Codémon: A Buggy World

A Debugging-Driven RPG for Learning Programming

Team Members

- Connor Allen — Project Planner
- Isaac Hutchison — UI / UX Design
- Lon Danna — GitHub & Infrastructure Coordinator

GitHub Repository:
https://github.com/Connor-Allen10/codemon

Communication: Discord
Version Control: GitHub (feature branches + pull requests)

Abstract (Executive Summary / TL;DR)

Codémon is a Pokémon-style role-playing game designed to teach one of the most critical yet under-practiced programming skills: debugging. Instead of writing code from scratch, players progress through the game by identifying and fixing bugs embedded directly into the world, battles, and creatures. By reframing debugging as the *core mechanic* rather than a punishment, Codémon offers an engaging, low-pressure way for students to build confidence, pattern recognition, and problem-solving skills essential to real-world software development.

Goal

The goal of Codémon is to help users practice and improve debugging skills in a fun, gamified environment. The system emphasizes understanding program behavior, spotting errors, and fixing bugs efficiently — skills that are central to professional programming but often underemphasized in traditional learning tools.

Current Practice

Most educational coding games and platforms focus on:

- Writing new code from scratch
- Solving algorithmic puzzles
- Debugging only as a failure state

In these systems, debugging is often treated as a secondary or punitive task rather than a primary learning objective. As a result, students may graduate with limited confidence in diagnosing real-world bugs.

Novelty

Codémon's novelty lies in making **debugging the main gameplay mechanic**:

- The game world itself is "buggy"
- Enemies behave incorrectly due to code errors
- Progression depends on diagnosing and fixing bugs

Rather than reinventing coding education, Codémon shifts focus to a *missing skill* by embedding debugging challenges directly into gameplay systems such as battles, exploration, and progression.

Effects (Who Cares?)

If successful, Codémon will:

- Reduce fear and frustration around debugging
- Help students develop faster bug-recognition skills
- Serve as a supplementary learning tool for CS students
- Provide a more realistic representation of professional programming work

---

1. Use Cases (Functional Requirements)

Use Case 1: Debugging Battles (Connor)

Actors: Player
Triggers: Player encounters a Codémon with broken behavior, battle is started
Preconditions: Player has a starting Codémon to battle with
Postconditions:
1) Player successfully fixes the bug and wins the battle.

2) The Codémon responds appropriately to the bug fix.
List of steps:
1) Player encounters a Codémon
2) Player goes to battle
3) Player tries to attack with their Codémon and observes its buggy behaviour
4) Player opens their Codémon's code and attempts to fix the bug.
5) Player returns to battle and the Codémon behaves correctly
6) Battle concludes

Use Case 2: World Progression Through Debugging (Isaac)

- Environmental obstacles (bridges, doors, paths) contain bugs
- Player fixes logic or syntax errors to unlock new areas
- Visual feedback confirms correct debugging

Actors: Player
Triggers: Player encounters a blocked path, door, bridge, or environmental obstacle that does not function correctly
Preconditions:
1) Player has access to the area containing the obstacle
2) The obstacle is controlled by buggy logic or syntax
Postconditions:
1) Player successfully fixes the bug controlling the obstacle
2) The obstacle functions correctly and allows access to a new area
3) Visual feedback confirms the successful fix
List of steps:
1) Player explores the world and encounters a blocked environmental obstacle
2) Player interacts with the obstacle and observes incorrect or broken behavior
3) Player opens the debugging interface associated with the obstacle
4) Player inspects the provided code snippet containing logic or syntax errors
5) Player corrects the bug and submits the fix
6) System validates the fix and updates the world state
7) The obstacle visually changes to indicate it is now functional
8) Player proceeds into the newly unlocked area

Use Case 3: Codémon Growth & Evolution (Lon)

Actors: Players
Triggers: A Codémon reaches the level required to evolve, but evolution isn't triggered.
Preconditions: Codémon is at the level required for evolving.
Postconditions: Evolution logic is corrected, Codémon transforms and updates its stats.
List of steps:

1. The player opens the Codémon menu and selects specific Codémon.
2. The player clicks the Inspect Logic button to see why evolution is failing.
3. The system displays an if statement: if current_level >= 100.

4. The player identifies that the constant 100 is a logic error for low-level evolution.
5. The player changes value to 10 and saves changes.
6. The Evolve button becomes active, the player clicks it to trigger the evolution.

Extensions: Correcting the code with cleaner logic (refactoring) grants a small evolve bonus.
Exceptions: If a player enters a non-numeric value, the system displays “Input Mismatch” and prevents the save.

---

2. Non-Functional Requirements

1. **Usability:**
   - Debugging interfaces must be intuitive for beginners
   - Clear visual feedback for correct/incorrect fixes
2. **Performance:**
   - Game must run smoothly on standard student laptops
   - Low latency for input and rendering
3. **Maintainability:**
   - Modular game and debugging systems
   - Clear separation between gameplay and bug logic

---

3. External Requirements

- **Error handling**: The product will include robust input-validation. The game should not crash if a player types something unexpected. For example, if the game asks for a number and the player enters a word or gibberish, the game should simply show an error message and let them try again rather than crashing.
- **Installability**: Anyone should be able to play the game without having to be a programmer. If it’s a computer program, we will provide a ready to use file. You shouldn’t have to download a bunch of tools to get the game to open.
- **Buildability**: We will provide the source code and clear instructions. The repository will include a set up guide to help install the C++ compiler and SFML headers.
- **Scope**: The project is scaled for 3 members, focusing on one polished zone and robust battle system rather than an over extended open world.

- Must use GitHub for version control and collaboration
- Must be implemented using C++
- Must comply with CS 362 project scope and timeline
- Must be publicly accessible for evaluation

---

4. Team process description

Software Toolset / Technical Approach

- **Language:** C++
- **Graphics & Input:** SFML (Simple and Fast Multimedia Library)
- **GitHub**: Version control and feature branches and pull requests
- **Architecture:**
    - Core game loop
    - Debugging challenge engine
    - Modular enemy and world systems

Bug challenges will be represented as controlled code snippets with predefined failure modes and validation logic.

Risks & Mitigation

Primary Risk: Scope creep due to creative ambition

Mitigation Strategy:

- Define a simple core gameplay loop early
- Implement minimum viable features first
- Treat advanced mechanics as stretch goals

Secondary Risk: Technical issues, feature difficult to implement

Mitigation Strategy:

- Plan each core feature as soon as possible so that this risk may only apply to optional features/stretch goals
- Modify feature to make technical implementation easier

Tertiary Risk: Code is overwritten due to faulty version control

Mitigation Strategy:

- Communication – Ensure code is up to date before pushing to repo
- Keep a backup of the latest working version
- Version history is preserved in GitHub

Team Roles & Responsibilities

- **Connor Allen:** Project planning, gameplay logic, requirements tracking
    - These roles ensure the scope of the project stays in check, managing this is very important.
- **Isaac Hutchison:** UI/UX design, player interaction, visual feedback
    - The UI/UX designer role requires an experienced UI designer.
- **Lon Danna:** GitHub management, CI setup, documentation, integration
    - Justification: GitHub version control is an immensely important part of developing software in a team, so having one person be in charge of making sure everything is clean and organized is important.

Schedule/Timeline

Week	Focus
1	Requirements, core design, repo setup
2	Core game loop prototype
3	Debugging battle system
4	World progression mechanics
5	Codémon capture & evolution
6	UI polish & feedback
7	Testing & iteration
8	Bug fixing & refinement
9	Final polish & presentation

External feedback will be most useful around week 6 or 7 of our timeline, where the program has the functional requirements implemented, and we can iterate and improve based on test users’ feedback.