CS49C Project Report: Gameboy Advance Game

by

Terry Hong, Ryan Tran, and Timothy Yang

ABSTRACT:

This project applies our current knowledge of the C programming language and uses external software to create our own Gameboy Advance game. Although assembly is the strongest programming language for Gameboy Advance games because it is flexible and provides improved performance when implemented correctly, the C programming language offers many other benefits when compared to assembly. Complex program development is simpler and easier to implement in C, and it is possible to write assembly in C. Therefore, C is the most commonly used programming language for creating Gameboy Advance games.

Libraries Used:

Several libraries and software programs were used in the project. The Gameboy Development Kit (GBDK) is a freeware resource for non-commercial developments provided by Michael Hope, and it includes the necessary Gameboy Development libraries as well as a .exe application to compile all files into a runnable Gameboy Advance game. The GBDK can be installed anywhere on the computer, but the directory chosen for this project is at “C:\gbdk” so that the “make.bat” file that compiles the code and the code editor can easily access the GDBK.

All seven software programs in the project are free to the public. The Gameboy Tile Designer and Gameboy Map Builder programs are provided by Harry Mulder. Gingemonster built the GameboyPngConverter for free and the Gameboy emulator came from http://bgb.bircd.org/. Microsoft Paint and GIMP are photo editing programs used to draw the welcome screen of the game. A code editor named Visual Studio Code was chosen over an integrated development environment (IDE) such as Microsoft Visual Studio because the code was compiled from an executable in the GDBK.

In addition to these software programs and libraries, an online video tutorial about Gameboy Advance game development created by GamingMonsters from Youtube was followed as a reference to learn about how Gameboy Advance games functioned. Once we learned how to implement sprites, backgrounds, player input, text, and collision detection in our game, we were able to write our code and draw our art to create our very own Gameboy Advance game.

Background Information:

A Gameboy Advance game stores information in the Read-Only Memory (ROM) of the program. The memory consists of an array of pockets of bytes called “tiles.” Each tile is one byte—8 bits—and is represented by unsigned chars in a hexadecimal format (0x00). These tiles are crucial for Gameboy Advance games because they allow sprites and other background images to appear in the game. The tile table for sprites resides at the top of the VRAM memory, and the tile table for the background is at the bottom.

The tiles for the sprites and background data were created with the Gameboy Tile Designer and Gameboy Map Builder programs. The welcome screen was drawn in Microsoft Paint and modified with the GameboyPngConverter program. The converter imposed a maximum of 4 colors, so GIMP was required to index the colors of the image. All image files were converted to arrays of type unsigned char so that they could be loaded into memory. Once the art was completed, they were exported into 6 files: “Sprites.c”, “Sprites.h”, “Background.c”, “Background.h”, “BackgroundTiles.c”, and “BackgroundTiles.h”. The #include directives were used to access the generated .c files in “main.c”. Tile management is crucial because memory is not unlimited; the entire tileset for one map cannot exceed 512 tiles. Therefore, all tiles in memory are reset once the welcome menu has ended to open up space.

Implementation:

All of the code in this project is written in the “main.c” class, which is compiled using a “make.bat” file that contains the instructions, file paths, and file names necessary to create a file with the “.gb” extension that runs the Gameboy Advance game with the Gameboy emulator. The code is implemented with the goal of maintainability and modularized with functions that perform a specific set of instructions. The program is compiled by running the command “./make.bat” in the terminal of Visual Studio Code.

The main method loads all sprites, background tiles, and font data into the memory and builds the player and enemies. Afterward, all loaded data and the display are turned on. An infinite loop keeps the game running until there is a collision between the player and the enemy. The score and character positions are updated in each loop iteration, as well as checks for collisions. There is a delay in each loop to slow down each game frame so that none of the sprites move “instantly” since each loop occurs in milliseconds. The game is ended by a printf statement that says prints Game Over on the screen.

Each sprite in the game is a character that represents the enemy or the player. A separate class “GameCharacter.c” was created to represent the characters. There are two different structs; one for the enemy and one for the character. Each of them uses a typedef alias to improve readability. Each game character struct has a sprite ID, the x and y position, the width, and the height of the character. The player is an array of type unsigned 8-bit int because the player is represented as 4 of the same sprites glued together as a square shape, and all 4 sprites must be updated when moving. The move functions for the game characters update the x and y position of the sprites by having pointers to structs, so each member is accessed with the arrow operator.

The update functions move the game characters and ensure that they do not move out of the map's boundaries. The player is updated using a switch statement with the joypad for each case at a speed of 5 pixels. The player cannot move when they reach the boundary of the map, similar to the Turtle Graphics assignment done previously in the semester. Enemies reset their y-position at the top of the map and their x-position equal to the player to track the player. The enemy speed is calculated using a formula in a for-loop statement. The formula "enemy[i].y += 4 * (i / 3) + 1" moves each enemy according to their ith position multiplied by 4/3. The division by the denominator results in two fast enemies and three slow enemies, and the addition at the end ensures that the first three enemies in the loop do not have a speed of zero .

Collisions between the player and the enemies are checked by examining the x and y coordinates at the top left corner of the sprites. In addition, the weight and height of each sprite is added accordingly to each case to allow detection for the entire sprite. The player cannot be inside or touching the enemy, and the enemy cannot be inside or touching the player. An unsigned 8-bit integer value is returned using the entire conditional in one line.

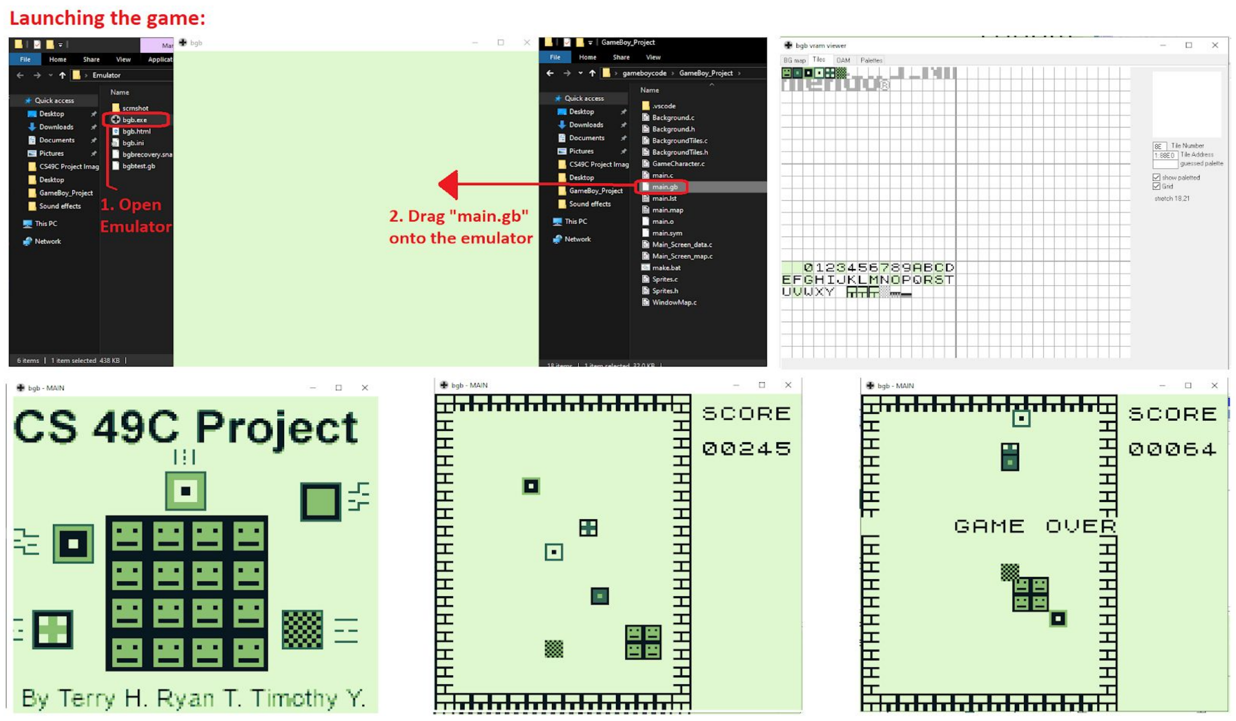
Since the score is represented by an array of unsigned chars rather than a numerical value, each char has to be updated for the score to change. If a char represents a nine, it is reset to 0. If the char in front is a 0, the char behind is incremented by 1. It is important to check for all char values at once because all values are 0 by default. Short-circuiting is implemented by making the condition most likely to be false appear first in the conditional statement.

C Concepts learned:

Interestingly, we noticed that we could utilize the branching goto statement from chapter 14.10 of the textbook for loop termination when the game ends after a collision is detected. The goto statement allows a change in the flow of control to the first statement after the label. As a result, the program will exit the infinite game loop if the label is placed after the loop.

We encountered a peculiar error during the coding of the program when we tried to initialize a counter variable in the for loop statement. If we initialized “int i = 0” in the for loop, a parse error would occur that complains about undefined identifiers. This issue was resolved by declaring the counter variable at the beginning of the method outside the loop then updating the counter variable inside the loop. We believe the error to be either a part of a secure C programming or an old compiler issue since the compiler is made for the Gameboy Advance.

Screenshots:



Additional project gameplay footage: https://youtu.be/EDdWl3Z1DLU