컴퓨터 공학 설계 및 실험 I 기말 프로젝트 보고서

2021.6.21 (월)

20181202 김수미

1. 프로젝트 환경

- macOS Catalina : Windows Parallels Desktop
- Visual Studio 2017 : OpenFramework

2. 프로젝트 목표

수업시간에 배운 다양한 자료구조와 알고리즘, Openframework 사용법을 기반으로 새로운 프로그램을 만드는 것을 주된 목표로 잡았다. 테트리스 수업 직후 waterfall 수업을 들었는데, 테트리스 블럭과 waterfall 의 물줄기를 보자마자 파이프 게임이 떠올랐다. 수업에서 파이프 게임 관련 내용은 다루지 않았기 때문에, 학기말 프로젝트를 기회로 파이프 게임을 직접 구현해보기로 하였다.

사람들이 알고있는 일반적인 파이프 게임이라 하면, 물줄기가 특정 위치에서 시작되고 게임 플레이어는 화면에 무작위로 배치된 여러 파이프를 회전시켜 해당 물줄기가 목적지에 도착할 수 있도록 경로를 만드는 게임이라고 할 수 있다.

하지만 해당 프로젝트의 파이프 게임은 이미 배치되어있는 파이프를 회전시키는게 아니라, 매 경로마다 새로운 파이프를 배치하는 것으로 구현했다. 플레이어는 물줄기가 목적지까지 도착할 수 있도록 경로를 설계해야 할 뿐만 아니라, 물줄기가 흐르는 시간에 비례해 점수가 부여되기 때문에 높은 점수를 기록하기 위해선 물이 최대한 오래 흐를 수 있도록 파이프를 배치해야 한다.

게임오버 조건은 일반적인 파이프 게임과 유사하게 설계했다. 한번 파이프가 배치된 자리에 또 파이프를 놓을 수 없으며, 게임 스크린을 벗어나는 위치에 파이프를 놓을 수 없다. 또한 물줄기가 지나가는 경로에 파이프가 이어져 있지 않거나, 파이프를 배치하기도 전에 물줄기가 먼저 도착한다면 게임이 오버된다. 보고서의 3, 4 번 부분에서 해당 프로젝트가 어떤 모습으로 실행되는지, 그리고 각 기능은 어떻게 구현하였는지를 기술해 두었다.

3. 프로젝트 실행 결과

1) 프로젝트를 빌드하면 가장 먼저 아래와 같은 화면을 확인할 수 있다.

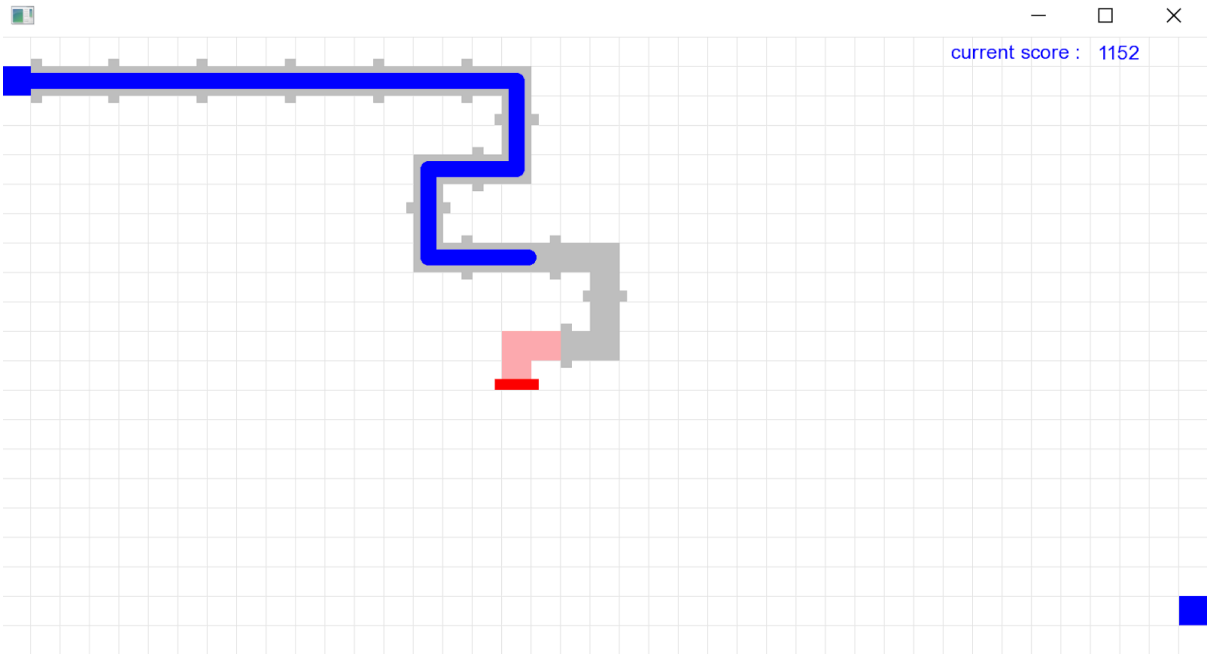
〈 그림 1. 파이프 게임 시작 화면 〉

플레이어는 키보드의 S 키를 눌러 게임을 시작할 수 있다. 물줄기는 좌측 상단에서 시작되며, 우측 하단이 물줄기의 목적지가 된다. 물이 시작되는 위치 앞에 직선 파이프 한개가 기본으로 셋팅되어 있다.

Press S to start 아래에는 조작 방법에 대한 간단한 설명이 나와있다. 키보드 좌/우 화살표 키로 파이프의 모양을 바꿀 수 있고 위/아래 화살표 키로 파이프를 회전시킬 수 있다. 스페이스키로 파이프를 배치할 수 있으며 파이프를 배치한 후에는 다음 위치에 파이프를 놓을 수 있게 된다. 게임 도중 Q 키로 게임을 중단할 수 있으며, 게임을 중단하는 경우 이는 게임오버로 간주된다.

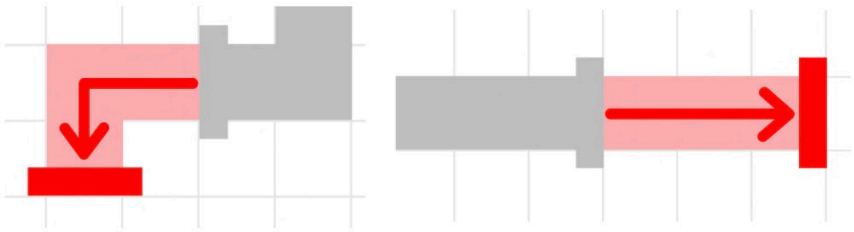
2) S 키를 눌러 게임을 시작하면 좌측 상단의 시작점에서 물줄기가 흐르기 시작한다.

플레이어는 물줄기가 목적지까지 도착할 수 있도록 파이프를 배치해야 하며, 물줄기는 플레이어가 배치하는 파이프를 따라 흐르게 된다. 물줄기가 흘러감에 따라 점수는 증가한다. 핑크색 파이프는 아직 놓지 않은 파이프이며, 회색 파이프는 이미 필드에 배치되어 그 모양이나 위치를 수정할 수 없는 파이프이다. 스페이스바를 눌러 파이프를 배치하면 핑크색 파이프가 회색으로 바뀌고, 다음으로 파이프를 배치해야 할 위치에 새로운 핑크색 파이프가 생긴다.

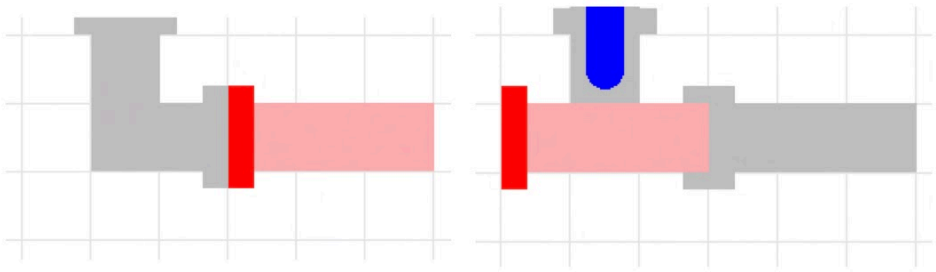


〈 그림 2. 파이프 게임 플레이 화면 〉

물줄기는 파이프의 빨간색 부분을 향해서만 흐를 수 있으며, 다음으로 배치할 파이프의 위치도 빨간색 부분의 방향으로 정해진다. 즉 같은 모양의 파이프를 배치하더라도 방향을 고려하지 않으면 게임오버가 되니 주의해야 한다.

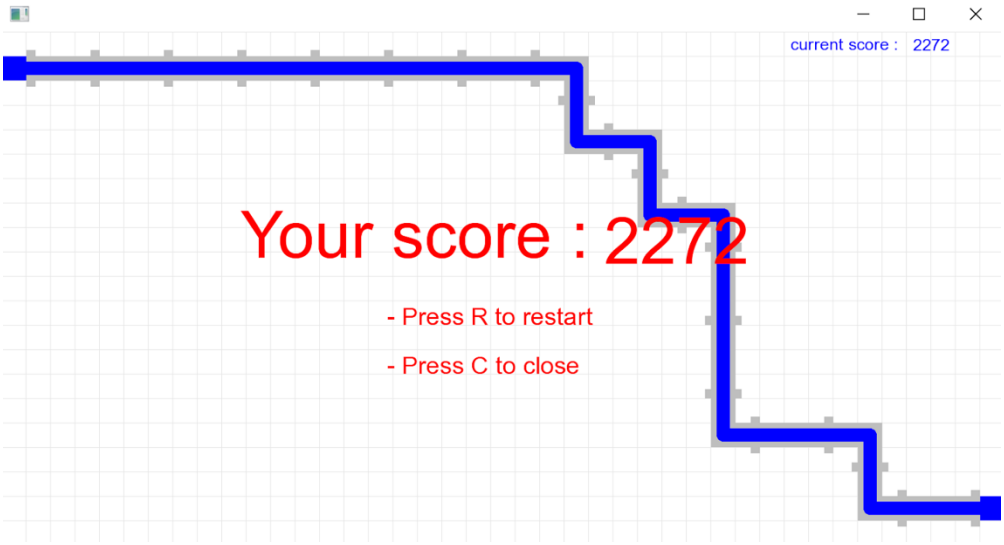


〈 그림 3. 파이프에서 물줄기가 흐르는 방향 및 다음 파이프가 놓일 위치 방향 〉



〈 그림 4. 방향을 고려하지 않아 파이프가 놓인 곳에 또 파이프를 놓는 상황이 되었다 : 게임오버 〉

3) 물줄기가 목적지에 무사히 도착한 경우 게임은 끝이 난다.



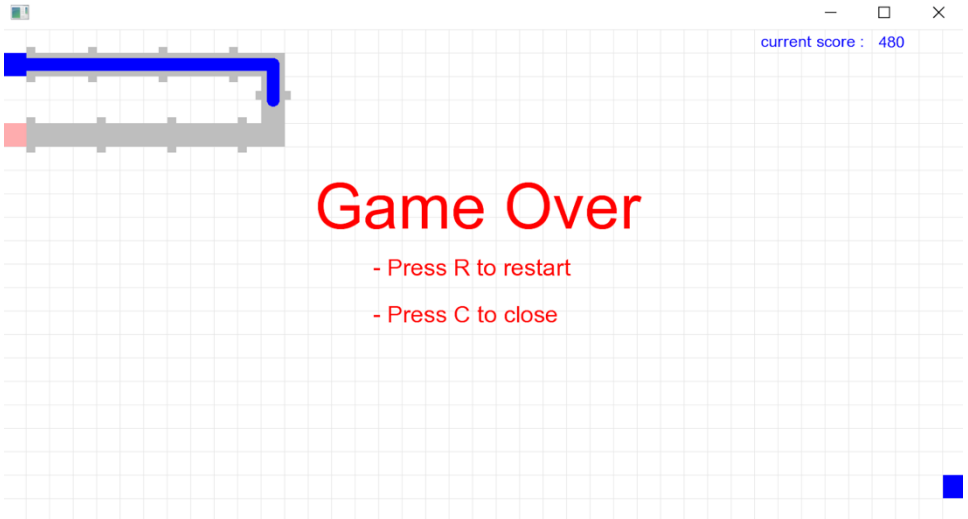
〈 그림 5. 게임이 정상적으로 끝난 경우의 화면 〉

이 때 내가 얻은 점수를 화면에서 확인할 수 있으며, R 키를 누르면 화면이 초기 상태로 돌아가 게임을 재시작 할 수 있다. C 키를 누르면 게임이 종료되고 게임 화면이 꺼진다.

물줄기가 도착하기 전에 이미 목적지까지의 파이프 경로 배치를 완료한 경우, 더이상 파이프를 배치할수는 없다. 이 때는 물줄기가 파이프를 따라 목적지에 무사히 도착할 때 까지 기다리면 된다.

4) 게임오버가 되는 경우는 아래 총 4 가지 경우이다

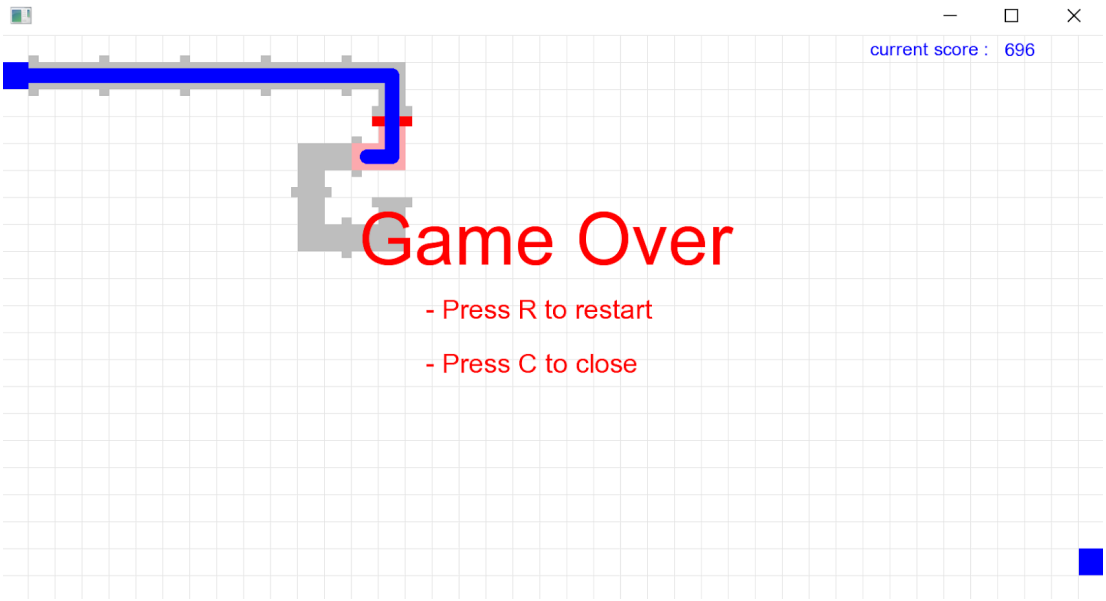
- 파이프를 게임 스크린 범위 바깥에 배치하는 상황



〈 그림 6. 게임오버 1 : 파이프가 게임 스크린을 벗어남 〉

파이프를 게임 스크린 범위 바깥에 배치해야하는 상황에 놓이는 경우 게임오버 처리된다.

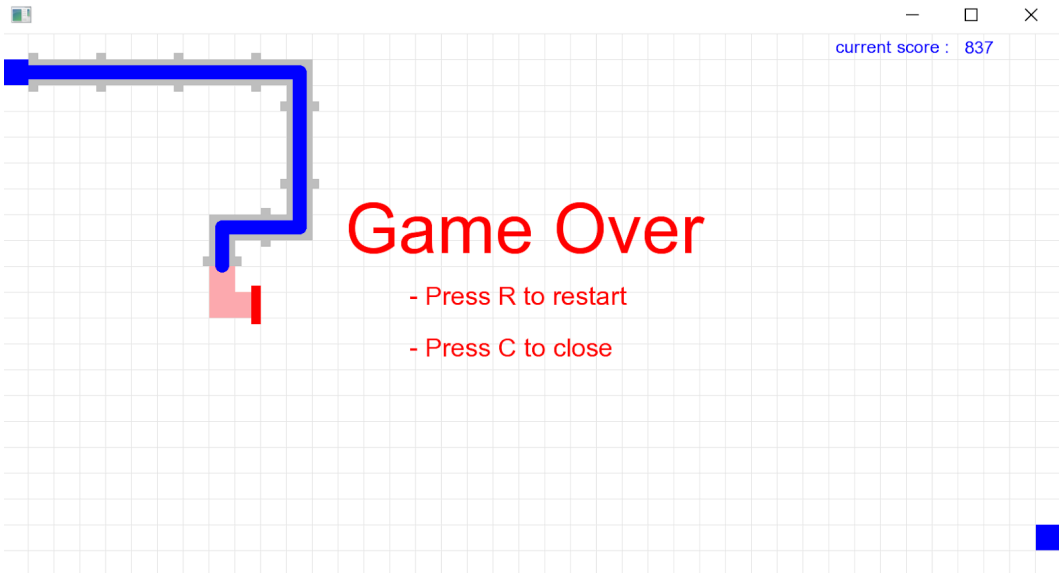
- 파이프를 놓은 자리에 또 파이프를 배치하는 상황



〈 그림 7. 게임오버 2 : 파이프 중복 〉

게임을 3D 로 구현한다면 파이프의 앞이나 뒤쪽에 배치할 수 있겠지만 해당 게임은 2D 이기 때문에, 파이프를 놓은 자리에 또 파이프를 배치해야 하는 상황에 놓이는 경우 게임오버로 처리했다.

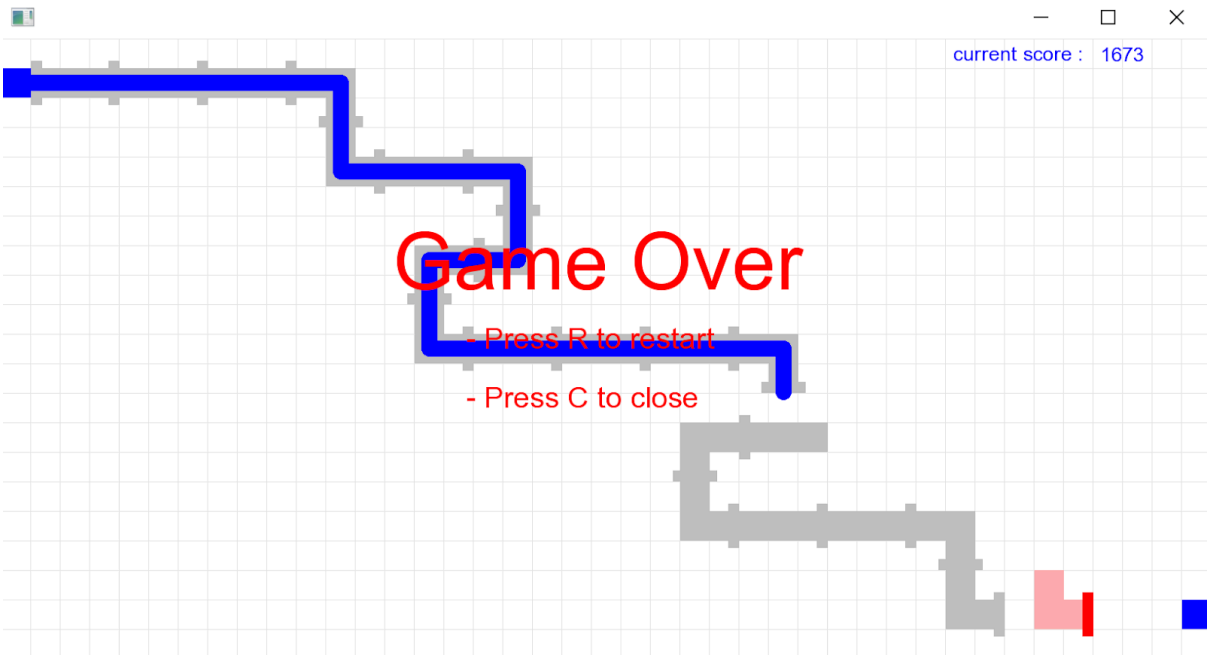
- 파이프를 배치하기 전에 물줄기가 먼저 도착하는 경우



〈 그림 8. 게임오버 3 : 파이프를 배치하기 전에 물줄기가 도착 〉

물줄기가 흘러가야 할 자리에 파이프가 아직 놓여져 있지 않은 경우 물이 샐 것이다. 따라서 이 경우 게임오버 처리된다.

- 물줄기가 지나가야 할 자리에 파이프가 끊어져 있는 경우



〈 그림 9. 게임오버 4 : 물줄기가 지나가야 하는 자리가 끊어져 있음 〉

물줄기가 흘러가야 할 자리에 파이프가 놓여져 있긴 하나 그 경로가 끊어져 있는 경우 물이 샐 것이다. 따라서 게임오버가 된다.

위의 4 가지 경우가 발생해 게임오버가 되면 R 키를 눌러 게임을 재시작하거나 C 키를 눌러 게임을 종료할 수 있다.

4. 코드 세부 설명

1) 자료구조

해당 프로그램 작성에 핵심적으로 이용한 자료구조는 연결리스트이다. Openframework 의 draw( ) 함수는 프로그램이 실행되는 동안 일정한 주기마다 반복적으로 다시 호출되기 때문에, 함수가 매번 호출될 때 마다 그려야 할 오브젝트의 정보를 저장하고 있는, 그리고 지속적으로 저장 정보를 업데이트 할 수 있는 전역적인 저장공간이 필요하다.

파이프 게임 프로그램에서 저장하고 있어야 하는 정보는 필드 위에 놓여진 파이프의 정보와 시작점에서 시작해 파이프를 따라 흐르는 물의 경로이다. 위 두가지 정보는 시간이 흐름에 따라 새로운 정보가 추가되며 추가된 정보는 새로운 노드로 생성되어 연결리스트의 맨 마지막에 연결된다. 예를들어 필드에 파이프를 새롭게 배치하는 경우, 배치한 파이프의 모양, 회전수, 배치된 위치 정보는 새로운 pipe 구조체 포인터 노드에 저장되어 현존하는 pipe 노드 연결리스트의 맨 마지막

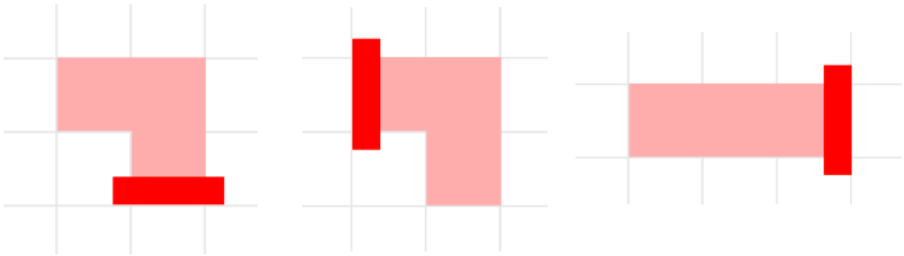
노드의 next 로 연결된다. 그 다음에 또 새롭게 배치되는 파이프 정보는 똑같이 직전에 생성되었던 노드의 next 에 연결된다.

물의 경로를 저장하는 연결리스트 역시 파이프가 새롭게 필드에 배치될 때 마다 새로운 값이 추가된다. 새로운 파이프가 필드에 추가되면, 파이프의 가운데를 가로지르는 경로가 물의 경로를 저장하는 연결리스트에 차례대로 저장된다. 예를 들어 A 에서 B 로 물이 이동하는 파이프가 새롭게 배치되는 경우, 물이 지나가야하는 경로를 저장하는 연결리스트에는 A 에서 B 까지 파이프를 따라 이동할때 지나는 좌표점 정보가 1 의 간격으로 각각 연결리스트의 뒤에 차례대로 저장된다.

수업시간에 구현한 프로그램에서 테트리스 블럭 정보는 2 차원 배열을, waterfall 에서의 물이 흐르는 경로는 1 차원 배열을 이용하여 저장했는데 파이프 게임의 경우 사용자에 따라 만들어지는 파이프의 개수와 물줄기의 길이가 다르기 때문에 정적 배열을 이용하면 공간 낭비가 생길 수 있어 그 크기를 동적으로 관리할 수 있는 연결리스트를 사용했다.

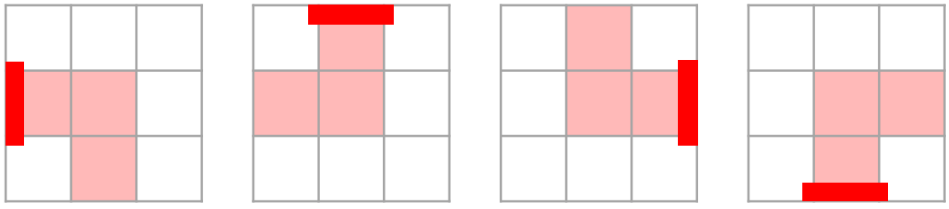
2) 알고리즘

파이프를 필드에 배치하고, 물이 파이프가 배치된 것을 따라 흐르게 하는 알고리즘은 간단하다. 먼저 파이프를 필드에 배치하는 과정은 다음과 같다. 파이프는 언제나 3 개의 정사각형으로 이루어져 있으며, 일직선으로 놓이거나 'ㄱ' 자로 놓이는 경우 두가지가 있다. 하지만 물이 흐르는 방향을 고려해야 하기 때문에 ㄱ자로 놓이는 경우에는 두가지 모양이 있다고 할 수 있다.



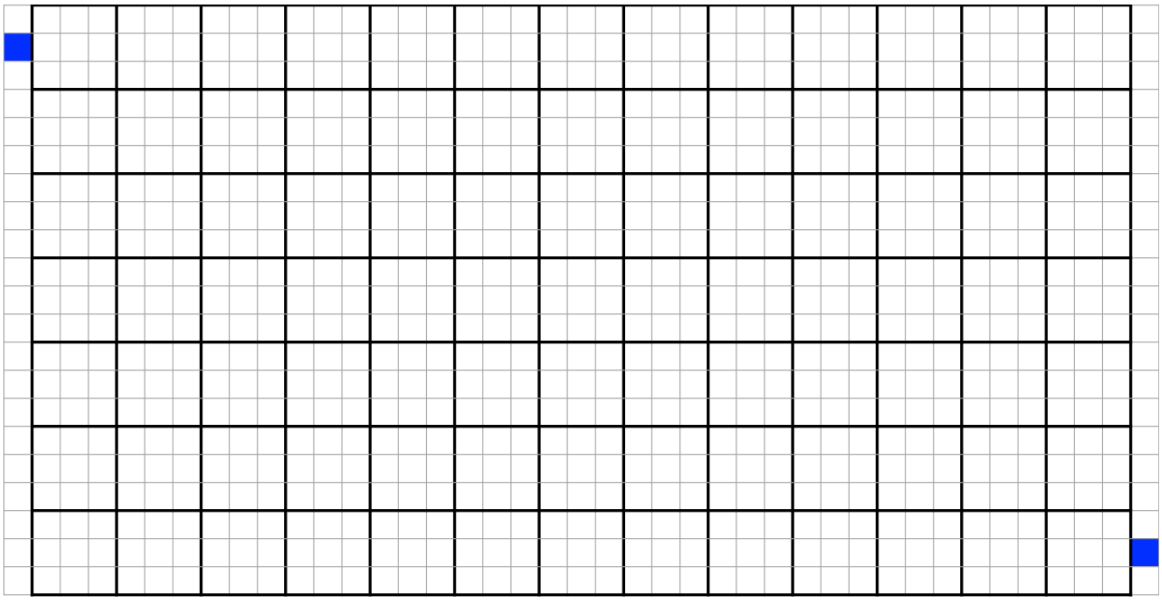
〈 그림 10. 파이프 종류 3 가지(빨간색 쪽으로 물이 흘러나간다) 〉

각 파이프는 90 도씩 4 방향으로 돌릴 수 있기 때문에, 필드에서 3*3 칸 만큼의 자리를 잡아놓으면 블럭의 모양과 회전을 자유롭게 변경할 수 있다.



〈 그림 11. 파이프를 4 방향으로 회전하기 위해서는 3*3 만큼의 자리가 필요하다 〉

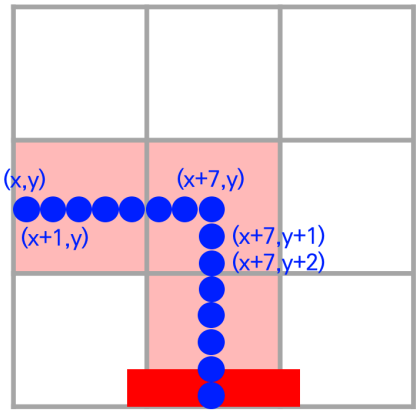
따라서 3*3 의 파이프 블럭들을 빈틈없이 배치할 수 있도록 필드를 크기를 구성하면 파이프를 3*3 칸마다 배치할 수 있다.



〈 그림 12. 3*3 의 파이프 블럭들을 빈틈없이 배치할 수 있도록 구성된 필드 〉

이렇게 구성하면 블럭의 좌측 상단 꼭짓점의 좌표값, 파이프의 모양, 회전수만 알고 있으면 계산식을 통해 파이프를 필드에 그려넣을 수 있다. 때문에 pipe 노드는 파이프의 type 번호, 회전수, 좌측 상단 꼭짓점의 좌표값 x · y, 다음 노드를 가리키는 구조체 포인터 next 로 이루어진다.

다음으로 물이 파이프가 배치된 것을 따라 흐르게 하는 과정을 다음과 같다. 파이프를 필드에 새롭게 배치하면 해당 파이프 정보는 pipe 구조체 포인터 노드에 저장되어 pipe 노드 연결리스트의 끝에 저장된다. 이 때 물이 지나가는 경로 역시 물의 경로를 저장하는 water 구조체 포인터 노드에 저장되어 water 노드 연결리스트의 끝에 저장된다. water 노드는 파이프의 정 중앙 경로의 좌표값을 1 간격으로 저장한다. 예를들어 일직선 파이프를 지나는 경로를 저장하는 경우, 파이프가 (x,y) 점에서 (x+10, y+10) 좌표점으로 물이 흐르게 한다면 water 노드에는 (x+1,y) 의 x, y 좌표가 저장되고, 이 노드가 가리키는 next 노드는 (x+2,y)의 좌표를, 또 이 노드가 가리키는 next 노드는(x+3,y) 좌표를 저장하는 식으로 차례대로 경로를 따라 노드가 생성되고 저장되는 것이다.



〈 그림 13. 물이 지나가는 경로가 저장되는 과정 〉

이처럼 파이프와 물의 경로를 연결리스트에 저장해두면 draw 함수가 호출될 때 마다 연결리스트의 header 노드부터 next 노드가 NULL 인 노드까지 훑으며 저장된 파이프들과 물줄기를 화면에 그릴 수 있는 것이다. 결과적으로 이 프로그램의 시공간 복잡도는 배치하는 파이프의 개수에 비례한다.
다음으로 게임 오버를 감지하는 알고리즘이다. 게임오버가 일어나는 경우는 위에서 4 가지 라고 설명했지만, 크게 3 가지로 나눠 볼 수 있다. 파이프가 스크린 범위를 벗어나는 경우, 파이프가 놓인 곳에 또 파이프를 놓는 경우, 물줄기가 지나갈 경로가 끊어진 경우가 그 3 가지 이다.

먼저 파이프가 스크린 범위를 벗어나는 경우는 파이프를 그릴 3*3 영역의 좌측 상단 꼭짓점의 좌표가 스크린의 가로 세로 길이를 초과하지 않았는지만 살펴보면 된다. 스크린의 크기는 1640*840 으로 설정했는데 이 중에서 파이프가 놓일 수 있는 영역은 가로 0~1560, 세로 0~840 이므로 좌측 상단 꼭짓점의 x 좌표가 0 보다 작거나 1560 보다 커지는 경우, 좌측 상단 꼭짓점의 y 좌표가 0 보다 작거나 840 보다 커지는 경우를 게임오버 처리하면 된다.

두번째로 파이프가 놓인 곳에 또 파이프를 놓는 경우는 스크린에서 특정 위치를 색상을 추출하는 방법을 이용했다. 현재 파이프를 놓고자 하는 자리에 이미 파이프가 있으면 해당 3*3 영역의 정 중앙은 회색이거나(파이프만 있음), 파란색일 것이다(물줄기가 이미 지나감). 따라서 현재 파이프를 놓고자 하는 자리의 정 중앙 좌표점의 색상을 확인해 회색이거나 파란색인 경우 게임오버로 처리해 주면 된다.

마지막으로 물줄기가 지나갈 경로가 끊어진 경우이다. 이는 두가지 경우가 있을 수 있다. 첫번째는 아예 다음 파이프가 놓이지 않은 경우이다. 이는 물줄기의 경로를 저장하는 노드를 훑다가 next 가 NULL 인 노드에 도착했는데 현재 좌표점이 목적지 좌표가 아닌 경우이며, 이 때 게임오버 처리를 해주면 된다. 두번째는 다음 파이프가 놓여졌으나, 그 경로가 끊어진 경우이다. water 노드 연결리스트에 저장된 정보를 바탕으로 물줄기 좌표를 그려나갈때 좌표 사이의 거리가 1 보다 클 수가 없다. 물줄기는 쭉 이어지기 때문에 언제나 x 축 또는 y 축으로 1 씩만 이동하기 때문이다. 따라서 물줄기 표현을 위해 파란색의 작은 원을 그려나갈 때, 현재 노드가 가리키는 좌표와 현재 노드의 next 에 연결된 좌표 사이의 거리를 측정해 이것이 1 보다 커지는 경우가 발생하면 이를 게임오버 처리해줘야 한다.

3) 핵심 변수 및 함수

핵심 변수는 위에서 설명한 바와 같이 pipe 구조체 포인터와 water 구조체 포인터 변수이다. 핵심 함수는 물줄기가 그려질 위치를 계산해서 water 노드를 생성하는 함수(addPath), pipe 노드에 저장된 정보를 이용해 필드 위에 배치된 파이프(회색 파이프)와 현재 배치하고자 하는 파이프(분홍색 파이프)를 그리는 함수이다(drawPipe 와 placePipe 함수).

pipe 노드를 생성하는 것은 현재 배치하고자 하는 파이프의 정보를 그대로 가져오면 되기 때문에 따로 함수로 구현해두지는 않았다. 또한 water 노드에 저장된 정보를 이용해 필드 위에 물줄기를 그리는 것은 OpenFramework 의 ofDrawCircle 함수와 원의 중심 좌표(water 노드에 저장되어 있는 각 좌표)만 있으면 되기 때문에 굳이 함수로 작성하지는 않았다.

4) 창의적 구현

테트리스 수업 시간에 배운 블럭 배치 원리와 waterfall 시간에 배운 물줄기 그리는 방법을 적절하게 조합하여 새로운 프로그램을 제작했다는 부분에서 창의적인 발상이 사용되었다고 이야기할 수 있다.

5. 느낀 점 및 개선사항

어떤 기능들을 구현하고 싶은지를 생각하는 것은 어렵지 않았지만, 그 기능들을 구현할 자료구조/알고리즘을 설계하고 코드로 구현하는 것이 생각보다 어려웠다. 특히 물은 파이프를 따라 흘러야 하기 때문에 물의 경로를 배치된 파이프로부터 계산해 내는 과정이 계산식이 많이 필요해 코드가 복잡해졌다. 그리고 처음에 파이프 블럭을 어떻게 화면에 일관되게 배치할 수 있을지를 고민했는데, 테트리스에서 사용한 방식(4*4 의 정사각형 grid 공간 안에서 색칠될 블럭을 정한다)을 활용하니

이를 쉽게 해결할 수 있었다. 결과적으로 머릿속에 어렴풋이 떠올리기만 했던 파이프 게임을 직접 구현하는데 성공하여 뿌듯했다.

개선하고 싶은 부분은 물의 경로상에서 파이프가 끊어졌거나 파이프를 미처 배치하지 못해 게임오버가 되는 경우 물이 파이프의 끝에서 넘쳐흐르는 모션을 추가하고 싶었는데, 이미 존재하는 자료구조 만으로도 프로그램이 무거워 해당 부분은 생략하였다. 더 나은 프로그램 작성 툴을 사용하거나 자료구조 및 알고리즘을 간소화 시켰다면 구현할 수 있었을 것 같은 부분이었기에 조금 아쉬웠다.