Felix Eder 950624-7353

Henrik Kälvegren 960305-5451

Stefan’s Journey - A 2D Sidescrolling Adventure

1. **Objective and Requirements**

We set out to do a side-scrolling game in which the player has to jump over various obstacles as well as shoot enemies and collect items. The player was going to have two buttons, one for jumping and one for shooting. We planned on having both a score, health and ammo counter for the player, which were to be displayed using either LEDs or the screen. We thought this would be a reasonable goal and wouldn’t be too much work. As we started taking on this task we noticed just how hard the task really was. We still wanted to implement everything that we said that we should do, but make a minimum viable product (MVP) to begin with. This includes the character, rectangles moving toward the player, as well as the ground. The only input from the player is one button to make the character jump over the rectangle. If the player fails, health is drained. Even this goal was hard to reach and we decided that we would try to reach the MVP and then add the other features if we had time.

2. **Solution**

Our biggest problem with the project was understanding how we should update the screen with our own data. We started to look into how the older labs from the course did it. We read through the C- and Header-files in order to understand how they used it. We also looked through the ChipKit reference manual on how the display worked. We understood that it was somehow divided up in four rows, where each row contained 32 columns consisting of eight pixels each. In order to update the display, you have to set it to an array of 128 values (one for each column on the entire display). Each value can be 0 to 255 which is 2^8. If we converted each number into an 8-bit binary number, each bit would represent one pixel in the column of the current row. If we wanted to draw something, we had to make sure to enter the correct value for the correct column in order to turn on the correct pixels. Once we figured this out, it became a whole lot easier to draw our simple figures.

We decided to draw each of the figures with it’s own array and function call. This meant that some pictures overlapped each other when they came too close, which was a major problem. Our solution for that was to use a special merge array when this occurred. It copied the value of the first image’s values and then calculated the number of pixel columns that overlapped. It then OR-ed these pixels with the second image and then displayed the result. This was done for each frame and works just as intended.

Collision was handled by creating hitboxes for each game object. Hitboxes were calculated from the objects’ widths and heights relative to their X and Y coordinates. When two hitboxes overlapped each other, for example the character and a rectangle, one life was drained from the player’s health.

The rest of the game was simpler to make. After we set up everything, we start a while-loop which runs throughout the game. In each iteration, input from the player is taken in, a logic function is called that uses that input and makes the player-struct do the appropriate thing (could be to jump or change a value). After this, a screen function is called that updates the screen with the new information and then finally the LEDs are updated as well. This loop goes on and on until the player has no more health.

We also use some simple text prints in order to display information for the player and the story of the game, this is made in the same way as the display_string function of the labs. It is however only a minor part of the game.

3. **Verification**

In order to verify that our game actually worked as intended, we used a variety of methods. Our most common practice was to actually play the game and see if it worked as we wanted it to. Most often than not, it didn't and we had to change something. Since our game doesn't use very complicated logic we could therefore debug the majority of the code in this way. Other methods included printing out information to the screen when a specific thing was going to happen. For example, we had trouble getting the character to jump on input and therefore added print-statements on two places in the code. The first was printed when the game recognised the input and the second after the character should have jumped. By doing this we saw were the fault was (wrong use of pointers) and managed to fix it.

4. **Contributions**

We didn't assign out work beforehand but as we made the game, we did both take on different tasks. Henrik wrote most of the code that updated the screen with the correct information. This was by far the most difficult part of the project and it did take him a long time to get it right. Felix of coursed helped out but Henrik was the one who made the most of it. Henrik also wrote the code about collision detection and the merge process. Felix handled the I/O, initialisation, the main game loop as well as the logic. No single part was as complex as Henrik's, but it was still a lot of code to write and it did take him a while. We both contributed equally to the player's file as well as the header-files. We both used them for different reasons and both added to them.

5. **Reflections**

We learned two things from making this game. The first thing, don't bite off more than you can chew. We had high ambitions for this game and that is a good thing. Even if we never quite got there, it was always nice to have this idyllic picture of our perfect game in our heads that made us push forward through all manner of hardships. The problem was that it would simply take too much time to actually reach that goal. Setting goals is great, but we now know to keep our expectations somewhat in check. And that leads us in on our second lesson we learned, start in time. We did wait a long time before we started and it did take a while before we made any meaningful progress. This of course resulted that we had to do a 3-day marathon of coding by ourselves just before the deadline in order finish in time. It was however a very fun project overall and we learned some valuable lessons as the cherry on top.