# openGL Maze Project

김동현
2astwis2@gmail.com

# 목차

1. 개요

2. 구현 내용

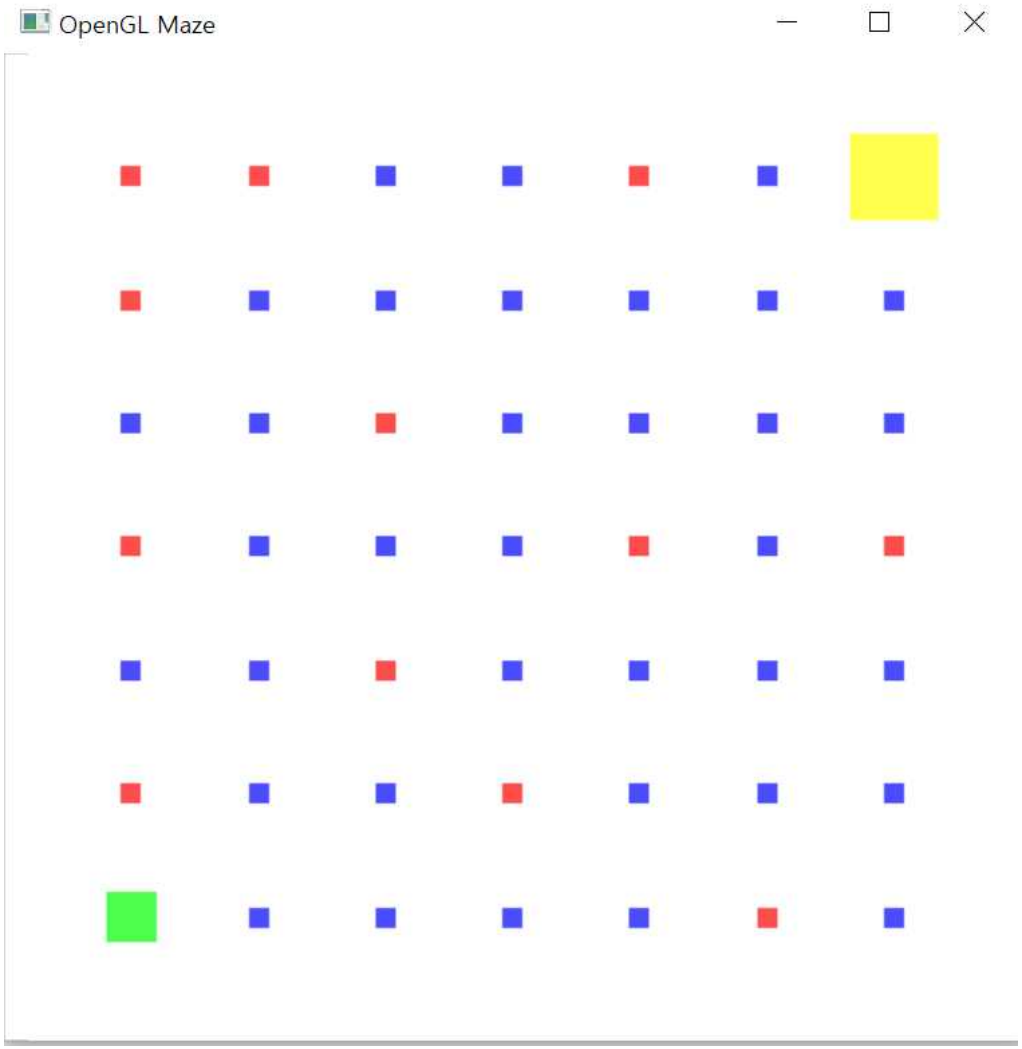
3. 아쉬운 점 및 더 발전시키고 싶은 점

# 1. 개요

## 개발 동기

- 주우석 교수님의 ‘openGL 로 배우는 3차원 컴퓨터 그래픽스’ 책을 보며 공부하던 중, 배운 지식을 활용한 간단한 게임을 만들어 보고 싶었습니다.
- openGL 을 활용한 포트폴리오를 만들고 싶어 도전하게 됐습니다.

# 1. 개요




- 2차원으로 나타나 있는 미로를 푸는 간단한 게임
- 좌 하단 점은 항상 시작점으로 (초록색 큐브)

게임 실행 시 플레이어가 조종할 수 있는 큐브가 존재한다.

- 우 상단 점은 항상 도착점으로 (노란색 큐브)

시작점과 도착점은 항상 고정이다.

- 플레이어가 큐브를 조종하여 도착점에 도착하는 것이 게임의 목표입니다.

# 1. 개요

## 전략

- ‘모델 뷰 변환’ 을 활용하여, 플레이어의 큐브를 이동시키고자 했습니다.
- ‘스택’ 자료구조를 이용하여 길 찾기 알고리즘을 구현해 보고자 했습니다.
- 키보드 입력과 창을 띄우는 것은 glut 의 기능을 활용했습니다.
- 미로는 유저가 원하는 대로 지정할 수 있게 하여, 다양한 게임을 즐길 수 있도록 하였습니다.

  (단, 미로는 오로지 정사각형만 가능하다고 가정)

# 2. 구현 내용

## 전체 플레이 영상 링크

- https://youtu.be/4wHe9rgrPFI
- 총 3가지의 다양한 맵에서 플레이가 가능함을 설명하고자 했습니다.
- 그 외에도 플레이어의 마음대로 맵을 설정할 수 있습니다.

# 2. 구현 내용

## 조작 방법

- 키보드 화살표 키를 이용하여 상하좌우 이동을 할 수 있습니다.
- 빨간색 점은 벽을 뜻하며, 지나갈 수 없습니다.
  - Map 지정 시 “1” 값에 해당합니다.
- 파란색 점은 길을 뜻하며, 길 위로만 지나다닐 수 있습니다.
  - Map 지정 시 “0” 값에 해당합니다.
- Map 지정은 2차 int 배열에 0 과 1 값을 이용하여 지정할 수 있습니다.
- ‘R’ 키를 통해, 처음 상황으로 돌아갈 수 있습니다. ( 리셋 )
- ‘S’ 키를 통해, 자동으로 미로의 해답 중 하나를 찾고 그 경로를 그릴 수 있습니다.

# 2. 구현 내용 – 세부 사항

## 플레이어의 큐브를 움직이는 메서드 – Move

- bool CubeCoor::Move(bool direction, float length, float value)
- 매개변수를 통해 상하좌우 중 방향을 결정
- 애니메이션 구현을 위해 한번 호출 시 조금만 이동하도록 구현
- static 변수를 이용하여 이동량을 검사한다.
  - 이동량을 통해 한 칸을 이동했는 지 확인한다.
  - 한 칸을 이동했다면 반올림을 통해, 위치 좌표를 정수값으로 조정한다.

```cpp
/*
    전역 좌표계 기준, 모델 좌표계의 위치 값을 갱신하는 메서드
    위치 값 갱신에 성공했다면 true 를 리턴하고 더이상 갱신이 없다면 false
    를 리턴한다.

    bool direction
        : 큐브 좌표계가 이동할 방향
        : true => x 축 방향 이동
        : false => y 축 방향 이동

    float length
        : 이동하고자 하는 거리

    int value
        : 한번에 이동하는 값인 이동량.

*/
bool CubeCoor::Move(bool direction, float length, float value) {
    /*
        direction 값을 기준으로 갱신하고자 하는 멤버를 선택한다.
        dirVal 은 갱신하고자 하는 위치 값의 레퍼런스
        direction == true => x 축 방향 위치 갱신
        direction == false => y 축 방향 위치 갱신
    */
    float& dirVal = (direction) ? this->x : this->y;

    /*
        현재 이동한 총량을 static 변수에 기록한다.
    */
    static float totalMove = 0;


    /*
        finish 변수를 이용해, 이동이 끝났는 지 아닌지를 확인한다
        if totalMove > length => 이동이 끝났다.

    if(finish) { // 아직 이동이 끝나지 않았다면,
        /*
            value 값 만큼 이동을 하게 된다.
            이동하는 값을 dirVal 에 더하여 위치 값을 갱신한다.
            또한 이동하는 값의 절대값을 totalMove 에 더하여 이동한 총량을
            체크한다.
            위치 값 갱신에 성공했으므로, true 를 리턴한다.
        */
        dirVal += value;
        totalMove += abs( value );

        return true;
    }
    else { // 더이상 이동할 수 없다면,
        /*
            주어진 방향으로의 이동이 끝났으므로 totalMove 의 값을 초기화
            시킨다.
            dirVal 값은 float 타입이지만, 정확히 각 점에 위치하도록 하기
            위해 반올림값을 넣어 보정한다.
            위치 값이 더이상 갱신되지 않으므로 false 를 리턴한다.
        */
        totalMove = 0;
        dirVal = static_cast<float>(floor(dirVal+0.5f));
        return false;
    }
}

```

# 2. 구현 내용 – 세부 사항

## 큐브 이동 시 애니메이션 구현 - CubeMoving

- void CubeMoving(int value)
- 키 입력에 따라서 한 칸을 이동하는 애니메이션을 실행시키는 메서드
- CubeMoving 은 Move 메서드를 호출하고 그 리턴 값을 이용해 충분히 한 칸을 이동했는 지 확인한다.
- glutTimerFunc 을 이용하여 CubeMoving 을 일정 시간 간격으로 호출해 애니메이션을 구현한다.
- 큐브가 이동 중임을 알리는 변수를 통해, 이동 중에는 추가적인 키 입력을 받지 않도록 한다.

```
/*
    Timer Callback
    Player Cube 가 움직이는 것을 애니메이션으로 표현하기 위해 Timer 를
    이용하였다.
*/
void CubeMoving(int value) {
    /*
        dir : Player 가 움직이려는 방향
            : x축 이동 => true
            : y축 이동 => false
        tip : 이동 시 위치 값을 더할 지 뺄 지를 결정
            : 이번 코드에서는 -0.1 or +0.1 만을 값으로 가진다.
    */
    bool dir;
    float tip = 0;


    switch(value) {
        case GLUT_KEY_UP:
            dir = false; tip = 0.1;
            break;

        case GLUT_KEY_DOWN:
            dir = false; tip = -0.1;
            break;

        case GLUT_KEY_LEFT:
            dir = true; tip = -0.1;
            break;

        case GLUT_KEY_RIGHT:
            dir = true; tip = 0.1;
            break;
    }

    // Player Cube 의 위치 값이 잘 갱신 되었다면
    if(Player.Move(dir, 1.0, tip)) {
        /*
            20ms 후 다시 Timer Callback 을 호출하여
            Player 가 움직이는 애니메이션을 표현한다.
        */
        glutPostRedisplay(); // 다음 이벤트 루프 때 Display Callback 을
        요구
        glutTimerFunc(10, CubeMoving, value);
    }
    // Player Cube 위치 값이 더이상 갱신이 되지 않는다면
    else {
        /*
            키 입력에 의한 이동이 모두 끝났으므로, on_going 값을 false 로
            설정
            이제 다시 키 입력을 받을 수 있도록 한다.
        */
        on_going = false;
        return;
    }
}
```

# 2. 구현 내용 – 세부 사항

## ‘S’ 키 입력 시 자동으로 길을 찾는 알고리즘

- bool findTheAnswer(std::stack<int>& st)
- 주어진 맵과 똑같은 배열을 복사한 후, 경로를 찾으며 기록을 한다.
- 각 점에 대해 적절한 값을 저장한다.
  - 아직 지나가지 않았다 = 0
  - 지나갈 수 없는 길이다 = 1
  - 지나온 경로이다 = 2
- 위 → 오른쪽 → 왼쪽 → 아래 순서로 갈 수 있는 점인지를 확인한다
  - 갈 수 있는 점이라면 경로에 2를 저장한다.
  - 해당 점으로 이동하기 위한 키 값을 스택에 저장한다.

# 2. 구현 내용 – 세부 사항

## ‘S’ 키 입력 시 자동으로 길을 찾는 알고리즘

- 더 이상 나아갈 경로가 없는 경우
  - 스택에 POP 을 가하여, 이전에 적용한 키 값을 확인한다.
  - 이번에 진행한 점에 1을 저장한 후, 키 값을 통해 이전 위치로 돌아간다.
- 우측 상단의 도착점까지 도달할 때까지, 혹은 명령 스택이 결국 다 비어 버릴 때까지 반복한다.
- 스택에 담았던 명령어는 이후, inverse 하여 경로를 그릴 때 재활용한다.

```
173
174
175 /*
176     출발 지점에서 골인 지점까지 도달하는 경로 중 한가지 경로를 찾는
    함수이다.
177     그러한 경로가 존재하면 true 를 반환하고, 경로가 존재하지 않으면 false
    를 리턴한다.
178 */
179 bool findTheAnswer(std::stack<int>& st) {
180     // processedMap 을 복사
181     int checkTheRoute[mapSize][mapSize];
182     for(int i = 0; i < mapSize; i++) {
183         for(int j = 0; j < mapSize; j++)
184             checkTheRoute[i][j] = processedMap[i][j];
185     }
186
187     /*
188         경로 조사는 출발지점인 (0, 0) 에서 시작한다.
189         cx, cy 는 현재 조사중인 좌표를 의미한다.
190     */
191     int cx = 0, cy = 0;
192
193     /*
194         아직 지나가지 않은 점 = 0
195         지나갈 수 없는 점 = 1
196         지나온 길 = 2
197         의 값을 부여한다.
198     */
199     checkTheRoute[cx][cy] = 2; // 출발 지점에서 경로를 시작하므로 2를
    부여한다.
200
201
202
```

```
202     /*
203         골인 지점에 도착할 때까지, 현재 점의 상,하,좌,우 에 갈 수 있는
        길이 있는 지 조사한다.
204         골인 지점이 출발 지점의 우상향 방향에 있기 때문에,
205         위로 가는 것과 오른쪽으로 가는 것에 우선순위를 두었다.
206     */
207     while(cx != mapSize-1 || cy != mapSize-1) {
208         // 위로 갈 수 있다
209         if(cy+1 < mapSize && checkTheRoute[cx][cy+1] == 0) {
210             /*
211                 해당 방향으로 이동 시, 그러한 이동을 지시하는 화살표 키
                값을 스택에 저장한다.
212             */
213             st.push(GLUT_KEY_UP);
214             cy++;
215             checkTheRoute[cx][cy] = 2;
216         }
217
218         // 오른쪽으로 갈 수 있다
219         else if(cx+1 < mapSize && checkTheRoute[cx+1][cy] == 0) {
220             st.push(GLUT_KEY_RIGHT);
221             cx++;
222             checkTheRoute[cx][cy] = 2; // 오른쪽으로 이동
223         }
224
225         // 왼쪽으로 갈 수 있다
226         else if(cx-1 >= 0 && checkTheRoute[cx-1][cy] == 0) {
227             st.push(GLUT_KEY_LEFT);
228             cx--;
229             checkTheRoute[cx][cy] = 2;
230         }
231
232         // 아래로 갈 수 있다
233         else if(checkTheRoute[cx][cy-1] == 0) {
234             st.push(GLUT_KEY_DOWN);
235             cy--;
236             checkTheRoute[cx][cy] = 2;
237         }
238
```

```
238
239         // 사방이 다 막혔다면
240         else {
241             // 현재 좌표는 경로가 될수 없으니, 막혔다는 의미에서 1을 배치
242             checkTheRoute[cx][cy] = 1;
243
244             // 스택을 확인하여 바로 이전에 이동한 방향을 확인한다.
245             int action = st.top();
246             st.pop();
247
248             // action 값을 이용하여 이전 좌표로 되돌아 간다.
249             switch(action) {
250                 case GLUT_KEY_UP:
251                     cy--;
252                     break;
253
254                 case GLUT_KEY_RIGHT:
255                     cx--;
256                     break;
257
258                 case GLUT_KEY_LEFT:
259                     cx++;
260                     break;
261
262                 case GLUT_KEY_DOWN:
263                     cy++;
264                     break;
265
266             }
267             /*
268                 모든 경로가 막혀 다시 처음 출발 지점으로 돌아온 경우.
269                 이동 명령어를 저장한 스택이 비어버렸다.
270                 경로가 없음을 뜻하는 false 를 반환한다.
271             */
272             if(st.empty()) return false;
273         }
274     }
275     return true;
276 }
```

# 3. 아쉬운 점 및 더 발전시키고 싶은 점

## 아쉬운 점

- 모델 뷰 변환, 그 중에서도 모델 변환만을 이용한 점이 아쉽다.
- XY 평면 상의 이동만 존재하며, Z 축에 평행한 이동이 존재하지 않는다.
- 엄연히 따지면, 3차원 게임이 아닌 2차원 게임이다.

## 더 발전시키고 싶은 점

- 카메라 이동을 구현하여, 뷰 변환도 고려해보고 싶다.

감사합니다