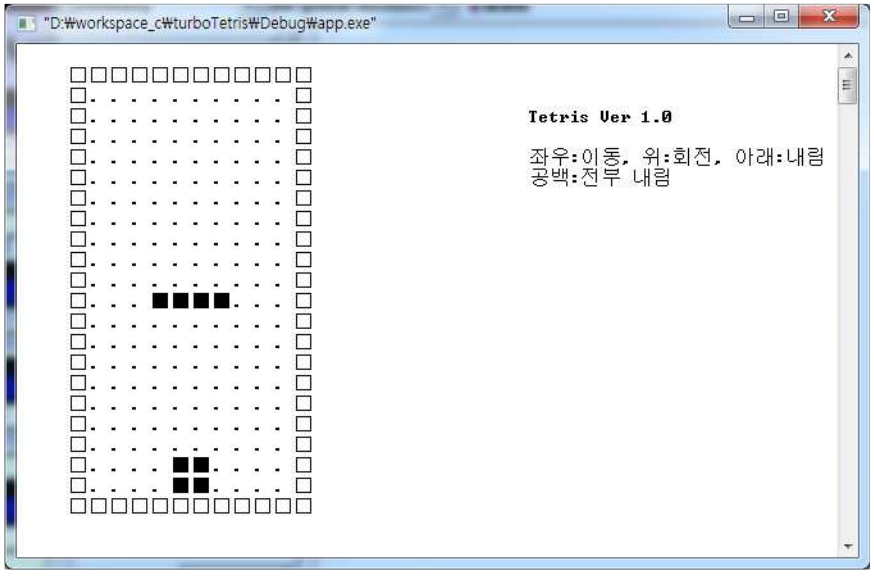
I 프로젝트 개요

01 게임디자인 확인하기

우리가 만들어 볼 테트리스 게임은 아주 기초적인 기능만을 가지고 있고 다음과 같이 디자인되어 있다. 콘솔환경에서 그래픽을 사용하지 않고 특수문자만을 사용하여 게임판의 벽과 안쪽 공간 그리고 테트리스 벽돌들을 표현하며, 게임의 진행방식은 보통의 테트리스와 똑같다.



게임 화면

게임 종료시에는 다음과 같은 메시지가 나타나며 프로그램이 종료된다.

게임 종료화면

02 사용자정의 함수의 기능과 개념도

▌기호

■ : 일반 C 함수

Ⓕ : 터보 C 함수

(단, 프로그램 내에서 정의된 사용자정의 함수는 기호 없음.)

전체 프로그램에 사용되는 함수의 기능과 개념도는 다음과 같다.

DrawScreen() – 게임의 배경화면인 게임판을 그린다.

DrawBoard() – 게임판의 내부화면을 그린다.

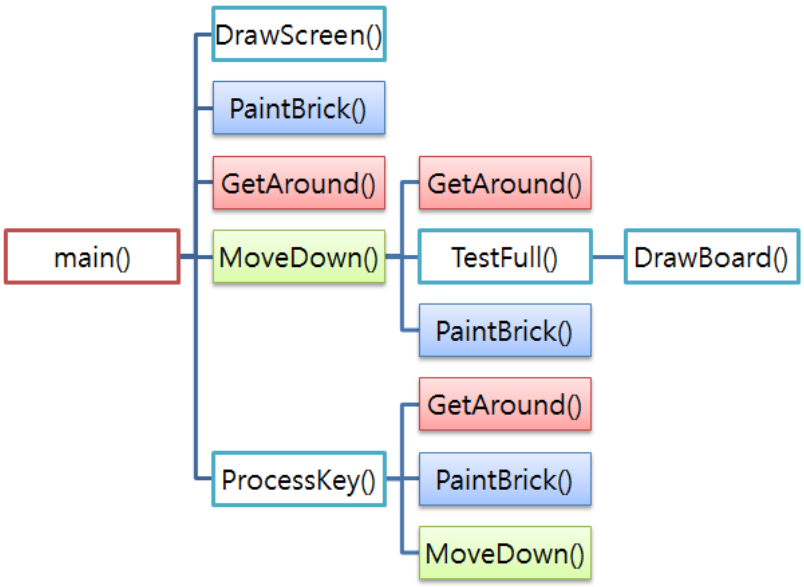
PrintBrick() – 벽돌을 그린다.

MoveDown() – 벽돌을 좌우, 아래로 움직인다.

GetAround() – 벽돌의 주변을 점검하여 빈공간이 있는지 체크한다.

TestFull() – 게임판이 벽돌로 다 찼는지 체크한다.

ProcessKey() – 키를 눌렀을 때 동작을 정의한다.



함수의 구성

03 터보 C에 대한 설명

터보 C는 볼랜드사에서 만든 것으로 setcursortype()나 gotoxy() 등 일반 C에서 제공하지 않는 편리한 기능들이 있다. 우리는 게임을 제작하기 위해 터보 C에서 제공했던 함수들을 우리 수업에서도 사용할 수 있도록 정의해 놓은 <turboc.h>파일을 같은 폴더에 저장한 후 위 함수들을 사용할 것이다. 이렇게 일반 C에서는 제공하지 않지만 터보 C에서 제공하는 함수는 본문 중에 Ⓕ로 표시하였다. <turboc.h>파일을 메모장에서 열어보면 파일의 처음 부분에는 <stdio.h>와 <conio.h> 파일들이 include되어 있는 것을 확인할 수 있다. 이 코드들 덕분에 게임소스에서 <turboc.h>만 include해도 일반 C 함수들을 자유롭게 사용할 수 있다.

테트리스의 역사와 종류

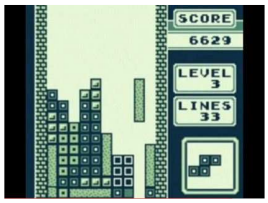


그림
테트리스

테트리스 원작자인 파지노프(Alexey Pajitnov)는 러시아 태생으로 컴퓨터 공학연구원으로 일하던 중 1984년에 테트리스를 만들었다. 테트리스란 그리스어 숫자 접두어인 Tetra와 파지노프가 평소 좋아했던 테니스를 합쳐 만든 단어이다. 그는 전통 퍼즐게임인 펜토미노(정사각형 5개로 이루어진 12개의 서로 다른 조각을 박스 안에 특정 순서대로 배치하는 퍼즐게임) 조각의 회전을 프로그래밍 하다가 테트리스를 만들게 된다. 펜토미노 조각을 단순화시켜 정사각형 4개로 이루어진 7개의 모양을 만들었는데 이 7개의 모양은 I, J, L, O, S, T, Z를 형상화한 것이다. 2000년 이후에는 테트리스 컴퍼니가 블록의 색을 규정하여 현재는 하늘, 파랑, 오렌지, 노랑, 초록, 보라, 빨강의 색을 가진다.

러시아 컴퓨터에서 구현한 테트리스는 한계가 있어서 친구에게 부탁하여 그 당시 많은 사람들이 사용하던 IBM PC버전 테트리스를 제작하였다. 이때부터 테트리스는 모스크바 전역으로 퍼져나가기 시작한다. 그러나 PC버전이 나오기도 전에 헝가리의 프로그래머에 의해 PC 버전이 출시가 되었고 그는 파지노프와 PC 버전의 권리에 대한 아무런 협의 없이 다른 회사에 권리를 팔았다. 이 회사는 1986년 미국에서 엄청난 인기를 누리며 수익을 올렸다. 이 회사는 이어 다른 컴퓨터 시장의 모든 테트리스 라이센스를 획득하고 시장을 점령한다.

파지노프는 테트리스에 대한 권리를 찾기 위해 끊임없이 노력하다가 1996년에야 비로소 얻게 된다. 그러나 이 위대한 게임을 만든 파지노프는 실상 큰 돈을 벌지는 못했다. 그는 저작권을 되찾은 후 테트리스 컴퍼니를 설립하고 세계 모든 나라의 테트리스 등록 상표를 소유하게 되었다.

우리나라에서도 그동안 각종 게임회사에서 오프라인, 온라인으로 제공되던 테트리스 게임이 테트리스 컴퍼니에 의해 중단되고, 이후 이 회사의 라이센스를 받은 일부 회사들이 현재 게임 서비스 중이다. PC버전으로는 한게임, 닌텐도 버전은 닌텐도 코리아, 모바일 버전으로는 컴투스 등이 있다.

한편, 테트리스는 역사가 깊어지며 다양한 모델들이 만들어졌다. 네트워크 테트리스는 온라인으로 친구와 함께 경쟁적으로 게임할 수 있도록 고안되었고, 블록의 모양을 변경하거나 블록 이외에 다른 아이템들을 삽입하여 게임의 재미를 추가하기도 하였다. 테트리스와 비슷한 개념의 퍼즐게임이면서도 다른 차원의 테트리스가 선보이기도 했다. 최근에도 3D테트리스와 같이 새로운 형태의 테트리스 게임이 지속적으로 개발되고 많은 사람들에게 사랑받고 있다.

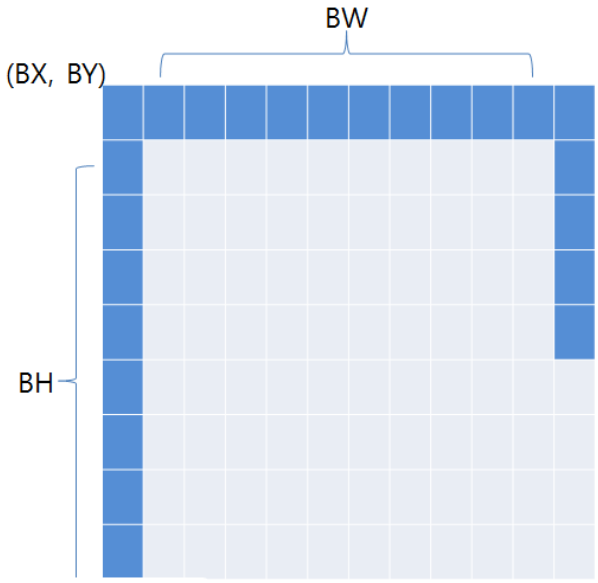
II 프로젝트 분석

이번 장에서는 테트리스 코드를 하나하나씩 분석해가며 콘솔모드에서 어떻게 동적인 테트리스를 구현하는지에 대해 자세히 배울 것이다. 코드에서 사용되는 함수에 대한 철저한 이해를 바탕으로 게임요소와 이 요소들을 조작하는 알고리즘에 대해서 공부한다. 우리가 배우는 테트리스는 가장 기본적인 기능만을 가지고 있다. 이 장을 배우고 난 후 여러분들의 독창적이고 산뜻한 아이디어를 적용해 좀 더 발전된 테트리스를 만들어보자.

01 게임판 그리기

1.1 게임판 설정

게임판을 만들어 보자. 게임판은 다음과 같은 구조로 되어 있다.



▌게임판의 위치와 크기

우선 게임판이 위치할 곳을 정한다. 콘솔창의 좌상단에서부터 (x, y) 좌표값으로 표현하는데 게임판의 시작 좌표는 BX, BY라는 매크로 상수에 저장한다. 같은 방법으로 게임판의 너비와 높이를 설정하는데 정수 값을 직접 넣어

도 되지만 우리는 BX, BY처럼 BW, BH라는 정수형 **매크로 상수**로 정의하여 사용한다. 매크로 상수를 사용하면 나중에 게임판의 위치와 크기를 수정할 때 편리하다. 우리 코드에는 (5,1)에서 시작하여 너비(BW) 10, 높이(BH) 20을 가지도록 정의되어 있다.

▌게임판의 현재 상태 – board배열

게임판의 현재 상태는 board 배열에 저장된다. 이 배열은 어디에 얼마만큼의 벽돌이 쌓여 있는지에 대한 정보를 보관하고 있다. 전역변수로 선언되어 있고 main()에서 초기화되는데 게임판 바깥쪽 테두리를 표현하는 배열요소는 모두 정수 2로 설정하고, 안쪽 빈공간은 모두 0, 벽돌은 모두 1로 설정한다.

설정값	의미
0	안쪽 빈 공간
1	벽돌
2	바깥 테두리

▌표 1 board배열 설정값과 의미

▌참고
열거체에 관한 내용은 9주차 복합자료형을 참고할 것

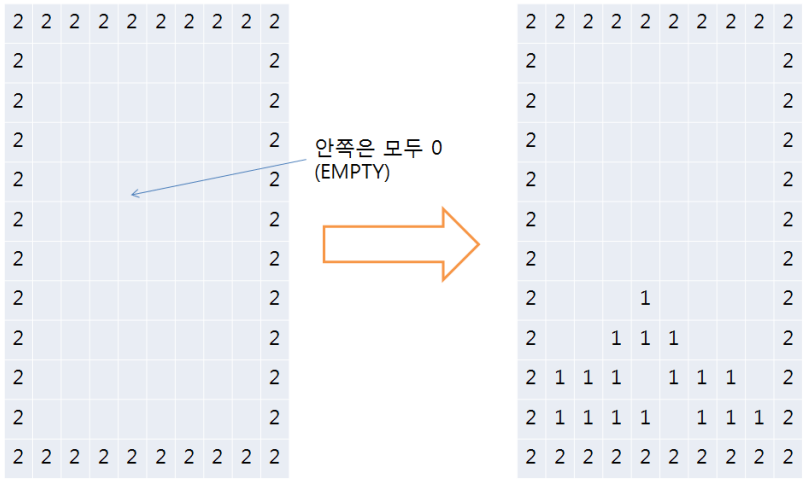
프로그램의 가독성을 향상시키기 위해 board배열이 가지고 있는 설정값 0, 1, 2 대신 열거체를 사용해 보자. 열거체을 정의하는 방법은 간단하다.

```
enum { EMPTY, BRICK, WALL };
```

이와 같이 정의하면 EMPTY는 정수 0, BRICK은 정수 1, WALL은 정수 2가 설정된다. 이것은 정수형 상수로 사용되며, 숫자 0, 1, 2를 사용하는 것보다 정수형 상수 EMPTY, BRICK, WALL을 사용하는 것이 이해하기 빠르고 오류도 줄일 수 있다.

```
enum { EMPTY, BRICK, WALL };

int board[BW+2][BH+2];
        ⋮
    for (x=0;x<BW+2;x++) {
        for (y=0;y<BH+2;y++) {
            board[x][y] = (y==0 || y==BH+1 || x==0 || x==BW+1) ? WALL:EMPTY;
        }
    }
```

위 코드를 실행하면 board배열에 0, 2가 저장되면서 게임판이 아래와 같이 설정된다.



초기화 직후 게임 진행 중

게임이 진행되면 아래쪽에 벽돌이 하나 둘 씩 쌓이는데 이 위치에 1(BRICK)이 기록된다. 게임을 진행하는 코드는 이 배열의 상태를 보고 어디에 벽돌이 쌓여 있는지, 한 줄이 다 채워졌는지를 판단한다.

board 배열의 크기는 게임판 안쪽(벽돌이 쌓이는 영역)보다 폭, 높이를 각각 2씩 더 크게 잡아 주고 상하 좌우에 2(WALL)라는 값을 나열해 두었다. 이 값이 배열의 바깥쪽을 완전히 에워싸고 있기 때문에 게임 진행 코드에서는 GetAround()의 도움을 받아 벽돌이 화면 밖으로 나갈 수 없도록 에러 처리를 할 수 있다. 만약 board 배열이 외부벽에 대한 정보없이 순수하게 게임판에 대한 정보만 가지도록 한다면 이동할 때마다 벽돌의 좌표를 일일이 점검하여 벽돌이 게임판 밖으로 못나가도록 해야 하는 번거로움이 있다.

1.2 게임판 그리기

▌DrawScreen()

게임판을 그리는 함수는 두 개가 작성되어 있는데 DrawScreen()은 화면 전체를 다시 그린다. 0부터 외부벽까지(BW+2, BH+2) 순회하면서 board 배열에 기록된 값대로 타일을 순서대로 출력하기만 하면 된다. 그리고 몇 가지 안내 문자열도 같이 출력했다. DrawBoard()는 벽돌이 움직일 때마다 호출되어 외부벽은 제외하고 벽돌이 움직이는 공간만 출력하는데 board 배열의 1부터 BW+1, BH+1까지만 순회하면 된다. 벽돌이 수평으로 꽉 채워진 줄을 삭제한 후 이 함수를 호출하면 게임판 안쪽만 다시 그릴 수 있다. DrawBoard()는 TestFull()에서 자세히 보도록 한다.

해보기 1 게임판 그리기

```
#include <Turboc.h>

#define BX 5              // 게임판의 위치
#define BY 1
#define BW 10             // 게임판의 크기
#define BH 20

void DrawScreen();

enum { EMPTY, BRICK, WALL };
char *arTile[]={". ","■","□"};
int board[BW+2][BH+2];

void main()
{
     int x,y;

     setcursortype(NOCURSOR);  // 커서를 안보이게 하는 함수
     clrscr();                 // 화면을 깨끗이 지우는 함수

     for (x=0;x<BW+2;x++) {    // board 배열 초기화
          for (y=0;y<BH+2;y++) {
              board[x][y] = (y==0 || y==BH+1 || x==0 ||
x==BW+1) ? WALL : EMPTY;
          }
     }

     DrawScreen();   // 게임화면 그리는 함수 호출
     getch();        // 한 개 문자를 입력받기 위해 대기하는 함수.
                     // (화면을 일시정지하기 위해 사용)
     clrscr();       // 화면을 깨끗이 지우는 함수
}

void DrawScreen()
{
     int x,y;

     for (x=0;x<BW+2;x++) {
          for (y=0;y<BH+2;y++) {
              gotoxy(BX+x*2,BY+y);
```

```
                puts(arTile[board[x][y]]);
            }
        }

        gotoxy(50,3);puts("Tetris Ver 1.0");
        gotoxy(50,5);puts("좌우:이동, 위:회전, 아래:내림");
        gotoxy(50,6);puts("공백:전부 내림");
}
```

setcursortype()는 콘솔화면에서 커서의 모양을 바꾸는 기능을 가지고 있다. 콘솔 어플리케이션 프로그래밍에서 커서가 필요하지 않을 때 사용한다. 원래 터보 C에서 지원하는 함수이고 일반 C에서는 지원하지 않는 함수지만 일반 C에서 사용할 수 있도록 <turboc.h>파일 안에 정의되어 있다.

Ⓕ setcursortype			
함수원형	void setcursortype(int x)		
함수인자	int x	NOCURSOR(0)	커서를 제거한다
		SOLIDCURSOR(1)	한글자 크기의 커서를 그린다
		NORMALCURSOR(2)	보통 모양의 커서로 복원한다
기능	커서의 모양을 결정한다.		

getch()는 키보드로부터 문자 한 개를 입력받는 함수로 키보드에서 문자가 입력될 때까지 대기한다. 이런 특성 때문에 키보드 입력 시까지 대기시키는 목적으로 많이 사용한다. 이 함수와 유사한 함수로 getchar(), getche()가 있다.

■ getch	
헤더파일	conio.h
함수원형	void getch(void)
함수인자	해당없음
기능	입력버퍼에서 1byte를 가져온다.

clrscr함수는 콘솔화면의 텍스트를 지우고 커서를 화면의 좌상단 (1,1)의 위치로 옮긴다.

Ⓕ clrscr	
함수원형	void clrscr(void)
함수인자	해당없음
기능	화면을 지우고 커서를 (1,1)로 옮긴다.

gotoxy함수는 콘솔화면에서 커서를 원하는 위치로 이동하는 함수이다.

Ⓕ gotoxy		
함수원형	void gotoxy(int x, int y)	
함수인자	int x	콘솔화면에서의 가로 위치를 지정
	int y	콘솔화면에서의 세로 위치를 지정
기능	콘솔화면에서 커서를 (x, y)로 이동시킨다.	

1.3 난수함수

전체적인 흐름을 보기 전에 우선 난수함수에 대해서 알아보자. 난수란 무작위 수로 C언어에서 난수를 만들기 위해서는 srand()를 이용하여 난수 발생기를 초기화 한다. srand() 호출시 입력 값에 따라 다른 난수표가 선택된다. 그러나 입력 값이 같으면 같은 난수표가 나오기 때문에 결국 동일한 값들이 나오게 된다. 이러한 문제를 해결하기 위해 일반적으로 입력 값으로 시간정보를 입력해준다. <time.h>에 정의되어 있는 time()의 경우 매 초당 다른 값을 주기 때문에 매 초마다 다른 난수표를 선택하게 되어 srand()를 호출할 때마다 서로 다른 난수를 만들어낼 수 있다. 실제로 대부분의 경우 srand()에는 time(NULL) 값으로 초기화하는 것이 일반적이다.

■ srand	
헤더파일	stdlib.h
함수원형	void srand(unsigned int seed)
함수인자	unsigned int seed
기능	난수 발생기를 초기화한다.

rand()는 srand()에 의해 선택된 난수표에서 하나의 난수를 가져온다. 이때 가져온 값을 %연산자를 사용하면 주어진 값 이하의 난수로 변형할 수 있다.

(value % 100)을 하면 0 ~ 99 사이의 난수를 돌려준다.

(value % 100 + 1)을 하면 1 ~ 100 사이의 난수를 돌려준다.

(value % 34 + 1979)을 하면 1979 ~ 2012 사이의 난수를 돌려준다.

■ rand	
헤더파일	stdlib.h
함수원형	int rand(void)
함수인자	해당없음
기능	난수표에서 난수를 선택한다.

해보기 2 난수 만들기

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void main ()
{
    printf ("첫번째 숫자: %d\n", rand() % 100);
    srand ( time(NULL) );
    printf ("무작위 수: %d\n", rand() % 100);
    srand ( 1 );
    printf ("첫번째 수와 비교: %d\n", rand() %100);
}
```

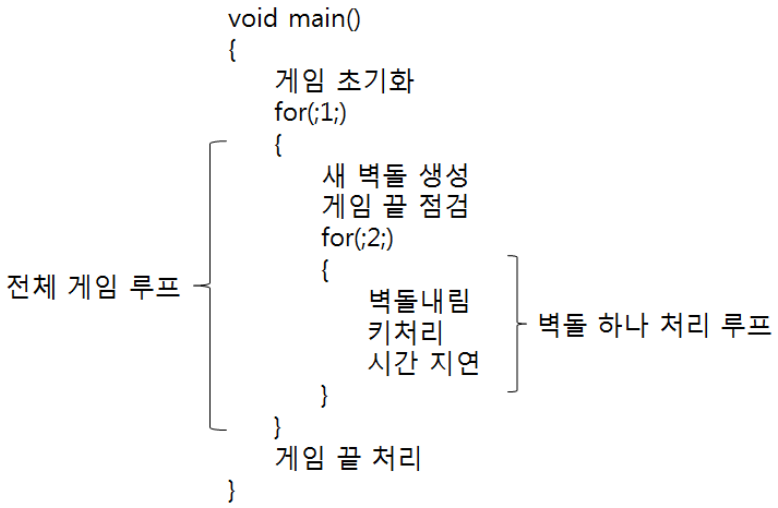
1.4 메인함수 분석하기

다음은 main 함수 코드의 일부다.

```
#include <Turboc.h>
int brick, rot;      // brick은 벽돌번호, rot은 각 벽돌의 회전번호

void main()
```

```
{
    int nFrame, nStay;                      // 벽돌이 내려가는 속도
        ⋮
    srand((unsigned)time(NULL));            // 난수발생기 초기화
        ⋮
    nFrame=20;                      // 벽돌이 떨어지는 속도변수

    for (;1;) {
        // 벽돌 선택하기
        brick=rand() % (sizeof(Shape)/sizeof(Shape[0]));

        nx=BW/2;                    // 벽돌의 초기 x좌표
        ny=2;                       // 벽돌의 초기 y좌표
        rot=0;                      // 벽돌의 회전모양 번호
        PrintBrick(TRUE);           // 벽돌 그리기 함수 호출

        // 벽돌주변이 비었는지 체크
        if (GetAround(nx,ny,brick,rot) != EMPTY) break;
        nStay=nFrame;
        for (;2;) {
            if (--nStay == 0) {
                nStay=nFrame;
                // 벽돌이 더 내려갈 수 없으면(false)
                // 루프 빠져나감
                if (MoveDown()) break;
            }
            if (ProcessKey()) break;
            delay(1000/20);
        }
    }
    clrscr();
    gotoxy(30,12);puts("G A M E  O V E R");
    setcursortype(NORMALCURSOR);
}
```

main()의 전체 구조는 다음과 같다.



▌for(;1;)루프

전체적으로 두 개의 무한 루프로 구성되어 있는데 무한 루프에 for(;1;), for(;2;) 등으로 이름을 붙여 놓았다. 루프에 들어가기 전에 커서 숨김, 난수 생성기 초기화, 화면 삭제 등의 처리를 하고 board 배열의 바깥쪽을 벽으로 초기화한 후 DrawScreen()를 호출하여 화면 전체를 모두 그린다. 벽돌이 떨어지는 속도인 nFrame값을 20으로 초기화한 후 for(;1;)루프로 진입하여 게임을 진행한다.

for(;1;)루프는 게임 전체를 감싸는 루프이고 for(;2;)루프는 벽돌 하나가 바닥에 닿을 때까지의 루프이다. for(;1;)에서 새로운 벽돌을 위쪽 중앙 좌표인 (BW/2, 2) 위치에 생성하고 이 벽돌을 일단 화면에 출력한다. 새로 만드는 벽돌의 번호는 물론 난수로 선택하였으며 회전 모양은 0번부터 시작한다.

벽돌을 생성한 후 게임 끝 처리를 하는데 만약 새로 생성한 벽돌의 주변이 공백이 아니라면 다른 벽돌이 이미 화면 위쪽까지 가득 차 있는 상태이므로 이때가 게임을 끝낼 때이다. GetAround()로 새로 생성한 벽돌의 주변을 점검해 보고 EMPTY가 아니면 for (;1;)루프를 탈출하도록 했다.

▌for(;2;)루프

for(;2;)루프의 끝에서 0.05초씩 시간을 지연시키고 있으므로 이 루프는 초당 20번 반복된다. delay()의 인자값의 단위는 millisecond로 1000이면 1초를 지연시켜준다. 그러므로 delay(1000/20)은 0.05초를 지연시킨다.

벽돌 하나가 내려오는 주기인 nFrame이 20으로 초기화되어 있고 nStay가 이 값을 거꾸로 카운트하면서 0이 될 때 벽돌을 한 칸 내린다. 그래서 벽돌은 정확하게 1초에 한 번 한 칸 아래로 이동할 것이다. 이렇게 하는 이유는 delay(1000)으로 설정하여 1초에 한 번씩 벽돌을 이동시킬 수도 있지만 그럴 경우 키 입력도 1초에 한 번밖에 못 받으므로 반응성이 떨어지기 때문이다. 그래서 시간을 1/20초로 분할하고 벽돌은 20프레임에 한 번씩 움직이되

키는 매 프레임마다 입력받을 수 있도록 했다.

for(;2;)루프는 벽돌이 생성되어 게임판 위에서부터 아래로 이동할 때까지 계속 반복된다. 시간이 지나 벽돌이 바닥에 닿으면 MoveDown()이 TRUE를 리턴하며, 사용자가 직접 벽돌을 바닥으로 떨어뜨리면 ProcessKey()가 TRUE를 리턴하는데 이 두 경우에 break로 for(;2;)루프를 탈출한다. 이 루프를 벗어나면 다시 for(;1;)루프의 처음으로 돌아가 새로운 벽돌을 생성하며 이 과정은 새로 생성한 벽돌의 주변이 비어있지 않을 때까지 반복된다.

키 입력 처리는 ProcessKey()에 위임되어 있다. 이 부분은 특별히 반복이 필요해서 함수로 분리시킨 것은 아니며 키 입력 처리를 별도의 함수로 분리시킴으로써 main()의 부담을 줄이고자 한 것이다. main은 전체 프로그램의 흐름을 통제하는 중요한 함수이기 때문에 세부적인 처리는 가급적이면 사용자정의 함수를 이용하여 분리하는 것이 좋다.

02 테트리스 벽돌 만들기

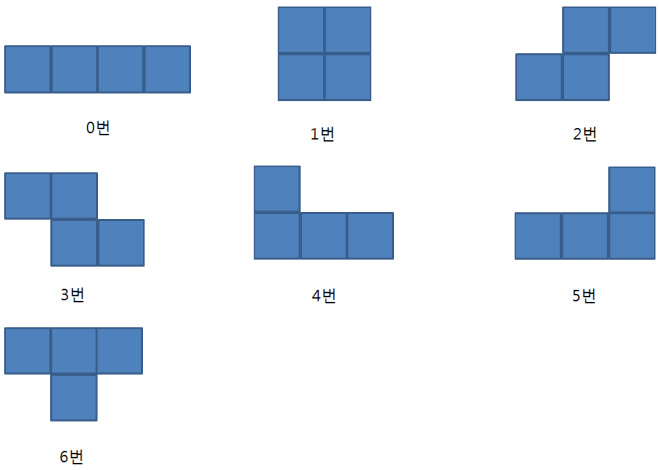
```
#include <Turboc.h>

void PrintBrick(BOOL Show);

struct Point {
    int x,y;
};
struct Point Shape[][4][4]={
    { {0,0,1,0,2,0,-1,0}, {0,0,0,1,0,-1,0,-2},
      {0,0,1,0,2,0,-1,0}, {0,0,0,1,0,-1,0,-2} },    //0번 벽돌
    { {0,0,1,0,0,1,1,1}, {0,0,1,0,0,1,1,1},
      {0,0,1,0,0,1,1,1}, {0,0,1,0,0,1,1,1} },       //1번 벽돌
    { {0,0,-1,0,0,-1,1,-1}, {0,0,0,1,-1,0,-1,-1},
      {0,0,-1,0,0,-1,1,-1}, {0,0,0,1,-1,0,-1,-1} }, //2번 벽돌
    { {0,0,-1,-1,0,-1,1,0}, {0,0,-1,0,-1,1,0,-1},
      {0,0,-1,-1,0,-1,1,0}, {0,0,-1,0,-1,1,0,-1} }, //3번 벽돌
    { {0,0,-1,0,1,0,-1,-1}, {0,0,0,-1,0,1,-1,1},
      {0,0,-1,0,1,0,1,1}, {0,0,0,-1,0,1,1,-1} },    //4번 벽돌
    { {0,0,1,0,-1,0,1,-1}, {0,0,0,1,0,-1,-1,-1},
      {0,0,1,0,-1,0,-1,1}, {0,0,0,-1,0,1,1,1} },    //5번 벽돌
    { {0,0,-1,0,1,0,0,1}, {0,0,0,-1,0,1,1,0},
      {0,0,-1,0,1,0,0,-1}, {0,0,-1,0,0,-1,0,1} },   //6번 벽돌
```

```
};

int nx, ny;
int brick, rot;

void main()
{
            ⋮
}

void PrintBrick(BOOL Show)
{
      int i;

      for (i=0;i<4;i++) {

gotoxy(BX+(Shape[brick][rot][i].x+nx)*2,BY+Shape[brick][rot][i
].y+ny);
            puts(arTile[Show ? BRICK:EMPTY]);
      }
}
```

2.1 벽돌 정의

▌벽돌 모양

테트리스 게임의 주인공은 7개의 벽돌들인데 이 벽돌들의 모양이 어떻게 정의되어 있는지 분석해 보자. 컴퓨터에서는 모든 것들이 숫자이기 때문에 각 벽돌에도 숫자를 붙여야 하며 벽돌의 모양도 숫자로 표현한다. 7개의 벽돌은 다음과 같은 모양을 가지며 번호가 부여되어 있다.



▌구조체 배열변수 Shape

게임 중에 생성되는 벽돌들은 어차피 난수로 선택되므로 순서는 별 의미가 없다. 각 벽돌 당 최대 네 방향으로 회전 가능하므로 각 회전 방향별로 4개의 모양을 정의해야 하며 각각의 모양은 또 4개의 단위 벽돌로 구성된다. 4개 단위 벽돌의 좌표가 모여 하나의 모양을 구성하고 4개의 모양이 모여야 하나의 벽돌이 정의되며 이런 벽돌이 7개 있다. 그래서 다음과 같은 3차원 구조체 배열로 전체 벽돌의 모양을 정의한다.

```
struct Point {
        int  x, y;
};
struct Point Shape[][4][4]={
        ....
};
```

좌표를 정의하는 구조체를 먼저 정의한 후 구조체의 이름을 Point라고 붙였다(9주차 복합자료형 참고). 구조체가 정의됨으로써 우리는 Point라는 이름의 새로운 데이터형을 얻게 되었다. 이런 구조체의 3차원 배열 Shape를 정의한 것이다. 여기서 Point는 구조체로 정의된 사용자정의 데이터형이고 Shape는 배열변수명이다. Shape 배열의 각 첨자는 다음과 같은 의미를 가진다.

$$\text{Shape[벽돌번호][회전번호][일련번호]}\begin{cases}.x - \text{멤버변수} \\ .y - \text{멤버변수}\end{cases}$$

첫 번째 첨자가 벽돌의 번호, 두 번째 첨자가 회전 번호이며 각 모양별로 4개의 단위 벽돌 좌표를 가지는데 세 번째 첨자가 이 좌표들의 일련번호이다. 최종 배열 요소는 Point 구조체이며 이 구조체의 멤버 (x, y)는 단위 벽돌의 좌표값이다. 5번 벽돌의 2번 회전 모양에 대한 좌표들을 얻고 싶으면 Shape[5][2][0] ~ Shape[5][2][3]까지의 좌표를 읽어 조립해 보면 된다.

Shape 배열은 아주 많은 숫자들로 구성되어 있는데 이 숫자들이 어떤 의미를 가지는지 살펴보자. 다음은 2번 벽돌의 모양을 정의하는 초기값이다.

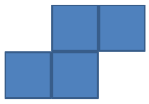
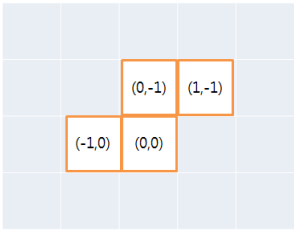
벽돌모양	벽돌의 좌표코드	회전수
 2번	{ {0,0,−1,0,0,−1,1,−1},	회전 없음(최초 생성시)
	{0,0,0,1,−1,0,−1,−1},	반시계 방향 한 번 회전
	{0,0,−1,0,0,−1,1,−1},	반시계 방향 두 번 회전
	{0,0,0,1,−1,0,−1,−1} },	반시계 방향 세 번 회전

표 2 벽돌 좌표코드 분석

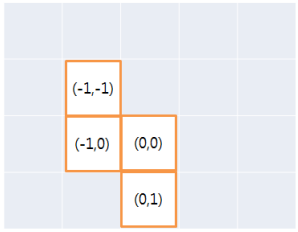
4개의 부분 배열로 구성되는데 각 부분 배열은 회전 모양 하나를 정의한다. 벽돌이 최초 생성되었을 때의 모양인 첫 번째 부분 배열 {0,0,−1,0,0,−1,1,−1}은 두 개의 숫자끼리 한 쌍을 이루어 4개의 Point 구조체를 초기화한다. 두 개씩 괄호로 묶어 {{0,0},{−1,0},{0,−1},{1,−1}}처럼 표현하면 이해하기 쉬워지는 반면 코드가 복잡해지므로 가독성이 떨어져 중간의 괄호들은 생략했다. 첫 번째 좌표 (0,0)이 이 모양의 기준점이며 회전할 때의 중심점이 된다. 나머지 세 좌표는 기준점으로부터의 상대적 거리 좌표이다.

각 좌표를 순회하면서 기준점 주변에 타일을 출력하면 하나의 벽돌 모양이 만들어진다. 2번째 부분 배열 {0,0,0,1,−1,0,−1,−1}은 벽돌이 최초로 생성되고 반시계 방향으로 90도 회전(한 번 회전)했을 때의 모양을 정의한다. 부분 배열을 구성하는 4개의 좌표를 전개해 보면 다음과 같은 벽돌 모양이 만들어 진다.



{0,0, -1,0, 0,-1, 1,-1}

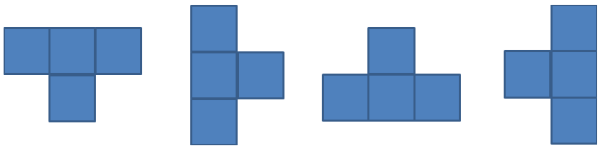
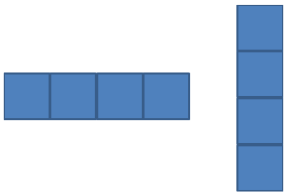
최초 생성시



{0,0, 0,1, -1,0, -1,-1}

반시계 방향으로 1번 회전

회전 모양 개수는 각 벽돌별로 다른데 0번 벽돌은 2가지 모양밖에 없다. 그래서 0, 1번 회전 모양이 2, 3번과 같으며 2, 3번 모양을 정의하는 값은 0, 1번과 동일하게 되어 있다. 2번, 3번 벽돌도 2가지 회전 모양을 가지며 4, 5, 6번 벽돌은 각 회전 모양이 모두 다르다.

벽돌의 회전형태	모양의 개수
	4가지 모양
	2가지 모양
	1가지 모양

▌표 1 표 제목

1번 벽돌은 한 가지 모양만 정의하면 된다. 그래서 Shape[1][0] ~ Shape[1][3]까지의 부분 배열은 모두 같은 초기값들을 반복적으로 정의한다. Shape 배열에 들어 있는 좌표값들로부터 벽돌의 모양을 직접 그려 보면 이 구조체 배열의 초기값을 이해할 수 있을 것이다. 이런 모양을 만들 때는 모눈 종이에 원하는 도형을 그려 놓고 기준점을 잡은 후 나머지 벽돌의 상대적 좌표를 나열해 보면 된다.

2.2 PrintBrick 함수

PrintBrick	
함수원형	void PrintBrick(BOOL)
함수인자	BOOL show
기능	벽돌을 그린다.

▌BOOL형

우선 BOOL형 자료형에 대해서 알아보자. BOOL형은 TRUE 또는 FALSE를 갖는 자료형으로 <windows.h>에 정의되어 있다. <windows.h>를 살펴보면 BOOL은 다음과 같이 정의되어 있다.

```
typedef int     BOOL;
```

C++컴파일러는 기본적으로 bool타입을 제공하지만 C에서는 <windows.h>를 삽입해야만 사용할 수 있다. C++의 bool과 <windows.h>의 BOOL은 참(1) 또는 거짓(0)을 나타내기 위해 사용한다는 측면에서는 같지만, bool은 1바이트 BOOL은 4바이트라는 차이가 있다.

▌PrintBrick 함수

벽돌을 화면에 그리거나 벽돌의 주변을 검사하는 함수는 모두 Shape 배열을 순회하면서 각 단위 벽돌의 좌표값을 참조한다. 실제로 이 정보를 참조하여 벽돌을 그리는 PrintBrick()를 분석해 보자. 이 함수는 전역변수 brick, rot가 지정하는 벽돌 모양을 현재 위치 nx, ny에 출력하거나 지운다.

```
void PrintBrick(BOOL Show)
{
        int  i;

        for  (i=0;i<4;i++) {
            gotoxy(BX+(Shape[brick][rot][i].x+nx)*2,
                    BY+Shape[brick][rot][i].y+ny);
            puts(arTile[Show ? BRICK : EMPTY]);
        }
}
```

gotoxy()에서 Shape 배열 참조식을 빼면 gotoxy(BX+nx*2,BY+ny);가 되는데 이 좌표는 nx, ny의 화면상 좌표이며 Shape 배열의 기준점 (0,0), 즉 벽돌이 최초에 생성되었을 때의 좌표가 된다. 0~3까지 루프를 돌면서 단위 벽돌의 상대적 좌표를 참조하면 벽돌이 화면에 그려지며 공백 타일을 출력하면 벽돌이 지워진다. 3차원 배열 참조문이 좀 길어서 복잡해 보이지만 알고 보면 무척 간단한 문장이다.

2.3 GetAround 함수

▌GetAround 함수

벽돌 주변에 무엇이 있는지를 조사하는 GetAround()도 비슷한 구조를 가

GetAround		
함수원형	int GetAround(int x, int y, int b, int r)	
함수인자	int x	현재 x좌표
	int y	현재 y좌표
	int b	벽돌 번호
	int r	벽돌의 회전 번호
기능	벽돌의 주변이 공백인지, 다른 벽돌이 있는지 또는 벽인지를 판단하고 공백이면 EMPTY, 다른 벽돌이 있으면 BRICK, 벽이면 WALL을 반환한다.	

진다. 단, 이 함수는 이동 중인 벽돌의 주변 정보를 조사해야 하므로 전역변수를 참조하지 않고 조사할 벽돌의 정보를 인수로 전달받는다.

```
int GetAround(int x, int y, int b, int r)
{
        int  i, k=EMPTY;

        for  (i=0;i<4;i++) {
                k=max(k, board[x+Shape[b][r][i].x]
                        [y+Shape[b][r][i].y]);
        }
        return  k;
}
```

■ max	
헤더파일	stdlib.h
함수원형	#define max(a, b) (((a) > (b)) ? (a) : (b))
기능	두 수중 큰 수를 반환한다.

k는 최초 EMPTY로 초기화되고 벽돌이 있는 좌표를 순회하면서 최대값을 조사하는데 EMPTY보다는 BRICK을, BRICK보다는 WALL을 더 큰 값으로 평가한다. 만약 이 위치에 아무 것도 없다면 EMPTY가 리턴될 것이고 벽돌

과 충돌했으면 BRICK이 리턴되고 벽과 충돌되었다면 WALL이 리턴될 것이다. 이 함수가 조사하는 주변값은 벽돌의 이동 가능 여부, 게임의 계속 여부 등을 판단하는 중요한 기준이 된다.

이 예제는 벽돌의 모양을 정의하기 위해 3차원의 좌표 구조체를 사용했는데 직관적으로 쉽게 분석되지 않는 다소 어려운 구조이다. 이 외에도 벽돌의 모양을 정의하는 다양한 방법을 생각해 볼 수 있다. 어떤 방법들이 가능한지 생각해 보고 각각의 장단점을 비교해 보자.

벽돌코드의 개선

① Shape 배열의 요소인 Point 구조체는 좌표값 하나를 기억하는데 상대 좌표는 -2~2 사이의 좁은 범위에 있으므로 x, y를 int형으로 정의하는 것은 낭비가 심하다. 현재 Shape 구조체의 총 크기는 896바이트이며 x, y를 short로 선언하면 448 바이트가 되어 절반으로 줄어들고 char로 바꾸면 224바이트가 된다. 좀 더 메모리를 절약하고 싶다면 Shape를 아예 char의 3차 배열로 만들고 각 요소의 상, 하위 4비트에 x, y좌표를 저장할 수도 있는데 이 경우 Shape는 또 절반이 되어 112바이트가 될 것이다. 메모리 절약 효과는 분명히 있지만 이런 작은 게임에서는 굳이 그럴 필요가 없어 일반적인 정수형인 int형을 사용했다.

② Shape 배열의 두 번째 첨자 크기가 4로 정의되어 있는데 회전 모양이 2가지나 1가지 밖에 없는 벽돌은 기억 장소를 낭비하는 경향이 있다. 같은 모양을 가지는 벽돌에 대해 동일한 초기값을 반복하는 대신 벽돌별로 회전 모양수를 따로 정의하고 꼭 필요한 만큼만 모양을 정의하는 방법을 생각해 볼 수 있다. 그러나 Shape 배열이 어차피 직사각형 배열이기 때문에 초기값을 생략해도 메모리 절약 효과는 없는 셈이며 코드만 복잡해져서 반복되는 값을 계속 써 주는 것이 더 좋다.

③ 각 모양의 기준점인 첫 번째 좌표는 항상 (0,0)이다. 이 좌표는 모든 모양에 공통적으로 존재하므로 초기값에서 제외하고 코드에서 (0,0)을 따로 처리할 수도 있다. 그러나 이렇게 되면 PrintBrick(), GetAround()에서 루프 하나로 단위 벽돌을 순회할 수 없고 기준점에 대해 별도의 코드를 작성해야 하므로 오히려 더 번거로워진다. 즉, 데이터는 작아지는 대신 코드의 부담이 늘어나게 되는데 벽돌의 개수가 아주 많다면 몰라도 이 정도 수준에서는 현재의 방법이 더 좋다.

④ 기준점으로부터의 상대좌표를 지정하는 방식 대신 일정한 크기의 비트맵을 정의하고 이 비트맵을 화면에 출력할 수도 있다. 이 방법은 모양을 디자인하고 수정하기에는 편리하지만 자료의 양이 많은 것이 단점이고 x, y 각 방향에 대해 이중 루프로 비트맵을 순회해야 하므로 코드도 조금 더 복잡해진다. 일반적으로 가장 많이 채택하는 방법이기는 하지만 이 예제는 상대 좌표 방식을 사용했다.

⑤ 각 벽돌별로 회전 모양 4가지를 배열에 미리 정의해 놓았는데 한 가지 모양만 정의하고 실행 중에 나머지 회전 모양을 계산해서 만들 수도 있다. 기준점은 그대로 두고 나머지 좌표는 반시계 방향으로 이동시키기만 하면 된다. 이렇게 하면 벽돌 모양을 정의하는 데이터가 작아지고 새로운 벽돌을 추가하기 쉬워지지만 코드는 더 큰 부담을 지게 된다.

이와 같이 벽돌의 모양을 정의하는 이런 간단한 문제도 여러 가지 해결책이 있다는 것을 알 수 있다. 개발자는 이런 다양한 방법 중 작성하는 프로그램에 가장 적합한 알고리즘을 선택해야 한다. 여기에서 선택한 방법은 처음 배열을 작성하는 수고를 하더라도 가급적이면 코드의 부담을 줄이는 쪽이다.

03 벽돌의 이동

```
BOOL ProcessKey()
{
	int ch, trot;

	if (kbhit()) {
		//입력버퍼에서 1바이트(아스키코드)를 가져온다.
		ch=getch();
		//ch 값이 224 또는 0 이면 다음 줄 실행
		if (ch == 0xE0 || ch == 0) {
			//다시 1바이트(스캔코드)를 가져온다.
			ch=getch();
			switch (ch) {
			case LEFT:                    //왼쪽 키 눌렀을 때
				if (GetAround(nx-1, ny, brick, rot) == EMPTY) {
					PrintBrick(FALSE);    //벽돌 지우기
					nx--;
```

```
                    PrintBrick(TRUE);      //벽돌 그리기
                }
                break;
            case RIGHT:                    //오른쪽 키 눌렀을 때
                if (GetAround(nx+1, ny, brick, rot) == EMPTY) {
                    PrintBrick(FALSE);
                    nx++;
                    PrintBrick(TRUE);
                }
                break;
            case UP:                       //벽돌 회전
                trot=(rot == 3 ? 0:rot+1);
                if (GetAround(nx, ny, brick, trot) == EMPTY) {
                    PrintBrick(FALSE);
                    rot=trot;
                    PrintBrick(TRUE);
                }
                break;
            case DOWN:                     //벽돌 한 번에 내리기
                if (MoveDown()) {
                    return TRUE;
                }
                break;
            }
        } else {
            switch (ch) {
            case ' ':                      //스페이스바 키 입력시
                while(MoveDown()==FALSE) {;}
                return TRUE;
            }
        }
    }
    return FALSE;
}
```

3.1 PressKey 함수

PressKey

함수원형	BOOL PressKey(void)
함수인자	해당 없음
기능	키입력을 받아 그에 맞는 처리를 한다.

▌스캔코드

벽돌을 이동시키는 대부분의 루틴은 사용자가 키를 입력할 때인 ProcessKey()에 포함되어 있다. 이 함수는 우선 kbhit()를 호출하고 눌러진 키가 있을 때만 키 입력을 처리하며 그렇지 않을 경우는 바로 리턴하여 게임을 계속 진행하도록 한다. 사용자가 가만히 있어도 벽돌은 계속 한 칸씩 내려와야 하므로 입력을 무한히 대기하는 getch()를 바로 호출해서는 안 된다. kbhit()는 키보드로부터 입력된 키가 있는지 조사하는 역할을 한다.

■ kbhit	
헤더파일	conio.h
함수원형	int getch(void)
함수인자	해당없음
기능	키보드로부터 입력된 키가 있는지 조사한다.

그리고 다음 코드를 보자.

```
ch=getch();
if (ch == 0xE0 || ch == 0) {
    ch=getch();
```

이 코드를 이해하기 위해 스캔코드에 대해 알아보자(부록 스캔코드 참고). 일반적으로 문자입출력을 위해서 우리는 아스키코드를 사용한다. 그리고 이 아스키코드 1byte는 이미 모두 영문자와 특수문자들로 예약되어 있다. 그래서 키보드에 존재하는 ctrl, alt, tab, F1~F12, 방향키, num lock 등과 같은 확장키코드 값은 아스키코드로는 표현할 수 없다. 이 확장키를 표현하기 위해 스캔코드(2byte)를 사용한다. 스캔코드 중 문자영역은 아스키코드와 1:1로 대응하기 때문에 문자를 입력할 때에는 별 차이가 없다.

(참고: http://support.microsoft.com/kb/57888/ko)

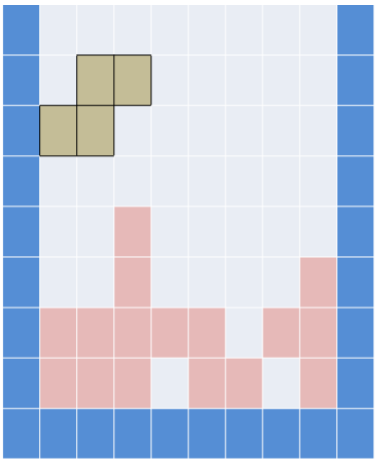
getch()함수는 키입력이 들어오면 입력버퍼에서 들어온 스캔코드값을 확인한다. 문자코드들은 1byte이며 그 값을 바로 돌려주면서 입력버퍼를 비운다. 그러나 확장키가 입력되면 2byte가 입력되므로 이중 1byte만 돌려주는데 이

때 반환된 값이 224(0xE0)이거나 0이다. 그리고 다시 getch()를 호출하면 입력버퍼에 남은 1byte를 가져와 돌려준다. 이때 남은 1byte의 값이 LEFT 키일 경우 75, UP키 일 경우 72이다.

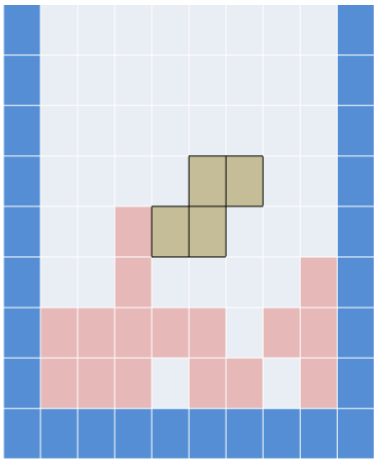
만약 K(아스키코드 75)를 입력한다해도 이것은 문자코드(1byte)이기 때문에 리턴값이 224나 0이 아니므로 if문의 조건문을 충족하지 못해 두 번째 getch()함수가 실행되지 않는다.

▌벽돌의 이동

입력된 방향키에 따라 대응되는 방향으로 벽돌을 이동시키는데 대표적으로 왼쪽 이동의 경우를 보자. 왼쪽 키가 눌러지면 GetAround()를 호출하여 현재 좌표보다 한 칸 왼쪽, 즉 (nx-1, ny)위치에 무엇이 있는지를 조사한다. 만약 이 위치에 벽이나 쌓여져 있는 벽돌이 있다면 더 이상 왼쪽으로 이동할 수 없다.



왼쪽이 벽인 상태



왼쪽이 벽돌인 상태

GetAround()로 왼쪽 위치를 조사한 결과가 빈 공간(EMPTY)일 때만 왼쪽으로 이동할 수 있다. 이동하는 방식은 아주 간단한다. PrintBrick(FALSE)로 현재 위치의 벽돌을 지운 후, nx를 1감소시키고 PrintBrick(TRUE)로 새 위치에 다시 출력하면 된다. 쉽게 말하면 현재 위치의 벽돌을 지우고 왼쪽에 다시 출력하는 것이다. 오른쪽 이동도 점검하는 좌표와 nx가 1증가한다는 것만 다를 뿐 왼쪽 이동과 다를 바 없다.

▌벽돌의 회전

회전의 경우는 trot에 다음 회전시킬 모양을 먼저 구하고 회전했을 때 주변에 무엇이 있는지를 조사한다.

```
trot=(rot == 3 ? 0:rot+1);
// trot : (0 ->) 1 -> 2 -> 3 -> 0 -> 1 -> 2 -> ...
```

이 경우도 회전한 결과 벽이나 벽돌이 부딪힌다면 회전할 수 없다. 회전에 방해가 되지 않는 충분한 공간이 있을 때만 회전할 수 있다. 회전 모양은 단순히 증감하기만 하는 것이 아니고 순환하기 때문에 임시 변수 trot에 다음 회전 모양(trot)을 미리 구해 놓고 회전 가능성을 점검(GetAround())하도록 했다. 회전 가능하다면 현재 위치에 trot모양으로 다시 벽돌을 출력한다.

3.2 MoveDown 함수

```
BOOL MoveDown()
{
    if (GetAround(nx, ny+1, brick, rot) != EMPTY) {
        TestFull();
        return TRUE;
    }
    PrintBrick(FALSE);
    ny++;
    PrintBrick(TRUE);
    return FALSE;
}
```

▌벽돌 내리기

벽돌을 아래쪽으로 이동시키는 동작은 사용자의 키 입력이 있을 때뿐만 아니라 시간이 경과할 때도 주기적으로 처리해야 하므로 MoveDown()이라는 별도의 함수로 분리되어 있다.

MoveDown	
함수원형	BOOL MoveDown(void)
함수인자	해당 없음
기능	벽돌을 한 칸 씩 아래로 내린다.

아래쪽으로 내리는 동작도 왼쪽, 오른쪽 이동과 유사하다. 다만, 한 칸 아래(ny+1)를 점검(GetAround())한 후 바닥에 닿은 경우 채워진 줄 제거를

위해 TestFull()를 호출한다는 것이 다르다. 또한 바닥에 닿은 경우 TRUE를 리턴하여 main()의 for(;2;)루프를 탈출하도록 한다. for(;2;)루프는 벽돌 하나가 바닥에 닿을 때까지의 루프이므로 MoveDown()이 TRUE를 리턴하면 즉시 루프를 탈출하여 새로운 벽돌을 만들도록 해야 한다. 빈 공간에서 한 칸 아래로 이동만 했을 경우는 FALSE를 리턴하여 for(;2;)루프를 계속 돌도록 한다.

▌벽돌 한 번에 내리기

MoveDown() 함수는 세 군데서 호출되는데 main()의 for(;2;)루프에서 1초가 경과되었을 때, 그리고 아래쪽 커서 이동키를 눌렀을 때 한 번씩 호출되며 스페이스 키를 눌렀을 때는 바닥에 닿을 때까지 이 함수가 반복적으로 호출되어 이동 중인 벽돌을 바닥까지 한 번에 내린다.

```
while(MoveDown()==FALSE) {;}
return TRUE;
```

벽돌을 한 번에 내릴 때는 MoveDown()이 TRUE를 리턴할 때까지 계속 호출해 대기만 하면 된다. 그래서 while문으로 MoveDown()이 FALSE를 리턴하는 동안 계속 반복하는데 이 반복문은 반복 자체가 목적이며 반복 중에 따로 할 일이 없다.

04 벽돌의 제거

4.1 TestFull 함수

```
void TestFull()
{
    int i,x,y,ty;

    for (i=0;i<4;i++) {
```

```
board[nx+Shape[brick][rot][i].x][ny+Shape[brick][rot][i].y]=BRI
CK;
    }

    for (y=1;y<BH+1;y++) {
        for (x=1;x<BW+1;x++) {
            if (board[x][y] != BRICK) break;
        }
        if (x == BW+1) {
            for (ty=y;ty>1;ty--) {
                for (x=1;x<BW+1;x++) {
                    board[x][ty]=board[x][ty-1];
                }
            }
            DrawBoard();
            delay(200);
        }
    }
}
```

▌벽돌이 바닥에 닿았는지 점검하기

수평으로 한 줄이 다 채워졌을 때는 가득 찬 줄을 제거해야 하는데 이는 TestFull()에서 처리한다. 이 함수는 MoveDown()에서 벽돌이 바닥에 닿았을 때만 호출되는데 벽돌이 허공에서 이동 중일 때는 board 배열 상에 변화가 없으므로 벽돌 제거 조건이 성립되지 않기 때문이다.

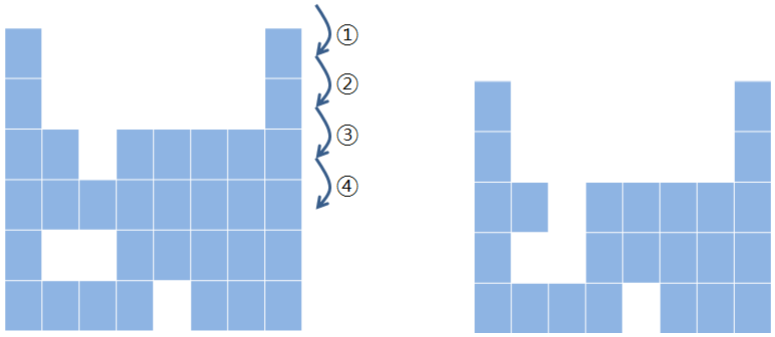
TestFull()는 우선 이동 중인 벽돌을 board 배열에 기록한다. 새로운 벽돌이 착륙했음을 게임판에 먼저 기록해야 게임판에 변화가 발생하고 제거 점검도 할 수 있다. board 배열에서 현재 위치(nx, ny)에 해당하는 벽돌의 위치를 찾아 전부 BRICK으로 바꿔 주면 된다. 벽돌을 화면에 출력할 때는 x 좌표에 2를 곱해 주어야 하지만 배열 상에서 조작할 때는 x좌표를 그냥 사용하면 된다.

바닥에 닿은 벽돌을 board 배열에 기록하면 새로 도착한 벽돌로 인해 채워진 줄이 발생할 수 있으므로 이때마다 제거 점검을 한다. 수직으로 제일 윗줄부터 검사해 나가되 한 줄이 모두 BRICK이면 이 줄은 제거해야 한다. 수평

으로 같은 줄에 있는 칸을 모두 검사하여 BRICK이 아닌 칸이 발견되면 루프를 빠져 나가되 끝까지 루프를 다 돌았으면(x == BW+1) 이 줄은 벽돌로 가득 찬 것으로 판단할 수 있다.

▌한 줄 제거하기

한 줄 제거는 두 단계의 처리로 이루어지는데 우선 board 배열에서 제거될 벽돌의 정보를 삭제한다. 한 줄이 삭제되면 제거되는 줄 위쪽의 벽돌이 한 칸씩 아래로 내려와야 하므로 제거될 줄부터 위쪽으로 루프를 돌며 윗줄로 아랫줄을 덮어쓰면 된다. 복사 방향이 위에서 아래로 이므로 반드시 아래에서 위로 루프를 돌아야 하는데 방향이 바뀌면 제일 윗줄이 반복적으로 아랫줄로 복사될 것이며 채워진 줄 위쪽의 벽돌은 한꺼번에 사라져 버릴 것이다.



board 배열 상에서 한 줄을 제거한 후 화면상에서도 이 줄을 제거하는데 방법은 아주 간단한다. DrawBoard()를 호출하면 게임판 안쪽만 board 배열을 참조하여 완전히 다시 그려 주므로 복잡한 이동 처리를 할 필요가 없다. 화면에 이미 출력된 벽돌을 일일이 이동시키는 것보다는 다시 그리는 것이 더 간단하다.

4.2 DrawBoard 함수

```
void DrawBoard()
{
    int x,y;

    for (x=1;x<BW+1;x++) {
        for (y=1;y<BH+1;y++) {
```

```
                gotoxy(BX+x*2,BY+y);
                puts(arTile[board[x][y]]);
            }
        }
}
```

DrawBoard()는 게임판의 내부를 돌면서 board 배열의 정보에 따라 빈 공백 또는 벽돌을 그려준다. 따라서 가로로는 x=1부터 x<BW+1까지, 세로로는 y=1부터 y<BH+1까지만 순회한다. arTile 배열에 있는 벽돌과 벽의 타일인 특수문자("■", "□")는 한글과 같은 2바이트 문자이다. 이 특수문자들은 콘솔화면에서도 가로방향으로 2칸을 차지하므로 특수문자 1개를 그린 후 바로 옆에 다른 특수문자를 그리기 위해서는 x좌표로 2칸을 좌표이동을 해야 한다.

```
gotoxy(BX+x*2, BY+y);
```

다시 TestFull()함수로 넘어가자.

```
delay(200);
```

한 줄을 지울 때마다 0.2초씩 시간을 지연시켜 여러 줄이 동시에 제거될 때 순서대로 한 줄씩 사라지는 것을 보여준다. 일종의 애니메이션 효과처럼 보이고 사용자에게 무슨 일이 일어나고 있는지를 분명히 알려줄 수 있다. 만약 4줄을 한꺼번에 없애 버린다면 화면상의 변화가 너무 급격해서 사용자가 당황스러워 할 것이다.

Ⅲ 부록

01 키보드 스캔코드

스캔코드는 키보드에 부여되어 있는 고유한 번호이다. 이 스캔코드를 이용하면 ASCII 값이 같을지라도 키조합이 다른 경우를 찾을 수 있다. 예를 들면 백스페이스와 Ctrl+H는 둘 다 ASCII값으로 8을 가지고 있다. 그러나 분명히 두 키는 다르다. 이를 구분하고자 할 때 스캔코드를 이용한다.

```
Const SCANKEY_ESC = 1(0x01)                ' ESC
Const SCANKEY_1 = 2(0x02)                  ' 1
Const SCANKEY_2 = 3(0x03)                  ' 2
Const SCANKEY_3 = 4(0x04)                  ' 3
Const SCANKEY_4 = 5(0x05)                  ' 4
Const SCANKEY_5 = 6(0x06)                  ' 5
Const SCANKEY_6 = 7(0x07)                  ' 6
Const SCANKEY_7 = 8(0x08)                  ' 7
Const SCANKEY_8 = 9(0x09)                  ' 8
Const SCANKEY_9 = 10(0x0A)                 ' 9
Const SCANKEY_0 = 11(0x0B)                 ' 0
Const SCANKEY_MINUS = 12(0x0C)             ' -
Const SCANKEY_EQUAL = 13(0x0D)             ' =
Const SCANKEY_BS = 14(0x0E)                ' ←

Const SCANKEY_TAB = 15((0x0F)              'TAB
Const SCANKEY_Q = 16(0x10)                 ' Q
Const SCANKEY_W = 17(0x11)                 ' W
Const SCANKEY_E = 18(0x12)                 ' E
Const SCANKEY_R = 19(0x13)                 ' R
```

```
Const SCANKEY_T = 20(0x14)                  ' T
Const SCANKEY_Y = 21(0x15)                  ' Y
Const SCANKEY_U = 22(0x16)                  ' U
Const SCANKEY_I = 23(0x17)                  ' I
Const SCANKEY_O = 24(0x18)                  ' O
Const SCANKEY_P = 25(0x19)                  ' P
Const SCANKEY_SQUARE_OPEN = 26(0x1A)        '[
Const SCANKEY_SQUARE_CLOSE = 27(0x1B)       ']
Const SCANKEY_ENTER = 28(0x1C)              ' ENTER

Const SCANKEY_CTRL = 29(0x1D)               ' CTRL
Const SCANKEY_A = 30(0x1E)                  ' A
Const SCANKEY_S = 31(0x1F)                  ' S
Const SCANKEY_D = 32(0x20)                  ' D
Const SCANKEY_F = 33(0x21)                  ' F
Const SCANKEY_G = 34(0x22)                  ' G
Const SCANKEY_H = 35(0x23)                  ' H
Const SCANKEY_J = 36(0x24)                  ' J
Const SCANKEY_K = 37(0x25)                  ' K
Const SCANKEY_L = 38(0x26)                  ' L
Const SCANKEY_SEMICOLON = 39(0x27)          ' ;
Const SCANKEY_QUOTATION = 40(0x28)          ' '

Const SCANKEY_QUOTATION2 = 41(0x29)         ' `
Const SCANKEY_LSHIFT = 42(0x2A)             ' LEFT SHIFT
Const SCANKEY_WON = 43(0x2B)                ' \

Const SCANKEY_Z = 44(0x2C)                  ' Z
Const SCANKEY_X = 45(0x2D)                  ' X
Const SCANKEY_C = 46(0x2E)                  ' C
Const SCANKEY_V = 47(0x2F)                  ' V
Const SCANKEY_B = 48(0x30)                  ' B
```

```
Const SCANKEY_N = 49(0x31)                    ' N
Const SCANKEY_M = 50(0x32)                    ' M
Const SCANKEY_COMMA = 51(0x33)                ' ,
Const SCANKEY_PERIOD = 52(0x34)               ' .
Const SCANKEY_SLASH = 53(0x35)                ' /
Const SCANKEY_RSHIFT = 54(0x36)               ' RIGHT SHIFT

Const SCANKEY_PRTSC = 55(0x37)      ' PRINT SCREEN SYS RQ
Const SCANKEY_ALT = 56(0x38)                  ' ALT
Const SCANKEY_SPACE = 57(0x39)                ' SPACE
Const SCANKEY_CAPS = 58(0x3A)                 ' CAPS
Const SCANKEY_F1 = 59(0x3B)                   ' F1
Const SCANKEY_F2 = 60(0x3C)                   ' F2
Const SCANKEY_F3 = 61(0x3D)                   ' F3
Const SCANKEY_F4 = 62(0x3E)                   ' F4
Const SCANKEY_F5 = 63(0x3F)                   ' F5
Const SCANKEY_F6 = 64(0x40)                   ' F6
Const SCANKEY_F7 = 65(0x41)                   ' F7
Const SCANKEY_F8 = 66(0x42)                   ' F8
Const SCANKEY_F9 = 67(0x43)                   ' F9
Const SCANKEY_F10 = 68(0x44)                  ' F10
Const SCANKEY_NUM = 69(0x45)                  'NUM ROCK
Const SCANKEY_SCROLL = 70(0x46)               ' SCROLL ROCK

Const SCANKEY_GRAY_HOME = 71(0x47)            ' HOME
Const SCANKEY_GRAY_UP = 72(0x48)              ' UP
Const SCANKEY_GRAY_PGUP = 73(0x49)            ' PAGE UP
Const SCANKEY_GRAY_MINUS = 74(0x4A)           ' -
Const SCANKEY_GRAY_LEFT = 75(0x4B)            ' LEFT
Const SCANKEY_GRAY_CENTER = 76(0x4C)          ' CENTER
Const SCANKEY_GRAY_RIGHT = 77(0x4D)           ' RIGHT
Const SCANKEY_GRAY_PLUS = 78(0x4E)            ' +
```

```
Const SCANKEY_GRAY_END = 79(0x4F)          ' END
Const SCANKEY_GRAY_DOWN = 80(0x50)         ' DOWN
Const SCANKEY_GRAY_PGDN = 81(0x51)         ' PAGE DOWN
Const SCANKEY_GRAY_INS = 82(0x52)          ' INS
Const SCANKEY_GRAY_DEL = 83(0x53)          ' DEL

Const SCANKEY_F11 = 87(0x54)               ' F11
Const SCANKEY_F12 = 88(0x55)               ' F12
```