**Questology**

# Introduction

## Project Overview

1. The term project focused on developing a 2D quest game using Java and Gradle. My goal was to apply everything I've learned over the past few months in class, particularly with Java and Gradle, and combine it with the experience and skills I gained from working with AI tools during hackathons this semester. It was a way to bring together all the concepts and techniques I've picked up, essentially wrapping up the semester by integrating everything I've learned.

## Introduction of the Quest Game

2. The quest game I built is my way of gamifying learning. Players are placed on a map where they interact with an NPC to choose a topic they want to learn about. The NPC then explains the rules of the quest, which involve visiting four landmarks scattered across the map (you'll recognize them when you see them). Each landmark features a unique minigame designed to teach more about the chosen topic. Once all four minigames are completed, the game is finished. Players can also interact with NPCs throughout the quest to gain additional information, however not during a minigame.

## Development Environment

- **The version of Java Used:**
  OpenJDK 19.0.2
- **IDE Used:**
  Visual Studio Code
- **Were there any special libraries or special resources where you got them from and how to install them:**
  Everything I used is encapsulated inside of gradle ~~and any API keys are provided or already pushed to the repo.~~ Special resources are: LMNT(speech-to-text), Groq(LLM: Llama 3.3 70b), CloudFlare Workers AI(stable diffusion image generation model).
- **How to build or import your game in the IDE you used:**

  Clone the repo into Visual Studio Code and enter the following commands via the CLI.

Linux/MacOS:

```
export LMNT_API_KEY=redacted

./gradlew clean build run
```

Windows:

```
set LMNT_API_KEY=redacted

gradlew.bat clean build run
```

Note: I imported the repo into IntelliJ to build it into a JAR, however ran into multiple issues. Thus, I just listed the steps I took to build it through Gradle. For the LMNT SDK, a virtual environment is created, and pip installs are passed directly through the command line using Gradle. This means you don't have to do anything other than build with Gradle. I set it up to support both macOS/Linux and Windows, since they use different commands, but I've only tested it on macOS.

---

## How to run and Play your game

1. WASD keys.

Tips however not required:

2. The circles are other NPC's you can chat to. Start the game by navigating to them.

3.The first message you send in chat will be your chosen topic to learn.

4.To change the topic, press the "Back to Menu" button, then "Start Game" again.

5.If you encounter issues answering questions(poorly generated question/answer), check the console output for the answers, they may be hidden sometimes (use Cmd+F for "answer"). You may also exit the minigame and return for a new question.

6.Turning off the sound disables text-to-speech (won't work mid speech output/generation).

7.Important: You will need to close the chat after you **send** a message and want to move your character.

Known Bugs & Solutions: If your character is continuously moving in one direction (e.g., to the right), press the corresponding key again (e.g. right) to reset the movement. The PDF reader button used to work but has been removed. It no longer functions due to rate-limiting when uploading large PDFs due to switching to a larger model. The button remains, but it's curently turned off–I'd like to add it in the future if the rate-limits change.

**Assumptions Made when designing and implementing your game:**

Players will run the game through Gradle, as there isn't a JAR file available.

Players will expect to use WASD to move around.

Players will eventually interact with an NPC and realize that the circles are NPCs.

Players may and are encouraged to explore the map or landmarks first, as the landmark mini-games will prompt them to visit the NPC.

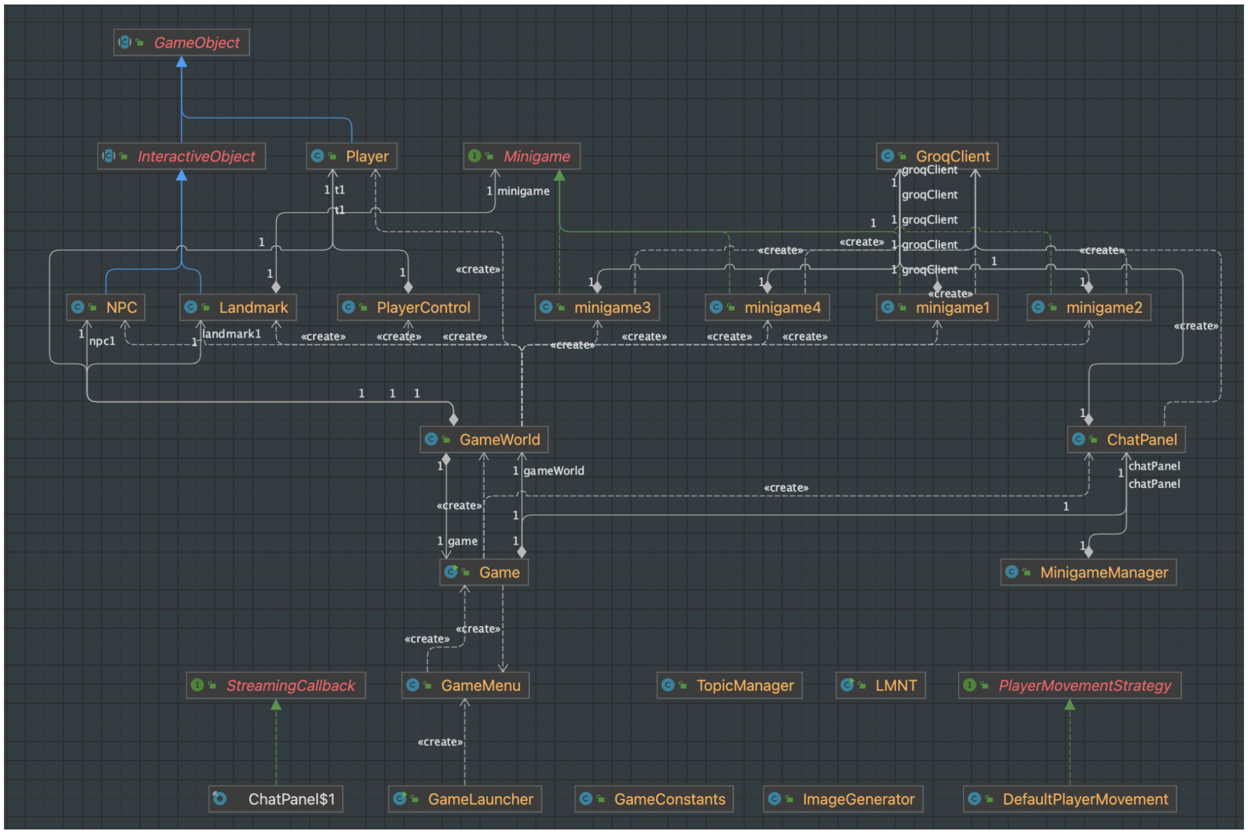
Players will figure out that the landmarks are generated images related to the topic they entered.

The chat system is designed to feel natural, but the first message players send will always be the topic, even if they say something like "hello" or "how are you?"

Any harmful words will be filtered by Llama, and using them as the topic might break the game.

---

# Tank Game Class Diagram



---

# Class Descriptions of classes implemented in the Tank Game

### ChatPanel

The ChatPanel class is a GUI component that gives the player a chat interface within the game. It lets the player interact with an AI guide to learn about any singular topic during their quest. The class manages user input, displays chat messages, and includes text-to-speech functionality to play the AI's responses. It also uses a LLama through the GroqClient to generate dynamic replies based on what the player types and the chosen topic.

### DefaultPlayerMovement

This class implements the PlayerMovementStrategy interface and handles the default movement behavior for the player. It checks which directional keys are pressed, adjusts the player's velocity (Vx and Vy) accordingly, and then updates the player's position based on those velocities.

**Game**

This class is the main entry point for the game. It sets up the game window, initializes the game world, and manages features like the chat panel and toolbar along with user interactions, such as toggling chat visibility (“Toggle Chat”), returning to the main menu, or turning speech on/off.

**GameConstants**

Defines constants for the game's screen dimensions. These dimensions are dynamically calculated based on the user's screen size, ensuring the game scales properly across different devices.

**GameLauncher**

Launcher class that starts the game by displaying the main menu. It ensures that all Swing components are initialized using SwingUtilities.invokeLater.

**GameMenu**

The main menu of the game. It features buttons for starting the game, accessing options (which was never used), and exiting. The menu uses a grid layout.

**GameObject**

An abstract class for all objects in the game world. It includes position, sprite, and hitbox, along with methods for updating and rendering objects.

**GameWorld**

This class is the core gameplay area where all interactions take place. It handles rendering, player movement, NPC interactions, and landmarks. It also manages viewport for the scrolling screen. The class includes methods for updating player positions, detecting interactions with NPCs and landmarks, and reloading game objects.

**GroqClient**

This class communicates with the Groq API to generate responses using Llama 3.3 70b. It constructs HTTP requests using OkHttp, sends them to the API, and processes the responses. The class has methods for both synchronous and streaming responses, however, using streaming allows the message to be sent to LMNT as quickly as possible for quicker speech generation.

**ImageGenerator**

The ImageGenerator class handles the creation of images based on specific topics. It reads configuration properties, generates a short prompt for image creation, and communicates with CloudFlare Workers AI REST API to generate and fetch images using a stable diffusion model. The class also has methods for converting plural words to singular forms due the model generating multiple items (i.e. saying you would want to learn about cat**s** would generate and images of multiple tiny cats that may be difficult to see) allowing for more accurate image generation.

**InteractiveObject**

An abstract class that extends GameObject, designed for objects that can interact with players within a certain range. It defines a method to check if a player is close enough to interact (e.g. NPC's and Landmarks).

**Landmark**

Landmark is a subclass of InteractiveObject representing special locations in the game world. These landmarks trigger minigames when a player interacts with them. The class includes methods for loading the generated images and updating landmark sprites, drawing them on the screen, and starting associated minigames.

**LMNT.java**

This class is responsible for text-to-speech using a Python script. It executes the script to generate audio files(mp3) from text and plays them using Java's sound libraries.

**Minigame Interface**

This interface makes sure all minigames in the game have a start method to initialize and run the minigames below.

**minigame2** (Multiple Choice Quiz)

This class implements a multiple-choice quiz as a minigame. It uses GroqClient to generate quiz content based on selected topics. The UI presents questions and options, tracks scores, and tells the user if it's a correct or incorrect answer.

**minigame3** (Matching Game)

A matching game where players match terms with their definitions. The class generates pairs using GroqClient, sets up a grid of cards you can flip for interaction, and tracks matches found by the player.

**minigame4** (Fill in the Blanks)

This minigame presents players with sentences containing blanks that need to be filled in with the correct words. It uses GroqClient to generate content and hints related to the current topic. The UI allows players to submit answers and request hints.

**MinigameManager**

This class manages the progress and completion of minigames in the game. It tracks completed minigames, checks if all are finished. Once finished, a chat message will notify you congratulations on finishing along with more details. It was also supposed to update quest progression in chat, however the feature had issues and was removed.

**NPC**

Represents non-playable characters in the game. NPCs have a position, sprite, and dialog, and they can interact with the player when approached. They can also be “reached” through the toggle chat button.

**PDFReader**

A class that reads text from PDF files using the PDFBox library and sends it to Groq. It was removed after upgrading to a newer model with stricter per-minute token rate limits, making it unsuitable for handling large volumes of text.

**Player**

The Player class represents the main character controlled by the user. It handles movement, boundary checks within the game world, and updates its position and hitbox.

**PlayerControl**

Handles keyboard input for controlling the player's movement.

**PlayerMovementStrategy**

Interface for defining different movement for player.

**StreamingCallback**

Interface for handling real-time streaming responses. For a more dynamic chat and faster response time for LMNT(speech-to-text).

**TopicManager**

Manages the current topic selected in the game. It allows setting and retrieving topics. Allowing it to be called by minigames and image generation classes.

**GameWorld**

Central gameplay area where all interactions occur. Manages rendering the game map, player movement, NPCs, landmarks, and viewport scrolling.

**lmnt_tts.py** (only non-java class)

This Python class handles text-to-speech using the LMNT API. It pulls API keys from environment variables for secure access and converts text into speech asynchronously, saving the output as an MP3 file. Error handling is built in for missing packages or API keys, mainly because pip installs and API setup are run directly through Gradle using command lines. It works on macOS and Linux and should work on Windows too—though I couldn't test that, so I added error handling just in case. Since there's no Java SDK for LMNT, I wrote it in Python. By default, it uses Souza's cloned voice for speech-to-text, but if you'd rather use the default voice, that's commented out on line 28.

---

# Self-reflection on the Development process during the term project

This idea originated after a hackathon project where I worked on replicating Duolingo's process but for buying houses. After that, I started seeing videos about Duolingo in my recommended feed (even though I'd never used it) and learned they were testing AI in their app. Around the same time, I had the idea of creating something similar to Duolingo, but where you could learn anything.

When the term project was announced, I decided to incorporate this idea into it. I began the game at a llama hackathon, where I laid the foundation and integrated Llama as a chatbot, using Groq for processing—tools I was already familiar with from previous projects, so it felt pretty straightforward. However, the hackathon had specific tracks, and I couldn't fully pursue my idea, so I pivoted to creating a game that would gamify taxes. The idea was to parse tax documents and "fill them out" through a chatbot conversation. Unfortunately, the project didn't work well and was mostly mocked up.

Afterward, I continued working on the game, shifting away from taxes and refocusing on the concept of learning anything. I also added image generation using Cloudflare's AI Stable

Diffusion model, something I had implemented in a previous hackathon project. As weeks passed, I further refined the game, and just before presentations, I took it to another hackathon. Initially, I worked on integrating Simli, an AI voice avatar, to replace the chat. However, implementing it into Gradle proved more difficult than expected, so I scrapped the idea midway through the hackathon. With limited time left, I switched to using LMNT another sponsor at the hackathon, which I had worked with before at another hackathon. LMNT provides speech-to-text and voice cloning, and I used their Python SDK to integrate it.

To make it work with Gradle, I created a virtual environment and installed the necessary requirements via pip directly through Gradle's command-line interface. From there, I sent the text to the Python script, which would save the speech as an MP3 file, then play it. The following day, I had the fun idea of cloning Souza's voice, which added a nice touch. Before the presentation, I made a few minor tweaks to make everything presentable.

A few days later, I attended a HeyGen hackathon, which focused on creating projects with their AI avatars. Once again, I tried and failed to integrate the avatar into my game. I worked on my game for the remained of the hackathon until in the final hour, I had a change of heart and wanted to present so I quickly implemented one of their quick-start Node.js demos, which worked in a simple UI where you could enter text, and the avatar would respond. I set up a WebSocket to stream Groq's responses to where the "enter text" was. The Node.js server, and it was somewhat functional, but when the avatar spoke, I realized since it was a demo it only responded with messages as the following, "(topic here) sounds cool, but let's get back to talking about HeyGen." I just copied the SDK to get it to run and didn't have time to dive deeper into it or documentation as I had literal minutes before the projects were due, so I just gave up afterwards. Given how behind I was on the game, I didn't revisit HeyGen after the hackathon. However, I do have a short video snippet of what the avatar would have looked like in the game here, which I am slightly bummed out I didn't get to integrate as it looks very realistic. The avatar is saying "Hello from Taxplorer," which was the initial name of my game when it focused filling out taxes and I never got around to coming up with a new name so I kept it as is.

I also spent a lot of time refining my prompts. Originally, I thought I could rely on Llama to handle most of the work for the landmarks and navigation, but that plan quickly fell apart. Llama often hallucinated, made things up, or just didn't follow instructions. So, I ended up spending a good amount of time adjusting the prompts, being more specific with the requests to get better results, and sometimes hardcoding chunks of the prompt.

# Project Conclusion

Overall, I'm glad I worked on the game, but I'm not entirely satisfied with how it turned out. It wasn't exactly what I envisioned, especially visually, the game doesn't look good or appealing at all. I spent a lot of time adding AI features, and a few didn't even make it into the final version, but I don't regret any of it. I do think, though, that I could've benefited from spending more time focusing on the core foundation of the game rather than getting distracted by additional features.

I also pivoted ideas a lot during the process, which was fun in some ways, as it let me explore my creativity and work on something that excited me at the time. But I also know I wasted more time taking breaks where I'd stare at my code and debate whether to change things up. Also most of my projects so far have been hackathons or short-term fun projects, so I didn't have to think too deeply about long-term consequences. I could just throw something together, keep iterating, and it would be fine. But with this project, where I was working on it for over a month, I quickly noticed how unorganized the codebase was, especially with some of the file names. That's something I'm going to work on improving with smaller projects so that when I do get to bigger ones, I won't make the same mistakes I've gotten away with before.