Intro. to Game Programming Languages (GPL)

Objective

In this lab, the students will explore the basics of **Game Programming Languages** (**GPL**) by writing simple programs and executing them with the GPL interpreter. Understanding the input before developing a compiler (or any program) is important.

- How to declare and initialize game objects.
- How to declare and initialize GPL arrays.
- How to declare and initialize an animation block.
- How to declare and initialize a text box.
- How to declare and initialize event handler.

By the end of this lab, you should have grasped a basic understanding of GPL to start working on the project's first phase, i.e., develop your own arcade game in GPL.

Prerequisites

- Make sure **g++** is installed. `$g++ --version$`
- Download **lab1.tar.gz** file from Moodle.
- Decompress the file. `$tar -xvf lab1.tar.gz`

Detail

In the decompressed lab1 directory, you will find three sub-directories: **MacOS**, **Linux**, and **tests**. You will use the appropriate binary executable for your operating system. For example, if you are on MacOS, you will use *gpl* binary file in the **MacOS** directory.

In the **tests** directory, you will find multiple different GPL files. Your first task is to run all test files and see the outputs.

```
$./gpl <test_file>
```

Then, open each test file to learn the input code.

Game Objects

GPL has five different game object types: `rectangle`, `circle`, `triangle`, `textbox`, and `pixmap`. You can declare a game object either without parameters (e.g., `rectangle my_rectangle();`) or with parameters (e.g., `rectangle my_rectangle(x = 42, y = 2, h = 100, w = 200, red = 0, green = 1, blue = 0);`). Parameters specify the properties of the object. For the `rectangle` object, the default values are 0 for the x and y coordinates, and 10 pixels for both height and width (I know this is a square, but GPL uses the name rectangle to simplify the implementation).

The *rectangle.gpl* file contains a single line of code that declares a rectangle object with default property values. When you run this code (`$ ./gpl tests/rectangle.gpl`), you should see something similar to Figure 1. A new window will open, displaying a small red square in the bottom-left corner.

The *rectangle_w_properties.gpl* file contains a single line of code that declares a rectangle object with manually set property values. When you run this code, you should see something similar to Figure . A new window will open, displaying a green rectangle.

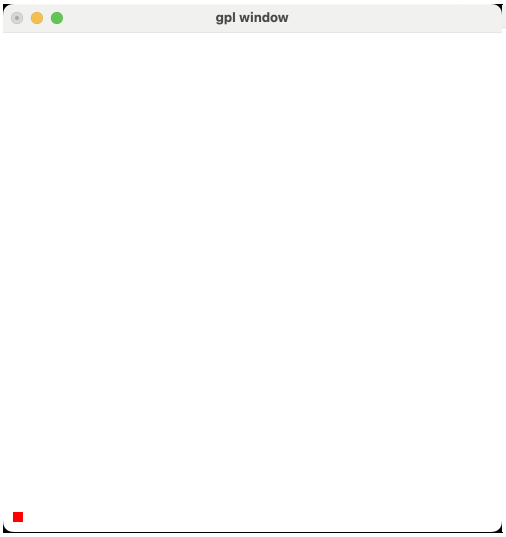

Figure 1: *rectangle.gpl*

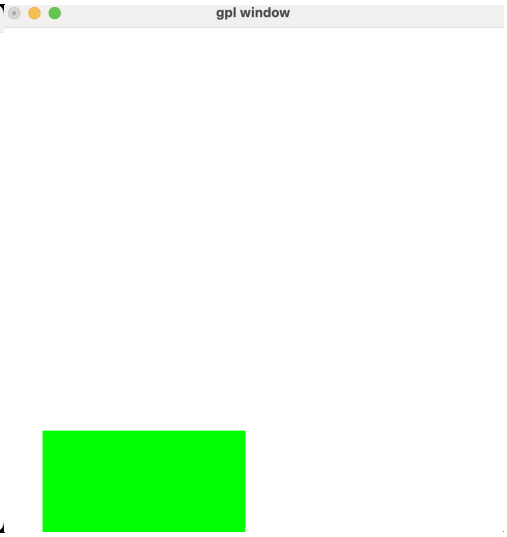

Figure 2: *rectangle_w_properties.gpl*

Using the provided GPL manual, try out different types of game objects.

Arrays

Like many programming languages, we can create arrays with all supported types in GPL (e.g., `int`, `double`, `rectangle`, etc.). For example, in the *tests/multiple_rectangles.gpl* file, you will find the following code:

```
rectangle my_rectangles[3];

initialization {

    my_rectangles[0].x = 10;

    my_rectangles[1].x = 30;
    my_rectangles[1].red = 0;
    my_rectangles[1].green = 1;

    my_rectangles[2].x = 50;
    my_rectangles[2].red = 0;
    my_rectangles[2].blue = 1;
}
```

The code declares an array type `rectangle` of size 3. This will create three different `rectangle` objects accessible in each index. Then, the code manually initializes each object's properties. Note that the code uses an `initialization` block to initialize the objects. The initialization must happen before the main. The output can be found in Figure 3.

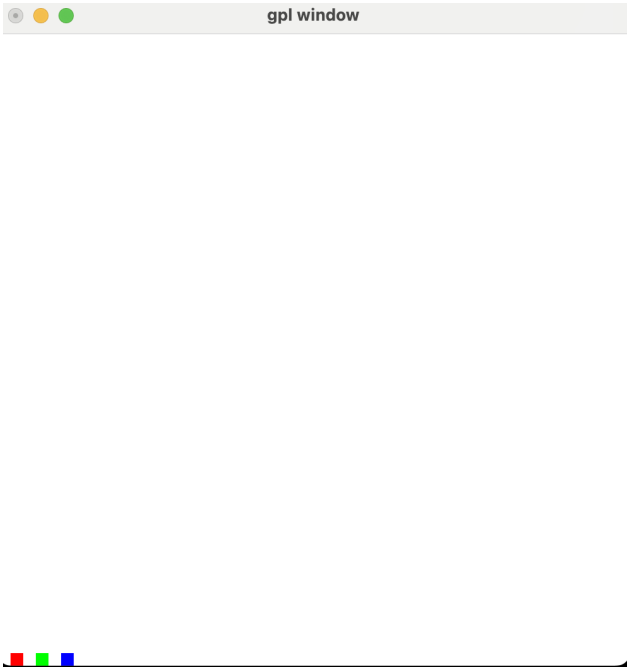

Figure 3: *multiple_rectangles.gpl*

Using the provided GPL manual, try out different types of game objects.

Animation Block

Animation blocks are named statement (see GPL manual for the supported statements) blocks. Animation blocks require a single typed parameter defined in the block's forward statement and the animation block header. This parameter can be of any game object type, such as `rectangle`, `circle`, or `triangle`. The parameter's name can be any valid identifier, but its type must correspond to the type of the object that will use the animation block. The following shows the syntax for declaring and defining an animation block.

```
// Think of the following as a function declaration.
forward animation <animation_block_name>(p_1, p_2, ..., p_n);
// Think of the following as a function definition.
animation <animation_block_name>(p_1, p_2, ..., p_n) {
    stmt_1
    stmt_2
    ...
    stmt_m
}
```

After defining an animation block, we can assign it to a game object by passing the animation block as an argument. This process specifies which animation block the game object should use.

```
<game_object_type> <object_name>(animation_block=<animation_block_name>);
```

The example code in *simple_animation_declaration.gpl* demonstrates the declaration of an animation block named `foo` that takes one argument of type `rectangle`. A `rectangle` game object, `my_rectangle`, is declared with all properties set to their default values, except for the `animation_block` property, which is set to `foo`. The animation block `foo` checks if the x-axis of the rectangle is zero (0). If true, it changes the x-axis value to 10 and sets the red color component to 0 and the green color component to 0.5.

```
forward animation foo(rectangle cur_rectangle);
rectangle my_rectangle(animation_block = foo);
animation foo (rectangle cur_rectangle) {
    if (cur_rectangle.x == 0) {
        cur_rectangle.x = 10;
        cur_rectangle.red = 0;
        cur_rectangle.green = 0.5;
```

```
    }
}
```

Execute the file to see the output, and try out the animation block using the provided GPL manual.

Event Handler

An event handler is a block of statements to execute when a specific event has occurred, e.g., pressing the move keys on the keyboard or moving the mouse cursor on the opened window etc. You can find the full list of GPL-supported events in the manual. The event handler block has the following syntax:

```
on <event_name> {
    stmt_1
    stmt_2
    ...
    stmt_n
}
```

To give you a better idea, let's look at the example that allows the user to move a rectangle using the keyboard's up, down, left, and right keys within the window. The first two lines declare two variables mapped to the reserved variables for the window's width and height. `window_title` is another reserved variable for setting the window name. In this example, it's set to "A Moving Rectangle". Then, the code declares a `rectangle` game object and initializes the properties using the initialization block. Finally, the code specifies the actions for each event. `leftarrow`, `rightarrow`, `uparrow`, and `downarrow` are reserved keywords.

```
int window_width = 1000;
int window_height = 500;

string window_title = "A Moving Rectangle";

rectangle rect();

initialization {
    rect.x = 10;
    rect.y = 10;
```

```
    rect.h = 15;
    rect.w = 15;
    rect.red = 1.0;
}

on leftarrow
{
    if (rect.x > 10)
    {
        rect.x -= 10;
    }
}

on rightarrow
{
    if (rect.x+40 < window_width)
    {
        rect.x += 10;
    }
}

on uparrow
{
    if (rect.y+40 < window_height)
    {
        rect.y += 10;
    }
}

on downarrow
{
    if (rect.y > 10)
    {
        rect.y -= 10;
    }
}
```

Run the program and try out different events using the GPL manual.

Task

Your task is to write a GPL program that implements everything we have covered in this lab. This can be a foundation for your first phase of the project. Write a program in which you declare a game object(s) and move your object(s). You can check out the provided *attack.gpl* program to start your program.

How to Compile and Test Your Code

Since GPL is an interpreter, you do not have to compile your *.gpl file. You just pass your *.gpl file to the GPL interpreter as a command-line argument: `$./gpl <file.gpl>$`

Expected Output

A working GPL program of your design.

How to Submit Your Code

The only file you will submit is your gpl file. Under the `Labs` directory, create a sub-directory called `Lab1`. You will add your gpl file. Add, commit, and push your file to GitHub.