게임 제작 프로젝트

- 교육 과정 명: ARM 아키텍처
- 프로젝트 명: 시골 쥐 도시에 오다!
- 이름: 장찬원

목차

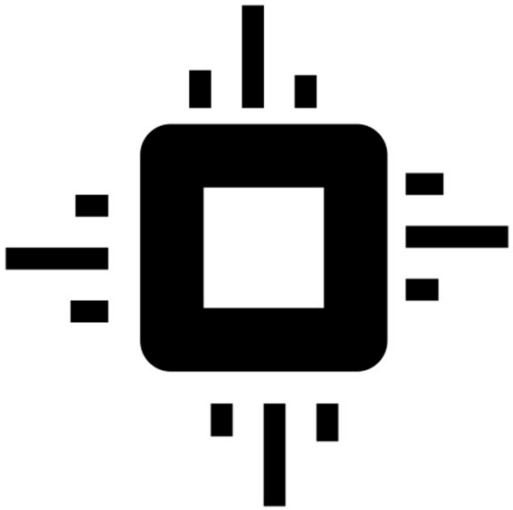
과제 개요

게임 컨셉

"시골쥐 도시쥐" 이야기를 모티브한 게임

게임 설명

플레이어는 시골쥐가 되어 고양이와 올빼미를 피해 다가오는 치즈를 먹으며, 가능한 한 많은 점수를 획득해야 합니다.



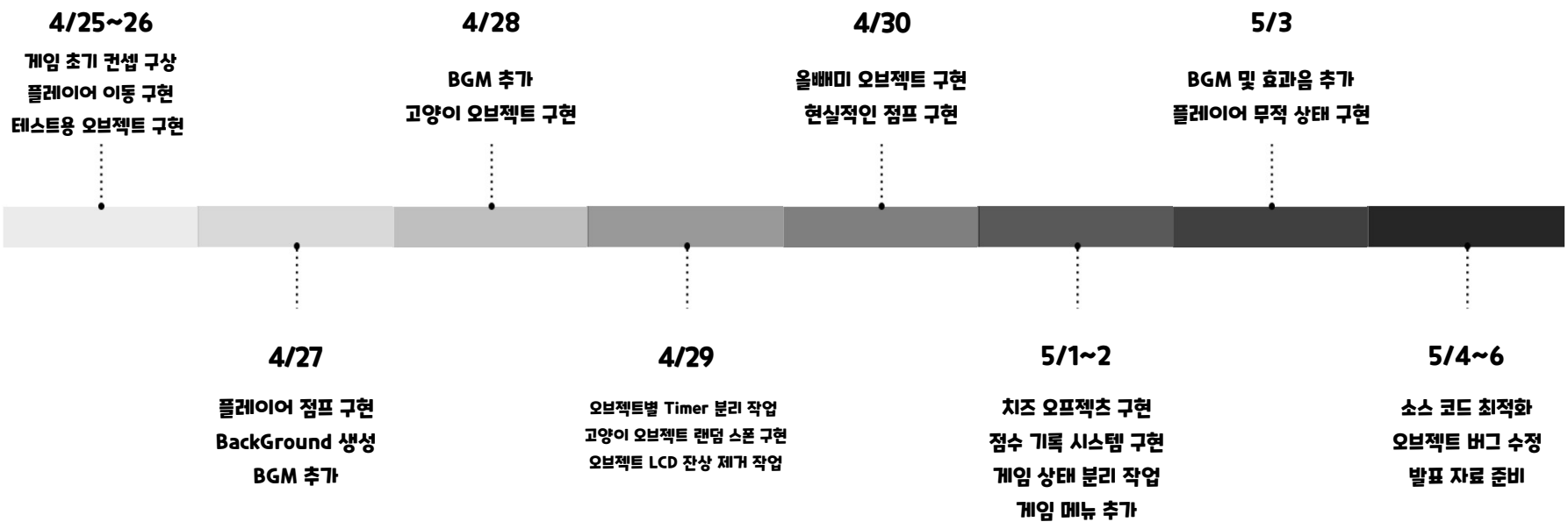
장비 소개

프로세서: 윌텍, Cortex-M3 MiNi

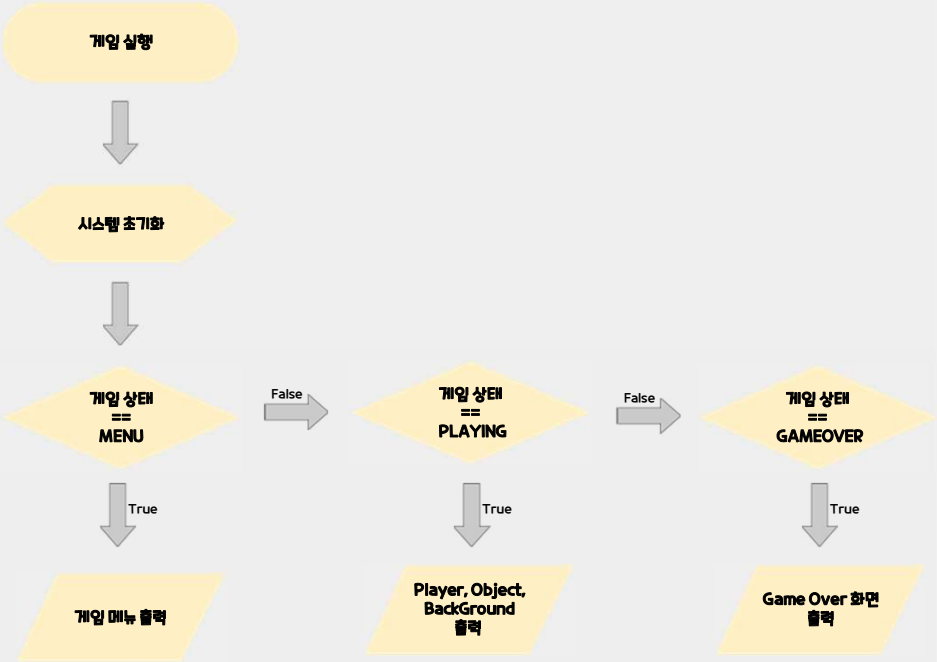
실습 환경

Visual Studio code
C 프로그래밍 언어

개발 일정


4/25~26
게임 초기 컨셉 구상
플레이어 이동 구현
테스트용 오브젝트 구현
4/27
플레이어 점프 구현
BackGround 생성
BGM 추가
4/28
BGM 추가
고양이 오브젝트 구현
4/29
오브젝트별 Timer 분리 작업
고양이 오브젝트 랜덤 스폰 구현
오브젝트 LCD 잔상 제거 작업
4/30
올배미 오브젝트 구현
현실적인 점프 구현
5/1~2
치즈 오프젝츠 구현
점수 기록 시스템 구현
게임 상태 분리 작업
게임 메뉴 추가
5/3
BGM 및 효과음 추가
플레이어 무적 상태 구현
5/4~6
소스 코드 최적화
오브젝트 버그 수정
발표 자료 준비

개발 결과



MENU 화면

PLAYING 화면

GAMEOVER 화면

게임 코드 분석

- **main() 함수에서 게임 상태에 따라 특정한 함수를 출력하도록 설계**

- **MENU 상태**

– Draw_Start_Menu() 함수, 시작 메뉴 lcd 출력

– Music_On() 함수, 메뉴 BGM 출력

– ResetGame() 함수, PLAYING 상태 전환 및 초기화

- **PLAYING 상태**

– MainLoop() 함수, Player-Object 스폰

– Game_Updata_All() 함수, Player-Object 실시간 위치 이동

– Player_Jump_Start() 함수, 플레이어 점프 실행

- **GAMEOVER 상태**

– Draw_GameOver_Screen() 함수, Game over 시 Lcd 출력

– ResetGame() 함수, SW1 입력 시 PLAYING 상태 전환

개발 결과

사용된 함수 목록

- 핵심 함수

- MainLoop()
  주기마다 게임 상태에 따라 화면, 사운드, 객체 등을 처리하는 핵심 함수입니다.

- 초기화 함수

- Game_Init()
  게임 시작 시 필요한 모든 초기 설정을 수행하는 함수입니다.

- Game_Updata_All()
  주기 마다 플레이어 이동, 충돌 검사, 아이템 처리, 점수·시간 갱신을 수행해 게임 상태를 업데이트 합니다.

- ResetGame()
  게임 상태를 초기화하고 화면을 되돌리며, 메뉴나 게임오버일 땐 이후 로직을 건너뛰는 역할입니다.

- GameOver_Cleanup()
  게임오버 시 LCD와 충돌 잔상을 완전히 지워 화면을 깔끔히 초기화 합니다.

- 객체 동작 함수

- Player_Jump_Start()
  Player가 땅에 있을 때만 점프 가능하게 점프 플래그를 설정 합니다.

- Player_Draw(color index)
  현재 Player의 위치에 그림을 그립니다.

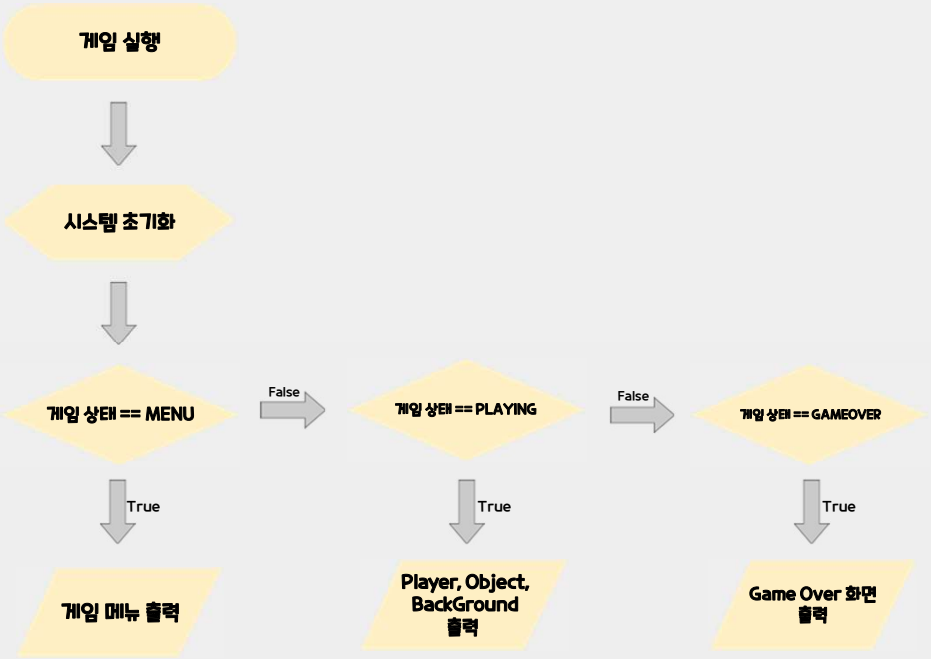
- Player_Update()
  Player 위치와 속도를 갱신하고, 바닥에 닿으면 위치 고정 및 점프 종료 처리합니다.

- Object_Init()
  Object(Cat, Owl, Cheese)를 초기화 합니다.

- Object_Updata()
  Object(Cat, Owl, Cheese)를 좌측에서 우측으로 이동시키고, 우측 끝에 도달하면 Object 지운 후 재생성 합니다.

- Object_Spawn()
  Object(Cat, Owl, Cheese) 최대 수를 넘지 않게 생성하는 역할입니다.

핵심 기술



- 시스템 초기화

```
System_Init();
Uart_Printf("Jump Game Start\n");
Lcd_Init(3);
Jog_Poll_Init();
Jog_ISR_Enable(1);
Uart1_RX_Interrupt_Enable(1);
Lcd_Clr_Screen();

// TIM2, TIM4: 주기 반복 인터럽트
TIM4_Repeat_Interrupt_Enable(1, TIMER_PERIOD);
TIM2_Repeat_Interrupt_Enable(1, 1);
TIM3_Out_Init();

game_state = GAME_STATE_MENU;
start_drawn = false;
gameover_drawn = false;
```

- 게임 상태 로직

```
for (;;)
{
    // 1) 게임 메뉴
    if (game_state == GAME_STATE_MENU)
    {…
    }

    // 2) 플레이 중
    else if (game_state == GAME_STATE_PLAYING)
    {…
    }

    // 3) 게임 오버
    else if (game_state == GAME_STATE_GAMEOVER)
    {…
    }
}
```

초기 문제점

- 가독성이 떨어져 코드 수정 및 추가 작업이 힘듦
- 모든 로직이 항상 실행되어 의도치 않은 버그 발생

게임 상태 알고리즘

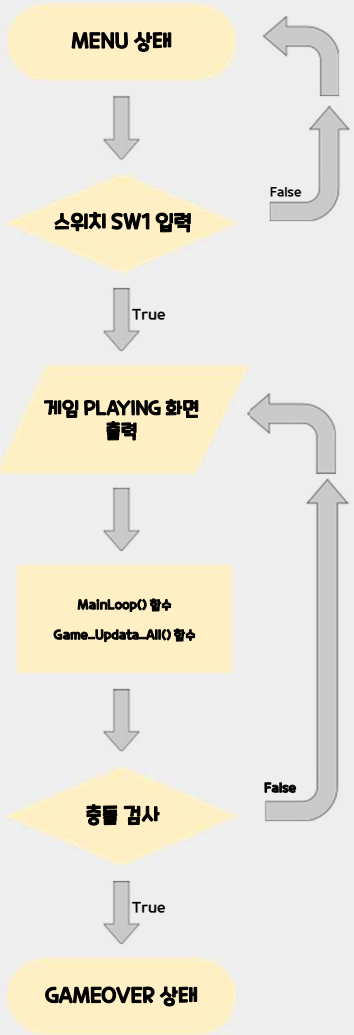
초기 시스템, 타이머, 인터럽트, 게임 설정 초기화 후

게임 상태에 따른 특정 함수 출력하는 알고리즘 설계

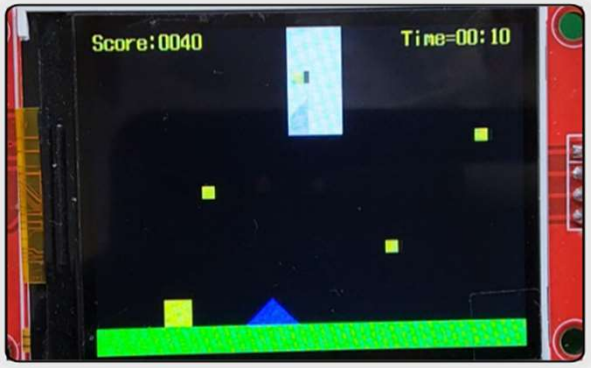
게임 상태 알고리즘 장점

- 게임 상태 코드에 따른 명확성
- 로직 분리로 인한 안정성
- 코드 유지보수 간편화
- 테스트 편의성

핵심 기술



PLAYING 화면



PLAYING 화면

PLAYING 상태 알고리즘

- MENU 상태에서 SW1 입력 시 PLAYING 상태 전환
- Player, Object 출력하는 함수 실행
- Player와 Object 간에 충돌 검사 실행
- 충돌 검사에 따른 게임 상태 변화

- **MainLoop() 함수 역할**

– PLAYING 상태일 때 Object Spawn

– MENU 상태, GAMEOVER 상태 LCD 출력

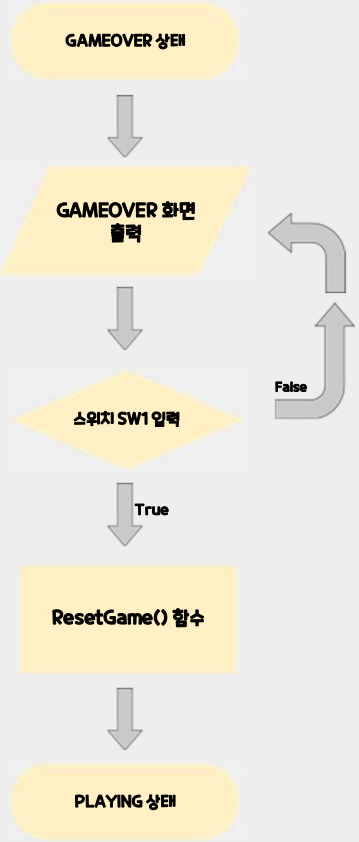
- **Game_Updata_All() 함수 역할**

– Player, Object 실시간 위치 이동

– Object 충돌 검사

– Score, Time 기능 LCD 출력

핵심 기술



GAMEOVER 화면

GAMEOVER 화면

GAMEOVER 상태 알고리즘

- Game_Updata_All() 함수를 통해 화면 출력
- 스위치 입력 시 ResetGame() 함수 실행
- PLAYING 상태 전환

- **ResetGame() 함수 역할**

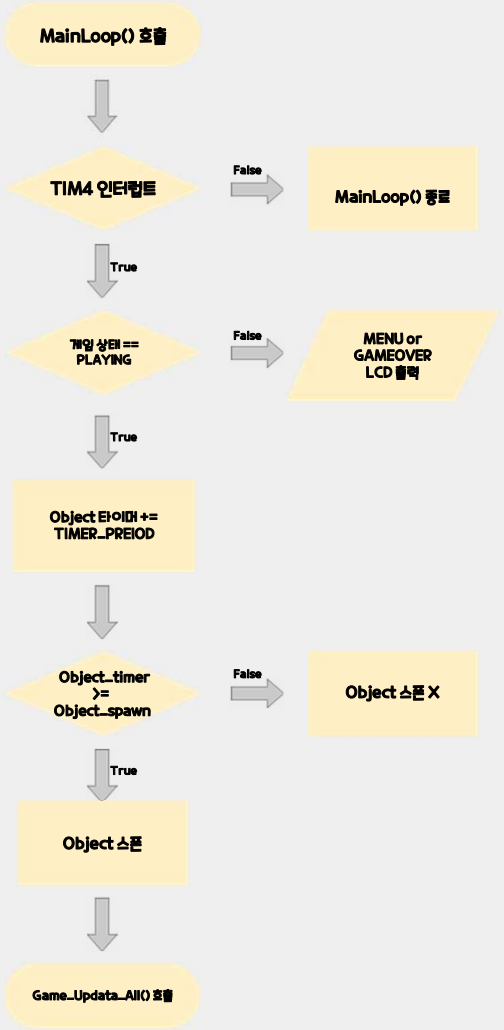
– 스폰, 이동 Timer 초기화

– Player 상태 초기화

– Score, Time 기능 초기화

– 모든 Object 비활성화

핵심 기술



초기 문제점

- 하나의 Timer 사용하여 Object가 서로 같은 움직임을 가짐
- Object 별로 개별 TIM2~4 사용함으로 Timer 기능 한계
- BGM 함수에 사용되는 TIM2~3와 서로 간섭 발생

S/W Timer 분리 알고리즘 장점

- Object 각각의 스폰 타이밍, 이동 등 별로 관리
- 다양한 Object 추가 가능
- BGM 작동 문제 해결

S/W Timer 분리 알고리즘

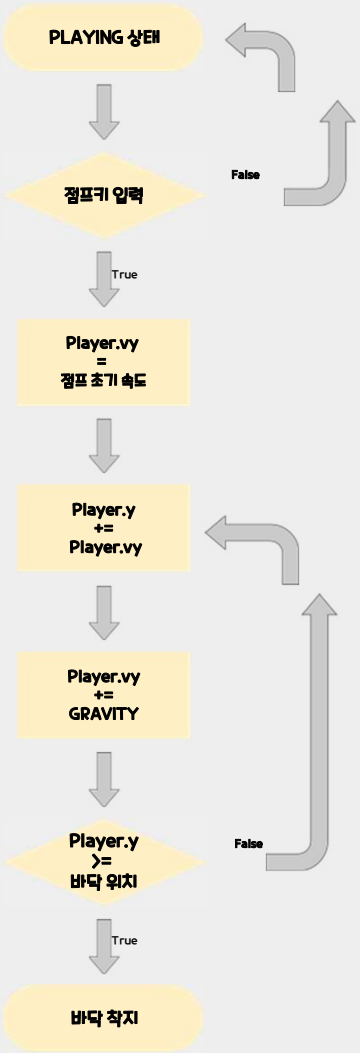
- TIM4는 Player 와 Object의 동작을 담당
- TIM4 인터럽트가 실행될 때 마다 루프를 진행 시킴
- TIM4 타임아웃 될 때 마다 Object 타이머에 Valuer 값 누적

```
// 타이머 누적
cat_spawn_timer  += TIMER_PERIOD;
owl_spawn_timer  += TIMER_PERIOD;
cat_move_timer   += TIMER_PERIOD;
owl_move_timer   += TIMER_PERIOD;
cheese_spawn_timer += TIMER_PERIOD;
elapsed_time_ms += TIMER_PERIOD;
```

- 조건 문을 통해 Object의 독립적인 주기 생성

```
// 스폰
if (cat_spawn_timer >= next_spawn_interval)
{
    cat_spawn_timer = 0;
    Cat_Spawn();
    next_spawn_interval = (rand() % 5000) + 500;
}

if(owl_spawn_timer >= next_owl_spawn)
{
    owl_spawn_timer = 0;
    Owl_Spawn();
    next_owl_spawn = (rand() % 5000) + 500;
}

if(cheese_spawn_timer >= CHEESE_SPAWN_INTERVAL)
{
    cheese_spawn_timer = 0;
    Cheese_Spawn();
}
```

핵심 기술



초기 문제점

- Player 점프 시 즉시 올라가고, 갑자기 떨어지는 형태
- 부자연스러운 모션, 불편한 조작감
- Object 회피 판정에 대한 어려움

물리 점프 알고리즘 장점

- 점프 정도 세밀한 조정 가능
- 자연스러운 점프로 인한 조작감 향상
- 명확한 Object 충돌 판정 가능

물리 점프 알고리즘

- 점프 시작 시

  Player.vy = -20 정의

  점프 시 다시 점프 할수 없게 플래그 생성

- 프레임마다 Player.y와 Player.vy 값 업데이트

  Player.y 위치: 위로 이동하다가 낙하

  Player.vy 속도: 2씩 증가 하다가 낙하로 전환

- 점프 후 바닥 착지 시

  Player.y 위치: 초반 위치로 고정

  Player.vy 속도: 0으로 초기화

  다시 점프할 수 있도록 플래그 초기화

핵심 코드

```
//충돌검사
static inline bool Check_Collision_Rect(
    int ax, int ay, int aw, int ah,
    int bx, int by, int bw, int bh)
    {
        return  (ax < bx + bw) &&
                (ax + aw > bx) &&
                (ay < by + bh) &&
                (ay + ah > by);
    }
```

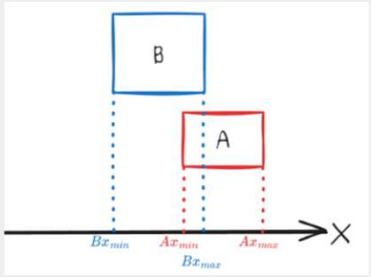
```
for (i = 0; i < MAX_CATS; i++)
{
    if (Cats[i].active &&
        Check_Collision_Rect(Player.x, Player.y, Player.w, Player.h,
                             Cats[i].x, Cats[i].y, Cats[i].size, Cats[i].size))
    {
        game_state = GAME_STATE_GAMEOVER;
        return;
    }
}

for (i = 0; i < MAX_OWLS; i++)
{
    if (Owls[i].active &&
        Check_Collision_Rect(Player.x, Player.y, Player.w, Player.h,
                             Owls[i].x, Owls[i].y, Owls[i].w, Owls[i].h))
    {
        game_state = GAME_STATE_GAMEOVER;
        return;
    }
}
```

AABB 충돌 감지

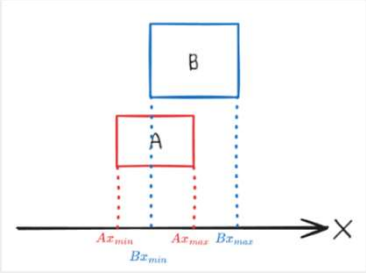
- Axis Aligned Bounding Box (축 정렬 직사각형)
- 2D 충돌 감지 방식 중 하나
- 두 사각형 A, B가 다음 조건에 만족하면 충돌 한다고 판단

코드 분석

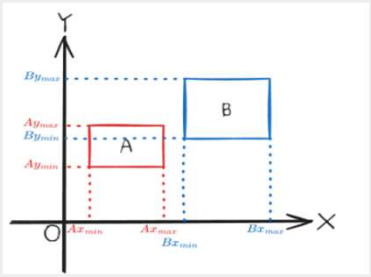
- ax < bx + bw == AxMin < BxMax



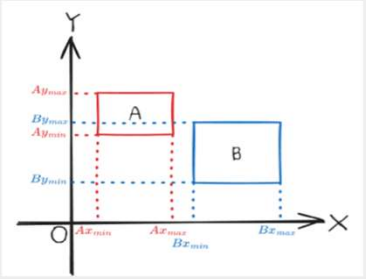
- ax + aw > bx == AxMax > BxMax



- ay < by + bh == AyMin < ByMax



- ay + ah > by == AyMax < ByMin

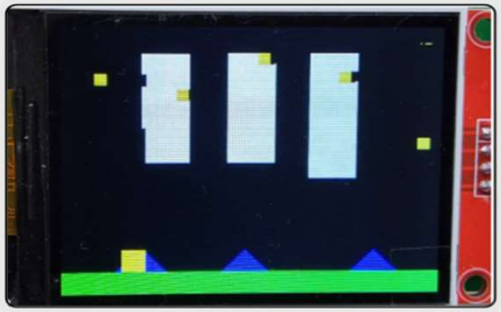


핵심 코드

```
void GameOver_Cleanup(void)
{
    Lcd_Draw_Box(0, 0, LCDH, LCDW, color[BACKGROUND_COLOR]);

    int i;
    for(i=0; i<MAX_CATS; i++)
    {
        if(Cats[i].active)
        {
            Lcd_Draw_Box(현재 Cats[i].위치 -> 검정색);
            Cats[i].active = 0;
        }
    }

    for(i=0; i<MAX_OWLS; i++)
    {
        if(Owls[i].active)
        {
            Lcd_Draw_Box(현재 Owl[i].위치 -> 검정색);
            Owls[i].active = 0;
        }
    }

    for(i = 0; i < MAX_CHEESE; i++)
    {
        if(Cheese[i].active)
        {
            Lcd_Draw_Box(현재 Cheese[i].위치 -> 검정색);
            Cheese[i].active = 0;
        }
    }
}
```

발견한 문제

- Timer로 실시간으로 움직이는 Object는 Lcd_Clr_Screen() 함수를 사용해도 Object가 계속 Lcd에 남아있는 문제 발견
- 독립적인 Timer 사용으로 인해 Lcd_Clr_Screen() 함수 효과 없음



문제 해결

- 각 Object 현재 위치를 배경색으로 덮는 GameOver_Cleanup() 함수 설계하여 GAMEOVER 시 남아있던 Object 잔상들 제거 성공

결론

게임 완성도

- 초기 계획한 게임과 비슷하게 나옴
- 스스로 생각하기에 완성도 70%정도 생각함

해결하지 못한 문제

- Player 점프 시 회전하는 애니메이션을 추가했을 때 Lcd 잔상 문제
- Lcd에 문장을 출력 후 Clear가 되지 않은 문제
- GAMEOVER 시 SW1 입력하지 않아도 재시작되는 문제

추가 아이디어

- 게임 경과 시간이 증가할 수록 난이도 증가
- 일반 스테이지 클리어 후 보스 스테이즈 추가
- 2단 점프, 엎드리기, 구르기 등 다양한 모션 추가

개발 후기

- 내가 구성했던 게임이 생각보다 잘 나와서 다행이라고 생각함
- Lcd에 문장을 출력 후 Clear가 되지 않은 문제를 해결하지 못해 아쉬움이 남았음
- Lcd에 문장 출력을 하기 위해 직접 함수를 제작했지만 graphics.c에 Lcd_print() 함수가 있다는 사실에 허탈감을 느꼈음
- AABB 충돌 검사에 대해 어렵게 생각했지만 알고보니 쉬운 이론이었음
- 스스로에게 알고리즘 설계 능력이 많이 부족하다는 것을 느낌