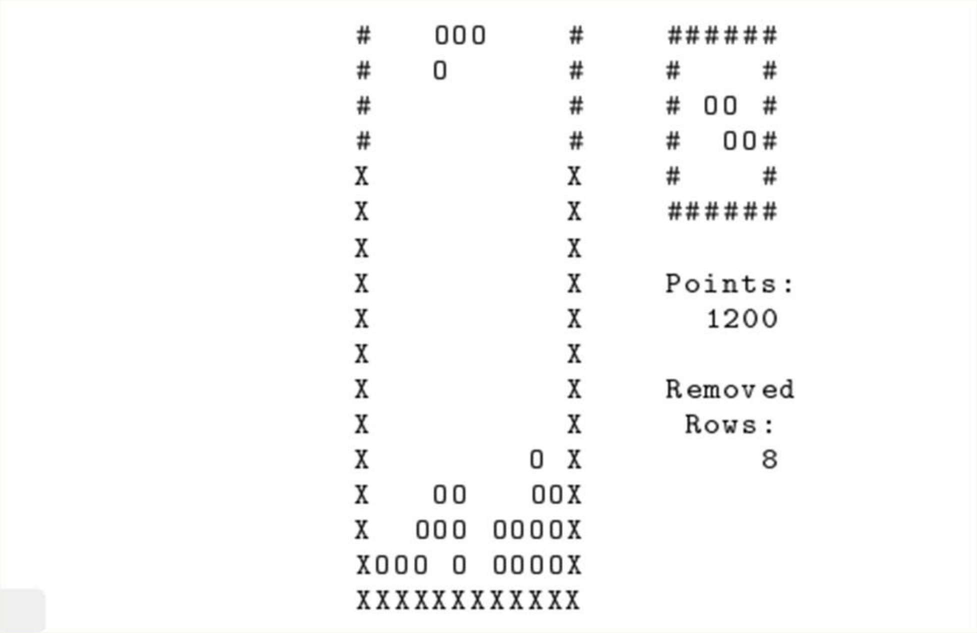
# Blocktris Game



```
#   000    #    ######
#   0      #    #    #
#          #    # 00 #
#          #    #  00#
X          X    #    #
X          X    ######
X          X
X          X    Points:
X          X      1200
X          X
X          X    Removed
X          X     Rows:
X        0 X         8
X   00   00X
X  000 0000X
X000 0 0000X
XXXXXXXXXXXX
```

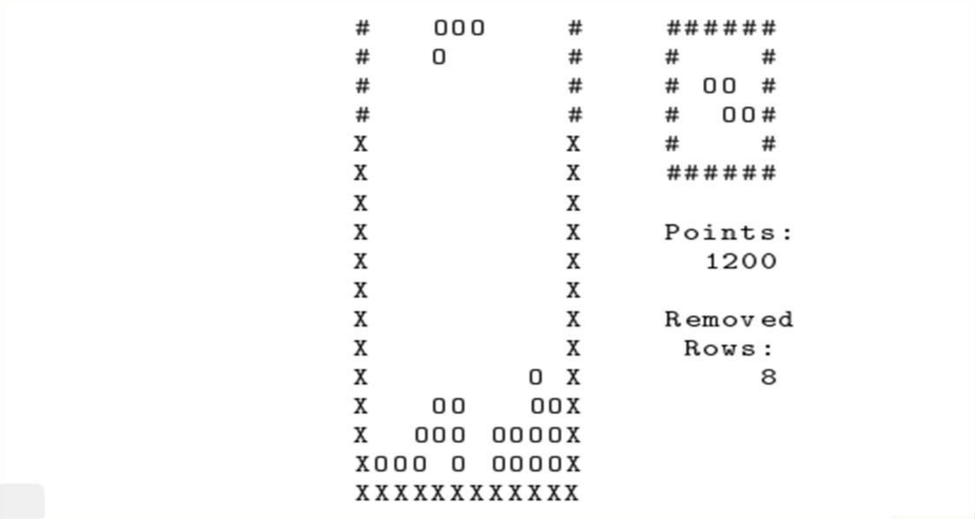
# Outline of the Presentation

- About Blocktris Game
- History of Blocktris Game
- Flow chat for our Blocktris game
- Main concept of the game.
- My Device class
- **Artificial intelligent (AI)** Deign
- Problem of AI Deign
- Future plan

# About Blocktris Game

# About

- Blocktris is a tile-matching puzzle video game.
- This is a variants of the game *Tetris.*
- There have 7 Block. I,L,T,J,S,O,Z
- Top 4 Row use for changing block



```
#   000    #    ######
#   0      #    #    #
#          #    # 00 #
#          #    #  00#
X          X    #    #
X          X    ######
X          X
X          X    Points:
X          X      1200
X          X
X          X    Removed
X          X     Rows:
X        0 X         8
X   00   00X
X  000 0000X
X000 0 0000X
XXXXXXXXXXXX
```

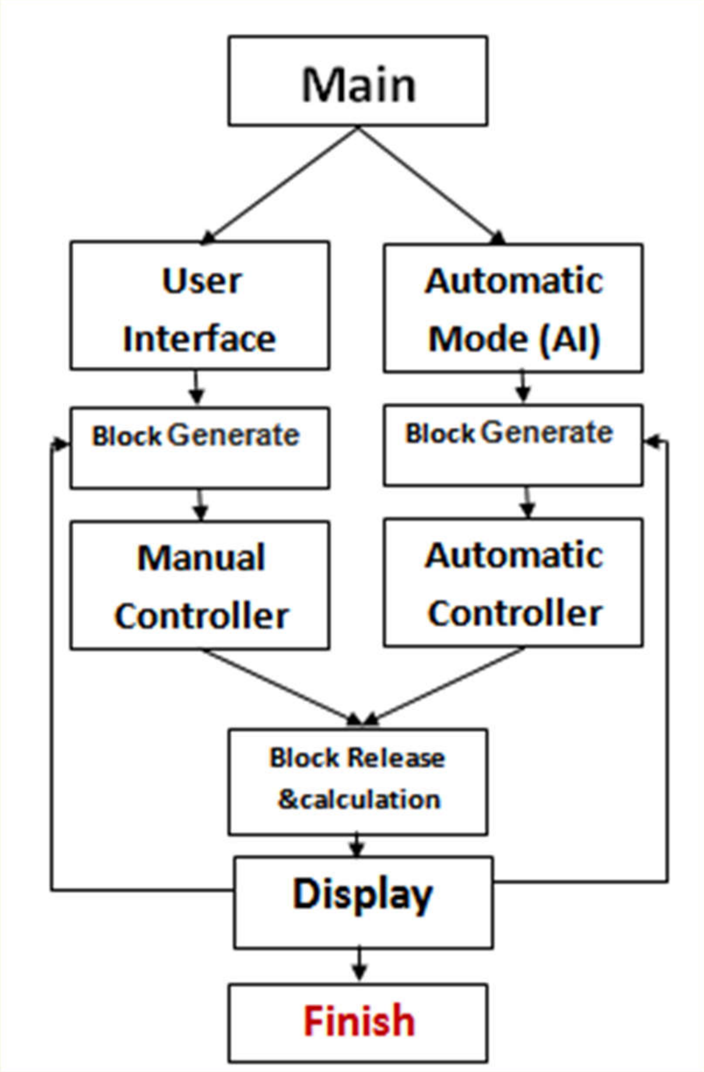
# History of Blocktris Game

# History

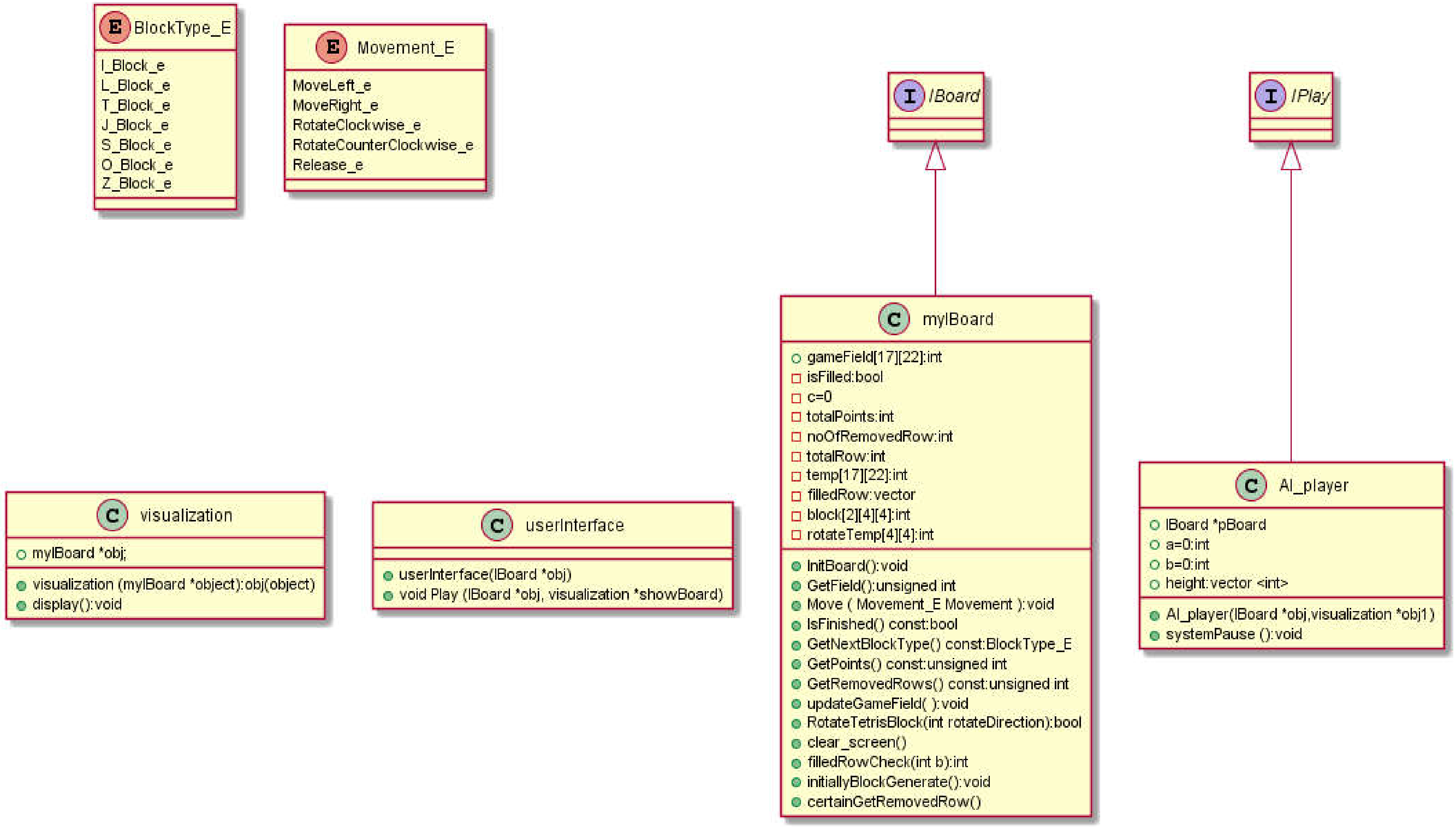
- This is a variants of the game *Tetris.*
- ***Tetris*** is first originally designed and programmed by Russian game designer Alexey Pajitnov (Wikipedia)
- It was first released on June 6, 1984
- In 19th century it was first place as "Greatest Game of All Time".
- In 2007, *It was* come in second place in "100 Greatest Video Games of All Time"

# Flow & UML chat for our Blocktris game

# Flow chat


Main
User Interface
Automatic Mode (AI)
Block Generate
Block Generate
Manual Controller
Automatic Controller
Block Release &calculation
Display
Finish

# UML chat



E BlockType_E

I_Block_e
L_Block_e
T_Block_e
J_Block_e
S_Block_e
O_Block_e
Z_Block_e

E Movement_E

MoveLeft_e
MoveRight_e
RotateClockwise_e
RotateCounterClockwise_e
Release_e

I *IBoard*

I *IPlay*

C myIBoard

- ○ gameField[17][22]:int
- □ isFilled:bool
- □ c=0
- □ totalPoints:int
- □ noOfRemovedRow:int
- □ totalRow:int
- □ temp[17][22]:int
- □ filledRow:vector
- □ block[2][4][4]:int
- □ rotateTemp[4][4]:int

- ● InitBoard():void
- ● GetField():unsigned int
- ● Move ( Movement_E Movement ):void
- ● IsFinished() const:bool
- ● GetNextBlockType() const:BlockType_E
- ● GetPoints() const:unsigned int
- ● GetRemovedRows() const:unsigned int
- ● updateGameField( ):void
- ● RotateTetrisBlock(int rotateDirection):bool
- ● clear_screen()
- ● filledRowCheck(int b):int
- ● initiallyBlockGenerate():void
- ● certainGetRemovedRow()

C AI_player

- ○ IBoard *pBoard
- ○ a=0:int
- ○ b=0:int
- ○ height:vector <int>

- ● AI_player(IBoard *obj,visualization *obj1)
- ● systemPause ():void

C visualization

- ○ myIBoard *obj;

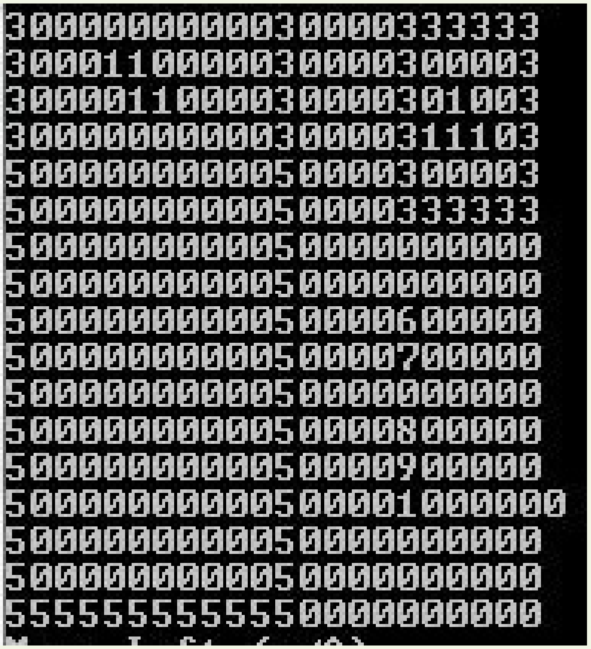
- ● visualization (myIBoard *object):obj(object)
- ● display():void

C userInterface

- ● userInterface(IBoard *obj)
- ● void Play (IBoard *obj, visualization *showBoard)

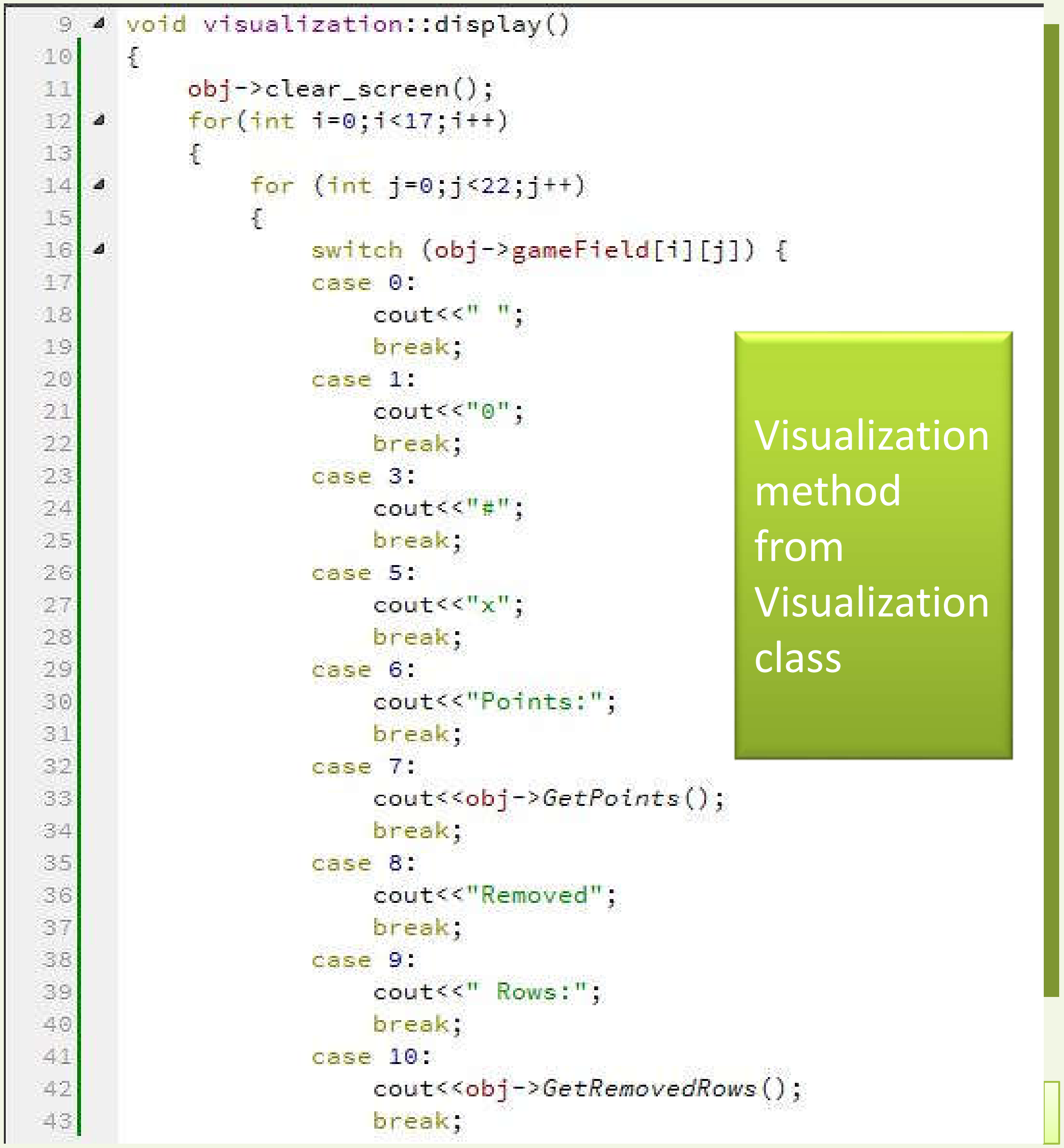
# Main concept of the game.

# Matrix concept



```
3000000000030000333333
3000110000030000300003
3000011000030000301003
3000000000030000311103
5000000000050000300003
5000000000050000333333
5000000000050000000000
5000000000050000000000
5000000000050000600000
5000000000050000700000
5000000000050000000000
5000000000050000800000
5000000000050000900000
50000000000500001000000
5000000000050000000000
5000000000050000000000
5555555555550000000000
```

- It has 22 rows And 17 column.
- Among them 12 rows is fixed for game Field.



```
void visualization::display()
{
    obj->clear_screen();
    for(int i=0;i<17;i++)
    {
        for (int j=0;j<22;j++)
        {
            switch (obj->gameField[i][j]) {
            case 0:
                cout<<" ";
                break;
            case 1:
                cout<<"0";
                break;
            case 3:
                cout<<"#";
                break;
            case 5:
                cout<<"x";
                break;
            case 6:
                cout<<"Points:";
                break;
            case 7:
                cout<<obj->GetPoints();
                break;
            case 8:
                cout<<"Removed";
                break;
            case 9:
                cout<<" Rows:";
                break;
            case 10:
                cout<<obj->GetRemovedRows();
                break;
```

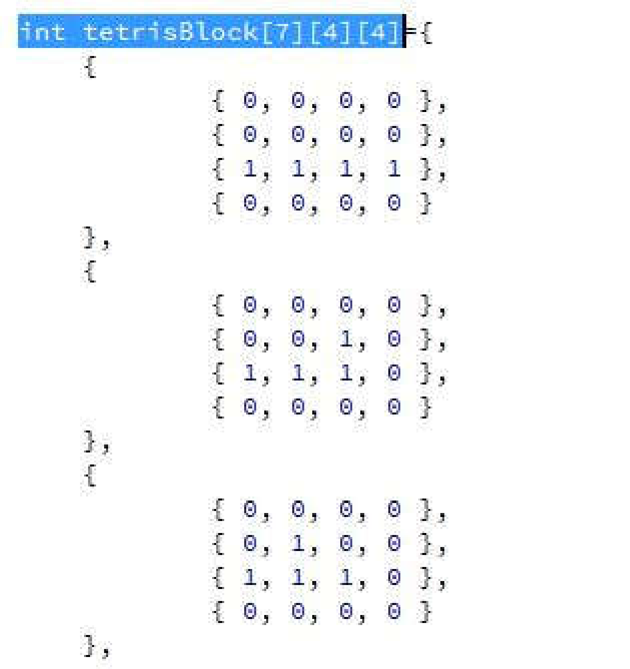
Visualization method from Visualization class

# MyDerived Class -> Few Important Variables

- int gameField[17][22]
- int temp[17][22]
- int block[2][4][4]
- int tetrisBlock[7][4][4]

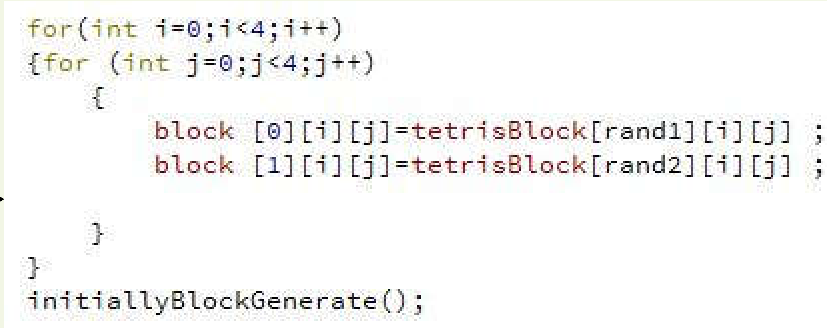
# MyDerived Class -> Few Important Variables

- **int tetrisBlock[7][4][4]**



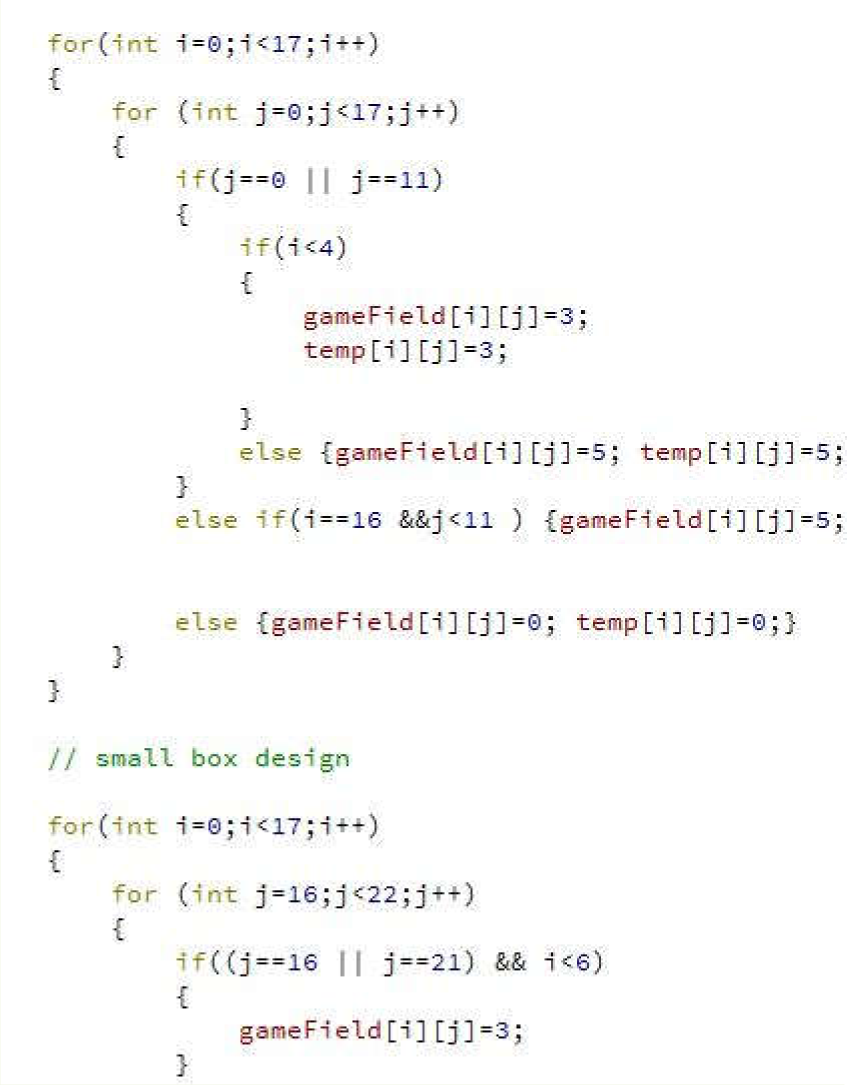
```
int tetrisBlock[7][4][4]={
    {
            { 0, 0, 0, 0 },
            { 0, 0, 0, 0 },
            { 1, 1, 1, 1 },
            { 0, 0, 0, 0 }
    },
    {
            { 0, 0, 0, 0 },
            { 0, 0, 1, 0 },
            { 1, 1, 1, 0 },
            { 0, 0, 0, 0 }
    },
    {
            { 0, 0, 0, 0 },
            { 0, 1, 0, 0 },
            { 1, 1, 1, 0 },
            { 0, 0, 0, 0 }
    },
```

- **How Block is Copying into block**



```
for(int i=0;i<4;i++)
{for (int j=0;j<4;j++)
    {
        block [0][i][j]=tetrisBlock[rand1][i][j] ;
        block [1][i][j]=tetrisBlock[rand2][i][j] ;

    }
}
initiallyBlockGenerate();
```

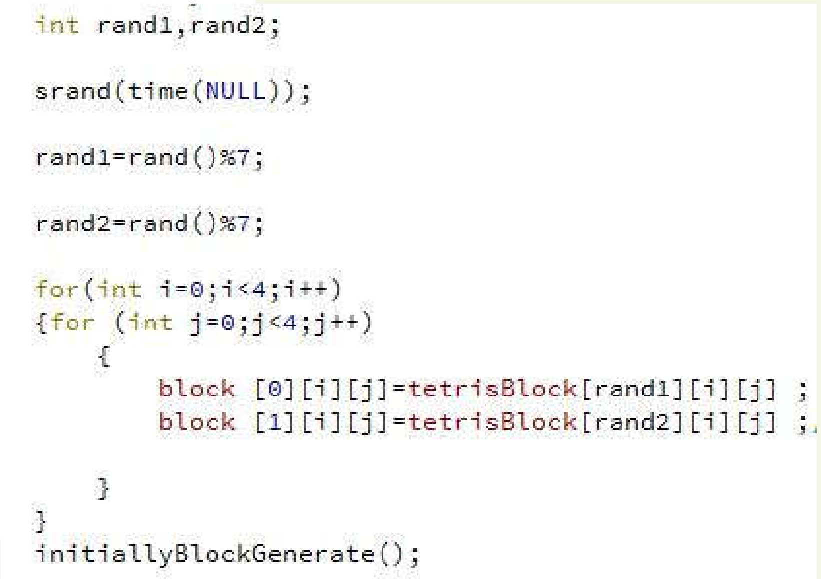
# MyDerived Class-> void *InitBoard*()



```
for(int i=0;i<17;i++)
{
    for (int j=0;j<17;j++)
    {
        if(j==0 || j==11)
        {
            if(i<4)
            {
                gameField[i][j]=3;
                temp[i][j]=3;

            }
            else {gameField[i][j]=5; temp[i][j]=5;}
        }
        else if(i==16 &&j<11 ) {gameField[i][j]=5; temp[i][j]=5;}


        else {gameField[i][j]=0; temp[i][j]=0;}
    }
}

// small box design

for(int i=0;i<17;i++)
{
    for (int j=16;j<22;j++)
    {
        if((j==16 || j==21) && i<6)
        {
            gameField[i][j]=3;
        }
```

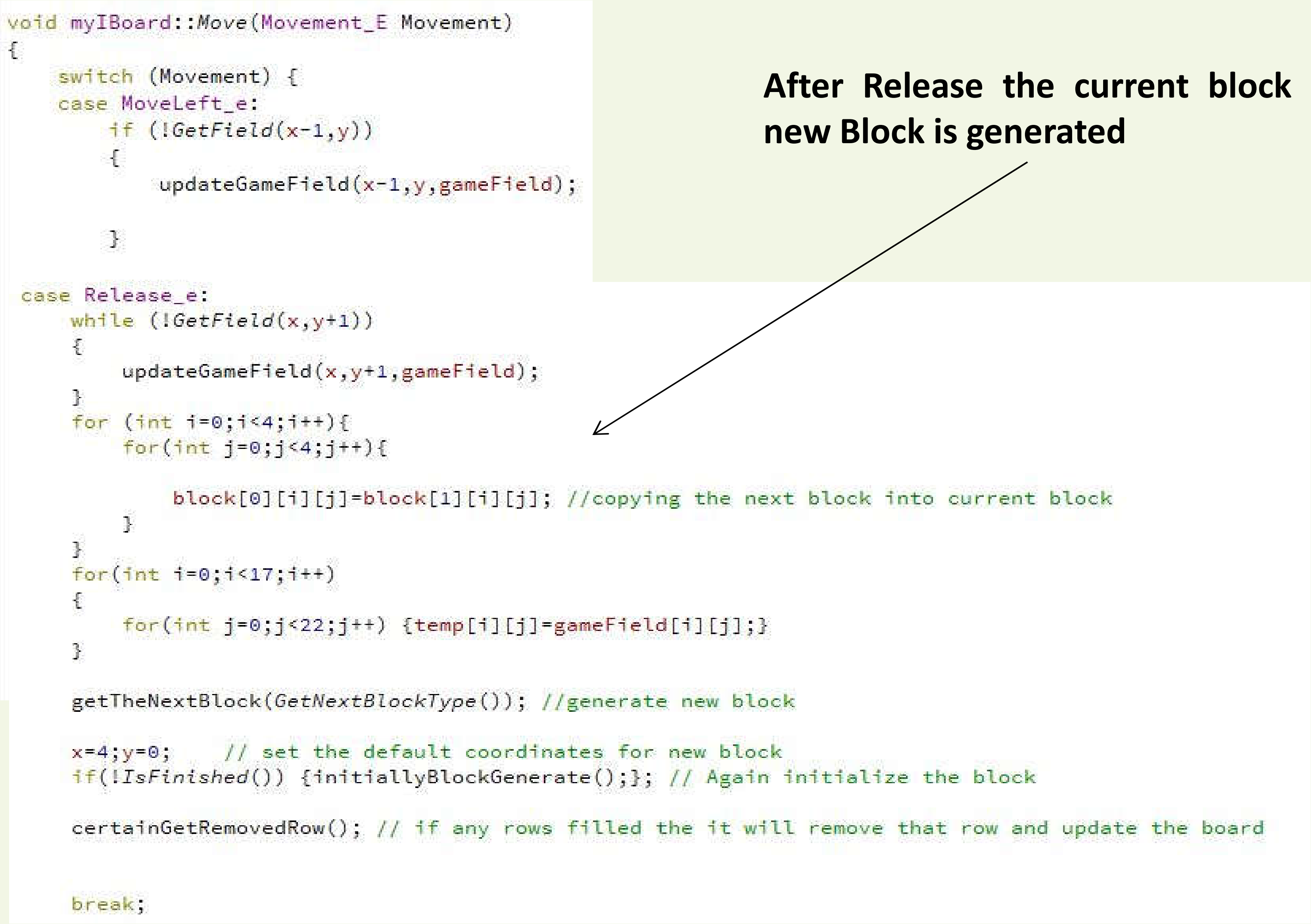
Designing the game Field

Initially Block Generate



```
int rand1,rand2;

srand(time(NULL));

rand1=rand()%7;

rand2=rand()%7;

for(int i=0;i<4;i++)
{for (int j=0;j<4;j++)
    {
        block [0][i][j]=tetrisBlock[rand1][i][j] ;
        block [1][i][j]=tetrisBlock[rand2][i][j] ;

    }
}
initiallyBlockGenerate();
```

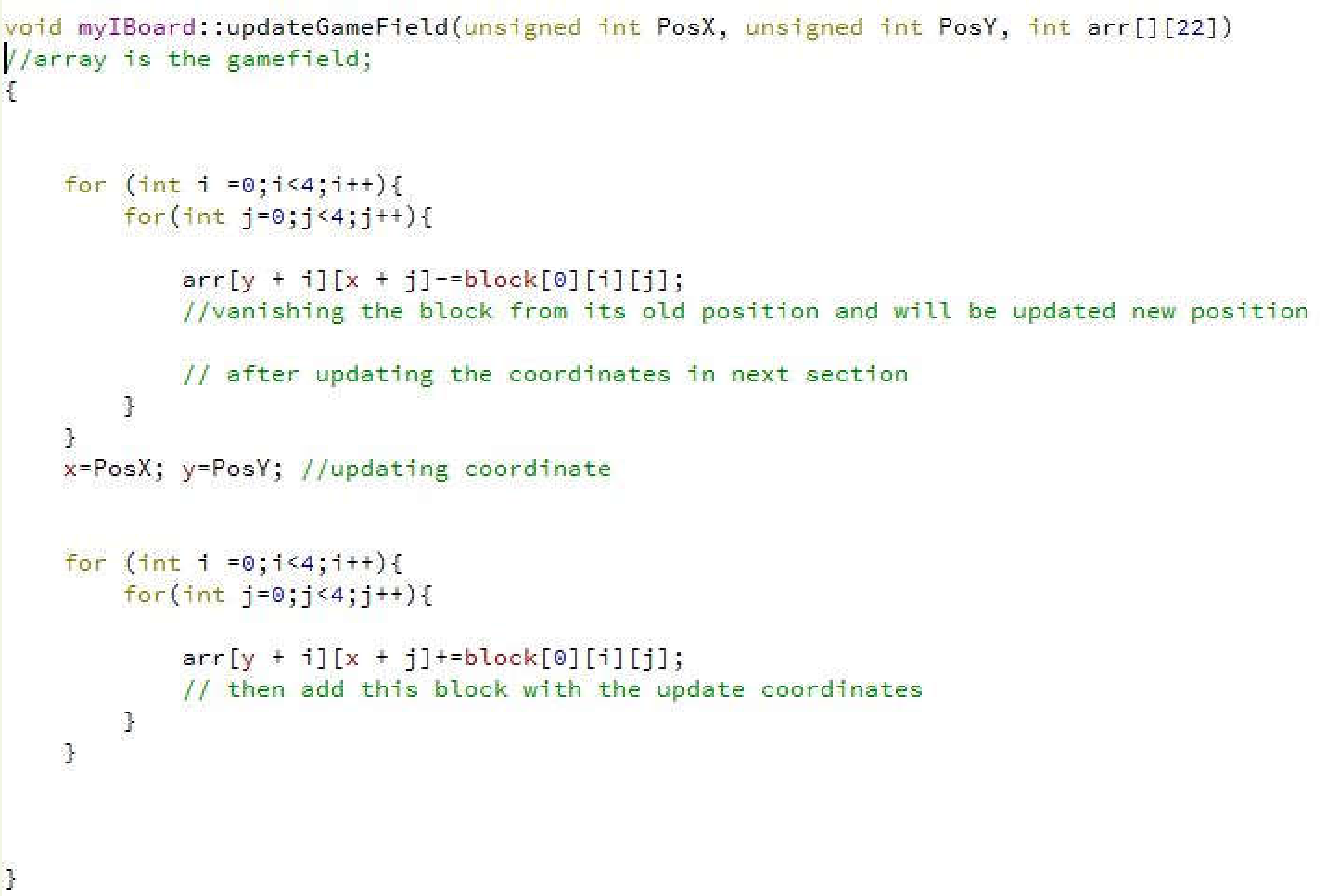
# MyDerived Class ->*Move*(Movement_E Movement)



```
void myIBoard::Move(Movement_E Movement)
{
    switch (Movement) {
    case MoveLeft_e:
        if (!GetField(x-1,y))
        {
            updateGameField(x-1,y,gameField);

        }

 case Release_e:
    while (!GetField(x,y+1))
    {
        updateGameField(x,y+1,gameField);
    }
    for (int i=0;i<4;i++){
        for(int j=0;j<4;j++){

            block[0][i][j]=block[1][i][j]; //copying the next block into current block
        }
    }
    for(int i=0;i<17;i++)
    {
        for(int j=0;j<22;j++) {temp[i][j]=gameField[i][j];}
    }

    getTheNextBlock(GetNextBlockType()); //generate new block

    x=4;y=0;    // set the default coordinates for new block
    if(!IsFinished()) {initiallyBlockGenerate();}; // Again initialize the block

    certainGetRemovedRow(); // if any rows filled the it will remove that row and update the board


    break;
```

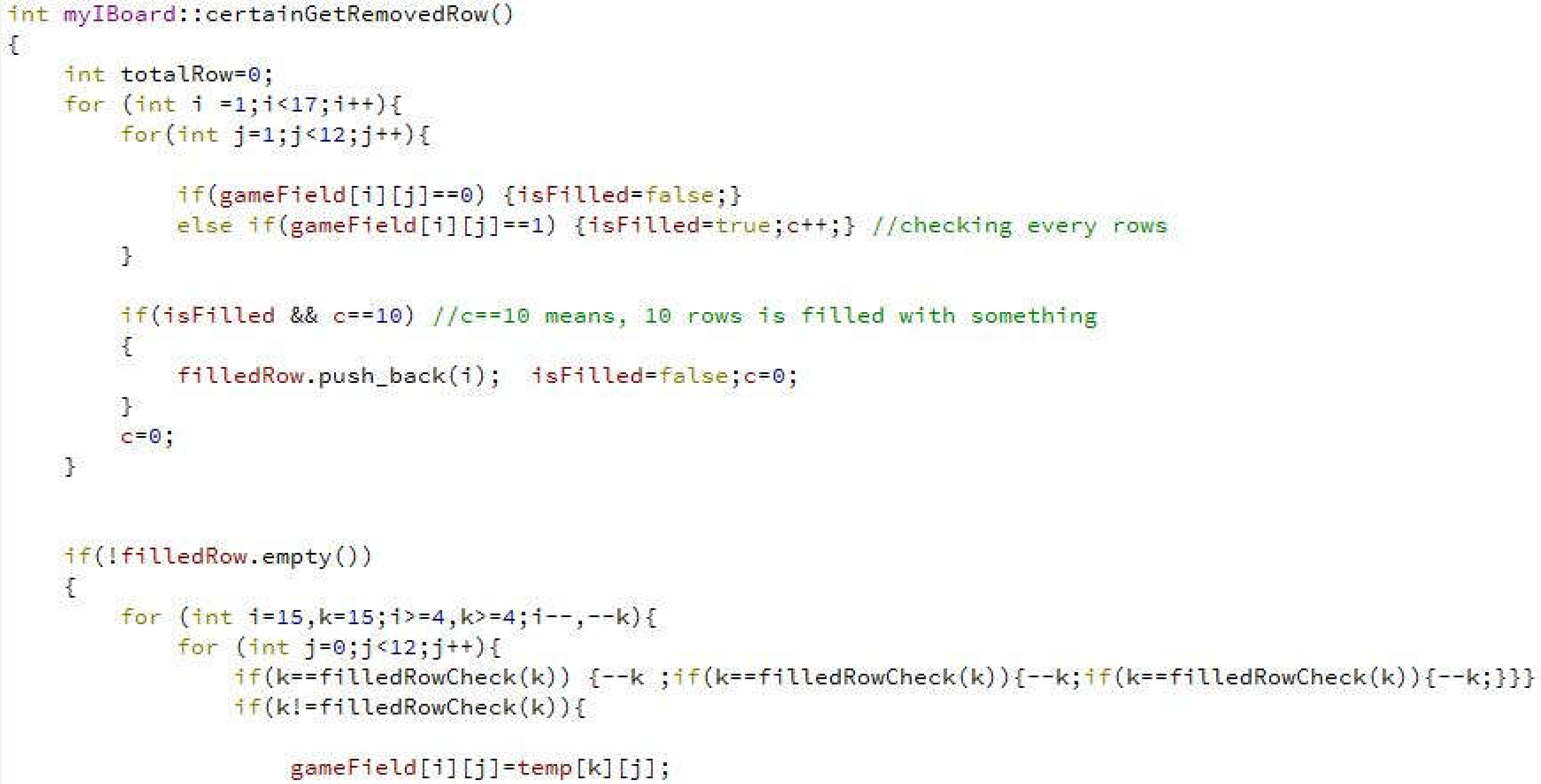
**After Release the current block new Block is generated**

# MyDerived Class -> updateGameField

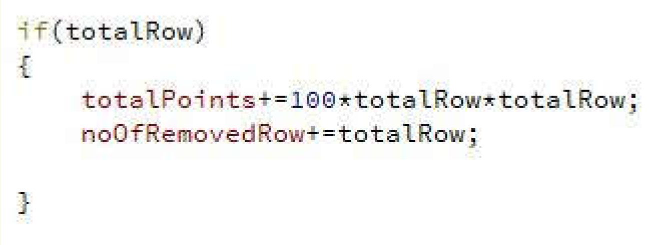


```
void myIBoard::updateGameField(unsigned int PosX, unsigned int PosY, int arr[][22])
//array is the gamefield;
{


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

            arr[y + i][x + j]-=block[0][i][j];
            //vanishing the block from its old position and will be updated new position

            // after updating the coordinates in next section
        }
    }
    x=PosX; y=PosY; //updating coordinate


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

            arr[y + i][x + j]+=block[0][i][j];
            // then add this block with the update coordinates
        }
    }




}
```

# MyDerived Class -> CertainGetRemovedRow

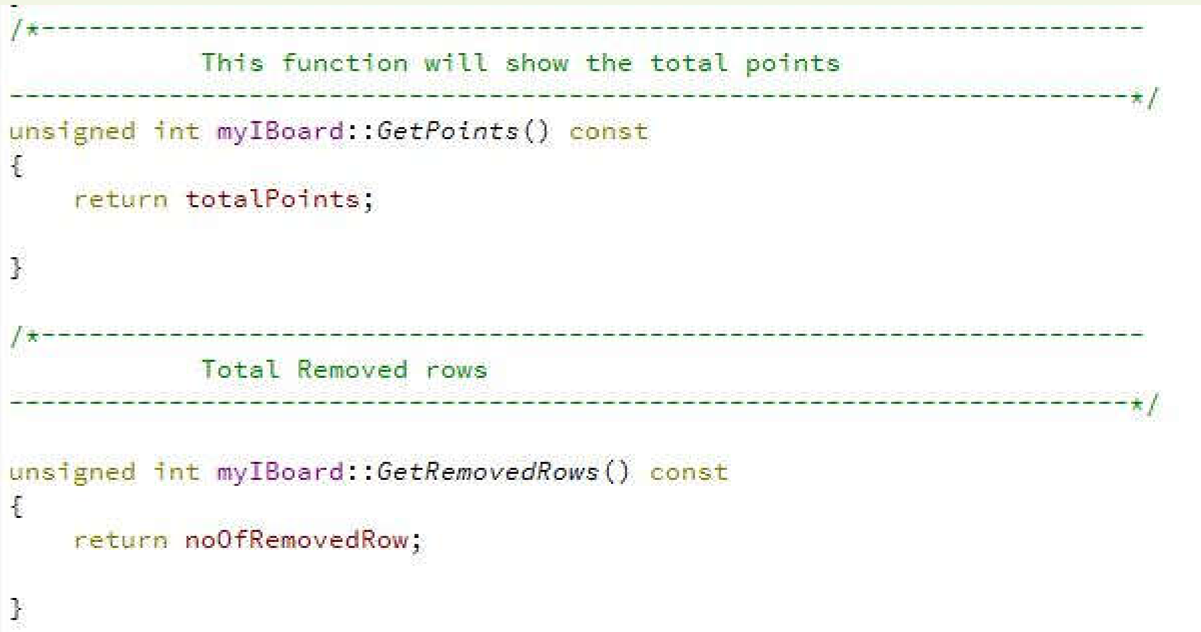


```
int myIBoard::certainGetRemovedRow()
{
    int totalRow=0;
    for (int i =1;i<17;i++){
        for(int j=1;j<12;j++){

            if(gameField[i][j]==0) {isFilled=false;}
            else if(gameField[i][j]==1) {isFilled=true;c++;} //checking every rows
        }

        if(isFilled && c==10) //c==10 means, 10 rows is filled with something
        {
            filledRow.push_back(i);  isFilled=false;c=0;
        }
        c=0;
    }


    if(!filledRow.empty())
    {
        for (int i=15,k=15;i>=4,k>=4;i--,--k){
            for (int j=0;j<12;j++){
                if(k==filledRowCheck(k)) {--k ;if(k==filledRowCheck(k)){--k;if(k==filledRowCheck(k)){--k;}}}
                if(k!=filledRowCheck(k)){

                    gameField[i][j]=temp[k][j];
```



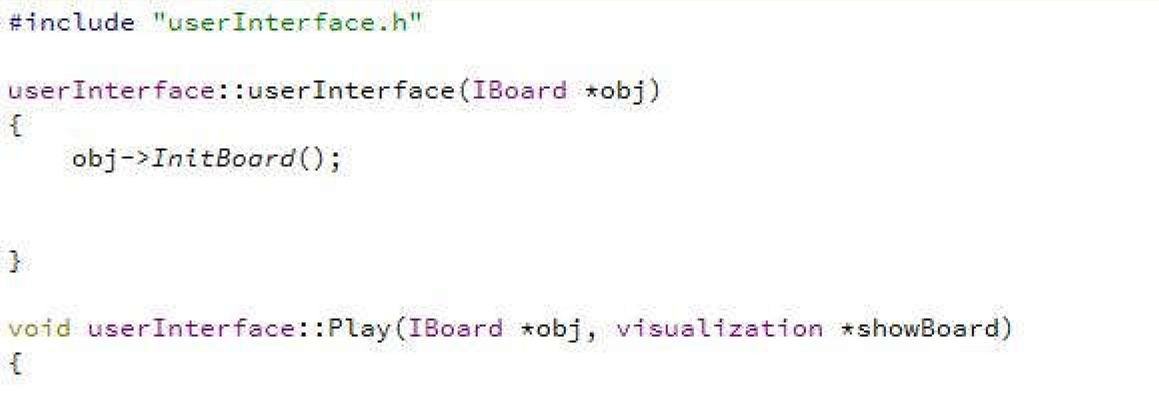
```
if(totalRow)
{
    totalPoints+=100*totalRow*totalRow;
    noOfRemovedRow+=totalRow;

}
```

# MyDerived Class-> total points & removed rows

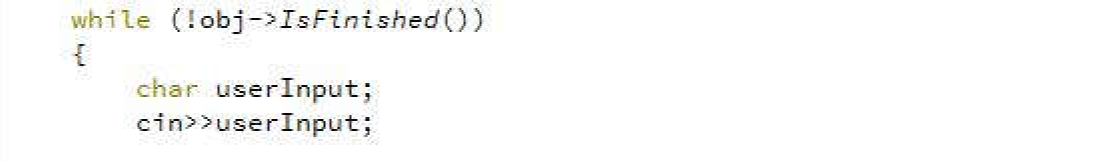


```
/*----------------------------------------------------------------------
            This function will show the total points
----------------------------------------------------------------------*/
unsigned int myIBoard::GetPoints() const
{
    return totalPoints;

}

/*----------------------------------------------------------------------
            Total Removed rows
----------------------------------------------------------------------*/

unsigned int myIBoard::GetRemovedRows() const
{
    return noOfRemovedRow;

}
```

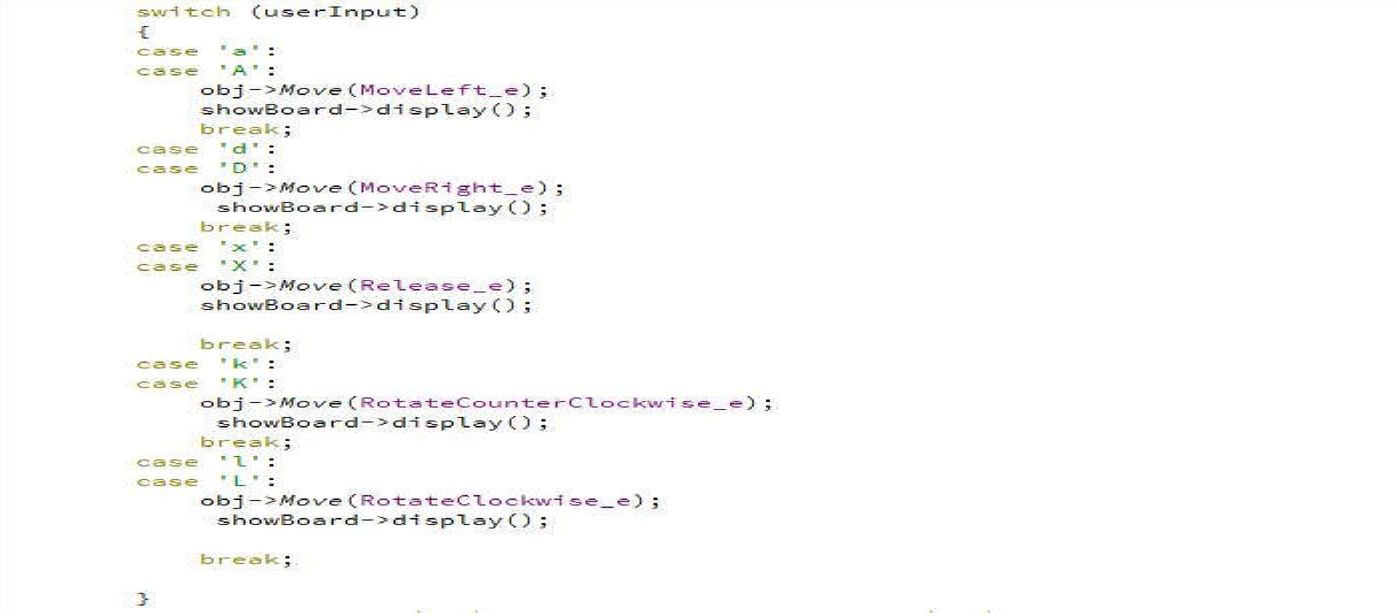
# MyDerived Class-> UserInterfaceClass



```
#include "userInterface.h"

userInterface::userInterface(IBoard *obj)
{
    obj->InitBoard();


}

void userInterface::Play(IBoard *obj, visualization *showBoard)
{
```



```
    while (!obj->IsFinished())
    {
        char userInput;
        cin>>userInput;
```

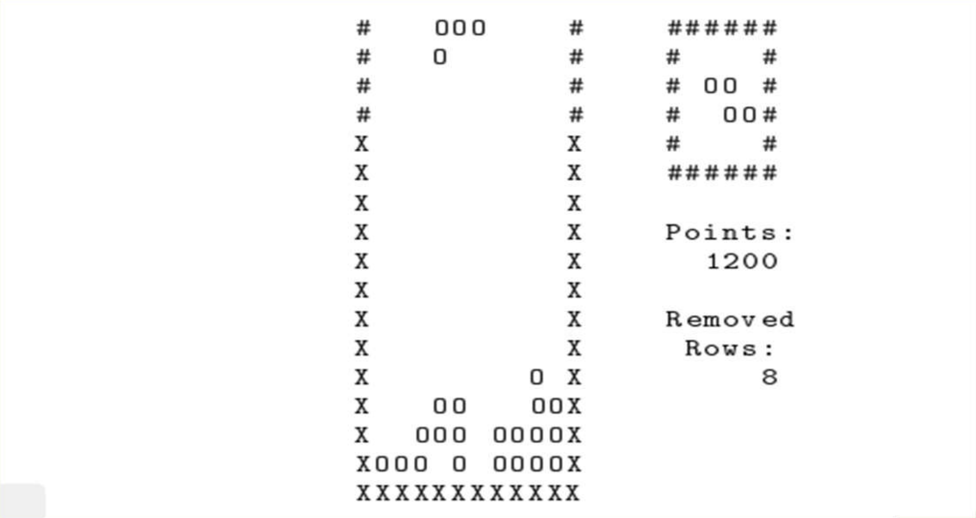


```
        switch (userInput)
        {
        case 'a':
        case 'A':
            obj->Move(MoveLeft_e);
            showBoard->display();
            break;
        case 'd':
        case 'D':
            obj->Move(MoveRight_e);
             showBoard->display();
            break;
        case 'x':
        case 'X':
            obj->Move(Release_e);
            showBoard->display();

            break;
        case 'k':
        case 'K':
            obj->Move(RotateCounterClockwise_e);
             showBoard->display();
            break;
        case 'l':
        case 'L':
            obj->Move(RotateClockwise_e);
             showBoard->display();

            break;

        }
```

# Artificial intelligent (AI)

# AI

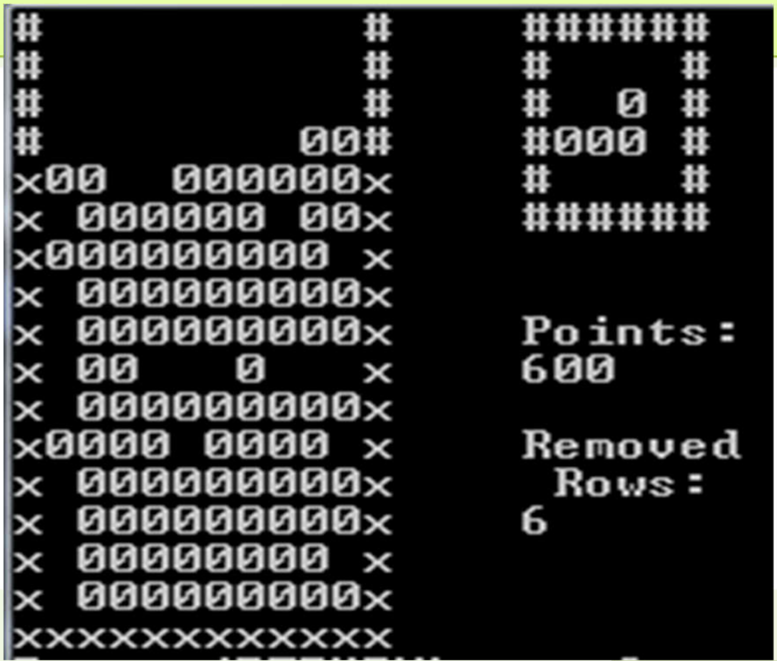
- Sum Height - Remove line + inter Hole + Sum of difference every Colum height to privies Colum  height=Lowest number.
- Use one block 4*10=40/4*8=32.
- **Problem is not use another matrix or we use fixed interface in  AI.**
- So we think another way……….



```
#   000    #    ######
#   0      #    #    #
#          #    # 00 #
#          #    #  00#
X          X    #    #
X          X    ######
X          X
X          X    Points:
X          X      1200
X          X
X          X    Removed
X          X     Rows:
X        0 X         8
X   00    00X
X  000 0000X
X000 0 0000X
XXXXXXXXXXXX
```

# AI Design and Problem

- Only Height
- Use one block 4*10=40 / 4*8=32.
- **Which height lowest .**
- Problem : light side first Colum not fill up .



```
#          #    ######
#          #    #    #
#          #    #  0 #
#        00#    #000 #
x00  000000x    #    #
x 000000 00x    ######
x000000000 x
x 000000000x
x 000000000x    Points:
x 00   0   x    600
x 000000000x
x0000 0000 x    Removed
x 000000000x     Rows:
x 000000000x    6
x 00000000 x
x 000000000x
xxxxxxxxxxxx
```

# Future Plan

# Future Plan

- Try to solve my problem left side first Colum .
- Use fist 5 left to right than change right to left
- Apply another method.

# Thank you

# Question & Answer

?