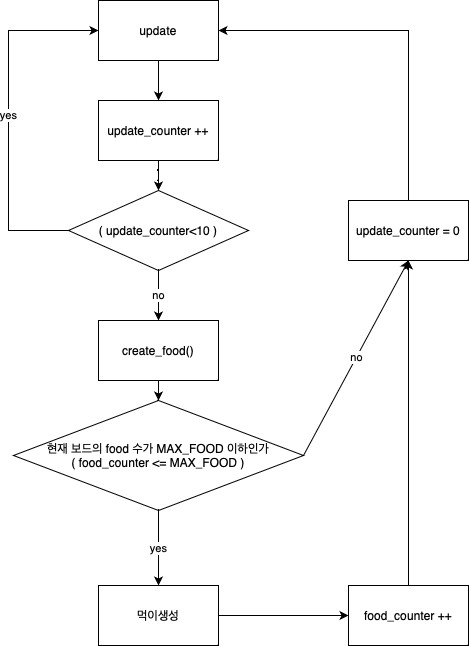
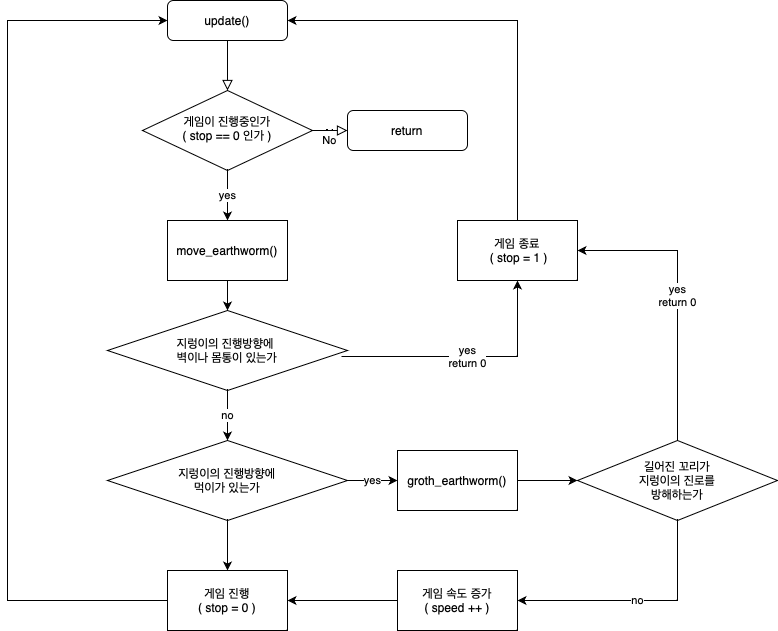
**Final Project**

**지렁이 게임**

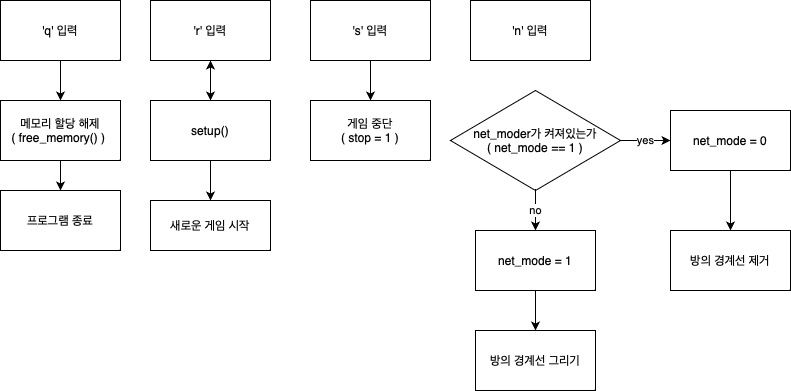
학번 / 이름 : 20170024 / 김병천

1. **프로젝트 목표, 실험환경**
   1. **프로젝트 목표**OpenFrameWork를 이용하여 Snake Game으로 알려진 게임을 모방한 지렁이 게임을 구현한다.   
      지렁이는 방을 돌아다니며, 먹이를 먹으면 성장한다.   
      벽에 머리를 박거나 자신의 몸통에 머리를 박으면 게임이 종료된다.
   2. **실험환경**Xcode 14.1 version과 Openframeworks of\_v0.11.2\_osx\_release환경에서 진행
2. **각 변수에 대한 설명**
   1. **#define BOARD\_SIZE**전체 게임보드의 크기. BOARD\_SIZE \* BOARD\_SIZE 규모의 게임보드에서 게임을 진행한다.
   2. **#define ROOM\_SIZE**게임보드의 각 방의 크기. 한 방의 크기는 ROOM\_SIZE \* ROOM\_SIZE이다.
   3. **#define MAX\_FOOD**전체 먹이의 갯수. 게임 보드에 동시에 존재하는 먹이의 갯수가 MAX\_FOOD갯수를 넘지 않는다.
   4. **struct room**강 방에 대한 정보를 저장할 구조체
      1. **x\_pos:** 방의 x좌표 정보를 저장
      2. **y\_pos:** 방의 y좌표 정보를 저장
      3. **state:** 방의 상태를 저장  
         방의 상태는 enum room\_condition{EMPTY,BODY,FOOD} 중 하나.
   5. **room board[][]**struct room[BOARD\_SIZE][BOARD\_SIZE]로 BOARD\_SIZE \* BOARD\_SIZE 규모의 게임보드 구성
   6. **struct body\_node**지렁이의 몸통에 대한 정보를 저장.   
      각 몸통은 doubly linked list로 머리부터 꼬리까지 연결되어있다.
      1. **room\_row:** 몸통이 위치한 방의 row정보 저장.
      2. **room\_col:** 몸통이 위치한 방의 column정보 저장
      3. **pre\_body:** 현재 몸통 바로 전의 몸통에 대한 포인터 저장
      4. **belong\_body:** 현재 몸통 바로 뒤의 몸통에 대한 포인터 저장
   7. **body\_node\* head**지렁이의 머리 역할을 하는 몸통의 포인터 정보를 저장
   8. **body\_node \* tail**지렁이의 꼬리 역할을 하는 몸통의 포인터 정보를 저장.
   9. **int direction**지렁이의 진행방향에 대한 정보를 저장  
      지렁이의 진행방향은 enum direction\_information{NORTH, SOUTH, EAST, WEST} 중 하나
   10. **double speed**게임의 진행 속도.  
       speed값은 지렁이가 먹이를 먹을 때마다 일정 정도씩 증가하며, ofSetFrameRate((int)(speed))로 프레임을 갱신하는 속도를 증가시켜가며 게임 속도를 조절
   11. **int stop**‘s’키를 누르거나 지렁이가 더 이상 진행이 불가능한 상황이 오면 활성화. stop이 활성화되면 update작업과 draw작업을 중단한다.
   12. **int is\_update**update()도중 update()가 호출되지 못하도록 lock.   
       update에 락을 따로 걸어주지 않아도 당연히 openFrameWork 내부적으로 update에 대한 락이 걸려있다고 믿지만, 만에 하나 자체적 update락이 없을경우 게임에 심각한 오류가 발생할 수 있기 때문에 update락을 걸어줬다.
   13. **int update\_counter**이 게임에서 먹이는 10번의 프레임갱신 마다 생성되도록 짜여져있다. 이를 위해 10번의 프래임갱신을 카운트해줄 변수이다.
   14. **direction\_changed\_flag**한번 방향을 바꿔준 후에는 한번의 draw가 적용된 이후에 방향을 바꿔줄 수 있도록 설정한 lock.   
       이 변수가 없을 경우, 현재 NORTH방향으로 가고있고, EAST와 SOUTH를 한번의 draw가 실행되기 이전에 빠르게 누를 경우, direction이 NORTH에서 SOUTH로 변하게된다. 그 경우, 현재의 진행방향과 반대방향에는 당연히 몸통이 있기 때문에 몸통을 만나 게임이 바로 종료되게된다.  
       따라서 진행방향과 반대방향으로는 진행하지 못하도록 lock을 걸었다.
   15. **net\_mode**방의 경계들이 명확히 알 수 있도록 방의 경계를 그려주는 모드.   
       ‘n’을 눌러 활성화 할 수 있다.
   16. **food\_counter**현재 게임보드 내에 동시에 존재하고 있는 먹이의 갯수를 카운트.
3. **각 함수에 대한 설명**
   1. **keyPressed()**
      1. **key == ‘s’**게임을 중단.
      2. **key == ‘r’**새로운 게임을 시작.   
         ‘s’를 눌러 중단된 시점부터 재개하는 명령어가 아님에 유의.
      3. **key == ‘n’**net\_mode활성화/비활성화
      4. **key == ‘q’**메모리 할당해제 및 게임 완전 종료
   2. **#define GET\_POS**
   3. **초기화 함수들**
      1. **board\_init :**게임판 초기화  
         전체 게임판을 BOARD\_SIZE \* BOARD\_SIZE로 설정  
         각 방마다의 x좌표와 y좌표를 표기  
         각 방의 상태를 EMPTY로 설정
      2. **earthworm\_init:**지렁이 초기화.   
         처음에는 머리와 꼬리 2개의 몸통만으로 게임을 시작.   
         지렁이의 시작방향은 북쪽  
         지렁이의 머리와 꼬리가 위치한 방의 상태를 BODY로 설정
   4. **게임 진행 함수들**
      1. **int move\_earthworm:**매 프레임이 갱신될 때마다 지렁이 이동  
         지렁이가 더이상 움직일 수 없을 경우 0을 반환 **🡪 게임 종료**그 외의 경우 1은 반환 **🡪 게임 진행**update시 호출되며, 지렁이의 꼬리를 머리로 가져와 지렁이가 움직이는 것처럼 보이게 만든다.  
         꼬리가 머리에 붙는 위치는 direction변수에 따라 달라진다.   
         지렁이가 이동함에 따라 방의 상태도 새 머리가 들어선 방을 BODY로, 꼬리가 없어진 방을 EMPTY로 변경  
           
         지렁이가 먹이를 만날 경우 groth\_earthworm함수 호출
      2. **create\_food:**10번의 프레임 갱신마다 랜덤한 빈 방에 FOOD를 생성
      3. **int groth\_earthworm:**지렁이가 먹이를 먹을 때마다 지렁이의 길이 증가  
         linked list의 tail의 next\_pointer에 새로 생긴 몸통을 붙인 후, tail을 새로 생긴 몸통으로 이동.  
         지렁이의 길이가 증가되서 지렁이의 머리의 진로를 방해할 경우 0을 반환 **🡪 게임 종료**그 외의 경우 1을 반환 **🡪 게임 진행**  
         지렁이가 먹이를 먹으면 속도가 증가하고, 증가된 속도를 ofSetFrameRate (speed)에 넣어 게임의 속도를 조정한다.   
         게임 속도는 먹이를 3~4번 먹을 때마다 증가한다.
   5. **draw 함수들**
      1. **draw\_empty\_board**게임판의 경계를 검은색 선으로 그려준다.
      2. **draw\_net**각 방의 경계들을 초록색 선으로 그려준다.
      3. **draw\_earthworm**지렁이를 그려준다  
         지렁이의 몸통이 linked\_list형태로 구현되어 있기 때문에 머리부터 꼬리까지 while문을 통해 몸통 하나씩 탐색해가며 그려준다.  
         struct body\_node에 저장된 room\_rom, room\_col정보를 이용해 몸통이 있는 방의 번호를 알아오고, GET\_POS함수를 이용해 몸통이 그려질 x,y좌표를 구해와 몸통을 그려준다.  
         몸통이 머리일경우, 눈알을 그려주는 작업을 추가적으로 진행해준다.  
         눈동자의 방향은 지렁이의 머리와 그 다음 몸통의 방향을 계산하는 수식을 통해 결정된다.
      4. **draw\_food**모든 방을 순회하면서 방의 상태가 FOOD인 방에 먹이를 그려준다.
   6. **종료 함수**
      1. **free\_memory**지렁이의 몸통의 메모리할당을 해제해준다.  
         지렁이가 링크드리스트형태로 구현되어 있기 때문에 while문으로 몸통을 하나씩 탐색해가며 할당을 해제한다.   
           
         free\_memory함수는 setup()과 EXIT\_ 직전에 수행된다.  
         ‘r’키를 눌러 게임을 다시할 경우 기존에 할당되어 있던 부분의 할당을 해제해야하기 때문에 setup()시에도 free\_memory()를 호출했다.
4. **프로젝트의 Flow차트 및 시간/공간복잡도**
   1. **먹이생성  
      **
   2. **지렁이 움직임**

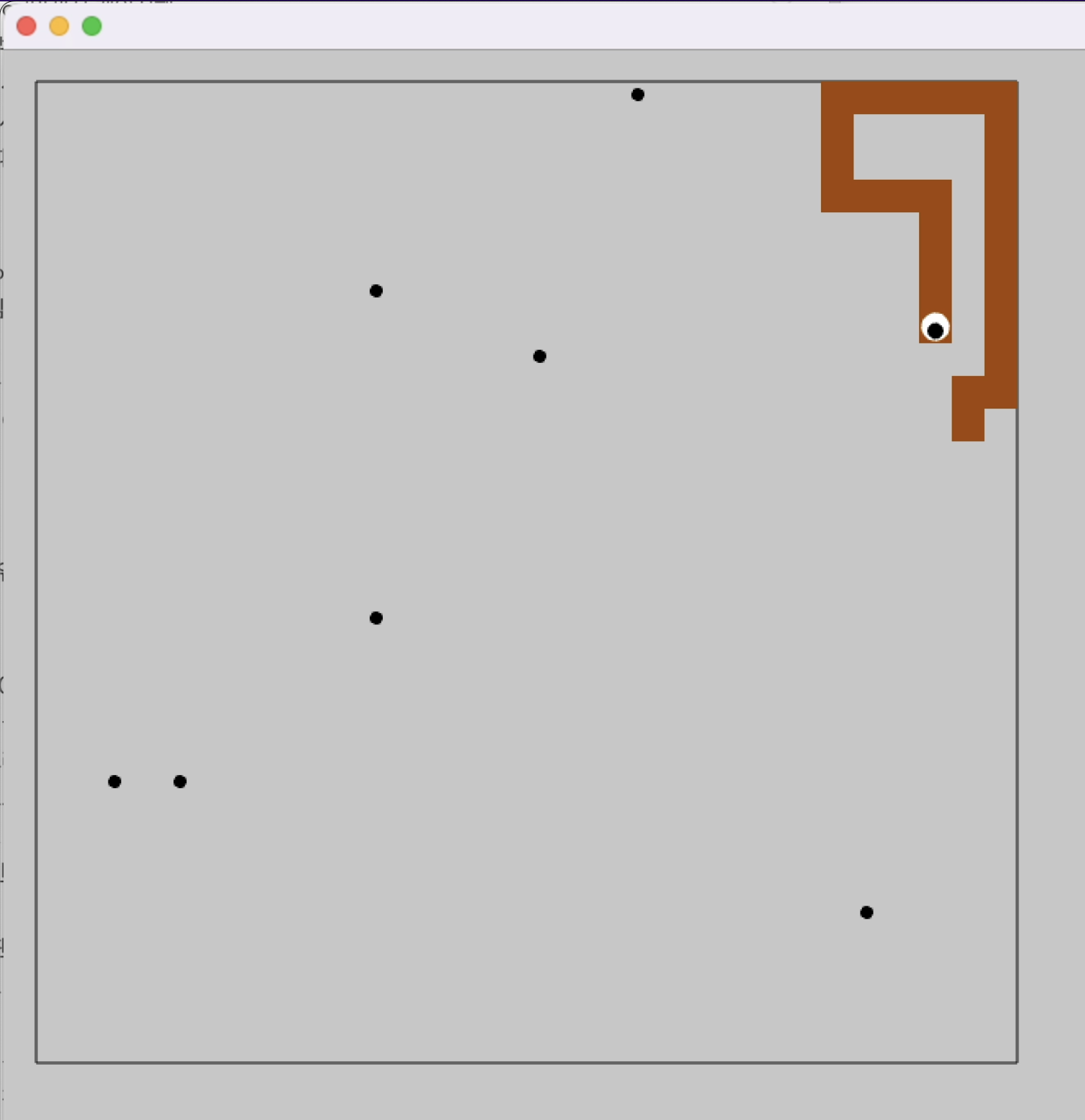


* 1. **지렁이 방향변경 ( 방향키 입력 )  
     텍스트, 명함이(가) 표시된 사진

     자동 생성된 설명**
  2. **키보드 입력 ( 방향키 제외 )**



* 1. **시간/공간복잡도**
     1. **시간복잡도**openframeworks의 특성상 update()가 꾸준히 이뤄지고, update()시 draw()가 함께 진행되기 때문에, 시간복잡도를 update()를 기준으로 설명하고, update()의 시간복잡도에 draw()의 시간복잡도를 포함하겠다.  
        이 프로그램의 핵심적인 구성요소 두개는 게임보드와 지렁이이다.  
        게임보드는 BOARD\_SIZE ^ 2의 사이즈를 갖는다.  
        지렁이의 사이즈를 계산하는 변수는 없지만 편의상 length라고 부르겠다.  
        이 프로그램은 board\_init(), draw\_empty\_board(), draw\_net()에 각각 O(BOARD\_SIZE ^ 2)의 시간을 소모하고,  
        draw\_earthworm(), free\_memory()에 각각 O(length)만큼의 시간을 소모한다.   
          
        그러나 지렁이는 게임보드의 방 내에 존재하는 생물이기 때문에 지렁이의 길이는 전체 방의 갯수보다 많을 수 없다.   
        따라서 이 프로그램의 update()시의 시간복잡도는 O(BOARD\_SIZE ^ 2)이다.
     2. **공간복잡도**이 프로그램의 공간복잡도는 O(BOARD\_SIZE ^ 2)이다.   
        이는 게임보드의 크기이며, 지렁이는 게임보드 내에 존재하기 때문에 이 프로그램 전체의 공간복잡도는 O(BOARD\_SIZE ^ 2)가 된다.

1. **창의적 구현**지렁이의 꼬리의 생성방향을 지렁이의 꼬리쪽 몸통의 위치를 기반으로 자동으로 붙도록 설정하였으며, 지렁이의 눈동자의 위치를 지렁이의 머리쪽 몸통의 위치를 기반으로 눈동자가 돌아가도록 구현했다. 또한, 방을 구분하기 쉽도록 net\_mode를 추가했다.
2. **실행결과 캡쳐  
   **
3. **느낀 점 및 개선사항**이 수업을 듣기 전에 자바를 공부한 적이 있었다. 당시, 자바의 그래픽 인터페이스인 awt를 배웠는데, 자바의 awt 역시 openframework와 비슷하게 x와 y좌표를 적어주며 선과 원을 그려준다.   
   당시 이거를 하나하나 어떻게 좌표를 써주고 그려주냐라고 불평하던 것이 기억난다. 아마 당시는 awt로 본격적인 프로그램을 만드는 것이 아닌 그냥 awt로 네모를 그리는 수준이였기에 그랬었을 것이다.   
   이번에 openFrameworks에서의 프로그램은 본격적으로 방들을 구성하고, 각 방별로 for문을 통해 x값과 y값을 저장해두니 사각형과 원을 식을 하나만 구해두면 편하게 그려지는 것이 매우 인상적이였다. 지렁이도 doubly linked list를 이용해 머리부터 꼬리까지의 방의 위치와 포지션을 쉽게 구할 수 있으니 방향키만 눌러주면 지렁이가 내 방향키의 조작에 맞게 움직이게 하는 것이 가능했다.  
   역시 컴퓨터는 자동화를 위해 존재하는 장치구나라는 것을 새삼스럽게 느꼈던 개발이였던 것 같다.   
     
   또한, 지렁이를 움직이려할 때 본래는 지렁이의 모든 몸뚱이를 한칸 앞의 몸뚱이의 위치로 변경시켜주려고 했었다. 그러나 문득 꼬리만 머리앞으로 옮겨주면 되지 않을까라는 생각이 들었고, 그에 가장 적합한 자료구조가 doubly linked list라는 것이 문뜩 떠올라 해당 자료구조를 적용함으로써 지렁이가 움직일때의 연산을 O(지렁이의 길이)에서 O(1)로 극적으로 줄일 수 있었다.  
   그동안 다른 수업들의 과제들은 대개 목적과 방향성이 매우 구체적이여서, 어떤 자료구조를 사용해야할지까지도 명확했기에 이렇게 직접 내가 목표한 바에 가장 효율적인 자료구조를 생각할 일이 없었다.   
   그러나 이번 과제에서는 내가 처음부터 설계해나가는 과제였기에 나의 목표를 어떻게 해야 가장 효율적으로 달성할 수 있을지에 대한 많은 고민이 있었기에 더 배운 것이 많은 프로젝트였다고 생각된다.  
     
   아쉬운 점으로는 지렁이가 움직이거나 먹이를 먹으면 소리가 나고 먹이의 그래픽을 지렁이가 먹는 먹이의 사진으로 넣을 수 있으면 좋았을 것이라는 생각이 들었다. 이에 대해서는 시험이 끝난 후, 방학에 관련 방법에 대해 조사해서 추가해볼 예정이다.