7주차 결과보고서

전공 : 경영학과 학년 : 4학년 학번 : 20190808 이름 : 방지혁

1. water_line.h와 water_line.cpp 설명

LineSegment 구조체

```
struct LineSegment {
        int x1, x2, y1, y2;
        double slope, x_coef, y_coef, constant;
};
```

물이 부딪히는 선분을 저장하는 구조체입니다. x1, x2, y1, y2는 선분의 양 끝점 좌표입니다. Slope는 기울기를 나타내는 변수입니다. 그 외 변수들은 실제로 사용되지는 않았습니다.

Dot 구조체

```
struct Dot{
        int x1, y1;
};
```

물의 시작점을 저장하는 구조체입니다.

다음 장에 이어서

WaterLine 클래스

```
class WaterLine{

public:
	WaterLine(int num_of_line);
	~WaterLine();

	void reset();
	void update();
	void draw();
	void calculate_path(LineSegment *lineseg, int num_of_line);

	Dot* path = NULL;

	float uniqueColor_r, uniqueColor_g, uniqueColor_b;
	float scale;
	int hexcolor;

	int path_idx;

	/* 그리기가 완료되었는지 체크하는 flag */
	int draw_complete;
	/* 경로계산이 완료되었는지 체크하는 flag, draw() 함수에서 이게 1이여야 시작 */
	int calc_complete;

	ofPoint start_dot;
	float dot_radius;

};
```

멤버 변수들에 대해 설명하자면 Dot* path는 물의 경로를 나타낼 동적배열입니다. uniqueColor_r, uniqueColor_g, uniqueColor_b는 물줄기의 색깔을 나타낼 변수입니다. 이는 파란색 계열로 랜덤하게 정해질 예정입니다. path_idx 변수는 현재 path 배열에서 몇 번째 점까지 저장했는지 사용되는 변수입니다. draw_complete는 그리기가 완료되었는지 확인하는 flag, calc_complete는 경로계산이 완료되었는지 체크하는 flag로 draw()함수에서 이게 1이여야 그림을 그립니다.

start_dot은 시작점, dot_radius는 점 반지름을 저장할 변수입니다. 이제 water_line.cpp에 구현되어 있는 멤버 함수들에 대해 설명하겠습니다.

WaterLine(int num_of_line)은 물줄기를 처음 만들 때 실행되는 생성자이며 변수들을 초기화하며 선분 하나와 만났을 때 최대 2개의 점이 필요하고 여유분으로 +4개의 점을 경로배열을 위해 더 할당합니다. **~WaterLine()**은 물줄기가 사라질 때 즉, delete waterLine;을 호출하면 실행되는 소멸자로 메모리를 해제해줍니다.

```
WaterLine::WaterLine(int num_of_line) {
	draw_complete = 0; // 그리기가 완료되었는지 체크하는 flag
	calc_complete = 0; // 경로계산이 완료되었는지 체크하는 flas
	/* 항상 파란색 계열에서 랜덤으로 선택 */
	uniqueColor_r = ofRandom(0,255);
	uniqueColor_g = ofRandom(0,255);
	uniqueColor_b = ofRandom(185, 255);
	path_idx = 0;
	if(!path)
	 path = (Dot*)malloc(sizeof(Dot) * (2 * num_of_line + 4));
}
WaterLine::~WaterLine() {
	free(path);
}
```

void WaterLine::draw()는 물줄기를 그리는 함수입니다.

```
void WaterLine::draw() {
	if(calc_complete){
	// 경로 계산이 끝났음!
		ofSetLineWidth(5);
		ofSetColor(uniqueColor_r, uniqueColor_g, uniqueColor_b);
		for (int i=0 ; i < path_idx - 1 ; i++) {
			uniqueColor_r = ofRandom(0,100);
			uniqueColor_g = ofRandom(0,100);
			uniqueColor_b = ofRandom(185, 255);
			ofSetColor(uniqueColor_r, uniqueColor_g, uniqueColor_b);
			ofDrawLine(path[i].x1-1, path[i].y1-1, path[i+1].x1+1, path[i+1].y1+1);
		}
	}
	draw_complete = 1;
}
```

해당 함수에 대해 설명하자면 경로 계산이 끝나야 그릴 수 있기 때문에 calc_complete이 1인지 확인합니다. ofSetLineWidth(5)로 물줄기 선의 굵기를 5로 설정하고 ofSetColor로 RGB색상을 설정합니다. 그리고 반복문을 돌리는데 주의 깊게 볼 것은 path_idx -1만큼 반복문을 돌리는데 여기서 path_idx는 점의 개수를 의미하고 예를 들어 점이 4개가 있다면 필요한 선은 3개뿐이기 때문입니다. 이후 매 선마다 색을 파란색 계열에서 살짝씩 바꾸고 i번째 점에서 i+1번째 점까지 선을 그려줍니다.

void WaterLine::reset() 함수는 물줄기를 처음 상태로 돌려줍니다.

```
void WaterLine::reset() {
	//the unique val allows us to set color for each water line.
	uniqueColor_r = ofRandom(0,100);
	uniqueColor_g = ofRandom(0,100);
	uniqueColor_b = ofRandom(185, 255);

	calc_complete = 0;
	draw_complete = 0;

	path_idx = 0;
}
```

마지막으로는 **void WaterLine::calculate_path(LineSegment *lineseg, int num_of_line)** 함수입니다.

```
void WaterLine::calculate_path(LineSegment *lineseg, int num_of_line) {
    /* 시작점 저장 */
    path[path_idx].x1 = start_dot.x;
    path[path_idx].y1 = start_dot.y;
    path_idx++;

    for( ; start_dot.y <= ofGetHeight()-50 ; start_dot.y++){
        for( int i=0 ; i<num_of_line ; i++){
            // 1) Ignore line that located higher than water particle.
            if( start_dot.y >= lineseg[i].y1 && start_dot.y >= lineseg[i].y2 ) continue;
            // 2) Check whether the dot point exists between the each end of line segment.
            // If pos.x exists btw. y1 and y2 then, eventually hit the line segment.q
            if( lineseg[i].x1 < lineseg[i].x2){
                if( start_dot.x <= lineseg[i].x1 || start_dot.x >= lineseg[i].x2)
                    continue;
            }
            else if ( lineseg[i].x1 > lineseg[i].x2){
                if( start_dot.x <= lineseg[i].x2 || start_dot.x >= lineseg[i].x1)
                    continue;
            }
            // Slope 계산
            double temp_slope = (double)(start_dot.y - lineseg[i].y1)/(start_dot.x - lineseg[i].x1);
            // Dot exists in line
            if( abs(temp_slope - lineseg[i].slope) <= EPSILON){
                path[path_idx].x1 = start_dot.x;
                path[path_idx].y1 = start_dot.y+2;
                path_idx++;
                // Debug output
                //cout << "[" << i << "]" << "x: " << start_dot.x << " " << "y: " << start_dot.y << endl;
                if( lineseg[i].slope < 0){
                    path[path_idx-1].x1++;
                    start_dot.x = lineseg[i].x1;
                    start_dot.y = lineseg[i].y1-2;
                }
                else{
                    path[path_idx-1].x1--;
                    start_dot.x = lineseg[i].x2;
                    start_dot.y = lineseg[i].y2-2;
                }
                path[path_idx].x1 = start_dot.x;
                path[path_idx].y1 = start_dot.y;
                path_idx++;
                // Debug output
                //cout << "[" << i << "]" << "x: " << start_dot.x << " " << "y: " << start_dot.y << endl;
            }
        }
    }
    // Last path
    path[path_idx].x1 = start_dot.x;
    path[path_idx].y1 = start_dot.y;
    path_idx++;

    calc_complete = 1;
}
```

path[path_idx].x1 = start_dot.x; path[path_idx].y1 = start_dot.y; path_idx++; 우선 배열에 시작점을 저장하고, 다음 점을 저장해야하기 때문에 path_idx를 1 증가시켜줍니다. 그 다음 반복문을 바닥까지 물을 떨어뜨리는데, 반복문을 돌 때마다 start_dot.y를 1씩 증가시킵니다. 이는 아래로 한 픽셀씩 이동한다는 것으로 화면 높이 -50, 즉 바닥 직전까지 반복합니다. 그리고 다음 루프는 현재 물방울 위치에서 모든 선분을 하나하나 검사하여 충돌 여부를 체크하는 것입니다. if(start_dot.y >= lineseg[i].y1 && start_dot.y >= lineseg[i].y2) continue; 해당 줄은 물방울 위에 있는 선분은 무시하는 것입니다.

다음 장에 이어서

if(lineseg[i].x1 < lineseg[i].x2){ if(start_dot.x <= lineseg[i].x1 || start_dot.x >= lineseg[i].x2) continue;} else if (lineseg[i].x1 > lineseg[i].x2){ if(start_dot.x <= lineseg[i].x2 || start_dot.x >= lineseg[i].x1) continue;} 해당 코드는 물방울의 x 좌표가 선분의 범위 밖이면 충돌하지 않기 때문에 무시하는 것입니다.

이후 선분의 기울기인 lineseg[i].slope와 현재 점과 선분의 끝점을 이었을 때 기울기인 temp_slope의 차이를 계산합니다. 이것이 epsilon 이하일때 즉, 기울기가 거의 같을 때 물방울과 선분이 만난다는 것을 의미합니다. 해당 경우 path[path_idx].x1 = start_dot.x; path[path_idx].y1 = start_dot.y+2; path_idx++;를 수행하여 충돌지점을 배열에 저장합니다.

이후 물이 튀는 방향을 계산하는 코드입니다. 선분의 기울기가 음수이면, 우리가 직관적으로 생각하는 바와 달리 y값이 클수록 아래쪽입니다. 음의 기울기를 가진다는 것은 y2보다 y1이 더 크고 x2가 화면 상 위쪽, x1이 아래쪽에 있다는 것입니다. 그래서 y값이 더 큰 x1으로 가야하기 때문에 x1과 y1 좌표를 다음 이동 좌표로 넣어줍니다. 반대의 경우는 x2와 y2 좌표를 다음 이동 좌표로 넣어줍니다. 그리고 path_idx를 두 경우 다 증가시켜줍니다. 이후 루프가 끝나면 마지막 점의 위치를 저장하고 calc_complete을 1로 설정함으로써 계산이 끝났다는 것을 알립니다.

다음 장에 이어서

2. ofApp.h와 ofApp.cpp 설명

```
class ofApp : public ofBaseApp{
  public:
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y);
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void mouseEntered(int x, int y);
    void mouseExited(int x, int y);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    /* WaterFall-related member variables Regions */
    // flag variables
    int draw_flag;
    int load_flag;
    int simulation_flag;
    int selected_dot_index;
    LineSegment* lineseg = nullptr;
    Dot* dot = nullptr;
    WaterLine* waterline = nullptr;
    // Line segment and dot related variables
    int num_of_line, num_of_dot;
    float dot_diameter;

    /* WaterFall-related member functions */
    void processOpenFileSelection(ofFileDialogResult openFileResult);
    void initializeWaterLines(); // 2nd week portion.
};
```

ofApp.h에서 멤버 함수들을 선언하고, 멤버 변수들을 선언하는데 draw_flag는 화면에 그릴 지 여부를 나타내는 flag, load_flag는 input 파일을 성공적으로 로드했는지 나타내는 flag, simulation_flag는 물줄기 시뮬레이션이 실행되었는지 나타내는 flag, selected_dot_index는 현재 선택된 dot의 index를 의미합니다. 그 외의 변수들은 선부 개수, 점수 개수, 점의 지름을 저장하는 변수입니다.

이제 ofApp.ccp의 실제 구현된 코드에 대해 분석하겠습니다.

void ofApp::setup()

```
void ofApp::setup(){
  ofSetFrameRate(15); // Limit the speed of our program to 15 frames per second

  // We still want to draw on a black background, so we need to draw
  // the background before we do anything with the brush
  ofBackground(255,255,255);
  ofSetLineWidth(4);

  draw_flag = 0;
  load_flag = 0;
  simulation_flag = 0;
  dot_diameter = 20.0f;
  selected_dot_index = 0;
}
```

프레임 속도를 1초에 15번 갱신으로 설정하고 배경색은 흰색으로 설정한다. 선 굵기는 4픽셀로 설정하고 모든 flag를 0으로 초기화해줍니다. 점 크기(지름)은 20, 선택한 dot의 index는 0부터 시작되도록 설정해줍니다.

void ofApp::draw()

```
void ofApp::draw(){
  ofSetColor(127,23,31);  // Set the drawing color to brown

  // Draw shapes for ceiling and floor
  ofDrawRectangle(0, 0, 1024, 40); // Top left corner at (50, 50), 100 wide x 100
high
  ofDrawRectangle(0, 728, 1024, 40); // Top left corner at (50, 50), 100 wide x 100
high
  ofSetLineWidth(5);


  ofSetLineWidth(5);
  if(draw_flag){

    /* COMSIL1-TODO 3 : Draw the line segment and dot in which water starts to flow
in the screen.
     Note that after drawing line segment and dot, you have to make selected water
start dot colored in red.
     */
    ofSetColor(0, 0, 0); // 검은색으로 설정
    for(int i = 0; i < num_of_line; i++) {
      ofDrawLine(lineseg[i].x1, lineseg[i].y1, lineseg[i].x2, lineseg[i].y2);
    }

    for(int i = 0; i < num_of_dot; i++){
      if(i == selected_dot_index){
```

```
      ofSetColor(255, 0, 0); // 선택된 점은 빨간색!
    }
    else {
      ofSetColor(0, 0, 0); // 나머지는 검정색
    }
    ofDrawCircle(dot[i].x1, dot[i].y1, dot_diameter/2);
  }

  // 2nd week portion.
  ofSetLineWidth(2);

  if(simulation_flag && waterline != nullptr) {
    waterline->draw();
  }
 }
}
```

그리는 함수에서 가장 처음으로 그리는 것은 우선 천장과 바닥입니다. 그리고, draw_flag가 1이라면 즉 사용자가 'd'키를 입력했을 때에만 동작하도록 합니다. 이후 색을 검정색으로 설정하고, 모든 선분을 반복문을 돌며 (x1, y1)에서 (x2, y2)까지 선을 그립니다. 그리고 모든 점들에 대해 선택된 점은 빨간색, 나머지는 검정색으로 반지름은 20으로 해서 원을 그립니다. 그리고 마지막은 물줄기를 그리는 코드로 simulation_flag와 Waterline이 존재할 때 waterline의 draw()함수를 호출하여 경로를 그립니다.

keyPressed() 함수

```
void ofApp::keyPressed(int key) {
 if (key == 'v') {
  // HACK: only needed on windows, when using ofSetAutoBackground(false)
  glReadBuffer(GL_FRONT);
  ofSaveScreen("savedScreenshot_" + ofGetTimestampString() + ".png");
 }
 if (key == 'q') {
  // Reset flags
  draw_flag = 0;

  // Free the dynamically allocated memory exits.
  if (lineseg != nullptr) {
    free(lineseg);
    lineseg = nullptr;
  }
  if (dot != nullptr) {
    free(dot);
    dot = nullptr;
  }
  if (waterline != nullptr) {
```

```
    delete waterline;
    waterline = nullptr;
  }
  cout << "Dynamically allocated memory has been freed." << endl;
  _Exit(0);
 }
 if (key == 'd') {
  if (!load_flag) return;

  draw_flag = 1;
 }
 if (key == 's') {
  // 2nd week portion.
  if (!draw_flag) return;
  simulation_flag = 1;
  initializeWaterLines();
 }
 if (key == 'e') {
  // 2nd week portion.
  simulation_flag = 0;
  if (waterline != nullptr) {
    delete waterline;
    waterline = nullptr;
  }
 }
}
```

V키가 입력되었을 때는 스크린샷을 찍고, q키가 입력되었을 때는 기존의 flag들을 리셋해주고 메모리를 해제해주고 포인터를 nullptr로 설정해줍니다. D키가 입력되었을 경우에는 파일이 이전에 로드되어 있어야 Drawflag를 1로 설정해주며, draw()함수에서 선분과 점이 그려집니다. S키가 입력되었을 경우 draw_flag가 1인 상태여야 simultation_flag를 1로 설정하고 initializeWaterLines()를 호출합니다. E키의 경우 simulation_flag를 0으로 설정하고 waterline 객체를 삭제하고 포인터를 null로 설정해줍니다.

다음 장에 이어서

processOpenFileSelection()함수

```
void ofApp::processOpenFileSelection(ofFileDialogResult openFileResult) {
  // Path to the comma delimited file
  // string fileName = "input.txt";

  string fileName = openFileResult.getName();
  ofFile file(fileName);

  if (!file.exists())
    cout << "Target file does not exists." << endl;
  else
    cout << "We found the target file." << endl;

  ofBuffer buffer(file);

  /* This variable is for indicating which type of input is being received.
   IF input_type == 0, then work of getting line input is in progress.
   IF input_type == 1, then work of getting dot input is in progress.
   */
  int input_type = 0;
  int line_idx = 0;
  int dot_idx = 0;

  /* COMSIL1-TODO 1 : Below code is for getting the number of line and dot,
   getting coordinates. You must maintain those information. But, currently
   below code is not complete. Also, note that all of coordinate should not be
   out of screen size. However, all of coordinate do not always turn out to be
   the case. So, You have to develop some error handling code that can detect
   whether coordinate is out of screen size.
   */

  // Read file line by line
  for (ofBuffer::Line it = buffer.getLines().begin(),
                  end = buffer.getLines().end();
       it != end; ++it) {
    string line = *it;  // 현재 줄의 내용을 line 변수에 저장

    // 한 줄을 공백 기준으로 분리
    vector<string> words = ofSplitString(line, " ");

    if (words.size() == 1) {
      // 숫자 한 개만 있는 줄이라면
      if (input_type == 0) {
        // 선분 입력 모드
        num_of_line = atoi(words[0].c_str());  // 선분의 개수
        cout << "The number of line is: " << num_of_line << endl;

        if (!lineseg)
          lineseg = (LineSegment*)malloc(sizeof(LineSegment) * num_of_line);
```

```
    } else {
      // 점 입력 모드
      num_of_dot = atoi(words[0].c_str());  // 점의 개수
      cout << "The number of dot is: " << num_of_dot << endl;

      if (!dot) dot = (Dot*)malloc(sizeof(Dot) * num_of_dot);
    }
  } else if (words.size() >= 2) {
    int x1, y1, x2, y2;
    if (input_type == 0) {
      // 선분 정보
      x1 = atoi(words[0].c_str());
      y1 = atoi(words[1].c_str());
      x2 = atoi(words[2].c_str());
      y2 = atoi(words[3].c_str());

      if (x1 < 0 || x1 > ofGetWidth() || x2 < 0 || x2 > ofGetWidth() ||
          y1 < 0 || y1 > ofGetHeight() || y2 < 0 || y2 > ofGetHeight()) {
        cout << "Out-of-range!" << endl;
        return;
      }

      lineseg[line_idx].x1 = x1;
      lineseg[line_idx].y1 = y1;
      lineseg[line_idx].x2 = x2;
      lineseg[line_idx].y2 = y2;

      if (x2 - x1 != 0) {
        lineseg[line_idx].slope = (double)(y2 - y1) / (x2 - x1);
      } else {
        lineseg[line_idx].slope = 0;
      }
      line_idx++;

      if (line_idx == num_of_line) {
        input_type = 1;
      }
    } else {
      // 점 정보
      x1 = atoi(words[0].c_str());
      y1 = atoi(words[1].c_str());

      if (x1 < 0 || x1 > ofGetWidth() || y1 < 0 || y1 > ofGetHeight()) {
        cout << "Out-of-range!" << endl;
        return;
      }

      // 점 정보 저장
      dot[dot_idx].x1 = x1;
      dot[dot_idx].y1 = y1;
```

```
      dot_idx++;
    }
  } // End of else if.
 } // End of for-loop (Read file line by line).
 // initializeWaterLines();
}
```

파일을 열고 공백을 기준으로 단어를 분리합니다. 선분인지 점인지에 따라 하는 동작이 다르며 선분이라면 좌표를 저장하고, 기울기를 계산하고 인덱스도 증가시키고 마지막 선분이라면 input_type을 1 즉 점 입력 모드로 변경시킵니다. 점의 경우 좌표를 읽고 범위를 체크하고 저장하고 마찬가지로 인덱스를 증가시킵니다.

InitializeWaterLines() 함수

```
void ofApp::initializeWaterLines() {
 if (waterline != nullptr) {
  delete waterline;
  waterline = nullptr;
 }

 waterline = new WaterLine(num_of_line);

 waterline->start_dot.x = dot[selected_dot_index].x1;
 waterline->start_dot.y = dot[selected_dot_index].y1;

 waterline->calculate_path(lineseg, num_of_line);
}
```

우선 기존 물줄기가 있다면 삭제하고 새 waterline 객체를 생성하고 선택도니 점을 시작된 점으로 설정해서 경로계산을 실행합니다.

정리하자면 setup은 프로그램을 초기화 시키고, draw는 매 프레임마다 화면을 그립니다. keyPressed(), keyReleased()함수 같은 경우 키보드 입력을 처리하고 processOpenFileSelection()은 선분과 점의 정보를 읽어 메모리에 저장하고 마지막으로 initializeWaterLines()는 물줄기 객체를 생성하고 경로를 계산합니다.