C++프로그래밍및실습

테트리스 만들기

최종 보고서

제출일자: 2024.12.22
제출자명: 주소연
제출자학번: 223310

1. 프로젝트 목표

1) 배경 및 필요성

테트리스는 1984년 출시 이후 전 세계적으로 사랑받아 온 클래식 퍼즐 게임으로, 간단하고 직관적인 규칙과 높은 접근성 덕분에 폭넓은 인기를 얻어 왔습니다. 하지만 테트리스는 겉보기와 달리 심도 있는 게임성을 제공하며, 플레이어는 블록을 빠르게 회전시키고 적절한 위치에 맞추어 쌓아야 하는 도전 과제에 몰입하게 됩니다.

프로그래밍 학습의 관점에서, 콘솔 기반의 테트리스 개발은 자료구조와 알고리즘, 사용자 인터페이스 같은 다양한 프로그래밍 개념을 실제로 적용해볼 수 있는 이상적인 프로젝트입니다. 게임의 블록 생성과 회전, 충돌 감지, 라인 제거 같은 기능은 다양한 자료구조와 알고리즘을 적용해볼 수 있는 기회를 제공합니다. 또한, 게임의 흐름과 상태를 관리하기 위해 인터페이스와 게임 루프 설계가 필요하며, 이 과정에서 유용한 프로그래밍 패턴과 설계 기법을 학습할 수 있습니다.

2) 프로젝트 목표

이 프로젝트는 콘솔 기반의 테트리스 게임을 C++로 구현하여, 블록의 이동과 회전, 라인 제거 등 게임의 핵심 메커니즘을 학습하고, 이를 통해 자료구조, 알고리즘, 객체지향 설계의 이해를 높이는 것을 목표로 합니다.

3) 차별점

단순하고 직관적인 구조를 통해 테트리스의 핵심 기능을 구현하고, 코드의 가독성과 효율성을 높이는 데 초점을 둡니다. 이를 통해 유지보수와 확장성을 고려한 깨끗한 코드 작성을 목표로 합니다.

2. 기능 계획

1) 기능 1: 블록이 상단에서 하단으로 내려오는 기능

(1) 세부 기능 1 : 일정 시간 간격으로 블록이 한 칸씩 내려오도록 설정

– 설명: 블록의 초기 위치를 설정해서 구현한다.

2) 기능 2: 총 7가지 모양의 도형

- 설명: 새로운 블록이 생성될 때 7가지 중 하나를 무작위로 선택하여 생성한다.

3) 기능 3: 특정 키를 눌렀을 때 도형 회전

- 설명: 특정 키를 눌렀을 때 도형이 회전한다. (4가지 방향으로 회전)

4) 기능 4: 블록이 바닥 또는 다른 블록과 닿으면 다음 도형으로 넘어감

- 설명: 블록이 바닥에 닿았는지 또는 다른 블록과 닿았는지 검사하고 블록이 닿은 경우 현재 위치에 고정하고 새로운 블록 생성한다. 새로운 블록을 게임 상단에서 생성하고 다시 하강 시작한다.

5) 기능 5: 도형을 맞추어 일자가 되면 제거하고 다른 블록을 아래로 이동

- 설명: 블록이 고정될 때마다 가로 라인이 꽉 찼는지 검사하고 가득 찬 라인이 있으면 해당 라인을 제거한다. 제거된 라인 위에 있는 블록들을 한 줄씩 아래로 이동한다.

3. 기능 구현

(1) 블록이 상단에서 하단으로 내려오는 기능

- 입출력: 입력: Map[MAPHEIGHT][MAPWIDTH]: 게임 맵의 2차원 배열, blockShape[5][5]: 현재 움직이는 블록의 모양을 저장한 5x5 배열, *BlockPos: 블록의 현재 위치(X,Y 좌표)를 담은 구조체 포인터 출력: bool 타입: true(바닥에 닿음) 또는 false(아직 내려갈 수 있음)

- 설명: 블록이 아래로 한 칸 더 내려갈 수 있는지 체크하고 이동시키는 함수이다 각 열마다 가장 아래에 있는 블록의 위치를 BottomArray에 저장합니다. 블록이 바닥이나 다른 블록에 닿았는지 검사하고 닿지 않았다면 블록을 한 칸 아래로 이동합니다.

- 적용된 배운 내용: 4주차 조건문, 반복문, 7주차 함수, 11주차 포인터

- 코드 스크린샷

```
bool GoDown(char Map[MAPHEIGHT][MAPWIDTH], int blockShape[5][5], Position *BlockPos)
{
    int BottomArray[5] = {}; // 각 열마다 제일 아래에 있는 행을 저장하는 배열
    int BottomRow = 0;       // 제일 아래의 행

    Block.LimitBottom(blockShape, BottomArray, &BottomRow); // botRow에 블럭의 제일 아래 행이 저장, Array는 각 열에 마지막 행을 저장
    // Left, Right와 다르게 Array가 각 행이 아닌, 각 열을 기준으로 저장됐다.
    // 블록 제일 아랫부분의 y좌표가 blocky에 저장, 각 x축에 가장 아랫부분이 bottomArray에 저장
    for (int i = 0; i <= 4; i++)
    {
        if (BottomArray[i] != 0)
        {
            if (Map[(BlockPos->Y) + BottomArray[i] + 1][(BlockPos->X) + i] == '1' || (BlockPos->Y) + BottomRow + 1 == MAPHEIGHT)
            { // 각 열의 제일 아래 블록 한 칸 아래에 블록이 있거나 한칸 아래가 벽일 경우, true를 반환하여 바닥에 닿았다는 것을 알려준다.
                return true;
            }
        }
    }
    RemoveShape(Map, blockShape, BlockPos);
    Sleep(80);
    (BlockPos->Y)++;
    return false; // 닿지 않았으면 블록을 한 칸 아래로 내린다.
}
```

(2) 총 7가지 모양의 도형

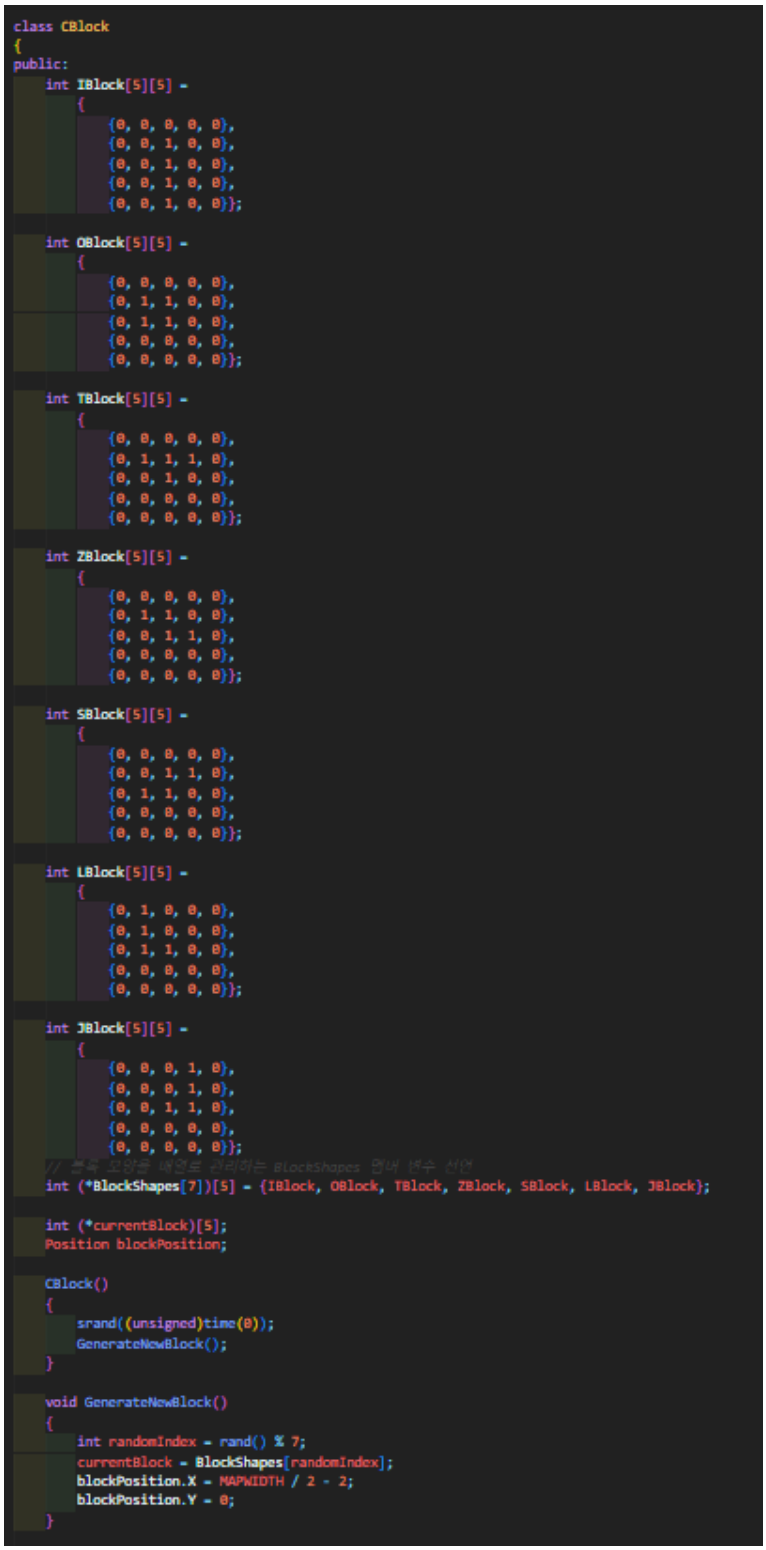
- 입출력: 입력: #define MAPWIDTH 15

출력: 7가지 도형 중 하나가 랜덤하게 생성되어 초기위치에 설정됩니다.

- 설명: 7가지 테트리스 블록의 모양을 2차원 배열로 정의하고 이들을 포인터 배열로 관리하며 새 블록이 필요할 때마다 랜덤하게 하나를 선택하여 맵 상단 중앙에 생성하는 기능을 구현합니다.

- 적용된 배운 내용: 5주차 배열, 9주차 클래스, 11주차 포인터

- 코드 스크린샷



```
class CBlock
{
public:
    int IBlock[5][5] =
        {
            {0, 0, 0, 0, 0},
            {0, 0, 1, 0, 0},
            {0, 0, 1, 0, 0},
            {0, 0, 1, 0, 0},
            {0, 0, 1, 0, 0}};

    int OBlock[5][5] =
        {
            {0, 0, 0, 0, 0},
            {0, 1, 1, 0, 0},
            {0, 1, 1, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0}};

    int TBlock[5][5] =
        {
            {0, 0, 0, 0, 0},
            {0, 1, 1, 1, 0},
            {0, 0, 1, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0}};

    int ZBlock[5][5] =
        {
            {0, 0, 0, 0, 0},
            {0, 1, 1, 0, 0},
            {0, 0, 1, 1, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0}};

    int SBlock[5][5] =
        {
            {0, 0, 0, 0, 0},
            {0, 0, 1, 1, 0},
            {0, 1, 1, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0}};

    int LBlock[5][5] =
        {
            {0, 1, 0, 0, 0},
            {0, 1, 0, 0, 0},
            {0, 1, 1, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0}};

    int JBlock[5][5] =
        {
            {0, 0, 0, 1, 0},
            {0, 0, 0, 1, 0},
            {0, 0, 1, 1, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0}};
    // 블록 모양을 배열로 관리하는 BlockShapes 멤버 변수 선언
    int (*BlockShapes[7])[5] = {IBlock, OBlock, TBlock, ZBlock, SBlock, LBlock, JBlock};

    int (*currentBlock)[5];
    Position blockPosition;

    CBlock()
    {
        srand((unsigned)time(0));
        GenerateNewBlock();
    }

    void GenerateNewBlock()
    {
        int randomIndex = rand() % 7;
        currentBlock = BlockShapes[randomIndex];
        blockPosition.X = MAPWIDTH / 2 - 2;
        blockPosition.Y = 0;
    }
```

(3) 특정 키를 눌렀을 때 도형 회전 및 이동

- 입출력: 입력: GetAsyncKeyState() 함수를 통한 키보드 입력 (VK_SPACE (스페이스바), 'A' (A키), 'D' (D키)), Map: 게임 맵의 2차원 배열, BlockShape: 현재 블록의 모양, BlockPos: 블록의 현재 위치 정보 출력: 블록의 회전 또는 이동된 상태

- 설명: 회전이 가능한 블록인지 먼저 확인하고 스페이스바를 누르면 블록이 회전합니다. A키를 누르면 블록이 왼쪽으로 이동합니다. D키를 누르면 블록이 오른쪽으로 이동합니다. 모든 동작은 현재 위치의 블록을 먼저 지우고(RemoveShape) 새로운 위치나 모양으로 블록을 그리는 순서로 진행됩니다.

- 적용된 배운 내용: 4주차 조건문, 9주차 클래스, 11주차 포인터

- 코드 스크린샷

```
if (GetAsyncKeyState(VK_SPACE) & 0x8000)
{ // Press 'Space' : 블록 회전
    if (noRotate == false)
    {
        map.RemoveShape(Map, BlockShape, &BlockPos);
        map.Rotate(Map, BlockShape, &BlockPos);
    }
}
if (GetAsyncKeyState('A') & 0x8000)
{ // Press 'A' : 블록 좌측 이동
    map.RemoveShape(Map, BlockShape, &BlockPos);
    map.GoLeft(Map, BlockShape, &BlockPos);
}
if (GetAsyncKeyState('D') & 0x8000)
{ // Press 'D' : 블록 우측 이동
    map.RemoveShape(Map, BlockShape, &BlockPos);
    map.GoRight(Map, BlockShape, &BlockPos);
}
```

(4) 블록이 바닥 또는 다른 블록과 닿으면 다음 도형으로 넘어감

- 입출력: 입력: Map: 게임 맵의 상태를 저장하는 2차원 배열, BlockShape: 현재 블록의 모양, NextBlock: 다음 블록의 모양, BlockPos: 블록의 현재 위치 정보, Bottom: 블록이 바닥이나 다른 블록과 닿았는지 여부 출력: 새로운 블록의 생성과 이동, 갱신된 맵

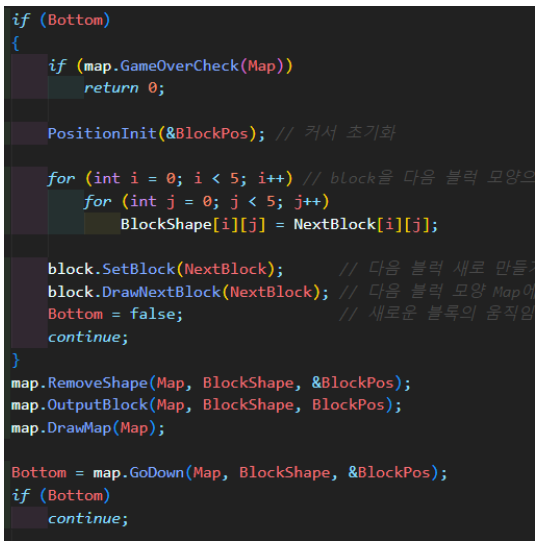
- 설명: 블록이 바닥이나 다른 블록과 닿았을 경우에는 먼저 게임오버 조건을 확인합니다. 게임이 계속될 수 있다면 새로운 블록을 생성하는 과정이 시작됩니다. 새 블록의 시작 위치를 맵 상단에 설정하고 미리 보여졌던 다음 블록을 현재 블록으로 가져옵니다. 그리고 또 다음에 나올 새로운 블록을 생성하여 화면에 표시합니다. 마지막으로 Bottom 상태를 false로 재설정하여 새 블록이 움직일 수 있도록 합니다.

두번째 블록이 아직 이동 중인 경우에는 블록의 현재 위치에서 모양을 지웁니다. 그 다

음 블록을 새로운 위치에 그리고 전체 맵을 다시 그려 화면을 갱신합니다. 이어서 블록을 한 칸 아래로 이동시키고 이 움직임으로 인해 바닥이나 다른 블록과 닿게 되었는지 확인합니다.

- 적용된 배운 내용: 4주차 조건문, 반복문, 5주차 배열, 11주차 포인터

- 코드 스크린샷



```
if (Bottom)
{
    if (map.GameOverCheck(Map))
        return 0;

    PositionInit(&BlockPos); // 커서 초기화

    for (int i = 0; i < 5; i++) // block을 다음 블럭 모양으
        for (int j = 0; j < 5; j++)
            BlockShape[i][j] = NextBlock[i][j];

    block.SetBlock(NextBlock);       // 다음 블럭 새로 만들
    block.DrawNextBlock(NextBlock); // 다음 블럭 모양 Map에
    Bottom = false;                 // 새로운 블록의 움직임
    continue;
}
map.RemoveShape(Map, BlockShape, &BlockPos);
map.OutputBlock(Map, BlockShape, BlockPos);
map.DrawMap(Map);

Bottom = map.GoDown(Map, BlockShape, &BlockPos);
if (Bottom)
    continue;
```

(5) 도형을 맞추어 일자가 되면 제거하고 다른 블록을 아래로 이동

- 입출력: 입력: Map[MAPHEIGHT][MAPWIDTH]: 게임 맵의 현재 상태, BlockPos: 현재 블록의 위치 정보, score: 점수를 저장하는 포인터 변수 출력: 완성된 줄 제거, 위의 블록들을 아래로 이동, 점수 업데이트

- 설명: 맵의 각 줄을 검사하여 블록으로 가득 찬 줄이 있는지 확인합니다. 가득 찬 줄을 발견하면 해당 줄을 제거(0으로 초기화)합니다. 제거된 줄 위의 모든 블록을 한 칸씩 아래로 이동합니다 완성된 줄 수에 따라 점수 부여 (1줄:500점, 2줄:2500점, 3줄:5000점, 4줄:10000점)합니다.

- 적용된 배운 내용: 4주차 조건문, 반복문, 5주차 배열, 11주차 포인터

- 코드 스크린샷

```
void CheckLine(char Map[MAPHEIGHT][MAPWIDTH], Position BlockPos, int *score)
    for (int i = MAPHEIGHT; i >= (BlockPos.Y); i--)
    {
        for (int j = 0; j < MAPWIDTH; j++)
        {
            if (Map[i][j] == '0')
            {
                break;
            }
            else if (Map[i][j] == '1')
            {
                height = i;
                count++;
            }

            if (count == MAPWIDTH)
            {
                linecount++;
                for (int j = 0; j <= MAPWIDTH; j++)
                {
                    Map[height][j] = '0';
                }

                while (height > 1)
                {
                    for (int j = 0; j <= MAPWIDTH; j++)
                    {
                        Map[height][j] = Map[height - 1][j];
                    }
                    height--;
                }
                i++; // 한 줄씩 다 내렸으므로 다시 그 줄부터 체크
            }
        }
        count = 0;
    }

    if (linecount == 1)
        (*score) += 500;
    else if (linecount == 2)
        (*score) += 2500;
    else if (linecount == 3)
        (*score) += 5000;
    else if (linecount == 4)
        (*score) += 10000;
}
```

프로그램 실행방법

프로그램 실행 시 '1' 입력: 게임 시작, '2' 입력: 프로그램 종료를 선택하여 실행합니다.

A키는 왼쪽으로 이동 D키는 오른쪽으로 이동 스페이스바는 회전입니다.

4. 테스트 결과

(1) 블록이 상단에서 하단으로 내려오는 기능

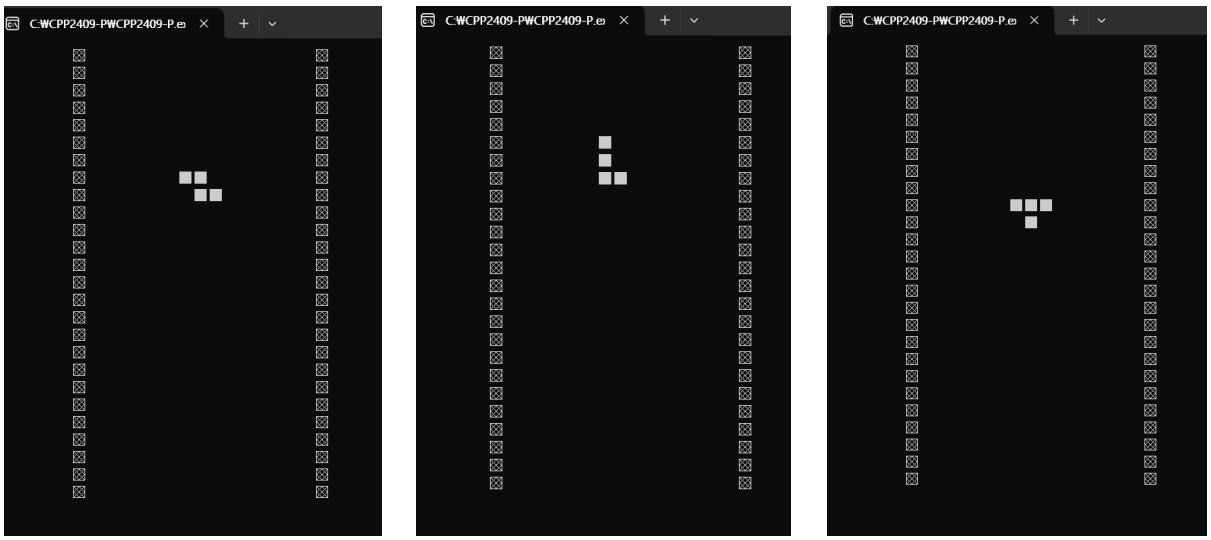
- 설명: 맵의 상단 중앙에서 새 블록이 생성되어 80ms 간격으로 한 칸씩 아래로 이동한다.

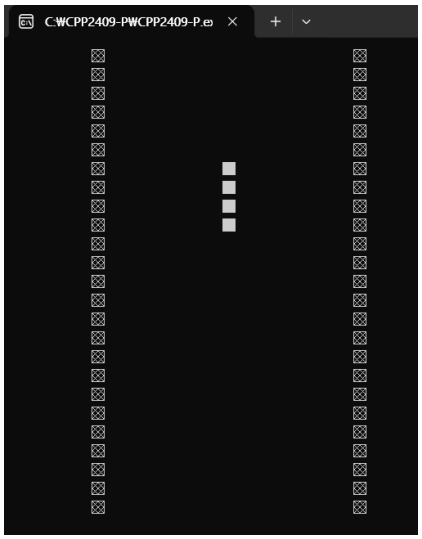
- 테스트 결과 스크린샷

(2) 총 7가지 모양의 도형

- 설명: 모든 블록 모양이 정상적으로 구현되었는지 테스트합니다.

- 테스트 결과 스크린샷

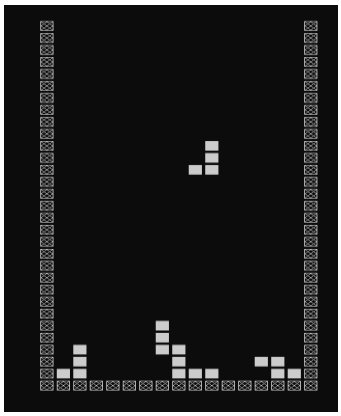




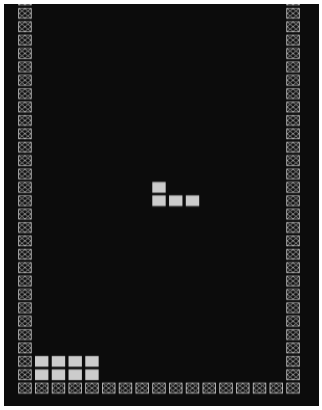
(3) 특정 키를 눌렀을 때 도형 회전 및 이동

- 설명: A키, D키, 스페이스바를 눌렀을 때 정상적으로 회전 및 이동하는 지 테스트합니다.

- 테스트 결과 스크린샷



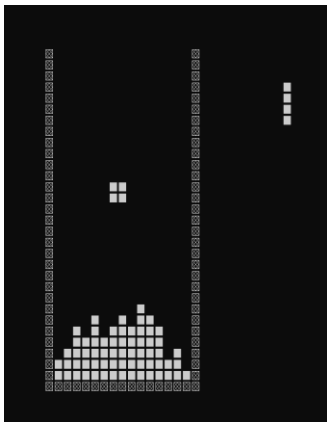
A키와 D키를 이용해 왼쪽과 오른쪽에 블록 이동

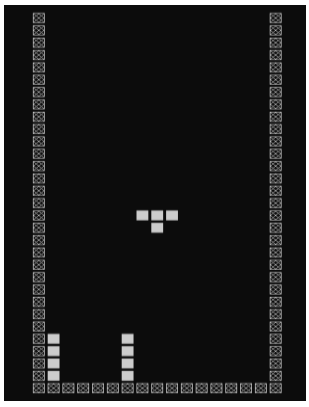
 스페이스바를 이용해 도형 회전

(4) 블록이 바닥 또는 다른 블록과 닿으면 다음 도형으로 넘어감

- 설명: 블록이 바닥 또는 다른 블록과 닿으면 다음 도형이 정상적으로 내려오는지 테스트합니다.

- 테스트 결과 스크린샷

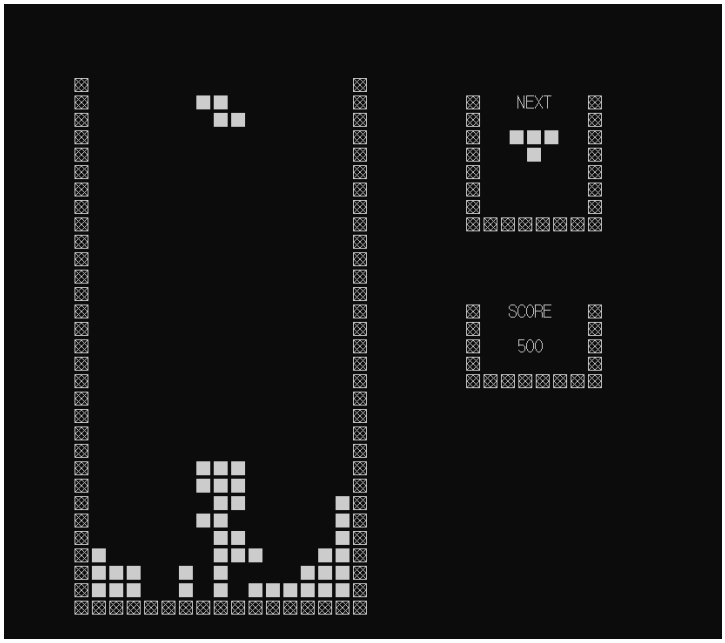
 다른블록과 닿았을 때

 블록이 바닥에 닿았을 때

(5) 도형을 맞추어 일자가 되면 제거하고 다른 블록을 아래로 이동

- 설명: 블록들이 가로로 한 줄을 가득 채우면 해당 줄이 제거되고 그 위의 블록들이 아래로 내려오는지 테스트합니다.

- 테스트 스크린샷



가로로 한 줄 가득 채우면 그 줄이 제거되고 점수가 +500올라감 그리고 위에 있던 블록들은 한칸 내려옴

블록들이 위에 까지 다 차면 점수를 알려주고 끝냄

5. 계획 대비 변경 사항 및 달성 내용

1) 블록이 상단에서 하단으로 내려오는 기능

- 이전: 단순히 블록의 Y좌표만 증가시키는 기본적인 하강 로직

- 이후: BottomArray를 사용하여 블록의 각 열마다 가장 아래 부분을 추적, 충돌시 블록을 고정시키는 기능 추가, RemoveShape를 통한 이전 위치 블록 제거 기능 추가

- 사유: 블록 충돌 처리의 정확성을 향상시키기 위해 변경

2) 서브맵을 만들어 점수 기능을 추가

- 내용: 메인 게임 화면 옆에 추가 정보를 표시하는 서브맵 구현, 다음 블록 미리보기 기능과 점수 표시 기능 추가, 라인 완성에 따른 차등 점수 시스템 도입 (1줄:500점, 2줄:2500점, 3줄:5000점, 4줄:10000점), 게임 오버 시 최종 점수 표시 기능 구현

- 추가 사유: 여러 줄을 한 번에 제거하는 것에 더 높은 점수를 부여하여 게임의 전략성 강화, 플레이어에게 다음 블록 정보를 제공하여 전략적 플레이 유도

6. 느낀점

테트리스 프로젝트를 진행하면서 많은 것을 배울 수 있었습니다.

가장 먼저 느낀 점은 **프로그래밍 개념의 실제 적용**입니다. 수업에서 배운 조건문, 반복문, 배열, 구조체, 클래스와 같은 개념들이 실제 게임 개발 과정에서 어떻게 유기적으로 연결되어 동작하는지를 체감할 수 있었습니다. 특히 객체 지향 프로그래밍을 활용하여 코드의 재사용성과 유지보수성을 높이는 작업에서 그 장점을 직접 경험했습니다.

블록 회전, 충돌 감지, 라인 제거와 같은 기능을 구현하면서 예상치 못한 기술적 문제들에 직면했지만, 하나씩 해결해 나가며 논리적 사고력과 디버깅 능력을 키울 수 있었습니다. 버그를 분석하고 해결하는 과정은 도전적이었지만 동시에 가장 보람 있는 부분이기도 했습니다.

또한, **프로젝트 관리의 중요성**을 절실히 느꼈습니다. 처음에는 계획한 대로 모든 것이 진행될 거라 생각했지만, 실제 구현 과정에서는 많은 변수가 생겼습니다. 이를 통해 유연한 사고와 관리 능력이 얼마나 중요한지 알게 되었고, 특히 코드의 구조화를 통해 유지보수

와 확장성을 고려하는 자세를 배울 수 있었습니다.

마지막으로, 단순히 작동하는 게임을 만드는 것에서 나아가 **사용자 경험**을 고려하는 것이 얼마나 중요한지도 깨달았습니다. 사용자가 게임을 더 재미있게 즐길 수 있도록 점수 시스템과 같은 추가 기능의 필요성을 고민하게 되었고, 게임을 만드는 입장에서 사용자 관점을 이해하려는 노력이 필요하다는 것을 알게 되었습니다.

이번 프로젝트는 프로그래밍을 이론이 아닌 실제로 체험할 수 있는 값진 기회였습니다. 게임 개발 과정 전반에 대한 이해도와 자신감을 얻을 수 있었고, 앞으로의 프로젝트에서도 이러한 경험들이 큰 밑거름이 될 것이라 생각합니다.