Programming Consoles Report CT5PROGC

Przemyslaw Hendel, UP739235

Your critical analysis and reflection on the development of the Game Boy Advance artefact, demonstrating a clear understanding at the appropriate technical level. References to literature and documentation (in Harvard APA format) should be used to justify your points as required. This deliverable covers learning outcomes 1 and 2.

In this report I will critically analyse and reflect on the development process of an artifact for Game Boy Advance. I will cover the problems I had, and any issues I found while creating my game.

# devKitPro

First of all, I’m going to start with devKitPro, the library that we had to use for creating our artifact. It does not have any documentation that tells a developer how to use the header files from the library and its utilities to bring more understanding to someone who has never had any experience with Game Boy Advance development. Although the code from the header files is understandable and can make the programmers life easier it is not commented everywhere so either previous experience with GBA is required. Furthermore, the developer has to spend extra time for doing research, for example different Display Control Register Modes and what they can be used for or about parts of memory that are responsible for different things such as palettes, sprites and so on.

Secondly, the MaxMod sound library has no documentation either, after doing research I found that the website that used to have the documentation is down, and some different non-programming related website is using that domain. Mukunda Johnson is the creator of Maxmod, and his website also doesn’t have any documentation links.

# Assembly

The THUMB mode that we were told to use when using assembly for our game is very limited, and only allows the use 8 registers. Whereas the ARM instruction set lets the developer use all of the 16 registers available. THUMB has an advantage over ARM in speed in the ROM but it’s stripped out of many instructions that are available in ARM and only branches can be used to create *for* loops or *if* statements. Also it can be confusing to write or read code written in the assembly language. What I also noticed when creating the score multiplication in ASM is that it is slightly slower when comparing it to the same code I firstly created using C. I also found the ARMv7 documentation when I was looking for some explanations on how to store a hexadecimal value in a hexadecimal memory address using ASM, I have spent quite a bit of time reading it and trying to understand how it works.

I could have done a little more in ASM, apart from multiplication of score and setting the palettes colour in it. So my C code would be shorter and my game would take advantage of partly working directly on the ARMv7 processor registers.

# Development

After finishing most of the tutorials available on Moodle apart from the sound/Maxmod as I couldn’t get it to work as it didn’t generate the files it supposed to after modifying the makefile (header file and binary) I started working on my game. I created sprites and added them into my game, unfortunately I ran out of time and didn’t animate them like I planned to do.

Then I wanted to create a menu screen, as I managed to create a menu, game and a screen that would appear when the player loses. So for the Menu state I have used Display Control Register’s Mode 4 to display my image after reading a little about it on the internet and looking at other people’s source code on GitHub. The problems I quickly noticed were that I was not allowed to use JPEG or PNG formats, and again after doing research I found out that I must convert image data into a format that the GBA accepts, so I used the Usenti converter to convert a .pcx image file format into .c and .h file that contains tile and palette data which could be copied into memory creating a for loop that drew each pixel from the data in C and H file. While I tried to get the bitmap to show on screen I encountered another problem that caused my image not to display correctly. This was a memory problem, DSPCNT Mode 4 uses page flipping that draws an image into another frame before swapping and displaying the frame buffers that have been drawn to. At the beginning I used the Frame 0 that had the address of 0x06000000 which messed up my bitmap, and has overwritten the default text tiles that are held in the same bit of memory. So I changed the address of my menu/splash image to use Frame 1 at 0x0600A000 by setting bit 4 of DSPCNT to 1. As I didn’t animate my bitmap I only needed one frame so nothing was overwritten and then the menu bitmap showed correctly, 240x160 pixels which is this modes resolution (Harbour, 2005).

Then after looking on source code on GitHub, I found GBA source code where a parallax scrolling background was implemented (idolize, 2013) and I have used this as a resource to implement one myself using Mode 3 of Display Control Register. I have used dmaCopy from gba\_dma.h in libgba to copy the bitmap and palette data to memory, that I could later use to draw two of my images in different BG Layers with different display priorities. I also used BGHOFS to make the bitmaps wrap, and start showing on the other side of the screen when the bitmap moves into the invisible part of the screen.

Collision detection on the enemies could be improved and work like it’s supposed to, I have struggled with making it, and I’m not yet sure what I would have to change to make it work. I could possibly do more research as there might be an explanation somewhere on how this problem should be approached to get the results wanted. For the collision detection I have also got lots of code that repeats which could be changed so I have a function that deals with creating sprites and collision detection. Each sprite is created manually, basically copied and pasted like it shouldn’t be. I have used stdlib.h to make the enemies change their X and Y positions to a random value between as soon as they disappeared from the screen. While doing that I found out that after the enemy character leaves the visible part of screen which is 240 pixels in width it travels another 256 pixels before it shows up on the other side of the screen.

# Conclusion

This part of the unit was really fun and definitely has improved my programming knowledge, because I had to deal with memory directly to create a game where the memory had areas that were responsible for different things, such as palettes or sprites. I have also learned about bit shifting and the ways of how to use up the memory when creating variables efficiently so no memory would be wasted. It has also made me think about how companies had to manage the memory and think carefully about implementing a certain feature with the limited memory that Game Bot Advanced has, to make their games use its full potential. My game has many things that could be improved or added such as sound, and previously mentioned collision detection, sprite animations and code that could be written more efficiently. Also I could use more of the ASM to move the code from my C files into it for better performance.

# Bibliography

Harbour, J. S. (2005). *Programming The Nintendo Game Boy Advance: The Unofficial Guide.* Course Technology PTR; 1 edition.

idolize. (2013). Game boy Advanced game: RavenX. Retrieved from https://github.com/idolize/ravenx-gba

ishaangrover. (n.d.). gameboy\_advanced. Retrieved from https://github.com/ishaangrover/gameboy\_advanced

Johnson, M. (n.d.). Retrieved from mukunda: http://www.mukunda.com/projects.html

Mike Huber, G. (n.d.). *GBA ASM Tutorial*. Retrieved from patater: https://patater.com/gbaguy/gba/ch1.htm

Vijn, J. (n.d.). Usenti - A Bitmap editor. Retrieved from http://www.coranac.com/projects/usenti/