2022 AMD-Xilinx

PYNQ Summer Hackathon Report

Yonsei University

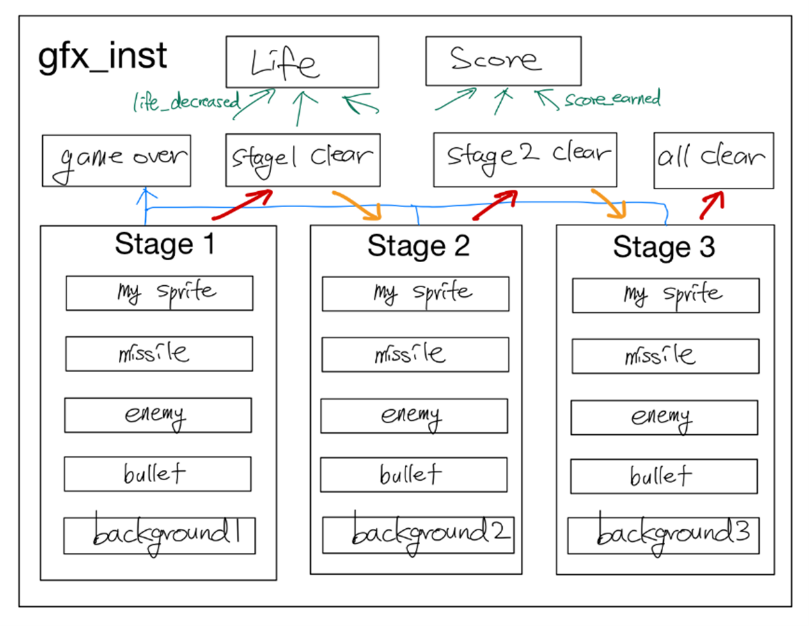
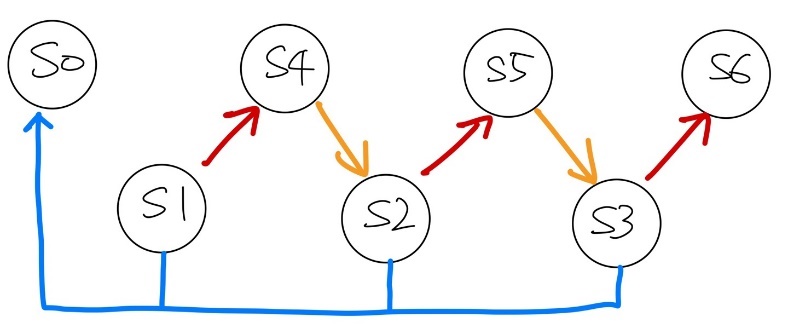
Electronic & Electrical Engineering dept.

2019142062 정성민

2020142173 가금민

# Step 0. Structure

소스코드의 전체적인 구성은 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| State | 게임 상태 | State transition | |
| S0 | 게임 오버 (life = 0) | X | |
| S1 | Stage 1 | 모든 적 클리어 -> S4 | Life = 0 -> S0 |
| S2 | Stage 2 | 모든 적 클리어 -> S5 | Life = 0 -> S0 |
| S3 | Stage 3 | 모든 적 클리어 -> S6 | Life = 0 -> S0 |
| S4 | Stage 1 clear 화면 | 3초 후 -> S2 | |
| S5 | Stage 2 clear 화면 | 3초 후 -> S3 | |
| S6 | All clear 화면 | X | |

위의 state 다이어그램에서 life, 점수, 각 스테이지 모듈은 gfx 바로 아래에 위치해 있고, 캐릭터들은 스테이지 모듈 안에 위치해 있다. 또한 전체적인 게임의 구성은 finite state machine인데, 각 state에 대한 설명은 위의 표와 같다. Diagram의 빨간색, 주황색, 파란색 화살표는 state transition을 나타낸다. 빨간색 화살표는 각 스테이지에서 모든 적을 클리어했을 때 state가 변하는 것을 나타내고, 주황색 화살표는 stage1 clear, stage2 clear 화면에서 일정 시간 후 자동으로 각각 stage2와 stage3으로 넘어가는 것을 나타낸다. 또한 각 스테이지에서 life가 0으로 떨어지면 파란색 화살표를 따라 game over로 넘어간다.

또한 각 스테이지 모듈은 라이프와 점수 정보를 gfx\_inst에 넘겨주어 출력될 수 있도록 하는데, 이 데이터의 흐름을 맨 위 다이어그램의 초록색 화살표로 표현하였다.

# Step 1. Make basic of the game

**[Display background graphic, the Player, the Enemies]**

텍스트, 전자기기, 컴퓨터, 디스플레이이(가) 표시된 사진

자동 생성된 설명기본적인 게임 화면 구성은 다음과 같다. 게임 화면을 보면 내 캐릭터, 적 캐릭터, 라이프, 점수가 배경과 함께 sprite로 나타난다.

**[Display life and score]**

왼쪽 위에는 life, 오른쪽 위에는 점수가 표시되어 있다. 이 세 가지는 sprite로 구현하였는데, 파일명은 <life\_sprite.v>, <life\_heart.v>, <score\_sprite.v>, <score\_number.v>이다. 위치를 변화시키는 코드 없이 처음 지정된 좌표에 고정되어 있다.

텍스트이(가) 표시된 사진

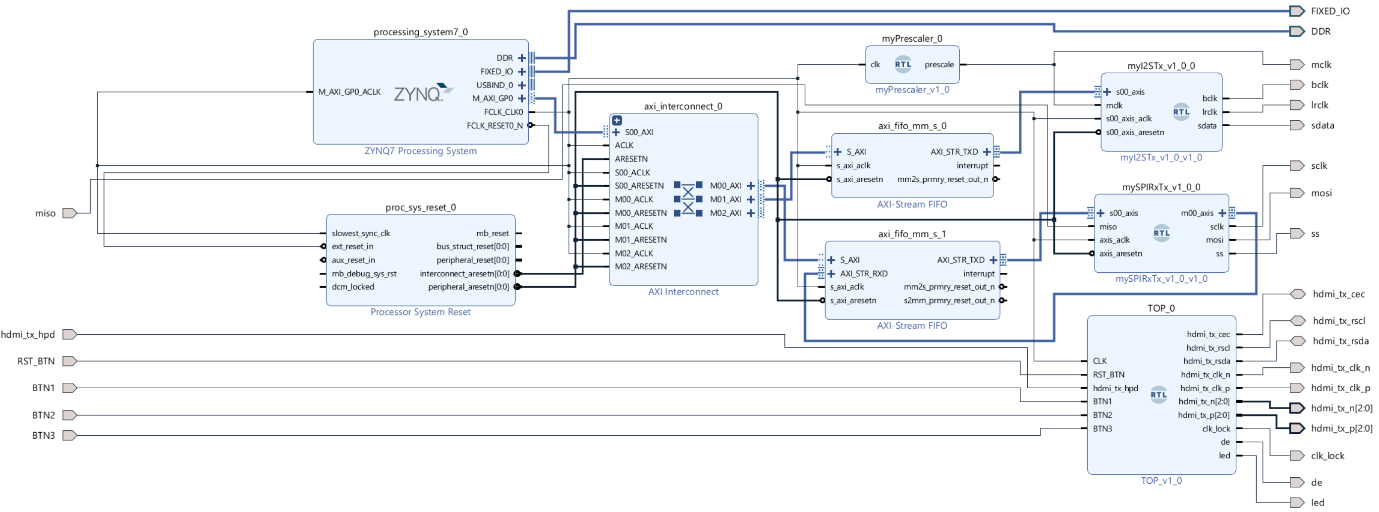
자동 생성된 설명

Sprite의 구현에 가장 핵심적인 부분은 위 코드인데, sprite\_hit\_x, y는 각각 sprite가 표현될 x축, y축 범위이고, sprite\_render\_x, y는 sprite의 맨 왼쪽 위의 좌표로부터의 상대적인 좌표값이다. 이때 >>2를 하게 되면 상대적인 좌표값에서 LSB 2비트를 무시하기 때문에 2^2=4배 확대된 이미지가 나타난다.

추가적으로, 3개의 stage와 각 enemy, 총알 등을 별개의 파일로 만들었다. 따라서 이 보고서에서는 stage1, my\_sprite1, missile1, enemy1, bullet1을 기준으로 설명할 것이다. 숫자만 다른 모듈들은 구조와 기능이 동일하다.

# Step 2. Make background music

Block diagram은 아래 그림과 같다. 배경음악을 재생하기 위해 axi\_interconnect, proc\_sys\_reset, axi\_stream\_fifo 등의 IP를 추가하였고, myI2STx\_v1\_0.v, myPrescaler.v, mySPIRxTx\_v1\_0.v 등의 모듈들을 생성하였다. 또한 이 파일들에서 사용하는 clk, mclk등 변수를 사용하기 위해 xdc 파일을 수정하였다. Block diagram은 아래와 같다.



마지막으로 오른쪽 아래에 TOP 블록을 추가하였는데, 이는 HDMI 통신과 게임 구현 코드를 한꺼번에 모아 놓은 것이다. TOP 블록에는 인풋으로 hdmi\_tx\_hpd와 버튼 4개, 클럭 신호가 들어간다. 버튼은 게임에 사용되는 버튼과 동일하고, 클럭은 ZYNQ 보드에서 사용하는 클럭이 들어간다. 또한 출력으로는 hdmi 관련 신호들이 출력되는데, 이러한 입력 포트와 출력 포트들은 xdc 파일에 정의된 이름과 기능(버튼, 입출력 등)을 그대로 사용한다.

# Step 3. Make the Player move

**[The Player moves left and right with a push button input]**

|  |  |
| --- | --- |
| BTN3 | BTN2 |
| 왼쪽으로 이동 | 오른쪽으로 이돟 |

각 버튼의 동작과 state diagram은 위와 같다. 리셋 버튼을 사용하는 state는 없으므로 diagram에서는 생략하였다.

BTN3을 누르면 왼쪽, BTN2를 누르면 오른쪽으로 내 캐릭터가 움직여야 하므로 각 버튼 변수를 xdc 파일을 통해 불러와 TOP -> gfx -> 각 stage -> my\_sprite 순서로 전달하였다.

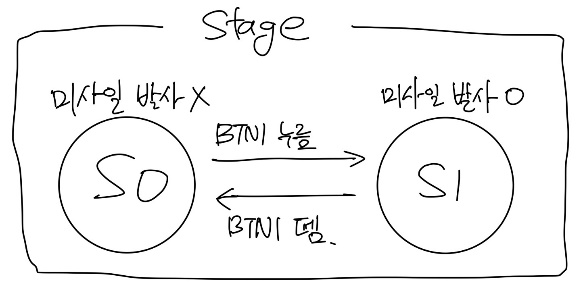
<my\_sprite1.v>의 78-86번 줄에서는 always문 안에서 각 버튼이 눌렸을 때 sprite\_x, sprite\_y 변수의 값을 바꾸는 방식으로 캐릭터의 위치를 이동시킨다. BTN3이 눌렸을 때는 sprite\_x에 매 i\_v\_sync마다 2를 더하고, 반대로 BTN2가 눌렸을 때는 2를 빼는 것을 알 수 있다. 1이 아닌 2를 더하고 빼는 것은 속도를 두 배로 만들기 위해서이다.

# Step 4. Make the Player attack the Enemies

**[The Player shoots projectile if the push button is pressed]**

Missile 모듈을 만들어 BTN1이 눌렸을 때 전방으로 발사되도록 했다. 우선 버튼이 눌리기 전에는 아래 사진과 같이 미사일이 내 캐릭터 앞에 붙어서 함께 움직인다. 이는 내 캐릭터 좌표를 미사일 모듈로 전달한 후, 받은 좌표를 미사일의 좌표에 대입했기 때문이다. <my\_sprite1.v>의 65~66번 줄에서 현재 좌표를 스테이지 모듈로 출력하고, <stage1.v>의 146~147번 줄에서 다시 미사일 모듈로 넣어 주는 것을 확인할 수 있다.

 텍스트, 옅은이(가) 표시된 사진

자동 생성된 설명   
 <변수의 이동>

<missile\_sprite1.v>의 100~102번 줄에서는 미사일의 초기 위치를 설정해 준다. 버튼이 눌리기 전에는 93-95번 줄에 의해 내 sprite와 함께 이동한다. 85번 줄에서는 미사일을 발사한다. 82번 줄에서 BTN1이 눌려지면 launched 변수를 1로 만들어 발사 상태가 되도록 하고, y좌표를 4씩 감소시킴으로써 위로 이동하게 만든다. 또한 BTN1을 눌렀다 떼더라도 이동 상태를 유지하기 위해 78~80번 줄을 추가하였고, 화면 밖으로 벗어났을 때 다시 플레이어의 현재 위치로 복귀시키기 위해 87~95번 줄을 추가하였다.

**[The Enemy crushes if it collides with the Player’s projectile] (동영상 11초)**

<stage1.v>의 228~236번 줄에서는 적이 미사일에 맞았는지를 판단한다. missile\_sprite\_hit 변수는 현재 픽셀에 미사일 스프라이트가 있는지를 나타내고, enemy\_hit 변수는 현재 픽셀에 적 스프라이트가 있는지를 나타낸다. 두 변수가 동시에 1이 된다는 것은 적이 미사일에 맞았다는 뜻이므로 이때 각 enemy에 해당하는 enemy\_killed 변수를 1로 만든다.

Stage3의 Boss는 미사일을 두 번 맞아야 죽는다. 또한 Boss의 life를 표시하기 위해 boss 아래에 두 개의 하트를 출력하였다. <stage3.v line 62-65>에서는 Boss의 life 2개의 기본 위치를 설정하고 있다. 또한 221-232번 줄은 boss가 움직이는 것에 맞춰 life도 같이 자동으로 좌우 이동을 시키는 부분이다. 252-254번 줄에서는, 만약 player의 미사일이 boss의 하단 8픽셀 안에 들어오면 \_hit 변수에 1을 더한다. 이는 hit 변수를 1로 만든다. 이때 219번 줄에서는 hit이 1이 될 때 오른쪽 하트가 출력되지 않도록 한다. 이후 235번 줄에서는 2초 후에 \_hit1 변수에 1을 더한다. 이는 hit1 변수를 1로 만든다. 256번 줄에서는 hit1이 1이 된 상태에서 player의 미사일이 boss의 하단 8픽셀 안에 들어왔을 때 enemy9\_killed 변수를 1로 만드는 부분이다.

enemy1\_killed 변수는 enemy 모듈과 bullet 모듈에 적용한다. (bullet은 적이 나에게 쏘는 총알) enemy1\_killed 변수가 1이 되면 그 적은 죽은 것으로 판단하여 화면 상에서 적과 그 적이 쏜 총알을 없앤다. <enemy\_sprite1.v>의 53~54번 줄에서 stage가 1이고 killed 변수가 0이면 원래 값을 출력하지만, killed가 1이 되면 hit를 항상 0으로 만든다. <bullet\_sprite1.v>의 41~42번 줄에서도 마찬가지다. Hit가 0이 되면 <stage1.v>의 272~276번 줄, 242~246번 줄에서 그 sprite를 출력하지 않는다.

# Step 5. Make score system

**[Score increases if the Player destroys the Enemies] (동영상 11초)**

우선 각 스테이지마다 적 캐릭터가 죽었는지 여부를 enemy\_killed 변수로 설정하였다(<stage1.v> line 228-236). 그리고 모든 적의 enemy\_killed 변수를 총합하여 그 스테이지의 점수를 만든다(<stage1.v> line 239). gfx 모듈에서는 stage 1~3의 점수를 모두 합하여 하나의 총점을 만들고(<gfx\_inst.v> line 244), 이를 score\_number 모듈에 넣어 준다(<gfx\_inst.v> 165번 줄).

따라서 처음에는 enemy\_killed 변수가 모두 0이므로 총점은 0이지만, 적이 미사일에 맞을 때마다 맞은 적의 enemy\_killed 변수가 1로 바뀌게 되므로 총점이 늘어난다.

**[Display the current score on the screen] (동영상 11초)**

Score\_sprite.v 모듈은 단순히 “SCORE: ” 텍스트를 출력하는 부분이고, score\_number.v 모듈이 실제 숫자를 출력하는 부분이다. score\_numbers 모듈에서는 총점을 받아 우선 일의 자리, 십의 자리, 백의 자리로 분리한다(<score\_numbers.v> line 22-24). 이후 각 자릿수를 삼항연산자( ? : )로 처리하여 출력하게 된다(<score\_numbers.v> line 140-180). 이때 0부터 9까지 픽셀 matrix를 만들어 놓고, 각 자릿수에서 불러와 사용하였다.

# Step 6. Make the stages with clearing conditions

**(동영상 18, 45초)**



텍스트, 전자기기, 디스플레이, 컴퓨터이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

Gfx 모듈 아래에는 stage1, stage2, stage3 모듈과 함께 stage1\_clear, stage2\_clear, all\_clear 모듈이 존재한다. 이는 각 스테이지가 끝난 뒤 clear effect를 보여주기 위한 모듈이다. 또한 life가 0이 되었을 경우 game over 화면을 띄운다.

**[Show stage-clear effect when there is no Enemy left]**

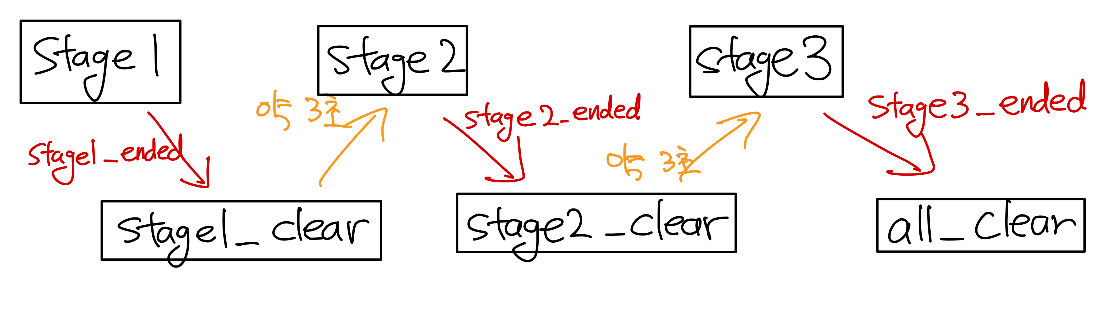
**(동영상 18, 45초)**

Enemy가 죽었는지 여부를 판단하는 방법은 위에서 사용했던 enemy\_killed 변수를 이용하는 것이다. 모든 적이 죽었을 때 스테이지가 종료되는 것이므로, 모든 enemy\_killed 변수를 and 처리하여 참일 때 stage\_ended 변수를 1로 만든다(<state1.v> line 238). gfx\_inst 모듈에서는 이 변수를 인식하여 state를 바꾼다(<gfx\_inst.v> line 204-231).

**[Move to the next stage if you have cleared the stage]**

**(동영상 18, 45초)**

우선 gfx 모듈에는 각 스테이지가 끝났는지를 판단하기 위한 stage1\_ended, stage2\_ended, stage3\_ended 변수가 있다(<gfx\_inst.v> line 32-34). 각 스테이지에서는 그 스테이지가 끝났는지 여부를 출력하는데, 그 출력을 담는 변수들이다. <gfx\_inst.v> line 204-231은 finite state machine의 state transition을 담당하는 부분인데, stage ended 신호를 받았을 때 state를 바꾸게 된다. State diagram은 아래 그림과 같다.



각 스테이지가 끝나면 약 3초간 stage clear 화면을 보여 준다. 이 화면은 별다른 기능 없이 3초가 지나면 다음 stage로 넘어간다. 이때 i\_v\_sync 신호의 주파수는 100000000 \* 37.12 / 10 / 5 /1650 / 750 = 59.99Hz이다. 따라서 이론적으로 i\_v\_sync가 180번 들어오는 것을 인식하면 3초의 시간을 잴 수 있다(<gfx\_inst.v> line 215-221). 이러한 방식으로 clear effect를 구현하였다.

**[Game complete effect if we complete the final stage] (동영상 60초)**

Stage3에서 적들을 모두 파괴하면 stage3\_ended 변수가 1이 되면서 all\_clear 상태로 state가 넘어간다. 해당 상태에서는 game complete effect를 보여 주게 된다.

**[At least three-stage with different conditions]**

우선 1단계에서는 체력이 1인 적 세 마리가 집단적으로 움직이며 총알을 발사한다. 집단적으로 움직이도록 하기 위해 각 sprite별 가동 범위를 제한하였다. 이는 아래에 자세히 설명하였다.

2단계에서는 5마리의 적을 구현하였다. 앞 3마리, 뒤 2마리로 묶여서 움직인다. 뒤의 2마리는 크기가 더 크고, 점수는 한 마리당 3점이다. 이때 점수의 가중치는 enemy\_killed 변수를 더하여 스테이지 점수를 계산하는 과정에서 구현하였다(<stage2.v> line 318).

3단계에서는 거대한 한 마리의 적이 등장한다. 이 적은 life가 2개라 두 번을 맞춰야 파괴할 수 있다. 점수는 10점이다.

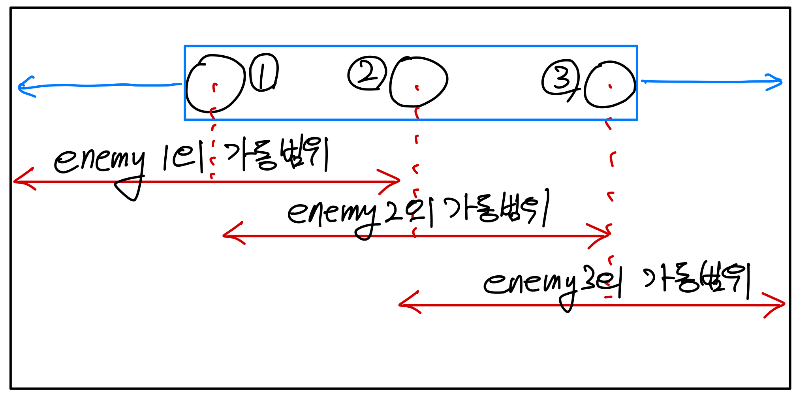
# Step 7. Make the Enemies move

**[Horizontal movement within display boundary]**

적 스프라이트 모듈 내에서 매 i\_v\_sync 신호마다 왼쪽 또는 오른쪽으로 한 픽셀씩 이동하도록 설정하였다(<enemy\_sprite1.v> line 72-82). 이때 적이 화면 경계 안에서만 움직이도록 하기 위해 sprite\_x\_direction 변수를 사용하였다. 이 변수가 1이면 오른쪽으로 움직이는 것, 0이면 왼쪽으로 움직이는 것인데, 조건문을 이용하여 변수가 1이면 sprite\_x 변수에 1을 더하고, 0이면 1을 빼는 방식으로 이동한다(line 73).

Sprite\_x\_direction 변수는 sprite가 화면 끝에 도달하면 invert된다. Sprite가 화면 끝에 도달했는지를 확인하는 방법은 x좌표값을 사용하는 것이다. x좌표값이 0이면 화면 왼쪽 끝에 도달했다는 뜻이고, (1280 – sprite 너비)이면 오른쪽 끝에 도달했다는 뜻이다.

**[Aligned, collective movement]**



적이 aligned, collective하게 움직이려면 위와 같이 가동범위를 설정해야 한다. Stage1에서는 적이 총 3마리이고, 초기 x좌표는 각각 400, 600, 800이다. 화면의 경계는 x=0, x=1280이므로 각 적의 가동 범위를 설정하면 아래와 같다. 이때 x좌표 최댓값을 설정할 때는 sprite의 너비를 빼야 한다.

|  |  |  |
| --- | --- | --- |
| 이름 | x좌표 최솟값 | x좌표 최댓값 |
| Enemy1 | 0 | 880 - 64 |
| Enemy2 | 200 | 1080 - 64 |
| Enemy3 | 400 | 1280 - 64 |

다른 stage들도 같은 방식으로 가동 범위를 고려하여 이동을 구현하였다.

# Step 8. Make the Enemies attack

**[The enemy shoots projectiles]**

각 enemy마다 그 enemy가 쏘는 bullet 모듈이 있다. Bullet은 미사일과 다르게 자동으로 발사되어야 한다. 또한 총알이 발사된 후에는 수직으로 이동하지만, 매 발사마다 적이 이동하므로 매 발사마다 발사 위치가 바뀐다.

우선 <bullet\_sprite1.v>의 68~71번 줄에서는 미사일의 초기 위치를 설정해 준다. 초기값이 설정되었다면, 58~64번 줄에서 총알을 발사한다. 총알이 발사된 후에는 매 i\_v\_sync 신호마다 y좌표에 1을 더하여 1픽셀씩 내려오도록 했다. 또한 총알의 y좌표인 sprite\_y 변수가 720 – 20이 되었다는 것은 화면 맨 아래에 닿았다는 것이므로 이 경우 y좌표에 1을 더하는 것이 아닌, x, y좌표에 적의 현재 좌표를 넣어 준다(line 62-65). +15, +32가 있는 이유는 총알이 적의 아랫부분에서 시작되도록 조정하기 위해서이다. 또한 720에서 20을 뺀 이유는 총알의 높이가 20픽셀이기 때문이다.

이렇게 하면 총알이 화면 맨 아래에 도달한 이후 적의 위치로 이동하여 다시 수직으로 내려오게 된다. 이를 통해 총알이 화면 밖으로 나가고 두 번째 총알이 발사되는 것처럼 구현할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

적에게 공격받는 것은 미사일로 적을 맞추는 것과 알고리즘이 유사하다. 다만 미사일로 적을 맞출 때는 맞기만 하면 적을 없애 버리면 되지만, 적에게 공격받을 때는 life를 1씩만 줄여야 하기 때문에 한 총알로 공격받았을 때 한 번만 인식하는 것이 중요하다.

<stage1.v>의 204~212번 줄을 보면, 조건문에 크게 2개의 조건이 있다.

1. 내 x좌표 < 총알 x좌표 < 내 x좌표+44

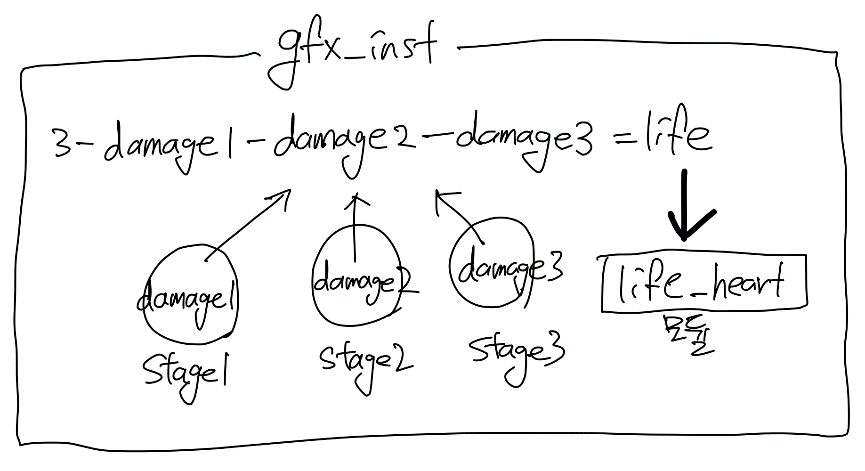
이는 총알이 내 sprite의 가로 범위 안에 들어와야 한다는 조건이다. +44를 한 이유는, 내 sprite의 너비가 64, 총알의 너비가 20이기 때문에 오른쪽 경계를 맞추기 위해 44만을 더하였다.

1. 내 y좌표 - 10 < 총알 y좌표 < 내 y좌표

위와 유사한 이유로, sprite의 y좌표의 기준은 왼쪽 위인데, 총알의 맨 아랫부분과 내 sprite의 맨 위가 맞았을 때를 인식하기 위해 y좌표 조건을 추가하였다. 이때 인식을 정확하게 하기 위해 y좌표를 범위로 지정하였다. 이 조건으로 인해 내 sprite의 맨 앞부분이 맞았을 때만 맞은 것으로 인정된다.

**[Life decreases if the player collides with the enemy’s projectile] (동영상 66, 84초)**

이 게임에서 라이프를 측정하는 과정은 아래 그림에서 볼 수 있듯 각 스테이지마다 받은 데미지를 3에서 빼는 방식이다.

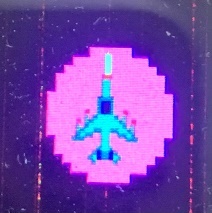
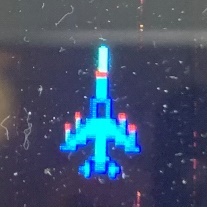
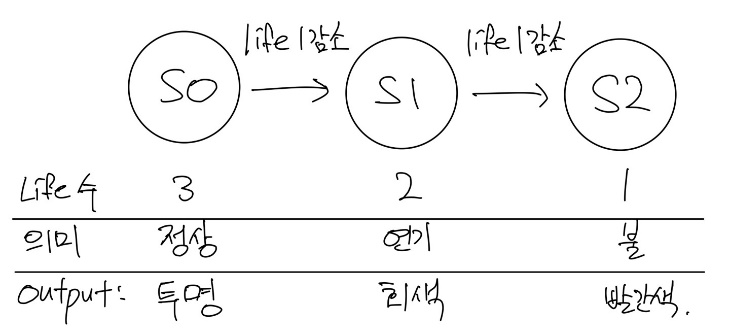


앞선 조건을 충족하여 적에게 공격받은 것으로 인식되면 life가 1 감소하게 되고, 이는 스테이지, gfx를 거쳐 life\_heart 모듈로 들어간다. Life 모듈은 life\_sprite 모듈과 life\_heart 모듈로 구분되어 있는데, life\_sprite 모듈은 단순히 “LIFE: ”를 출력하는 모듈이고, life\_heart 모듈이 들어온 life 수에 따라 하트 개수를 다르게 하여 내보내는 부분이다.

<life\_heart.v> 모듈의 46번 줄을 통해 별다른 구현 없이도 간단하게 하트의 개수를 조절할 수 있는데, sprite\_data의 크기보다 sprite\_hit\_x의 범위가 더 클 때 모양이 반복되어 출력되는 점을 이용하였다. 하트의 크기는 64픽셀이므로 sprite\_hit\_x의 너비를 64 \* 2 = 128로 하면 하트 2개, 64 \* 3 = 192로 하면 3개가 출력된다.

**[Some effect if the Player collides with the Enemy’s projectile] (동영상 66, 84초)**

라이프의 개수에 따라 아래의 그림과 같이 주변에 원형의 sprite를 구현하였다.

위 과정을 state diagram으로 표현하면 위와 같다. S0는 life=3인 상태로, 주변에 아무런 effect가 없다. S1은 life=2인 상태로, 주변의 연기를 표현하기 위해 회색의 원형 effect를 구현하였다. S2는 life=1인 상태로, 주변의 불을 표현하기 위해 빨간색의 원형 effect를 구현하였다.

코드 상에서는 이를 <explode.v> 모듈로 구현하였다. State가 변하는 것을 인식하는 방법으로는 explode 모듈에 life 변수를 넣어 준 후, life의 값에 따라 조건문을 통해 색을 변화시키는 방법을 사용하였다(<explode.v> line 24-59). 우선 24-29번 줄은 사용할 색을 정의하는 부분인데, 25번 줄의 경우 기존에 투명으로 지정되어 있으므로 이를 S0 state에 사용한다. 26번 줄은 S1, 27번 줄은 S2에 사용할 색을 정의한다. 28번 줄은 life=0이 되어 게임이 종료되었을 때 에러를 방지하기 위해 추가하였다. 이후 31-47번 줄에서는 원을 0과 1으로만 표현하였다. 이는 추후 1에 할당될 색을 바꿈으로써 출력을 바꾸기 위함이다. 56-58번 줄은 각각 RGB 색을 출력하는 내용으로, 구조가 동일하므로 56번 줄만 살펴보면 selected\_palatte가 1일 때 (즉 원 안의 픽셀일 때) palette\_colors[3-life][2]를 출력한다. 3-life를 한 이유는, life=3일 때 팔레트의 0번 줄, life=2일 때 팔레트의 1번 줄, life=1일 때 팔레트의 2번 줄을 사용하기 위해서이다.