CSED211

Project Final

**굳세어라 간조메**

good se o ra GAN JO ME

플 젝 충

CSE 20140658 최유정

CSE 20140996 황일환

Content

[1. 프로그램 개요 3](#_Toc438500644)

[2. 프로그램 소개 4](#_Toc438500645)

[1) 게임의 내용 4](#_Toc438500646)

[A. 시작 4](#_Toc438500647)

[B. 게임진행 6](#_Toc438500649)

[C. GAME OVER 7](#_Toc438500650)

[D. Happy Ending 8](#_Toc438500651)

[2) 프로그램을 만들기 위해 활용한 devices 8](#_Toc438500652)

[A. 망고보드 8](#_Toc438500653)

[B. 망고보드 - Grapics2D 8](#_Toc438500654)

[C. 망고보드 - Post Processor 9](#_Toc438500655)

[D. LAN 선의 활용 9](#_Toc438500656)

[3) 프로그램에 사용된 소스 10](#_Toc438500657)

[3. 프로그램 실행 방법 설명 10](#_Toc438500658)

[4. 토론 및 개선 사항 10](#_Toc438500659)

[5. 결론 12](#_Toc438500660)

[6. 느낀점 12](#_Toc438500661)

# 프로그램 개요

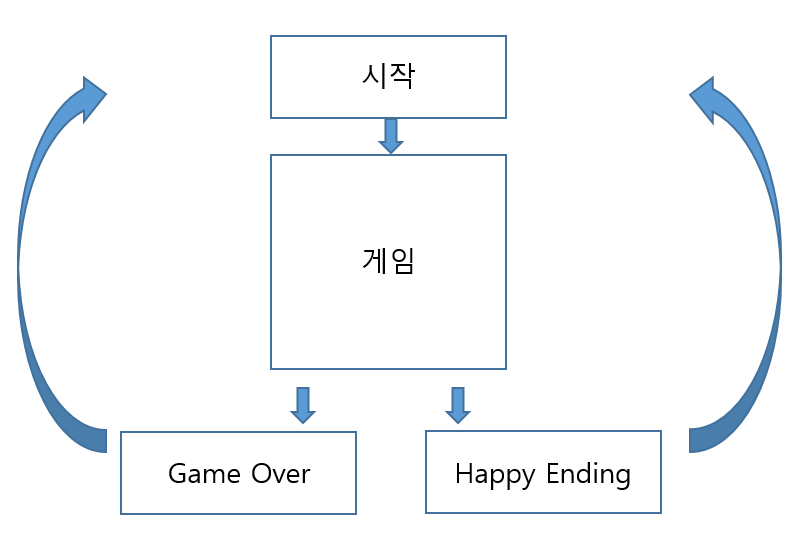
망고보드를 슈팅 게임을 만들기로 한 이상 정말 재미있고 흥미로운 게임을 만들고 싶었다. 띠라서 본 프로젝트의 목표는 수업시간에 배웠던 여러 지식과 랩시간에 배웠던 망고보드를 다루는 방법을 녹여내서 아름답고 재미있고 흥미로운 게임을 만드는 것이었다.

그래서 여러가지 게임 시나리오들을 생각하다 일상생활에서 발견할 수 있는 재미있는 상황들에서 게임의 아이디어를 착안해서 게임을 만들도록 했고, “굳세어라 간조메”가 탄생하게 되었다. 평소 최유정 학생은 검고 동그란 얼굴에 간장에 조린 메추리알 일명 간조메라는 별명을 가지고 있었다. 이 아주 해맑은 학생이 임범수 학생을 부를 때면 그는 언제나 일관적으로 부정적인 대답과 폭언과 폭행을 일삼았다. 이런 상황 속에서 유정 학생이 굴하지 않고 범수 학생과 잘 노는데, 이런 상황을 십분 게임에 반영하여 게임을 만들었다. 이 게임은 간조메가 범수가 깔아 놓은 모든 방어에도 불구하고 꿋꿋하고 굳세게 범수를 찾아가서 범수와 즐겁게 놀려고 하는 게임이다.

이러한 시나리오를 바탕으로 범수를 쫓아가는 슈팅게임이 만들어 졌으며, 아름다운 그림을 제공하는 실시간 게임을 만들기 위해서 아주 많은 공을 들인 게임이다.

게임의 전체적인 구조는 다음과 같다. 크게는 4가지 state로 구분될 수 있다.

게임의 제목이 떴을 때 터치를 하면 총 3 STAGE로 구성되어 있는 게임이 시작되며, 모든 life를 다 사용하게 되면 간조메가 범수를 만나지 못하는 Game Over state가 나오게 되며, “잉 8ㅅ8” 이 표현되며, 만약 모든 stage를 clear하게 되면 범수를 만나는 Happy Ending state에 도달하게 된다.

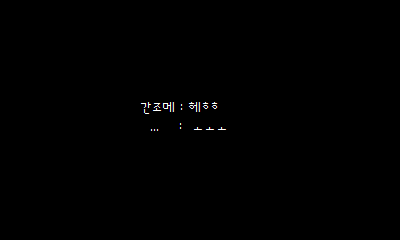
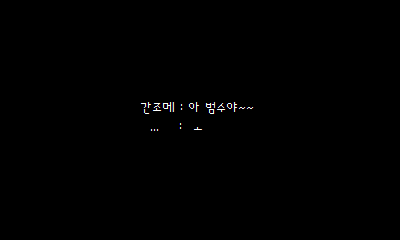
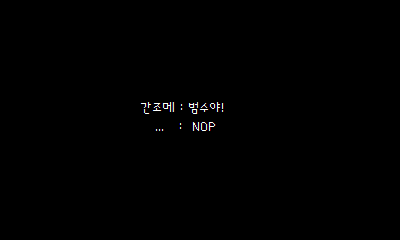


# 프로그램 소개

## 게임의 내용

## 시작

## (다음 사진들이 연속적으로 진행이 되어야 했으나, 파일의 크기가 크기도 했거니와 게임의 단순함과 심플함을 증가시키기 위해서 실제 구현에서는 제외시켰다.)





실제로 구현되어 있는 시작화면에는 굳세어라 간조메라는 게임의 제목이 있으며 이 화면을 touch하게 되면 다음 state인 게임 중인 state로 넘어가도록 된다.

## 게임진행

* PLAYER

게임에서 player는 간조메이다.

* ITEM

간조메는 간장이 잘 조려졌기 때문에 default 아이템으로 간장을 뿜으면서 게임을 진행한다. 아이템은 총 3개가 존재하는데, 적들을 무찌르면 아이템이 나온다. 간장, 컬러 “ㅎ” 빔, 꽃의 3가지 아이템이 존재하는데, 간장은 직선으로 3갈래로 갈라져서 나가고 있으며, 컬러 “ㅎ”은 빨주노초파남보를 번갈아가면서 싸인의 파동을 그리면서 “ㅎ”이 나간다. 이것은 간조메가 평소에 아주 시끄러운 소리로 웃는 것과 아주 유사하다. 그리고 꽃은 아주 흩날리면서 사방으로 날아가는데 사실 아주 화려한 효과가 있는 것에 비해서 그 데미지가 아주 약하기 때문에 이 아이템을 피하는 것도 일이다.

* CONTROL

게임의 조작은 왼쪽 아래에 있는 d-pad를 통해서 이루어진다. 이 패드는 중간 지점을 기준으로 상하좌우로 움직이면 패드 상에서의 위치에 따라서 player의 속도를 조절해 준다.

-STAGE

게임은 총 3 stage로 이루어져 있다.

첫 번째 스테이지는 범수가 가장 많이 찾는 파트너 쿤 STAGE이다. 이 스테이지에서는 마운틴 듀를 좋아하는 승규를 반영하여서 적으로 마운틴 듀가 등장하며, 술을 못 먹는 간조메에게 술을 먹여서 죽이려 하기 때문에 순하리 들이 등장한다. 그리고 보스로 승규가 등장하는데 평소 따봉충인 승규는 게임 내부에서도 따봉을 날린다.

두 번째 스테이지는 디지털 STAGE이다. 이 STAGE는 꾸준히 간조메의 project 파트너였던 황일환 군이 프로젝트를 하기 위해서 간조메를 잡아가려하는데 이를 막는 것이 굉장히 중요하다. 이 스테이지 에서는 장벽으로 나오는 NOR gate를 뚫고, 쏟아지는 NAND gate를 피하며 던져지는 AND gate를 피해야만 한다. 이를 피했지만 방심할 수 없는게, 디그다가 그려진 망고보드가 나와서 마구 망고를 던지기 때문에 이를 조심해야한다. 그리고 이 스테이지의 보스는 광광이라는 바이크를 탄 황일환이 등장하는데, 무자비 하게 gate와 망고를 뿌리는데 이를 조심해야한다.

그리고 세 번째 스테이지에서는 드디어 범수를 만날 수 있다. 평소 아이폰을 잡고 “ㅗ” 와 같은 압정을 아주 많이 날려대는 범수를 반영하여 게임을 진행하기 시작하면 범수가 압정을 펑펑 뿜어낸다. 그리고 그 압정을 피하면 분홍색과 노란색의 탄막들이 웃는 범수의 얼굴과 화난 범수의 얼굴에서 쏟아져 나온다. 이 탄막들을 모두 피하는 것은 아주 힘든 일이다.

매 스테이지에서 보스를 죽이면 라이프가 하나 더 생겨서 게임을 더 진행할 수 있도록 해준다.

이 모든 스테이지와 보스를 이기고 나면 드디어 간조메는 범수와 함께 행복한 happy ending을 맞이할 수 있다.

## GAME OVER

모든 LIFE를 소진시 Game Over 화면이 출력된다. “잉 8ㅅ8” 이 나와서 범수가 놀아주지 않아서 한껏 슬픈 간조메의 마음을 담는다.

## Happy Ending



범수를 만나서 행복한 간조메가 광광 뛰고 있는 모습을 나타낸다. 실제 게임에서는 위 그림은 크기가 크기 때문에 올라가 있지 않고, 흰 배경에 범수가 웃고 있는 모습이 나타나지만 그 그림을 본 기억은 단 한 번뿐이다.

게임의 난이도 혹은 간조메를 보고 싶지 않은 범수의 굳은 의지 때문에 happy ending을 보기는 아주 힘들며 마치 unreachable state와 같은 느낌을 준다.

## 프로그램을 만들기 위해 활용한 devices

### 망고보드

S3C6410X Microprocessor가 탑재되어 있으며 OS가 깔려 있지 않은 Mango board 64

### 망고보드 - Grapics2D

실시간 게임을 만들기 위해서는 Mango board에 탑재되어 있는 Grapics 2D chip을 활용하여야만 했다. 만약 이 chip의 힘을 빌리지 않고 전 화면(480 \* 800)을 흰 색과 검은 색을 번갈아 덮도록 했더니 약 1초에 3 frame 정도의 속도를 보였다. LCD를 다루는 lab시간에 배경이 바뀌도록 했을 때에는 화면이 불러지는 속도 때문에 그림이 밀리는 것을 볼 수 있었다. 게임을 하기 위해서는 적어도 30frame이 필요했다. 따라서 그 내부의 그래픽을 가속시켜주도록 했다. Off-screen 이미지를 screen에 copy를 함으로써 아주 빠르게 그림을 그려줄 수 있었고, 게임을 하기 적합했다.

또한 V-sync(vertical syncronization)을 구현시켜서 그림이 왜곡되지 않고 출력되지 않으면서 60 frame이 올바르게 출력되도록 하였다.

### 망고보드 - Post Processor

우리는 480\*800의 모든 해상도를 활용할 수도 있지만, 그림이 고전 도트 스타일의 그림이 되는 것도 아주 귀여울 뿐 더러 보다 빠른 속도를 위해서 해상도를 낮추는 것이 필요하다. 따라서 post processor를 이용해서 길이 1/2 (넓이 1/4)의 사이즈 그림을 전 화면에 출력시키도록 함으로써 더욱 속도를 높였다.

### LAN 선의 활용

tftp server을 이용해 UART 대신 LAN 선을 통해서 data를 전송했다. 그림이 많이 올라가기 때문에, 빠른 속도의 전송이 필요했다.

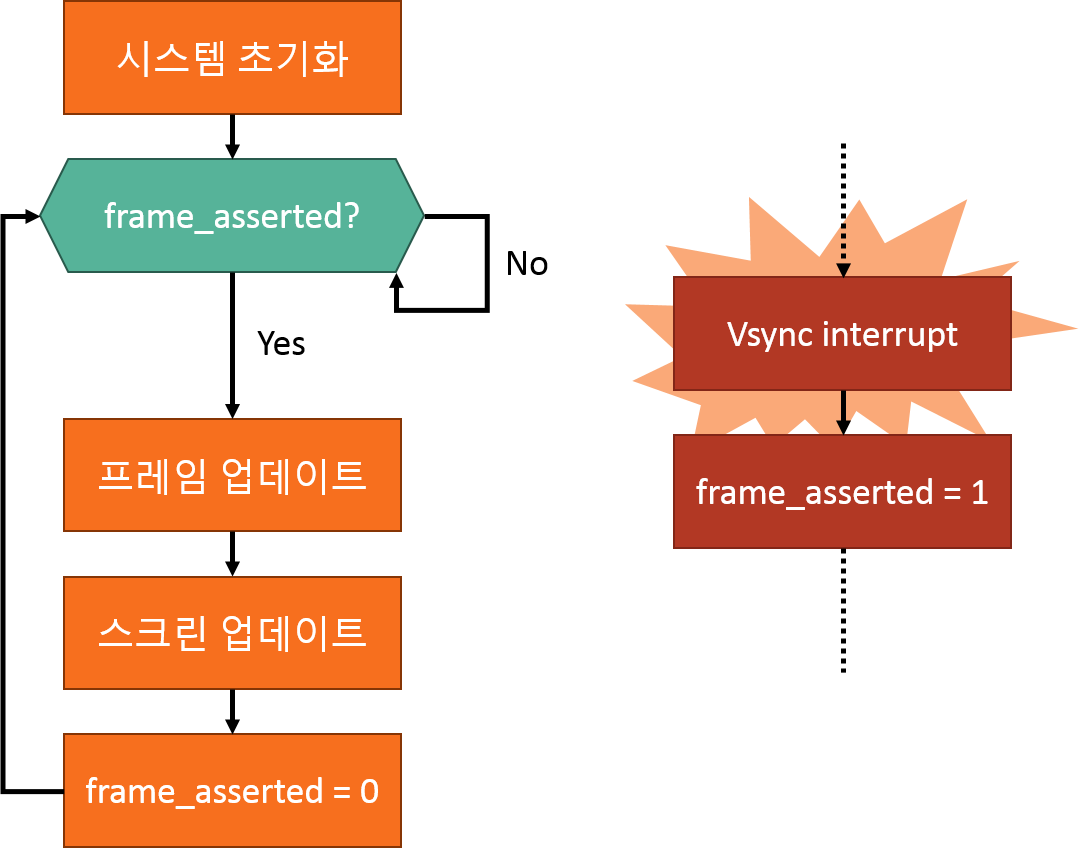
## 프로그램에 사용된 소스

## 개요

다음은 이번 프로젝트에 사용된 소스코드 목록이다. 총 23개의 소스코드 파일이 사용되었으며 이에 따른 헤더파일 또한 목록에 포함된다.

|  |
| --- |
| **망고보드 시스템** |
| s3c\_uart.c  s3c6410.h  support.c  interrupt.c  vic.h  io.h |
| **그래픽 프로세싱** |
| lcd.c  graphics.c  postproc.c |
| **엔트리포인트** |
| main.c  frame.c |
| **유틸리티 함수** |
| util.c |
| **게임 백엔드 시스템** |
| sprite.c  cbox.c  timeline.c |
| **게임 미들엔드 시스템** |
| bullet.c  enemy.c  effect.c  item.c |
| **게임 프론트엔드 시스템** |
| player.c  pad.c |
| **게임 진행 관리** |
| action.c  smanage.c  stage1.c  stage2.c  stage3.c |

## 흐름도



## 소스코드 설명

### 망고보드 시스템

망고보드 기본 시스템과 관련된 소스들이다. 이들은 랩 세션에서 제공되었던 소스코드를 거의 그대로 사용했다. 대신 타이머 인터럽트는 제거하였고 터치 인터럽트에서 실제 위치를 계산해서 외부로 인터페이스를 제공하도록 수정하였다. 또한 소스코드를 조금 정리해서 vic.h등의 헤더파일로 모았다.

#### vic.h

Vectored interrupt controller와 관련된 레지스터들을 모아놓았다. 기존에 제공된 소스코드에는 타이머와 터치를 관리하는 인터럽트가 모두 각각의 레지스터 정의를 따로 가지고 있었기 때문에 다른 인터럽트 확장성을 위해 따로 빼서 정리했다.

### 그래픽 프로세싱

CPU를 이용한 단순 메모리 연산은 굉장히 느리기 때문에 LCD화면 전체를 커버하는 그래픽을 제공하기에 적합하지 않다. 그래서 s3c6410x 마이크로프로세서가 제공하는 Graphics2D와 Postprocessor기능을 이용해 그래픽 가속을 구현했다. 또한 LCD와 수직동기화를 하기 위해 타이머 인터럽트 대신 LCD Vsync인터럽트를 사용해 깔끔한 그래픽을 꾀하였다.

#### lcd.c

|  |
| --- |
| void vsync\_interrupt\_service\_routine(void) {  static int serviced = 0;  unsigned temp;  temp = VIC0INTENABLE\_REG;  VIC0INTENCLEAR\_REG = 0xffffffff;  if (S3C\_VIDINTCON1 & BIT1) {  frame\_assert();  if (serviced >= 60) {  printf("Serving %dfps\n", frame\_count);  serviced = 0;  frame\_count = 0;  }  S3C\_VIDINTCON1 = 0x2;  }  serviced++;  VIC0INTENABLE\_REG = temp;  }  void lcd\_init(void){  lcd\_bl\_on(MAX\_BL\_LEV-1);  lcd\_pwr\_on();  init\_lcd\_reg();  set\_lcd\_pos(0, 0, S3CFB\_HRES, S3CFB\_VRES);  frame\_init();  set\_wincon0\_enable();  set\_vidcon0\_enable();  S3C\_VIDINTCON0 |= (3 << 15) | (1 << 12) | 1;  VIC0INTENABLE\_REG |= (1 << 30);  VIC0VECTADDR30 = (unsigned)vsync\_interrupt\_service\_routine;  } |

lcd.c 는 기존에 랩 세션에서 제공되었던 기능을 거의 그대로 사용했다. 대신 lcd 자체의 vsync 인터럽트에 프레임 시작 함수를 넣어 화면을 수직동기화 시켰다. 또한 프레임 연산 자체를 인터럽트 컨텍스트에서 하면 다른 인터럽트가 영향을 받을 것이므로 실제 프레임 연산은 뒤에서 구현했다. 이는 이후에 더 자세히 설명하도록 하겠다.

#### graphics.c

|  |
| --- |
| #include "graphics.h"  #include "lcd.h"  #include "util.h"  #include "vic.h"  #include "s3c6410.h"  void gfx\_poll() {  while (GFXFIFOSTAT & 0x7E);  }  void gfx\_bitblck\_ext(unsigned \*dest, unsigned \*src,  int dx1, int dy1, int dx2, int dy2,  int destw, int desth,  int sx1, int sy1, int sx2, int sy2,  int srcw, int srch) {  int stretch = 0;  gfx\_poll();  GFXCOORD0 = sx1 | (sy1 << 16);  GFXCOORD1 = (sx2-1) | ((sy2-1) << 16);  GFXCOORD2 = dx1 | (dy1 << 16);  GFXCOORD3 = (dx2-1) | ((dy2-1) << 16);  if (dx2 - dx1 != sx2 - sx1) {  float f;  f = 2048.0 \* (sx2 - sx1) / (dx2 - dx1);  GFXXINC = (int)f;  stretch = 1;  }  else  GFXXINC = 1 << 11;  if (dy2 - dy1 != sy2 - sy1) {  float f;  f = 2048.0 \* (sy2 - sy1) / (dy2 - dy1);  GFXYINC = (int)f;  stretch = 1;  }  else  GFXYINC = 1 << 11;  GFXSRCBASE = src;  GFXDESTBASE = dest;  GFXCWLTX = 0;  GFXCWLTY = 0;  GFXCWRBX = destw;  GFXCWRBY = desth;  GFXRESH = destw;  GFXRESV = desth;  GFXSRCRESH = srcw;  GFXFGCOL = 0xFFFFFFFF;  GFXBGCOL = 0xFF0000FF;  GFXBSCOL = 0xFF0000FF;  GFXROP = (1 << 13) | (1 << 10) | (0xF0);  GFXSRCCOLMOD = 4;  GFXDESTCOLMOD = 4;  GFXCMD1 = stretch ? 2 : 1;  } |

Graphics2D란 s3c6410x가 제공하는 그래픽 가속 기능으로써, 이름에서 알 수 있듯 아주 빠른 속도로 그림을 그릴 수 있게 해준다. 먼저 parameter레지스터에 우리가 원하는 값을 모두 넣고 명령 레지스터에 값을 쓰면 parameter에 입력된대로 그래픽 명령이 실행된다. 위에서 구현된 부분은 bit block transfer라고 불리는 기능이다. 이 기능은 메모리 메모리의 한 영역을 참조하여 래스터 연산을 실행한 후 그 결과를 다른 영역에 덮어쓰는데, 이를 화면에 그림을 그리는 용도로 사용할 수 있다. 또한 명령이 실행되는 동안 parameter 레지스터에 있는 값이 영향을 받지 않게 하기 위해서 명령 FIFO가 모두 비워질 때 까지 기다리도록 구현하였다.

#### postproc.c

|  |
| --- |
| #include "postproc.h"  #include "lcd.h"  #include "s3c6410.h"  #include "vic.h"  #include "frame.h"  void ppc\_interrupt\_service\_routine(void) {  PPCMODE &= ~BIT6;  }  static int logi(unsigned x) {  int ret = 0;  x >>= 1;  while (x) {  x >>= 1;  ret++;  }  return ret;  }  static int maxi(unsigned x, unsigned y) {  return x > y ? x : y;  }  void ppc\_init() {  unsigned hclk = get\_hclk();  unsigned pres\_h\_ratio = maxi(SCREENW / S3CFB\_HRES, 1);  unsigned pres\_v\_ratio = maxi(SCREENH / S3CFB\_VRES, 1);  unsigned pres\_dest\_h = SCREENW / pres\_h\_ratio;  unsigned pres\_dest\_v = SCREENH / pres\_v\_ratio;  unsigned h\_shift = logi(pres\_h\_ratio);  unsigned v\_shift = logi(pres\_v\_ratio);  VIC0INTENABLE\_REG |= (1 << 9);  VIC0VECTADDR9 = (unsigned)ppc\_interrupt\_service\_routine;  PPCSRCIMGSIZE = (SCREENH << 12) | SCREENW;  PPCDESTIMGSIZE = (S3CFB\_VRES << 12) | S3CFB\_HRES;  PPCPRESCLRATIO = (pres\_v\_ratio << 7) | pres\_h\_ratio;  PPCPRESCLSIZE = (pres\_dest\_v << 12) | pres\_dest\_h;  PPCMSCLHRATIOH = (SCREENW << 8) / (S3CFB\_HRES << h\_shift);  PPCMSCLHRATIOV = (SCREENH << 8) / (S3CFB\_VRES << v\_shift);  PPCPRESCLSH = (10 - h\_shift - v\_shift);  PPCMODE = BIT18 | BIT7 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1;  PPCMODE2 = BIT4;  printf("Postprocessor initialized.\n");  }  void ppc\_set(unsigned \*src, unsigned \*dest) {  PPCADDRSY = (unsigned)src;  PPCADDRSRGB = (unsigned)dest;  PPCADDREY = (unsigned)src + (SCREENSIZE << 2);  PPCADDRERGB = (unsigned)dest + (S3CFB\_SIZE << 2);  PPCPOSTENVID = BIT31;  } |

Postprocessor란 이미지의 후처리 연산(화면에 맞게 해상도 조절, 컬러 스페이스 변환)에 쓰이는 기능이다. 망고보드의 성능 한계 상 800x480화면을 매번 새로 덮어서 새로 그리는 것이 힘들 수 있기 때문에 우리는 400x240로 스크린 해상도를 낮추고 그것을 프레임 버퍼에 2배 확대하여 붙여넣는 형식으로 성능 향상을 유도했다. 이때 프레임 버퍼에 실제로 2배 작은 스크린을 붙여넣을 때 쓰이는 것이 postprocessor이다. 포스트 프로세서에 해당하는 적절한 명령어를 넣고 enable 레지스터를 활성화시키면 입력된 클락에 맞춰 메모리 복사가 일어난다.

### 엔트리 포인트

#### main.c

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <s3c\_uart.h>  #include <lcd.h>  #include "graphics.h"  #include "vic.h"  #include "frame.h"  #include "enemy.h"  #include "stage1.h"  #include "stage2.h"  #include "stage3.h"  #include "timeline.h"  #include "effect.h"  #include "item.h"  static void init(void)  {  disable\_interrupts();  uart\_init(1, 115200);  lcd\_init();  gfx\_init();  ppc\_init();  mango\_interrupt\_init();  enable\_interrupts();  sprite\_resource\_init();  stg1\_init();  stg2\_init();  stg3\_init();  }  int main()  {  init();  while(1){  frame\_service();  }  return 0;  } |

main함수는 프로젝트 바이너리 자체의 엔트리 포인트이다. 여기서 기본적으로 frame\_service함수를 무한루프로 작동시키는데, 이에 대해서는 다음에서 자세히 알아보자.

#### frame.c

|  |
| --- |
| #include <stdio.h>  #include "frame.h"  #include "s3c6410.h"  #include "lcd.h"  #include "util.h"  #include "graphics.h"  #include "postproc.h"  #include "interrupt.h"  #include "timeline.h"  #include "cbox.h"  #include "bullet.h"  #include "interrupt.h"  #include "pad.h"  #include "player.h"  #include "enemy.h"  #include "smanage.h"  #include "effect.h"  #include "item.h"  #include "stage1.h"  int frame\_count = 0;  cbox cb\_screen = {(float)SCREENW, (float)SCREENH, 0.0, 0.0, (float)SCREENW, (float)SCREENH};  cbox cb\_scr\_pad = {(float)(SCREENW + SCREENPAD), (float)(SCREENH + SCREENPAD),  -(float)SCREENPAD, -(float)SCREENPAD,  (float)(SCREENW + SCREENPAD), (float)(SCREENH + SCREENPAD)};  cbox cb\_scr\_unpad = {(float)(SCREENW - SCREENUNPAD), (float)(SCREENH - SCREENUNPAD),  (float)SCREENUNPAD, (float)SCREENUNPAD,  (float)(SCREENW - SCREENUNPAD), (float)(SCREENH - SCREENUNPAD)};  unsigned \*scr\_target;  static int frame\_asserted = 0;  static unsigned screen[SCREENSIZE];  static unsigned fb\_odd[S3CFB\_SIZE];  static unsigned fb\_even[S3CFB\_SIZE];  static unsigned \*fb\_shown;  static unsigned \*fb\_working;  extern timeline tl\_stg1\_main;  static unsigned state = 0;  static unsigned deadcnt = 0;  unsigned endcount = 0;  extern sprite spr\_title;  extern sprite spr\_gameover;  extern sprite spr\_ending;  void (\*bg\_draw)();  int frame\_is\_asserted(void) {  return frame\_asserted;  }  void frame\_set\_fb(unsigned \*fb) {  unsigned ufb = (unsigned)fb;  S3C\_VIDW00ADD0B0 = ufb;  S3C\_VIDW00ADD1B0 = S3C\_VIDWxxADD1\_VBASEL\_F(ufb +  (PAGE\_WIDTH + S3CFB\_OFFSET) \* S3CFB\_VRES);  S3C\_VIDW00ADD2 = S3C\_VIDWxxADD2\_OFFSIZE\_F(S3CFB\_OFFSET) |  S3C\_VIDWxxADD2\_PAGEWIDTH\_F(PAGE\_WIDTH);  }  void frame\_init(void) {  frame\_set\_fb(fb\_even);  printf("Frame initialized.\n");  printf("Screen collision box is (%d, %d) (%d, %d).\n", (int)cb\_screen.x1, (int)cb\_screen.y1, (int)cb\_screen.x2, (int)cb\_screen.y2);  }  void frame\_assert(void) {  frame\_asserted = 1;  }  static void screen\_refresh();  void frame\_service(void) {  static int parity = 0;  if (frame\_asserted) {  fb\_shown = parity ? fb\_odd : fb\_even;  fb\_working = !parity ? fb\_odd : fb\_even;  parity = !parity;  frame\_set\_fb(fb\_shown);  scr\_target = screen;  if (state == 0) {  if (touch\_is\_valid()) {  bullet\_init();  player\_init();  enemy\_init();  effect\_init();  item\_init();  smanage\_set(&tl\_stg1\_main);  bg\_draw = stg1\_draw;  deadcnt = 0;  endcount = 0;  state = 1;  }  }  else if (state == 1) {  smanage\_update();  pad\_update();  player\_update();  enemy\_update();  item\_update();  bullet\_update\_master();  effect\_update();  if (player\_get\_life() == 0)  deadcnt++;  if (endcount > 0)  endcount++;  if (deadcnt >= TRANSITIONTIME)  state = 2;  if (endcount >= TRANSITIONTIME)  state = 3;  }  else if (state == 2) {  if (touch\_is\_valid()) {  state = 0;  }  }  else if (state == 3) {  if (touch\_is\_valid()) {  state = 0;  }  }  screen\_refresh();  gfx\_poll();  screen\_fetch();  touch\_clear();  frame\_count++;  frame\_asserted = 0;  }  }  void screen\_fetch(void) {  ppc\_set(screen, fb\_working);  }  static void screen\_refresh() {  if (state == 0) {  sprite\_draw\_clear(0xFFFFFFFF);  sprite\_draw(&spr\_title, 200.0, 120.0);  }  if (state == 1) {  bg\_draw();  smanage\_draw();  player\_draw();  enemy\_draw();  bullet\_draw\_master();  item\_draw();  effect\_draw();  player\_life\_draw();  pad\_draw();  if (player\_get\_life() == 0 || endcount > 0) {  unsigned alpha = (deadcnt + endcount) \* 255 / TRANSITIONTIME;  sprite\_draw\_clear(0xFFFFFF | (alpha << 24));  }  }  if (state == 2) {  sprite\_draw\_clear(0xFFFFFFFF);  sprite\_draw(&spr\_gameover, 200.0, 120.0);  }  if (state == 3) {  sprite\_draw\_clear(0xFFFFFFFF);  sprite\_draw(&spr\_ending, 200.0, 120.0);  }  } |

frame개념은 앞서 설명했던 lcd.c와 협업하여 작동한다. LCD가 vsync 인터럽트를 발생시키면 그때 main loop의 흐름이 frame의 엔트리 포인트로 바뀌면서 해당 프레임에 필요한 모든 연산을 행한다. 한 프레임은 시스템 업데이트와 이미지 프로세싱이라는 두개의 큰 부분으로 나뉘어있다. 1번 state가 바로 실제 게임을 진행하는 상태로, 각종 upate함수가 있는 것을 확인할 수 있다.

여기서 update가 끝나면 스크린을 refresh하는데 이 refresh하는 과정과 LCD 물리적 스크린에 그림을 복사하는 과정이 서로 overlap되면 화면에 가로 선이 보이거나 배경과 플레이어가 서로 깜빡이면서 보이는 등의 문제가 발생한다. 이를 막기 위해 두개의 프레임 버퍼를 놓고 한 쪽에 그림을 그리는 동안 다른 한 쪽을 LCD 스크린에 표시함으로써 이 현상을 막았다.

### 유틸리티 함수

#### util.c

util.c에는 각종 기본적인 수학적 연산이 정의되어있다. math.h를 include하는 것이 불가능했던 관계로 직접 Pade’s approximant를 이용해 sin함수를 정의했고, Quake사에서 사용했던 inverse square root함수를 가져와 사용했다. 음수 domain에서 sin함수가 제대로 동작하지 않는 문제가 개발 후반에 밝혀졌지만 실제로 프로젝트에 영향을 주지는 않았다.

### 게임 백엔드 시스템

#### sprite.c

|  |
| --- |
| typedef struct {  int ofx, ofy, w, h;  unsigned \*data;  } sprite; |
| #include <stdio.h>  #include "sprite.h"  #include "graphics.h"  #include "frame.h"  #include "../img/ganjome.h"  ...  #include "../img/ending.h"  extern unsigned \*scr\_target;  void sprite\_resource\_init() {  int i;  for (i = 0; i < 80\*80; i++) {  int alpha = (spr\_key.data[i] >> 24) & 0xFF;  alpha /= 3;  spr\_key.data[i] = (spr\_key.data[i] & 0xFFFFFF) | (alpha << 24);  }  for (i = 0; i < 120; i++) {  spr\_pixelsky.data[i] = spr\_pixelsky.data[239];  }  spr\_key.ofx = 0;  spr\_key.ofy = 0;  spr\_dgt.ofx = 0;  spr\_dgt.ofy = 0;  }  void sprite\_draw(sprite \*spr, float \_\_x, float \_\_y) {  int x, y;  int dx1, dy1, dx2, dy2;  int cx1, cy1, cx2, cy2;  x = (int)\_\_x - spr->ofx;  y = (int)\_\_y - spr->ofy;  dx1 = x;  dy1 = y;  dx2 = x + spr->w;  dy2 = y + spr->h;  if (dx1 >= SCREENW || dx2 <= 0 || dy1 >= SCREENH || dy2 <= 0)  return;  cx1 = 0;  cy1 = 0;  cx2 = 0;  cy2 = 0;  if (dx1 < 0) {  cx1 = -dx1;  dx1 = 0;  }  if (dy1 < 0) {  cy1 = -dy1;  dy1 = 0;  }  if (dx2 >= SCREENW) {  cx2 = dx2 - SCREENW;  dx2 = SCREENW;  }  if (dy2 >= SCREENH) {  cy2 = dy2 - SCREENH;  dy2 = SCREENH;  }  gfx\_bitblck\_ext(scr\_target, spr->data,  dx1, dy1, dx2, dy2,  SCREENW, SCREENH,  cx1, cy1, spr->w - cx2, spr->h - cy2,  spr->w, spr->h);  }  void sprite\_draw\_clear(unsigned wrapper) {  gfx\_bitblck\_ext(scr\_target, &wrapper,  0, 0, SCREENW, SCREENH,  SCREENW, SCREENH,  0, 0, 1, 1,  1, 1);  } |

본 프로젝트에서는 이미지 정보를 배열로 관리한다. 컴파일 타임에 모든 이미지 정보가 바이너리에 저장되는데, sprite개념은 이 데이터를 효율적으로 관리하기위해 만들어졌다. Sprite는 구조체로써 내부에 이미지 오프셋과 이미지 크기, 그리고 이미지 정보 어레이를 가지고 잇다. 나중에 각 게임 객체는 이 sprite포인터를 들고있다가 그림을 그릴 타이밍에 sprite\_draw함수 인자로 넘겨주면 된다.

#### cbox.c

|  |
| --- |
| typedef struct {  float w, h;  float x1, y1, x2, y2;  } cbox; |
| #include "cbox.h"  inline int cbox\_inclusion(cbox \*cb, float x, float y) {  return (cb->x1 <= x && cb->x2 > x) && (cb->y1 <= y && cb->y2 > y);  }  void cbox\_refresh(cbox \*cb, float x, float y) {  cb->x1 = x;  cb->y1 = y;  cb->x2 = x + cb->w;  cb->y2 = y + cb->h;  }  void cbox\_refresh\_centered(cbox \*cb, float x, float y) {  cb->x1 = x - cb->w\*0.5;  cb->y1 = y - cb->h\*0.5;  cb->x2 = x + cb->w\*0.5;  cb->y2 = y + cb->h\*0.5;  }  int cbox\_collide(cbox \*c1, cbox \*c2) {  return c1->x1 <= c2->x2 && c1->x2 >= c2->x1 &&  c1->y1 <= c2->y2 && c1->y2 >= c2->y1;  } |

cbox는 충돌 판정과 관련된 일을 담당하는 구조체이다. Cbox는 자기 자신 안에 어떤 점이 있는지 검사할 수 있으며 또한 cbox서로 끼리의 충돌도 검사할 수 있다. 이는 이후 플레이어와 적 탄알, 혹은 플레이어 탄알과 적 기체의 충돌 판정 등에 사용된다.

#### timeline.c

|  |
| --- |
| typedef struct {  void (\*func)(void\*, void\*);  void \*aux;  unsigned frame;  } action;  typedef struct {  action\* act;  unsigned size;  } timeline;  typedef struct {  timeline \*tl;  unsigned seek;  unsigned fcnt;  } watch; |
| #include "timeline.h"  #include "enemy.h"  #include "player.h"  #include "util.h"  action action\_pool[ACTION\_MAX];  static int pool\_seek = 0;  action \*action\_alloc(int size) {  int idx = pool\_seek;  pool\_seek += size;  if (pool\_seek >= ACTION\_MAX)  printf("\*\*\*\*\*\*\*\*\nError::Action pool size exceeded!!\n\*\*\*\*\*\*\*\*\n");  return &action\_pool[idx];  }  void watch\_update(watch \*w, void \*self) {  timeline \*tl = w->tl;  action \*act = tl->act;  while (1) {  if (tl->size <= w->seek)  break;  else if (act[w->seek].frame != w->fcnt)  break;  else {  unsigned idx = w->seek++;  act[idx].func(self, act[idx].aux);  }  }  w->fcnt++;  }  void watch\_set\_frame(watch \*w, unsigned fcnt) {  timeline \*tl = w->tl;  action \*act = tl->act;  w->fcnt = fcnt;  w->seek = 0;  while (act[w->seek].frame < w->fcnt) {  w->seek++;  }  } |

timeline은 예정된 행위를 관리하기 위해 만들어진 개념이다. 이 개념은 세가지 구조체로 이루어져있다 먼저 이 timeline의 원소에 해당하는 action. Action은 프레임 수와 function pointer의 쌍으로, 어떤 객체가 프레임 수를 센지 어느 정도 시간이 지나면 예정된 function pointer가 작동하는 식이다. Action의 function은 실행의 주체 자기 자신을 가리키는 void 포인터와 기타 인자를 넣을 수 있는 또 다른 void 포인터 두개를 인자로 받는다. 다음 Timeline은 action의 집합으로, action이 시간 순서대로 정렬된 array와 그 크기를 가지고있는 wrapper이다. 마지막으로 watch는 이 timeline개념을 동작시키는 부분으로, watch는 자기 자신이 들고있는 timeline구조체를 보면서 자신이 세고있는 카운터가 일정 시간에 도달하면 timeline에 예정된 action을 실행시킨다. 이후에 설명될 적군과 스테이지는 이 timeline개념에 기반하여 작동한다.

### 게임 미들엔드 시스템

#### Pool

소스코드를 설명하기에 앞서 pool을 먼저 짚고 넘어가자. 본 프로젝트에는 동적 할당이 전혀 사용되지 않는다. 모든 정보는 compile time에 정해진 공간에 모두 들어가며, 이를 관리하기 위해 pool이라는 개념을 도입했다. Pool은 기본적으로 메모리상에 정적으로 만들어진 array이다. 만약 사용자가 할당을 요청하면 pool은 array에서 아직 쓰이지 않은 원소를 꺼내어 제공한다. 그러므로 complie time에서 정한 양 보다 많은 원소를 쓸 수 없지만 시스템을 안정적으로 만들 수 있다. 게임 시스템에 쓰이는 객체들은 대부분 pool로 만들어져있다.

#### bullet.c

|  |
| --- |
| typedef struct {  sprite\* spr;  cbox cb;  float x, y, vx, vy, ax, ay;  int valid;  int power;  int ef;  } bullet;  typedef struct {  bullet\* b\_array;  unsigned count;  unsigned size;  unsigned seek;  } bulletpool; |
| #include <stdio.h>  #include "bullet.h"  #include "cbox.h"  #include "sprite.h"  #include "player.h"  #include "util.h"  #include "effect.h"  bullet enemy\_bullet\_array[EBULLET\_MAX];  bullet player\_bullet\_array[PBULLET\_MAX];  bulletpool ebullet\_pool = {enemy\_bullet\_array, 0, EBULLET\_MAX};  bulletpool pbullet\_pool = {player\_bullet\_array, 0, PBULLET\_MAX};  extern cbox cb\_screen;  void bullet\_init() {  int i;  for (i = 0; i < ebullet\_pool.size; i++)  ebullet\_pool.b\_array[i].valid = 0;  for (i = 0; i < pbullet\_pool.size; i++)  pbullet\_pool.b\_array[i].valid = 0;  }  bullet\* bullet\_find\_empty(bulletpool \*bp) {  int i;  for (i = 0; i < bp->size; i++) {  if (!bp->b\_array[bp->seek].valid)  return &bp->b\_array[bp->seek];  bp->seek++;  bp->seek %= bp->size;  }  return NULL;  }  bullet\* bullet\_create\_enemy(sprite \*spr,  float x, float y, float vx, float vy, float w, float h, int power) {  return bullet\_create(&ebullet\_pool, spr, x, y, vx, vy, w, h, power);  }  bullet\* bullet\_create\_player(sprite \*spr,  float x, float y, float vx, float vy, float w, float h, int power) {  return bullet\_create(&pbullet\_pool, spr, x, y, vx, vy, w, h, power);  }  bullet\* bullet\_create(bulletpool \*bp, sprite \*spr,  float x, float y, float vx, float vy, float w, float h, int power) {  bullet \*b = bullet\_find\_empty(bp);  if (b == NULL) {  printf("No bullet pool left!!\n");  return NULL;  }  b->spr = spr;  b->ax = 0.0;  b->ay = 0.0;  b->x = x;  b->y = y;  b->vx = vx;  b->vy = vy;  b->ef = 0;  b->cb.w = w;  b->cb.h = h;  cbox\_refresh\_centered(&b->cb, x, y);  b->power = power;  b->valid = 1;  bp->count++;  return b;  }  void bullet\_update(bulletpool \*bp) {  int i;  for (i = 0; i < bp->size; i++) {  bullet \*b = &bp->b\_array[i];  if (b->valid) {  b->x += b->vx;  b->y += b->vy;  b->vx += b->ax;  b->vy += b->ay;  cbox\_refresh\_centered(&b->cb, b->x, b->y);  if (!cbox\_inclusion(&cb\_screen, b->x, b->y)) {  bullet\_delete(bp, i);  }  }  }  }  void bullet\_update\_master() {  bullet\_update(&pbullet\_pool);  bullet\_update(&ebullet\_pool);  }  void bullet\_delete(bulletpool \*bp, int i) {  bullet \*b = &bp->b\_array[i];  if (b->ef) {  effect\_small(b->x, b->y);  }  b->ef = 0;  b->valid = 0;  b->ax = 0.0;  b->ay = 0.0;  bp->count--;  }  void bullet\_draw(bulletpool \*bp) {  int i;  for (i = 0; i < bp->size; i++) {  bullet \*b = &bp->b\_array[i];  if (b->valid) {  sprite\_draw(b->spr, b->x, b->y);  }  }  }  void bullet\_draw\_master() {  bullet\_draw(&pbullet\_pool);  bullet\_draw(&ebullet\_pool);  }  void bullet\_towards\_player(bullet \*b, float v, float dir) {  float dx, dy, d, dx2, dy2;  dx = player\_get\_x() - b->x;  dy = player\_get\_y() - b->y;  d = v \* Q\_rsqrt(dx\*dx + dy\*dy);  dx2 = dx \* cos(dir) - dy \* sin(dir);  dy2 = dy \* cos(dir) + dx \* sin(dir);  b->vx = dx2 \* d;  b->vy = dy2 \* d;  } |

bullet에는 플레이어와 적군 편의 두가지 pool이 존재한다. 두 pool이 가지는 내용물 구조는 완전히 같다. 소스에서 확일할 수 있듯, pool에서 빈 엔트리를 찾고, 엔트리를 해제하고, 현재 할당되어있는 엔트리를 모두 한번에 그릴 수 있는 함수 등이 준비되어있다. 특히 \*\_update와 \*\_draw류 함수들은 frame.c에서 매 프레임마다 호출된다. 이후 설명될 다른 개념들도 bullet과 거의 같은 방식으로 작동한다.

#### enemy.c

|  |
| --- |
| typedef struct {  cbox cb;  watch w;  sprite \*spr;  void (\*dest)(void \*);  float x, y, vx, vy, ax, ay;  int health;  int valid;  int destroy;  int ef;  int item\_prob;  } enemy; |
| void enemy\_update() {  int i;  for (i = 0; i < ENEMY\_MAX; i++) {  if (enemy\_pool[i].valid) {  enemy \*e = &enemy\_pool[i];  int j;  watch\_update(&e->w, (void \*)e);  e->vx += e->ax;  e->vy += e->ay;  e->x += e->vx;  e->y += e->vy;  cbox\_refresh\_centered(&e->cb, e->x, e->y);  if (e->destroy || !cbox\_inclusion(&cb\_scr\_pad, e->x, e->y)) {  enemy\_delete(i);  continue;  }  for (j = 0; j < pbullet\_pool.size; j++) {  bullet \*b = &pbullet\_pool.b\_array[j];  if (b->valid) {  if (cbox\_collide(&e->cb, &b->cb)) {  bullet\_delete(&pbullet\_pool, j);  if (enemy\_damage(i, b->power))  break;  }  }  }  }  }  } |

enemy.c는 적군에 대한 정보를 담고 있는 소스코드이다. 특히 enemy는 watch구조를 가지고 있는데, 이는 적군 객체 하나마다 자신의 시간과 일정을 따로 가지고 있음을 의미한다. enemy\_update에서 이 watch를 업데이트함으로써 각 적군 개체마다 자신에게 예정된 일을 실행할 수 있게 하였다. 또한 플레이어의 bullet pool에 할당된 모든 entry와 자기 자신의 cbox를 연산하여 대미지 계산을 행한다.

#### effect.c

|  |
| --- |
| typedef struct {  sprite \*\*spr;  unsigned size;  } animation;  typedef struct {  animation \*ani;  int frame;  float x, y;  } effect; |
| extern sprite spr\_bomb1;  extern sprite spr\_bomb2;  extern sprite spr\_bomb3;  extern sprite spr\_sbamb1;  extern sprite spr\_sbamb2;  extern sprite spr\_sbamb3;  sprite \*ani\_large\_seqeunce[3] = {&spr\_bomb1, &spr\_bomb2, &spr\_bomb3};  sprite \*ani\_small\_seqeunce[3] = {&spr\_sbamb1, &spr\_sbamb2, &spr\_sbamb3};  animation ani\_large = {ani\_large\_seqeunce, 3};  animation ani\_small = {ani\_small\_seqeunce, 3}; |

effect는 플레이어의 탄알이 적군에게 명중했을 때, 적군이 플레이어에 의해 파괴될 때, 플레이어가 피탄당해 화면의 적군 탄알이 사라질 때 나타난다. 각 개체들은 적당한 순간과 위치에 effect\_create를 호출함으로써 효과를 나타낸다. effect는 animation포인터와 현재 재생되고있는 frame으로 이루어져있다. animation구조는 sprite어레이와 그 어레이의 크기로 이루어져있다. effect는 create된 순간부터 재생되다가 어레이의 크기를 넘어가면 자동으로 해제된다.

#### item.c

|  |
| --- |
| #define ITEM\_SOY 1  #define ITEM\_HA 2  #define ITEM\_FLOWER 3  #define ITEM\_LIFE 4  #include "cbox.h"  typedef struct {  int type;  int span;  cbox cb;  float x, y, vx, vy;  } item; |

item.c는 아이템을 관리하는 역할을 하지만 그 아이템에 의한 실제 작용은 모두 player측에서 관리한다. item.c는 아이템의 존재와 타입만을 관리한다.

### 게임 프론트엔드 시스템

#### pad.c

|  |
| --- |
| static int pvalid = 0;  static float ppadx = 0.0;  static float ppady = 0.0;  static float padvx = 0.0;  static float padvy = 0.0;  extern unsigned \*cb\_screen;  extern sprite spr\_key;  void pad\_update() {  if (touch\_is\_valid()) {  float x, y;  x = touch\_get\_x();  y = touch\_get\_y();  if (!pvalid && !cbox\_inclusion(&cb\_pad, x, y)) {  return;  }  x = fclamp(x, cb\_pad.x1, cb\_pad.x2);  y = fclamp(y, cb\_pad.y1, cb\_pad.y2);  if (!pvalid) {  ppadx = x;  ppady = y;  pvalid = 1;  }  ppadx = flerp(ppadx, x, 0.5);  ppady = flerp(ppady, y, 0.5);  padvx = (ppadx - PADX) / PADW \* (2.0 + 2.0 \* PADTHRESH) - (1.0 + PADTHRESH);  padvy = (ppady - PADY) / PADH \* (2.0 + 2.0 \* PADTHRESH) - (1.0 + PADTHRESH);  if (padvx > PADTHRESH)  padvx -= PADTHRESH;  else if (padvx > -PADTHRESH)  padvx = 0.0;  else  padvx += PADTHRESH;  if (padvy > PADTHRESH)  padvy -= PADTHRESH;  else if (padvy > -PADTHRESH)  padvy = 0.0;  else  padvy += PADTHRESH;  padvx \*= PADVMAX;  padvy \*= PADVMAX;  }  else {  pvalid = 0;  padvx = 0.0;  padvy = 0.0;  }  } |

pad는 화면상에 플레이어를 조작할 수 있게 해주는 아날로그 패드와 관련된 처리를 한다. pad는 현재 터치가 행해지고 있는지 알아내어 터치가 패드 영역 위라면 패드 속도 출력을 알맞게 준다. 패드 영역 위에서 드래그 하다가 밖으로 드래그 하면 여전히 드래그가 되지만 밖에서 시작하면 패드의 속도에 영향을 줄 수 없다. 또한 pad는 이 터치 좌표를 그대로 사용하지 않고 이전 좌표와 내분하여 노이즈 성분을 어느 정도 제거한다.

#### player.c

|  |
| --- |
| static float x;  static float y;  static int cooldown = 0;  static int life = 5;  static int type = ITEM\_SOY;  static int invinc = 60;  static int over = 0;  static cbox cb = {10.0, 10.0};  ...  void player\_update() {  int i;  if (over)  return;  x += pad\_get\_vx();  y += pad\_get\_vy();  x = fclamp(x, (float)(PLAYERPADDING), (float)(SCREENW - PLAYERPADDING));  y = fclamp(y, (float)(PLAYERPADDING), (float)(SCREENH - PLAYERPADDING));  cbox\_refresh\_centered(&cb, x, y);  if (cooldown > 0)  cooldown--;  if (touch\_is\_valid() && cooldown <= 0) {  switch (type) {  case ITEM\_SOY: shoot\_bbrown(); break;  case ITEM\_HA: shoot\_hi(); break;  case ITEM\_FLOWER: shoot\_bflower(); break;  }  }  for (i = 0; i < ITEM\_MAX; i++) {  item \*it = &item\_pool[i];  if (it->type != -1) {  if (cbox\_collide(&cb, &it->cb)) {  switch (it->type) {  case ITEM\_SOY:  case ITEM\_HA:  case ITEM\_FLOWER:  type = it->type;  break;  case ITEM\_LIFE:  life++;  break;  }  effect\_large(x, y);  item\_delete(i);  }  }  }  if (invinc <= 0) {  for (i = 0; i < ebullet\_pool.size; i++) {  bullet \*b = &ebullet\_pool.b\_array[i];  if (b->valid) {  if (cbox\_collide(&cb, &b->cb)) {  life--;  invinc = 60;  printf("Collide with bullet\n");  break;  }  }  }  if (invinc <= 0) {  for (i = 0; i < ENEMY\_MAX; i++) {  enemy \*e = &enemy\_pool[i];  if (e->valid) {  if (cbox\_collide(&cb, &e->cb)) {  life--;  invinc = 60;  printf("Collide with enemy\n");  break;  }  }  }  }  if (invinc > 0) {  for (i = 0; i < ebullet\_pool.size; i++) {  bullet \*b = &ebullet\_pool.b\_array[i];  if (b->valid) {  b->ef = 1;  bullet\_delete(&ebullet\_pool, i);  }  }  for (i = 0; i < ENEMY\_MAX; i++) {  enemy \*e = &enemy\_pool[i];  if (e->valid) {  e->ef = 1;  enemy\_damage(i, 10);  }  }  if (life <= 0) {  effect\_large(x, y);  over = 1;  }  }  }  else {  invinc--;  }  } |

player.c는 화면상에 있는 플레이어와 관련된 처리를 모두 한다. 플레이어는 좌표, 무적시간, 공격 타입, 공격 쿨다운, 생명력, 게임오버 여부 등의 정보를 가지고 있다. player\_update함수는 pad에서 받아온 속도 정보를 가지고 플레이어의 위치를 갱신하고 아이템, 적군, 탄알등의 다른 개체와 충돌을 처리한다.

### 게임 진행 관리

#### action.c

action.c에는 이후 적군의 행동 패턴 설정에 있어 자주 쓰일 수 있는 간단한 행동 함수들이 정의되어있다. 예를 들면 가로로 이동, 세로로 이동, 방향 바꾸기, 멈추기, 플레이어를 향해 움직이기, 자기 자신 제거하기, watch 되감기 등이 있다.

#### smanage.c

|  |
| --- |
| void smanage\_update() {  if (next\_phase > 0) {  next\_phase += SCREENW / 30;  if (bg\_draw\_next != NULL && next\_phase > SCREENW) {  bg\_draw = bg\_draw\_next;  bg\_draw\_next = NULL;  }  if (next\_phase > SCREENW \* 2) {  main\_watch.seek = 0;  main\_watch.fcnt = 0;  next\_phase = 0;  }  }  else {  watch\_update(&main\_watch, &main\_watch);  }  }  void smanage\_next(timeline \*tl, void (\*next)()) {  main\_watch.tl = tl;  next\_phase = 1;  bg\_draw\_next = next;  }  void smanage\_draw() {  unsigned wrapper = 0xFF202020;  if (next\_phase == 0)  return;  if (next\_phase < SCREENW) {  gfx\_bitblck\_ext(scr\_target, &wrapper,  SCREENW - next\_phase, 0, SCREENW, SCREENH,  SCREENW, SCREENH,  0, 0, 1, 1,  1, 1);  }  else {  gfx\_bitblck\_ext(scr\_target, &wrapper,  0, 0, 2 \* SCREENW - next\_phase, SCREENH,  SCREENW, SCREENH,  0, 0, 1, 1,  1, 1);  }  } |

smanage.c는 게임 전체 스테이지 진행을 관리하는 소스이다. 이 소스는 글로벌 스코프의 watch를 하나 가지고 프레임마다 업데이트 하면서 정해진 위치에 적군을 생성하는 등의 예정된 일을 실행한다. 또한 스테이지가 넘어갈 때가 됐을 때 다음 스테이지에 대한 정보를 가지고 있다가 화면을 덮은 후 스테이지를 넘기고나서 다시 watch를 초기화해 스케줄을 실행한다.

#### stage1.c

#### stage2.c

#### stage3.c

이들은 모두 스테이지 진행에 관한 타임라인 정보, 그리고 그 스테이지의 배경을 어떻게 그려야 하는지에 대한 정보를 담고있다. 여기서 정의된 타임라인은 나중에 smanage에 있던 main\_watch에 의해 reference되면서 실제 스케줄로 실행된다. 모든 타임라인은 여태까지 있던 인터페이스를 망라하여 하드코딩으로 구현되었다. 다음 스테이지로 넘어가는 로직은 enemy의 dest멤버에 의해 실행된다. 해당 스테이지의 보스의 dest멤버에 적절한 다음 스테이지 명령 함수를 넣어놓는 것으로 다음 스테이지가 감지된다. 이 다음스테이지 명령 함수는 smanage의 함수를 호출하여 스테이지를 전환한다. 다음은 stage1 소스코드의 일부이다.

|  |
| --- |
| #include <stdio.h>  #include "stage1.h"  #include "stage2.h"  #include "timeline.h"  #include "action.h"  #include "enemy.h"  #include "util.h"  #include "frame.h"  #include "player.h"  #include "bullet.h"  #include "smanage.h"  #include "item.h"  timeline tl\_stg1\_main;  extern timeline tl\_stg2\_main;  ...  extern void (\*bg\_draw)();  extern unsigned \*scr\_target;  static void to\_stage2(void \*self) {  smanage\_next(&tl\_stg2\_main, stg2\_draw);  item\_create(ITEM\_LIFE, 200.0, 120.0);  }  static void create\_soonpink1(void \*self, void \*aux) {  if ((int)aux == 0) {  enemy\_create(&spr\_soonpink, &soonpink1, NULL,  420.0, 60.0, ENEMY\_W\_SMALL, ENEMY\_H\_SMALL, ENEMY\_HEALTH\_SMALL);  }  else {  enemy\_create(&spr\_soonpink, &soonpink2, NULL,  420.0, 180.0, ENEMY\_W\_SMALL, ENEMY\_H\_SMALL, ENEMY\_HEALTH\_SMALL);  }  }  ...  static void create\_partner(void \*self, void \*aux) {  enemy\_create(&spr\_partner, &partner, to\_stage2,  440.0, 120.0, 73.0, 96.0, 500);  }  static void partner1(void \*self, void \*aux) {  enemy \* e = enemy\_create(&spr\_ddabong, &ddabong, NULL,  420.0, (float)(int)aux, 22, 34, 100);  e->ax = -0.2;  e->vx = -1.0;  }  static void ddabong1(void \*self, void \*aux) {  static float phase = 0.0;  enemy \*e = (enemy \*)self;  float f;  int parity = 0;  phase += 0.5;  for (f = 0.0; f < CONST\_PIDOUBLE; f += CONST\_PI / 6.0) {  sprite \*spr = parity ? &spr\_byellow : &spr\_bpink;  bullet \*b;  parity = !parity;  b = bullet\_create\_enemy(spr, e->x, e->y, 0.0, 0.0, 10, 10, 0);  b->ax = 0.15 \* cos(f+phase);  b->ay = 0.2 \* sin(f+phase);  }  }  ...  void stg1\_init() {  action \*act;  int i = 0;  int frame = 0;  bg\_draw = stg1\_draw;  tl\_stg1\_main.size = 60;  tl\_stg1\_main.act = action\_alloc(tl\_stg1\_main.size);  act = tl\_stg1\_main.act;  frame = 10; action\_set(&act[i++], create\_soonpink1, (void \*)0, frame);  frame += 10; action\_set(&act[i++], create\_soonpink1, (void \*)0, frame);  frame += 10; action\_set(&act[i++], create\_soonpink1, (void \*)0, frame);  frame += 10; action\_set(&act[i++], create\_soonpink1, (void \*)0, frame);  frame += 10; action\_set(&act[i++], create\_soonpink1, (void \*)0, frame);  ...  frame += 30; action\_set(&act[i++], create\_partner, NULL, frame);  //soonpink1  soonpink1.size = 4;  soonpink1.act = action\_alloc(soonpink1.size);  act = soonpink1.act;  i = 0;  frame = 0; action\_set(&act[i++], action\_enemy\_horizontal, FLOATNEG4, frame);  frame += 40; action\_set(&act[i++], action\_enemy\_change\_direction, FLOAT45RAD, frame);  action\_set(&act[i++], bpink\_towards, NULL, frame);  frame += 20; action\_set(&act[i++], action\_enemy\_horizontal, FLOATNEG4, frame);  ...  //partner  partner.size = 7;  partner.act = action\_alloc(partner.size);  act = partner.act;  i = 0;  frame = 0; action\_set(&act[i++], action\_enemy\_horizontal, FLOATNEG2, frame);  frame += 60; action\_set(&act[i++], action\_enemy\_stop, NULL, frame);  //frame += 60; action\_set(&act[i++], action\_enemy\_vertical, FLOATNEG1, frame);  frame += 40; action\_set(&act[i++], partner1, (void \*)60, frame);  frame += 40; action\_set(&act[i++], partner1, (void \*)180, frame);  frame += 80; action\_set(&act[i++], partner1, (void \*)200, frame);  frame += 40; action\_set(&act[i++], partner1, (void \*)40, frame);  frame += 40; action\_set(&act[i++], action\_enemy\_rewind, (void \*)60, frame);  //ddabong  ddabong.size = 3;  ddabong.act = action\_alloc(ddabong.size);  act = ddabong.act;  i = 0;  frame = 0; action\_set(&act[i++], action\_enemy\_horizontal, FLOATNEG2, frame);  frame += 20; action\_set(&act[i++], ddabong1, NULL, frame);  frame += 20; action\_set(&act[i++], action\_enemy\_rewind, (void \*)15, frame);  }  void stg1\_draw() {  unsigned \*src = spr\_pixelsky.data;  gfx\_bitblck\_ext(scr\_target, src,  0, 0, SCREENW, SCREENH,  SCREENW, SCREENH,  0, 0, 1, SCREENH,  1, SCREENH);  } |

# 프로그램 실행 방법 설명

make command를 통해서 cross – compile을 시키면 project라는 ARM binary가 만들어진다. 이 binary를 LAN선을 이용하든, UART를 이용하든 (본 프로젝트를 제작할 당시에는 그림이 많은 것을 기반으로 LAN선과 tftp 서버를 통한 빠른 전송을 하였다.) 망고 보드에 전송시킨다. 전송이 된 후에 bootelf를 입력하면 게임이 실행된다.

# 토론 및 개선 사항

우선 무엇인가를 개발할 때에 가장 기본적으로 해야하는 것은 철저한 저장과 철저한 버전관리라는 것을 다시 상기시켜주었던 프로젝트였다. 최유정 학생의 경우에도 밤새워 작성했던 그림이 컴퓨터가 꺼지면서 파일이 손상되어서 다시 하루 종일 그것을 복구시켜야 했고, 황일환 학생의 경우에도 작성했던 Grapics 2D chip을 활용하는 파일에 대한 버전관리가 되지 않아, 실행이 전 날 작성했던 것과 다른 일이 생겨서 그 이후 둘 모두 버전관리를 아주 철저하게 하게 되었다.

그리고 UART를 통해서 전송하다가 안석준군의 도움으로 ubuntu에서 tftp server와 옆 컴퓨터의 LAN선을 활용하여 아주 빠른 속도로 program을 전송할 수 있게 되어서 개발 속도에 혁신이 있을 수 있었다.

또한 bmp 파일을 HEX로 바꾸는 작업 또한 정유철 학생이 작성한 python script의 활용을 통해서 빠른 속도로 그림을 저장할 수 있었다. 특히 이 그림을 후 처리하는 것 또한 더 빨리 하기 위해서 python script에 조금 수정을 가해서 작성된 .h 파일이 바로 include 될 수 있는 형태로 만들었고, 따라서 그림을 저장한 후 몇 가지 command만을 이용해 사용하기 쉽도록 가공된 이미지를 얻을 수 있었다.

사실 게임의 본 내용을 만들기 시작한 것은 일요일부터였으며, 모든 랩 시간과 프로젝트를 시작한 약 일주일 정도에서 가장 중요하고 많은 시간을 걸렸던 것은 실시간 게임을 만들기 위한 하드웨어적 기반과 게임에 대한 아이디어였다. 게임을 잘 알지 못하는 최유정 학생은 슈팅게임이라는 컨셉을 이해하기 위해서 황일환 학생의 도움을 받아 (잘하지는 못하지만) 슈팅게임을 이래저래 시도해 보면서 슈팅게임이라는 것에 대해서 이해를 할 수 있도록 했고, 마침내 게임에 대한 전반적인 시나리오와 게임 내용을 만들어 낼 수 있었다. 사실 이렇게 게임을 확정할 수 있었던 것은 하드웨어적인 문제가 모두 해결이 되었기 때문이다. 아무리 슈팅게임을 만들고자 하는 포부가 있다고 하더라도 실시간 게임을 실행시킬 수 있는 하드웨어적 기반이 튼튼하다는 확신이 없이는 어떤 게임도 만들어 낼 수가 없다.

Manual에 대한 깊은 고찰과 스스로에 대한 반성과 고찰을 통해서 Grapics2D chip의 모든 레지스터와 작동방법을 알아낸 후에 이 chip을 작동시킬 수 있었다. 그러나 chip이 작동한다고 모든 것이 해결되는 것이 아니었다. 색을 출력하는 것도 아주 많은 시도와 고찰을 통해 뇌를 갈고 닦으면서 chip을 작동시켰으며, 기어코 아름다운 그림을 만들어 냈다. 그리고 매뉴얼을 보다가 3D chip이 있다는 것을 알게 되었고, 혹시 아주 현실적인 그림이 보고싶다면, 이 chip을 사용할 수도 있을 것만 같았다.

처음에는 sound를 넣고 싶은 마음에 audio 에 대해서 계속적으로 알아 보았으나, 우선 실습실에있는 책자에서는 구체적인 정보를 주지않았으며 audio codec 과 관련된 manual에는 사용하는 방법에 대한 구체적인 정보는 들어있지 않았다. 또한 망고보드 부분에 대한 설명이 있을 뿐 chip이 대한 data sheet는 따로 찾아보아야 했고, 인터넷에 audio와 관련해서는 linux 기반에서 작동하는 pcm 과 관련된 file을 딱 하나 찾을 수 있었기에 audio 출력을 얻는 것은 어려운 일이라는 것을 알게 되었으며, 특히나 Grapics 2D chip을 동작시켜본 결과 audio를 작동시키는 것은 아주 어렵다는 것을 알게 되었기 때문에 audio 부분을 활용하는 것을 포기하게 되었다.

그 외 게임의 내부적인 구현은 hard coding 적인 부분과 게임을 설계한 대로 구현해 내는 일이었다. 게임의 내용 부분을 구현 할 때에 가장 중요한 알고리즘은 collision을 check하는 것이었는데, 초기에 작성해서 두 사람 모두 괜찮다고 생각했던 collision check가 올바르게 동작하지 않는 예외적인 경우가 있다는 것을 알게 되어서, 여러 번의 시도 끝에, 현재와 같이, 4개의 조건이 and를 통해서 판별이 되는 collision check 알고리즘을 얻게 되었다.

게임에 그림을 하나씩 추가하다보니 어느 순간 배경에 해당하는 것을 통째로 올리려고 하다보니 bootelf를 하면 망고보드가 작동하지 않는 것을 알게 되었는데, 이가 .data section의 용량이 부족해서라는 것을 알게 되었고, 이를 make file의 .Text의 주소를 수정함으로써 고쳐질 수 있다는 것을 알게되었으나, 이렇게 주소를 옮기는 것이 혹시나 충돌을 만들지도 모른다는 생각과, 작은 그림을 놓고 그것을 반복시키는 연산등을 통해 배경을 나타냄으로써 보다 가벼운 프로그램을 만들 수 있었기에 Text의 주소를 옮기지 않고 진행했다. 혹시 더 그래픽적으로 풍부한 게임을 만들고 싶다면 이 주소를 수정시키는 방법이 필요할 것이다.

# 결론

아름다운 게임을 보는 것과, 느끼는 것과, 만드는 일은 정말 훌륭한 일이다. 평소에 상시 게임을 해서 칼을 갈고 닦아서 이렇게 프로젝트를 할 때 그런 능력들을 잘 활용하도록 해야겠다. 게임의 플레이를 확인하고 싶다면 황일환 군의 face book을 참고해주길 바란다.

# 느낀점

최유정(컴퓨터 공학과, 20140658) - 010-6243-0461

yjchoi0606@postech.ac.kr

프로젝트는 집에 얼른 갈 수 없다는 단 하나의 단점을 제외한다면 학기 중에 참 기다려지는 이벤트 중 하나인 것 같다. 한 학기 동안 배운 모든 지식들을 쏟아서 무엇인가를 집중해서 하나의 재미있는 것으로 만들어 내는 것은 정말 재미있는 일인 것 같다. 특히 몇 날 몇일 밤을 새면서 친구들이랑 놀면서 만드는 프로젝트는 재미있는 것 같다. 이번 프로젝트를 만들기 위해서 슈팅게임을 새로 다운 받아서 해봤는데, 게임을 안한지가 오래되었는데, 게임을 해 보니 정말 재미있었다. (물론 잘 하지 못해서 많이 슬프다.) 일환이가 추천해준 게임을 하는데, 세상 모든 것이 그렇겠지만 주의 깊게 안보면 잘모르는 것들이 자세히 보면 보인다. 게임을 만들기 위해서 게임의 시나리오, 게임의 캐릭터, 그림 이미지, 게임의 내용을 자세히 관찰하면서 이것을 구현하기 위한 것들을 생각하면서 게임을 하니 잘 만들어진 게임은 정말 아름답다는 생각을 하게 되기까지 했다. 이번 프로젝트에서 그림을 많이 그리고 코딩 부분을 많이 하지 않았음에도, 보드에 전송방법, 보드의 성능, 게임의 실제적인 구현, 외부 device 등을 생각하다보니 수업시간과 랩시간에 배웠던 지식들을 꽤나 활용할 수 있어서 더욱 흥미로웠다.

물론 나도 한 숨도 안자고 계속 프로젝트를 했지만, 그 옆에서 부들부들 떨며 코딩을 하는 일환이를 보니 정말 수고를 많이 한다고 생각을 했다. 흐릿해지는 의식 속에서도 꿋꿋하게 코딩하는 모습에 고맙다고 생각을 했다.

프로젝트가 무사히 끝나서 다행이다. ㅎㅎ.

Ps0. 실습실은 너무나도 추웠다.

Ps1. 범수를 그린다고 범수한테 많이 얻어맞았다. 8ㅅ8

Ps2. 일환아 정말 수고 많이 했어. 니랑 플젝하면 정말 재미있다.

ㅎ 다음 학기 플젝도 ㄱ? 플젝은 넘나 재미있는 것

황일환(컴퓨터공학과, 20140996) - 010-5120-8593

[dlf0325@postech.ac.kr](mailto:dlf0325@postech.ac.kr)

프로젝트 같은 일을 할 때 나는 항상 보이는 것에 많은 신경을 쓴다. 그렇게 찾은 것이 바로 Graphics2D와 Postprocessor였다. 처음 이 기능들을 사용하려고 했을 땐 구글링에 의존해보려했다. 하지만 아무리 구글링을 해도 관련된 정보를 찾을 수 없었고, 오로지 s3c6410x매뉴얼에만 의존해서 코드를 작성했다. 초반엔 정말 머릿속이 하얬다. 하지만 랩 세션에서 제공된 소스와 매뉴얼들을 참고하면서 하나씩 코드를 만들어갔고, 결국에는 이 기능들을 성공적으로 다룰 수 있었다. 남들이 보면 뭐 그런 쓸데없는 고생을 사서 하느냐고 말할 수도 있겠지만 ‘프로젝트’라는 젊은 날의 선물을 이렇게 쓰지 않으면 대체 어떻게 쓰겠는가. 그래서 나는 프로젝트가 너무나 좋다. 다음 프로젝트도 최대한의 힘을 쏟을 것이다!