1. Project Description
   1. Program Flow Chart



* 1. Detailed Description

設計思路：

一開始在寫project的時候發現，方塊的形狀都是固定的，於是想要用二為矩陣去儲存方塊的相關資料，例如：

bool T1[2][3] = {1, 1, 1, 0, 1, 0};

但是當我要寫入function的時候，會有很大的麻煩，因為我們無法直接將一個固定的二維矩陣的pointer to pointer直接傳入至一個function裡（除非一開始是用new去建立二維矩陣），我以以下的code為例子：

#include <iostream>

using namespace std;

void print(int \*matrix) {

    for (int i = 0; i < 3; ++i) {

        for (int j = 0; j < 3; ++j) {

            cout << matrix[3\*i+j] << ' ';

        }

        cout << endl;

    }

}

int main() {

    int matrix[3][3];

    int value = 1;

    for (int i = 0; i < 3; ++i) {

        for (int j = 0; j < 3; ++j) {

            matrix[i][j] = value++;

        }

    }

    print(\*matrix);

    return 0;

}

從上面的example中會發現在傳入print的是matrix的row的address，因此在function裡matrix變為一維矩陣來使用，那還不如一開始全部都用一維矩陣來執行，反而更有效率且方便。於是我決定使用一維矩陣並以pointer來進行演算。

演算過程：

首先先定義變數以及方塊形狀（方塊的布林值取的順序是由上到下由左到右）：

    bool T1[6] = {1, 1, 1, 0, 1, 0};

    bool T2[6] = {0, 1, 1, 1, 0, 1};

    bool T3[6] = {0, 1, 0, 1, 1, 1};

    bool T4[6] = {1, 0, 1, 1, 1, 0};

    bool L1[6] = {1, 0, 1, 0, 1, 1};

    bool L2[6] = {1, 1, 1, 1, 0, 0};

    bool L3[6] = {1, 1, 0, 1, 0, 1};

    bool L4[6] = {0, 0, 1, 1, 1, 1};

    bool J1[6] = {0, 1, 0, 1, 1, 1};

    bool J2[6] = {1, 0, 0, 1, 1, 1};

    bool J3[6] = {1, 1, 1, 0, 1, 0};

    bool J4[6] = {1, 1, 1, 0, 0, 1};

    bool S1[6] = {0, 1, 1, 1, 1, 0};

    bool S2[6] = {1, 0, 1, 1, 0, 1};

    bool Z1[6] = {1, 1, 0, 0, 1, 1};

    bool Z2[6] = {0, 1, 1, 1, 1, 0};

    bool I1[4] = {1, 1, 1, 1};

    bool I2[4] = {1, 1, 1, 1};

    bool O[4] = {1, 1, 1, 1};

    bool is\_game\_over = 0;

    int rows, cols;

    string s;

然後開啟檔案tetris.data，並決定玩家想要的長度和寬度：

ifstream fin("tetris.data");

fin >> rows >> cols;

接著判斷玩家所輸入的內容是否合法，接著增加row的數量，主要是給予方塊掉下來的空間：

    if (rows > 40 || cols > 15) {

        throw "Error! The size of row or col is too large!";

    }

    if (rows < 1 || cols < 1) {

        throw "Error! The size of row or col is too small!";

    }

rows += 4;

然後建立一個一維matrix並且initialize。

    bool \*matrix = new bool[rows \* cols];

    for (int i = 0; i < rows \* cols; ++i) {

        matrix[i] = 0;

    }

接下來我們就可以開始輸入我們想丟入的方塊了。我們的input主要有兩個，s和col，其中s代表我們所輸入的字串，col代表我們想讓方塊在哪一列掉落。由於switch無法接受字串的判斷，因此這裡全部由if…else來控制。我們所輸入的值經過判斷後，會輸入至一個叫做block的function，其中function所輸入的變數有7個：

void block(bool \*matrix, int matrix\_rows, int matrix\_cols,

bool \*block, int block\_rows, int block\_cols, int col);

在block中，先定義幾個重要變數，並且判斷玩家輸入的內容是否合法（方塊的位置是否超出範圍），其中p代表一開始方塊的左上角的address，row代表一開始方塊的最底部在第幾行：

    bool \*p = matrix + col - 1;

    int row = block\_rows - 1;

    if (col + block\_cols - 1 > matrix\_cols) {

        throw "Error input!";

    }

接著就是判斷方塊掉落的行數，由於row不能超出matrix的範圍，因此該迴圈由while (row < matrix\_rows)控制。在迴圈中定義一些非常重要的變數：q代表同一列中p下一行的address，r代表該方塊的address。第一個雙重for loop是用來check該方塊若再往下一格，是否與其他方塊重疊，若有重疊或是已經到底，則進入第二個雙重for loop，注意這裡的q等於p，r變回原本的block。

第二個雙重for loop是用來記錄該方塊最後掉落的地方，\*q = (\*r) ? 1 : \*q;這條式子就是將block的值紀錄再matrix上，要注意不能直接讓q = p，否則會讓本來已經為１的值變為０。記錄完後，必須跳出迴圈while，直接break。

    while (row < matrix\_rows) {

        bool \*q = p + matrix\_cols;

        bool \*r = block;

        bool check = 0;

        for (int i = 0; i < block\_rows && row < matrix\_rows; ++i) {

            for (int j = 0; j < block\_cols; ++j) {

                if (\*q && \*r) {

                    check = 1;

                }

                ++r; ++q;

            }

            q = q + matrix\_cols - block\_cols;

        }

        q = p; r = block;

        if (check || row == matrix\_rows - 1) {

            for (int i = 0; i < block\_rows; ++i) {

                for (int j = 0; j < block\_cols; ++j) {

                    \*q = (\*r) ? 1 : \*q;

                    ++r; ++q;

                }

                q = q + matrix\_cols - block\_cols;

            }

            break;

        }

        p += matrix\_cols; ++row;

    }

接著，既然位置已經記錄好了，那就要判斷該行能否進行消除，輸入至eliminate的變數只要有三個就好，因為不必再對matrix進行增加方塊的動作：

eliminate(matrix, matrix\_rows, matrix\_cols);

首先先再函數裡定義幾個重要的變數：p為pointer，作為預備用，is\_full預設為１，為了讓後面能夠進入while檢查，eliminate\_row紀錄要被削除的行數，其預設值為０。

進入while迴圈時，第一個for迴圈不會被執行，因為eliminate\_row的預設值為０。然後讓p為matrix，讓ｐ能夠從頭到尾檢查哪一行全部都為１，若其中該行有一個element為０，則is\_full就會一直為零，直到進入下一行為止。若檢查到有一行全部為１時，eliminate\_row就會記錄第幾行，然後再執行一次while loop，這時候第一個for loop就會執行，\*p = \*(p - matrix\_cols);這條式子主要是位移被削除的那行以上所有的element往下移動一行。移動完後，就會再次檢查，直到沒有能夠被削除的element。

    bool \*p;

    bool is\_full = 1;

    int eliminate\_row = 0;

    while (is\_full) {

        for (int i = eliminate\_row; i > 0; --i) {

            p = matrix + matrix\_cols \* i;

            for (int j = 0; j < matrix\_cols; ++j) {

                \*p = \*(p - matrix\_cols);

                ++p;

            }

        }

        p = matrix;

        for (int i = 0; i < matrix\_rows; ++i) {

            is\_full = 1;

            for (int j = 0; j < matrix\_cols; ++j) {

                is\_full \*= \*p;

                ++p;

            }

            if (is\_full) {

                eliminate\_row = i;

                break;

            }

        }

    }

回到原本的main，的while loop裡，後面有一個判斷式為：

        for (int i = 0; i < cols \* 4; ++i)

            if (matrix[i])

                is\_game\_over = 1;

其主要在於檢查matrix上面四行是否有布林值為１的element，若有，則結束遊戲。

最後關閉tetris.data，並且建立tetris.output，然後印出最後的結果，再關閉tetris.output：

    fin.close();

    ofstream fout("tetris.output");

    for (int i = 4; i < rows; ++i) {

        for (int j = 0; j < cols; ++j) {

            fout << matrix[i \* cols + j];

        }

        fout << endl;

    }

    fout.close();

到這裡該project已結束。

1. Test case Design
   1. Detailed Description of the Test Case

Test case:

|  |  |  |  |
| --- | --- | --- | --- |
| 12 9  T3 1  T2 3  O 5  J2 7  O 8  I1 1  O 2  T2 4  O 6  L4 2  J4 6  O 1  I2 6  Z2 5  Z2 3  S2 4  O 8  I1 7  O 1  Z1 1  T2 5 | O 8  O 8  T2 5  T2 5  I1 7  Z1 1  Z1 1  I1 4  O 8  Z1 2  O 7  L2 1  L3 8  Z2 4  T1 6  O 1  L2 4  J4 7  I1 3  L3 4  T4 1  I2 6 | T1 1  O 4  S2 5  L1 7  T2 8  O 1  L3 2  J3 4  T1 7  J3 6  O 8  Z2 1  L2 3  O 6  L3 4  T1 2  I1 1  S1 2  T2 6  I1 5  O 8  O 8 | L2 2  I2 1  T2 5  L1 7  L3 8  I2 2  L2 1  I2 4  J1 8  End |