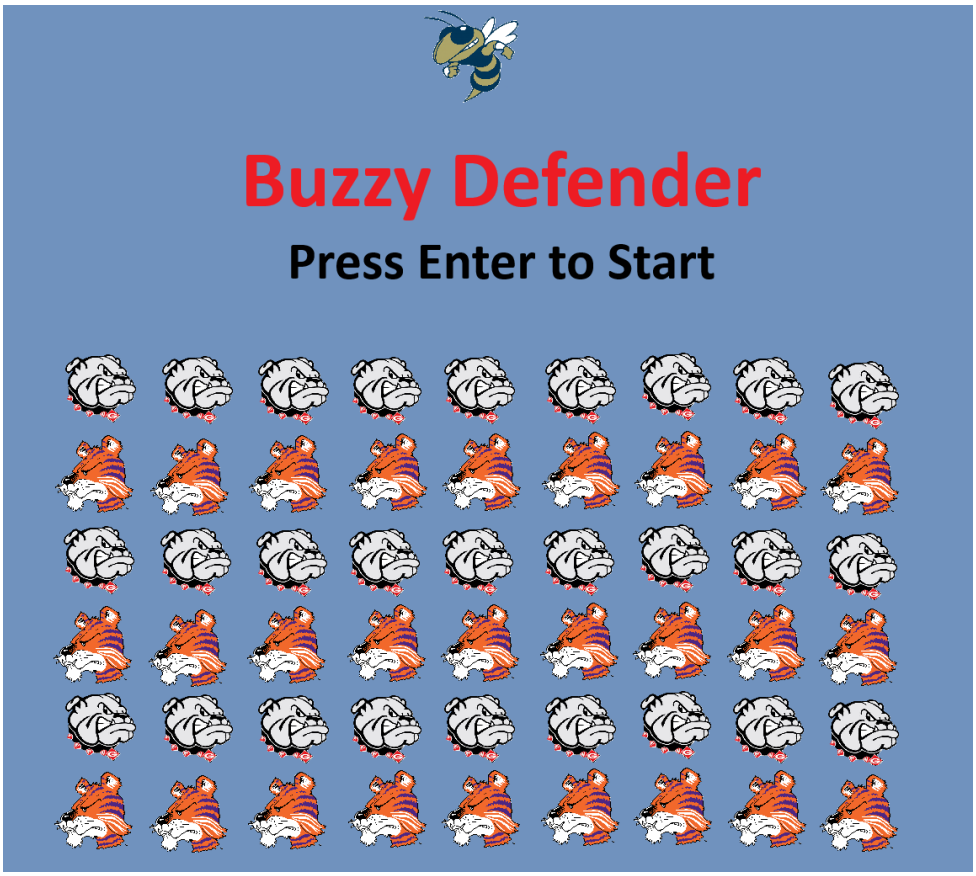
ECE 4122/6122 Lab 1: Buzzy Defender

(100 pts)

Category: Class creation and 2D Graphics with SFML
Due: Wednesday September 22nd, 2025, 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 Buzzy Defender (aka Space Invaders). A similar online version of the game can be found at (https://www.mysteinbach.ca/game-zone/1755/alien-invaders/). 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 enemy elements are destroyed or the player dies. Buzzy starts in the upper middle part of the screen. The enemy elements march left and right and move upward each time a side is encountered.

Game specifics:

1. Create an ECE_Buzzy class derived from the SFML::Sprite.
    a. The class is responsible for maintaining the texture, location, and displaying of the Buzzy player..
    b. The class is also responsible for detecting collisions with other objects and taking appropriate action.
2. 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. This class is used both by Buzzy and the enemy.
3. Create an ECE_Enemy class derived from the SFML::Sprite.
    a. The class is responsible for maintaining the texture, location, and displaying of the enemy element.
    b. The class is also responsible for detecting collisions with other objects and taking appropriate action.
4. Each time an enemy is hit by the laser blast it disappears.
5. Use the left/right/ arrow keys to move Buzzy and use the space key to fire a laser blast.
6. When the game is over just go back to the Start screen.

Sample images are included with the assignment. You are free to use different images to enhance the game.

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 into 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 enemy elements are displayed in evenly place rows.
3. (15 pts) Buzzy starts at the top center of the game and the is moved around by pressing the left/right 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) A random number generator picks on of the enemy elements to fire a laser blast at Buzzy.
6. (20 pts) The enemy elements moves correctly and advance upwards when side wall is encountered.
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.