2026 ASEE ANNUAL CONFERENCE June 21–24, 2026 | Charlotte Convention Center, NC
Host Sponsor UNC Charlotte
ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION
Paper ID #52508

Measuring the Effects of Open-Ended Programming Project Tasks in a Programming Practicum Course

Dr. Scott J. Reckinger, University of Illinois at Chicago

Scott J. Reckinger is a Clinical Associate Professor in the Department of Computer Science at the University of Illinois Chicago. He earned a PhD in Mechanical Engineering at the University of Colorado Boulder. His research areas within computational fluid dynamics include numerical methods for simulations of fluid instabilities, turbulence modeling, and parameterizations for ocean models. He has also published papers in computer science and engineering education, specifically on oral proficiency exams (both synchronous and asynchronous), two-stage exams, project-based learning, and active learning methods.

Levell Shelomi Kensey, The University of Illinois at Chicago
Miss Han Le Gia Dang, The University of Illinois at Chicago

Measuring the Effects of Open-Ended Programming Project Tasks in a Programming Practicum Course

Scott J. Reckinger[†], Levell S. Kensey[†], Han L.G. Dang[†]
[†]Department of Computer Science, University of Illinois at Chicago

ABSTRACT

Programming projects have long been a cornerstone of core CS courses, engaging students in higher-order cognitive skills such as applying concepts, analyzing algorithmic approaches, and evaluating program outputs. The increased use of autograders is beneficial for providing rapid feedback and grading efficiency, but often necessitates fully-specified, highly-scaffolded project designs, which may constrain opportunities for open-ended problem solving and independent decision-making. These concerns are amplified by widespread availability of generative AI (genAI) tools, which can often produce solutions quickly. In order to investigate the impact of incorporating open-ended components into otherwise fully-specified programming projects in a second-year programming course, two counterbalanced programming projects, centered on word puzzle games, namely Spelling Bee and Word Ladder, are used to compare student experiences, learning gains, and independence. Data collection instruments focus on student perceptions, concept proficiency, self-efficacy, belongingness, and development/submission behavior. Results show that students prefer fully-specified project tasks, open-ended project tasks are more stressful, but open-ended project tasks lead to a stronger sense of accomplishment. Participants self-report that they tended to utilize genAI tools for idea generation and debugging during project development. Student feedback on perceptions, AI usage, and project experience give more insight to the trends found within the data.

INTRODUCTION

While autograders offer significant value to students and instructors, such as timely, consistent feedback and reduced grading workload, they can also put limits on the educational benefits of tackling a programming project, which may be exasperated with wide access to modern generative AI (genAI) tools. Highly-scaffolded, fully-specified tasks often limit opportunities for open-ended problem solving, design decision-making, and independent testing/debugging, all of which are critical for fostering higher-order cognitive skills. When students focus primarily on passing test cases, they may engage less deeply with the underlying concepts, analysis of alternative approaches, or evaluation of their own design choices. As a result, the development of general problem-solving abilities, critical thinking, and independence skills may be constrained unless autograded projects are intentionally complemented with activities that encourage reflection, exploration, and self-directed learning. This study involves the design, implementation, and results of incorporating open-ended programming project tasks in a second-year programming (CS2) course.

BACKGROUND

Over the past several decades, programming projects have served as a central pedagogical tool in core programming courses, providing a workspace for students to put concepts into practice. Programming projects move learners beyond basic proficiency by engaging them at higher levels of cognition. Through designing, implementing, and testing programs, students apply course

concepts to solve complex problems, analyze trade-offs among alternative designs, and evaluate the correctness, efficiency, and maintainability of their solutions. Sustained emphasis on project-based learning can help cultivate deeper conceptual understanding, critical thinking, and professional skills aligned with real-world software development.

When designing programming projects, instructors regularly develop an autograded test suite to provide immediate feedback to the student, while streamlining the grading process. The implementation of an autograder typically requires the project to be designed with fully-specified programming tasks, with starter code that is highly-scaffolded and modularized, and where function headers are provided. Due to these limitations, the impact of autograders on the pedagogical value of programming projects is varied in the eyes of CS educators [1]. A previous study focused on the evolution of autograder usage over the past decade from the pre- to post-pandemic era, noting a recent decrease in the quantity of student submissions to the autograder, which may be an effect of modern advancements in, popularity of, and wide-spread access to genAI tools [2]. A primary concern is that the educational benefits of completing a programming project may be lost when students rely on genAI tools to produce solutions.

One potential approach to mitigating concerns about genAI overuse is to shift from fully-specified, highly-scaffolded programming tasks toward more open-ended projects that require greater student initiative and decision-making. A previous study examined the effects of including open-ended tasks in programming projects implemented in a CS1 course [3]. Investigation of the role of open-ended assignments in CS2, particularly considering recent developments of genAI tools, is limited.

RESEARCH QUESTIONS

The primary aim of this study is to develop interventions that enhance student higher-order cognitive skill development by incorporating open-ended programming project tasks that require problem formulation, design decision-making, and independent testing and debugging. The goal is to address the limitations of highly-scaffolded, autograded programming assignments while shifting student focus from merely passing test cases toward analyzing alternative approaches, evaluating design choices, and reflecting on their solutions. Efficiency benefits provided by autograders must be balanced by assignments that support exploration and self-directed learning.

The three research questions of this study are as follows

(1) What are the impacts of open-ended tasks on student confidence with course-specific programming concepts?
(2) How do student perceptions of open-ended tasks vs. fully-specified/scaffolded tasks differ, and what are the major contributors to those perceptions?
(3) How are genAI tools used, if at all, to tackle the open-ended tasks?

To address these research questions, students enrolled in a second-year programming (CS2) course completed both open-ended and fully-specified programming project tasks and then completed a survey near the end of the term.

INTERVENTION DESIGN – OPEN-ENDED PROGRAMMING TASKS

For this study, open-ended components are incorporated into an otherwise fully-specified programming project in a second-year programming (CS2) core course, called *Programming Practicum*, that is designed to bridge the introductory programming concepts in CS1 and the

more advanced topics covered in Data Structures and Systems Programming courses. The course also introduces students to many useful software development tools, such as working in a UNIX terminal, writing/using makefiles, continual use of debugging tools/utilities/environments (e.g. valgrind, gdb, VScode, etc.), and a system timing utility. A primary course goal is to improve software development skills, such as diagramming, outlining code, testing, debugging, and optimization. All programs are written in C for low-level implementation practice.

The study ran in Fall 2025, when the course had a total enrollment of 330 split between two lecture sections and twelve lab sections. In the required two-hour weekly labs, students work collaboratively on non-coding activities, and coding exercises aligned with weekly lecture topics. Throughout the term, students complete five programming projects, each spanning about three weeks. The two mid-semester projects, specifically Project 3 and Project 4, form the primary intervention for this study.

All 330 enrolled students completed the follow-up survey after the deadline for both Project 3 and Project 4 where they were assigned open-ended and fully-specified programming tasks. The student participants included 232 males and 84 females. Full details on the student demographics (gender, age, race/ethnicity, and highest parental educational level) for this study, which took place at the University of Illinois Chicago, a minority-serving institution, are given in Table 1 and are reflective of an urban Chicago campus and national norms [4].

Demographics	n
All Students	330
Gender	
Male	232
Female	84
Non-binary/third gender	5
Prefer not to say	9
Age	
18 - 24	307
25+	23
Race/Ethnicity	
Asian	170
Hispanic or Latino	52
White or Caucasian	30
Black or African American	18
Middle Eastern or North African	17
Multi-race	24
Prefer not to say	19
Parental Education Level	
Graduate or Professional Degree	65
Bachelors Degree	88
Technical or Associates Degree	30
Some college, but no degree	32
High school diploma or GED	57
Some high school or less	36
Prefer not to say	22

Table 1 – Survey Participant Demographics.

Two projects were selected for repeated measures, where the same student experiences both project versions, one with fully-specified tasks, and the other with an open-ended component. To counterbalance any potential ordering bias, enrolled students are put into two groups, based on their lab section; i.e. six lab sections are assigned to Group 1 for a total of 135 students, and the other six lab sections are assigned to Group 2 for a total of 195 students. Whereas Group 1 is assigned the fully-specified version of the first project and the open-ended version of the second project, Group 2 experiences the project versions in the opposite order. Forming the groups by lab section is done for convenience, since students are already separated by lab sections within the grading software used by the class instructors for projects and lab assignments. However, this has the added benefit that students can discuss their general approaches to the projects (whether open-ended or not) within their groups during lab sessions, which is designed to be the most collaborative component of the course learning environment.

There are a few practical constraints to consider when designing the programming projects:

- The programming project must align with course modules; i.e. it must require application of the module concepts from the course schedule (i.e. covered in lecture) during the appropriate time frame.
- The programming project must naturally have opportunities for creative, open-ended tasks.
- Two sequential projects that have a similar overall structure are required for counterbalanced repeated measures.

These constraints are all met by designing projects around simple word puzzle games, since there are many options, many of which have been published as Nifty assignments at past SIGCSE conferences [5]. Examples include Word Ladders [6], Evil Hangman [7], Spelling Bee [8], Wordle [8], LetterBoxed [9]. Additionally, projects centered on word puzzle games naturally promote creativity, making them well-suited for open-ended programming tasks.

The first project of this study, Project 3 in the course, is a **Spelling Bee Game and Solver**. The primary programming concepts are C-strings, file input, algorithm development, and optimization using binary search, which are all aligned with relevant course modules. The Spelling Bee project consists of three main parts: (a) build the hive, (b) play the game, and (c) solve the game. These parts are broken down into tasks for the students to complete, with full autograder testing for the first and third parts. For example, Task 1 involves building the dictionary word list utilized during gameplay and the solver, with an autograded testing suite that determines if a word fits all given requirements, i.e., word length, dictionary size, and hive letters. As another example, in Task 6, students are required to solve a given puzzle with a recursive binary search. The autograder tests the implementation of pre-specified functions that are required to complete each task, such as `findWord()` (finds a given word in the dictionary using binary search), `isPrefix()` (tests if a word is a prefix to any other word in the dictionary), and `findAllMatches()` (implements an efficient algorithm to find all solution words). The open-ended portion of the project is incorporated in the second part, *play the game*. Two versions of the project were developed for this part, a fully-specified version with a full autograder to test a basic gameplay implementation, and an open-ended version with only a few autograded tests of the core functions. As stated in the open-ended project description, “The final component of this task is purposefully left open-ended for whatever ideas you have for making an interactive and enjoyable experience for the user.” Below is sample output from the basic gameplay, showing the list of solution words, the score for each word, the total score, and identifications of *pangrams* (*), i.e. words that use all hive letters, and *perfect pangrams* (***), i.e. pangrams that use each hive letter only once, as shown in Figure 1.1:

```
Word List:
*** (16) computer
    ( 7) compute
    ( 1) mute
 *  (17) recompute
    ( 5) comet
Total Score: 46
```

Note: the hive for this sample output is "cemoprtu" (hiveSize = 8) with required letter 'e'.

Figure 1.1 – Sample output from the Spelling Bee project: the solution list for a hive with required letters ‘cemoprtu’ shows how the output should appear to the user.

The second project of the study, Project 4 in the course, is a **Word Ladder Game and Solver**. The primary programming concepts practiced in the second project overlap with the first project, with the additional concept of using linked lists and linked structures, again aligned with the relevant course modules. A word ladder is a bridge between one word an another, formed by changing one letter at a time, with the constraint that at each step the sequence of letters still forms a valid word. The sample output below is from the basic gameplay in the fully-specified version of the project, and it shows a word ladder display that signifies which letter is changed at each rung of the ladder, as shown in Figure 1.2:

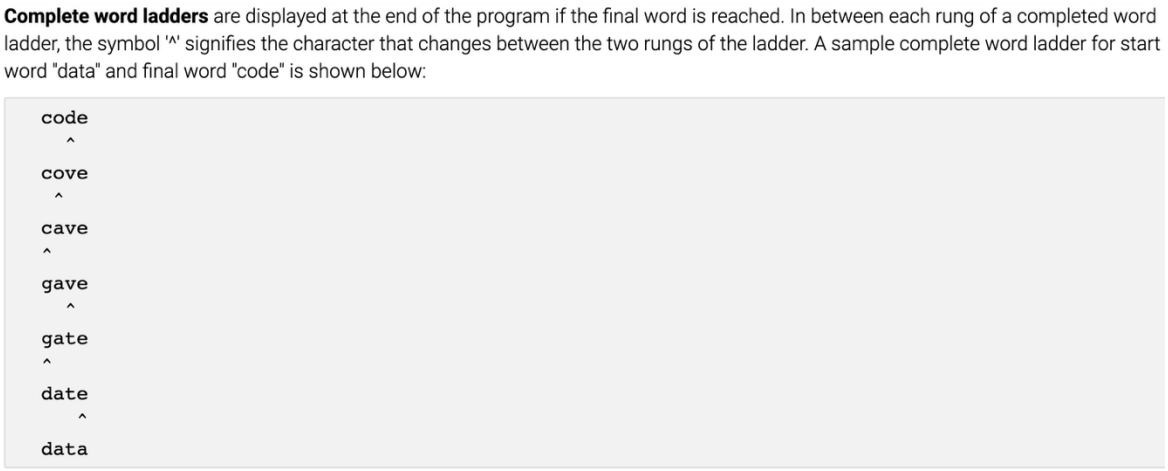


Figure 1.2 – Sample output from the Word Ladder project: an example walkthrough of a word ladder that starts at ‘code’ and ends at ‘data’.

The goal of the project is to find the smallest possible ladder between any two given words of the same length. The early project tasks involve foundational elements, such as building a dictionary that contains only words of the specified length, setting gameplay parameters from user input, and functions related to building a linked list of C-structs that represent word ladders (to be built later in the project). The open-ended portion is again the gameplay, the intermediate part of the program where the user interacts with the program to construct a word ladder of their own. Students are tasked with implementing a creative extension to the basic game play to provide the user with an enhanced experience.

Both projects require the students to develop an engaging game play interface, followed by an implementation of an optimized solver to find the full set of solution words for the Spelling Bee hive or to find the shortest Word Ladder. In both projects, the game play interface component has two different versions. One version has fully-specified tasks where the student is required to implement a preset, classic version of the game with all program outputs predetermined, along with a full-suite of autograded test cases to check all required functions and the full program output for sample game plays. The second, open-ended version has a small autograded test suite to check only 1-2 required functions that focus specifically on core programming concepts, e.g. C-string analysis. The purpose is to provide a solid, jumping off point for students to create their own modified and/or extended game play.

For the open-ended version of the project, a demo program executable is provided for the students to experience a *standard game play*. Their task is to implement a *creative game play extension*. Some example extensions of the Spelling Bee game include the following:

- Allow users to play against "the computer," which selects words (randomly or via the solver) until either side reaches a target score.
- A randomly selected required letter for each round of play.
- Add specific constraints each round, such as "words must begin with a vowel" or "words must be exactly 5 letters long" or "must be a pangram."

The open-ended game play implementation is graded for incorporating key programming concepts (e.g. valid logic for game rules, C-string analysis for identifying *pangrams*, keeping accurate score, etc.) and for creativity. The grading rubric for both projects are included in Appendix I. Additional grading time and resources are required for the open-ended tasks beyond what is necessary for the autograded, fully-specified project versions. Grading of the open-ended tasks is broken down into two items, (1) proper application of required programming constructs and (2) creativity/complexity of the gameplay extension (as compared to the basic gameplay). Each item is worth 5% of the total project score and is graded using a four-level proficiency scale, where the *Excellent* and *Acceptable* levels signify *Meets (or Above) Expectations* and the *Developing* and *Minimal* levels signify *Below Expectations*. The rubrics are designed to offer clear performance distinctions while remaining simple and reliable to apply, which supports consistent scoring across multiple evaluators while utilizing limited grading resources efficiently.

Below are a few example programming extension descriptions left by teaching assistants when leaving feedback for the students using the rubric. These example all received *Excellent* creativity scores.

- *They introduced a new point system by taking in a target score from the user and starting the timer when the player starts the game. The final score is a function of time taken to complete the round and words guessed. They also maintain a local file and write the highest score to it (so it's a persistent change!) and displays a leaderboard of highest score, which gets updated if someone beats it. It's pretty cool.*
- *They increase the difficulty of the game by introducing a second required letter as well which changes in every round so it's fun to play.*
- *The added a 'bingo' concept for entire word guesses, slow progression through the game with interactive UX messages which demonstrate immersion, reward signaling, and state progression.*

An anonymous survey was administered via Qualtrics shortly after the submission deadline for the second project. The survey has five sections, including: Project Modality Feedback, Gen-AI Usage, Self-Efficacy, Preparedness and Belongingness, and Demographics. The first section focuses student perceptions of the different project task versions, fully-specified vs. open-ended. The second section asks students to self-report their usage of genAI tools, specifically regarding the open-ended project tasks, which requires additional creative thinking. By course policy, students were not allowed to use genAI tools for code development, but it is not specified if genAI is allowed for other software development steps. The third section measures self-efficacy using questions from the CPSES instrument [10] (with a few additional questions related to open-ended project development, e.g., asking if students felt they could create a program without an autograder). No data is presented in this paper on the self-efficacy results; analysis is reserved for future work. The fourth section asks questions about belongingness and preparedness. No data is presented in this paper on the self-efficacy, belongingness, or preparedness results; analysis is reserved for future work. The last section is demographics.

RESULTS & DISCUSSION

Self-Reported Understanding & Perceptions

The survey includes questions related to student perceptions of the projects, specifically on the open-ended tasks. Nine questions asked the students to self-report the impact of the open-ended tasks on their understanding of a core concept. The response distributions for these questions are shown in Figure 2.

The concepts with the greatest reported levels of confidence improvement (30% or more of students) are makefiles/make utility, C-strings, and search/sort algorithms. Whereas the use of C-strings is ubiquitous in all tasks for both projects, the other two concepts with high levels of reported confidence improvement are less obvious. Many students use makefiles to compile and run the gameplay portion of the program with various inputs, causing students to gradually become more comfortable with the make utility. Search and sort algorithms are used primarily in the form of a binary search on the full dictionary of words, which could imply students came to better understand how binary search works by completing the open-ended task, or they may choose to implement a different, but related, algorithm.

Other concepts with moderately high (about 25% of students) reported levels of confidence improvement include debugging tools, functional decomposition, and pointers and dynamic memory. Debugging and pointers/dynamic memory are related concepts; i.e. debugging tools are vital for tackling dynamic memory issues, which are often challenging to track down, possibly leading to frustration and lower confidence. However, successfully debugging a tricky memory leak is often a rewarding experience that may contribute to a greater sense of accomplishment, especially when done for the open-ended gameplay task, which has no autograder. Functional decomposition is broad concept where a student's confidence can depend on many factors. For example, developing a function to achieve a specific component of a their own gameplay ideas may increase satisfaction and confidence, but unsuccessful attempts or high-levels of difficulty in writing the function may lead to a decrease in confidence in their ideas and independence. The free-response student feedback reported below provides additional insights.

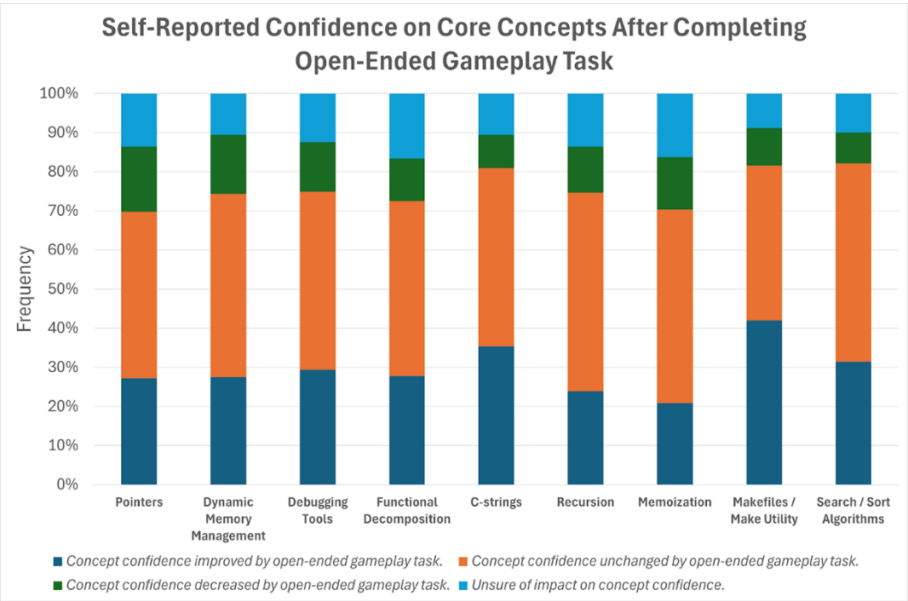


Figure 2 – Stacked relative frequency distribution for four answer choices across nine programming concepts (e.g., Pointers).

Recursion and memoization are among the lowest for reported confidence improvement. Students were not directly encouraged (nor discouraged) to use either topic in the open-ended task, so it is unlikely that students would utilize these concepts without explicit requirements.

Figure 3 shows the project task modality preference distribution. The leftward skew implies that students tended to prefer the fully-specified project tasks over the open-ended tasks. When the data is quantified using -1 = *strongly prefer open-ended*, 0 = *no preference*, 1 = *strongly prefer fully-specified*, the mean response is 0.189.

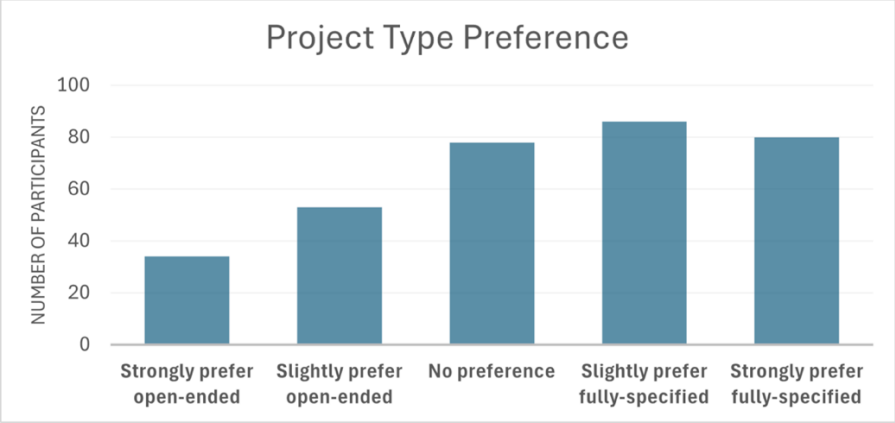


Figure 3 – Project task modality preference, open-ended or fully-specified.

Figure 4 shows the distribution of perceived stress level comparison between project task modalities. When the data is quantified using -1 = *open-ended much more stressful*, 0 = *equally stressful*, 1 = *fully-specified much more stressful*, the mean response is -0.468. The vast majority (71.6%) of students felt that the open-ended gameplay tasks were more stressful to work on than the fully-specified tasks, likely related to open-ended project tasks requiring more effort and critical thinking.

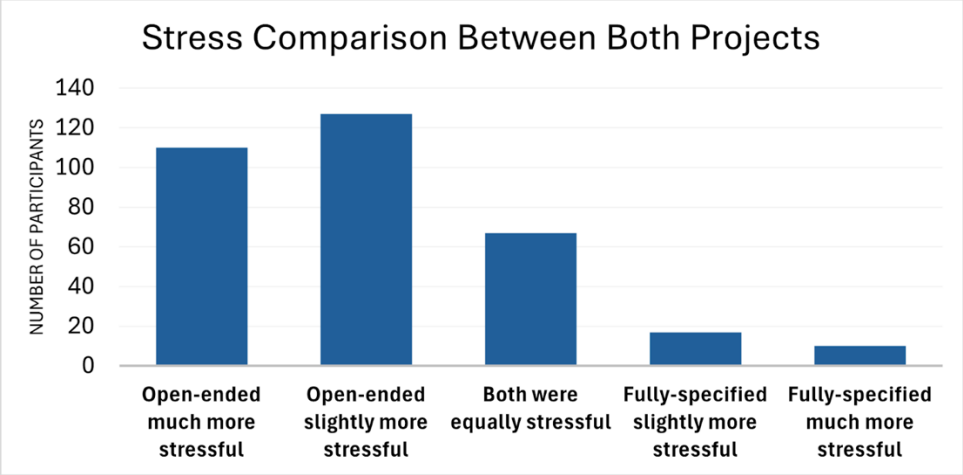


Figure 4 – Stress comparison, open-ended or fully-specified.

Students may have found the open-ended tasks more stressful due to the additional creativity or functionality components. To provide additional insights, the following are notable samples from the free-response student feedback:

- *"I usually prefer fully-specified tasks because the clear instructions make it easier to know what's expected. Open-ended tasks are fun and creative, but sometimes they can feel a little overwhelming since you have to decide more on your own." - Student 1*
- *"The open-ended programming task was more creative and can bring out more ideas. The pro of specified programming was that it was simpler in terms knowing what to do. The con of open-ended was that it took more time, and the con of specified tasks was that it arguably led to less creativity." - Student 2*
- *"Open-ended gives me much more freedom to toy around with the code as I see fit. It especially helps considering I am in CS + Design [major], so I get to exercise my design skills to projects such as these. Fully-specified leaves a lot less to the imagination and creativity by locking you into doing things one specific way, but it at least gives you an opportunity to work on a tougher method you wouldn't have normally gone with, boosting your coding skills overall." - Student 3*
- *"My only problem with the open-ended option was that I got too engaged in the task because it was open ended, which resulted in me spending too much time creating my own version and missing the deadline" - Student 4*

In order to further investigate project modality preferences, students selected from a list of potential stressors that contributed to their stress while engaging in both project modalities. The top 3 stressors for the open-ended task include 'Being unsure my implementation met the correct standards' (count = 206), 'Not knowing where to start' (count = 181), and 'Generating my own ideas' (164). The top 3 stressors for the fully-specified modality include 'Matching the program' (count = 163), 'Making sure I met every detailed requirement or specification' (count = 149), and 'Feeling pressure to avoid small mistakes that could cause the program to fail testing' (count = 118). The overall counts for the open-ended task stressors outweigh the fully-specified modality stressors, which aligns with the drastic rightward skew in Figure 4.

One interesting observation from Figure 3 and Figure 4 is that 48 students reported a preference for open-ended despite also reporting that it is more stressful, which is more than half of the students who prefer the open-ended modality. Thus, stress is not the only contributor to project mode preference; there exists other factors, such as sense of accomplishment.

Figure 5 shows the distribution of student-reported sense of accomplishment, which is higher for the open-ended gameplay task than the fully-specified project task, as evidenced by the slight rightward skew. When the data is quantified using -1 = *open-ended much more*, 0 = *both equal*, 1 = *fully-specified much more*, the mean response is -0.116.

Similar to the above presentation of stress factors, students also selected from a list of potential contributors to their sense of accomplishment resulting from their experiences with both project modalities. The top 3 accomplishment contributors for the open-ended modality are 'Solving Challenging Problems Independently' (count = 152), 'Developing Real-World Applications' (count = 147), and 'Cultivating a Unique Experience' (count = 119). With over half of students reporting that 'Solving Challenging Problems Independently' contributed to their sense of accomplishment when working on the open-ended task, it is likely that this is a primary reason many students prefer the open-ended project modality. The top 3 accomplishment contributors for the fully-specified modality are 'Passing All Autograded Tests' (count = 189), 'Completing Task Correctly and Efficiently' (count = 169), and 'Matching Exact Program Output' (count = 162). As with stress factors, the total counts for contributors to sense of accomplishment for the fully-specified tasks outweigh the open-ended modality counts.

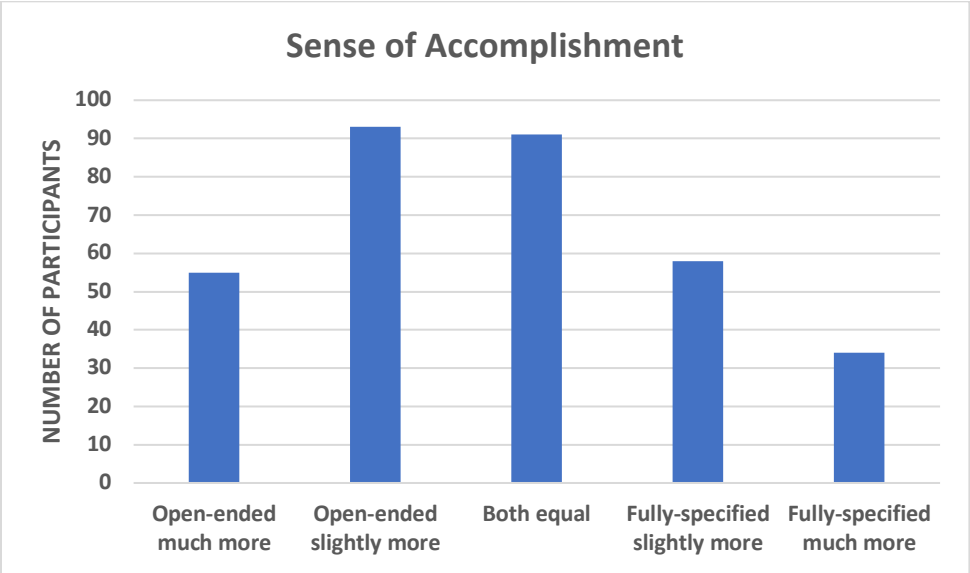


Figure 5 – Sense of accomplishment comparison, open-ended or fully-specified.

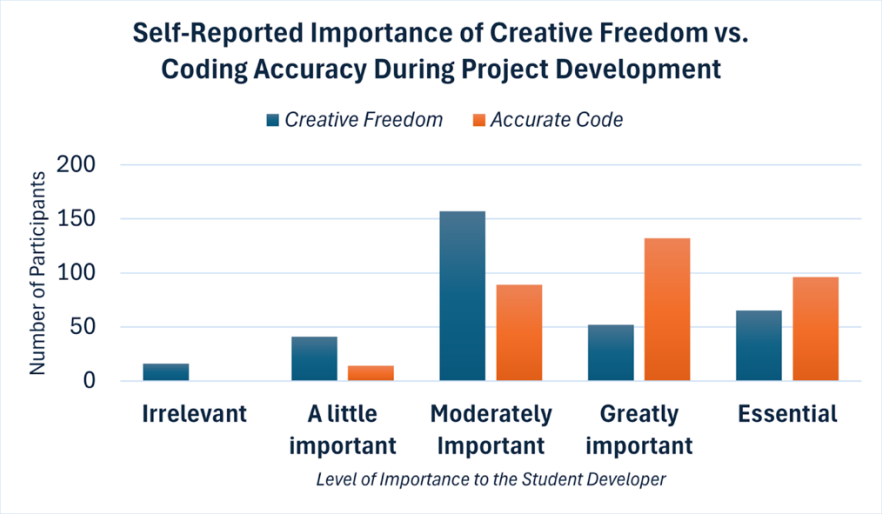


Figure 6 – Comparison of student-reported levels of importance for creative freedom and coding accuracy during project development.

Students also reported the level of importance they place on two aspects of a programming project's design: (1) creative freedom and (2) the sense of knowing their code is accurate. Figure 6 shows that while students value both creative freedom and a sense of accuracy, the latter is more important. When quantified as 0 = *Irrelevant*, 2 = *Moderatly Important*, 4 = *Essential*, the mean responses for creative freedom and sense of accuracy are 2.33 and 2.94, respectively. This aligns with the student-reported preference for fully-specified projects, as they allow students to utilize the autograder for testing. Overall, while students may feel more pride in their work by completing a task without an autograder and predefined functions, they report a preference toward the traditional project modality. While students value the creative freedom offered by open-ended tasks, they more strongly value the assurance of knowing that their code is accurate and achieves the required functionality. The open-endedness leads to higher stress and can be overwhelming, especially since students are not immediately made aware of their errors.

GenAI Involvement

To investigate the impact of genAI on the projects used in this study, the survey asks students to reflect on their usage of genAI for course projects. One free-response student feedback stated:

> *"I create ALL my code, comments, and ideas on my own. I DO NOT use a LLM to do my work for me. I believe it is seriously important for students to be able to proficiently write and analyze code. I do however think that AI is a tool that can be seriously useful for us programmers, and learning how to use it to write better code would do us students a huge favor. […]" - Student 5*

Student perception on genAI were mixed, ranging from heavy use to little use. Certain students, such as Student 5, stated that they tend to stay away from heavy reliance on LLMs (Large Language Models). The way in which students reported using LLMs also vary. For instance, one student explained that it could be beneficial to use an LLM to help debug the program, especially after dedicating a great amount of time towards attempting to find a tricky issue. Many students shared similar feelings to Student 5, particularly in describing how genAI can be helpful for programmers to learn how to write better code or minimize time spent on a certain steps of the development process, whether that be programming projects for school or personal projects; while also showing concern for overuse of AI as an early programmer. Here are some additional sample student responses that express a need to moderation in genAI usage:

> *"genAI makes smaller tasks easier for students but also stunts their growth by relying on other sources instead of their own critical thinking" - Student 6*

> *"AI is useful to get a plan going and to give some creative ideas especially if you're not that creative yourself. You can also use it to teach concepts on the spot if you are lacking understanding" - Student 7*

Students also self-reported their usage of genAI directly for the projects assigned for this study. The survey was wholly anonymous with no identifying information collected, which was explained to the students in the survey description. Despite these assurances, student responses may be biased by not wanting to admit to genAI usage, whether related to policy or stigma.

Figure 7 shows the relative frequency of student-reported genAI usage for four steps of the project development process. The overwhelming majority of students reported that they did not use genAI to help with any part of the development process (brainstorming, writing code, debugging, commenting) of the open-ended gameplay task. Students were more likely to use genAI for help brainstorming ideas or debugging than for code development and commenting. It is important to point out that genAI usage for code development is strictly against policy for this course. Students were given a few samples of creative gameplay extension, which may have some impact on the number of students using genAI for brainstorming ideas. The reported tendency to use genAI

The survey also asks students to report the frequency of their genAI usage, ranging from *Never*, *Rarely*, *Sometimes*, *Often*, and *Always*. These responses were used to assign students to two groups, where the *Not Likely to Use AI* group includes the *Never* and *Rarely* responses, while the rest of the responses are assigned to the *Likely to Use AI* group. Whereas there is very little difference in project modality preference between these two groups, there is a notable difference in the way these two groups reported their level in confidence in developing a program from scratch. Figure 8 shows the relative frequency distribution for the reported confidence levels for

these two groups. The overall reported level of confidence is significantly higher for the Not Likely to use AI group.

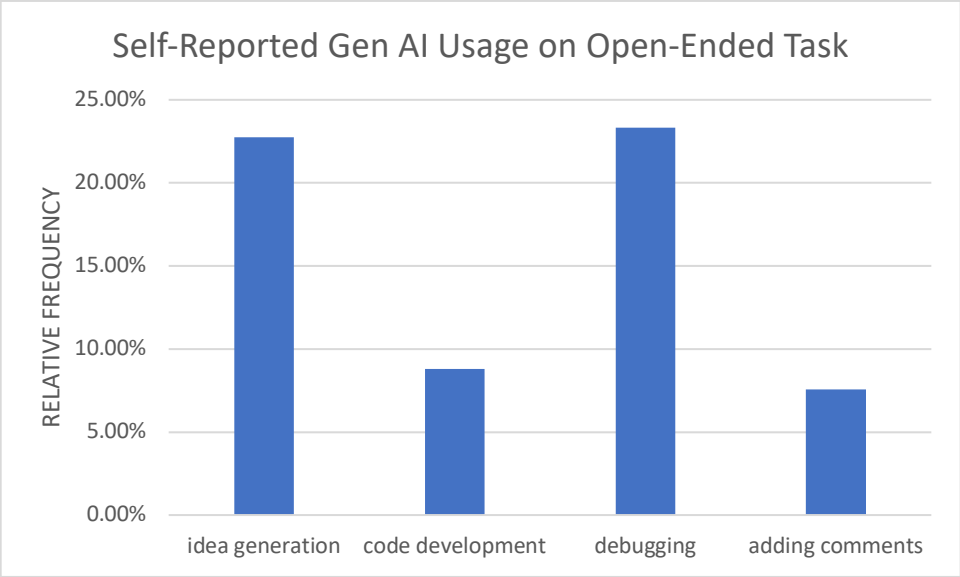


Figure 7 – Student-reported usage of genAI for specific steps of the project development process, including idea generation, code development, debugging, and commenting.

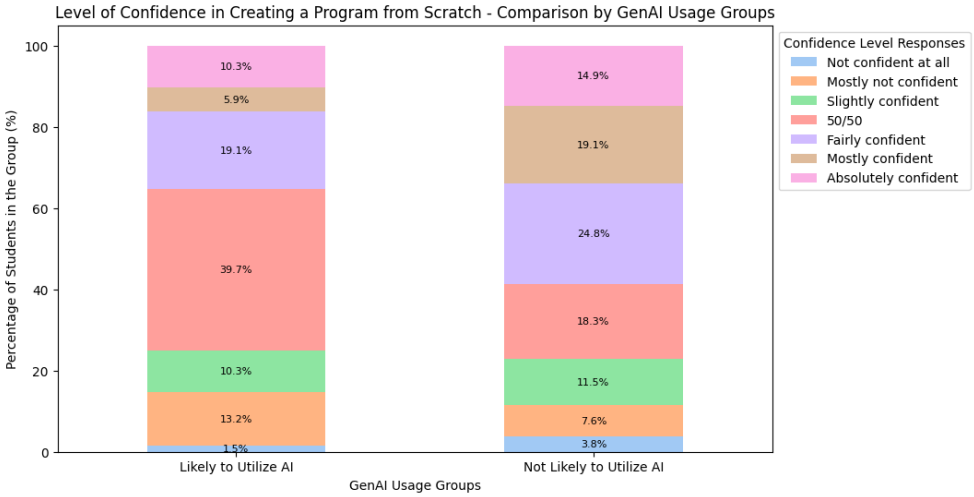


Figure 8 – A comparison of two groups of students (those likely vs. not likely to utilize AI) in the reported confidence level when creating a program from scratch.

CONCLUSIONS

For research question (1), survey responses indicated that although the highest frequency across all concepts was unchanged confidence, the second highest response category was "improved confidence by open-ended gameplay task." Students most commonly reported increased confidence (≈30%) in C-strings, makefiles/make utility, and search/sort algorithms after completing the open-ended gameplay task. Two concepts - recursion and memoization - were expectedly lowest in reported improvement, likely due to their limited application within the gameplay context and the time constraints of the projects. These findings suggest that open-ended gameplay tasks may be particularly effective at strengthening confidence in practical,

repeatedly exercised concepts that arise naturally through iterative testing, debugging, and refinement, while having less impact on advanced or peripheral concepts that are not directly prompted by the task structure.

For research question (2), students' perceptions revealed a clear and important tradeoff. While a majority of students reported a preference for the fully-specified gameplay tasks (most often citing clearer expectations, autograder feedback, and stronger assurance of correctness) they also reported a greater sense of accomplishment from completing the open-ended gameplay task. Students overwhelmingly indicated that open-ended tasks were more stressful, attributing this stress to uncertainty about correctness and expectations, difficulty deciding where to start, and the cognitive load associated with generating and refining original ideas. At the same time, students associated open-ended work with deeper engagement, creative ownership, and satisfaction from independently overcoming technical challenges. One student captured this tension succinctly: *"Open-ended tasks are fun and creative, but sometimes they can feel overwhelming since you have to decide more on your own,"* while another reflected that becoming "too engaged" led them to invest significantly more time, even at the expense of deadlines. Together, these findings indicate that although students value the security and efficiency of fully-specified, autograded assignments, open-ended components can uniquely foster personal investment, autonomy, and intrinsic accomplishment that are less commonly reported in fully-specified, autograded project formats.

For research question (3), most students self-reported that they did not use generative AI tools to complete the open-ended gameplay task. Among those who did report genAI use, it was more commonly for brainstorming ideas or debugging rather than for generating complete solutions. Qualitative responses further revealed that many students viewed genAI as a potentially beneficial support tool when used selectively (for example, to overcome creative blocks or identify difficult bugs), while simultaneously expressing concern that reliance on AI to generate code could undermine learning and critical thinking. One student noted that AI could be valuable for "getting a plan going" or clarifying concepts, whereas another emphasized its appropriateness only after sustained independent effort. These findings suggest that open-ended tasks may encourage more restrained and reflective genAI use, positioning AI as a supplementary aid rather than a primary solution source.

Despite students having more stress from the open-ended task and an overall preference for the fully-specified projects, students on average felt a greater sense of accomplishment by successfully completing a program task generated from their own ideas. This leads to an important question for CS educators: "Which version is better to implement?" Is stress a factor that should be overlooked in order to achieve a goal of increased pride in the students, or should they stick to the method that students have grown accustomed to and rely on?

Taken together, this study highlights the complex instructional tradeoffs involved in incorporating open-ended programming tasks into CS2 coursework. Open-ended gameplay extensions increased reported confidence in several foundational programming concepts and promoted feelings of accomplishment, creative ownership, and independence, yet also introduced higher stress and uncertainty, contributing to students' overall preference for fully-specified, autograded tasks. These outcomes suggest that future programming projects may benefit from hybrid designs that preserve the motivational and developmental advantages of

open-ended work while strategically supporting students' strong desire for clarity and correctness. Potential approaches include lightweight correctness checks, staged milestones, structured ideation prompts, and transparent rubrics that provide guidance without constraining creative space. Thoughtfully designed, such interventions may better align project experiences with both student preferences and the broader educational goals of fostering autonomy, problem formulation, and professional programming practices in CS2 contexts.

Future work will incorporate performance-based analyses using project scores and project-aligned exam items to evaluate whether the self-reported confidence gains and perceptions observed in this study correspond to measurable differences in concept mastery and problem-solving ability.

ACKNOWLEDGMENTS

We would like to thank the two instructors, five graduate teaching assistants, and seven undergraduate teaching assistants who supported students on their programming project tasks (open-ended and fully-specified) during Help Sessions. We would also like to thank the students in CS 211 who gave their time and effort to their course and this study. Lastly, we would like to thank the University of Illinois at Chicago and the Department of Computer Science for providing the resources needed to complete this study.

REFERENCES

[1] Barrett Ruth and John R. Hott. 2025. "Auto-grading in Computing Education: Perceptions and Use." In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education* V. 1 (Pittsburgh, PA, USA) (SIGCSETS 2025). Association for Computing Machinery, New York, NY, USA, 1008–1014. doi:10.1145/3641554.3701900.

[2] Jason Lee Weber, Hyunjun Park, Daniel J. Song, Jared Apillanes, Barbara Martinez Neda, Jennifer Wong-Ma, and Sergio Gago-Masague. 2025. "Investigating Autograder Usage in the Post-Pandemic and LLM Era." In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education* V. 2 (Pittsburgh, PA, USA) (SIGCSETS 2025). Association for Computing Machinery, New York, NY, USA, 1653–1654. doi:10.1145/3641555.3705208.

[3] Sadia Sharmin, Daniel Zingaro, Lisa Zhang, and Clare Brett. 2019. "Impact of Open-Ended Assignments on Student Self-Efficacy in CS1." In *Proceedings of the ACM Conference on Global Computing Education* (Chengdu,Sichuan, China) (CompEd '19). Association for Computing Machinery, New York, NY, USA, 215–221. doi:10.1145/3300115.3309532.

[4] Valerie Barr, Carla E. Brodley, and Manuel Pérez-Quiñones. 2024. "Visualizing Progress in Broadening Participation in Computing: The Value of Context." *Communications of the ACM*. https://cacm.acm.org/research/visualizing-progress-in-broadening-participation-in-computing-the-value-of-context/

[5] Nick Parlante, Owen Astrachan, Mike Clancy, Richard E. Pattis, Julie Zelenski, and Stuart Reges. 1999. "Nifty assignments panel." *SIGCSE Bulletin* 31, 1 (March 1999), 354–355. doi:10.1145/384266.299809.

[6] Owen L. Astrachan. 2001. "Word Ladder Nifty Assignment." Retrieved October 5, 2025 from https://www2.cs.duke.edu/csed/poop/ladder/

[7] Keith Schwarz. 2011. "Evil Hangman Nifty Assignment." Retrieved October 5, 2025 from http://nifty.stanford.edu/2011/schwarz-evil-hangman/

[8] Eric S. Roberts. 2022. "Spelling Bee + Wordle Nifty Assignment." Retrieved October 5, 2025 from http://nifty.stanford.edu/2022/eroberts-spelling-bee-wordle/

[9] New York Times. 2025. Letter Boxed Game. Retrieved October 5, 2025 from https://www.nytimes.com/puzzles/letter-boxed

[10] Vennila Ramalingam, and Susan Wiedenbeck. "Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy." *Journal of Educational Computing Research* 19.4 (1998): 367-381.

Appendix I: Programming Project Rubrics

Programming Practicum
CS 211

Open-Ended Programming Project 3 Task – Grading Rubric

Criterion	Excellent (5 pts)	Acceptable (3 pts)	Developing (1 pts)	Minimal (0 pts)	Score
Open-Ended Gameplay requirements: 1. accurately checks that user-entered words are valid: (a) in dictionary, (2) only use hive letter, (3) includes required letter. 2. finds and identifies pangrams and perfect pangrams **3.** regularly displays the list of valid user-entered words, with the point value for each word and the total score displayed	Open-Ended Gameplay is well described, with sample output, and **all three requirements** are met	Open-Ended Gameplay is well described, with sample output, but **only two requirements** are met	Open-Ended Gameplay is well described, with sample output, but **only one requirement** is met	Open-Ended Gameplay is not described OR **no requirements** are met	
Open-Ended Gameplay creativity score for creating an enjoyable gameplay experience for the user. The gameplay implementation should NOT exactly match the demo program, but instead the implementation is a creative way to uniquely extend the basic gameplay.*	**High creativity**: Open-Ended Gameplay is well described, with sample output and implements a **unique idea of high complexity** similar to the suggestions given in the project description. The implementation **significantly enhances the basic playMode** and creates an enjoyable experience for the user playing the game.	**Moderate creativity**: Open-Ended Gameplay is well described, with sample output and implements a **distinctive idea of moderate complexity** similar, but NOT exactly matching, the basic gameplay presented in the demo executable. The implementation is simpler than the suggestions given in the project description. The implementation **presents slightly modified version of the basic playMode** and creates an average experience for the user playing the game.	**Developing creativity**: Open-Ended Gameplay is well described, with sample output and implements a **basic gameplay experience** that is extremely similar to or simpler than the basic gameplay presented in the demo executable. The implementation is vastly simpler than the suggestions given in the project description. The implementation **is an inferior version of the basic playMode** and creates a diminished experience for the user playing the game.	**Minimal or missing**: Open-Ended Gameplay description is missing or the description does not include sample output or the submission does not show a valid gameplay implementation.	
				TOTAL (out of 10)	

***Suggestions for game play extension:**

1. Develop a new game mode to allow the user to play against "the computer". This may involve "the computer" randomly picking words from the dictionary OR running the solver (see Task 5) first, so that "the computer" can enter valid words. The game continues until either the user or the computer reaches a threshold score.
2. Develop a new game mode where the required letter is randomized during each round of the game play.
3. Develop a new game mode where specific constraints are added at each round, e.g. “words must start with a vowel for this round” or “words must be exactly 5 letters long for this round” or "the next word must be a pangram."

Programming Practicum
CS 211

Open-Ended Programming Project 4 Task – Grading Rubric

Criterion	Excellent (5 pts)	Acceptable (3 pts)	Developing (1 pts)	Minimal (0 pts)	Score
Open-Ended Gameplay requirements: 1. Accurately checks that user-entered words are valid: (a) in dictionary and (b) only one-character difference from the previous ladder word. 2. Regularly displays incomplete word ladders using clean formatting that encourages the user to keep building. 3. Displays complete word ladder(s) (if one is constructed during game play), in a manner unique from the incomplete ladder display, that highlights how individual characters are changed at each rung of the ladder.	Open-Ended Gameplay is well described, with sample output, and **all three requirements** are met	Open-Ended Gameplay is well described, with sample output, but **only two requirements** are met	Open-Ended Gameplay is well described, with sample output, but **only one requirement** is met	Open-Ended Gameplay is not described OR **no requirements** are met	
Open-Ended Gameplay creativity score for creating an enjoyable gameplay experience for the user. The gameplay implementation should NOT exactly match the demo program, but instead the implementation is a creative way to uniquely extend the basic gameplay.*	**High creativity**: Open-Ended Gameplay is well described, with sample output and implements a **unique idea of high complexity** similar to the suggestions given in the project description. The implementation **significantly enhances the basic playMode** and creates an enjoyable experience for the user playing the game.	**Moderate creativity**: Open-Ended Gameplay is well described, with sample output and implements a **distinctive idea of moderate complexity** similar, but NOT exactly matching, the basic gameplay presented in the demo executable. The implementation is simpler than the suggestions given in the project description. The implementation **presents slightly modified version of the basic playMode** and creates an average experience for the user playing the game.	**Developing creativity**: Open-Ended Gameplay is well described, with sample output and implements a **basic gameplay experience** that is extremely similar to or simpler than the basic gameplay presented in the demo executable. The implementation is vastly simpler than the suggestions given in the project description. The implementation **is an inferior version of the basic playMode** and creates a diminished experience for the user playing the game.	**Minimal or missing**: Open-Ended Gameplay description is missing or the description does not include sample output or the submission does not show a valid gameplay implementation.	
				TOTAL (out of 10)	

***Suggestions for game play extension:**

1. Develop a new game mode to allow the user to play against "the computer". This may involve "the computer" randomly picking words from the dictionary (*easy-mode*) OR running the solver first, so that "the computer" always enters an optimal word (*hard-mode*), OR some combination (*medium-mode*). The game continues until either the user or the computer completes the word ladder.
2. Develop a new game mode with limitations on which letters can or must change each round of the game play.
3. Develop a new game mode that allows the word size to grow or shrink as the word ladder is developed, e.g. a word ladder connecting **rate -> ring** could now be done as **rate -> rat -> ran -> rang -> ring**, which has a ladder height of 5, which is one better than can be done without this new game mode.
4. Develop a new game mode where the user can play multiple times with randomly generated start/final words and a record is kept of completion vs. incompletion AND the ladder lengths. Note: make sure to keep the original start/final words for running the solver.
5. Develop a new game mode to allow the user to play the same start->final word ladder multiple times by introducing a "reset" option.

Appendix II: Open-Ended vs. Fully-Specified Programming Tasks Survey Questions

PROGRAMMING PROJECT TASK MODALITY FEEDBACK:

1. [SINGLE SELECTION] Lecture Section: 4pm or 5pm
2. [SINGLE SELECTION] Lab Section: 8am, 10am, 12pm, 2pm, or 4 pm
3. [SINGLE SELECTION] For which project were you assigned an open-ended creative gameplay extension (instead of the fully-specified gameplay tasks)?
    - Project 03 – Spelling Bee Game & Solver
    - Project 04 – Word Ladder Building Game & Solver
    - I don't know/remember
4. [SINGLE SELECTION] As part of either Project 03 or Project 04, you were tasked with developing an open-ended creative gameplay for either the Spelling Bee game (Project 03) or a Word Ladder Building game (Project 04). Which of the following best describes your engagement with the open-ended gameplay task?
    - No engagement - I did no work on the open-ended gameplay extension
    - Little engagement - I did some early planning for the open-ended gameplay task, but did not get my ideas implemented in code
    - Partial engagement - I implemented some ideas for the open-ended gameplay task, but never got it fully functioning
    - Standard engagement - I fully implemented a creative extension for the open-ended gameplay task by closely following one of the examples given in the project description
    - Extensive engagement - I fully implemented a creative extension for the open-ended gameplay task that stems wholly from my own ideas and is of greater complexity than the examples given in the project description.
5. [SINGLE SELECTION] As part of either Project 03 or Project 04, you were tasked with developing a fully-specified, highly-scaffolded gameplay for either the Spelling Bee game (Project 03) or a Word Ladder Building game (Project 04), with a full set of autograded test cases to check your implementation. Which of the following best describes your engagement with the fully-specified gameplay task?
    - No engagement - I did no work on the fully-specified gameplay task
    - Little engagement - I outlined rough steps for the fully-specified gameplay task, but did not implement anything in code.
    - Partial engagement - I implemented some code for the fully-specified gameplay task, but never got anything functioning.
    - Standard engagement - I fully implemented the fully-specified gameplay task by closely following the project description closely and relying on the autograder to test my code.
    - Extensive engagement - I fully implemented the fully-specified gameplay task using my own set of test cases to determine if the code was working correctly.

Questions 6-14 [SINGLE SELECTION] have identical answer options:

- Concept confidence improved by open-ended gameplay task.
- Concept confidence unchanged by open-ended gameplay task.
- Concept confidence decreased by open-ended gameplay task.
- Unsure of impact on concept confidence.

6. Evaluate the impact of your open-ended gameplay task experience on your confidence with the programming concept of pointers.
7. Evaluate the impact of your open-ended gameplay task experience on your confidence with the programming concept of dynamic memory management.
8. Evaluate the impact of your open-ended gameplay task experience on your confidence with the programming concept of debugging Tools (valgrind, gdb, VScode, etc.).
9. Evaluate the impact of your open-ended gameplay task experience on your confidence with the programming concept of functional decomposition.
10. Evaluate the impact of your open-ended gameplay task experience on your confidence with the programming concept of C-strings.
11. Evaluate the impact of your open-ended gameplay task experience on your confidence with the programming concept of recursion.
12. Evaluate the impact of your open-ended gameplay task experience on your confidence with the programming concept of memoization.
13. Evaluate the impact of your open-ended gameplay task experience on your confidence with the programming concept of makefiles and the make utility.
14. Evaluate the impact of your open-ended gameplay task experience on your confidence with the programming concept of search/sort algorithms.
15. [FREE RESPONSE] Describe your experience with the open-ended, creative gameplay task. Was this a new experience for you, or have you done something similar in the past? How did you approach the task? How successful were you at implementing your creative ideas?
16. [SINGLE SELECTION] Both Project 03 (Spelling Bee) and Project 04 (Word Ladder) involved the implementation of a gameplay interface. In one project, you were tasked with developing a fully-specified, highly-scaffolded play mode with a full set of autograded test cases requiring specific program outputs. In the other project, you were tasked with an open-ended creative play mode extension without autograded test cases to allow for unique program outputs. If you had the choice between an open-ended and a fully-specified gameplay task in a future project, which gameplay modality would you prefer?
    - I strongly prefer the open-ended gameplay task over the fully-specified task.
    - I slightly prefer the open-ended gameplay task over the fully-specified task.
    - I have no preference between the open-ended and fully-specified gameplay tasks.
    - I slightly prefer the fully-specified gameplay task over the open-ended task.
    - I strongly prefer the fully-specified gameplay task over the open-ended task.

17. [FREE RESPONSE] Explain your preference between open-ended and fully-specified programming tasks. What do you see as the pros and cons of each modality?
18. [SINGLE SELECTION] Which gameplay modality did you find more stressful to work on?
    - The open-ended gameplay task was much more stressful to work on than the fully-specified task.
    - The open-ended gameplay task was slightly more stressful to work on than the fully-specified task.
    - The open-ended and fully-specified gameplay tasks were equally stressful to work on.
    - The fully-specified gameplay task was slightly more stressful to work on than the open-ended task.
    - The fully-specified gameplay task was much more stressful to work on than the open-ended task.
19. [MULTIPLE SELECTION] Which of the following are primary contributors to your stress level when working on open-ended programming tasks? (Select all that apply.)
    - Generating my own ideas
    - Being unsure whether my implementation met the expected standards
    - Being unsure whether my implementation was correct
    - Not knowing where to start
    - Feeling overwhelmed by too many possible design choices
    - Difficulty breaking the task into smaller steps
    - Feeling uncertain about whether I was “on the right track”
    - Lack of clear examples or reference solutions
    - Limited time to explore or iterate on my ideas
    - Feeling responsible for creating something original or creative
    - Technical challenges (bugs, errors, or unexpected behavior)
    - Difficulty understanding the task description or requirements
    - Worrying that my solution would be judged or compared to others
    - Other (please specify)
    - None of the above
20. [MULTIPLE SELECTION] Which of the following are primary contributors to your stress level when working on fully-specified programming tasks? (Select all that apply.)
    - Matching the program’s output format exactly to satisfy the autograder
    - Having to develop a program that did not align with my creative ideas
    - Making sure I met every detailed requirement or specification
    - Feeling pressure to avoid small mistakes that could cause the program to fail testing
    - Difficulty understanding or interpreting the specifications
    - Feeling constrained by the lack of flexibility or opportunities to be creative
    - Challenges debugging strict, rule-based behavior
    - Difficulty ensuring my implementation matched the instructor’s and/or autograder’s expectations
    - Limited opportunity to explore alternative solutions
    - Technical challenges
    - Managing time to meet all specified requirements
    - Other (please specify)
    - None of the above
21. [SINGLE SELECTION] For which gameplay modality do you feel the strongest sense of accomplishment or pride in the work you completed?
    - The open-ended gameplay task led to a much greater sense of accomplishment and pride than the fully specified task
    - The open-ended gameplay task led to a slightly greater sense of accomplishment and pride than the fully-specified task.
    - The open-ended and fully-specified gameplay tasks led to a similar level of accomplishment and pride.
    - The fully-specified gameplay task led to a slightly greater sense of accomplishment and pride than the open-ended task.
    - The fully-specified gameplay task led to a much greater sense of accomplishment and pride than the open-ended task.
22. [MULTIPLE SELECTION] Which of the following are primary contributors to your sense of accomplishment when working on open-ended programming tasks? (Select all that apply.)
    - Cultivating a unique experience for the program user
    - Applying concepts to my own creative ideas to develop impactful, real-world applications
    - Solving challenging problems independently
    - Designing elegant, efficient, or well-structured code
    - Exploring multiple approaches and making design decisions
    - Overcoming technical obstacles or debugging difficult issues
    - Creating something original that reflects my personal style or ideas
    - Receiving positive feedback from peers, instructors, or users
    - Completing a task that matches or exceeds my own expectations
    - Other (please specify)

- None of the above

23. [MULTIPLE SELECTION] Which of the following are primary contributors to your sense of accomplishment when working on fully-specified programming tasks? (Select all that apply.)
    - Matching the program output exactly to the specified requirements
    - Achieving green check marks or passing all autograded tests
    - Completing the task correctly and efficiently
    - Debugging and resolving errors or unexpected behavior
    - Meeting all specified criteria without missing any details
    - Developing a correct solution under time constraints
    - Successfully following a step-by-step process to achieve the expected result
    - Gaining confidence in applying programming concepts accurately
    - Receiving confirmation the system or instructor that the solution is correct
    - Other (please specify)
    - None of the above

Questions 24-25 [SINGLE SELECTION] have identical answer options:

- A great deal
- A lot
- A moderate amount
- A little
- None at all

24. How important is it for you to have creative freedom when developing programs to solve problems?
25. How important is it for you to have a sense of security that comes from knowing your program is accurate when solving a problem?
26. [FREE RESPONSE] Use this space for any additional thoughts, suggestions, challenges you faced, or notes regarding the open-ended programming tasks in CS 211 and their potential future implementations.

GEN-AI USAGE:

Questions 27-30 [NO or YES] have identical answer options: NO or YES

27. Did you use genAI to generate ideas for the open-ended programming task?
28. Did you use genAI to develop code for the open-ended programming task?
29. Did you use genAI to debug your program for the open-ended programming task?
30. Did you use genAI to write comments for the open-ended programming task?
31. [SINGLE SELECTION] In general, what is your level of generative AI (genAI) use to complete programming projects in this course is?
    - Never - I do not use genAI at all
    - Rarely - I use genAI only occasionally
    - Sometimes - I use genAI for some parts of my projects
    - Often - I use genAI for most parts of my projects
    - Always - I rely on genAI for all or nearly all of my project work
32. [FREE RESPONSE] Describe your general use of genAI for completing programming projects in this course? In what ways is genAI beneficial or detrimental to your learning?

Questions 33-42 [SINGLE SELECTION] have identical answer options:

- Absolutely confident
- Mostly confident
- Fairly confident
- 50/50
- Slightly confident
- Mostly not confident
- Not confident at all

SELF-EFFICACY:

33. I could complete a programming project if I had a lot of time to complete the program.
34. I could complete a programming project if someone showed me how to solve the problem first.
35. I could debug (correct all the errors) a long and complex program that I had written, and make it work.
36. I could comprehend a long, complex multi-file program.
37. I could write a program that someone else could comprehend and add features to at a later date.
38. I could make use of a pre-written function, given a clearly labeled declaration of the function.
39. I could manage my time efficiently if I had a pressing deadline on a programming project.
40. I could understand the language structure of C/C++ and the usage of the reserved words.
41. If given a specific project task with a starter code template that is scaffolded into individual tasks and an autograder to check my work, I could develop fully-functioning code for that specific task.

42. If given only a set of overall project goals (with NO starter code template and NO autograder), I could develop a fully-functioning program from scratch.

PREPAREDNESS AND BELONGINGNESS:

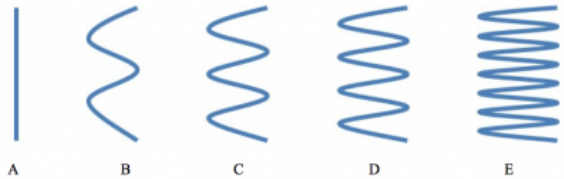
43. [MULTIPLE SELECTION] Major: What is your major?
    - Computer Science
    - Data Science
    - CS + Design
    - CS + Linguistics
    - Other (please specify)
44. [MULTIPLE SELECTION] Minor: If you have a minor, select it below. If you do not have a minor, select 'N/A'.
    - Computer Science
    - Business
    - Mathematics
    - Other (please specify)
    - N/A
45. [SINGLE SELECTION] Class Standing: What is your current class standing?
    - Freshman (0-29 earned college credits)
    - Sophomore (30-59 earned college credits)
    - Junior (60-89 earned college credits)
    - Senior (90+ earned college credits)
46. [SINGLE SELECTION] Prior CS Courses: How did you earn credit for CS 141 - Programming Design II, which is the only prerequisite for this course?
    - I took CS 141 at UIC
    - I took CS 207 at UIC
    - I took a class at a community college that transferred credit for CS 141
    - I took a class at another four-year college that transferred credit for CS 141
    - I took a high school class and/or took an AP test that transferred credit for CS 141
    - I took a proficiency exam at UIC that gave credit for CS 141
    - Other (please explain)
47. [SINGLE SELECTION] Prior Exposure to C: My exposure to Programming in C now in this class is best described as:
    - Non User (zero exposure)
    - Basic User (knows some functionality)
    - Intermediate User (confidently programs in C)
    - Proficient User (utilizes C at work-level intensity)

Questions 48-49 [SINGLE SELECTION] have identical answer options:

- Strongly agree
- Somewhat agree
- Neither agree nor disagree
- Somewhat disagree
- Strongly disagree

48. Confidence: I feel confident about succeeding in CS 211.
49. Growth mindset: I believe I can improve my CS skills with effort.
50. [SINGLE SELECTION] Belongingness: Given the following images, which picture best represents how well you feel you fit in and belong in this CS course?

51. [SINGLE SELECTION] Belongingness Variability: Think about how you feel about yourself at different times. Some people pretty much always feel the same way about themselves. Other people feel differently about themselves at different times. The lines below represent a person's feelings of belonging in this CS course. The straight line represents someone who always feels the same about how much they belong in this CS course. The curved lines represent someone who sometimes feels like they belong in this CS course and sometimes feels like they don't belong in this CS course. The more curved the line, the more a person's feelings about their belonging in this CS course vary. Select the line that best represents how much your feelings about whether you belong in this CS course are different at different times.



DEMOGRAPHICS:

52. [SINGLE SELECTION] How old are you?
    - Under 18
    - 18-24 years old
    - 25-34 years old
    - 35-44 years old
    - 45-54 years old
    - 55-64 years old
    - 65+ years old
53. [SINGLE SELECTION] How do you describe yourself?
    - Male
    - Female
    - Non-binary / third gender
    - Prefer to self-describe
    - Prefer not to say
54. [MULTIPLE SELECTION] Which of the following best describes your race or ethnicity? (Select all that apply.)
    - Hispanic, Latino, or Spanish origin
    - White or Caucasian
    - Black or African American
    - Asian
    - Middle Eastern or North African
    - American Indian/Native American or Alaska Native
    - Native Hawaiian or Other Pacific Islander
    - Another race or ethnicity (please specify)
    - Prefer not to say
55. [SINGLE SELECTION] What is the highest level of education completed by any of your parent(s) or guardians(s)?
    - Some high school or less
    - High school diploma or GED
    - Some college, but no degree
    - Associates or technical degree
    - Bachelor's degree
    - Graduate or professional degree (MA, MS, MBA, PhD, JD, MD, DDS etc.)
    - Prefer not to say