최종 프로젝트 보고서

전공: 기계공학과 학년: 3학년 학번: 20191820 이름: 김형준

**1. 실험 환경**

OS : Windows 10 Home

CPU : Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz 2.60 GHz

RAM : 16.00GB

Compiler : Visual Studio 2022 Release Mode

**2. 프로젝트 목표**

실습 때 진행했던 maze 프로젝트와 waterfall 프로젝트를 합쳐서 새로운 미로탈출 게임을 제작한다.

프로그램 실행시 시작화면이 출력되고, 버튼을 클릭해 (또는 키보드 ‘S’키를 눌러) 스테이지 선택창으로 이동한다. 시작화면에서 How To play버튼을 클릭해 (또는 키보드 ‘H’키를 눌러) 게임설명 화면으로 넘어가 게임에 대한 설명을 출력한다. 이후, 게임설명 화면에서 버튼을 눌러(또는 키보드 ‘S’키를 눌러) 스테이지 선택창으로 넘어간다.

스테이지는 총 5개의 스테이지로 구성되어 있으며, 각 스테이지는 이전 스테이지를 클리어해야 플레이 할 수 있다. 각 스테이지는 해당 스테이지의 버튼을 눌러 (또는 키보드 ‘1’~’5’키를 눌러) 플레이 할 수 있으며, 스테이지 화면에서 ‘Q’키로 다시 스테이지 선택창으로 돌아올 수 있다.

스테이지 시작시 정해진 시간제한 내로 시작점 (맨 왼쪽 위)부터 도착점 (맨 오른쪽 아래) 까지 도달해야 한다. 제한시간 이내로 도달하지 못할 경우 게임오버 처리되며, 도달시 해당 스테이지를 클리어 한 것으로 처리한다. 각 스테이지는 정해진 미로를 사용하여 (프로그램 시작시 로드함) 해당 미로를 시간에 따라 달라지는 3x3 변환 행렬을 사용하여 선형변환을 적용해 다양한 효과를 준다. 미로 내부의 모든 물체(플레이어 캐릭터 포함)들도 해당 선형변환을 적용해 미로와 함께 움직인다. 이때, 플레이어 캐릭터의 이동방향 및 거리는 변환 후의 좌표를 기준으로 상하좌우로 정해진 거리만큼 움직이게 하여 선형변환과 관계없이 캐릭터의 조작은 항상 일정하도록 유지한다. (3x3 변환행렬의 역행렬을 사용하여 구현한다.) 미로 내부에는 3개의 동전이 랜덤한 위치에 배치되며, 캐릭터에 동전이 닿은 경우 해당 동전을 먹은 것으로 처리하고, 점수 계산에 포함시킨다.

키보드의 ‘H’키를 눌러 (게임당 1회 제한) 현재 위치에서 10초간 도착점까지 가는 경로를 출력한다. 미로 내부에 물줄기 구멍과 용암 구멍을 설치하여 미로 탈출시 추가적인 장애물로 기능하도록 한다. 물줄기에 캐릭터가 닿는 경우, 닿은 시간으로부터 5초간 둔화 상태에 걸리게 되며, 둔화상태인 경우 이동속도가 일반 상태의 절반으로 설정한다.

용암에 캐릭터가 닿는 경우, 즉시 게임오버 처리되게 설정한다. 물줄기와 용암은 구멍으로부터 흘러나오며, 미로의 맨 밑 바닥에 닿거나 움푹 파여진 구덩이 모양 (V자 같이 오목한 부분으로 들어가는 경우)에 도달하면 멈추게 된다.

게임 클리어시 점수에 따라 획득한 코인의 개수를 결정한다. 점수는 남은 제한시간이 많을수록, 미로 내의 동전을 많이 먹었을수록 높아지며, 힌트를 사용했을 경우 총 점수의 절반을 감점한다. 클리어시 현재 최고기록과 클리어한 스테이지 수를 세이브 파일(save.txt)에 별도로 저장하고, 프로그램 재실행시 해당 세이브 파일을 로드해 이전 기록을 불러올 수 있게 한다.

스테이지 선택창에서의 버튼에는 각 스테이지의 최고기록 (최단시간, 최고 획득 코인 개수)를 표시하며, 최고기록이 갱신되었을 경우 해당 사항을 반영하도록 한다. 게임오버시 재도전 버튼을 눌러 (또는 키보드 ‘R’키를 눌러) 해당 스테이지를 재도전 할 수 있게하고, 메뉴로 돌아가는 버튼을 눌러 (또는 키보드 ‘Q’키를 눌러) 스테이지 선택창으로 돌아갈 수 있게 만든다.

클리어 화면에서는 마지막 스테이지가 아니면 버튼을 눌러 다음 스테이지를 바로 도전할 수 있게 만들고, 메뉴로 돌아가는 버튼을 눌러 스테이지 선택창으로 돌아갈 수 있게 만든다.

**3. 각 변수에 대한 설명**

ofTrueTypeFont myFont; // 사용할 폰트

ofImage Title, StageTitle, tipTitle, tip; // 사용할 이미지들

Node\* Mtop = NULL; // DFS용 스택의 Top

FILE\* fptr; // 세이브 파일 포인터

MData MArr[Stage\_Num]; // 미로 데이터를 저장하는 배열

vector<Line> Path; // 힌트 경로를 저장하는 배열

Player Box; // 플레이어 데이터 저장

int Stage\_Clear, Star\_Clear[Stage\_Num], Screen=1, prevScreen; // 각각 클리어 한 스테이지 수, 각 스테이지의 최고 코인 갯수, 현재 화면 번호, 이전 화면 번호

double Time\_Clear[Stage\_Num], Limit\_Clear[Stage\_Num] = {60.0,90.0,60.0,90.0,45.0}; // 각각 클리어한 최단시간, 스테이지별 시간제한

ofTrueTypeFont myFont24, myFont16, myFont12, myFont48, myFont60, myFont36, myFont20; // 폰트용 변수

Button BArr[Stage\_Num]; // 버튼의 정보를 저장하는 배열

double Matrix[3][3]; // 변환용 행렬

double Inverse[3][3] = { {1,0,0},{0,1,0},{0,0,1} }; // 역변환용 행렬

int WaterNum[Stage\_Num] = { 2,4,10,12,8 }; // 스테이지별 물줄기 갯수

double ScrX[Stage\_Num], ScrY[Stage\_Num]; // DrawMaze 첫 호출시 화면 크기 (각각 가로, 세로크기)

**4. 각 함수에 대한 설명**

void setup(); // 초기설정 : 미로, 폰트, 이미지파일들을 로드함

void draw(); // 게임화면을 그리는 함수

void keyPressed(int key); // 키인식 관련 함수

void mousePressed(int x, int y, int button); // 마우스 클릭인식 관련 함수

void SetCoinLoc(int Num, int Limit); // 코인의 위치를 재설정함

void SetStageSetting(int Num, double Zoom); // 스테이지를 설정을 초기화함

void StageReset(int Num); // 스테이지를 초기화함

void update\_file(); // 세이브 데이터를 파일로 출력하는 함수

void Ensnare(); // 둔화 상태를 설정하는 함수

void DrawWater(int N); // 물줄기 및 용암 경로 계산후 그림

void Water\_addpath(Water& Wobj, double X, double Y, int index); // Path에 물줄기의 경로(점)을 추가함

void Water\_addpath(Water& Wobj, Point P, int index); // 바로 윗 함수의 overloading 버젼, X,Y대신 Point를 받음

void Water\_addblock(Water& Wobj, Block B, int index); // wblock에 플레이어-물줄기 충돌판정 Block을 추가함

void Water\_calcpath(int N, Water& Wobj); // 물줄기의 경로를 계산하는 함수

void Matrix\_Clear(double(\*A)[3]); // 변환행렬 초기화 함수

void Matrix\_Mul(double(\*A)[3], double(\*B)[3], double(\*C)[3]); // 두 행렬을 곱함 (C = A \* B)

void Matrix\_Copy(double(\*A)[3], double(\*B)[3]); // 행렬을 복사함 (B = A)

double Matrix\_GetDet2(double(\*A)[2]); // 2x2 행렬의 determinant값을 구함

double Matrix\_GetDet3(double(\*A)[3]); // 3x3 행렬의 determinant값을 구함

int Matrix\_Inverse(double(\*A)[3], double(\*B)[3]); // 역행렬을 구함 (B = A^-1)

Point Matrix\_Trans(double(\*A)[3], double X, double Y); // X,Y 좌표를 행렬A로 변환한 값을 구함

Point Matrix\_Trans(double(\*A)[3], Point P); // 바로 위 함수의 overloading 버젼, X,Y대신 Point로 받음

Block Matrix\_TransBlock(double(\*A)[3], Block B); // Block을 행렬A로 변환한 값을 구함

Line Matrix\_TransLine(double(\*A)[3], Line L); // Line을 행렬A로 변환한 값을 구함

void Stack\_Move(double X, double Y); // 변환행렬에 평행이동 행렬을 곱하는 함수

void Stack\_Scale(double X, double Y); // 변환행렬에 크기조절 행렬을 곱하는 함수

void Stack\_Reflect(double X, double Y); // 변환행렬에 반사변환 행렬을 곱하는 함수

void Stack\_Rotate(double Angle); // 변환행렬에 회전변환 행렬을 곱하는 함수

void Stack\_RotateAt(double Angle, double X, double Y); // 변환행렬에 회전의 중심을 X,Y로 하는 회전변환 행렬을 곱하는 함수

void Stack\_Shear(double X, double Y); // 변환행렬에 전단변형 행렬을 곱하는함수

int Stack\_Project(int N, double LUX, double LUY, double LDX, double LDY, double RUX, double RUY, double RDX, double RDY); // 변환행렬에 투영행렬을 곱하는 함수

void DrawMaze(int N); // Draw Maze (힌트 사용시 DFS 사용으로 힌트도 출력함, 첫 호출시 현재 화면 크기를 기준으로 미로의 Orig, WOrig, WType을 계산함)

int CheckMaze(int N); // 미로의 벽(플레이어-벽 충돌판정)과 충돌하는지 체크함

int CheckCollide(Block Target); // Block내부에 플레이어가 존재하는지 검사함

int CheckCircle(double X, double Y, double R); // Circle내부에 플레이어가 존재하는지 검사함 (마름모로 근사함)

void DFS(int N); // DFS를 수행하는 함수

void dfsdraw(); // DFS로 구한 도착점까지의 경로를 출력함

void Push(int row, int col); // DFS용 스택 Push

void Pop(int\* row, int\* col); // DFS용 스택 Pop

void ClearStack(); // DFS용 스택을 초기화하는 함수

void calc\_Loc(int\* X, int\* Y, int N); // 현재 위치에서 인덱스 x, y를 계산함 (어느 방에 있는지를 계산)

double mapX(double X); // 인덱스 X에 해당하는 X좌표를 계산함

double mapY(double Y); // 인덱스 Y에 해당하는 X좌표를 계산함

Block MakeBlock(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4); // 네 점으로 block을 생성함

Line MakeLine(double x1, double y1, double x2, double y2); // 두 점으로 선을 생성함

void TR\_ofDrawLine(double x1, double y1, double x2, double y2); // 중심이 (x,y), 반지름이 r인 원을 변환후 좌표로 그림

void TR\_ofDrawCircle(double x, double y, double r); // 두 꼭짓점이 (x1,y1), (x2,y2)인 선분을 변환후 좌표로 그림

**5. 사용한 자료구조 설명**

typedef struct \_point { // 점의 좌표를 저장하는 구조체

double X; // X좌표

double Y; // Y좌표

} Point;

typedef struct \_room { // 미로의 각 방을 나타내는 구조체 (1 : 벽 존재, 0 : 벽 없음)

int visited; // 중복방지

int r; // 우측 벽

int d; // 아래 벽

}Room;

typedef struct \_node { // DFS 호출시 사용하는 스택

int row; // Y좌표

int col; // X좌표

struct \_node\* next; // 연결리스트용 포인터

}Node;

typedef struct \_block { // 점 4개로 구성되는 직사각형의 자표를 저장하는 구조체

Point P[4]; // 꼭짓점의 좌표

}Block;

typedef struct \_line { // 선분의 좌표를 저장하는 구조체

float x1; // 시작점의 X좌표

float y1; // 시작점의 Y좌표

float x2; // 끝점의 X좌표

float y2; // 끝점의 Y좌표

}Line;

typedef struct \_water { // 물줄기의 정보를 저장하는 구조체

double TotalTime; // 주기

double EnableTime; // 한 주기동안 물줄기가 켜지는 시간

double StartTime; // 물줄기가 처음으로 켜지는데 걸리는 시간 (대기시간)

double t; // 물줄기의 두께

int size; // 물줄기 경로의 갯수

int run; // 물줄기의 작동여부 (0:꺼짐, 1:켜짐)

int type; // 물줄기의 종류 (0 : 물, 1: 용암)

Point Dot; // 물줄기 시작점의 좌표

vector<Point> Path; // 물줄기의 경로를 저장한 배열

vector<Block> wblock; // 물줄기의 접촉 판정을 저장한 배열 (한 직사각형당 Block 1개를 저장함, 플레이어-물줄기 간의 충돌판정 계산시 사용함)

}Water;

typedef struct \_mdata { // 미로의 정보를 저장하는 구조체

int Row; // 미로의 세로크기

int Col; // 미로의 가로크기

Room\*\* Maze; // 미로의 정보를 저장하는 배열

vector<Block> Orig; // 변환전 미로의 벽의 위치 (플레이어-벽의 충돌판정에 사용함)

vector<Block> Trans; // 변환후 미로의 벽의 위치 (플레이어-벽의 충돌판정에 사용함)

vector<Water> Hole; // 미로 내의 물줄기를 저장하는 배열

vector<Line> WOrig; // 변환전 미로의 벽의 위치 (물줄기-벽의 충돌판정에 사용함)

vector<Line> Wall; // 변환후 미로의 벽의 위치 (물줄기-벽의 충돌판정에 사용함)

vector<int> Wtype; // 미로 벽의 종류 (WTest와 동일하나, DrawMaze최초 호출시 계산되어 WTest에 type값이 3인 항목을 제외한 모든 항목을 WTest에 저장함)

vector<int> WTest; // 미로 벽의 종류 (물줄기의 경로계산에 사용, 0:세로선 1:가로선 2:두께(가로및세로) 3:임시용(실제로 사용X) 4:가로선(내부) 5:세로선(내부)

}MData;

typedef struct \_button { // 스테이지 선택창의 스테이지 버튼의 정보를 저장하는 구조체

double X; // X좌표

double Y; // Y좌표

char name[10]; // 스테이지 이름

}Button;

typedef struct \_player { // 플레이어의 정보를 저장하는 구조체

Point Loc; // 변환전 플레이어 위치 좌표

Point TLoc; // 변환후 플레이어 위치 좌표

double SlowTimer; // 남은 둔화시간

double SlowStart; // 둔화가 시작된 시간

int Slowflag; // 둔화 상태 (0:정상, 1:둔화 걸림)

double size; // 플레이어의 크기

double Time; // 남은시간

int Coin; // 먹은 동전의 갯수

int hint; // 남은 힌트 갯수 (1: 사용안함, 0:사용함)

double baseX; // 미로를 그리기 시작할 X좌표

double baseY; // 미로를 그리기 시작할 Y좌표

double defaultX; // 첫 DrawMaze실행시 계산된 baseX (화면크기 보정용)

double defaultY; // 첫 DrawMaze실행시 계산된 baseY (화면크기 보정용)

double L; // 미로 한 칸의 길이

double zoom; // 미로를 확대할 배율

double speed; // 플레이어 이동속도

double slow\_speed; // 플레이어 둔화시 이동속도

Point Dest; // 도착점의 위치

double hintTimer; // 남은 힌트출력 시간

double StartTime; // 스테이지 시작 시간

double hintStartTime; // 힌트출력 시작시간

Point CoinDest[3]; // 동전의 위치 배열

int GetCoin[3]; // 각 동전의 먹었는지의 여부 (0:안먹음, 1:먹음)

Block Move; // 이동시 움직일 변위 배열(UDLR순서)

Block SlowMove; // 둔화중 이동시 움직일 변위 배열(UDLR순서)

}Player;

이 중에서 MData타입 전역변수 MArr[Stage\_Num]에서 Maze 멤버변수에 2차원 동적할당 배열을 사용하여 각 스테이지의 미로 데이터를 저장하고, DFS() 호출시 연결리스트 스택을 사용하여 현재 위치에서 도착점까지의 경로탐색을 수행한다.

**6. 알고리즘 및 시간/공간 복잡도 설명**

DFS 알고리즘 : 힌트 사용시 현재 위치에서 도착점까지의 경로를 표시하기 위해 현재 플레이어의 위치를 시작점으로 해서 (calc\_Loc()으로 현재 위치에 해당하는 미로의 인덱스를 구함), 위의 연결리스트 스택을 사용한 iterative DFS으로 탐색을 진행하고, 탐색 도중 도착점에 도달하면 루프에서 탈출한 다음 스택에 남아있는 노드들을 하나씩 꺼내면서 도착위치부터 시작위치까지의 경로를 전역변수 Path에 추가해 힌트 경로를 dfsdraw()로 출력한다.

DFS 탐색시 시간복잡도는 미로의 가로, 세로 크기를 각각 Width, Height라고 하면,

미로의 Edge개수는 최대 2\*Width\*Height이므로 O(Width\*Height)이다. 공간복잡도는 최악의 경우 탐색시 모든 edge를 탐색해야 하므로 O(Width\*Height)이다.

선형 변환 및 이동 좌표 계산 : 전역변수로 3x3 변환행렬 Matrix를 사용한다.

스테이지 시작시 변환행렬을 초기화 한 뒤 (항등 행렬로 초기화) 해당 스테이지에서 정해진 변환들을 변환 행렬에 순서대로 곱해서 해당 스테이지의 최종 변환 행렬을 계산한다. (시간에 따라 변환식이 달라지며, 매 프레임마다 시간에 따라 달라지는 변환식을 새로 계산함)

계산된 변환 행렬의 역행렬을 계산해 Inverse에 저장하고, 미로의 구성요소들에 (x’,y’,1) = Matrix \* (x,y,1) 연산을 통해 변환 후의 좌표를 계산해 화면에 출력한다.

플레이어 캐릭터의 이동방향과 크기는 변환행렬과 관계없이 항상 일정해야 하므로 변환 후의 값을 기준으로 변환 전의 플레이어 위치 좌표를 변경하는 방식으로 구현하였다.

매 프레임마다 Move, SlowMove에 변환후의 현재 위치에서 상하좌우 각각의 방향으로 움직인 후의 좌표를 저장하고 (SlowMove의 경우 둔화상태이므로 절반만 이동), Inverse를 사용해 역변환 한 다음 변환전 원래 위치의 좌표를 빼서 변환 후 상하좌우 방향으로 이동하는 움직임에 해당하는 변환전의 x, y방향 이동거리를 구한다.

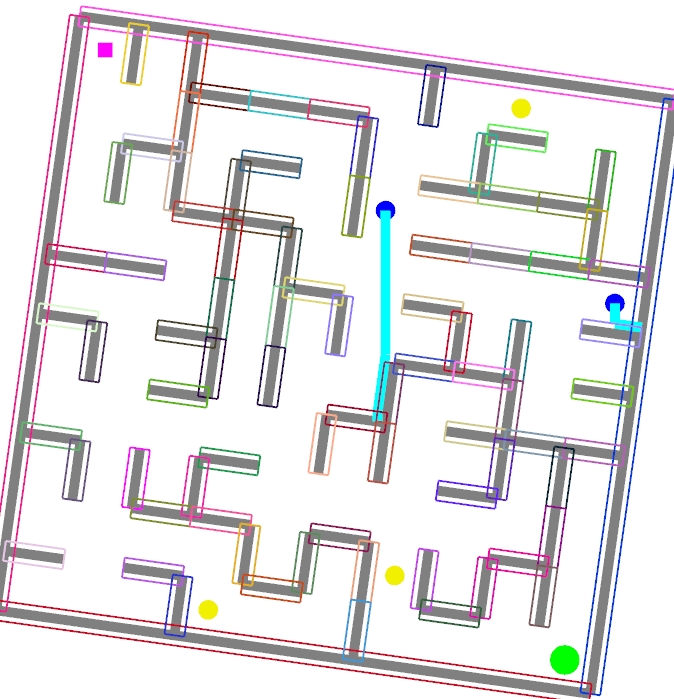
이후 방향키로 캐릭터를 조작할 때, 해당방향으로 진행할 수 있다면 (CheckMaze() 사용) 앞에서 구한 변환전의 x, y방향 이동거리를 현재 캐릭터의 좌표에 더해서 이동시 변환행렬과 관계없이 상하좌우 방향으로 일정하게 움직이게 구현하였다.

변환 행렬로 선형변환을 실행할 때의 시간복잡도는 고정된 연산을 수행하므로 (3x3 행렬곱) O(1)이고, 공간복잡도는 O(1)이다.

미로 데이터 처리 (DrawMaze 첫 호출시) : 프로그램 실행 후 스테이지를 플레이 하면, 해당하는Maze 데이터로 DrawMaze를 호출하는데, 해당 Maze 데이터의 Orig, WOrig, WType이 계산이 되어있지 않은 경우, 해당 미로의 데이터를 사용해 계산한다.

캐릭터 – 미로벽 간의 충돌판정을 체크하기 위해 (캐릭터가 벽을 뚫는 것을 방지) 미로 데이터의 Orig는 미로의 벽의 정보를 Block 타입으로 저장한다. DrawMaze에서 먼저 미로의 상하좌우 외각 벽들을 각각 하나의 Block으로 간주해서 저장한다. 그 다음 Maze를 2중 for문으로 순회하면서 벽이 있는 경우 해당 위치의 벽의 좌표로 Block을 생성해 Orig에 저장한다.

이렇게 저장된 Orig는 DrawMaze() 호출 이후 각각을 변환행렬로 변환한 값을 Trans에 저장하고

이 Trans와 변환후의 캐릭터 좌표값을 비교해 (CheckCollide 사용) 캐릭터 – 미로벽간의 충돌판정을 체크한다. (아래의 사진에서 회색 벽을 감싸는 얇은 사각형이 Orig->Trans의 결과로 만들어진 캐릭터 – 미로벽 사이의 충돌판정이다.)

물줄기 – 미로벽 간의 충돌판정을 체크하기 위해 (물줄기가 벽을 뚫고 흘러가는 것을 방지) 미로 데이터의 WOrig은 미로의 벽의 정보를 Line 타입으로 저장하고, WType은 WOrig과 1:1 대응되며, 해당 Line의 종류를 저장한다.

WOrig을 저장할 때, 물줄기의 흐름을 제어하기 위해 내부선과 외부선으로 나눠서 저장하는데, 외부선은 type=0 (세로선), type=1 (가로선), type=2 (두께선), type=4 (내부가로선), type=5 (외부가로선)

으로 분류된다. 물줄기가 외부선을 만나면 선을 따라 아래로 흐르면서 계속 진행하고, 내부선을 만나면 그 지점에서 물줄기의 경로가 끝나게 된다.

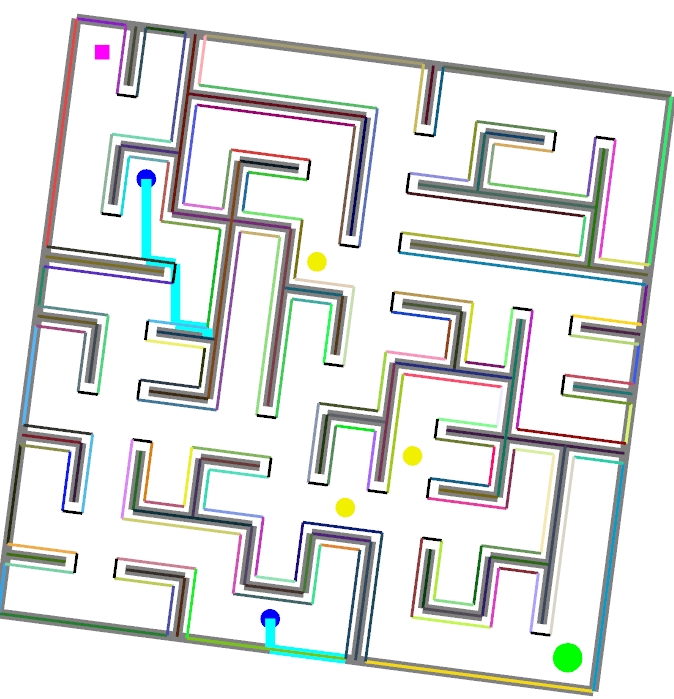
DrawMaze에서 먼저 미로의 상하좌우 외각 벽들을 따라가면서 외각선을 추가한다.

위쪽 외각 벽의 경우, 현재 방에서 왼쪽에 벽이 존재하면 벽의 두께(T) 때문에 위쪽 벽의 외각선은 두께만큼 왼쪽으로부터 짧아져야 한다. 마찬가지로 오른쪽 벽이 존재하면 오른쪽으로부터 두께만큼 짧아져야한다. (미로의 한 변의 길이 L은 2T보다 항상 크게 설정함)

이런 식으로 상하좌우 외각 벽과 인접한 두 벽의 존재 여부를 체크하여 외각벽 위치의 외각선의 길이를 결정하여 WOrig과 Wtype에 외각선을 추가한다.

그 다음 2중 for문을 사용해 각 방을 순회하면서 Orig을 구성하는 벽의 4개의 선분들 중 서로 겹치지 않는 선분만을 추출하여 각각 추가한다. 또한, 세로벽의 경우 벽 중심에 내부세로선을, 가로벽의 경우 내부가로선을 각각 추가한다. 이후, 위에서 WOrig에 추가했던 선분들을 같은 방향으로 이어진 것끼리 서로 연결하는데, WOrig을 순회하면서 현재 선분이 0,1,4,5 타입이면, 내부에서 WOrig을 처음부터 순회하면서 현재 선분과 같은 타입이며, 서로 다른 것들 중에서 한 좌표가 같고, 다른 좌표가 한 선분이 다른 선분 내부에 있을 때 (경계점 포함) 해당 선분을 합쳐서 WOrig의 값을 갱신한다. (합쳐지고 남은 한 선분은 타입을 3으로 변경해 더 이상 연산이 일어나지 않도록 설정함) 이후, WOrig에서 WType이 3이 아닌 선분들만 뽑아서 변환행렬로 변환한 값을 Wall에 저장하고, 해당 선분의 Type을 WTest에 저장한다.

이 Wall, WTest의 값과 변환후의 물줄기 좌표값을 비교해 (Water\_calcpath 사용) 물줄기 경로 계산시 물줄기 – 미로벽간의 충돌판정을 체크한다. (아래의 사진에서 회색 벽을 감싸는 선분들이 WOrig->Wall의 결과로 만들어진 외각선이고, 벽 내부의 선분들이 내부선이다.)

시간복잡도는 선분들을 합칠 때 3중 루프문을 돌고 (while의 최대 루프 횟수는 total개임)

추가되는 선분의 총 개수는 O(Width\*Height) (=total)이므로 O(Width^3\*Height^3)이다.

(2번째 해당 스테이지로 DrawMaze호출시 Orig, WOrig, WType 계산을 다시 하지는 않으므로 2번째 DrawMaze 호출부터는 O(Height\*Width)가 됨) 공간복잡도는 total개의 선분을 저장하므로 O(Width\*Height)이다.

WaterFall의 경로 계산 (Water\_calcpath) : 물줄기 – 캐릭터간의 충돌판정을 계산하기 위해 물줄기의 경로를 계산한다. 먼저, 변환후의 물줄기 시작점 좌표를 경로에 추가한다. (시작점 추가)

그 다음 Wall을 순회하면서 현재 선택된 벽(선분)의 시작점과 끝 점이 현재위치보다 위에 있거나, 둘 다 왼쪽에 있거나 오른쪽에 있거나 또는 수직으로 세워져 있으면 스킵한다.

그렇지 않으면, 벽의 기울기를 계산하고, 현재 위치에서 물줄기가 내려가서 벽에 닿을 때의 Y좌표 (currentY)를 계산한다. 이때, 닿은 좌표가 현재 Y좌표보다 위에 있어도 스킵한다.

벽의 종류가 외각선이고, 벽에 도달위치가 이전위치와 같은 경우 벽을 타고 흘러내렸을 때의 Y좌표가 이전위치보다 더 내려가지 않으면 스킵한다.

이전에 선택한 벽과 다르고, 벽의 도달위치가 minY보다 위에 있는 경우 최소값을 갱신한다.

최소값 계산연산이 실행되지 않았거나, 도달위치가 현재위치보다 올라가는 경우 경로추가를 마친다. 그 다음, 벽 위의 도달점을 경로에 추가하고, 이 벽이 내부선이면 경로추가를 마친다.

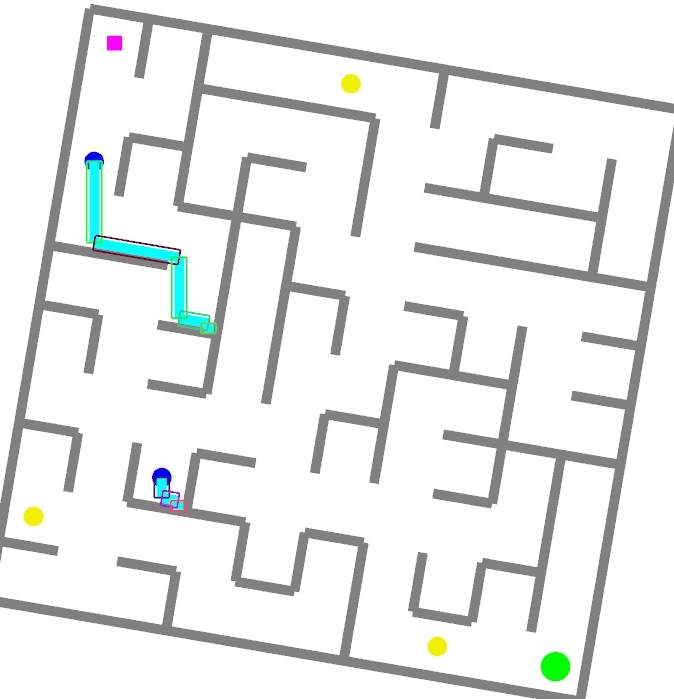
그렇지 않으면 벽의 탈출위치(벽의 양쪽 끝 중 더 아래에 있는 점)를 경로에 추가한다.

이전 점과 현재 점의 Y좌표가 같은 경우 횟수를 카운트해 2회 연속이면 경로 추가를 멈춘다.

그 다음 이전 좌표 변수에 현재 좌표를 저장하고 다음 루프로 넘긴다.

이렇게 계산된 Path값을 사용하여 두께를 고려한 물줄기-캐릭터 충돌판정을 Block 형태로 wblock에 저장한다. 위에서 구한 Path값으로 DrawWater에서 물줄기 및 용암을 화면에 그리고,

wblock과 캐릭터 좌표를 비교하여 (CheckCollide) 물줄기 및 용암에 캐릭터가 닿았는지를 체크한다. (아래의 사진에서 물줄기를 감싸는 선분들이 Path->wBlock의 결과로 만들어진 물줄기-캐릭터 충돌판정 이다.) 시간복잡도 : O(Width\*Height), 공간복잡도 : O(Width\*Height)



충돌판정 (CheckCollide) :

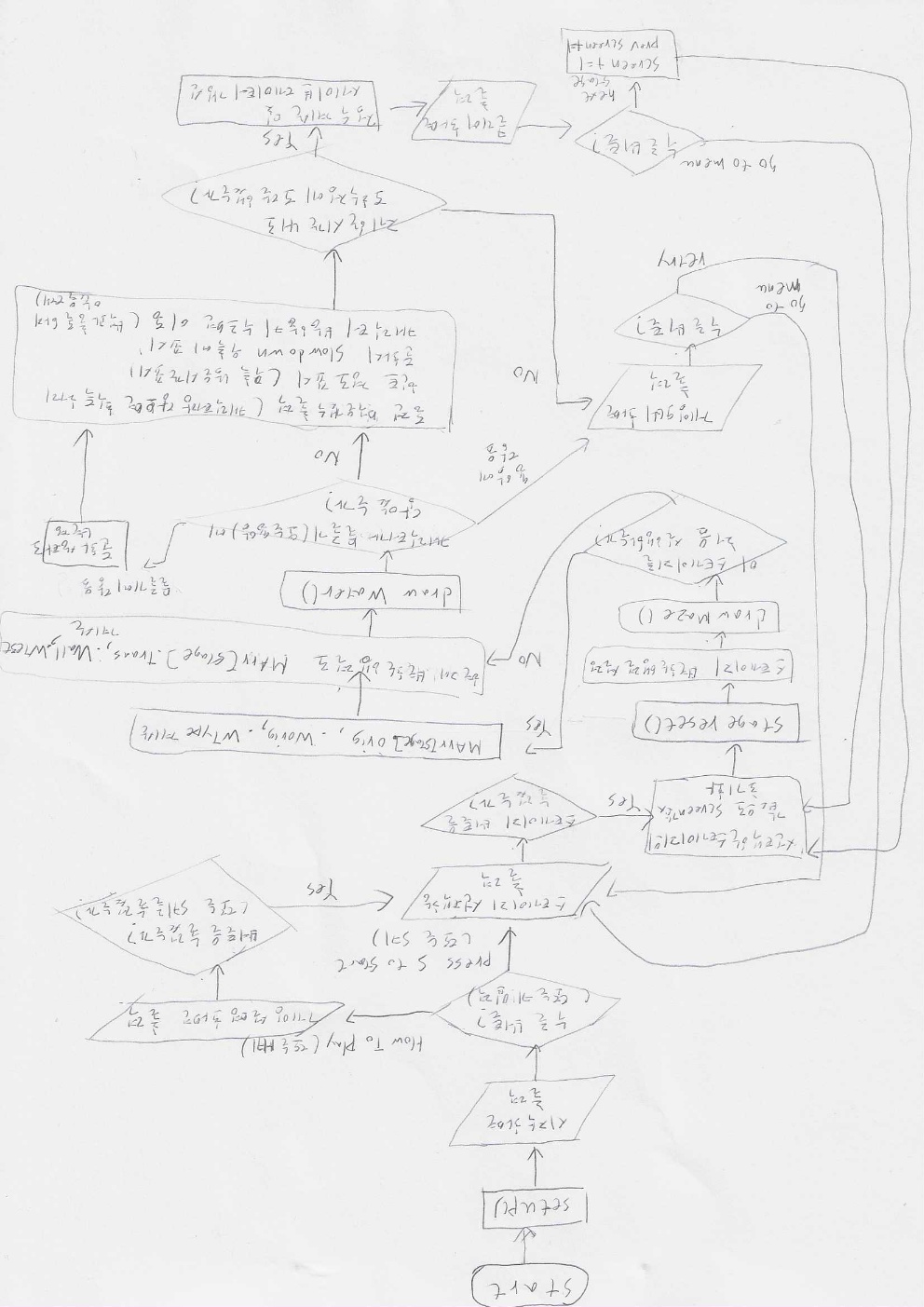
입력으로 받은 Block 내부에 캐릭터가 있는지 없는지를 판정하기 위해 4개의 변 각각에 대해서

캐릭터 좌표를 기준으로 해당 변의 시작점과 끝점이 모두 캐릭터의 위 또는 아래쪽에 있으면 해당 변과의 교점이 없는 것으로 처리한다. 아니면 캐릭터의 위치에서 수평선을 그어서 변과 만나는 점의 X좌표를 구한 뒤, 이 X좌표가 캐릭터의 x좌표보다 오른쪽에 있으면 교점의 개수를 1 증가한다. 현재 변이 수평선인 경우, 캐릭터의 y좌표와 같은 경우 교점을 1 증가한다.

교점의 합이 짝수면 외부에 있다고 판정하고, 교점의 합이 홀수면 내부에 있다고 판정한다.

시간복잡도 : O(1), 공간복잡도 : O(1)

**7. 전체 프로그램 플로우 차트**

****

**8. 창의적 구현 항목에 관한 설명**

기존 maze 프로젝트를 응용하여 시간제한형 미로탈출 게임을 만드는데, waterfall 프로젝트에서 구현한 물줄기 시스템을 결합하여 미로를 탈출하는데 일종의 장애물로 기능할 수 있도록 했다.

그러나 미로가 자체가 움직이지 않으면 물줄기는 벽에 가로막혀 고정된 경로만을 항상 출력하게 되므로 미로를 포함한 전체 필드가 시간에 따라 달라지는 3x3 변환행렬을 사용하여 선형 변환을 통해 움직이게 만들고 물줄기의 경로는 항상 아래 방향을 향하게 해, 시간에 따라서 물줄기의 경로가 변할 수 있도록 하였다.

waterfall 프로젝트의 물줄기를 게임에 추가할 때 서로 다른 종류의 물줄기를 구현했는데, 물줄기 (닿으면 5초간 둔화), 용암 (닿으면 게임오버)은 같은 자료형(Water)을 사용하지만, 내부 타입변수(type)로 구분하여 같은 물줄기 시스템에서 서로 다른 효과를 낼 수 있도록 구현하였다.

물줄기를 꺼지고 켜지는 타이밍을 조절할 수 있도록 하여 각 스테이지마다 다양한 기믹을 넣을 수 있도록 하였다.

또한, dfs를 활용한 힌트를 게임 도중에 사용할 수 있도록 하여 도착지까지의 남은 경로를 실시간으로 보여줄 수 있도록 하였다.

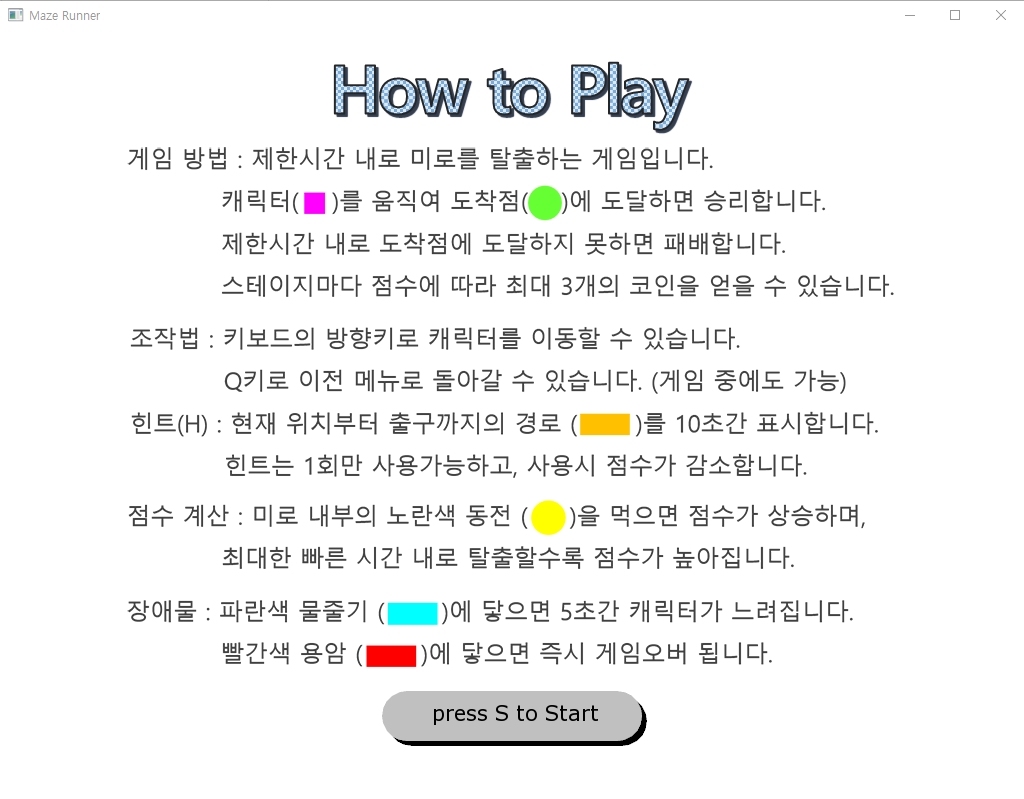
필드에 동전을 임의의 위치에 3개씩 배치하고, 이것을 먹으면 점수를 올릴 수 있고, 빠르게 클리어 할수록 높은 점수를 부여한 다음 점수에 따라 획득 코인량을 결정해 일종의 도전과제 같은 시스템을 구현하였다.

클리어 한 스테이지 수, 각 스테이지별 최단시간, 최대 획득 코인량을 세이브 파일에 저장하고, 프로그램 실행시 그것을 로드함으로써 자동 세이브로드 시스템을 구현하였다.

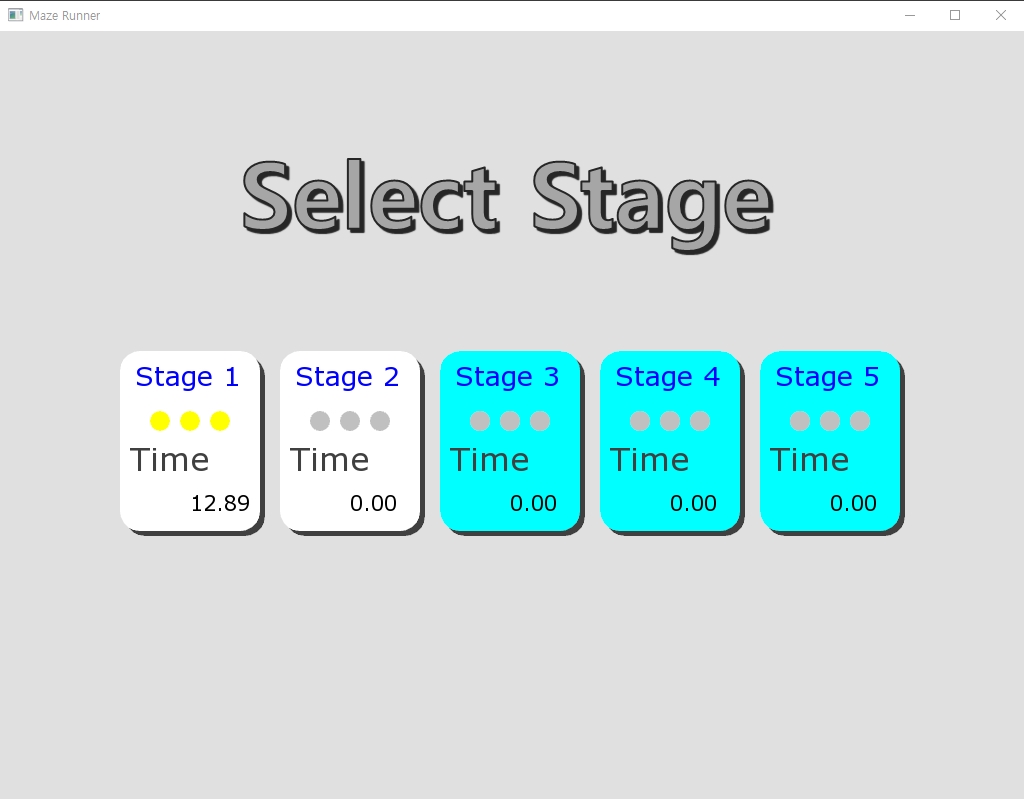
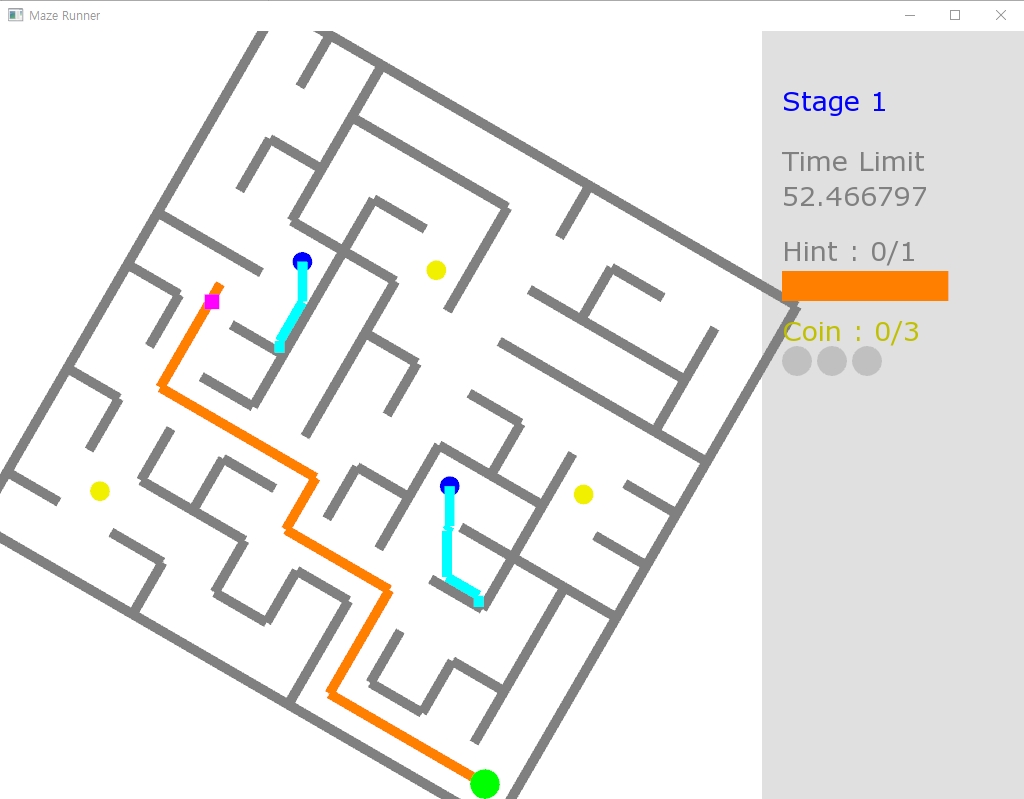
시작화면, 게임설명 화면, 스테이지 선택창에 이미지를 로드하여 화면을 다채롭게 구성하였다.

한 화면에서 다른 화면으로 넘어갈 때, 마우스 버튼 클릭과 키보드 키 입력 방식 모두를 지원함으로써 게임 플레이의 편의성을 높였다.

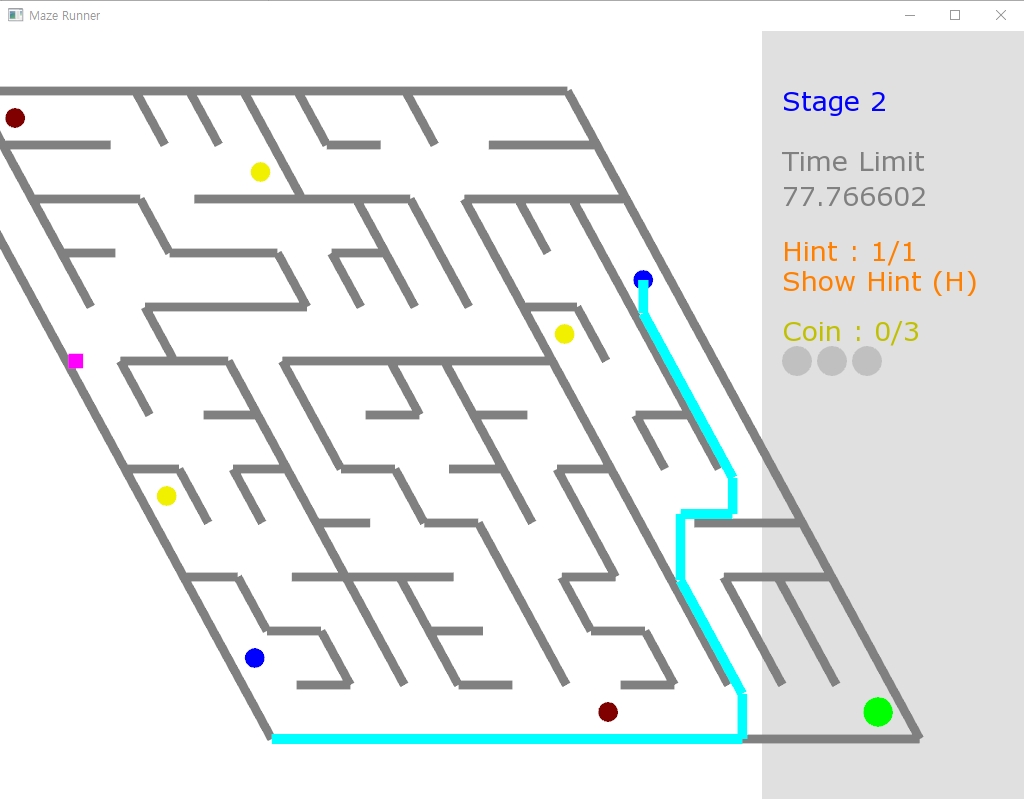
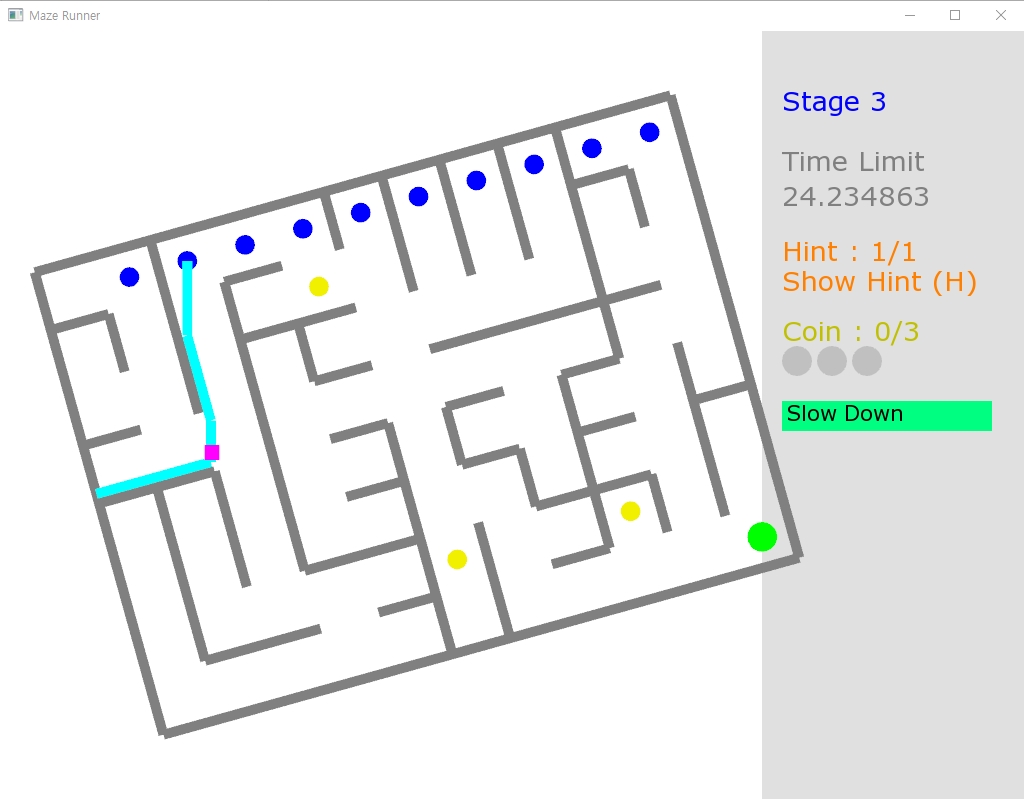
**9. 프로젝트 실행결과 캡쳐**

시작 화면

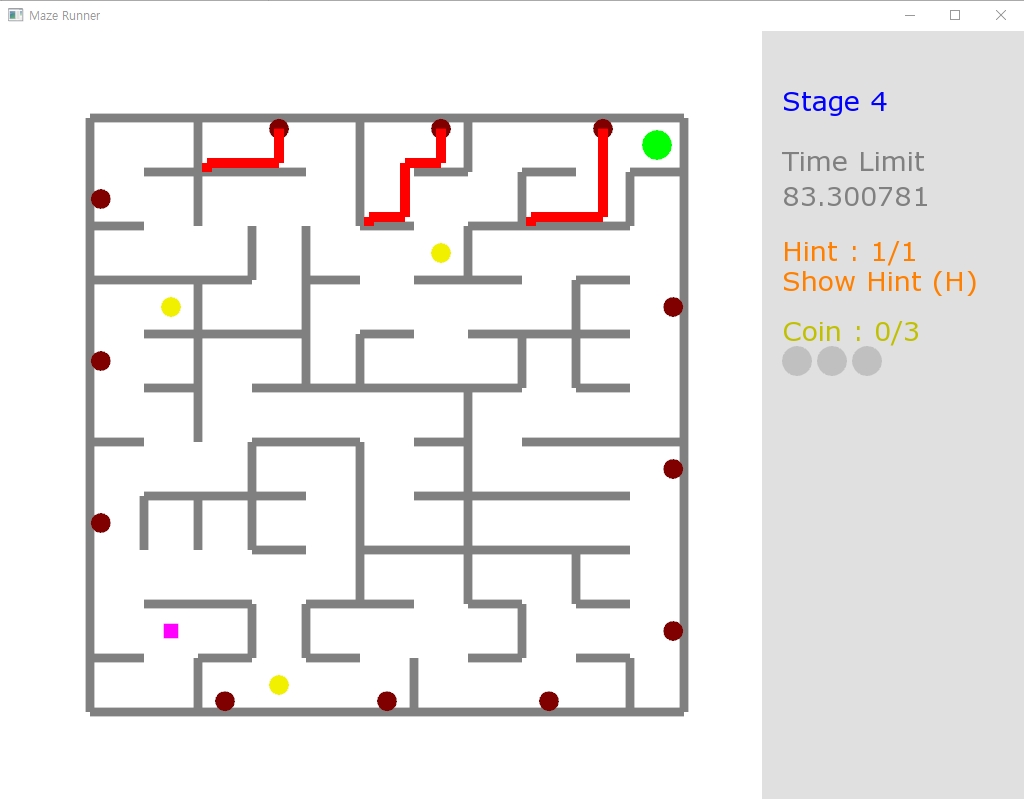
게임설명 화면

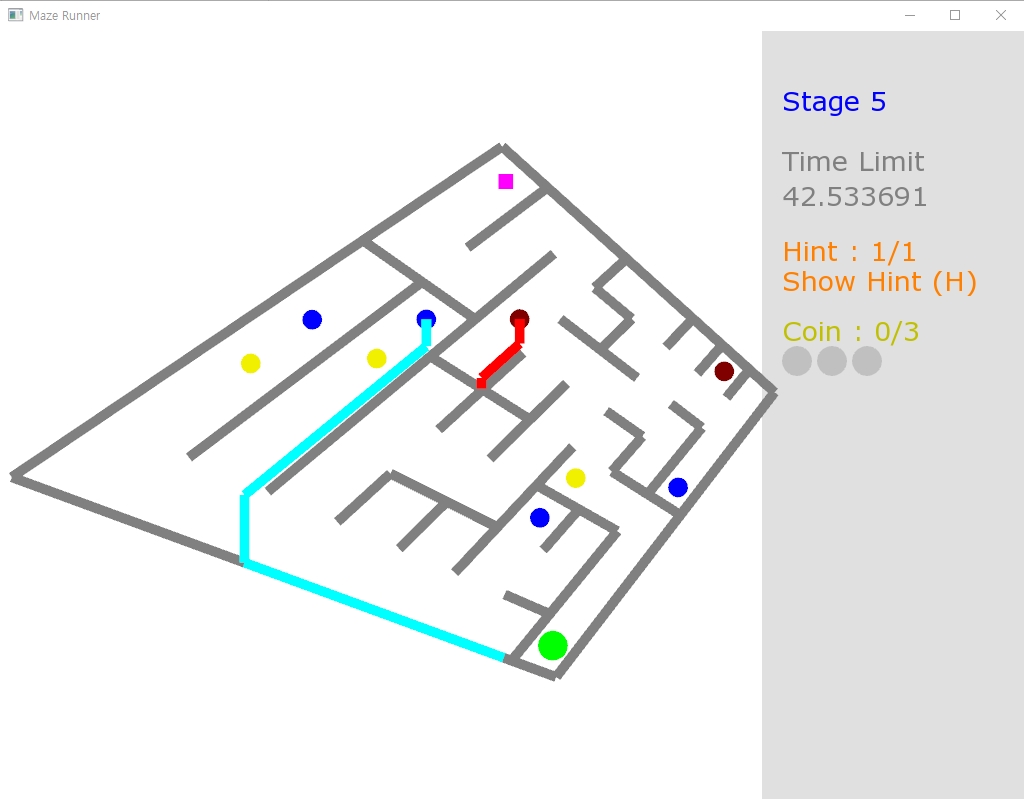
스테이지 선택창 (1스테이지까지 클리어한 상태)

1스테이지 (힌트를 쓴 모습)

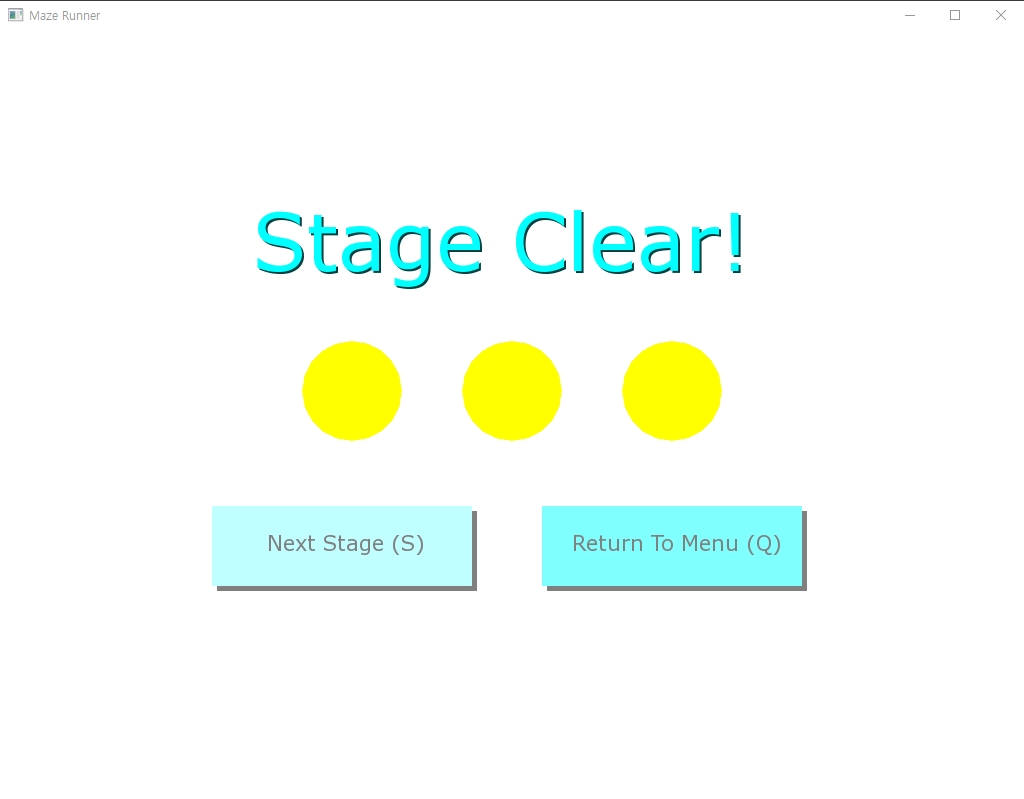
2스테이지

3스테이지 (둔화에 걸린 모습)

4스테이지

5스테이지



게임오버 화면

클리어 화면 (마지막 스테이지 클리어시 Next Stage 버튼이 없음)

**10. 느낀점 및 개선사항**

이번 프로젝트를 진행하면서, 두 프로젝트의 컨셉을 살려 새로운 게임을 만들어 볼려고 했는데, 실제로 구현을 하다보니, 처음에 생각한 구현 난이도보다 훨씬 더 어려워서 원하는 모양이 나올 때까지 시행착오를 여러 번 하게 되어 좀 많이 애를 먹은 것 같다.

특히, 물줄기-미로벽 사이의 충돌 판정을 계산할 때, 단순하게 미로 벽들을 한 칸짜리 선분들의 집합으로 구현하면 (=MArr[Stage].Orig 같은 형식) 물줄기가 벽을 타고 내려가면서 선분의 연결지점에서 벽을 뚫고 내려가 버린다던가, 두 미로 벽의 모서리가 볼록 형상(^)인지 오목 형상인지(v) 구분하지 못해 오목한 지점에 닿으면 벽을 뚫거나, 볼록한 지점에 닿으면 경로계산이 멈춰버린다던가 하는 문제들이 많았었다. 이를 해결하기 위해 몇 시간의 고민 끝에 현재와 같은 이어지는 선분은 하나로 합친 두께를 고려한 내부선/외각선을 활용해 물줄기가 벽을 뚫지 않고, 오목할때는 내부선을 만나서 멈추고, 볼록할 때는 내부선을 만나지 않아 계속 진행할 수 있도록 구현하였다. 이번 프로젝트를 만든 경험으로 다른 프로젝트를 진행할 때 많은 도움이 될 것 같다.

마지막으로, 추가로 개선할 점에 대해서는

물줄기가 미로의 벽을 타고 이동해 시간에 따라 위치가 변하게 만들어서 좀 더 다양한 효과를 낼 수도 있을 것 같고, 물줄기의 종류를 더 늘려서 (ex 닿으면 가속이 되는 물줄기 등) 스테이지를 더욱 다채롭게 구성할 수도 있을 것 같다.

또한, 미로 내부에 캐릭터가 닿으면 눌러지는 스위치를 추가해 스위치를 누를 경우 특정 물줄기가 켜지고 꺼지는 효과를 설정해 게임에 퍼즐요소를 더할 수도 있을 것 같다.