积木游戏实验报告

1.作业目标

实现一个类似俄罗斯方块的程序。要求支持图形界面；支持玩家随机取出包括正方形、三角形、长方形、梯形在内的方块；能够在方块堆满一或几行时完全消去并更新游戏分数；支持玩家随时存档、读档、退出游戏。

2.需求分析

由于这学期接触了面向对象的思路，所以试图将作业中涉及的问题逐一拆解为各个对象的动作以及对象之间的关系与协作的问题。

首先遇到的是面板的概念。通过借助面板来确定积木所在的位置、积木位置是否会冲突，也通过面板来储存每一个方格的颜色。其次，各类积木也拥有相同的属性，如下移、旋转、中心位置等，因此把各个积木归为一类，从 CBrick 中各自派生。之后玩家的各种操作也需要一个类，来接纳来自 windows 操作系统的信息，也需要将实时的状态显示出来的 CDisplay 类。

3.数据结构

3.1 积木方块的存储

各个方块都由三角形构成，而一个正方形可由两个三角形覆盖，再加上表示两个三角形颜色的部分，正好可以用一个 unsigned short 来表示一个正方形的状态。新填充进一个正方形的三角形被放入低八位中，后填充进的三角形只需和两个字节的低四位分别取与运算，即可判别是否冲突。

每一个积木方块都拥有自己中心的 xy 坐标，同时也拥有表示方向的数据，这样在积木进行旋转时，要通过位运算来循环移动表示方向的数据的各个位。

3.2 面板的数据

面板包括 30*11 个方格，但是为了规避数组越界需要做的特殊处理，在左右下三个方向各增加了一行/列。方格用 unsigned short 的数组存储。另一方面，为了方便判别是否能消去方块，用一个单独的结构来表示某一行是否已经放满、是否与其他行之间连通。最后，将已经落下的方块的指针存放在 vector 中方便管理。

3.3 各种积木方块的关系

各种积木方块拥有相同的属性，此外在消去、显示、下移时并没有必要特地考虑积木的类型，所以将各种积木都设置为 CBrick 派生出的子类，将它们共有的属性提炼到基类中，利用多态的机制在各个子类中分别实现一些相同的方法。

4.数据结构的代码实现

4.1 头文件

```
#include <windows.h>
#include <windowsx.h>
#include <vector>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <algorithm>
```

前两个头文件用来与 windows 进行交互。因为要对存放在容器中的各个积木的指针排序，引入了 vector 和 algorithm 头文件。

4.2 宏定义

```
#define HRECTANGLE 0xAA        //horizonal rectangle: 1010 1010
#define VRECTANGLE 0x55        //vertical rectangle: 0101 0101
#define LUTRIANGLE 0x0C        //triangle on the left-up conner: 0000 1100
#define RUTRIANGLE 0x06        //triangle on the right-up conner: 0000 0110
#define RDTRIANGLE 0x03        //triangle on the right-down conner: 0000 0011
#define LDTRIANGLE 0x09        //triangle on the left-down conner: 0000 1001
#define HUTRAPEZIUM 0x39       //horizonal trapezium shrinking up: 0011 1001
#define VRTRAPEZIUM 0xC9       //vertical trapezium shrinking right: 1100 1001
#define HDTRAPEZIUM 0xC6       //horizonal trapezium shrinking down: 1100 0110
#define VLTRAPEZIUM 0x36       //vertical trapezium shrinking left: 0011 0110
```

定义了各种积木在各种方向下的代号，同时也便于通过位运算表示积木的结构，如水平向上的梯形的高低四位分别代表着左右的两个三角形。

4.3CDisplay 类的成员变量与成员函数

```
class CDisplay {
        HWND hWnd,
                hBnNew, //child-windown for new-button
                hBnSave, //child-windown for save-button
                hBnOpen, //child-windown for open-button
                hBnExit, //child-windown for exit-button
                hBnMvLeft, //child-windown for mvLeft-button
                hBnMvRight, //child-windown for mvRight-button
                hBnMvDown, //child-windown for mvDown-button
                hBnRtClck, //child-windown for rotate-clockwise-button
                hBnRtAntiClck;//child-windown for rotate-anti-clockwise-button

        HDC    hDC;
        COLORREF colorRef[COLOR_NUM+7];
        RECT    boardRect, scoreRect;
        int     winWidth, winHeight, boardLegend;
        bool fillTriangle(int color, int x, int y, int triangle) const;
public:
         CDisplay(HWND hWnd_);
         ~CDisplay();
         void refresh(int score);
         int    parseCommand(LPARAM hButton) const;
         bool show(int x, int y, int triangle) const;
         bool erase(int x, int y, int triangle) const;
         void show(int arg) const;
         void show(char arg[])const;
```

HWND 是一个窗口句柄，用来操作图形界面。HDC 用来绘制图形。Refresh()函数在消去某一行或几行后刷新界面，更新分数。Show()函数向面板显示三角形或向分数框中输出分数。

4.4CBoard 的成员变量与成员函数

```
class CBoard {
        CDisplay* pDisplay;
        unsigned short pixels[31][13];
        vector<CBrick*> vecPTRBricks;
        SRowState rowStates[31];
public:
        CBoard();
        ~CBoard();
        void repaint();
        bool init(CDisplay* pDisplay_);
        unsigned short   getRim(short x, short y);
        CBoard& operator<<(CTriangle triangle);
        CBoard& operator>>(CTriangle triangle);
        int tryclear(CBrick*);
        int clear();
};
```

CBoard 中有 CDisplay 的指针，通过它来控制面板的状态；存放的 rowStates 数组存储各个行是否全部填充、是否互相连通；存储着各个已经落下的积木的指针的容器方便消去落满几行的积木。

getRim()函数返回某一个方块的占用情况，可以用来判断别的积木是否能占用这块方块；重载的两个运算符用于显示或抹去某块积木。Tryclear()是用于一块积木落下之后消去已经落满的行的(如果有的话)，返回值是消去这些积木的得分。Clear()用来存读档、开始新游戏时清空面板。

4.5CBrick 的成员变量与成员函数

```
class CBrick {
        const static unsigned char brickType[SHAPE_NUM];
protected:
        static CBoard &board;           short   x,y;
        unsigned char color, direction;
        virtual void    erase()=0;      virtual void    show()=0;
        virtual bool isProped()=0;
public:
        short   score;
        CBrick(short x_, short y_, char color_, unsigned char direction_);
        virtual ~CBrick();
        static CBrick* newBrick();
        void stop();
        int   mvDown(int command=UNDEFINED);
```

```
        virtual int    mvLeft()=0;
        virtual int    mvRight()=0;
        virtual int    rotateLeft();
        virtual int    rotateRight();
friend class CDisplay;
friend bool compare_by_y(const CBrick*, const CBrick*);
friend int CBoard::tryclear(CBrick* pBrick);
};
```

brickType 用来定义各种积木的类型，在 main.cpp 中得到初始化，随机生成积木时只需生成一个随机数模去 SHAPE_SUM 就取到随机的积木类型；board 的引用方便得到某一个方块的占用情况，判断积木能否移动；各个积木的下移操作思路都是一致的(没有阻挡就可下移)，所以在 CBrick 中实现；而各个积木的左右移动、旋转留给各自去实现，在 CBrick 中定义为纯虚函数；每一块积木的分数计算方式为下落的格子数；添加了 compare_by_y()函数来对 CBoard 中的指针容器排序。

从 CBrick 类中派生出来的各个积木类都只是实现基类的虚函数，并没有增加其他的成员变量或者成员方法。

5.具体问题的代码实现

5.1 新积木的取出(以正方形为例)

```
CBrick* CBrick::newBrick(){
        CBrick* result=NULL;
        int color = rand()%COLOR_NUM + 1;
        unsigned short curBrickType = rand()%SHAPE_NUM;

        if ( board.getRim(6, 30)&LUTRIANGLE&RUTRIANGLE ) return result;
        switch ( brickType[curBrickType] ) {
        case SQUARE:
            if(board.getRim(6, 30)){
                return newBrick();
             }
             result = new CSquare(6, 30, color, brickType[curBrickType]);
             break;
        ......
        default:
                break;
        }
        if (result!=NULL) result->show();
        return result;
}
```

首先利用随机数来确定新的方块的形状与颜色；同时因为新的积木无论如何都要占用最上方一行的中间方块，如果这个方块全部被占用，判定游戏结束；如果没有全被占用，那么继续判断能否产生随机出来的方块类型，对

正方形来说，它要求坐标为(6,30)的方块全为空，不满足要求就递归来尝试一个新的类型的积木是否可行；如果产生了一个新的积木，那么在屏幕上显示出来，并返回它的指针来进行后续操作。

5.2 积木的显示与擦去(以长方形为例)

```
void CRectangle::show(){
	CTriangle* parray[6];
	switch(direction){
		case HRECTANGLE:
			for(int i = 0; i < 3; i += 1){
				parray[2 * i] = new CTriangle(x - 1 + i, y, color, LUTRIANGLE);
				parray[2 * i + 1] = new CTriangle(x - 1 + i, y, color, RDTRIANGLE);
			}
			for(int i = 0; i < 6; i += 1){
				board << *parray[i];
			}
			break;
		 ......
	}
	for(int i = 0; i < 6; i += 1){
		delete parray[i];
	}
}
```

通过产生临时的三角形对象来显示一个长方形，最后回收临时对象的内存。擦去积木时只需将’<<’换成’>>’。

5.3 积木的移动(以梯形的旋转为例)

```
int CTrapezium::rotateRight(){
	bool stopped = true;
	char lPixel = ' ';
	char rPixel = ' ';
	switch (direction){
		case HUTRAPEZIUM:
			if(board.getRim(x + 1, y - 1)){
				break;
			}
			if(board.getRim(x, y - 1)){
				break;
			}
			if(board.getRim(x - 1, y) & LUTRIANGLE){
				break;
			}
			if(board.getRim(x - 1, y + 1)){
				break;
			}
```

```
                stopped = false;//没有卡住
                break;
                ……
        }
        if(stopped){
              return mvDown();//卡住，尝试下移
        }
        erase();
        lPixel = direction >> 4;//取左四位
        rPixel = direction & 0xF;//取右四位
        rPixel = (rPixel >> 1)|((rPixel & 1) << 3);//前三位后移，第四位置首
        lPixel = (lPixel >> 1)|((lPixel & 1) << 3);
        direction = (rPixel << 4) | lPixel;//左右对调
        show();
         return RT_RIGHT;
```

对每种梯形，看旋转过程中需要占用的各个方格/三角形是否为空；如果卡住，就尝试下移(如果返回 stopped 会出现方块卡在边框上却被判定为不能移动的情况)；之后擦去自己，分别轮换表示方向的数据的前后四位，再显示出新的位置/方向。

5.4 三角形积木的消去与显示(运算符重载)

```
CBoard& CBoard::operator<<(CTriangle triangle) {
        int x=triangle.x;
        int y=triangle.y;
        unsigned char color = triangle.color;
        unsigned char direction = triangle.direction;

        if ( x<1 || x>11 || y<1 || y>30 ) return (*this);
        switch (((pixels[y][x]>>8)|pixels[y][x])&0xF) {
                case 0xF:
                        return (*this);
                case 0:
                        pixels[y][x] = (color<<4)|direction;
                        break;
                default:
                        if (direction&pixels[y][x]&0xF) return (*this);
                        pixels[y][x] = ((color<<4)|direction)|(pixels[y][x]<<8);
                        break;
        }
        if (pDisplay==NULL) return (*this);
        pDisplay->show(x, y, (color<<4)|direction);
        rowStates[y].blankTriangles--;
        return (*this);
}
```

需要考虑到几种情况，如果传入三角形在边界以外，不需处理，直接返回；如果这个方块已经占满(两个字节的后四位并为 0xF)，也直接返回；这个方块原来是空的，就把三角形存到低字节中；原来有一个三角形(如果不与现在的冲突)。就放到低字节中，原来的三角形放到高字节中；同时减少这一行中空白的三角形数量。

```
CBoard& CBoard::operator>>(CTriangle triangle) {
        int x=triangle.x;
        int y=triangle.y;
        unsigned char direction = triangle.direction;



        if ( x<1 || x>11 || y<1 || y>30 ) return (*this);
        if ( (direction&0xF)== (pixels[y][x]&0xF) ) {
                triangle.color = (pixels[y][x]&0xF0)>>4;
                pDisplay->erase(x, y, direction);
                pixels[y][x]=pixels[y][x]>>8;
        }
        if ( (direction&0xF)== ((pixels[y][x]>>8)&0xF) ) {
                triangle.color = (pixels[y][x]&0xF000)>>12;
                pDisplay->erase(x, y, direction);
                pixels[y][x]=pixels[y][x]&0xFF;
        }
        rowStates[y].blankTriangles++;
        return (*this);
}
```

抹去三角形是类似的，如果抹去的是高字节的三角形，调用 erase()函数后将高字节清空，是低字节的话，将高字节的三角形右移到低字节中。

5.5 消去占满的行

```
int CBoard::tryclear(CBrick* pBrick){
      short result = 0;
      vecPTRBricks.push_back(pBrick);
      sort(vecPTRBricks.begin(), vecPTRBricks.end(), compare_by_y);

      for(vector<CBrick*>::iterator i = vecPTRBricks.begin(); i != vecPTRBricks.end(); i ++){
           if((*i)->direction == VRECTANGLE || (*i)->direction == VRTRAPEZIUM ||
(*i)->direction == VLTRAPEZIUM){
                rowStates[(*i)->y + 1].linked |= 1;//01 ==> linked down
                rowStates[(*i)->y - 1].linked |= 2;//10 ==> linked up
                rowStates[(*i)->y].linked |= 3;       //10 ==> linked up and down
           }
      }
```

首先插入新落下来的积木的指针，并对 vector 排序，并更新行之间连接的状态，因为向上连通和向下连通是不同的，这里区分标记。

```
int upperbound, lowerbound;
for(int i = 1; i <= 30; i += 1){
    if(rowStates[i].blankTriangles == 0){
        upperbound = i;
        lowerbound = i;
        while((rowStates[upperbound].linked & 2) && (rowStates[upperbound +
1].linked & 1)){
            upperbound += 1;
        }
        while((rowStates[lowerbound].linked & 1) && (rowStates[lowerbound -
1].linked & 2)){
            lowerbound -= 1;
        }
        bool allfilled = true;
        for(int j = lowerbound; j != upperbound + 1; j += 1){
            if(rowStates[j].blankTriangles != 0){
                allfilled = false;
                break;
            }
        }
```

寻找所有被占满的行，并找到与它们相连的行的上下界(有可能上面的行向上连通，下面的行向下连通，区分标记这两种连通形式之后，不会出现扩大消去行范围的情况)。如果这些行之间全部被占满，就消去这些行之间的积木。

```
if(allfilled == true){
    for(vector<CBrick*>::iterator it = vecPTRBricks.begin(); it != vecPTRBricks.end(); ){
        if((*it)->y <= upperbound && (*it)->y >= lowerbound){
            result += (*it)->score;
            (*it)->erase();
            delete (*it);
            it = vecPTRBricks.erase(it);
        }else{
            it++;
            }
    }
```

删去在上下界之间的积木，并加上这些积木的得分，作为之后的返回值；从容器中删去积木后，迭代器的位置可能会出现问题，所以利用 erase()的返回值来保证迭代器指向下一个元素，或者手动自增迭代器

```
for(vector<CBrick*>::iterator it = vecPTRBricks.begin(); it != vecPTRBricks.end(); ){
    if((*it)->y > upperbound){
        (*it)->erase();
        (*it)->y -= upperbound - lowerbound + 1;
        (*it)->show();
        it++;
    }else{
        it++;
    }
```

之后将删去的行上面的积木向下移，并显示在屏幕上；在整个循环的结束，更新行之间连通的状态(没有必要在积木下移时就更新，因为每一次只落下一块积木，也就只能消去一整块连通的行，不会出现下面的行被消去了，上面的行也要消去，但是上面的积木已经被移下去的情况)。

5.6 处理玩家输入

```
void CPlayer::executeCommand() {
    short result;
    char    str_gameover[] = "Game Over";
    if (pBrick==NULL) return;
    switch ( command ) {
    case MV_LEFT:
        result = pBrick->mvLeft();
        break;
    ......
    default:
        result = pBrick->mvDown(command);
        break;
    }
    command = UNDEFINED;
    if ( result != STOPPED ) return;

    score += board.tryclear(pBrick);
    repaint();
    pBrick = CBrick::newBrick();
    if ( pBrick==NULL ) pDisplay->show(str_gameover);
    return;
}
```

如果执行指令之后，积木未被卡住，则返回；如果被卡住，则看是否能消去一些积木，并加上这些积木带来的分数，更新界面，取到一块新的积木；没有释放上一块积木的空间，因为此时这块积木由 vector 中的指针管理，提前释放掉会出现错误。

5.7 存储游戏&读入存档

因为是在游戏进行过程中存读档，所以要考虑到当前的积木是否会影响

游戏进程，所以在进行文件之前，先将现在的积木预先处理，并对面板做相应的处理，再进行文件操作。

5.7.1 存档

```
void CPlayer::saveGame() {
	if(pBrick){
		int result = pBrick->mvDown(command);
		score += board.tryclear(pBrick);
		repaint();
	}
```

先让当前的积木落下，并尝试消去落满的行，之后再打开文件。

```
	ofstream gameFile(gameFilePath, ios::out);
	gameFile << score << endl;
	for(int i = 0; i < 31; i += 1){
		for(int j = 0; j < 13; j += 1){
			gameFile << board.pixels[i][j] << ' ';
		}
		gameFile << endl;
	}
	for(int i = 0; i < 31; i += 1){
		gameFile << board.rowStates[i].blankTriangles << ' ' <<
board.rowStates[i].linked << endl;
	}
	gameFile << endl;
	vector<CBrick*>::iterator it = board.vecPTRBricks.begin();
	gameFile << board.vecPTRBricks.size() << endl;
	while(it != board.vecPTRBricks.end()){
		gameFile << (*it)->x << ' ' << (*it)->y << ' ' << (*it)->color << ' ' <<
(*it)->direction << ' ' << (*it)->score << endl;
		it++;
	}
	gameFile.close();
```

依次向文件中输出分数，各个像素点的信息，行的状态，已经落下的方块的信息，最后关闭文件。

```
	if(pBrick){
		delete pBrick;
	}
	pBrick = CBrick::newBrick();		command = UNDEFINED;
	if ( pBrick==NULL ) return;
	pDisplay->show(score);
	hTimer = SetTimer(hWnd, TIMER_ID, 500, NULL);
```

之后重新申请一块积木(因为之前的已经落下了)，显示新的游戏信息。

下面是以记事本形式看到的存档文件。

```
0//分数
780 780 780 780 780 780 780 780 780 780 780 780 780//行列信息
780 0 0 0 0 38 11299 44 0 0 0 0 780
780 0 0 0 0 23635 23635 23635 0 0 0 0 780
780 0 0 0 0 0 83 0 0 0 0 0 780
780 0 0 0 0 0 70 0 0 0 0 0 780
……
18 0//每一行的空白三角形数，连接情况
16 0
21 0
21 1
20 3
21 2
22 0
……
4//容器中积木数
6 1    ?29//每个积木的状态(方向与颜色是乱码)
6 2    ?28
6 3      27
6 5    6 24
```

5.7.2 读档

```
void CPlayer::openGame() {
	score = 0;
	if(pBrick){
		pBrick->erase();
		delete pBrick;
	}
	int wellclear = board.clear();
	if(!wellclear){
		return ;
	};
	gameFile >> score;
	for(int i = 0; i < 31; i += 1){
		for(int j = 0; j < 13; j += 1){
			gameFile >> board.pixels[i][j];
		}
	}
	for(int i = 0; i < 31; i += 1){
		gameFile >> board.rowStates[i].blankTriangles >>
board.rowStates[i].linked;
	}
```

与存档类似，先清除现有积木的指针，再清空面板，为读入存档做准备，之后按照存档的顺序，先读入分数，之后读入每一个像素块的信息，每行的空白方格数和连接情况。

```
int size = 0;
gameFile >> size;
vector<CBrick*>::iterator it;
while(size--){
    int x, y, score;
    unsigned char color, direction;
    gameFile >> x >> y >> color >> direction >> score;
    switch(direction){
        case SQUARE:
            board.vecPTRBricks.push_back(new CSquare(x, y, color, direction));
            it = board.vecPTRBricks.end() - 1;
            break;
        ......
    }

    (*it)->score = score;
    (*it)->show();
}
gameFile.close();
```

因为 CBrick 类是抽象类，不能定义对象，所以在读入容器中的积木的时候，要按照不同积木的类型分别生成不同类的对象，放入容器中，并显示在屏幕上；最后关闭文件。

```
pBrick = CBrick::newBrick();
command = UNDEFINED;
if ( pBrick==NULL ) return;
pDisplay->show(score);
hTimer = SetTimer(hWnd, TIMER_ID, 500, NULL);
 return;
```

最后重新申请新的积木，显示分数，继续游戏。

5.8 开始新游戏

```
void CPlayer::newGame() {
    score = 0;
    if(pBrick){
        pBrick->erase();
        delete pBrick;
    }
   for(vector<CBrick*>::iterator i = board.vecPTRBricks.begin(); i !=
board.vecPTRBricks.end(); i += 1){
        (*i)->erase();
```

```
        }// compile error if this code is put in board.clear()
        int wellclear = board.clear();
        if(!wellclear){
            return ;
        };
        pBrick = CBrick::newBrick();//newBrick();
        if ( pBrick==NULL ) return;
        command = UNDEFINED;
        if ( hTimer!=0 ) KillTimer(hWnd, TIMER_ID);
        hTimer = SetTimer(hWnd, TIMER_ID, MV_SPEED, NULL);
        pDisplay->show(score);
        return;
}
```

分数置零，释放当前指针，擦去所有在容器中的积木，清除面板；重新申请一块新的积木，开始新游戏。

5.9 清除面板现有状态(CBoard::clear 函数)

```
int CBoard::clear(){
        for( int i=0; i<13; i++ ) pixels[0][i] = 0x030C;
        for( int i=1; i<31; i++ ) pixels[i][0] = pixels[i][12] = 0x030C;
        for( int i=0; i<30; i++ )
            for( int j=0; j<11; j++) pixels[i+1][j+1]=0;
        for( int i=1; i<31; i++ ) {
            rowStates[i].linked = 0;
            rowStates[i].blankTriangles = 22;
        }
        vecPTRBricks.clear();
        pDisplay->show(rowStates);
        return 1;
}
```

这个函数在开始新游戏、读档时会被调用；它会清除面板上每一个像素的信息，将每一行的状态都初始化，并清空容器中的积木；返回 1 表示清除成功。

6.C++的应用

6.1 抽象与封装

在作业的一开始就用类和对象的概念来解析题目，从积木的相同属性抽象出 CBrick 类，从玩家的操作抽象出 CPlayer 类，从面板抽象出 CBoard 类，从显示图形界面的需求抽象出 CDisplay 类。借助于类的概念通过不同作用域的声明，不同功能被封装在各自的类中，例如每一个像素点和行的状态被设置为私有成员，不允许在类的外部任意修改；而积木的移动旋转操作是公有的，方便在其他的类中操作积木。

6.2 继承与多态

在积木游戏中，不同的积木共享相同的属性，可以考虑在它们之间建立

关系，因为不能强行认为某一种积木可以成为另一种的基类，所以将所有积木都认为是一个抽象类 CBrick 的子类，由基类声明积木共有的成员与方法，各个子类分别给出自己的实现。在游戏过程中，利用多态的机制，对基类的指针进行移动、旋转、显示、擦除的操作，而具体调用的函数则由指针指向的对象的类型决定。

6.3 运算符重载

在作业中涉及到对三角形的输入输出，是采用类似 iostream 一样用流插入、提取运算符进行的；在 CBoard.cpp 中，给出重载的运算符的代码实现。这样使得处理其他类型的积木的输入输出变得方便，编写代码变得简单了。

6.4 标准模板库 算法

在涉及消去积木时，需要存储已经落下的积木，因为积木的数量不定，所以不能用定长数组来存储；因为积木的类型不相同，再加上如果直接存入对象本身的话，实际上存入的是一个副本，并不会释放原有的积木的内存，反而会增加空间的占用，所以采用存入基类的指针的形式管理已经落下的积木；在需要对积木排序(按纵坐标)时，还需要调用 sort()模板，并编写自己的比较函数来排序。

6.5 文件操作

在涉及存档读档时，定义了文件流对象，使得向文件输出/从文件读入数据和平时的标准输入输出类似。

7.遇到的困难及解决方法

7.1 消去行的算法

在引入 rowState 结构之后，消去几行时，只需看这些行是否全为满，并找到相连通的行的上下界就可以了。但是确定上下界时遇到了一些困难，如果只记录每一行是否与别的行连接的话，向上连通与向下连通是不能加以区分的，可能会出现扩大范围的情况。所以将结构中的变量 linked 改为 int 型，利用二进制的第零位、第一位分别表示向下、向上连通，这样可以确定找到的上下界是准确的。又因为一次只会落下一个积木，所以只可能消去一次，所以在消去积木的代码之后加上 break 减少没有必要的时间开销。

7.2 向 vector 中加入积木后释放空间

在 vector 中插入的是积木的指针。一开始在插入指针之后，回收了积木的内存，而消去时试图再次回收内存，出现程序崩溃的情况，而一时没有定位到这里出错。之后删去了回收空间的代码，能够成功消去一行并回收积木占据的内存。

7.3 重新申请积木后初始化

在存档/读档/新游戏时需要先处理正在下落的积木，而在文件操作结束之后，需要重新申请一块积木，在运行时发现新申请的积木并不会慢慢下落，而是直接落下，干扰游戏。在对比了其他地方申请积木的代码之后，发现是在申请积木后并没有对 command 做初始化，而赋值为 UNDEFINED 之后就没有了这种问题。

7.4 访问权限限制

在作业的过程中经常出现在一个类中，试图访问另一个类的私有成员或私有方法的情况，如果一味加友元声明的话，有违封装的原则，也容易由于编码错误出现比较大的问题。采取的解决方法是在相应的类中添加成员函数，作为另一个类访问的接口，例如 CBoard 中的 clear 函数用于将 board 恢复到

开始游戏的状态，在 CPlayer 中被多次调用。

8.收获与问题

8.1 这次作业与之前的作业最明显的不同是代码不再局限于一个文件，而是分散于各个文件中。通过几次作业的练习，了解了在有多个文件时，应该在头文件中声明，在源文件中定义的做法，同时静态成员也最好初始化。但是也遇到了一些没有解决的问题，比如 CTrapezium 的代码在同名源文件中链接不通过而被迫转到 CBrick 中。

8.2 通过这次作业，对类与对象的概念有了初步的认识；也在不断报错的过程中，有了类的成员变量，成员函数的使用范围的概念，在编写代码的过程中也会考虑要访问那些变量，这个功能是写在哪一个类中比较合适的想法；虽然说最终的实现仍然有很多需要改进的地方，没有贯彻封装的思路，但是在做作业的过程中有了这种思维模式的雏形。

8.3 这次作业是第一次接触 windows 程序，尽管与操作系统对接的部分并不是自己编写，但是在阅读代码的过程中，通过在网上搜索和课上老师的讲解，也对 Windows 程序的运行机制有了一点浅显的认识。

9.总结

这一次作业是比较有难度的作业，但是拆解成一次次的小作业加上老师的讲解之后，作业的难度分散开了，也降低了一些难度。通过这次作业的训练，我对 C++的面向对象的程序设计有了基本的认识，也锻炼了用这种思想去解决问题的能力。