**СОДЕРЖАНИЕ**

1 ПОСТАНОВКА ЗАДАЧИ…………………………………………… 5

1.1 Выбор логических структур для хранения данных………….. 5

2 ВЫБОР СРЕДЫ РАЗРАБОТКИ…………………………………….. 6

3 СТРУКТУРА ВХОДНЫХ И ВЫХОДНЫХ ДАННЫХ……………. 8

4 ДИАГРАММА КЛАССОВ…………………………………………... 95 ОПИСАНИЕ КЛАССОВ…………………………………………….. 11

6 БЛОК-СХЕМА АЛГОРИТМА……………………………………… 15 6.1 Метод bool MoveFigure( Cell& nextCell )…………………….. 15 6.2 Функция int EnterMsgLoop( bool ( \*ptr\_display )…………….. 16

7 АЛГОРИТМЫ ПО ШАГАМ………………………………………... 17

7.1 Метод void Draw( )…………………………………………….. 17

7.2 Метод bool CheckMove(Cell& currentCell, Cell& nextCell)…. 18

8 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ……………………………. 20

ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ……………………... 25

**ВВЕДЕНИЕ**

C# — элегантный, типобезопасный объектно-ориентированный язык, предназначенный для разработки разнообразных безопасных и мощных приложений, выполняемых в среде .NET Framework. С помощью языка C# можно создавать обычные приложения Windows, XML-веб-службы, распределенные компоненты, приложения "клиент-сервер", приложения баз данных и т. д. Visual C# предоставляет развитый редактор кода, конструкторы с удобным пользовательским интерфейсом, встроенный отладчик и множество других средств, упрощающих разработку приложений на базе языка C# и .NET Framework.

Синтаксис C# очень выразителен, но прост в изучении. Все, кто знаком с языками C, C++ или Java с легкостью узнают синтаксис с фигурными скобками, характерный для языка C#. Разработчики, знающие любой из этих языков, как правило, смогут добиться эффективной работы с языком C# за очень короткое время. Синтаксис C# делает проще то, что было сложно в C++, и обеспечивает мощные возможности, такие как типы значений Nullable, перечисления, делегаты, лямбда-выражения и прямой доступ к памяти, чего нет в Java.

C# поддерживает универсальные методы и типы, обеспечивая более высокий уровень безопасности и производительности, а также итераторы, позволяющие при реализации коллекций классов определять собственное поведение итерации, которое может легко использоваться в клиентском коде. Выражения LINQ делают строго типизированный запрос очень удобной языковой конструкцией.

Как объектно-ориентированный язык, C# поддерживает понятия инкапсуляции, наследования и полиморфизма. Все переменные и методы, включая метод Main – точку входа приложения – инкапсулируются в определения классов. Класс может наследовать непосредственно из одного родительного класса, но может реализовывать любое число интерфейсов. Для методов, которые переопределяют виртуальные методы в родительском классе, необходимо ключевое слово override, чтобы исключить случайное повторное определение. В языке C# структура похожа на облегченный класс: это тип, распределяемый в стеке, реализующий интерфейсы, но не поддерживающий наследование.

Программа на языке C# выполняется в среде .NET Framework – интегрированном компоненте Windows, содержащем виртуальную систему выполнения (среда CLR) и унифицированный набор библиотек классов.

**1 ПОСТАНОВКА ЗАДАЧИ**

Разработать классический вариант архиватора файлов для операционных систем семейства Windows, используя собственный алгоритм сжатия и разжатия.

К реализации предъявляются следующие требования:

1. Удобный интерфейс.
2. Возможность из программы выбирать файл для работы.
3. Процент сжатия не менее 15%
   1. **Обзор методов и алгоритмов решения поставленной задачи**

Для реализации приложения используется алгоритм Хаффмана – адаптивный жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. Был разработан в 1952 году аспирантом Массачусетского технологического института Дэвидом Хаффманом при написании им курсовой работы. В настоящее время используется во многих программах сжатия данных.

Идея алгоритма состоит в следующем: зная вероятности символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана.

Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.

Выбираются два свободных узла дерева с наименьшими весами.

Создается их родитель с весом, равным их суммарному весу.

Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.

Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0.

Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

Допустим, у нас есть следующая таблица частот:

15 7 6 6 5

А Б В Г Д

Этот процесс можно представить как построение дерева, корень которого — символ с суммой вероятностей объединенных символов, получившийся при объединении символов из последнего шага, его n0 потомков — символы из предыдущего шага и т. д.

Чтобы определить код для каждого из символов, входящих в сообщение, мы должны пройти путь от листа дерева, соответствующего текущему символу, до его корня, накапливая биты при перемещении по ветвям дерева (первая ветвь в пути соответствует младшему биту). Полученная таким образом последовательность битов является кодом данного символа, записанным в обратном порядке.

Для данной таблицы символов коды Хаффмана будут выглядеть следующим образом:

А Б В Г Д

0 100 101 110 111

Поскольку ни один из полученных кодов не является префиксом другого, они могут быть однозначно декодированы при чтении их из потока. Кроме того, наиболее частый символ сообщения А закодирован наименьшим количеством бит, а наиболее редкий символ Д — наибольшим.

Классический алгоритм Хаффмана имеет ряд существенных недостатков. Во-первых, для восстановления содержимого сжатого сообщения декодер должен знать таблицу частот, которой пользовался кодер. Следовательно, длина сжатого сообщения увеличивается на длину таблицы частот, которая должна посылаться впереди данных, что может свести на нет все усилия по сжатию сообщения. Кроме того, необходимость наличия полной частотной статистики перед началом собственно кодирования требует двух проходов по сообщению: одного для построения модели сообщения (таблицы частот и Н-дерева), другого для собственно кодирования. Во-вторых, избыточность кодирования обращается в ноль лишь в тех случаях, когда вероятности кодируемых символов являются обратными степенями числа 2. В-третьих, для источника с энтропией, не превышающей 1 (например, для двоичного источника), непосредственное применение кода Хаффмана бессмысленно.

* 1. **Обоснование выбранных методов и алгоритмов**

С тех пор, как Д. А. Хаффман опубликовал в 1952 году свою работу "Метод построения кодов с минимальной избыточностью", его алгоритм кодирования стал базой для огромного количества дальнейших исследований в этой области.

По сей день в компьютерных журналах можно найти большое количество публикаций, посвященных как различным реализациям алгоритма Хаффмана, так и поискам его лучшего применения. Кодирование Хаффмана используется в коммерческих программах сжатия (например в PKZIP и LHA), встроено в некоторые телефаксы и даже используется в алгоритме JPEG сжатия графических изображений с потерями.

Алгоритм обладает рядом преимуществ перед другими:

1. Прост – легко запоминается и сравнительно легко программируется.
2. Подходит для всех схем сжатия(статической, полуадаптивной и адаптивной)
3. Быстро работает
4. Генерирует таблицу кодов элементов

**2 ВЫБОР СРЕДЫ РАЗРАБОТКИ**

Для реализации приложения был выбран язык разработки – C#.

Сами разработчики языка описывают его, как простой, современный, объектно-ориентированный и безопасный язык программирования. Синтаксически C# напоминает C++ и Java, что позволило программистам за достаточно короткое время изучить тонкости нового языка.

Несмотря на то, что C# и .NET предназначены в первую очередь для веб-разработки, их также активно применяют для создания приложений, которые должны устанавливаться на машине конечного пользователя, где и будет выполняться вся обработка данных. Разработку таких приложений обеспечивает библиотека Windows Forms, позволяющая проектировать графический интерфейс.

Приложение, описанное в данной работе, разработано именно с помощью библиотеки Windows Forms.

Язык был выбран по следующим причинам:

1. Претендует на подлинную объектную ориентированность.
2. Призван реализовать компонентно-ориентированный подход к программированию, который способствует меньшей машинно-архитектурной зависимости результирующего программного кода, большей гибкости, переносимости и легкости повторного использования программ.
3. Отличием от предшественников является изначальная ориентация на безопасность кода.
4. Расширенная поддержка событийно-ориентированного программирования.
5. Язык является «родным» для создания приложений в среде Microsoft .NET, поскольку наиболее тесно и эффективно интегрирован с ней.

**3 СТРУКТУРА ВХОДНЫХ**  **И ВЫХОДНЫХ ДАННЫХ**

Входные данные:

Camera – экземпляр класса Camera, позволяет управлять камерой в игровом мире;

Board – экземпляр класса Chessboard, представляет игровое поле с расположенными на нем фигурами;

g\_pDevice – указатель на интерфейс IDirect3DDevice9, предоставляет возможности работы с графикой;

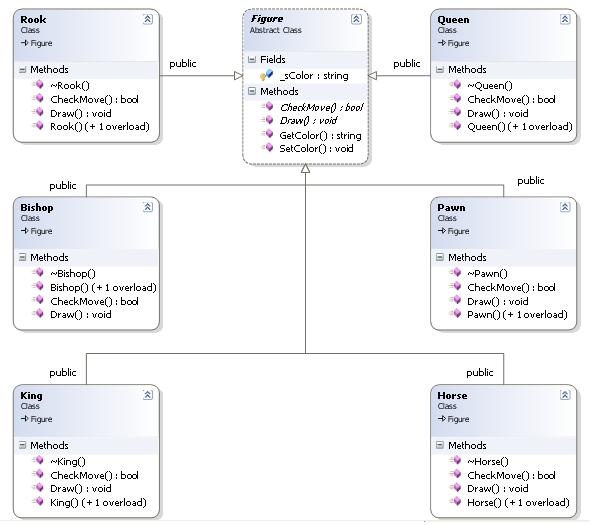
g\_Matrix – экземпляр структуры D3DXMATRIX, является матрицей преобразования;

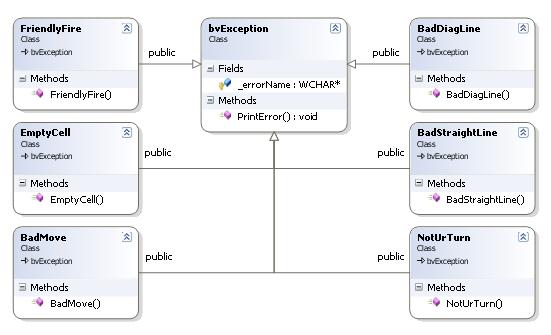
Выходные данные:

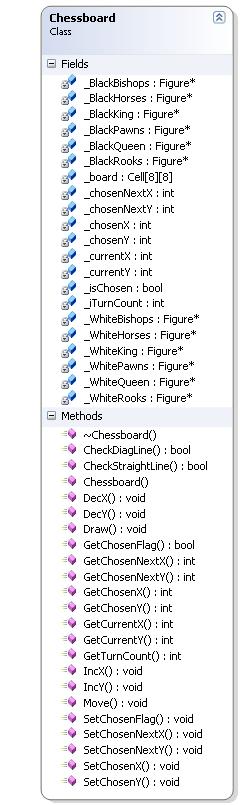
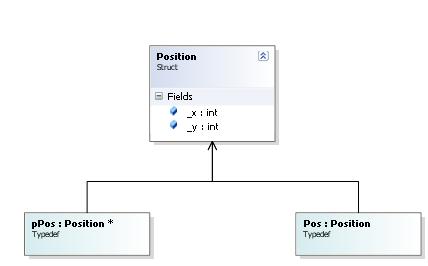
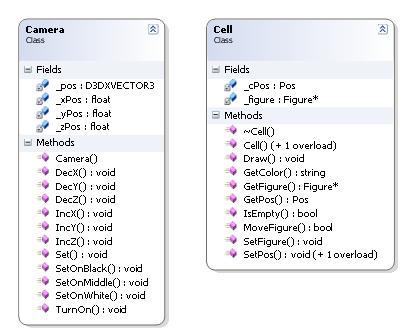
hWnd – дескриптор окна с прорисованным игровым миром;

hResult – переменная типа HRESULT, используется как статус состояния игры.

**4 ДИАГРАММА КЛАССОВ**

****

****

**** ****

**5 ОПИСАНИЕ КЛАССОВ**

|  |  |
| --- | --- |
| Абстрактный класс для описания фигуры, в котором определяются ее цвет, метод его установки и получения | Figure |
| Экземпляр класса является фигурой – король | King |
| Экземпляр класса является фигурой – королева | Queen |
| Экземпляр класса является фигурой – пешка | Pawn |
| Экземпляр класса является фигурой – слон | Bishop |
| Экземпляр класса является фигурой – ладья | Rook |
| Экземпляр класса является фигурой – конь | Horse |
| Структура для хранения двумерных координат | Position |
| Класс исключений | bvException |
| Класс исключения: ход на дружественную фигуру | FriendlyFire |
| Класс исключения: выбора пустой клетки | EmptyCell |
| Класс исключения: неверного хода | BadMove |
| Класс исключения: наличие фигур на диагональной линии | BadDiagLine |
| Класс исключения: наличие фигур на прямой линии | BadStraightLine |
| Класс исключения: нарушение очередности хода | NotUrTurn |
| Класс служит для обзора игрового мира | Camera |
| Класс, представляющий собой структурную ячейку класса Chessboard | Cell |
| Класс служит для представления шахматной доски | Chessboard |

Figure – абстрактный класс для описания фигуры, в определяются ее цвет, метод его установки и получения:

class Figure

{

protected:

std::string \_sColor; // figure's color

public:

std::string GetColor( ); // Method for receiving figure's color

void SetColor( std::string sColor ); // Method for set figure's color

//

// abstract method for check if move is correct

//

virtual bool CheckMove( Cell& currentCell, Cell& nextCell ) = 0;

//

// abstract method for draw figure

//

virtual void Draw( int material ) = 0;

};

Cell – класс, представляющий собой структурную ячейку класса Chessboard

class Cell

{

Figure\* \_figure; // pointer to figure object

Pos \_cPos; // cell's position

public:

Cell( int x, int y, Figure\* figure ); // constructor with parameters

Cell( ); // default constructor

~Cell( ); // destructor

Pos GetPos( ); // method for receiving cell's position

Figure\* GetFigure( ); // method for receiving figure's pointer

void SetPos( Pos& cPos ); // method for set cell's position ( Pos& )

void SetPos( int x, int y ); // method for set cell's position ( int, int )

void SetFigure( Figure\* figure ); // method for set figure

bool MoveFigure( Cell& nextCell ); // method for move figure to next cell

bool IsEmpty( ); // method for check is cell is empty

std::string GetColor( ); // method for receiving figure's color

void Draw( int material ); // method for draw cell

};

Chessboard – класс служит для представления шахматной доски

class Chessboard

{

Figure\* \_WhitePawns;

Figure\* \_BlackPawns;

Figure\* \_WhiteKing;

Figure\* \_BlackKing;

Figure\* \_WhiteHorses;

Figure\* \_BlackHorses; // Figure pointers

Figure\* \_WhiteRooks;

Figure\* \_BlackRooks;

Figure\* \_WhiteBishops;

Figure\* \_BlackBishops;

Figure\* \_WhiteQueen;

Figure\* \_BlackQueen;

Cell \_board[MAX\_CELL\_NUMBER][MAX\_CELL\_NUMBER]; // Matrix of cell object

int \_iTurnCount; // int variable to choose player's turn

int \_currentX; // int variable that mark on current cell

int \_currentY; // int variable that mark on current cell

int \_chosenX;

int \_chosenY;

int \_chosenNextX; // variables for making player's move

int \_chosenNextY;

bool \_isChosen;

public:

Chessboard( ); // constructor

~Chessboard( ); // destructor

void Move( int x, int y, int nx, int ny ); // method for move figure

// method for draw chessboard

void Draw( D3DXMATRIX g\_Matrix, IDirect3DDevice9\* g\_pDevice );

void IncX( ); // method for increase current X coordinate

void IncY( ); // method for increase current Y coordinate

void DecX( ); // method for decrease current X coordinate

void DecY( ); // method for decrease current Y coordinate

int GetCurrentX( ); // method for receiving current X coordinate

int GetCurrentY( ); // method for receiving current Y coordinate

void SetChosenX( int x ); // method for set chosen X coordinate

void SetChosenY( int y ); // method for set chosen Y coordinate

void SetChosenNextX( int nx ); // method for set next chosen X coordinate

void SetChosenNextY( int ny ); // method for set next chosen Y coordinate

void SetChosenFlag( bool flag ); // method for set flag of chosen cell

bool GetChosenFlag( ); // method for receiving flag of chosen cell

int GetChosenX( ); // method for receiving chosen X coordinate

int GetChosenY( ); // method for receiving chosen Y coordinate

int GetChosenNextX( ); // method for receiving next chosen X coordinate

int GetChosenNextY( ); // method for receiving next chosen Y coordinate

int GetTurnCount( ); // method for receiving turn count

//

// method for check is striaght line is empty

//

bool CheckStraightLine( Cell& currentCell, Cell& nextCell );

//

// method for check is diagonal line is empty

//

bool CheckDiagLine( Cell& currentCell, Cell& nextCell );

};

**6 БЛОК-СХЕМА АЛГОРИТМА**

**6.1 Метод bool MoveFigure( Cell& nextCell ) класса Cell –** метод перемещает фигуру с текущей клетку на nextCell клетку



**6.2 Функция int EnterMsgLoop( bool ( \*ptr\_display ) ( float timeDelta ) ) –** функция отображения игрового мира



**7 АЛГОРИТМЫ ПО ШАГАМ**

**7.1 Метод void Draw(D3DXMATRIX g\_Matrix, IDirect3DDevice9\* g\_pDevice) класса Chessboard –** метод визуализации шахматной доски:

1. Начало.
2. Входные данные:

\_board – массив клеток;

\_currentX – Х-координата клетки, на которую установлен курсор;

\_currentY – Y-координата клетки, на которую установлен курсор;

Выходные данные:

\_board – массив клеток;

1. Вызов метода очистки экрана Clear() из интерфейса IDirect3DDevice9.
2. Вызов метода начала сцены BeginScene() из интерфейса IDirect3DDevice9.
3. Цикл с параметром y по вертикальному ряду шахматной доски.
   1. Цикл с параметром x по горизонтальному ряду шахматной доски.
      1. Вызов функции D3DXMatrixTranslation() для создания матрицы преобразования в g\_Matrix.
      2. Вызов метода SetTransform() из интерфейса IDirect3DDevice9 для установки мировой матрицы с параметрами g\_Matrix.
      3. Если \_board[x][y] не содержит фигуры, то
         1. Если x = \_currentX и y = \_currentY, то
            1. Вызываем метод Draw() для клетки, устанавливаем зеленый цвет.
         2. Иначе если x – четное число, то
            1. Если y – четное число, то

Вызываем метод Draw() для клетки, устанавливаем белый цвет.

* + - * 1. Иначе

Вызываем метод Draw() для клетки, устанавливаем черный цвет.

* + - 1. Иначе
         1. Если y – четное число, то

Вызываем метод Draw() для клетки, устанавливаем черный цвет.

* + - * 1. Иначе

Вызываем метод Draw() для клетки, устанавливаем белый цвет.

* + 1. Иначе
       1. Если x = \_currentX и y = \_currentY, то
          1. Вызываем метод Draw для фигуры на клетке, устанавливаем зеленый цвет.
       2. Иначе если фигура на клетке белая, то
          1. Вызываем метод Draw для фигуры на клетке, устанавливаем желтый цвет.
       3. Иначе если фигура на клетке черная, то
          1. Вызываем метод Draw для фигуры на клетке, устанавливаем синий цвет.
  1. Конец цикла с параметром x.

1. Конец цикла с параметром у.
2. Вызов метода конца сцены EndScene() из интерфейса IDirect3DDevice9.
3. Вызов метода отображения вторичного буфера Present() из интерфейса IDirect3DDevice9.
4. Конец.

**7.2 Метод bool CheckMove(Cell& currentCell, Cell& nextCell) класса Rook –** проверка правильности хода ладьи:

1. Начало.
2. Входные данные:

currentCell – ссылка на текущую клетку;

nextCell – ссылка на следующую клетку;

Промежуточные данные:

nextPos – координаты клетки nextCell;

currentPos – координаты клетки currentCell;

Выходные данные:

currentCell – ссылка на текущую клетку;

nextCell – ссылка на следующую клетку;

1. Если nextPos.\_x = currentPos.\_x и nextPos.\_y = currentPos.\_y, то
   1. Генерация исключения BadMove().
2. Если nextPos.\_x != currentPos.\_x и nextPos.\_y != currentPos.\_y, то
   1. Генерация исключения BadMove().
3. Если следующая клетка содержит фигуру, то
   1. Если цвет фигуры на текущей клетке совпадает с цветом фигуры на следующей клетке, то
      1. Генерация исключения FriendlyFire().
4. Возврат истинного значения.
5. Конец.

**8 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

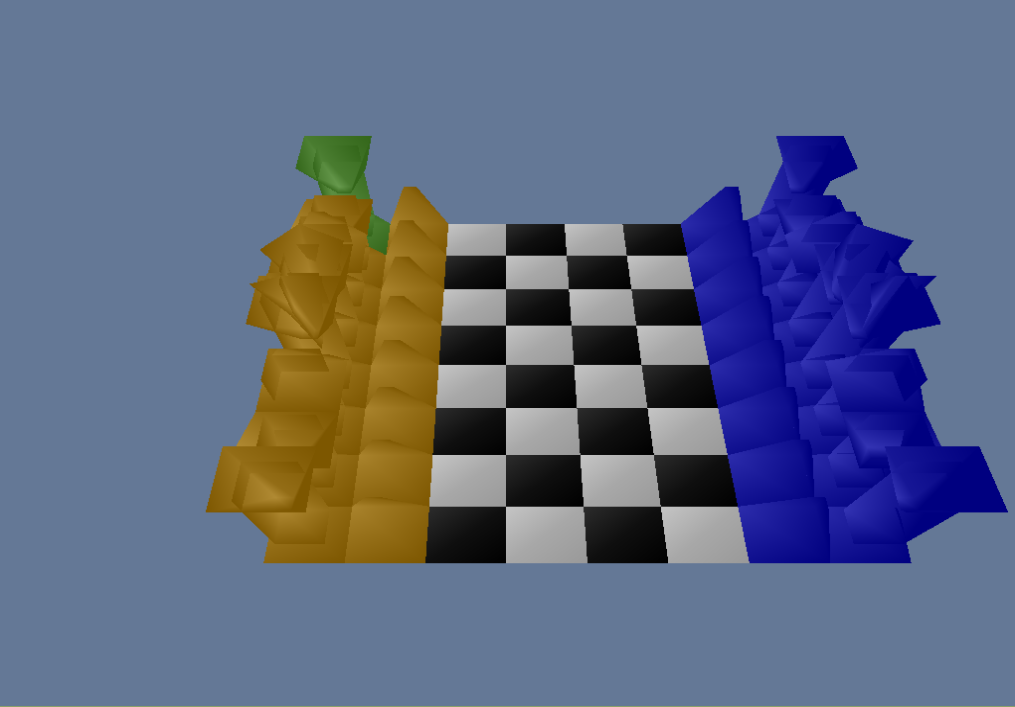
****

Рисунок 8.1 – Начало игры

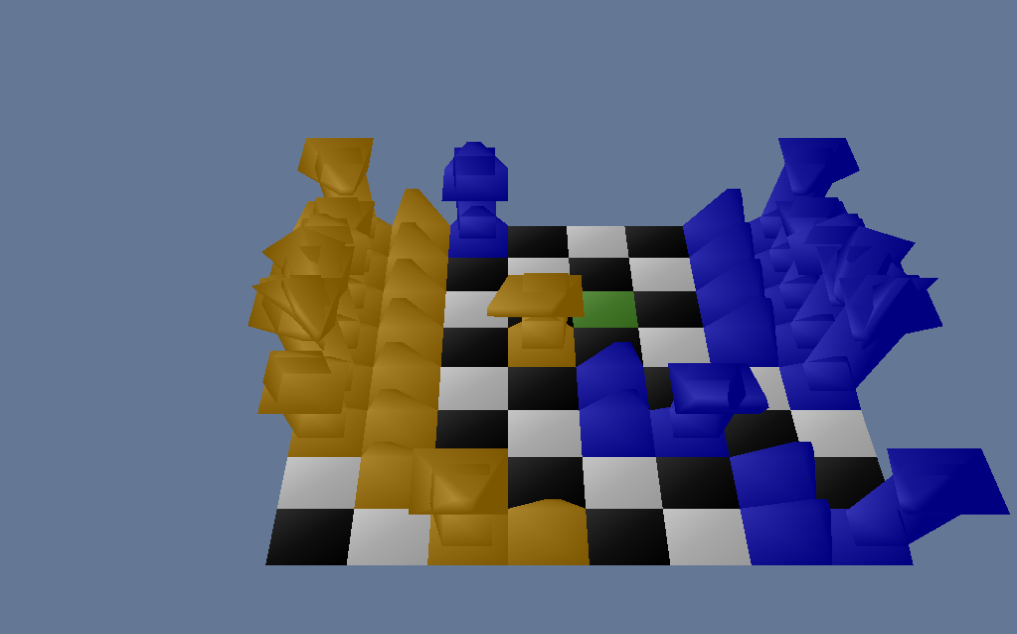


Рисунок 8.2 – Игровое поле после нескольких шагов

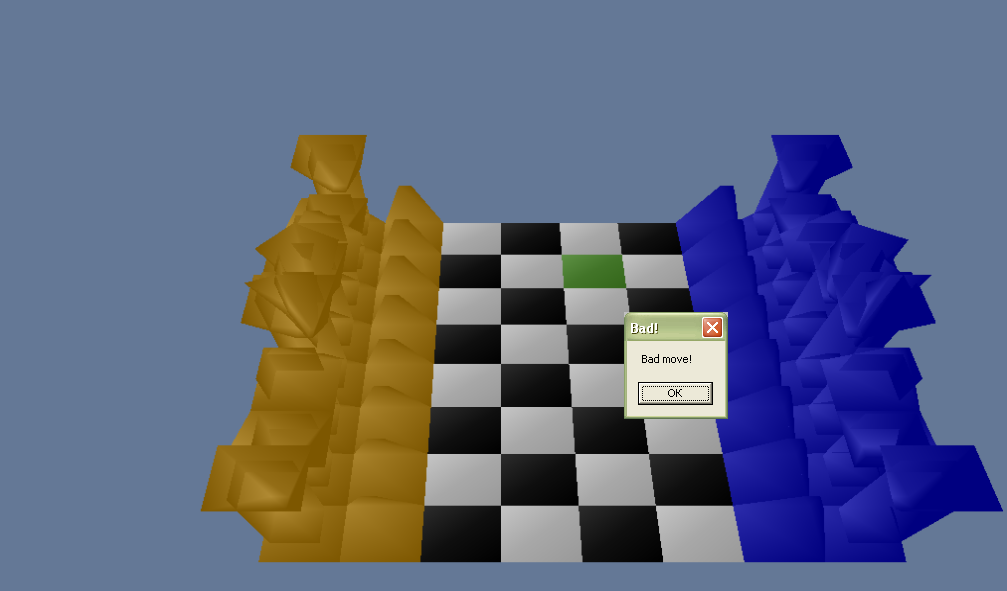


Рисунок 8.3 – Попытка походить пешкой на 3 клетки привела к ошибке

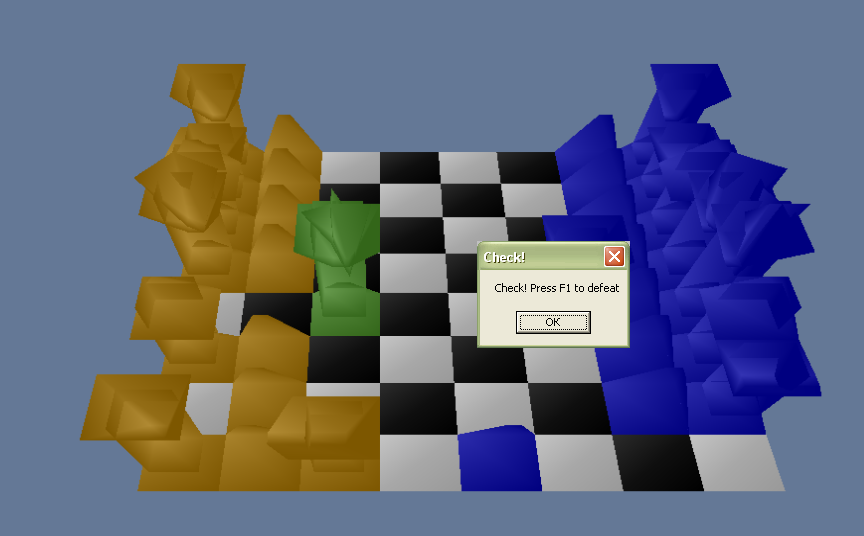


Рисунок 8.4 – Ход слоном на короля приводит к шаху

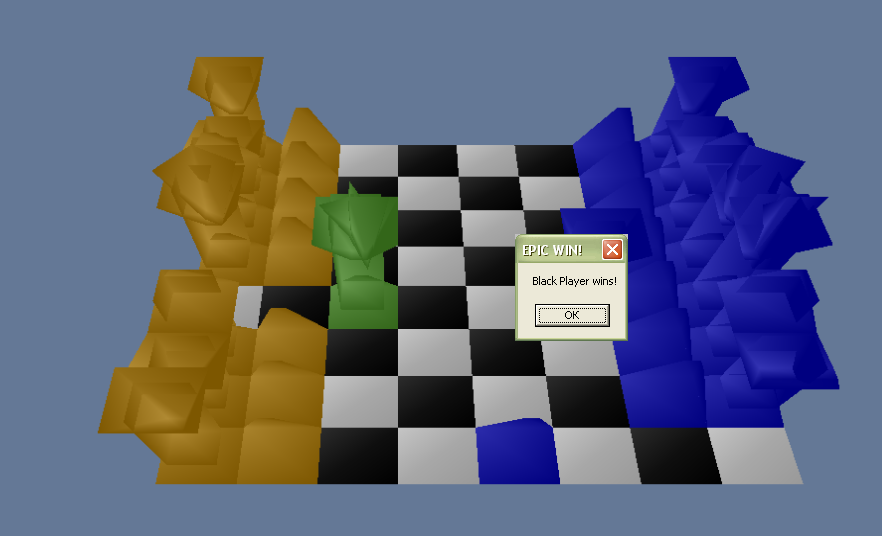


Рисунок 8.5 – Игрок 2 победил

**ЗАКЛЮЧЕНИЕ**

В рамках данной работы был разработан компьютерный вариант классической игры «Шахматы». Учитывая свою специфику, она будет интересна всем возрастным группам. Возможность одновременного участия двух живых соперников должна разнообразить игровой процесс.

В будущем этот проект можно усовершенствовать, добавив искусственный интеллект (планируется использование ChessPro v. 4.3), трехмерные модели, созданные в графическом редакторе 3D Studio MAX, а также мультиплатформенность, включив в проект отдельный модуль, созданный с помощью библиотеки OpenGL.

**Ограничения, накладываемые на программу**

Для корректной работы программы на компьютере должен быть установлен DirectX 9.0с и выше.

**Поддерживаемые операционные системы:**

* Microsoft Windows XP, Vista, 7

**СПИСОК ЛИТЕРАТУРЫ**

1. Luna, Frank D. «Introduction to 3D game programming with DirectX 9.0» – Wordware Publishing Inc., 2003. – 421 p. ISBN 1-55622-913-5
2. Gornakov S. «DirectX 9: Programming with C++» – Wordware Pulishing Inc., 2005. – 400 p. ISBN 5-94157-482-7
3. Макконел С. «Совершенный код. Мастер-класс». СПб.: Питер, 2005 – 896 стр.; ил. ISBN 5-7502-0064-7
4. Шилдт Г. «Самоучитель С++» СПб.:БХВ-Петербург, 2003. – 688 с. ISBN 5-7791-0086-1
5. Мейер С. «Эффективное использование С++. 50 рекомендаций по улучшению ваших программ и проектов» СПб.:Питер, 2006. – 240 с. ISBN 5-469-01213-1

**ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ**

#ifndef \_FIGURE\_H\_

#define \_FIGURE\_H\_

#include <string>

class Cell;

class Figure

{

protected:

//

// figure's color

//

std::string \_sColor;

public:

//

// Method for receiving figure's color

//

std::string GetColor( )

{

return \_sColor;

}

//

// Method for set figure's color

//

void SetColor( std::string sColor )

{

\_sColor = sColor;

}

//

// abstract method for check if move is correct

//

virtual bool CheckMove( Cell& currentCell, Cell& nextCell ) = 0;

//

// abstract method for draw figure

//

virtual void Draw( int material ) = 0;

};

#endif

#ifndef \_CELL\_H\_

#define \_CELL\_H\_

#include "figure.h"

#include "globals.h"

#include <iostream>

class Cell

{

//

// pointer to figure object

//

Figure\* \_figure;

//

// cell's position

//

Pos \_cPos;

public:

//

// constructor with parameters

//

Cell( int x, int y, Figure\* figure );

//

// default constructor

//

Cell( );

//

// destructor

//

~Cell( );

//

// method for receiving cell's position

//

Pos GetPos( );

//

// method for receiving figure's pointer

//

Figure\* GetFigure( );

//

// method for set cell's position ( Pos& )

//

void SetPos( Pos& cPos );

//

// method for set cell's position ( int, int )

//

void SetPos( int x, int y );

//

// method for set figure

//

void SetFigure( Figure\* figure );

//

// method for move figure to next cell

//

bool MoveFigure( Cell& nextCell );

//

// method for check is cell is empty

//

bool IsEmpty( );

//

// method for receiving figure's color

//

std::string GetColor( );

//

// method for draw cell

//

void Draw( int material );

};

#endif

#include <iostream>

#include <typeinfo.h>

#include "cell.h"

//

// Default ctor

//

Cell::Cell( )

{

\_cPos.\_x = 0;

\_cPos.\_y = 0;

\_figure = NULL;

}

//

// Ctor with parameters

//

Cell::Cell( int x, int y, Figure \*figure )

{

\_cPos.\_x = x;

\_cPos.\_y = y;

\_figure = figure;

}

//

// Dtor

//

Cell::~Cell( )

{

\_figure = NULL;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: GetPos

// Method task: Get Pos struct from Cell

// Access: public

// Returns: Pos

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Pos Cell::GetPos( )

{

return \_cPos;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: GetFigure

// Method task: Get Figure ptr from cell

// Access: public

// Returns: Figure\*

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Figure\* Cell::GetFigure( )

{

return \_figure;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: SetPos

// Method task: Set Cell's Pos struct

// Access: public

// Returns: void

// Parameters: Pos&

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Cell::SetPos( Pos& cPos )

{

\_cPos = cPos;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: SetPos

// Method task: Set Cell's Pos struct

// Access: public

// Returns: void

// Parameters: int, int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Cell::SetPos(int x, int y)

{

\_cPos.\_x = x;

\_cPos.\_y = y;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: SetFigure

// Method task: Set Figure on Cell

// Access: public

// Returns: void

// Parameters: Figure\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Cell::SetFigure( Figure\* figure )

{

\_figure = figure;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: MoveFigure

// Method task: Move Figure from current cell to next cell

// Access: public

// Returns: bool

// Parameters: Cell&

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Cell::MoveFigure( Cell& nextCell )

{

if ( \_figure != NULL && \_figure->CheckMove( \*this, nextCell ) ) {

if ( !nextCell.IsEmpty( ) ) {

if ( strcmp( "class King", typeid( \*nextCell.\_figure ).name( ) ) == 0 ) {

MessageBox(0, L"Check! Press F1 to defeat", L"Check!", 0);

return true;

}

}

nextCell.SetFigure( \_figure );

\_figure = NULL;

return true;

}

else

{

::MessageBox( 0, L"Wrong cell!", 0, 0 );

return false;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: IsEmpty

// Method task: Check is cell is empty

// Access: public

// Returns: bool

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Cell::IsEmpty()

{

if ( \_figure == NULL) {

return true;

}

return false;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: GetColor

// Method task: Get figure's color from current cell

// Access: public

// Returns: std::string

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

std::string Cell::GetColor( )

{

if ( \_figure ) {

return \_figure->GetColor( );

}

}

#ifndef \_CHESSBOARD\_H\_

#define \_CHESSBOARD\_H\_

#include "globals.h"

#include "figure.h"

#include "cell.h"

#include "pawn.h"

#include "horse.h"

#include "rook.h"

#include "bishop.h"

#include "king.h"

#include "queen.h"

class Chessboard

{

//

// Figure pointers

//

Figure\* \_WhitePawns;

Figure\* \_BlackPawns;

Figure\* \_WhiteKing;

Figure\* \_BlackKing;

Figure\* \_WhiteHorses;

Figure\* \_BlackHorses;

Figure\* \_WhiteRooks;

Figure\* \_BlackRooks;

Figure\* \_WhiteBishops;

Figure\* \_BlackBishops;

Figure\* \_WhiteQueen;

Figure\* \_BlackQueen;

//

// Matrix of cell object

//

Cell \_board[MAX\_CELL\_NUMBER][MAX\_CELL\_NUMBER];

//

// int variable to choose player's turn

//

int \_iTurnCount;

//

// int variables that mark on current cell

//

int \_currentX;

int \_currentY;

//

// variables for making player's move

//

int \_chosenX;

int \_chosenY;

int \_chosenNextX;

int \_chosenNextY;

bool \_isChosen;

public:

//

// constructor

//

Chessboard( );

//

// destructor

//

~Chessboard( );

//

// method for move figure

//

void Move( int x, int y, int nx, int ny );

//

// method for draw chessboard

//

void Draw( D3DXMATRIX g\_Matrix, IDirect3DDevice9\* g\_pDevice );

//

// method for increase current X coordinate

//

void IncX( );

//

// method for increase current Y coordinate

//

void IncY( );

//

// method for decrease current X coordinate

//

void DecX( );

//

// method for decrease current Y coordinate

//

void DecY( );

//

// method for receiving current X coordinate

//

int GetCurrentX( );

//

// method for receiving current Y coordinate

//

int GetCurrentY( );

//

// method for set chosen X coordinate

//

void SetChosenX( int x );

//

// method for set chosen Y coordinate

//

void SetChosenY( int y );

//

// method for set next chosen X coordinate

//

void SetChosenNextX( int nx );

//

// method for set next chosen Y coordinate

//

void SetChosenNextY( int ny );

//

// method for set flag of chosen cell

//

void SetChosenFlag( bool flag );

//

// method for receiving flag of chosen cell

//

bool GetChosenFlag( );

//

// method for receiving chosen X coordinate

//

int GetChosenX( );

//

// method for receiving chosen Y coordinate

//

int GetChosenY( );

//

// method for receiving next chosen X coordinate

//

int GetChosenNextX( );

//

// method for receiving next chosen Y coordinate

//

int GetChosenNextY( );

//

// method for receiving turn count

//

int GetTurnCount( );

//

// method for check is striaght line is empty

//

bool CheckStraightLine( Cell& currentCell, Cell& nextCell );

//

// method for check is diagonal line is empty

//

bool CheckDiagLine( Cell& currentCell, Cell& nextCell );

};

#endif

#include "chessboard.h"

#include <iostream>

//

// Default ctor

//

Chessboard::Chessboard( )

{

\_iTurnCount = 2;

\_currentX = 0;

\_currentY = 0;

\_chosenX = 0;

\_chosenY = 0;

\_chosenNextX = 0;

\_chosenNextY = 0;

\_isChosen = false;

//

// Fill Position structure for each cell

//

for ( int i = 0; i < MAX\_CELL\_NUMBER; i++ ) {

for ( int j = 0; j < MAX\_CELL\_NUMBER; j++ ) {

\_board[ i ][ j ].SetPos( i, j );

\_board[ i ][ j ].SetFigure( NULL );

}

}

//

// Memory allocation for each figure

//

try

{

\_WhiteKing = new King( "white" );

\_BlackKing = new King( "black" );

\_WhitePawns = new Pawn[ MAX\_CELL\_NUMBER ];

\_BlackPawns = new Pawn[ MAX\_CELL\_NUMBER ];

\_WhiteHorses = new Horse[ 2 ];

\_BlackHorses = new Horse[ 2 ];

\_WhiteRooks = new Rook[ 2 ];

\_BlackRooks = new Rook[ 2 ];

\_WhiteBishops = new Bishop[ 2 ];

\_BlackBishops = new Bishop[ 2 ];

\_WhiteQueen = new Queen( "white" );

\_BlackQueen = new Queen( "black" );

}

catch( std::bad\_alloc badAlloc )

{

std::cout << "\_WhiteKing ptr " << \_WhiteKing << std::endl;

std::cout << "\_WhiteQueen ptr " << \_WhiteQueen << std::endl;

std::cout << "\_WhitePawns ptr " << \_WhitePawns << std::endl;

std::cout << "\_WhiteHorses ptr " << \_WhiteHorses << std::endl;

std::cout << "\_WhiteRooks ptr " << \_WhiteRooks << std::endl;

std::cout << "\_WhiteBishops ptr " << \_WhiteBishops << std::endl;

std::cout << "\_BlackKing ptr " << \_BlackKing << std::endl;

std::cout << "\_BlackQueen ptr " << \_BlackQueen << std::endl;

std::cout << "\_BlackPawns ptr " << \_BlackPawns << std::endl;

std::cout << "\_BlackHorses ptr " << \_BlackHorses << std::endl;

std::cout << "\_BlackRooks ptr " << \_BlackRooks << std::endl;

std::cout << "\_BlackBishops ptr " << \_BlackBishops << std::endl;

}

//

// Set figure's color

//

for ( int i = 0; i < 2; i++ ) {

\_WhiteHorses[ i ].SetColor( "white" );

\_WhiteRooks[ i ].SetColor( "white" );

\_WhiteBishops[ i ].SetColor( "white" );

\_BlackRooks[ i ].SetColor( "black" );

\_BlackHorses[ i ].SetColor( "black" );

\_BlackBishops[ i ].SetColor( "black" );

}

for ( int i = 0; i < MAX\_CELL\_NUMBER; i++ ) {

\_WhitePawns[ i ].SetColor( "white" );

\_BlackPawns[ i ].SetColor( "black" );

}

//

// Set Figure

//

\_board[ 4 ][ 0 ].SetFigure( \_WhiteKing );

\_board[ 3 ][ 0 ].SetFigure( \_WhiteQueen );

\_board[ 1 ][ 0 ].SetFigure( &\_WhiteHorses[ 0 ] );

\_board[ 6 ][ 0 ].SetFigure( &\_WhiteHorses[ 1 ] );

\_board[ 0 ][ 0 ].SetFigure( &\_WhiteRooks[ 0 ] );

\_board[ 7 ][ 0 ].SetFigure( &\_WhiteRooks[ 1 ] );

\_board[ 2 ][ 0 ].SetFigure( &\_WhiteBishops[ 0 ] );

\_board[ 5 ][ 0 ].SetFigure( &\_WhiteBishops[ 1 ] );

\_board[ 4 ][ 7 ].SetFigure( \_BlackKing );

\_board[ 3 ][ 7 ].SetFigure( \_BlackQueen );

\_board[ 1 ][ 7 ].SetFigure( &\_BlackHorses[ 0 ] );

\_board[ 6 ][ 7 ].SetFigure( &\_BlackHorses[ 1 ] );

\_board[ 0 ][ 7 ].SetFigure( &\_BlackRooks[ 0 ] );

\_board[ 7 ][ 7 ].SetFigure( &\_BlackRooks[ 1 ] );

\_board[ 2 ][ 7 ].SetFigure( &\_BlackBishops[ 0 ] );

\_board[ 5 ][ 7 ].SetFigure( &\_BlackBishops[ 1 ] );

for ( int i = 0; i < MAX\_CELL\_NUMBER; i++ ) {

\_board[ i ][ 1 ].SetFigure( &\_WhitePawns[ i ] );

\_board[ i ][ 6 ].SetFigure( &\_BlackPawns[ i ] );

}

}

//

// Dtor

//

Chessboard::~Chessboard( )

{

delete \_WhiteKing;

delete \_BlackKing;

delete [] \_WhitePawns;

delete [] \_BlackPawns;

delete [] \_WhiteHorses;

delete [] \_BlackHorses;

delete [] \_WhiteRooks;

delete [] \_BlackRooks;

delete [] \_WhiteBishops;

delete [] \_BlackBishops;

delete \_WhiteQueen;

delete \_BlackQueen;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Move

// Method task: Move x,y cell to nx, ny cell

// Access: public

// Returns: void

// Parameters: int, int, int, int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Chessboard::Move( int x, int y, int nx, int ny )

{

try

{

if ( \_board[ x ][ y ].IsEmpty( ) ) {

throw EmptyCell( );

}

}

catch( bvException bv )

{

bv.PrintError( );

return;

}

try

{

if ( \_iTurnCount % 2 == 0 && \_board[ x ][ y ].GetColor( ) == "black" ) {

throw NotUrTurn( );

}

else if( \_iTurnCount % 2 != 0 && \_board[ x ][ y ].GetColor( ) == "white" ) {

throw NotUrTurn( );

}

}

catch( bvException bv )

{

bv.PrintError( );

return;

}

if ( strcmp( "class Rook", typeid(\*(\_board[ x ][ y ].GetFigure( ))).name() ) == 0 ) {

try

{

CheckStraightLine( \_board[ x ][ y ], \_board[ nx ][ ny ] );

}

catch( bvException bv )

{

bv.PrintError( );

return;

}

}

else if( strcmp( "class Bishop", typeid(\*(\_board[ x ][ y ].GetFigure( ))).name() ) == 0 ) {

try

{

CheckDiagLine( \_board[ x ][ y ], \_board[ nx ][ ny ] );

}

catch (bvException bv)

{

bv.PrintError( );

return;

}

}

else if ( strcmp( "class Queen", typeid(\*(\_board[ x ][ y ].GetFigure( ))).name() ) == 0 ) {

if ( ( nx == x && ny != y ) || ( nx != x && ny == y ) ) {

try

{

CheckStraightLine( \_board[ x ][ y ], \_board[ nx ][ ny ] );

}

catch ( bvException bv )

{

bv.PrintError( );

return;

}

}

else if ( nx != x && ny != y ) {

try

{

CheckDiagLine( \_board[ x ][ y ], \_board[ nx ][ ny ] );

}

catch ( bvException bv )

{

bv.PrintError( );

return;

}

}

}

try

{

\_board[ x ][ y ].MoveFigure( \_board[ nx ][ ny ] );

}

catch( bvException bv )

{

bv.PrintError( );

return;

}

\_iTurnCount++;

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Draw

// Method task: Draw Chessboard

// Access: public

// Returns: void

// Parameters: D3DXMATRIX, IDirect3DDevice9\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Chessboard::Draw( D3DXMATRIX g\_Matrix, IDirect3DDevice9 \*g\_pDevice )

{

g\_pDevice->Clear( NULL, NULL, D3DCLEAR\_TARGET | D3DCLEAR\_ZBUFFER, D3DCOLOR\_XRGB( 100, 120, 150 ), 1.0f, NULL );

g\_pDevice->BeginScene( );

for ( int y = 0; y < MAX\_CELL\_NUMBER; y++ ) {

for ( int x = 0; x < MAX\_CELL\_NUMBER; x++ ) {

D3DXMatrixTranslation( &g\_Matrix, ( float )x - 3.5f, 0.5f, ( float )y - 2.5f );

g\_pDevice->SetTransform( D3DTS\_WORLD, &g\_Matrix );

if ( \_board[ x ][ y ].GetFigure( ) == NULL ) {

if ( x == \_currentX && y == \_currentY ) {

\_board[ x ][ y ].Draw( 2 );

}

else if ( ( x % 2 ) == 0 )

{

if ( ( y % 2 ) == 0 ) {

\_board[ x ][ y ].Draw( 1 );

}

else

\_board[ x ][ y ].Draw( 0 );

}

else

{

if ( ( y % 2 ) == 0 ) {

\_board[ x ][ y ].Draw( 0 );

}

else

\_board[ x ][ y ].Draw( 1 );

}

}

else

{

if ( x == \_currentX && y == \_currentY ) {

\_board[ x ][ y ].GetFigure( )->Draw( 2 );

}

else if ( \_board[ x ][ y ].GetColor( ) == "white" ) {

\_board[ x ][ y ].GetFigure( )->Draw( 7 );

}

else if ( \_board[ x ][ y ].GetColor( ) == "black" ) {

\_board[ x ][ y ].GetFigure( )->Draw( 5 );

}

}

}

}

g\_pDevice->EndScene( );

g\_pDevice->Present( NULL, NULL, NULL, NULL );

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: CheckStraightLine

// Method task: Check straight line from current cell to next cell

// Access: public

// Returns: bool

// Parameters: Cell&, Cell&

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Chessboard::CheckStraightLine( Cell& currentCell, Cell& nextCell )

{

Pos nextPos = nextCell.GetPos( );

Pos currentPos = currentCell.GetPos( );

if ( nextPos.\_x == currentPos.\_x )

{

if ( ( nextPos.\_y - currentPos.\_y ) > 0 ) {

for ( int i = 1; ( currentPos.\_y + i ) != nextPos.\_y; i++ ) {

if ( \_board[ nextPos.\_x ][ currentPos.\_y + i ].IsEmpty( ) ) {

continue;

}

else

throw BadStraightLine( );

}

}

else if( ( nextPos.\_y - currentPos.\_y ) < 0 ) {

for ( int i = 1; ( currentPos.\_y - i ) != nextPos.\_y; i++ ) {

if ( \_board[ nextPos.\_x ][ currentPos.\_y - i ].IsEmpty( ) ) {

continue;

}

else

throw BadStraightLine( );

}

}

}

else if ( nextPos.\_y == currentPos.\_y )

{

if ( ( nextPos.\_x - currentPos.\_x ) > 0 ) {

for ( int i = 1; ( currentPos.\_x + i ) != nextPos.\_x; i++ ) {

if ( \_board[ currentPos.\_x + i ][ nextPos.\_y ].IsEmpty( ) ) {

continue;

}

else

throw BadStraightLine( );

}

}

else if ( ( nextPos.\_x - currentPos.\_x ) < 0 ) {

for ( int i = 1; ( currentPos.\_x - i ) != nextPos.\_x; i++ ) {

if ( \_board[ currentPos.\_x - i ][ nextPos.\_y ].IsEmpty( ) ) {

continue;

}

else

throw BadStraightLine( );

}

}

}

return true;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: CheckDiagLine

// Method task: Check Diagonal Line from current cell to next cell

// Access: public

// Returns: bool

// Parameters: Cell&, Cell&

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Chessboard::CheckDiagLine( Cell& currentCell, Cell& nextCell )

{

Pos nextPos = nextCell.GetPos( );

Pos currentPos = currentCell.GetPos( );

if ( abs( nextPos.\_x - currentPos.\_x ) == abs( nextPos.\_y - currentPos.\_y ) ) {

if ( ( nextPos.\_x - currentPos.\_x ) > 0 && ( nextPos.\_y - currentPos.\_y ) > 0 ) {

for ( int i = 1; ( currentPos.\_x + i ) != nextPos.\_x , ( currentPos.\_y + i ) != nextPos.\_y; i++ ) {

if ( \_board[ currentPos.\_x + i ][ currentPos.\_y + i ].IsEmpty( ) ) {

continue;

}

else

throw BadDiagLine();

}

}

else if ( ( nextPos.\_x - currentPos.\_x ) < 0 && ( nextPos.\_y - currentPos.\_y ) < 0 ) {

for ( int i = 1; ( currentPos.\_x - i ) != nextPos.\_x , ( currentPos.\_y - i ) != nextPos.\_y; i++ ) {

if ( \_board[ currentPos.\_x - i ][ currentPos.\_y - i ].IsEmpty( ) ) {

continue;

}

else

throw BadDiagLine( );

}

}

else if ( ( nextPos.\_x - currentPos.\_x ) > 0 && ( nextPos.\_y - currentPos.\_y ) < 0 ) {

for ( int i = 1; ( currentPos.\_x + i ) != nextPos.\_x , ( currentPos.\_y - i ) != nextPos.\_y; i++ ) {

if ( \_board[ currentPos.\_x + i ][ currentPos.\_y - i ].IsEmpty( ) ) {

continue;

}

else

throw BadDiagLine( );

}

}

else if ( ( nextPos.\_x - currentPos.\_x ) < 0 && ( nextPos.\_y - currentPos.\_y ) > 0 ) {

for ( int i = 1; ( currentPos.\_x - i ) != nextPos.\_x , ( currentPos.\_y + i ) != nextPos.\_y; i++ ) {

if ( \_board[ currentPos.\_x - i ][ currentPos.\_y + i ].IsEmpty( ) ) {

continue;

}

else

throw BadDiagLine( );

}

}

}

else

throw BadDiagLine( );

return true;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: IncX

// Method task: Increase current cell's X coordinate

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Chessboard::IncX( )

{

if ( \_currentX + 1 == MAX\_CELL\_NUMBER ) {

return;

}

\_currentX++;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: IncY

// Method task: Increase current cell's Y coordinate

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Chessboard::IncY( )

{

if ( \_currentY + 1 == MAX\_CELL\_NUMBER ) {

return;

}

\_currentY++;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: DecX

// Method task: Decrease current cell's X coordinate

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Chessboard::DecX( )

{

if ( \_currentX == 0 ) {

return;

}

\_currentX--;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: DecY

// Method task: Decrease current cell's Y coordinate

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Chessboard::DecY( )

{

if ( \_currentY == 0 ) {

return;

}

\_currentY--;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: GeCurrentX

// Method task: Get current cell's X coordinate

// Access: public

// Returns: int

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Chessboard::GetCurrentX( )

{

return \_currentX;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: GetCurrentY

// Method task: Get current cell's Y coordinate

// Access: public

// Returns: int

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Chessboard::GetCurrentY( )

{

return \_currentY;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: SetChosenX

// Method task: Set source cell's X coordinate

// Access: public

// Returns: void

// Parameters: int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Chessboard::SetChosenX( int x )

{

\_chosenX = x;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: SetChosenY

// Method task: Set source cell's Y coordinate

// Access: public

// Returns: void

// Parameters: int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Chessboard::SetChosenY( int y )

{

\_chosenY = y;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: SetChosenNextX

// Method task: Set destination cell's X coordinate

// Access: public

// Returns: void

// Parameters: int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Chessboard::SetChosenNextX( int nx )

{

\_chosenNextX = nx;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: SetChosenNextY

// Method task: Set destination cell's Y coordinate

// Access: public

// Returns: void

// Parameters: int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Chessboard::SetChosenNextY(int ny)

{

\_chosenNextY = ny;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: SetChosenFlag

// Method task: Set chosen flag (check if src cell is chosen)

// Access: public

// Returns: void

// Parameters: bool

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Chessboard::SetChosenFlag( bool flag )

{

\_isChosen = flag;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: GetChosenFlag

// Method task: Get chosen flag

// Access: public

// Returns: bool

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Chessboard::GetChosenFlag( )

{

return \_isChosen;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: GetChosenX

// Method task: Get source cell's X coordinate

// Access: public

// Returns: int

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Chessboard::GetChosenX( )

{

return \_chosenX;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: GetChosenY

// Method task: Get source cell's Y coordinate

// Access: public

// Returns: int

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Chessboard::GetChosenY( )

{

return \_chosenY;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: GetChosenNextX

// Method task: Get destination cell's X coordinate

// Access: public

// Returns: int

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Chessboard::GetChosenNextX( )

{

return \_chosenNextX;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: GetChosenNextY

// Method task: Get destination cell's Y coordinate

// Access: public

// Returns: int

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Chessboard::GetChosenNextY( )

{

return \_chosenNextY;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: GetTurnCount

// Method task: Get turn count

// Access: public

// Returns: int

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Chessboard::GetTurnCount( )

{

return \_iTurnCount;

}

#ifndef \_BISHOP\_H\_

#define \_BISHOP\_H\_

#include "figure.h"

#include "cell.h"

class Bishop : public Figure

{

public:

Bishop( );

Bishop( std::string sColor );

~Bishop( );

bool CheckMove( Cell& currentCell, Cell& nextCell );

void Draw( int material );

};

#endif

#include "bishop.h"

//

// Default ctor

//

Bishop::Bishop( )

{

}

//

// Ctor with params

//

Bishop::Bishop( std::string sColor )

{

\_sColor = sColor;

}

//

// Dtor

//

Bishop::~Bishop( )

{

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: CheckMove

// Method task: Check Cell for movement

// Access: public

// Returns: bool

// Parameters: Cell&, Cell&

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Bishop::CheckMove( Cell &currentCell, Cell &nextCell )

{

Pos nextPos = nextCell.GetPos( );

Pos currentPos = currentCell.GetPos( );

if ( nextPos.\_x == currentPos.\_x && nextPos.\_y == currentPos.\_y ) {

throw BadMove( );

}

if ( !nextCell.IsEmpty( ) ) {

if ( nextCell.GetColor( ) == currentCell.GetColor( ) ) {

throw FriendlyFire();

}

}

return true;

}

#ifndef \_HORSE\_H\_

#define \_HORSE\_H\_

#include "figure.h"

#include "cell.h"

#include "globals.h"

#include <d3dx9.h>

#include <d3d9.h>

class Horse : public Figure

{

public:

Horse( );

Horse( std::string sColor );

~Horse( );

bool CheckMove( Cell& currentCell, Cell& nextCell );

void Draw( int material );

};

#endif

#include "horse.h"

//

// Default ctor

//

Horse::Horse( )

{

}

//

// Ctor with params

//

Horse::Horse( std::string sColor )

{

\_sColor = sColor;

}

//

// Dtor

//

Horse::~Horse( )

{

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: CheckMove

// Method task: Check Cell for movement

// Access: public

// Returns: bool

// Parameters: Cell&, Cell&

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Horse::CheckMove( Cell& currentCell, Cell& nextCell )

{

Pos nextPos = nextCell.GetPos( );

Pos currentPos = currentCell.GetPos( );

if ( nextPos.\_x == currentPos.\_x && nextPos.\_y == currentPos.\_y ) {\

throw BadMove( );

}

if ( ( abs( nextPos.\_x - currentPos.\_x ) == 2 && abs( nextPos.\_y - currentPos.\_y ) == 1 ) ||

( abs( nextPos.\_y - currentPos.\_y ) == 2 && abs( nextPos.\_x - currentPos.\_x ) == 1 ) ) {

if ( !nextCell.IsEmpty( ) ) {

if ( nextCell.GetColor( ) == currentCell.GetColor( ) ) {

throw FriendlyFire( );

}

else

return true;

}

else {

return true;

}

}

throw BadMove( );

}

#ifndef \_KING\_H\_

#define \_KING\_H\_

#include "figure.h"

#include "cell.h"

#include "globals.h"

#include "config.h"

class King : public Figure

{

public:

King( );

King( std::string sColor );

~King( );

bool CheckMove( Cell& currentCell, Cell& nextCell );

void Draw( int material );

};

#endif

#include "king.h"

//

// Default ctor

//

King::King( )

{

}

//

// Ctor with params

//

King::King( std::string sColor )

{

\_sColor = sColor;

}

//

// Dtor

//

King::~King( )

{

}

//

// CheckMove( ) method

//

bool King::CheckMove( Cell& currentCell, Cell& nextCell )

{

Pos nextPos = nextCell.GetPos( );

Pos currentPos = currentCell.GetPos( );

if ( nextPos.\_x == currentPos.\_x && nextPos.\_y == currentPos.\_y ) {

throw BadMove( );

}

if ( abs( nextPos.\_x - currentPos.\_x ) > 1 ||

abs( nextPos.\_y - currentPos.\_y ) > 1 ) {

throw BadMove( );

}

if ( !nextCell.IsEmpty( ) ) {

if ( nextCell.GetColor( ) == currentCell.GetColor( ) ) {

throw FriendlyFire( );

}

}

return true;

}

#ifndef \_PAWN\_H\_

#define \_PAWN\_H\_

#include "figure.h"

#include "cell.h"

#include "globals.h"

#include <d3dx9.h>

#include <d3d9.h>

class Pawn : public Figure

{

public:

Pawn( );

Pawn( std::string sColor );

~Pawn();

bool CheckMove( Cell& currentCell, Cell& nextCell );

void Draw( int material );

};

#endif

#include "pawn.h"

//

// Default ctor

//

Pawn::Pawn( )

{

}

//

// Ctor with params

//

Pawn::Pawn( std::string sColor )

{

\_sColor = sColor;

}

//

// Dtor

//

Pawn::~Pawn( )

{

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: CheckMove

// Method task: Check cell for movement

// Access: public

// Returns: bool

// Parameters: Cell&, Cell&

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool Pawn::CheckMove( Cell& currentCell, Cell &nextCell )

{

Pos nextPos = nextCell.GetPos( );

Pos currentPos = currentCell.GetPos( );

if ( nextPos.\_x == currentPos.\_x && nextPos.\_y == currentPos.\_y ) {

throw BadMove( );

}

if ( ( abs( nextPos.\_y - currentPos.\_y ) > 1 ) &&

( currentPos.\_y != 1 && currentPos.\_y != 6 ) ) {

throw BadMove( );

}

if ( ( abs( nextPos.\_x - currentPos.\_x ) > 2 ||

abs(nextPos.\_y - currentPos.\_y ) > 2 ) &&

( currentPos.\_y == 1 || currentPos.\_y == 6 ) ) {

throw BadMove( );

}

if ( nextPos.\_x == currentPos.\_x && !nextCell.IsEmpty( ) ) {

throw BadMove( );

}

if ( nextPos.\_y == currentPos.\_y && nextPos.\_x != currentPos.\_x ) {

throw BadMove( );

}

if ( ( abs( nextPos.\_x - currentPos.\_x) >= 1 &&

abs( nextPos.\_y - currentPos.\_y) >= 1 ) ) {

if ( nextCell.IsEmpty( ) ) {

throw BadMove( );;

}

if ( nextCell.GetColor( ) == currentCell.GetColor( ) ) {

throw FriendlyFire( );

}

}

return true;

}

#ifndef \_QUEEN\_H\_

#define \_QUEEN\_H\_

#include "figure.h"

#include "cell.h"

#include "globals.h"

#include "config.h"

class Queen : public Figure

{

public:

Queen( );

Queen( std::string sColor );

~Queen( );

bool CheckMove( Cell& currentCell, Cell& nextCell );

void Draw( int material );

};

#endif

#include "queen.h"

//

// Default ctor

//

Queen::Queen( )

{

}

//

// Ctor with params

//

Queen::Queen( std::string sColor )

{

\_sColor = sColor;

}

//

// Dtor

//

Queen::~Queen( )

{

}

//

// CheckMove( ) method

//

bool Queen::CheckMove( Cell& currentCell, Cell& nextCell )

{

Pos nextPos = nextCell.GetPos( );

Pos currentPos = currentCell.GetPos( );

if ( nextPos.\_x == currentPos.\_x && nextPos.\_y == currentPos.\_y ) {

throw BadMove( );

}

if ( !nextCell.IsEmpty( ) ) {

if ( nextCell.GetColor( ) == currentCell.GetColor( ) ) {

throw FriendlyFire( );

}

}

return true;

}

#ifndef \_ROOK\_H\_

#define \_ROOK\_H\_

#include "cell.h"

#include "figure.h"

#include "globals.h"

#include "config.h"

class Rook : public Figure

{

public:

Rook( );

Rook( std::string sColor );

~Rook( );

bool CheckMove( Cell& currentCell, Cell& nextCell );

void Draw( int material );

};

#endif

#include "rook.h"

//

// Default ctor

//

Rook::Rook( )

{

}

//

// Ctor with params

//

Rook::Rook( std::string sColor )

{

\_sColor = sColor;

}

//

// Dtor

//

Rook::~Rook( )

{

}

//

// CheckMove( ) method

//

bool Rook::CheckMove( Cell& currentCell, Cell& nextCell )

{

Pos nextPos = nextCell.GetPos( );

Pos currentPos = currentCell.GetPos( );

if ( nextPos.\_x == currentPos.\_x && nextPos.\_y == currentPos.\_y ) {

throw BadMove( );

}

if ( nextPos.\_x != currentPos.\_x && nextPos.\_y != currentPos.\_y ) {

throw BadMove( );

}

if ( !nextCell.IsEmpty( ) ) {

if ( nextCell.GetColor( ) == currentCell.GetColor( ) ) {

throw FriendlyFire( );

}

}

return true;

}

#ifndef \_CAMERA\_H\_

#define \_CAMERA\_H\_

#include "globals.h"

#define MOVEMENT\_OFFSET 0.5f

class Camera

{

//

// Camera's position

//

float \_xPos;

float \_yPos;

float \_zPos;

//

// Camera's vector

//

D3DXVECTOR3 \_pos;

public:

Camera( );

void Set( IDirect3DDevice9\* g\_pDevice );

//

// Set camera's position

//

void SetOnWhite( );

void SetOnBlack( );

void SetOnMiddle( );

//

// Init camera

//

void TurnOn( IDirect3DDevice9\* g\_pDevice );

//

// Increase camera's postion methods

//

void IncX( );

void IncY( );

void IncZ( );

//

// Decrease camera's position methods

//

void DecX( );

void DecY( );

void DecZ( );

};

#endif

#include "camera.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Camera

// Method task: Camera class' default constructor

// Access: public

// Returns: -

// Parameters: -

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Camera::Camera( )

{

\_xPos = 0.0f;

\_yPos = 0.0f;

\_zPos = 0.0f;

\_pos = D3DXVECTOR3( \_xPos, \_yPos, \_zPos );

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: TurnOn

// Method task: Camera's init

// Access: public

// Returns: void

// Parameters: IDirect3DDevice9\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Camera::TurnOn( IDirect3DDevice9\* g\_pDevice )

{

\_xPos = 0.0f;

\_yPos = 10.0f;

\_zPos = -6.0f;

\_pos = D3DXVECTOR3( \_xPos, \_yPos, \_zPos );

D3DXMATRIX matProj;

D3DXMatrixLookAtLH( &matProj, &\_pos, &D3DXVECTOR3( 0.0f, 0.0f, 0.0f ), &D3DXVECTOR3( 0.0f, 1.0f, 0.0f ) );

g\_pDevice->SetTransform( D3DTS\_VIEW, &matProj );

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Set

// Method task: Set Camera on Scene

// Access: public

// Returns: void

// Parameters: IDirect3DDevice9\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Camera::Set( IDirect3DDevice9\* g\_pDevice )

{

D3DXMATRIX matProj;

\_pos = D3DXVECTOR3( \_xPos, \_yPos, \_zPos );

D3DXMatrixLookAtLH( &matProj, &\_pos, &D3DXVECTOR3( 0.0f, 0.0f, 0.0f ), &D3DXVECTOR3( 0.0f, 1.0f, 0.0f ) );

g\_pDevice->SetTransform( D3DTS\_VIEW, &matProj );

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: SetOnBlack

// Method task: Set Camera behind black figures

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Camera::SetOnBlack( )

{

\_xPos = 0.0f;

\_yPos = 10.0f;

\_zPos = 6.0f;

\_pos = D3DXVECTOR3( \_xPos, \_yPos, \_zPos );

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: SetOnWhite

// Method task: Set camera behind white figures

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Camera::SetOnWhite( )

{

\_xPos = 0.0f;

\_yPos = 10.0f;

\_zPos = -6.0f;

\_pos = D3DXVECTOR3( \_xPos, \_yPos, \_zPos );

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: SetOnMiddle

// Method task: Set camera on the middle of chessboard

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Camera::SetOnMiddle( )

{

\_xPos = 7.0f;

\_yPos = 11.0f;

\_zPos = 0.0f;

\_pos = D3DXVECTOR3( \_xPos, \_yPos, \_zPos );

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: IncX

// Method task: Increase X coordinate

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Camera::IncX( )

{

\_xPos += MOVEMENT\_OFFSET;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: IncY

// Method task: Increase Y coordinate

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Camera::IncY( )

{

\_yPos += MOVEMENT\_OFFSET;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: IncZ

// Method task: Increase Z coordinate

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Camera::IncZ( )

{

\_zPos += MOVEMENT\_OFFSET;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: DecX

// Method task: Decrease X coordinate

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Camera::DecX( )

{

\_xPos -= MOVEMENT\_OFFSET;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: DecY

// Method task: Decrease Y coordinate

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Camera::DecY( )

{

\_yPos -= MOVEMENT\_OFFSET;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: DecZ

// Method task: Decrease Z coordinate

// Access: public

// Returns: void

// Parameters: void

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Camera::DecZ( )

{

\_zPos -= MOVEMENT\_OFFSET;

}

#ifndef \_EXCEPTION\_H\_

#define \_EXCEPTION\_H\_

class bvException

{

protected:

WCHAR\* \_errorName;

public:

virtual void PrintError( )

{

::MessageBox(0, \_errorName, L"Bad!", 0);

}

};

class BadMove: public bvException

{

public:

BadMove( )

{

\_errorName = L"Bad move!";

}

};

class EmptyCell: public bvException

{

public:

EmptyCell( )

{

\_errorName = L"Chosen cell is empty!";

}

};

class NotUrTurn: public bvException

{

public:

NotUrTurn( )

{

\_errorName = L"It's mot ur turn!";

}

};

class FriendlyFire: public bvException

{

public:

FriendlyFire( )

{

\_errorName = L"U cant beat ur friend!";

}

};

class BadDiagLine: public bvException

{

public:

BadDiagLine( )

{

\_errorName = L"Check diagonal line!";

}

};

class BadStraightLine: public bvException

{

public:

BadStraightLine( )

{

\_errorName = L"Check Straight Line!";

}

};

#endif

#ifndef \_\_d3dUtilityH\_\_

#define \_\_d3dUtilityH\_\_

#include <d3dx9.h>

#include <string>

namespace d3d

{

bool InitD3D(

HINSTANCE hInstance, // [in] Application instance.

int width, int height, // [in] Backbuffer dimensions.

bool windowed, // [in] Windowed (true)or full screen (false).

D3DDEVTYPE deviceType, // [in] HAL or REF

IDirect3DDevice9\*\* device);// [out]The created device.

int EnterMsgLoop(

bool ( \*ptr\_display )( float timeDelta ) );

LRESULT CALLBACK WndProc(

HWND hwnd,

UINT msg,

WPARAM wParam,

LPARAM lParam);

template<class T> void Release( T t )

{

if( t )

{

t->Release( );

t = 0;

}

}

template<class T> void Delete( T t )

{

if( t )

{

delete t;

t = 0;

}

}

const D3DXCOLOR WHITE( D3DCOLOR\_XRGB(255, 255, 255) );

const D3DXCOLOR BLACK( D3DCOLOR\_XRGB( 0, 0, 0) );

const D3DXCOLOR RED( D3DCOLOR\_XRGB(255, 0, 0) );

const D3DXCOLOR GREEN( D3DCOLOR\_XRGB( 0, 255, 0) );

const D3DXCOLOR BLUE( D3DCOLOR\_XRGB( 0, 0, 255) );

const D3DXCOLOR YELLOW( D3DCOLOR\_XRGB(255, 255, 0) );

const D3DXCOLOR CYAN( D3DCOLOR\_XRGB( 0, 255, 255) );

const D3DXCOLOR MAGENTA( D3DCOLOR\_XRGB(255, 0, 255) );

//

// Lights

//

D3DLIGHT9 InitDirectionalLight( D3DXVECTOR3\* direction, D3DXCOLOR\* color );

D3DLIGHT9 InitPointLight( D3DXVECTOR3\* position, D3DXCOLOR\* color );

D3DLIGHT9 InitSpotLight( D3DXVECTOR3\* position, D3DXVECTOR3\* direction, D3DXCOLOR\* color );

//

// Materials

//

D3DMATERIAL9 InitMtrl( D3DXCOLOR a, D3DXCOLOR d, D3DXCOLOR s, D3DXCOLOR e, float p );

const D3DMATERIAL9 WHITE\_MTRL = InitMtrl( WHITE, WHITE, WHITE, BLACK, 2.0f );

const D3DMATERIAL9 RED\_MTRL = InitMtrl( RED, RED, RED, BLACK, 2.0f );

const D3DMATERIAL9 GREEN\_MTRL = InitMtrl( GREEN, GREEN, GREEN, BLACK, 2.0f );

const D3DMATERIAL9 BLUE\_MTRL = InitMtrl( BLUE, BLUE, BLUE, BLACK, 2.0f );

const D3DMATERIAL9 YELLOW\_MTRL = InitMtrl( YELLOW, YELLOW, YELLOW, BLACK, 2.0f );

}

#endif // \_\_d3dUtilityH\_\_

#include "d3dUtility.h"

bool d3d::InitD3D(

HINSTANCE hInstance,

int width, int height,

bool windowed,

D3DDEVTYPE deviceType,

IDirect3DDevice9\*\* device)

{

//

// Create the main application window.

//

WNDCLASS wc;

wc.style = CS\_HREDRAW | CS\_VREDRAW;

wc.lpfnWndProc = ( WNDPROC )d3d::WndProc;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hInstance = hInstance;

wc.hIcon = LoadIcon( 0, IDI\_APPLICATION );

wc.hCursor = LoadCursor( 0, IDC\_ARROW );

wc.hbrBackground = (HBRUSH)GetStockObject( LTGRAY\_BRUSH );

wc.lpszMenuName = 0;

wc.lpszClassName = L"Chess3D";

if( !RegisterClass( &wc ) )

{

::MessageBox( 0, L"RegisterClass() - FAILED", 0, 0 );

return false;

}

HWND hwnd = 0;

hwnd = ::CreateWindow( L"Chess3D", L"Chess3D",

WS\_EX\_TOPMOST,

0, 0, width, height,

0 /\*parent hwnd\*/, 0 /\* menu \*/, hInstance, 0 /\*extra\*/ );

if( !hwnd )

{

::MessageBox( 0, L"CreateWindow() - FAILED", 0, 0 );

return false;

}

::ShowWindow( hwnd, SW\_SHOW );

::UpdateWindow( hwnd );

//

// Init D3D:

//

HRESULT hr = 0;

// Step 1: Create the IDirect3D9 object.

IDirect3D9\* d3d9 = 0;

d3d9 = Direct3DCreate9( D3D\_SDK\_VERSION );

if( !d3d9 )

{

::MessageBox( 0, L"Direct3DCreate9( ) - FAILED", 0, 0 );

return false;

}

// Step 2: Check for hardware vp.

D3DCAPS9 caps;

d3d9->GetDeviceCaps( D3DADAPTER\_DEFAULT, deviceType, &caps );

int vp = 0;

if( caps.DevCaps & D3DDEVCAPS\_HWTRANSFORMANDLIGHT )

vp = D3DCREATE\_HARDWARE\_VERTEXPROCESSING;

else

vp = D3DCREATE\_SOFTWARE\_VERTEXPROCESSING;

// Step 3: Fill out the D3DPRESENT\_PARAMETERS structure.

D3DPRESENT\_PARAMETERS d3dpp;

d3dpp.BackBufferWidth = width;

d3dpp.BackBufferHeight = height;

d3dpp.BackBufferFormat = D3DFMT\_A8R8G8B8;

d3dpp.BackBufferCount = 1;

d3dpp.MultiSampleType = D3DMULTISAMPLE\_NONE;

d3dpp.MultiSampleQuality = 0;

d3dpp.SwapEffect = D3DSWAPEFFECT\_DISCARD;

d3dpp.hDeviceWindow = hwnd;

d3dpp.Windowed = windowed;

d3dpp.EnableAutoDepthStencil = true;

// d3dpp.AutoDepthStencilFormat = D3DFMT\_D24X4S4;

d3dpp.AutoDepthStencilFormat = D3DFMT\_D16;

d3dpp.Flags = 0;

d3dpp.FullScreen\_RefreshRateInHz = D3DPRESENT\_RATE\_DEFAULT;

d3dpp.PresentationInterval = D3DPRESENT\_INTERVAL\_IMMEDIATE;

// Step 4: Create the device.

hr = d3d9->CreateDevice(

D3DADAPTER\_DEFAULT, // primary adapter

deviceType, // device type

hwnd, // window associated with device

vp, // vertex processing

&d3dpp, // present parameters

device ); // return created device

if( FAILED( hr ) )

{

// try again using a 16-bit depth buffer

d3dpp.AutoDepthStencilFormat = D3DFMT\_D16;

hr = d3d9->CreateDevice(

D3DADAPTER\_DEFAULT,

deviceType,

hwnd,

vp,

&d3dpp,

device );

if( FAILED( hr ) )

{

d3d9->Release( ); // done with d3d9 object

::MessageBox( 0, L"CreateDevice() - FAILED", 0, 0 );

return false;

}

}

d3d9->Release( ); // done with d3d9 object

return true;

}

int d3d::EnterMsgLoop( bool ( \*ptr\_display )( float timeDelta ) )

{

MSG msg;

::ZeroMemory( &msg, sizeof( MSG ) );

static float lastTime = ( float )timeGetTime();

while( msg.message != WM\_QUIT )

{

if( ::PeekMessage( &msg, 0, 0, 0, PM\_REMOVE ) )

{

::TranslateMessage( &msg );

::DispatchMessage( &msg );

}

else

{

float currTime = ( float )timeGetTime( );

float timeDelta = ( currTime - lastTime )\*0.001f;

ptr\_display( timeDelta );

lastTime = currTime;

}

}

return msg.wParam;

}

D3DLIGHT9 d3d::InitDirectionalLight( D3DXVECTOR3\* direction, D3DXCOLOR\* color )

{

D3DLIGHT9 light;

::ZeroMemory( &light, sizeof( light ) );

light.Type = D3DLIGHT\_DIRECTIONAL;

light.Ambient = \*color \* 0.6f;

light.Diffuse = \*color;

light.Specular = \*color \* 0.6f;

light.Direction = \*direction;

return light;

}

D3DLIGHT9 d3d::InitPointLight( D3DXVECTOR3\* position, D3DXCOLOR\* color )

{

D3DLIGHT9 light;

::ZeroMemory( &light, sizeof( light ) );

light.Type = D3DLIGHT\_POINT;

light.Ambient = \*color \* 0.6f;

light.Diffuse = \*color;

light.Specular = \*color \* 0.6f;

light.Position = \*position;

light.Range = 1000.0f;

light.Falloff = 1.0f;

light.Attenuation0 = 1.0f;

light.Attenuation1 = 0.0f;

light.Attenuation2 = 0.0f;

return light;

}

D3DLIGHT9 d3d::InitSpotLight( D3DXVECTOR3\* position, D3DXVECTOR3\* direction, D3DXCOLOR\* color )

{

D3DLIGHT9 light;

::ZeroMemory( &light, sizeof( light ) );

light.Type = D3DLIGHT\_SPOT;

light.Ambient = \*color \* 0.0f;

light.Diffuse = \*color;

light.Specular = \*color \* 0.6f;

light.Position = \*position;

light.Direction = \*direction;

light.Range = 1000.0f;

light.Falloff = 1.0f;

light.Attenuation0 = 1.0f;

light.Attenuation1 = 0.0f;

light.Attenuation2 = 0.0f;

light.Theta = 0.4f;

light.Phi = 0.9f;

return light;

}

D3DMATERIAL9 d3d::InitMtrl( D3DXCOLOR a, D3DXCOLOR d,

D3DXCOLOR s, D3DXCOLOR e, float p )

{

D3DMATERIAL9 mtrl;

mtrl.Ambient = a;

mtrl.Diffuse = d;

mtrl.Specular = s;

mtrl.Emissive = e;

mtrl.Power = p;

return mtrl;

}

#include "d3dUtility.h"

#include "config.h"

#include <vector>

//

// Global variables

struct CUSTOMVERTEX

{

D3DXVECTOR3 position;

D3DXVECTOR3 normal;

};

//

// Camera object

//

Camera Camera;

//

// Chessboard object

//

Chessboard board;

//

// Indices array

//

WORD g\_Cube[64];

//

// Device Pointer

//

IDirect3DDevice9\* g\_pDevice = 0;

//

// Material Array

//

D3DMATERIAL9 g\_Material[ 9 ];

//

// Matrix for drawin objects

//

D3DXMATRIX g\_Matrix;

//

// Parameters of our window

//

const int Width = 1024;

const int Height = 768;

//

// CUSTOMVERTEX region: here we have declaration arrays of vectors for each figure

//

CUSTOMVERTEX g\_BoardCube[ 8 ];

CUSTOMVERTEX g\_BoardPawn[ 8 ];

CUSTOMVERTEX g\_BoardRook[ 8 ], g\_BoardRook1[ 8 ], g\_BoardRook2[ 9 ];

CUSTOMVERTEX g\_BoardBishop[ 8 ], g\_BoardBishop1[ 8 ], g\_BoardBishop2[ 8 ];

CUSTOMVERTEX g\_BoardHorse[ 8 ], g\_BoardHorse1[ 8 ], g\_BoardHorse2[ 8 ], g\_BoardHorse3[ 8 ];

CUSTOMVERTEX g\_BoardQueen[ 8 ], g\_BoardQueen1[ 8 ], g\_BoardQueen2[ 8 ], g\_BoardQueen3[ 8 ];

CUSTOMVERTEX g\_BoardKing[ 8 ], g\_BoardKing1[ 8 ], g\_BoardKing2[ 8 ], g\_BoardKing3[ 8 ], g\_BoardKing4[ 8 ];

CUSTOMVERTEX g\_BoardFloor[ 4 ];

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Draw

// Method task: Draw Bishop Figure

// Access: public

// Returns: void

// Parameters: int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Bishop::Draw( int material )

{

g\_pDevice->SetMaterial( &g\_Material[ material ] );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardBishop, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardBishop1, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardBishop2, sizeof( CUSTOMVERTEX ) );

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Draw

// Method task: Draw Horse Figure

// Access: public

// Returns: void

// Parameters: int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Horse::Draw( int material )

{

g\_pDevice->SetMaterial( &g\_Material[ material ] );

if ( this->\_sColor == "black" ) {

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardHorse, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardHorse1, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardHorse2, sizeof( CUSTOMVERTEX ) );

return;

}

else {

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12,&g\_Cube, D3DFMT\_INDEX16, &g\_BoardHorse, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12,&g\_Cube, D3DFMT\_INDEX16, &g\_BoardHorse3, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12,&g\_Cube, D3DFMT\_INDEX16, &g\_BoardHorse2, sizeof( CUSTOMVERTEX ) );

return;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Draw

// Method task: Draw King Figure

// Access: public

// Returns: void

// Parameters: int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void King::Draw( int material )

{

g\_pDevice->SetMaterial( &g\_Material[ material ] );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12,&g\_Cube, D3DFMT\_INDEX16, &g\_BoardKing, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12,&g\_Cube, D3DFMT\_INDEX16, &g\_BoardKing1, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12,&g\_Cube, D3DFMT\_INDEX16, &g\_BoardKing2, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12,&g\_Cube, D3DFMT\_INDEX16, &g\_BoardKing3, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12,&g\_Cube, D3DFMT\_INDEX16, &g\_BoardKing4, sizeof( CUSTOMVERTEX ) );

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Draw

// Method task: Draw Pawn Figure

// Access: public

// Returns: void

// Parameters: int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Pawn::Draw( int material )

{

g\_pDevice->SetMaterial( &g\_Material[ material ] );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST, 0, 8, 12,&g\_Cube, D3DFMT\_INDEX16, &g\_BoardPawn, sizeof( CUSTOMVERTEX ) );

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Draw

// Method task: Draw Queen Figure

// Access: public

// Returns: void

// Parameters: int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Queen::Draw( int material )

{

g\_pDevice->SetMaterial( &g\_Material[ material ] );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST, 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardQueen, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST, 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardQueen1, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST, 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardQueen2, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST, 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardQueen3, sizeof( CUSTOMVERTEX ) );

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Draw

// Method task: Rook

// Access: public

// Returns: void

// Parameters: int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Rook::Draw( int material )

{

g\_pDevice->SetMaterial( &g\_Material[ material ] );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardRook, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardRook1, sizeof( CUSTOMVERTEX ) );

g\_pDevice->DrawIndexedPrimitiveUP( D3DPT\_TRIANGLELIST , 0, 8, 12, &g\_Cube, D3DFMT\_INDEX16, &g\_BoardRook2, sizeof( CUSTOMVERTEX ) );

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Method: Draw

// Method task: Draw Cell

// Access: public

// Returns: void

// Parameters: int

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Cell::Draw( int material )

{

g\_pDevice->SetMaterial( &g\_Material[ material ] );

g\_pDevice->DrawPrimitiveUP( D3DPT\_TRIANGLESTRIP, 2, &g\_BoardFloor, sizeof( CUSTOMVERTEX ) );

return;

}

//

// Setup function - here we'll do all "before\_drawing" job

//

bool Setup( )

{

//

// White

//

g\_Material[ 0 ].Diffuse.r = g\_Material[ 0 ].Diffuse.g = g\_Material[ 0 ].Diffuse.b = 0.5f;

g\_Material[ 0 ].Emissive.g = 0.0f;

g\_Material[ 0 ].Emissive.b = 0.0f;

g\_Material[ 0 ].Emissive.r = 0.0f;

//

// Black cell of chessboard

//

g\_Material[1].Diffuse = g\_Material[0].Diffuse;

g\_Material[1].Emissive.r = 0.6f;

g\_Material[1].Emissive.g = 0.6f;

g\_Material[1].Emissive.b = 0.6f;

//

// Green cell

//

g\_Material[2].Diffuse = g\_Material[0].Diffuse;

g\_Material[2].Emissive.r = 0.2f;

g\_Material[2].Emissive.g = 0.4f;

g\_Material[2].Emissive.b = 0.1f;

g\_Material[5].Diffuse = g\_Material[0].Diffuse;

g\_Material[5].Emissive.r = 0.0f;

g\_Material[5].Emissive.g = 0.0f;

g\_Material[5].Emissive.b = 0.5f;

g\_Material[7].Diffuse = g\_Material[0].Diffuse;

g\_Material[7].Emissive.r = 0.49f; //49

g\_Material[7].Emissive.g = 0.34f; //34

g\_Material[7].Emissive.b = 0.0f; //0

//

// sets the view perspective

//

D3DXMatrixPerspectiveFovLH( &g\_Matrix, D3DX\_PI / 3, 1.0f, 1.0f, 100.0f );

g\_pDevice->SetTransform( D3DTS\_PROJECTION, &g\_Matrix );

g\_BoardCube[ 0 ].position = D3DXVECTOR3( -0.5f, 0.1f, -0.5f );

g\_BoardCube[ 0 ].normal = D3DXVECTOR3( -1.0f, 1.0f, -1.0f );

g\_BoardCube[ 1 ].position = D3DXVECTOR3( 0.5f, 0.1f, -0.5f );

g\_BoardCube[ 1 ].normal = D3DXVECTOR3( 1.0f, 1.0f, -1.0f );

g\_BoardCube[ 2 ].position = D3DXVECTOR3( -0.5f, -0.5f, -0.5f );

g\_BoardCube[ 2 ].normal = D3DXVECTOR3( -1.0f, -1.0f, -1.0f );

g\_BoardCube[ 3 ].position = D3DXVECTOR3( 0.5f, -0.5f, -0.5f );

g\_BoardCube[ 3 ].normal = D3DXVECTOR3( 1.0f, -1.0f, -1.0f );

g\_BoardCube[ 4 ].position = D3DXVECTOR3( -0.5f, 0.1f, 0.5f );

g\_BoardCube[ 4 ].normal = D3DXVECTOR3( -1.0f, 1.0f, 1.0f );

g\_BoardCube[ 5 ].position = D3DXVECTOR3( 0.5f, 0.1f, 0.5f );

g\_BoardCube[ 5 ].normal = D3DXVECTOR3( 1.0f, 1.0f, 1.0f );

g\_BoardCube[ 6 ].position = D3DXVECTOR3( -0.5f, -0.5f, 0.5f );

g\_BoardCube[ 6 ].normal = D3DXVECTOR3( -1.0f, -1.0f, 1.0f );

g\_BoardCube[ 7 ].position = D3DXVECTOR3( 0.5f, -0.5f, 0.5f );

g\_BoardCube[ 7 ].normal = D3DXVECTOR3( 1.0f, -1.0f, 1.0f );

//

// Pawn Init

//

g\_BoardPawn[ 0 ].position = D3DXVECTOR3( -0.1f, 1.0f, -0.1f );

g\_BoardPawn[ 0 ].normal = D3DXVECTOR3( -1.0f, 1.0f, -1.0f );

g\_BoardPawn[ 1 ].position = D3DXVECTOR3( 0.1f, 1.0f, -0.1f );

g\_BoardPawn[ 1 ].normal = D3DXVECTOR3( 1.0f, 1.0f, -1.0f );

g\_BoardPawn[ 2 ].position = D3DXVECTOR3( -0.5f, -0.5f, -0.5f );

g\_BoardPawn[ 2 ].normal = D3DXVECTOR3( -1.0f, -1.0f, -1.0f );

g\_BoardPawn[ 3 ].position = D3DXVECTOR3( 0.5f, -0.5f, -0.5f );

g\_BoardPawn[ 3 ].normal = D3DXVECTOR3( 1.0f, -1.0f, -1.0f );

g\_BoardPawn[ 4 ].position = D3DXVECTOR3( -0.1f, 1.0f, 0.1f );

g\_BoardPawn[ 4 ].normal = D3DXVECTOR3( -1.0f, 1.0f, 1.0f );

g\_BoardPawn[ 5 ].position = D3DXVECTOR3( 0.1f, 1.0f, 0.1f );

g\_BoardPawn[ 5 ].normal = D3DXVECTOR3( 1.0f, 1.0f, 1.0f );

g\_BoardPawn[ 6 ].position = D3DXVECTOR3( -0.5f, -0.5f, 0.5f );

g\_BoardPawn[ 6 ].normal = D3DXVECTOR3( -1.0f, -1.0f, 1.0f );

g\_BoardPawn[ 7 ].position = D3DXVECTOR3( 0.5f, -0.5f, 0.5f );

g\_BoardPawn[ 7 ].normal = D3DXVECTOR3( 1.0f, -1.0f, 1.0f );

//

// Rook init

//

g\_BoardRook[ 0 ].position = D3DXVECTOR3( -0.1f, 0.5f, -0.1f );

g\_BoardRook[ 0 ].normal = D3DXVECTOR3( -1.0f, 1.0f, -1.0f );

g\_BoardRook[ 1 ].position = D3DXVECTOR3( 0.1f, 0.5f, -0.1f );

g\_BoardRook[ 1 ].normal = D3DXVECTOR3( 1.0f, 1.0f, -1.0f );

g\_BoardRook[ 2 ].position = D3DXVECTOR3( -0.5f, -0.5f, -0.5f );

g\_BoardRook[ 2 ].normal = D3DXVECTOR3( -1.0f, -1.0f, -1.0f );

g\_BoardRook[ 3 ].position = D3DXVECTOR3( 0.5f, -0.5f, -0.5f );

g\_BoardRook[ 3 ].normal = D3DXVECTOR3( 1.0f, -1.0f, -1.0f );

g\_BoardRook[ 4 ].position = D3DXVECTOR3( -0.1f, 0.5f, 0.1f );

g\_BoardRook[ 4 ].normal = D3DXVECTOR3( -1.0f, 1.0f, 1.0f );

g\_BoardRook[ 5 ].position = D3DXVECTOR3( 0.1f, 0.5f, 0.1f );

g\_BoardRook[ 5 ].normal = D3DXVECTOR3( 1.0f, 1.0f, 1.0f );

g\_BoardRook[ 6 ].position = D3DXVECTOR3( -0.5f, -0.5f, 0.5f );

g\_BoardRook[ 6 ].normal = D3DXVECTOR3( -1.0f, -1.0f, 1.0f );

g\_BoardRook[ 7 ].position = D3DXVECTOR3( 0.5f, -0.5f, 0.5f );

g\_BoardRook[ 7 ].normal = D3DXVECTOR3( 1.0f, -1.0f, 1.0f );

g\_BoardRook1[ 0 ].position = D3DXVECTOR3( -0.5f, 2.0f, -0.5f );

g\_BoardRook1[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardRook1[ 1 ].position = D3DXVECTOR3( 0.5f, 2.0f, -0.5f );

g\_BoardRook1[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardRook1[ 2 ].position = D3DXVECTOR3( -0.1f, +1.0f, -0.1f );

g\_BoardRook1[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardRook1[ 3 ].position = D3DXVECTOR3( 0.1f, +1.0f, -0.1f );

g\_BoardRook1[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardRook1[ 4 ].position = D3DXVECTOR3( -0.5f, 2.0f, 0.5f );

g\_BoardRook1[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardRook1[ 5 ].position = D3DXVECTOR3( 0.5f, 2.0f, 0.5f );

g\_BoardRook1[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardRook1[ 6 ].position = D3DXVECTOR3( -0.1f, +1.0f, 0.1f );

g\_BoardRook1[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardRook1[ 7 ].position = D3DXVECTOR3( 0.1f, +1.0f, 0.1f );

g\_BoardRook1[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

g\_BoardRook2[ 0 ].position = D3DXVECTOR3( -0.3f, 1.9f, -0.3f );

g\_BoardRook2[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardRook2[ 1 ].position = D3DXVECTOR3( 0.3f, 1.9f, -0.3f );

g\_BoardRook2[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardRook2[ 2 ].position = D3DXVECTOR3( -0.3f, -0.1f, -0.3f );

g\_BoardRook2[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardRook2[ 3 ].position = D3DXVECTOR3( 0.3f, -0.1f, -0.3f );

g\_BoardRook2[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardRook2[ 4 ].position = D3DXVECTOR3( -0.3f, 1.9f, 0.3f );

g\_BoardRook2[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardRook2[ 5 ].position = D3DXVECTOR3( 0.3f, 1.9f, 0.3f );

g\_BoardRook2[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardRook2[ 6 ].position = D3DXVECTOR3( -0.3f, -0.1f, 0.3f );

g\_BoardRook2[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardRook2[ 7 ].position = D3DXVECTOR3( 0.3f, -0.1f, 0.3f );

g\_BoardRook2[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

//

// Horse init

//

g\_BoardHorse[ 0 ].position = D3DXVECTOR3( -0.1f, 0.5f, -0.1f );

g\_BoardHorse[ 0 ].normal = D3DXVECTOR3( -1.0f, 1.0f, -1.0f );

g\_BoardHorse[ 1 ].position = D3DXVECTOR3( 0.1f, 0.5f, -0.1f );

g\_BoardHorse[ 1 ].normal = D3DXVECTOR3( 1.0f, 1.0f, -1.0f );

g\_BoardHorse[ 2 ].position = D3DXVECTOR3( -0.5f, -0.5f, -0.5f );

g\_BoardHorse[ 2 ].normal = D3DXVECTOR3( -1.0f, -1.0f, -1.0f );

g\_BoardHorse[ 3 ].position = D3DXVECTOR3( 0.5f, -0.5f, -0.5f );

g\_BoardHorse[ 3 ].normal = D3DXVECTOR3( 1.0f, -1.0f, -1.0f );

g\_BoardHorse[ 4 ].position = D3DXVECTOR3( -0.1f, 0.5f, 0.1f );

g\_BoardHorse[ 4 ].normal = D3DXVECTOR3( -1.0f, 1.0f, 1.0f );

g\_BoardHorse[ 5 ].position = D3DXVECTOR3( 0.1f, 0.5f, 0.1f );

g\_BoardHorse[ 5 ].normal = D3DXVECTOR3( 1.0f, 1.0f, 1.0f );

g\_BoardHorse[ 6 ].position = D3DXVECTOR3( -0.5f, -0.5f, 0.5f );

g\_BoardHorse[ 6 ].normal = D3DXVECTOR3( -1.0f, -1.0f, 1.0f );

g\_BoardHorse[ 7 ].position = D3DXVECTOR3( 0.5f, -0.5f, 0.5f );

g\_BoardHorse[ 7 ].normal = D3DXVECTOR3( 1.0f, -1.0f, 1.0f );

g\_BoardHorse1[ 0 ].position = D3DXVECTOR3( -0.5f, 1.2f, -0.5f );

g\_BoardHorse1[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardHorse1[ 1 ].position = D3DXVECTOR3( 0.5f, 1.2f, -0.5f );

g\_BoardHorse1[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardHorse1[ 2 ].position = D3DXVECTOR3( -0.1f, +0.7f, -0.1f );

g\_BoardHorse1[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardHorse1[ 3 ].position = D3DXVECTOR3( 0.1f, +0.7f, -0.1f );

g\_BoardHorse1[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardHorse1[ 4 ].position = D3DXVECTOR3( -0.5f, 1.2f, 0.5f );

g\_BoardHorse1[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardHorse1[ 5 ].position = D3DXVECTOR3( 0.5f, 1.2f, 0.5f );

g\_BoardHorse1[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardHorse1[ 6 ].position = D3DXVECTOR3( -0.1f, +0.7f, 0.8f );

g\_BoardHorse1[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardHorse1[ 7 ].position = D3DXVECTOR3( 0.1f, +0.7f, 0.8f );

g\_BoardHorse1[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

g\_BoardHorse2[ 0 ].position = D3DXVECTOR3( -0.3f, 1.5f, -0.3f );

g\_BoardHorse2[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardHorse2[ 1 ].position = D3DXVECTOR3( 0.3f, 1.5f, -0.3f );

g\_BoardHorse2[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardHorse2[ 2 ].position = D3DXVECTOR3( -0.3f, -0.1f, -0.3f );

g\_BoardHorse2[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardHorse2[ 3 ].position = D3DXVECTOR3( 0.3f, -0.1f, -0.3f );

g\_BoardHorse2[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardHorse2[ 4 ].position = D3DXVECTOR3( -0.3f, 1.5f, 0.3f );

g\_BoardHorse2[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardHorse2[ 5 ].position = D3DXVECTOR3( 0.3f, 1.5f, 0.3f );

g\_BoardHorse2[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardHorse2[ 6 ].position = D3DXVECTOR3( -0.3f, -0.1f, 0.3f );

g\_BoardHorse2[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardHorse2[ 7 ].position = D3DXVECTOR3( 0.3f, -0.1f, 0.3f );

g\_BoardHorse2[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

g\_BoardHorse3[ 0 ].position = D3DXVECTOR3( -0.5f, 1.2f, -0.5f );

g\_BoardHorse3[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardHorse3[ 1 ].position = D3DXVECTOR3( 0.5f, 1.2f, -0.5f );

g\_BoardHorse3[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardHorse3[ 2 ].position = D3DXVECTOR3( -0.1f, +0.7f, -0.8f );

g\_BoardHorse3[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardHorse3[ 3 ].position = D3DXVECTOR3( 0.1f, +0.7f, -0.8f );

g\_BoardHorse3[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardHorse3[ 4 ].position = D3DXVECTOR3( -0.5f, 1.2f, 0.5f );

g\_BoardHorse3[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardHorse3[ 5 ].position = D3DXVECTOR3( 0.5f, 1.2f, 0.5f );

g\_BoardHorse3[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardHorse3[ 6 ].position = D3DXVECTOR3( -0.1f, +0.7f, 0.1f );

g\_BoardHorse3[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardHorse3[ 7 ].position = D3DXVECTOR3( 0.1f, +0.7f, 0.1f );

g\_BoardHorse3[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

//

// Bishop Init

//

g\_BoardBishop[ 0 ].position = D3DXVECTOR3( -0.1f, 0.5f, -0.1f );

g\_BoardBishop[ 0 ].normal = D3DXVECTOR3( -1.0f, 1.0f, -1.0f );

g\_BoardBishop[ 1 ].position = D3DXVECTOR3( 0.1f, 0.5f, -0.1f );

g\_BoardBishop[ 1 ].normal = D3DXVECTOR3( 1.0f, 1.0f, -1.0f );

g\_BoardBishop[ 2 ].position = D3DXVECTOR3( -0.5f, -0.5f, -0.5f );

g\_BoardBishop[ 2 ].normal = D3DXVECTOR3( -1.0f, -1.0f, -1.0f );

g\_BoardBishop[ 3 ].position = D3DXVECTOR3( 0.5f, -0.5f, -0.5f );

g\_BoardBishop[ 3 ].normal = D3DXVECTOR3( 1.0f, -1.0f, -1.0f );

g\_BoardBishop[ 4 ].position = D3DXVECTOR3( -0.1f, 0.5f, 0.1f );

g\_BoardBishop[ 4 ].normal = D3DXVECTOR3( -1.0f, 1.0f, 1.0f );

g\_BoardBishop[ 5 ].position = D3DXVECTOR3( 0.1f, 0.5f, 0.1f );

g\_BoardBishop[ 5 ].normal = D3DXVECTOR3( 1.0f, 1.0f, 1.0f );

g\_BoardBishop[ 6 ].position = D3DXVECTOR3( -0.5f, -0.5f, 0.5f );

g\_BoardBishop[ 6 ].normal = D3DXVECTOR3( -1.0f, -1.0f, 1.0f );

g\_BoardBishop[ 7 ].position = D3DXVECTOR3( 0.5f, -0.5f, 0.5f );

g\_BoardBishop[ 7 ].normal = D3DXVECTOR3( 1.0f, -1.0f, 1.0f );

g\_BoardBishop1[ 0 ].position = D3DXVECTOR3( -0.1f, 2.2f, -0.1f );

g\_BoardBishop1[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardBishop1[ 1 ].position = D3DXVECTOR3( 0.1f, 2.2f, -0.1f );

g\_BoardBishop1[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardBishop1[ 2 ].position = D3DXVECTOR3( -0.5f, +1.2f, -0.5f );

g\_BoardBishop1[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardBishop1[ 3 ].position = D3DXVECTOR3( 0.5f, +1.2f, -0.5f );

g\_BoardBishop1[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardBishop1[ 4 ].position = D3DXVECTOR3( -0.1f, 2.2f, 0.1f );

g\_BoardBishop1[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardBishop1[ 5 ].position = D3DXVECTOR3( 0.1f, 2.2f, 0.1f );

g\_BoardBishop1[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardBishop1[ 6 ].position = D3DXVECTOR3( -0.5f, +1.2f, 0.5f );

g\_BoardBishop1[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardBishop1[ 7 ].position = D3DXVECTOR3( 0.5f, +1.2f, 0.5f );

g\_BoardBishop1[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

g\_BoardBishop2[ 0 ].position = D3DXVECTOR3( -0.3f, 1.9f, -0.3f );

g\_BoardBishop2[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardBishop2[ 1 ].position = D3DXVECTOR3( 0.3f, 1.9f, -0.3f );

g\_BoardBishop2[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardBishop2[ 2 ].position = D3DXVECTOR3( -0.3f, -0.1f, -0.3f );

g\_BoardBishop2[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardBishop2[ 3 ].position = D3DXVECTOR3( 0.3f, -0.1f, -0.3f );

g\_BoardBishop2[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardBishop2[ 4 ].position = D3DXVECTOR3( -0.3f, 1.9f, 0.3f );

g\_BoardBishop2[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardBishop2[ 5 ].position = D3DXVECTOR3( 0.3f, 1.9f, 0.3f );

g\_BoardBishop2[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardBishop2[ 6 ].position = D3DXVECTOR3( -0.3f, -0.1f, 0.3f );

g\_BoardBishop2[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardBishop2[ 7 ].position = D3DXVECTOR3( 0.3f, -0.1f, 0.3f );

g\_BoardBishop2[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

//

// Queen init

//

g\_BoardQueen[ 0 ].position = D3DXVECTOR3( -0.1f, 0.5f, -0.1f );

g\_BoardQueen[ 0 ].normal = D3DXVECTOR3( -1.0f, 1.0f, -1.0f );

g\_BoardQueen[ 1 ].position = D3DXVECTOR3( 0.1f, 0.5f, -0.1f );

g\_BoardQueen[ 1 ].normal = D3DXVECTOR3( 1.0f, 1.0f, -1.0f );

g\_BoardQueen[ 2 ].position = D3DXVECTOR3( -0.5f, -0.5f, -0.5f );

g\_BoardQueen[ 2 ].normal = D3DXVECTOR3( -1.0f, -1.0f, -1.0f );

g\_BoardQueen[ 3 ].position = D3DXVECTOR3( 0.5f, -0.5f, -0.5f );

g\_BoardQueen[ 3 ].normal = D3DXVECTOR3( 1.0f, -1.0f, -1.0f );

g\_BoardQueen[ 4 ].position = D3DXVECTOR3( -0.1f, 0.5f, 0.1f );

g\_BoardQueen[ 4 ].normal = D3DXVECTOR3( -1.0f, 1.0f, 1.0f );

g\_BoardQueen[ 5 ].position = D3DXVECTOR3( 0.1f, 0.5f, 0.1f );

g\_BoardQueen[ 5 ].normal = D3DXVECTOR3( 1.0f, 1.0f, 1.0f );

g\_BoardQueen[ 6 ].position = D3DXVECTOR3( -0.5f, -0.5f, 0.5f );

g\_BoardQueen[ 6 ].normal = D3DXVECTOR3( -1.0f, -1.0f, 1.0f );

g\_BoardQueen[ 7 ].position = D3DXVECTOR3( 0.5f, -0.5f, 0.5f );

g\_BoardQueen[ 7 ].normal = D3DXVECTOR3( 1.0f, -1.0f, 1.0f );

g\_BoardQueen1[ 0 ].position = D3DXVECTOR3( -0.3f, 2.2f, -0.3f );

g\_BoardQueen1[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardQueen1[ 1 ].position = D3DXVECTOR3( 0.3f, 2.2f, -0.3f );

g\_BoardQueen1[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardQueen1[ 2 ].position = D3DXVECTOR3( -0.3f, -0.1f, -0.3f );

g\_BoardQueen1[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardQueen1[ 3 ].position = D3DXVECTOR3( 0.3f, -0.1f, -0.3f );

g\_BoardQueen1[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardQueen1[ 4 ].position = D3DXVECTOR3( -0.3f, 2.2f, 0.3f );

g\_BoardQueen1[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardQueen1[ 5 ].position = D3DXVECTOR3( 0.3f, 2.2f, 0.3f );

g\_BoardQueen1[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardQueen1[ 6 ].position = D3DXVECTOR3( -0.3f, -0.1f, 0.3f );

g\_BoardQueen1[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardQueen1[ 7 ].position = D3DXVECTOR3( 0.3f, -0.1f, 0.3f );

g\_BoardQueen1[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

g\_BoardQueen2[ 0 ].position = D3DXVECTOR3( -0.5f, 2.5f, -0.1f );

g\_BoardQueen2[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardQueen2[ 1 ].position = D3DXVECTOR3( 0.1f, 2.5f, -0.5f );

g\_BoardQueen2[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardQueen2[ 2 ].position = D3DXVECTOR3( -0.1f, +1.2f, -0.1f );

g\_BoardQueen2[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardQueen2[ 3 ].position = D3DXVECTOR3( 0.1f, +1.2f, -0.1f );

g\_BoardQueen2[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardQueen2[ 4 ].position = D3DXVECTOR3( -0.1f, 2.5f, 0.5f );

g\_BoardQueen2[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardQueen2[ 5 ].position = D3DXVECTOR3( 0.5f, 2.5f, 0.1f );

g\_BoardQueen2[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardQueen2[ 6 ].position = D3DXVECTOR3( -0.1f, +1.0f, 0.1f );

g\_BoardQueen2[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardQueen2[ 7 ].position = D3DXVECTOR3( 0.1f, +1.0f, 0.1f );

g\_BoardQueen2[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

g\_BoardQueen3[ 0 ].position = D3DXVECTOR3( -0.5f, 2.0f, -0.5f );

g\_BoardQueen3[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardQueen3[ 1 ].position = D3DXVECTOR3( 0.5f, 2.0f, -0.5f );

g\_BoardQueen3[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardQueen3[ 2 ].position = D3DXVECTOR3( -0.1f, +1.0f, -0.1f );

g\_BoardQueen3[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardQueen3[ 3 ].position = D3DXVECTOR3( 0.1f, +1.0f, -0.1f );

g\_BoardQueen3[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardQueen3[ 4 ].position = D3DXVECTOR3( -0.5f, 2.0f, 0.5f );

g\_BoardQueen3[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardQueen3[ 5 ].position = D3DXVECTOR3( 0.5f, 2.0f, 0.5f );

g\_BoardQueen3[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardQueen3[ 6 ].position = D3DXVECTOR3( -0.1f, +1.0f, 0.1f );

g\_BoardQueen3[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardQueen3[ 7 ].position = D3DXVECTOR3( 0.1f, +1.0f, 0.1f );

g\_BoardQueen3[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

//

// King init

//

g\_BoardKing[ 0 ].position = D3DXVECTOR3( -0.1f, 0.5f, -0.1f );

g\_BoardKing[ 0 ].normal = D3DXVECTOR3( -1.0f, 1.0f, -1.0f );

g\_BoardKing[ 1 ] .position = D3DXVECTOR3( 0.1f, 0.5f, -0.1f );

g\_BoardKing[ 1 ].normal = D3DXVECTOR3( 1.0f, 1.0f, -1.0f );

g\_BoardKing[ 2 ].position = D3DXVECTOR3( -0.5f, -0.5f, -0.5f );

g\_BoardKing[ 2 ].normal = D3DXVECTOR3( -1.0f, -1.0f, -1.0f );

g\_BoardKing[ 3 ].position = D3DXVECTOR3( 0.5f, -0.5f, -0.5f );

g\_BoardKing[ 3 ].normal = D3DXVECTOR3( 1.0f, -1.0f, -1.0f );

g\_BoardKing[ 4 ].position = D3DXVECTOR3( -0.1f, 0.5f, 0.1f );

g\_BoardKing[ 4 ].normal = D3DXVECTOR3( -1.0f, 1.0f, 1.0f );

g\_BoardKing[ 5 ].position = D3DXVECTOR3( 0.1f, 0.5f, 0.1f );

g\_BoardKing[ 5 ].normal = D3DXVECTOR3( 1.0f, 1.0f, 1.0f );

g\_BoardKing[ 6 ].position = D3DXVECTOR3( -0.5f, -0.5f, 0.5f );

g\_BoardKing[ 6 ].normal = D3DXVECTOR3( -1.0f, -1.0f, 1.0f );

g\_BoardKing[ 7 ].position = D3DXVECTOR3( 0.5f, -0.5f, 0.5f );

g\_BoardKing[ 7 ].normal = D3DXVECTOR3( 1.0f, -1.0f, 1.0f );

g\_BoardKing1[ 0 ].position = D3DXVECTOR3( -0.3f, 2.6f, -0.3f );

g\_BoardKing1[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardKing1[ 1 ].position = D3DXVECTOR3( 0.3f, 2.6f, -0.3f );

g\_BoardKing1[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardKing1[ 2 ].position = D3DXVECTOR3( -0.3f, -0.1f, -0.3f );

g\_BoardKing1[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardKing1[ 3 ].position = D3DXVECTOR3( 0.3f, -0.1f, -0.3f );

g\_BoardKing1[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardKing1[ 4 ].position = D3DXVECTOR3( -0.3f, 2.6f, 0.3f );

g\_BoardKing1[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardKing1[ 5 ].position = D3DXVECTOR3( 0.3f, 2.6f, 0.3f );

g\_BoardKing1[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardKing1[ 6 ].position = D3DXVECTOR3( -0.3f, -0.1f, 0.3f );

g\_BoardKing1[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardKing1[ 7 ].position = D3DXVECTOR3( 0.3f, -0.1f, 0.3f );

g\_BoardKing1[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

g\_BoardKing2[ 0 ].position = D3DXVECTOR3( -0.07f, 2.8f, -0.07f );

g\_BoardKing2[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardKing2[ 1 ].position = D3DXVECTOR3( 0.1f, 2.8f, -0.4f );

g\_BoardKing2[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardKing2[ 2 ].position = D3DXVECTOR3( -0.07f, +2.0f, -0.07f );

g\_BoardKing2[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardKing2[ 3 ].position = D3DXVECTOR3( 0.1f, +2.0f, -0.1f );

g\_BoardKing2[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardKing2[ 4 ].position = D3DXVECTOR3( -0.07f, 2.8f, 0.4f );

g\_BoardKing2[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardKing2[ 5 ].position = D3DXVECTOR3( 0.07f, 2.8f, 0.1f );

g\_BoardKing2[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardKing2[ 6 ].position = D3DXVECTOR3( -0.1f, +1.0f, 0.1f );

g\_BoardKing2[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardKing2[ 7 ].position = D3DXVECTOR3( 0.1f, +1.0f, 0.1f );

g\_BoardKing2[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

g\_BoardKing3[ 0 ].position = D3DXVECTOR3( -0.5f, 2.3f, -0.5f );

g\_BoardKing3[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardKing3[ 1 ].position = D3DXVECTOR3( 0.5f, 2.3f, -0.5f );

g\_BoardKing3[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardKing3[ 2 ].position = D3DXVECTOR3( -0.1f, +1.3f, -0.1f );

g\_BoardKing3[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardKing3[ 3 ].position = D3DXVECTOR3( 0.1f, +1.3f, -0.1f );

g\_BoardKing3[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardKing3[ 4 ].position = D3DXVECTOR3( -0.5f, 2.3f, 0.5f );

g\_BoardKing3[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardKing3[ 5 ].position = D3DXVECTOR3( 0.5f, 2.3f, 0.5f );

g\_BoardKing3[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardKing3[ 6 ].position = D3DXVECTOR3( -0.1f, +1.3f, 0.1f );

g\_BoardKing3[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardKing3[ 7 ].position = D3DXVECTOR3( 0.1f, +1.3f, 0.1f );

g\_BoardKing3[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

g\_BoardKing4[ 0 ].position = D3DXVECTOR3( -0.4f, 2.8f, -0.07f );

g\_BoardKing4[ 0 ].normal = D3DXVECTOR3( -1.5f, 1.0f, -1.0f );

g\_BoardKing4[ 1 ].position = D3DXVECTOR3( 0.07f, 2.8f, -0.07f );

g\_BoardKing4[ 1 ].normal = D3DXVECTOR3( 1.5f, 1.0f, -1.0f );

g\_BoardKing4[ 2 ].position = D3DXVECTOR3( -0.07f, +2.0f, -0.07f );

g\_BoardKing4[ 2 ].normal = D3DXVECTOR3( -1.5f, -1.0f, -1.0f );

g\_BoardKing4[ 3 ].position = D3DXVECTOR3( 0.07f, +2.0f, -0.07f );

g\_BoardKing4[ 3 ].normal = D3DXVECTOR3( 1.5f, -1.0f, -1.0f );

g\_BoardKing4[ 4 ].position = D3DXVECTOR3( -0.07f, 2.8f, 0.07f );

g\_BoardKing4[ 4 ].normal = D3DXVECTOR3( -1.5f, 1.0f, 1.0f );

g\_BoardKing4[ 5 ].position = D3DXVECTOR3( 0.4f, 2.8f, 0.1f );

g\_BoardKing4[ 5 ].normal = D3DXVECTOR3( 1.5f, 1.0f, 1.0f );

g\_BoardKing4[ 6 ].position = D3DXVECTOR3( -0.1f, +1.0f, 0.1f );

g\_BoardKing4[ 6 ].normal = D3DXVECTOR3( -1.5f, -1.0f, 1.0f );

g\_BoardKing4[ 7 ].position = D3DXVECTOR3( 0.1f, +1.0f, 0.1f );

g\_BoardKing4[ 7 ].normal = D3DXVECTOR3( 1.5f, -1.0f, 1.0f );

//

// Cell init

//

g\_BoardFloor[ 0 ].position = D3DXVECTOR3( -0.5f, -0.5f, -0.5f );

g\_BoardFloor[ 0 ].normal = D3DXVECTOR3( -1.0f, -1.0f, -1.0f );

g\_BoardFloor[ 1 ].position = D3DXVECTOR3( 0.5f, -0.5f, -0.5f );

g\_BoardFloor[ 1 ].normal = D3DXVECTOR3( 1.0f, -1.0f, -1.0f );

g\_BoardFloor[ 2 ].position = D3DXVECTOR3( -0.5f, -0.5f, 0.5f );

g\_BoardFloor[ 2 ].normal = D3DXVECTOR3( -1.0f, -1.0f, 1.0f );

g\_BoardFloor[ 3 ].position = D3DXVECTOR3( 0.5f, -0.5f, 0.5f );

g\_BoardFloor[ 3 ].normal = D3DXVECTOR3( 1.0f, -1.0f, 1.0f );

g\_Cube[ 0] = 0; g\_Cube[ 1] = 1; g\_Cube[ 2] = 2;

g\_Cube[ 3] = 1; g\_Cube[ 4] = 3; g\_Cube[ 5] = 2;

g\_Cube[ 6] = 1; g\_Cube[ 7] = 5; g\_Cube[ 8] = 3;

g\_Cube[ 9] = 3; g\_Cube[10] = 5; g\_Cube[11] = 7;

g\_Cube[12] = 0; g\_Cube[13] = 5; g\_Cube[14] = 1;

g\_Cube[15] = 0; g\_Cube[16] = 4; g\_Cube[17] = 5;

g\_Cube[18] = 4; g\_Cube[19] = 6; g\_Cube[20] = 7;

g\_Cube[21] = 5; g\_Cube[22] = 4; g\_Cube[23] = 7;

g\_Cube[24] = 6; g\_Cube[25] = 4; g\_Cube[26] = 0;

g\_Cube[27] = 6; g\_Cube[28] = 0; g\_Cube[29] = 2;

g\_Cube[30] = 6; g\_Cube[31] = 2; g\_Cube[32] = 7;

g\_Cube[33] = 7; g\_Cube[34] = 2; g\_Cube[35] = 3;

//

// Set light RENDERSTATE params

//

g\_pDevice->SetRenderState ( D3DRS\_LIGHTING, TRUE );

g\_pDevice->SetRenderState ( D3DRS\_CULLMODE, D3DCULL\_CW );

//

// Create Light

//

D3DLIGHT9 light;

//

// Set light params

//

light.Type = D3DLIGHT\_DIRECTIONAL;

light.Diffuse.r = 0.2f;

light.Diffuse.g = 0.2f;

light.Diffuse.b = 0.2f;

light.Direction.x = 1.0f;

light.Direction.y = 1.0f;

light.Direction.z = 1.0f;

//

// Set Device's params for light

//

g\_pDevice->SetLight( 0, &light );

g\_pDevice->LightEnable( 0, TRUE );

//

// Set View matrix

//

g\_pDevice->SetTransform( D3DTS\_VIEW, &g\_Matrix );

//g\_pDevice->SetFVF( D3DFVF\_XYZ | D3DFVF\_NORMAL );

g\_pDevice->SetFVF( D3D\_CUSTOMFVF );

return true;

}

//

// Cleanup function - here we'll delete all buffers/clean memory, using d3d::Release

//

void Cleanup( )

{

}

//

// Display function - in fact, the most important function in our program, created for draw all objects

//

bool Display( float timeDelta )

{

if ( g\_pDevice ) {

board.Draw( g\_Matrix, g\_pDevice );

}

return true;

}

//

// WndProc

//

LRESULT CALLBACK d3d::WndProc( HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam )

{

switch ( msg )

{

case WM\_DESTROY:

{

::PostQuitMessage( 0 );

break;

}

case WM\_KEYDOWN:

{

switch ( wParam )

{

case VK\_ESCAPE:

{

::DestroyWindow( hWnd );

}

case VK\_F1:

{

if ( board.GetTurnCount( ) % 2 == 0 ) {

::MessageBox(0, L"Black Player wins!", L"EPIC WIN!", 0);

::DestroyWindow( hWnd );

break;

}

::MessageBox(0, L"White Player wins!", L"EPIC WIN!", 0);

::DestroyWindow( hWnd );

break;

}

case VK\_NUMPAD0:

{

Camera.TurnOn( g\_pDevice );

break;

}

case VK\_NUMPAD8:

{

Camera.IncY( );

break;

}

case VK\_NUMPAD5:

{

Camera.DecY( );

break;

}

case VK\_NUMPAD4:

{

Camera.DecX( );

break;

}

case VK\_NUMPAD6:

{

Camera.IncX( );

break;

}

case VK\_NUMPAD7:

{

Camera.DecZ( );

break;

}

case VK\_NUMPAD9:

{

Camera.IncZ( );

break;

}

case VK\_NUMPAD1:

{

Camera.SetOnWhite( );

break;

}

case VK\_NUMPAD2:

{

Camera.SetOnMiddle( );

break;

}

case VK\_NUMPAD3:

{

Camera.SetOnBlack( );

break;

}

case VK\_UP:

{

board.DecX( );

break;

}

case VK\_DOWN:

{

board.IncX( );

break;

}

case VK\_LEFT:

{

board.DecY( );

break;

}

case VK\_RIGHT:

{

board.IncY( );

break;

}

case VK\_SPACE:

{

if ( !board.GetChosenFlag( ) ) {

board.SetChosenX( board.GetCurrentX( ) );

board.SetChosenY( board.GetCurrentY( ) );

board.SetChosenFlag( true );

break;

}

board.SetChosenNextX( board.GetCurrentX( ) );

board.SetChosenNextY( board.GetCurrentY( ) );

board.Move( board.GetChosenX( ), board.GetChosenY( ), board.GetChosenNextX( ), board.GetChosenNextY( ) );

board.SetChosenFlag( false );

break;

}

}

Camera.Set( g\_pDevice );

}

}

return ::DefWindowProc( hWnd, msg, wParam, lParam );

}

//

// WinMain function - in fact, our program

//

int WINAPI WinMain ( HINSTANCE hinstance,

HINSTANCE prevInstance,

PSTR cmdLine,

int showCmd )

{

if ( !d3d::InitD3D( hinstance,

Width, Height, true, D3DDEVTYPE\_HAL, &g\_pDevice ) ) {

::MessageBox( 0, L"InitD3D - FAILED", 0, 0 );

return 0;

}

if ( !Setup( ) ) {

::MessageBox( 0, L"Setup() - FAILED", 0, 0 );

return 0;

}

d3d::EnterMsgLoop( Display );

Cleanup( );

g\_pDevice->Release( );

return 0;

}