2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
Paper ID #52179

Scaling Open-Ended Projects in CS1: A Framework for Large Enrollment Courses

Dr. Shanon Marie Reckinger, University of Illinois at Chicago

Shanon Reckinger is a Clinical Associate Professor in the department of Computer Science at the University of Illinois at Chicago. She received her PhD in Mechanical Engineering at the University of Colorado Boulder in August of 2011 and an MS degree in Computer Science at Stanford University.

Scaling Open-Ended Projects in CS1: A Framework for Large Enrollment Courses

Abstract

This paper presents a scalable framework for implementing open-ended group projects in large-enrollment CS1 courses. With the increasing availability and capabilities of generative AI tools, it has become essential to rethink how we design curriculum and assessments for introductory programming students. Many traditional CS1 projects can now be easily completed using freely available AI tools, raising concerns about academic integrity and learning outcomes. Additionally, increasing the weight of traditional assessments may heighten anxiety and disproportionately disadvantage students who are less prepared for high-stakes testing. The approach described in this paper offers an alternative model for integrating core programming concepts through open-ended, collaborative projects. While managing such projects at scale poses logistical and assessment challenges, this work outlines detailed procedures for project design, team formation, grading, and student support. Projects are introduced to students in a hackathon-style format, with groups selecting from one of three thematic tracks. Teams are formed based on student ability levels and assigned a TA who serves as a project manager throughout the multi-week experience. Lab sessions in the final weeks of the semester are used for progress updates, with grading based on iterative development and team collaboration. In the final week, students present their projects, and selected showcase teams are allowed to substitute their project for a final exam grade. This model has been implemented over 10 semesters and has shown to be a motivating and rewarding way to conclude the course for both students and instructors. The paper includes student feedback, project examples, and practical insights for instructors looking to adopt or adapt this format.

Introduction

With the growth in the use and availability of generative AI tools in education, there has been increased motivation to develop instructional approaches that encourage students to build authentic programming skills while still allowing instructors to accurately assess both content knowledge and skill development. Traditional introductory programming projects face growing limitations, as many can now be easily solved using publicly available generative AI tools. As a result, students often struggle to use these tools productively when the most accessible option is to simply generate a complete solution. In response, many CS1 courses have increased the weight of exams and shifted toward in-class, pen-and-paper assessments. However, in introductory computing and other engineering courses, this shift has the potential to exacerbate equity concerns and increase student anxiety due to the reliance on high-stakes assessments.

This work aims to describe a scalable, integrity-preserving, and pedagogically sound model for integrating open-ended projects into a CS1 course. The purpose of this paper is to present the framework for these open-ended, group-based CS1 projects, which is a repeatable project model that has been implemented across more than ten semesters. The paper provides practical strategies for project logistics, assessment, and student support, and presents evidence of student engagement and instructional sustainability.

Background

Project based learning is well studied in CS education [2]. PBL has been shown to increase student motivation [8] and enthusiasm [9]. Similarly, it has been shown to increase engagement [10] and attendance [11]. Lastly, some work has shown increases in cognitive skill development like critical thinking [12].

One benefit is that students find PBL more useful and enjoyable when they are able to apply their personal interests to the choice of the project [1]. Additionally, open-end projects have been shown to improve self-efficacy and grades [4]. When projects are student-led and open-ended, studies have found students felt a sense of ownership in bringing their own ideas to fruition [10]. Other work has shown that first year students are increasingly more interested in free-choice open-ended projects [14].

Course Context

These large-enrollment, open-ended group projects were initially developed in small summer courses with approximately 40 students. This smaller setting allowed for close collaboration between student teams and teaching assistants (TAs), which was essential for refining the framework and developing strategies for scaling the approach to larger enrollments. After establishing this foundation in the summer courses, the model was expanded and implemented in regular semester CS1 courses with much larger enrollments.

The first rollout in a large-enrollment course was in a CS1 class of 450 students. The project was then repeated the following spring in a class of 250 students, followed by two more semesters with enrollments of approximately 400 students in the fall and 200 students in the spring. Since then, this open-ended project model has continued to be used in subsequent semesters and remains in use today. It has been successfully applied across multiple instructors and a wide variety of instructional and TA team configurations.

Our experience spans class sizes from as few as 20 students to as many as 450. We have implemented the projects with a range of instructional team sizes—from a single instructor supported by a handful of undergraduate and graduate TAs, to teams including two instructors and as many as 25 TAs. The success of running these large-enrollment, open-ended projects depends heavily on having sufficient instructional support at the TA level. Adequate interaction between TAs and student groups, coupled with a structured series of milestones and check-ins, helps maintain fairness and consistency in assessment.

The primary goals of our CS1 course are:

1. To develop basic proficiency in the Python programming language, including concepts such as input/output, variables, data types, lists, dictionaries, branching, loops, and user-defined functions.

2. To foster problem-solving skills, enabling students to understand problems, think logically and computationally, and use programming to implement solutions.

3. To expose students to computing topics and technologies, helping them acquire foundational knowledge of the computing field and emerging technologies.

Project Framework Overview

This open-ended group project serves as the final project for the CS1 course, allowing students to apply all programming concepts learned throughout the semester, as well as their problem-solving and computational thinking skills. In most course iterations, students have the option to work independently or with a partner using pair programming.

The design thinking method taught in our class is based on Stanford's d.school, which provides a free starter kit featuring numerous activities [3]. The full design thinking training can be adapted to fit various time frames.

Typically, we dedicate about one class period—approximately 50 minutes—to design thinking. However, it can also be integrated across multiple class periods and combined with other lectures on the design thinking process.

This training takes place before brainstorming begins and serves as a way for partners to get to know each other better. It works best in smaller laboratory sections because it is difficult to ensure both partners' availability in large lecture settings. For this reason, design thinking has primarily been incorporated into our small summer sections. Due to the large enrollment and the structure of TA-led lab sections, integrating design thinking into large lecture courses has proven more challenging.

The d.school's design thinking materials are exceptionally well crafted, but they require a skilled instructor to facilitate effectively. The figure below shows students using the design thinking exercise of drawing each other—an activity 100% authored by Stanford's d.school—intended to emphasize that design is often messy, uncomfortable, and nonlinear, which contrasts with the more structured processes many computer science students are accustomed to.

In most large-enrollment implementations of the open-ended project, we have not integrated the design thinking component.

Project Timeline

The project timeline is crucial for the success of these scaled open-ended projects. To ensure steady student progress and allow effective evaluation, we design multiple check-ins with partners. Below is a week-by-week breakdown of the final project rollout.

Weeks 1–11: During the first eleven weeks of the course, students complete between six and twelve weekly assignments. These assignments focus on developing Python code to solve specific programming prompts and are graded automatically using autograders. This sequence builds foundational programming skills and prepares students for the more open-ended final project.

Week 12 Lab: In week 12, students finish their last mini-project before transitioning to the final project. During the lab session, they are introduced to the final project prompts, which typically offer three thematic options to choose from. Students are paired with partners, usually assigned by the instructor based on similar ability levels and prior observations of effective collaboration. At this stage, students commit either to working with their assigned partner or independently. The lab also includes brainstorming and initial discussions to gauge project interest and ensure partner alignment. Logistical considerations such as availability, including nights and weekends, are discussed to help ensure that partners' schedules are compatible.

Week 13 Lab: The entire lab session in week 13 is devoted to brainstorming. Students are required to answer targeted questions about their brainstorming progress and project ideas. TAs check in with each group to provide guidance, evaluate brainstorming efforts, and help clarify the project direction. Grading for this lab is based on a rubric that assesses whether students have met the required brainstorming milestones (see figure).

Week 14 Lab: In week 14, students meet with their TA and partner to formally propose their project solution. To receive full credit at this stage, students must have completed at least 20% of their code, which typically

includes one working screen and a functional button. Attendance at this lab is mandatory and graded, as it strongly correlates with project success. Students are evaluated on the appropriateness of their project idea relative to the chosen theme, whether the scope fits within the expected three-week development timeline, and their active engagement and progress during lab sessions. Engagement is graded on a scale from zero to two.

Week 15 Lab (80% Draft Check-in): By week 15, students should have approximately 80% of their project completed. Grading criteria at this checkpoint include attendance, the quality and feasibility of the project concept, and the functionality of their code. While some features may still have errors, the base code must run without error. Progress made during lab time is also evaluated, and students who have completed less than 80% of their project receive reduced credit.

Final Submission: The final step of the project is submission via an online form. Students must submit a 3-minute video in which both partners demo their project, explain their work, and show how it meets their chosen prompt. Submissions also include slides. TA managers already have access to the full code and development history. TAs then complete the final rubric section, which is quick since they have tracked progress up to 80% completion. In small sections, students also present live, graded by TAs and peers, while in large sections, only the videos are used.

Sample Project Prompts

Over time, we have developed a variety of project prompts to guide students' final project ideas. In the early semesters, these prompts were intentionally broad. Because projects were built using Python's turtle graphics, which are inherently visual, the initial prompt framed the final project as an art exhibit. Students created either static or animated graphics, and the project was positioned as a semester-ending art exhibit or competition.

As the course evolved, students increasingly wanted to incorporate interactive and event-driven elements. In response, the project prompts shifted toward more application-like designs that explicitly required event-driven programming with turtle graphics. Both approaches have proven effective. The art exhibit prompt works well when there is limited time to cover turtle graphics or when a simpler project is desired, while the event-driven prompts support richer interaction but introduce additional complexity, as turtle graphics is not well suited for large, multi-screen applications. We will describe each.

Art Exhibit: For the art exhibit prompt, the project description was intentionally broad. Students were tasked with creating their own work of art using everything they had learned in Python. The project was open-ended, allowing students the freedom to explore, experiment, and create something they were excited to share. Students were encouraged to express themselves, highlight personal interests or passions, or incorporate other creative or artistic talents.

The art exhibit prompt was particularly successful in smaller summer sections. However, when it was used in a large fall offering with approximately 450 students, the wide range of interpretations revealed limitations in the prompt's scope. In the large lecture setting, some submissions moved beyond artistic graphics, including standard games and projects that appeared to closely resemble widely available online examples. As a result, the prompt was reworked to be more specific for subsequent large-enrollment offerings.

Industry-Sponsored Prompts: The project prompts were developed by the companies themselves and are therefore not included verbatim here; however, we summarize their general focus. We partnered with an

insurance company, a mobile carrier company, and a financial company. The insurance company asked students to design an application to educate young consumers about insurance. The mobile carrier prompt involved creating an interactive training dashboard for employees, exposing students to internal business applications rather than consumer-facing products. The financial company prompt was more narrowly defined and asked students to develop an educational game simulating stock trading based on probabilistic events.

Each prompt presented trade-offs. The first two were more open-ended and resulted in greater diversity among student projects, while the third was more constrained and produced more similar solutions across teams. Student preferences varied: some appreciated the clarity and reduced ambiguity of the more structured prompt, while others preferred the flexibility of the more open-ended options.

Overall, the industry partnership offered several positive experiences and learning opportunities. However, company participation was inconsistent. While some mentors were highly engaged and reliably attended labs and milestones, others were difficult to coordinate with or did not meet agreed-upon expectations. Due to these challenges, the nonprofit and participating companies did not pursue the partnership beyond this single semester.

Community-Sponsored Prompts: Early on, we also used a variation of the final project prompt that partnered with external stakeholders without centering on a specific business. This prompt asked students to design an application addressing food insecurity. Students heard from guest speakers from local food pantries, including the campus food pantry and other community organizations working in this space. These partners served as authentic stakeholders, giving students a clear "customer" to design for while grounding the project in social justice and community impact rather than commercial goals.

This version of the open-ended project was implemented with students who had less programming experience than a typical CS1 cohort, closer to a CS0-level course. Despite this, the prompt proved highly effective and engaging. Its success directly informed and inspired later iterations of project prompts used in the CS1 course.

Instructor-Created Prompts: We have run many iterations of the course using instructor-provided prompts developed by the instructor and TAs. In one example semester (Appendix 3), students chose from four prompts. The first focused on data visualization, where students built applications using provided clean datasets to apply the data science and plotting skills taught in the course. The second prompt asked students to design tools to support international students experiencing homesickness, grounded in issues specific to our campus and city. The third prompt involved creating a highly visual educational tool to teach coding concepts to learners at various levels. The fourth prompt was intentionally open-ended, asking whether technology brings people together or pushes them apart, allowing for a wide range of interpretations.

Across iterations, we vary prompts and aim to include one or two that are socially or community oriented. Informed by student-run hackathons, we also include prompts that encourage students to build applications for themselves or their peers, allowing them to identify a customer within the university community.

CS Student Organization Provided Prompts: Another prompt framework we used involved having CS student organizations contribute project prompts to the CS1 course. Student organization leaders introduced their prompts during lab sessions, which also gave them an opportunity to promote their organizations to CS1 students, many of whom were first-year students. Over the final three weeks of the project, student leaders or senior members visited labs to provide feedback and discuss student ideas, mimicking industry-sponsored projects but with lower logistical overhead.

This approach was easier to implement because student organizations were motivated to recruit new members. The student-created prompts, listed in Appendix 3, included data visualization and fintech prompts from the ACM student chapter, “bridging the gap in technology” and time management prompts from Women in Computer Science, and a cultural identity informational app from Lohana. Among these, the time management and cultural identity prompts were the most popular with students.

This model offered several benefits, including organic promotion of student organizations and increased interaction between first-year and senior students. However, commitment levels varied among student volunteers. Having TAs who were active members of these organizations helped maintain consistency, though it limited the introduction of entirely new external partners.

Finally, Sample Student Work is provided in Appendix V to give a glimpse at some responses to these prompts.

Pair Formation and Support Structure

We have experimented with different approaches to partnerships, including requiring partners versus allowing independent work, and assigned versus self-selected partners. When assigning partners, we have tried various methods. One approach groups students by ability, pairing those with similar performance in the course. Another method involves collecting student responses to questions about their interests, skills, and availability, then matching partners based on these criteria.

We have found that pair programming works best for open-ended CS1 projects. Teams larger than two tend to struggle with coordination, and there is often not enough work to justify larger groups. As a result, we limit projects to pairs or individual work.

Through trial and error, we learned, and confirmed what is known in CS education literature, that students will not naturally pair program unless it is made explicit. In Lab 12, we dedicate time to teaching what pair programming is, why it is required, and how it differs from other approaches. We explicitly prohibit “divide and conquer,” “switch-with-shifts” (where students alternate working in the same IDE without collaborating), and having one student do all the work. We explain that these methods often lead to nonfunctional or error-prone code and emphasize the technical and collaborative benefits of true pair programming.

We also provide practical guidance, including tips for collaborating over Zoom, and teach the driver/navigator approach. Experience over 10 semesters shows that students who genuinely work together are far more productive, make fewer errors, and have a better learning experience. Small sections with more contact hours allow us to observe pair programming in action, and it is evident that well-coordinated pairs make much faster progress.

In large-enrollment semesters (200–450+ students), offering the option to work individually is essential, as managing partner conflicts at that scale becomes unmanageable. During the Week 12 lab, students work with an assigned partner and then must commit either to continuing with that partner or to working independently, with no option to switch later. This commitment process significantly reduces downstream complaints, though some issues still arise. Students do submit partner evaluations with their final submission, which helps keep both partners accountable to their work, as other work has shown [5].

In smaller summer sections (fewer than 40 students), students are typically required to work in pairs. The smaller class size and more frequent interaction make it easier to support teams and address partnership issues as they occur.

A key factor in successfully scaling these open-ended projects is assigning each team a dedicated TA who acts as a project manager. Lab sections typically include 20–30 students and are staffed by one graduate TA and two undergraduate TAs, allowing each TA to manage approximately 4–8 project groups. Nearly all rubric-based assessment occurs during lab sessions, eliminating the need for additional grading hours.

With this structure, each group receives 5–10 minutes of focused feedback per lab. In practice, the number of active groups per TA is closer to four due to late-semester attrition. Groups are assigned only after the drop deadline, and students who disengage after that point are evenly distributed to TAs and also grouped together as pairs to minimize their impact on active teams.

Requiring students to meet with their assigned TA three times during the semester increases accountability and supports steady progress. During these check-ins, TAs review code, require students to run their applications, and provide feedback on both design and programming challenges. These meetings also allow TAs to identify concerns early, such as repeated absences or lack of good-faith effort, and escalate them to the instructor when necessary.

Assessment and Grading Approach

Appendix I provides the full project rubric, aligned with the project timeline. This rubric is shared with students to ensure transparency and clearly show how to earn full credit. We are explicit about how the project will be graded on technical criteria. While this may seem overly detailed, we have found that specifying low-level coding requirements helps students understand exactly what is expected. For example, Appendix IV lists the minimum technical requirements, such as ensuring the code is more complex than their three prior semester projects and including a certain number of loops, if statements, and user-defined functions. These benchmarks give students a clear way to gauge whether their open-ended project meets the expectations.

We emphasize that students should satisfy these requirements “without a doubt,” which helps avoid ambiguity in pre-grading questions (e.g., whether a single if statement branch counts). This also makes grading more consistent, as teams and TAs can straightforwardly verify each requirement. Additionally, we provide a repository of sample projects, showing both projects that meet the minimum requirements and those selected for the showcase, to guide and inspire students.

Another key aspect of our grading approach is that TA managers do not make subjective grading decisions; they simply complete a detailed rubric, which is shared with students for full transparency. This ensures consistent and fair grading, even across both graduate and undergraduate TAs. Rather than evaluating the projects themselves, TAs focus on applying the rubric consistently. In a later section, we show that this process produces an accurate and equitable grade distribution compared to more traditional grading methods.

Implementation at Scale

Successfully implementing open-ended CS1 projects at scale depends on having sufficient TAs and small lab sections for regular student meetings. With these components, most grading occurs during lab time, with only a minimal amount required after the final submission. TAs report that this post-submission grading is quick and

easy, as they are already engaged with their teams and familiar with the code. The final project videos are only three minutes long, and grading typically takes 5–10 minutes per project using the rubric. Many TAs describe this as their favorite and most rewarding part of the course, as they enjoy seeing the students' final creations.

The instructor's workload for the final project differs from a traditional autograded CS1 project. Planning requires working well ahead of deadlines to train and remind TAs about check-ins and milestones, but this largely replaces the usual prep for each lab section. Coordinating with external partners adds complexity and moving parts, though it provides the benefit of a real "customer" for the projects. Using instructor- or TA-created prompts is much less time-intensive. A key factor in managing workload at scale is having a TA team with prior experience in the open-ended project, either as former students or previous TAs, which significantly reduces the effort required. However, most other work points to integrating projects like this increasing teacher workload [6, 7].

Showcase

To encourage creativity and experimentation beyond the minimum requirements, we run a showcase for the top final project submissions. Eligibility for the showcase requires students to complete their projects by the week 15 lab, which typically includes about 20 to 30% of the class, which can be over 100 submissions in large sections. TA managers review submissions throughout the semester and flag potential showcase candidates, allowing the instructor to focus on a smaller subset for final selection, if needed.

The incentive for students is significant: their lowest exam score is dropped from the final grade calculation. This encourages students who excel in project work and especially those who struggled with exams.

Showcase submissions go well beyond the minimum project requirements. While a basic project might include a simple multi-screen graphics application with minimal interaction, showcase winners often demonstrate:

- **Artistic creativity**: incorporating hand-drawn or custom graphics.
- **Unique interpretation**: approaching the prompt in a novel or unexpected way.
- **Technical innovation**: exploring Python and Turtle Graphics beyond what is required, creating sophisticated, interactive, or realistic-looking applications.

Despite Turtle Graphics being a simple API that students learn in only about two weeks, students have created applications that appear highly polished and professional. The showcase allows us to highlight submissions that stand out for creativity, originality, and technical skill, rewarding students who push beyond the baseline expectations.

Evaluation and Student Feedback

One common question from prospective instructors is how grading distributions for open-ended CS1 projects compare to traditional projects and exams. Analysis of both large and small enrollment sections shows that final project grade distributions resemble those of standard projects. For example, a typical mid-semester project using loops has a mean score around 70%, a median score near 90%, and a high standard deviation (~40%). In contrast, the final project mean score is about 75%, with a medianscore around 90% and a slightly lower standard deviation (~30%) due to frequent check-ins and milestones, which reduce extremes of performance.

Compared to exams, final project mean and median scores are roughly 10% higher. Final project mean, median, and standard deviation are more similar to later exam statistics (e.g. midterm exams 2 and 3).

Correlation analyses show that final project scores align closely with overall course performance, despite being only 10% of the final grade, and correlate more strongly with later exams and projects than earlier ones. This suggests that the skills demonstrated in the final project reflect broader competencies developed throughout the course.

Comparing large and small enrollment sections reveals notable differences. In large sections (200–450 students), final project distributions mirror other high-enrollment grade items, with wider variation due to differences in student engagement. In small sections (≤40 students), averages and medians are 10–20% higher than the large enrollment classes, and standard deviations are much lower (10–15 points vs. ~30 points), reflecting near-universal completion and baseline competency. Observations of small-section presentations show almost all students complete and present their projects, whereas large-section presentations often reveal incomplete participation.

Overall, open-ended final projects are effective at scale, enabling most students to engage meaningfully, but they achieve the highest participation and consistency in small sections.

Student and TA feedback has consistently been very positive about the open-ended final projects.

Reflections on Influence of Generative AI Tools

We analyzed final project scores across multiple semesters, both before and after the widespread availability of generative AI tools, and found no statistically significant differences. Observationally, the quality and type of final projects have also remained consistent. In the early semesters, generative AI tools were not widely used and were ineffective for Turtle Graphics. In later semesters, students occasionally use AI to assist in generating Turtle Graphics code, but this requires crafting their own prompts due to the open-ended nature of the project.

Misuse of generative AI is minimal compared to traditional, highly specified, auto-graded assignments. The required check-ins prevent students from relying on a single AI-generated solution. Additionally, generating images, which are an integral part of these projects, still requires significant manual effort. While students sometimes start from AI-generated code, it often becomes messy and difficult to debug, encouraging them to rely on in-class templates and lab exercises. Consequently, most final projects use only the subset of Turtle Graphics content taught in the course.

Generative AI is most commonly used during the brainstorming stage. Students occasionally produce similar ideas in response to prompts, such as creating a “day in the life” game for the time management prompt. When this happens, instructors use check-ins to guide students toward more creative approaches. This tendency is not unique to to our class, even in hackathons, multiple teams often converge on similar solutions, so unique or unconventional projects naturally stand out. It is eye opening for students to see how many similar solutions there are.

Overall, generative A tools appear to have little impact on project outcomes.

Conclusions and Recommendations for Instructors

We highly recommend that instructors consider open-ended final projects for CS1. These projects engage students in coding early, provide meaningful practice even in the era of generative AI tools, and serve as a strong alternative or complement to highly specified traditional projects. While we do not have formal quantitative evidence, over 10 semesters of running this course, our observations suggest that this open-ended framework is highly effective and has been successful across multiple instructors.

There are differences in implementing these projects in large versus small sections. Small sections tend to achieve higher overall participation and engagement, but even at scale, the projects are logistically manageable. Success at scale requires weekly lab sections with roughly one TA per 10 students. This aligns with the emphasis on the importance of TAs found by other work in PBL [13]. Frequent milestones, check-ins, and a clear rubric maintain grading consistency, even with many TAs, provided the straightforward nature of the rubric. Grading and rubric transparency with the students also helps with fairness.

Key lessons from running this model include:

1. **Varied, semester-specific prompts** encourage creative, unique solutions and minimize code reuse semester to semester or overreliance on generative AI.

2. **Pair programming and optional individual work** are essential. In large sections, mandatory partners create unmanageable conflicts, so students must have the option to work alone or in pairs. In small section, required pair programming works well.

3. **Incentives for going above and beyond** such as early submission deadlines or project showcases with additional rewards motivate students to exceed baseline technical requirements. When the showcase is not offered, student submissions tend to be less creative.

Overall, with clear structure, TA support, and thoughtful incentives, open-ended final projects provide a scalable, engaging, and effective learning experience for CS1 students.

References

[1]. E. O. Barros, J. Hauck, and L. Gery, "Engaging CS1 Students with Project-Based Learning," in *Proc. IEEE Frontiers in Education Conf.*, San Jose, CA, Oct. 2018, pp. 1–8. [Online]. Available: https://dl.acm.org/doi/10.1109/FIE.2018.8659242

[2]. A. D. Suleiman, D. Hou, Y. Liu, J. E. DeWaters, D. C. Shepherd, and J. G. de Souza, "Systematic Literature Review on Project-Based Learning in Computing Education," *ACM Trans. Comput. Educ.*, vol. 25, no. 4, pp. 1–53, Oct. 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3743684

[3]. Stanford d.school, "An Introduction to Design Thinking PROCESS GUIDE," [Online]. Available: https://dschool.stanford.edu/resources/design-thinking-process-guide

[4]. S. Sharmin, D. Zingaro, L. Zhang, and C. Brett, "Impact of open-ended assignments on student self-efficacy in CS1," in *Proceedings of the ACM Conference on Global Computing Education (CompEd '19)*, Chengdu, Sichuan, China, 2019, pp. 215–221, doi: 10.1145/3300115.3309532.

[5]. M. Souza, R. Moreira, and E. Figueiredo, “Students’ perception on the use of project-based learning in software engineering education,” in *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, 2019.

[6]. I. Bosnić, F. Ciccozzi, I. Crnković, I. Čavrak, E. Di Nitto, R. Mirandola, and M. Žagar, “Managing diversity in distributed software development education—A longitudinal case study,” *ACM Trans. Comput. Educ.*, vol. 19, no. 2, Art. no. 10, pp. 1–23, Jun. 2019, doi: 10.1145/3218310.

[7]. A. García-Holgado, A. Vázquez-Ingelmo, F. J. García-Peñalvo, and M. R. Conde, “Improvement of learning outcomes in software engineering: Active methodologies supported through the virtual campus,” *IEEE Revista Iberoamericana de Tecnologías del Aprendizaje*, vol. 16, no. 2, pp. 143–153, May 2021, doi: 10.1109/RITA.2021.3089926.

[8]. S. M. Souza and R. A. Bittencourt, “Sentiments and performance in an introductory programming course based on PBL,” in *Proc. IEEE Global Engineering Education Conf. (EDUCON)*, Vienna, Austria, 2021, pp. 831–840, doi: 10.1109/EDUCON46332.2021.9453963.

[9]. Z. Song, N. Shah, J. Guo, and Q. Zhu, “Applying project-based learning to improve computer networks courses: An experience report,” in *Proc. IEEE Global Engineering Education Conf. (EDUCON)*, Tunis, Tunisia, 2022, pp. 148–156, doi: 10.1109/EDUCON52537.2022.9766750.

[10]. G. Tewolde, “Student-inspired project-based learning in an embedded systems course,” in *Proc. IEEE Integrated STEM Education Conf. (ISEC)*, Princeton, NJ, USA, 2020, pp. 1–5, doi: 10.1109/ISEC49744.2020.9280656.

[11]. M. Vidoni, J. M. Montagna, and A. Vecchietti, “Project and team-based strategies for teaching software architecture,” *International Journal of Engineering Education*, vol. 34, no. 5, pp. 1701–1708, 2018.

[12]. M. Vijayalakshmi and M. M. Raikar, “Development of network applications and services through project-based learning to meet 21st century skills,” in *Proceedings of the 2021 IEEE Global Engineering Education Conference (EDUCON)*, Vienna, Austria, Apr. 2021, pp. 1167–1174, doi: 10.1109/EDUCON46332.2021.9454133.

[13]. T. Andernach and G. N. Saunders-Smits, “The use of teaching assistants in project-based learning at aerospace engineering,” in *Proceedings of the 36th Annual Frontiers in Education Conference (FIE 2006)*, San Diego, CA, USA, Oct. 2006, pp. 11–14, doi: 10.1109/FIE.2006.322468.

[14]. K. A. Dunnigan, J. Bringardner, and G. W. Georgi, “From pre-defined to open-ended projects: Evaluating first-year ability to innovate and problem solve,” in *2019 ASEE Annual Conference & Exposition*, Tampa, FL, USA, Jun. 2019, doi: 10.18260/1-2--32866.

Appendix I - Open-Ended Project Rubric

Week 13 - Brainstorming		
Rubric	**Points**	**Scoring / Notes**
Attendance	2	0 = absent, 1 = late or left early, 2 = present and submitted form
Prompt	2	Group must select one of the prompts.
Logistics	2	0 = no plans, 1 = very little, 2 = discussed schedule, timing, goals, availability, etc.
Brainstorming	2	0 = no progress, 1 = very little progress, 2 = has come up with several ideas or one/two well-developed ideas
Comments		Please describe briefly what the group/individual's idea for their project
Week 14 - Project Proposal		
Attendance	2	0 = absent, 1 = late or left early / no form, 2 = present and submitted form
Theme	2	0 = no direction/theme, 1 = not well thought out, 2 = well thought out
Started Coding?	2	0 = no, 1 = very little, 2 = has started and ~20% complete
Scope	2	0 = too much/too little, 1 = can't tell, 2 = well scoped
Progress in lab	2	0 = did no work, 1 = not productive, 2 = worked all lab
Week 15 - Project Draft		
Attendance	2	0 = absent, 1 = late or left early / no form, 2 = present and submitted form
Theme	2	0-no direction/theme; 1 - not well thought out; 2 - well thought out
Code Runs	2	0 - Code does not run without error; 1 - runs but minimal graphics; 2 - runs and has graphics
Code Progress	2	0-less than 50% 1-coded; 50-80% done; 2-80%+ done
Progress in lab	2	0-did no work; 1-not productive; 2-worked all lab or done
Showcase	2	0-not eligible; 2 - should be considered for showcase.
Final Submission		
Slides	2	0-no slide or no access; 1-not 2, unacceptable quality, does not meeting requirements; 2 - meets all slide requirements
Video	2	0-no video; 1-does not meet requirements (not 3 min, no demo, no presentation, bad quality, face of both partners does not appear in video), 2 - meets requirements
Code Requirements	10	Enter 0-10 10 for meeting first 10 requirements; -1 for each missing;
Theme	5	Enter 1-5: 1 - weak, 2 - below average, 3 - average, 4 - above average. 5 - strong
Partner Evaluation	5	Partner's evaluation to "how likely would I agree to working with this partner again: 5 - very likely; 1 - very unlikely

Appendix II - Instructions for TAs by Week

Who	Task
Week 12	**PARTNER ASSIGNMENT WEEK**
Head TAs	1. Assign Lab 12 partners. You should pair students by ability and who will work well together. This is our proposed partnering for the Final Projects.
Head TAs	2. Review Lab 12 submission and review if students want to work individually or with their partner. Update the Grading tab in this document so that all final project groups are labelled. If any student misses the lab (and does not notify you), they will work alone. If they notify you, you can coordinate with them on their preferences.
Head TAs	3. Split up the groups for each of your lab section so that each one is assigned to one of the three TAs. Put assigned TA in the Grading tab. Notify your TAs once you have done so.
Week 13	**BRAINSTORMING WEEK**
Head TAs	1. Present each prompt
Head TAs	2. Present on Pair Programming and present Rubrics for grading -- see slides.
All TAs	3. Students should brainstorm and submit their ideas. They should "choose" a prompt (although they can switch later).
All TAs	4. Each TA should meet with each of their assigned groups and talk to them for at least 5-10 minutes. Discuss (and make sure they have discussed): how they will communicate, their timeline, their schedules on when they can/want to work, whether or not they plan to submit early and to the show case, etc.
Week 14	**PROPOSAL/ROUGH DRAFT WEEK**
All TAs	1. Individually meet with each of your assigned groups during lab and have them present their final project proposal. Make sure to schedule it out so each group gets equal amount of time with you.
All TAs	2. Fill out the grading for Week 14 in the Grading tab for each of your groups. Do this while in lab and discussing with them.
Week 15	**MOSTLY FINAL DRAFT WEEK**
All TAs	1. Get all your assigned groups together (if too large, please split). Each group should demo their draft to the rest of the groups.
All TAs	2. Fill out the grading for Week 15 in the Grading tab for each of your groups. Do this in the lab while watching the demo. If you need to split groups, you might need to ask some groups to show you a second time.
All TAs	3. Encourage students to share aspects about each project that they found interesting or impressive. Encourage them to ask "how did you do xyz"
All Tas	4. With any time left, students should continue to work on the projects. If they are done, they can leave.
All TAs	5. Make sure you know which groups are submitting to be considered for the Showcase. They must submit their final slides, code, video by Wed. at 11:59 pm to be considered. Please also mark their names in the Grading tab so we don't miss anyone.
Ugrad TAs	6. We prefer for you to fill out the rubrics for your final projects for your pod during final weeks. However, if that is not possible, please notify your grad TA ASAP so they can complete the last stage for you.
Final Exam Week	
All TAs	1. Using the "Form Responses 1" tab, please review each of your groups final submissions once they have submitted. They are due Sunday at midnight, but you can review on a rolling basis.
Ugrad TAs	2. If you prefer not to work during final exam week, please let your grad TA know and they will finish the final grading. We prefer to split it so that each student group gets a more fresh evaluation. However, we respect your choice not to grade during final exam week. Please do not claim hours if you are opting out.

Appendix III - Sample Prompts

Prompt 1 - Data Visualization

We live in a time of tremendous access to data, but more data doesn't necessarily equate to more useful *information*. With our programming skills, we can help users turn data *into* information by building tools to let users summarize, filter and visualize data.

We've collected some datasets that you could use for your project, with data files on health, finance, crime in the city, politics, and more. You can download any of these, then upload them into your ZyBooks IDE.

Your project should let users interact with data through a combination of visualization, filtering, statistical calculations or more. Incorporate animations to make the data more engaging and easier to understand.

You can find the data files at: <redacted>

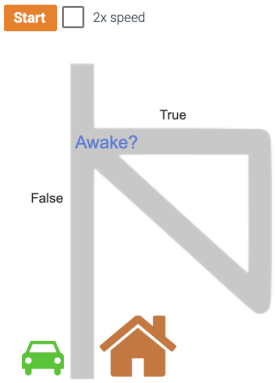
Prompt 2 - Diversity at <Redacted-School> and <Redacted-City>

<Redacted> has a diverse student population, including 12% international students as of Fall 2023. The top areas of study for international students at <Redacted> are STEM, English language acquisition, and business. The top five sending countries include India, China, Pakistan, South Korea, and Vietnam. The City of <Redacted> and the <Redacted> area make up a diverse community. Design an app or graphic to do one of two things (either/or):

1. Help international students adjust to the culture of <Redacted> and/or <Redacted>. We recommend focusing on a particular sending country, culture, or group. Regardless of whether or not you identify with the group you have selected, we recommend that you do some research, interviews, etc. before solutioning.
2. Informational approach that brings awareness or education to the diversity of people at <Redacted> or in <Redacted>. For example, take a look at <Redacted> 1979 graphic of <Redacted>. Remember that your project must use events and animation, so this is only an example. This is a broad theme, you may be as specific or general as you'd like.

Prompt 3 - Teaching Coding Concepts using Graphics

Design at least three lessons that use graphics to explain coding concepts to students. The three lessons should be graphical in nature using events and animation. Themes of the lessons are your choice but some ideas: variables, branching, loops, functions, an algorithm, sorting, etc. Consider exploring ideas of how these ideas are visually presented in Zybooks or Code.org. As just one example, consider how Zybooks introduced loops in section 5.1:

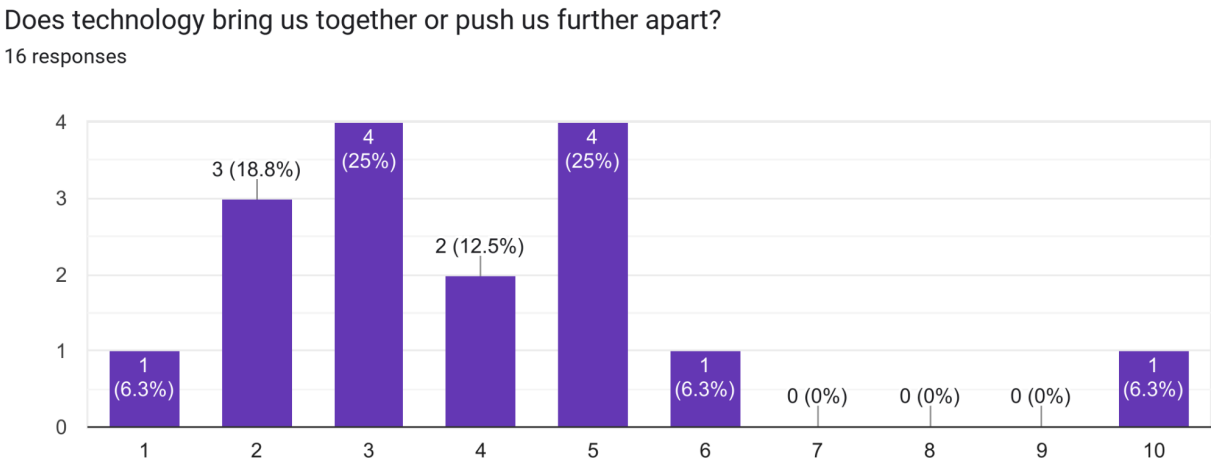


The audience can be your choice: children (elementary school), middle school students, high school students, or college students. We encourage you to be creative and/or game-ify.

Prompt 4 - Technology?

Does technology bring people together or push us apart?

Vote here and see your classmates responses here:



1 - together, 10-apart.

This theme is very open. After thinking about this question (and perhaps researching it), you can brainstorm a final project graphic/application.

Association for Computing Machinery (ACM) Prompt

1. **Data Visualization** - Work on a project that visualizes data related to a social issue over the years. This could be crime rates in a city, population growth rates, or school/college dropout rates. Incorporate animations to make the data more engaging and easier to understand.

2. **FinTech Financial Advisor-** Create a financial advisor app that can track your monthly expenses and provides you with a spending recommendation for the future to reach you to a financial goal (some amount of money in the future).

Women in Computer Science (WiCs) Prompt

3. **Bridge the Gap in Technology** - Create an application or graphic that will include information about different resources for young students to learn about bridging the gap in technology

   Features:
   - Display conferences events, or opportunities happening about CS or technology can be in or out of <Redacted>
   - Display other resources on upskilling

4. **Time management** - Create an interactive game of the life of a CS major where each decision the student makes leads to a different outcome in their life.

   Features:
   - Make it an interactive game
   - At each step, give different options, where each option leads to a different outcome
   - Should reflect upon time management

Latinx Organization for Growth in Computing and Academics - LOGiCA Prompt

5. **Cultural Identity Informative App -** Develop an application or graphic that educates users about a cultural identity of choice, for example: a nationality, ethnicity, tradition, or any other cultural category.

   The app should include at least 2 tabs. Specifically, 1 of type A and 1 of type B:

- **Type A:**
  - **An informational tab** with text and images relevant to the identity of choice. For example: basic information, interesting facts, history of the culture, or recent news centered around the culture.
  - **A graphical tab** where a meaningful visual piece is the main content, preferably based on Turtle Graphics. For example: a piece of art made with Turtle graphics, a Turtle animation, a map, or symbols and patterns that are representative.
- **Type B:**
  - **An interactive tab** with more complex functionality that closely involves user interaction. For example: a mini game, a trivia/questionnaire, or a translator for the language of the culture.

   You should choose a culture that at least one team member identifies with. One partner should move forward as an application expert (or client) and the other should be asking lots of questions to further understand how the app could best serve the client. If neither identifies as the client (which we generally would not recommend), we expect that you do some research to discuss how to serve the client.

   The goal is to share parts of your chosen cultural identity with others, to inspire or educate about our world's diversity!

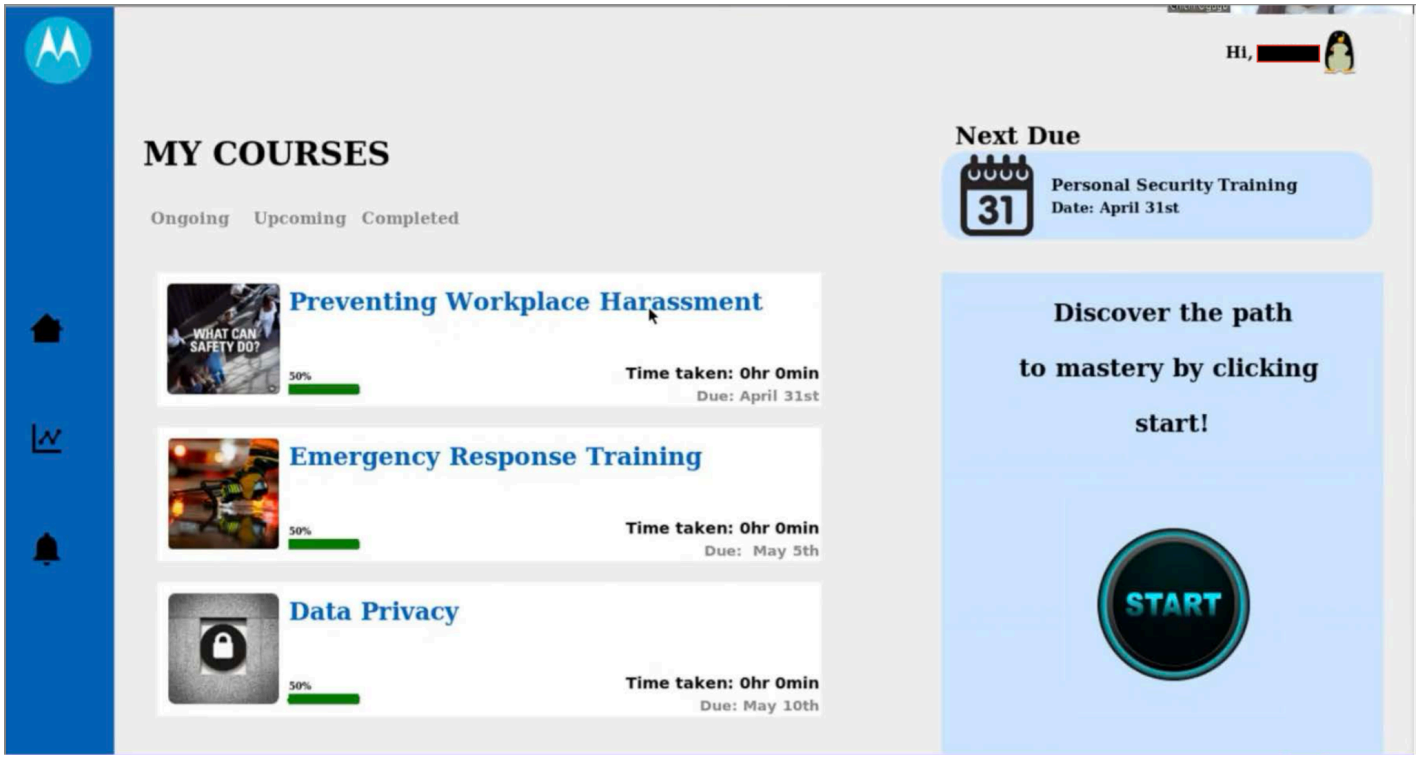
Appendix IV - Minimum Technical Requirements

To receive full credit for your project, it should be more complex than Project 3, 4 or 5 and twice the length of any of those three. If you turn in a project that has minimal lines of code, minimal functionality, and could be completed in a very short amount of time, you will not receive much credit for your work. This explanation alone should be sufficient to describe the requirements. However, in order to be extra clear, here is a detailed list of minimum requirements that can help guide your project, the project must include a code:

1. more complexity than Projects 3, 4, and 5 (individually, not the three together).
2. that contains two or more loops.
3. that contains two or more if statements.
4. that contains four or more user-defined functions and calls or uses those functions in a repeated way.
5. that imports data from a file and uses it in the final product. You should choose or create the data.
6. that uses five or more built in Python functions (turtle functions do not count).
7. that is animated (not looking for a still graphic). The animation should include drawing and moving turtles.
8. that contains at least two event-driven components.
9. that runs in Zybooks IDE with Turtle graphics. Code must run. If code has errors and does not run, no credit will be given. Development must be entirely in Zybooks. Code playback should reveal entire code development progress. Developing in an outside IDE is not permitted. You may not use advanced concepts not covered in this class. If you use it, you should be able to explain your code.
10. that responds to one of the three prompts and is a high quality theme.

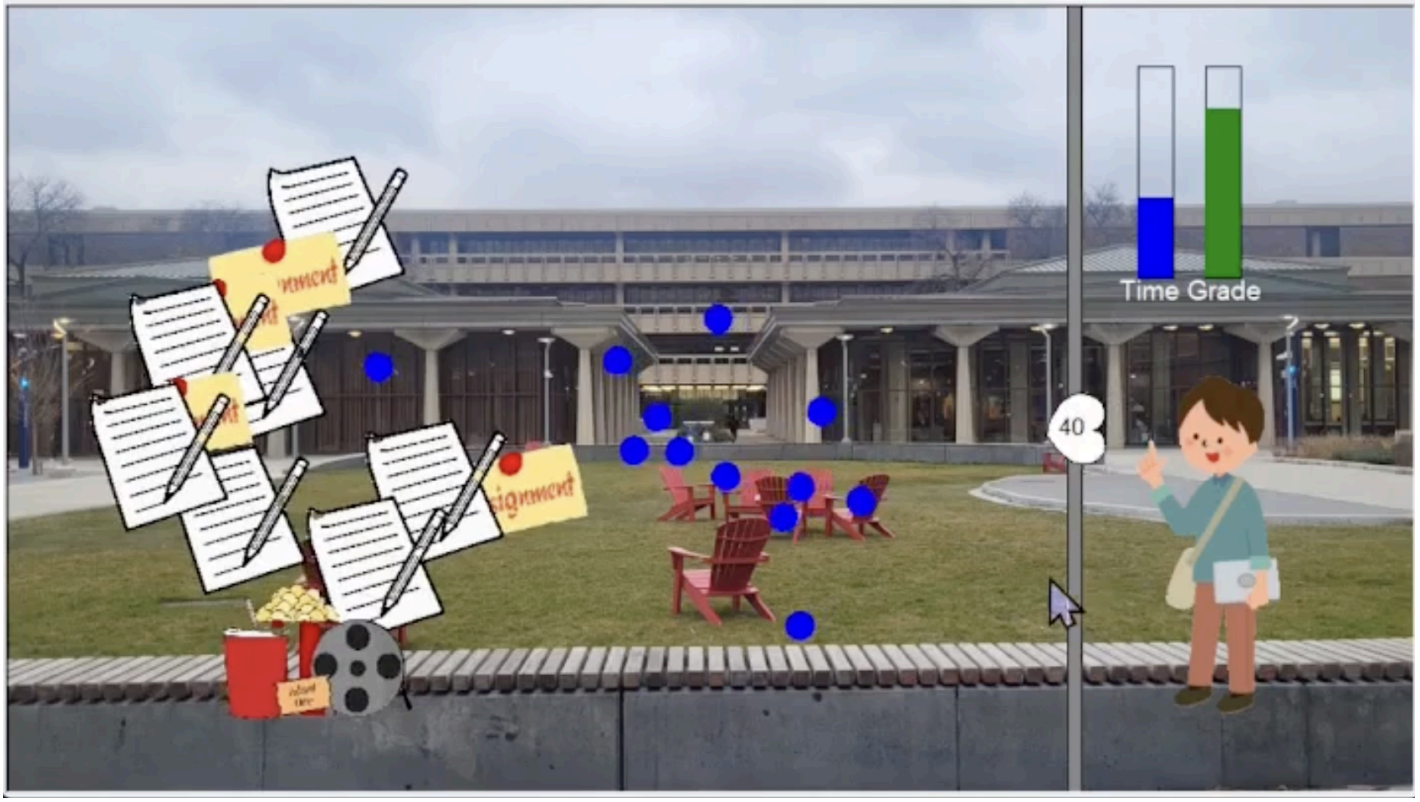
With all the requirements above, they must contribute to the functionality of your project. If removing them would not change the final product, then they don't count. If you are trying to minimize or "minimally satisfy" the requirements, your project is not sufficient. For example, if you ask "Do two branches of the same if statement count for requirement 3?", the answer is "No". These are a list of minimum requirements and your goal should be to turn in a final product that without a doubt meets them. Do not ask us if your project satisfies them, it is your job to convince us that it does. If you aren't sure, then it does not meet requirements. Most students who struggle to find enough branches, Python functions, etc. are usually "hard coding" or have poor coding style.

Appendix V - Sample Student Work

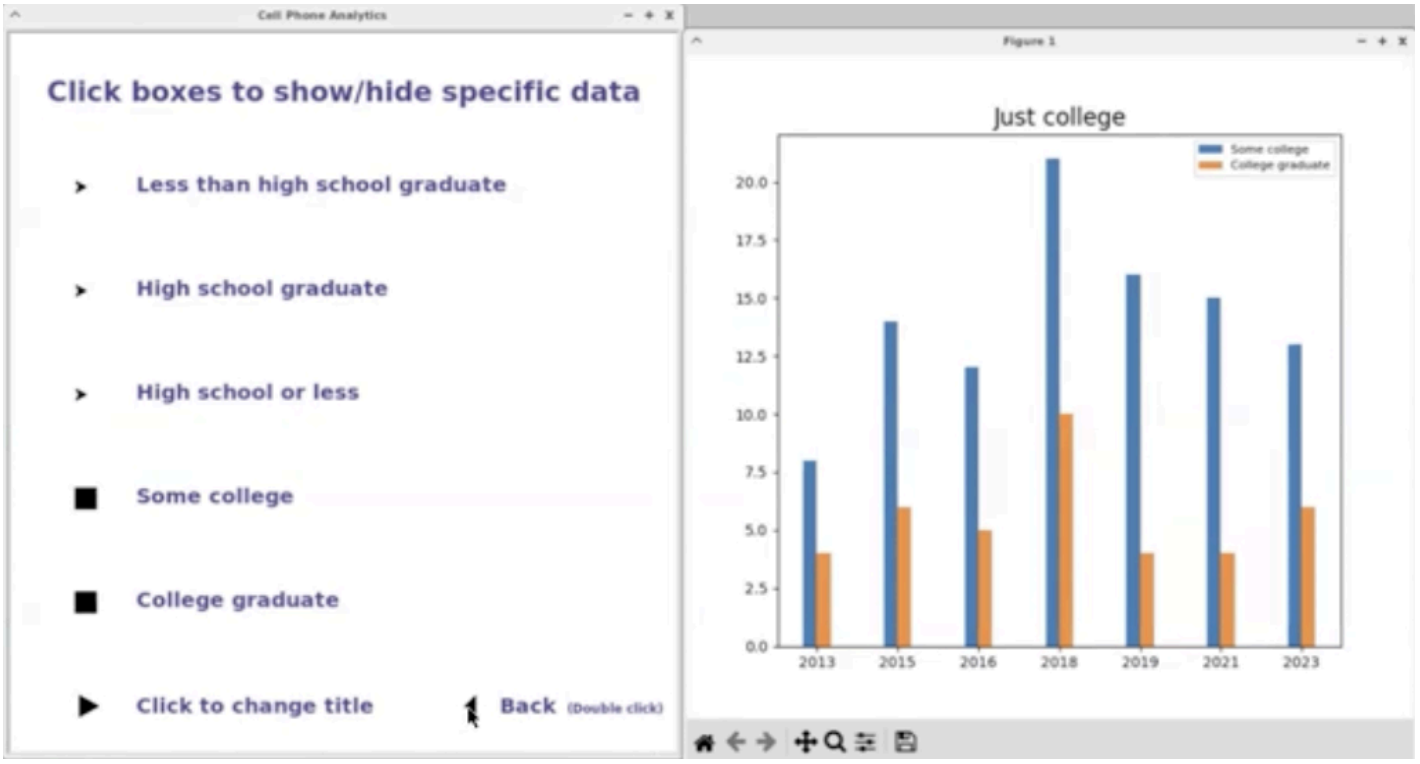


A student's response to the industry sponsored prompt to build an application for employee training. This is an example of students using Turtle Graphics to produce a very professional looking application.

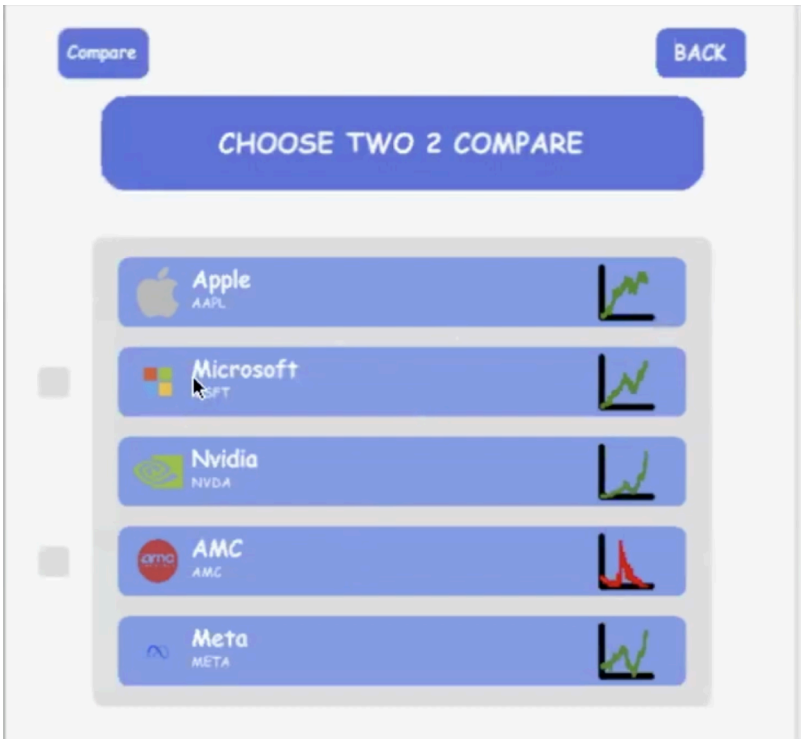
A student's response to a community focused prompt at educating city youth. This team designed a module to teach city youth about the importance of vaccinating dogs. There were many pages of education material, quizzes, and interact games. All images and graphics were hand drawn.



A student's response to the time management prompt. They created a game where you get points for finishing assignments but your mental health goes up when you take a break (watch a movie).

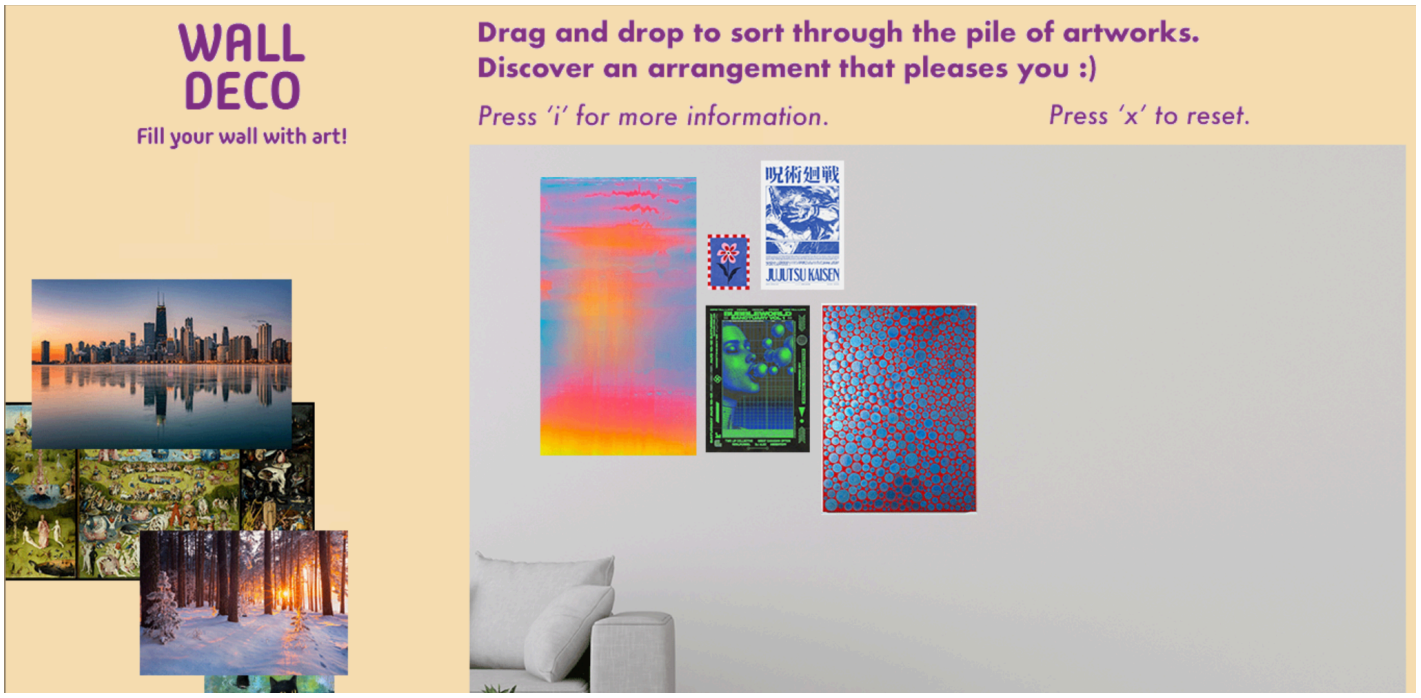


An example of a Data Visualization project. Students created a tool that could plot data live via various toggles and button pushes. Here they are showing cell photo data usage over time for people with some college vs people with college degrees.



Here is a UI designed by a student to compare stocks between multiple tech companies in response to a FinTech industry sponsored prompt..

This is a student's response to our generic "Art Exhibit" prompt. This student used Turtle Graphics to draw everything on the screen. The vehicle moves across the screen at a random speed and the "Your Speed" sign displays its speed.



This student responded to a prompt “Design a <Fill in the Blank>”. Their solution was to build a tool that helped designers create an Art Deco Wal.

This is a response to the “Bridging the Gap in Technology” prompt. Users interact with the old Nokia phone by pushing the buttons which the students have programmed to respond to clicks. The text on the phone is Turtle Graphics!