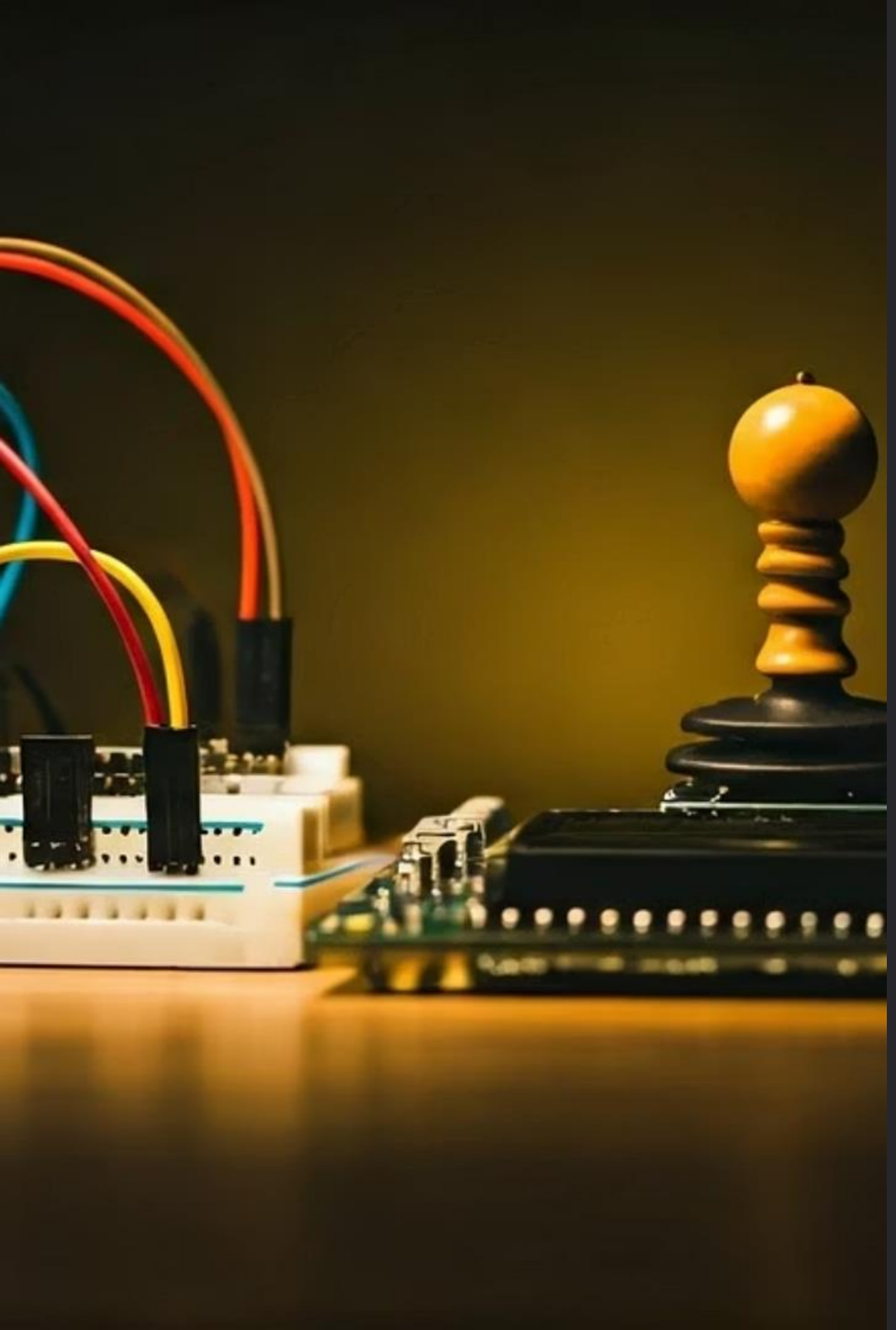
ARM 디바이스 프로그래밍
Dodge Rush

작성자: 임 도엽



과제 개요

게임 설명

플레이어가 적 미사일을 피하며 생존합니다. 아이템으로 무적 효과나 추가 생명을 획득할 수 있습니다.

사용 실습 환경

개발 MCU: STM32(Cortex-M3기반)

개발 툴: VS Code

디버깅 방식: USB 연결 + Tera Term을 통한 UART 출력 확인

빌드 환경: Windows 11

장비 소개

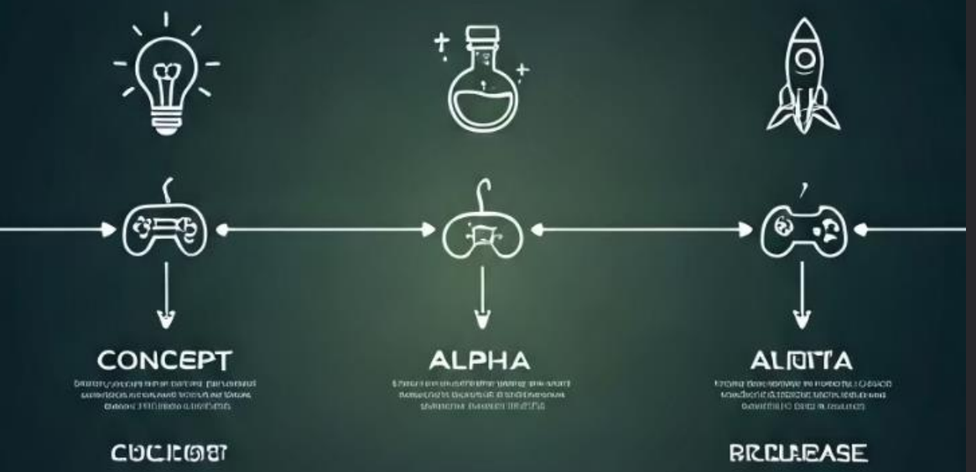
STM32 보드: Timer, GPIO, UART등 기본 기능 제공

조이스틱: 4방향 입력으로 플레이어 이동 제어

LCD 디스플레이: 게임 그래픽 출력에 사용

Buzzer: TIM3을 이용한 PWM 신호로 BGM출력

UART: UART1 포트를 통해 점수 출력 및 디버깅 메시지 확인



개발 일정

1

기획 단계

2025년 4월 25일

- 게임 컨셉 확정

2

개발 단계

2025년 04월 26일 ~ 5월 3일

- 26일: 플레이어 및 적 이동 구현
- 28일: 미사일 구현 및 충돌 체크
- 30일: 점수 및 게임 BGM 설정
- 2일: 아이템 구현 및 아이템 충돌 체크
- 3일: UI 개선 및 플레이어와 생명 이미지 대입

3

마무리 단계

2025년 5월 4일 ~ 5일

- 최종 발표 준비

개발 결과

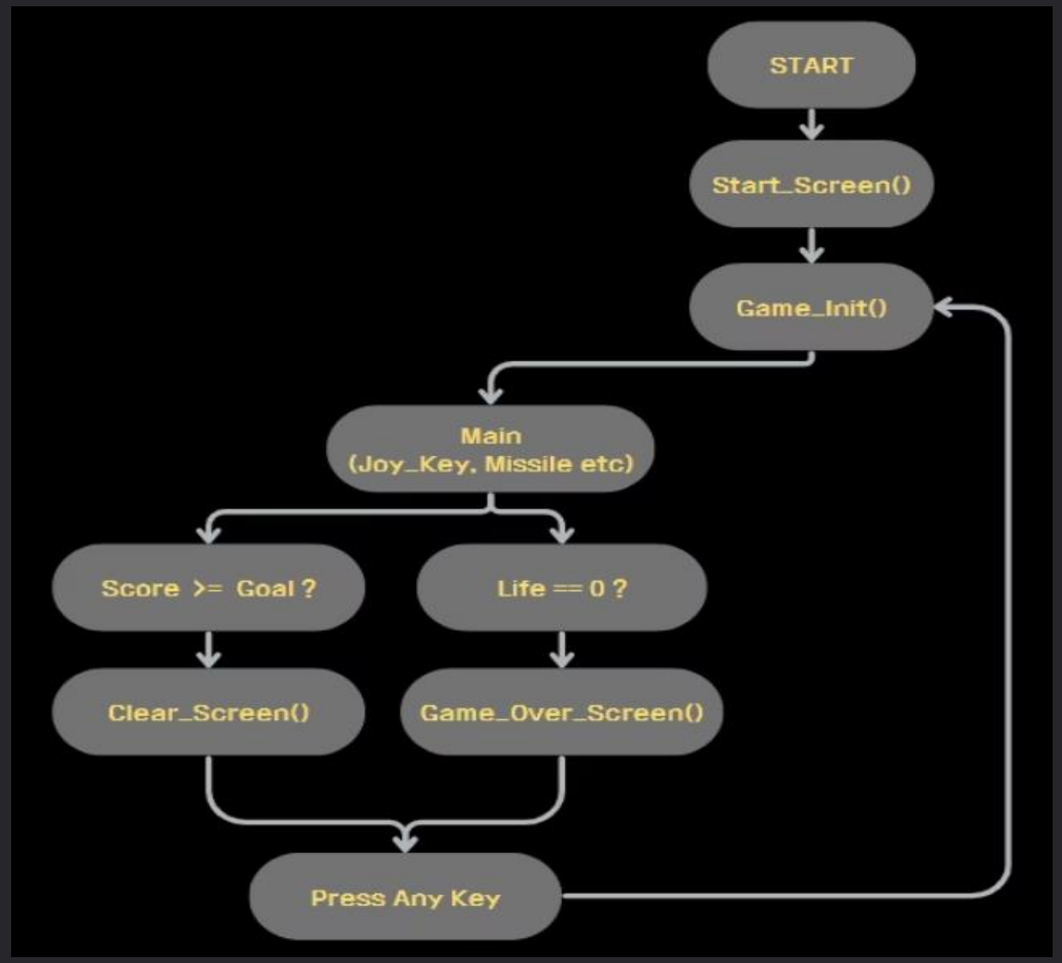
개발 결과

주요 함수 설명 표

함수명	역할
`Main()`	전체 게임 루프
`Game_Init()`	게임 상태 초기화
`Restart_BGM()`	배경음 재시작
`Player_Move()`	조이스틱 이동 처리
`Check_Collision()`	미사일, 아이템과의 충돌 확인
`Update_Missiles()`	미사일 이동 / 삭제
`Update_Item_Move_And_Draw()`	아이템 생성 및 이동 함

개발 결과

Flow Chart



상태전이도

핵심 기술

1. 충돌 판정 시스템

문제

• 미사일과 플레이어가 겹치는 순간 판정이 불안정함

해결 방법

• AABB (Axis-Aligned Bounding Box) 방식 적용

• x축, y축 별로 분리해서 충돌 판정

추가 해결

• 무적 상태 시 충돌 무시

• life 감소 이후 미사일 비활성화를 통해 중복 감소 방지

핵심 기술

2. 상태 기반 아이템 처리

아이디어

• 아이템 종류를 나눠 무적, 생명 효과 제공

핵심 설계

• item.type을 도입하여 조건 분기

• 아이템 색상으로 효과 구분

난제 해결

• 아이템을 먹은 뒤에도 잔상이 남는 문제

-> 배경색으로 지우고 다시 그림

한 함수에서 아이템 이동, 충돌, 효과 모두 관리

핵심 기술

3. BGM 타이밍

문제

- BGM이 시작할 때 '드르르르' 잡음 발생

해결

- Play_BGM_Init() + Play_BGM_Lock(1) → 안정화된 음정 출력까지 대기
- TIM3_Repeat_Interrupt_Enable() 호출 위치 조정

구성

- 16비트 타이머 기반, 음계별 주기 출력, 루프 재생생

핵심 기술

4. 생명 시스템과 UI 출력

설계 목표

- 시각적으로 직관적인 생명 표현

기술 요소

- 생명 수만큼 하트 이미지 출력
- Life_Update() 함수로 중앙 집중 관리

업그레이드 아이디어

- 생명 초과 제한, 깜빡임 효과 등 추가 가능

핵심 기술

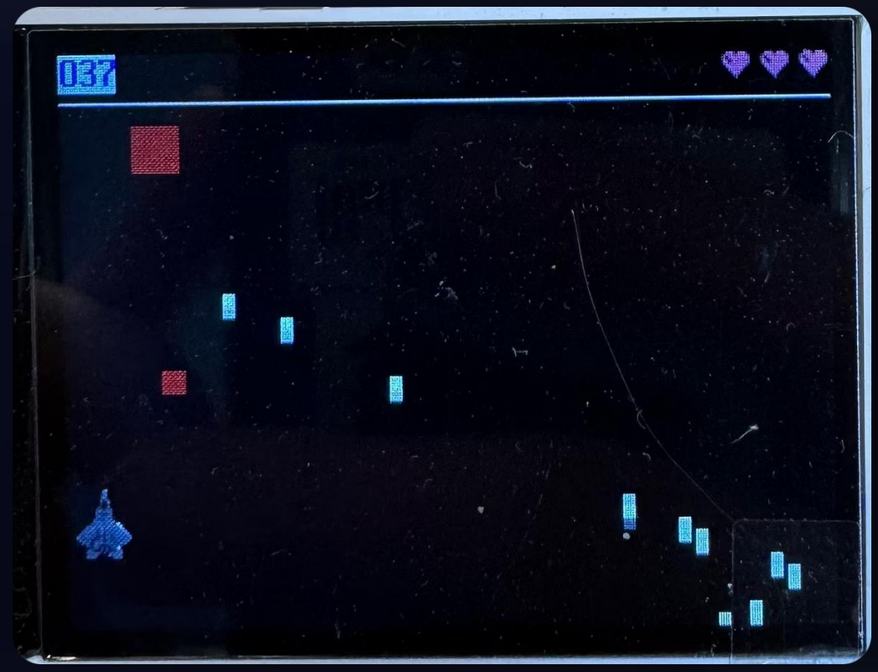
5. UI 개선 (이미지 기반, 선 추가 등)

문제

- 기존에는 모든 그래픽이 색상만 다른 네모 박스

기술 요소

- 플레이어 이미지 출력 함수 사용
- 하트 이미지 출력
- 화면 상단 UI 선 그리기 (Lcd_Put_Pixel() 기반)



핵심 코드

1. 게임 상태 흐름 핵심 - Main ( ) 루프

요약 설명:

• 전체 게임 흐름을 구성하는 메인 루프

• 초기화 → 시작화면 → 게임 루프 → 게임 오버 흐름으로 순환

핵심 코드 (요약)

```
for(;;){
    Game_Init();
    TIM4_Repeat_Interrupt_Enable(1, TIMER_PERIOD*10);
    TIM3_Repeat_Interrupt_Enable(1, 100);
    Play_BGM_Lock(0);

    while (!game_over) {
        Score_Update();
        if (Jog_key_in) player_Move();
        if (item.active) Update_Item_Move_And_Draw();
        if (TIM4_expired) Game_Tick_Handler();
    }
    Game_Over_Screen();
}
```

포인트:

• game_over 상태를 기준으로 for 루프 구성

• 타이머/인터럽트에 의한 게임 요소 분리

핵심 코드

2. 충돌 판정 처리 - Check_Collision( )

요약 설명:

- 플레이어와 미사일 충돌 감지
- 무적 상태 고려, 생명 감소 로직 포함

핵심 코드 (요약)

```
for (i = 0; i < MAX_MISSILES; i++) {
  if (!missiles[i].active) continue;

  if (충돌 발생) {
    if (!invincible) {
      life--;
      if (life <= 0) return GAME_OVER;
    }
    missiles[i].active = 0; // 미사일 무력화
  }
}
```

핵심 코드

3. 아이템 로직 - Update_Item_Move_And_Draw ( )

요약 설명:

• 아이템 이동, 화면 출력, 충돌 감지용

핵심 코드 (요약)

```
item.y += 속도;

if (item.active) {
   if (충돌) {
     item.active = 0;
     if (item.type == ITEM_INVINCIBLE) invincible = 1;
     else if (item.type == ITEM_LIFE) life++;
       Life_Update();
   }
}
```

포인트:

• 아이템 종류 분기 처리 (ITEM_LIFE, ITEM_INVINCIBLE)

• 충돌 후 효과 변화

핵심 코드

4. BGM 재생 - Play_BGM_Run ( )

요약:

• TIM3 인터럽트를 통한 주기적 BGM 음정 출력

핵심 코드 (요약)

```
if (my_bgm_lock) return;
    my_bgm_tick++;

    if (my_bgm_tick * 10 >= cur_dur) {
        my_bgm_index++;
        if (my_bgm_index >= my_bgm_len) my_bgm_index = 0;
        my_bgm_tick = 0;
        TIM3_Out_Freq_Generation(tone_table[cur_note]);
    }
```

포인트

• bgm_tick에 따라 음계 타이밍 결정

• 타이머 인터럽트 기반 BGM 순환 재생생

핵심 코드

5. 생명 UI 표시 - Life_Update ( )

요약:

- 플레이어 생명 수에 따라 하트 이미지 출력
- 위치 계산 및 이미지 출력 함수 활

핵심 코드 (요약)

```
for (i = 0; i < 3; i++) {
        int x = LCDW - heart_w - i * (heart_w+2);
        int y = 0;
        Lcd_Draw_Box(x, y, heart_w, heart_h, color[BACK_COLOR]);
    }
for (i = 0; i < life; i++) {
                    int x = LCDW - heart_w - i * (heart_w+2);
        int y = 0;
                    Lcd_Draw_Image(x, y, heart_small.width, heart_small.height, heart_small.data);
    }
```

포인트

- 게임 몰입도를 높이는 시각적 디테일

결론

프로젝트 성과

이번 프로젝트를 통해 강의시간동안 배웠던 다양한 기능을 통합하여 원하는 형태의 게임을 완성할 수 있었다는 점이 뿌듯했습니다.

아쉬운 사항

게임의 추가적인 디테일을 구현하지 못한 점이 아쉬웠습니다.

업그레이드 아이디어

점수에 따른 난이도 조정, 보스 스테이지 추가, 다양한 아이템과 적 패턴 구현 등을 고려할 수 있습니다.

개발 후기

느낀 점

처음엔 단순히 "이렇게 하면 되지 않을까? 생각보다 쉽겠는데?"라고 생각했지만, 실제 구현은 생각보다 복잡하고 타이밍 조절, 상태 전이, UI 흐름 까지 세심한 설계가 필요하다는 것을 느꼈습니다. 실수하고 디버깅하는 과정을 통해 진짜 내 손으로 만든 게임이라는 자부심이 생겼습니다.

과제를 통해 얻은 점

TIMx, UART, GPIO, LCD, 인터럽트 등을 직접 제어하면서 통합적으로 설계하는 방법을 익힐 수 있었습니다. 삽질을 하면서도 스스로 원인을 추적해 해결한 경험이 가장 값졌습니다.

아쉬운 점

이미지를 다양하게 넣고 싶었지만 메모리·속도 한계로 하지 못한 것이 아쉬웠습니다. 다양한 적의 패턴이나 스테이지별 난이도 증가 같이 다양한 기능을 넣고 싶은 욕심이 생겼지만 구현하지 못해 아쉬움이 남습니다.

삽질의 추억

플레이어가 미사일과 충돌 시 life감소가 두 번 발생하여 즉사하는 문제가 발생했습니다. 3시간넘게 문제를 붙잡고 있었는데 알고보니 충돌한 미사일을 비활성화하지 않아 다음 루프에서 또 life 감소가 발생한 것이었습니다. 다른 c파일에서 초기화한 줄 알고 확인을 안 했다가 충돌 후 처리 누락이라는 단순한 실수였습니다. 너무 허탈했지만 더 꼼꼼히 코드를 작성해야겠다는 생각이 들었습니다.