University of Portsmouth

Computer Games Technology

CT5PROGC

GBA REPORT

UP784120

I started the project by thinking about how I want to organize the data into the memory. It was important to select from the start the base blocks that are going to contain character data and the ones that are going to contain screen data in order to avoid the overlapping between these two.

**Map Design:**

When it came to designing the map, I had to choose from two possible solutions. One was to individually draw each tile on the screen by indicating it’s x and y position and the other one was to generate the map by setting some conditions in order to place the wanted tiles in the right place. I used the formula that describes the position of the tiles, base\_block\_addresss [(y\*32) +x], in order to jump through memory and draw the map data. The screen location of each tile is updated by using a variable that is indicating the x position of the tile on the screen, variable which takes values from 0 to 32 because there are 32x32 tiles on the screen.

I had to define the y position of the tiles manually because I had some custom tile designs that needed to be defined in a certain order to get the desired design. I made us of the multiple layers to create the scrolling visual effect. At first I tried to update the position of each tile to get the scrolling effect but, using the documentation, I managed to get access to the x and y values of the backgrounds and this made the process of moving the backgrounds easier (GBATEK, n.d).

I tried to generate the map data and move the backgrounds by using LLC code. At first the map data was design by using a for loop but this approach lead to performance issues, because all the other processes were stopped while the map was drawn on screen.

There is a small graphical error with the 1st layer of the map that happens because the objects are drawn continuously. Normally, the drawing process should happen just once.

**Objects:**

Each object that is presented in the game has a defined space in memory even if it appears on screen or not. I think that during some stage of the game the memory occupied by objects that are not part of the respective scene can be emptied and overwritten. I tried to use this approach in my artefact but the code became too ambiguous and it was hard to read and understand. Because of this I had to manually turn on and off the visibility of the objects when game states were changing. This memory management wasn’t a real problem for my artefact because there was enough memory to store all the objects and tiles but it might become a problem when working with a larger number of tiles and objects.

I made use of the objects attributes such as color pallets and rotation in order to create different looking objects without adding more data to the character data base block. Each object has some predefined variables that indicate their position and different states during the game. I believe that I didn’t manage to organize my code in a professional way and this affected my artefact stability because it was hard to manage a large number of variables.

For most of the objects I created functions that are used to set the object’s x and y coordinates and their visibility status in order to make it easier to set objects and to remove them from the screen.

**Game Logic**

The game is built on three game states, PLAY, PAUSE and SET\_LEVEL, the PLAY game state enabling three enemy waves. Additional states were used for the ending of the game .The three waves are identical and the only thing that is different is represented by the enemy plane position. At first I tried to create a function that had as arguments the coordinates of the enemy plane and the next wave that should be played. This approach wasn’t working because of the large number of variables that should have been handled. I came up with a new solution that consists in adding if statements that are checking the state of the current wave every frame and are updating the games states if it is required.

**Collision test:**

I had some problems when it came to implementing the collision detection. The best solution would have been to create a collision test that calculates the distance between two points using the Pythagorean theorem. I had problems with the math.h header file and this didn’t allow me to use the sqrt function. The collision test was done by manually checking if the x position of the objects is the same and if the y positions are situated in defined range, for example: if (rocketY>=planeY-5 && rocketY<=planeY+5).

I tried to implement the collision test by fully using the low-level programming. One of the problems was transforming an if statement that has more condition (the one presented above) into assembly language. I did find out that using conditional execution might be helpful but unfortunately it seems like Thumb does not support conditional execution. (Fuber, 2000, p65)

I managed to find another solution that consist in using the BLT and BGT branch conditions in order to check if the y position of rocket is in range of the y position of the plane (Fuber,200, p 64). Each of the two condition adds one two the variable that returns the final result of the function so if the returned value is 2 it means that both conditions are true.

The collision test for x-axis was consist only in one if statement that was transformed intro low-level programming code by using the BNE branch and returns one if collision is detected.

**Sound:**

As I stated before, the usage of for loops lead to performance issues but it was also affecting the sound. The sound was distorted because of the delays that were caused by for loops.

I had to add some control variables that helped me to control the sounds in my artefact. I used both ‘jingles’ and ‘modules’. Jingles allowed me to play more than one sound in the same time. One of the problem of modules is that the song stops when another one is starting. My artefact still has some sound problems, for example if two sound effects such as ‘hit sound’ or ‘shoot sound’ are played one after another, the first one that was played will be will be canceled.

**LLC**

In addition to the collision test and map drawing, I also used the low-level programming language to create some basic functions that use the ADD, SUB AND MOV commands. These were mainly used to update object’s position.

One important thing that I learned while working with LLC is that you have to pay attention to the variables that are returned back into C from the assembly function. If the variables that are passed as arguments modify their values inside the assembly function and then their new value is stored, there is a possibility of breaking the code because LLC works with memory addresses and the values of the arguments are going to be updated through the whole code and this can affect the behavior of other functions.

**Reflection:**

I believe that my artefact shows good evidence of GBA’s technical features and also, I managed replace some bits of the C code with LLC code. Most of the Low-Level Code that is implemented is based on basic functions such as add or sub but I also managed to use some unusual commands such as BLT and BGT. I think that I focused too much on other aspects of the game, such as graphics because, I postponed the implementation of LLC until the very end. My initial approach was to firstly finish the game and then try to convert it in LLC code.

I consider that I should work more on organizing and structuring my code because when I am going to work in the industry, I’ll have to share my code with other persons and it should be easy to understand. Also, I think that a good organization of the code can help you to find and fix the bugs quickly. I admit that know my code might be hard to follow.

Even if in my artefact, the LLC code doesn’t have a key role in increasing my code’s efficiency probably it will be extremely useful in the future when I am going to work on more complex games that need to be optimized. From what I remember from the first part of the year, even the PS4 profiling tool was presenting the code under LLC format.

Reference:

Fuber, S. (2000). *ARM System-On-Chip Architecture*. Harlow: Pearson Education Ltd.

GBATEK. (n.d). Retrieved from: http://problemkaputt.de/gbatek.htm