Empowering Computing Students with Large Language Models by Developing an Escape Room Game

Nasser Giacaman
n.giacaman@auckland.ac.nz
University of Auckland
Auckland, New Zealand

Valerio Terragni
v.terragni@auckland.ac.nz
University of Auckland
Auckland, New Zealand

Course HCI
Programming Language Java
Knowledge Unit Programming Concepts
CS Topics Human Computer Interaction, Object-Oriented Programming, Design Patterns and Program Organization
Resource Type Project

Synopsis

In this project, computing students learn to integrate large language models (LLMs) into a software system. Students develop a Java application with a basic graphical user interface (GUI) using JavaFX, gain practical experience with prompt engineering, and learn about the impact of LLM parameters and conversational roles. Students are provided with a Java-based API that connects with OpenAI's GPT model. The project emphasizes teaching students to manage LLM API calls, enhance GUI responsiveness, and improve the user experience all in the context of an AI-powered application. This experience equips them with critical skills in software development and AI application. It prepares them for advanced software development by learning how to create effective LLM prompts to create intelligent and user-friendly applications. We share the experience of using this project and provide guidelines for assessing it in a second-year software engineering undergraduate course, where students' prior programming experience is limited to the prerequisite CS2 course on object-oriented programming. In the case study we present, the project involved developing a riddle-solving escape room, which we called *EscAIpe Room*.

ACM EngageCSEdu, January 2025.

ACM ISBN 979-8-4007-1445-0/2025/01
https://doi.org/10.1145/3708897

Keywords

Computing Education, Large Language Models, Prompt Engineering, Project-Based Learning, Human-Computer Interaction, Java

ACM Reference Format:
Nasser Giacaman and Valerio Terragni. January 2025. Empowering Computing Students with Large Language Models by Developing an Escape Room Game. In *ACM EngageCSEdu.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3708897

1 Engagement Highlights

The project aligns with the NCWIT Engagement Practices Framework [5] through the following teaching practices:

Using Meaningful and Relevant Content: The project engages students by integrating the latest AI technologies, such as OpenAI's GPT model. By working on a riddle-solving escape room, students discover the potential of LLMs and how to best integrate them to create interactive conversational applications. This real-world practical experience motivates students to learn about AI, and help them better understand the capabilities and limitations of LLMs. The conversational nature of the application also raises the importance of students considering the user experience in software design, which is a key aspect of human-computer interaction.

Incorporating Student Choice: By only providing students with simple starter code, the project requires them to be creative and extend the application in a number of ways. Students need to propose their own theme for the escape room, then design additional rooms and their own riddles. This autonomy encourages students to pursue a direction most interesting and meaningful to them, allowing them to take ownership of their learning. Instructors should still impose constraints to guide students in scope, depending on course level and time limitations. For example, must have three rooms, and riddles should be solvable within two minutes.

Make Interdisciplinary Connections to CS:. The open-ended nature of the project allows making connections with various fields. Within computing, cross-disciplinary fields could

include *computational linguistics* and *human-computer interaction*. Outside computing, cross-disciplinary fields could include *cognitive science*, where students explore how humans solve problems, or *psychology*, which provides insight into how humans behave in escape rooms.

2 Project Modifications

One of the key strengths of this project is its flexibility, which allows instructors to modify the starter code and requirements to suit student abilities and learning objectives. We categorize possible modifications as follows.

2.1 Minor Modifications

The simplest modifications involve replacing the existing room images with new ones. This simple alteration is an immediate visual change, which could be themed to match particular interests or holidays, such as Christmas or Halloween. Instructors can also easily modify the starting prompt and riddle, for example having a history-themed escape room with riddles related to historical events.

2.2 Extensions and Scalable Complexity

Depending on the course level and time constraints, instructors can easily scale the complexity of the project by requiring students to implement additional features. For example, students could be tasked with increasing the number of rooms and riddles. Advanced students could be required to create an experience that requires multiple interactions to solve. Additional features could be integrated, such as a timer, hint system, or a multi-user scoring system. More technical students could be tasked with integrating speech synthesis and recognition, integrating the application with a database to store user progress, or integrating with free public APIs [6] to provide additional functionality.

2.3 Core Essence of the Application

The initial prompts are the core of the application. By changing these prompts, the core essence of the application changes from an escape room to a different type of conversational application. The starter code, however, still remains as a solid foundation for the project. Here are alternative ideas to an escape room, purely based on changing the initial prompts:

Educational Quiz. The AI asks questions related to a specific topic, and the user is challenged to answer them. Examples include a quiz on history questions or language vocabulary.

Interactive Storytelling. The user and the AI take turns in creating a story. Based on how the user responds, the AI continues the story in a different direction.

Virtual Museum Tour. By integrating elements of art and history, the user can select different artifacts to learn more about them, and ask the AI questions about them.

2.4 Future Updates

In addition to the downloadable stater code, we also provide a dedicated GitHub repository [3] to ensure future updates and modifications. For example, as OpenAI releases new models or updates to their API, we will update the GitHub repository with the latest version of the API wrapper to ensure continued compatibility with the project. Instructors can also contribute to the repository by adding new features, fixing bugs, or providing additional documentation.

3 Recommendations

We provide specific recommendations and advice for instructors who may want to adapt this project for their own course.

3.1 Prerequisite Knowledge

We recommend that students have a basic understanding of Java programming, including object-oriented concepts (in particular, classes, inheritance, polymorphism, and basic data structures). It is not necessary for students to have prior experience with JavaFX, as the starter code provides a basic GUI that students will build on. Similarly, students do not need prior experience with cloud-based APIs or large language models, as the project provides a simple wrapper that students will use to interact with OpenAI's GPT model in an object-oriented manner.

3.2 Time Management

Depending on the course level and the desired complexity of the project, instructors should allocate between 5–10 weeks for this project. This would allow students to become familiar with JavaFX and concurrency in JavaFX (1–2 weeks), explore the starter code and LLM interactions (1–2 weeks), and then implement extensions for the application (3–6 weeks) as defined by the instructor. For example, an "*alpha*" version could be developed individually by the end of the first month, followed by a "*beta*" version developed in groups.

To put into context, the example applications shown in Figure 1 were developed by students after a total of 10 weeks of work (includes learning JavaFX, prompt engineering, and developing the application in the alpha and beta versions). The additional time is particularly encouraged to allow students to experiment with different features, explore different prompts, and refine the user interface.

3.3 GUI Development

We strongly recommend that students use JavaFX Scene Builder [4] for the development of the GUI. Scene Builder

Figure 1: Examples of student-created applications that built on the provided starter code. Students are encouraged to be creative and add their own features to the application.

provides a simple drag-and-drop interface that simplifies the design without requiring manual coding of the GUI components. Instructors should guide students to initially focus on core functionality, such as adding a hint system and countdown timer, rather than initially investing too much time in the visual aspects. This ensures that essential features are operating, and provides a solid foundation for the project.

It is also important for students to learn and implement GUI concurrency. Without this, the user interface will freeze during long-running operations, such as when waiting for results to return from the LLM. To ensure that the interface remains responsive, background threads should be used to ensure the main application thread is not blocked. JavaFX provides good support with `javafx.concurrent.Task`.

3.4 Prompt Engineering

We recommend that instructors include a lecture on prompt engineering techniques. In our course, we conducted a two-hour lecture that covered (i) a high-level and simplified explanation of how LLMs work, which is important for understanding their limitations and capabilities, (ii) the role of the parameters `temperature` and `top-p` in tuning the creativity or determinism of LLM responses, and (iii) basic prompt engineering techniques such as zero-shot, few-shot, and chain-of-thought prompting. We have included this material for instructors to use in the supplementary materials.

3.5 Ethical Considerations

While integrating OpenAI's GPT into the project provides valuable learning, we also acknowledge potential concerns.

Privacy Concerns. Some students may have privacy concerns with cloud-based models such as OpenAI's GPT. This concern can be reduced by ensuring the project theme is one that does not require any personal or sensitive conversations. By focusing on neutral topics like riddles, students can engage with the LLM without involving private data.

Cost Barriers and Fair Access. Although OpenAI requires payment, we managed this by providing individual API keys to students and capping the usage at $1 per student (covered by the department's central OpenAI account). This amount was more than sufficient when restricted to cost-effective models such as `gpt-4o-mini`, and could be reduced further for smaller projects. This approach ensured that no student was disadvantaged due to financial constraints or lack of a credit card, and that all students had equal access to the API.

Content Appropriateness. Since we allowed students to select any theme for their project, it is important to require that such themes are appropriate without being sensitive or offensive. We included explicit wording in the handout to ensure students choose suitable content for their applications.

Compliance with OpenAI's Usage Policies. We recommend reminding students of OpenAI's usage policies[1], which require that applications do not generate harmful content. The OpenAI API also includes built-in content moderation; if inappropriate content is detected, the API does not complete the request and returns `content_filter` as the stop reason. On top of that, OpenAI also offers a free Moderation API[2] that can be used to detect and filter inappropriate content.

3.6 Use of AI Tools in Development

Whether students use tools like GitHub Copilot or ChatGPT to generate code for their application is a course-specific decision left to the instructor. In our experience, we allowed and even encouraged students to utilize these tools during development. The primary objective of this assignment is for students to learn how to incorporate LLMs into the software they develop, which is independent of whether they use generative AI to assist in writing code.

Allowing the use of AI coding tools reflects real-world practices where developers are expected to leverage such

[1]https://openai.com/policies/usage-policies
[2]https://platform.openai.com/docs/guides/moderation

technologies to enhance productivity. We believe that gaining experience with AI-assisted development is beneficial for students as it prepares them for modern software engineering workflows [1, 2]. However, instructors should establish and communicate clear policies regarding using such tools.

3.7 Team Collaboration Tools

If instructors wish to assign this project as a group project, we recommend providing guidance on collaboration tools. *GitHub* encourages good software development practices, enabling students to follow a branch workflow, use pull requests for code reviews, and manage issues. *Discord* or *Slack* can be used for discussions and sharing resources, while *Trello* can help organize tasks and track progress. Providing guidance on these tools (or similar) can help students work more efficiently as a team and develop skills that are valuable in professional software development environments.

3.8 Pitfalls and Struggles

The main challenge for students was learning JavaFX. While JavaFX is currently the most modern and best framework for developing GUI applications in Java, Java itself is not the optimal language for this purpose. We however chose Java because our students had previous courses in Java, and we did not want them to spend time learning a new language.

Another challenge was prompt engineering; students struggled to craft the right prompts to achieve the intended behavior from the LLMs. Although they used OpenAI's GPT, one of the best LLMs available, it sometimes did not comply with the commands. We have included a lecture on prompt engineering in the supplementary materials to help students understand how to use LLMs effectively.

From the students' perspective, these challenges can be frustrating. However, we believe they are highly beneficial. Hands-on practice with prompt engineering allows students to understand its limitations. LLMs are expected to play a pivotal role in future software development, not only because more software engineers are relying on LLMs to boost productivity in writing code, but also because more companies are incorporating LLMs and other AI tools into their systems. Prompt Engineering will be an essential skill that every software engineering graduate should master.

3.9 Lessons Learned

We provide the following suggestions, based on the context of our course being a second-year full-semester project course.

Balancing Autonomy with Guidance. While we have seen creative autonomy enhance motivation, we found it is important to balance this with some structured guidance. As instructors, we achieve this by acting as clients to help students define realistic scope. This allows us to make decisions on project boundaries, so as to prevent students from pursuing an overly ambitious project.

Supporting Hands-On Learning. This project demands a lot of hands-on learning. While this is valuable, it is also challenging for students. We found it important to provide resources to students (such as tutorials on JavaFX basics, and JavaFX concurrency) and scheduling regular check-ins for students to seek more clarity (in the form of "client meetings").

Incremental Milestones. Structured milestones are important to help students develop something impressive. The *alpha* version was a solo endeavour to ensure all students got exposed to the necessary technologies. The *beta* version transitions to a team effort, expanding the requirements. The *final* version does not introduce new requirements, but instead requires students to polish the usability of their application.

Understanding Client Requirements. Given the technical challenges that students might face, we believe a "less is more" approach emphasizes simplicity. This provides students with more clarity to achieve a quality outcome. Using lectures as interactive Q&A sessions allows students to seek clarification, and instructors (as clients) define achievable goals.

4 Student Experience

4.1 Methodology

A total of 120 second-year undergraduate students enrolled in a software engineering course for computing majors were enrolled in the course. All students had completed a prerequisite CS2 course on object-oriented programming in Java and had limited prior programming experience beyond that. To gather insights into the student experience, we analysed the written reflections submitted at the end of the project. One researcher reviewed all the student reflections using an open coding approach. Recurring ideas and observations were identified and grouped into overarching themes. Both positive and negative sentiments were noted for each theme to provide a balanced view of the student experience.

4.2 Findings

The analysis revealed several key themes in the students' feedback, including engagement and enjoyment, creative freedom and autonomy, and challenges related to learning new concepts and technologies. Students appreciated the opportunity to work with cutting-edge AI technologies while reinforcing their programming skills. Below are key themes, in both positive and negative perspectives:

Engaging and Fun.

> **Positive:** *"The project style of the course was very engaging and fun to do."*

Negative: *"The most challenging thing about this course was that once you started, it was hard to stop and it was easy to start getting carried away and then end up not finishing everything you wanted to finish."*

Creative Freedom and Autonomy.

Positive: *"The creativity they gave us allowed me to learn how to code with something I am interested in. This made me passionate."*

Negative: *"There were times when my code wasn't working as I intended and I wasn't sure how to fix it because the technique I was using hadn't been taught in the lectures."*

Challenging Learning Curve and Self-Study.

Positive: *"Being left to our own devices and search up what we needed for our project [was helpful for my learning]."*

Negative: *"We only learnt basic JavaFX skills. So I had to search on Google to get cool functions. Finding new functions was a challenge."*

Practical Experience and Skill Development.

Positive: *"[Instructors] only taught us the bare minimum to do with JavaFX and the rest was up to us. This made it necessary to do some self study and learn some skills using resources online to achieve the end result I had in mind."*

Negative: *"Learning multithreading and concurrency (especially for GPT) proved to be very difficult and I feel could be a topic that gets spent more time on throughout the year."*

Grappling with Prompt Engineering.

Positive: *"We had the ability to adjust the game master's personality (powered by the GPT) via prompting. This provided the player with a more authentic experience by making the game more dynamic and alive."*

Negative: *"When I design prompt engineering for the GPT, I feel like I am dealing with an unpredictable word processor. Always random and hard to predict."*

5 Materials

In this section, we summarize the handout and starter code provided to students to help guide them complete the project.

5.1 Handout

The `handout.pdf` file is provided to students to make sure they have a clear understanding on the key requirements. In addition to the project description and requirements, it also includes a background story to the (simulated) clients, played by the instructors. This provides students with a realistic development scenario, detailing why the clients are hiring the students to develop the application. This adds a layer of realism to the project, aiming to replicate a real-world scenario where understanding client needs is important. For further guidance on the manner in which this is conducted, refer to our publication on this teaching approach [7].

5.2 Starter Code

The starter code is included in the `starter-code.zip` file. The project is structured as a *Maven* project, and only requires students to have *Java JDK 21+* installed on their machines. They are however encouraged to use *Scene Builder* for designing the graphical user interface of the application, and *VS Code* as their IDE. The rest of the libraries and dependencies are included in the project's Maven configuration file. The project includes the following core files:

- `README.md` - Instructions for running the project.
- `pom.xml` - Maven configuration file.
- `apiconfig.properties` - Configuration file storing credentials to access the OpenAI API, to be created from https://platform.openai.com/account/api-keys.
- `src/main/java/.../App.java` - Main class to run.
- `src/main/java/.../controllers/RoomController.java` Controller class associated with `room.fxml`, which is the main view of the application.
- `src/main/java/.../controllers/ChatController.java` Controller class associated with `chat.fxml`, which is the chat view of the application.
- `src/main/java/.../gpt/openai/*.java` - Classes that interact with the OpenAI GPT API.
- `src/main/resources/fxml/room.fxml` - FXML file for the room view, to be designed using Scene Builder.
- `src/main/resources/fxml/chat.fxml` - FXML file for the chat view, to be designed using Scene Builder.

The provided Maven wrapper runs the project, where the following command compiles and runs the application:

```
./mvnw clean javafx:run      # macOS/Linux
.\mvn.cmd clean javafx:run   # Windows
```

An example of the starter code running is shown in Figure 2 and Figure 3. The application is a simple escape room game where the user interacts with the room to solve a riddle and escape. The user interacts with the AI through a chat interface, and the AI responds to the user's guesses. The user can also ask for hints, and the AI will provide them until the user solves the riddle.

The application provided to students is a starting point, and they are encouraged to extend it by adding more rooms,

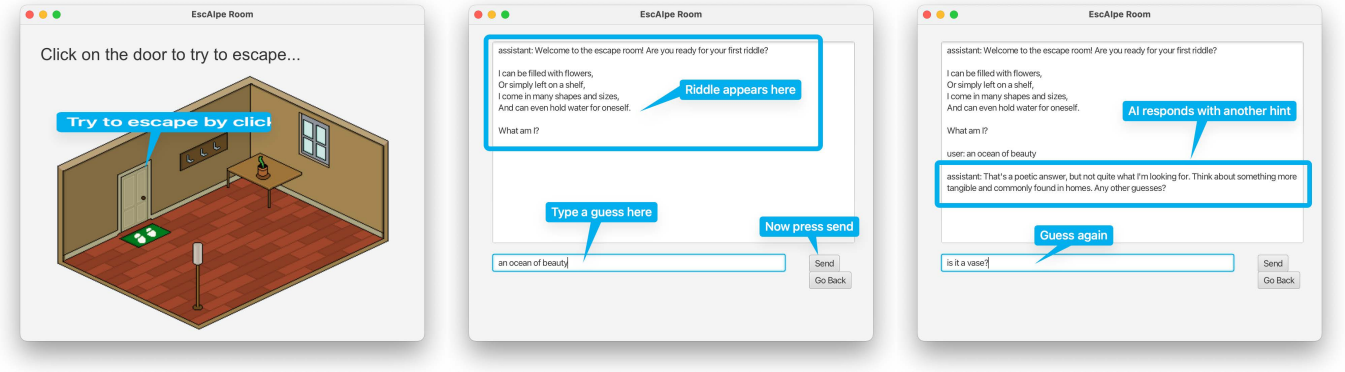

Figure 2: Screenshots of the starter code provided to students. The first image shows the main view where the user interacts with the escape room. The second image shows the chat view, where the user interacts with the AI by guessing the riddle. The third image shows a response to an incorrect guess, while the user is about to guess again.

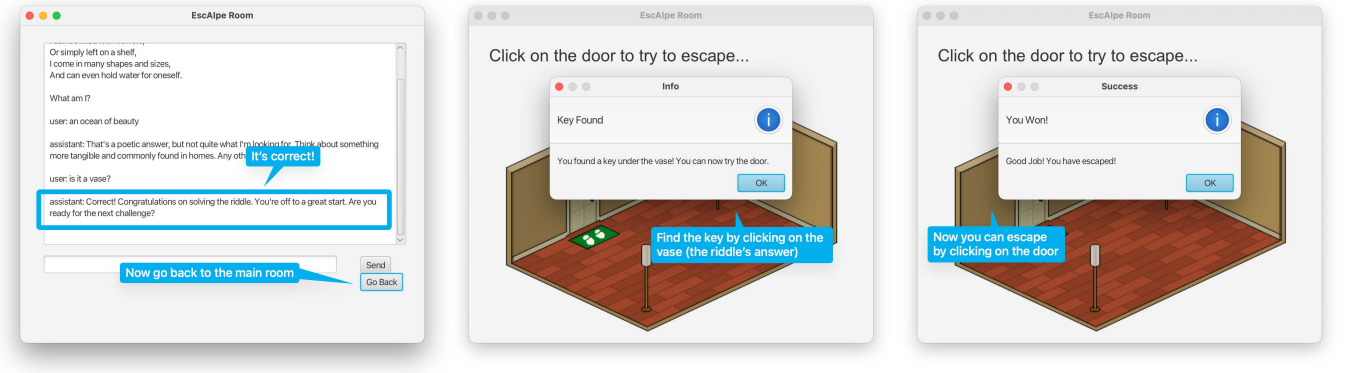

Figure 3: The riddle is correctly solved in the first image, and the user is congratulated. In the second image, the user goes to where the key is hidden, and in the third image, the user is congratulated for escaping the room.

more riddles, and more interactions with the AI. They can also add more features to the application, such as text-to-speech, or a more complex graphical interface.

Figure 1 shows screenshots of three final student-created applications. These applications are the result of the students' creativity and effort in extending the provided starter code. It is impressive to see the variety of features that students added to their applications, such as different themes, more rooms, and more complex interactions with the AI. What is more impressive is that these applications were developed by students with limited prior programming experience, and in a relatively short period of time (about 10 weeks).

References

[1] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1136–1142. doi:10.1145/3545945.3569823

[2] Paul Denny, James Prather, Brett A. Becker, James Finnie-Ansley, Arto Hellas, Juho Leinonen, Andrew Luxton-Reilly, Brent N. Reeves, Eddie Antonio Santos, and Sami Sarsa. 2024. Computing Education in the Era of Generative AI. *Commun. ACM* 67, 2 (Jan. 2024), 56–67. doi:10.1145/3624720

[3] Nasser Giacaman and Valerio Terragni. 2024. EscAIpe Room assignment for EngageCSEdu. https://github.com/Digital-Educational-Engineering/engagecsedu-escaipe-room. Accessed: 2024-12-17.

[4] Gluon. 2024. Scene Builder. https://gluonhq.com/products/scene-builder. Accessed: 2024-12-17.

[5] National Center for Women & Information Technology (NCWIT). 2024. Engagement Practices Framework. https://ncwit.org/engagement-practices-framework. Accessed: 2024-12-17.

[6] Public APIs. 2024. A collective list of free APIs. https://github.com/public-apis/public-apis. Accessed: 2024-12-17.

[7] Valerio Terragni, Catherine Watson, Nicholas Rowe, and Nasser Giacaman. 2023. Fostering Professionalism in Software Engineering: An Early-Exposure Approach. *IEEE Software* 40, 6 (2023), 47–54. doi:10.1109/MS.2023.3291711