**Tower of HANOI**

컴실1 기말프로젝트 보고서

전공 : 수학 / 학년 : 3 / 학번 : 20171273 / 이름 : 심현우

1. Tower of Hanoi 소개



Tower of Hanoi는 유명한 퍼즐이다. 위의 그림을 참고하여 보면 세 개의 기둥(A, B, C)가 있고 원판들이 A기둥에 꽂혀 있다. 원판은 크기 순서대로 크기가 큰 것이 밑에 있게 배치되어 있다. 여기서 하노이의 탑의 퍼즐 규칙은 매우 단순하다. A기둥에 있는 원판들을 동일한 모양으로 C로 옮기면 된다. 이 때, 한 번의 행동으로 하나의 원판을 옮길 수 있으며 작은 원판 위에 큰 원판이 올 수 없다는 규칙만 지키면 된다. 이를 해결하기 위해 최소한 몇 번의 행동을 해야 하는지 구하는 것이 이 Tower of Hanoi의 퍼즐 규칙 및 목표이다.

1. Tower of Hanoi 해결 방법

Tower of Hanoi를 해결하기 위해서는 규칙에 대한 이해가 필요하다. 원판의 개수를 적은 것부터 하면서 이해하도록 하자.

1. 원판의 개수가 1개

A에 있는 1 원판을 C로 옮기면 목표를 달성한다.

1. 원판의 개수가 2개

A에 있는 원판 중 위에 있는 1 원판을 B로 옮긴 후 2 원판을 C로 옮긴다. 그 후 1 원판을 다시 C로 옮겨 목표를 달성한다.

1. 원판의 개수가 3개

A에 있는 원판을 C로 옮겨 다음 옮길 2 원판이 B로 3 원판이 C로 옮겨질 수 있도록 발판을 마련한다. 1 원판을 C로 옮기고 2원판을 B로 다시 1원판을 B로 옮기면 결과적으로 A에 3, B에 1 2 원판이 오도록 된다. 그 후, 3원판을 C로 옮겨 목표를 달성할 수 있도록 한다.

이와 같은 작업을 반복하면 A에서 제일 밑에 있는 원판이 C에 오기 위해서 제일 밑을 제외한 Tower가 B에 전부 이동해 있어야 마지막으로 제일 큰 원판을 C로 옮길 수 있다. 이를 N개의 원판으로 생각하면 다음과 같다.

1. 원판의 개수가 N개

A에 있는 원판 중 제일 큰 원판을 제외한 나머지 N-1개의 원판을 일련의 과정을 통해 그대로 B로 옮긴다. 이 후, N번째 원판을 C로 옮긴 후 다시 일련의 과정을 거쳐 N-1개의 원판 타워를 C로 옮기면 목표를 달성할 수 있다.

이와 같은 해결 방법을 자료구조 및 알고리즘으로 구현하여 C++로 구성할 수 있다.

1. C++에서의 자료구조 및 알고리즘
2. 재귀 함수 알고리즘을 통한 구현

위의 해결 방법 과정을 재귀 함수 알고리즘을 통해 구현하면 간단하게 구현이 가능하다. 여기서 재귀 함수를 통해 구현하여 타워를 옮기는 최소 행동의 수를 구할 수 있다. Pseudo code로 작성하면 다음과 같다.

//하노이 타워를 움직이는 함수이다. N은 원판의 개수 각각 A, B, C는 기둥의 이름이다.

MoveTower(N, from, to, 나머지) {

재귀 함수를 탈출하기 위해 N=1일 때 조건을 걸어준다.

If(N==1)

A에 있는 원판 한 개를 C로 옮긴다.

Else

MoveTower(N-1, from, 나머지, to);

//위에서 확인한 것처럼 N개의 원판을 옮길 때는 N-1개의 원판을 우선 B로 옮긴 후 마지막 제일 큰 원판 N을 C로 옮긴다.

A에 있는 ‘N’ 원판을 C로 옮긴다.

MoveTower(N-1, 나머지, from, to);

위와 같은 수도코드에 따르면 N개의 원판은 재귀 함수를 돌면서 A를 제외한 나머지 기둥 B, C로 타워 전체를 이동시킬 수 있다. 여기에 따로 변수를 두어 count해주면 최소 이동 횟수를 구할 수 있다.

이동 횟수를 계산해서 구하면 다음과 같다.

an을 n개의 원판을 옮기는 최소한 횟수라고 할 때, a1=1이다. 여기서 n+1번째를 보면 n개의 원판을 나머지로 보낸 후 n+1번째 원판을 도착점으로 옮기는 횟수 한번, 다시n개의 원판을 도착점으로 보내는 방법으로 계산이 가능하다. 따라서 식으로 세워보면 an+1=2an+1이 된다. 이 점화식의 일반항을 구하게 되면 an=2n-1이 된다.

1. Stack 자료구조를 이용한 구현

Stack이란 LIFO(Last In First Out) 자료구조로 데이터를 입력할 시 마지막 배열에 데이터가 저장되고, 데이터를 제거하면 마지막으로 넣은 데이터가 빠져나오는 자료구조이다. 이는 하노이 탑을 구현하기에 최적화된 자료구조로 볼 수 있는데, 이는 하노이 탑 자체가 stack과 같은 역할을 하기 때문이다. 하노이 탑에서는 마지막에 넣은 원판을 제일 먼저 그 기둥에서 제거하기 때문에 stack과 완전히 동일하다고 할 수 있다. 재귀함수를 이용하지 않고 반복문과 stack을 통해 하노이탑을 구현할 수 있다.

간단한 pseudo code로 보면 다음과 같다.

While(1)

While(n>1) {

목적지 기둥 stack에 데이터 추가

나머지 기둥 stack에 데이터 추가

출발점 기둥 stack에 데이터 추가

scale감소

목적지와 나머지 기둥 stack 교환

}

If(stack이 비어있는 경우) {

stack에서 데이터를 꺼냄

나머지와 도착점 stack 교환

}

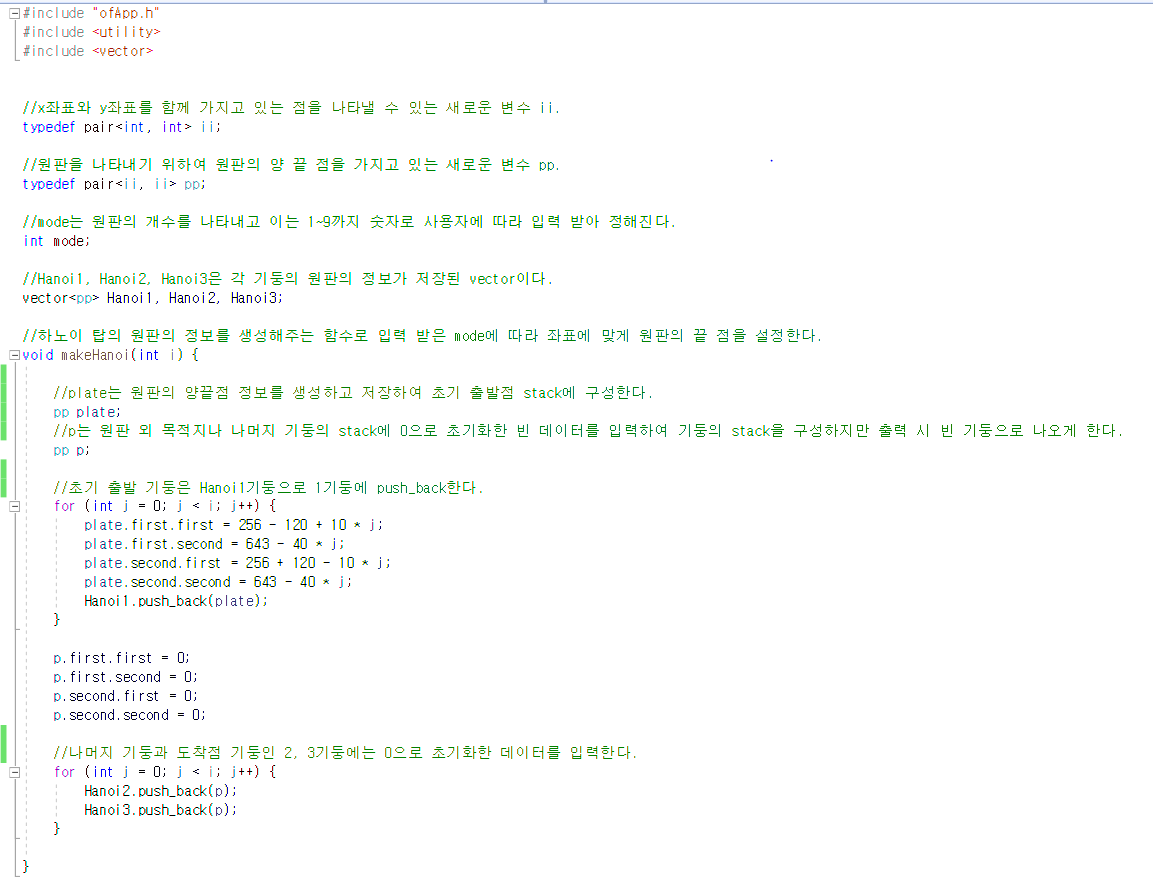
1. openframeworks에서의 구성

Hanoi의 탑을 of에서 구성하기 위해 최소한의 횟수를 구하는 것이 아닌 of의 visual 기능을 이용하여 직접 하노이의 탑의 원판을 움직이고, 자신이 플레이한 횟수를 구하는 프로그램을 구성했다. 기존의 목표달성인 횟수 구하는 것을 구성하지 않기에 재귀 함수를 이용하지 않고 자료구조인 stack과 반복문에서의 과정을 하나의 과정으로 나눈 알고리즘을 이용하여 각각의 케이스를 출력할 수 있도록 했다. 이를 통해 사용자가 직접 작동하는 하노이탑을 구성하는 것이 목표였다.

1. 자료구조

실제 코드 구성에서는 stack대신 vector를 사용했다. Vector는 배열의 일종으로 크기가 정해져 있지 않은, 동적 할당이 가능하다. 또한 배열의 특성을 가져 index로 배열의 데이터에 접근할 수 있으며 마지막 위치나 첫 위치에 데이터를 추가하거나 삭제하는 것에 용이하다. 이 프로젝트에서 stack으로 구성하지 않은 이유는 사용자의 입력에 따라 변화하는 원판을 화면에 출력해야 하는데 stack은 데이터를 출력함에 있어 모든 데이터를 pop을 통해 접근해야 하기 때문에 임의의 temporary stack을 두어 데이터를 저장하고 출력할 수 있도록 해야 하기 때문이다. 따라서 vector를 통해 stack구조를 이용해 데이터를 추가하고 마지막 데이터를 가져오는 작업과 출력하는 작업을 동시에 수행했다.

1. 구성한 코드

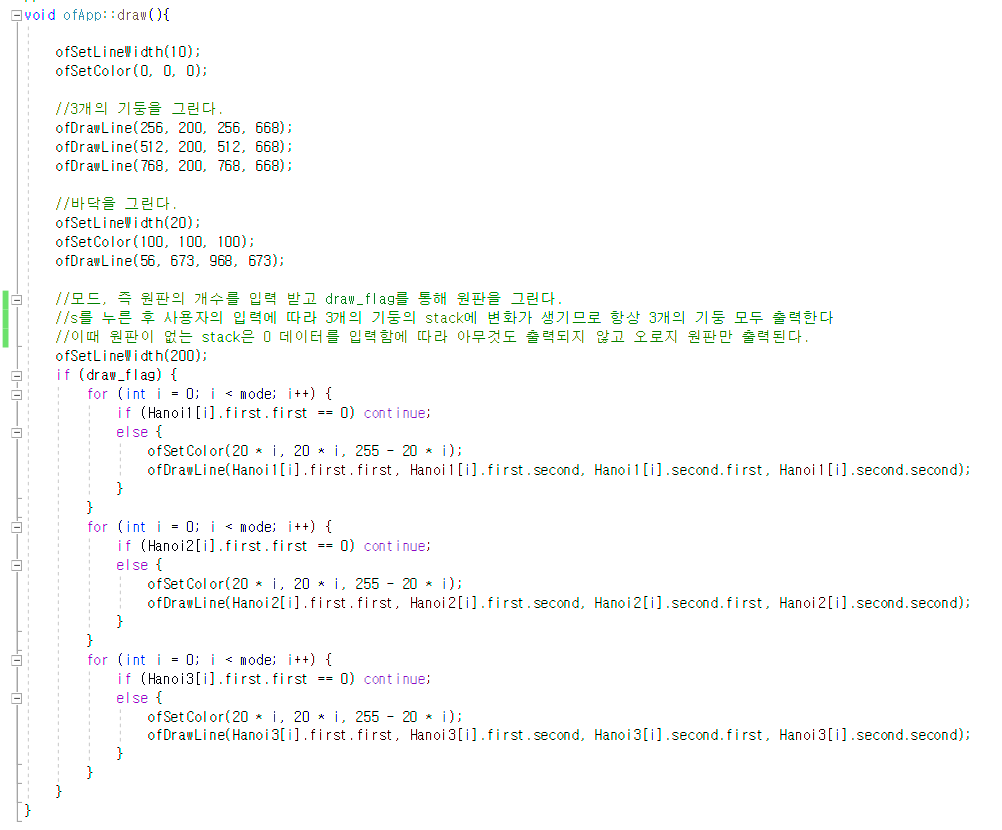


-변수 설정

우선 원판의 정보를 저장하기 위한 변수를 생성했다. Pair를 통해 ii는 x좌표와 y좌표를 가지고 있는 하나의 점인 변수로 설정했고, pp 또한 pair를 통해 ii와 ii를 가지는 즉, 양끝점의 좌표를 가질 수 있는 변수로 구성했다. 이는 pp a라고 선언했을 때, a.first.first , a.first.second가 a 원판의 왼쪽 점의 좌표, a.second.first, a.second.second가 a 원판의 오른쪽 점의 좌표이다. 이를 vector<pp>를 통해 pp를 데이터로 갖는 vector를 선언했다. 여기서 기둥이 3개이므로 각각 Hanoi1, Hanoi2, Hanoi3로 선언했다.

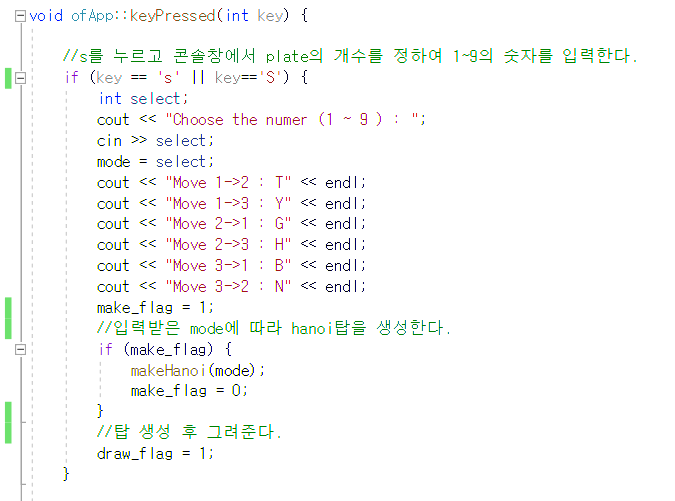
-makeHanoi

makeHanoi는 Hanoi의 원판의 데이터를 구성하여 원판을 만드는 함수다. 이는 계산에 의하여 배경 인터페이스에 맞게 생성되고 1번 째 기둥에 차례대로 존재할 수 있도록 설계되었다. 나머지 Hanoi2, Hanoi3에도 0의 데이터를 넣어 출력을 할 수 있도록 초기화 했다.



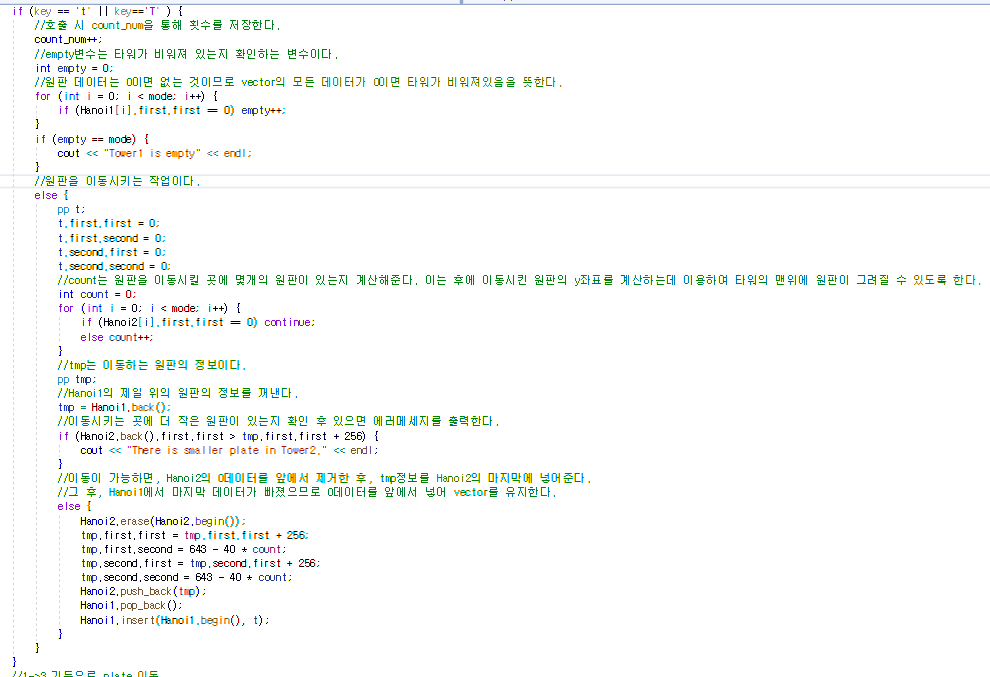
-draw

배경에 맞게 좌표를 추적하여 바닥과 기둥 3개를 그렸다. 그 후, 사용자에 입력에 따라 draw flag가 1이 되어 활성화되면 원판의 데이터를 받아 원판을 그려줬다. 이 때, 처음 0으로 초기화해줬던 배열의 데이터들은 원판이 아니기 때문에 draw에서 해당되지 않는다.



-keypressed s

사용자는 프로그램을 실행 후, s를 누르면 콘솔창에서 숫자를 입력하게 되어있다. 사용자의 입력에 맞게 원판의 개수가 정해지며 makehanoi로 원판의 데이터를 생성하고 화면에 draw flag를 통해 draw해준다. 여기서 기본적인 조작법을 콘솔창에 띄워준다.



-keypressed t

사용자의 입력에 맞게 원판을 이동시키는 코드이다. vector를 이용하여 원판이 빠지는 기둥에서는 원판의 데이터를 넘겨주고 0의 데이터를 다시 입력해 넣는 것으로 원판을 제거한다. 원판을 받는 기둥에서는 0의 데이터를 제거해주고, 원판의 데이터를 넘겨받아 원판의 개수에 맞게 위치를 조정하여 데이터를 받는 것으로 한다.

이 때, vector의 상황은 다음과 같다. 초기 상황 plate는 3개인 것으로 예시를 들어본다.

<Vector 끝 부분>

|  |  |  |
| --- | --- | --- |
| **원판 1** | **0** | **0** |
| **원판 2** | **0** | **0** |
| **원판 3** | **0** | **0** |

<vector 시작 부분>

왼쪽부터 오른쪽으로 Hanoi1, Hanoi2, Hanoi3이다. Vector의 index는 위쪽이 더 큰 인덱스를 가진다. 즉, 아래쪽이 vector의 앞, 위쪽이 vector의 뒤이다. Makehanoi를 마치면 위와 같은 상태가 된다. 여기서 vector의 흐름에 따라 Hanoi1에서 Hanoi2로 원판을 옮겨보도록 하자. 우선 옮기기 전에 Hanoi1이 비어 있는지, Hanoi2 마지막 데이터에 원판 1보다 큰 원판이 있는지 확인한다. 지금은 해당 없으므로 진행한다. 옮기기 전 vector의 시작 부분에서 데이터 0을 지운다.

|  |  |  |
| --- | --- | --- |
| **원판 1** | **0** | **0** |
| **원판 2** | **0** | **0** |
| **원판 3** |  | **0** |

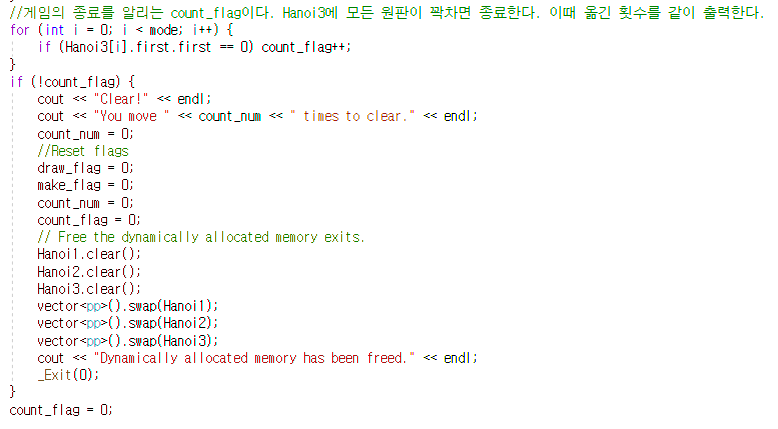
이 후, 원판 1을 vector의 끝 부분에 삽입한다.

|  |  |  |
| --- | --- | --- |
| **원판 1** | **원판 1** | **0** |
| **원판 2** | **0** | **0** |
| **원판 3** | **0** | **0** |

이 후, Hanoi1에서 vector의 끝 부분 데이터를 삭제하고 vector의 시작 부분에 0데이터를 삽입한다.

|  |  |  |
| --- | --- | --- |
| **원판 2** | **원판 1** | **0** |
| **원판 3** | **0** | **0** |
| **0** | **0** | **0** |

위와 같은 알고리즘을 반복하여 원판을 이동시킨다. 여기서 원판의 위치는 y좌표를 통해서 따로 조정해주기 때문에 출력에는 문제가 없다. 또한 0은 출력이 안되기 때문에 문제가 없다.



-종료

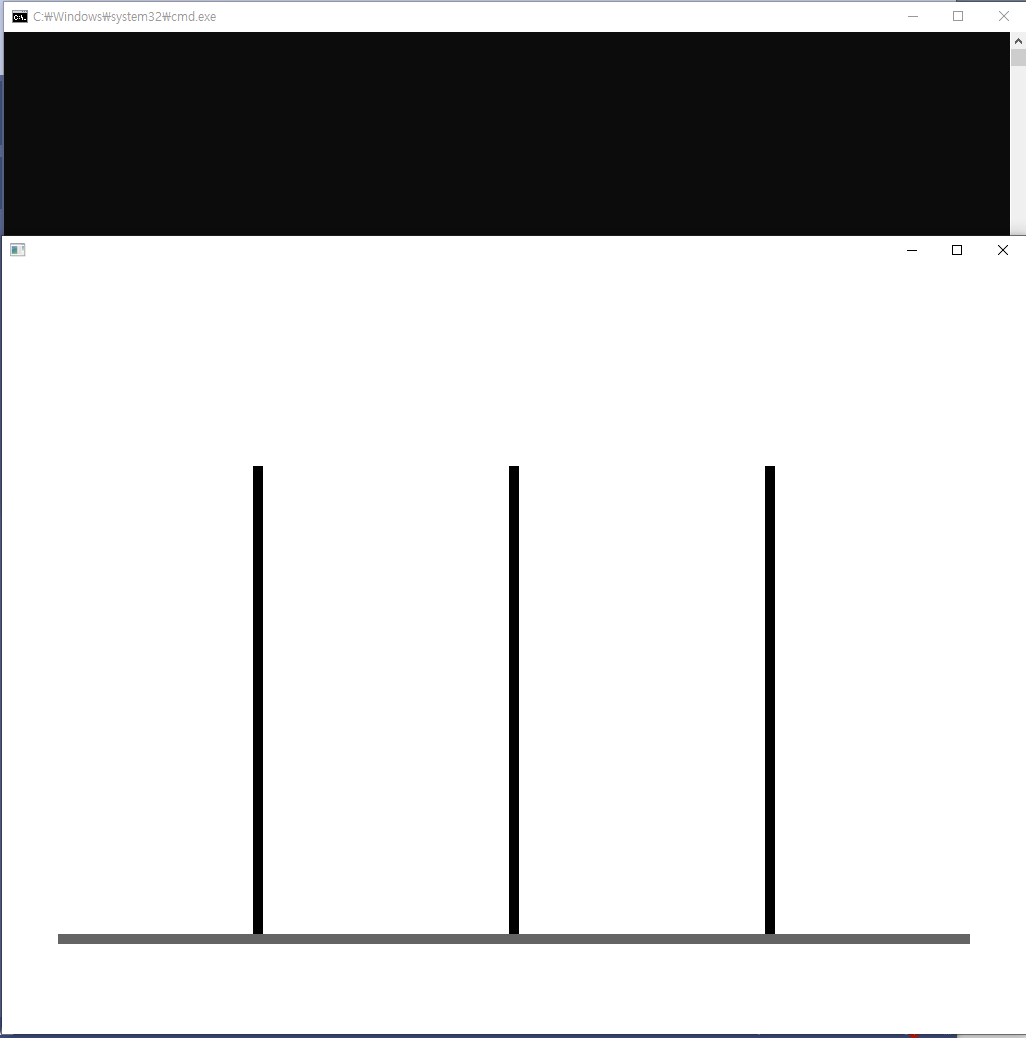
Count\_flag를 통해 도착지점인 hanoi3에 원판의 개수를 판단하고 모든 원판이 3으로 이동되면 콘솔 창에 종료 메세지와 횟수와 함께 프로그램이 종료된다. 이는 1->3으로 원판을 옮기거나 2->3으로 원판을 옮길 때만 호출된다.

-시간 및 공간 복잡도

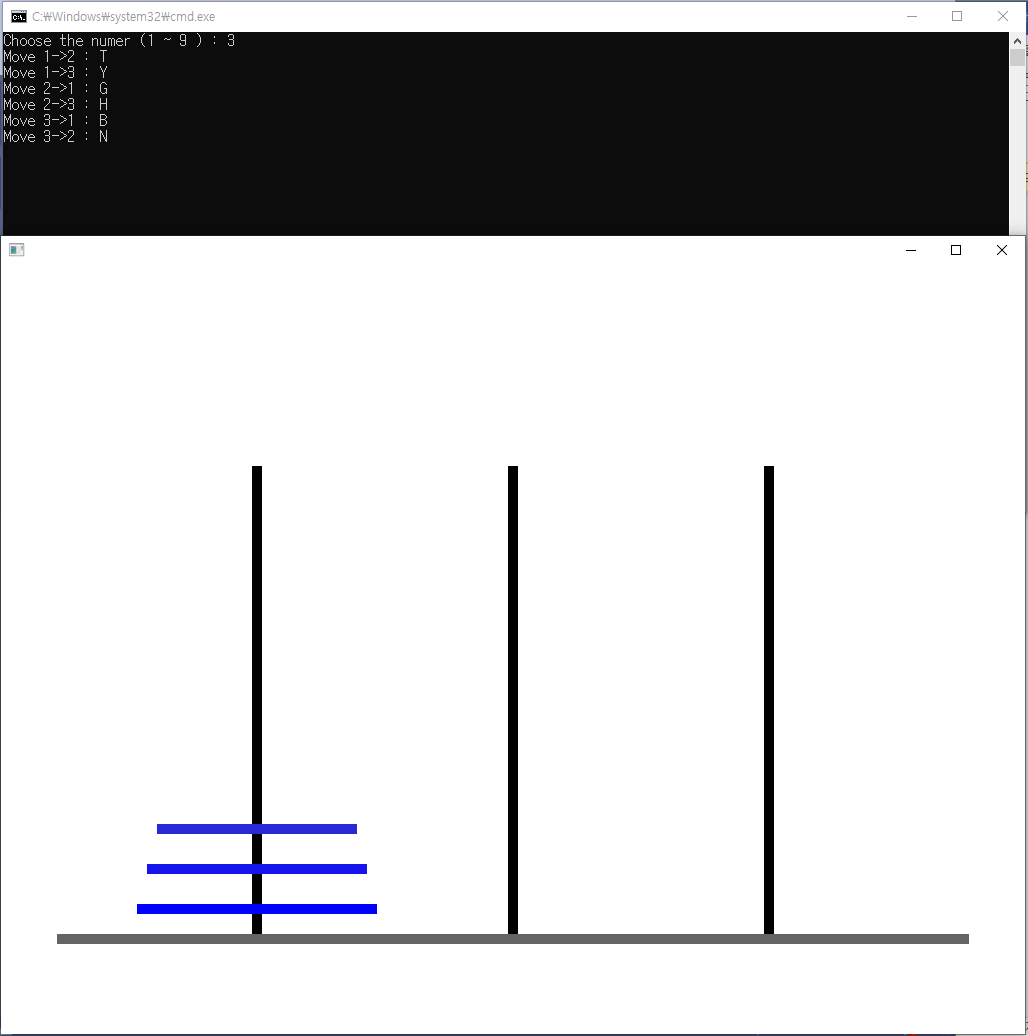
MakeHanoi나 draw는 하노이탑의 원판 개수만큼 반복해야 하는 함수이므로 O(원판의 개수)의 시간 복잡도를 갖는다. 이 외, s키는 makehanoi를 호출하므로 또한 O(원판의 개수)이다. 원판을 움직이는 키에 경우 원판을 옮기는 과정 자체에는 반복문이 없고 데이터에 접근, 삭제, 삽입 과정만 있기 때문에 O(1)이나, 원판의 y좌표를 계산하거나, 원판의 개수를 통해 게임을 종료하거나 빈 기둥일 경우를 계산할 때 vector를 반복적으로 돌아 확인해야 하므로 O(원판의 개수)의 시간 복잡도를 갖는다.

공간 복잡도는 vector를 사용하여 dynamic allocation을 하기 때문에, 사용자가 입력하는 만큼의 메모리를 할당한다. 따라서 공간 복잡도 또한 O(원판의 개수)이다.

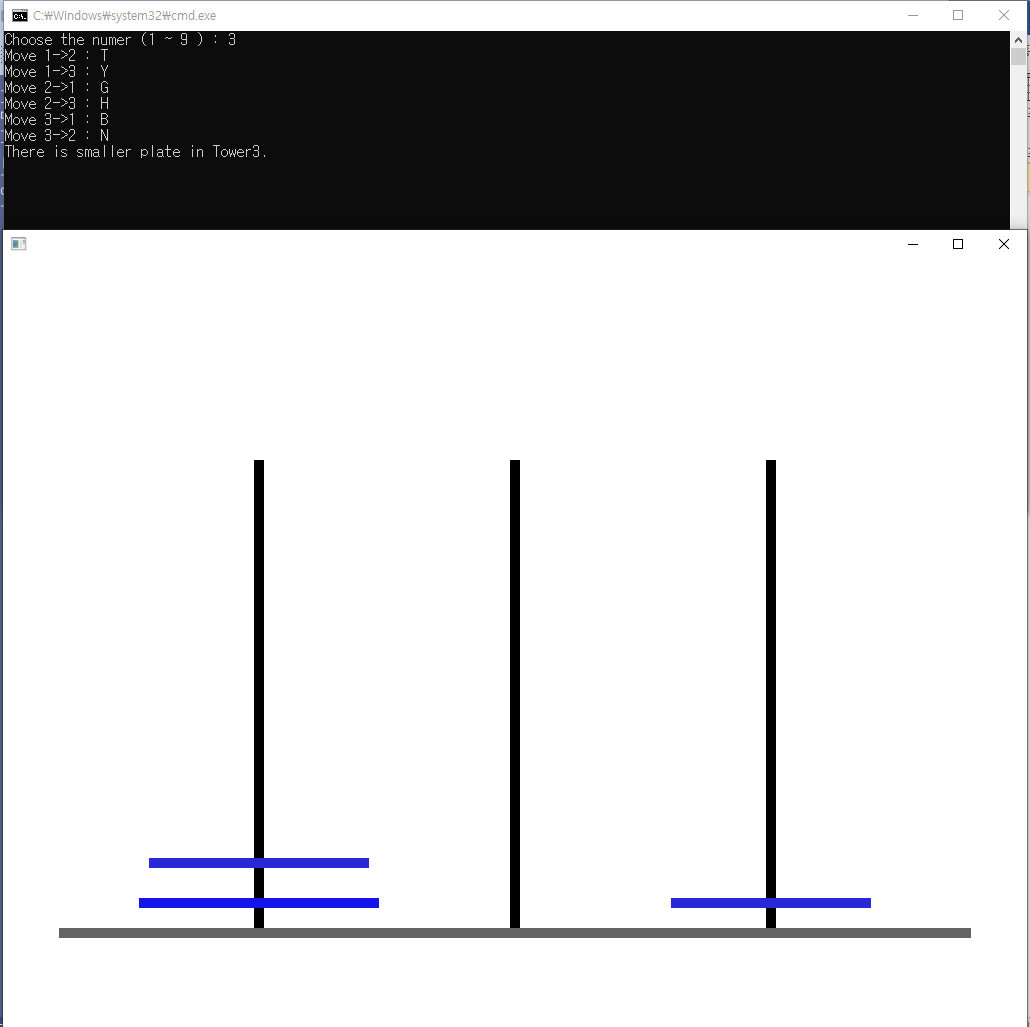
1. 프로젝트 실행 결과



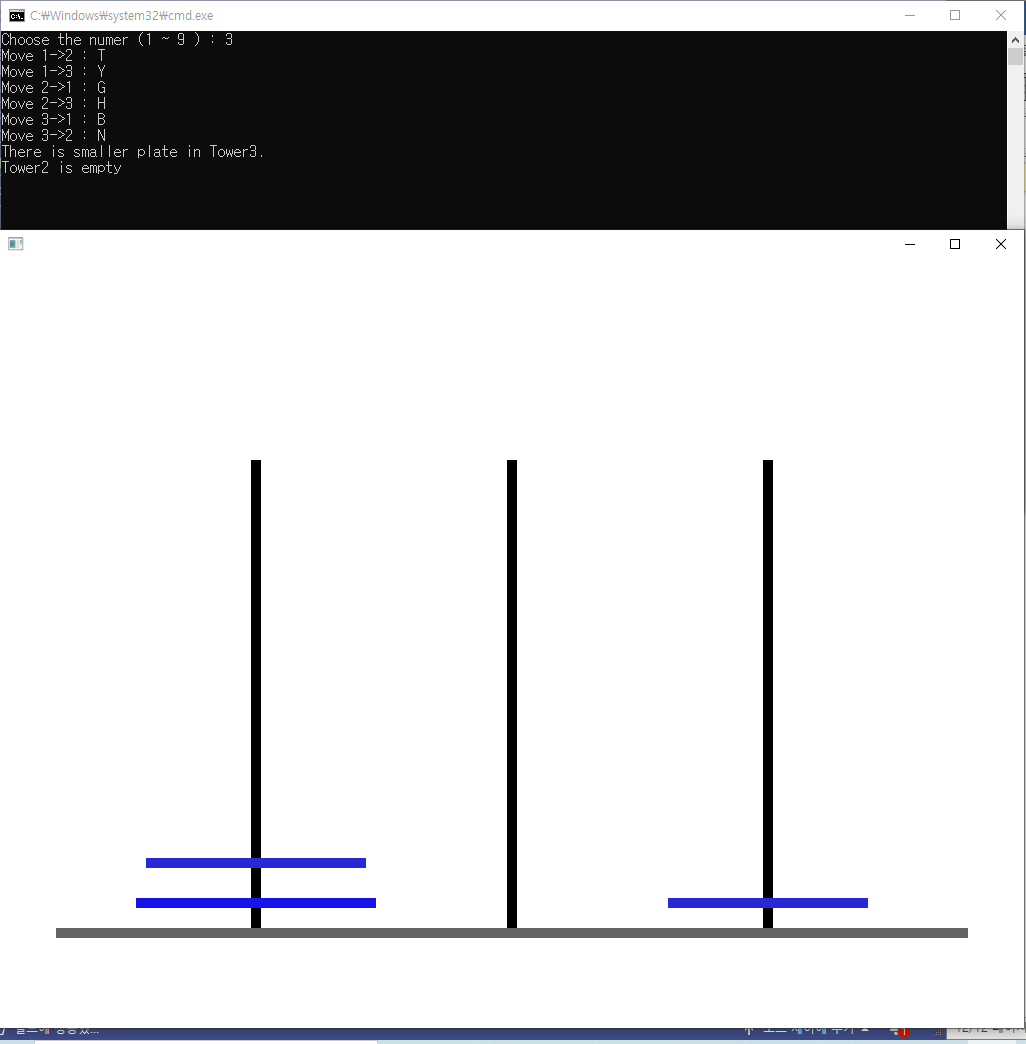
초기 실행 결과이다. 기둥과 바닥이 draw된 상태이다. 이 상태에서 s를 누르고 콘솔창을 확인한다.



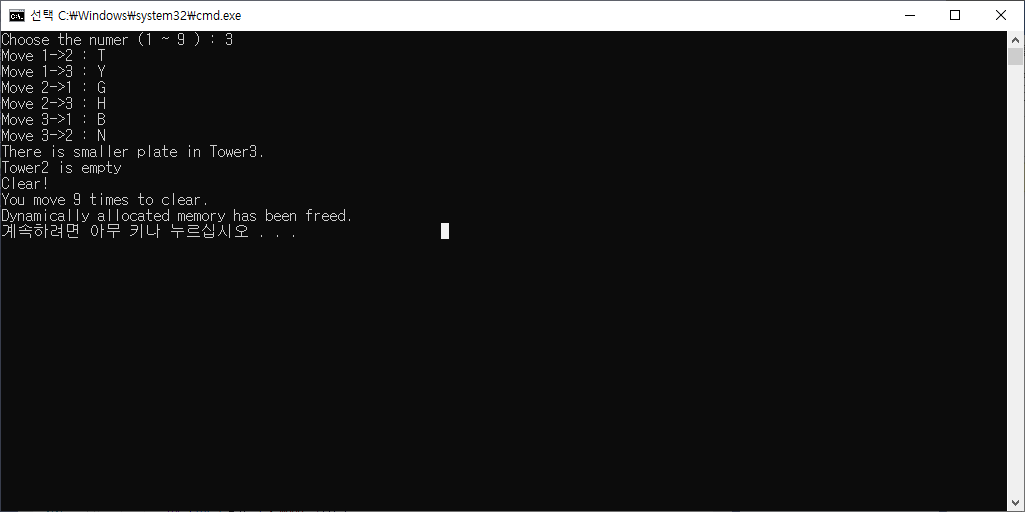
콘솔창에 3을 입력한 결과이다. 안내문과 함께 타워에 원판이 표시된다. 이 후, 조작키를 이용해 움직인다.



이 상황에서 한번 더 1에서 3기둥으로 원판을 옮기려 하면 에러메세지가 출력되면서 옮겨지지 않는다.



또한 2기둥에서 옮기려 하면 에러메세지 출력 후 옮겨지지 않는다.



하노이탑을 완료하면 메시지를 출력하면서 프로그램이 종료된다.

1. 느낀 점 및 개선사항.

하노이 탑을 처음 생각하고 재귀 함수와 stack으로 구성을 하려고 생각을 했지만 실제 of에서 프로그램 구성 중 단계 별로 실행하는 것이 of에 더 적합하고 이는 재귀 함수를 사용하지 못하는 상황을 맞았다. stack또한 of에서 draw 시 구현하기 어려워 고민 끝에 vector를 선택했다. 하노이탑에서는 중간에 있는 원판의 데이터에 접근하거나 삭제, 삽입 하는 경우가 없으므로 vector를 사용해도 무방했으나 처음 계획했던 stack을 사용하지 못한 것에 고민을 많이 했다. 하지만 vector를 충분히 stack처럼 활용할 수 있으며, index의 접근하는 것도 용이하기에 vector의 장점들을 사용하여 자료구조를 구성했다.