최종 프로젝트 보고서

전공: 수학과,컴퓨터공학과 학년: 3학년 학번: 20181256 이름: 김도현

1.

최종 프로젝트에서는 지금까지 실험에서 코드를 작성한 과정들을 종합하고 또 추가적으로 자신만의 창의적인 생각을 넣어 새로운 게임을 만드는 것을 목표로 하였다. 우선 winodw의 visual studio 2017 버전을 실험 환경으로 잡고 openframework를 사용하였다.

maze 혹은 waterfall 둘 중 하나를 기반으로 잡고자 하였는데 게임을 만들려는 특성상 waterfall 에서 응용될 것이 훨씬 많아 보였다. 그래서 이렇게 기틀을 잡고 코드를 작성하였다.

해당 프로젝트의 목표는 어떻게 보면 5번 질문과 연관이 있다. 서로 관련 지어 이 보고서를 평가하면 좋다.

프로젝트의 목표는 공이 위에서부터 떨어지는데 이 공은 랜덤한 line을 따라서 이동할 수 도 있다. 이때 line 은 겹칠 수 있는데 이 경우 공이 line을 타고 갈지 말지 랜덤하게 결정하여 난이도를 조절하였다. 채점 시 주의해야 할 점은 line을 타고 가지 않아도 잘못 작성한 코드가 아니라는 것이다.

꼭 readme.txt 파일을 읽기 바란다.

그래서 마지막 바닥에 떨어지는 순간에 바구니로 공을 담아 score를 높이는 게임이다. 이때 게임의 모드는 2가지가 있다. Easy mode는 공이 1개만 떨어져 난이도가 쉬운 대신 공 1개당 score는 10점으로 하였고 게임이 가능한 목숨 또한 2개로 설정하였다. Hard mode에서는 공이 2개 떨어져 난이도가 어려운 대신 공 1개당 score는 20점으로 하고 게임이 가능한 목숨 또한 3개로 설정하였다. 이때 주의사항은 특정 모드에 따라 공의 개수를 국한하지 않고 ofApp.h 파일에서 #define 로 공의 개수를 정해두고 만약 공의 개수를 조절하고 싶다면 이 #define만 다시 설정해주면 공의 개수가 다르게 해서 돌아간다. 즉 테트리스 프로젝트의 visible block 을 응용한 것이다.

또한 앞에서 설명한 것을 구현하는 것 역시 자료구조 및 알고리즘을 사용하는데 정확한 이름이 부여된 새로운 알고리즘을 사용하기 위해 점수 ranking 시스템을 도입하였다. 여기서는 eaymode, hardmode 각각의 ranking이 게임이 끝난 후 알 수 있다. 이때 게임이 끝난 점수 역시 새로 기입을 해주어야 한다. 기입하는 파일 이름은 각 easy.txt, hard.txt 로 한다. 만약 이 두 파일이 없다면 첫 프로젝트 실행 시 자동으로 만들어 초기화를 시켜준다. 기존의 내림차순에서 새로운 값을 넣기 위해서는 탐색 및 수정에 유리한 linkedlist를 사용하였다. 이때 화면 상 크기가 모든 rank를 보여줄 수는 없으니 최대 5개까지만 보여주도록 한다. 만약 5개가 없다면 그 개수만큼만 보여준다.

반면에 가장 첫 메뉴 화면에서는 이 두 ranking을 종합하여 점수의 total rank를 보여주어야 한다. 그러려면 두 개의 Ranking을 입력 받아 다시 내림차순으로 sorting 을 해야 한다. 이 때 새로운 알고리즘을 도입하였다. 가장 효율적인 sorting 방법에 대해 구상해 보았다. 바로 시간 복잡도가 최악의 상황에서도 O(nlogn)을 넘지 않고 공간 복잡도 또한 O(n) 인 heap sorting 알고리즘을 도입하였다. 비록 같은 O(nlogn) 이라도 그렇게 빠른 성능을 안보일 수도 있지만 결국 평균적인 속도 면에서 안정적으로 나오기 때문에 이를 선택하였다. 일반적으로 많이 정렬하는 데는 bubble sort, selection sort 등이 있는데 이는 for문을 2번 돌며 거의 모든 경우를 다 탐색하여 정렬하기 때문에 시간 복잡도가 O(n^2)이 걸린다. 이때 자료구조는 배열을 사용하였는데 그 이유도 추후에 설명할 예정이다.

이때 화면 상 크기가 모든 rank를 보여줄 수는 없으니 최대 5개까지만 보여주도록 한다. 만약 5개가 없다면 그 개수만큼만 보여준다.

해당 프로젝트로 linkedlist와 배열의 Ranking을 입력 받을 때나 sorting할 때의 차이 또한 알 수 있다.

간단한 키 설명으로 게임 메뉴에서 hard mode, easy mode는 마우스 클릭으로 실행이 되고 게임 화면에서는 s 키를 누르면 게임이 시작된다. 이때 바구니는 방향키의 오른쪽 왼쪽 키로 조절 가능하다. 이후 게임이 종료되면 다시 RE\_GAME으로 메뉴화면으로 돌아가거나 QUIT 을 눌러 정상적으로 메모리 해제 등을 하고 프로그램을 종료한다. 만약 게임 도중 q를 누르면 정상적으로 메모리 해제 등을 하고 프로그램을 종료한다.

2. 핵심적인 변수들을 설명하기 위해 ofApp.h 파일을 가져왔다.

우선 함수에 대한 자세한 설명은 3장에서 다루니 여기는 핵심적인 글로벌 변수 설명만 하겠다.

사진의 주석도 함께 보면 이해가 더 쉬울 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

draw\_flag는 공과 line에 대한 정보가 다 받아들여져 정상적으로 자료구조에 저장이 되어 해당 정보를 윈도우 창에 그림으로 그릴 수 있는지 없는지를 확인하는 flag 이다. Start\_flag는 draw\_flag로 게임 화면이 그려졌다면 s 키를 눌러 게임이 시작했는지 하지 않았는지 확인하는 flag 이다. Vertical\_flag 는 공이 수직으로 떨어지는지 혹은 line 을 타고 이동하는지 확인하는 flag 이다. Random 변수는 이미 srand 함수를 통해 실제 rand() 함수로 계속해서 난수를 받아오고 있지만 여기에 조금 더 random 한 경우를 추가하려고 도입한 정수형 변수이다. 다음 life는 해당 목숨이 얼마나 남았는지 이를 체크하는 변수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음은 line\_ball 이라는 struct 구조체로 line의 시작 좌표 끝 좌표를 저장하는 구조체이다.

Dot\_ball 이라는 struct 구조체는 공의 x,y 좌표를 저장하는 구조체이다. 이때 공 역시 구조체로 주어 야하는 이유는 공이 2개 이상으로 게임화면에 나타날 수 있고 이 역시 좌표를 한번에 저장하는 것이 코드 작성상 편리하기 때문이다. 그리고 바구니를 표현할 구조체인 Box 역시 Box의 왼쪽 상단의 좌표를 저장하고 있다. 다음은 scorenode 구조체인데 hardmode 혹은 easymode의 score rank 를 불러올 때 사용한다. 여기서 새로 게임이 진행될 때마다 score를 추가적으로 hard.txt, easy.txt에 기입해야 하는데 이 경우 linkedlist로 구현하는 것이 배열로 구현하는 것보다복잡도 측면이나 코드를 구현하는데 훨씬 유리하다. Linkedlist의 경우 그냥 해당 위치까지 탐색하여 link만 다시 연결해주면 완료가 되어 추가적으로 다시 자료구조의 메모리 할당이 필요 없다. 때문에 해당 각 모드에서의 rank 알고리즘의 linkedlist 자료구조는 적절한 것이다.

여기서 top은 해당 mode에서의 가장 높은 score를 가리키고 있다. Ranknumber 변수는 easy, hard 각각의 경우의 ranking 개수를 저장하고 있다. Level 변수는 hard 모드인지 easy 모드 인지를 저장하고 이에 따라 공의 개수 등을 다르게 표현하기 위해 사용하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Game 메뉴 실행 여부나 종료 여부를 확인할 수 있는 bool 타입의 game\_menu, game\_over가 있다. 초기값으로 game\_menu는 true, game\_over는 false 로 두어 프로젝트 생성 시 기본값으로 메뉴가 보이도록 해주었다. 다음은 폰트를 지정하는 font1, font2가 있다. font1은 verdana 폰트에 포인트가 32 인 폰트이다. font2은 verdana 폰트에 포인트가 20 인 폰트이다. 필수적인 요소는 아니지만 openframework 는 디자인도 큰 요소이기 때문에 이를 추가하였다. 또한 게임의 점수를 저장하고 있는 score 역시 전역변수로 두었다. Num\_of\_line은 line을 rand 함수로 랜덤하게 만들 때 그 개수를 저장하는 정수형 변수이다.

heap 1차원 동적 할당 배열은 eaymode, hardmode 에서 각각 rank를 받아와 이를 heap sorting 으로 정렬 할 때 사용하는 배열이다. Heap을 사용한 ranking 시스템 구현 시 배열을 사용한 이유는 우선 sorting만 진행을 할 것이기 때문에 추가적으로 정보를 집어 넣거나 아니면 삭제하는 일이 전혀 없다. 배열은 여기서 단점이 있는데 만약 추가 정보를 중간에 넣는다면 그 만큼 모든 노드들을 뒤로 보내야 해서 다시 메모리를 재 할당 받아야 하기 때문에 공간 복잡도나 시간 복잡도 측면에서 손해가 있다. 또한 정보를 삭제할 때도 중간에 빈 공간을 다시 채워야하기 때문에 그 뒤에 있는 배열 Index들을 앞으로 한칸 씩 당겨야 하는 복잡도 측면의 손해, 코드의 복잡함이 늘어난다. 하지만 sorting만 진행할 경우 이 경우는 생기지 않고 무엇보다 heap 은 완전 이진 트리를 사용해야 하는데 index가 1부터 시작할 경우 자식 노드를 /2 하면 부모 노드가 나와 구현하기 매우 유리하다. 때문에 알고리즘에 따른 적절한 자료구조를 사용했음을 알 수 있다.

Heapsize는 나중에 heap 배열을 이용하는 등에 사용할 heap size를 저장하고 있는 변수이다. (이 size를 추가한 이유 역시 코드 설명이나 마지막 느낀점에 설명하겠다.)

3. 각 함수에 대해 설명을 할 것인데 이미 주석 처리로 자세한 설명이 되어 있어 핵심적인 부분을 집고 넘어가겠다. 그리고 어차피 함수이기 때문에 설명하는 함수의 순서는 코드에 적힌 함수의 순서와 약간 다를 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Setup 함수로 여기서는 초기값을 세팅한다. 우선 뒷 배경을 흰색(ghostwhite) 으로 설정 하였다. 그리고 draw\_flag를 0으로 두어 초기에는 아무것도 안 했으니 당연히 화면에 라인 및 바구니 그림을 그려서는 안된다. 또한 여기서 font 설정을 다음 사진과 같이 설정하였다. 이는 font1은 verdana 폰트에 포인트가 32 인 폰트이다. font2은 verdana 폰트에 포인트가 20 인 폰트이다. 최대한 openframework의 장점을 이용하고자 디잔인에도 충실했다. 실제로 rand함수가 난수를 지속적으로 받기 위해 srand 함수를 통해 계속해서 다른 값으로 들어오도록 설정하고 현재 easy\_mode, hard\_mode 둘다 false 로 설정된 상태이다.

가장 첫 실행화면에서는 menu 화면에 total\_rank가 출력이 되어야 하니 total\_rank() 함수를 실행 하여 현재 easymode, hardmode의 rank 정보들을 받아와 sorting 할 수 있도록 해준다. 그리고 게임의 점수를 저장하는 score 역시 0을 초기화 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음은 핵심적으로 게임 구동이 프레임에 따라 돌아가도록 하는 알고리즘을 구현한 update 함수이다. 우선 s 키가 눌렸다면 게임을 실행한다. 이후 반복문을 통해 level 수 만큼(공의 개수 만큼) for문이 돌아가도록 설정하였다. 이렇게 하여 기존의 waterfall 에서 물이 1줄기만 떨어지는 것이 아닌 2줄기 이상 떨어지도록 구현하는 것처럼 새로운 알고리즘을 도입한 것이다. 공 역시 이렇게 해서 1개뿐만 아니라 2개 이상을 구현할 수 있다. 이게 가능한 이유는 update함수는 각 프레임 당 실행이 되기 때문에 거의 동시에 공들에 대한 좌표 계산이 되는 것처럼 보인다. 이후 수직 경로를 의미하는 vertical \_flag를 1로 두어 공이 수직을 떨어질 수 있도록 한다. 이후 각 랜덤한 line들 수 만큼 반복문을 돌면서 random 변수를 1씩 늘려 rand 함수 실행 시 더욱 값이 분산되도록 설정한다. 그리고 시간 복잡도를 줄이기 위해 이미 line들이 해당 공들의 좌표보다 더 위에 있다면 바로 다음 for문으로 넘어가도록 하였다. 이후 선분의 방정식에 공의 좌표를 대입하여 그 차가 절대값 3 보다 작으면 해당 Line을 타고 내려가는 것으로 간주하였다. 이때 당연히 공의 y 좌표는 선분의 각 양 끝 점 y 좌표 사이에 있고 공의 x 좌표는 선분의 각 양 끝 점 x 좌표 사이에 있어야 한다. (랜덤한 선분들을 받아와 기울기가 거의 0에 가까운 선분도 존재하기 때문에 이 경우를 추가해야 정상적으로 게임이 작동한다.)

프로젝트의 목표인 여기서 ball\_line이 겹쳐 있을 경우 어느 line을 타고 가는지 random 으로 결정되는 곳이다.

그렇지 않다면 vertical\_flag를 1로 수정하여 계속해서 공이 수직방향으로 떨어지게 한다. 만약 수직방향이라면 y값만 조절해주면 된다. 이때 난이도를 올리기 위해서 어떤 공은 속도를 y 값을 3만큼 증가시키고 어떤 공은 속도를 2만큼 증가시켜 속도를 다르게 하였다.

수직방향이 아니라면 line을 타고 공이 흘러야 한다. 이때는 line의 기울기가 0보다 크면 공을 오른쪽으로 이동시키고 0보다 작으면 공을 왼쪽으로 이동시키면 된다. 이후 위에서 바뀐 x좌표에 따라 해당 선분의 방정식에 대입하여 y 좌표도 구할 수 있을 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

여기서는 update 함수에서 꼭 해줘야 할 일인 공을 바구니에 담는 과정 및 게임 종료 여부를 판단하는 새로운 알고리즘이 추가된 핵심적인 부분이다.

만약 바구니의 좌표 기준으로 공이 일정 수준까지 가까이 왔다면 공을 받는 것으로 간주한다. 이 경우가 생기면 아까 random 변수를 이용하여 50프로 확률로 새로운 공이 x 좌표 기준으로 윈도우 창 오른쪽에서 시작할 수 있도록 해주고 50프로 확률로 새로운 공이 x 좌표 기준으로 윈도우 창 왼쪽에서 시작하도록 해준다.

그리고 공을 바구니에 담을 때마다 새로운 line을 만들어 난이도를 다르게 해주어야 하니 makeline 함수를 실행시켜 새로운 line 정보를 받아온다. 그리고 기본 score 점수를 10점을 추가해주고 random 변수역시 초기화를 한번씩 시켜준다. 이때 level 이 1 이 아니라는 것은 easy mode가 아닌 hard 모드이기 때문에 score를 10를 더 추가해 공 하나당 20점을 부여하도록 한다. 그렇지 않고 공을 못 받았다면 life를 1 줄여 목숨을 하나 줄이고 만약 목숨이 0이 되었다면 get\_rank 함수를 실행시켜 각 모드에 따라 다른 rank.txt(hard.txt, easy.txt) 정보를 읽어 들인다. 이후 해당 score를 write\_rank 함수에 넘겨주어 각 mode에 맞는 txt 파일에 새로운 점수를 정확한 위치에 기입 해준다. 이후 게임이 종료되었다는 game\_over 를 true로 해주고 각종 flag 및 변수들을 초기화 해준다. 만약 아직 목숨이 남아 있다면 게임을 계속 진행해야 하므로 위에서 진행한 것처럼 다시 random 변수를 이용하여 50프로 확률로 새로운 공이 x 좌표 기준으로 윈도우 창 오른쪽에서 시작할 수 있도록 해주고 50프로 확률로 새로운 공이 x 좌표 기준으로 윈도우 창 왼쪽에서 시작하도록 해준다. 또한 다시 해당 공이 위에서 시작하므로 makeline 함수를 실행시켜 새로운 line 정보를 받아온다. 차이점은 여기서는 점수를 추가적으로 부여하지는 않는다.

텍스트이(가) 표시된 사진

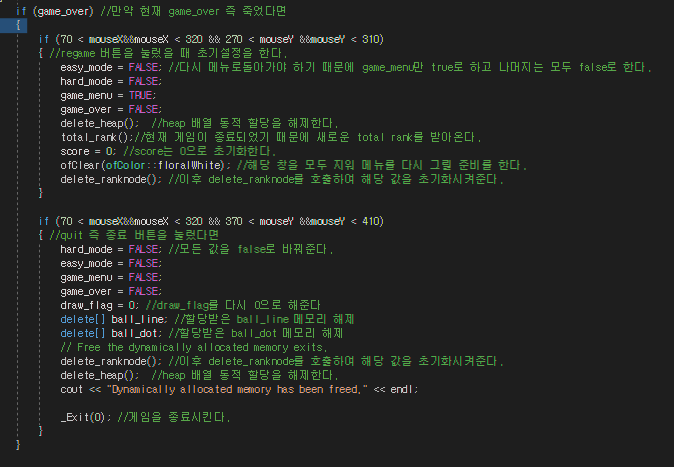
자동 생성된 설명

다음은 keypressed 함수로 만약 q 를 눌렀다면 프로그램을 종료시켜야 하기 때문에 각종 flag를 0으로 해주고 할당받은 ball\_line, ball\_dot 메모리 해제를 해준다. 그리고 delete\_heap 함수를 통해 heap 1차원 배열 동적 할당 역시 해제 시켜주어 메모리 leaking이 일어나지 않도록 한다. 만약 game\_over 인 상태라면 해당 mode에서의 ranking이 보여지고 있다는 뜻으로 delete\_ranknode 함수를 통해 linkedlist 메모리를 정상적으로 해제 시켜준다. 만약 s 키를 눌렀다면 draw\_flag가 1 일 경우에 star\_flag를 1로 설정하여 게임을 구동하도록 한다. 만약 draw\_flag가 1인 경우( 게임을 실행하지 않아도 미리 바구니의 위치를 움직일 수 있다.) 오른쪽 방향키를 누를 때 바구니의 x 좌표가 1024보다 작으면 한번에 30만큼 바구니를 오른쪽으로 이동시키고 왼쪽 방향키를 누를 때 바구니의 x 좌표가 0보다 크다면 한번에 30만큼 바구니를 왼쪽으로 이동시킨다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음은 mousePressed 함수로 마우스를 화면에 클릭했을 때 일어나는 일들이다. 만약 게임 메뉴가 true 라면 게임 화면에는 easy mode, hard mode 버튼이 있다. 해당 위치에 알맞게 마우스로 클릭을 하면 각종 flag들을 새롭게 설정하여 원하는 모드의 게임이 실행된다. 이때 life 목숨 설정을 다르게 해주고 ofclear를 통해 한번 화면을 지워 메뉴 화면이 아닌 게임 화면을 띄워야 한다. 그리고 get\_line\_ball\_box() 함수를 실행시켜 line이나 ball, box 에 대한 정보들을 받아도록 한다.



Mousepressed 함수에서 gameover 가 true 라면 게임이 종료되었다는 의미로 만약 regame 버튼을 누르면 다시 초기설정을 해야해서 game\_menu를 true 로 하고 새로운 score 가 추가 되었기에 먼저 heap 1차원 동적 배열 메모리를 해제 하고 이 정보를 반영한 total\_rank 함수를 실행해 다시 heap sorting을 해준다. 그리고 score는 0으로 초기화하고 gameover에 관련한 화면을 지우고 현재 화면에 출력 중인 해당 모드의 자체 rank 를 받아온 ranknode linkedlist 역시 delete\_ranknode 함수를 통해 메모리 해제 시킨다.

만약 quit 버튼을 눌렀다면 모든 값을 false로 설정하고 할당 받은 ball\_line 이나 ball\_dot 등의 메모리를 해제 시키고 현재 화면에 출력 중인 해당 모드의 자체 rank 를 받아온 ranknode linkedlist 역시 delete\_ranknode 함수를 통해 메모리 해제 시킨다. 그리고 delete\_heap을 통해 heap 배열의 메모리 해제 를 시켜 memory leaking 이 일어나지 않도록 한다. 이후 정상적으로 프로그램을 종료 시킨다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음은 draw 함수로 design을 결정하고 rank 및 정보들을 화면으로 출력하는 함수이다.

우선 기본은 ofNoFill로 도형에 색을 칠하지 않도록 하였다. 이후 상단 하단에 직사각형 도형을 두어 상하의 경계선을 구분하였고 게임 메뉴가 실행중일 때는 draw\_menu 함수, 게임에서 죽었을 때는 draw\_gameover, draw\_flag가 1 일때는 draw\_line\_box 함수, startflag가 1일 때는 게임을 시작해야 하니 draw\_game 함수를 불러온다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Draw\_menu 함수는 다음과 같이 drawstring 및 ofDrawRectangle를 통해 원하는 위치에 easy mode, hard mode 버튼 및 해당 문구들을 출력하고( 자세한 건 코드 사진의 주석을 보면 된다.) 그리고 여기 핵심은 heap sorting을 통해 total rank5를 출력해줘야 하는 것이다. 그래서 heapsize를 돌면서 string형 rank 에 to\_string으로 정수형을 string 으로 변환하여 저장한다. 그리고 해당 순위는 loop 변수의 i 를 추가적으로 rank에 다시 넣어주어 rank에는 최종적으로 순위.rank 가 들어가게 된다. 이를 원하는 위치에 순차적으로 font2 폰트를 사용하여 출력해준다. 그리고 i가 5가 되었으면 이미 Top 5 rank가 출력되었으니 이 for문을 멈춘다. 이 경우 i를 1부터 출력하는데 그 이유는 heap sorting의 경우 heap 배열을 통해 완전 이진 트리를 구현해야 하는데 첫 시작 index를 1로 둔다면 부모노드에 대한 접근은 자식노드/2 를 하면 갈 수 있기 때문이다. (이 점 이 유의사항이다.)

텍스트이(가) 표시된 사진

자동 생성된 설명

Draw\_gameover 함수 에서 역시 RE\_GAME, QUIT 버튼들을 drawstring, ofdrawRectangle 등으로 만들어 준다. 자세한 것은 앞에서 설명했기에 생략하겠다. 이때 여기서는 모드에 따른 자체 ranking 을 출력해야 하는데 만약 easy\_mode라면 title을 Easy\_Mode\_top\_5 로 설정한다. 아니라면 Hard\_Mode\_Top\_5 로 설정한다. 이후 각 모드의 ranking은 linkedlist 형태고 저장되어 있기 때문에 current 노드를 순차적으로 노드를 탐색하면서 current->rankscore를 font2 폰트로 출력할 수 있도록 한다. 이때 i==4 즉 rank가 5까지 출력되었기 때문에 더이상 출력을 진행하지 않는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Draw\_Game 함수에서는 해당 떨어지는 공들을 level 수에 따라 ball\_dot struct 구조체에 저장된 공의 (x,y) 좌표를 화면에 원 모양으로 출력해준다. 색깔은 랜덤하게 주어 공이 빛이 나는 것처럼 해주었고 오른쪽 상단에는 current Score 문구 및 실제 현재의 score를 출력하였다. 그리고 life 인 목숨 또한 해당 개수만큼의 반복문을 돌며 score 아래에 빨간색 원으로 ofFill() 즉 내부 색을 채워 나타냈다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Draw\_line\_box 함수는 line 및 box 바구니를 그리는 함수인데 여기서는 디자인상 ofNoFill을 하여 내부를 채우지 않았다. Loop 변수인 i가 num\_of\_line이 될때까지 반복문을 돌며 ofdrawline으로 해당 선분들을 그려준다. 또한 하단에는 box.X,box.Y 좌표를 가지는 바구니 또한 가로 70, 세로 40 으로 그려준다. 이후 게임 방법 설명을 위해 왼쪽 상단에 시작 안내 및 종료 안내 문구도 font2 폰트로 기입하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음은 delete\_ranknode() 함수 및 detet\_heap() 함수이다. 여기서는 각 rank 정보들을 저장하고 있는 자료구조의 메모리 해제를 하는데 delete\_ranknode() 에서는 각 mode의 rank 를 담는 linkedlist 메모리를 해제해야 해서 topnode로 계속해서 linkedlist를 순차적으로 돌면서 null 이 될때까지 current=top 한 current를 삭제 시킨다.

Delete\_heap() 함수에서는 easy, hard 모드의 rank를 합친 total rank에 대한 정보를 담고 있는 자료구조 동적 할당 1차원 heap 배열의 메로리를 delete 로 삭제한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Get\_line\_ball\_box 함수는 이름그대로 line, ball, box를 받아오는 함수이다. 우선 makeline 함수를 통해 line을 받아오고 easy\_mode가 실행되었다면 level을 1로 하여 ball\_dot 에 공의 정보를 저장하고 있는 dot\_ball 구조체에 level 만큼의 동적 할당으로 받는다. Hard\_mode가 실행되었다면 level 을 2로 하여 dot\_ball 구조체를 동적 할당 받는다. 그리고 바구니의 x,y 좌표는 윈도우창 기준 가장 왼쪽으로 초기화 한다. 이후 게임에 필요한 정보들이 모두 받아졌으니 draw\_flag=1를 통해 해당 정보를 화면에 출력하도록 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Makeline 함수는 line에 대한 개수 및 좌표들을 업데이트 해주는 함수이다. 즉 이미 start\_flag가 1이라면 게임이 시작했다는 의미이므로 ball\_line에 대한 메모리를 해제 해주고 다시 기입 해준다. (메모리 오류 안생기도록 함)

그리고 num\_of\_line은 rand 함수로 하여 3~5개의 num\_of\_line 이 생기도록 설정하였다. 이후 ball\_line에 해당 num\_of\_line 만큼의 line\_ball인 Line 정보를 담고있는 struct 구조체로 메모리 할당을 해준다. 이후 반복문을 돌면서 각 선분의 시작 점 끝 점을 rand 함수를 통해 난수를 받아와 이를 각 Ball\_line[i]의 시작(x,y)좌표, 끝 (x,y) 좌표 로 저장한다. 그리고 slope 즉 기울기 역시 update 함수에서 공이 Line을 따라 움직여야 하기 때문에 계산하여 저장한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Get\_rank 함수는 mode에 따라 easy.txt 파일 혹은 hard.txt 파일을 읽어들여 해당 정보를 scorenode에 저장하는 함수이다. 우선 mode를 확인하여 file string 변수에 해당하는 파일 이름을 준다. 그리고 Ifstream readfile(file)을 통해 파일을 읽는다. 만약 readfile.is\_open이 true 이면 getline 함수를 통해 순차적으로 한 줄 씩 num\_line에 정보를 받는다. 이때 가장 상위 정보는 해당 rank의 개수를 의미한다. 그래서 이 숫자를 ranknumber에 int형으로 변환하여 넣어주고 scorenode linkedlist에 넣기 위해 top을 null로 하고 반복문을 돌면서 getline(readfile, num\_line)이 0이 될 때까지 다음을 시행한다. 우선 해당 정보를 newnode라는 새로운 노드를 만들어서 거기에 정수형으로 변환해 rankscore에 저장하고 link를 null로 한다. 만약 top이 null이라면 해당 newnode를 top으로 하고 prev node를 통해 다음 노드를 계속 linking 시킨다. 이후 top이 null이 아니라면 prev->link를 newnode로 할당하고 prev를 newnode로 하여 다음 node를 연결시키는 준비를 한다. 그리고 해당과정이 끝나면 readfile.close를 통해 파일을 닫는다.

만약 파일이 존재하지 않는다면 ofstream writefile로 해당 이름의 file 을 새로 생성해서 writefile.write ranknumber 및 숫자를 기입해준다. 여기서 초기값은 ranknumber 1, score의 최고 점수는 0 점으로 기입해준다. Writefile.write(“0”,1) 뒤의 1은 입력 문자 수라고 생각하면 된다.

이후 해당 정보를 newnode를 만들어 정보를 집어 넣고 top을 newnode로 하여 top 역시 해당 정보로 위치시킨다. 이때 ranknumber는 0이 하나 있기 때문에 1이된다. 이후 writefile을 닫는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Write\_Rank 함수는 게임에서 종료한 이후 새로운 score를 linkedlist에 해당 점수의 위치에 저장하여 이를 mode에 따라 각 easy.txt 나 hard.txt 파일에 다시 기입해주는 함수이다. 우선 이 과정까지 왔다는 것은 무조건 file이 있다는 것으로 예외없이 ofstream.writefile로 입력할 파일을 열어 입력준비를 한다. 그리고 우선 해당 score를 절적한 linkedlis의 위치에 넣어야 하기 때문에 currentnode를 top으로 하여 우선적으로 top의 rankscore와 비교연산 후 top node를 바꿀지 안바꿀지 결정한다. Top 보다 작다면.이제 top은 바뀌지않는다는 의미로 반복문을 통해 해당 원하는 score의 위치를 찾을 수 있다. 이때 i==ranknumber-1 즉 가장 마지막 노드까지 왔는데 원하는 위치가 없다면 결국 그 score는 newnode로 만들어 가장 아래 node와 연결 한다. 이때 ranknumber는 +1 하여 노드가 하나 추가됐음을 알린다.

이후 tmp string형에 ranknumber를 to\_string화 하여 우선적으로 .txt 파일의 가장 위에 ranknumber 를 입력한다. 이후 for 반복문으로 순서대로 currentnode=top 부터 마지막 노드 까지 순차적으로 link를 통해 움직이며 .txt 파일에 입력을 한다.

최종적으로 writefile.close로 파일을 닫는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Total rank 함수는 새로운 알고리즘을 적용한 함수로 heap sorting을 통해 easy mode, hard mode의 rank 정보를 받아와 이를 정렬하는 함수이다. 그래서 우선 get\_rank함수를 통해 easy\_mode에 대한 정보를 읽어들인다. 이때 easy\_tmp라는 동적할당 배열을 통해 get\_rank로 부터 받아들인 scorenode 정보를 for문을 돌면서 저장한다. 이때 easymode에 대한 get\_rank 함수가 실행되니 scorenode에는 easy\_mode rank 정보만 담겨져 있다. 그래서 currentnode를 top으로 하여 순차적으로 link를 타고 이를 easy\_tmp 동적할당 배열에 주면 된다. 이때 지역변수로써 heapsorting에 사용할 변수인 totalnumber를 ranknumber로 저장해야한다. 이후 delete\_ranknode를 통해 easy\_mode에 대한 scorenode 정보를 모두 지우고 get\_rank 함수를 통해 hard\_mode 정보를 받아온다. 이경우도 마찬가지로 scorenode에는 hard\_mode rank 정보만 담겨져 있다. 그래서 currentnode를 다시 top으로 하여 hard\_mode에 대한 정보를 hard\_tmp 1차원 동적할당 배열에 저장한다. 또한 totalnumber에 hard\_mode에 대한 ranknumber역시 추가해서 더해줘야 전체 totalnumber가 완성 된다.

이후 heap 1차원 동적할당 메모리를 [totalnumber+1] 만큼 준다. 그 이유는 heap[1] 부터 heap[totalnumber]로 설정하면 heap 완전 이진 트리를 구현하는데 매우 유리하다. 그 이유는 자식노드에서 부모노드를 접근할 때는 그냥 자식노드/2 만 하면 부모 노드가 되기 때문이다. 그리고 heapsize역시 totalnumber+1로 하여 heapsize를 따로 기억하고 있는다. 그 이유는 totalnumber는 heap sorting 알고리즘 구현 시 계속해서 줄어들기 때문에 heap의 size를 기억하는데 좋지 않다. 그 이후 delete\_ranknode() 함수를 실행시켜 읽어들인 hard\_mode의 scorenode를 모두 메모리 해제 시켜 게임 진행에 영향이 없도록 한다. 이후 heap 배열에 easy\_tmp, hard\_tmp 배열에 저장된 정보들을 순서에 큰 상관 없이 집어 넣는다.(어차피 sorting과정을 거치기 때문이다.) 물론 순차적으로 넣는 코드를 짜긴 했다. 이후 minheap 함수를 실행하여 minheap 완전이진트리를 처음에 구성하도록 한다. 이후 heapsize-1 만큼 for문을 돌면서 sorting\_heap 함수를 실행시킨다. 이때 actual paramter를 &totalnumber로 주는데 그 이유는 sorting을 할수록 가장 아래 노드들은 내림차순 sorting이 완료되어 더이상 접근을 하면 안되기 때문이다. 이런 이유들로 시간 및 공간복잡도 측면에서 heap sorting이 유리하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다음은 minheap 함수이다. 여기서는 totalnumber를 받아와 반복문을 i부터 totalnumber 까지 모든 노드들을 확인한다. 이때 childnode를 i, parentnode=i/2로 두어 완전 이진트리에서의 부모와 자식관계를 만든다. 만약 childnode가 1이 아닐때 까지 즉 root node가 아닐 때까지 부모와 자식 노드를 비교하여 만약 부모 노드가 더 큰 숫자를 가진다면 min heap 개념에 어긋나기 때문에 이를 서로 바꿔준다. 그리고 다시 childnode를 parentnode, parentnode를 childnode/2로 하여 다음 while문에서 계속 해당 line에서 탐색을 하도록 한다.

이 과정이 끝나면 minheap 이 완성된다. 가장 높은 root 노드는 가장 작은 값을 가지고 자식 노드로 갈 수록 큰 값을 가지는 완전 이진 트리 구조이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

마지막으로 살펴볼 함수는 sorting\_heap 함수이다. 이 함수는 totalnumber 를 포인터로 받아오는데 totalnumber를 통해 이미 sorting 된 노드들에게는 접근 하지 못하도록 하기 위해서이다. 우선 위의 부모노드부터 아래로 탐색할 건데 부모 노드의 자식 노드 중 어떤 노드가 더 작은지 확인하고 더 작은 노드의 인덱스를 check 변수에 둔다. (즉 만약 부모노드와 비교해 자식 노드가 더 작다면 이 인덱스에 해당하는 heap[check]와 바꿔야 하기 때문이다.)

이때 주의해야 할 점은 오른쪽 자식 노드가 더 작을 때이다. 이때는 childnode가 \*totalnumbe 보다 작은지 (정확히는 \*childnode+1<\*totalnumber+1 을 확인해야 하는데 양 쪽에 1을 빼도 같은 값이다.) 확인하여 접근 가능할 때만 check=childnode+1을 해준다. 아니라면 이 경우도 우선적으로 오른쪽 자식 노드는 접근을 못하기 때문에 왼쪽 자식노드의 인덱스로 check 해준다. (매우 중요한 포인트이다!!)

이후 반복문을 통해 check가 (\*totalnumber)+1 보다 작고 heap[parentnode]가 heap[check]보다 클 때 두 개의 노드를 바꿔주고 계속해서 그 과정을 진행할 수 있도록 처음에 했던 parentnode=check, childnode=parentnode\*2, check 변수에 대한 값 할당을 해준다. 즉 sorting을 진행함에 있어 계속해서 minheap 의 개념이 유지되는 것이다.

그 이후 가장 root 노드와 heap[\*totalnumber] 를 서로 바꿔 heap[\*totalnumber] 에는 현재 접근 가능한 한 heap tree 중 가장 작은 값이 들어 오게 된다.

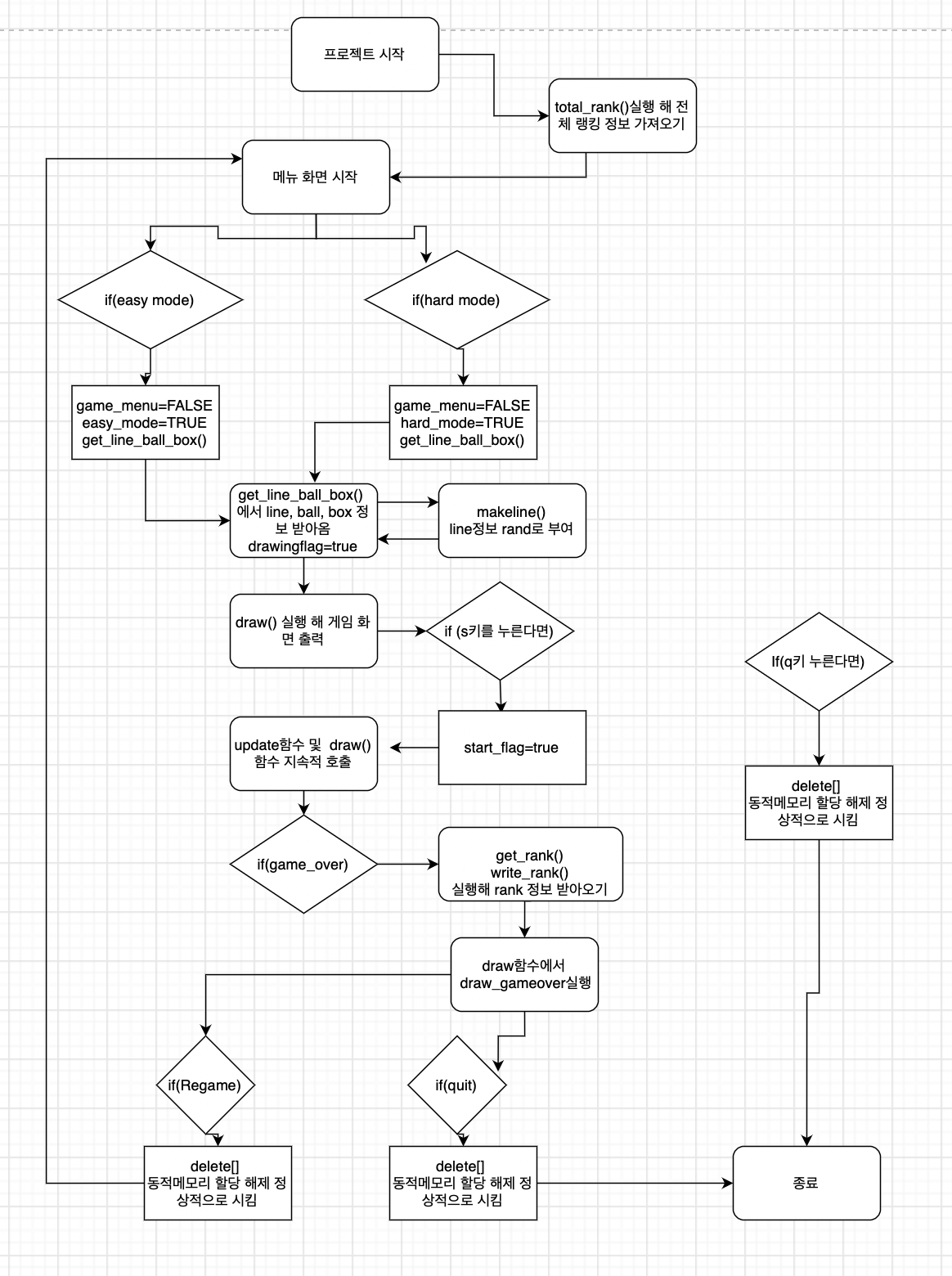
다시 (\*totalnumber)-- 를 하여 해당 저장한 인덱스에 더 이상 부모노드와 자식노드간의 비교에서 접근 못하도록 한다. Sorting이 진행될 수록 \*totalnumber 수는 줄어들기 때문에 접근 가능한 노드 수가 줄어 더 빠른 속도를 보인다.

이 과정을 통해 결국 totalrank를 내림차순 할 수 있게 되었다.

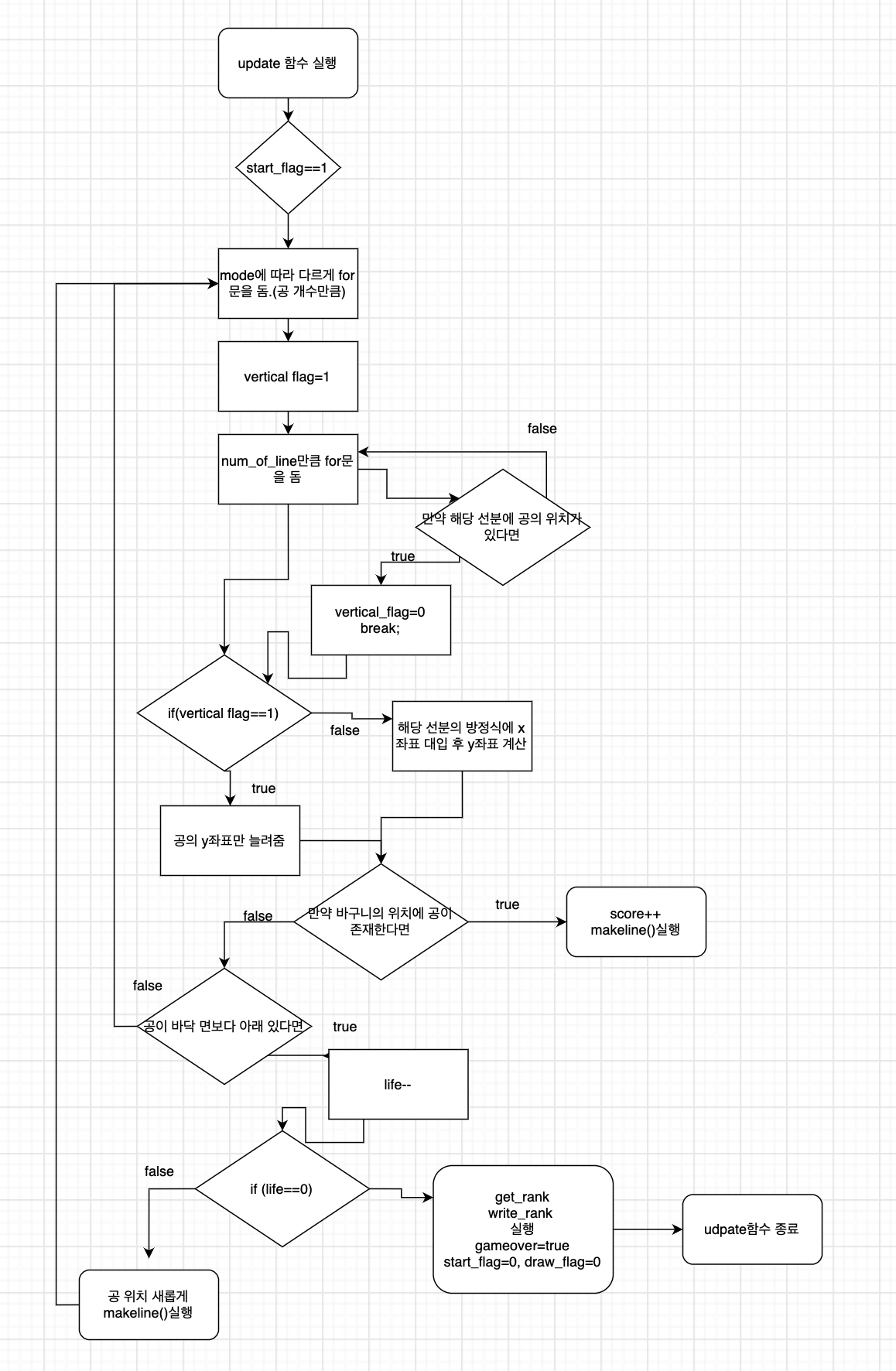
4.

핵심적인 이미 코드에 대해서는 설명 했으니 핵심적인 알고리즘의 핵심적인 flow 차트에 대해서 살펴보겠다.

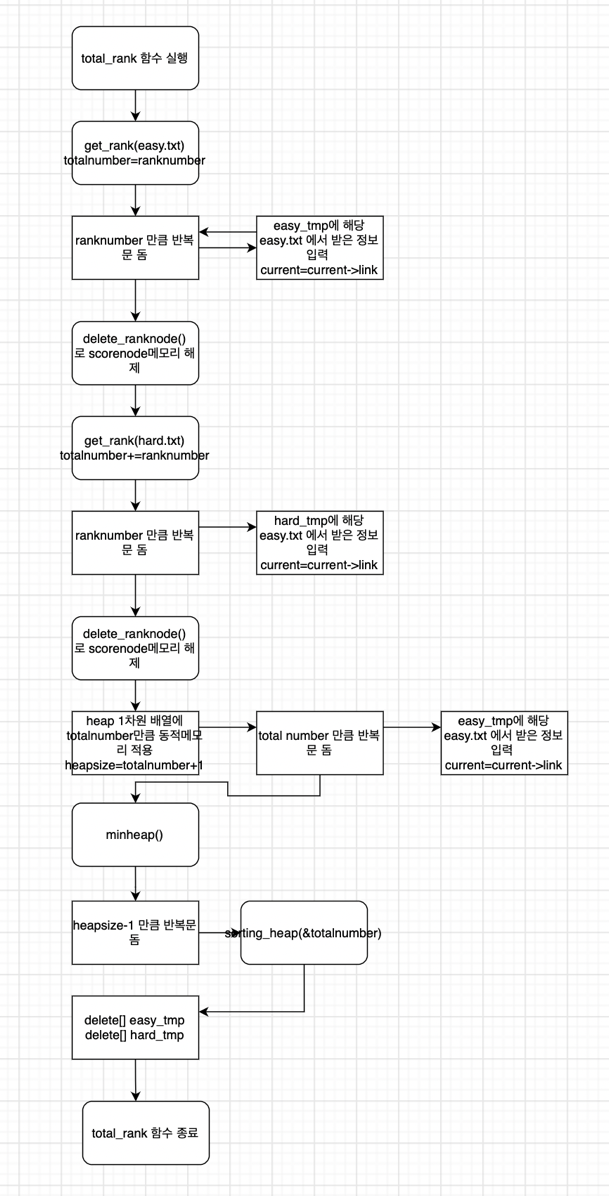
다음은 프로젝트 전체의 플로우 차트이다. 조건문에서는 모두 다음 flow이동은 true 일 때이다.

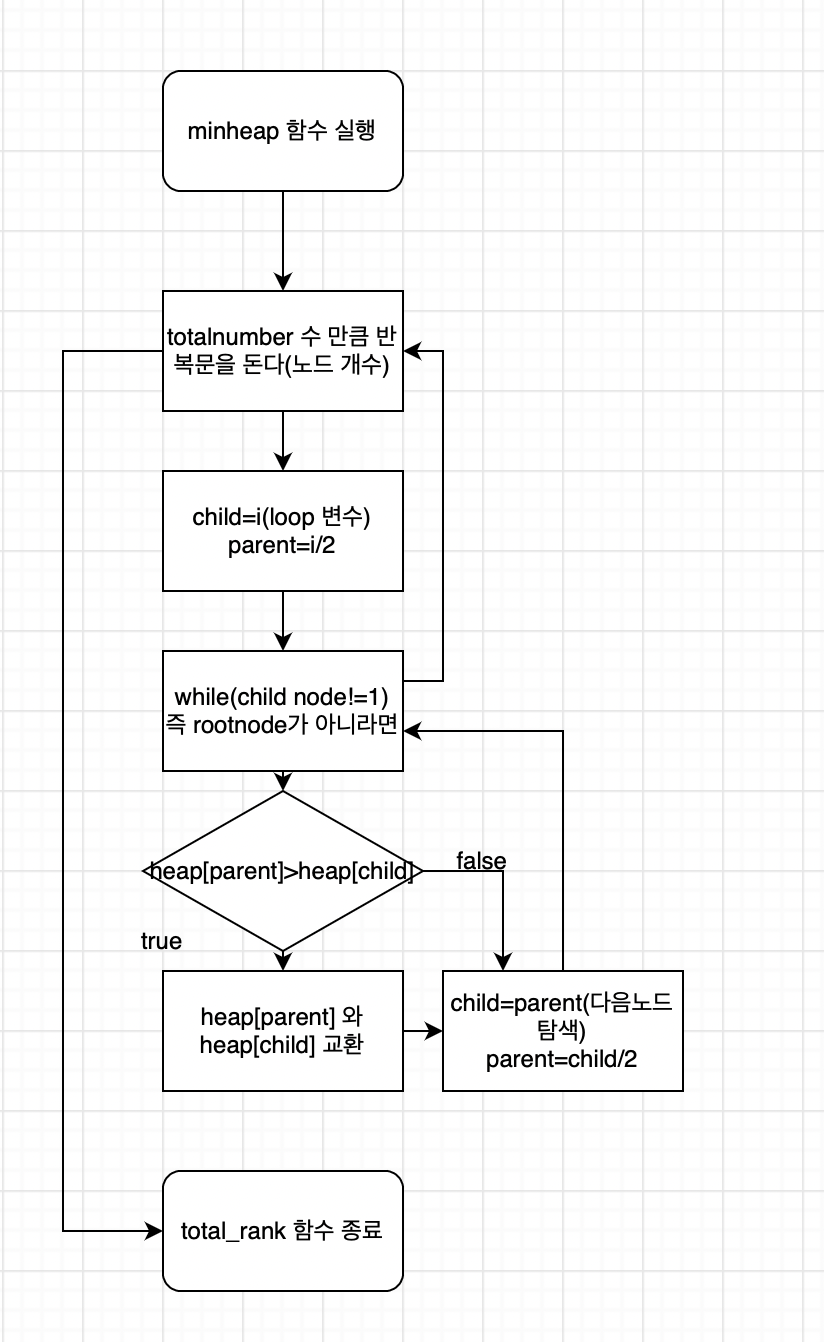


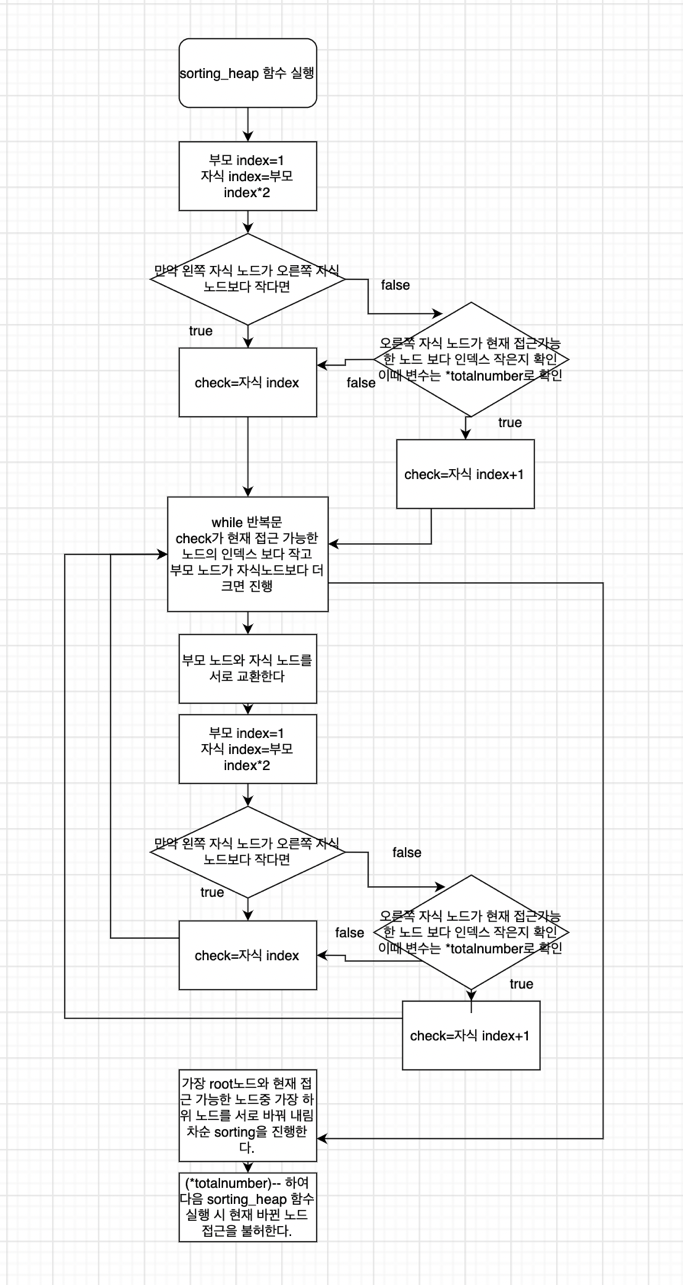
다음은 update 함수의 플로우 차트이다.



다음은 total\_rank() 함수 및 새로운 알고리즘인 minheap tree 구성 heap sorting의 flow chart 이다.







함수에 대한 자세한 알고리즘, 자료구조 설명은 위에 함수 설명하면서 같이 기입했으니 시간 복잡도, 공간 복잡도에 대해서만 기술하겠다.

우선 update 함수에 대한 시간, 공간 복잡도에 대해 알아보겠다. 현재 update 함수는 mode에 따라 공의 개수가 달라 각각의 공에 대해 게임의 계산을 해야 한다. 그래서 우선 for문을 통해 각 공의 경우를 나누었다. 그 다음 random 숫자인 num\_of\_line 으로 for문을 돌면서 각 선분과 공의 위치를 계산하였다. 이때 여기의 시간 복잡도는 O(n) 이다. 그 외에는 조건문으로 해당 update 함수가 진행된다. 전역변수로 ball\_dot, ball\_line 구조체를 사용하였다. 그리고 makeline이 실행될때는 for문을 num\_of\_line만큼만 돌기 때문에 O(n) 의 시간 복잡도가 나온다. 여기서 메모리 해제를 하고 다시 ball\_line에 값을 입력 받는데 전역 변수 고려 시 O(n)의 공간 복잡도로 고려할 수 있겠다. 이때 각 공간 복잡도는 차원이 1차원으로 저장되어 있기 때문에 O(n) 이다.

그리고 get\_rank, write\_rank 함수 역시 호출되는데 결국 rank의 개수에 따라 ranknumber 만큼 반복문을 도니 O(n)의 시간 복잡도가 나옴을 알 수 있다. 여기서는 linkedlist 를 만들어서 해당 Rank 에 대한 정보를 저장한다. 즉 O(n) 만큼의 공간 복잡도가 나옴을 알 수 있다.

최종적으로 update 함수는 O(n^2) 의 시간 복잡도와 O(n)의 공간 복잡도가 나옴을 확인할 수 있다.

다음은 heap sorting 과정이다. 우선 get\_rank 함수를 통해 easymode, hardmode 의 정보들을 읽어들인다. 이때 get\_rank 함수내 에서 각 txt 파일의 정보를 읽어들이기 위해 while문을 ranknumber 만큼 돌아가니 시간 복잡도는 O(n) 이다. 이후 반복문을 돌며 받아들인 정보를 tmp에 저장하고 후에 다시 heap 동적 할당 배열에 저장한다. 여기서 for문을 돌기 때문에 시간 복잡도는 O(n) 이다. 그리고 minheap() 함수를 호출한다.

여기가 중요한데 minheap 에서는 처음 heap 완전 이진 트리를 만든다. 우선 모든 노드를 순차적으로 부모 노드와 확인하여 부모 노드가 더 크다면 부모와 자식 노드를 서로 바꾼다. 이때 O(logn) 의 시간 복잡도가 발생한다. 그리고 이 과정을 n 번 반복하니 최악의 시간 복잡도는 O(nlogn) 으로 고려할 수 있다. 이후 for문을 돌며 sorting\_heap 함수를 호출 한다. 여기서는 가장 root 노드와 접근 가능한 가장 하위 노드를 서로 바꿔 내림차순 sorting을 진행하는데 처음엔 이미 heapsorting이 되어 있지만 그 다음 순간부터는 접근 가능한 노드들을 다시 heap 완전 이진 트리로 만들어 주어야 한다. 이 과정은 root 노드를 기준으로 더 작은 자식 노드들과 바꾸면 된다. 이때 시간 복잡도는 O(logn) 만큼 필요하다. 결국 sorting\_heap 을 여러번 호출하니 최대 O(nlogn) 의 시간 복잡도를 가진다.

최종적으로 heap sorting 알고리즘 과정은 O(nlogn) 만큼의 시간 복잡도를 가지니 우리가 일반적으로 사용하는 bubble sorting 의 시간 복잡도 O(n^2) 보다 훨씬 더 효과적임을 알 수 있다. 그리고 최악의 경우 역시 O(nlogn) 이기 때문에 sorting만 하는 경우 안정적으로 사용 가능하다.

공간 복잡도는 우선 get\_rank 함수 내에서 linkedlist 에 정보들을 저장하는데 여기서 공간 복잡도 O(n) 이 필요하다. 그리고 tmp 동적 할당 변수를 통해서 임시로 easymode, hardmode 정보들을 저장하고 있는데 이 역시 O(n) 만큼 공간 복잡도가 필요하다. 마지막으로 heap 배열 동적 할당을 하는 O(n) 만큼의 공간 복잡도가 필요하다.

최종적으로 heap sorting 알고리즘의 공간 복잡도는 O(n) 이다.

번외적으로 게임 화면에 따라 그림을 그리는 draw 함수의 경우 이 역시 for문을 통해 line, box, 공 등의 여러 정보를 표현하는데 여기서의 시간 복잡도는 O(n)이다. 만약 전역 변수를 고려한다면 모두 O(n)만큼 가지고 있기 때문에 공간 복잡도는 O(n) 이다.

이제 자료구조를 설명하겠다.

여기서 알고리즘에 따라 쓰인 자료구조는 우선 struct 구조체가 있다. 이는 한번의 선언으로 여러 개의 자료형을 가질 수 있는데 그래서 좌표 축 및 선분을 정의할 때 매우 유용하게 사용 가능하다. 그리고 easymode, hardmode 의 각 Rank 정보를 저장할 때는 linkedlist를 사용하였다. 그 이유는 배열을 사용해도 되지만 여기서는 새로운 score에 대한 정보를 기입을 해주는 과정이 필요하다. 배열은 특정 노드를 호출하거나 탐색 면에서는 유리할지 모르지만 어떤 정보를 넣거나 뺄 때 최악의 경우 for 문으로 탐색 후 다시 거의 대부분의 노드들을 전부 이동해야 하기 때문에 정보 입력 혹은 삭제 시 최악의 경우 O(n^2) 의 시간 복잡도가 생길 수 도 있다. 그리고 메모리 역시 동적으로 할당하였다면 다시 재할당 해줘야 하는 공간적 측면에서도 불편함이 생긴다. 이를 방지하기 위해 linkedlist 자료구조를 사용하였다.

반면 heap sorting을 하는 과정에서는 1차원 배열을 사용하였다. 그 이유는 heap sorting 자체의 특성 때문이다. Heap sorting은 완전 이진 트리를 이용한다. 컴퓨터 특성상 홀수든 짝수든 /2로 하면 트리로 표현했을 때 부모 노드가 나옴을 알 수 있다. 그래서 linkedlist 보다 더 편하게 부모 노드를 찾아 갈 수 있는 장점이 있다. 가장 첫 노드와 가장 접근 가능한 마지막 노드를 변환하는 과정에서도 굳이 추가적인 메모리 할당 없이 두 노드를 바로 교환이 가능하다.

만약 Linkedlist로 구현했다면 for문을 통해 마지막 노드까지 순차적으로 탐색을 해야 하고 이 과정에서 첫 노드를 추가적인 메모리 할당으로 임시적으로 저장하고 있어야한다. 즉 시간 복잡도가 이과정에서만 O(n) 이 발생하고 어떤 정보일 지는 모르겠지만 최소 임시 저장할 때 추가적 메모리 할당을 인해 공간 복잡도 역시 O(1) 이상이 발생한다.

결과적으로 heap sorting이 공간 복잡도, 시간 복잡도 측면에서 유리하면서 여기에 배열 자료구조를 사용하는 것이 최고의 공간, 시간 복잡도를 보임을 알 수 있었다.

5.

사실 여기에 대한 설명은 1번에 많이 기술하였다. 우선 실험자는 waterfall에 관련해서 게임을 만들고 싶었다. 다만 그 당시는 물이 한줄기만 떨어져서 많이 아쉬웠다. 그래서 어떻게 하면 두 개 이상의 물줄기가 떨어질 수 있을 까 고민했었다. 그 고민을 한 번 이번 프로젝트에 대입하고 싶어 다음과 같은 공이 두 개 이상 떨어질 수 있는 알고리즘을 구현 한 것이다. (#define을 바꾸면 여러개 공이 떨어지는 것도 가능하다.) 그리고 여기서 그치는 게 아니라 바구니 개념을 도입해 공을 담으면 다시 위에서부터 rand 함수를 도입하여 랜덤 한 위치에서 떨어지도록 하는 게임을 구현했다. 선분 역시 왜 항상 타고 내려 가야 하는지에 대해 고민하고 만약 선분이 공을 담을 때마다 계속해서 random 하게 주어지고 random하게 선분을 타고 내려가지 않는다면 단순히 공만 줍는 게임이 아닌 조금 더 특별한 난이도를 가지는 게임을 만드는 방법을 생각했다.

그리고 원래는 이 과정에서 멈추려고 했었는데 특정 수업에서 bubble sort 과정에 대해 배웠다. 이 과정은 일반적으로 우리가 많이 사용하는 sorting 개념이다. 비록 시간 복잡도 측면에서는 O(n^2) 이지만 매우 단순한 코드를 구현할 수 있다. 하지만 우리는 시간 복잡도, 공간 복잡도를 모두 고려해야 한다. 그래서 최악의 경우에서도 O(nlogn) 의 시간 복잡도, heap 배열로 할당 시 O(n)의 공간 복잡도를 보이는 heap sorting에 대해 조사하였다.

그래서 조금 더 창의적인 코드 구현 및 각 모드에 따른 rank 정보, 이 둘을 종합한 total rank 정보 역시 확인 가능하다.

linkedlist, 배열에 대한 자료 구조 특성 또한 다시 공부 할 수 있는 프로젝트이다.

실험자는 openframework 를 최대한 이용하고 싶었다. 이는 단순히 도형만 표현되는 프로그램이 아니라 마우스 클릭, 키보드 이용 등 다양한 작동이 가능하다. 그래서 알고리즘, 자료구조뿐만 아니라 디자인에도 신경을 많이 썼다. 우선 색깔을 ofColor:: 를 통해 다양한 색깔을 줄 수 있었고 rand 함수를 통해 난수를 받아와 이 난수로 r,g,b 를 주어 떨어지는 공의 색깔을 계속 해서 달라지게 해 반짝이는 것처럼 표현했다. 그리고 게임 화면의 오른쪽 상단에 목숨 개수 및 score 도 고민해가며 구성했다. 글자를 화면에 출력하고 싶어서 찾아보니 drawString 를 이용하면 글자 출력이 가능하였다. 그리고 글꼴이 너무 단순해 보여 추가적으로 글꼴을 bin 폴더에 주었다. Verdana 글꼴을 사용하였다. 글꼴의 포인트를 조절하는 방법도 찾아보았다. loadFont(“verdana.ttf”,포인트) 이렇게 주면 되었다. 그 외에도 결국 특정 화면 공간을 클릭하면 다음 화면으로 넘어가는 듯한 느낌을 주는 방법도 고민하여 ofClear 를 통해 이를 구현하였다. 그로 인해 menu 화면 게임 진행 화면, 게임 over 화면 이 3가지가 생길 수 있었다.

이런 고민들을 통해 다음과 같은 게임 프로젝트가 나왔고 이는 시간, 공간 복잡도를 고려한 효과적인 알고리즘, 자료구조 등을 활용하고 충분히 창의적으로 이 프로젝트를 진행 했음을 자신한다.

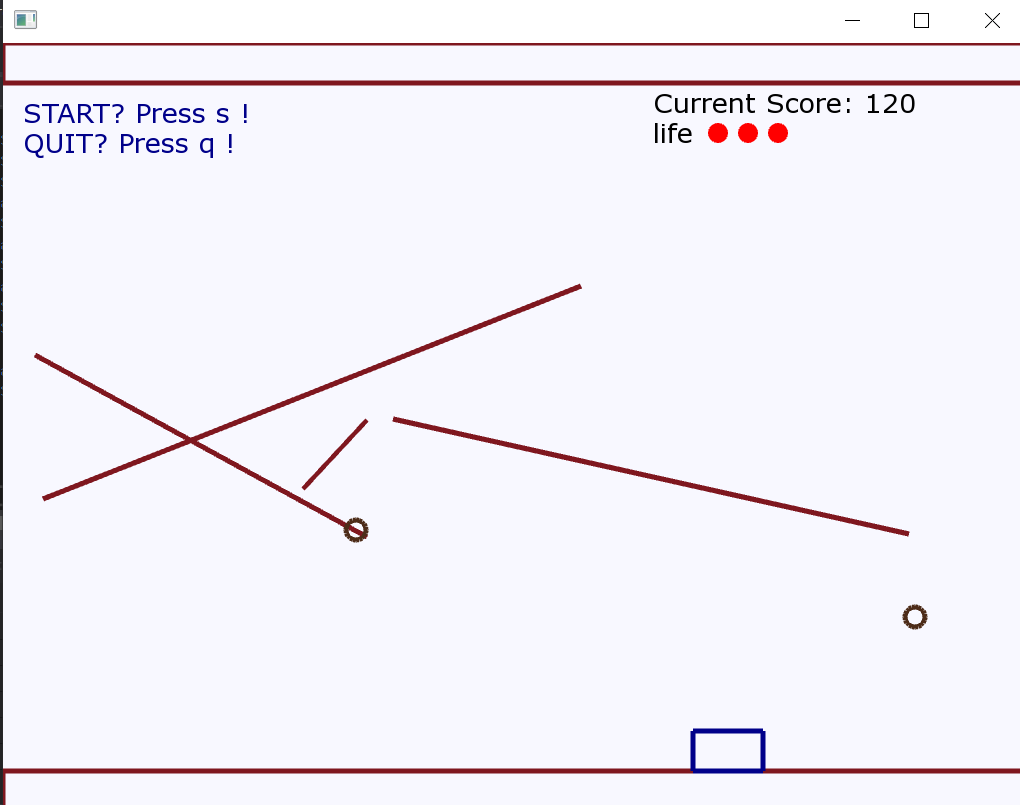
6.

테이블이(가) 표시된 사진

자동 생성된 설명

메뉴 화면이다.

오른쪽에 heap sorting으로 표현한 total 랭킹이 보인다.



Hard mode 클릭 시 떨어지는 공이 2개가 보임을 알 수 있다. 여기 현재 스코어 및 목숨도 화면에 표시가 된다. 아래 직사각형은 바구니이다.

테이블이(가) 표시된 사진

자동 생성된 설명

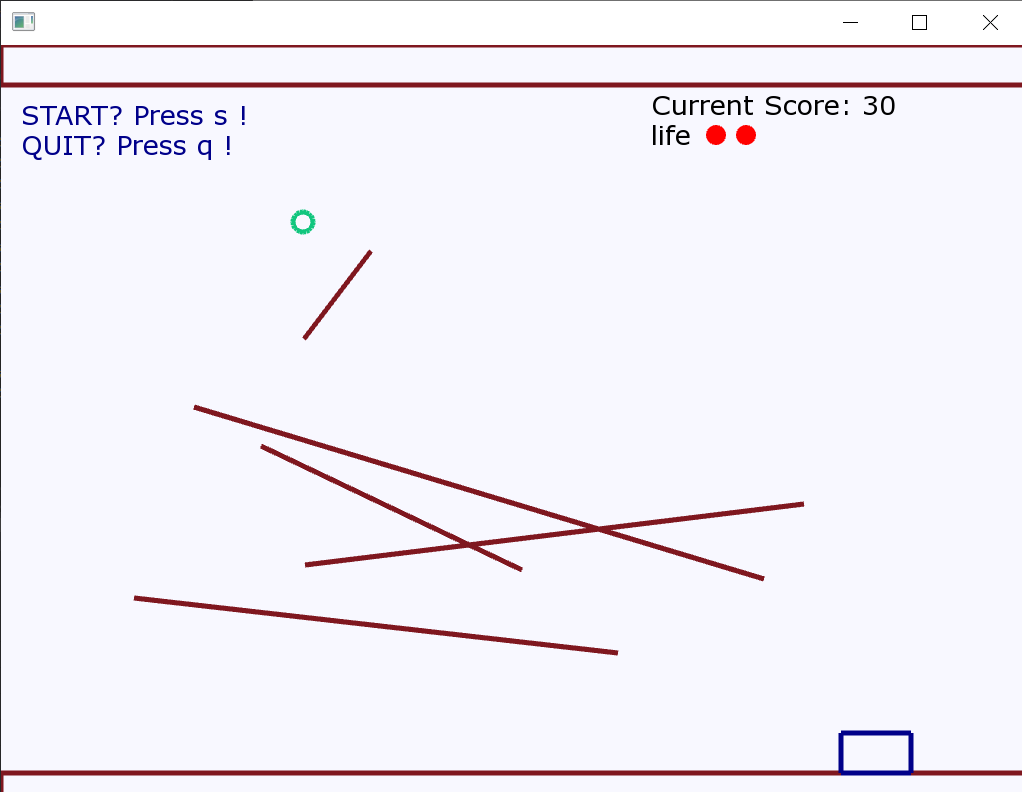
게임에서 죽으면 다음과 같은 화면이 나온다. 여기서 오른쪽은 hardmode의 top5 랭크가 나온다. 이때 현재 사용자의 점수가 120점이니 2등 점수임을 확인 가능하다.

Re\_game 버튼을 누르면 다시 메뉴화면으로 이동하고 quit을 누르면 종료된다.

테이블이(가) 표시된 사진

자동 생성된 설명

현재 오른쪽 전체 top 5에 아까 게임했던 3위에 120점이 추가된 것을 확인 할 수 있다.



Easy\_mode에서는 공이 하나만 떨어지는 것을 확인 할 수 있다.

테이블이(가) 표시된 사진

자동 생성된 설명

Easymode 게임 종료 후 결과 화면이다. 5위에 해당 점수가 들어있음을 확인 가능하다. Quit을 눌러 나가겠다.

7.

이렇게 처음으로 아무것도 주어지지 않은 상태에서 프로젝트를 진행 한 것은 거의 처음이다. 그래서 어떤 방식으로 짜야 할 지 상당히 고민했었다. 무엇보다 평가 기준에 알고리즘이 핵심인 것 같아 도대체 어떤 새로운 알고리즘을 구현해야 할까? 그리고 알고리즘 자체의 정의에 대해서도 많이 찾아본 것 같다. 어떻게 보면 이 게임이 돌아가는 것 자체가 하나의 알고리즘 일수도, 혹은 미로 경로 구현 시 사용한 dfs, bfs 처럼 heap sorting 이 알고리즘 일수도 있다. 결국 이 두 개를 모두 사용하는 방향으로 프로젝트 계획을 잡았다.

가장 처음 힘들었던 것은 공이 여러 개 떨어지는 것을 구현하는 것이다. 어느 순간 공이 두 개 떨어지다가 하나가 사라지는 일들이 발생하였다. 이 문제를 해결하기 위해 1차적으로 엄청나게 시간을 많이 썼다. 결국 어느 위치에 공의 위치를 업데이트 하는 지 알고리즘을 문제 였다. 그래서 그 방법을 해결했다.

두번째는 heap sorting 구현 방법이다. 자료 조사 시 이해는 되었지만 당연히 copy를 하면 안되기 때문에 자료만 읽고 직접 구현하였다. 첫 min heap 은 잘 만들었지만 이 min heap 에서 어떻게 heap sorting을 해야 하는지 코드를 구현했지만 잘 안 되었다. 이 과정만 거의 7시간 동안 고민했다. 비록 많이 알고리즘을 짜보지 않은 문제일 수 있지만 다시 정신을 차리고 차근차근 노트에 완전 이진트리를 그려가며 문제점을 찾아보았다. 여기서 문제점은 계속해서 root노드와 접근 가능한 노드 중 가장 하위 노드를 서로 바꾸는데 그 접근 가능한 노드를 어떻게 설정하는지에 대한 문제였다. 그래서 check 변수를 추가하여 현재 부모노드에서 왼쪽 자식과 오른쪽 자식중 어느 노드가 더 작은지 확인하여 그 check 변수에 index를 넣고 부모노드와 heap[check] 를 비교하는 방식으로 표현하니 다행히 내림차순 sorting이 잘 되었다.

테트리스 프로젝트 정도는 아니지만 상당히 시간이 오래 걸렸다. 그래도 이런 프로젝트를 혼자서 진행하고 창의적인 생각을 코드에 직접 구현해보고 에러나 표현이 안되는 점을 차근차근 고민해봤다는 것에서 상당히 큰 의미가 있다. 그리고 linkedlist array의 자료구조 차이도 다시 한번 알게 되는 좋은 기회였다.

물론 개선 사항도 존재한다. 먼저 시간이 부족해서 랭킹 정보를 출력할 때 top 5 랭킹만 출력하도록 하였다. 이때 특정 버튼을 만들어서 이를 누르면 사용자의 점수를 탐색해서 점수 기준 위 2개 아래 2개의 점수도 보여주면 더욱 좋을 것 같다. 그리고 openframework는 키보드 입력에 따라 이름 입력도 가능할 것으로 보인다. 하지만 이 방법은 제대로 조사를 못하기도 했고(아는 방법은 글자 하나당 하나씩 누를 때 각 글자를 출력하는 것이다.) 큰 의미는 없을 것 같기 때문에 이를 제외하기는 했다. 하지만 만약 추가한다면 더 좋은 rank 정보를 출력할 것이다.

그리고 사용자가 직접 공의 개수를 입력했으면 어땠을까? 이런 생각도 든다. 아마 입력 방법에 차이가 있겠지만 전체적인 코드 구현에는 큰 차이는 없다.

마지막으로는 rand 함수에 관한 것이다. Rand 함수로 선분 및 공의 위치를 구현 했지만 특정 게임 상황일 때는 거의 비슷한 곳으로 공들이 떨어지거나 공이 선분에 전혀 닿지 않는 경우도 생긴다. 그리고 공이 line을 타고 내려갈 때 아주 드물지만 두 선분 사이에 끼는 경우가 생길 수도 있다. Random하게 line을 타도록 설정하여 이 경우는 거의 없지만 지속적으로 두 선분에 대해 접근하려는 경우가 발생했었다. (하지만 현재는 line을 타고 내려가는 조건문에 대한 코드 개선을 했기 때문에 이런 경우가 발생 안 할 것이다.)

이런 개선 사항들을 추가하여 코드를 구현하면 더 좋은 게임이 될 것 이다.