2020년도 2학기 컴퓨터공학설계및실험Ⅰ

11주차 WaterFall-2 week 결과 보고서

20161663 허재성

1. 실습 목적

WaterFall 문제에서 물이 떨어져 흐르는 경로를 계산하여 화면에 나타내는 것을 openFrame Works로 구현한다.

2. 실습 구현 내용

11주차 실습에서는 10주차에서 정의한 Point 클래스와 LineSegment 클래스를 다음과 같이 변경하였다.

Point 클래스

|  |
| --- |
| class Point {  private:  **float** xpos, ypos;  public:  Point();  Point(**float** \_x, **float** \_y);  void setX(**float** \_x);  void setY(**float** \_y);  **float** getX() const;  **float** getY() const;  }; |

LineSegment 클래스

|  |
| --- |
| class LineSegment {  **protected:**  Point leftP, rightP;  **float slope;**  public:  LineSegment(**float** xl, **float** yl, **float** xr, **float** yr);  **float** getLX() const;  **float** getLY() const;  **float** getRX() const;  **float** getRY() const;  // 추가된 함수  **float getSlope() const;**  }; |

2차원 평면 상의 점을 표현하는 Point 클래스는 점의 x 좌표와 y 좌표를 정수형 변수가 아닌 실수형 변수가 되도록 float로 자료형을 변경하였다. 이에 따라 생성자 및 setter 함수에서의 입력 인자의 자료형도 float로 변경되었고, getter 함수의 반환형도 float로 변경되었다.

Point 객체를 포함하는 LineSegment 클래스의 경우에도 생성자의 입력 인자의 자료형을 float로 변경했고, getter 함수의 반환형도 float로 변경하였다. 기존에 접근 제어자가 private로 선언되었던 Point 객체 leftP, rightP는 상속을 위해 protected로 접근제어자를 변경하였다. 또한 선분의 기울기를 나타내는 float형 멤버 변수 slope를 접근 제어자 protected로 추가했으며, slope의 값을 반환하는 getter 함수 getSlope()도 추가하였다.

11주차 실습에서 물 줄기를 표현하기 위해 WaterSegment 클래스를 새로 정의하였다. WaterSegment 클래스는 이미 정의된 LineSegment 클래스를 기반 클래스로 상속하는 유도 클래스로 작성되었다.

WaterSegment 클래스

|  |
| --- |
| class WaterSegment : public LineSegment {  public:  WaterSegment\* link;  WaterSegment(Point& start);  void setLP(Point& lp);  void setRP(Point& rp);  }; |

WaterSegment 클래스는 LineSegment 클래스를 상속하므로 LineSegment의 멤버 변수인 leftP, rightP, slope를 가지고 있으며 또한 protected로 선언되어 있으므로 WaterSegment 클래스에서 접근이 가능하다. 기반 클래스인 LineSegment에서는 leftP, rightP가 각각 선분의 왼쪽 끝 점, 오른쪽 끝 점을 의미했지만, WaterSegment에서는 각각 왼쪽, 오른쪽의 의미는 사라지고 물줄기가 흐르기 시작하는 점, 끝나는 점을 의미한다.

추가된 멤버 변수로 다른 WaterSegment 객체를 가리키는 포인터 link가 추가되었다. WaterSegment 객체를 연결 리스트의 노드로 사용하여 WaterSegment 객체로 이루어진 연결 리스트를 구현할 수 있도록 하기 위해 추가한다..

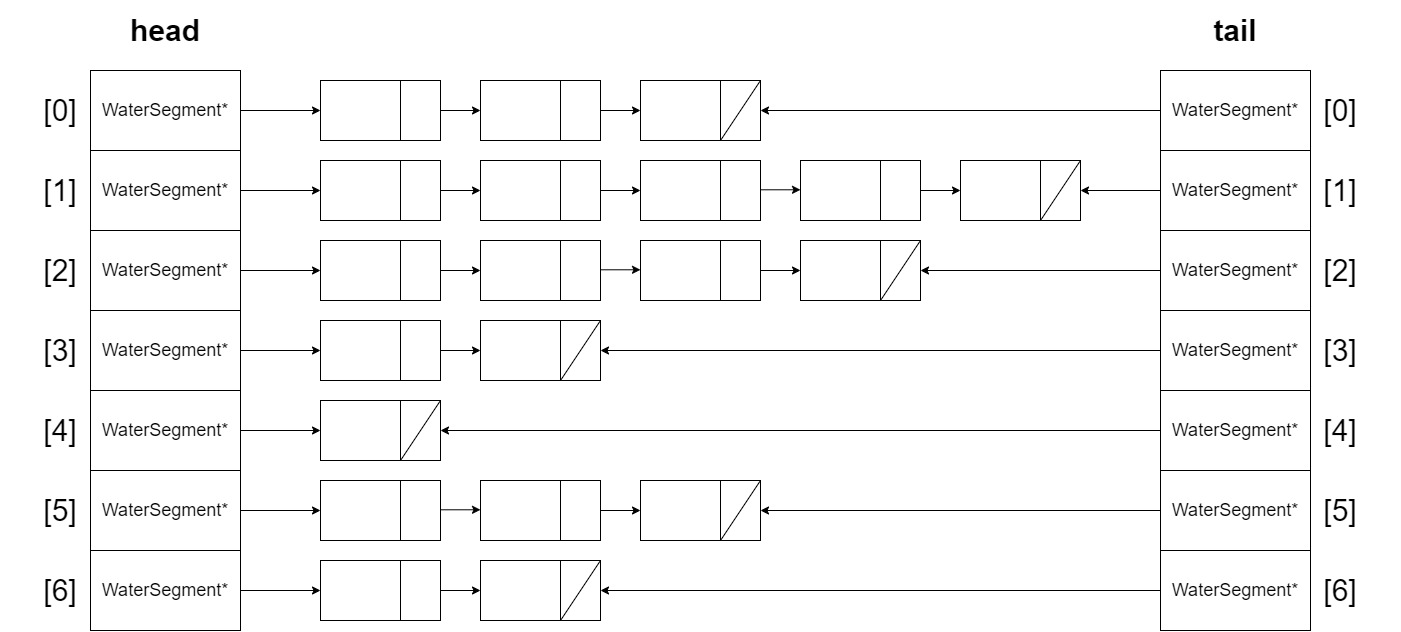
WaterSegment의 생성자로 WaterSegment의 시작점 start를 인자로 받는 생성자를 정의했다. 또한 leftP와 rightP를 각각 lp와 rp로 변경할 수 있도록 setter 함수 setLP()와 setRP()를 정의하였다.

WaterSegment 클래스가 추가됨에 따라 기존의 물이 나오는 구멍과 선반을 통합적으로 관리하는 클래스인 Waterfall 클래스가 WaterSegment 연결 리스트를 관리할 수 있도록 수정되어야 한다. 다음과 같이 수정한다.

Waterfall 클래스

|  |
| --- |
| class Waterfall {  private:  int wh; // 물 구멍의 개수  int sh; // 선반의 개수  int pos; // 선택한 점(물 구멍)의 index  public:  Point\*\* waterhole;  LineSegment\*\* shelf;  **WaterSegment\*\* head; // 물 줄기 표현 LinkedList head 배열**  **WaterSegment\*\* tail; // 물 줄기 표현 LinkedList tail 배열**  **Waterfall();**  **~Waterfall();**  void setWH(int \_wh);  void setSH(int \_sh);  void setPos(int \_pos);  int getWH() const;  int getSH() const;  int getPos() const;  **void createWaterfallList();**  }; |

물 구멍의 개수가 wh개일 때, WaterSegment를 노드로 가지는 연결 리스트의 개수 또한 wh개가 된다. wh개의 각각의 연결 리스트의 가장 앞, 가장 뒤의 원소를 가리키는 wh 크기의 포인터의 배열을 각각 동적 할당하기 위해 WaterSegment 이중 포인터 head와 tail을 멤버 변수로 추가한다. 추가적으로 연결 리스트를 생성하는 멤버 함수 createWaterfallList() 함수를 추가해야 한다. 새로운 포인터 배열을 동적 할당 및 해제해야 하고 연결 리스트를 생성 및 해제해야 하므로 기존의 생성자와 소멸자도 변경해야 한다.



createWaterfallList 함수로 만들어진 연결 리스트는 위와 같다. 물 구멍이 7개일 경우 wh = 7이 되어 위와 같이 7개의 연결 리스트가 만들어진다. 포인터 배열 head의 i번째 원소(0<= i < wh) head[i]는 i번째 연결 리스트의 가장 앞의 노드를 가리키며, 포인터 배열 tail의 i번째 원소 tail[i]는 i번째 연결 리스트의 가장 뒤의 노드를 가리킨다. 연결 리스트에 노드를 추가할 때에는 가장 뒤에 추가한다. 따라서 tail[i]가 가리키는 노드는 노드가 추가될 때마다 변경된다.

수정된 클래스, 생성자, 소멸자, 멤버 함수

단순히 입력, 반환형의 자료형이 변경된 경우는 생략한다.

헤더파일 ofApp.h의 ofApp 클래스에 멤버 변수 water\_flag를 선언한다. water\_flag가 0일 경우, 물 줄기를 그리지 않으며, water\_flag가 1일 경우 물줄기를 그린다.

(1) LineSegment::LineSegment(float xl, float yl, float xr, float yr);

기존의 LineSegment 함수처럼 leftP, rightP를 초기화한다. 추가적으로 선분의 기울기를 의미하는 slope의 값을 기울기 공식을 이용해 초기화한다.

|  |
| --- |
| slope = (yr - yl) / (xr - xl); |

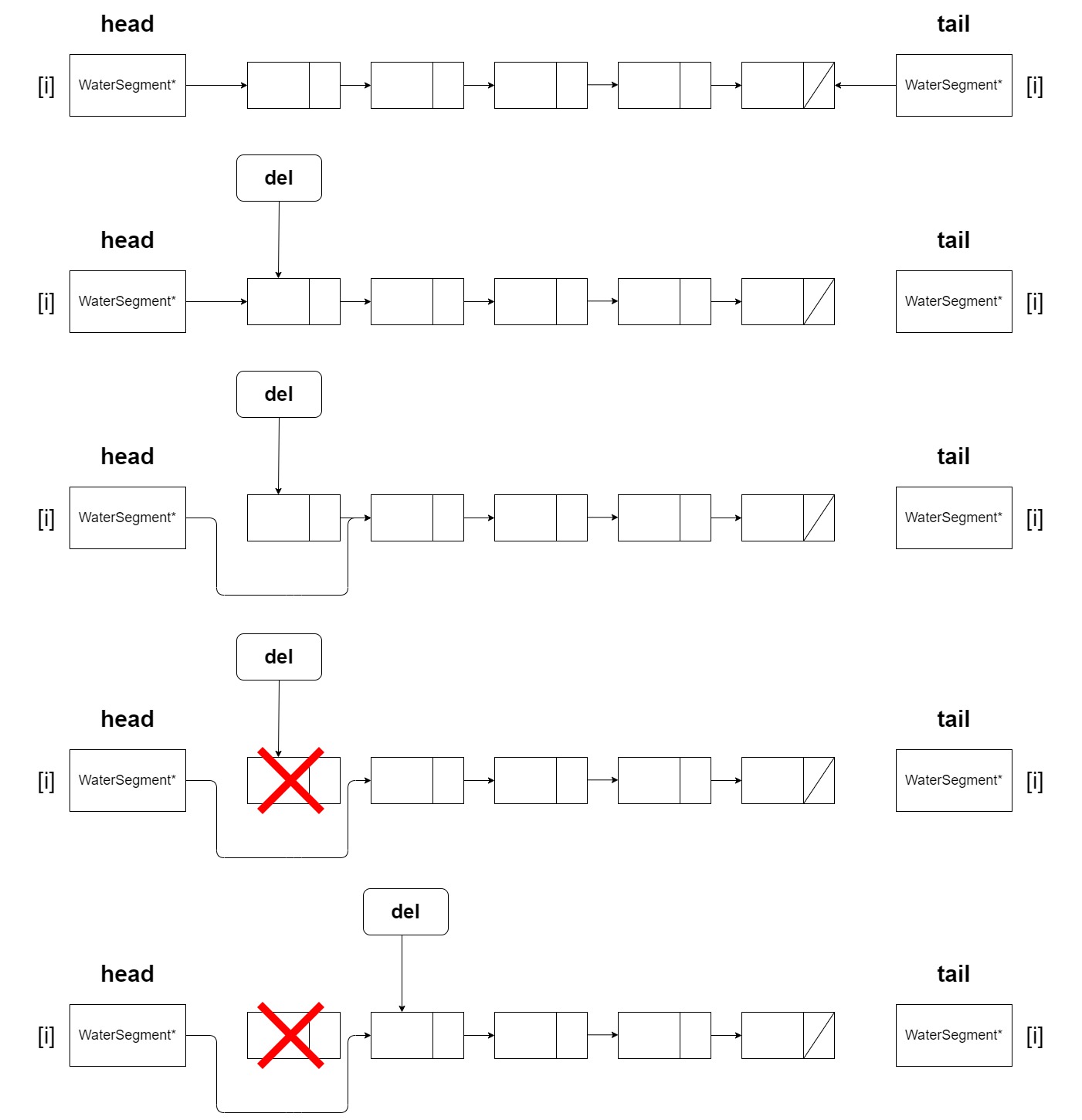
(2) Waterfall::Waterfall()

기존의 Waterfall 클래스의 생성자와 동일한 역할을 한다. 추가적으로 새로 추가된 멤버 변수 head와 tail의 값을 NULL로 초기화한다

(3) Waterfall::~Waterfall()

기존의 Waterfall 클래스의 소멸자와 동일하게 동적 할당된 메모리를 해제하는 역할을 한다. 추가적으로 새로 추가된 포인터 배열 head의 원소들이 가리키는 연결 리스트들을 전부 해제 후, 동적 할당된 포인터 배열 head와 tail을 해제해준 뒤, head와 tail을 NULL로 변경한다. 그 외 동적 할당된 메모리의 해제 방법은 이전과 동일하다.

head[i]가 가리키는 연결 리스트의 해제 과정은 다음 그림과 같다. 가장 앞의 노드부터 차례로 제거한다.



위의 과정을 의사 코드로 나타내면 다음과 같다. 연결 리스트를 해제하는 부분만 나타내었다.

|  |
| --- |
| Waterfall::~Waterfall() {  if (head) {  for (int i = 0; i < wh; i++) {  tail[i] = NULL;  for (WaterSegment\* del = head[i]; del;) {  if (삭제하려는 현재 노드 del의 다음 노드가 있을 경우)  head[i]가 del의 다음 노드를 가리킴;  else 다음 노드가 없을 경우 head[i]를 NULL로 설정;    del->link = NULL; // 다음 노드와 연결된 link 제거  현재 노드 del 해제;  포인터 del이 연결 리스트의 첫 번째 노드를 가리킴;  }  }  동적 할당된 포인터 배열 head, tail 해제;  head = NULL;  tail = NULL;  }  } |

선반의 개수를 N개, 물 구멍의 개수를 M개라 할 때, 하나의 연결 리스트의 노드의 개수는 선반의 개수에 비례하므로 O(N)개이다. 따라서 하나의 연결 리스트를 해제하는 과정의 시간 복잡도는 O(N)이다. 연결 리스트는 물 구멍의 개수만큼 존재하므로 모든 연결 리스트를 해제하는 과정의 시간 복잡도는 O(MN)이다.

(4) void ofApp::processOpenFileSelection(ofFileDialogResult openFileResult)

기존의 기능에 추가로 입력받은 선분 중 수평 선분이나 수직 성분이 존재하는 지 확인 후 존재하면 입력 범위를 벗어났으므로 range\_flag를 1로 설정 후 종료한다. 물 구멍의 개수만큼 Waterfall 포인터 wf의 head와 tail을 동적 할당해 WaterSegment 포인터 배열을 만든다. 그리고 head, tail 포인터 배열의 각 원소마다 물구멍을 시작점으로 하는 WaterSegment 객체를 만들고 리스트에 삽입한다. 마지막으로 initializeWaterLines 함수를 호출하여 물줄기를 표현하는 연결 리스트를 생성한다.

선반의 개수를 M, 물 구멍의 개수를 M이라 하자. 연결 리스트를 생성하는 initializeWaterLines 함수를 제외하고 시간, 공간 복잡도를 계산해보면 시간 복잡도는 물 구멍의 개수만큼 반복하여 WaterSegment 개수를 만들어야 하므로 O(M)이다. 공간 복잡도도 O(M)이다.

(5) void ofApp::keyPressed(int key)

‘s’ 키를 누를 경우 water\_flag를 1로 설정하여 물 줄기가 그려지도록 한다. ‘e’ 키를 누를 경우 water\_flag를 0으로 설정하여 물 줄기가 나타나지 않도록 한다. ‘d’ 키를 누를 경우, water\_flag가 1이면 물줄기가 그려진 상태이므로 어떤 동작도 하지 않고, water\_flag가 0일 경우 선반과 물 구멍을 다시 그린다.

(6) void ofApp::keyReleased(int key)

‘l’ 키를 누를 경우 파일을 열기 전에 water\_flag를 0으로 재설정한다. 방향키를 누를 경우 water\_flag가 1일 경우 물줄기가 그려진 상태이므로 어떤 동작도 하지 않는다. 반대로 water\_flag가 0일 경우 물이 나올 물 구멍의 위치를 왼쪽, 또는 오른쪽 점으로 변경한다. 변경 방법은 이전과 같다.

(7) void ofApp::draw()

water\_flag가 1일 경우, 이미 만들어진 선택된 물구멍에 해당하는 WaterSegment 연결 리스트를 이용하여 물 줄기를 그린다. 물 줄기를 그리기 위해 ofSetColor 함수와 ofSetLineWidth 함수를 이용한다. 연결 리스트의 노드의 개수는 O(N)개이고 연결 리스트에 저장된 모든 WaterSegment에 대해서 선분을 그려야 하므로 시간 복잡도는 O(N)이다.

추가된 생성자, 멤버 함수

(1) float LineSegment::getSlope() const;

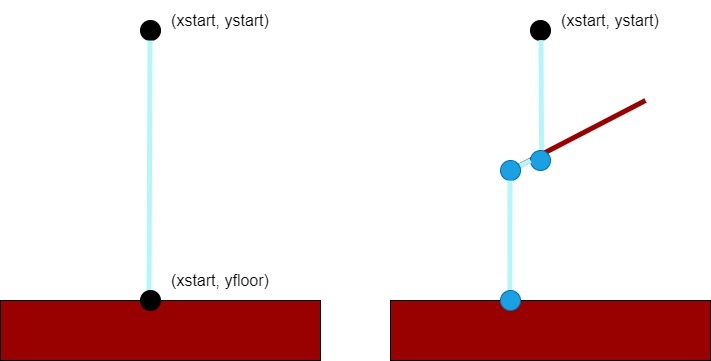
기존의 LineSegment 클래스에 추가된 멤버 함수로 선분의 기울기를 의미하는 멤버 변수 slope의 값을 반환하는 getter 함수이다.

input : 없음

output : 선분의 기울기 값 slope

(2) WaterSegment::WaterSegment(Point& start)

새로 추가된 WaterSegment 클래스의 생성자로 입력으로 물줄기를 표현하는 선분 WaterSegment 객체의 시작점(물줄기가 흐르기 시작하는 점)을 나타내는 객체 start를 참조자로 받는다. WaterSegment의 기반 클래스인 LineSegment 클래스의 생성자에 start 점의 x 좌표, start 점의 y 좌표, start 점의 x 좌표, 바닥의 y 좌표를 인자로 전달하여 호출해서 시작점 start와 start에서 바닥에 수직으로 내린 점을 끝점으로 잇는 선분(LineSegment)를 생성한다. 또한 다른 WaterSegment 객체를 가리키는 포인터 link를 NULL로 초기화한다.



기본적으로 WaterSegment는 왼쪽 그림처럼 시작점으로부터 바닥까지 물이 수직으로 떨어지는 것을 가정하고 생성된다. 따라서 시작점만 입력으로 받고, 끝점의 좌표는 시작점의 x 좌표와 바닥의 좌표를 이용하여 결정된다. 만약 오른쪽 그림처럼 물이 수직으로 떨어지는 중간에 선분에 의해 가로막힐 경우 물의 경로가 변경된다. 이러한 경우는 createWaterfallList 함수에서 해결한다.

input : 물줄기가 흐르기 시작하는 점 객체의 참조자 start

output : 없음

(3) void WaterSegment::setLP(Point& \_lp); void WaterSegment::setRP(Point& \_rp);

WaterSegment의 시작점 leftP와 끝 점 rightP를 각각 점 \_lp, 점 \_rp로 변경하는 함수이다. 각각 leftP, rightP의 setter 함수 setX, setY를 호출하여 점 \_lp, 점 \_rp의 좌표와 같게 변경한다. \_lp, \_rp의 x, y 좌표는 getter 함수 getX, getY를 호출하여 얻어 온다.

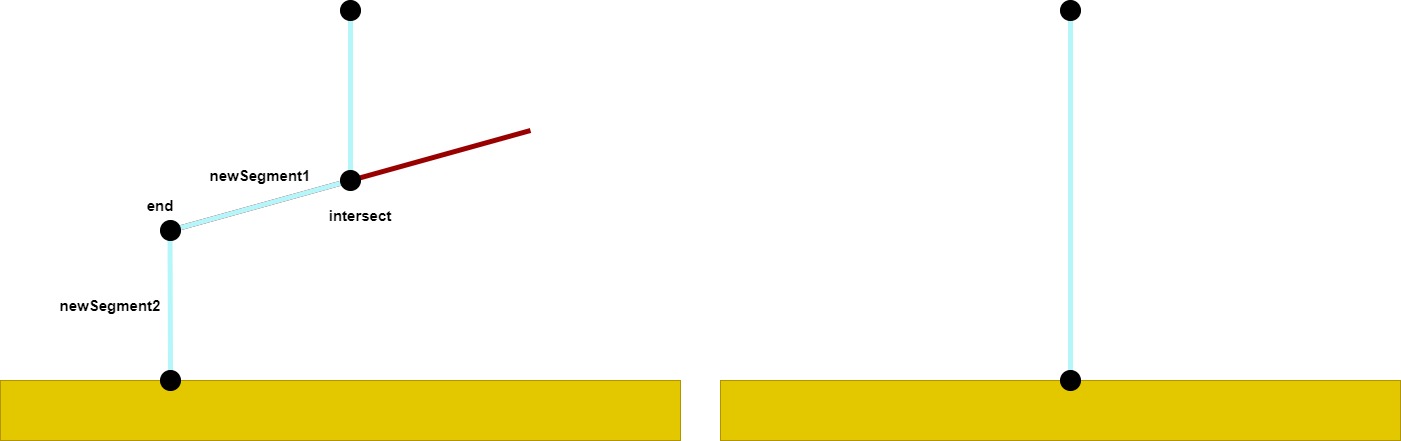
input : 각각 시작점, 끝점으로 할 점 객체 참조자 \_lp, \_rp

output : 없음

(4) void Waterfall::createWaterfallList()

Waterfall 객체에 저장된 물 구멍의 개수만큼 WaterSegment 연결 리스트가 존재한다. 각 연결 리스트의 첫 번째 노드는 각 물구멍에서 시작해서 바닥으로 수직으로 바로 떨어지는 WaterSegment 노드를 가지고 있다. 각각의 연결 리스트에 대하여 물 줄기와 만나는 선반이 있는지 모든 선반을 검사한다. 모든 선반을 검사하여 가장 가까이 만나는 선반을 찾는다. 가장 가까이 만나는 선반을 찾았으면 현재 물줄기의 끝점을 해당 선반과의 교점 intersect으로 변경한다. 변경 후, 교점을 시작점으로 하는 WaterSegment 노드 newSeg1을 생성한다. newSeg1의 끝 점을 선반의 양 끝점 중 더 아래의 있는 점 end로 변경한 뒤 연.결 리스트에 삽입한다.

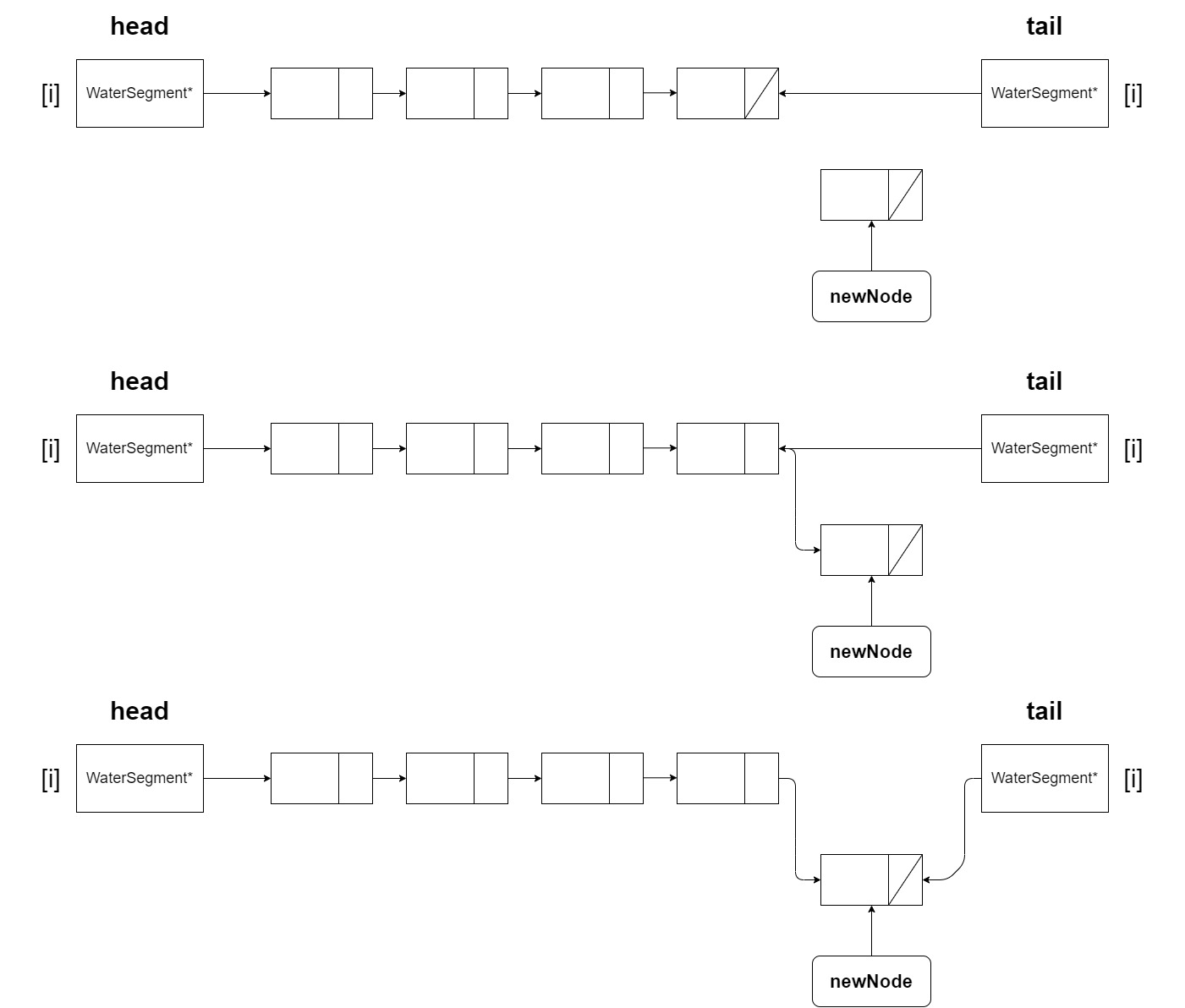
end를 시작점으로 하는 WaterSegment 노드 newSeg2 생성하고 newSeg2를 연결 리스트에 삽입에 삽입한다.



왼쪽 그림이 가로막는 선분이 있는 경우이다. 위의 설명처럼 2개의 WaterSegment가 더 생기는 것을 알 수 있다. 반대로 오른쪽 그림처럼 가로막는 선분이 있을 경우 처음 WaterSegment를 그대로 유지한다.

가로막는 선분이 존재해서 새로운 WaterSegment 노드의 삽입이 이루어졌을 경우, 마지막 노드의 시작점을 새로운 물구멍으로 하여 위의 과정을 반복한다.

연결 리스트의 삽입 과정은 다음 그림과 같다. 해당 연결 리스트의 마지막 노드를 가리키는 tail[i]를 이용하여 연결 리스트의 가장 뒤에 삽입한다.

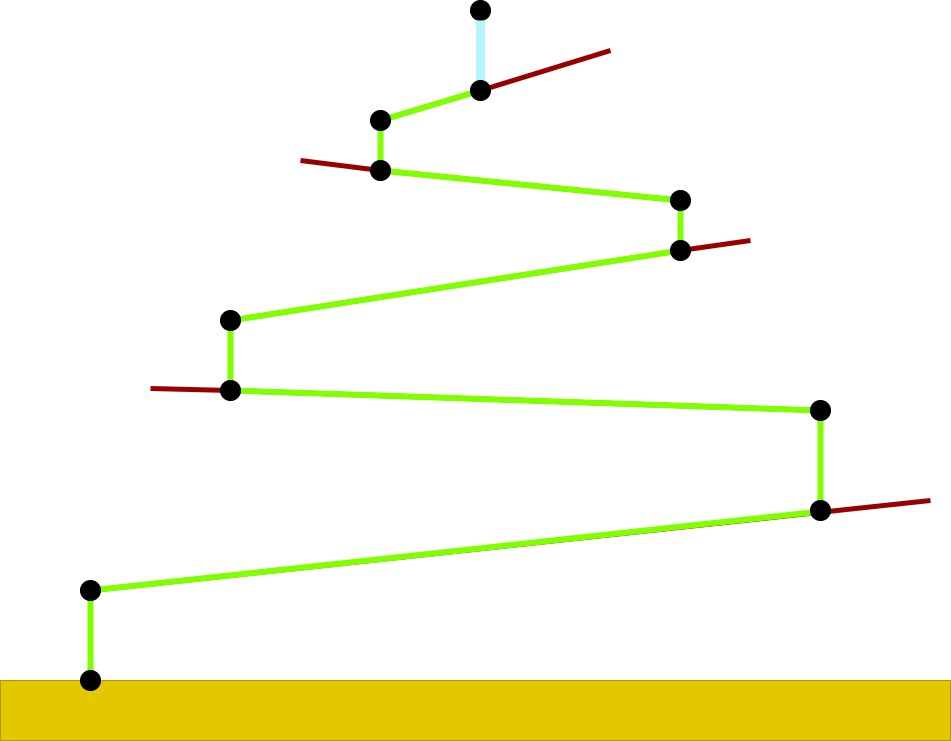


위의 설명을 토대로 작성한 의사 코드는 다음과 같다.

|  |
| --- |
| void Waterfall::createWaterfallList() {  if (연결 리스트가 존재하지 않을 경우) {  에러 메시지 출력;  return;  }  for (int i = 0; i < wh; i++) { // 각 물 구멍에 대하여  가로막는 선반의 개수 intersection;  do {  가장 먼저 가로막는 선반과의 교점의 y 좌표 minY;  가로막는 선반과의 교점의 y 좌표 yIntersect;  가장 먼저 가로막는 선반의 index(몇 번째 선반인지)    intersection = 0으로 초기화;  for (int k = 0; k < sh; k++) { // 모든 선반을 탐색함  if (물 구멍의 x 좌표가 선반의 양 끝점 x 좌표 사이 범위를 벗어날 경우) {  해당 선반이 물 줄기를 가로막을 수 없으므로 다음 선반으로 넘어감;  }  else {  기울기를 이용해 교점의 y 좌표 yIntersect 찾기;  if (선반과 물줄기의 교점이 물 구멍보다 위에 존재) {  해당 선반이 물 줄기를 가로막을 수 없으므로 다음 선반으로 넘어감;  }  else {  가장 먼저 물 줄기를 가로막는 선반을 찾음;  intersection++; // 가로막는 선반의 개수 증가  }  }  }  if (가로막는 선반이 있을 경우) {  float endX, endY;  선반의 양 끝점 중 더 아래에 있는 점의 x, y 좌표를 각각 endX, endY로 함;  선반과 물줄기의 교점 intersect 생성;  선반의 물줄기가 흐르는 방향 끝점 end 생성;  연결 리스트의 마지막 WaterSegment 노드의 끝점을 교점 intersect로 변경;  교점 intersect를 시작점으로 하는 WaterSegment 노드 newSeg1 생성;  newSeg1의 끝점을 end로 변경;  newSeg1을 연결 리스트에 삽입;  end를 시작점으로 하는 WaterSegment 노드 newSeg2 생성;  newSeg2를 연결 리스트에 삽입;  }  else 가로막는 선반이 없을 경우 바닥까지 떨어지는 물 줄기를 그대로 유지;  } while (가로막는 선반이 있을 경우);  }  } |

createWaterfallList 함수의 시간, 공간 복잡도는 다음과 같다.

입력받은 선반의 개수를 N, 물 구멍의 개수를 M이라 했을 때, M개의 각 물구멍에서 나오는 물 줄기에 대하여 물 줄기를 가로막는 선분을 찾기 위해 N개의 선분을 모두 탐색해야 한다. 이 과정에서의 시간 복잡도는 O(N)이다. 이 과정을 통해 가장 가까이 만나는 선반을 찾을 수 있다. 가로막는 선반의 끝점을 기준으로 다시 N개의 선반을 탐색하여 새로운 가로막는 선반을 찾는다. 가로막는 선반이 없을 때까지 반복한다. 따라서 물 구멍 하나에 대한 시간 복잡도는 O(가로막는 선반의 개수 \* N)이다. 가로막는 선반의 개수는 최대 N개이므로 물 구멍 하나에 대한 시간 복잡도는 O(N2)이다. 따라서 전체 시간 복잡도는 O(MN2)이다.



위의 그림에서 알 수 있듯이 물줄기가 선분 하나에 가로막힐 때마다 2개의 WaterSegment 노드가 새로 생성된다. 따라서 물 구멍 하나 당 새로 생성되는 WatetSegment의 개수는 2 \* 가로막는 선분의 개수이다. 가로막는 선분의 개수는 최대 N개이므로 WaterSegment 노드는 최대 2N개가 새로 생성된다. 따라서 물 구멍 하나당 연결 리스트 생성의 공간 복잡도는 O(2N) = O(N)이다. 모든 물구멍에 대하여 공간 복잡도는 O(MN)이다.

(5) void ofApp::initializeWaterLines()

전역적으로 선언된 Waterfall 포인터 wf로 함수 createWaterfallList 함수를 호출해 물 줄기를 표현하는 연결 리스트를 만든다.

input : 없음

output : 없음

3. 실습 환경

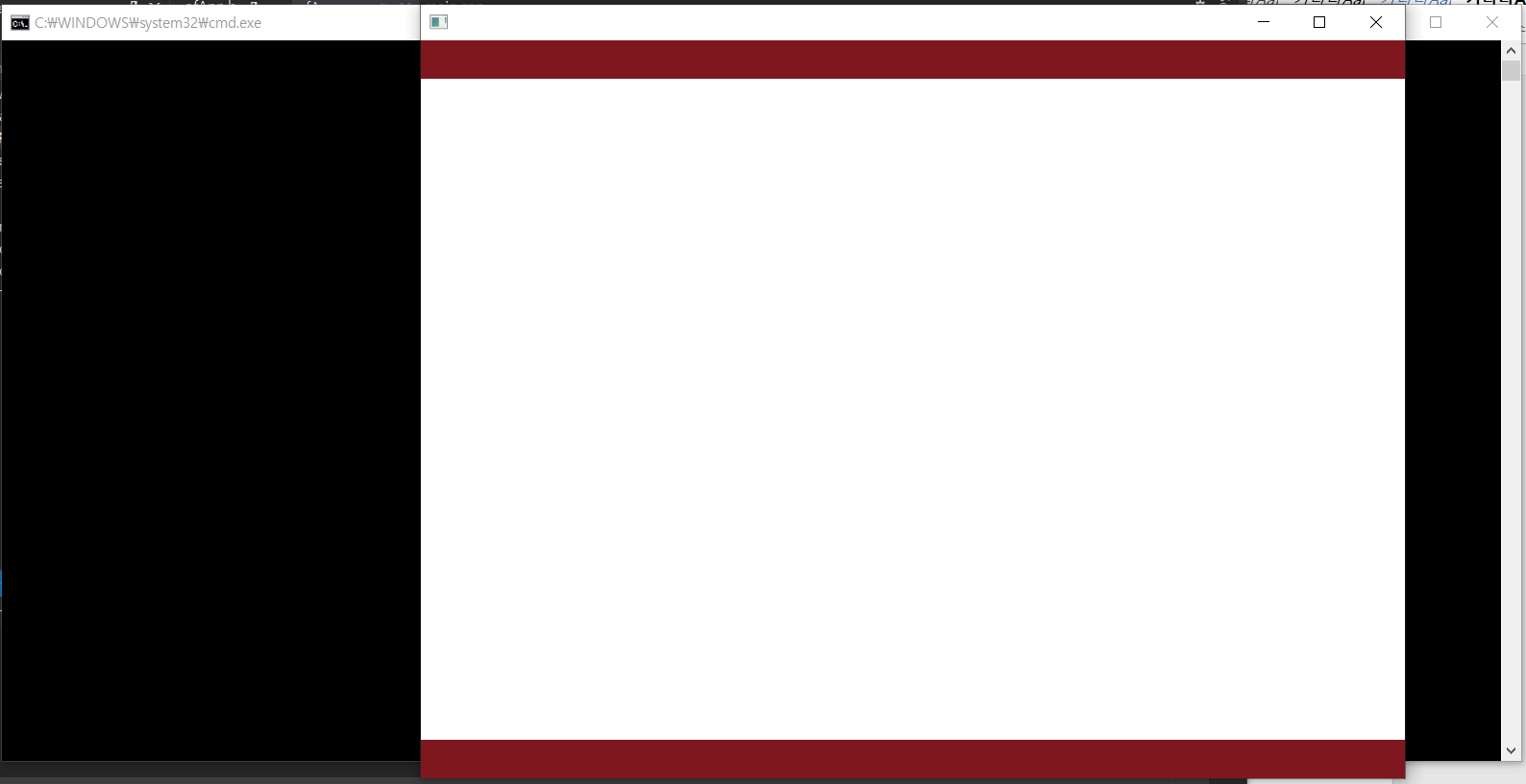
OS : Windows 10

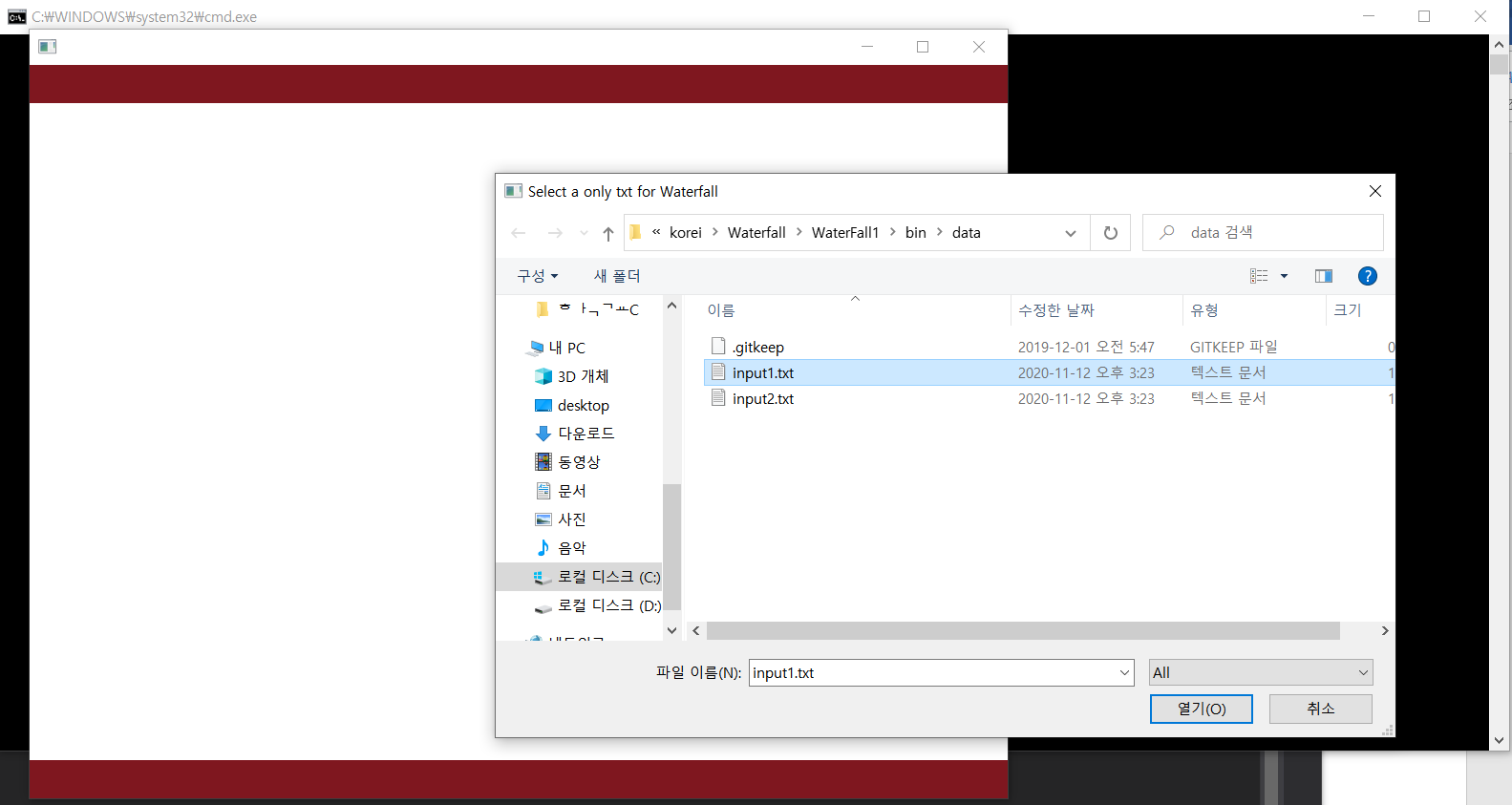
CPU : Intel® Core(TM) i3-5005U CPU @ 2.00GHz

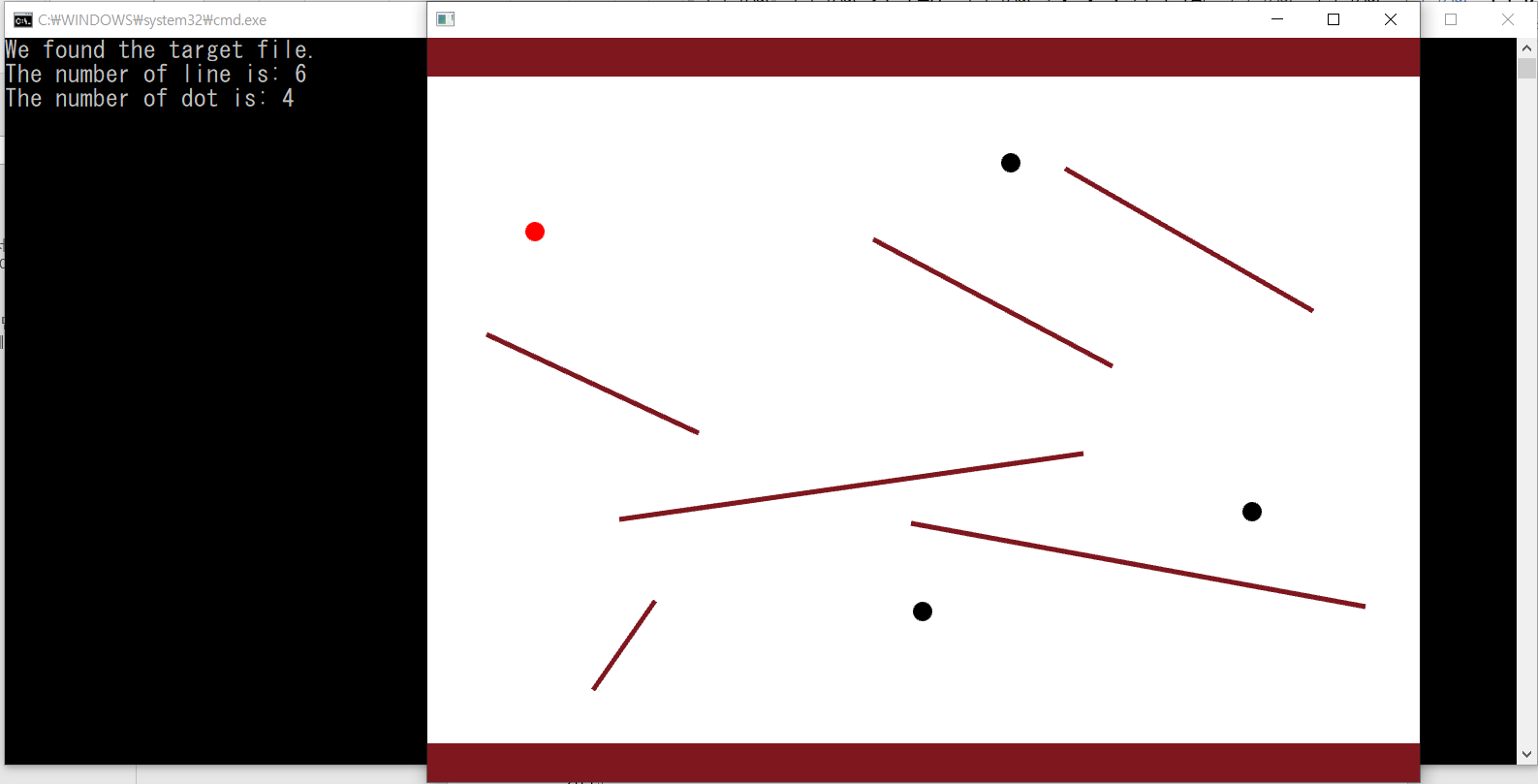
RAM : 8.00GB

IDE : Visual Studio 2019 community

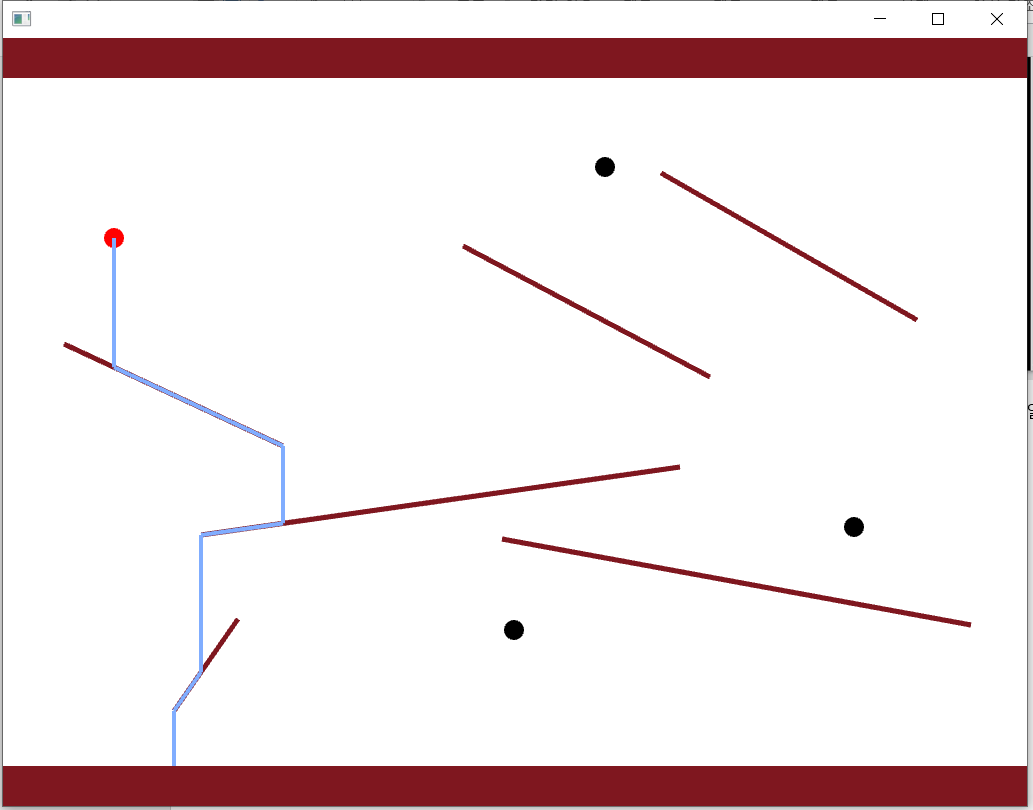
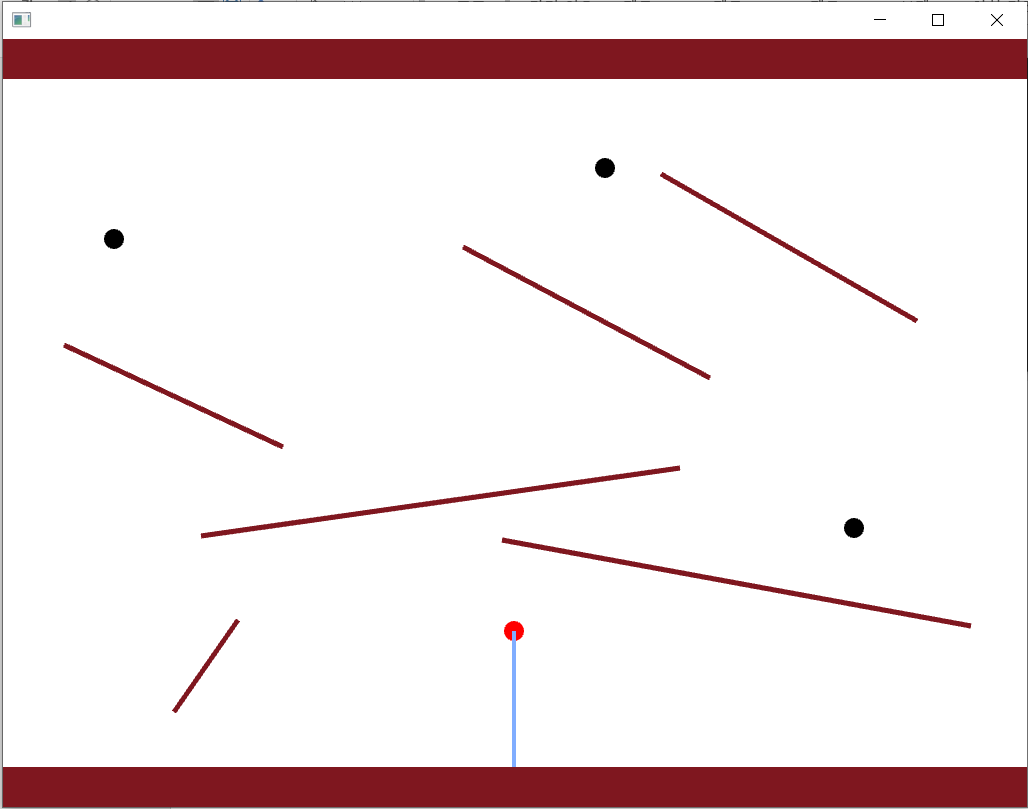
4. 실습 결과 및 분석

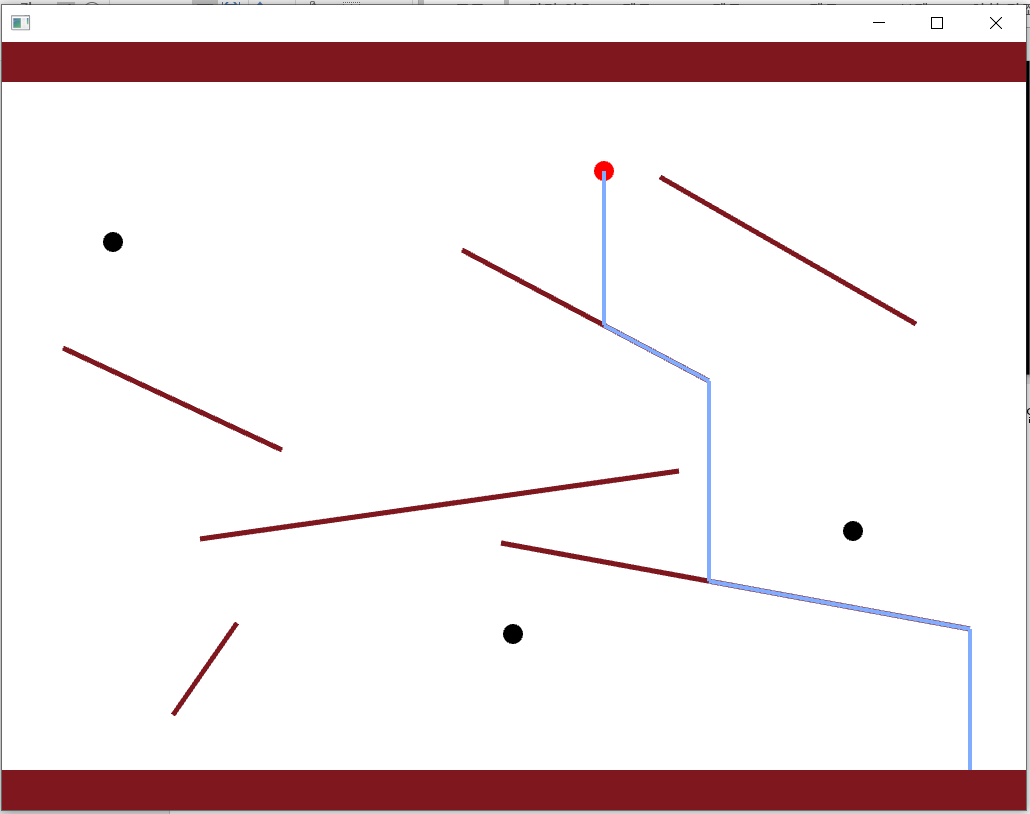
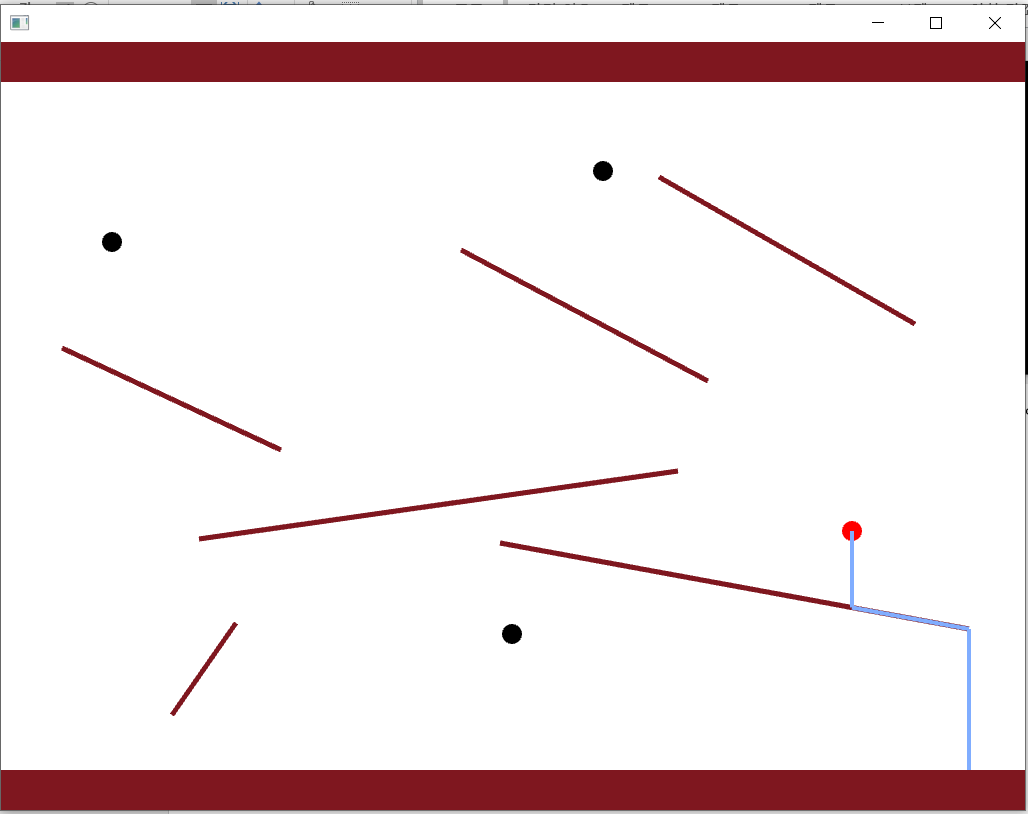






프로그램을 실행하고 파일을 선택 후 ‘d’ 키를 눌러 선반과 구멍을 그리는 것까지는 이전과 동일하다.

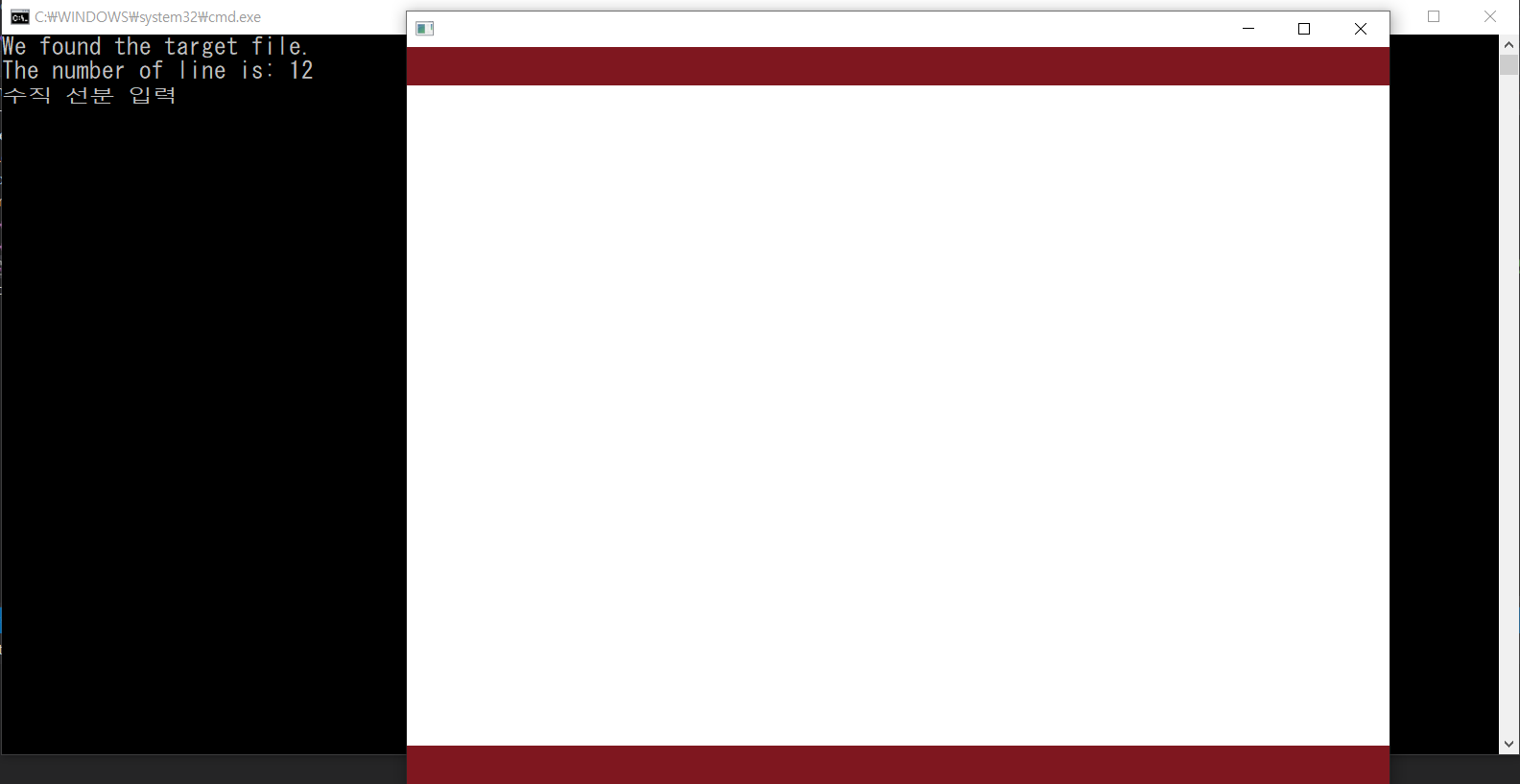
 

각 구멍에서 ‘s’ 키를 누르면 구멍에서 물이 흘러나오기 시작해 선반을 마주칠 때마다 물의 경로가 바뀌는 것을 알 수 있다. 두 번째 구멍에서 나오는 물 줄기는 가로막는 선반이 없어서 바로 바닥으로 수직하게 떨어지는 것을 알 수 있다.

|  |
| --- |
| 1012 321 1012 632 |

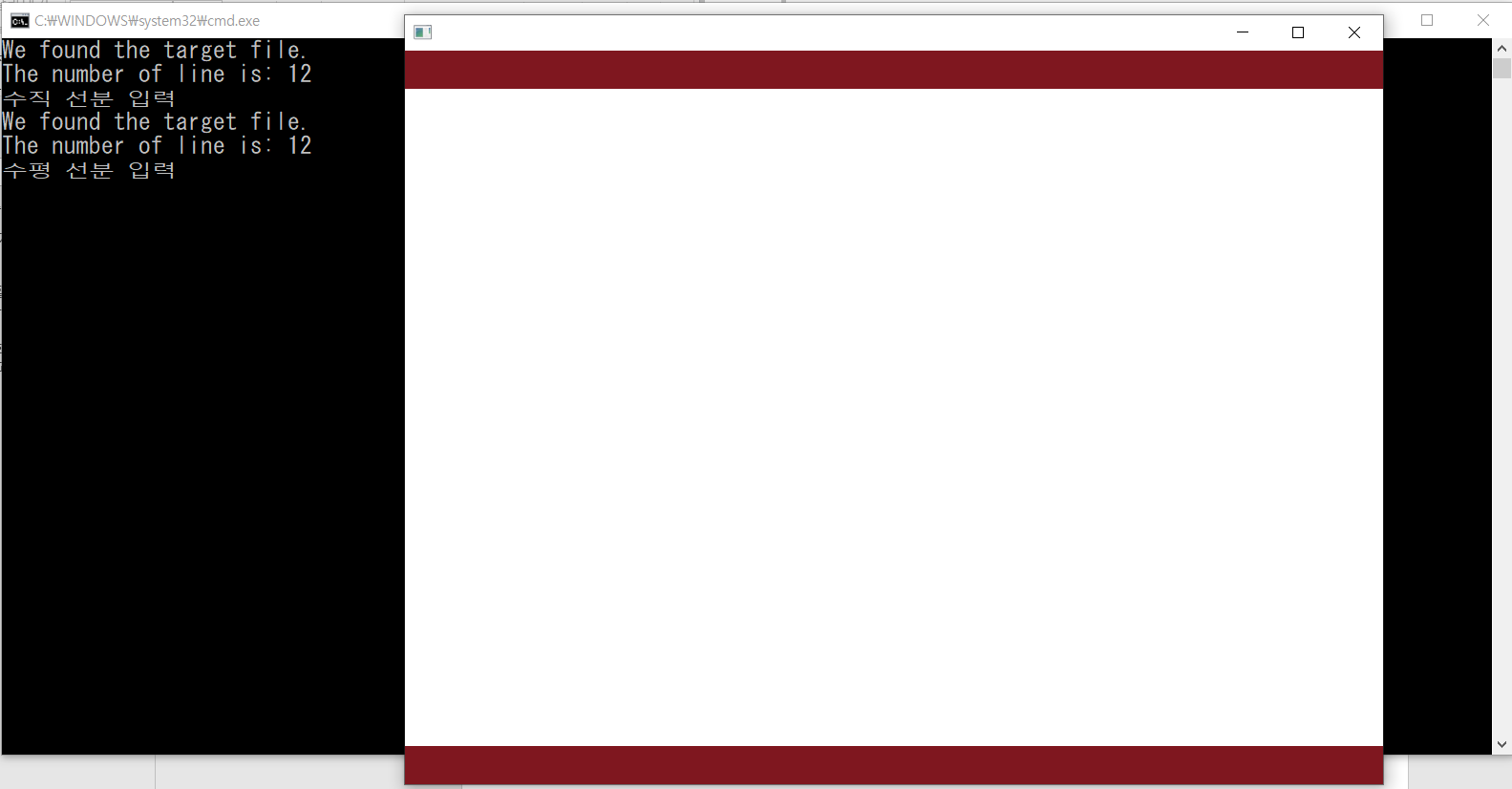
‘l’ 키를 누르고 양 끝점의 x 좌표가 모두 1012인 수직한 선분이 있는 텍스트 파일을 읽어본다.



수직 선분 입력이라는 오류 메시지가 콘솔에 출력되며 스크린에는 어떤 그림도 그려지지 않는다.

|  |
| --- |
| 673 345 782 345 |

‘l’ 키를 누르고 양 끝점의 y 좌표가 모두 345인 수평한 선분이 있는 텍스트 파일을 읽어본다.



마찬가지로 수평 선분 입력이라는 오류 메시지가 콘솔에 출력되며 스크린에는 어떤 그림도 그려지지 않는다.

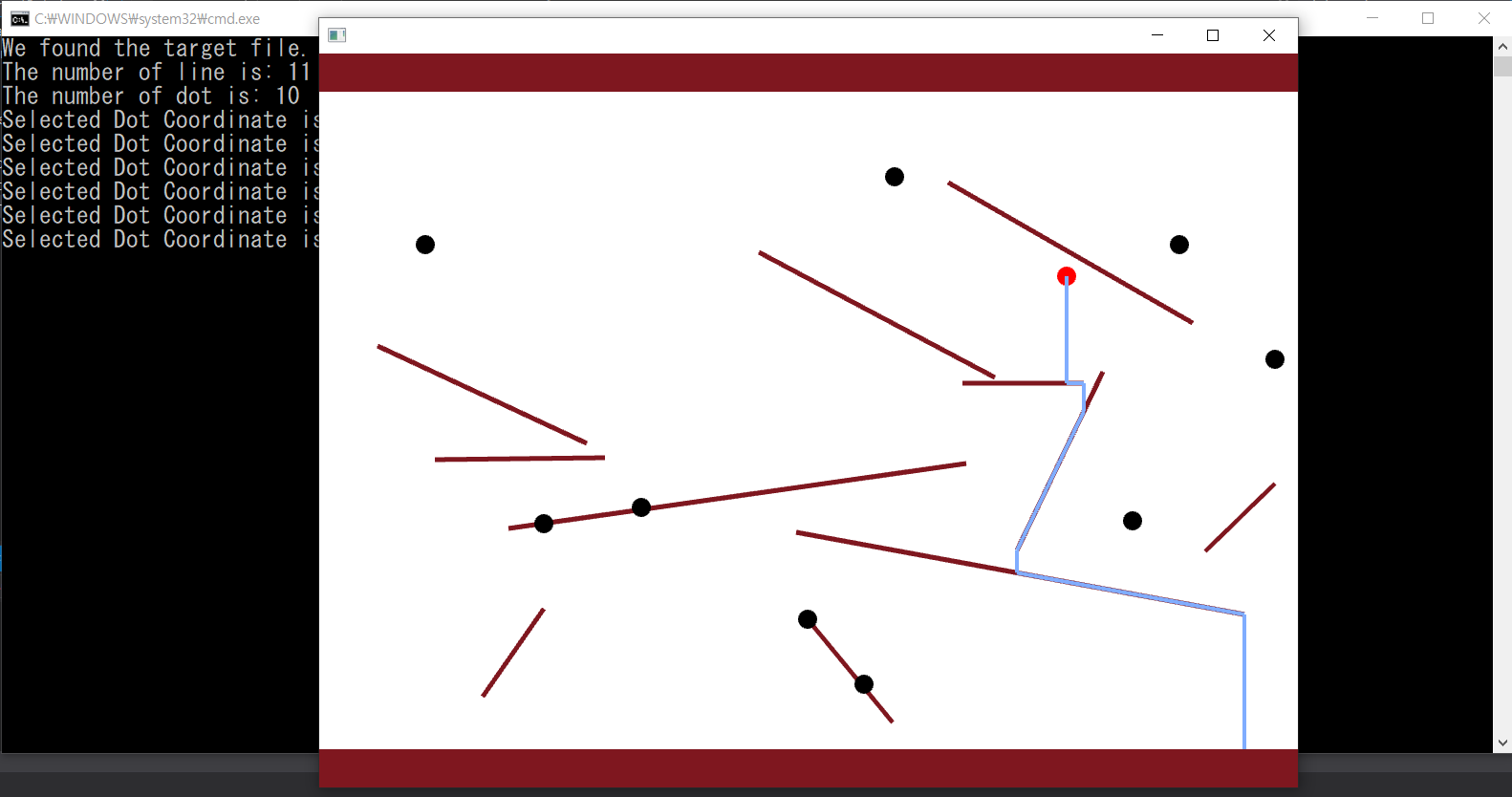
기타

수직 선반의 입력의 경우, 위와 같은 예외 처리를 해주지 않으면 입력 시 오류가 발생할 수 있다. 수직 선반의 경우 기울기 slope가 무한대가 되어 문제가 발생한다. 따라서 위와 같이 예외 처리로 수직 선반이 있을 경우 입력 받지 않도록 한다.

수평 선반의 입력의 경우 위와 같이 예외 처리를 해주지 않아도 오류는 발생하지는 않는다. 다음은 물줄기가 만나는 선반의 양 끝점을 비교하는 의사 코드이다.

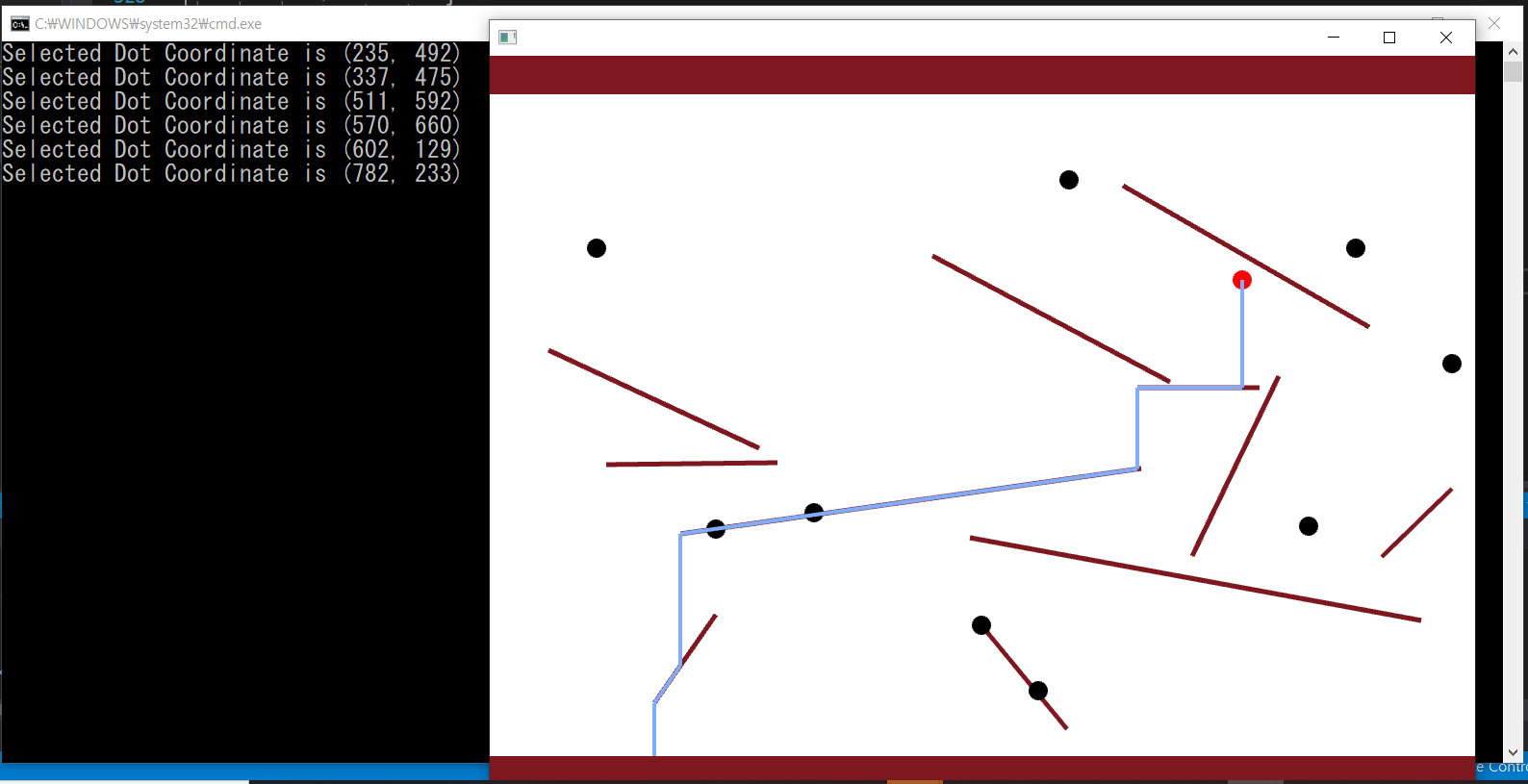
|  |
| --- |
| if (선반의 왼쪽 끝점의 y 좌표 > 선반의 오른쪽 끝점의 y 좌표) {  점 end의 x 좌표 = 선반의 왼쪽 끝 점의 x 좌표;  점 end의 y 좌표 = 선반의 왼쪽 끝 점의 y 좌표;  } else {  점 end의 x 좌표 = 선반의 오른쪽 끝 점의 x 좌표;  점 end의 y 좌표 = 선반의 오른쪽 끝 점의 y 좌표;  } |

위의 의사 코드를 그대로 따르면 양 끝점의 y 좌표가 같을 경우, 오른쪽 점이 선택된다. 따라서 오류는 발생하지 않는다.



물줄기가 수평 선분을 만나서 오른쪽 점으로 흐르는 것을 알 수 있다.

하지만 특별한 기준이 없으면 구현 방법에 따라 왼쪽 점이 선택될 수도 있다. 위의 의사 코드에서 부등호 >를 >= 바꾸면 수평 선분의 경우 왼쪽 점이 선택된다.



물 줄기가 수평 선분을 만나서 왼쪽 점으로 흐르는 것을 알 수 있다.

수평 선분의 경우 오류는 발생하지 않아도, 특별한 기준이 없으면 구현 방법에 따라 결과가 상이하다. 따라서 수평 선분이 주어질 경우도 예외 처리를 해준다.

만약 수평 선분을 만났을 경우 만난 점을 기준으로 더 가까운 점으로 흐르도록 구현하는 방법도 생각해볼 수 있을 것 같다. (위의 그림의 경우 오른쪽 점으로 흐른다.) 하지만 이 경우도 물줄기가 선분의 정 중앙에 떨어질 경우 양 쪽 점과의 거리가 같으므로 어떤 점을 선택해야 하는 지가 명확하지 않다.