Knock Knock

A survival-based quiz game built in Godot with C++

Team Members: Alberto Alvarez Alcaraz, Jose Sanchez Menchen, Mikel Sabina Bazaco, Adi Karamustafic

Overview

- Problem
- Background
- Design Plan (UML + architecture)
- Implementation Details
- Example Usage
- User Feedback
- Reflection and Future Improvements

The Problem We Address

- Students often struggle to study effectively for Computer Science
- It is difficult to generate meaningful practice questions independently
- Traditional studying feels repetitive and unengaging
- There is no existing tool that turns CMSC240 revision into an interactive experience

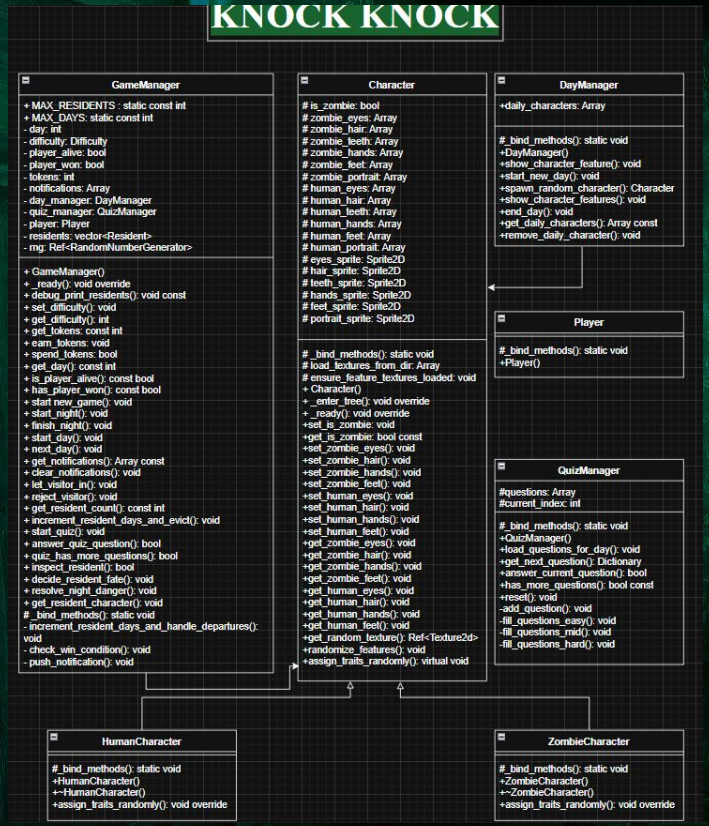
Background and Motivation

- Inspired by No, I’m Not a Human survival game.
- Survival decision-making creates tension and engagement.
- Educational games help increase retention of technical content.
- Target users include students with beginner to advanced coding experience.

Project Purpose

- Transform CMSC240 practice into an interactive survival scenario.
- Use daily rounds, character inspection, and quiz mechanics.
- Encourage learning through consequences and reward loops.

Class Architecture (UML)



- Central controller: GameManager
- Character system with shared base class Character
- Specializations: HumanCharacter, ZombieCharacter
- DayManager spawns characters and displays features
- QuizManager handles question loading and evaluation

Design Rationale

- Object-oriented approach allows modular extension.
- Separate managers reflect different gameplay systems.
- Character inheritance allows humans and zombies to share behavior where appropriate.
- Godot + GDExtension C++ used for performance and control.

Key Implementation: GameManager

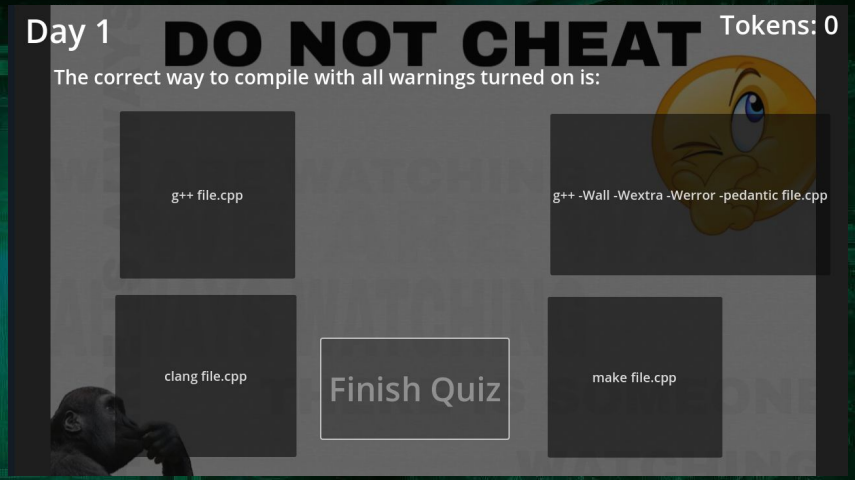
- Controls full game loop: day → quiz → inspection → night.
- Tracks player tokens, days survived, win/loss condition.
- Calls other managers (DayManager, QuizManager).
- Handles core logic: start_day(), start_night(), inspect_resident(), resolve_night_danger().

Key Implementation: Character System

- Characters have randomized traits: hair, eyes, hands, feet and teeth.
- Both human and zombie traits stored in arrays.
- Features loaded from directories using C++ texture loading.
- Randomization supports inspection-based gameplay.

Key Implementation: Quiz Manager

- Loads CMSC240 questions per difficulty level.
- Tracks current question index.
- Validates player answers.
- Influences gameplay by granting tokens.

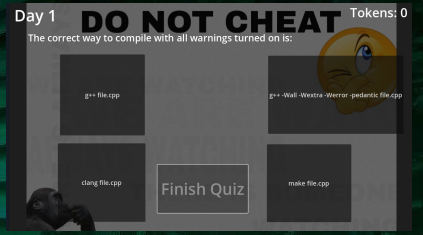


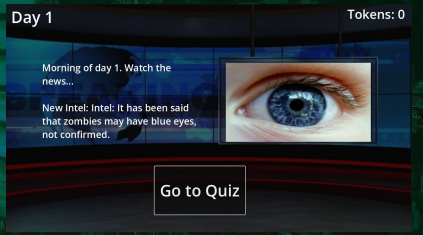
How Everything Connects

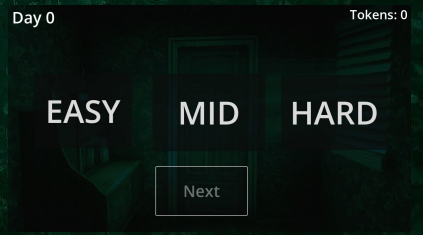
- GameManager orchestrates all actions.
- DayManager creates characters → QuizManager activates questions.
- Correct answers give tokens; wrong answers increase risk.
- Every subsystem supports the main learning objective.

Example Usage (Gameplay Flow)

1. Introductory.
2. Choose a difficulty level.
3. Each day, a resident knocks on your door. Let them in or reject them.
4. Sleep.
5. Wake up and watch the news.
6. Do the quiz. Answer CMSC240 questions correctly to earn tokens. Inspect features to decide: human or zombie? Or just go to sleep, but remember, your choices determine who survives the night.
7. Night outcomes resolved.







What We Learned from Surveys

All core requirements scored extremely high, with most users giving 5 (Beyond expectations) for:

- Difficulty system
- Reward/inspection system
- Use of CMSC240 questions
- Day/round tracking
- Photos of Character Features

Users consistently rated the game as easy or intuitive to use

- Most selected 4 or 5 on the ease-of-use scale
- One user selected 3 (neutral), but no one found it confusing or difficult

The core gameplay loop is functioning as intended

- Difficulty felt appropriate
- The daily structure and survival mechanic were clearly understood
- The CMSC240 questions were viewed as appropriate and useful
- All users rated this requirement at the maximum score (5)

No requirement scored below 4 from any user, meaning players felt the game was complete, coherent, and aligned with the project goals

Reflection: What We Learned from the Project

- Working with C++ in Godot required careful binding and debugging.
- Team collaboration was essential to align mechanics and design.
- Balancing gameplay and educational value was challenging.
- Setting internal deadlines kept us on track and prevented the project from becoming overwhelming.
- We gained practical experience integrating a native C++ layer inside a game engine environment.

Future Improvements

- Expand question bank with more CMSC240 topics.
- Improve character visuals and inspection clarity.
- Implement audio + atmospheric effects.

Conclusion

- Knock Knock turns CMS2C40 studying into an interactive experience.
- Architecture supports expansion and modularity.
- Surveys will inform next iteration.
- Thank you!

YOU WIN