스타크래프트(우주전쟁)

대한상공회의소 넥스트칩 AI 반도체 설계(2기) 최윤석 제작

1.과제개요

- 스타크래프트 - 실시간 전략 시뮬레이션 게임(RTS)
  - 전장과 본진 화면을 옮겨가며 실시간 제어하는 특징을 가짐
- 사용 실습환경: STM32 2개, C언어로 제작
- 전장 STM32(Jog Key, LCD, 핀(통신) 사용)
- 본진 STM32(Jog Key, LCD, 핀(통신) 사용)
- UART통신으로 RTS게임 구현

2.개발일정

- 목표일정

4월26일 카메라뷰 구현 및 적유닛 구조체 작성 → 4월27일 적유닛 이동 구현 → 4월28~29일 총알 구현 → 4월30일 데미지 계산 및 사망처리 → 5월1~2일 통신구현 및 본진화면 구현 → 5월3~6일 아군유닛 생산 통신 확인후 게임 업그레이드

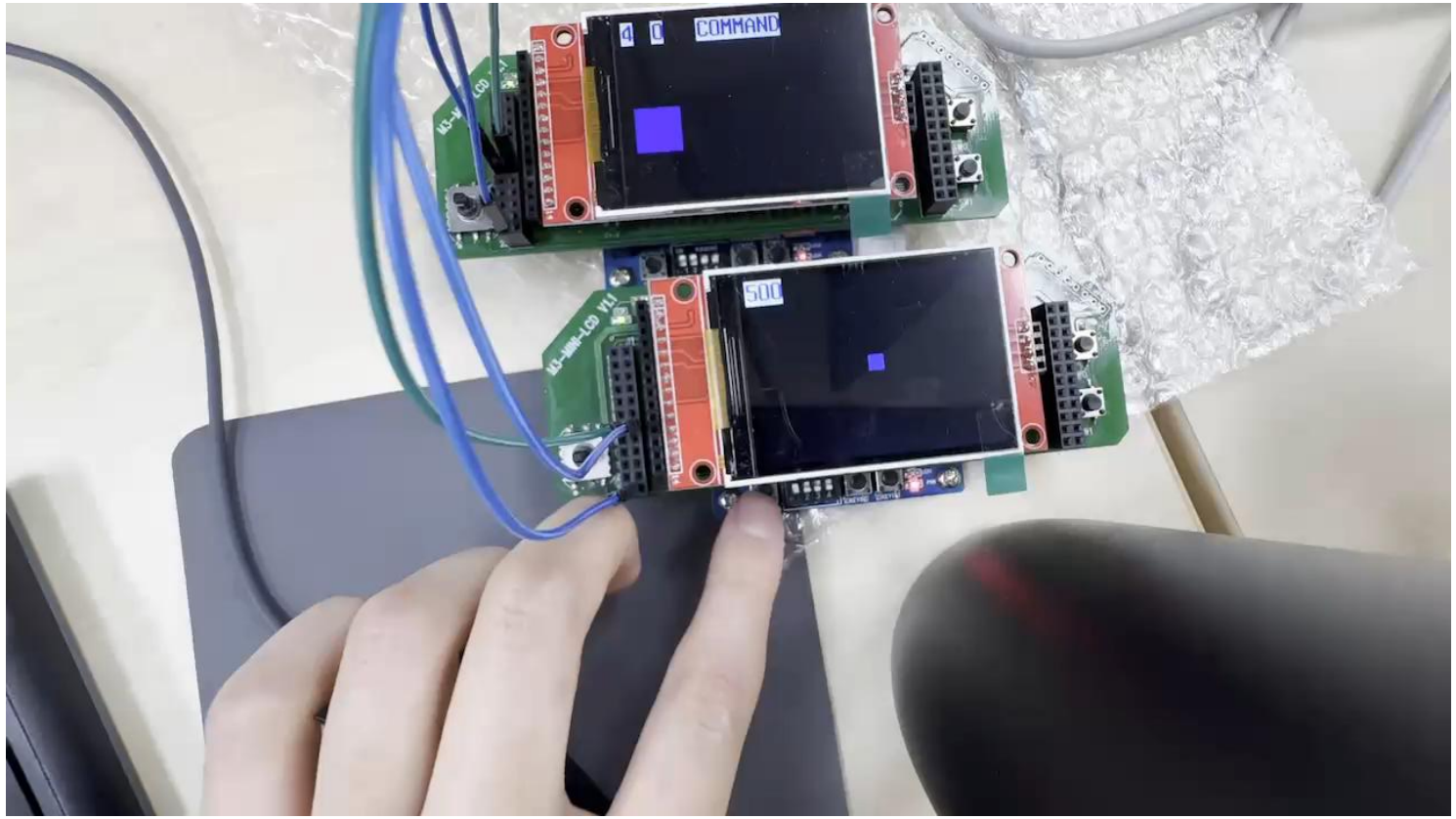
- 실제일정(코드 보완시간 부족)

4월27~28일 카메라뷰 구현 및 적유닛 구현 → 4월29~30일 적유닛 이동 구현 → 5월 1~3일 총알구현 및 데미지 계산/사망처리 → 5월 4~5일 통신구현 및 본진화면 구현

3.개발결과

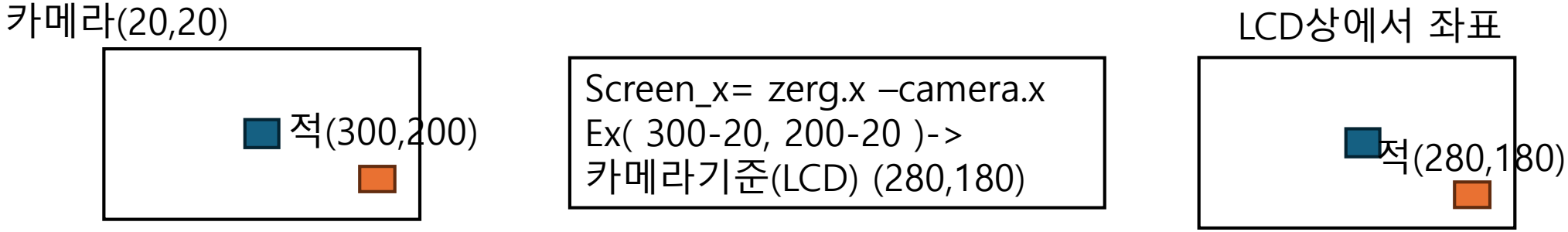
- 주요함수들
  ->UART명령 수신(UART 송신에서 온 명령어 문자열 저장후 확인)
  ->가장 가까운 적찾기(공격자의 사거리내에서 가장 가까운 적 찾기)
  ->다방향 총알 발사(가장가까운 적의 위치를 받아서 대각선도 발사가능)
  ->jogkey제어(카메라뷰제어, 아군,주인공 유닛 상태제어)
  ->유닛 소환(유닛 배열중 active=0인 부분에 초기값 부여후 소환)
  ->충돌 체크박스(유닛들 고유의 크기의 체크박스로 충돌여부 확인)

3.개발 결과 – 실제 플레이 영상



4.핵심기술

- 카메라뷰: 적유닛의 맵좌표값과 LCD상 에서의 좌표의 괴리



- 카메라뷰 크기는 320 X 240 -> LCD상 좌표가 (0~320, 0~240) 일때 그림
- (맵좌표–카메라좌표)->카메라값 변화->LCD상에서 적 위치변화 ->주인공 중앙에 고정후에도 이동 묘사 가능

4.핵심기술

- 적 유닛 이동/공격 알고리즘 :

  처음부터 모든적 유닛 달려옴 -> 너무 높은 난이도
- 상태모드 제어: 저그 초기모드(STOP) -> 카메라뷰에 저그 나타남
  -> 들어온 저그 유닛 모드(ATTACK) -> 끝까지 아군 추격
- 이를위해 구조체에 상태모드 변수 추가, 카메라뷰 함수에 적유닛 들어올시 저그유닛 상태 변환 코드 작성
- EX) if(LCD상 좌표가 카메라뷰 크기안에 들어올시(아군제외))
  -> unit.mode = mode_attack;

4.핵심기술

- 총알 알고리즘: 다방향에서 오는적 -> 총알 방향의 다방향 필요성, 가까운 적을 찾는 알고리즘 필요성
- 가까운 적을 찾는 알고리즘: 모든 적과의 거리 계산($(x1-x2)^2 + (y1-y2)^2$) -> 사거리 제곱과 비교 -> 첫번째로 사거리안 적 거리 변수에 저장-> 변수값 보다 작은 거리 유닛 -> 변수값 최신화->마지막 변수값의 좌표의 적유닛 -> 가장 가까운 적유닛으로 타겟 설정
- EX) if(사거리보다 작고 변수값(min_distance)보다 작으면) -> 변수값= 다른 유닛의 적과의 거리; (변수값 최신화)

4.핵심기술

- 총알 다방향 알고리즘: 적과의 거리 정확한 연산(sqrt)->피타고라스 연산 ->루트연산(속도 느림)
- dx = (x1-x2), dy = (y1-y2);
- argument로 공격자와 위에서 구한 타겟 좌표 ->abs연산(절대값)-> 적과의 거리 abs(dx), abs(dy)-> max_abs=둘중 가장 큰 절대값 -> dx= (dx * 총알속도) / max_abs , dy= (dy * 총알속도) / max_abs ->총알이 다양한적위치에 맞게 방향과 속력을 가지게됨
- 피타고라스 연산에 비해 정확도 떨어지지만 연산속도 빠르고 게임 영향에 문제없음

4.핵심기술

- 데미지 계산,충돌 확인: 충돌확인 좌표로 비교-> 유닛크기 고려X
- 체크박스 확인: 유닛의 크기를 고려하여 서로의 실제 모양 비교->겹치는 부분 발생시 충돌확인->데미지: target.hp-attacker.damage
- w=가로크기, h= 세로크기, self 1번, other 2번

- 코드:

```
int Check_Collision_Box(UNIT *self, UNIT *other) {
    return (self->x < other->x + other->w && self->x + self->w > other->x &&
            self->y < other->y + other->h && self->y + self->h > other->y);
}
```

-원리: 1번 왼쪽세로선이 2번 오른쪽 세로선보다 왼쪽인지, 1번 오른쪽 세로선이 2번 왼쪽 세로선보다 오른쪽인지, 1번 위쪽 가로선이 2번 아래쪽 가로선보다 위쪽인지, 1번 아래쪽 가로선이 2번 위쪽 가로선보다 아래쪽인지 확인후 모두 만족한다면 겹치는 부분이 있다는걸 확일할수 있음

4.핵심기술

- 통신구현: 전장STM32에서는 유닛생산 불가능-> 다른STM32에서 유닛생산을 해도 보낼방법 없음
- UART2통신: STM32_1(전장), STM32_2(본진) 에서 각각 TX_1<->RX_2, TX_2 <-> RX_1, GND_1<->GND_2 선연결 ->두 STM32 수신 인터럽트허용->본진 유닛 생산시 문자열 송신->전장에서 수신을 받은후 유닛 소환 함수 실행
- 문자열송신: 배열에 문자열 저장->USART 상태 레지스터 7번비트(TX 비어있음 플래그)가 1일때마다 문자열 하나씩 USART->DR(데이터 레지스터)에 보냄
- 문자열수신:수신 인터럽트 발생->문자열에 '₩n'나올때까지 저장-> if(소환명령(문자열) ==수신 문자열)-> 유닛소환 함수호출

4. 핵심 기술

- 아군 유닛 배치: 아군유닛이 한곳에서 생산->주인공주위로 이동-> 한곳에 뭉쳐서 몇 마리인지 알수 없음
- 배치 리스트사용: 처음부터 유닛별 배치 정함->{20,-20}, {0,20} ..... ->ex)배치위치(destination)=hero.x +{20,-20};

- 실제코드 :

```
const int destination_list[][2] = {
    { 0, -20 }, { 20, 0 }, { 0, 20 }, { -20, 0 },
    {-20,-20},{ 20, -20 }, { 20, 20 }, { -20, 20 }
};
for(i = 0; i < marine_count; i++){
    if(marine[i].active == 1 && marine[i].mode==mode_move){ // 마린이 활성화된 경우
        destination_x = hero.x + destination_list[i][0];
        destination_y = hero.y + destination_list[i][1];
        if (marine[i].x <= destination_x) marine[i].x += step;
        else if (marine[i].x > destination_x) marine[i].x -= step;
        if(marine[i].y <= destination_y) marine[i].y += step;
        else if (marine[i].y > destination_y) marine[i].y -= step;
        marine[i].attack_speed+=1; // 공격 쿨타임 증가
    }
}
```

5. 핵심코드

- UART송신

```
//본진 송신
void Send_Spawn_Command(const char* unit_type)
{
    char buffer[32];
    sprintf(buffer, "SPAWN_%s\n", unit_type);  // 예: SPAWN_MAARINE
    for ( i = 0; buffer[i]; i++) {
        while (!Macro_Check_Bit_Set(USART2->SR, 7)); // 송신 준비 대기
        USART2->DR = buffer[i];
    }
}
```

- UART수신

```
//수신 인터럽트 발생
  if (Macro_Check_Bit_Set(USART2->SR, 5)) {
    char data = USART2->DR;

    // 간단한 수신 버퍼에 저장
    if (uart2_rx_index < sizeof(uart2_rx_buffer) - 1) {
        uart2_rx_buffer[uart2_rx_index++] = data;

        // 예시: '\n' 도달 시 명령 처리
        if (data == '\n') {
            uart2_rx_buffer[uart2_rx_index] = 0;  // null-terminate
            Process_Command((char*)uart2_rx_buffer);
            uart2_rx_index = 0;
        }
    } else {
        // overflow 방지
        uart2_rx_index = 0;
    }
}
```

코드설명

Buffer배열에 sprintf을 이용하여 원하는 명령어 문자열을 저장(unit_type은 생성유닛이름)-> 문자열에서 첫번째부터 하나씩 송신 데이터 레지스터가 빌때 마다 데이터 레지스터에 넣어 전송

문자를 수신 받을 때마다 수신 버퍼에 저장->₩n를 수신 받으면 명령어 확인하는 함수 호출(if문으로 기다리던 명령어 같은지 확인후 유닛 소환 함수 호출)

5.핵심코드

- 가장 가까운 적 찾기

```
UNIT* Find_Closest_Enemy(UNIT *attacker) {

    UNIT* closest = NULL;
    int min_distance_squared = 9999999;

    int range = attacker->attack_range; // 공격자의 사거리
    int range_squared = range * range;

    UNIT* groups[] = {zergling, hydralisk, ultralisk};
    int counts[] = {zergling_count, hydralisk_count, ultralisk_count};
    int g,i;
    for ( g = 0; g < 3; g++) {
        for ( i = 0; i < counts[g]; i++) {
            UNIT* enemy = &groups[g][i];
            if (!enemy->active || enemy->hp <= 0) continue;

            int dx = enemy->x - attacker->x;
            int dy = enemy->y - attacker->y;
            int dist_sq = dx * dx + dy * dy;
            if (dist_sq <= range_squared && dist_sq < min_distance_squared) {
                min_distance_squared = dist_sq;
                closest = enemy;
            }
        }
    }
    return closest;
}
```

공격자를 argument로 받아서 공격자의 사거리를 제곱하여 제곱비교를할 준비를 합니다. 생존해있는 모든적유닛들의 좌표를 가지고 공격자의 좌표와 비교해 상대 거리를 제곱하여 사거리와 상대거리를 제곱 비교(연산속도 빠름) 합니다. 또한 비교중 사거리내 상대거리가 더 작은 얘가 나타나면 가장작은거리 변수에 최신화 하며 계속 비교후 마지막으로 남은 사거리내 가장 가까운 적 유닛을 찾게 됩니다.

6.결론

- 게임진행에 있어 필수적 요소들 구현 성공
  -> UART통신: 유닛 생산 시 소환 명령어 전달
  -> 총알: 가까운 적을 향하는 다방향 총알 구현
  -> 두개의 STM32화면 생성
  -> 카메라 뷰 : 넓은 맵 활용
  -> 저그 유닛 : 이동/공격 알고리즘 구현

6.결론(아쉬운 점)

- 적 유닛들의 이동/공격 알고리즘 좀더 손 볼 필요가 있음
- SCV 자원수집 움직임 구현 필요
- 인구수로 총 유닛 제어
- 적 유닛 리스폰이 없는 것과 배치위치의 아쉬움
- 제3의 종족 프로토스 유닛
- 본진 STM에서 건물 선택방식이 너무 단조로움
- 더 다양한 유닛들과 공격 방식 구현 못함
  ex)스플래시(광역) 데미지(럴커 가시, 파이어 뱃 화염, 탱크 포격)

7.개발후기

- 삽질의 추억: 최근 기억 나는 것은 UART통신 오류로 화면이 안나오는데 통신은 가능한 이상한 상황에 마주해서 하루종일 수정에 매달렸는데 못 고쳤습니다.
  그런데 통신은 가능하다는점에서 포기하기 아쉬워 다음날 보니 문득 송수신 하려해서 송신 인터럽트를 켰지만 송신 인터럽트는 송신 데이터 레지스터가 비면 발생한다는게 기억나서 인터럽트에 갇힌 상황을 뒤늦게 알고 송신인터럽트 끄고 송신을 해봤습니다.
  그런데 송신도 되고 수신 인터럽트도 잘 발생해서 정상 작동돼서 매우 당황한 삽질 중에 삽질 입니다.
  이밖에도 객체지향 흉내 내려다 실패,

7.개발후기

- 교훈: 코드를 짜기전에 계획과 절차를 만들어야 하며 코딩 중간 중간 디버깅을 해야합니다. 또한 후회되는것이 메인에 다 작성해버렸지만 모듈화(move.c, attack.c등)하였다면 유지보수에 좋았을 것이고, 공통 알고리즘(공통 이동/공격 함수)을 염두해서 코드를 작성했다면 완성속도가 좀더 빨라 졌을것입니다.
  그리고 삽질할때는 꼭 쉬고 오십쇼...
- 얻은것: 코드 문제분석력, 알고리즘 이해, 포인터 활용능력, 나름의 최적화 능력, 구조체 이해,