

CST8110: Introduction to Computing

Lab 10: Using Structures in a Game Design

Overview

This will be an individual lab. You will work with existing game code supplied by me on my website. The code functions correctly as far as it goes. You will modify the code to do the following:

- Add appropriate *struct* specifications to cluster related data.
- Nest at least one structure inside another (for example, the **Coordinate** *struct* will likely be nested inside the **Actor** *struct*.)
- **Essential Game Enhancements:**
 - Add at least one data member to the players (Godzilla and human player). You can decide on the purpose for this extra data member. Of course, you will need to add code to manage this extra data member.
 - Add a time limit to game play and display the time remaining.
- **Optional Game Enhancements:** None of the following are required, but they do offer an opportunity to explore advanced features of the C++ language.
 - When the human player encounters Godzilla; currently, the human player dies and Godzilla wins. In the enhanced version, an encounter results in a battle. The human player and Godzilla will battle each other, inflicting randomly determined damage. The battle will be implemented using a small loop inside the main game loop. On each iteration of the loop, each **Actor** object's health points decline (both the human player and Godzilla). The loop terminates when one **Actor**'s health points is less-than-or-equal-to zero. You need to define the rules that affect their health points. It could involve an arithmetic calculation that blends each Actor's strength with some random value. The decision is yours.
 - Currently, Godzilla moves approximately 60% of the time. In the enhanced version, have Godzilla move more frequently as game play progresses.
 - In the current implementation, program execution always stops and waits at the `_getch()` function (until the player types a key). As a result, the display of time remaining will freeze (even though the timer really continues to count). Each time the player types a character, `_getch()` returns with this character value; and program execution sweeps through one more iteration of the main loop, stopping again at `_getch()`. I'll post some sample code on my website that shows a technique to change this.
 - Adding colour to your game. I'll post some sample code that shows a technique to implement colour.
 - You can implement other enhancements, but check with me in advance to ensure that you are working within reasonable limits.
 - Organize game play using functions.

You will need the *Problem Statement (Parts 1 and 2)*, *algorithm* and *test plan*, and a complete, working C++ implementation.

Problem Statement: Part 1: Original Code

You are using my code as a foundation, so *Part 1* of your problem statement must describe how my original program behaves. This is a useful method of understanding code that is written by someone else¹.

¹ This mirrors the real world of programming. In a production environment, you will commonly work with someone else's code. It is rare that you invent
Lab 10: Using Structures in a Game Design

Problem Statement: Part 2: Code Enhancements

Part 2 of your *problem statement* must identify the enhancements that you intend to implement. As a minimum, you are implementing the two basic enhancements, but you can add as many as you want.

Algorithm: Part 1: Original Code

You need an algorithm for the entire program, but I want to simplify your work. For my code, you can write a more descriptive high-level algorithm. For example, one section of C++ code contains two nested **for** loops to display the map. You need not replicate the details of these nested loops. You can put a single statement in your algorithm that announces that the map will be displayed.

Handling player movement can also be expressed as a descriptive high-level algorithm.

Writing this descriptive high-level algorithm helps you to understand my original code.

You will still need to show the loop boundaries of the main game loop. This is important since you need to identify where you expect to insert your game enhancements.

Algorithm: Part 2: Code Enhancements

All your enhancements must be expressed in the detailed algorithm style (the style used in earlier lab work).

For example, to implement the game's time limit you need to identify where current time is captured; where the calculation of time remaining takes place; where the display of time remaining takes place; where the test to see if time has run out takes place. These detailed steps will be placed in the descriptive high-level

Strategies to Complete the Lab

- Capture my code, compile and run.
- Write the *Problem Statement: Part 1: Original Code*.
- Write the *Algorithm: Part 1: Original Code*.

At this point, you should understand the original code very clearly. You are now ready to implement the enhancements.

- Implement and use the **struct** specifications to cluster related data. Do not implement any enhancements yet.
- Compile and run the code.
- Write the *Problem Statement: Part 2: Code Enhancements*. Essentially, what are you proposing to do?
- Write the *Algorithm: Part 2: Code Enhancements*. Essentially, how are you going to do it?
- Write the *Test Plan*. The *Test Plan* will guide you during your coding process.
- Write and test the C++ code to implement the algorithm changes.

an entire program from scratch. So here, you'll be working with my code.
CST8110: Introduction to Computing

Submission Standards

- The due date is posted on my website.
- Cover page:
 - Course Name / Number (including lab section)
 - Lab Number
 - You name
 - Date of Submission
- Game solution containing *Problem Statement, Algorithm and Test Plan*.
- Use the variable-naming convention shown in Lab 1.
- Use a consistent approach to indentation to communicate program structure.
- Include many useful comments:
 - Multi-line sections of code need a short description of that section's purpose. These comments will be aligned at the same left margin as the associated code (that is, indent comment the same amount as the code)
 - Most individual lines need a clarifying comment to the right.

Submission Process

- Staple all paper components together with the evaluation sheet (shown on the right side of this document).

Evaluation Criteria

CST8110: Introduction to Computing Lab 10: Using Structures in Game Design

Your lab assignment has been evaluated using the following criteria: This lab is to be completed on your own. Look to Lab 1b for coding standards (sample code, variable naming conventions, etc.)

- Statement of Assumptions, Test Plan, Algorithm (*Included from previous lab work with any revisions*)
- Program and function headers (remember, *main()* is a function), including history.
- Comment each significant line of code.
- Comment each related section of your program.
- Variable naming as defined in handout.
- Indentation follows standard in handout.
- (4 marks) Program performs runs correctly from source code (Including meaningful user prompts).

Subtotal /10

- Penalties (up to 3 marks deducted): Penalty for redundant code, unused code, unnecessary extra steps. Penalty for poor organization and binding of program and documentation.
- Bonus marks (up to 3 marks added): Clever, clear, useful enhancements..

Total /10