

AI 시스템반도체 설계 (2기)

STM32
CORTEX-M3
LCD 게임 프로젝트

엄 찬 하

목차

과제 개요

개발 일정

개발 결과

핵심 기술

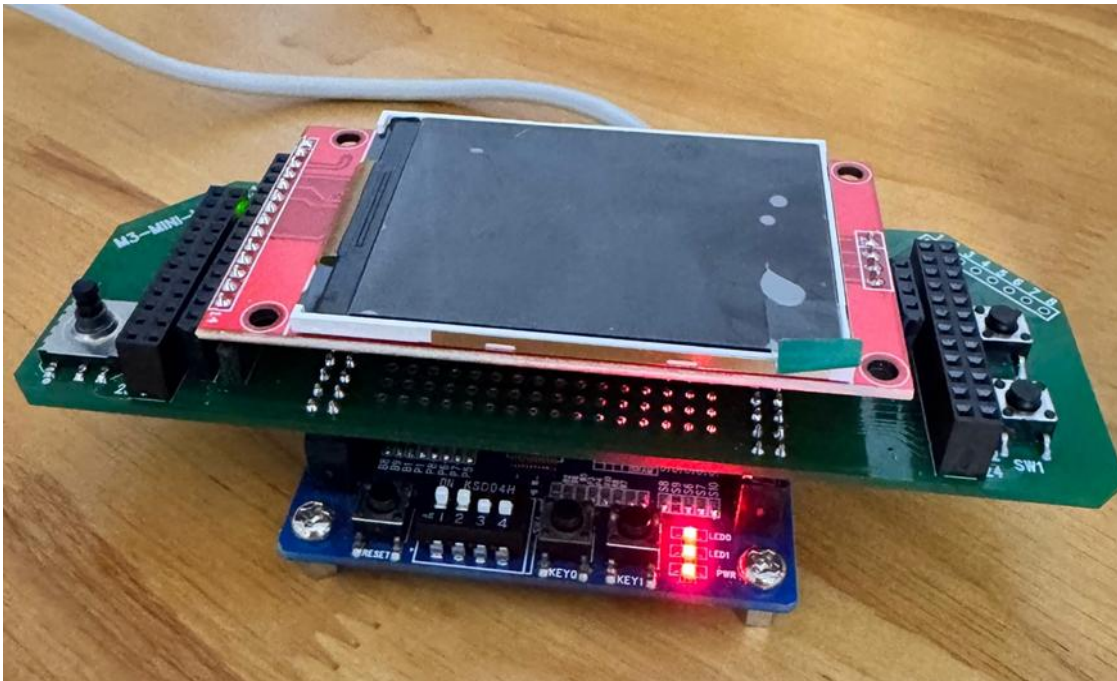
핵심 코드

결론

개발 후기

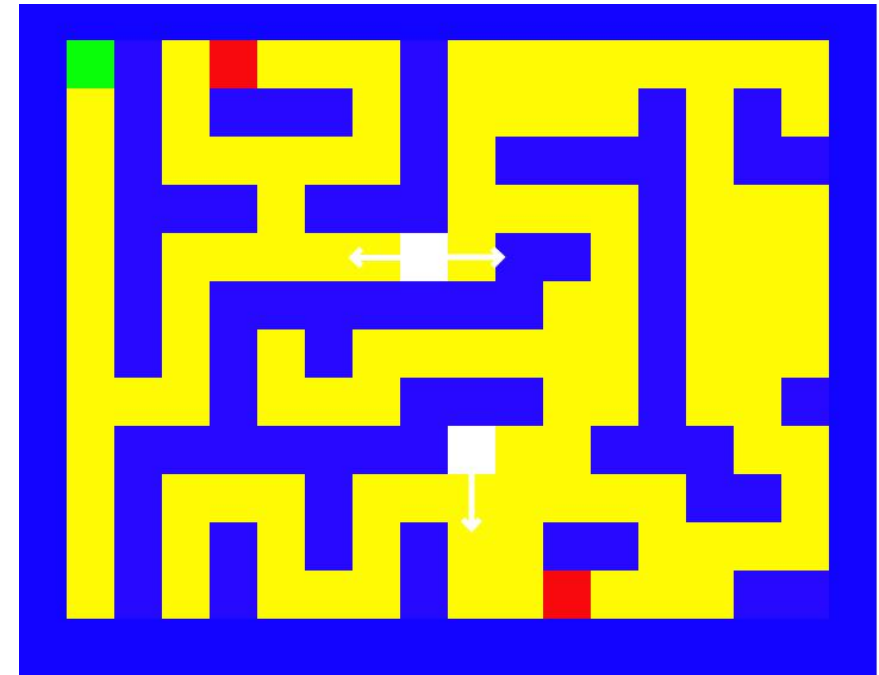
과제 개요 <루팡:경찰과 도둑>

실습 장비 및 환경

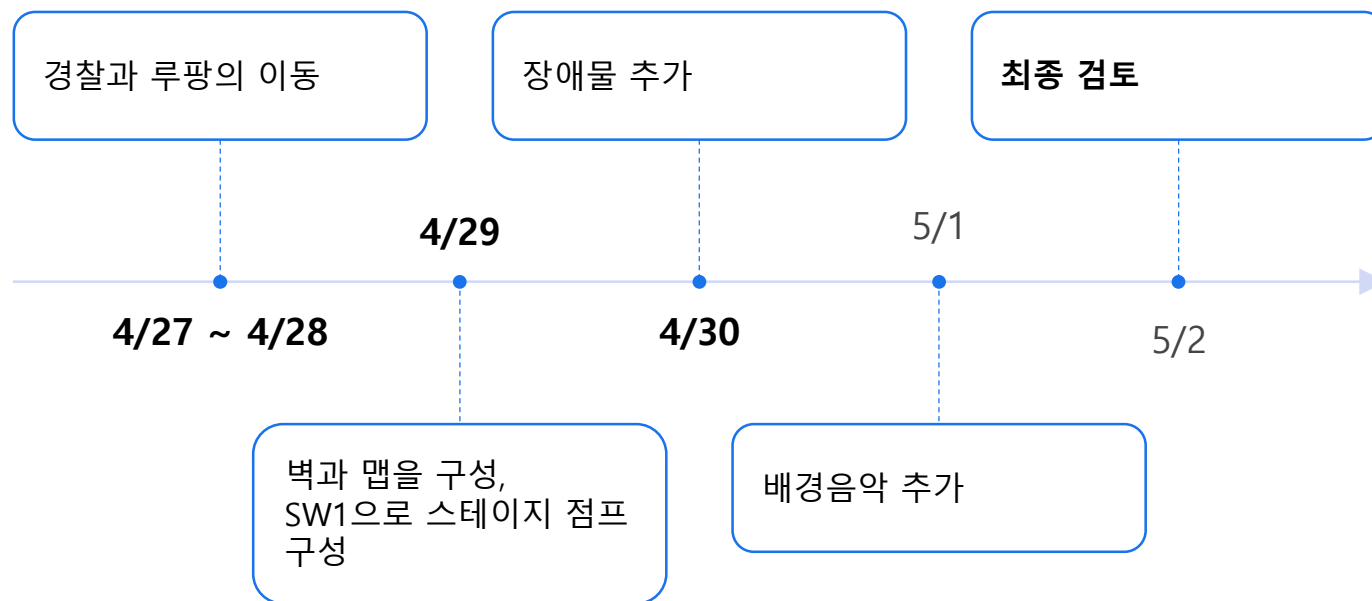
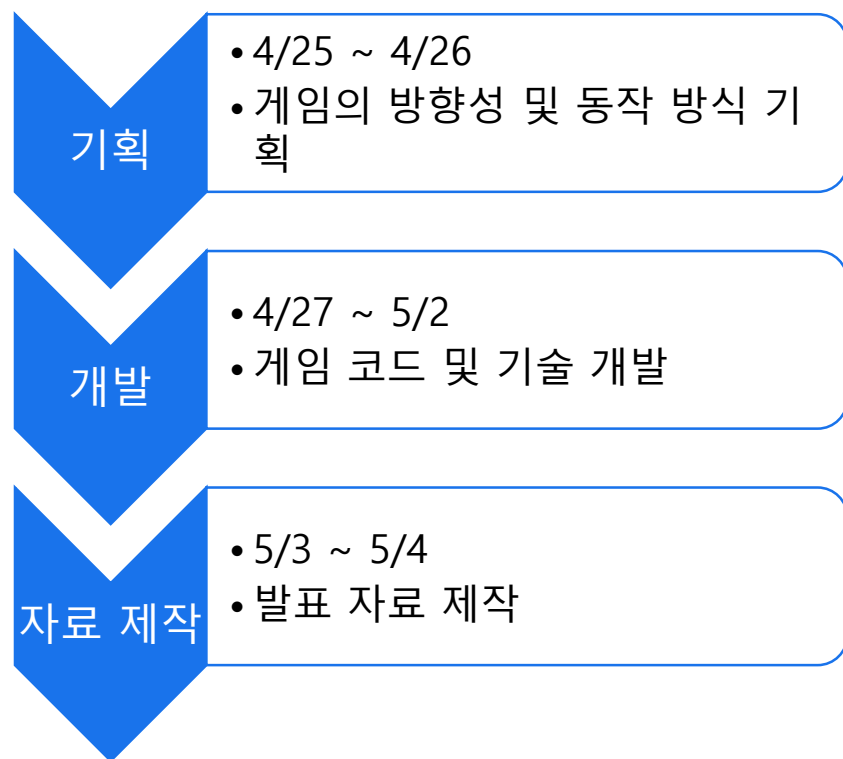


루팡(녹색)이, 장애물(흰색)과 경찰(빨강)을 피해 노랑(포인트)를 모두 먹는 게임!!

게임 설명

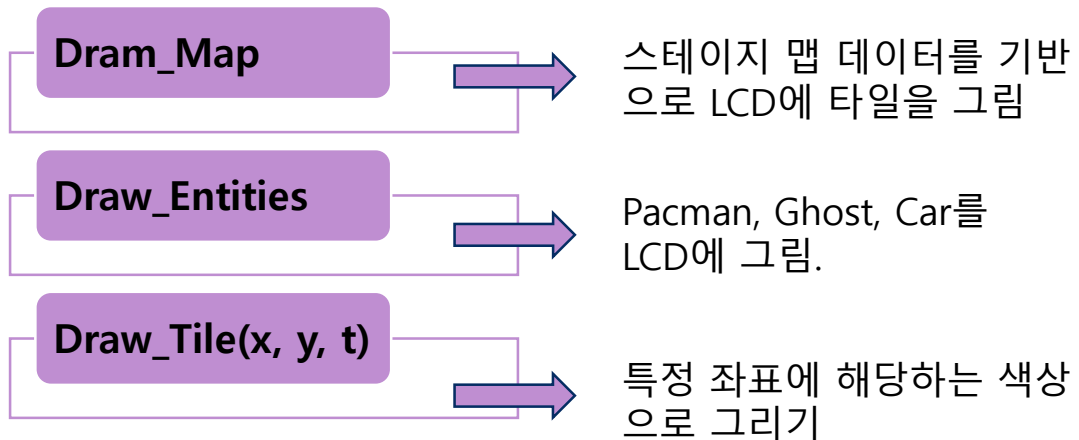


개발 일정

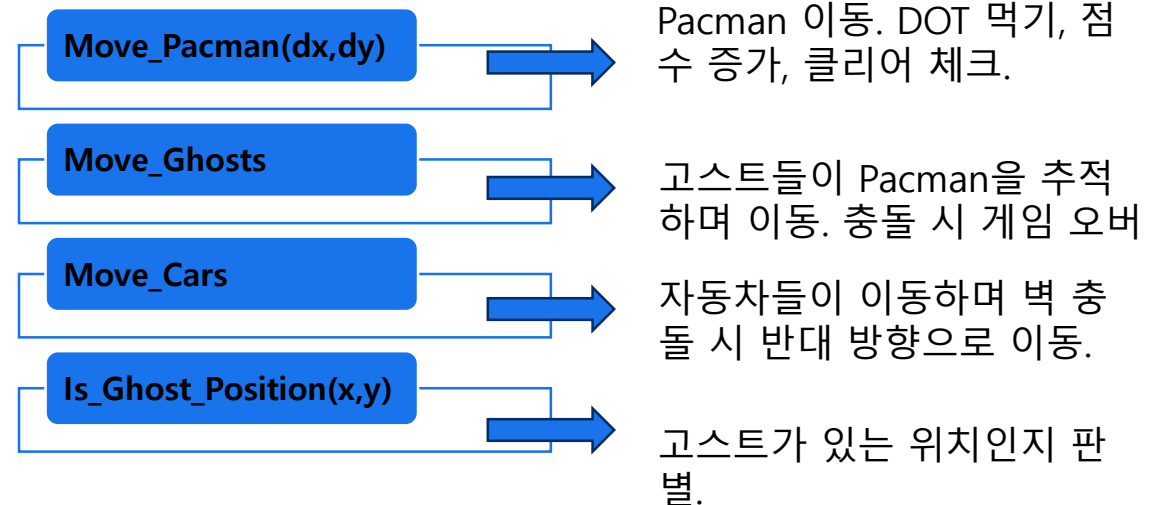


<개발 결과> 1.함수 목록

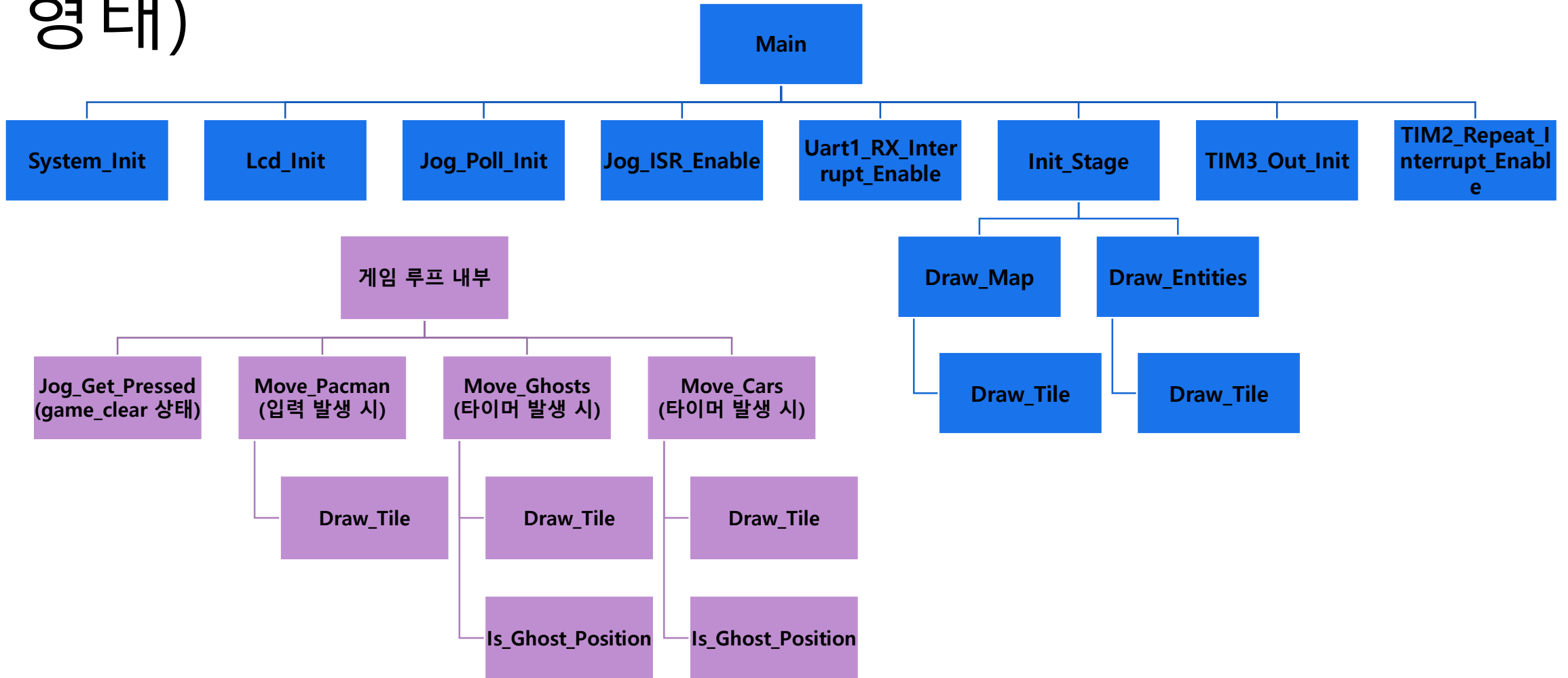
맵 그리기



몹 이동



<개발 결과> 2.함수 호출 관계 (호출 트리 형태)



<개발 결과> 3. 영상 및 Flowchart

맵 초기화

Init_Stage

게임 루프 반복

조그 입력 발생?

타이머 발생?

게임 상태 출력

세부 내용 판별

Pacman 이동

Ghost 이동

Car 이동

충돌 여부 확인 (게임 오버 or 클리어)

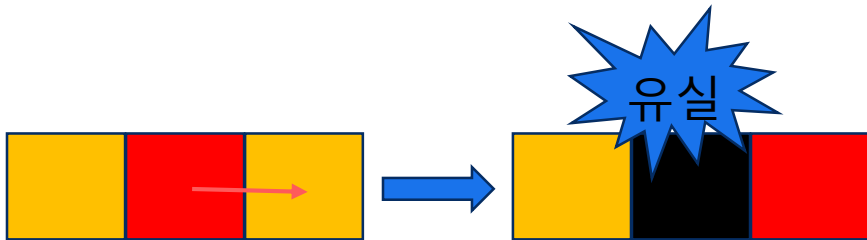
클리어 or out 판별

스테이지 이동 or 아웃

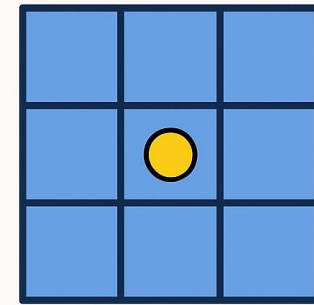


핵심 기술 1 -> 물체의 이전 상태 기억

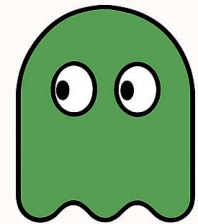
- 문제: 경찰(Ghost), 장애물(Car)가 지나간 자리에 DOT 유령이 생성되는 문제
- 해결: ghost_under[] 배열로 이전 상태 기억
- 효과: 시각적 정합성 유지, DOT 소멸 방지



각 유령마다 ghost_under에
직전 자리의 타일 정보를 저장하여



DOT

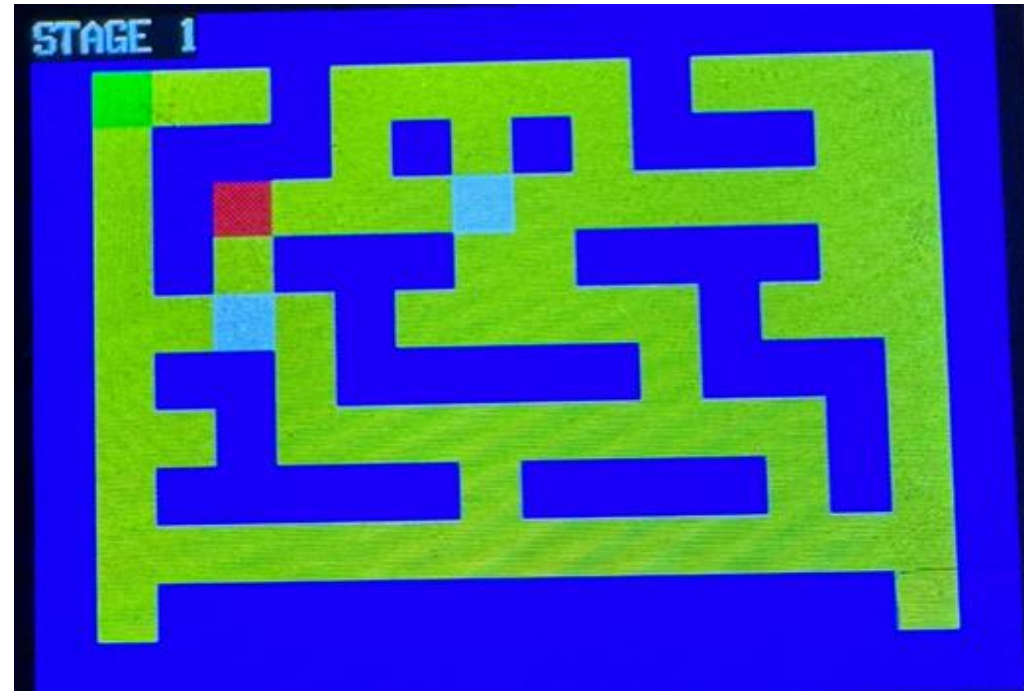


GOST

- 현재 위치: 유령이 위치한 타일 GHOST
- 직전 자리: 유령이 이동하기 전에 위치한 타일
- ghost_under: DOT 유령 생성 직전의 타일

핵심 기술 2 -> 맵 그리기

```
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},  
{1,2,2,2,1,2,2,2,2,2,1,2,2,2,1},  
{1,2,1,1,1,2,1,2,1,2,1,1,1,2,2,1},  
{1,2,1,2,2,2,2,2,2,2,2,2,2,2,1},  
{1,2,1,2,1,1,1,2,2,1,1,1,1,2,2,1},  
{1,2,2,2,2,1,2,2,2,2,2,1,2,2,2,1},  
{1,2,1,1,2,1,1,1,1,1,2,1,1,1,2,1},  
{1,2,2,1,2,2,2,2,2,2,2,2,2,1,2,1},  
{1,2,1,1,1,1,1,2,1,1,1,1,2,1,2,1},  
{1,2,2,2,2,2,2,2,2,2,2,2,2,2,1},  
{1,2,1,1,1,1,1,1,1,1,1,1,1,1,2,1},  
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
```

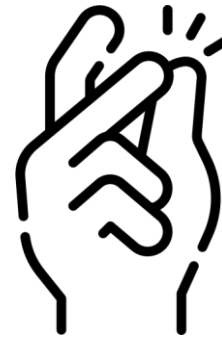


핵심 기술 3 -> 스테이지별 난이도 설정

- 문제: 난이도 조절을 하드코딩하면 유지보수 어려움
- 해결: stage_ghost_count[], ghost_speed 배열 활용
- 효과: 동적 난이도 조절, 스테이지 확장 용이

```
const int stage_ghost_count[MAX_STAGE] = {1, 2, 2, 3, 3};

int ghost_speed = 800;
if (stage == 3 || stage == 5) {
    ghost_speed = 533;
}
TIM4_Repeat_Interrupt_Enable(1,
ghost_speed);
```

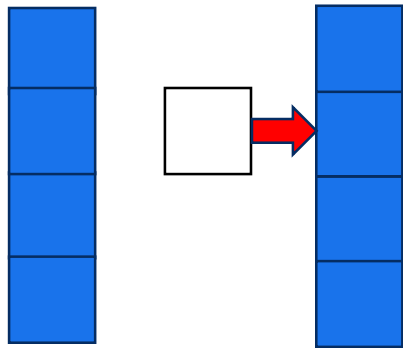


{1, 3, 5, 7, 9}

Ghost_Speed = 10000

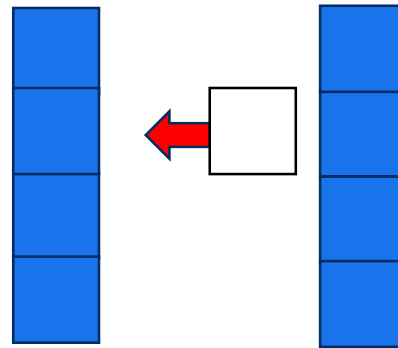
핵심기술 4 -> X, Y 자동차 이동

- 문제: 단순 이동체도 충돌 및 벽 감지를 해야 함
- 해결: 방향 배열 (`car_dir_x/y[]`) + 벽 감지 후 방향 반전
- 효과: 최소한의 로직으로 장애물 회피 구현



만약 벽을 만난다면??

`if (map[ny][nx] == WALL)`



방향 반전

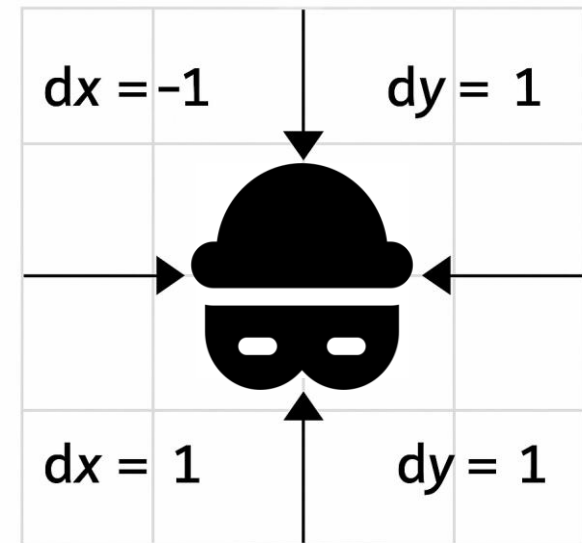
```
car_dir_x[i] = -car_dir_x[i];  
car_dir_y[i] = -car_dir_y[i];
```

반대 방향으로 위치 계산

```
nx = cars[i].x + car_dir_x[i];  
ny = cars[i].y + car_dir_y[i];
```


핵심 기술 5 -> 경찰(Ghost)의 루팡 추적

- 문제: 실시간으로 움직이는 Pacman을 추적
- 해결: 좌표 비교 기반의 단순 직선 추적 로직 사용
(pacman.x/y와 ghost.x/y를 비교해 한 칸씩 이동)
- 효과: 간단한 조건문만으로 추적 효과 구현
- 추후 AI 고도화 가능성 확보 (랜덤/회피 등으로 확장 가능)



핵심 기술 6 -> 객체 간 상호작용 규칙 (충돌,

통과)

상호작용 대상

루팡 ↔ Ghost

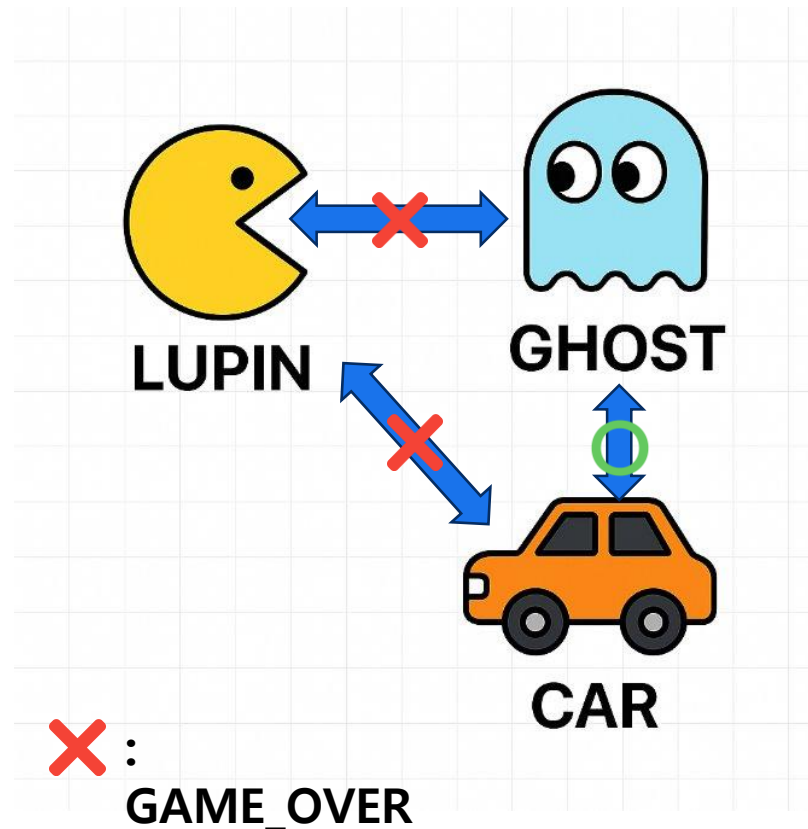
루팡 ↔ Car

Car ↔ 벽(WALL)

Ghost ↔ 벽(WALL)

Ghost ↔ Car

Car ↔ Ghost



충돌 결과 / 통과 여부

충돌 시 game_over = 1 → 게임 종료

충돌 시 game_over = 1 → 게임 종료

벽에 부딪히면 이동 방향 반전

벽은 통과 불가. 이동 제한됨

통과 가능! Car와 같은 타일에 위치 가능

무시 (Car는 Ghost 고려 안 함)

핵심 기술 7 -> FSM 구조 적용

- 문제: 다양한 게임 상황(클리어, 게임 오버 등)을 직관적으로 구분하고 처리하기 어렵고, 각 조건마다 분기문이 복잡해져 가독성이 떨어짐
- 상태를 FSM 방식으로 분리하여 관리
state 변수를 중심으로 조건 분기 → DOT 모두 먹으면 STAGE_CLEAR, SW1 입력 시 다음 스테이지로 전이



핵심 기술 8 -> 배경음악



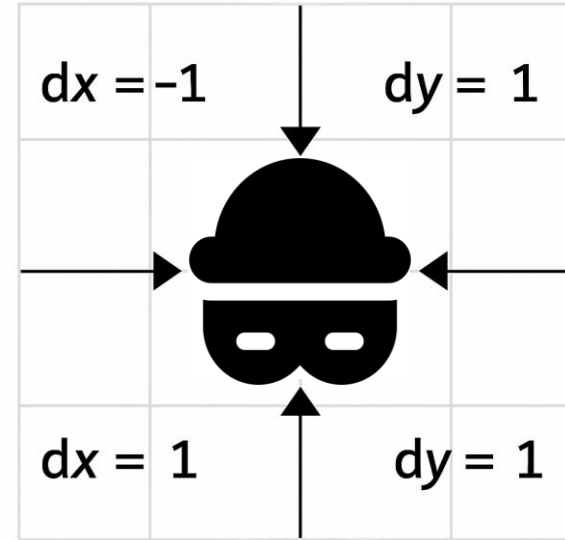
Timer 4 -> 게임 동작



Timer 2 -> 음악 재생

핵심 코드 1 -> 추적

- Move_Ghosts()의 직선 추적 알고리즘
- if (ghosts[i].x < pacman.x) dx = 1;
- else if (ghosts[i].x > pacman.x) dx = -1;
- else if (ghosts[i].y < pacman.y) dy = 1;
- else if (ghosts[i].y > pacman.y) dy = -1;



- 고스트는 루팡의 현재 위치 (pacman.x, pacman.y)와 자기 위치 (ghosts[i].x, ghosts[i].y)를 비교합니다.



핵심 코드 2 -> 다방향 장애물

```
if (map[ny][nx] == DOT || map[ny][nx] == EMPTY)
{
    cars[i].x = nx;
    cars[i].y = ny;
}
```



DOT 또는 빈 공간만 이동 가능

```
if (!Is_Ghost_Position(cars[i].x, cars[i].y)){
    Draw_Tile(cars[i].x, cars[i].y, CAR);
}
```



이동 후 LCD에 자동차 타일 그림 (유령이 있는 곳은 무시)

```
if (cars[i].x == pacman.x && cars[i].y ==
pacman.y)
    game_over = 1;
```



루팡과 충돌하면 게임 종료

핵심 코드 3 -> 게임 세팅, 맵 제작

```
void Draw_Map(void) {
    int x, y;
    total_dots = 0;
    for (y = 0; y < MAP_HEIGHT; y++) {
        for (x = 0; x < MAP_WIDTH; x++) {
            map[y][x] = maps[stage-1][y][x];
            Draw_Tile(x, y, map[y][x]);
            if (maps[stage-1][y][x] == DOT)
                total_dots++;
        }
    }
    int i;
    for (i = 0; i < stage_ghost_count[stage-1]; i++) {
        int gx = ghosts[i].x;
        int gy = ghosts[i].y;
        ghost_under[i] = (maps[stage-1][gy][gx] == DOT) ? DOT : EMPTY;
    }
}
```

확장성과 재사용성

새로운 맵 추가 시 배열 한 블록만 더
추가하면 코드 수정 없이 무한히 스테이지
확장 가능

```
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
{1,2,2,2,1,2,2,2,2,2,1,2,2,2,1},
{1,2,1,1,1,2,1,2,1,2,1,1,1,2,2,1},
{1,2,1,2,2,2,2,2,2,2,2,2,2,2,1},
{1,2,1,2,1,1,1,2,2,1,1,1,1,2,2,1},
{1,2,2,2,2,1,2,2,2,2,2,1,2,2,2,1},
{1,2,1,1,2,1,1,1,1,1,2,1,1,1,2,1},
{1,2,2,1,2,2,2,2,2,2,2,2,2,1,2,1},
{1,2,1,1,1,1,1,2,1,1,1,1,2,1,2,1},
{1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
{1,2,1,1,1,1,1,1,1,1,1,1,1,1,2,1},
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
```

핵심 코드 3 -> 게임 세팅, 맵 제작

```
void Draw_Map(void) {
    int x, y;
    total_dots = 0;
    for (y = 0; y < MAP_HEIGHT; y++) {
        for (x = 0; x < MAP_WIDTH; x++) {
            map[y][x] = maps[stage-1][y][x];
            Draw_Tile(x, y, map[y][x]);
            if (maps[stage-1][y][x] == DOT)
                total_dots++;
        }
    }
    int i;
    for (i = 0; i < stage_ghost_count[stage-1]; i++) {
        int gx = ghosts[i].x;
        int gy = ghosts[i].y;
        ghost_under[i] = (maps[stage-1][gy][gx] == DOT) ? DOT : EMPTY;
    }
}
```

맵 복사 + 타일 그리기 +
DOT 카운트

```
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
{1,2,2,2,1,2,2,2,2,2,1,2,2,2,2,1},
{1,2,1,1,1,2,1,2,1,2,1,1,1,2,2,1},
{1,2,1,2,2,2,2,2,2,2,2,2,2,2,2,1},
```

유령 하부 타일 정보 초기화

```
{1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
{1,2,1,1,1,1,1,1,1,1,1,1,1,1,2,1},
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
```


핵심 코드 4 -> 게임 동작

루팡이 DOT을 모두 먹어서
game_clear == 1인 상태

LCD 화면에 다음 메시지를 표시:
"STAGE CLEAR!"
"PRESS SW1 TO CONTINUE"

조이스틱의 SW1 버튼이 눌렸
는지 감지
다음 스테이지로 넘어감

모든 스테이지를 클리어했을
경우 무한 루프 진입 → 게임
완전 종료

새로운 맵을 불러오기 위해
Init_Stage() 호출

```
if (game_clear) {  
    Lcd_Printf(10, 100, GREEN, BLACK, 2, 2, "STAGE CLEAR!");  
    Lcd_Printf(10, 140, WHITE, BLACK, 1, 1, "PRESS SW1 TO CONTINUE");  
    int key = Jog_Get_Pressed();  
    if (key == 32) {  
        game_clear = 0;  
        stage++;  
        if (stage > MAX_STAGE) while(1);  
        Init_Stage();  
    }  
}
```

결론: 아쉬운 점 + 아이디어

UI 요소 개선: 점수, 타이머, 목숨 표시 추가 가능

사운드 다양화 (배경음악 + 효과음 분리)

스테이지 전환 연출 -> 스테이지 클리어 후 스크롤 애니메이션 또는 "Loading..." 화면

아이템 시스템 도입

- 🕒 **시계**: 유령 정지
- 🧨 **폭탄**: 벽 일부 제거
- 🌀 **포탈**: 해당 위치로 이동



포탈을 구현하고자 하였지만

시간부족 및 기술 부족으로 인해 실패!

결론: 아쉬운 점 + 아이디어

⊖ 현재: Ghost AI가 단순함 (Pacman 추적 직선 방식)

✓ 향후: 랜덤성 + 경로 탐색 (A*, DFS 등) 추가 고려

알고리즘

핵심 특징

DFS (깊이 우선 탐색)

가능한 경로를 끝까지 파고드는 방식. 빠르지만 최단 경로는 보장되지 않음

BFS (너비 우선 탐색)

한 칸씩 넓게 퍼지며 탐색. 최단 경로 보장, 느릴 수 있음

Dijkstra

가중치가 있는 그래프에서 최단 경로 찾음. A*의 기반

A* (에이스타)

휴리스틱(예상 거리) 사용하여 빠르고 정확하게 최단 경로 탐색

개발 후기

삽질: DOT 개수에 따른 Stage Clear 구현이 안되었다. 하지만 알고보니 매 스테이지마다 초기화하지 않아서 생긴 단순한 문제로 시간을 하루나 잡아먹었다. ㅁ

삽질2: 노래를 구현하는 과정에서 지속적인 배경음악인데 인터럽트로 굳이 제어를 하려다 시간을 많이 소모했다! 기존 과제를 참고하여 Timer.c에 추가하여 간단히 구현했다.

느낀점: 오히려 과제를 풀며 진행했던 난이도보다. 게임 구현 난이도가 더 쉬웠던 것 같다. 사실 구현 초기엔 두려움 때문에 낮은 난이도를 선택했다. 하지만 진행하고 보니 더 많은 시간 투자와 용기가 있었다면 충분히 4단계 레벨을 구현할 수 있었을 거라 확신한다!!

개발 후기

얻은 점!!!

1. 임베디드 시스템 전체 구조 이해

- MCU에서 입력 → 처리 → 출력 흐름이 어떻게 구성되는지 실전 경험으로 체득
- 타이머, 인터럽트, 메모리 맵, GPIO 동작 등 MCU 동작 방식에 대한 감각

2. 상태 기반 설계(FSM) 능력

- 게임 진행을 IDLE → PLAYING → STAGE_CLEAR → IDLE처럼 명확한 상태로 나눔
- 각 상태에 맞는 이벤트 처리 경험 → 유지보수 쉬운 구조적 사고 능력

3. 모듈화/추상화 사고력

- Move_Ghosts(), Move_Cars(), Draw_Map() 같은 함수를 만들며 반복 로직 분리, 역할 단위로 함수 설계
- 실무에서 요구하는 모듈 단위 설계 능력에 대한 감각 획득

4. 자원 제약 하에서의 최적화 경험

- MCU의 한정된 성능과 메모리를 고려해, 간단한 직선이동 구현
- ghost_under[]로 타일 복원 처리