ECE 4122/6122 Lab 1: Retro Centipede Arcade Game

(100 pts)

Category: Class creation and 2D Graphics with SFML
Due: Wednesday September 25th, 2024 by 11:59 PM

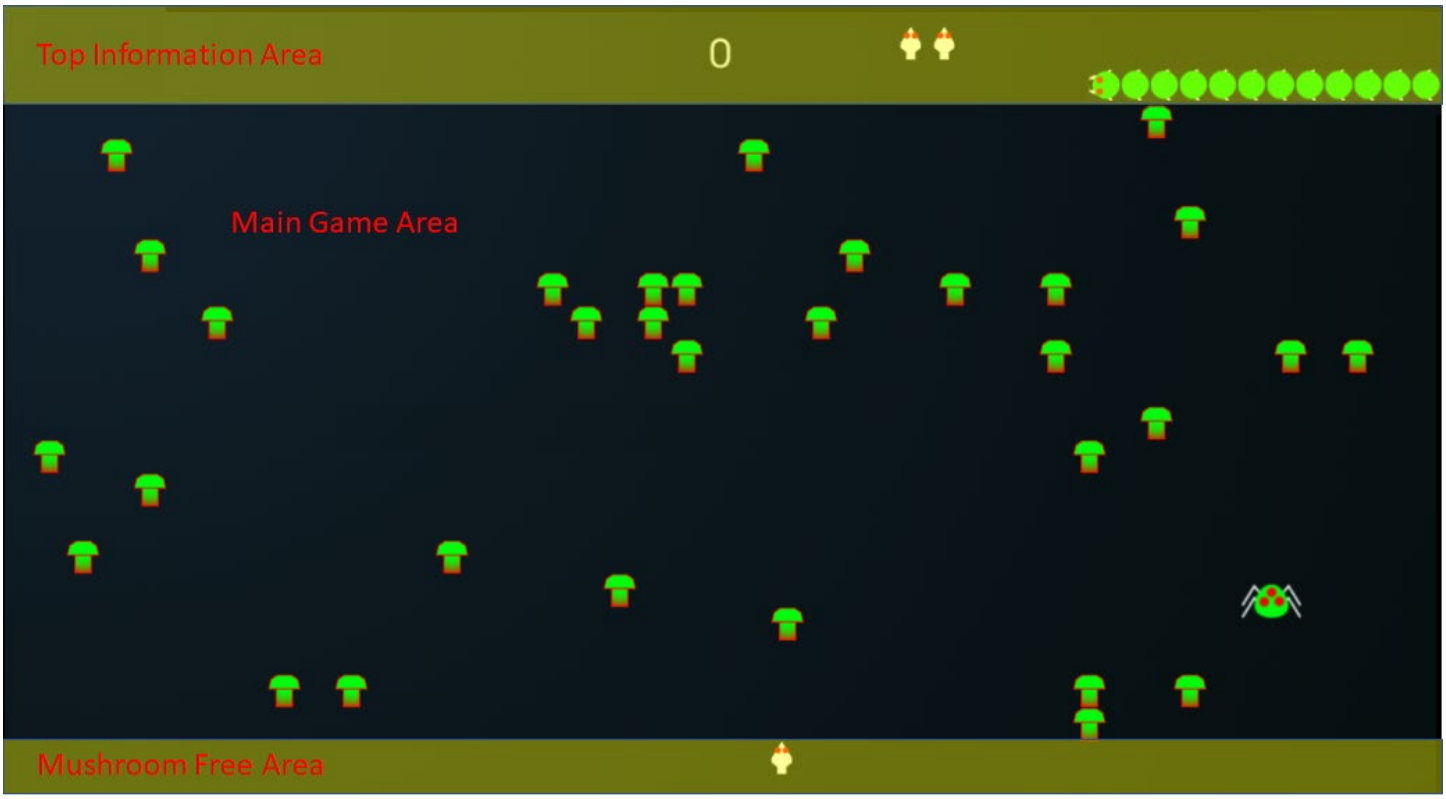
Objective:

To understand and apply the principles of 2D graphics and **class creation** in C++.

Description:

Create a simple version of the classic arcade game called Centipede. An online version of the game can be found at (https://www.mysteinbach.ca/game-zone/2153/atari-centipede/). The goal of the assignment is to try to reproduce the game play present on the listed web site. You only need to support the first level and the game ends when either all the centipede body elements are destroyed or the player runs out of lives. The game screen is made up of three section (1) the top information area, (2) the main game area and (3) the bottom mushroom free area, as seen below. The game score is shown in the middle of the top information area and the number of remaining live is show by the number of starship icons. The centipede starts in the lower right part of the top information band. The centipede is made up of 11 body sections and one head section. Reproduce the action of the centipede as shown in the game on the website. You only need to support the spider in your version

of the game.



Game specifics:

1. Create an ECE_Centipede class derived from the SFML::Sprite.
    a. The class is responsible for calculating the location of all the segments of the centipede and which segments have broken apart into new centipedes with its own head.
    b. The class is also responsible for detecting collisions with other objects and taking the appropriate action.
2. Use the random number generator (std::uniform_int_distribution) to randomly locate 30 mushrooms in the main game play area. Use a std::list to maintain the state and location of the mushrooms. Feel free to use either a class or struct to hold the information.
3. Create an ECE_LaserBlast class derived from the SFML::Sprite that calculates the current location of the laser blast and detecting collisions with objects and taking the appropriate action. Make sure to allow for the movement of multiple laser blasts by using a std::list.
4. Each time a mushroom is hit by the laser blast change the image to the ones provided. As the mushroom is hit the lower part disappears until it is hit twice and is removed from the game.
5. Use the left/right/up/down arrow keys to move the spaceship and use the space key to fire a laser blast.
6. Keep track of the score and the number or remaining lives and display it at the top of the screen.
7. When the game is over just go back to the Start screen.

Turn-In Instructions

Create your assignment in a folder called **Lab1** at the same level as the **Chapterxx** folders in the *"Beginning-Cpp-Game-Programming-Second-Edition"*. Inside your Lab1 folder create a CMakelists.txt file that will be used to build your assignment for testing. Inside your Lab1 folder create a **code** folder to hold any *.cpp and *.h files you create to complete your assignment. Also, inside your Lab1 folder create a **graphics** folder to hold any graphics needed by your game.

When you are finished zip up your Lab1 folder in to a zip file called **Lab1.zip** and upload this zip file on the assignment section of Canvas.

Grading Rubric:

1. (10 pts) Main start screen is shown at startup and pressing the enter key starts the game.
2. (20 pts) The 30 mushrooms are randomly placed in the main game area.
3. (15 pts) The starship starts at the center of the bottom of the game and the starship is moved around by pressing the left/right/up/down keys.
4. (10 pts) Pressing the space key fires laser blasts. Multiple laser blasts need to be supported and collisions need to be correctly handled.
5. (10 pts) The spider moves randomly around the main game area. When a spider collides with a mushroom, the mushroom is destroyed. When the spider collides with the starship, the starship moves back to the starting location and a life is used up.
6. (20 pts) The centipede moves correctly through the mushrooms and a segment is destroyed when hit by a laser blast and is divided, if hit in the middle creating a new smaller centipede.
7. (15 pts) The game play is fun with the individual components moving at reasonable speeds and collision correctly detected.

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Does Not Compile	40%	Code does not compile on PACE-ICE!
Does Not Match Output	Up to 90%	The code compiles but does not produce correct outputs.
Clear Self-Documenting Coding Styles	Up to 25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score – 0.5 * H	H = number of hours (ceiling function) passed deadline

Appendix A: Coding Standards

***Indentation*:**

When using *if/for/while* statements, make sure you indent **4** spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird).

This applies for functions and member functions as well!

The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function represents. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:

Every file should have the following header at the top

```
/*
Author: your name
Class: ECE4122 or ECE6122 (section)
Last Date Modified: date

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.