COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

The Programmers of Madagascar

Seth Krause

Dylan Cormican

Landon Brown

Colin Brush

Revised December 10th, 2023

HiLabs Logo

CSCI 370 Fall 2023

Prof. Kathleen Kelly

Table 1: Revision history

Revision	Date	Comments
New	09/03/2023	Completed Requirements Doc
Rev – 2	09/17/2023	Laid out preliminary design components in System Architecture section
Rev – 3	10/20/2023	Finished our quality and ethics section of the document
Rev – 4	11/12/2023	Completed the results section
Rev - 5	12/10/2023	Final touches

*** see appendix**

Table of Contents

*** see appendix**

I. Introduction

This is a document of HiLabs Content Gamification product. The Content Gamification product is a tool that reads in open-source educational content, such as lecture slides, textbook pages, etc. and utilizes a LLM* to convert the content into a web-based game that will make learning the content more fun and engaging. It was designed with the ultimate goal of integrating it within a larger 'virtual TA' product created by HiLabs to be used by college students.

It consists of 4 components: a game, a web application, an interface for the LLM, and a server. The game was built using GameMaker*, the web application with React*, the LLM interface with Python + Langchain*, and the server using Python + Flask*.

II. Functional Requirements

- Take open-source educational content as input, such as lecture slides, textbook PDFs, etc. into an LLM.
- Utilizes prompt engineering* to tailor output from LLM to create a web-based game, which makes content more fun and engaging for the learner/end-user.
- The time it takes to create the game file from the lecture content should take no more than 1-3 minutes.
- Have the ability to edit game files once they have been generated.
- Game files should be able to be downloaded and uploaded in-place of being generated every time.

III. Non-Functional Requirements

- Modularity, product will need to be able to integrate within the larger virtual TA platform, and account for developing and new LLMs that may come out. Additionally, we may want to use the output from the LLM to create a variety of games, not just a particular type. So LLM prompts must be engineered in such a way that their outputs can be used in a variety of ways.
- Usability, web interface should be easy and intuitive to use.
- Maintainability, code should be well-commented and documented to keep later developers up-to-date.
- Utilizes Python, Typescript, and Javascript; is consistent and compatible with existing HiLabs software.

*** see appendix**

IV. System Architecture

Extractor (LLM):

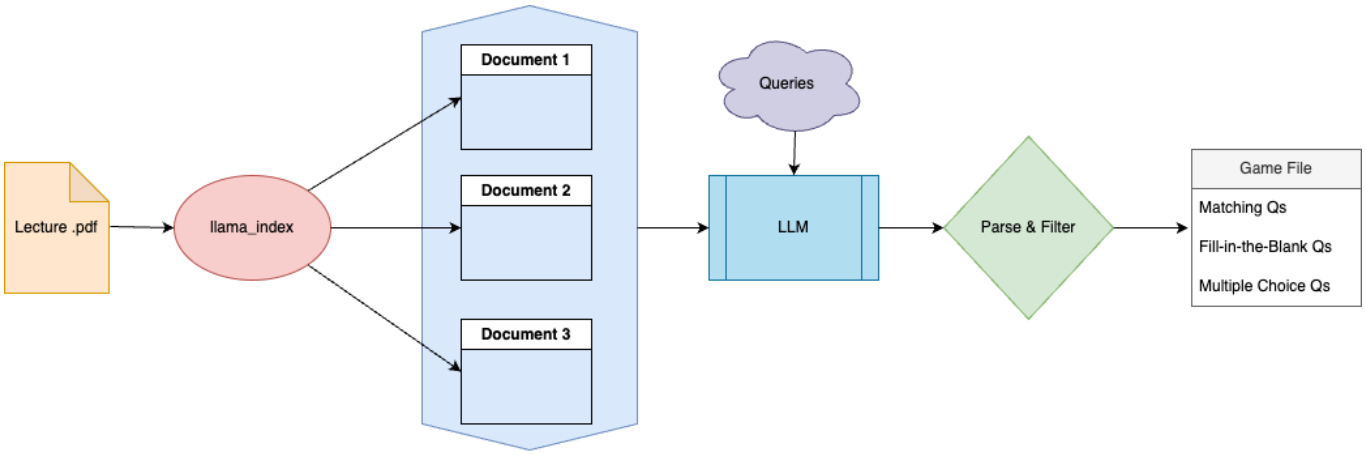

Figure 1: Extractor Architecture

Overview:

- The Extractor takes in lecture content in the form of a PDF or text file and utilizes a LLM to convert it into a game file that tells the game how to build itself and what questions to include.
- The extractor class was split up into different methods which are described below

Component	Description
llama_index*	Takes in lecture content and converts it into vector embeddings*
documents	Vector embeddings fed into the LLM along with a series of queries
LLM	Large Language Model used to extract questions from lecture. We used OLlama, a MacOS based application that enabled us to run Meta's Llama-2 model locally.
Queries	Prompts asking LLM to extract certain types of questions and output in json format.
Parse & Filter	Series of methods used to breakdown LLM string output into JSON objects and write them to a game file. Game file is then uploaded to the server

*** see appendix**

Web Application:

Figure 2: React Application Breakdown, Including File Upload, Load Bar and Edit Widgets

Overview:

- The web application was built using React.js and is split up into separate widgets, or components, according to functionality.
- Each widget is placed on a home splash screen, a loading page, and an edit page based on where they are needed and what information the user might require on that page.

Component	Description	Location
File upload	Allows the user to drag and drop or select a file from their file system to upload to the server via a POST request. Can be either a lecture file (pdf, txt) or a game file (json).	Home page
Load bar	Receives updates from the server about the LLM's progress and displays them.	Loading page
Edit cards	Allows user to edit the questions from the generated game files	Edit page
Disclaimer	Message that informs users they must have the rights to the content they upload.	Home page, loading page
Instructions	Walks the user through how to use the web app. Available after scrolling down.	Home page

*** see appendix**

Game:

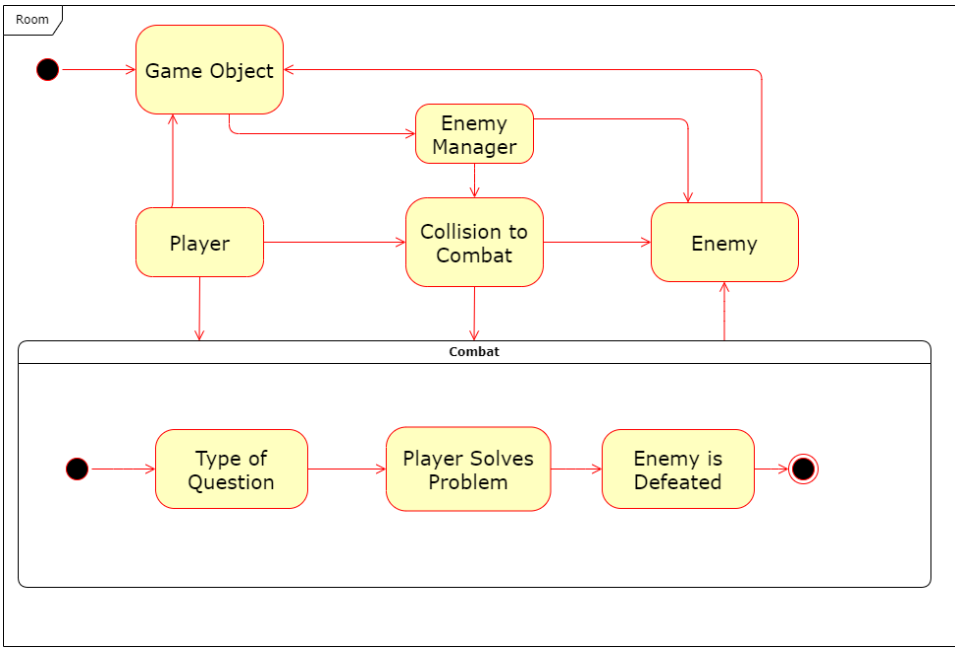

Figure 3: Game Flowchart

Overview:

- The game takes in data extracted from the lecture materials and presents it to the user.
- It is a dungeon crawler type game which revolves around answering questions and defeating enemies to progress.
- There are 3 question types: matching, multiple choice, and fill-in-the-blank; which correspond to different enemies: skeleton, ghost, and slime respectively.
- Each question/enemy type corresponds with a minigame to go along with the question.
- Each minigame takes the player through a series of questions to defeat the enemy and progress in the lesson.
- Each room contains one of each question type. Once the player has answered all the questions in the room, they progress to the next one through a portal and gain one heart.
- By default, there are 10 matching questions, 5 multiple choice, and 5 fill-in-the-blank.
- The player wins the game once they have answered all the questions correctly, or loses once they run out of health from answering questions incorrectly.
- This game is designed in such a way that it makes learning fun, while also ensuring efficiency, avoiding the idea of too much game, not enough learning and vice versa.

*** see appendix**

Server:

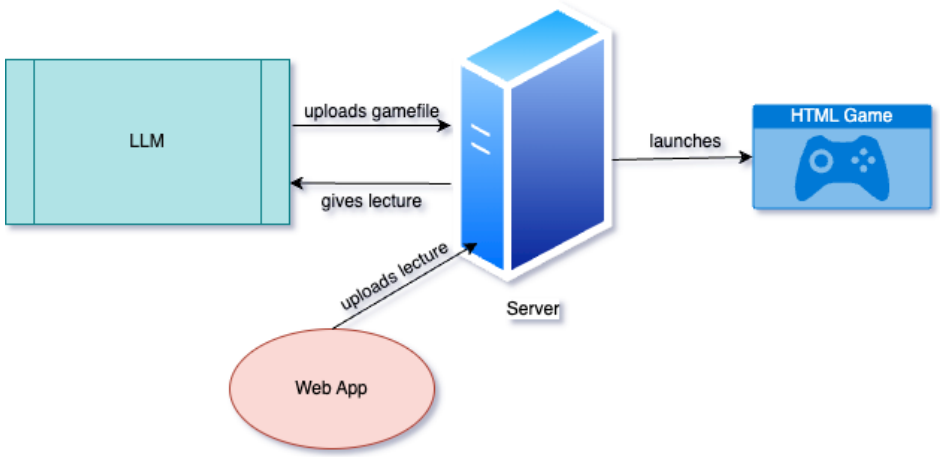

Figure 4: Server Diagram

Overview:

- The server ties all of the components together.
- Hosts the HTML5 game built using GameMaker on a Flask server.
- Provides routes for the LLM and web app to upload or pull files.
- Hosts and calls the Extractor script.
- Updates the web app loading screen with progress from the Extractor.
- Handles hosting game and web application via a Flask application

V. Technical Design

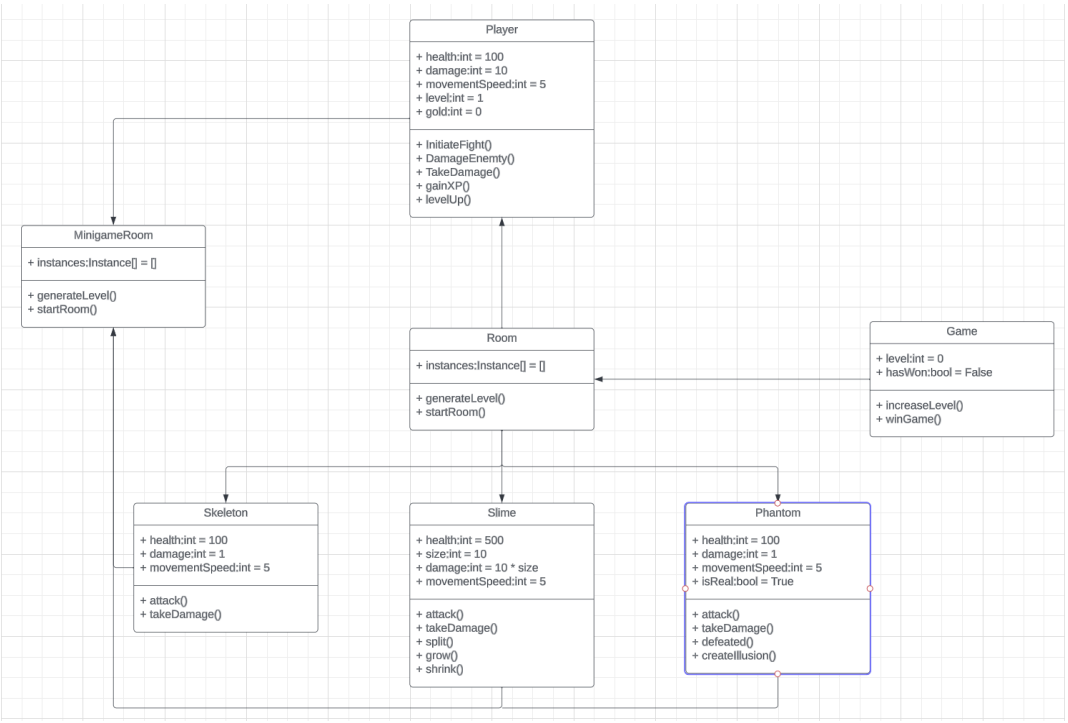

Figure 5: Game Class Structure

*** see appendix**

The class structure above is a representation of the game and its components at a high level. These components each interact with one another to create the game experience. GameMakerStudio offers an object oriented approach to game development and allows us to implement the above class structure along with a hierarchical structure utilizing the elements needed to create a GameMaker game.

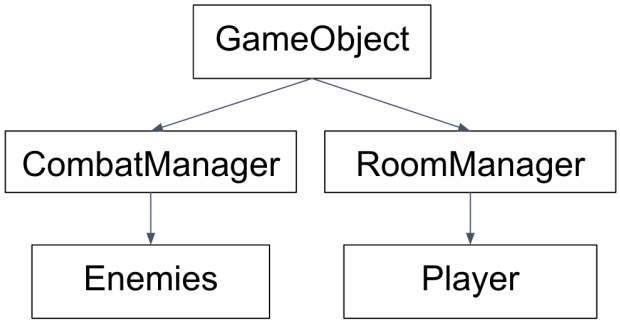

Figure 6: Game Hierarchical Structure

The hierarchy structure shown above in Figure 6 displays how the game is set up in terms of the objects and managers which are required to make the game function as expected. The GameObject manages the game as a whole, it creates the managers which manage the instances across the game and handles things such as game state. When the game is won or lost the game object is what determines what to do next.

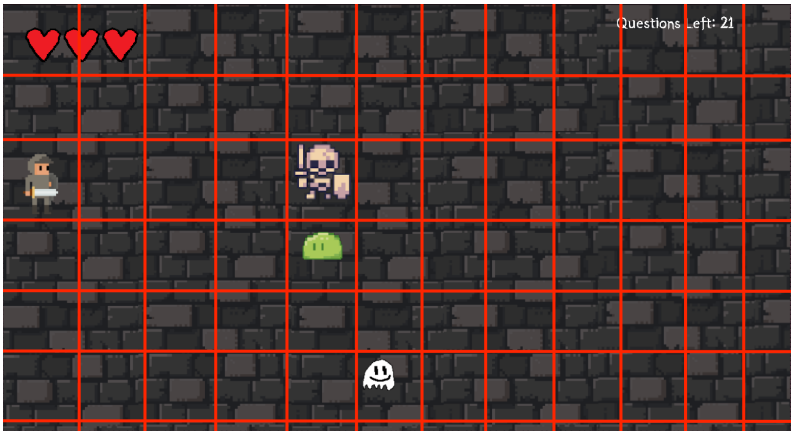

Figure 7: Main game room with drawn grid

The room manager handles spawning enemies and the player and ensuring room switching is completed smoothly. This also manages where enemies are spawned and ensures they do not spawn on top of one another. Enemies are assigned a random position in our positioning grid, where they remain still until the player collides with them, when the room is initially started. A representative example of how the room is generated with the grid can be seen in Figure 7 above. The combat manager handles the enemies and how each of their respective mini games function.

*** see appendix**

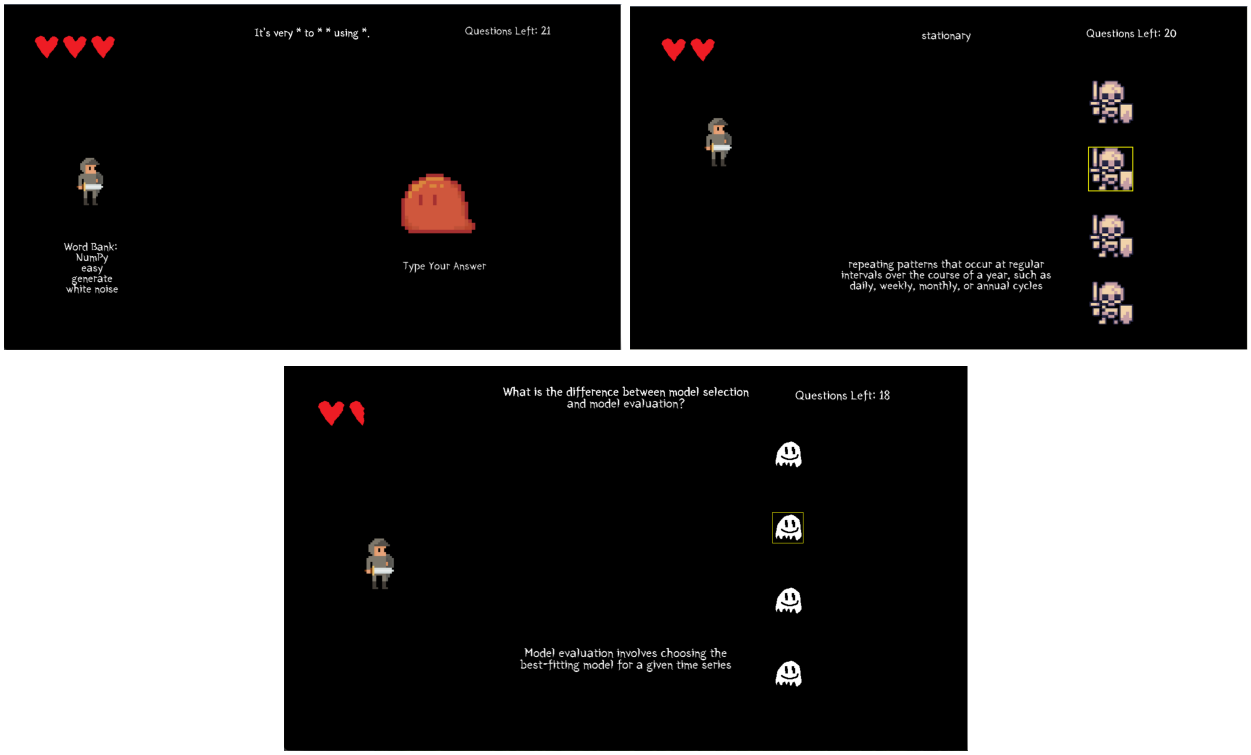

Figure 8: Minigame examples

Shown in Figure 8 are examples of the 3 minigames corresponding to the 3 different question types: multiple choice (ghost), fill-in-the-blank (slime), and matching (skeleton). The multiple choice questions consist of 4 enemies which the player can choose from. One contains the correct answer and will advance the enemy forward whereas the other three enemies will result in the player losing health. The fill-in-the-blank minigame is a game where the user must type in their answer which they feel best corresponds to the blank in the paragraph shown at the top of the screen. This matches a list of answers to the words typed by the user exactly. In the future this would preferably be able to account for simple spelling mistakes. While the player is attempting to match the values, the enemy is constantly moving to reach the player. If the player does not provide all terms needed, they will lose some health. Finally, the matching game consists of key value pairs which can be matched by the user. If the user matches correctly, the enemy is defeated, if not the player loses health.

VI. Software Test and Quality Assurance

To ensure the maximum quality of the product we are producing we implemented different forms of quality assurance across the three separate portions of our project (LLM, web application, game)

Large Language Model (LLM)

Unit Testing

To unit test the LLM portion we wrote tests for each of its methods and its expected outputs:

*** see appendix**

Method	Test/Expected Output
pdf_to_documents	Pass in a pdf, returns a list of objects/vector embeddings
setup_llm	Connects to the LLM using langchain, sets up a query engine using the documents, returns a query engine object.
replace_term_with_asterisks	Given a sentence and a list of terms, correctly removes the terms from the sentence and replaces them with *s. Used for fill-in-the-blank questions.
parse_llm_output	Given a long string as the LLM output, correctly converts the output into JSON question objects.
build_json	Given a list of JSON question objects, correctly converts them into a single JSON game file
upload_json	Correctly uploads the generated game file to the server.
send_progress_update	Sends a POST request to the server which updates it on the LLM's progress.

Output QA

- One of our client's requirements was that the LLM call take no more than around a minute to process. To accommodate this, we have switched over to a vectorized embedding model which requires far fewer LLM calls and meets this time requirement.
- Throughout our process we went through several query prompts and we discovered that the LLM is the most consistent when we ask it to output in JSON format specifically, and then parse around that.
- If we encounter errors, such as if the LLM API is down, the Extractor sends an error update message to the server which then in turn is displayed on the Web App to keep the user informed and maintain transparency.

Web Application

Unit + End-To-End Testing

Similarly to the LLM, we broke down the Web App into smaller components and then tested each as a black box. We used React testing frameworks Jest* and Puppeteer* to test each of the components. Here are some example test cases:

Component	Test
File Drop	We can upload a file and the name of the file now appears on the web page, indicating it was correctly uploaded
Instructions	If the user scrolls down the instructions appear
Make Game	If the user clicks 'make game', the web page is redirected to the /making-game route
Load Bar	Once the loading screen appears, the load bar should receive at least 4 different status updates. Receiving these status updates indicates that the game file has been

*** see appendix**

	generated correctly.
Edit Widget	Expects an edit card to be loaded correctly according to each question type. Browser should prompt the user for a new value when ‘edit’ is clicked.

Game

Our game itself is being built using GameMaker Studio 2 which runs its own C based language to develop simple 2D games. Utilizing its own language and systems, GameMaker does not have a great way to perform unit tests for our game. Due to this the primary ways we will ensure quality here will be through user testing and code reviews.

User Play Testing

- We provided a working, pre-release version of our game to fellow college students, who are the demographic of HiLab’s stakeholders.
- After playing the game, they were asked to answer a survey with some basic questions inquiring about the feel of the game, as well as a section asking about suggestions for how to make the game more engaging as well as reporting any bugs found within

Code Reviews

- Weekly code reviews between the developers took place to ensure the software we are developing is up to standards
- We playtested regularly to ensure any bugs we came across were fixed as quickly as possible

Server

Unit Testing

Similarly to the LLM and Web App, we broke the server down into each of its methods, or routes, and tested each individually. Here is a table of each of the routes and the tests that were written for them:

Route	Test
‘/upload-lecture’	Uploads a mock file and asserts that the response code is 200
‘/upload-gamefile’	Uploads a mock game file and asserts that the response code is 200
‘/get-gamefile’	Server should send a valid json response object and a response code of 200
‘/receive-data’	Server receives an update from the LLM about its status correctly
‘/’	Base URL where the game is run. Should send a response code of 200

*** see appendix**

VII. Results

The goal of our software was to develop a pipeline capable of reading educational content such as lecture slides or class readings and convert the key ideas and main points into a game which could be used to better comprehend course content. In order to make these conversions and read the content effectively, we implemented novel large language model (LLM) technologies to complete these tasks that would normally require a large amount of human intervention. With the LLM, we spent a large amount of time engineering our prompts given to the model to produce the most optimal output. We ensured the quality of our output through continuous quality assurance during the development of our product. We also allowed potential users the chance to check to see if the output quality was up to standard. With this technology being so new, our output did not always make sense. This led us to begin to implement a feature that would allow users to edit the game files before publishing the game. We also spent a large amount of time trying to reduce the amount of time it took for our model to read the content and generate our game. Throughout the course of development we were able to drive the timing of this process from 6 minutes to under a minute by utilizing vector embedding software which allowed the LLM to select what part of the lecture it used to answer the queries more efficiently as opposed to our initial chunking method.

Performance Testing Results:

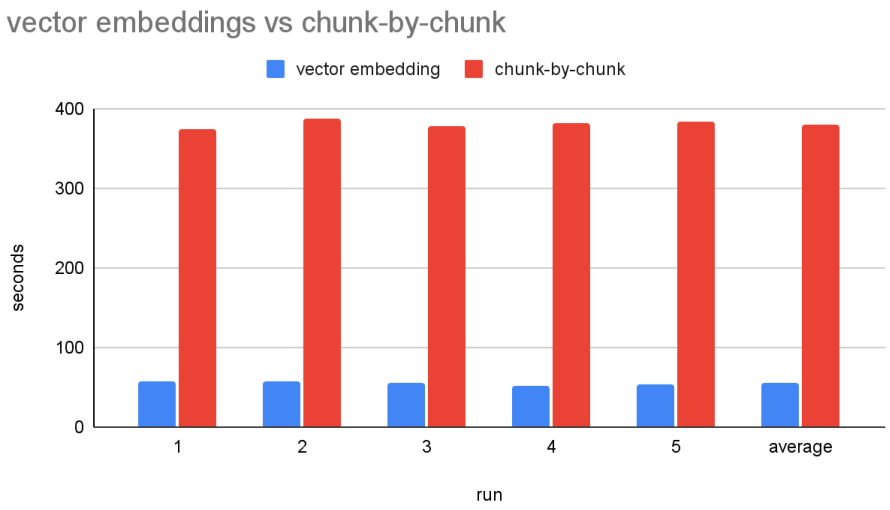

Figure 9: Performance testing vector embedding versus chunk-by-chunk loading

The primary performance testing was done with the LLM to determine the build times of the game between uploading a file and receiving the LLM output. Initial tests utilizing our own chunking method to feed prompts to the LLM proved to be successful but very slow with most runs being close to 6 minutes to complete the game file synthesization. Our client, when laying out the requirements for this project, was hopeful that the synthesis process would only take about a minute to complete so we made some optimizations. We switched over to a vector embedding method that allowed us to much more efficiently send our prompts to the LLM leading to us being able to make a constant 3 calls as opposed to a variable amount of calls depending on the length of the lecture. This cut our times down to just under a minute making the new way of synthesizing the game file six times faster than our initial iterations.

*** see appendix**

Usability Tests:

For our usability tests we took prospective users through a demo experience using an open source psychology lecture found online. The questions were generated and the game was started. We gave a brief introduction about the controls and testers were able to play through a few rounds of our game. Almost all of the users who play tested our game enjoyed themselves and said that they would recommend it for others to use. Most users also reported that they felt as though they learned something during the experience which is the main goal of the product. Most users had a positive experience even if they experienced a few issues.

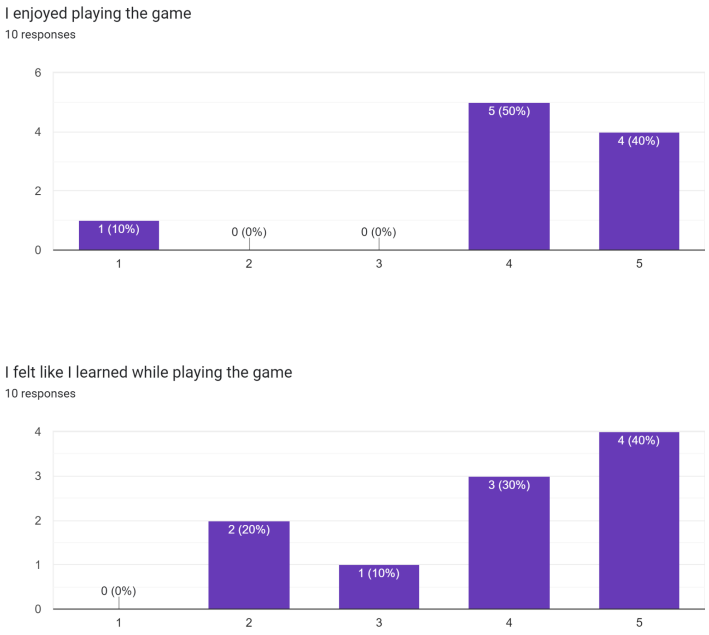

Figure 10: User testing survey results

The issues experienced by users were primarily the result of the repetitive nature of the game as well as some UI bugs here and there. Many questions were repeated multiple times resulting in some redundant gameplay; however, users did note that this helped them remember information better. Although the output from the LLM did prove to be slightly buggy (e.g. sometimes it asked questions that were not completely relevant to the topic, such as “who is the author of this lecture?” and “when was this written?”), overall the game performed quite well throughout the user testing.

VIII. Future Work

Features Unable to Implement:

- Game diversity in terms of enemies and level environments
- Initial difficulty ranking system
- Enemy movement within primary game room
- Enhanced game graphics and controls
- More minigames for each question type
- File storage for games that have been created in the past

*** see appendix**

- User-query tailoring. Ability for users to specify what area of the lecture they would like the questions to focus on.
- Ability for users to specify the difficulty of the questions.
- Ability to not only edit but also create additional questions once LLM processing has been completed.

In the future, the next big step would be to integrate the project with the existing platform HiLabs has created. Making the platform more usable beyond uploading files and playing the game (such as including leaderboards, ability to add questions, etc.) would also be ways we could continue to improve upon our product. Ultimately the goal would be to use it in the context of a college class with a library of games built by the LLM to help make learning the content more fun and efficient. For example, adding a leaderboard system for a class could be extremely helpful in providing some friendly competition between classmates. Competition in this case would strive to be productive, rewarding students for taking the time to study for their classes. In order to do this, a larger backend, database schema, and class systems would need to be built to allow professors or teaching assistants to add lectures and create games from their own content and ensure the quality is up to their standards.

Additionally, as previously mentioned, the LLM output is not always reliable. Aside from outputting questions that don't always make sense, if the Extractor script has an issue parsing a question, it will skip over it and not create a question object out of it. The result is instances of game files that are lacking question types, which can create issues when generating the game. Our team suggests a few ways of working around this. First, is to use a higher-quality LLM that is less likely to deviate from the prompts. Second, is to add a system on the web app to also add questions rather than just editing them. Third is to make the game more fail-safe in that if a certain question type is not present, it will not create an enemy of that question type.

IX. Lessons Learned

- There is a lot of software already available for usage with generative AI technology. Oftentimes problems such as large wait times or inconsistent outputs were able to be solved by discovering some new software which helped improve the model's efficacy. For example, with the help of our advisor, we discovered the llama_index tool which allowed us to make the LLM call times constant and meet our client's requirements.
- UI and usability design are extremely important when it comes to designing a game. If a game is difficult to understand or navigate the odds of it holding a user's attention is extremely slim. In order to meditate this a lot of steps had to be taken to ensure there was enough instruction for the user to play the game effectively and that the controls were easy to grasp.
- Game design takes a lot of time! Throughout the beginnings of the project, designing a game that was both functional and graphically impressive was one of our goals. This however proved to be quite the challenge as building and fleshing out the game systems took more planning and coordination than initially thought. In the end, much of the game's functionality was properly integrated, however there didn't end up being quite enough time to enhance the look and feel of the game as much as we had hoped.
- GameMaker Studio and React applications do not work well together. We built out our web application medium between the LLM and game in React as it was something we had experience with going into the project. We did not realize until later into the development process that hosting our game within the React app proved to be a challenge with little to no documentation on the internet. We were able to

*** see appendix**

work our way around this by hosting the HTML 5 game on the Flask server and achieved a fully functioning pipeline by the end of our project.

- Building a proof of concept (POC) is a lot different from designing a production product. This project was primarily designed to develop a POC for our client to determine whether or not they would move forward with this idea to add to their existing suite of products. A lot of what we thought about going into the project was the end user experience, what it would look like and how we would build it. Ultimately this was not the end point we had in sight, oftentimes we had to take a step back and focus on the important basic functionality before beginning to jump into features that may not be needed just yet.

X. Acknowledgments

We would like to thank our client from HiLabs, Loc Hoang, for providing great insight into the project requirements and helping us gather as much understanding about the task at hand as possible, especially when gathering open source tools. We would also like to thank our advisor, Kathleen Kelly, for helping us stay organized and providing some very useful information about hosting our game on a Flask application and vector embeddings which helped speed up our LLM output wait time.

*** see appendix**

XI. Team Profile

Landon Brown

Major: Computer Science

Work Experience: Software Development Intern at Poly, Software Engineer at Lockheed Martin

Role on the team: Landon was in charge of designing and implementing the Extractor, the server, as well as frontend and backend testing. Conducted the other portion of the usability testing.

Seth Krause

Major: Computer Science - Robotics and Intelligent Systems

Work Experience: Platform Engineering Intern at Thomson Reuters, Software Engineering Intern at Renaissance learning

Role on the team: Seth was in charge of developing game mechanics and graphics as well as designing some of the core systems for the game. He also conducted a large portion of the usability testing.

*** see appendix**

Dylan Cormican

Major: Computer Science - Data Science

Work Experience: Software development intern at Poly, Software engineering intern at HP

Role on the team: Dylan was the lead developer for the front end platform, which ties the files, LLM, and game together.

Colin Brush

Major: Computer Science

Work Experience: Low-level embedded systems software development at Garmin

Role on the team: Colin handled the development and coding of the game aspect including the interpretation of the json file and the implementation of the games necessary mechanics.

*** see appendix**

XII. References

[1] “ACM Code of Ethics and Professional Conduct,” ACM, https://www.acm.org/code-of-ethics (accessed Oct. 19, 2023).
[2] “IEEE code of Ethics,” IEEE, https://www.ieee.org/about/corporate/governance/p7-8.html (accessed Oct. 19, 2023).

XIII. Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
Flask	*Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions*
Jest	*JavaScript test framework used for testing React applications.*
Langchain	*Python framework used to create applications using LLMs*
Llama_Index	*Data framework designed to connect custom data sources to LLMs, used to store and index data in vector embeddings.*
LLM	*Abbreviation for “large language model” - A large language model is a type of language model notable for its ability to achieve general-purpose language understanding and generation.*
Puppeteer	*Library used to control web browsers, used for end-to-end testing.*
React	*React is a JavaScript library for building user interfaces that enables the creation of interactive and dynamic web applications by efficiently managing and updating the UI components in response to data changes.*
Vector Embedding	*Numerical representation of data that captures semantic relationships and similarities.*
Prompt Engineering	*The creation of specific prompts to pass into an LLM in order to receive a specific type/format of output.*
GameMaker	*Free 2d game development software used for implementing game designs*

*** see appendix**