**컴실1 프로젝트 (Pac-Man) 보고서**

20201597 신동준

**프로젝트 목표**

|  |
| --- |
| 저는 실습 때 했던 maze를 활용하고, 응용해서 1980년에 출시된 고전게임 Pac-Man 게임을 만들어보려고 합니다. 크게 3개의 오브젝트가 있는데, user가 플레이하는 주인공 pac-man, 유저가 목적지에 못가도록 방해하는 ghost, 유저가 최종적으로 먹어야 하는 fruit가 있습니다.  저는 각 스테이지 별로 다른 미로를 불러와서, 유저가 클리어할 수 있도록 할 것이며, 죽지 않고, 클리어한 유저는 자신의 점수를 txt파일에 저장할 수 있습니다.  또한, 유령은 특정한 알고리즘을 바탕으로 미로를 이동하며, 유저를 찾아다니게 됩니다. |

**각 주요 변수 설명**

|  |
| --- |
| 아래의 변수는 화면을 제어하며, 어떤화면을 표시해야 될지 bool type으로 전달합니다.  출력될 화면은 true, 그렇지 않은 화면은 false로 표시됩니다.  bool main\_wdw = true; // main화면  bool game\_start = false; // 인게임 화면  bool clear\_wdw = false; // 클리어 화면  bool ending\_wdw = false; // 엔딩 화면  bool rank\_wdw = false; // 랭크 화면  bool death\_wdw = false; // 죽었을 때 화면 |

|  |
| --- |
| 아래의 변수는 각각 폰트, 음악, 이미지를 포함합니다.  ofTrueTypeFont OEB; // 폰트  ofTrueTypeFont OEB\_1; // 작은 폰트  ofSoundPlayer begin; // 계속 반복되는 배경음악  ofSoundPlayer death; // 죽었을 때 사운드  ofSoundPlayer eatcoin; // 쿠키를 먹을 때 사운드  ofSoundPlayer eatfruit; // 체리를 먹을 때 사운드  ofImage Player\_left; // 플레이어의 입이 왼쪽  ofImage Player\_right; // 플레이어의 입이 오른쪽  ofImage Player\_up; // 플레이어의 입이 윗쪽  ofImage Player\_down; // 플레이어의 입이 아랫쪽  ofImage Ghost; // 유령 이미지  ofImage Fruit; // 체리 이미지  ofImage Title; // 제목 이미지 |

|  |
| --- |
| 아래의 변수들은 각각 유저, 유령, 체리의 위치, 크기, 속도 등을 저장하게 됩니다.  char rotate = 'r'; // player의 방향 저장  int Player\_x, Player\_y; // player의 좌표  int player\_size = 40; // player의 크기  int player\_speed = 2; // player의 속도  int ghost\_speed = 20; // ghost의 속도  int cookie\_size = 3; // 쿠키의 크기    int fruit\_x, fruit\_y; // 체리의 좌표  int Ghost\_x = shift + length \* 10 + 10; // 유령의 x좌표  int Ghost\_y = shift + length \* 10 + 10; // 유령의 y좌표  int Ghost\_pos\_x = 19; // 유령이 위치한 x 칸  int Ghost\_pos\_y = 19; // 유령이 위치한 y 칸 |

|  |
| --- |
| 아래의 변수들은 점수의 출력과 저장에 사용됩니다.  int score = 0; // 점수 저장  string score\_s; // 점수 int to string  priority\_queue<int> pq; // 랭킹을 입력받는 max heap |

|  |
| --- |
| 아래의 변수들은 미로의 크기와 높이, 넓이 등 미로의 정보를 저장하고, BFS를 위한 탐색 정보들을 저장합니다.  int length = 60; // 미로 한칸의 크기  int shift = 50; // 미로 전체적으로 shift  int HEIGHT; // 미로 높이  int WIDTH; // 미로 너비  int maze\_col, maze\_row; // bfs에 쓰이는 좌표  vector<string> input; // 미로 입력  vector<vector<int>> visited; // bfs의 방문 장소 확인용  queue<pair<int, int>> q; // bfs의 queue  int vec[4]; // 갈 수 있는 곳 4방향 탐색을 위한 array  int stage = 1; // game 스테이지  int MAX\_stage = 4; // 최대 스테이지 |

**사용된 자료구조 및 알고리즘**

|  |
| --- |
| **Stack(vector), queue**  C++에 있는 vector, queue stl을 사용하였습니다.  maz파일을 읽어서 미로를 저장하기 위해서 2차원 벡터를 사용하였습니다. 벡터 자료구조는 stack과 같이 LIFO구조를 가집니다.  Bfs를 실행할 때 queue 자료구조를 사용하였습니다. Queue에 먼저 들어간(depth가 작은) 곳을 먼저 탐색하는 것이 Breadth first search에 핵심이기 때문에 queue를 사용하였고, queue안에는 pair자료구조가 push되게 됩니다. Queue는 FIFO 구조를 가집니다.  **Max heap(priority queue)**  랭크를 높은 순서대로 효율적으로 정렬하고 가져오기 위해 max heap 자료구조를 선택하였습니다. 저는 넣을 때 빠르게 넣기위에서 insert를 O(1)안에 할 수있게 unsorted하게 txt파일에 저장하였고, 가져와서 정렬을 빠르게 하기 위해 max heap에 점수를 넣었습니다. 최종적으로 n개의 점수를 sort하는데 O(nlogn) 시간복잡도로 정렬할 수 있었습니다. |

|  |
| --- |
| **Eller’s algorithm**  미로를 생성하는 과정에서 엘러의 알고리즘을 사용하였습니다. 모든 벽이 있는 상태에서 각 칸을 다른 set으로 설정하고, 왼쪽이나 아랫쪽 중 랜덤하게 벽을 부수고 같은 set으로 설정해줍니다. 이를 반복하는데 같은 set이면 깨지못하고, 다른 set이면 깰 수 있도록하여 완전 미로를 만드는 알고리즘 입니다.  **BFS(Breadth First Search)**  유령이 미로를 탐색하며, 유저를 방해하러 다니기 위해 미로의 벽은 넘을 수 없고, 미로를 효율적으로 탐색하며 나아가야하기 때문에 BFS 알고리즘을 택하였습니다. BFS는 queue에 갈 수 있는 주변의 모든 공간을 넣고 하나씩 pop하면서 탐색하는 알고리즘입니다. Depth보다 breadth를 우선으로 하여 최초점 주변을 먼저 탐색하게 됩니다. |

**각 함수 설명**

|  |
| --- |
| 아래 함수는 .maz 파일을 스테이지 별로 읽어오는 역할을 합니다.  Input vector, stage등 전역변수에 접근하게 됩니다.  [ Input : X, output : X ]  void readFile(); // maz 파일을 읽어옴 |

|  |
| --- |
| 아래 함수는 rank.txt 파일을 읽어와서 전역변수 max heap에 넣어줍니다.  [ Input : X, output : X ]  void readRank(); // rank.txt 파일을 읽어옴 |

|  |
| --- |
| 아래의 함수는 user가 갈 수있는지 확인하고, 갈 수 있을 때 true를 리턴합니다.  [ Input : 유저좌표, output : true/false ]  bool check\_go(int delta\_x, int delta\_y); // user가 갈 수 있는지 확인 |

|  |
| --- |
| 아래의 함수는 발 밑에 쿠키가 있는지 확인하고, 쿠키가 있으면 먹고 점수를 올려줍니다.  [ Input : X, output : X ]  void check\_eat(); // 먹을 쿠키가 있는지 확인 |

|  |
| --- |
| 아래의 함수는 발 밑에 체리가 있는지 확인하고, 체리가 있으면 클리어 화면으로 전환합니다.  [ Input : X, output : X ]  void check\_clear(); // 체리를 먹었는지 확인 |

|  |
| --- |
| 아래의 함수는 유령과 충돌을 확인하고, 충돌했으면 사망화면으로 전환합니다.  [ Input : X, output : X ]  void check\_death(); // ghost와 충돌했는지 확인 |

|  |
| --- |
| 아래의 함수는 동적할당을 해제합니다.  [ Input : X, output : X ]  void freeMemory(); // 동적할당 해제 |

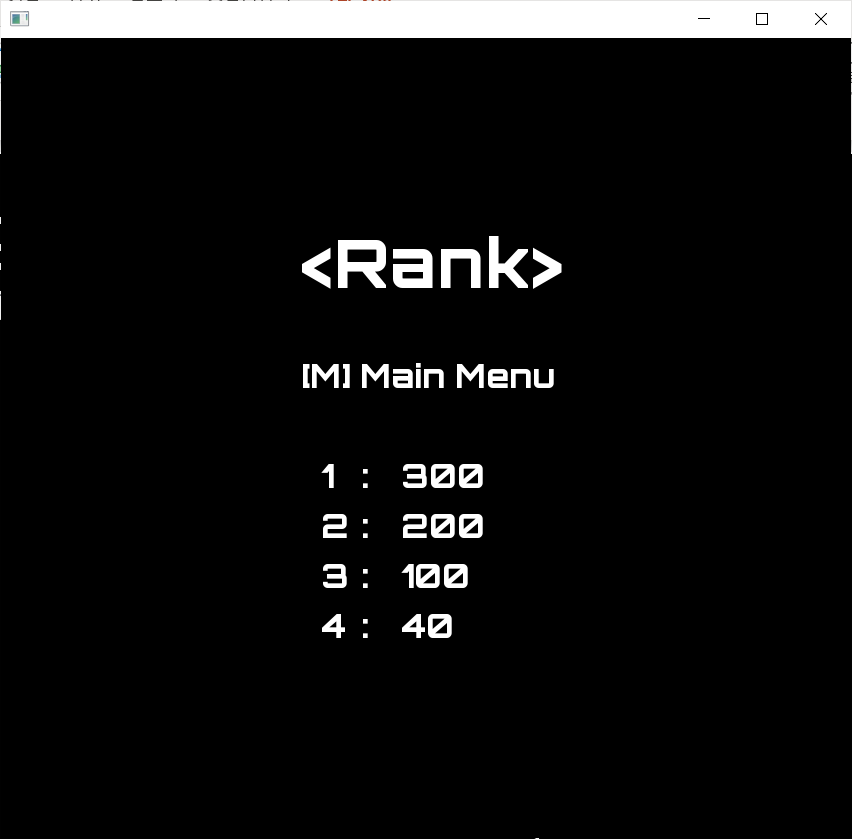
|  |
| --- |
| 아래 함수는 rank.txt를 불러오고, score를 txt 파일 마지막에 적어줍니다.  [ Input : score, output : X ]  void writeRank(int scr); // rank를 rank.txt에 write |

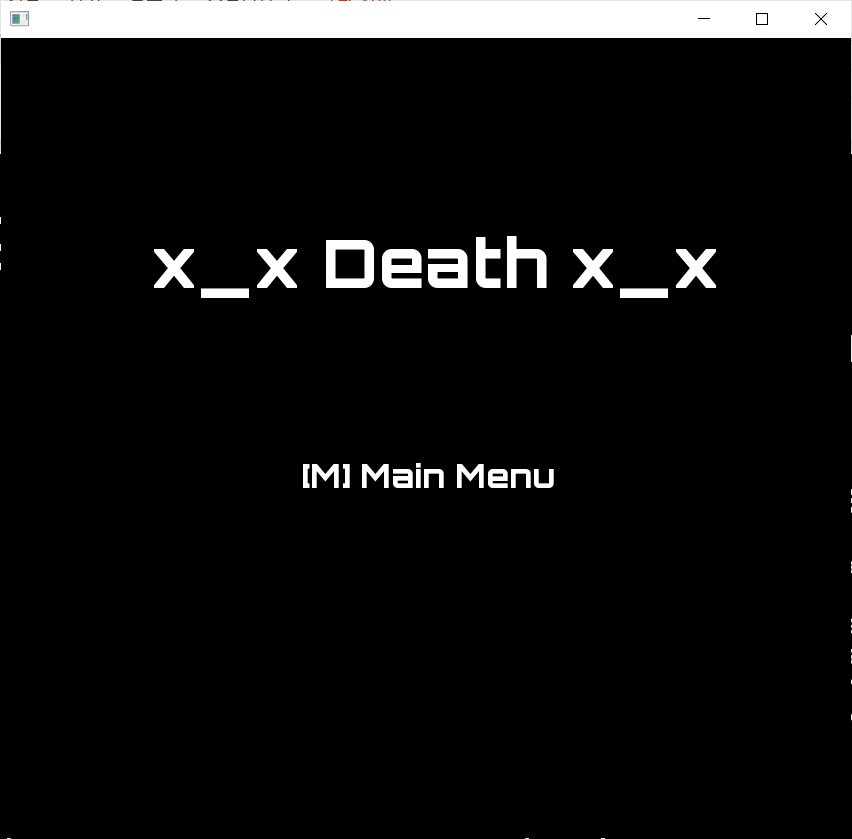
|  |
| --- |
| 아래 함수는 각각 정방향, 역방향으로 BFS를 돌고, 유령이 움직일 순서(visited) 2차원 벡터를 만드는 역할을 합니다.  [ Input : 탐색할 초기 위치, output : X ]  void BFSfe(int maze\_col, int maze\_row); // (1,1)에서 (19,19)로 BFS  void BFSef(int maze\_col, int maze\_row); // (19,19)에서 (1,1)로 BFS |

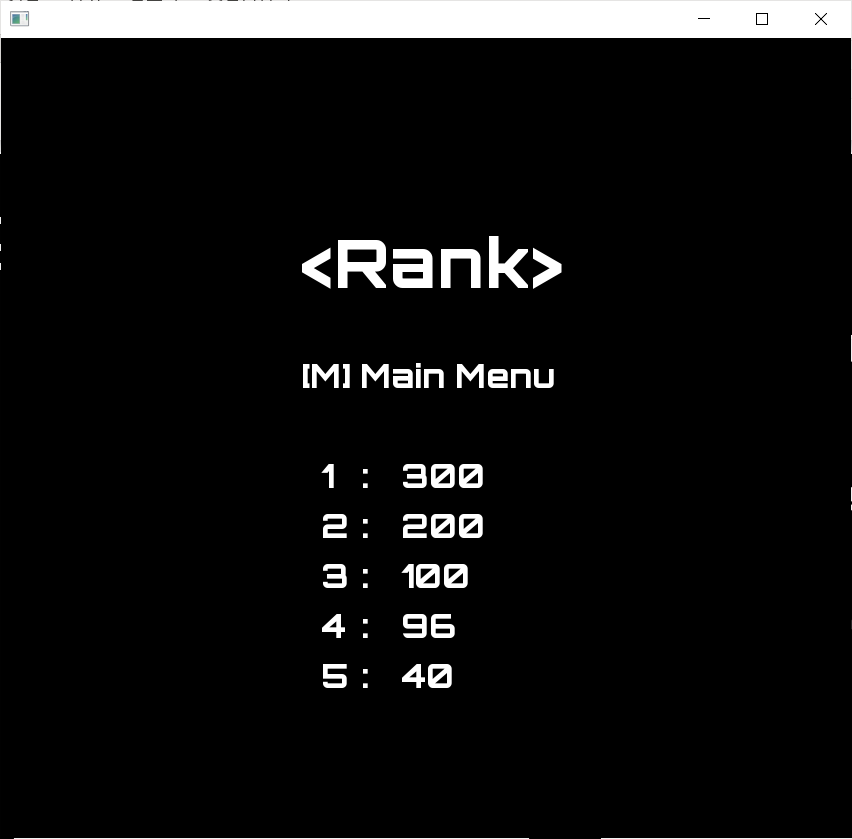
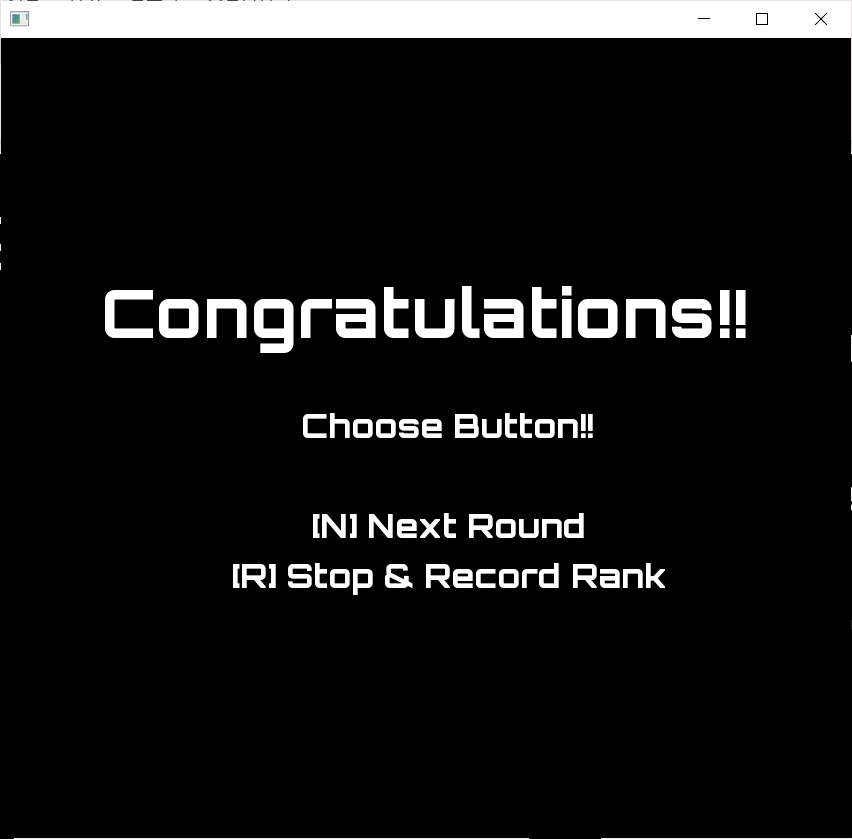
**본인의 창의적인 구현 설명**

|  |
| --- |
| 기존 pac-man 게임에서 다음 라운드에 도전하기 위해서는 지금까지 쌓아온 점수를 날릴 수도 있는 위험성을 넣어서, 재미적인 요소를 더 했습니다. 점수를 얻는 방법은 쿠키를 먹는 것으로 제한하여서, 쿠키를 많이 먹으려고 돌아다니다보면 유령과 부딪힐 확률도 높기에 게임적인 요소를 가미했습니다.  또한, 배경 음악과 이미지 등을 추가해 조금 더 생동감을 느낄 수 있도록 하였습니다.  랭킹 시스템 구현을 위해서 파일을 저장하은 빠르게 unsorted하게 하지만, 불러옴과 동시에 max heap에 넣어서 빠르게 sorting할 수 있도록 구현해봤습니다. |

**프로젝트 실행 결과**







**느낀 점 및 개선 사항**

|  |
| --- |
| 생각했던 것보다 오픈 프레임워크 내에 다양한 기능이 있다는 걸 알게되었고, 신기했다. 프로젝트를 구현하면서 이것저것 필요한 기능들을 찾아보고, 자료구조를 찾아보면서 스스로도 많은 공부가 되었다.  조금 더 게임적인 요소나 난이도적인 면을 개선하고 싶다. 유령을 지금은 한마리만 이동시키는데, n마리의 유령이 무작위 스폰된다던지, 스테이지 별로 아이템 등도 개선하고 싶다. |

**프로젝트 환경**

Visual studio 2019, window10