개복치 게임_유리멘탈 복치

과목명	임베디드 시스템
담당 교수	공기석
날짜	2020.12.10
팀 번호	5팀
팀원	2018152011 김채리 2018152010 김지안 2016156021 이기웅

임베디드 시스템 설계 목차

5. 실행 결과 및 분석

5-1. 실행 결과 정리

5-2. 실행 결과에 대한 분석

5-3. 디바이스 드라이버 개선사항

6. 결론

6-1. 평가 및 반성

6-2. 향후 계획

7. 참고문헌

1. 개요

1-1. 개발 목표 및 요구사항 분석

- 게임 시작 버튼을 누르면 알 그림이 뜨고 온도는 7°C, 용존산소량은 120mg/L에서 시작한다.
- 알 단계에서는 하나의 사망 이유가 존재한다.
- Push Switch 버튼으로 1°C 온도 조절이 가능하다.
- 온도를 3초마다 검사해서 적정 온도가 아닐 시에 LED를 하나씩 키고 8개가 다 켜지면 알에서 부화하지 못해서 사망하게 된다.
- 10~20°C 사이를 30초 동안 유지하면 알에서 부화해서 유년기로 진화한다.
- 유년기에는 세 가지의 사망 이유가 존재한다.
- 적정 수온 20~30°C, 적정 용존산소량 70~150mg/L를 유지해야 스트레스 지수가 상승하지 않는다.
- 수온과 용존산소량 모두 20초마다 랜덤함수로 값을 받아오되 받아오는 값의 범위는 수온은 10~40°C, 용존산소량은 20~150mg/L이다.
- Push Switch 버튼으로 온도는 1°C, 용존산소량은 10mg/L씩 조절 가능하다.
- LED로 스트레스 지수 설정하고 온도와 용존산소량을 20초마다 검사해서 적정 온도나 적정 용존산소량이 아닐 시에 LED를 하나씩 키게 된다.
- 특정 Push Switch 버튼을 클릭하면 개복치를 만질 수 있는데 만질 때마다 스트레스 지수가 상승하여 LED가 하나씩 켜진다.
- LED 8개가 다 켜지면 스트레스로 인해 사망하게 된다.
- Push Switch에서 플래시로 지정된 버튼을 누르면 플래시 때문에 놀라서 사망하게 된다.
- 햇빛에 관한 Dot Matrix의 값 3개를 미리 지정해두고 1분마다 랜덤함수로 받아온다.
- 햇빛은 Dot Matrix에 3가지 모양으로 출력되며 가장 강한 햇빛이 비춰지면 햇빛이 너무 강해서 사망한다.
- 아무일 없이 2분이 지나면 노년기로 진화한다.
- 노년기에는 유년기의 세 가지 사망 이유에 물살을 추가해 네 가지의 사망 이유가 존재한다.
- Step Motor로 물살의 세기를 표현하며 물살의 세기를 30초마다 랜덤함수로 받아오되 받아오는 값의 범위는 100~170으로 설정한다.
- 물살의 세기가 150이상이면 물살이 너무 세서 사망한다.
- 알에서 30초, 유년기에서 2분, 노년기에서 3분이 지나게 되면 자연사로 사망한다.

1-2. 문제점 및 해결방안

• 디바이스 드라이버와 관련한 여러 이벤트들이 동시에 제어가 안된다.
– 여러 개의 스레드를 생성해서 동시에 여러 이벤트들을 제어할 수 있게 한다.
– 알 단계에서는 온도를 제어하는 스레드, 온도를 체크하는 스레드, 총 2개의 스레드를 생성한다.
– 유년기 단계에서는 온도와 용존산소량을 조절하고 플래시를 누를 수 있는 스레드와, 햇빛 제어 스레드, 산소와 온도를 체크하는 스레드, 총 3개의 스레드를 생성한다.
– 노년기 단계에서는 유년기 단계의 3개의 스레드에 물살 제어 스레드를 추가해서 총 4개의 스레드를 생성한다.

• 알에서 30초가 지나면 유년기로, 유년기에서 2분이 지나면 노년기로, 노년기에서 3분이 지나면 자연사 해야하는데 시간을 재는 방법이 없다.
– int run(int t)이라는 함수를 따로 만들어 t가 1일 때는 알 단계, t가 2일 때는 유년기 단계, t가 3일 때는 노년기 단계로 단계에 맞는 스레드를 각각 생성해준다.
– 변수 t를 스레드 함수의 인자로 넘겨서 시간을 체크한다.
– 알인 경우, 즉 t가 1인 경우에 3초마다 검사를 하므로 10번 검사를 하면 30초가 지나게 된다. 그러므로 검사 횟수 변수를 count로 지정하고 count가 10이하일 동안만 검사를 진행하고 10보다 커지게 되면 값을 리턴해서 다음 단계로 넘어간다.
– 유년기의 경우, 즉 t가 2인 경우에는 20초마다 검사를 하므로 6번을 검사하면 2분이 지나게 된다. 알인 경우와 마찬가지로 검사 횟수를 count로 지정하고 count가 6이하일 동안만 검사를 진행한다.
– 노년기의 경우, 즉 t가 3인 경우에 20초마다 검사를 하므로 9번을 검사하면 3분이 지나게 된다. 알인 경우와 마찬가지로 검사 횟수를 count로 지정하고 count가 9이하일 동안만 검사를 진행한다.

1-3. 역할 분담

김채리	• 알 스레드 코드 작성 • 메인 헤더 파일 작성 • PPT 제작 • 보고서 목차 1~3과 알 핵심함수 작성
김지안	• 노년기 스레드 코드 작성 • run() 코드 작성 • 보고서 전체 소프트웨어 구성도와 노년기 핵심함수와 실행결과 분석 작성
이기웅	• 유년기 스레드 코드 작성 • 메인 ui 작성 • 보고서 흐름도와 성년기 핵심함수와 목차 6 작성

1-4. 일정 계획

월	11	11	12	12
주차	3	4,5	1	2
주제 선정 및 제안서 작성				
Qt 디바이스 드라이버 소스코드 분석				
알, 유년기, 성년기 단계별 스레드 설계				
전체 코드 작성 및 이미지 삽입				

2. 팀 활동 내역

2-1. 주차별 활동 내역

월	11	11	12	12
주차	3	4,5	1	2
김채리	• 제안서 PPT 작성 • 알, 성년기 시나리오작성	• LED, LCD Qt 디바이스 드라이버 소스코드 분석	• 알 스레드 작성 • 메인 헤더파일 작성	• 전체 코드 작성 • 보고서 작성 • PPT 작성
김지안	• 전체 시스템 수행 시나리오 작성 • 사용 디바이스 정리	• FND, Motor Qt 디바이스 드라이버 소스코드 분석	• 노년기 스레드 작성 • run() 코드 작성	• 전체 코드 작성 • 보고서 작성
이기웅	• 노년기 시나리오 작성	• Switch, LED Qt 디바이스 드라이버 소스코드 분석	• 성년기 스레드 작성 • 메인 ui 작성	• 전체 코드 작성 • 보고서 작성

3. 설계

3-1. 현실적인 제약사항

- 사망하거나 다음 단계로 넘어갈 때 멜로디를 출력하려 했으나 부저 디바이스의 오류로 구현하지 못했다.

- Qt에서 이미지를 넣을 때 개복치가 계속 움직이도록 좌표를 설정하고 싶었으나 여러 스레드를 실행시키다보니 단순히 이미지를 바꾸는 것조차 오래 걸려서 하지 못했다.

- 유년기 단계의 개복치부터는 3개의 스레드가 동시에 돌아가 관리하는 데 어려움이 있었다.
  특히, 3개의 스레드 중 하나라도 종료가 되면 나머지 2개의 스레드가 동기화되어 같이 종료되어야 하는 데, join함수를 호출한 시점에서 이를 구현하는 것에 큰 어려움이 있었다. 기간 내에 문제를 해결 하지못하여 유년기에서는 불가피하게 중요도가 떨어지는 스레드(햇빛) 하나를 빼게 되었다.

- 노년기에서 구현했던 물살 관련 함수를 포함하여 실행시키려 했으나 스레드를 조작하는데 문제가 생겨 포함시키지 못하였다.

- 스레드를 여러 개 동작시켜줄 때는, 스레드 하나가 종료될 때 다른 스레드가 다 같이 종료되어야 한다. 현재 코드에서는 각 스레드마다 값을 반환해주며 종료한다. 이를 join함수를 이용해서 받아야하는데 정해진 시간동안 돌아가는 스레드는 문제가 없다. 하지만 push_switch의 입력을 받는 스레드4는 이런 부분에서 문제가 생긴다.
  예를 들어 스레드 2개가 돌아가는 알 단계에서 스레드 하나는 다른 스레드가 값을 return해주기 전에 변수의 값을 이용해 그 스레드에게 종료를 알릴 수 있다.. 그러면 다른 스레드는 그 변수의 값을 확인하고 스레드를 종료시킨다.

- 노년기에서는 값을 받아야하는 스레드가 햇빛 스레드, 물살 스레드, 스트레스 스레드로 총 3개가 되는데, 이 때 각 스레드 내부에서는 변수의 값을 확인하고 스레드를 종료시킬 수 있다.
  하지만 이 부분에서 값을 받아야 하는 스레드가 3개이므로, join함수를 이용해 3개의 값을 다 받아야 할 때까지 스레드는 종료되지 않는다. 또, 이 때 join함수의 순서가 잘못되면 스레드가 종료되지 않아 프로그램은 오류가 발생한다.

- 햇빛 스레드, 물살 스레드, 스트레스 스레드는 동시에 조작하므로, 어떤 스레드에서 가장 먼저 값을 반환하는지 알 수 없다. 그렇기 때문에 join함수의 순서를 잘 배치한다해도, 정상적으로 스레드를 끝낼 수 없다.

- 스레드를 외부에서 한 번에 여러 개 죽일 수 있는 방법이 프로세스를 죽이는 방법 외에는 없다.
  이는 전역변수를 이용하거나 quit()함수를 써서 구현할 수 있을 줄 알았으나 코드에 다른 오류가 있었는지 스레드의 오류가 해결되지 않았다.

3-2. 설계 방법 및 설계 전략

설계전략
• 스레드를 사용하여 여러 개의 디바이스들을 동시 제어
• 알, 유년기, 노년기로 나누어 스레드를 생성
• game start버튼은 GUI를 통해 구현
• 코드의 구조화
• 전역 변수를 이용하여 스레드를 제어
• mutex를 이용하여 여러 스레드에서 접근 가능한 temp와 oxygen 데이터를 보호

유리멘탈 복치 프로그램에서는 여러 개의 디바이스를 동시에 제어해야 하므로 스레드를 사용했다.
실습실 가상머신의 코어 개수가 4개이므로 한 번에 실행시키는 스레드는 4개를 넘지 않도록 하였다.

알, 유년기, 노년기로 나누어 스레드를 따로 생성하고 넘어갈 때마다 이전 단계의 스레드는 종료시켜 자원을 반환해주었다.

Push Switch로 게임을 시작하게 될 경우, 온도 제어와 용존산소량 제어 등 Push Switch를 이용한 이벤트를 처리하는데 혼란을 줄 수 있어 game start의 버튼은 GUI를 이용하여 구현하였다.

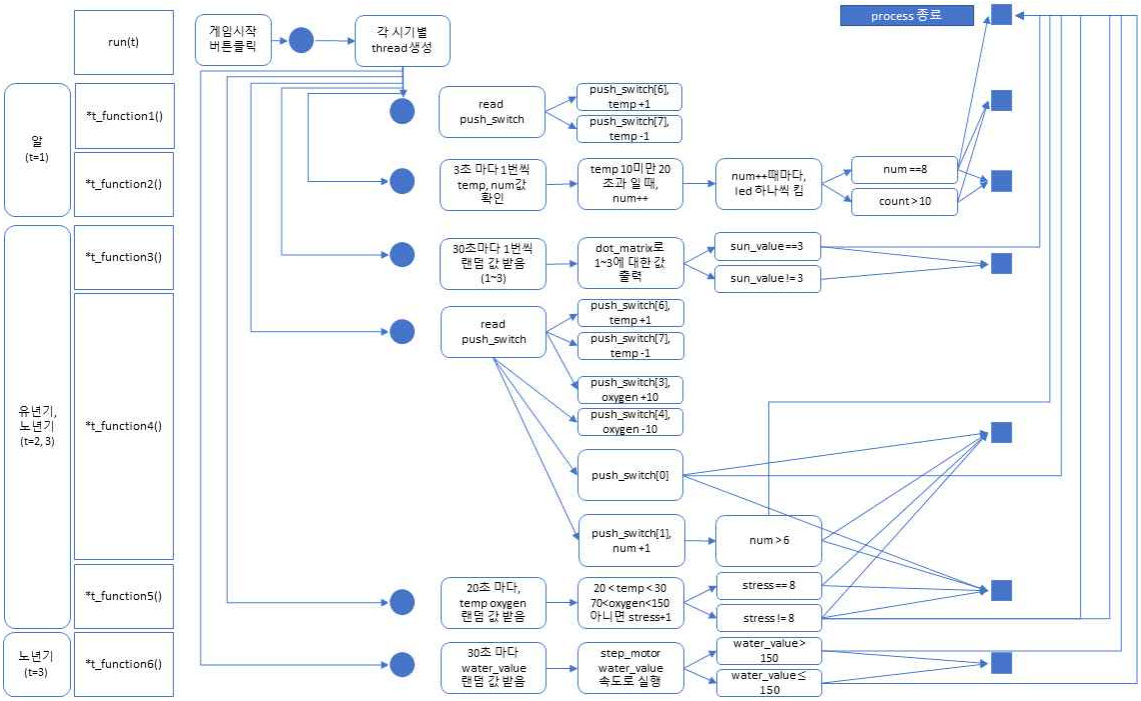
run() 함수를 생성하여 알, 유년기, 노년기로 나누어 작업을 수행하도록 해 조금 더 구조적으로 코드를 작성하였다.
run()함수 내에서 return 값을 하나씩 지정하여, 각 값에 따라 사망의 원인을 나타내었고 main이라 할 수 있는 buttonClicked()함수에서 그 값에 따라 이미지를 바꾸어 주었다.

전역변수로 temp_th1을 사용하여, 개복치가 알이었을 때 스레드를 제어한다. *t_function2()함수를 쓰는 스레드에서는 count를 이용하여 시간을 재고 그 시간 동안만 스레드를 돌릴 수 있게 하지만 *t_function1()함수에서는 시간을 측정할 수 있는 부분이 없다.
그래서 *t_function2()함수 내에서 스레드를 종료할 때 temp_th1의 값을 바꾸어준다. *t_function1()함수 내에서는 temp_th1의 값이 바뀌었을 때 스레드를 종료한다.

전역변수인 temp와 oxygen의 값을 여러 스레드에서 동시 접근하면 충돌이 발생하므로 mutex를 사용해 한 번에 하나의 스레드만 접근 가능하도록 해 temp와 oxygen변수의 데이터를 보호한다.

4. 소스코드 구성

4-1. 전체 소프트웨어 구성도



소프트웨어 구성은 크게 MainWindow 창을 생성하는 생정자 함수와, 버튼이 클릭되었을 때 발생하는 이벤트 buttonClicked()함수로 나뉘게 된다.
게임 시작 버튼을 클릭하면 buttonClicked()함수의 내용이 실행되고 이 때 main이라 할 수 있는 run()함수가 실행된다. 그리고 그 안에서 있는 단계에 맞는 스레드가 생성된다.

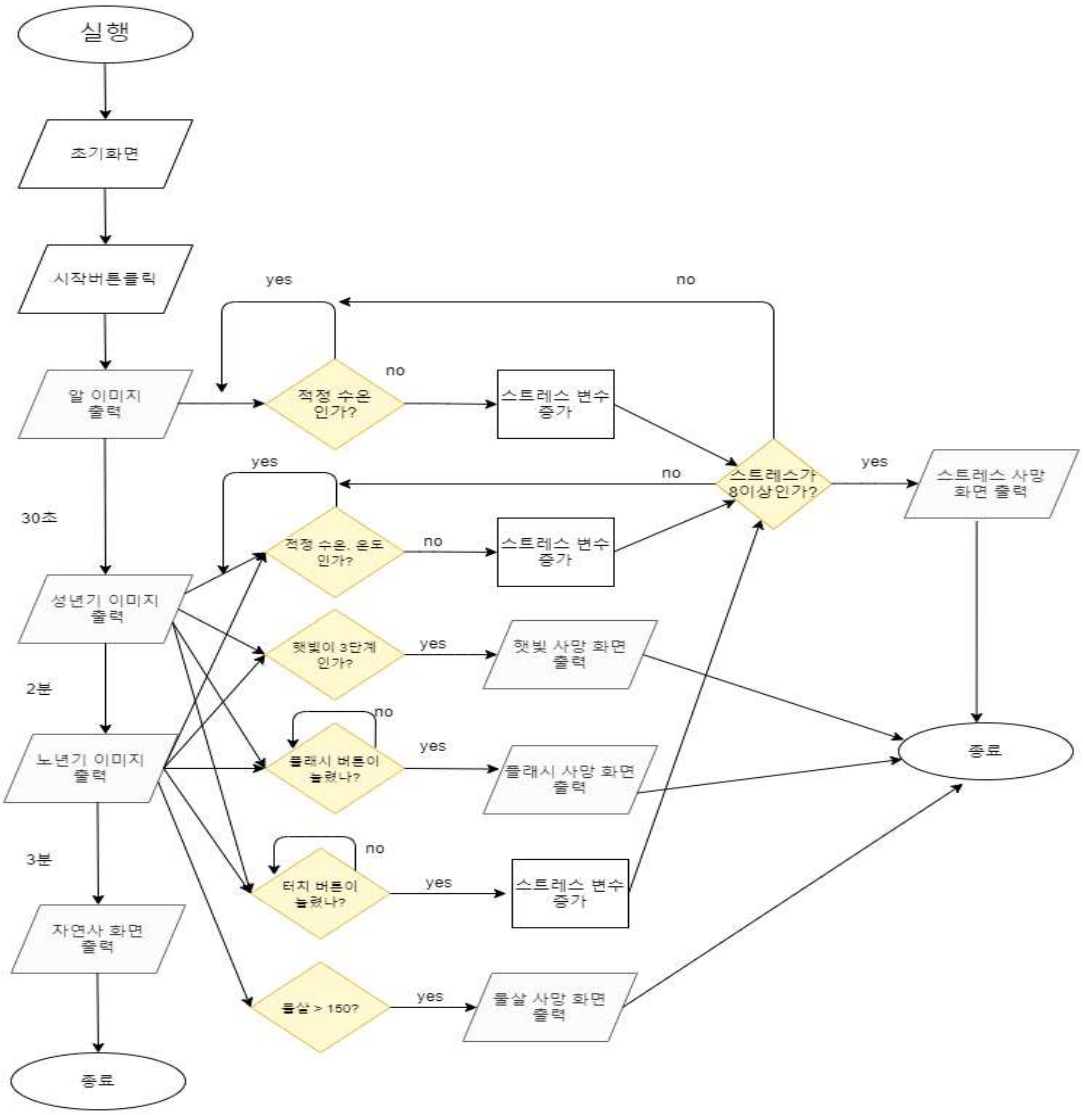
개복치가 알일 때 생성되는 스레드는 *t_function1(), *t_function2()함수를 쓰는 스레드로 총 2개이다. 먼저 *t_function1()에서는 Push Switch의 값을 읽어와 6번 값이면 온도를 1올리고, 7번 값이면 온도를 1을 내려준다.
*t_function2()함수에서는 3초 마다 한번 씩 온도와 그에 따른 스트레스 값(num)을 읽어온다. 온도가 적정온도인 10~20℃가 아니라면 스트레스가 1 증가한다.
이 때, 스트레스가 8이 되면 스레드2가 종료하면서 스레드1도 종료시키며 run()함수를 return해준다. 스트레스가 8이 아니고 count가 10이 넘으면(30초가 지나가면) 마찬가지로 스레드 2개를 종료시키며 run()함수에서 return해준다.

개복치가 유년기 때 생성되는 스레드는 *t_function3(), *t_function4(), *t_function5()함수를 쓰는 스레드로 총 3개이다. 스레드4에서는 알 단계에서 스레드1과 기능이 유사하다. Push Switch 값을 읽어와서 해당 기능을 실행한다. 알 스레드1에서는 온도만 제어했다면 스레드4에서는 온도, 산소, 플래시, 터치를 제어한다.
스레드4에서는 플래시를 눌렀을 때, 터치를 많이 하여 스트레스가 6이 넘었을 때만 스레드를 종료

시키며 run()함수를 return한다.
스레드5는 알 스레드2와 비슷하다. 적정온도의 범위가 바뀌고 온도뿐만 아니라 용존산소량도 검사하여 스트레스 값을 제어해준다. 알 스레드2와 똑같이 수행시간이 정해지지 않은 스레드4를 스레드2를 종료할 때, 같이 종료시켜준다.
유년기 때, 스레드3은 30초마다 1~3 값을 랜덤으로 받는다. 그리고 각 값마다 dot_matrix로 해 밝기 정도를 출력해준다. 값이 3일 때는 해가 너무 밝아 스레드를 종료시키며 run()함수를 return 해준다. 값이 3이 아니라면 스레드3만 종료시켜준다.

노년기 때는 *t_function6()함수를 사용하는 스레드6을 제외하고 유년기와 동일하다. 스레드6에서는 30초마다 100~170 값을 랜덤으로 받는다. 그리고 그 값을 Step Motor의 속도로 하여 출력한다. 이 때 값이 150이상이라면 속도가 너무 빨라 스레드를 종료시키며 run()함수를 return하여 게임을 종료시킨다. 값이 150미만이면, 스레드6을 종료하고 run()함수를 return해주며 게임은 종료된다.

4-2. 흐름도



프로그램 실행 시 개복치 게임의 메인 화면이 출력된다.
초기 화면에 있는 게임 시작 버튼을 클릭해 게임을 진행할 수 있다. 게임이 시작되면 동시에 개복치의 알 이미지가 출력된다.

적정 수온은 10~20˚C이고 알 단계에서는 적정 수온인가를 주기적으로 검사한다. 만약 적정 수온이 아니라면 개복치의 스트레스 지수가 올라가고 스트레스 지수가 8을 넘어가게 되면 개복치가 스트레스로 인해 사망하는 화면을 출력하고 게임이 종료된다. 적정 수온이라면 3초 뒤에 다시 검사를 수행한다.

30초가 지나면 개복치의 알에서 유년기 단계로 성장하게 된다.
유년기에서는 수온과 산소량을 20초마다 검사하는데 각각의 적정 수온은 20~30˚C, 적정 용존산소량은 70~120mg/L이다.

만약에 이 수치를 벗어나면 알에서와 마찬가지로 스트레스의 지수가 증가한다. 또 개복치를 만지게 되면 스트레스 지수가 증가한다.
그리고 30초마다 랜덤으로 출력되는 1~3단계의 햇빛 중 3단계의 햇빛이면 개복치를 햇빛에 의한 사망에 이르게 할 수 있다. 이 경우 햇빛에 의한 사망 화면이 출력되며 게임이 종료된다.
마지막으로 플래시 버튼을 누르게 되면 개복치가 바로 사망하는데 이때 플래시로 인한 사망 화면을 출력하고 게임이 종료된다.

2분이 지난 성년 개복치는 노년 개복치 단계로 성장하게 된다.
성년기의 조건검사문에 물살이 150 이상인지 확인하는 부분이 추가된다.
30초마다 랜덤 값으로 받아온 물살 값이 150 이상이면 개복치는 강한 물살에 의해 사망하게 되며, 물살에 의한 사망 화면을 출력하고 게임을 종료한다.
노년기에서 3분이 지나면 개복치의 수명이 다해 자연사하게 된다. 이 때, 자연사에 의한 사망 화면을 출력하며 게임이 종료된다.

4-3. 핵심 함수 설명

4-3-1. 알

알 단계에서는 run함수의 인자에 t=1가 들어가게 된다.

```
QApplication::processEvents(); //사진을 정상적으로 불러주기 위해 호출
pthread_mutex_init(&mutex, NULL); //뮤텍스를 초기화, 기본 특징 사용
clear_led(); //LED 초기화
//알 단계 호출 함수
int retVal1=run(1);
    //스레드로부터 0을 반환 받으면 알에서 부화한 유년기의 개복치 화면 출력
    if(retVal1==0){
        QPixmap pix3("/home/pi/Modules/adolescent_bokcci.png");
        ui->label_pic->setPixmap(pix3);
        ui->label_pic->setPixmap(pix3.scaled(w,h));
        return;
    }
    //스레드로부터 1을 반환 받으면 알에서 부화하지 못해 사망한 화면 출력
    else if(retVal1==1){
        QPixmap pix4("/home/pi/Modules/die_cannothatch.png");
        ui->label_pic->setPixmap(pix4);
        ui->label_pic->setPixmap(pix4.scaled(w,h));
        return;
    }
```

알 단계에서는 on_pushButton_clicked()함수에서 t를 1로 설정하여 run()함수에 넘겨주게 된다.
그리고 run()반환 값에 따라 다른 화면이 출력되게 된다.
스레드로 0을 반환받으면 알에서 부화에 성공한 것이므로 유년기의 개복치 화면을 출력한다
1을 반환받았다면 사망한 것이므로 알에서 부화하지 못해 사망한 화면을 출력한다.

```
//int run(int t) 함수에서 호출되는 알 단계(t==1) 스레드 생성 함수
int run(int t) {
    if(t == 1) {
        //p_thread[0],[1] : 생성된 스레드 id를 저장할 변수의 포인터
        //NULL : 스레드 특성 설정
        //(void*)&t : 함수에 전달하고자 하는 인자 값(알 단계에서는 1전달)
        thr_id = pthread_create(&p_thread[0], NULL, t_function1, (void *)&t);
        //t_fucntion1 : 스레드가 생성되고 나서 실행되는 함수(수온 제어 함수)
        thr_id = pthread_create(&p_thread[1], NULL, t_function2, (void *)&t);
        //t_fucntion2 : 스레드가 생성되고 나서 실행되는 함수(수온 체크 함수)
        if(thr_id<0){
            perror("thread create error : ");
            exit(0);
        }
        //스레드가 잘 생성되었을 때는 0을 반환, 이 외는 에러 코드 값을 리턴
        pthread_join(p_thread[1], &tret);
        //p_thread[1]가 종료할 때까지 실행을 지연시킴
        //temp_th1의 값이 초깃값 2가아닌 0이나 1일 경우 스레드 종료 요청
        if(temp_th1 == 0|| temp_th1 == 1) pthread_cancel(p_thread[0]);
        //tret에는 온도 체크 함수가 종료할 때 반환하는 값이 저장됨
        if(*((int*)tret) == 0) return 0;
        //tret의 값이 0일 때는 알에서 부화한 것이므로 0반환
        else return 1;
        //그 외의 값일 때는 알에서 부화하지 못해 사망한 것이므로 사망 값 1반환
}
```

int run(int t)에 포함된 알 단계에서 사용되는 스레드들을 실행시키는 코드이다.
알 단계, 즉 t=1이 주어지면 호출되고 관련 스레드들이 생성된다.
그리고 스레드들의 반환 값을 받아 on_pushButton_clicked()로 반환해서 그에 따른 이미지를 출력한다.

```
void *t_function1(void *data) //수온을 제어하는 스레드
{
    text_print(); //현재의 수온와 용존산소량 출력
    while(1){
        //Push Switch 값을 읽어옴
        read(push_switch_dev, &push_sw_buff, buff_size);
         //버튼 6,7은 수온의 값을 변경하고 변경된 값은 LCD에 출력
         if(temp_th1 == 0|| temp_th1 == 1) {
              pthread_exit(0);
         }  //다른 쓰레드와 동기화를 하기 위해 temp_th1가 2가 아니라면 의미없는 스레드 반환
               값을 리턴하고 종료
        //다른 스레드들이 OXYGEN, TEMP에 동시접근을 못 하도록 뮤텍스 사용
        if(push_sw_buff[6]==1){
            pthread_mutex_lock(&mutex);
            temp=temp+1;
            text_print();
            pthread_mutex_unlock(&mutex);
        }
        if(push_sw_buff[7]==1){
            pthread_mutex_lock(&mutex);
            temp=temp-1;
            text_print();
           pthread_mutex_unlock(&mutex);
        }
    }
    sleep(1);
}
```

수온 제어 스레드로 Push Switch의 값을 읽어온다.
6번이면 수온을 1°C씩 올려주고 7번이면 1°C씩 내려준다.
그리고 전역변수인 temp와 oxygen의 값을 여러 스레드에서 동시 접근하면 충돌이 발생하므로 뮤텍스를 사용해 temp와 oxygen변수의 사용을 제한하였다.
또 temp_th1은 전역변수로 초기의 값은 2로 설정되어 있다. 만약 다른 스레드에서 조건에 따라 종료가 되면 2에서 1이나 0으로 바뀌므로 이 스레드 또한 종료하게 된다.

```
void *t_function2(void *data){ //온도를 체크하는 스레드
    int count = 1; //검사 횟수
    static int retval=0; //스레드 반환값 선언
    retval = (int*)malloc(sizeof(int));
    while(count<=(*((int *)data))*10){
        //여기서 (*((int *)data))은 t값으로 1
        //알 단계는 30초이므로 검사를 3초마다 한다고 할 때 10번 검사를 하면 됨
        sleep(3); //3초마다 검사
        if(temp>20 || temp<10){ //적정 수온이 아닐시에
            write(led_dev,&led_data[num],1); //배열에 저장된 LED값을 하나씩 켬
            num++;
        }
        if(num==8){
            retval=1;
            temp_th1 = 1; //temp_th1을 1으로 바꾸면서 다른 스레드에 종료를 알려주는 동시에
                          스트레스에 의한 사망임을 알려줌
            pthread_exit((void*)&retval);
            //스레드의 리턴 값을 1로 설정하여 즉시 스레드 종료
        }
        count++;
        //수행 시간 측정을 위한 count 1증가, 3초 마다 1씩 증가
        //검사 횟수인 count가 10이 넘어가면 반복문 탈출(알 단계 끝)
    }
    temp_th1 = 0; //temp_th1을 0으로 바꾸면서 다른 스레드에 종료를 알려줌
    pthread_exit((void*)&retval); //스레드의 리턴 값이 0이면 알에서 부화
}
```

*data 함수에 넘어온 인자 값은 앞에서 본 run()함수의 t값으로 1이다.

검사 횟수 count를 1로 설정하고 sleep(3)으로 3초 마다 수온을 검사한다. 그러면 count의 횟수가 10번이면 알 단계 30초를 설정할 수 있게 된다.

만약 이때 LED가 다 켜졌다면 스레드 리턴 값을 1로 설정하고 즉시 스레드를 종료하는 동시에 temp_th1을 1로 설정하여 다른 스레드에게 종료를 알려준다.

만약 검사 횟수 count가 10이 넘어가서 반복문을 탈출했다면 리턴 값을 0을 반환하며 스레드를 종료한다. 그리고 temp_th1 또한 0으로 설정하여 다른 스레드를 동시에 종료시킨다.

4-3-2. 유년기

유년기 단계에서는 run함수의 인자에 t=2가 들어가게 된다.

```
clear_led(); //LED 초기화
QApplication::processEvents(); //사진을 정상적으로 불러주기 위해 호출
//유년기 단계 호출 함수
int retVal2=run(2);
    //스레드로부터 1를 반환받으면 햇빛으로 인한 사망 화면 출력
    if(retVal2 == 1) {
        QPixmap pix5("die_sun.png");
        ui->label_pic->setPixmap(pix5);
        ui->label_pic->setPixmap(pix5.scaled(w,h));
        sleep(3);
        return;
     }
     //스레드로부터 5를 반환받으면 스트레스로 인한 사망 화면 출력
     else if(retVal2 == 5) {
        QPixmap pix6("die_stress.jpg");
        ui->label_pic->setPixmap(pix6);
        ui->label_pic->setPixmap(pix6.scaled(w,h));
        return;
     }
    //스레드로부터 2를 반환받으면 노년기 단계화면을 출력
     else if(retVal2 == 2){
       QPixmap pix7("old_bokcci.png");
       ui->label_pic->setPixmap(pix7);
       ui->label_pic->setPixmap(pix7.scaled(w,h));
       return;
     }
    //스레드로부터 6를 반환받으면 플래시로 인한 사망 화면 출력
    else if(retVal3 == 6) {
        QPixmap pix12("flash.jpg");
        ui->label_pic->setPixmap(pix12);
        ui->label_pic->setPixmap(pix12.scaled(w,h));
        return;
    }
```

유년기 단계에서는 on_pushButton_clicked()함수에서 t를 2로 설정하여 run()함수에 넘겨주게 된다.
그리고 run()반환 값에 따라 다른 화면이 출력되게 된다.
스레드로 1을 반환받으면 햇빛으로 인해 사망한 화면을 출력한다.

5을 반환받았다면 스트레스로 인해 사망한 화면을 출력한다.
6을 반환받았다면 플래시로 인해 사망한 화면을 출력한다.
2를 받환받았다면 정상적으로 진화한 것이므로 노년기의 개복치 화면이 출력된다.

```
// int run(int t) 함수에서 호출되는 유년기 단계(t==2) 스레드 생성 함수
 if(t == 2) {
        //p_thread[2],[3],[4] : 생성된 쓰레드 id를 저장할 변수의 포인터
        //NULL : 쓰레드 특성 설정
        //(void*)&t : 함수에 전달하고자 하는 인자 값(유년기 단계에서는 2전달)
       thr_id = pthread_create(&p_thread[2], NULL, t_function3, (void *)&t);
        //t_fucntion3 : 쓰레드가 생성되고 나서 실행되는 함수(햇빛 관련 함수)
       thr_id = pthread_create(&p_thread[3], NULL, t_function4, (void *)&t);
        //t_function4 : 쓰레드가 생성되고 나서 실행되는 함수(Push Swtich 제어 함수)
       thr_id = pthread_create(&p_thread[4], NULL, t_function5, (void *)&t);
        //t_function5 : 쓰레드가 생성되고 나서 실행되는 함수(수온, 산소량 체크 함수)
         if(thr_id<0) {
             perror("thread create error : ");
             exit(0);
         } //스레드가 잘 생성되었을 때는 0을 반환, 이 외는 에러 코드 값을 리턴
         pthread_join(p_thread[2], &tret_sun);
         //p_thread[2]가 종료할 때까지 실행을 지연시킴, tret_sun은 햇빛 관련 함수가 종
           료할 때 반환하는 값이 저장됨
         pthread_join(p_thread[3], &tret_stress);
         //p_thread[3]가 종료할 때까지 실행을 지연시킴, tret_stress은 Push Swtich
           제어 함수가 종료할 때 반환하는 값이 저장됨
         pthread_join(p_thread[4], &tret_check);
         //p_thread[4]가 종료할 때까지 실행을 지연시킴, tret_check은 수온, 산소량 체
           크 함수가 종료할 때 반환하는 값이 저장됨
         if(temp_th2 == -1) return 6;
         //temp_th2값이 -1일 때는 플래시가 원인인 사망 값 6을 반환
         else if(*((int*)tret_stress == 2 || temp_th2 == 0) return 5;
         //tret_stress값이 2이거나 temp_th2값이 0 일때는 스트레스로 인한 사망 값 5
           를 반환
         else if(*((int*)tret_sun) == 3) return 1;
         //tret_sun이 3일 때는, 햇빛이 원인인 사망 값 1을 반환
         else return 2; //아무것도 해당하지 않으면 2를 반환하고 노년기 단계로 진화
    }
```

int run(int t)에 포함된 유년기 단계에서 사용되는 스레드들을 실행시키는 코드이다.
유년기 단계, 즉 t=2이 주어지면 호출되고 관련 스레드들이 생성된다.
그리고 스레드들의 반환 값을 받아 on_pushButton_clicked()로 반환해서 그에 따른 이미지를 출력한다.

```
void *t_function3(void *data){ //햇빛 이벤트 발생 스레드
        srand(time(NULL));
        //랜덤값을 받기 위해 초기화, seed를 고정하지 않기 위해 time(NULL)을 사용
        int sun_value = 0;
        static int retval = 999; //스레드 반환값 선언
        int count = 1; //검사 횟수
        write(dot_dev, setzero, sizeof(setzero)); //아무 출력이 없는 상태를 출력
        while(count <= (*((int *)data))*2) {
            //여기서 (*((int *)data))은 t값으로 2
            if(temp_th != 2) {
                pthread_exit((void*)&retval);
            }
            //다른 쓰레드와 동기화를 하기 위해 temp_th가 2인지를 확인 만약 다른 스레드에서 먼
              저 종료된 스레드가 있으면 의미없는 스레드 반환 값을 리턴하고 종료
            sleep(30); //30초를 sleep함으로 2분 동안 4번 수행하게 됨
            if(sun_value==3) { //햇빛 값이 3이 나오면 그 값을 반환
                pthread_exit((void*)&retval);
                //스레드 리턴 값을 3으로 하여 즉시 스레드 종료
            }
            sun_value=rand()%3+1; //햇빛의 값을 랜덤으로 1~3사이의 숫자를 받아옴
            retval = sun_value; //스레드 반환 값에 햇빛 값을 저장
            write(dot_dev, sun_dot[sun_value-1], sizeof(sun_dot[sun_value-1]));
            //dot matrix에 햇빛 모양 출력
            sleep(1);
            count ++;
        }
        pthread_exit((void*)&retval); //스레드 리턴 값 1 이나 2(다른 햇빛 값)반환
}
```

햇빛 이벤트 발생 스레드로 2분동안 4번 수행되게 된다.
sun_value의 값을 랜덤으로 1~3사이의 숫자를 받아온다. 만약 sun_value값이 3이라면 리턴 값을 3으로 하여 즉시 스레드를 종료한다.
4번 수행되었다면 반복문을 탈출해 다른 sun_value값(1이나 2)를 반환하며 스레드를 종료한다.
temp_th2는 전역변수로 유년기 단계의 스레드들이 같이 종료될 수 있게 공유한다. 초기의 값은 2로 설정되어있고, 스레드에서 조건에 따라 종료가 되면 2에서 다른 값으로 바꾸므로 다른 스레드들 역시 바뀐 temp_th2에 의해 종료된다.

```
void *t_function4(void *data) //Push Swtich 제어 스레드
{
    int retval; //스레드 반환값 선언
    retval = (int*)malloc(sizeof(int)); //사망 원인을 저장하기 위한 동적 할당

    while(1){
        if(temp_th != 2) {
            pthread_exit((void*)&retval);
        }
        //Push Switch 값을 읽어옴
        read(push_switch_dev, &push_sw_buff, buff_size);
        //버튼 0은 플래시 버튼으로 바로 사망
        if(push_sw_buff[0]==1){
            retval = 0;
            temp_th2 = -1; //temp_th2을 -1로 바꾸면서 다른 스레드에 종료를 알려주는 동시에
                           플래시 의한 사망임을 알려줌
            return (void*)retval;
        }
        //버튼 1은 터치 버튼으로 스트레스를 올림
        if(push_sw_buff[1]==1){
            stress = stress+1;
            //스트레스가 하나 올라갈 때마다 LED를 하나씩 켬
            write(led_dev,&led_data[stress], 1);
            if(stress > 6){ //LED 8개가 다 켜진 상태면 스트레스에 의한 사망
                retval = 2;
                temp_th2 = 0;//temp_th2을 0으로 바꾸면서 다른 스레드에 종료를 알려주는 동시에
                             스트레스에 의한 사망임을 알려줌
                pthread_exit((void*)&retval); //스레드 리턴값을 2로하여 즉시 스레드종료
            }
        }
        //버튼 3, 4는 산소값을 변경하고 변경된 값을 LCD에 출력
        if(push_sw_buff[3]==1){
           pthread_mutex_lock(&mutex);
           oxygen = oxygen+10;
           text_print();
           pthread_mutex_unlock(&mutex);
          //다른 스레드들이 OXYGEN, TEMP에 동시접근을 못 하도록 뮤텍스 사용
        }
        if(push_sw_buff[4]==1){
           pthread_mutex_lock(&mutex);
           oxygen = oxygen-10;
           text_print();
           pthread_mutex_unlock(&mutex);
        }
```

```
        //버튼 6, 7는 수온 값을 변경하고 변경된 값을 LCD에 출력
        if(push_sw_buff[6]==1){
          pthread_mutex_lock(&mutex);
          temp = temp+1;
          text_print();
          pthread_mutex_unlock(&mutex);
        }
        if(push_sw_buff[7]==1){
          pthread_mutex_lock(&mutex);
          temp = temp-1;
          text_print();
          pthread_mutex_unlock(&mutex);
        }
        sleep(1);
    }
}
```

Push Switch 제어 스레드로 Push Switch의 값을 읽어온다.
0번이면 플래시로 바로 사망하고 temp_th2 변수를 -1로 변경해 플래시 사망 이벤트를 run() 함수에 알리는 동시에 다른 스레드가 종료될 수 있게 한다.
1번이면 누를 때마다 스트레스가 1씩 증가한다. 스트레스가 7 이상이 되면 temp_th2 변수를 0으로 바꾸어 스레드가 종료됨을 다른 스레드에 알림과 동시에 스트레스로 인한 사망 이벤트를 run() 함수에 반환한다. 또한, retval=2를 반환해 스트레스로 인한 사망 이벤트를 run() 함수에 반환한다.
3번이면 산소를 10mg/L씩 올려주고 4번이면 10mg/L씩 내려준다.
6번이면 수온을 1°C씩 올려주고 7번이면 1°C씩 내려준다.
전역변수로 선언된 산소변수와 수온변수를 해당 스레드가 사용 중일 때 다른 스레드에서 사용을 제한하도록 뮤텍스를 사용했다.

```
void *t_function5(void *data) { //수온, 산소량 체크 스레드
    int retv = 999; //스레드 반환값
    srand(time(NULL));
    int count = 1; //검사 횟수
    while(count <= (*((int *)data))*3) {
        //여기서 (*((int *)data))은 t값으로 2
        //유년기 단계는 2분이므로 검사를 20초마다 한다고 할 때 6번 검사를 하면 됨
        if(temp_th != 2) {
            retv = 0;
            pthread_exit((void*)&retv);
        } //다른 쓰레드와 동기화를 하기 위해 temp_th2가 2인지를 확인 만약 다른 쓰레드에서 먼
            저 종료된 쓰레드가 있으면 의미 없는 쓰레드 반환 값을 리턴하고 종료
        if(num == 8) { //스트레스가 8이면 사망 값 반환
            retv = 2;
            temp_th2 = 0; //temp_th2을 0으로 바꾸면서 다른 스레드에게 종료를 알려주는 동시에
                          스트레스에의한 사망임을 알려줌
            pthread_exit((void*)&retv);//스레드 리턴값을 2로하여 즉시 스레드 종료
        }
        sleep(20); //20초마다 검사
        pthread_mutex_lock(&mutex);
        temp = rand()%40+10; //랜덤 값으로 산소,수온 변경
        oxygen = rand()%150 + 20;
        text_print(); //변경된 값 LCD 출력
        //적정 산소 70~150mg/L 수온 20~30°C 체크
        if(temp < 20 || temp > 30 || oxygen < 70 || oxygen > 150) {
            stress ++; //범위 밖이면 스트레스 증가
        }
        pthread_mutex_unlock(&mutex);
        count ++;
        //수행 시간 측정을 위한 count 1증가, 20초 마다 1씩 증가
        //검사 횟수인 count가 6이 넘어가면 반복문 탈출(유년기 단계 끝)
    }
    retv = 0; //생존 시 0 반환
    temp_th2=1;
    pthread_exit((void*)&retv); //스레드 리턴 값이 0이면 노년기 단계로 진화
}
```

*data 함수에 넘어온 인자 값은 앞에서 본 run()함수의 t값으로 2이다.
검사 횟수 count를 1로 설정하고 sleep(20)으로 20초마다 수온과 산소량을 검사한다. 그러면 count의 횟수가 6번이면 유년기 단계 2분을 설정할 수 있게 된다.
수온과 산소량을 검사할 때 전역변수로 선언된 oxygen과 temp를 사용함으로 다른 스레드와 충돌이 발생하지 않도록 뮤텍스를 사용해 제한을 두었다.

만약 이때 LED가 다 켜졌다면 스레드 리턴 값을 2로 설정하고, temp_th2를 0으로 변경하여 다른 스레드들이 모두 종료될 수 있게 하며, run() 함수에 스트레스로 인한 사망 이벤트를 알린다.
만약 검사 횟수 count가 6을 넘어가서 반복문을 탈출했다면 리턴 값을 0으로 설정하여 스레드를 종료한다.

4-3-3. 노년기

노년기 단계에서는 run함수의 인자에 t=3가 들어가게 된다.

```
clear_led(); //LED 초기화
QApplication::processEvents(); //사진을 정상적으로 불러주기 위해 호출
//노년기 단계 호출 함수
int retVal3=run(3);
    //스레드로부터 1를 반환받으면 햇빛으로 인한 사망 화면 출력
    if(retVal3 == 1) {
        QPixmap pix8("sun.png");
        ui->label_pic->setPixmap(pix8);
        ui->label_pic->setPixmap(pix8.scaled(w,h));
        sleep(3);
        return;
    }
    //스레드로부터 3를 반환받으면 강한 물살로 인한 사망 화면 출력
    else if(retVal3 == 3) {
        QPixmap pix9("water.jpg");
        ui->label_pic->setPixmap(pix9);
        ui->label_pic->setPixmap(pix9.scaled(w,h));
        sleep(3);
        return;
    }
   //스레드로부터 4를 반환받으면 자연사 사망 화면 출력
    else if(retVal3 == 4) {
        QPixmap pix10("die.jpg");
        ui->label_pic->setPixmap(pix10);
        ui->label_pic->setPixmap(pix10.scaled(w,h));
        sleep(3);
        return;
    }
    //스레드로부터 5를 반환받으면 스트레스로 인한 사망 화면 출력
    else if(retVal3 == 5) {
        QPixmap pix11("stress.jpg");
        ui->label_pic->setPixmap(pix11);
        ui->label_pic->setPixmap(pix11.scaled(w,h));
        sleep(3);
        return; //return으로 on_pushButton_clicked()함수 탈출
    }
```

```
    //스레드로부터 6를 반환받으면 플래시로 인한 사망 화면 출력
    else if(retVal3 == 6) {
        QPixmap pix12("flash.jpg");
        ui->label_pic->setPixmap(pix12);
        ui->label_pic->setPixmap(pix12.scaled(w,h));
        sleep(3);
        return;
    }
    pthread_mutex_destroy(&mutex); //모든 함수가 종료되므로, mutex를 소멸
```

노년기 단계에서는 on_pushButton_clicked()함수에서 t를 3으로 설정하여 run()함수에 넘겨주게 된다. 그리고 run()반환 값에 따라 다른 화면이 출력되게 된다.
대부분 유년기와 동일하나 다른 점은 스레드로 3을 반환받으면 강한 물살로 인해 사망한 화면을 출력한다. 그리고 4를 반환받으면 시간이 지나서 자연사한 화면을 출력한다.
스레드 내에서 전역변수의 값을 이용할 때가 있어 mutex를 사용하였고, 노년기의 실행이 끝난 후(프로그램이 종료될 때) mutex를 소멸시켜준다.

```
//int run(int t) 함수에서 호출되는 노년기 단계(t==3) 스레드 생성 함수
//대부분 유년기와 동일하므로 다른점만 설명
else if(t == 3) {
      thr_id = pthread_create(&p_thread[2], NULL, t_function3, (void *)&t);
      //t_function6 : 쓰레드가 생성되고 나서 실행되는 함수(물살 관련 함수)
      thr_id = pthread_create(&p_thread[5], NULL, t_function6, (void *)&t);
      if(thr_id<0){
            perror("thread create error : ");
            exit(0);
      }
      pthread_join(p_thread[2], &tret_sun);
      pthread_join(p_thread[5], &tret_current);
      thr_id = pthread_create(&p_thread[3], NULL, t_function4, (void *)&t);
      thr_id = pthread_create(&p_thread[4], NULL, t_function5, (void *)&t);
      if(thr_id<0) {
            perror("thread create error : ");
            exit(0);
       }
       pthread_join(p_thread[3], &tret_stress);
       pthread_join(p_thread[4], &tret_check);
       if(temp_th2 == -1) return 6;
       else if(*((int*)tret_stress) == 2 || *((int*)tret_check) == 999) return 5;
       else if((*((int*)tret_sun)) == 3) return 1;
       //tret_current 값이 150보다 클 때는, 강한 물살이 원인인 3을 반환
       else if((*((int*)tret_current)) > 150) return 3;
       else return 4; //아무것도 해당하지 않으면 4를 반환하고 자연사
}
```

int run(int t)에 포함된 노년기 단계에서 사용되는 스레드들을 실행시키는 코드이다.
노년기 단계, 즉 t=3이 주어지면 호출되고 관련 스레드들이 생성된다.
그리고 스레드들의 반환 값을 받아 on_pushButton_clicked()로 반환해서 그에 따른 이미지를 출력한다

이 함수를 제외하고 유년기와 동일

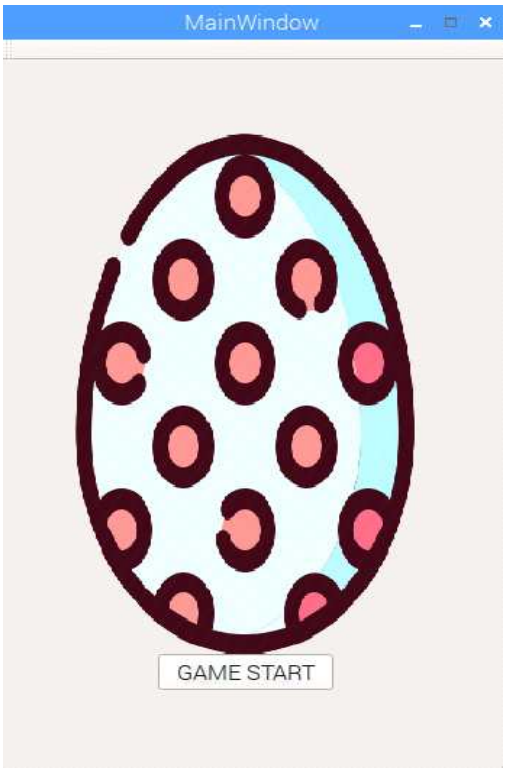
```
void *t_function6(void *data) //물살 이벤트 관련 스레드
{
    srand(time(NULL));
    //랜덤값을 받기 위해 초기화, seed를 고정하지 않기 위해 time(NULL)을 사용
    int water_value = 0;
    //물살 값, step_motor의 빠르기 값 (0:느림, 250: 빠름)
    static int retv = 0;//스레드 함수 반환값
    if(temp_th2 != 2) {
            pthread_exit((void*)&retv);
    } //다른 쓰레드와 동기화를 하기 위해 temp_th2가 2인지를 확인 만약 다른 쓰레드에서 먼
        저 종료된 쓰레드가 있으면 의미 없는 쓰레드 반환 값을 리턴하고 종료
    int temp =1; //step_motor의 실행값과 방향 값을 위한 임시값
    int temp_not = 0;  //step_motor의 초기화를 위한 임시값(작동x)
    memset(motor_state,0,sizeof(motor_state));
    //Step Motor를 제어하기 위한 배열 motor_state를 0으로 초기화
    motor_state[0] = (unsigned char)temp_not;
    //작동하지 않을 때이므로 motor action값은 temp_not(0)으로 설정
    motor_state[1] = (unsigned char)temp_not;
    //작동하지 않을 때이므로 motor 방향 값은 temp_not(0)으로 설정
    motor_state[2] = (unsigned char)temp_not;
    //작동하지 않을 때이므로 motor 속도 값은 temp_not(0)으로 설정

    write(step_motor_dev, motor_state, 3);
    //방향은 0(왼쪽), 빠르기는 0, action은 0(stop)으로 Step Motor를 실행

    int count = 1;
    while(count <= (*((int *)data))*2) {
        //여기서 (*((int *)data))은 t값으로 3
        //노년기 단계는 3분이므로 검사를 30초마다 한다고 할 때 6번 검사를 하면 됨
```

```
        sleep(30); //30초마다 검사
        if(water_value > 150) {
            //step_motor의 값이 150초과일 때, 물살 값이 너무 빨라서 사망
            temp_th2 = 0;  //스레드가 종료됨을 다른 스레드에게 알리기 위해 값 설정
            pthread_exit((void*)&retv);
            //스레드 리턴값을 water_value로 설정해 즉시 스레드 종료
        }
       //Step Motor의 빠르기는 100부터 170사이에서 랜덤으로 받아옴
        water_value=rand()%(170+1-100)+100;
        retv = water_value; //스레드 리턴값을 water_value(물살 값)으로 설정
        memset(motor_state,0,sizeof(motor_state));
        //Step Motor를 제어하기 위한 배열 motor_state를 0으로 초기화
        motor_state[0] = (unsigned  char)temp;
        //작동할 때이므로 motor action값은 temp(1)으로 설정
        motor_state[1] = (unsigned  char)temp;
        //작동할 때이므로 motor 회전 방향 값은 temp(1) 오른쪽으로 설정
        motor_state[2] = (unsigned  char)water_value;
        //작동할 때이므로 motor 속도값은 water_value(100~170사이의 값)로 설정
        write(step_motor_dev, motor_state, 3);
        //방향은 1(오른쪽), 빠르기는 랜덤 값인 water_value, action은 1(start)으로
          Step Motor를 실행
        count ++;
        //수행 시간 측정을 위한 count를 1증가, 30초마다 1씩 증가함
        //검사 횟수인 count가 6을 넘어가면 반복문 탈출(노년기 단계 끝)
    }
    temp_th2 = 1;  //스레드가 종료됨을 다른 스레드에게 알리기 위해 값 설정
    pthread_exit((void*)&retv);
    //스레드 반환값을 water_value로 설정해 스레드 종료
}
```

*data 함수에 넘어온 인자 값은 앞에서 본 run()함수의 t값으로3이다.
검사 횟수 count를 1로 설정하고 sleep(30)으로 20초마다 모터의 속도를 검사한다. 그러면 count의 횟수가 6번이면 노년기 단계 3분을 설정할 수 있게 된다.
만약 모터의 속도, 즉 물살의 세기가 150을 넘어가면 스레드 리턴 값을 water_value로 설정하여 즉시 스레드를 종료한다.
그리고 temp_th2의 값을 스레드를 종료할 때, 값을 0 또는 1로 설정하여 다른 스레드들을 종료할 수 있도록 한다. 즉, temp_th2의 값이 초기화시킨 2의 값이 아니라면 다른 스레드 하나가 종료되었기 때문에, 스레드를 종료시켜준다.
만약 검사 횟수 count가 6을 넘어가서 반복문을 탈출했다면 리턴 값을 water_value로 설정하여 스레드를 종료한다.

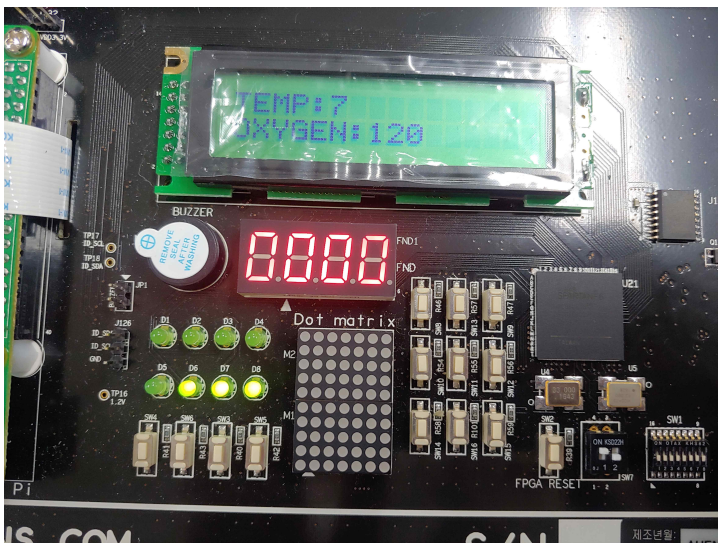
5. 실행결과 및 분석

5-1. 실행 결과 정리



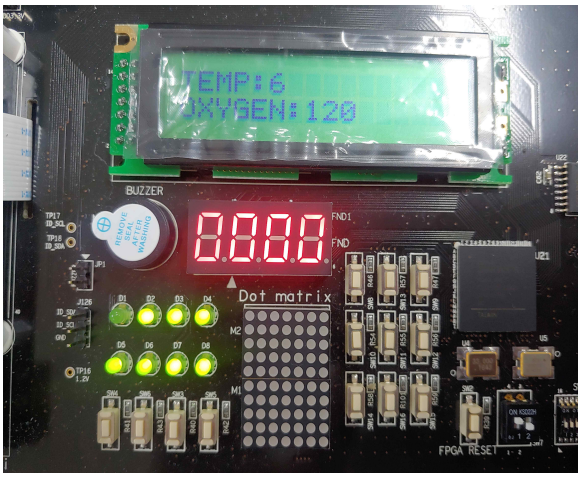
처음 게임 화면

게임 시작 버튼을 누르면 알 사진이 뜨며 게임 시작



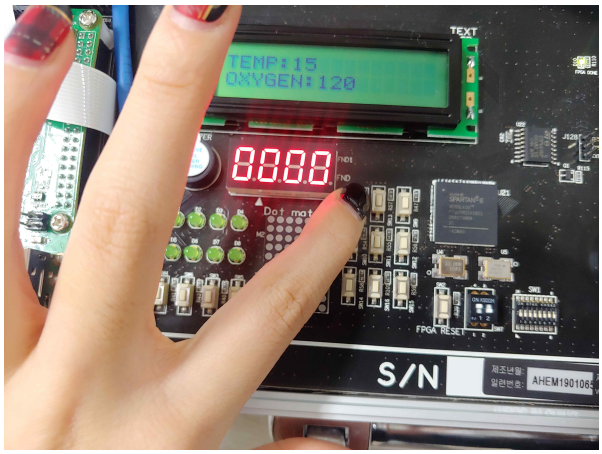
초기 LCD화면

온도 7°C, 산소량 120mg/L

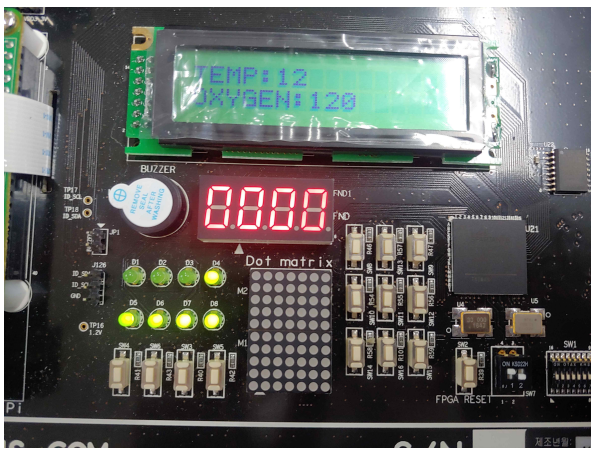


30초 동안 스트레스가 8이상 쌓여서 LED가 다켜지면 알에서 부화하지 못해서 사망

30초가 지나고 알에서 부화하면 유년기 단계로 진입



플래시로 지정된 Push Switch 버튼을 누르면 플래시로 인해서 사망



개복치를 터치해서 만지거나 수온이나 산소량을 맞추지 못해서 스트레스가 쌓이면 스트레스로 인한 돌연사로 사망

랜덤으로 받아온 햇빛 값에 의해 햇빛이 너무 강해서 사망(구현하지 못함)

노년기 단계에서 랜덤으로 받아온 물살 값에 의해 너무 물살이 빨라서 사망(구현하지 못함)

유년기에서 아무일 없이 2분이 지나면 노년기 단계로 진입
노년기에서는 유년기에서 발생한 사망이 모두 똑같이 발생할 수 있고 아무일 없이 3분이 지나면 자연사

5-2. 실행 결과에 대한 분석

제일 먼저 main()함수를 호출하는 과정에서 MainWindow창을 생성하고 만든 MainWindow창을 사용자에게 보여준다. 이 때, 사용자는 close 버튼을 누를 수 있고, Game Start 버튼을 누를 수도 있다. close 버튼을 누르게 되면 프로그램이 종료되며 게임 또한 종료된다. Game Start버튼을 누르게 되면 알부터 시작하여 게임을 진행한다.

알에서는 게임이 30초 동안 진행된다. 온도는 7°C, 산소는 120mg/L으로 시작하여, 적정 온도와 산소를 유지하면 알에서 유년기로 진행된다. 유년기로 진행하지 못했을 경우에는 '알에서 부화하지 못 함' 이미지를 띄우며 게임은 종료된다. 게임을 다시 시작하기 위해서는 프로그램을 다시 실행하여야 한다.

유년기 때, 사망 원인으로는 햇빛, 온도, 산소, 터치, 플래시가 있다. 햇빛이 너무 세면 dot matrix에 강한 햇빛을 표현하고 '아침해가 너무 강해서'이미지를 띄우며 사망한다.
온도와 산소, 터치를 적정 수준에 맞춰놓지 못할 경우에는 '돌연사'이미지를 띄우며 사망한다.
플래시의 버튼을 1번이라도 누르게 된다면 '플래시'이미지를 띄우며 사망하게 된다.
플래시는 햇빛과 똑같이 dot matrix에 표현하며, 온도와 산소는 lcd로 결과가 출력된다. 스트레스는 led로 최대 8까지 표현된다.
위와 같이 개복치가 사망했을 때는 게임을 다시 실행해야 한다. 플래시 버튼을 누르지 않고, 산소와 온도, 터치를 적정 수준에 맞추고 운이 좋아 햇빛이 강하지 않다면 개복치는 노년기로 넘어가게 된다.

노년기에서는 개복치는 하나를 제외하면 유년기와 모든 조건이 똑같다. 노년기에서는 물살 속도를 100~170 사이의 값으로 랜덤으로 받으며 이를 step motor에 표현한다.
150이 넘게 되면 속도가 너무 빨라 '착수 시의 충격'이미지를 띄우며 개복치는 사망하게 된다.
모든 조건에 걸리지 않으면 '죽음을 맞이하다'이미지를 띄우며 게임은 종료된다.

5-3. 디바이스 드라이버 개선사항

```
ssize_tiom_fpga_text_lcd_write(struct file *inode, const char *gdata, size_t
length, loff_t *off_what)
{
   int i;
   unsigned char value[33];
   const char* tmp = gdata;

   if (copy_from_user(&value, tmp, length))
      return-EFAULT;

   value[length] = 0;
   printk("GetSize:%d/String:%s\n", length, value);
   for (i = 0; i<length; i++)
   {
      iom_fpga_itf_write((unsigned  int)IOM_FPGA_TEXT_LCD_ADDRESS  +  i,
value[i]);
   }
   return length;
}
```

위의 코드는 LCD 드라이버의 write함수이다. 한번에 LCD창에 띄울 수 있는 글자 수는 32로 Write함수 내에서 입력받은 문자열의 크기를 계산하고 32가 넘어가면 예외를 띄우게 디바이스 드라이버를 개선하였다. 개선된 코드는 아래와 같다.

```
ssize_tiom_fpga_text_lcd_write(struct file *inode, const char *gdata, size_t
length, loff_t *off_what)
{
    int i;
    unsigned char value[33];
    const char* tmp = gdata;

    int data_length = strlen(tmp);
    //data_length변수는 사용자가 입력한 문자열의 크기를 나타냄
    if (data_length 〉 32) {
        printk("Too long string!");
        return;
    } //최대 출력할 수 있는 문자열의 크기는 32로, 넘으면 오류를 출력 후 반환

    if (copy_from_user(&value, tmp, length))
        return-EFAULT;
    value[length] = 0;
    printk("GetSize:%d/String:%s\n", length, value);
    for (i = 0; i〈length; i++)
    {
        iom_fpga_itf_write((unsigned  int)IOM_FPGA_TEXT_LCD_ADDRESS  +  i,
value[i]);
    }
    return length;
}
```

위와 같이 개선하면 응용 프로그램을 작성할 때, 예외처리를 디바이스 내에서 해주기 때문에 사용자가 따로 예외처리를 해주지 않아도 되어서 편리하다. 그리고 어떤 오류인지도 명확히 알 수 있다.

```
ssize_t iom_led_write(struct file *inode, const char *gdata, size_t length,
loff_t *off_what)
{
        unsigned char value;
        const char *tmp = gdata;

        if (copy_from_user(&value, tmp, 1))
                return -EFAULT;

        iom_fpga_itf_write((unsigned int)IOM_LED_ADDRESS, value);

        return length;
}
```

led에 데이터를 입력할 때 위와 같은 write 함수를 호출하게 된다. 이 때 led를 한 개씩 켜주는 경우에는 led 각 자리수의 이진수 값을 넣어주면 되지만 led를 1개, 2개, 3개 이렇게 순차적으로 다 키려고 할 경우에는 led 자리수 값을 일일이 계산해서 더해야 하므로 사용자에게 불편함을 일으킬 수있다.

```
#define IOM_LED_MAJOR 260
#define IOM_LED_NAME "fpga_led“
...
int led_data[8]={1,3,7,15,31,63,127,256};
```

만약 그런 사용자를 생각해 미리 디바이스 드라이버 파일에 led를 순차적으로 다 켜기 위한 이진수 값을 미리 계산해서 배열로 선언해 놓는다면 편할 것이다.

6. 결론

6-1. 평가 및 반성

다양한 요인에 의해 사망 이벤트가 발생하는 개복치 프로그램을 만들기 위해 여러 스레드 함수가 사용되었으며, 그로 인해 프로그램이 복잡해졌다.
돌아갈 것으로 생각하고 프로그램을 작성하였나 역시 다중 스레드의 오류로 결국 최종 시연에서 스레드 하나를 지우게 되었다. 시간이 부족해서 스레드를 한 번에 종료시키는 효율적인 방법을 생각하지 못한 것이 아쉽다.
또, 주로 sleep 함수를 사용해 각종 검사나 이벤트를 관리하기 때문에 예외사항이 많이 발생 될 것으로 예상한다. 또한, 게임인데 불구하고 화면에 출력되는 정적인 이미지는 게임의 충분한 시각화를 제공하지 못했다는 생각이 든다.
조금 더 시간을 들여 qt creator를 다루는 방법을 찾지 못한 점이 반성해야 할 부분이라고 생각한다.

6-2. 향후 계획

임베디드의 프로젝트를 통해 qt creator라는 개발환경을 알게 되었으며, 이를 이용해 GUI 프로그램을 효율적으로 개발할 수 있다는 생각을 하게 되었다.
따라서 팀원들과 작업할 시간이 생긴다면, 원래의 계획대로 화면의 좌표값을 이용해 개복치가 움직이는 모습이나 자연스럽게 화면이 넘어가는 모습을 구현할 것이다.
또한, 제대로 실행되지 못했던 QThread를 사용해 좀 더 체계적으로 스레드를 관리함으로써 좀 더 완성도 있는 모습으로 프로그램을 발전시켜나갈 계획이 있다.

7. 참고 문헌

・Happy Programmer 블로그, “C언어 스레드 생성” https://bitsoul.tistory.com/157 2020.12.2

・Notepad 블로그, “Qt Label 이미지 삽입” https://m.blog.naver.com/PostView.nhn?blogId=hextrial&logNo=221109232458&proxyReferer=https:%2F%2Fwww.google.com%2F 2020.12.5.

・위키 백과, “스레드” https://ko.wikipedia.org/wiki/%EC%8A%A4%EB%A0%88%EB%93%9C_(%EC%BB%B4%ED%93%A8%ED%8C%85) 2020.12.6.

・코딩초보의 블로그, “QApplication::processEvents()” https://coding-chobo.tistory.com/34 2020.12.07