Project: The Maze

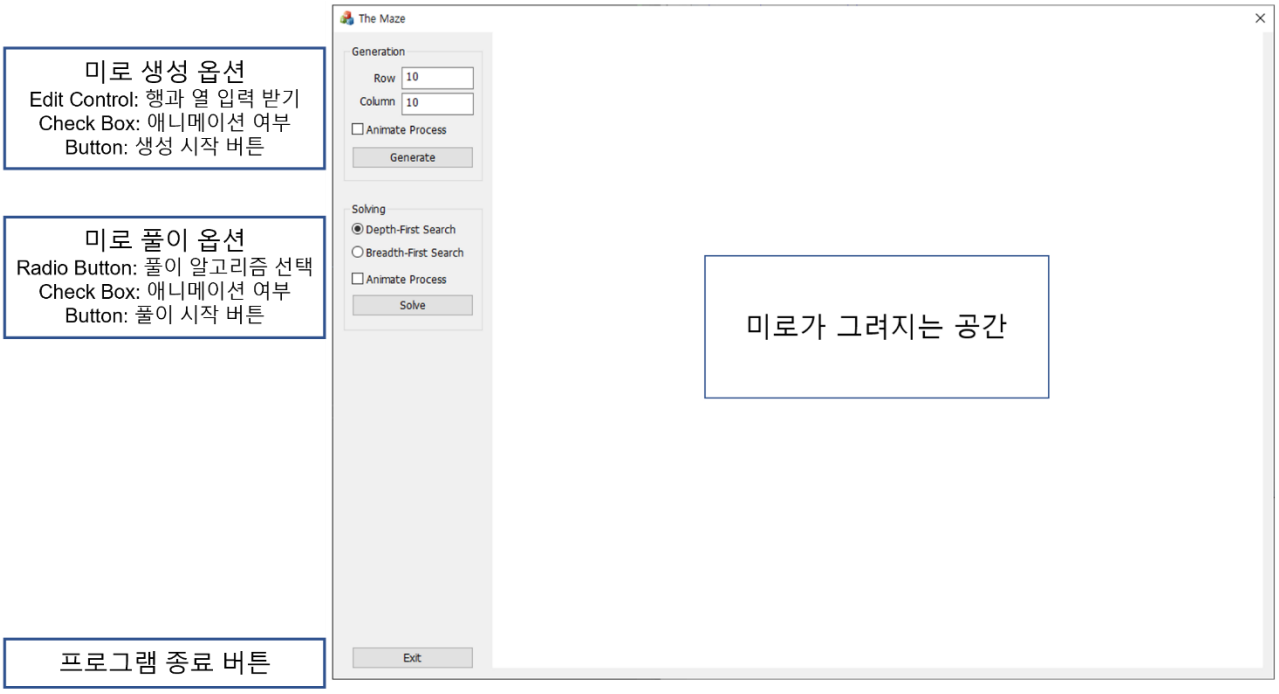
(Due date: 2023. 06. 16. 23:59:59)

[프로젝트 개요]

이번 프로젝트는 미로를 자동으로 생성하고, 생성된 미로의 길을 찾아주는 MFC 기반 윈도우 프로그램을 개발하는 것이 목적이다. 프로젝트를 위해 수행해야 하는 과업은 크게 (1) 사용자 인터페이스 (2) 미로 생성, (3) 사용자 인터랙션, (4) 미로 풀이, (5) 완성도 높이기, (6) 보고서 작성으로 나뉜다.

(1) 사용자 인터페이스 (User Interface) (배점 15%)

- 사용자 인터페이스의 구성은 아래의 그림과 같으며, 각 부분의 기능은 아래 그림을 참조한다.



- MFC 애플리케이션의 종류는 "대화 상자 기반"으로 만든다.
- 각 컨트롤들의 배치 등이 위와 완전히 동일할 필요는 없으나, 채점이 가능한 수준으로 그림의 컨트롤들이 모두 구분 가능하게 배치되어야 한다. (각 컨트롤들의 구분이 불가능할 경우 감점될 수 있음)
- 행과 열을 입력 받는 Edit Control은 숫자만 입력 가능하도록 해야 한다.
- 체크 박스를 통해 미로의 생성 과정 또는 풀이 과정을 애니메이션으로 표현할지 여부를 입력 받는다.
- Generate 버튼을 누르면 인터페이스의 가운데에 "미로가 그려지는 공간"에 미로가 그려진다.
- 초기에 미로가 생성되지 않았을 때에 "미로가 그려지는 공간"에는 아무것도 그려져 있지 않은 흰 바탕으로 시작한다.
- 미로가 생성된 후에 미로 풀이를 위해 어떤 알고리즘 기반으로 미로를 풀이할 것인지를 2개의 Radio Button으로 선택받는다.
- Solve 버튼을 누르면 이미 그려진 미로에 대한 풀이에 해당하는 **"경로"**와 **"탐색구간"**이 그려져야 한다.
- Exit 버튼을 누르면 프로그램이 종료된다.

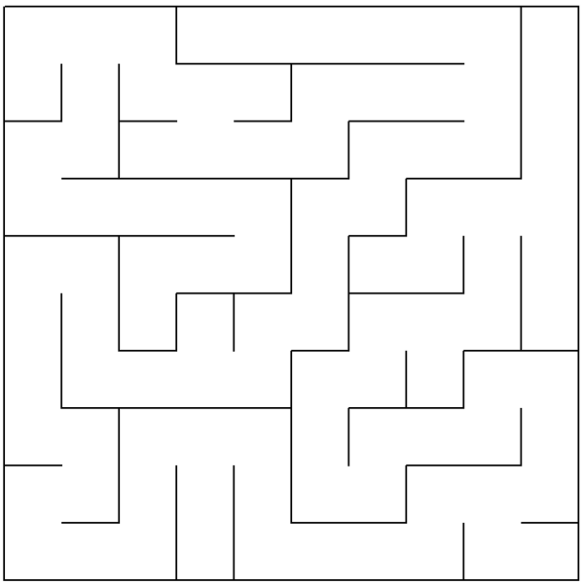
(2) 미로 생성 (Maze Generation) (배점 25%)

- 미로 생성 알고리즘을 선택은 자유이지만, 생성되는 미로는 반드시 다음의 조건을 만족해야 한다

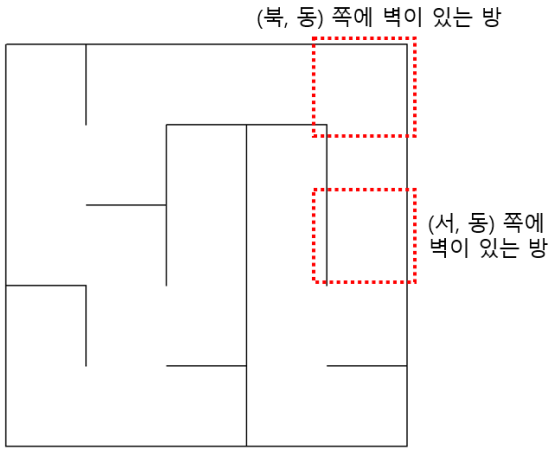
  - **무작위 생성 (Randomized)**: 미로를 생성할 때마다 항상 임의의 새로운 미로가 생성되어야 한다. 이는 rand() 함수를 통해 생성되는 난수를 활용하여 구현 가능하다.
  - **하나의 해답 (Single solution)**: 특정한 출발지에서 목적지까지는 반드시 하나의 경로만이 존재해야 한다.
  - **완전 연결 (Fully connected)**: 미로에 존재하는 모든 방 간에는 경로가 존재해야 한다. 즉, 미로 상에 완전히 막힌 방 혹은 접근 불가능한 영역은 존재하지 않아야 한다.

- **[구현 힌트]** 사용해볼 수 있는 미로 생성 알고리즘의 종류와 그 구현 방법 및 사용되는 자료구조를 알기 위해서는 다음의 웹사이트를 참고할 수 있다.
https://visualize-it.github.io/maze_generation/simulation.html

- 위 세 조건을 만족하는 미로의 예시는 아래와 같으며, 생성에 활용한 알고리즘은 DFS 기반 방식이다.



10 x 10 미로 예시



5 x 5 미로 예시

- 특정한 출발지와 목적지를 가정하지 않고 미로를 만들도록 한다. (출발지 및 목적지는 사용자가 직접 인터페이스 상에서 선택할 것이다.)

- 행과 열이 최소 4부터 최대 30인 미로까지 생성 가능해야 한다.

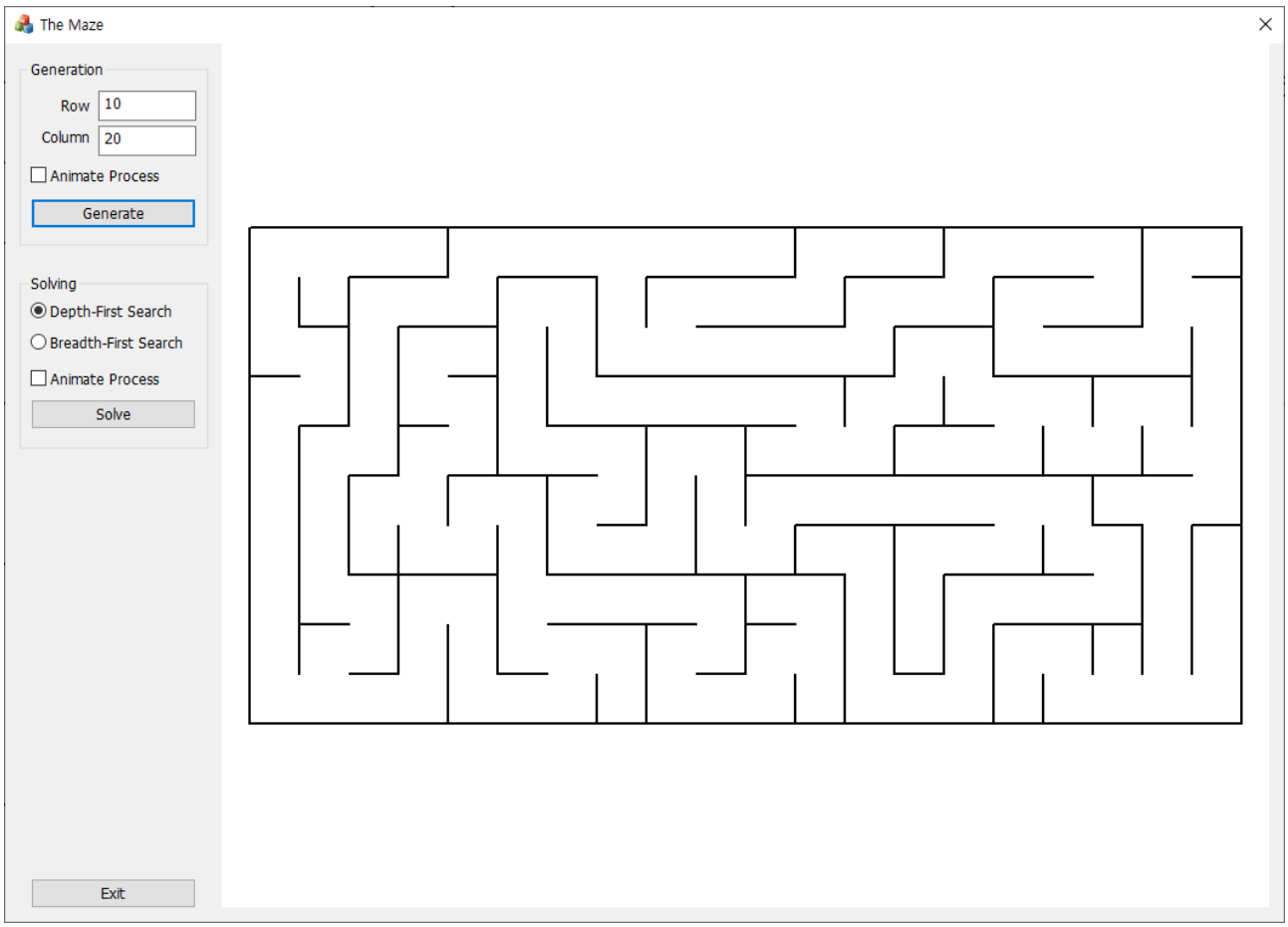
- 정사각형 미로 뿐 아니라 행과 열의 값이 다른 직사각형 미로도 생성가능해야 한다.

- 미로의 각 방들은 반드시 정사각형이어야 하며, 방은 동, 서, 남, 북 방향으로 벽을 가질 수 있다.

- 초기에 미로의 모든 방들은 사방에 벽을 가지고 있다가, 미로 생성 알고리즘이 수행됨에 따라 점차 벽이 없어지는 방식으로 미로 생성이 진행된다.

- 미로 생성 알고리즘을 수행할 시작 지점에 해당하는 방은 매번 임의로 결정되도록 한다.

- 다음의 그림은 직사각형 미로가 생성된 인터페이스의 모습이다.



- 각 방의 변의 길이는 미로의 행과 열의 개수 및 "미로를 그리는 공간"의 가로와 세로 길이를 고려하여 최대로 잡을 수 있는 값을 선정한다.

- 미로는 "미로를 그리는 공간" 상에서 항상 수평, 수직 방향으로 가운데 정렬된 채로 그려져야 한다.

- 미로 생성 과정을 애니메이션으로 구현한다.
  - Animate Process 체크박스가 체크해제 된 채로 Generate 버튼을 눌러 미로를 생성할 경우, 아무런 애니메이션 효과 없이 위와 같이 미로 생성 결과만을 그리면 된다.
  - Animate Process 체크박스가 체크된 채로 Generate 버튼을 누를 경우는, 따로 안내되는 동영상에서의 애니메이션 효과를 모방하여 구현한다.
  - **[구현 힌트]** 애니메이션은 사실 생성 알고리즘이 수행되는 과정 속에서 미로 그림을 지속적으로 그려주는 것이다. 따라서 알고리즘 내에서 미로에 변화가 일어날 때마다 미로 그림을 다시 그리는 함수를 호출하되, 약간의 지연시간을 주는 **Sleep 함수를 이용하여 그 속도를 조절**할 수 있다.

- 옵션을 바꿔가며 Generate 버튼을 여러 번 눌러 새롭게 미로를 생성하더라도 프로그램이 이상 없이 돌아가야 한다.

(3) 사용자 인터랙션 (User Interaction) (배점 20%)

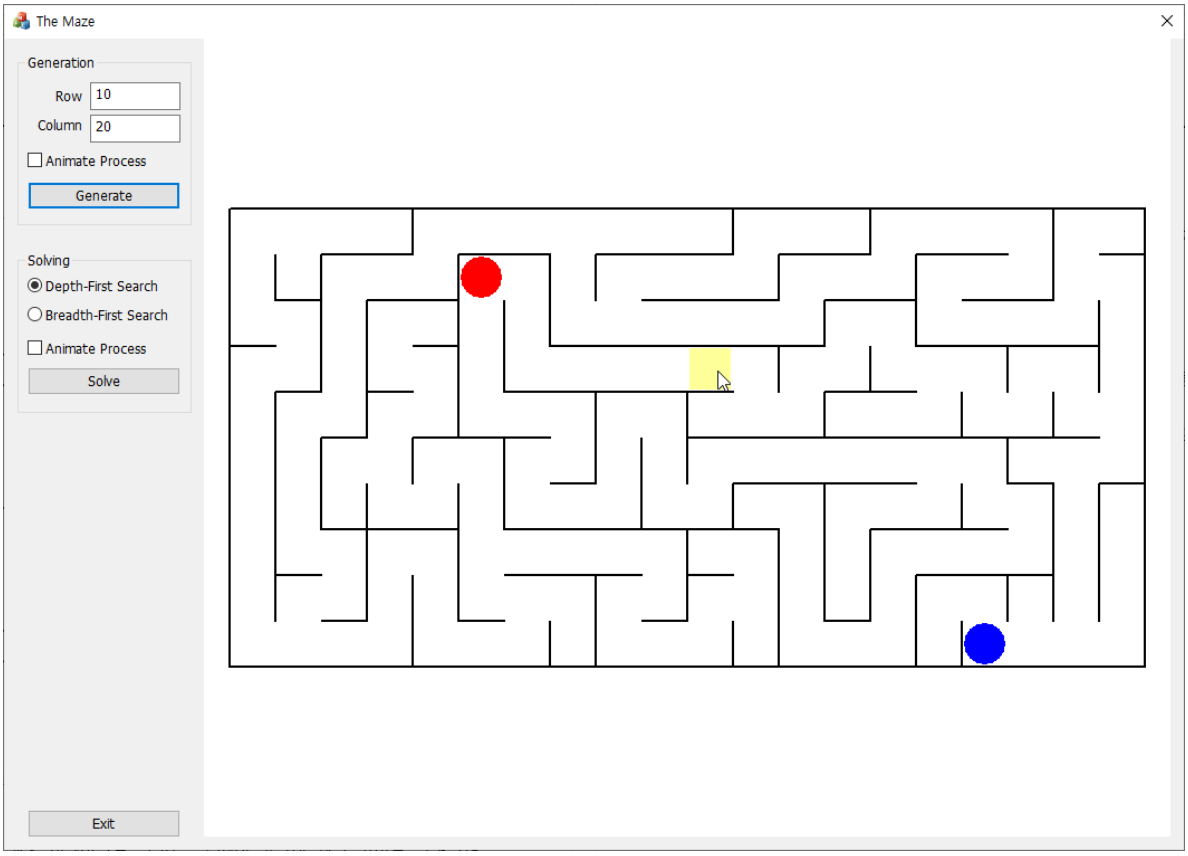
- 마우스 호버링 (Mouse Hovering)을 구현한다.

  - "미로가 그려지는 공간"에서 마우스가 움직이다가 미로의 각 방에 마우스 커서가 올라가면, 해당 방의 배경 색을 하얀색이 아닌 다른 색으로 강조해준다.
  - 마우스 커서가 방을 빠져나갔을 경우, 그 방은 다시 기본 배경 색인 하얀 색으로 복구한다.
  - **[구현 힌트]** 이 구현은 그림을 그리는 Control 내에서 마우스가 움직일 때마다 OS로부터 보내지는 **WM_MOUSEMOVE 메시지를 처리하는 Handler (함수)**에서 마우스 커서의 위치를 받아서 그 위치를 기반으로 현재 마우스가 어느 방에 위치해 있는지를 판단한 뒤 그에 맞게 그림을 그려야 한다.

- 출발지, 목적지 선택 기능을 구현한다.

  - 출발지 선택: 마우스 커서가 올려진 방을 **마우스 좌클릭**하면 해당 방은 출발지로 설정되며, 해당 방에는 빨간 원이 그려진다. **(WM_LBUTTONDOWN 메시지 핸들링 필요)**
  - 목적지 선택: 마우스 커서가 올려진 방을 **마우스 우클릭**하면 해당 방은 도착지로 설정되며, 해당 방에는 파란 원이 그려진다. **(WM_RBUTTONDOWN 메시지 핸들링 필요)**
  - 이미 출발지, 목적지가 선택된 상황에서도 다른 방을 클릭하면 출발지, 목적지가 갱신되고 그에 맞게 그림도 다시 그려져야 한다.

- 사용자 인터랙션의 동작 과정은 따로 안내되는 동영상을 통해 더욱 자세히 확인 가능하며, 그 결과에 해당하는 그림은 아래와 같다.



(4) 미로 풀이 (Maze Solving) (배점 25%)

- 미로의 풀이를 위한 알고리즘의 종류는 매우 다양하지만, 이번 프로젝트에서는 깊이 우선 탐색 (Depth-First Search)과 너비 우선 탐색 (Breadth-First Search)에 기반한 풀이 방식을 구현한다.

- **[구현 힌트]** DFS 및 BFS 기반의 미로 풀이 알고리즘과 그에 필요한 자료구조의 파악을 위해 다음의 웹 사이트를 참고할 수 있다.
https://makeschool.org/mediabook/oa/tutorials/trees-and-mazes/solving-the-maze/

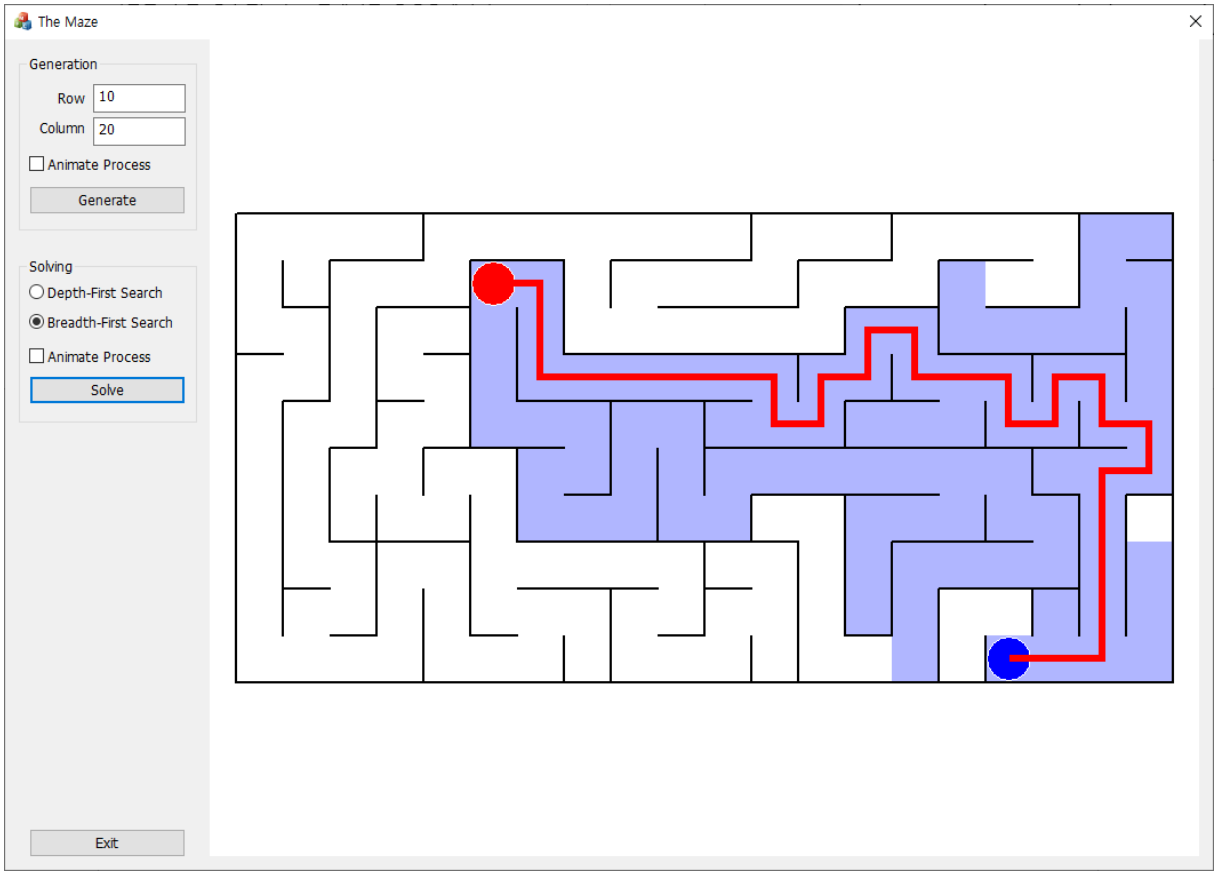
- 사용자 인터페이스 상의 Radio Button에서 선택된 방식으로 미로를 풀이한다.

- 옵션을 바꿔가며 Solve 버튼을 여러 번 눌러 새롭게 미로를 풀이하더라도 프로그램이 이상 없이 돌아가야 한다.

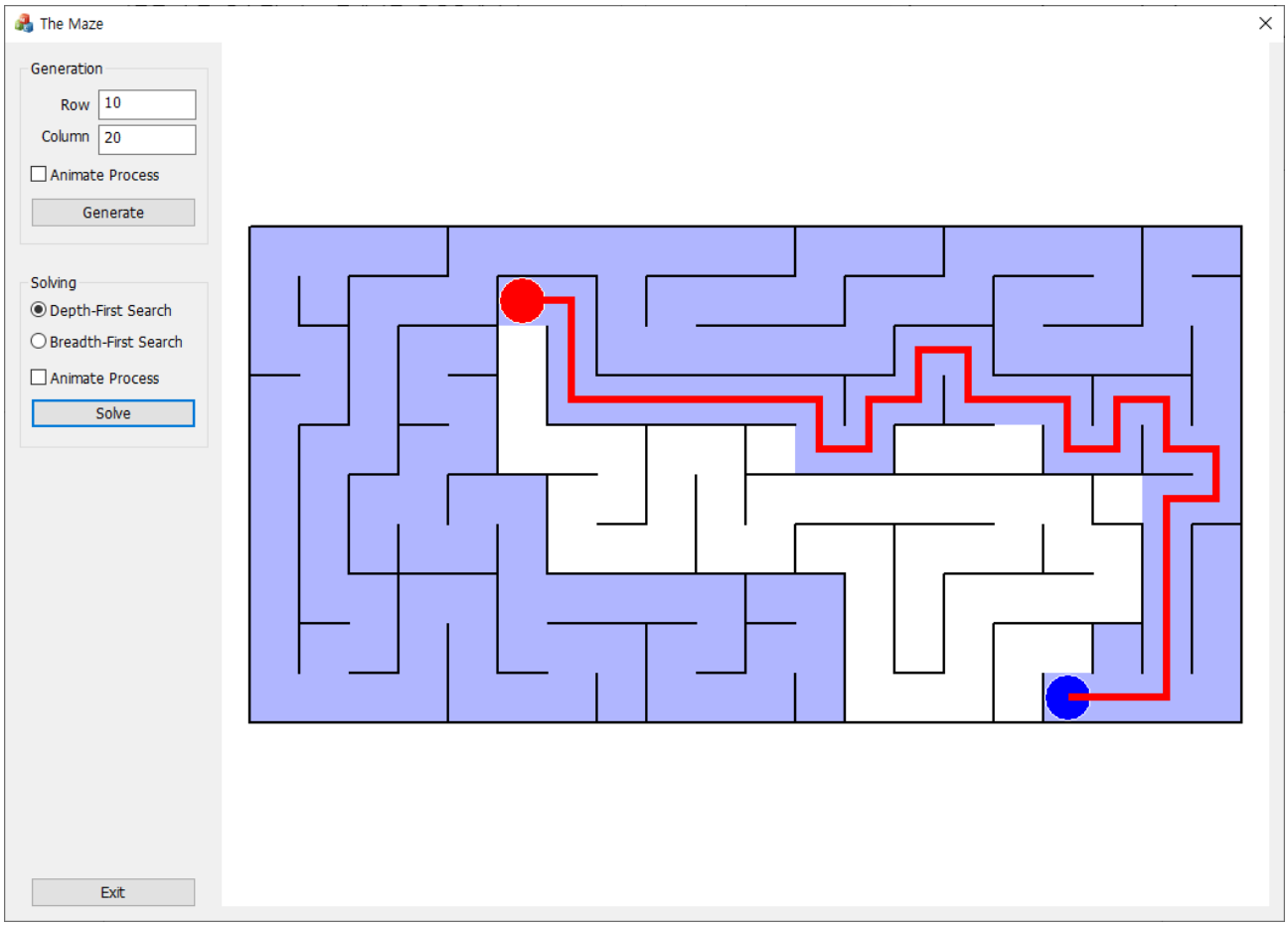
- 두 방식을 통해 나온 경로는 동일할 것이다. 다만, 탐색 구간에서 차이가 있을 것이므로, 반드시 미로 풀이 결과로 **경로와 탐색 구간을 모두 그려주어야** 한다. 여기서의 탐색 구간이란 알고리즘이 수행되면서 방문한 모든 방들을 의미한다.

- 탐색 구간 및 경로는 아래의 그림과 같이 나타내며, 미로 풀이의 경우도 마찬가지로 애니메이션의 경우 따로 안내되는 동영상에서의 애니메이션 효과를 모방하여 구현한다.

<BFS 방식 미로 풀이의 예시>



<DFS 방식 미로 풀이의 예시>



(5) 완성도 높이기 (배점 5%)

- 더블 버퍼링 (Double Buffering) 기법을 구현한다.

  - 미로 생성 혹은 풀이 그림을 그릴 때에, 특히 애니메이션 혹은 사용자 인터랙션에 대응하기 위해 그림을 지속적으로 다시 그려주어야 상황에서는 화면이 깜빡거리는 Flickering 현상이 발생하게 된다. 이러한 현상을 해결하기 위해서는 더블 버퍼링 기법이 있다.

  - **[구현 힌트]** 더블 버퍼링 기법의 개념은 아래의 웹사이트를 참고하도록 한다.
    https://en.wikipedia.org/wiki/Multiple_buffering

(6) 보고서 작성 (배점 10%)

- 보고서 작성 시 Introduction / Flowchart / Algorithm / 결과화면 / 고찰 의 구성에 따라 작성한다.
- 영문 또는 한글로 작성한다.
- 반드시 PDF로 제출한다.
- 표지를 제외하고 10페이지 이내로 작성
- 소스코드는 보고서에 포함하지 않는다.

(7) 제출 기한 및 제출 방법

- 제출 기한: 2023년 6월 16일 23:59:59 까지 제출
- 제출 방법: FTP Upload (Klas 과제 제출 X)
    - Address : ftp://223.194.8.1:1321
    - username : IPSL_OBJ
    - password : ipslobj_2023
    - 프로젝트, 솔루션, 소스파일 및 보고서 포함
    - Debug/Release 폴더 삭제 후 제출
- 제출 형식
    - 설계반_실습반_학번_이름.zip
        - 예) 설계1반 수강, 실습 A반: 1_A_학번_이름.zip
        - 예) 설계 수강, 실습 미수강: 2_0_학번_이름.zip
        - 예) 설계 미 수강, 실습 C반: 0_C_학번_이름.zip
    - 채점 시 코드를 수정해야 하는 일이 없도록 한다.
    - 과제 수정하여 업로드 시 버전 명시: 설계반_실습반_학번_이름_verX.zip

※ 구현 시 제한사항 및 유의 사항

- STL 컨테이너(vector, queue, stack, deque 등) 사용을 금한다.
- 본 프로젝트 제안서에 명시된 내용 이외의 내용에 대해서는 자유롭게 구현한다.
- 발생 가능한 모든 문제에 대하여 예외처리를 할 수 있도록 한다.
- 프로그램 구조에 대한 디자인이 최대한 간결하도록 고려하여 설계한다.
- 프로그램 내에서 할당된 모든 메모리는 프로그램이 종료될 때 모두 해제되어야 한다.