offer an underutilized pathway for computational thinking assessment. Many games require players to engage in strategic planning, iterative testing, and dynamic adaptation. Mirroring the problem solving cycle inherent in computational thinking. Unlike block based coding platforms or robotics kits, video games provide a familiar and engaging environment where students naturally exhibit computational thinking skills. Recognizing this potential, this project explores a tool that surfaces and assesses computational thinking behaviors in gaming contexts.

RELATED WORK

Computational thinking, initially popularized by (Wing 2006), emphasizes problem solving skills fundamental to computer science, yet applicable broadly across disciplines. (Grover 2013) further expanded this definition, advocating for computational thinking as a foundational literacy akin to reading and arithmetic. (Weintrop 2016) introduced a taxonomy defining specific competencies associated with computational thinking, highlighting skills such as abstraction, decomposition, algorithmic thinking, and debugging as central to the framework.

Research indicates video games offer a potent avenue for cultivating computational thinking. (Cipollone 2014) demonstrated that open ended games like *Minecraft* facilitate constructionist learning, allowing students to build knowledge through interactive exploration. (Kafai 2015) similarly explored how *Minecraft* could support meaningful computational thinking experiences, noting its particular efficacy in fostering collaborative problem solving skills. (Bavelier 2012) contributed further insights, finding that fast paced, action oriented games enhance players' adaptive decision making and pattern recognition abilities.

Several frameworks have emerged to measure computational thinking in digital environments. (Asbell 2021) developed the INFACT framework, emphasizing inclusivity in computational thinking assessments. (Wu 2025) expanded this concept, integrating design based thinking into computational thinking assessments. (Chng 2019) notably demonstrated that computational thinking skills could be identified and developed even among students with minimal coding experience.

Despite these advancements, existing assessment tools remain limited in their capacity to effectively identify and visualize computational thinking skills within mainstream, non coding games. The absence of accessible, intuitive tools specifically for non coding environments remains a significant gap that this project directly addresses.

DESIGN GOALS

The design of the tool was guided by several goals:

1. **Accessibility**: The interface had to be approachable for educators with no coding experience. Many existing computational thinking tools assume some familiarity with programming or at least a willingness to deal with complex interfaces. The design aimed to ensure any educator, regardless of technical background, to feel comfortable right away. That meant choosing intuitive UI elements like checkboxes and sliders instead of requiring code or specialized knowledge.

2. **Flexibility**: The tool needed to work across different kinds of games and scenarios. Teachers use games differently, and students play all sorts of genres. Limiting the tool to just one type of gameplay or a specific set of scenarios wouldn't capture the real classroom diversity. So, the tagging interface was built to handle screenshots, text descriptions, and even short video clips, allowing educators to choose the input format that works best for their context.

3. **Usability**: The whole process had to be simple enough that teachers could start tagging right after minimal onboarding. If it took too much training or felt cumbersome, educators wouldn't stick with it. Clear tooltips defining computational thinking skills, an uncluttered layout, and minimal required fields helped achieve this simplicity. The goal was to minimize friction, letting educators focus on understanding students' behaviors rather than figuring

out how to use the tool itself.

4. **Insightfulness**: The tool wasn't just about making it easy to tag scenarios, it needed to help educators quickly see patterns and make sense of their data. That meant including straightforward visualizations like bar charts to see skill frequency, radar charts to compare skills across different scenarios, and pie charts to capture educators' confidence levels. These visual summaries help teachers spot trends quickly and can inform instructional decisions without needing external analysis.

These goals align with prior work on educational technology design principles, emphasizing simplicity, immediate feedback, and low entry barriers.

SOLUTION OVERVIEW

The tool is built using Streamlit, a Python library that makes it easy to create clean, interactive web apps. The goal was to give educators a lightweight way to help students tag and reflect on computational thinking in gameplay without needing any programming experience. Educators can paste in gameplay scenarios as text and tag them using a simple checklist of computational thinking skills. Each skill comes with a built-in tooltip that defines what it means in context, so even if someone's new to the terminology, they're not guessing. The design keeps things flexible and approachable, while still grounded in core computational thinking concepts. Overall, it is meant to lower the barrier to recognizing computational thinking in the kinds of games students already love to play.

The tool allows educators to:

- Upload and view gameplay scenarios.
- Tag computational thinking skills based on a curated list.
- Set a confidence level for each tag.
- Add reflection notes to explain reasoning.

Computational thinking skills implemented in the tool include:

- Pattern Recognition
- Abstraction
- Decomposition
- Algorithmic Thinking

- Debugging
- Evaluation and Refinement

IMPLEMENTATION DETAILS

The tool was built using Streamlit for the frontend and Python to handle backend logic. Streamlit made it easy to get a clean, interactive UI up and running without having to mess with a lot of custom HTML or JavaScript.

All tagged data is stored in JSON format. JSON was selected over something heavier like a database to keep the project lightweight and portable. It also makes it easy to read, edit, or share data across systems without needing to set up an external service. Each annotation includes the selected computational thinking skills, confidence slider value, and any optional reflection notes entered by the user.

For visual feedback, the tool includes three real time charts that update as scenarios are tagged. The bar chart shows how often each computational thinking skill has been selected across the dataset which is great for spotting trends at a glance. The radar chart breaks it down per scenario, helping educators quickly compare the computational thinking skill distribution across different gameplay situations. The pie chart summarizes the confidence levels selected by users, which adds an extra layer of insight into how certain (or uncertain) users were while tagging.

All visualizations are generated using Plotly, which allowed for interactive and responsive charts without a steep learning curve. They're embedded directly in the tagging interface, so users don't have to switch pages or run separate reports to get insights.

Accessibility was another priority. The interface supports keyboard navigation for users who prefer or rely on non mouse input, and ARIA labels were added to ensure screen reader compatibility. These decisions were aimed at making the tool usable not just for a wide range of educators, but also for future accessibility audits if the tool is deployed in more formal settings.

Looking ahead, the current setup provides a solid base for expanding features like CSV export, user accounts, or even automated tagging without having to rewrite the core architecture.

METHODOLOGY

To evaluate how usable and effective the tool was, a small scale usability study was conducted with five peers. Each participant was asked to interact with the tool by tagging several gameplay scenarios and then provide feedback through a Google Form followed by short one on one interviews. The idea was to capture both quantitative impressions (ease of use, clarity) and qualitative insights about how the tool might actually fit into a classroom workflow.

Participants were given a short onboarding guide and a few sample scenarios to work through. They were encouraged to try out the full tagging process, including selecting computational thinking skills, adjusting the confidence slider, and writing reflection notes, which is optional. The feedback they gave focused on three main areas: whether the computational thinking skill definitions were clear and understandable, how easy the tagging interface was to navigate, and how useful the overall experience felt in terms of recognizing computational thinking.

Alongside the user testing, I manually tagged a dataset of 22 gameplay scenarios myself to build out the underlying data and test the tool's consistency across games. The scenarios were pulled from Minecraft, Teamfight Tactics (TFT), and Super Smash Bros. These three games that were chosen because they each demand different types of decision making and problem solving. Minecraft leans into open ended design and systems thinking, TFT emphasizes real time strategic adaptation, and Smash offers fast paced pattern recognition and reaction timing. This mix was intentional so that the computational thinking skills being tagged weren't limited to just one genre or gameplay style.

The manual tagging process also helped refine the tooltips and test whether certain computational thinking skills were showing up more frequently in particular games. That kind of pattern insight, even in a small sample, offered early signs that the tool could help surface interesting trends about how different games support different computational thinking practices.

Overall, the methodology wasn't about large scale validation but about collecting focused feedback to guide iteration and set a foundation for future classroom pilots.

SCENARIO ANNOTATION EXAMPLES

- **Game: Minecraft**
  Title: Torch Placement Optimization
  Scenario: A player optimizes the placement of torches to prevent monster spawning.
  Tagged Skills: Evaluation and Refinement, Pattern Recognition
  Confidence: 5

- **Game: Teamfight Tactics**
  Title: Opponent Scouting and Prediction
  Scenario: A player scouts opponents and predicts optimal positioning.
  Tagged Skills: Pattern Recognition, Algorithmic Thinking **Confidence:** 5

- **Game: Super Smash Bros**
  Title: Opening Pattern Recognition
  Scenario: A player identifies a repeatable opening against a specific playstyle.
  Tagged Skills: Pattern Recognition, Evaluation and Refinement
  Confidence: 4

These examples illustrate the versatility of the tagging tool across different game genres and styles.

USER FEEDBACK AND IMPROVEMENTS

Overall, the feedback from the pilot testers was encouraging. Most participants described the tool as clean, easy to use, and surprisingly intuitive and they were not familiar with tagging frameworks beforehand. The inclusion of tooltips for each computational thinking skill stood out as one of the most helpful features. A few users mentioned they had a better understanding of what "abstraction" or "evaluation" meant in practice just by hovering over those definitions. That was great to hear, since one of the goals was to make computational thinking more approachable for educators who don't have a technical background.

There were a few suggestions that helped refine the tool's usability. For example, several testers noted that the spacing between interface elements felt a little tight, so I added padding and improved layout alignment to make things easier on the eyes. The tooltips were also originally placed too close to the edge of the screen, which made them clip off depending on screen size. To fix that, I repositioned

those for better visibility. One other useful note was that the reflection prompt, while valuable, could feel like extra work. Making that field optional helped lower the barrier to completing a tagging session while still encouraging deeper thought for users who wanted to engage more critically.

I also created two basic educator personas to explore how different types of users might engage with the tool. An educator thats just starting out (maybe someone just getting into computational thinking) might use the tool as a way to get familiar with the terminology and start noticing computational thinking moments in games their students already play. On the other hand, a more experienced teacher could use it to guide formative assessment, track common patterns in student behavior, or even facilitate classroom discussions around decision making and problem solving. The tools simplicity worked well for both cases, which was a core goal from the beginning and that is to keep it lightweight but useful.

Overall, the feedback confirmed that the core interaction model was solid, and the tweaks that came out of testing helped smooth out rough edges to make the tool more classroom friendly.

LIMITATIONS

While the tool has shown a lot of promise so far, there are still some clear limitations that need to be acknowledged before it can be scaled or widely adopted.

First, the user testing was pretty limited. Just five participants, all with similar academic backgrounds and tech familiarity. That's fine for early feedback, but it does not reflect the range of actual educators who might use this in real classrooms. Teachers in different subjects, age groups, or even school environments may have totally different needs or challenges. Getting feedback from a more diverse group would definitely help shape future iterations.

Second, all the tagging in this version was done manually. Manual tagging is workable for now, but it opens the door for inconsistency, especially with subjective skills like "abstraction" or "algorithmic thinking." Without a rubric or multiple people tagging the same scenarios to compare results, it is hard to tell how consistent or reliable the tags really are. That becomes more of a concern if the tool is ever used in formal assessment or research.

Third, the tool uses a fixed set of computational thinking skills based on the

Weintrop framework. It is a solid foundation, but it does not leave room for custom tagging or adjustments. Some teachers might want to align tags with their own rubrics, state standards, or classroom goals. Right now, that's not something they can do in the tool.

Fourth, while the visualizations (bar, radar, pie charts) give a nice overview, they're pretty basic. There's no way to filter by confidence levels, track changes over time, or compare between different students or sessions. If the tool's going to be used for more in depth insights or instructional planning, the analytics side will need to evolve.

Finally, the tool assumes that short written descriptions can fully capture gameplay context. Also, that's not always the case. Some in game actions happen quickly or depend on subtle strategies that aren't easy to explain in a single sentence. Without richer context, like video or audio cues, some of the computational thinking behaviors might get missed or misinterpreted.

None of these are dealbreakers. They're good reminders of what still needs to happen to take this from a solid prototype to something that can hold up in real classrooms at scale.

CONCLUSION

This project introduces a simple but powerful tool designed to help educators recognize computational thinking in games students already play. By making it easy to tag and visualize computational thinking skills in titles like Minecraft, TFT, and Super Smash Bros, the tool bridges the gap between what students do naturally in gameplay and how teachers can make sense of that in an educational context. It gives educators a way to connect classroom goals with authentic student behavior. Which in this case would be without needing code, special training, or expensive platforms.

What makes this tool different is its accessibility. Most computational thinking tools require some programming background or a controlled learning environment. This one does not. It was built to be fast, lightweight, and usable by a wide range of educators, whether they're just starting to explore computational thinking or already have experience teaching it. The interface is straightforward, the skill tags are research backed, and the visualizations offer real time feedback without feeling overwhelming.

The feedback so far shows that this approach works. My peers appreciated how easy it was to use, and how the tooltips and reflection notes helped them make better sense of what they were seeing in gameplay. The tool isn't just about assessment, it is really about helping educators build a vocabulary for recognizing the kinds of thinking that often go unnoticed when students aren't writing code.

There's still work to do to make the tool more scalable and flexible, but the foundation is strong. If nothing else, this project shows that recognizing computational thinking in non coding environments does not have to be complicated. With the right design, focus and receiving feedback from legitimate educators, it can be simple, transparent, and actually useful in real classrooms.

FUTURE WORK

There's still a lot of room to grow this tool beyond the current prototype. One of the biggest next steps is exploring how to automate the tagging process using machine learning. Right now, everything is done manually, which works for small datasets but isn't scalable if educators start using this across dozens of students or long gameplay sessions. Using something like natural language processing or image recognition to suggest tags could save time and reduce friction, especially for teachers new to the computational thinking framework.

Another area to build on is the computational thinking skill taxonomy itself. The current list is based on the Weintrop framework, which is solid, but it does not cover every possible interpretation. Especially in more creative or design based games. Down the line, I'd like to experiment with letting educators customize the list or choose from different computational thinking frameworks depending on their curriculum or learning goals.

I also want to get this tool into actual classrooms. Not just for feedback on usability, but to see how it impacts teaching and learning. A small pilot study with teachers and students would go a long way in understanding how effective it really is when embedded into real instruction, especially if we can track how it shapes student reflection or assessment outcomes.

Some practical upgrades are also on the roadmap. Exporting tagged data as a CSV will make it easier for teachers to pull their results into gradebooks or spreadsheets. And making the whole project open source would invite others to contribute and build on what's already there. The core goal is to keep the tool

simple, flexible, and grounded in real educator needs and these next steps are focused on making that happen.

REFERENCES

Asbell-Clarke, J., Rowe, E., & Sylvan, E. (2021). The INFACT framework: Inclusive computational thinking assessment. *Computers in the Schools*, *38*(1), 44–64.

Bavelier, D., Green, C. S., Pouget, A., & Schrater, P. (2012). Brain plasticity through the life span: Learning to learn and action video games. *Annual Review of Neuroscience*, *35*, 391–416.

Ch'ng, Y. H., & Tai, C. L. (2019). Observing computational thinking development in players without coding experience. In *Proceedings of the International Conference on Learning Analytics and Knowledge* (pp. 405–409).

Cipollone, M., Schifter, C. C., & Moffat, R. A. (2014). Minecraft as a constructionist learning environment. In *Society for Information Technology & Teacher Education International Conference* (pp. 257–262).

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43.

Kafai, Y. B., & Burke, Q. (2015). *Connected gaming: What making video games can teach us about learning and literacy*. MIT Press.

Weintrop, D., Beheshti, E., & Horn, M. S. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Wu, Z., Lee, J., & Kumar, N. (2025). Integrating design-based thinking into computational thinking assessments. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems* (pp. 1–12).