國立高學大學一下OOP期末專題報告 第七組  
組員：

A1005504 戴佳燊

A1005539 李羿杰

A1005541 謝詠炘

目錄：

程式碼原文：

Ball.h p.3

Ball.cpp p.4

Board.h p.6

Board.cpp p.8

Boundary.h p.18

Boundary.cpp p.20

Game.h p.21

Game.cpp p.24

myKeyboard.h p.31

myKeyboard.cpp p.36

vCanvas.h p.40

vCanvas.cpp p.42

Wall.h p.60

Wall.cpp p.61

參考的技術： p.62

/\*Ball.h\*/

#ifndef \_BALL\_H\_

#define \_BALL\_H\_

#define dLeftDefaultBallX LConsoleMidPoint

#define dRightDefaultBallX LConsoleMidPoint

#define dDefaultBallY 12

#define dBallmoverate 0

class Ball

{

private:

int X;

int Y;

int Delay;

int MaxDelay;

bool isDown, isRight;

public:

Ball(int = 0, int = 0, bool = false, bool = false, int = dBallmoverate, int = dBallmoverate);

void setPosition(bool);

void setDirection(bool,bool);

void randDirection();

void move();

int getxpos()const;

int getypos()const;

bool getIsDown() const;

void setDelay(int);

int getDelay() const;

void setMaxDelay(int);

int getMaxDelay() const;

void setBall(bool);

void adjustYPosition(int);

};

#endif

/\*Ball.cpp\*/

#include "stdafx.h"

#include <iostream>

#include <cstdlib>

#include <ctime>

#include "Ball.h"

#include "vCanvas.h"

using namespace std;

/\*private:

int X;

int Y;

int Delay;

bool isDown, isRight;

\*/

Ball::Ball(int x, int y, bool isD, bool isR, int delay, int maxd)

:X(x), Y(y), isDown(isD), isRight(isR), Delay(delay), MaxDelay(maxd)

{

srand( (unsigned int)time(NULL) );

setPosition(true);

randDirection();

}

void Ball::setPosition(bool isLeftLose)

{

X = (isLeftLose)? dLeftDefaultBallX:dRightDefaultBallX, Y = dDefaultBallY-2+rand()%5;

}

void Ball::randDirection()

{

isDown = (rand()%2 == 1)? true:false;

isRight = (rand()%2 == 1)? true:false;

}

void Ball::move()

{

X += (isRight)? 2:(-2);

Y += (isDown)? 1:(-1);

}

int Ball::getxpos()const

{

return X;

}

int Ball::getypos()const

{

return Y;

}

void Ball::setDirection(bool ChangeIsR,bool ChangeIsD)

{

Beep(784,2000/64);

if(ChangeIsR)

{

isRight = !isRight;

}

if(ChangeIsD)

{

isDown = !isDown;

}

}

bool Ball::getIsDown() const

{

return isDown;

}

void Ball::setDelay(int d)

{

Delay = d;

}

int Ball::getDelay() const

{

return Delay;

}

void Ball::setMaxDelay(int d)

{

MaxDelay = d;

}

int Ball::getMaxDelay() const

{

return MaxDelay;

}

void Ball::setBall(bool isleftlose)

{

setPosition(isleftlose);

setMaxDelay(4);

}

void Ball::adjustYPosition(int newY)

{

if(newY>2&&newY<22)

Y = newY;

else

X = (rand()%2 == 1)? X+2 : X-2;

}

/\*Board.h\*/

#ifndef \_BOARD\_H\_

#define \_BOARD\_H\_

#include "Ball.h"

#define dLeftBoardLine 10

#define dRightBoardLine (ConsoleBufferWidth-12)

#define dDefaultUend 10

#define dDefaultDend 14

class Board

{

protected:

int X;

int Y\_UpperEnd;//small

int Y\_LowerEnd;//big

bool isUp;

bool isDown;

bool isLPlayer;

static int Countdown;

static int Delay;

public:

Board(int = 0, int = 0, int = 0, bool = false, bool = false, bool = true);

int getX() const;

int getYUpEnd() const;

int getYDownEnd() const;

bool getIsLPlayer() const;

virtual void setPosition(bool);

void move();

virtual void collision(Ball &);

void setIsUp(bool);

void setIsDown(bool);

static void setDelay(int);

int getDelay() const;

static void setCountDown(int);

int getCountDown() const;

};

#endif

/\*Board.cpp\*/

#include "stdafx.h"

#include "Board.h"

#include "vCanvas.h"

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

/\*

private:

int X;

int Y\_UpperEnd;//small

int Y\_LowerEnd;//big

bool isUp;

bool isDown;

bool isLPlayer;

static int Countdown;

static int Delay;

\*/

int Board::Countdown = 0;

int Board::Delay = 0;

Board::Board(int x, int yu, int yd, bool isu, bool isd, bool isl)

:X(x), Y\_UpperEnd(yu), Y\_LowerEnd(yd), isUp(isu), isDown(isd), isLPlayer(isl)

{

setPosition(isl);

}

int Board::getX() const

{

return X;

}

int Board::getYUpEnd() const

{

return Y\_UpperEnd;

}

int Board::getYDownEnd() const

{

return Y\_LowerEnd;

}

void Board::setPosition(bool isLeft)

{

X = (isLeft)? dLeftBoardLine:dRightBoardLine;

Y\_UpperEnd = dDefaultUend;

Y\_LowerEnd = dDefaultDend;

isLPlayer = isLeft;

}

void Board::move()

{

if(isUp&&isDown)

{

setIsUp(false);

setIsDown(false);

}

else if(isUp)

{

if(getYUpEnd()<=2)

{}

else

{

Y\_UpperEnd--;

Y\_LowerEnd--;

}

}

else if(isDown)

{

if(getYDownEnd()>=22)

{}

else

{

Y\_UpperEnd++;

Y\_LowerEnd++;

}

}

setIsUp(false);

setIsDown(false);

}

void Board::collision(Ball &ball)

{

if(isLPlayer)

{

if(ball.getxpos()-getX() == 2)

{

if( (ball.getypos()-getYUpEnd() == -1 && ball.getIsDown()) || (ball.getypos() - getYDownEnd() == 1 && !ball.getIsDown()) )

{

ball.setDirection(true,true);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

else if(ball.getypos() >= getYUpEnd() && ball.getypos() <= getYDownEnd() )

{

ball.setDirection(true,false);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

}

if(ball.getxpos()-getX() == 0)

{

if(ball.getypos()-getYUpEnd() == -1)//above

{

if(ball.getIsDown())

{

ball.setDirection(false,true);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

}

else if(ball.getypos() - getYDownEnd() == 1)//below

{

if(!ball.getIsDown())

{

ball.setDirection(false,true);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

}

else if(ball.getypos() == getYUpEnd())//if the ball is inside the board(up)

{

ball.adjustYPosition(getYUpEnd()-1);

if(ball.getIsDown())

ball.setDirection(false,true);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

else if(ball.getypos() == getYDownEnd() )//if the ball is inside the board(down)

{

ball.adjustYPosition(getYDownEnd()+1);

if(!ball.getIsDown())

ball.setDirection(false,true);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

}

}

else

{

if(getX()-ball.getxpos() == 2)

{

if( (ball.getypos()-getYUpEnd() == -1 && ball.getIsDown()) || (ball.getypos() - getYDownEnd() == 1 && !ball.getIsDown()) )

{

ball.setDirection(true,true);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

else if(ball.getypos() >= getYUpEnd() && ball.getypos() <= getYDownEnd() )

{

ball.setDirection(true,false);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

}

if(ball.getxpos()-getX() == 0)

{

if(ball.getypos()-getYUpEnd() == -1)//above

{

if(ball.getIsDown())

{

ball.setDirection(false,true);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

}

else if(ball.getypos() - getYDownEnd() == 1)//below

{

if(!ball.getIsDown())

{

ball.setDirection(false,true);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

}

else if(ball.getypos() == getYUpEnd())//if the ball is inside the board(up)

{

ball.adjustYPosition(getYUpEnd()-1);

if(ball.getIsDown())

ball.setDirection(false,true);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

else if(ball.getypos() == getYDownEnd() )//if the ball is inside the board(down)

{

ball.adjustYPosition(getYDownEnd()+1);

if(!ball.getIsDown())

ball.setDirection(false,true);

if(Countdown == 0)

{

if(ball.getMaxDelay()>1)

{

ball.setMaxDelay(ball.getMaxDelay()-1);

}

Countdown = 9;

}

else

{

Countdown--;

}

}

}

}

}

bool Board::getIsLPlayer() const

{

return isLPlayer;

}

void Board::setIsUp(bool in)

{

isUp = in;

}

void Board::setIsDown(bool in)

{

isDown = in;

}

void Board::setDelay(int c)

{

Delay = c;

}

int Board::getDelay() const

{

return Delay;

}

void Board::setCountDown(int c)

{

Countdown = c;

}

int Board::getCountDown() const

{

return Countdown;

}

/\*Boundary.h\*/

#ifndef \_BOUNDARY\_H\_

#define \_BOUNDARY\_H\_

#include "Board.h"

#include "Ball.h"

class Boundary : public Board

{

private:

static int LPoint ;

static int RPoint ;

public:

Boundary(int = 0, int = 0, bool = true);

virtual void collision(Ball &);

void resetPoint();

void plusPoint(Ball&, bool);//reset ball here;

static int getLpoint();

static int getRpoint();

virtual void setPosition(bool);

};

#endif

/\*Boundary.cpp\*/

#include "stdafx.h"

#include "Boundary.h"

#include "vCanvas.h"

#include "Board.h"

#include "Ball.h"

#define dLeftBoundary 0

#define dRightBoundary (ConsoleBufferWidth-2)

/\*

class Boundary : public Board

{

private:

static int LPoint;

static int RPoint;

\*/

int Boundary::LPoint = 0;

int Boundary::RPoint = 0;

Boundary::Boundary(int x, int y, bool isl)

:Board(x, y, y, false, false, isl)

{

resetPoint();

}

void Boundary::collision(Ball &ball)

{

if(ball.getxpos() == Board::getX())

{

if(Board::getIsLPlayer())

{

plusPoint(ball, true);

}

else

{

plusPoint(ball, false);

}

}

}

void Boundary::resetPoint()

{

LPoint = 0, RPoint = 0;

}

void Boundary::plusPoint(Ball& ball, bool isLeftBoundaryHit)//reset ball here;

{

if(isLeftBoundaryHit)

{

RPoint++;

ball.setBall(true);

}

else

{

LPoint++;

ball.setBall(false);

}

ball.randDirection();

}

int Boundary::getLpoint()

{

return LPoint;

}

int Boundary::getRpoint()

{

return RPoint;

}

void Boundary::setPosition(bool isLeft)

{

Board::X = (isLeft)?dLeftBoundary:dRightBoundary;

Board::isLPlayer = isLeft;

}

/\*Game.h\*/

#ifndef \_GAME\_H\_

#define \_GAME\_H\_

#include <iostream>

#include "Ball.h"

#include "Board.h"

#include "Wall.h"

#include "Ball.h"

#include "Boundary.h"

using namespace std;

class Game

{

private:

Ball NowBall;

Board Player1, Player2;

Wall UpWall, DownWall;

Boundary LeftBound, RightBound;

bool isPause;

bool isCounting;

bool isEnd;

int GameStatus;//0 = gaming 1 = p1 win, 2 = p2 win, 3 = pause 4 = countdown"3" 5 = countdown"2" 6 = countdown"1"

static int CountdownPauseFrame;

public:

Game();

void ResetGame();

void Calculate(unsigned int);

void pmove();

void bmove();

void collision();

int getLscore()const;

int getRscore()const;

bool getIsPause()const;

void changePause();

bool getIsCounting()const;

void changeCounting();

void setGameStatus(int);

int getGameStatus()const;

const Ball& getBallRef() const;

const Board& getP1Ref() const;

const Board& getP2Ref() const;

const Wall& getWallURef() const;

const Wall& getWallDRef() const;

const Boundary& getLBRef() const;

const Boundary& getRBRef() const;

static void setCountdownPauseFrame(int);

static int getCountdownPauseFrame();

void checkBallAndBoard();

};

#endif

/\*Game.cpp\*/

#include "stdafx.h"

#include <windows.h>

#include "Game.h"

#include "Ball.h"

#include "Board.h"

#include "Wall.h"

#include "Boundary.h"

#include "vCanvas.h"

/\*private:

Ball NowBall;

Board Player1, Player2;

Wall UpWall, DownWall;

Boundary LeftBound, RightBound;

bool isPause;

bool isCounting;

bool isEnd;

int GameStatus;//0 = gaming 1 = p1 win, 2 = p2 win, 3 = pause 4 = countdown"3" 5 = countdown"2" 6 = countdown"1"

static int CountdownPauseFrame;

\*/

int Game::CountdownPauseFrame = 64;

Game::Game()

{

ResetGame();

}

void Game::ResetGame()

{

NowBall.setBall(true);//reset ball;

LeftBound.resetPoint();//reset point;

Player1.setPosition(true);//reset player position

Player2.setPosition(false);

UpWall.setPosition(true);//reset wall position

DownWall.setPosition(false);

LeftBound.setPosition(true);//reset boundary position

RightBound.setPosition(false);

isPause = false;

isCounting = true;

isEnd = false;

Board::setCountDown(0);

setGameStatus(4);

Beep(1046,2000/64);

}

void Game::Calculate(unsigned int keycode)

{

if(keycode == 13&&!getIsCounting())//enter

{

changePause();

if(getIsPause())

{

Beep(1046,100);

Beep(784,100);

Beep(1046,100);

Beep(784,100);

setGameStatus(3);

}

else if(getGameStatus() == 1 || getGameStatus() == 2)

ResetGame();

else

setGameStatus(0);

}

if(!getIsPause())

{

if(keycode == 87)//W

{

Player1.setIsUp(true);

}

else if(keycode == 83)//S

{

Player1.setIsDown(true);

}

else if(keycode == 38)//UP

{

Player2.setIsUp(true);

}

else if(keycode == 40)//DOWN

{

Player2.setIsDown(true);

}

}

}

void Game::pmove()

{

if(!getIsPause())

{

if(Player1.getDelay() == 0)

{

Player1.move();

Player2.move();

Player1.setDelay(1);

}

else

{

Player1.setDelay(Player1.getDelay()-1);

}

}

}

void Game::bmove()

{

if(!getIsPause()&&!isCounting)

{

if(NowBall.getDelay() == 0)

{

//move ball

NowBall.move();

NowBall.setDelay(NowBall.getMaxDelay());

//determine score up

int LeftP = Boundary::getLpoint();

int RightP = Boundary::getRpoint();

RightBound.collision(NowBall);

LeftBound.collision(NowBall);

if(LeftP!=Boundary::getLpoint()||RightP!=Boundary::getRpoint())

{

Board::setCountDown(0);

changeCounting();

setGameStatus(4);

Beep(1046,2000/64);

}

}

else

{

NowBall.setDelay(NowBall.getDelay()-1);

}

if(Boundary::getLpoint() == 7 || Boundary::getRpoint() == 7)

{

if(Boundary::getLpoint()>Boundary::getRpoint())

{

ResetGame();

setGameStatus(1);

}

else if(Boundary::getLpoint()<Boundary::getRpoint())

{

ResetGame();

setGameStatus(2);

}

Beep(523,100);

Beep(659,100);

Beep(784,100);

Beep(1046,300);

}

}

}

void Game::collision()

{

if(!getIsPause()&&!isCounting)

{

if(NowBall.getDelay() == 0)

{

//polymorphism here

//determine ball direction

Player2.collision(NowBall);

Player1.collision(NowBall);

UpWall.collision(NowBall);

DownWall.collision(NowBall);

}

}

}

int Game::getLscore()const

{

return Boundary::getLpoint();

}

int Game::getRscore()const

{

return Boundary::getRpoint();

}

bool Game::getIsPause()const

{

return isPause;

}

void Game::changePause()

{

isPause = !isPause;

}

bool Game::getIsCounting()const

{

return isCounting;

}

void Game::changeCounting()

{

isCounting = !isCounting;

}

void Game::setGameStatus(int stat)

{

GameStatus = stat;

}

int Game::getGameStatus()const

{

return GameStatus;

}

const Ball& Game::getBallRef() const

{return NowBall;}

const Board& Game::getP1Ref() const

{return Player1;}

const Board& Game::getP2Ref() const

{return Player2;}

const Wall& Game::getWallURef() const

{return UpWall;}

const Wall& Game::getWallDRef() const

{return DownWall;}

const Boundary& Game::getLBRef() const

{return LeftBound;}

const Boundary& Game::getRBRef() const

{return RightBound;}

void Game::setCountdownPauseFrame(int frame)

{

CountdownPauseFrame = frame;

}

int Game::getCountdownPauseFrame()

{

return CountdownPauseFrame;

}

void Game::checkBallAndBoard()

{

if(NowBall.getxpos() == Player1.getX())

{

if(NowBall.getypos() == Player1.getYUpEnd())//if the ball is inside the board(up)

{

NowBall.adjustYPosition(Player1.getYUpEnd()-1);

}

if(NowBall.getypos() == Player1.getYDownEnd() )//if the ball is inside the board(down)

{

NowBall.adjustYPosition(Player1.getYDownEnd()+1);

}

}

if(NowBall.getxpos() == Player2.getX())

{

if(NowBall.getypos() == Player2.getYUpEnd())//if the ball is inside the board(up)

{

NowBall.adjustYPosition(Player2.getYUpEnd()-1);

}

if(NowBall.getypos() == Player2.getYDownEnd() )//if the ball is inside the board(down)

{

NowBall.adjustYPosition(Player2.getYDownEnd()+1);

}

}

}

/\*myKeyboard.h\*/

//出自遊戲程式設計概論EX5-2 鍵盤訊息的處理p.5-14 ~ p.5-26

//這是為了乓遊戲進行的最佳化版本

//預計將GetKeyboardState改成含有GetKeyState的GetMyKeyState

//承認鍵有W(0x57),S(0x53),上(VK\_UP),下(VK\_DOWN),ENTER(VK\_RETURN),ESCAPE(VK\_ESCAPE)共計六鍵

//其中W,S,上,下不能衝突且為連發鍵

//修正Keyboard.h帶來的錯誤

//本標頭檔及CPP參考： 遊戲程式設計概論呂建德著ISBN 986-125-455-2

// msdn.microsoft.com

#pragma once

#include <windows.h>

class clQueue

{

private:

unsigned int mCount;

unsigned int mBufSize;

int \*mBuffer;

unsigned int PushIndex, PopIndex;

CRITICAL\_SECTION CriticalSection;

private:

void Critical\_Enter(void)

{ EnterCriticalSection(&CriticalSection); }

void Critical\_Leave(void)

{ LeaveCriticalSection(&CriticalSection); }

public:

clQueue() : mCount(0), mBufSize(50), mBuffer(NULL), PushIndex(0), PopIndex(0)

{

InitializeCriticalSection( &CriticalSection );

mBuffer = new int[ mBufSize ];

}

~clQueue()

{

if( mBuffer != NULL )

{

delete [] mBuffer;

mBuffer = NULL;

}

DeleteCriticalSection( &CriticalSection );

}

bool Push ( int nNum )

{

Critical\_Enter();

if( mCount >= mBufSize )

{

Critical\_Leave();

return false;

}

mBuffer[ PushIndex ] = nNum;

PushIndex++;

if( PushIndex >= mBufSize )

{

PushIndex = 0;

}

mCount++;

Critical\_Leave();

return true;

}

bool Pop ( int &nNum )

{

Critical\_Enter();

if( mCount == 0 )

{

Critical\_Leave();

return false;

}

nNum = mBuffer[ PopIndex ];

PopIndex++;

if( PopIndex >= mBufSize )

{

PopIndex = 0;

}

mCount--;

Critical\_Leave();

return true;

}

void Clear ( void )

{

Critical\_Enter();

PushIndex = PopIndex;

mCount = 0;

Critical\_Leave();

}

};

/////////

//佇列類別不更動

/////////

/////////////////////////////////////////////////////////

//define key type:

//

//hotkey:VK\_UP, 0x57, VK\_DOWN, 0x53

//

//couplekey:dont cares

//

//generalkey:VK\_RETURN, VK\_ESCAPE

//

//note:only use in game, use method getch() in menu;

/////////////////////////////////////////////////////////

#define D\_HotKeyNumber 4

#define D\_CoupleKeyNumber 0

#define D\_KeyReadRate 64 //times per second

#define D\_KeyReadSpace (1000/D\_KeyReadRate) //span per read(ms)

#define D\_QuickKeyDelay 0//10\*D\_KeyReadSpace = 0.5s

//////////////////////////////////////////////////////////

//key read class

//

class clKeyBoard

{

private:

DWORD dwThreadId; //DWORD = double word

protected:

clQueue Keys;

private:

void StartKeyThread ( void );

public:

clKeyBoard()

{

StartKeyThread();

}

virtual ~clKeyBoard()

{

}

bool ReadKey ( unsigned int &KeyCode )

{

int nKeyCode;

if( Keys.Pop( nKeyCode ) )

{

KeyCode = nKeyCode;

return true;

}

return false;

}

bool WriteKey ( unsigned int KeyCode )

{

if ( Keys.Push( KeyCode ) )

{

return true;

}

return false;

}

void ClearBuffer( void )

{

Keys.Clear();

}

};

/\*myKeyboard.cpp\*/

#include "stdafx.h"

#include <windows.h>

#include <iostream>

#include "myKeyboard.h"

int HotKeys [ D\_HotKeyNumber ] =

{ VK\_UP, 0x57, VK\_DOWN, 0x53 };

BOOL GetMyKeyState(PBYTE lpKeyState)

{

const int all = 6;

int allKeys [ all ] =

{ VK\_UP, 0x57, VK\_DOWN, 0x53, VK\_ESCAPE, VK\_RETURN };

int loop;

for( loop = 0 ; loop < all ; loop++ )

{

lpKeyState[loop] = (GetKeyState(allKeys[loop])>>8);

}

return TRUE;

};

//key read function

DWORD WINAPI ThreadFunc( LPVOID lpParam )

{

const int Count = 6;

int loop;

BYTE KeyTable[Count];

BYTE BackupTable[Count];

clKeyBoard \*lpKB = (clKeyBoard \*)lpParam;

bool HotKey, HotKeyPress;

int HKdelay;

short Res;

if( lpKB == NULL ){ return 1; }

memset ( KeyTable, 0, Count );

memset ( BackupTable, 0, Count );

HotKeyPress = false;

HKdelay = 0;

HotKey = false;

while(true)

{

Res = GetKeyState( VK\_F1 );

if ( GetMyKeyState( PBYTE ( &KeyTable ) ) )

{

if((KeyTable[5]>>7) == 0 && (KeyTable[5]>>7) != (BackupTable[5]>>7) )

{

if( HotKey )

{

lpKB->ClearBuffer();

HotKey = false;

}

lpKB->WriteKey(VK\_RETURN);

}

if((KeyTable[4]>>7) == 0 && (KeyTable[4]>>7) != (BackupTable[4]>>7) )

{

if( HotKey )

{

lpKB->ClearBuffer();

HotKey = false;

}

lpKB->WriteKey(VK\_ESCAPE);

}

for( loop = 0 ; loop<D\_HotKeyNumber ; loop++ )

{

if((KeyTable[loop]>>7) != 0 && (KeyTable[loop]>>7) != (BackupTable[loop]>>7) && !HotKey )

{

HotKey = true;

}

if((KeyTable[loop]>>7) == 0 && (KeyTable[loop]>>7) != (BackupTable[loop]>>7) && HotKey)

{

lpKB->WriteKey( HotKeys[loop] );

HotKeyPress = false;

}

else if ( (KeyTable[loop]>>7) != 0 && !HotKeyPress && (KeyTable[loop]>>7) == (BackupTable[loop]>>7) && HotKey )

{

HotKeyPress = true;

HKdelay = D\_QuickKeyDelay;

}

}

if( HotKeyPress )

{

if( HKdelay > 0 )

{

--HKdelay;

}

if( HKdelay == 0 )

{

for( loop = 0 ; loop < D\_HotKeyNumber ; loop++ )

{

if( (KeyTable[loop]>>7) != 0 && HotKey)

{

lpKB->WriteKey(HotKeys[loop]);

}

}

}

}

memcpy ( BackupTable, KeyTable, Count );

}

Sleep( D\_KeyReadSpace );

}

return 0;

}

////////////////////////////////////////////////////////////////////////

void clKeyBoard::StartKeyThread( void )

{

HANDLE hThread;

hThread = CreateThread( NULL, 0, ThreadFunc, this, 0, &dwThreadId );

if(hThread = NULL)

{

std::cout << "CreateThread failed." << std::endl;

}

else

{

CloseHandle( hThread );

}

}

/\*vCanvas.h\*/

//參考http://benryves.com/tutorials/winconsole/ 及遊戲程式設計概論的"虛擬畫布"部分

#ifndef \_vCANVAS\_H\_

#define \_vCANVAS\_H\_

#include <windows.h>

#include "Ball.h"

#include "Board.h"

#include "Wall.h"

#include "Boundary.h"

#include "Game.h"

#define ConsoleBufferWidth 120//must can be divided by 4 and should be greater than 80

#define ConsoleBufferHeight 25//should be 25

#define LConsoleMidPoint (ConsoleBufferWidth/2)

#define RConsoleMidPoint (ConsoleBufferWidth/2+1)

extern HANDLE wHnd; // Handle to write to the console.

extern HANDLE rHnd; // Handle to read from the console.

class cl\_vCanvas

{

friend ostream& operator<< (ostream&, const cl\_vCanvas&);

private:

CHAR\_INFO VirtualCanvas[ConsoleBufferWidth\*ConsoleBufferHeight];

public:

cl\_vCanvas();

void setWindow( void ); //get handles ready here too;

void PrinttoConsole( void ) const;

void ClearCanvas( void );

void WriteCanvas(const Ball&, cl\_vCanvas&);

void WriteCanvas(const Board&, cl\_vCanvas&);

void WriteCanvas(const Wall&, cl\_vCanvas&);

void WriteCanvas(const Boundary&, cl\_vCanvas&);

void WriteCanvas(Game&, cl\_vCanvas&);

};

/\*

// Set up the handles for reading/writing:

wHnd = GetStdHandle(STD\_OUTPUT\_HANDLE);

rHnd = GetStdHandle(STD\_INPUT\_HANDLE);

SetConsoleTitle(TEXT("PONG"));

// Set up the required window size:

SMALL\_RECT windowSize = {0, 0, ConsoleBufferWidth-1, ConsoleBufferHeight-1};

// Change the console window size:

SetConsoleWindowInfo(wHnd, TRUE, &windowSize);

// Create a COORD to hold the buffer size:

COORD bufferSize = {ConsoleBufferWidth, ConsoleBufferHeight};

// Change the internal buffer size:

SetConsoleScreenBufferSize(wHnd, bufferSize);

getch();

\*/

#endif

/\*vCanvas.cpp\*/

#include "stdafx.h"

#include <windows.h>

#include "vCanvas.h"

#include "Ball.h"

#include "Board.h"

#include "Wall.h"

#include "Boundary.h"

#include <iostream>

using namespace std;

// CHAR\_INFO VirtualCanvas[ConsoleBufferWidth][ConsoleBufferHeight];

ostream& operator<< (ostream& Output, const cl\_vCanvas& Canvas)

{

Canvas.PrinttoConsole();

return Output;

}

cl\_vCanvas::cl\_vCanvas()

{

setWindow();

ClearCanvas();

}

void cl\_vCanvas::setWindow( void ) //get handles ready here too;

{

wHnd = GetStdHandle(STD\_OUTPUT\_HANDLE);

rHnd = GetStdHandle(STD\_INPUT\_HANDLE);

SetConsoleTitle(TEXT("PONG"));

// Set up the required window size:

SMALL\_RECT windowSize = {0, 0, ConsoleBufferWidth-1, ConsoleBufferHeight-1};

// Change the console window size:

SetConsoleWindowInfo(wHnd, TRUE, &windowSize);

// Create a COORD to hold the buffer size:

COORD bufferSize = {ConsoleBufferWidth, ConsoleBufferHeight};

// Change the internal buffer size:

SetConsoleScreenBufferSize(wHnd, bufferSize);

// Create a cursor info to hide cursor:

CONSOLE\_CURSOR\_INFO CursorInfo = {1,FALSE};

// Hide cursor

SetConsoleCursorInfo(wHnd, &CursorInfo);

}

void cl\_vCanvas::PrinttoConsole( void )const

{

COORD BufferSize = {ConsoleBufferWidth, ConsoleBufferHeight};

COORD StartPoint = {0,0};

SMALL\_RECT WriteArea = {0,0,ConsoleBufferWidth-1,ConsoleBufferHeight-1};

WriteConsoleOutputA(wHnd, VirtualCanvas, BufferSize, StartPoint, &WriteArea);

}

void cl\_vCanvas::ClearCanvas( void )

{

// Clear the VirtualCanvas buffer:

for (int i=0; i < ConsoleBufferWidth\*ConsoleBufferHeight; i++)

{

VirtualCanvas[i].Char.AsciiChar = ' ';

VirtualCanvas[i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

}

}

void cl\_vCanvas::WriteCanvas(const Ball& ball, cl\_vCanvas& Canvas)

{

Canvas.VirtualCanvas[ball.getxpos() + ConsoleBufferWidth \* ball.getypos()].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[(ball.getxpos() + ConsoleBufferWidth \* ball.getypos())+1].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

}

void cl\_vCanvas::WriteCanvas(const Board& board, cl\_vCanvas& Canvas)

{

if(board.getIsLPlayer())

{

for(int i = board.getYUpEnd() ; i <= board.getYDownEnd() ; i++)

{

Canvas.VirtualCanvas[board.getX()+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[board.getX()+ConsoleBufferWidth\*i+1].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

}

}

else

{

for(int i = board.getYUpEnd() ; i <= board.getYDownEnd() ; i++)

{

Canvas.VirtualCanvas[board.getX()+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[board.getX()+ConsoleBufferWidth\*i+1].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

}

}

}

void cl\_vCanvas::WriteCanvas(const Wall& wall, cl\_vCanvas& Canvas)

{

for(int i = 0 ; i <= ConsoleBufferWidth-1 ; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*wall.getYDownEnd()].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

}

}

void cl\_vCanvas::WriteCanvas(const Boundary& bound, cl\_vCanvas& Canvas)

{

switch(bound.getLpoint())

{

case 0:

for(int i = LConsoleMidPoint-9; i<=LConsoleMidPoint-4; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*8].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

for(int i = 4; i<=8; i++)

{

Canvas.VirtualCanvas[LConsoleMidPoint-5+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-4+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-9+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-8+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

case 1:

for(int i = LConsoleMidPoint-7; i<=LConsoleMidPoint-4; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

for(int i = 5; i<=8; i++)

{

Canvas.VirtualCanvas[LConsoleMidPoint-5+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-4+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

break;

case 2:

for(int i = LConsoleMidPoint-9; i<=LConsoleMidPoint-4; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*6].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*8].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

Canvas.VirtualCanvas[LConsoleMidPoint-5+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-4+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-9+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-8+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

break;

case 3:

for(int i = LConsoleMidPoint-9; i<=LConsoleMidPoint-4; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*6].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*8].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

Canvas.VirtualCanvas[LConsoleMidPoint-5+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-4+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-5+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-4+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

break;

case 4:

for(int i = LConsoleMidPoint-9; i<=LConsoleMidPoint-4; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*6].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

for(int i = 4; i<=6; i++)

{

Canvas.VirtualCanvas[LConsoleMidPoint-9+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-8+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

for(int i = 4; i<=8; i++)

{

Canvas.VirtualCanvas[LConsoleMidPoint-5+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-4+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

break;

case 5:

for(int i = LConsoleMidPoint-9; i<=LConsoleMidPoint-4; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*6].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*8].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

Canvas.VirtualCanvas[LConsoleMidPoint-9+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-8+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-5+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-4+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

break;

case 6:

for(int i = LConsoleMidPoint-9; i<=LConsoleMidPoint-4; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*6].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*8].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

for(int i = 4; i<=8; i++)

{

Canvas.VirtualCanvas[LConsoleMidPoint-9+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-8+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

Canvas.VirtualCanvas[LConsoleMidPoint-5+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[LConsoleMidPoint-4+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

break;

default:

break;

}

switch(bound.getRpoint())

{

case 0:

for(int i = RConsoleMidPoint+4; i<=RConsoleMidPoint+9; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*8].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

for(int i = 4; i<=8; i++)

{

Canvas.VirtualCanvas[RConsoleMidPoint+8+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+9+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+4+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+5+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

case 1:

for(int i = RConsoleMidPoint+6; i<=RConsoleMidPoint+9; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

for(int i = 5; i<=8; i++)

{

Canvas.VirtualCanvas[RConsoleMidPoint+8+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+9+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

break;

case 2:

for(int i = RConsoleMidPoint+4; i<=RConsoleMidPoint+9; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*6].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*8].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

Canvas.VirtualCanvas[RConsoleMidPoint+8+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+9+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+4+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+5+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

break;

case 3:

for(int i = RConsoleMidPoint+4; i<=RConsoleMidPoint+9; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*6].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*8].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

Canvas.VirtualCanvas[RConsoleMidPoint+8+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+9+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+8+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+9+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

break;

case 4:

for(int i = RConsoleMidPoint+4; i<=RConsoleMidPoint+9; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*6].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

for(int i = 4; i<=6; i++)

{

Canvas.VirtualCanvas[RConsoleMidPoint+4+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+5+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

for(int i = 4; i<=8; i++)

{

Canvas.VirtualCanvas[RConsoleMidPoint+8+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+9+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

break;

case 5:

for(int i = RConsoleMidPoint+4; i<=RConsoleMidPoint+9; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*6].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*8].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

Canvas.VirtualCanvas[RConsoleMidPoint+4+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+5+ConsoleBufferWidth\*5].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+8+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+9+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

break;

case 6:

for(int i = RConsoleMidPoint+4; i<=RConsoleMidPoint+9; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*4].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*6].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*8].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

for(int i = 4; i<=8; i++)

{

Canvas.VirtualCanvas[RConsoleMidPoint+4+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+5+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

Canvas.VirtualCanvas[RConsoleMidPoint+8+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+9+ConsoleBufferWidth\*7].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

break;

default:

break;

}

for(int i = +2; i<=ConsoleBufferHeight-1-2; i+=2)

{

Canvas.VirtualCanvas[LConsoleMidPoint+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

Canvas.VirtualCanvas[RConsoleMidPoint+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE;

}

}

void cl\_vCanvas::WriteCanvas(Game& game, cl\_vCanvas& Canvas)

{

char \*block;

char \*command;

char \*ptr;

switch(game.getGameStatus())

{

case 0:

break;

case 1:

if(!game.getIsPause())

game.changePause();

if(game.getIsCounting())

game.changeCounting();

block = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

command = "\* Player 1 Wins! \*";

ptr = block;

for(int i = LConsoleMidPoint-8; i <= RConsoleMidPoint+8; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*11].Char.AsciiChar = \*ptr;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*11].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*13].Char.AsciiChar = \*ptr;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*13].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

ptr++;

}

ptr = command;

for(int i = LConsoleMidPoint-8; i <= RConsoleMidPoint+8; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*12].Char.AsciiChar = \*ptr;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*12].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

ptr++;

}

break;

case 2:

if(!game.getIsPause())

game.changePause();

if(game.getIsCounting())

game.changeCounting();

block = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

command = "\* Player 2 Wins! \*";

ptr = block;

for(int i = LConsoleMidPoint-8; i <= RConsoleMidPoint+8; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*11].Char.AsciiChar = \*ptr;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*11].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*13].Char.AsciiChar = \*ptr;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*13].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

ptr++;

}

ptr = command;

for(int i = LConsoleMidPoint-8; i <= RConsoleMidPoint+8; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*12].Char.AsciiChar = \*ptr;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*12].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

ptr++;

}

break;

case 3:

block = "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";

command = "\* Game Paused. \*";

ptr = block;

for(int i = LConsoleMidPoint-7; i <= RConsoleMidPoint+7; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*11].Char.AsciiChar = \*ptr;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*11].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*13].Char.AsciiChar = \*ptr;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*13].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

ptr++;

}

ptr = command;

for(int i = LConsoleMidPoint-7; i <= RConsoleMidPoint+7; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*12].Char.AsciiChar = \*ptr;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*12].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

ptr++;

}

break;

case 4:

for(int i = 10; i<=14; i++)

{

Canvas.VirtualCanvas[LConsoleMidPoint-2+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[LConsoleMidPoint-1+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[LConsoleMidPoint+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[RConsoleMidPoint+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[RConsoleMidPoint+1+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[RConsoleMidPoint+2+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

}

for(int i = LConsoleMidPoint-2; i<=RConsoleMidPoint+2; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*10].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*12].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*14].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

}

Canvas.VirtualCanvas[RConsoleMidPoint+1+ConsoleBufferWidth\*11].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[RConsoleMidPoint+2+ConsoleBufferWidth\*11].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[RConsoleMidPoint+1+ConsoleBufferWidth\*13].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[RConsoleMidPoint+2+ConsoleBufferWidth\*13].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

if(game.getCountdownPauseFrame() == 0)

{

game.setCountdownPauseFrame(60);

game.setGameStatus(5);

Beep(784,2000/64);

}

else

{

game.setCountdownPauseFrame(game.getCountdownPauseFrame()-1);

}

break;

case 5:

for(int i = 10; i<=14; i++)

{

Canvas.VirtualCanvas[LConsoleMidPoint-2+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[LConsoleMidPoint-1+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[LConsoleMidPoint+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[RConsoleMidPoint+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[RConsoleMidPoint+1+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[RConsoleMidPoint+2+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

}

for(int i = LConsoleMidPoint-2; i<=RConsoleMidPoint+2; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*10].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*12].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*14].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

}

Canvas.VirtualCanvas[RConsoleMidPoint+2+ConsoleBufferWidth\*11].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[RConsoleMidPoint+1+ConsoleBufferWidth\*11].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[LConsoleMidPoint-2+ConsoleBufferWidth\*13].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[LConsoleMidPoint-1+ConsoleBufferWidth\*13].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

if(game.getCountdownPauseFrame() == 0)

{

game.setCountdownPauseFrame(60);

game.setGameStatus(6);

Beep(523,2000/64);

}

else

{

game.setCountdownPauseFrame(game.getCountdownPauseFrame()-1);

}

break;

case 6:

for(int i = 10; i<=14; i++)

{

Canvas.VirtualCanvas[LConsoleMidPoint-2+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[LConsoleMidPoint-1+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[LConsoleMidPoint+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[RConsoleMidPoint+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[RConsoleMidPoint+1+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

Canvas.VirtualCanvas[RConsoleMidPoint+2+ConsoleBufferWidth\*i].Attributes = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY | 0;

}

for(int i = LConsoleMidPoint-1; i<=RConsoleMidPoint; i++)

{

Canvas.VirtualCanvas[i+ConsoleBufferWidth\*10].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

}

for(int i = 11; i<=14; i++)

{

Canvas.VirtualCanvas[LConsoleMidPoint+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

Canvas.VirtualCanvas[RConsoleMidPoint+ConsoleBufferWidth\*i].Attributes = 0 | BACKGROUND\_RED | BACKGROUND\_GREEN | BACKGROUND\_BLUE | BACKGROUND\_INTENSITY;

}

if(game.getCountdownPauseFrame() == 0)

{

game.setCountdownPauseFrame(60);

game.changeCounting();

game.setGameStatus(0);

}

else

{

game.setCountdownPauseFrame(game.getCountdownPauseFrame()-1);

}

break;

default:

break;

}

}

/\*

// Set up the handles for reading/writing:

wHnd = GetStdHandle(STD\_OUTPUT\_HANDLE);

rHnd = GetStdHandle(STD\_INPUT\_HANDLE);

SetConsoleTitle(TEXT("PONG"));

// Set up the required window size:

SMALL\_RECT windowSize = {0, 0, ConsoleBufferWidth-1, ConsoleBufferHeight-1};

// Change the console window size:

SetConsoleWindowInfo(wHnd, TRUE, &windowSize);

// Create a COORD to hold the buffer size:

COORD bufferSize = {ConsoleBufferWidth, ConsoleBufferHeight};

// Change the internal buffer size:

SetConsoleScreenBufferSize(wHnd, bufferSize);

getch();

\*/

/\*Wall.h\*/

#ifndef \_WALL\_H\_

#define \_WALL\_H\_

#include "Board.h"

#include "Ball.h"

class Wall : public Board

{

public:

Wall(int = 0, int = 0);

virtual void collision(Ball &);

virtual void setPosition(bool);

};

#endif

/\* Wall.cpp \*/

#include "stdafx.h"

#include "Wall.h"

#include "Board.h"

#include "Ball.h"

#define dUpWallY 1

#define dDownWallY 23

Wall::Wall(int x, int y)

:Board(x, y, y, false, false, false)

{

}

void Wall::collision(Ball &ball)

{

if(ball.getypos()-Board::getYDownEnd()==1||ball.getypos()-Board::getYDownEnd()==(-1))

{

ball.setDirection(false,true);

}

}

void Wall::setPosition(bool isUp)

{

Board::Y\_UpperEnd = (isUp)? dUpWallY:dDownWallY;

Board::Y\_LowerEnd = (isUp)? dUpWallY:dDownWallY;

}

參考的技術：

環狀佇列與按鍵輸入 —遊戲程式設計概論(ISBN:986-125-455-2)

虛擬畫布概念 —遊戲程式設計概論

Winconsole相關 —<http://benryves.com/tutorials/winconsole/>

Windows.h函式理解 —<http://msdn.microsoft.com/en-us/library/windows/desktop/ms682073(v=vs.85).aspx>

心得：

A1005504 戴佳燊

這次專題讓我學習到了windows.h的一些函式，如getkeystate跟printconsoleoutput等等，專案開發過程我也感受到封裝的好處與getset函式的重要性。由於exception handling跟讀寫檔不太熟所以沒有在專案中使用。希望下次能夠做得規模更大點。