C++프로그래밍및실습

2048게임 구현

프로젝트 제안서

제출일자: 2024 / 11 / 02 (토)
제출자명: 김혜신
제출자학번: 232053

1. 프로젝트 목표

1) 배경 및 필요성

일상 속에서 우리는 다양한 문제에 직면하고 이를 해결해야 하는 상황을 자주 맞닥뜨림. 2048 게임은 주어진 블록을 합쳐 목표 숫자에 도달하기 위해 전략적으로 생각하게 만듦. 이러한 사고방식은 일상에서 논리적이고 효율적인 문제 해결 방식을 찾는 데 도움이 됨. 게다가 게임이라는 점에서 바쁜 일상 속에서 잠시 휴식을 취하며 집중력과 몰입감을 높일 수 있고, 여가 시간을 유용하게 활용할 수 있는 취미가 될 수 있음.

2) 프로젝트 목표

같은 숫자가 충돌할 때마다 그 숫자가 합쳐져 2배의 숫자가 됨 (예: 2와 2가 만나면 4, 4와 4가 만나면 8). 목표는 숫자를 계속 합쳐서 **2048**을 만드는 것.

3) 차별점

기존의 2048 게임과 다르게 다양한 모드를 추가할 예정. 예를 들면 주어진 목표를 달성하는 모드. 특수블록(폭발 블록, 점프 블록 등)을 이용한 아이템 모드, N x N 모드 등. 이러한 모드들을 통해 플레이어에게 다양한 경험을 제공하며 기본 게임에 독창성을 더함.

2. 기능 계획

1) 기능 1 (기본 2048 게임)

- 기본적인 2048게임을 구현

(1) 세부 기능 1 (3x3 게임판 만들기)

- tictactoe 활동을 참고하여 배열로 3x3 플레이 화면을 만들 예정

(2) 세부 기능 2 (랜덤 2의배수 숫자 생성)

- 10보다 작은 정수 중 2의 배수인 랜덤 숫자를 빈 공간에 생성 (한 턴에 하나 생성)

(3) 세부 기능 3 (블록 합체 및 이동 기능)

- 블록들을 주어진 방향으로 정렬시키되 같은 블록이 연속적으로 있을 경우 합침

- 블록이 2048이 되면 게임을 종료함

(4) 세부 기능 4 (사용자 이동기능)

- 편하게 w,a,s,d의 입력을 받아 사용자가 원하는 방향으로 블록들을 이동시킴

(5) 세부 기능 5 (NxN 확장)

- 플레이 화면을 3 이상의 정수 N으로 확장시킴

2) 기능 2 (아이템 모드 만들기)

- 기본적인 2048게임에 다양한 기능의 블록을 추가

(1) 세부 기능 1 (점프 블록)

- 사용자 입력 방향에 가장 근접한 블록이 아닌 한 칸 떨어진 블록이 같은 경우 해당 블록과 합체

(2) 세부 기능 2 (폭발 블록)

- 사용자 입력 방향으로 정렬 시 폭발 블록과 인접한 가로 세로 대각에 위치한 블록들 모두 파괴

3) 기능 3 (목표 달성 모드 만들기)

- 5x5 2048게임의 가운데(3x3 위치) 목표가 되는 블록(1024 혹은 2048)을 위치시키고, 해당 블록을 합치면 클리어 하는 모드

(1) 세부 기능 1 (난이도 설정)

- 3x3 위치에 선택한 난이도에 따라 목표가 되는 블록을 정하는 기능 (상-2048, 중- 1024, 하- 512)

(2) 세부 기능 2 (클리어 조건)

- 가운데 위치한 블록을 합치면 클리어 되는 것으로 기존의 조건을 변경함

3. 진척사항

1) 기능 구현 [기본적인 2048게임을 구현]

(1) 세부 기능 1 (4x4 게임판 만들기) & 세부 기능 5 (NxN 확장)

- 입출력

➔ 기본 numCell(= 4)을 받아 변수 i, j와 for문을 이용하여 4x4 게임판을 출력합니다.

- 설명

➔ 중간고사 tictactoe에서 numCell을 변경하여 게임판의 크기를 바꾸는 것을 참고하여 2048 기본 게임판을 구현하였습니다.

주로 for문에 numCell을 활용하여 유동적인 코드로 작성했으며, 특이사항은 숫자가 한 자리 수, 두 자리 수, 세 자리 수, 네 자리 수인때에 각각 공백을 달리하여 게임판이 붕괴되는 현상을 방지하였습니다.

(이후에 기능3 목표 달성 모드 만들기에 5x5를 사용할 계획입니다.)

- 적용된 배운 내용 (예: 반복문, 조건문, 클래스, 함수, 포인터 등)

➔ for문을 이용한 반복문, 반복되는 사용이 많기 때문에 draw라는 이름으로 함수화, 게임판의 모양은 수업시간에 배운 tictactoe를 참고하였습니다.

- 코드 스크린샷

```
14  const int numCell = 4; // 보드판의 가로 세로 길이 (기본 4x4)
15  int board[numCell][numCell] = {0}; // 게임판 초기화
```

```
// 세부 기능 1 (3x3 게임판 만들기)
void draw(void)
{
    int i, j;
    system("clear"); // 터미널 화면을 지우는 명령어
    for (int i = 0; i < numCell; i++)
    {
        for (int i = 0; i < numCell - 1; i++)
        {
            cout << "----|";
        }
        cout << "----" << endl;
        for (int j = 0; j < numCell; j++)
        {
            cout << board[i][j];
            if (j == numCell - 1)
            {
                break;
            }
            if (board[i][j] < 10){
                cout << "   |";}
            else if (board[i][j] < 100){
                cout << "  |";}
            else if (board[i][j] < 1000){
                cout << " |";}
            else if (board[i][j] < 10000){
                cout << "|";}
        }
        cout << endl;
    }
    for (int i = 0; i < numCell - 1; i++)
    {
        cout << "----|";
    }
    cout << "----" << endl;
    printf("\nScore : %d \n", score);
}
```

(2) 세부 기능 2 (랜덤 2의배수 숫자 생성)

- 입출력

➔ numCell의 제곱만큼 포인터 배열을 생성하여 빈 칸을 저장, 랜덤 선택하여 2또는 4를 생성하여 배열에 저장하는 함수입니다.

- 설명

➔ 게임판의 수만큼 포인터를 지정하고, 각 칸을 for문을 통해 확인하며 0인 경우 cnt의 값을 증가시켜 포인터 배열에 저장합니다.(최대 numCell의 제곱까지 저장 가능) p0 배열에 저장된 빈 칸 중에서 무작위로 하나를 선택하여 새로운 숫자를 생성하며, rand() % 100 < 80 조건을 사용해 80% 확률로 2를, 20% 확률로 4를 생성하여 선택된 빈 칸에 대

입합니다.

- 적용된 배운 내용 (예: 반복문, 조건문, 클래스, 함수, 포인터 등)

➔ 수업시간에 배운 반복문과 조건문, 포인터, 배열, 난수생성(srand(time(NULL)))이 사용되었고 이 코드를 new_num이라는 이름으로 함수화했습니다.

- 코드 스크린샷

```
50      srand(time(NULL)); // 난수생성 시드값
51      new_num();         // 초기값 2개 생성
52      new_num();
53      draw(); // 임의의 수 2개 생성 후 게임판 그리기
```

```
190 // 세부 기능 2 (랜덤 2의배수 숫자 생성)
191 void new_num(void)
192 {
193     int i, j, cnt = 0;
194     int *p0[numCell*numCell] = {0}; // 포인터 배열 p0를 선언
195
196     for (i = 0; i < numCell; i++)
197     {
198         for (j = 0; j < numCell; j++)
199         {
200             if (board[i][j] == 0)
201             {
202                 p0[cnt] = &board[i][j]; // board[i][j]가 0인 경우, 해당 칸의 주소를 p0 배열에 저장
203                 cnt++;                  // cnt를 1 증가하여 다음 빈 칸의 위치를 가리킴
204             }
205         }
206     }
207
208     *p0[rand() % cnt] = (rand() % 100 < 80) ? 2 : 4;
209     // p0 배열에 저장된 빈 칸 중에서 무작위로 하나를 선택하여 새로운 숫자를 생성
210     // rand() % 100 < 80 조건을 사용해 80% 확률로 2를, 20% 확률로 4를 생성하여 선택된 빈 칸에 대입
211 }
```

(3) 세부 기능 3 (블록 합체 및 이동 기능) & 세부 기능 4 (사용자 이동기능)

- 입출력

➔ player로부터 wasd를 입력받아 게임판 속 데이터들을 상하좌우로 정렬한 결과를 출력해주고, 만약 이웃한 블럭이 같은 수인 경우에 이동방향에 있는 수에 흡수되어 2배가 되는 결과를 출력해주는 기능입니다.

- 설명

➔ 사용자의 입력을 받는 코드에서 <termios.h>는 터미널 입출력 설정을 제어하는 헤더 파일이며,<unistd.h>는 기본적인 시스템 레벨의 입출력 작업, 프로세스 제어, 파일 관리 등을 위한 여러 함수들을 제공하는 헤더 파일입니다. 이 두 헤더파일을 사용한 궁극적 이유는 게임 플레이어가 input값을 입력하고 엔터를 누르는 번거러움을 없애고자 올바른 input값을 입력한 즉시 게임에 반영시키기 위함 입니다.

➔ 사용자 입력에 따라 블럭을 정렬하고 합체하는 기능을 구현하기 위해서 입력받은 key값에 따라 LEFT, RIGHT, UP, DOWN의 케이스를 나눴습니다. 각각의 경우에 구현된 구조가 비슷하기 때문에 간단하게 요약하자면 각 행과 열을 순회하며 단일 요소마다 case1. 칸이 비어있거나 이미 병합된 경우 case2. 병합이 불가능한 경우 case3. 움직일 부분이 비어있는 경우(비어있는 부분으로 이동함), case4. 같은 숫자가 충돌하는 경우(병합)를 체크하여 case 3,4인 경우에는 act를 1씩 증가시켜 act>0 일 때 동작발생 취급하여 게임을 진행시키도록 하였습니다. 이때 각각의 원소를 순차적으로 검사하기 때문에 중복으로 병합되는 경우가 생길 수 있음을 고려하여 병합된 숫자에는 +10000을 하여 이미 결합됨을 표시해 주었고 이후 모든 순회가 끝나면 결합된 숫자에 -10000을 하여 숫자를 복구해 주는 과정이 있습니다.

- 적용된 배운 내용 (예: 반복문, 조건문, 클래스, 함수, 포인터 등)

➔ switch문, for문, if문, 배열이 사용되었고 main함수 내에 있기 때문에 이전에 만든 기능들을 호출하는 부분도 있으며, break문을 통해 특정한 경우에 switch문을 탈출하는 내용도 있습니다. tictactoe에서 while(1)무한루프로 진행시키며 특정 조건에서 종료하는 코드를 참고하였습니다.

- 코드 스크린샷

(사용자로부터 wasd를 입력받는 함수)

```c
// 세부 기능 4 (사용자 이동기능-입력받기)
int getch(void)
{
    // termios는 비차단 입력을 설정하여 키 입력이 발생하는 즉시 프로그램이 이를 처리함
    struct termios oldattr, newattr;
    // oldattr는 기존의 터미널 속성을 저장하고, newattr는 변경된 속성을 설정하기 위해 사용
    int ch;                         // 입력받은 한 글자를 저장할 변수 ch를 선언
    tcgetattr(STDIN_FILENO, &oldattr);
    // 현재 터미널 속성을 oldattr에 저장. STDIN_FILENO는 표준 입력 파일 디스크립터(일반적으로 키보드 입력)
    newattr = oldattr;
    newattr.c_lflag &= ~(ICANON | ECHO); // newattr의 c_lflag 필드를 변경하여 입력 모드를 수정.
    // ICANON 비트를 꺼서 비정규 모드로 변경. Enter를 누르지 않아도 키 입력이 즉시 프로그램으로 전달됨
    // ECHO 비트를 꺼서 에코 기능을 비활성화. 입력한 문자가 화면에 표시되지 않음
    tcsetattr(STDIN_FILENO, TCSANOW, &newattr);
    // 새로운 설정(newattr)을 터미널에 적용합니다. TCSANOW는 즉시 속성을 변경
    ch = getchar(); // 한 문자를 입력받아 ch 변수에 저장
    tcsetattr(STDIN_FILENO, TCSANOW, &oldattr);
    // 원래 터미널 속성(oldattr)을 복원, 프로그램이 끝나고 나서 터미널에 영향을 주지 않도록 함
    return ch;
}
```

(입력받은 key값에 따라 블록을 합체하거나 이동시키는 코드)

```c
// adws로 입력받기
#define LEFT 'a'
#define RIGHT 'd'
#define UP 'w'
#define DOWN 's'
```

```c
    // 게임 시작
    while (1)
    {
        key = getch(); // 사용자 입력을 key에 저장
        act = 0;

        // 세부 기능 3 (사용자 입력에 따른 블록 합체 및 이동 기능)
        switch (key)
        {
        case LEFT:
            for (i = 0; i < numCell; i++)
            { // 각 행에 대해 처리
                for (j = 1; j <= numCell-1; j++)
                { // 두 번째 칸부터 마지막 칸까지
                    for (r = j; r > 0; r--)
                    { // 현재 칸을 왼쪽으로 이동
                        if (board[i][r] == 0 || board[i][r] > 10000)
                            break; // case1.비어있거나 이미 병합된 경우
                        if (board[i][r - 1] != 0 && board[i][r - 1] != board[i][r])
                            break; // case2.병합 불가능한 경우 (연속 같은 수)
                        if (board[i][r - 1] == 0)
                            board[i][r - 1] = board[i][r]; // case3.왼쪽이 비어있으면 이동
                        else if (board[i][r] == board[i][r - 1])
                        { // case4.같은 숫자가 충돌하면 병합
                            board[i][r - 1] *= 2;       // 2배로 병합
                            board[i][r - 1] += 10000;   // 병합된 숫자 표시
                            score += 2 * (board[i][r]); // 점수 업데이트
                        }
                        // case3,4(움직임 발생)인 경우만 해당
                        board[i][r] = 0; // 현재 칸 비우기
                        act++;           // 이동/병합 발생 표시
                    }
                }
            }
            break;
```

```
        case RIGHT:
            for (i = 0; i < numCell; i++)
            { // 각 행에 대해 처리
                for (j = numCell-2; j >= 0; j--)
                { // 마지막 -1 칸부터 첫 번째 칸까지
                    for (r = j; r < numCell-1; r++)
                    { // 현재 칸을 오른쪽으로 이동
                        if (board[i][r] == 0 || board[i][r] > 10000)
                            break;
                        if (board[i][r + 1] != 0 && board[i][r + 1] != board[i][r])
                            break;
                        if (board[i][r + 1] == 0)
                            board[i][r + 1] = board[i][r];
                        else if (board[i][r] == board[i][r + 1])
                        {
                            board[i][r + 1] *= 2;
                            board[i][r + 1] += 10000;
                            score += 2 * (board[i][r]);
                        }
                        board[i][r] = 0;
                        act++;
                    }
                }
            }
            break;

        case UP:
            for (j = 0; j < numCell; j++)
            { // 각 열에 대해 처리
                for (i = 1; i <= numCell-1; i++)
                { // 두 번째 행부터 마지막 행까지
                    for (r = i; r > 0; r--)
                    { // 현재 칸을 위로 이동
                        if (board[r][j] == 0 || board[r][j] > 10000)
                            break;
                        if (board[r - 1][j] != 0 && board[r - 1][j] != board[r][j])
                            break;
                        if (board[r - 1][j] == 0)
                            board[r - 1][j] = board[r][j];
                        else if (board[r][j] == board[r - 1][j])
                        {
                            board[r - 1][j] *= 2;
                            board[r - 1][j] += 10000;
                            score += 2 * (board[r][j]);
                        }
                        board[r][j] = 0;
                        act++;
                    }
                }
            }
            break;

        case DOWN:
            for (j = 0; j < numCell; j++)
            { // 각 열에 대해 처리
                for (i = numCell-2; i >= 0; i--)
                { // 마지막-1 행부터 첫 번째 행까지
                    for (r = i; r < numCell-1; r++)
                    { // 현재 칸을 아래로 이동
                        if (board[r][j] == 0 || board[r][j] > 10000)
                            break;
                        if (board[r + 1][j] != 0 && board[r + 1][j] != board[r][j])
                            break;
                        if (board[r + 1][j] == 0)
                            board[r + 1][j] = board[r][j];
                        else if (board[r][j] == board[r + 1][j])
                        {
                            board[r + 1][j] *= 2;
                            board[r + 1][j] += 10000;
                            score += 2 * (board[r][j]);
                        }
                        board[r][j] = 0;
                        act++;
                    }
                }
            }
```

```
167             break;
168         }
169         // 병합된 숫자 복구
170         for (i = 0; i < numCell; i++)
171         {
172             for (j = 0; j < numCell; j++)
173             {
174                 if (board[i][j] > 10000)
175                     board[i][j] -= 10000; // 병합된 숫자 복구
176             }
177         }
```

```
178         // 동작 발생
179         if (act > 0)
180         {
181             new_num();          // 새로운 숫자 추가
182             draw();             // 보드 출력
183             check_game_over(); // 게임 오버 상태 확인
184         }
185     }
186 }
```

(4) + 세부 기능 6 (승리 조건 확인하기)

- 입출력

➔ numCell을 이용하여 모든 게임 블럭을 검사하며 우승조건에 충족할 경우 혹은 게임을 더이상 진행시킬 수 없는 상황인 경우에 게임을 종료시키는 멘트를 출력하고 종료합니다.

- 설명

➔ numCell입력을 받아 게임판의 모든 데이터들을 체크하면서 최고점수(100000)을 돌파한 경우 멘트를 출력하고 게임을 종료시키고, 빈 칸이 있거나 합칠 수 있는 수가 이웃한 경우 return을 통해 게임을 재개하며, 모든 조건을 충족하지 못하는 경우 (빈 칸이 없는 경우) 게임오버가 되는 함수입니다.

➔ 이때 주의할 점으로 합칠 수 있는 수를 따지는 조건에서 마지막 행 / 마지막 열을 비교하는 경우 존재하지 않는 (numCell=4 기준 - 4행 또는 4열 존재x) 데이터와 비교하는 코드가 되기 때문에, 마지막 행과 열의 비교 조건문은 따로 작성해야 합니다.

- 적용된 배운 내용 (예: 반복문, 조건문, 클래스, 함수, 포인터 등)

➔ for문, if문, 배열을 통해 구현하였으며 check_game_over라는 이름의 함수를 구현했습니다. tictactoe에서 게임 종료 조건을 함수화 했던 내용을 적용시켰습니다.

- 코드 스크린샷

```
252  void check_game_over(void)
253  {
254      int i, j;
255
256      for (i = 0; i < numCell; i++)
257          for (j = 0; j < numCell; j++)
258              if (board[i][j] > 10000){ // 10000을 초과하는 경우
259                  cout << "10000점을 돌파했습니다. 게임을 종료합니다.";
260                  exit(0); }
261
262      for (i = 0; i < numCell; i++)
263          for (j = 0; j < numCell; j++)
264              if (board[i][j] == 0) // 빈 칸이 있는 경우 게임진행
265                  return;
266
267      for (i = 0; i < numCell-1; i++)
268          for (j = 0; j < numCell-1; j++)
269              if (board[i][j] == board[i + 1][j] || board[i][j] == board[i][j + 1])
270              // 합칠 수 있는 수가 있는 경우 재개
271                  return;
272
273  // 만약 위의 for문에 범위를 4미만으로 할 경우 board[4][j]라는 존재하지 않는 정보를 비교하게 됨
274  // 때문에 3행 비교와 3열 비교는 각 각 분리해야 함
275      for (i = 0; i < numCell-1; i++)
276          if (board[i][numCell-1] == board[i + 1][numCell-1]) //가장 오른쪽 열을 검사
277              return;
278
279      for (j = 0; j < numCell-1; j++)
280          if (board[numCell-1][j] == board[numCell-1][j + 1]) //가장 아래쪽 행을 검사
281              return;
282
283
284
285  // 위 4가지 경우를 모두 충족시키지 못한 경우
286      cout << "Game Over..";
287      exit(0);
288  }
```

2) 테스트 결과

(1) 세부 기능 1 (4x4 게임판 만들기) & 세부 기능 5 (NxN 확장)

- 설명

➔ 기본 4x4와 numCell을 각각 5, 10으로 변경했을 경우 입니다.

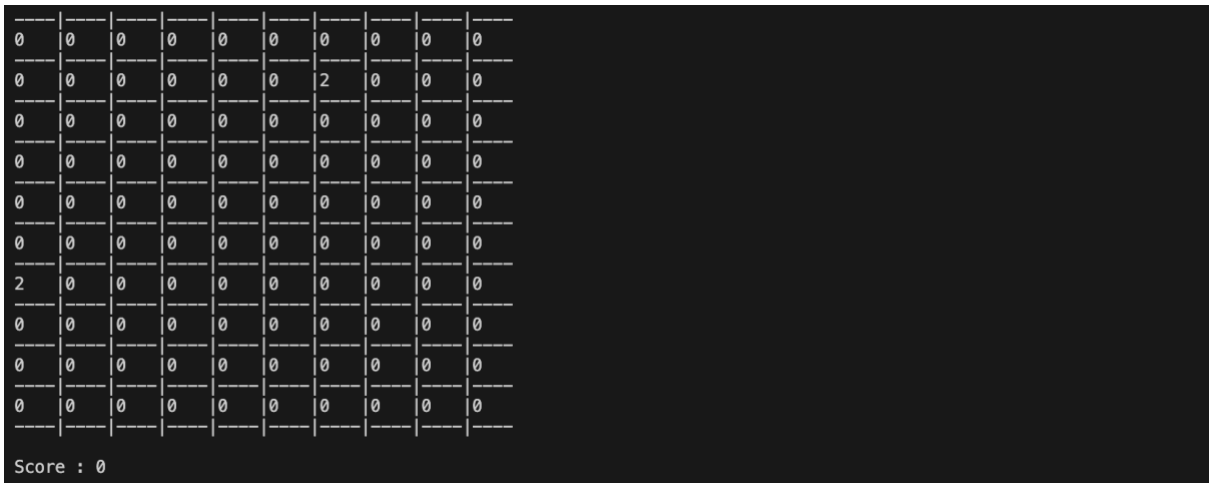
(4x4에서 한 자리, 두 자리, 세 자리 수에서 게임판이 파괴되지 않는지도 확인했습니다.)

- 테스트 결과 스크린샷

(4x4)

(5x5)

(10x10)



(2) 세부 기능 2 (랜덤 2의배수 숫자 생성)

- 설명

➔ 게임을 여러 번 실행시키며 위치와 숫자가 무작위로 생성되는지를 확인하겠습니다.

- 테스트 결과 스크린샷

(1차)

(2차)

(3차)

➔ 위치가 랜덤하다는 점과 4보다 2의 생성 비율이 높다는 점을 확인할 수 있습니다.

(3) 세부 기능 3 (블록 합체 및 이동 기능) & 세부 기능 4 (사용자 이동기능)

- 설명

➔ w,a,s,d 입력에 따라 게임판이 어떻게 변화하는 지를 알아보겠습니다.

- 테스트 결과 스크린샷

➔

(게임 시작 화면)

(w 를 입력한 경우 -UP)

➔ 기존의 0,2열의 2가 위로 정렬되고 2가 2행 1열에 무작위 생성됨.

(a 를 입력한 경우 -LEFT)

➔ 이전에 0행 2행의 데이터들이 왼쪽으로 정렬됨과 동시에 0행의 두 2가 병합되어 4가 생성되고, 마찬가지로 2가 랜덤생성됨을 확인.

(s 를 입력한 경우 -DOWN)

➔ 0열과 3열의 데이터가 아래로 정렬되면서 0열의 2와 4는 병합되지 않음을 확인 할 수 있고, 이번에는 4가 랜덤생성됨을 확인.

(d 를 입력한 경우 -RIGHT)

➔ 0행 2행 3행의 데이터가 우측으로 정렬, 3행의 2와 2가 병합, 2가 생성됨을 확인.

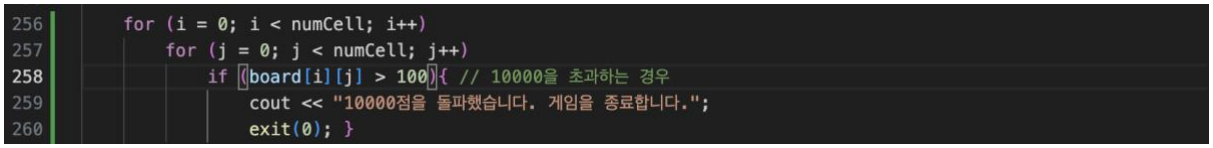
(4) + 세부 기능 6 (승리 조건 확인하기)

- 설명

➔ 10000점은 실제로 달성하기 힘들기 때문에 목표 점수를 100으로 임시 수정한 후 100 이상이 되었을 때 게임이 종료되는지, 보드가 병합할 수 없는 수들로 가득 찼을 때 게임 오버가 되는지를 테스트해보겠습니다.

- 테스트 결과 스크린샷

(100점이 목표점수인 경우)



➔ 목표점수를 도달했다는 멘트와 프로그램이 종료됨.

(보드가 가득 찬 경우)

➔ 게임오버 멘트를 마지막으로 프로그램이 종료됨을 확인함.

4. 계획 대비 변경 사항

1) 세부 기능 추가 및 numCell의 디폴트 값 변경

- 이전

➔ 세부기능 1의 목표 = 3x3 게임판을 만들기

➔ 게임 종료 조건 x

- 이후

➔ 세부기능 1의 목표를 4x4 게임판 만들기로 수정하였습니다.

➔ 게임 종료 조건을 세부기능 6으로 추가하게 되었습니다.

- 사유

➔ 직접 플레이 해 본 결과 3x3 보다는 4x4의 난이도가 기본값으로 적당하다고 느껴졌기 때문입니다.

➔ main함수 내에 자연스럽게 종료 조건을 녹여보려 했으나 함수화로 턴의 마지막에 한 번 호출해 버리는 편이 가독성이 높다고 생각하여 세부기능으로 따로 추가하게 되었습니다.

5. 프로젝트 일정

(진행한 작업과 진행 중인 작업 등을 표기)

업무		11/3	...	11/17	...
제안서 작성		완료			
기능1	세부기능1		완료		
	세부기능2		완료		
	세부기능3		완료		
	세부기능4		완료		
	세부기능5		완료		
기능2	세부기능1				----------
	세부기능2				----------
업무		12/1	...	12/15	...
기능2	세부기능1	---------->			
	세부기능2	---------->			
기능3	세부기능1		---------->		
	세부기능2		---------->		

(교수님 혹시 초록색으로 하면 눈이 아프시나요...?)