AI 시스템반도체설계 2기
FINAL PROJECT
DOOM : C SLAYER

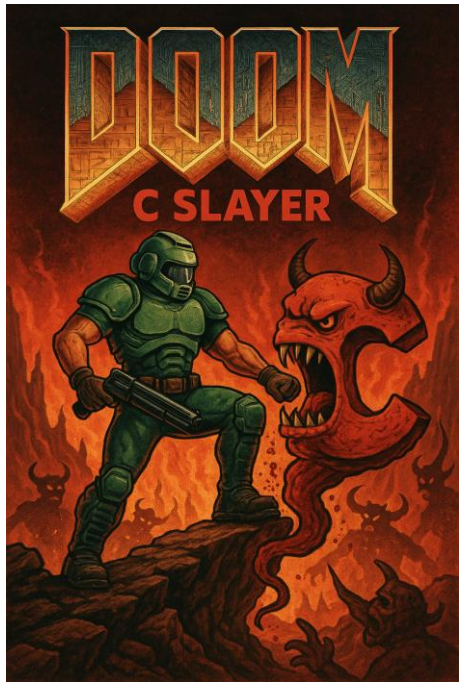
2025-05-07

JONG WAN KO (JONGWANKO@GMAIL.COM)

Contents

과제 개요

- **게임 제목:** DOOM : C Slayer
- **장르 :** 2D 슈팅 게임
- **컨셉 요약 :**
  “플레이어는 코딩 초보!”
  “지옥에서 탈출한 C언어 악마들과 전투”
  “스테이지별로 적을 처치”
  - 최종 보스(C보스)를 격파하여 게임 승리!



- 게임 컨셉 아트

- **개발 환경:** Visual Studio Code
- **개발 Tool Chain :** CodeSourcery Sourcery G++ Lite (GCC)
- **버전 :** 4.8.1
- **개발 보드 :** WILTEK 개발보드 (ARM Cortex-M3) + LCD

- 개발 보드 : WILTEK 개발보드 (ARM Cortex-M3) + LCD

개발 일정

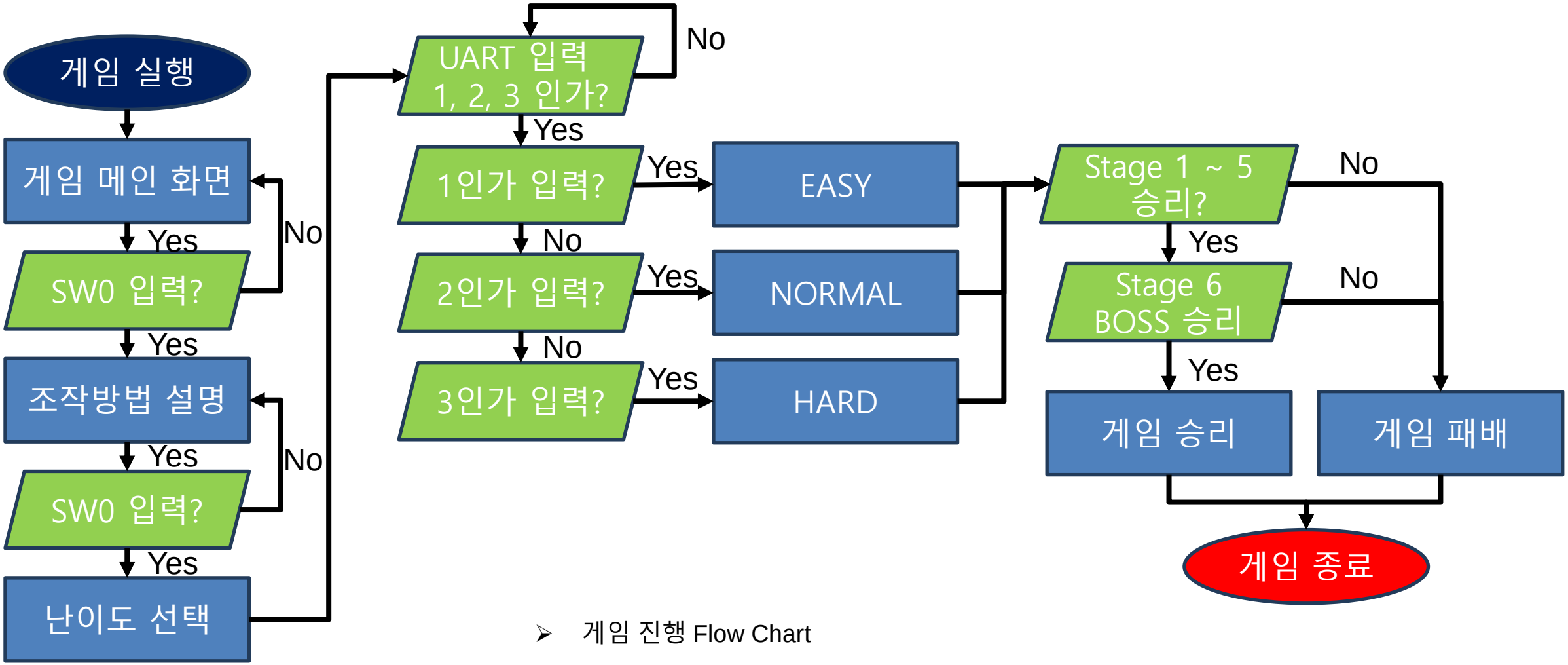
	4/25	4/26	4/27	4/28
전체 기획 및 코드 연구				
LCD 기반 타이틀 화면 구현				
Enemy 구조체 및 Spawn 기능 완성				
Enemy 이동 시스템 구현				
Player 공격 기능 구현				
Enemy 공격 기능 추가				
Boss 등장 로직 구현				
전체 시스템 통합				
디버깅 및 세부 수정				
시스템 최종 튜닝				
BGM 및 효과음 적용				
최종 점검 및 리허설				
프로젝트 발표				

	4/29	4/30	5/1	5/2
전체 기획 및 설계 완료				
LCD 기반 타이틀 화면 구현				
Enemy 구조체 및 Spawn 기능 완성				
Enemy 이동 시스템 구현				
Player 공격 기능 구현				
Enemy 공격 기능 추가				
Boss 등장 로직 구현				
전체 시스템 통합				
디버깅 및 세부 수정				
시스템 최종 튜닝				
BGM 및 효과음 적용				
최종 점검 및 리허설				
프로젝트 발표				

	5/3	5/4	5/5	5/6	5/7
전체 기획 및 설계 완료					
LCD 기반 타이틀 화면 구현					
Enemy 구조체 및 Spawn 기능 완성					
Enemy 이동 시스템 구현					
Player 공격 기능 구현					
Enemy 공격 기능 추가					
Boss 등장 로직 구현					
전체 시스템 통합					
디버깅 및 세부 수정					
시스템 최종 튜닝					
BGM 및 효과음 적용					
최종 점검 및 리허설					
프로젝트 발표					

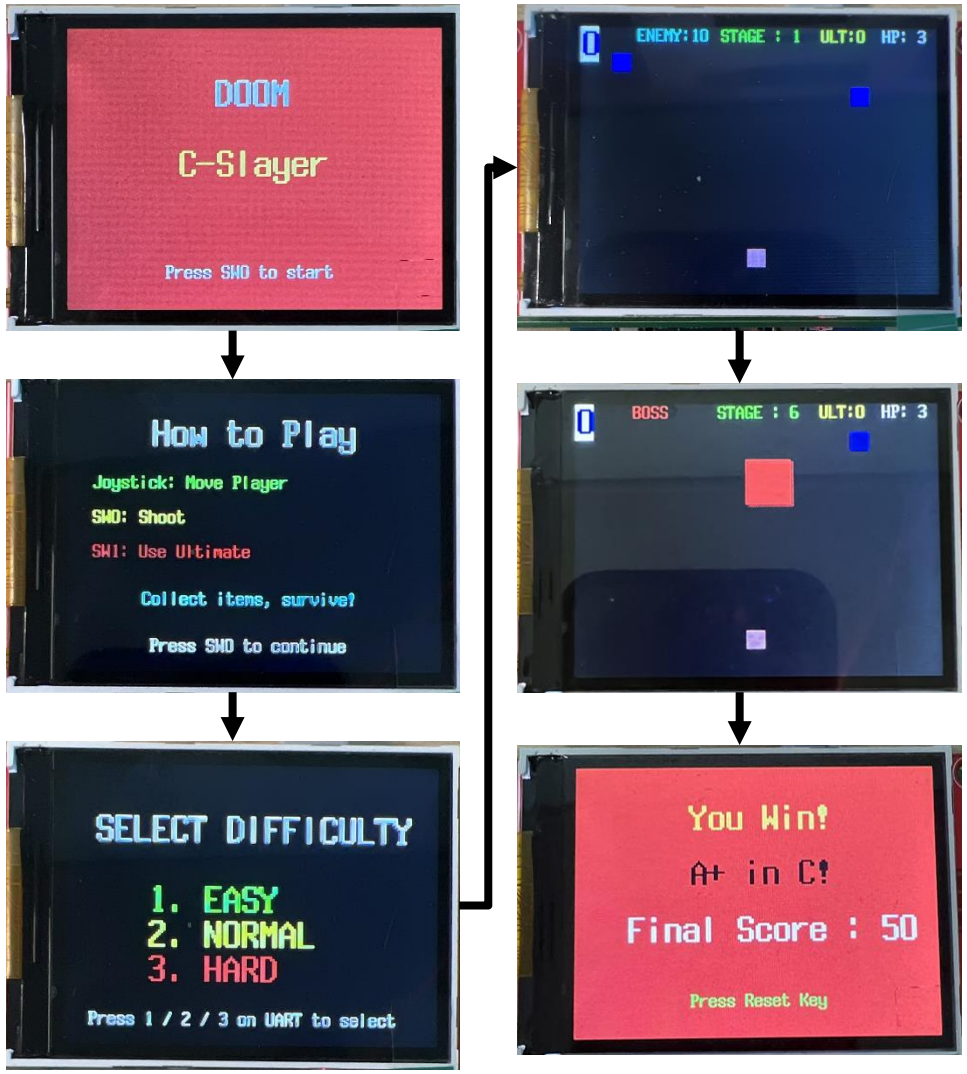
- 계획 대비 실제 개발을 하루 단축했습니다.

개발 결과



➢ 게임 진행 Flow Chart

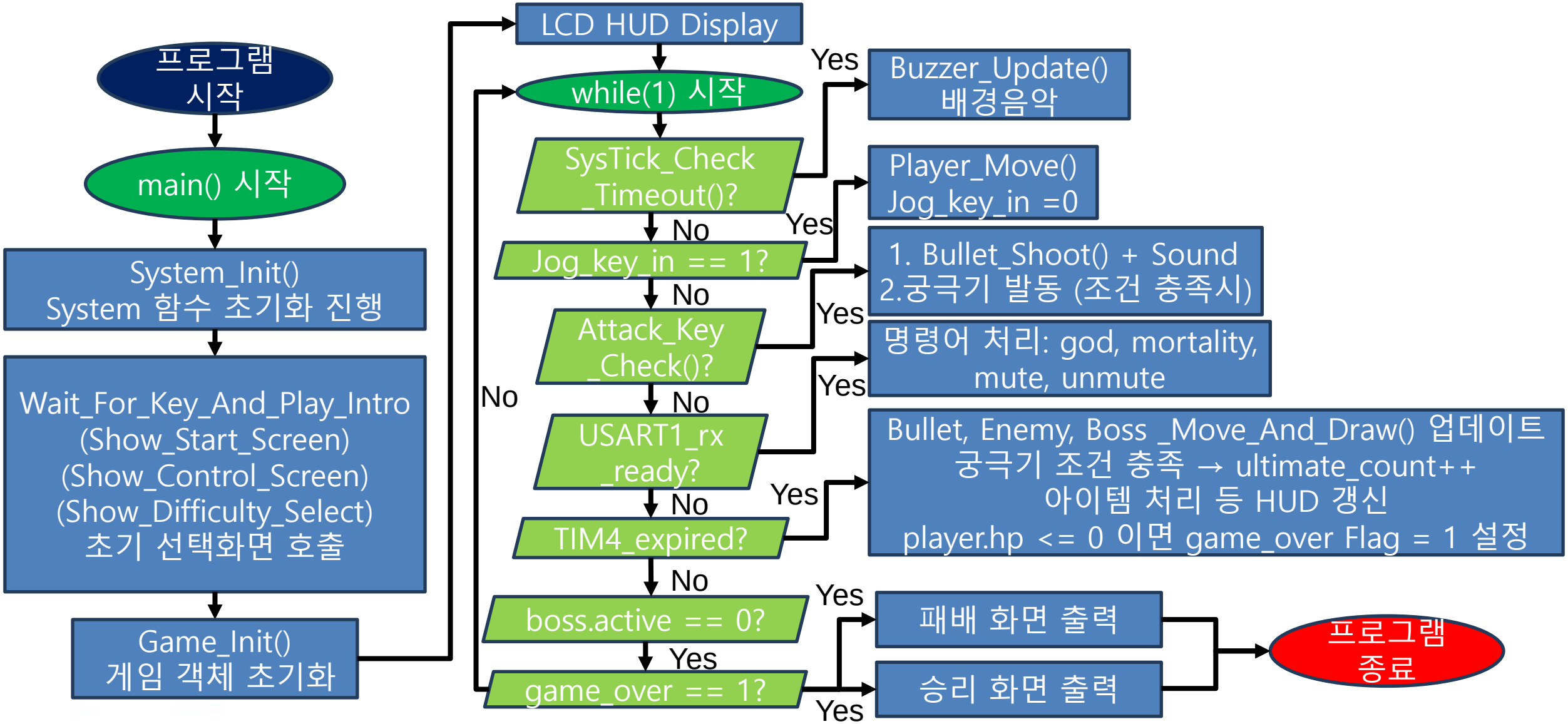
개발 결과 (Continued)



➢ 게임 실제 화면 및 단계별 사진

➢ 프로젝트 파일 구성

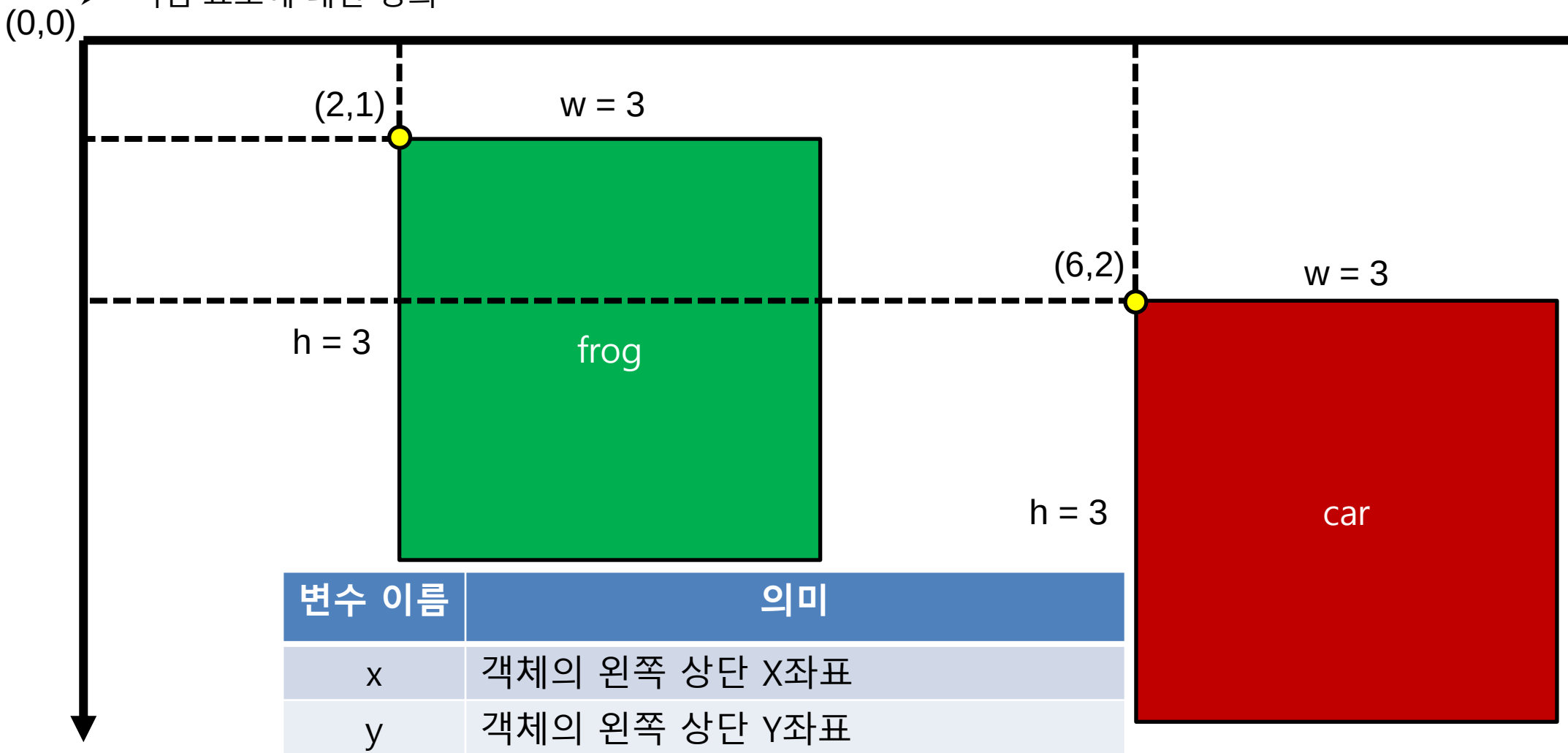
개발 결과 (Continued)



➢ main.c Flow Chart

핵심 기술(충돌 판정 알고리즘(AABB 방식))

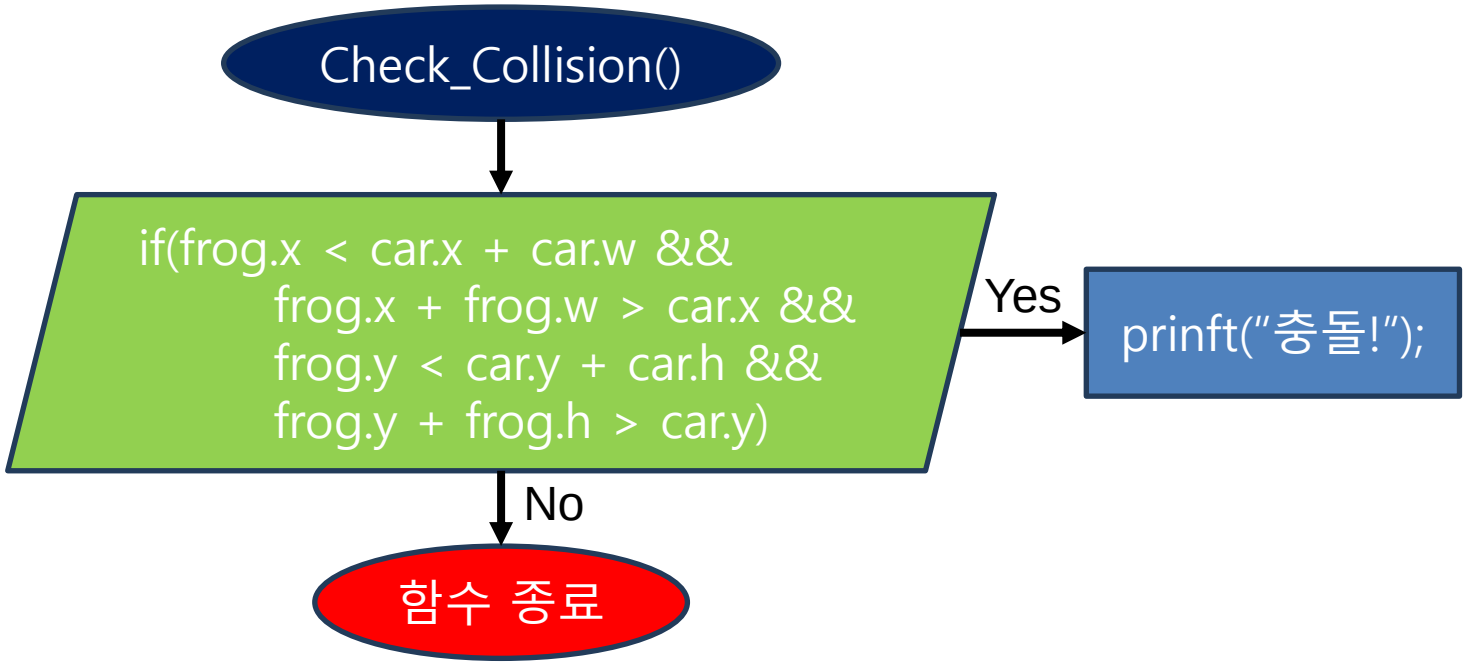
➢ 핵심 요소에 대한 정의

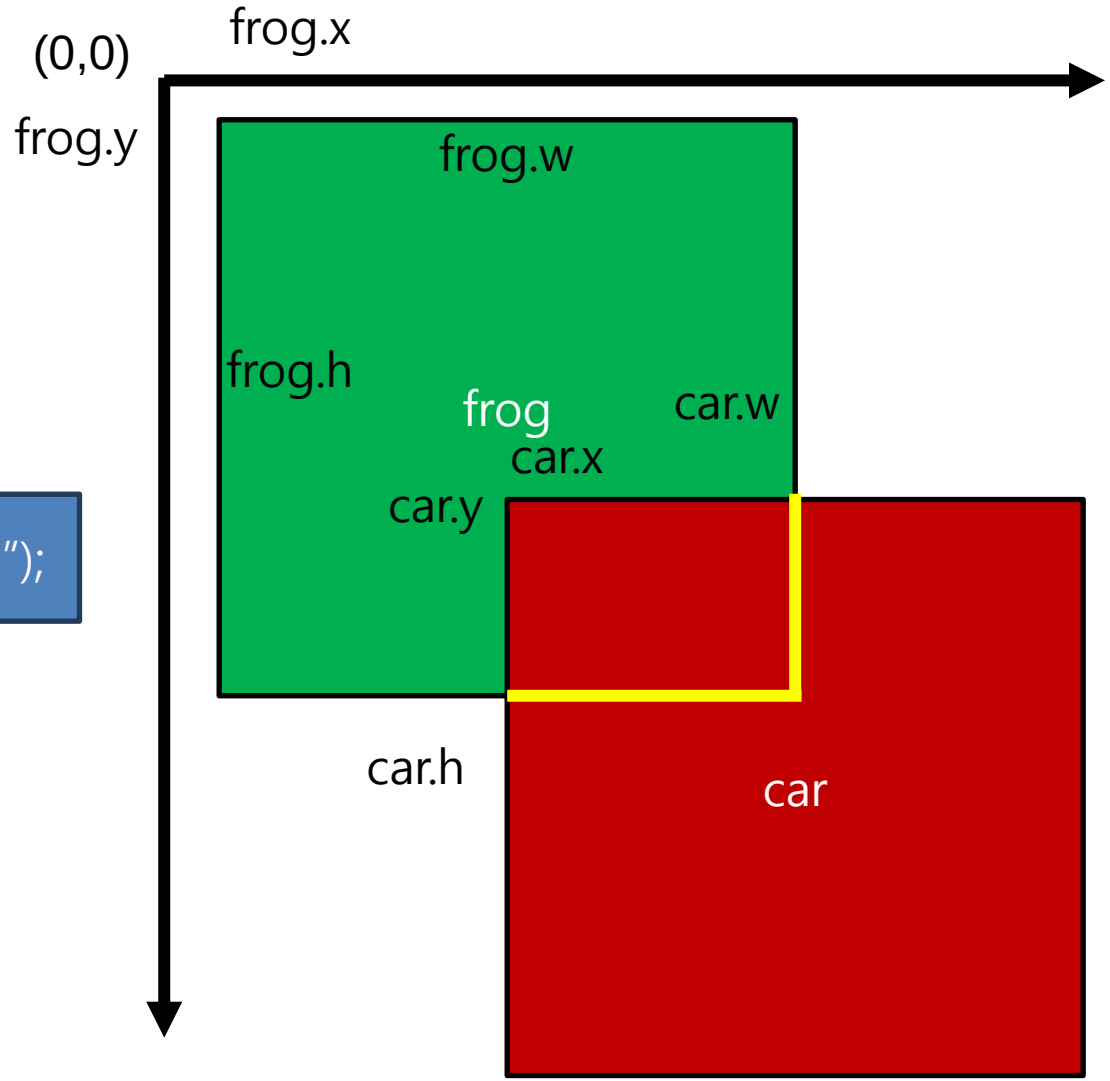


변수 이름	의미
x	객체의 왼쪽 상단 X좌표
y	객체의 왼쪽 상단 Y좌표
w	객체의 가로 길이 (width)
h	객체의 세로 길이 (height)

핵심 기술(충돌 판정 알고리즘(AABB 방식))

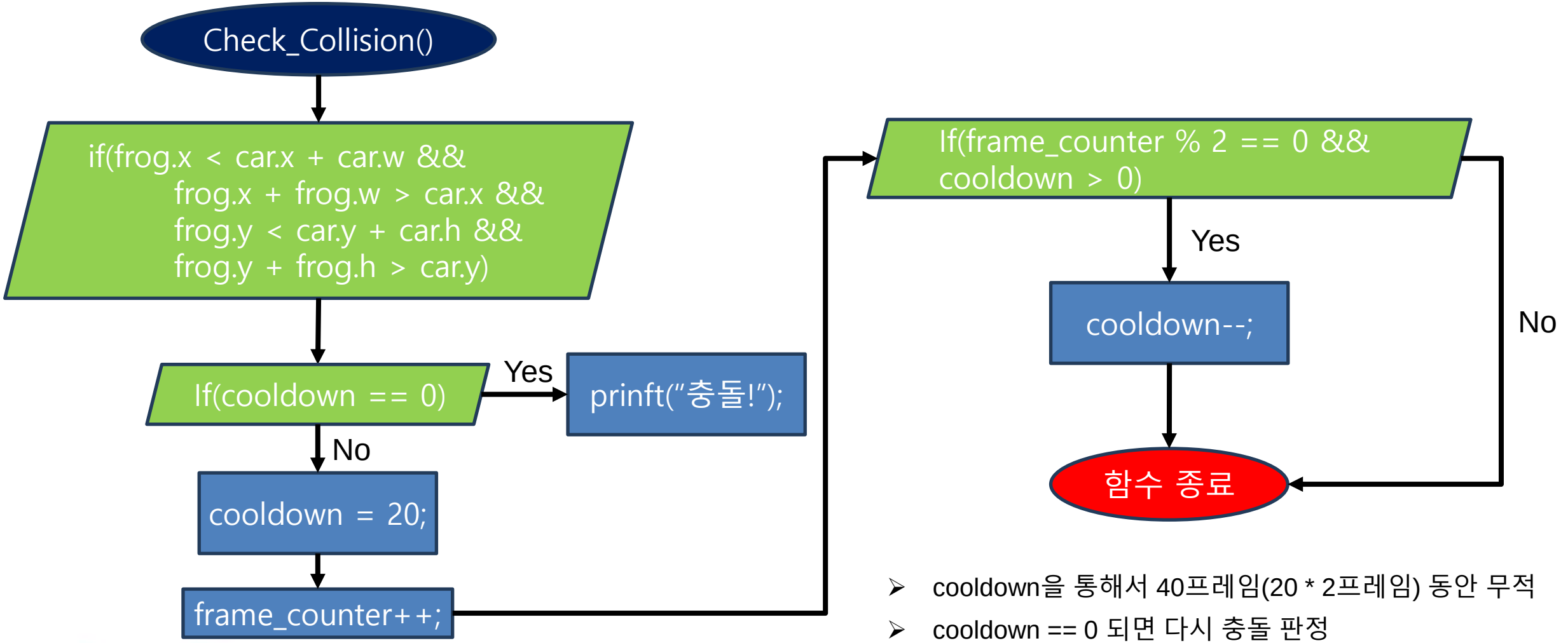
➢ 기본 논리 이해





핵심 기술(충돌 판정 알고리즘(AABB 방식))

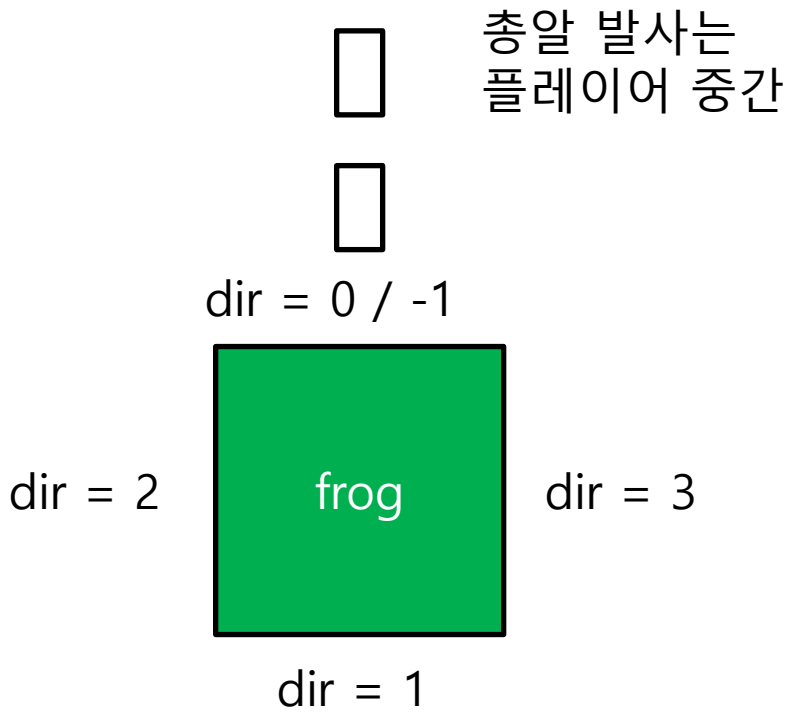
- 문제점: 중복하여 충돌 판정되어 체력이 무한히 감소 할 수 있음



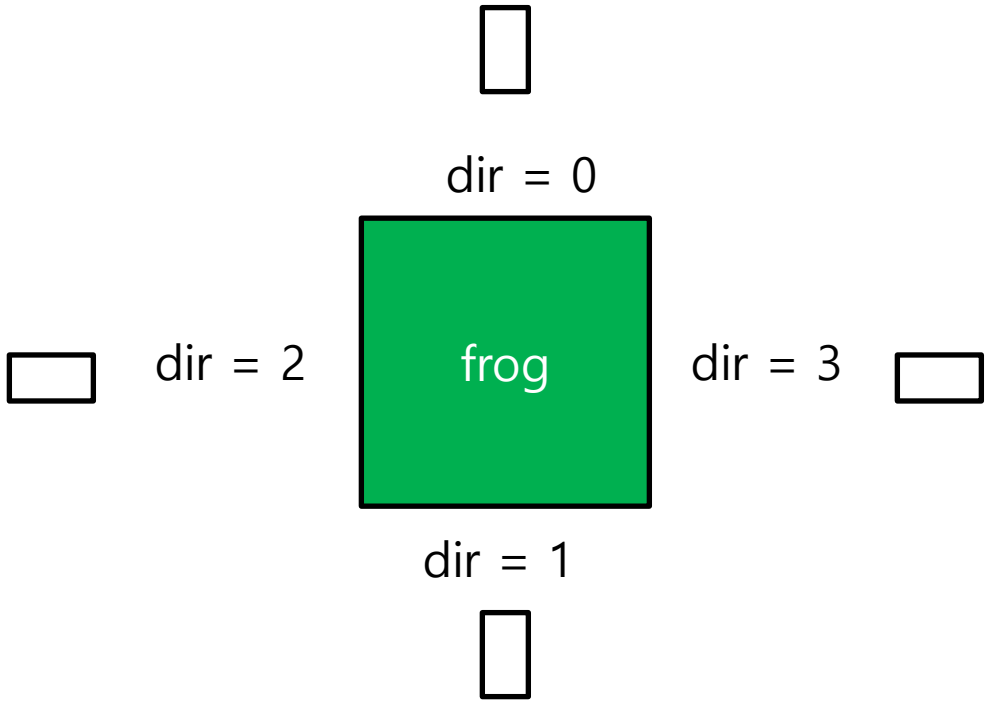
- cooldown을 통해서 40프레임(20 * 2프레임) 동안 무적
- cooldown == 0 되면 다시 충돌 판정

핵심 기술 (사격 알고리즘)

- 핵심 요소에 대한 정의



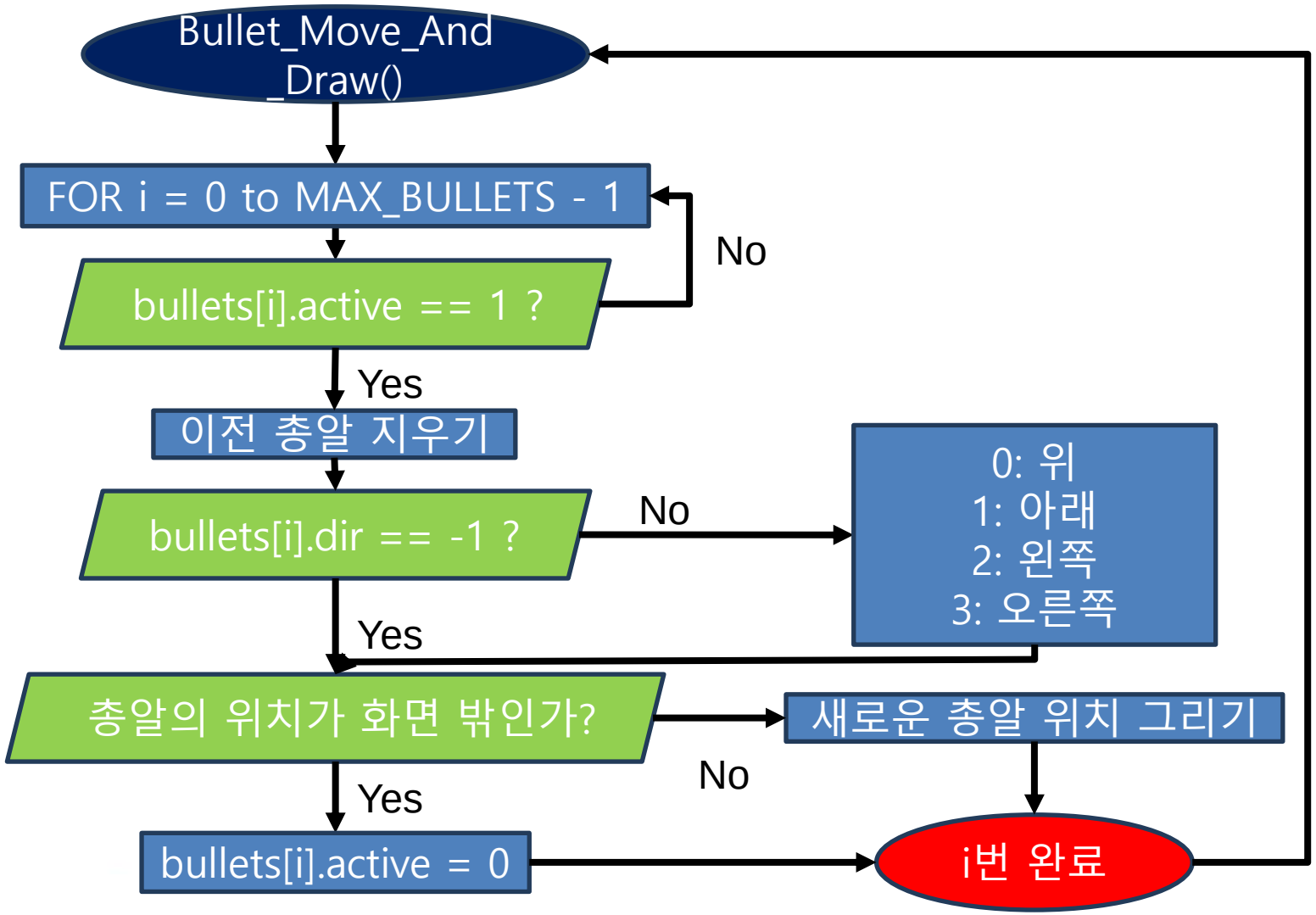
- SW0로 “위”로 발사



- SW1로 “위, 오른쪽, 아래, 왼쪽” 순서로 발사

핵심 기술(사격 알고리즘)

- 사격 알고리즘



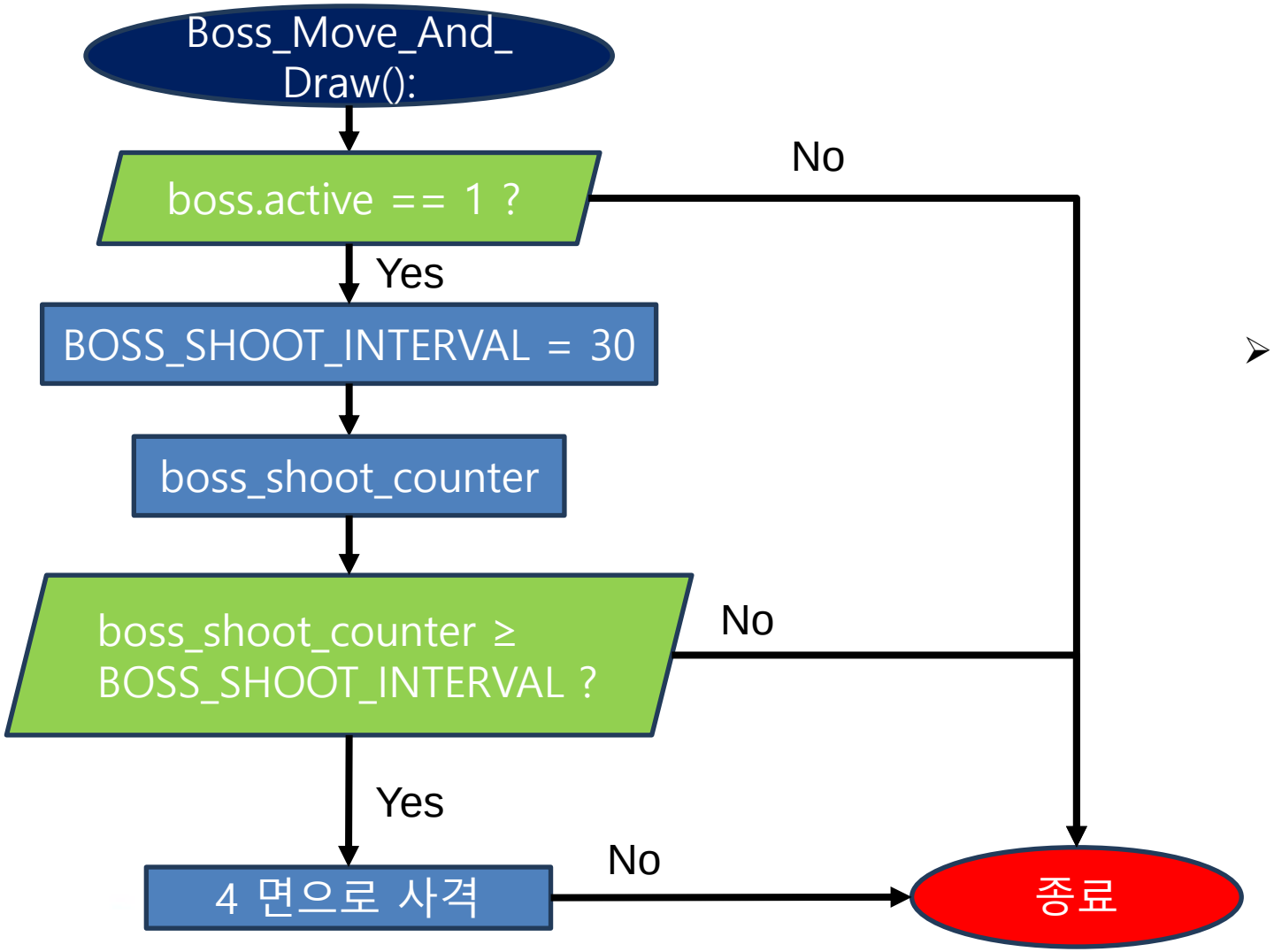
- 초기 설계 : Bullet_Shoot()
  - 위쪽 방향을 -1로 설정함

- 4면 설계 : Bullet_Shoot_Dir()
  - 위 0: 위
  - 1: 아래
  - 2: 왼쪽
  - 3: 오른쪽

- 총알 피격은 충돌 알고리즘과 동일

핵심 기술(사격 알고리즘)

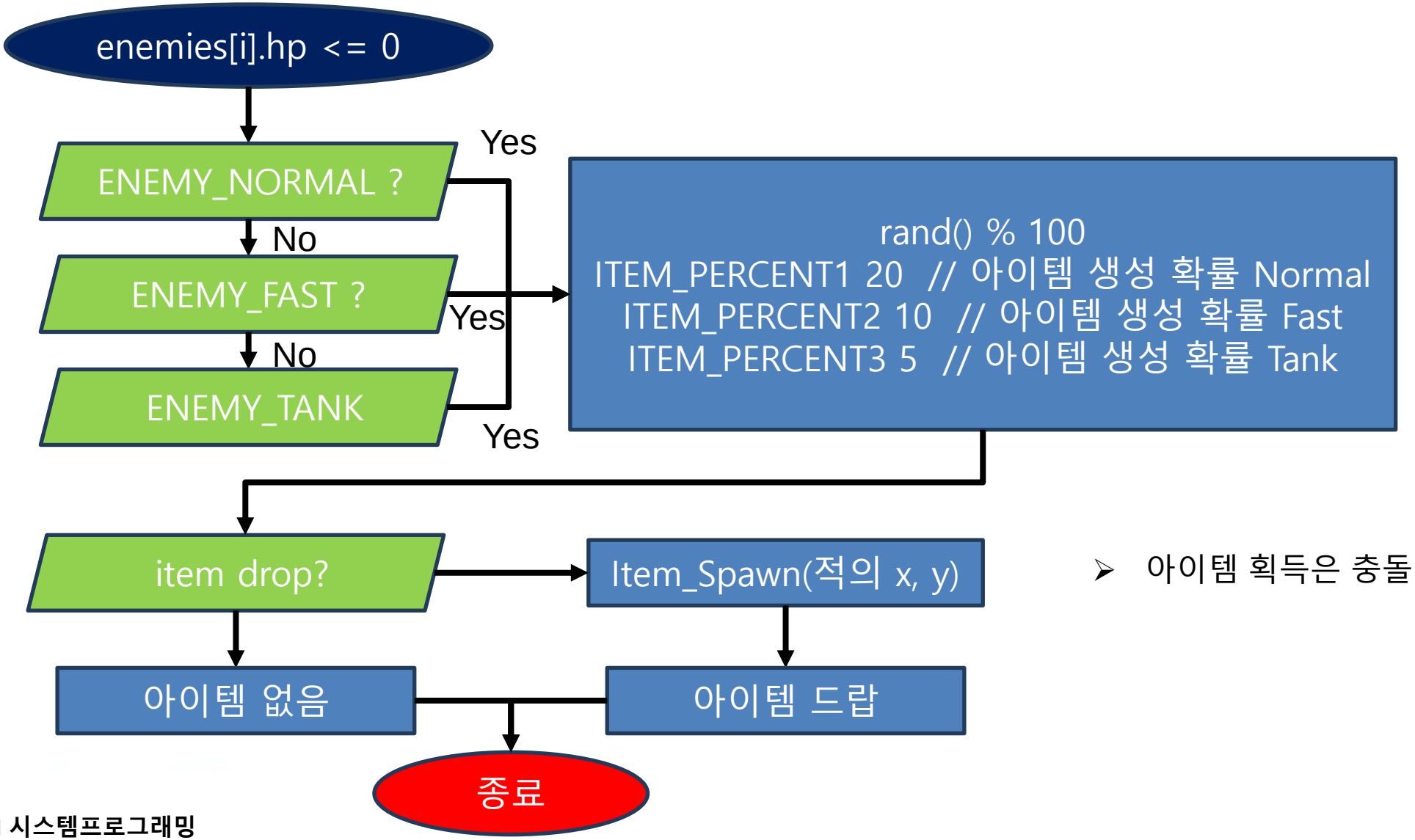
- 보스 사격 알고리즘



- cooldown과 유사한 방식으로 설계
  - BOSS_SHOOT_INTERVAL = 30 프레임
  - boss_shoot_counter로 30 될때마다 사격

핵심 기술 (아이템 드랍 알고리즘)

➢ 보스 사격 알고리즘



➢ 아이템 획득은 충돌 알고리즘과 동일합니다.

핵심 코드(궁극기 발사 시스템 pseudo code)

main.c

```
                                    …
if (ultimate_count)
{
    ultimate_remaining = 10;   // 발사 10회
    ultimate_count--;          // 보유 궁극기 감소
    Buzzer_Ultimate_Effect();  // 궁극기 사운드
                                    …
}
                                    …

if (ultimate_remaining > 0) {
    Bullet_Shoot_Dir(…); // 위쪽 발사
    Bullet_Shoot_Dir(…); // 오른쪽 발사
    Bullet_Shoot_Dir(…); // 아래쪽 발사
    Bullet_Shoot_Dir(…); // 왼쪽 발사
    ultimate_remaining--; // 10회중 총알 감소
}
                                    …
```

bullet.c

```
void Bullet_Shoot(x, y, w) // 플레이어 위치 x, y
{
    for (i = 0; i < MAX_BULLETS; i++)
    {
        if (!bullets_active)
        {
            // 총알의 가로 위치 = 플레이어 중심 (x + w/2 - 2)
            // 총알의 세로 위치 = 플레이어 바로 위 (y - 4)
            // 총알의 크기 = 가로 4, 세로 8
            // 총알의 속도 = -8 (위 방향)
            // 총알의 방향 = -1 (위쪽 의미)
            // 총알을 활성화함 (active = 1) 화면에 보이게
            break; // 총알 한 개씩 발사
        }
    }
}
```

➢ 궁극기 발사 시스템 pseudo code

핵심 코드(게임 난이도 시스템 pseudo code)

enemy.h

```
                                        …
typedef enum {
    ENEMY_NORMAL,
    ENEMY_FAST,
    ENEMY_TANK
} ENEMY_TYPE;
                                        …
```

enemy.c

```
                                        …
// 기본 속도는 3.0이며, 난이도와 현재 스테이지에 따라 증가
speed = base_speed + (stage × 속도 증가율 테이블[difficulty]);
                                        …
```

enemy.c

```
                                                    …
// (stage 증가에 따라 점진적 증가)
switch (enemy type)
{
    case ENEMY_NORMAL:
        enemies[i].hp = 2;  // 체력: 2
        enemies[i].speed = (int)speed; // 속도: 보통 속도
        break;

    case ENEMY_FAST:
        enemies[i].hp = 1;  // 체력: 1
        enemies[i].speed = (int)speed;
        break;

    case ENEMY_TANK:
        enemies[i].hp = 4; // 체력: 4
        enemies[i].speed = (int)speed;
        break;
}
                                                    …
```

- 게임 난이도 시스템 pseudo code

핵심 코드(디버깅 명령어 시스템 pseudo code)

main.c

```
                                                    …
if (UART 수신 완료)
{
    c = 수신된 문자;
    수신 완료 플래그 = 0;
    if (c가 개행 문자)
    {
        cmd[idx] = '₩0'; // 문자열 종료 문자 추가
        if (cmd == "god")
        {
         printf("[System] God mode enabled");
        }
        else if (cmd == "mortality")
        {
            무적 모드 해제;
        }
        else
        {
              Buzzer_Process_Command(cmd); // 음악 음소거
        }
                                                    …
    }
}
                                                    …
```

enemy.c

```
                                                    …
if (god_mode)
{
    printf("GOD MODE: Ignored ENEMY collision₩n");
}
else
{
    if (player.hp > 0) player.hp--;
    printf("PLAYER HIT! HP=%d₩n", player.hp);
}
                                                    …
```

buzzer.c

```
if (cmd == "mute")
{
    mute_enabled = 1;
    TIM3_Out_Stop (); // 음악 정지, output만 끄고 타이머 끄지 않음
    printf("[Buzzer] Background music muted");
}
else if (cmd == "unmute")
{
    mute_enabled = 0;
    printf("[Buzzer] Background music unmuted");
}
                                                    …
```

- 디버깅 명령어 시스템 pseudo code

핵심 코드(총알-적 충돌 및 아이템 시스템 pseudo code)

bullet.c

```
                                        ...
if (bullets[i].active && enemies[j].active)
{
    if (Check_Collision(bullets[i], enemies[j]))
    {
        bullets[i].active = 0;
        enemies[j].hp--;

        if (enemies[j].hp <= 0)
        {
            enemies[j].active = 0;
            score++;

            // 확률적으로 아이템 생성
            if (rand() % 100 < ITEM_PERCENT)
            {
                Item_Spawn(enemies[j].x, enemies[j].y);
            }
        }
    }
}
                                        ...
```

enemy.c

```
                                        ...
#define ITEM_PERCENT1 20  // 아이템 생성 확률 Normal
                                        ...
```

item.c

```
                                        ...
if (items[i].active && Check_Collision(player, items[i]))
{
    items[i].active = 0;

    if (player.hp < MAX_HP)
        player.hp++;  // 체력 회복

    score++;
}
                                        ...
```

- 총알-적 충돌 및 아이템 시스템 pseudo code

결론

- **과제 목표 및 실제 구현 상황**

과제 목표	목표	실제 구현
독립적인 Object	4	4
추가 구현 요소	2개(BGM과 효과음)	2개(BGM과 효과음)
스테이지별 난이도 조절	적의 속도 및 체력의 증가	적의 속도 및 체력의 증가
특수한 명령 기능	4개 명령어	4개 명령어

- **목표한 모든 과제 구현 완료 하였습니다.**

- **아쉬운 점 및 추후 개발 사항**
  - 기본적인 그래픽만 구현됨→ 캐릭터, 배경, 효과 등을 좀 더 생동감 있게 개선 필요
  - 사운드 시스템 단순→ 상황에 따라 다채로운 음악 전환 (Stage별 BGM, 타격 효과 다양화 등)
  - 적 AI 단순→ 추후 플레이어 추적, 랜덤 이동 등 추가하여 전략성 강화
  - 스테이지 구성이 반복적임→ 각스테이지 마다 다양한 미션을 제공
  - 보스전 단일 패턴→ 추후 패턴 공격 이나 타이밍 회피 등으로 게임성 향상 가능

개발 후기

- **프로젝트를 통해서 얻은 점**
  - **모듈화의 중요성 실감**
    - 이전 과정(1단계 C 프로그래밍)에서는 간단한 함수 단위 프로그램에 그쳤지만, 이번 과제에서는 여러 기능을 .c / .h로 나눠 함수화하고, main.c에서 통합하는 실전 구조를 직접 설계함으로써 프로그램의 구조적 설계 역량을 기를 수 있었습니다.
  - **실제 임베디드 흐름에 대한 이해**
    - ARM 시스템 프로그래밍 과정에서 배운 UART, Timer, GPIO, 인터럽트, LCD, Buzzer 등을 직접 다뤄보며, 하드웨어 제어 흐름이 어떻게 통합되어 동작하는지 실제 감각을 얻었습니다.

- **앞으로 기대 성과**
  - **플랫폼 독립적 개발 역량 기반 확보**
    - 하드웨어 제약이 있는 환경에서도 C 언어로 필요한 기능을 구현하고 디바이스를 제어하는 경험을 통해, 향후 어떤 플랫폼에서도 소프트웨어를 개발할 수 있는 기초적인 기반 역량을 갖추게 되었습니다.

Q & A