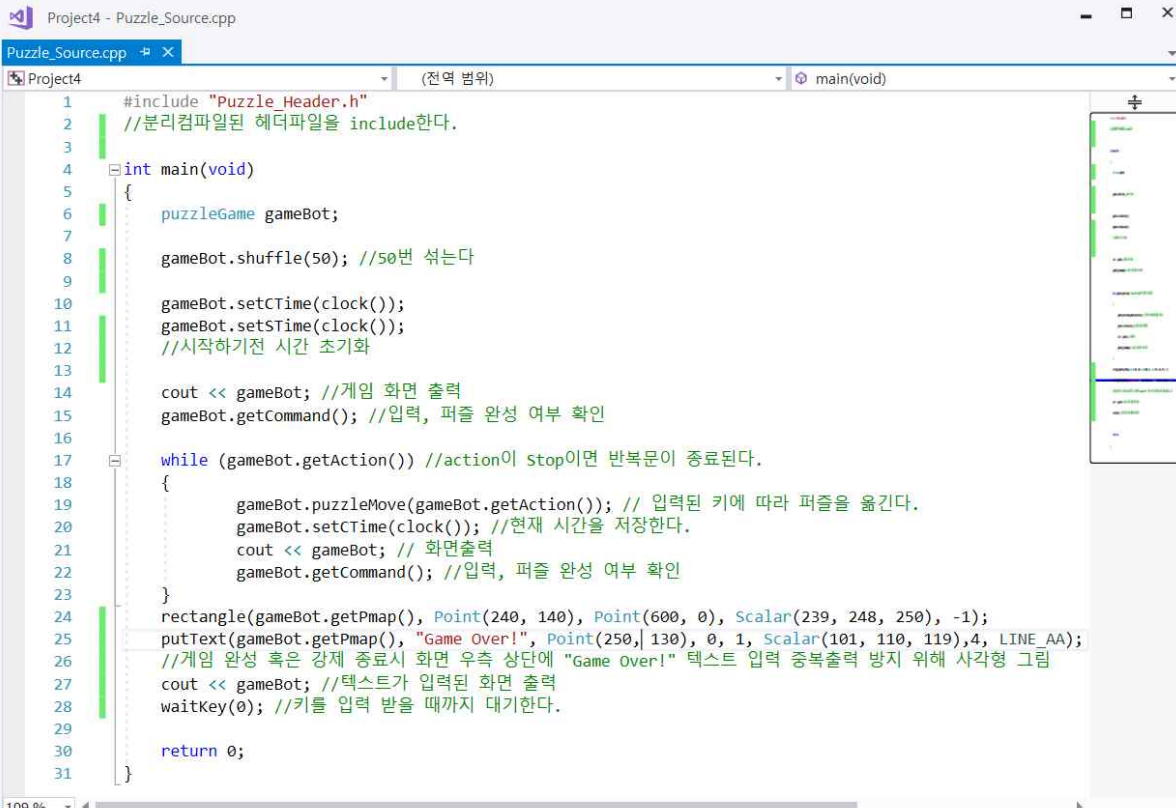


보 고 서 #4

제 목: 4x4 퍼즐 게임 구현

과 목 명:	고급프로그래밍및실습
학 과:	스마트시스템소프트웨어학과
학 번:	20170404
이 름:	한종수
제 출 일:	2018년 6월 7일(목)
담당교수:	한 영 준

- (1) 주석을 포함한 C++ 소스 코드를 프린트로 출력하여 제출
 - (2) 게임 시작 단계, 게임 도중 단계, 게임 종료 단계의 화면을 이미지 캡처하여 칼라 프린트 출력하여 제출
- 분리 컴파일 메인 함수



```
1 #include "Puzzle_Header.h"
2 //분리컴파일된 헤더파일을 include한다.
3
4 int main(void)
5 {
6     puzzleGame gameBot;
7
8     gameBot.shuffle(50); //50번 섞는다
9
10    gameBot.setTime(clock());
11    gameBot.setTime(clock());
12    //시작하기전 시간 초기화
13
14    cout << gameBot; //게임 화면 출력
15    gameBot.getCommand(); //입력, 퍼즐 완성 여부 확인
16
17    while (gameBot.getAction()) //action이 stop이면 반복문이 종료된다.
18    {
19        gameBot.puzzleMove(gameBot.getAction()); // 입력된 키에 따라 퍼즐을 옮긴다.
20        gameBot.setTime(clock()); //현재 시간을 저장한다.
21        cout << gameBot; // 화면출력
22        gameBot.getCommand(); //입력, 퍼즐 완성 여부 확인
23    }
24    rectangle(gameBot.getPmap(), Point(240, 140), Point(600, 0), Scalar(239, 248, 250), -1);
25    putText(gameBot.getPmap(), "Game Over!", Point(250, 130), 0, 1, Scalar(101, 110, 119), 4, LINE_AA);
26    //게임 완성 혹은 강제 종료시 화면 우측 상단에 "Game Over!" 텍스트 입력 중복출력 방지 위해 사각형 그림
27    cout << gameBot; //텍스트가 입력된 화면 출력
28    waitKey(0); //키를 입력 받을 때까지 대기한다.
29
30    return 0;
31 }
```

분리 컴파일 헤더 파일

```
Project4 - Puzzle_Header.h
Puzzle_Header.h
Project4 (전역 범위)

1  #pragma once
2
3  #ifndef Puzzle_Header
4  #define Puzzle_Header
5
6  #include <iostream>
7  #include <ctime> //clock() 함수 사용
8  #include <opencv2/opencv.hpp>
9  #include <stdio.h>
10
11  using namespace cv;
12  using namespace std;
13
14  #define FSIZE 120 //숫자 그림 하나 크기
15  #define FSTARTX 30 //숫자 그림 시작 위치 x
16  #define FSTARTY 30 //숫자 그림 시작 위치 y
17  #define Psize 82 //조절되어야 할 숫자의 크기
18
19  #define ESC 27 //게임 종료
20  #define LEFT 2424832 //왼쪽 화살표 키 ASCII값
21  #define RIGHT 2555904 //오른쪽 화살표 키 ASCII값
22  #define DOWN 2621440 //아래 화살표 키 ASCII값
23  #define UP 2490368 //위쪽 화살표 키 ASCII값
24
25  #define MDIM 4 //4x4 퍼즐맵의 크기
26  #define PDIM 7 //7x7 퍼즐의 크기
27  #define PNUM MDIM*MDIM // 네모칸의 수
28
29  //게임 Map 그리기 시작 위치: 필요하다면 출력 위치를 조정할 수 있음
30  #define MAP_STARTX 117
31  #define MAP_STARTY 206
32
33  enum COMMAND { Stop = 0, Left, Right, Down, Up, Non };
34  //action 에 대해 숫자 지정
35
36  class puzzleGame
37  {
38  public:
39      puzzleGame(); //default 생성자: 클래스 객체 초기화 함수, 필요시에 매개변수를 추가함
40      ~puzzleGame(); // 소멸자: 클래스 객체의 소멸전에 동적할당의 해제와 같은 객체의 정리를 위해 사용됨
41      friend ostream & operator <<(ostream & outputStream, const puzzleGame& gmBot);
42      // gameDrawd 멤버함수 대신에 << 연산자 오버로딩을 통해 사용하시오.
43      void puzzleDraw(Mat Mapping, const Mat Puzzle[], int x, int y, int k) const;
44      // 하나의 퍼즐을 그리는 멤버함수
45      void puzzleMove(int moveKey); //퍼즐 이동 함수; 필요시에 매개변수 추가함
46      void shuffle(int num); //퍼즐 맵 초기화; 50회 무작위 이동, 필요시에 인자 추가 할 수 있음
47      void getCommand(void); //모든 퍼즐이 제 위치에 있거나 ESC키가 눌러졌으면 정수 0을 반환
48      // 방향키를 눌렀을 때는 해당 키값을 반환(LEFT, RIGHT, DOWN, UP)
49      //필요시에 매개변수 인자를 추가할 수 있음
50      int getAction(void) const { return action; }; //메인함수에서 action의 값을 읽는다.
51      int solvable(void); //풀 수 있는 퍼즐인지 판별하는 함수
52      void setCTime(clock_t ct) { cTime = ct; }; //cTime설정 인라인 함수
53      void setSTime(clock_t st) { sTime = st; }; //sTime설정 인라인 함수
54      string Point2(string nums) const; //소숫점 두자리까지 출력하기위한 함수
55      Mat getPmap() {return this->Pmap; }; //메인함수에서 Pmap을 부를때 필요한 함수
56  private:
57
58      Mat Puzzle_num, Pmap; //숫자 전체 맵 저장 Puzzle_num, 전체 배경맵 저장 Pmap
59      Mat Num16[PNUM]; //1~15, 빈칸을 저장하기 위한 Num[16] 배열
60      int **map; //map 동적할당위해 선언
61
62      int action; //앞으로 동작 저장
63      int x, y; //빈칸의 위치
64      int moveNum; //이동회수
65      clock_t sTime; //시작 시간
66      clock_t cTime; //시작 시간
67  };
68  #endif // !Puzzle_Header
69
```

분리 컴파일 헤더파일 정의부

```
Puzzle_Header.cpp
Project4 (전역 범위) operator<<(ostream & outputStream, const |
1 #include <conio.h> //getch()와 kbhit() 함수 사용
2 #include <fstream>
3 #include <windows.h> //GetStdHandle(), Sleep(), SetConsoleCursorPosition() 함수 사용
4 #include "Puzzle_Header.h" //분리컴파일
5
6 ostream & operator <<(ostream & outputStream, const puzzleGame& gmBot)
7 { // gameDraw 멤버함수 대신에 << 연산자 오버로딩을 통해 사용하시오.
8   namedWindow("Puzzle_Game", CV_WINDOW_AUTOSIZE); //창 이름 지정 및 띄우기
9
10   for (int i = 0; i < MDIM; i++)
11     for (int j = 0; j < MDIM; j++)
12       gmBot.puzzleDraw(gmBot.Pmap, gmBot.Num16, MAP_STARTX + 95 * j, MAP_STARTY + 95 * i, gmBot.map[i][j]);
13   //puzzleDraw 함수 호출해 퍼즐 출력.
14
15   float timing = (gmBot.cTime - gmBot.sTime) / 1000.00, temp; //소요시간 계산
16
17   string time = gmBot.Point2(to_string(timing)); //두자리 수 까지 출력되게 뒷부분을 자름
18   string move = to_string(gmBot.moveNum);
19   //putText함수 사용위해 시간을 string형으로 바꾼다.
20
21   Size timetext = getTextSize(time, 0, 1, 2, 0);
22   Size movetext = getTextSize(move, 0, 1, 2, 0);
23   Point timepoint, movepoint;
24
25   movepoint.x = 185 - movetext.width/2; movepoint.y = 690;
26   timepoint.x = 415 - timetext.width/2; timepoint.y = 690;
27   //시간과 이동횟수를 왼쪽 아래부터가 아닌 중간부터출력되게 하기위해 텍스트크기를 재고 중간으로 맞춘다.
28
29   rectangle(gmBot.Pmap, Point(97, 655), Point(270, 700), Scalar(160, 173, 187), -1);
30   rectangle(gmBot.Pmap, Point(327, 653), Point(500, 700), Scalar(160, 173, 187), -1);
31   //이동횟수와 시간 출력시 이미지에 중첩되어 출력되는 문제가 발생하는데 이를 방지하기 위해
32   //중첩되는 부분에 계속해서 배경과 같은 색상의 채워진 사각형을 그린다.
33
34   putText(gmBot.Pmap, time, timepoint, 0, 1, Scalar(255, 255, 255), 2, LINE_AA);
35   putText(gmBot.Pmap, move, movepoint, 0, 1, Scalar(255, 255, 255), 2, LINE_AA);
36   //정해진 위치에 시간과 이동횟수를 출력한다.
37
38   imshow("Puzzle_Game", gmBot.Pmap);
39   waitKey(100);
40   //이미지를 창에 표시하며 보이게 하기위해 0.1초의 대기시간을 부여한다.
41
42   return outputStream;
43 }
44
45 puzzleGame::puzzleGame()
46 {
47   Pmap = imread("C:/users/young/Pictures/Map.png", CV_LOAD_IMAGE_COLOR);
48   //배경에 대한 사진을 Mat형의 Pmap에 저장한다.
49
50   Puzzle_num = imread("C:/users/young/Pictures/Puzzle_Num.png", CV_LOAD_IMAGE_COLOR);
51   Rect temp;
52   int num = 0;
53   // Draw circles on the detected faces
54   for (int i = 0; i < PNUM / 4; i++) {
55     for (int j = 0; j < PNUM / 4; j++) {
56       temp.x = FSTARTX + (FSIZE + 20)*j;
57       temp.y = FSTARTY + (FSIZE + 20)*i;
58       temp.width = FSIZE;
59       temp.height = FSIZE;
60       resize(Puzzle_num(temp).clone(), Num16[num++], Size(Psize, Psize));
61     }
62   }
63   //숫자에 관한 사진을 Mat형의 Puzzle_num에 저장한다.
64   //불러온 이미지에서 사각형의 영역을 지정하고 각각의 숫자들을 Num16배열에 저장한다.
65
66   map = new int*[MDIM]; // 이차원 포인터를 동적할당 한다.
67
68   for (int i = 0; i < MDIM; i++)
69     map[i] = new int[MDIM]; // 이차원 포인터를 동적할당한다.
70
71   for (int i = 0; i < MDIM; i++)
72     for (int j = 0; j < MDIM; j++)
73       map[i][j] = i * 4 + j;
74   //퍼즐 맵 동적할당 및 값 할당
75   x = MDIM - 1; y = MDIM - 1;
76   // 맵의 값을 1,2,3,...,MDIM*MDIM-1,0의 순서로 초기화 시키고 빈칸의 위치를 마지막으로 초기화시킨다.
77   sTime = 0.00; //시작시간을 초기화 시킨다.
78   moveNum = 0; //이동 회수를 초기화시킨다.
79
80   //게임 초기화 코드 작성
81 }
82
83 puzzleGame::~puzzleGame()
84 {
85   for (int i = 0; i < MDIM; i++)
86     delete[] map[i]; //배열 포인터를 초기화한다.
87
88   delete[] map; // 배열 포인터를 초기화한다.
89 }
90
91 void puzzleGame::puzzleDraw(Mat Mapping, const Mat Puzzle[], int x, int y, int k) const
92 { //숫자가 그려질 이미지와 숫자 이미지, 입력할 위치, 입력할 숫자를 입력받는다.
93   Mat imageROI = Mapping(Rect(x, y, Psize, Psize));
94   // 영상 ROI 정의
95   // Rect는 사각형 영역 지정
96   // x, y은 각각 숫자 크기의 x좌표, y좌표 시작지점
97   // Psize는 숫자 하나 이미지의 가로 세로 크기
98
99   Puzzle[k].copyTo(imageROI);
100   //지정한 관심영역에 숫자 하나 이미지를 복사하여 붙인다.
```



```

101 }
102
103 void puzzleGame::puzzleMove(int moveKey)
104 {
105     int temp = 0;
106     switch (moveKey) //getcommand를 통해 입력받은 키보드 값으로 퍼즐을 움직인다.
107     {
108     case Left:
109         if (x >= 1) { //예외 처리를 통해 빈칸의 위치가 판을 벗어나지 않도록 한다.
110             temp = map[y][x];
111             map[y][x] = map[y][x - 1];
112             map[y][x - 1] = temp;
113             x--; moveNum++; //퍼즐을 이동했기 때문에 이동 횟수를 1더한다.
114         }
115         break;
116
117     case Right:
118         if (x <= 2) {
119             temp = map[y][x];
120             map[y][x] = map[y][x + 1];
121             map[y][x + 1] = temp;
122             x++; moveNum++; //퍼즐을 이동했기 때문에 이동 횟수를 1더한다.
123         }
124         break;
125
126     case Up:
127         if (y >= 1) {
128             temp = map[y][x];
129             map[y][x] = map[y - 1][x];
130             map[y - 1][x] = temp;
131             y--; moveNum++; //퍼즐을 이동했기 때문에 이동 횟수를 1더한다.
132         }
133         break;
134
135     case Down:
136         if (y <= 2) {
137             temp = map[y][x];
138             map[y][x] = map[y + 1][x];
139             map[y + 1][x] = temp;
140             y++; moveNum++; //퍼즐을 이동했기 때문에 이동 횟수를 1더한다.
141         }
142         break;
143
144     default:
145         break; //아무것도 아니면 그냥 switch를 탈출한다.
146     }
147 }
148
149 void puzzleGame::getCommand(void)
150 {
151     bool Flag = FALSE; //퍼즐이 맞추어졌는지 확인한다.
152
153     for (int i = 0; i < MDIM; i++)
154     for (int j = 0; j < MDIM; j++) {
155         if ((map[i][j] != i * 4 + j) && (i * j < (MDIM - 1) * (MDIM - 1))) { Flag = TRUE; }
156     } //중간에 틀린 부분이 발생하면 Flag가 TRUE로 설정된다.
157     if (Flag == FALSE)
158         action = Stop; //퍼즐이 다 맞으면 action을 Stop으로 한다.
159
160     if (action != Stop) { //퍼즐이 맞추어지지 않은 경우 키보드로 키를 입력받는다.
161         switch (cWaitKey(0)) { //getDirectKey 함수를 통해 키를 입력받고 명령어를 디코딩해 action에 저장한다.
162         case ESC:
163             action = Stop;
164             moveNum = 0;
165             sTime = cTime = 0; //시간과 이동횟수를 초기화한다.
166             break;
167         case LEFT:
168             action = Left;
169             break;
170         case RIGHT:
171             action = Right;
172             break;
173         case DOWN:
174             action = Down;
175             break;
176         case UP:
177             action = Up;
178             break;
179         default:
180             action = Non;
181             break; // 방향키나 ESC키 말고 다른키가 입력되면 NON을 리턴한다.
182         }
183     }
184 }
185
186 void puzzleGame::shuffle(int num)
187 {
188     int numcheck[MDIM*MDIM]; //배열의 값이 중복되었는지 체크하기 위한 맵과 동일한 크기의 배열을 생성한다.
189     int random = 0; //생성된 난수를 저장해 놓는 변수이다.
190
191     srand(time(NULL)); //난수의 시드 값을 바꾸어 줌으로써 난수 생성을 한다.
192
193     putText(Pmap, "Shuffling...", Point(250, 130), 0, 1, Scalar(101, 110, 119), 4, LINE_AA);
194     //섞고 있다는 안내문을 출력한다.
195
196     for (int k = 0; k < num; k++)
197     {
198         while (1) {
199             for (int i = 0; i < MDIM*MDIM; i++)
200                 numcheck[i] = 1; //난수 설정하기전에 중복체크 배열을 사용되기 전 상태인 '1'로 초기화 시킨다.

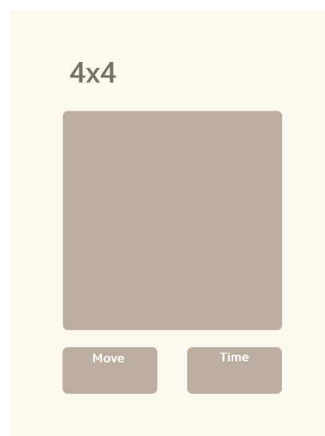
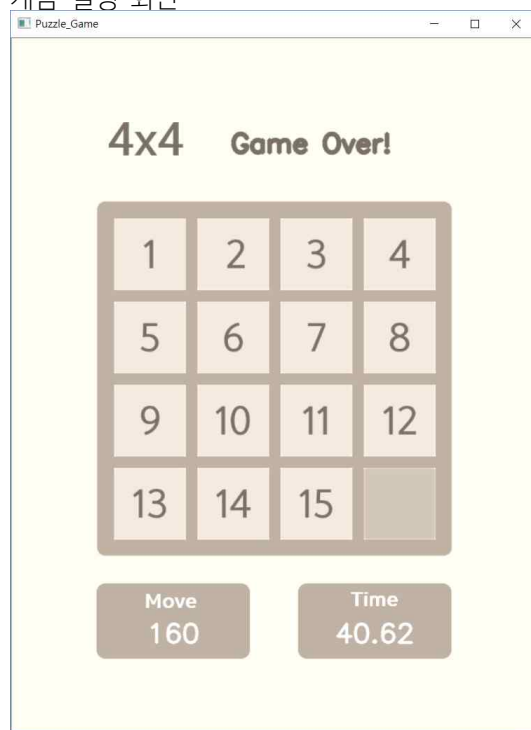
```

```

201         for (int i = 0; i < MDIM; i++)
202             for (int j = 0; j < MDIM; j++)
203                 while (1) {
204                     if ((numcheck[random = (rand() % (MDIM*MDIM))] != 0) {
205                         if (random == 15) { //만약 난수의 값이 빈칸의 좌표인 0이라면 그의 배열 좌표를 저장한다.
206                             x = j; //빈칸의 위치를 저장하기 위해 멤버 변수 x,y에 2차원 배열의 좌표를 저장한다.
207                             y = i;
208                         }
209                         map[i][j] = random;
210                         numcheck[random] = 0;
211                         break;
212                     } //생성된 난수가 사용되지 않은 난수이면 사용 표시를 하고 배열에 저장하며 루프를 탈출한다.
213                     // 난수가 사용 전이면 numcheck에 1의 형태로, 후면 0의 형태로 저장된다.
214                 }
215
216         if (solvable() == 1) //퍼즐을 풀지 못하는 경우로 설일 수 있다. 풀 수 있는지 판별하여
217             break; //풀 수 있으면 1을 리턴하며 풀지못하면 0을 리턴하고 다시 쉰다.
218     }
219     cTime = sTime = 0.00; //쉰을 때는 시작 시간을 계속 초기화 시켜줌으로써 시간이 0으로 보이도록 한다.
220     cout << *this; //현재 클래스를 출력하기 위해 this포인터를 사용한다.
221     Sleep(100); //맵 변경이 보이도록 대기시간을 준다.
222 }
223 rectangle(Pmap, Point(240, 140), Point(600, 0), Scalar(239,248,250), -1);
224 putText(Pmap, "Start!", Point(250, 130), 0, 1, Scalar(101, 110, 119), 4, LINE_AA);
225 //시작하라는 안내문구를 출력하며 문자의 중복출력을 막기위해 배경과 동일한 색의 사각형을 그린다.
226 }
227
228 int puzzleGame::solvable(void) //무작위로 배열된 퍼즐이 풀수있는지 판별하는 함수
229 {
230     int inv_count = 0, k = 0, temp, t_arr[MDIM*MDIM]; //반전된 숫자 배열 세는 변수 등을 선언
231
232     if (x != MDIM - 1 || y != MDIM - 1) //먼저 빈칸의 위치를 맨 아래 맨 오른쪽으로 이동시킨다.
233     {
234         temp = map[MDIM - 1][MDIM - 1];
235         map[MDIM - 1][MDIM - 1] = MDIM*MDIM-1;
236         map[y][x] = temp;
237         y = MDIM - 1;
238         x = MDIM - 1;
239     }
240
241     for (int i = 0; i < MDIM; i++)
242         for (int j = 0; j < MDIM; j++)
243             t_arr[i * 4 + j] = map[i][j];
244     //배열의 숫자들을 비교하기 위해 퍼즐 크기의 일차원 배열에 복사한다.
245
246     for (int i = 0; i < MDIM*MDIM - 2; i++)
247         for (int j = i + 1; j < MDIM * MDIM - 1; j++)
248             if (t_arr[i] > t_arr[j])
249                 inv_count++;
250     //일차원 배열들의 값을 순차적으로 비교하며 반전된 수의 수를 센다.
251
252     if (inv_count % 2 == 1) //만약 반전된 수의 수가 홀수 개이면 풀 수 없는 퍼즐이며 0을 리턴한다.
253         return 0;
254     else
255         return 1; //짝수 개이면 풀 수 있는 퍼즐이며 1을 리턴한다.
256 }
257
258 string puzzleGame::Point2(string nums) const
259 {
260     //소숫점 두자리까지 출력하기 위한 함수이다.
261     string temp; //임시 string temp를 설정한다.
262     temp = nums.substr(0, nums.find(".") + 3);
263     //문자열에서 .find를 통해 .의 위치를 찾고 .substr을 통해 0번째 인덱스부터 .포함 3개의 문자열을
264     //출력하면 소수점 두자리까지 출력하게 된다. 이를 temp에 저장후 temp를 리턴한다.
265     return temp;
266 }

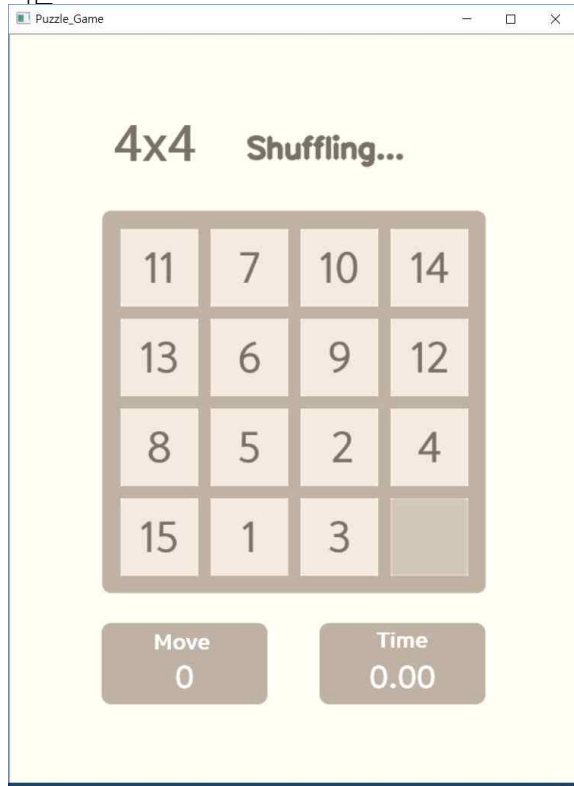
```

게임 실행 화면

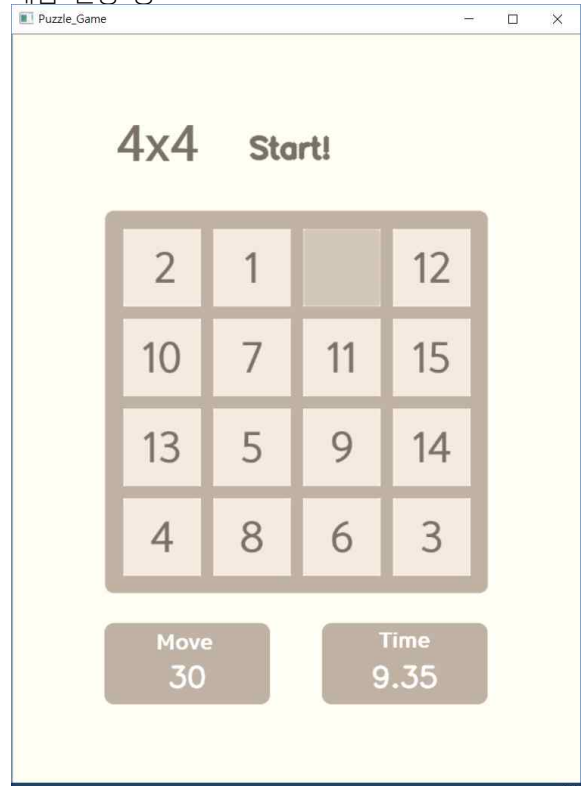


<- 퍼즐 완성 화면. 상단의 두 이미지는 기본 이미지

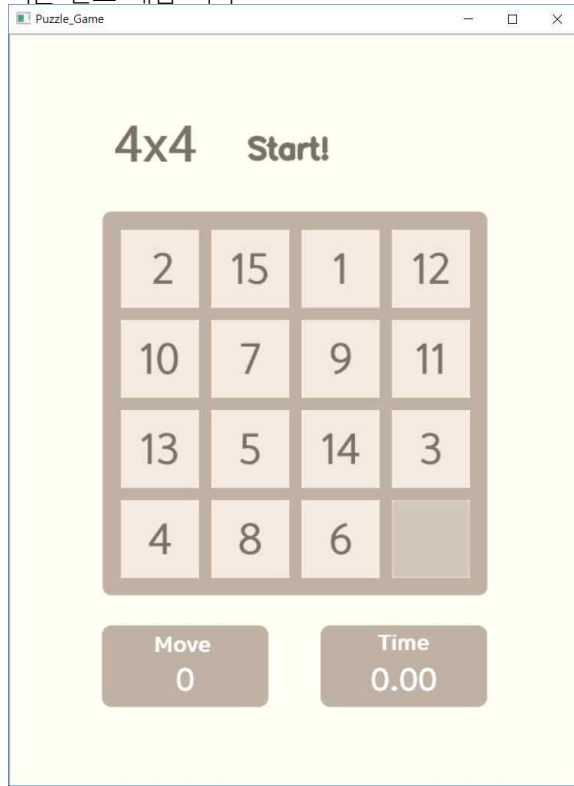
셔플



게임 진행 중



셔플 완료 게임 시작



게임 강제 종료

