|  |
| --- |
| Design and Implement an Original Functioning Turn Based Strategy Game Based on Chess  By James Southall  2012/13  Supervisor 1: Prof Andrew Ware  Supervisor 2: Dr Alex Lohfink |

**Summary**

The report documents the design, implementation and evaluation of an original Chess program. There are many chess programs on the web, but most tend to be standard chess. Therefore to entice more people into playing chess, making an original game with interesting game play techniques in it would be good to see on the web, hence the project. The functionality of the application was implemented within a console application and the graphical application within an OpenGL 3D scene. Software issues had to be considered, from choosing how to store the chess board cells to choosing the appropriate API for the graphical implementation.

The minimum requirements were:

* To research into current chess variant game rules
* To research into current developed chess games
* To research into chess board representation
* To research which API is more suited for the graphical application
* To design and implement an original console based chess program
* To implement special moves
* To implement a graphical demonstration of a chess program

The possible extensions were:

* To fully implement a working graphical chess program
* To implement turn based cool down power ups
* To design and implement unique chess pieces
* To implement animation into the chess pieces

**Acknowledgements**

I would not have been able to complete this project to a reasonable standard if it was not for a few people who deserve a mention.

**Professor Andrew Ware** my project supervisor for the needed push to hit deadlines for the weekly project meetings and general guidance where needed for the progression of my project. I extend this thanks to **Dr Alex Lohfink** for being my 2nd supervisor.

**Dr Paul Angel** for the use of his OpenGL framework within the graphical implementation and of course for his general professionalism when giving lectures over the past two years.

**Dr Ian Wilson** for his expertise over the three years of the degree. Passing on his vast knowledge through lectures and support when needed.

**Family** for providing support over the three years of my degree and throughout the stressful times of this project.

**Friends** for information and assistance when needed for certain aspects. Their friendship is greatly valued and has kept me sane through it all.

Table of Contents

[1 Introduction 6](#_Toc351375046)

[1.1 Aims 6](#_Toc351375047)

[1.2 Objectives 6](#_Toc351375048)

[1.3 Available resources 8](#_Toc351375049)

[1.4 Project Schedule 8](#_Toc351375050)

[2 Research 9](#_Toc351375051)

[2.1 Chess Rules 9](#_Toc351375052)

[2.2 Valid Moves 10](#_Toc351375053)

[2.2.1 Pawns 10](#_Toc351375054)

[2.2.2 Rook 11](#_Toc351375055)

[2.2.3 Bishop 11](#_Toc351375056)

[2.2.4 Queen 11](#_Toc351375057)

[2.2.5 Knight 11](#_Toc351375058)

[2.2.6 King 12](#_Toc351375059)

[2.4 Chess Board 13](#_Toc351375060)

[2.5 Chess Variants 14](#_Toc351375061)

[2.5.1 Eurasian Chess 14](#_Toc351375062)

[2.5.2 Grand Chess 16](#_Toc351375063)

[2.5.3 Eight-Stone Chess 17](#_Toc351375064)

[2.6 Current Chess Games 18](#_Toc351375065)

[2.6.1 Battle vs. Chess 18](#_Toc351375066)

[2.7 Graphical API 22](#_Toc351375067)

[2.7.1 Open GL 22](#_Toc351375068)

[2.7.2 DirectX (With focus on Direct3D) 23](#_Toc351375069)

[2.8 Chess Programming 25](#_Toc351375070)

[2.8.1 Board Representation 25](#_Toc351375071)

[3 Design 27](#_Toc351375072)

[3.1 UML Class Diagram 27](#_Toc351375073)

[3.2 Game Design Document 28](#_Toc351375074)

[4 Implementation 32](#_Toc351375075)

[4.1 Functionality 32](#_Toc351375076)

[4.1.1 Chess Class (BChess) 32](#_Toc351375077)

[4.1.2 Board Class (BBoard) 35](#_Toc351375078)

[4.1.3 Piece Class (BPiece) 37](#_Toc351375079)

[4.1.4 Individual Piece Classes 39](#_Toc351375080)

[4.2 Graphics 52](#_Toc351375081)

[5 Testing 53](#_Toc351375082)

[6 Evaluation 61](#_Toc351375083)

[6.1 Meeting Requirements 61](#_Toc351375084)

[6.2 Future Developments 64](#_Toc351375085)

[7 References 66](#_Toc351375086)

[8 Appendices 68](#_Toc351375087)

[8.1 Appendix 1 –UML Diagram 68](#_Toc351375088)

[8.2 Appendix 2 - LSEPI 69](#_Toc351375089)

[8.3 Appendix 3 81](#_Toc351375090)

[8.4 Appendix 4 83](#_Toc351375091)

[8.5 Appendix 5 85](#_Toc351375092)

# 1 Introduction

## 1.1 Aims

The aim was to design and implement a fully functional turn based strategy game based on the well known two player board game: Chess. The secondary aim was to make an original chess game by incorporating different pieces and special moves. A graphical version of the game will also be implemented.

## 1.2 Objectives

LSEPI

Objective 1: Intellectual Property and Copyright – Investigation into the legal issues of using open source material relevant to the project specification, as algorithms were looked at for the implemented for certain aspects of the project; storing valid moves, animation etc.

Objective 2: Ethics – The project was to include violent animations when taking pieces so an investigation was conducted into the ethical issues that can arise when developing a game with violence.

To meet the aims set, certain objectives will need to be made and complete during the timescale:

Research

As the game was to be a different style of chess game, a variety of chess games were researched to decide whether any aspects of them games can be utilised for the game. Special moves for the original pieces were to be implemented into the game to change up the play style compared to that of normal chess.

The research on chess was to be extensive as it is the main base of the project. Other chess games that have been made were researched to gain knowledge on the algorithms used for the functionality of the game such as how the pieces themselves know what moves are valid and which ones are invalid.

Research was to be conducted on OpenGL or Direct3D to decide which API would be the appropriate one to use to implement the graphical application.

The techniques that were to be made use of within the graphical implementation are the use of shaders, VAOs, VBOs and model loading, rendering and texturing.

For the models 3DS Max will be used to develop custom original models which will be textured and ready to be loaded and used with the game.

Design

The design was to feature a game design document that will be an in depth plan for the game. Graphical designs for the pieces, board and UI will be drawn out to give a rough guide on how the pieces will be modelled. Based on research the method for storing the board will be outlined with the layout of the pieces on the board.

Implementation

The main implementation was to be to develop a console application that will demonstrate the full functionality of the game. That would include algorithms for all the pieces. The implementing of the method used to store the board and pieces.

Secondary implementation is to develop a graphical application that will show the board and the pieces loaded onto a 3D scene.

Evaluation

The evaluation contains a detailed outline on how the minimum requirements were met and if they were not met what the reasoning for that was. If any extensions were completed this will also be documented.

Future enhancements were discussed within the evaluation. This could include functionality improvements or full features that could be added to the application/game.

## 1.3 Available resources

To make use of shaders which is included in higher versions of the APIs (OpenGL 3.3 is required for shader use) and also be able to use programs such as 3DS Max without having performance issues computers with reasonable hardware would be used. A detail specification of the hardware is:

* Processor – Intel i5 2500k Quad core
* Memory – 8GB DDR3 RAM
* Graphics – nVidia GTX670 2GB dedicated memory
* Harddrive – 120GB OCZ Agility SSD
* Monitor(s) – 2 x 24inch LCD.

This machine is capable of running 3DS max without any performance issue. It also has a very quick compile time when building programs on Microsoft Visual Studio 2010 due to the use of a solid state drive.

## 1.4 Project Schedule

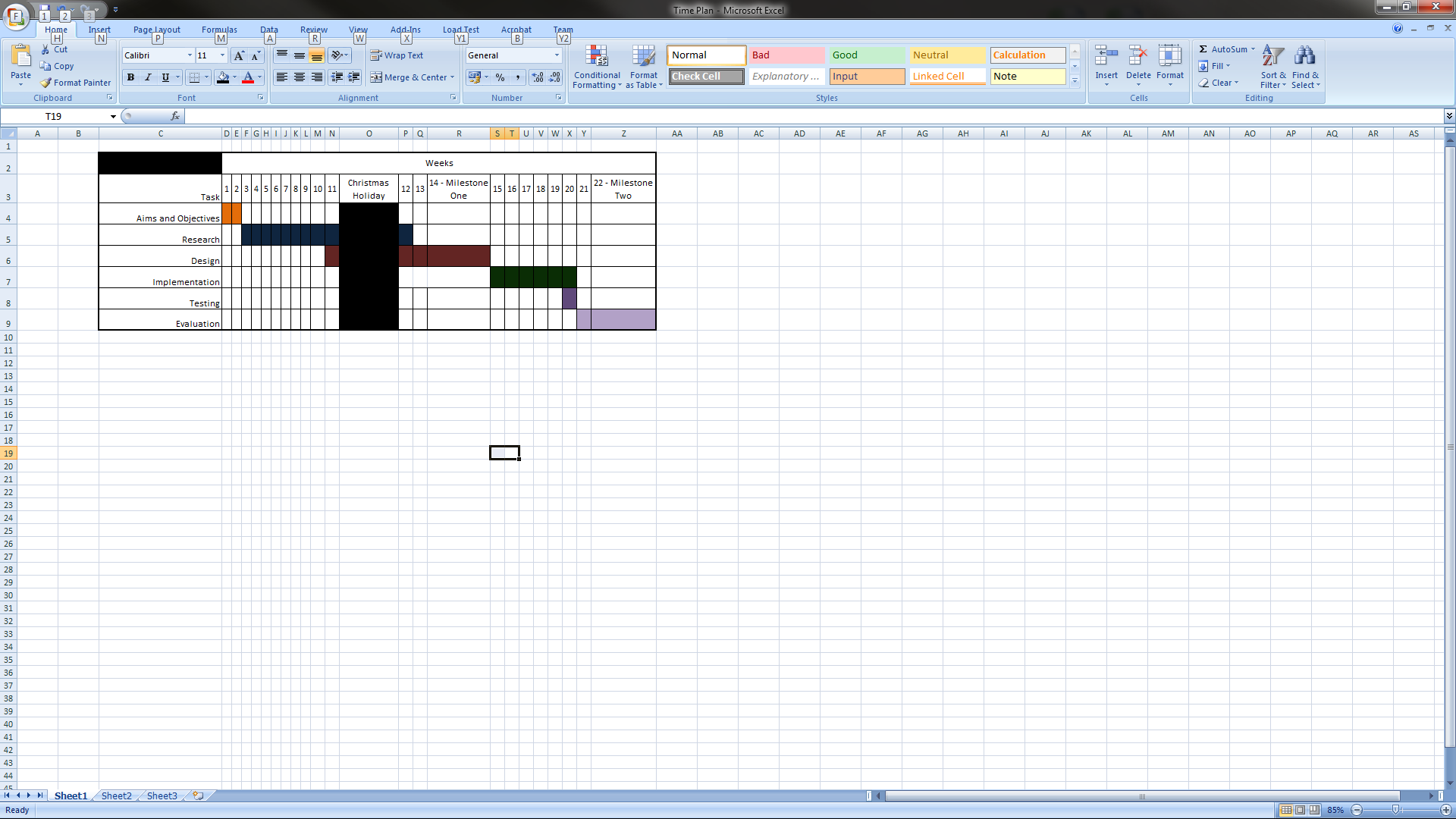
The Gantt chart below illustrates the predicted timescale for the tasks to be completed for the project. Due to being predicted it is inaccurate and could have change at any stage up to the deadline of the project.

Figure i - Gantt Chart of Project Schedule

A great deal of time is expected to be spent on research and design as these sections are believed to be the most important stages within the project. The research will cover background reading on algorithms, graphical techniques and which software and API to use for certain tasks.

Despite testing being given a schedule it will be ongoing throughout the implementation of the application to make sure functions are being developed correctly. At the end most testing will be conducted to make sure it works as intended.

# 2 Research

## 2.1 Chess Rules

A standard chess game is a two player board game. The chessboard is made up of 64 squares arranged by eight rows by eight columns. Each player starts with sixteen pieces on two rows opposite their opponent. The pieces consist of;

* eight pawns
* two rooks/castles
* two bishops
* two knights/horses
* one queen
* one king

The objective of the game is to get the opponents King into a situation it cannot get out of, known as Checkmate.

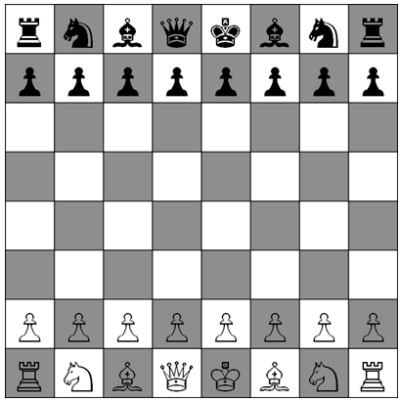


Figure ii - Standard Chess Board

## 2.2 Valid Moves

Each different type of chess piece has its own set of valid moves. In which it can be used in them ways to attack and take opponents pieces. These rules will be fully analysed to gain knowledge to use when developing the algorithms to store rules.

### Diagram showing how a pawn moves on the chess board.2.2.1 Pawns

Figure iii - Pawn Movement

Pawns are very limited pieces, but have some very unique and complex rules;

* Only able to move 1 space forward at a time (Not allowed to move backwards)
* Only able to take opponent pieces by moving in a diagonal space forward
* Can move 2 spaces forward if it’s in its original starting square

*Taking en-Passant* – This is a special rule which allows a pawn to take an opponent’s pawn in a certain circumstance. If a player does a double step with its pawn from second row to fourth row, and there is an enemy pawn on an adjacent space on the fourth row, then the enemy pawn is able to move diagonally to the space that was passed over during the pawns double step on the third row. The double stepping pawn is then taken within this move.

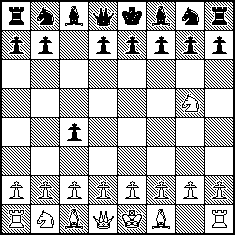
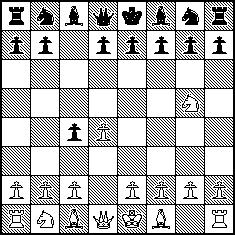
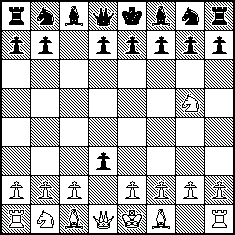


Figure iv - En Passant three

Figure v - En passant two

Figure vi - En passant one

*Promotion* – When pawns eventually reach the last row of the board they are promoted. The played is able to choose a; queen, rook, knight or bishop to replace the pawn. Most commonly a queen is select, but that does not rule out the other pieces as viable replacements. The player is not required to choose a piece that has been taken, thus allowing a single player to have two or more queens at one time.

### Diagram showing how a rook moves on the chess board.2.2.2 Rook

* Move in a straight line horizontally or vertically
* May move an unlimited amount of spaces as long as it’s a valid move
* May not jump over pieces

Figure vii - Rook Movement

### Diagram showing how a bishop moves on the chess board.2.2.3 Bishop

* Move straight in a diagonal line
* May move an unlimited amount of spaces as long as it’s a valid move
* May not jump over pieces

Figure viii - Bishop Movement

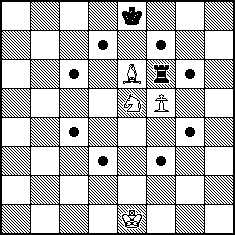
### Diagram showing how a queen moves on the chess board.2.2.4 Queen

* Has the combined moves of the rook and the bishop (Able to move straight; horizontal, vertical or diagonal)
* May move an unlimited amount of spaces as long as it’s a valid move

Figure ix - Queen Movement

* May not jump over pieces

### 2.2.5 Knight

The Knight (also known as the horse) has a unique way of moving:

* Only piece in chess that is able to jump over other pieces (both enemy and player pieces)
* A knights move starts with one step in a horizontal or vertical direction followed by one step in a diagonal outward direction

Figure x - Knight Movement

### Diagram showing how a King moves.2.2.6 King

* The king is the most important piece of the game; moves must be made to protect the king from the enemy moving in and placing it in a position to be taken.
* Able to move in ANY direction; horizontally vertically or diagonally by only 1 space.

Figure xi - King Movement

*Castling* – Providing neither the king nor a chosen rook has been moved then the player is able to complete the special move known as castling. This is where the king moves two spaces towards the rook and the rook moves to the space that the king has just past.

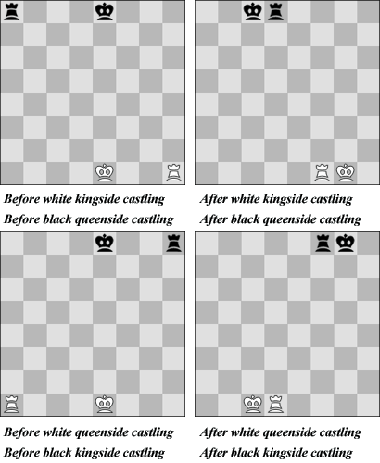
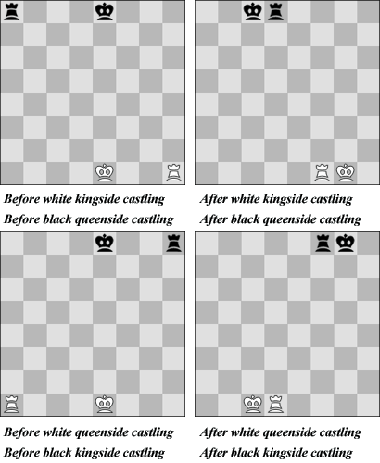


Figure xii - Castling Two

Figure xiii - Castling One

Check, Mate, and Stalemate

*Check* – When a player moves one of their pieces into a position where they can take the opponent’s king within the next move then the situation is known as ‘Check’. Once in check the player is not allowed to make any moves if it leaves the king still in check after. A player can get out of check via three different ways:

* Moving the king out of check
* Taking the piece that has the king in check
* Or moving another piece into the path of the enemy piece and the king

*Checkmate* – When a player is in check and they cannot make a move to get out of it then this is known as checkmate and the game has been won.

*Stalemate* – When a piece is not in check but cannot make a legal move then they are stalemated and the game is ended as a draw.

## 2.4 Chess Board

The layout of the chess is fairly simple: 64 squares laid out in an 8 by 8 grid. Chessboards are commonly labelled A-H for the columns or Files as more commonly known as in Chess, and 1-8 for the rows which are known as Ranks in Chess. Rank 1 and 2 are the Whites starting ranks and 7 and 8 are the Blacks starting ranks. The Queen for both colours starts in the D file and the first square (A1) should be a black square. [1][2][3]

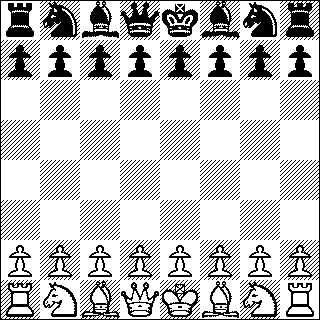


Figure xiv - Chess Board

## 2.5 Chess Variants

Bellum is designed to be unique from any other chess game. People may question ‘any other chess game’ because not many people know that there is more than one different type of chess game. Below a variety of different style of chess games will be discussed and analysed on whether or not the rules they use would be useful to use within Bellum.

### 2.5.1 Eurasian Chess

This chess variant by Fergus Duniho began with another chess variant called Yáng Qi also by Fergus which does not have a queen and is on a 10 x 10 board.

Figure xv - Eurasian Chess Board

*Setup*

Eurasian chess is on a 10 x 10 board where each player has 22 pieces. A noticeable addition to the game besides the bigger board and more pieces is the river in the middle of the board. Kings are restricted to their side of the board.

*Pieces*

http://www.chessvariants.org/graphics.dir/eurasian/BKing.gifKing – Just like in standard chess the king is limited to moving just one space in any direction. The objective of the game still stands at having to place the king in checkmate. Each players king like mentioned above is limited to just their side of the board/river. As the layout of the pieces is different there is no castling. Kings are also unable to face each other across an empty diagonal or vertical line.

http://www.chessvariants.org/graphics.dir/eurasian/BQueen.gif

Queen – Moves exactly like queen in standard chess. No difference what so ever.

http://www.chessvariants.org/graphics.dir/eurasian/BKnight.gifKnight/Horse – Uses the same play style as standard chess with also being the only piece that is able to jump over other pieces.

http://www.chessvariants.org/graphics.dir/eurasian/BRook.gifRook/Chariot – Standard rook play style moving in vertical and horizontal spaces for as many spaces.

http://www.chessvariants.org/graphics.dir/eurasian/BBishop.gif

Bishop/Sage – Moving for as many spaces as possible in a diagonal movement just like standard chess.

http://www.chessvariants.org/graphics.dir/eurasian/BCannon.gifCannon – First of the unique piece which is Asia’s contribution to Eurasian Chess. This is the Cannon from Chinese Chess, not to be confused with the one from Korean Chess. It uses the same movement style as the rook, expect the fact that it must jump over a piece for it to take a piece, similar to how pieces are taken in Checkers.

http://www.chessvariants.org/graphics.dir/eurasian/BVao.gifVao/Arrow – The Arrow is the counterpart of the Cannon as is the Bishop is the counterpart of the Rook. It moves in a diagonal style but has to jump over a piece to take it.

http://www.chessvariants.org/graphics.dir/eurasian/BPawn.gifPawn – Using the same style movement as a standard chess pawn, this includes moving 2 spaces on its initial move. The special move of ‘Taking en Passant’ is still used. Pawns can still get promoted but only into a piece that has already been taken, but can promote when in the last 3 rows of the board on the enemies side of the board. A pawn is not obligated to promote until the last row. They may not advance to the final row if there are no captured pieces to promote to but they are still able to place pieces in check that are on the last row [4][5][6].

Evaluation

Bellum will be designed to include addition pieces with special ways of movement so the Cannon and Vao/Arrow are good examples to consider in the implementation of the special pieces. Another unique aspect of Eurasian chess that would good to consider is the river which prevents the King from leaving their side.

### 2.5.2 Grand Chess

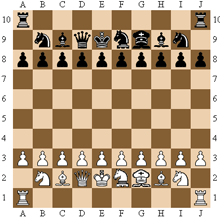
Created by Dutch game designer Christian Freeling. Just like Eurasian chess it is played on a 10 x 10 board and makes use of two different pieces; Marshall and Cardinal.

Figure xvi - Grand Chess Board

* The Marshall located on the board at F2 and F9 has the combined moves of a Rook and Knight.
* The Cardinal located at G2 and G9 has the combined moves of a Bishop and a Knight.

Grand chess follows the standard rules of Chess besides:

* Castling is not possible.
* Pawns can optionally be promoted when reaching the 8th or 9th row, but must promote on the final row.
* Pawns can only promote to already taken pieces.

[6][7]

Evaluation

Similar to the extra pieces in Eurasian Chess, it would be good to consider the movement style of the Marshall and Cardinal of Grand Chess. Also the special new rules of the pawn could be interesting to implement.

### 2.5.3 Eight-Stone Chess

This Chess variant was developed by Jim Aikin. It has all the usual Chess pieces and rules but has the addition of 8 stones placed on the board and is played on a 8x9 board so that the stones can be places in a square that is an equal distance away from each players side.

After moving a piece the player may then optionally move a stone. The stones are neutral pieces and may be moved by either player. Stones cannot capture pieces, nor can they be captured.

Stone moves:

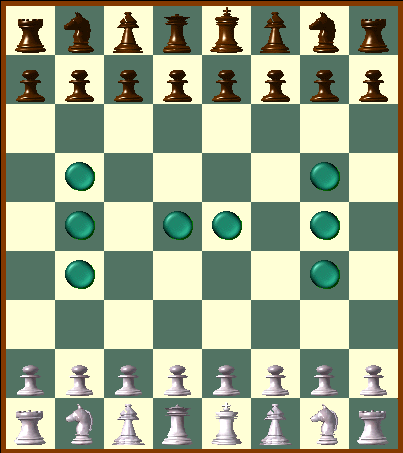
* A stone may be moved vertically or horizontally by a single square.
* If a friendly piece occupies a square vertically or horizontally adjacent to a stone, the stone may swap places with the piece.

Figure xvii - Eight Stone Chess Board

* When a group of two or more stones are arranged in an unbroken vertical or horizontal line with a vacant square at either end of the line, a stone at one end can be picked up and moved to the other end. The line must contain just stones, no pieces.
* Special rule: A stone that was previously moved by the opponent may not be moved back to its previous position.

[6][8]

Evaluation

The use of the stones is a very unique and game changing element which would be interesting if included into Bellum. Something similar may be considered as an extra.

## 2.6 Current Chess Games

### 2.6.1 Battle vs. Chess

“Battle vs. Chess is based on the simple yet ingenious idea of creating a fusion between professional chess software and upbeat, combat oriented game mechanics, complete with richly animated game pieces.”



Figure xviii - Battle vs. Chess Starting

Battle vs. Chess is a very recent game developed by Targem Games and Zuxxez Entertainment and published by TopWare Interactive for PC, Xbox 360, Playstation 3, Wii, DS and PSP. It is complete with a variety of single player game modes and also multiplayer which can be done cross-platform with 7th generation consoles (Xbox 360 and PS3) and PC. The two sides are Angels and Demons, with unique killing moves for each piece/side.



Figure xix - Battle vs. Chess Animation

Game Modes

For the single player there are 5 main game modes to choose from:

1. Skirmish – Standard chess rules/playstyle.
2. Battlegrounds – Standard chess rules with a twist, when a player selects to take an opponent’s piece they are engaged into a 1v1 battle. (See further on for information)
3. Tutorial – Helps players understand the rules of chess. Allows the player to choose from what they would like to learn, whether it’s a particular piece, castling or basic general rules.
4. Campaign – The player will complete missions just like a normal game of chess but the opponent will have a different amount of pieces or the pieces will be arranged differently and the goal in each mission is to get the opponent checkmated.
5. Mini Games – This includes a massive amount challenges ranging from standard to hard.



Figure xx - Battle vs. Chess Main Menu

Scenery

Before going into a game the player is able to choose from a few battlegrounds from a standard chess board to a fire chasm of doom.

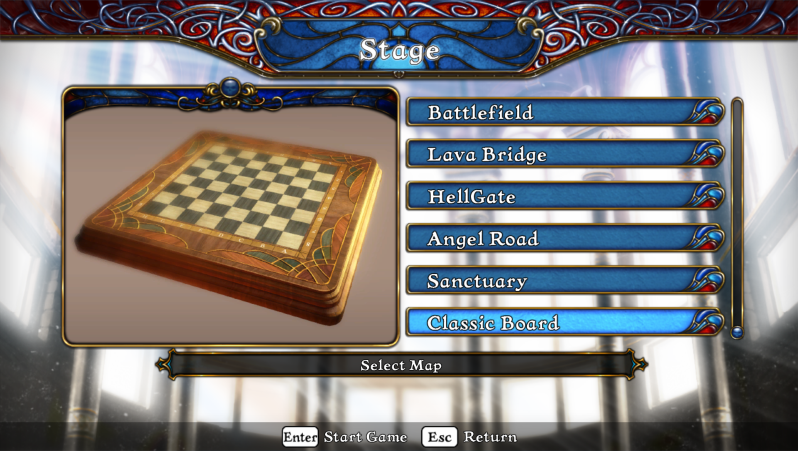


Figure xxi - Battle vs. Chess Board Selection

User Interface

The UI within Battle vs. Chess has player names displayed at the top and on either side the player’s moves that have been made and which pieces have been taken.



Figure xxii - Battle vs. Chess UI

One v One

When any player selects a move in which they wish to take an opponent’s piece they engage in a one on one battle with that opponent’s piece. The player is then require to input a sequence of arrow key inputs that displays on the screen to be able to attack the opponent, if they player fails the sequence then the opponent attacks the players piece. Depending on how valuable the piece is will depend on how many health that piece will have: Pawn will have the lowest health and the king with have the highest.



Figure xxiii - Battle vs. Chess 1v1

Figure xxiv - Battle vs. Chess 1v1 Win

Conclusion

Overall Battle vs. Chess is a very solid chess game; the animations are stunning and the varieties of styles of chess that can be played are great. In terms of the project the aim will be to have a very similar UI as its basic yet effective, the choice of scenery is a great idea also. As for the piece style, a different style will be approached to make it unique to other games. The one on one aspect of the game is a great feature. Bellum will feature less violent/graphical animations to this as this is a PEGI rating for 12 years old and above where as Bellums’ targeted audience are all ages.

[9]

## 2.7 Graphical API

### 2.7.1 Open GL

*Overview*

Silicon Graphics sold high-performance graphics terminals using a proprietary API called Iris GL (Graphics Library) back in 1982. Over the years Iris GL became hard to maintain, up until Silicon Graphics took a new step: they completely restructured Iris GL and made it an open standard. This allowed competitors to use it, but in return, they were required to help keep it up to date.

Figure xxv - OpenGL Logo

Nowadays Open GL is managed by the Khronos Group, who are a non-profit organisation with representatives from many companies that are interested in maintaining high-quality media APIs.

*Advantages*

More powerful than DirectX –OpenGL has faster draw calls than DirectX, this can be shown with a test completed by Valve for Left 4 Dead 2, where OpenGL outperforms DirectX by 40fps more [10]. Tessellation technology that Microsoft promotes for DX11 has been an extension for OpenGL for three years.

Cross Platform – OpenGL is used on every platform except the Xbox (a Microsoft product), this ranges from PC operating system to smart phones. Major game developers claim that supporting Mac and Linux is a waste of time yet Blizzard releases Mac versions of their games simultaneously, and they are one of the most successful game companies in the world.

Features – OpenGL has a wide range of features, both in core and through extensions. It’s extensions allow it to be constantly up to date with new hardware features. It’s features represent a wide range of interests making it useful for many different applications.

Understandable Syntax – Compared to Direct3D the syntax for OpenGL (written in C) is a lot more understandable and easier to grasp.

*Disadvantages*

Extensions – Although they were mentioned in the advantages they can also be classed as a disadvantage due to them making the code messy despite being powerful. Also many newer extensions are card or vendor specific, so may not be used commercially.

Function Naming – Not a big weakness but can seem a bit unnecessary at times. Most IDEs have context sensitive help which will display the required parameters, and in most cases if you know a function then the parameters needed should be known. Also having 12 different names for a function would seem strange to C++ programmer used to function overloading, but OpenGL was written in C which does not support function overloading.

[11][12]

### 2.7.2 DirectX (With focus on Direct3D)

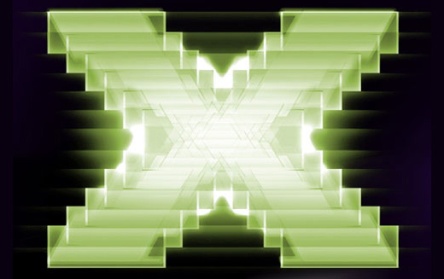
*Overview*

Figure xxvi - DirectX Logo

In 1995 Microsoft create a set of libraries to encourage exclusive games to their new Windows 95 OS as since MS-DOS they understood that games play a big role in user’s choice of OS. These libraries included Direct3D, DirectInput and DirectSound and all came together to be known as DirectX.

Microsoft then entered the gaming Market with the Xbox (DirectX Box) in 2001. This lost the company over 4 billion dollars but was intended to set the stage to dominate the game market in the next generation. Now looking at the current games scene it is clear that this strategy is succeeding. A vast majority of the major PC games at the moment use DirectX, and run on PC and Xbox360.

*Advantages*

OO – DirectXs interface is great for object oriented programming.

Improved Syntax – Over the years the syntax for DirectX has been greatly improved with developers saying that it is more intuitive than previous versions.

Shaders – This is Direct3Ds major strength. Back in DirectX 8 Programmable Pixel and Vertex Shaders were introduced which allow the user to replace parts of the rendering pipeline with custom code to make it more efficient. OpenGL supports shaders but only through extensions not as a standard.

*Disadvantages*

Yearly Release – DirectX is only updated every year or so, which is slow considering the pace of which the graphics industry moves.

More Code – To initialise DirectX requires a lot of code. This has been reduced by a considerable amount over the years. In Direct3D8 it takes 200 lines to draw your first triangle where-as in Direct3D7 it required 800 lines.

Not Open – DirectX is not an open standard. This means that only Microsoft can decide what gets included in a release, and if a bad decision is made then it is stuck in that release for the year.

[11][12]

Conclusion

The chosen API that will be used is OpenGL. DirectX has many useful things as it is a whole package including Direct3D as well as DirectInput and DirectSound. Despite it’s uses OpenGL has much more reasons for being the more suitable API; Open Standard, Understandable Syntax etc.

## 2.8 Chess Programming

### 2.8.1 Board Representation

The most important aspect of any computer chess program is the internal board representation to maintain chess positions for its search, evaluation and game-play. Other than its use to store piece placement, additional information is needed to fully specify a cess position, such as side to move, castling rights, possible en passant target square and to keep track of other things for other implemented rules.

Choosing the appropriate data structure to represent the board and its piece placement creates the foundation for a chess game. There are different representation solutions; piece centric and square centric.

*Piece Centric*

Piece centric representation keeps lists, arrays or sets of all pieces still on the board with the relevant information on which square they occupy.

Piece-Lists – These are lists or arrays of all 32 pieces on the board. The type and colour of the pieces are associated by a certain index range, disjoint lists or arrays. Each element of the list or array represents each piece on a square. [13].

Bitboards – Also known as bitsets or bitmaps, are used elsewhere besides being able to represent the board inside a chess program. Bitboards are finite sets of up to 64 elements (each square of a chessboard), a bit per square.

To represent the board one bitboard for each piece-type and colour will be needed and more often than not will be ecapsulated inside a class or structure, or as an array of bitboards as part of a position object. [14][15].

*Square Centric*

Where piece centric keeps information on pieces that are still on the board, square centric keeps information on each square of the board: Is the square empty or is it occupied?

8 by 8 – This can be represented in two different ways: either a two-dimensional array of integers, containing the relevant information on pieces or empty squares, indexed by file and rank index or a one-dimensional array which is the more common way of doing it. [16]

0x88 – A standard chess board has 64 squares or 0-63 in elements within an array. A1 would be 0, B3 would be 10 and so on. 0x88 uses a different board representation style of using a 16x8 board, 128 squares. The usual 8x8 board with a ‘dummy’ chess board to the right of it, A1 would still be 0 but B1 would be 16.

The formula for this would be**: index = rank \* 16 + file**

The critical reason why this is done is so that it can detect if an iterated ray traversal has reach the side edges of the board.

Now to explain that in more clearer terms: Basically detecting errors going up and down the file of the board is simple for the system, if a rook is at A1 then it’s index is 0 if it goes down it goes to index -8 or if it goes all the way and beyond its index would be 72, if an error checking bit of code is implemented such as: **if ((index < 0) || (index >= 64))** then errors would occur in both times. Whereas with going left and right along the rank there’s issues with error checking, if using a one-dimensional array to go right the rank would need to incremented by one, when the rook is in the final file ‘H0’ for example it would be at index 7. Now if it is incremented again it would go to index 8 which is A1 which is still on the board and this is where the 0x88 representation comes to the rescue.

If a piece steps into the dummy board from either side the error will be detected and a message can be displayed saying that it was an invalid move. [14][16][17]

# 3 Design

## 3.1 UML Class Diagram

Chess Class – This will be the central point class, which will call other classes when needed such as graphics, sound and more importantly the board class. This class will take in user input for piece movement. There will also be a function that will end the game if checkmate has occurred.

Board Class – When this class initialises it should place all the pieces in their correct positions ready for the start of the game. The board representation will be stored within this class. This class will have a function that when called will update the positions on the board. Another function will check after every move to see if a piece is in check.

Graphics Class – This class will load in models and textures. Separate functions will setup all the necessary graphical techniques up.

Piece Class – This will be an abstract class that all the other pieces will inherit from. The individual pieces will inherit piece colours and general legal moving rules (size of the board).

Individual Piece Classes – Each piece will hold its own valid moves and special rules if they hold them such as the special pieces, pawns and king.

For UML diagram see Appendix 1.

## 3.2 Game Design Document

Section I - Game Overview

* 1. Genre – Turn based strategy game.
  2. Target Audience – The target audience will be for all ages. Violence will be minimal.
  3. Project Scope
     1. Number of pieces – 20 pieces per player (2 players means 40 pieces on the battlefield). Each player will have the following pieces; King, Queen, two Bishops, 2 Knights, 2 Rooks, 8 Pawns, 2 Super Pawns and 2 Guards.

1. Section II - Gameplay and Mechanics
   1. Gameplay
      1. Objectives – The objective of the game is to place the opponent’s king in check.
      2. Play Flow – Being a turn based strategy game, the players will take it in turns to make a choice on what piece to move. So that the game doesn’t last more than a suitable time, each player will have 20seconds to make a move, else they lose their turn.
   2. Mechanics
      1. Movement – All the pieces will be in a class to which the game will call a certain function related to the piece that’s moving which will store the valid moves for that piece. The game class will then determine whether the piece will collide with another piece in a way that the opponent’s piece is taken. The board class will then be updated with the positions for each piece after every turn, which will be stored in multidimensional array of size 10 x 10.
         1. Pawn – Moves two spaces forward on the initial move. One space forward if the pawn has already been moved. Not able to jump over pieces. Can only take pieces diagonally one space ahead. Once the pawn gets to the other end of the board the player gets to choose a piece to exchange it for, the special pieces will not be a valid choice.
         2. Knight – Has two phases to their movement; Firstly they move one space in horizontal or vertical direction, secondly they move in a vertical outward direction. Also able to jump over other pieces.
         3. Bishop – Moves as many spaces as possible in a diagonal movement. Not able to jump over pieces.
         4. Rook – Moves as many spaces as possible in a vertical or horizontal movement. Not able to jump over other pieces.
         5. Queen –Has the combination of the Rook and Bishop movements and also not able to jump over pieces.
         6. King – Can move in any direction by just one space. Unable to jump over pieces.
         7. Super Pawn – Able to move in within a two block radius.
         8. Guard – Has a combination of the knight and bishops moves
      2. Other Movement
         1. Taking En-Passant – This is a special rule which allows a pawn to take an opponent’s pawn in a certain circumstance. If a player does a double step with its pawn from second row to fourth row, and there is an enemy pawn on an adjacent space on the fourth row, then the enemy pawn is able to move diagonally to the space that was passed over during the pawns double step on the third row. The double stepping pawn is then taken within this move.
         2. Castling – Providing neither the king nor a chosen rook has been moved then the player is able to complete the special move known as castling. This is where the king moves two spaces towards the rook and the rook moves to the space that the king has just past.
         3. Super Pawn Swap – Allows the player to sacrifice their super pawn to allow another piece to move to its space.
2. Section IV - Interface
   1. Visual System – As the main focus besides the functionality for Bellum is graphics then the aim is make all aspects visually pleasing for the user.
      1. HUD – The only thing that the HUD should display is a timer along with a border.
      2. Menus – The menu will display four buttons; Start Game, Load Game, Help and Exit.
         1. Main Menu Screen – From the main menu the user will be able to access the help screen and start up a new game.
         2. Help Screen – The help screen will be a go to screen for mainly new players for controls, piece movement and other general information/help.
         3. Replaying and Saving – If a game is taking a little too long then the user will have the choice to save the game. The game board array will be sent to a file and stored for when it is chosen to be loaded.
      3. Rendering System – The game will be rendered using the OpenGL graphical API. Reasoning for this is due to the better understanding for the OpenGL language compared to that of the DirectX language and from what research had gathered it appears to be the most powerful API.
      4. Camera – A choice of two cameras will be available a fixed top down view and free moving camera which will continue to look at the centre of the board but can be moved around and has the option to zoom in.
      5. Lighting Models – One light will be used and conveniently placed in the centre of the game area.
   2. Control System – The game will be controlled via keyboard input where the user will be required to input grid co-ordinates (G2 for example) then select a place to put that piece by inputting the desired grid co-ordinates.
   3. Audio - Some voice acting will be done for certain points within the game; Start and end of a game, sometimes when a piece is taken and when the king is in check.
   4. Music - Appropriate music will be used for general background music, when a piece takes another piece, when the player wins or loses the game.
   5. Sound Effects – Sounds effects will be used for menu selection, when a piece moves and a combat noise when a piece is taken.
   6. Help System –A help screen will be available at the main menu which will give assistance on how to control the pieces.
3. Section V – Technical
   1. Software – Different programs will be used for the development of Bellum, the programs and what they will be used for are detailed below.
      1. Programming – Microsoft Visual Studio will be used to program the game. Bellum will be programmed in C++ using the graphical API OpenGL for the graphics.

As a starting point OpenGL code provided by Dr Paul Angel of the University of Glamorgan will be used, as it contains all the useful functions such as texture loading, model loading and mathematical equations

* + 1. 3D Modelling – The models for the pieces will be developed within Autodesk 3D Studio Max.
    2. Texturing – Adobe Photoshop will be the image editor of choice for creating textures.

1. Section VI – Game Art
   1. Pieces
      1. Guardians – For the Guardians have been drawn out to be customised standard Chess pieces with a customised mask and weapons.
      2. Demons – For the Demons have been drawn out to be customised standard pieces with horns of all sorts and weapons.
   2. Board – Similar to the standard Chess board just bigger.
2. Section VII - Art
   1. Textures
      1. Pieces – The pieces will be given a marble style texture with a custom texture for mask or horns.
      2. Board – The board will have a stone style texture.
      3. Menu – A background texture along with textures for the buttons will be used.
      4. Scenery – An assortment of landscape textures will be used for the locations.

# 4 Implementation

## 4.1 Functionality

The functionality of the game is primarily being demonstrated in the form of a console base application. The board is outputted to the screen and movements are completed via text input. The game is fully Object Orientated to provide a more clear structure and allows it to be easily maintained and modified if needed.

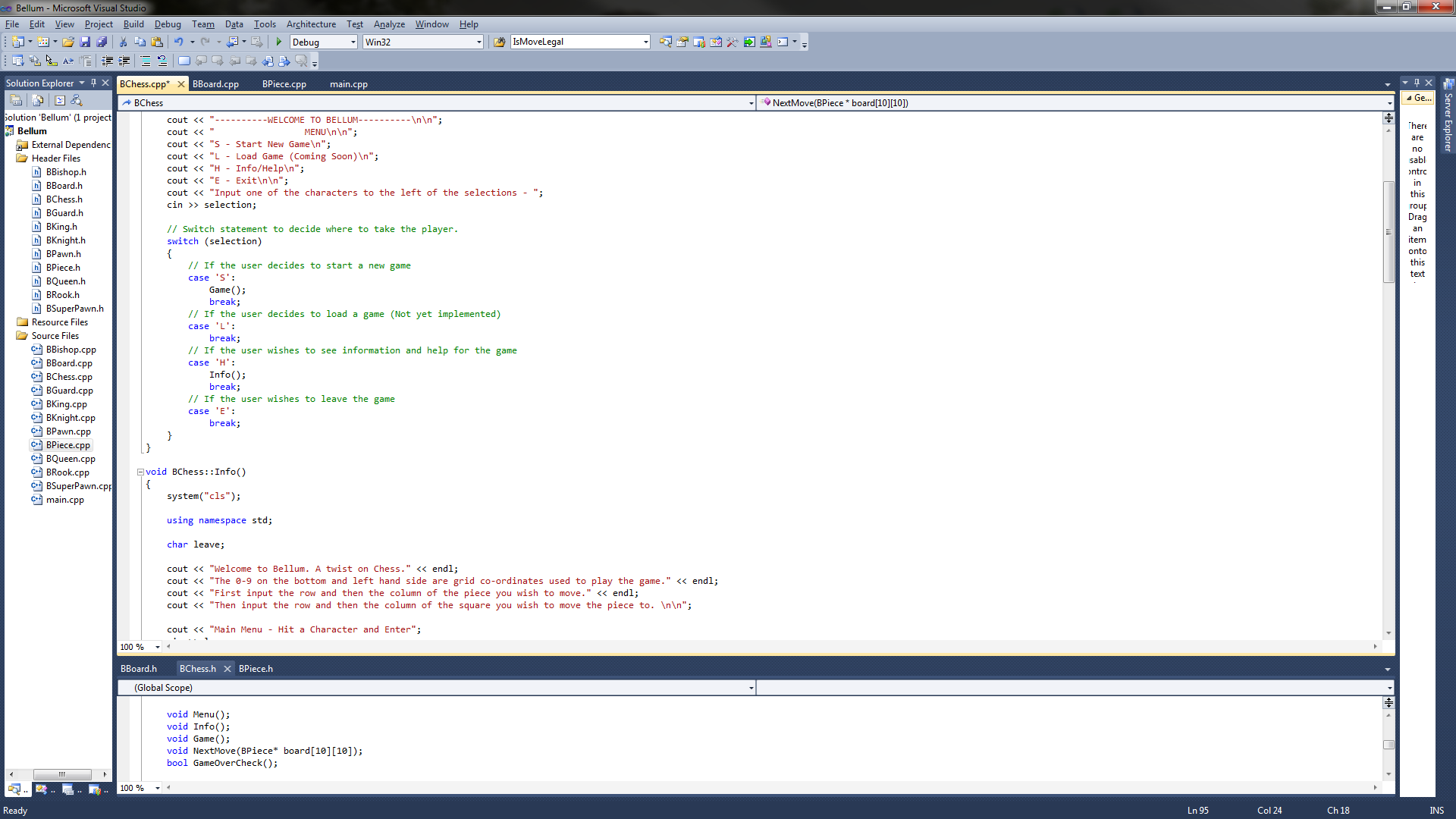
### 4.1.1 Chess Class (BChess)

This is the game control class where the important functions from the other classes are called as well as having its own important function members. BChess has access to the BBoard class and the BPiece class through #include’s. BChess contains nine members: two data members (playerTurn of type char which contains the character that denotes whether it’s the Guardians or Demons turn, and an object of the BBoard class called bellumBoard used to gain access to members of the BBoard class) and four function members, each will be given a detailed description of below:

BChess (Constructor) - Initialises the playerTurn data member with a ‘G’ which denotes the Guardian faction as they start first.

~BChess (Destructor) – A default destructor which is called when the class object is destroyed.

*(void)Main –* This is the function that will be called when the application is run. From here the user will be able to decide whether to start a new game, load a previous played game, display information/help for the game or to leave the application. The user inputs a character linked to an option which is then put into a switch statement to take the user to the chosen option (see below).

**

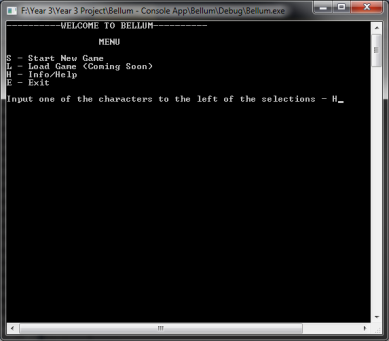
**

Figure xxvii - Menu Switch Statement

Figure xxviii - Menu

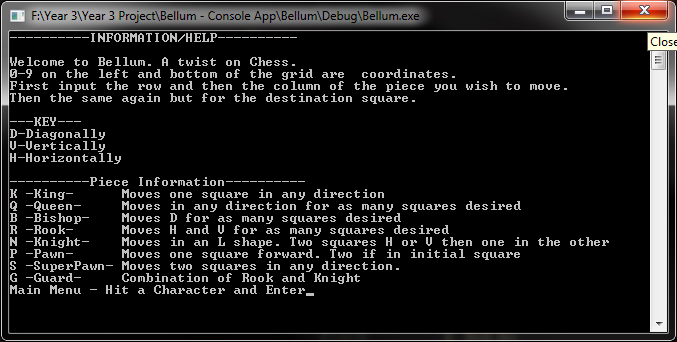
*(void)Info –* Information and help on how to play this game is outputted via this function. At the users discretion they can go back to the main menu.

Figure xxix - Info Screen

*(void)Game –* A do while loop is used to keep the game running, GameOverCheck a function member within the class is called to decide whether or not the game is over. Each time the do while loops it calls a function for the next move and changes the player. To alternate between the players turns instead of using a basic if-else statement, the conditional operator ‘?’ was used.

Figure xxx - Game Loop

*(void)NextMove –* NextMove is the only function member that has a parameter passed through and that is a pointer to the board. This function member is called in the Game functions do while loop. It will enter a while loop which will continue until a local variable (bool)validMove has been set to true upon a valid move being a completed. First it will output the board by accessing a BBoard function member via the object bellumBoard. Next it will ask the user to input coordinates of the piece they wish to move and the coordinates of the destination square. The coordinates will represent the row and column on the grid, the program reads it in as one two integer value and converts it into single integers for the row and column. The row and column coordinates of both the piece and destination are then taken through some error checking, the tests conducted are discussed below:

To make sure the coordinates given are within the 10x10 grid. Done with a simple if statement.

Next is a faction check to make sure if it’s the guardians turn that a guardian piece was selected and then the same with the demons turn. For this a pointer to the BPiece class was created to get access to its GetFaction function member.

The most important check is then done, to see if that move is a valid move for that piece. This is done by firstly requesting the valid move for the piece through the IsMoveLegal function member of the BPiece class. Next it creates a temporary move to the destination then the checks are done and an additional check to make sure the player is not in check is done.

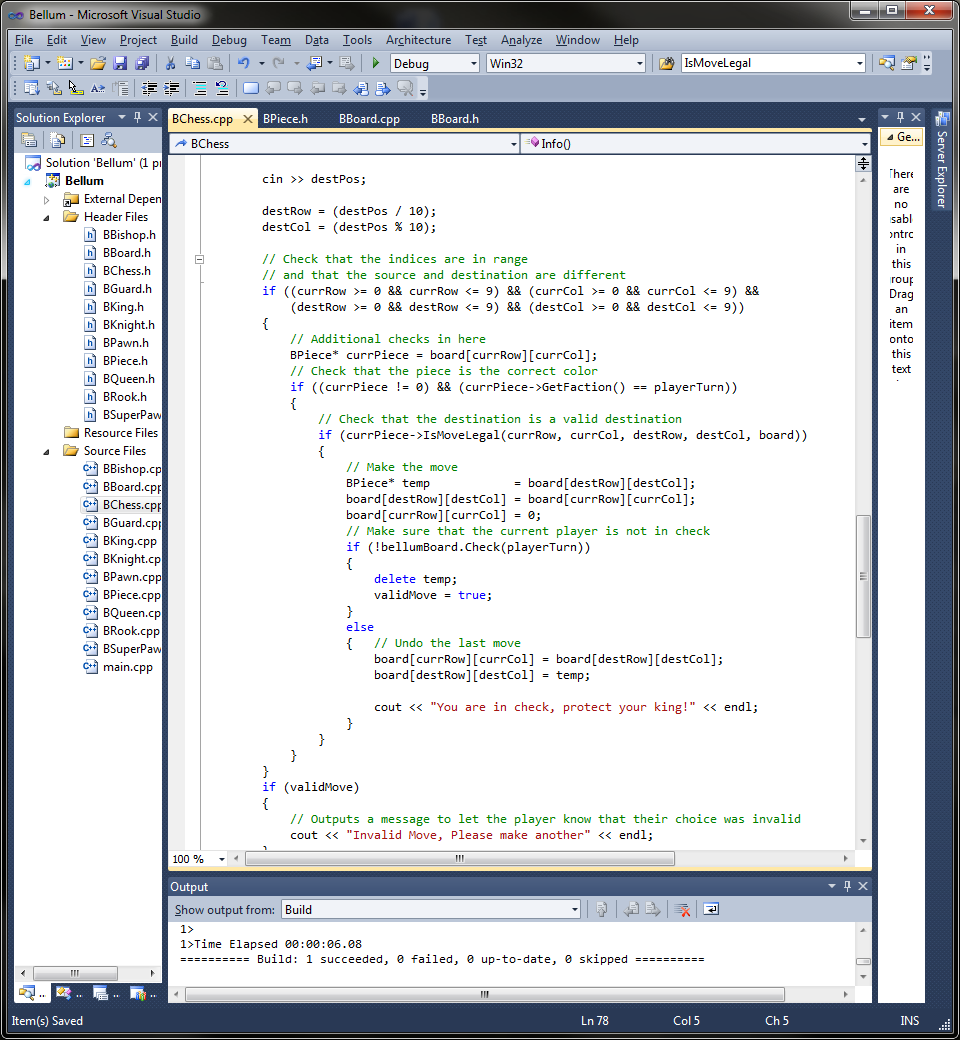
If all the checks are successful then the validMove Boolean is set to true and the move is made. If a single check is unsuccessful then the piece is sent back to its original position and the message ‘Invalid Move, Please make another’ is outputted to the console. An additional message is displayed if the user is in check which reads, ‘You are in check, protect your king!’.

Figure xxxi - Valid Move Check

*(bool)GameOverCheck* – As mention in the Game function description above this function is used as the parameter of the do while loop. If the returns true then the game is over. To be able to check whether the game is over or not it uses the BBoard object declared within the header file to access the CanMove member function to see if the current player is able to move. If the player is unable to move then function does further checking to see if the player is in check or not. If not then the message ‘Stalemate!’ is displayed otherwise ‘Checkmate <insert player> Wins!’.

### 4.1.2 Board Class (BBoard)

The board class deals with everything related to the board itself, this includes, the placement of the pieces, the displaying of the board and piece interaction. It has access to all the pieces classes and the piece class itself. The class contains six public members: five member functions and one data member which will be discussed in detail below.

The member function is the most important part of the whole game: The board itself. A pointer to an object of the BPiece class was created. As mentioned in the design the two dimensional array method for storing the board information was used.

*BBoard –* Within the constructor the array dynamically allocated with the pieces of the game. Instances of the pieces are created and placed in the appropriate array element. The instances of the pieces take a character parameter to denote which faction the piece belongs to.

*~BBoard –* The destructor is called when the object of this class in destroyed. In this case the destructor goes through each element of the array and deletes its contents to avoid memory leaks.

*(void)OutputBoard* – This member function is called within the BChess class both when the game has finished, showing the final piece locations and every time the game loops within the NextMove member function.

White squares are shown using the star notation ‘\*’ and black squares are black text. 0-9 is outputted on the left hand side and bottom so the user knows what to input for the row and column coordinates for movement. Within a for loop when the main board is being outputted the pieces are called and if the location on the board contains a piece then the characters that denote the piece are outputted instead of a ‘\*’ or blank text.

Below on the left is the board without any pieces then once the pieces have been dynamically allocated they output to the board as shown in the image on the right.

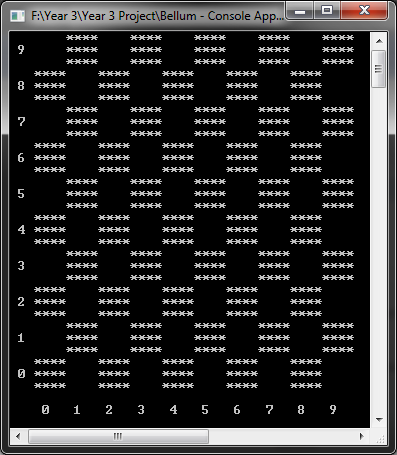
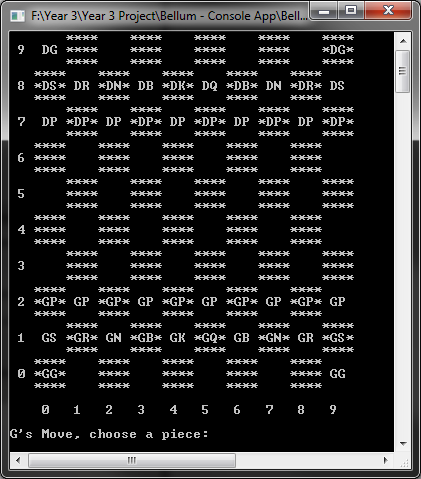


Figure xxxii - Board With Pieces

Figure xxxiii - Board

*(bool)CanMove* – This member function is used to check to see if the player can move. The function is used with the GameOverCheck member function within the BChess class. The check is done by going through each of the pieces left on the board, then getting the coordinates of the current players’ pieces. It then makes a temporary move and then checks to see if the king will be in check after if so then the function returns false otherwise if the piece can move then true if returned.

*(bool)Check* – Final function member within the BBoard class which is used to determine whether the king is in check or not. Firstly the function does a loop through each element of the array looking for the king. Once it finds the king it stores the row and column of the king in located variables.

Now with the kings coordinates known the function then goes into another loop to check to see if any of the opponents pieces are able to take the king.

### 4.1.3 Piece Class (BPiece)

This class handles all the individual pieces of the game. It does not have access to any other classes it’s just used for pieces to inherit certain members from. It contains seven members: five public function members(one of which is a virtual function inherited by child classes) and two encapsulated members (one data member for storing the faction when the class is used and a virtual function member used by one of its own public function members for getting valid moves from a desired piece.

*BPiece* – This constructor takes in the faction when an object is created and sets a private data member to this parameter.

~BPiece - A default destructor which is called when the class object is destroyed.

(virtual char)GetPiece – This is an inherited function member. It’s used to gain access to pieces.

(char)GetFaction – This is used to get the faction of the piece.

(bool)IsMoveLegal – This takes five parameters: the current row and column, the destination row and column and the board. The function is called within BBoard and BChess classes to check for legal moves. Within the function it calls the virtual function CheckSquares which will check for a valid move depending on the detected piece.

(virtual char)CheckSquares – Takes in the same five parameters as the above function. From the current row and column the function detects which piece it needs to get the valid moves from. Once it knows which piece it sends the parameters which will be used to check in move is legal.

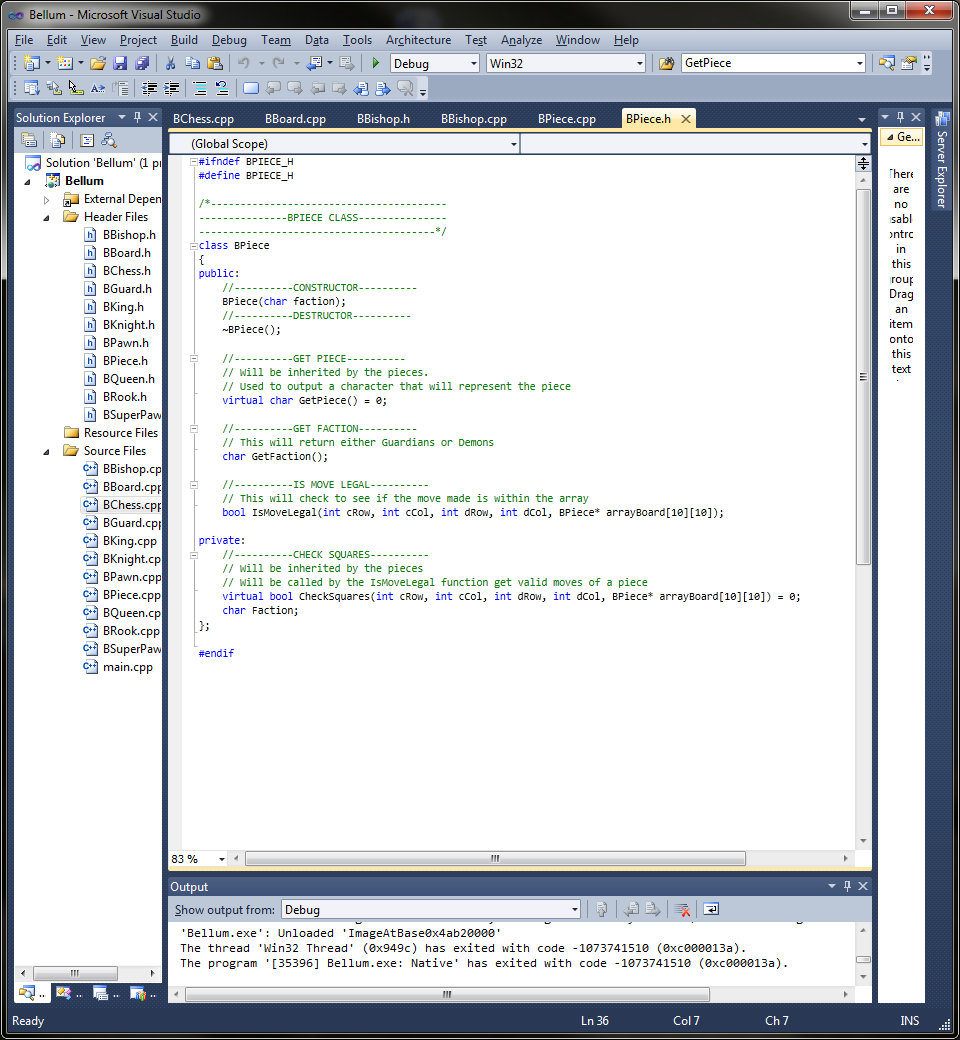
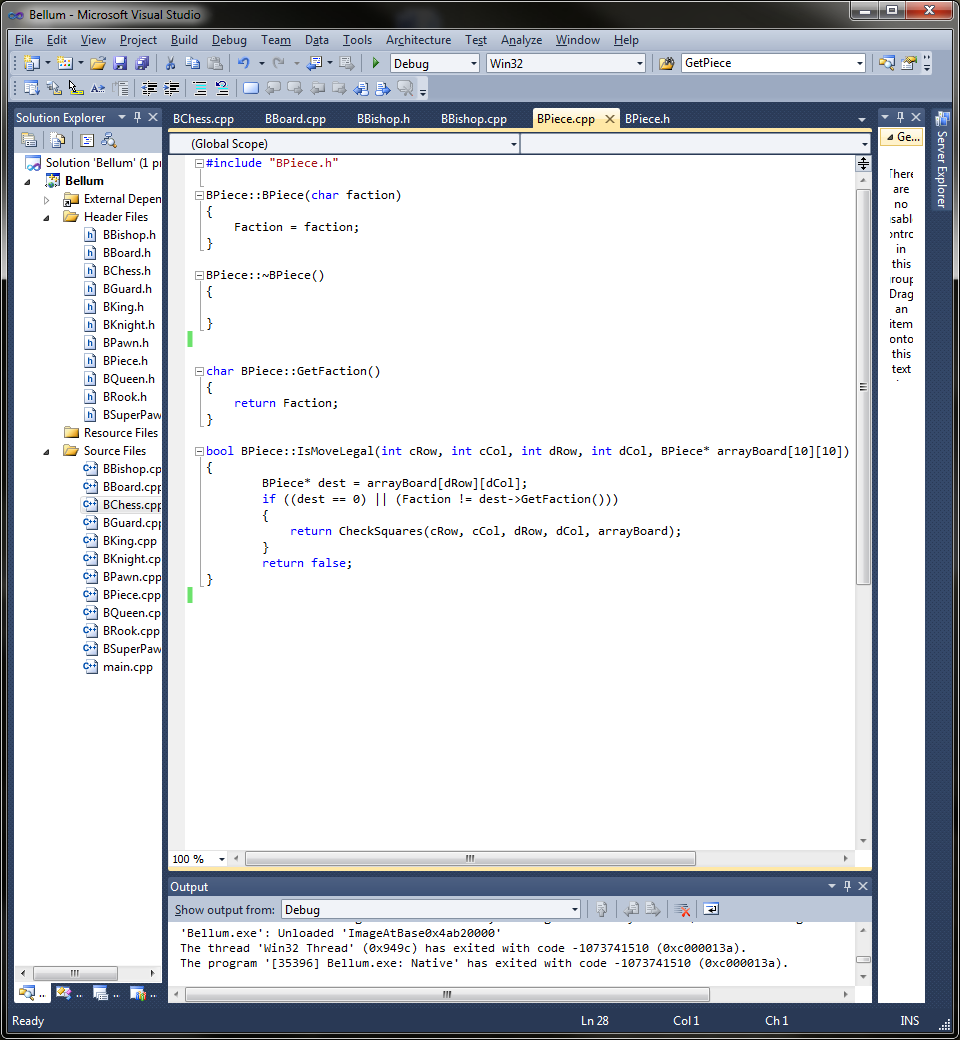


Figure xxxiv - IsMoveLegal Function

Figure xxxv - Pieces Class Header File

### 4.1.4 Individual Piece Classes

Each piece class are derived from the Piece class. Each has two private function members that have been inherited from the Piece class. Below each piece’s valid move algorithm that is implemented in the CheckSquares function will be explained via pseudo code with a screenshot of the code along with what will be returned by the GetPiece function. The CheckSquares function as mentioned above in the Piece class description takes in five parameters: current row and column, destination row and column and the board itself.

The layout of the piece header files like the pawn class shown below:

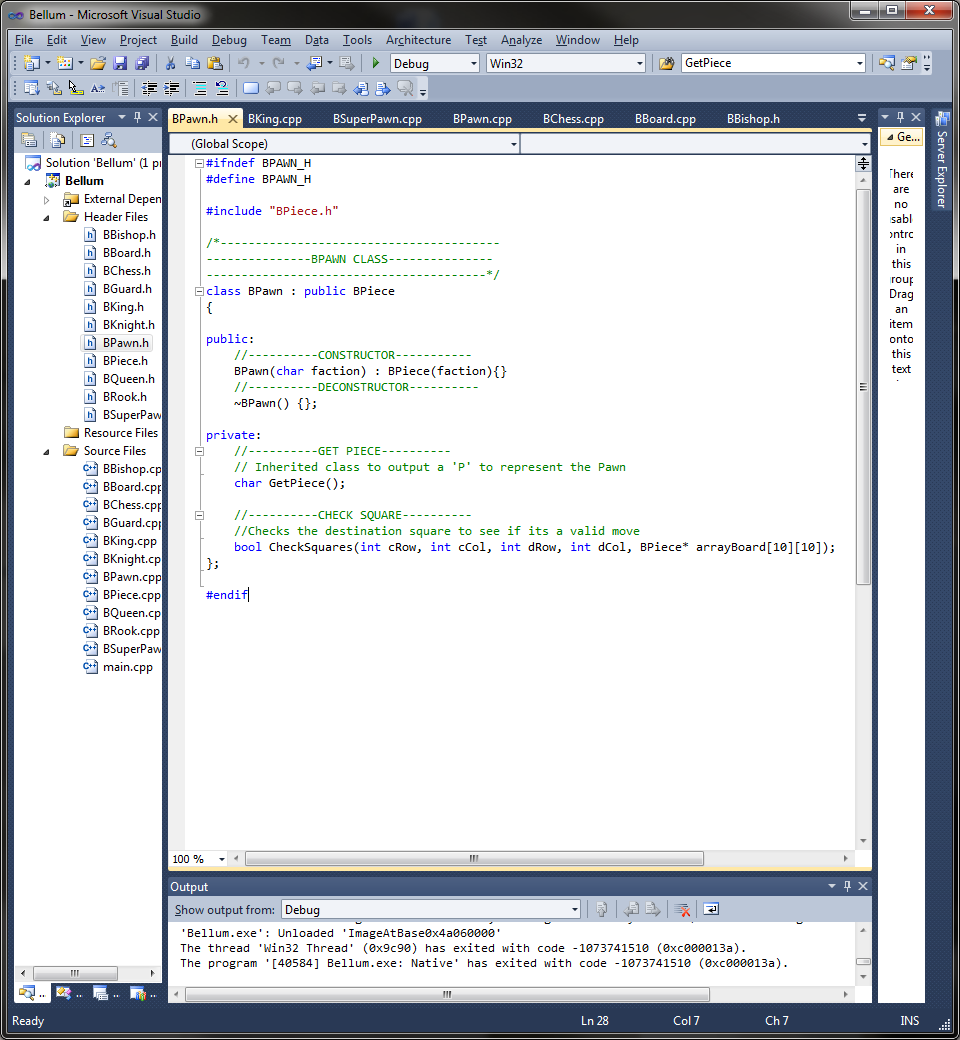


Figure xxxvi - Example of a Pieces Class (Pawn)

**Pawn**

*(char)GetPiece* – The character that would be returned to denote the pawn is ‘P’.

*(bool)CheckSquares*

As the pawn can only move forward and has a special move a few things need to be considered:

* Which faction the pawn belongs to
* The column in which it is in cannot change
* The only time the column can change is when taking a piece as pawns can only take diagonally forward
* Special move is that pawns can move two squares forward on their first move

-Pseudo Code-

*Create an object of Piece that has the coordinates of the destination square*

*If the object == 0 then space is empty*

*If current column == destination column*

*If faction == guardians*

*If destination row == current row + 1*

*Then move is valid – return true*

*Else if destination row == current row + 2*

*If current row == starting row*

*Then move is valid – return true*

*End if*

*End if*

*Else faction == demons*

*If destination row == current – 1*

*Then move is valid – return true*

*Else if destination row == current row – 2*

*If current row == starting row*

*Then move is valid – return true*

*End if*

*End if*

*End if*

*End if*

*Else if opponent piece infront diagonal square*

*If current column == destination column + or – 1*

*If faction == guardians*

*If destination row == current row + 1*

*Then move is valid – return true*

*End if*

*Else faction == demons*

*If destination row == current row – 1*

*Then move is valid – return true*

*End if*

*End if*

*End if*

*End if*

*If none of the statement combinations are true then move is not valid – return false*

**Knight**

(char)GetPiece – This returns an ‘N’ to denote the knight. It would have been a K but the king gets priority.

(bool)CheckSquares – Compared to other pieces the knight has somewhat unique valid move rules. The following needed to be considered to create the algorithm.

* First the knight moves one or 2 squares in a horizontally or vertical direction.
* Next depending on the previous move the knight does the other movement. If it moved one square it now moves two squares and if it moved horizontally it now moves vertically and vice versa.
* The knight can also jump over other pieces

-Pseudo Code-

*If current column == destination column + 1 or current column == destination column -1*

*If current row == destination row + 2 or current row == destination row – 2*

*Then the move is valid – return true*

*End if*

*Else if current column == destination column + 2 or current column == destination column – 2*

*If current row == destination row + 1 or current row == destination row – 1*

*Then the move is valid – return true*

*End if*

*End if*

*If the statement does not return then return false*

**Rook**

(char)GetPiece – This function returns the letter ‘R’ to denote the rook.

(bool)CheckSquares – To implement the rooks valid move algorithm the following rules must be consider:

* Can move horizontally and vertically
* Can move for as many squares possible in the chosen direction
* Cannot jump over pieces.

-Pseudo Code –

*If the movement it horizontal then check that the destination and current row are the same*

*Create an offset variable to see how many squares to check*

*Loop for however many squares the rook is moving*

*If a square contains a piece*

*Move is invalid – return false*

*End if*

*End loop*

*If all squares are empty then move is valid – return true*

*Else if the movement is vertical then check that the destination and current column are the same*

*Create an offset variable to see how many squares to check*

*Loop for however many squares the rook is moving*

*If a square contains a piece*

*Move is invalid – return false*

*End if*

*End loop*

*If all squares are empty then move is valid – return true*

*End if*

*If the attempted move does not satisfy the rules then the move is invalid – return false*

**Bishop**

(char)GetPiece – A ‘B’ is returned by this function to denote the bishop.

(bool)CheckSquares – The following rules were devised to be used when the algorithm was implemented:

* Can move in any diagonal direction
* Can move for as many space desired
* Cannot jump over pieces

-Pseudo Code-

*Create offsets to check all squares between destination and current*

*If destination column – current column == destination row – current row*

*Or destination column – current column == current row – destination row*

*Loop for the amount of squares moved*

*If a square contains a piece*

*Then move is invalid – return false*

*End if*

*End loop*

*If the squares are empty then move is valid – return true*

*End if*

*If the desired move does not satisfy the statement then move is invalid – return false*

**Queen**

(char)GetPiece – The letter ‘Q’ is returned to denote the queen.

(bool)CheckSquares – The queens valid moves are a combination of the bishops and the rooks, therefore the following rules needed to be followed to implement the algorithm:

* Can move in any direction
* Can move for as many squares as desired
* Cannot jump over pieces

-Pseudo Code-

*Create offsets to check all squares between destination and current*

*If destination column – current column == destination row – current row*

*Or destination column – current column == current row – destination row*

*Loop for the amount of squares moved*

*If a square contains a piece*

*Then move is invalid – return false*

*End if*

*End loop*

*If the squares are empty then move is valid – return true*

*Else If the movement it horizontal then check that the destination and current row are the same*

*Create an offset variable to see how many squares to check*

*Loop for however many squares the rook is moving*

*If a square contains a piece*

*Move is invalid – return false*

*End if*

*End loop*

*If all squares are empty then move is valid – return true*

*Else if the movement is vertical then check that the destination and current column are the same*

*Create an offset variable to see how many squares to check*

*Loop for however many squares the rook is moving*

*If a square contains a piece*

*Move is invalid – return false*

*End if*

*End loop*

*If all squares are empty then move is valid – return true*

*End if*

*If the attempted move does not satisfy the rules then the move is invalid – return false*

**King**

(char)GetPiece – The character that would be returned here that denotes the king is ‘K’.

(bool)CheckSquares – The king has a very basic valid move algorithm. To implement the algorithm the following rules needed to be followed:

* Can move in any direction
* Can only move one square each move

-Pseudo Code-

*Create delta variables for row and column*

*If row Delta >= -1 and <= 1 and column Delta >= - 1 and <= 1*

*Then move is valid – return true*

*End if*

*Otherwise the function returns false*

**Super Pawn**

(char)GetPiece – This function returns the character ‘S’ to denote the piece.

(bool)CheckSquares – The super pawn is one of two original pieces to Bellum. The way it can move was discussed within the Design section. To get these valid moves the following was considered:

* Can move in any direction
* Can move up to two squares at a time
* Can jump over pieces

-Pseudo Code-

*Create delta variables for row and column*

*If row Delta >= -2 and <= 2 and column Delta >= - 2 and <= 2*

*Then move is valid – return true*

*End If*

*Otherwise the function returns false*

**Guard**

(char)GetPiece – To denote the guard a ‘G’ is returned from this function

(bool)CheckSquares – The second of the original pieces has a combination of the bishop and knight valid moves. For the algorithm the follow was considered:

* Can move diagonally in any direction for however many squares
* If moving like the bishop it cannot jump over pieces
* Can move in an L shape: two squares in one direction and one in the other
* If moving like the knight then it is able to jump over pieces

*Create offsets to check all squares between destination and current*

*If destination column – current column == destination row – current row*

*Or destination column – current column == current row – destination row*

*Loop for the amount of squares moved*

*If a square contains a piece*

*Then move is invalid – return false*

*End if*

*End loop*

*If the squares are empty then move is valid – return true*

*End if*

*If current column == destination column + 1 or current column == destination column -1*

*If current row == destination row + 2 or current row == destination row – 2*

*Then the move is valid – return true*

*End if*

*Else if current column == destination column + 2 or current column == destination column – 2*

*If current row == destination row + 1 or current row == destination row – 1*

*Then the move is valid – return true*

*End if*

*End if*

*If the desired move does not satisfy the statement then move is invalid – return false*

-Pseudo Code-

**Special Moves**

In addition to the valid moves on the previous pages special moves were implemented.

**Castling**

The first of the special moves that was implemented was what’s known as castling; where the king can move next to the rook and then the rook moves to the other side of the king, providing that both the king and rook are in their original squares. This was implemented within the kings’ valid moves.

The following rules needed to be considered before implementing this rule:

* Make sure the rook and king are in their original squares
* Make sure there are no pieces in between the king and rook

-Pseudo Code-

*Make sure they are on the same row*

*If guardians piece*

*If destination column for the king == current column - 2*

*If king and rook are in original squares*

*If the squares between them are empty*

*Move rook to place*

*Move is valid – return true*

*Else*

*Squares contain pieces – return false*

*End if*

*End if*

*Else if destination column for the king == current column +3*

*If king and rook are in original squares*

*If the squares between them are empty*

*Move rook to place*

*Move is valid – return true*

*Else*

*Squares contain pieces – return false*

*End if*

*End if*

*End if*

*Else If demons piece*

*If destination column for the king == current column - 2*

*If king and rook are in original squares*

*If the squares between them are empty*

*Move rook to place*

*Move is valid – return true*

*Else*

*Squares contain pieces – return false*

*End if*

*End if*

*Else if destination column for the king == current column +3*

*If king and rook are in original squares*

*If the squares between them are empty*

*Move rook to place*

*Move is valid – return true*

*Else*

*Squares contain pieces – return false*

*End if*

*End if*

*End if*

*If not statement is satisfied then invalid move – return false*

From the pseudo code the following code was implemented:

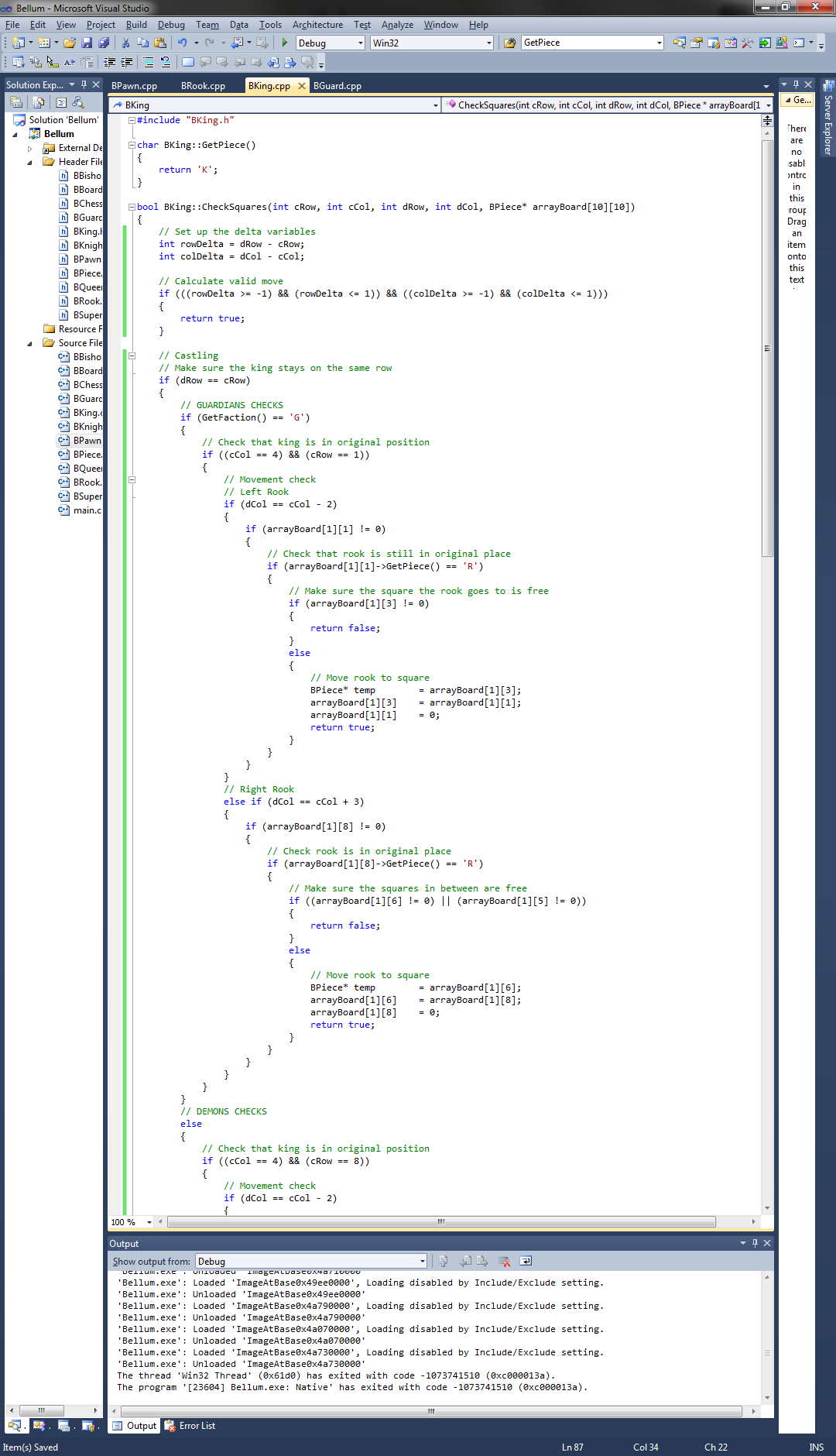
Guardians section

Figure xxxvii - Guardians Section of Castling

Demons section

Figure xxxviii - Demons Section of Castling

**Promotion**

Completing this special move required a separate function within the BBoard class that would be called at the end of the game loop to do a check. The check would see if any pawns are in the last row, if so then a queen would be created in the pawns position.

*Loop through all squares*

*If square contains a pawn of the current player*

*Delete pawn*

*Create new Queen*

*End if*

*End Loop*

**Taking En Passant**

This special move required two sections. One section within the pawn class to check if any enemy pawns were adjacent to it. The other section was a function in the BBoard class that would be called after the promotion function which would delete the enemy pawn.

Pawn Section

*If destination column == current column + 1 or – 1*

*If faction is guardians*

*If destination row == current row + 1*

*If destination row – 1 == enemy pawn*

*Move is valid – return true.*

*End if*

*End if*

*Else faction is demons*

*If destination row == current row - 1*

*If destination row + 1 == enemy pawn*

*Move is valid – return true.*

*End if*

*End if*

*End if*

*End if*

BBoard Function

*Loop through all squares*

*If square contains current players pawn*

*If behind pawn is enemy pawn*

*Delete enemy pawn*

*End if*

*End if*

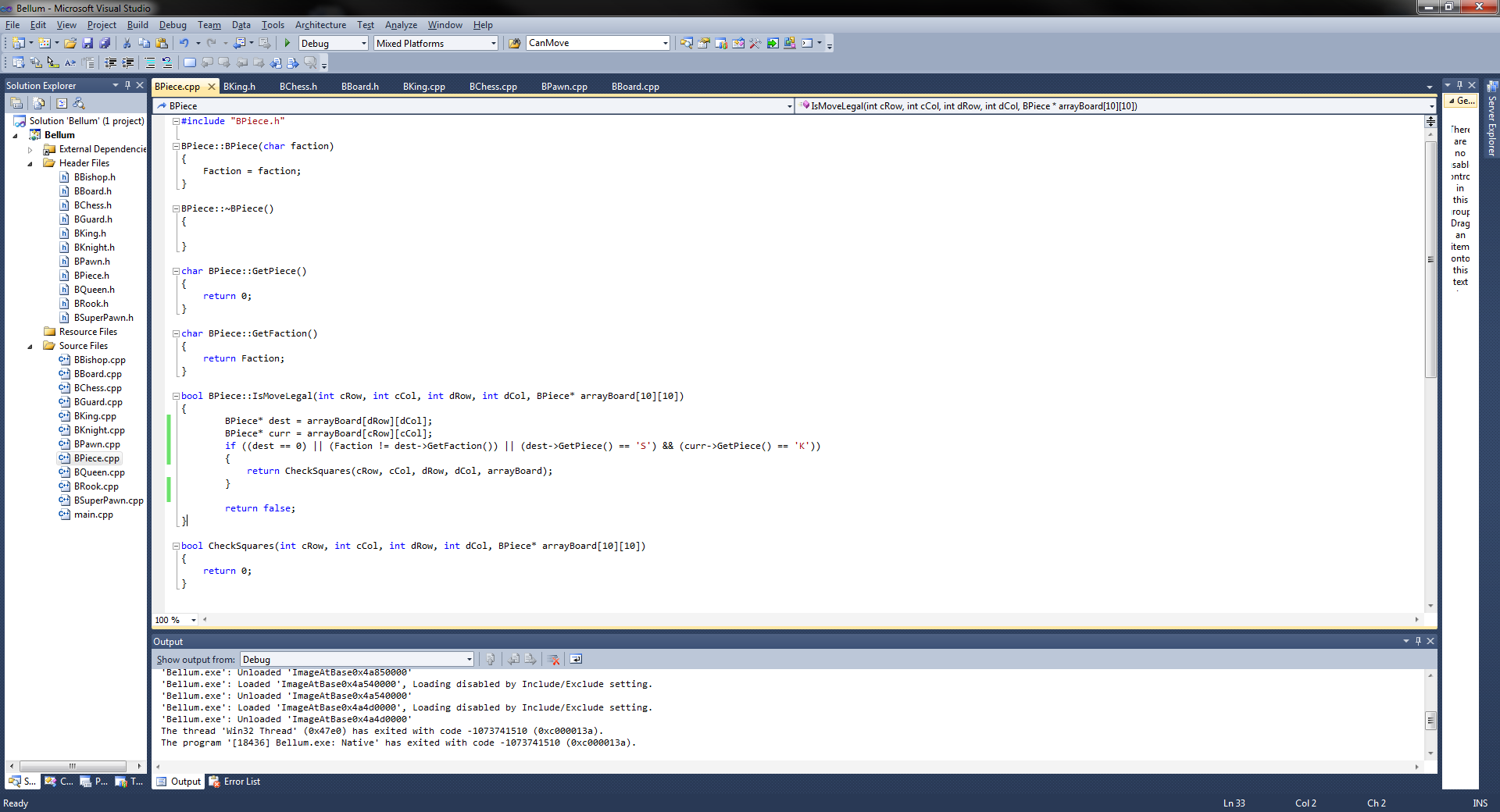
*End Loop*

**Super Pawn Sacrifice**

Firstly to do this the parameters within the IsMoveLegal function of the piece class needed to be added to. Allowing the move to be valid if the destination was the Super Pawn and the chosen piece to move was the King.

The next section was to add to the kings rules, allowing it to move however far providing the destination is the Super Pawn.

Modified IsMoveLegal function



Added section to king

*If destination == Super Pawn*

*Then move is valid – Return true*

*End if*

## 4.2 Graphics

# 5 Testing

Tests were completed while the game was being made to make sure functions and algorithms were working as they should.

Tests were documented along with screenshots as evidence. In some tests most of the pieces were removed to make it easier for demonstration.

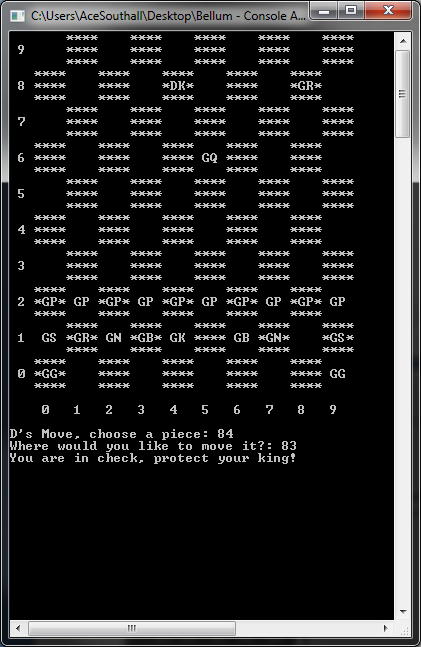
BChess Class Tests

Menu – This test checks to make sure the menu selections work. For this the characters ‘S, H and E’ were entered to make sure it would start a game, display the info screen or exit the game, respectively. *The test went as expected, it was successful.*

Info – To get out of the info screen the program requires you to input any character. *The test went as expected, it was successful.*

Game – This class contains the game loop which calls the NextMove function. When the game is started the game asks for a players input therefore the game function successfully works as it is the only function that calls the NextMove function. *This test was successful.*

NextMove – This function first asks for the coordinates for the piece the player wishes to move and the destination square for where they would like to place the piece. Based on the coordinates it works out if it is a valid move through the pieces class. If it is a valid move then the move is made and the next player has their go. If it is not a valid move it outputs an invalid move message and if the player is in check it outputs an in check message. See figures below for outputs. ***Tests were successful***



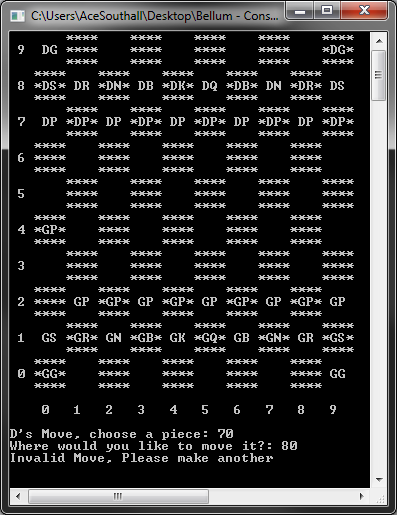


Figure xxxix – Move Invalid Message

Figure xl - Must Move Out of Check

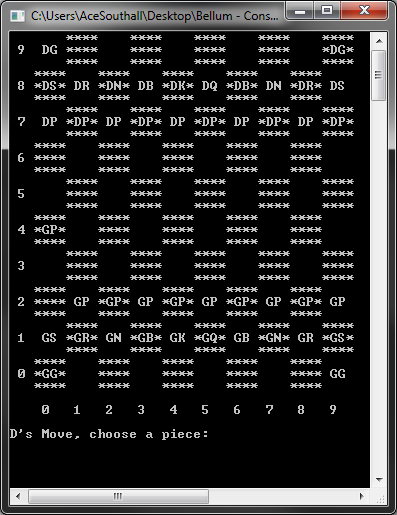
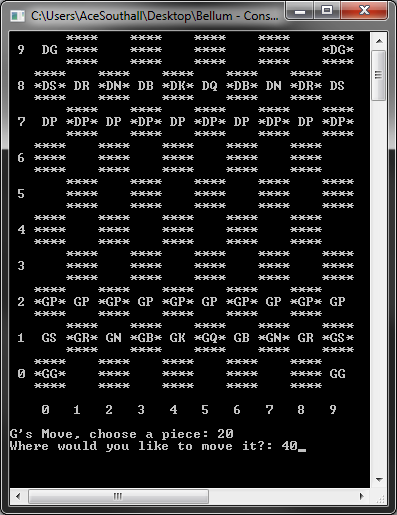
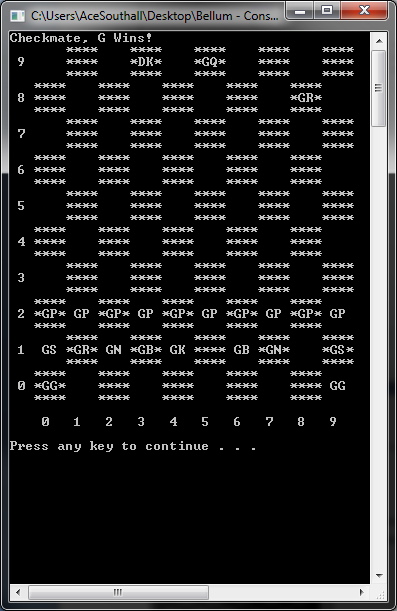


Figure xli - Successful Reads in Coordinates

Figure xlii - Move Pawn at (2,0) to (4,0)

GameOverCheck – This calls a function Check within the board class to check the kings locations and whether it can move. If it cannot move and nothing can protect it then it is in check. A basic scenario was made to demonstrate CheckMate. ***This test was successful.***

BBoard Class Tests

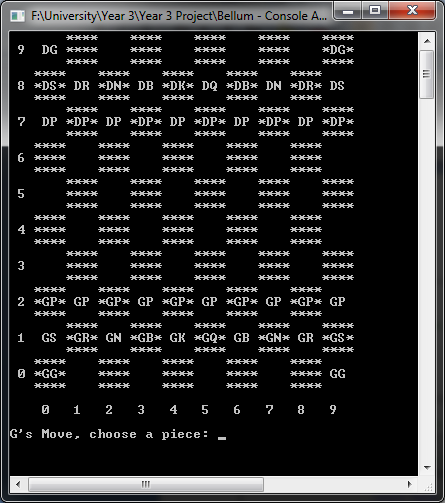
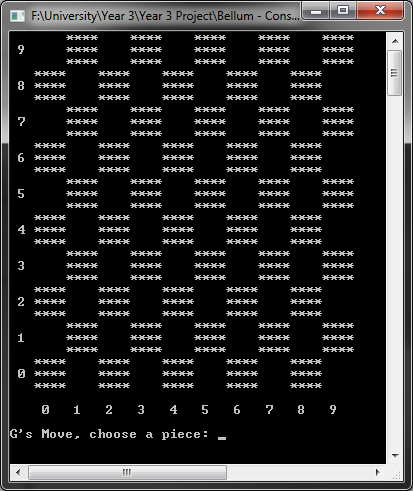
BBoard – This constructor dynamically allocates all the pieces to their board locations within the two dimensional array. From Figure xiiv it can be seen that the pieces are on the board where they should be. ***This test went successfully.***

Figure xliii - With Pieces

Figure xliv - Without Pieces

OutputBoard – As it would suggest this function is for outputting the board. On the left side and bottom of the board the numbers 0-9 are outputted for the coordinates for user input purposes. Above are images of the board with and without pieces. ***This test was a success***.

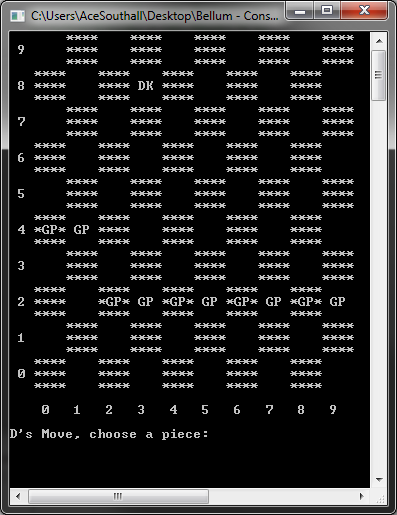
CanMove – This function is used to access the pieces to see if the piece can move. This was demonstrated within the NextMove function test.

Check – This function is used to check if the king is in check. This was tested within the GameOverCheck function test.

BPiece Tests

This section of the tests documents and shows the tests done to make sure the pieces valid moves are working as they should.

Pawn – Multiple tests for the pawn were completed; one move forward only, double move at the start, promotion and en passant. ***Tests were successful***



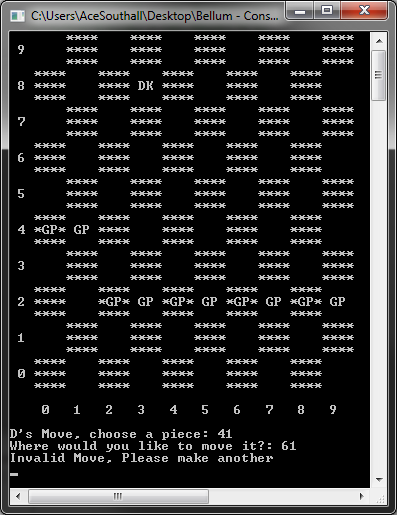
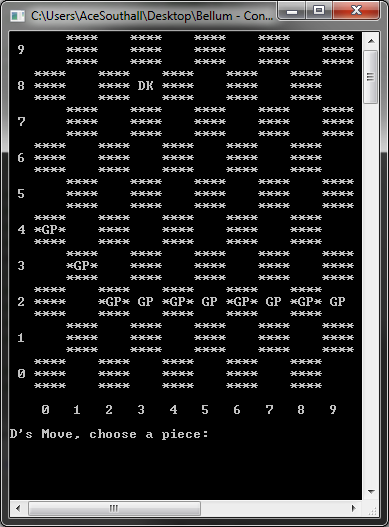
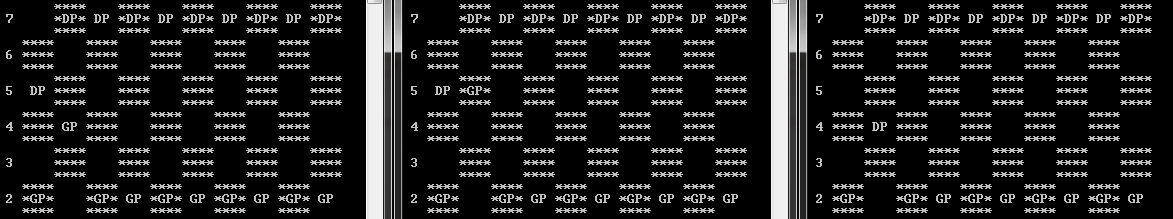


Figure xlv - En Passant works successfully

Figure xlvi - Promotion works successfully

Figure xlvii - Attempt to move same pawn by another 2 squares fails

Figure xlviii - Successful 1 square move

Figure xlix - Successful Double Move

Figure l - (2,1) to (4,1) Double Move

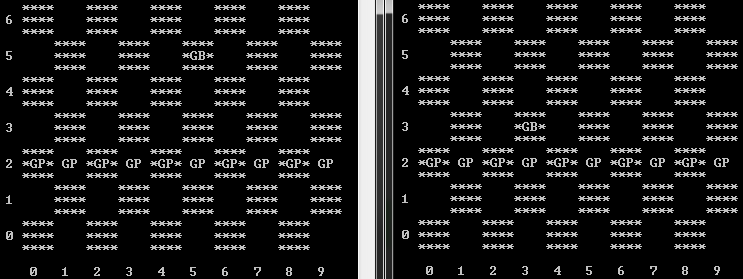
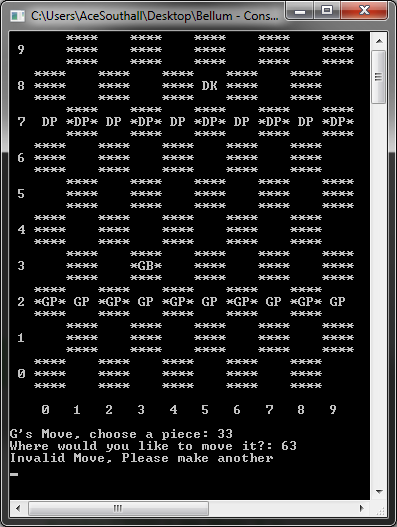
Bishop – Tests were conducted to make sure the bishop could only move diagonally. ***This test was successful.***

Figure li - Bishop Successful Diagonal Move



Rook – Just once test was conducted for this

Figure lii - Unsuccessful Vertical move

Figure liii - Unsuccessful Horizontal Move



Figure liv - Rook Successfully Moves Vertically and Horizontally

Figure lv - Unsuccessful Diagonal Move

Knight – Tests were done to make sure the knight could only move in an L shape, unable to move horizontally, vertically or diagonally. ***Tests were all successful.***

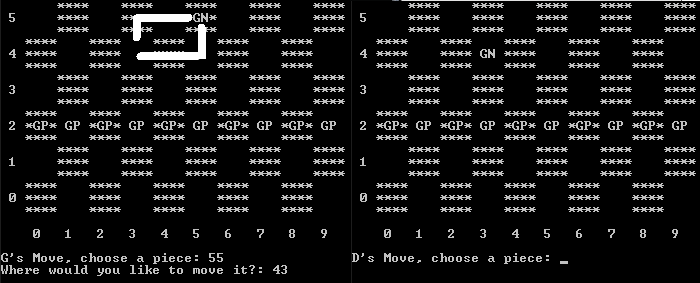


Figure lvi - Successful L Shape Movement

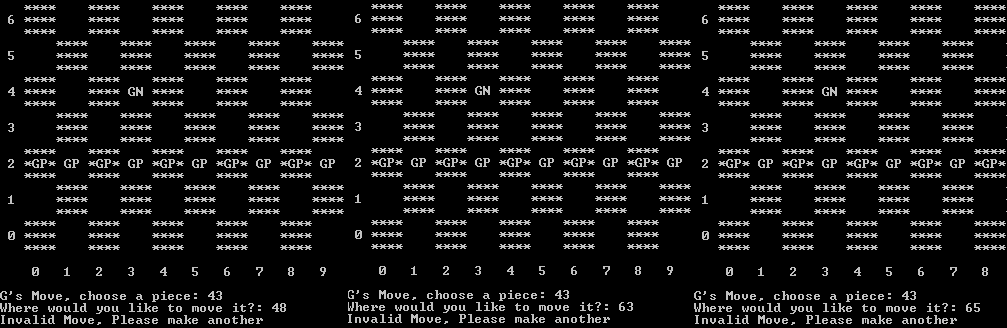
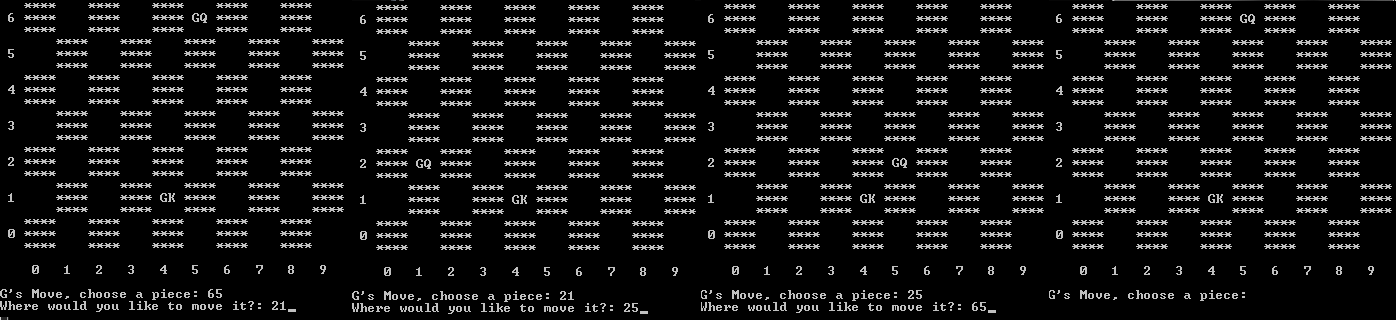


Figure lvii - Unsuccessful Moves. Left to Right: Horizontal, Vertical and Diagonally

Queen – Tests to make sure the queen can move in any direction for however much were conducted. ***Tests were successful.***

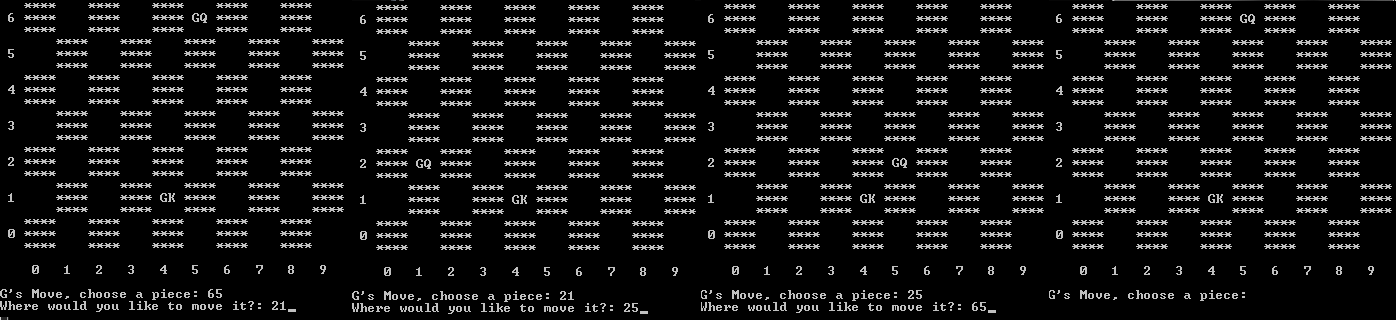
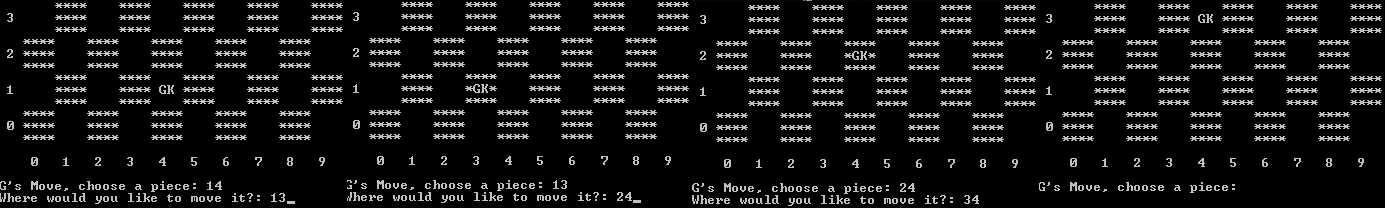


Figure lviii - Successful Horizontal and Vertical Movement

Figure lix - Successful Vertical Movement

King – Tests were completed to make sure it can only move one square horizontally, vertically and diagonally. Two extra tests were conducted to make sure castling work for both rooks. Final tests were to make sure the special super pawn sacrifice move works. ***Tests were successful.***

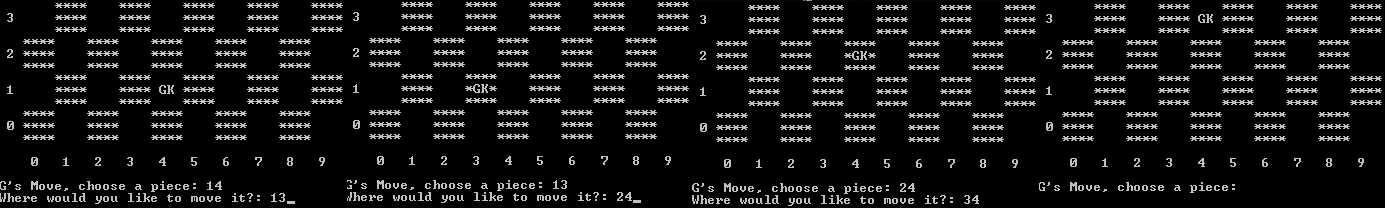


Figure lx - Super Pawn Sacrifice Successfully Done

Figure lxi - Successful Right Side Castling

Figure lxii - Successful Left Side Castling

Figure lxiii - Successfully Moves Diagonally and Vertically

Figure lxiv - Successfully Moves Horizontally

Super Pawn – Tests for this were the same as the kings testing but by two squares instead of one therefore no evidence needed. ***Tests went successful.***

Guard – The guard has a combination of the knight and bishops moves which were both successful therefore no evidence is needed. ***Tests went successful.***

# 6 Evaluation

## 6.1 Meeting Requirements

At the start of this project six minimum requirements were set that were to be completed through the timescale of the project. Below, each requirement will be marked whether it was completed or not. If the requirement was met then this will be elaborated on how it was met otherwise a explanation on why it was not completed will be given.

The first three requirements that were to be met were research orientated:

**To research into current chess variant game rules** - *This requirement was met successfully.*

This requirement was completed by using both website and book sources. ‘The Concise Guide to Chess Variants’ was a very interesting and useful source, containing over 50 possibly closer to the 100 mark of different style of chess games. It contained many useful links to other sources for more information on each of the variants.

Considering standard chess and most other chess variants contained an 8x8 or smaller board Grand Chess and Eurasian Chess were the most appropriate variants as they featured a 10x10 which was a desired Bellum feature. Most variants that were researched contained unique pieces that differentiated them from standard chess, this helped a lot with deciding how to implement the Guard and the Super Pawn from the valid moves used within the variants.

**To research into current developed chess games** – *This requirement was met successfully*.

This research was conducted to see how other chess games were implemented graphically, to assist with the graphical implementation of the project.

The used game was Battle vs Chess which was surprisingly a very well done chess game. There’s not that many chess games that stand out but this did. The graphics were stunning, the animations unique and brilliant for each piece type. The UI was clean and the scenery was pleasing to the eye. Overall it was good to see how visually pleasing chess game could be done. From research through there were no 3D graphical games like Battle vs Chess that contained chess variants such as Grand Chess which suggested an open market for such a game.

**To research into chess board representation** – *This requirement was met successfully*.

This was a very beneficial part of the research that made an impact on the overall project. Two different styles of representation were researched, the piece centric style kept information about individual piece whereas the board centric style held information about each element on the board.

The 0x88 method was the most advanced representation as it would avoid invalid moves without the developer having to implement a separate error check. The downside is that it is only optimised for 8x8 standard chess boards.

The method that was researched and then used within the implementing of the functionality is the two dimensional array method. It keeps information about each of the board squares and then implementing a basic if statement to make sure the move made by the player is within the 10 x 10 board is fairly straight forward.

**To research which API is more suited for the graphical application** – *This requirement was met successfully.*

To complete this requirement the two graphical API’s; OpenGL and Direct3D of DirectX were researched to see what the advantages and disadvantages of each were. The main feature of OpenGL that stood besides having more experience with it is that it us cross platform so making Bellum work on Linux required no extra work, where as if developed in Direct3D it would not work. What Direct3D did have were shaders already built into the API where with OpenGL it required extensions, which does not really require much effort.

From this OpenGL was the chosen API to be used for the graphical development.

**To design and implement an original console based chess program** – *This requirement was met successfully*

Firstly the structure of the application was designed. The decisions made were to have a main hub class which would contain the game loop and call necessary functions from the desired classes to make the game run. A board class was needed to contain the two dimensional array which would hold the information of the chess board. Then finally a piece class which would be inherited by each individual piece. Each individual piece would contain its valid moves and a character that would denote it so the program knows which one to check when a move is made.

When implementing the basic functions where created first such as the Game function which would just go between each player and then call the NextMove function which had been created and be used to take in a players input for the desired move. The OutputBoard function was also implemented along with the first few functions created. At the start all the pieces valid moves was same as the kings as it was the most basic, then once all other functions have been create such as Check which would see if the king was in check and CanMove which would see if the move made was valid then the valid moves for each piece were implemented correctly and not like the kings.

**To implement special moves –** *This requirement was successfully met.*

There were four special moves that needed to be implemented. All were successfully implemented.

The first of the special moves to be completed was the double move of the pawn if it is on its original square. This was done by checking its current square and if it was its original square when the move was valid and the function returned true.

The next to be implemented was castling. This was done by first making sure that king and rook are on their original squares, then making sure that the squares in between them are empty, if so then the move is valid and the function returns true.

Promotion was a basic implementation which was done in a separate function within the BBoard class. It would be called at the end of the game loop. It would check all pieces and if a pawn was in the end row it would be promoted to a queen.

En Passant was a fairly confusing implementation. First it implemented within the pawn valid moves function but once the move was made and the opponent took their next turn the game would break. So to solve this issue it was split into two parts. The first part within the pawn function would check to see if that if an enemy pawn was adjacent to the players pawn then it would be able to move diagonally behind it. The next part dealt with the removal of the enemy pawn which was implemented within a function in the BBoard class, which like the promotion function would be called at the end of the game loop. The function would check to see if any of the enemies’ pawns were behind the players’ pawns, if so then the pawn would be removed.

**To implement a graphical demonstration of a chess program** – *Partial met*

Originally the plan for this requirement was to implement the board and around five unique chess pieces which would be selectable and then squares would light up to show where the piece could move. This was to be done to show would a graphical version would be like. In the end just one piece was implemented as well as the board with squares lighting up to show the valid moves for this piece. The piece and the board were rendered using vertex array objects, vertex buffer objects and pixel and fragment shaders. These were described within the implementation section.

## 6.2 Future Developments

There are a few aspects of this game that could be improved, this includes graphical changes and additions and game play improvements.

The first thing that could be improved is a full graphical implementation this would include:

* Designed and implemented models for each of the pieces. Depending on whether it was a guardians piece or a demons piece would depend on whether it would have a white texture or a black texture.
* Implementing a sky box for scenery to be added.
* If the pieces were implemented like the demonstration pieces then animation could be added to them, so that when they take a piece it would do an animation to strike it off the board.

More game modes could be implemented by first creating a Menu class then being able to run different game modes. From the menu class it would open up different game and board classes depending on the desired chess game the player wanted to play. Each of the game and board classes would contain different piece setups and unique rules for that game. The game classes would be left the same as the pieces valid moves are the same within each game mode.

Elaborating more on adding more game modes; a challenge mode could be implemented, where the playing is challenged to either find a way to take a certain piece or get out of a situation where the player is in check for example.

Additional minor improvements that could be made to improve the overall quality of the game would be adding:

* Sound – Music playing the background and sounds made when a piece is taken
* UI – To show information like what the last move made was and what pieces have been taken.
* Options – To be able to set a timer for the game, to change between different sceneries etc.
* Achievements – For the accomplishment hunters out there to have something to work towards. It also adds to the replay value.
* Statistics – A section so players can look at how many games they’ve played and how many they’ve won out of them.
* Saving and Loading – This would allow the player to come back to the game at another point.
* Artificial Intelligence – Have it so that the game would be compatible with chess bots where one could be downloaded and installed straight into a section of the game.

# 7 References

[1] – Bodlaender, H (1996). The Chess Variant Pages. *The rules of chess*. Last Modified June 23, 2000. Available: <http://www.chessvariants.org/d.chess/chess.html> Date Accessed: 17/10/2012

[2] – Fide (Founded in 1924). FIDE – World Chess Federation. *Laws of Chess*. Last Modified Unknown.Available: <http://www.fide.com/component/handbook/?id=124&view=article> Date Accessed: 24/10/2012

[3] - Graham Burgess (2009). *The Mammoth Book of Chess*. 970-985.

[4] – Duniho, F (2005). The Chess Variants Pages. *Eurasian Chess.* Last Modified February 26, 2005.Availabe: <http://www.chessvariants.org/large.dir/eurasian.html> Date Accessed: 15/11/2012

[5] – Eurasia-Chess. (N/A). *Rules/Manual.* Available: <http://www.eurasia-chess.com/manual_en.php> . Date Accessed: 15th Nov 2012.

[6] - Pritchard, D. B. (2007). *The Classified Encyclopedia of Chess Variants*. John Beasley.

[7] – Rachunek, F (2002). BrainKing. *Grand Chess.* Available: <http://brainking.com/en/GameRules?tp=74> . Last Accessed on: 18th Dec 2012.

[8] – Aikin, J (1999). The Chess Variants Pages. *EIGHT-STONE CHESS*. Last Modified May 8, 1999. Available: <http://www.chessvariants.org/large.dir/contest/eightstones.html> . Date Accessed: 18th Dec 2012.

[9] – Battle vs. Chess. (2007). *Game Info*. Available: <http://www.battlevschess.com/en/> . Date Accessed: 19th Nov 2012.

[10] – Anthony, S (2012). ExtremeTech. *Valve: OpenGL is faster than DirectX – even on Windows.* Last Modified August 2, 2012. Available: <http://www.extremetech.com/gaming/133824-valve-opengl-is-faster-than-directx-even-on-windows> . Date Accessed: 4th Dec 2012.

[11] – Rosen, D (2010). Wolfire Blog. *Why you should use OpenGL and not DirectX*. Last Modified: January 8th, 2010. Available: <http://blog.wolfire.com/2010/01/Why-you-should-use-OpenGL-and-not-DirectX> . Date Accessed: 4th Dec 2012.

[12] – Promit, R (2002). Gamedev.net. *Direct3D vs. OpenGL: Which API to Use When, Where, and Why.* Last Modified: February 24th, 2002. Available: <http://www.gamedev.net/page/resources/_/technical/graphics-programming-and-theory/direct3d-vs-opengl-which-api-to-use-when-whe-r1775> . Date Accessed: 4th Dec 2012.

[13] – Isenberg, G (2008).Chess Programming WIKI. *Piece-Lists*. Last Modified: December 30th, 2012. Available: <http://chessprogramming.wikispaces.com/Piece-Lists> . Date Accessed: 9th Jan 2013.

[14] – Reul, F (2009). *New Architectures in Computer Chess*. 9-66.

[15] – Heinz, E A. (1997). *How DarkThought Plays Chess*. Vol. 20(3). 166-176.

[16] – Hyatt, R (N/A). *Chess program board representations*. Last Modified: Unknown. Available: <http://www.cis.uab.edu/hyatt/boardrep.html> . Date Accessed: 9th Jan 2013.

[17] – Moreland, B (2001). *0x88 Move Generation*. Last Modified: April 11th, 2002. Available: <http://web.archive.org/web/20071026130204/http://www.seanet.com/~brucemo/topics/0x88.htm> . Date Accessed: 9th Jan 2013.

# 8 Appendices

GRAPHICS

## 8.1 Appendix 1 –UML Diagram

**High Level Design**

Abstract

PIECE

ROOK

KNIGHT

BISHOP

KING

QUEEN

PAWN

SUPER PAWN

GUARD

CHESS

BOARD

## 8.2 Appendix 2 - LSEPI

IS3H605 Advanced LSEPI

LSEPI Assignment

Project Title

To Design and Implement a Turn Based Strategy Game Based on Chess with Focus on Graphical Detail

James Southall

January 2013

**Section 1 – Project Synopsis**

The project is to design and implement a turn based strategy game based on Chess with focus on graphical detail. The project will be named Bellum which is Latin for war. There will be two factions: angels (light) and demons (dark), which will include all the standard Chess pieces but also additional pieces and on a bigger board to house the increased number of pieces. With the main focus on graphical detail one of the noticeable techniques used will be animation, pieces will do an attacking animation when they take a piece.

Section 2 – LSEPI issue 1 – Intellectual Property and Copyright - Algorithms

As Bellum is a chess game, part of the implementation will be on storing valid moves for each of the pieces. Algorithms will be created to do this. As chess is overall a popular game on the internet there will have been an unlimited amount of games made all of which using very similar algorithms to store the valid moves. There’s a phrase “If it’s not broke, don’t fix it” therefore for Bellum current algorithms will be looked at and used as a referenced when the algorithms for Bellum are implemented.

With doing this it can put Bellum at risk from Intellectual Property and Copyright laws depending on if developers have patented algorithms if they are able to. So for this section research will be gathered to see if algorithms can be copyrighted and what sort of issues could be faced if this matter is not dealt with correctly.

**Intellectual Property**

“Intellectual property (IP) refers to creations of the mind: inventions, literary and artistic works, and symbols, names, images, and designs used in commerce.”[1]. IP rights fall principally into four main categories:

* Copyright – This applies to work this recorded in some way. The rights exist in items such as literary (novels, poems and plays), artistic (drawings, paintings, photographs and sculptures, and architectural designs), musical and dramatic work as well as films, sound recordings and typographical arrangements. It allows the author to have the right to take legal action if their work has been used without permission.
* Trademarks – Can be one of the follow: name (company name), word, slogan, design, symbol, or another representation for the identity of a product or company. Trademarking is done at a national or territory level with an appointed government body which can all take up to 18 months to process. If registering in countries such as the US, the UK, Japan, etc it will only protect the mark within that country, although within the European Union there is a Community Trade Mark (CTM) which covers marks in all of the EU countries. Registered trademarks can be identified with the abbreviation ‘TM’ or the ‘®’ symbol (It’s illegal to use one of these trademark identities until the trademark has been registered).
* Patents – These are applied to industrial processes and inventions, and protect against the unauthorised implementation of the invention. For a creator to gain exclusive right to use, sell or manufacture the invention then the national governments grant patents. They take up to 2 years to be granted and are like trademarks and are registered at a national or territory level.
* Design rights – Similar to copyright and patents but for designs of the appearance of a product in shape, texture, colour, materials used, contours and ornamentation. The design must be completely unique and original. [2].

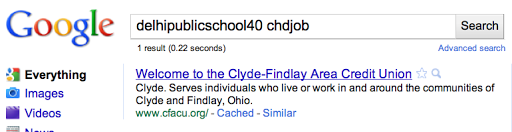
**Can Algorithms be Patented?**

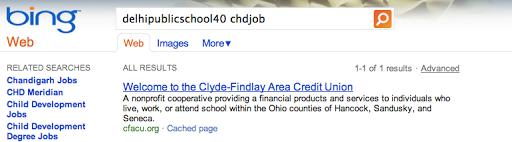
Yes is the basic answer here. They can be patented in many countries but not all. Like stated above a patent protects an invention so that it can be built, marketed and then profited from. Without this someone else with more resources and money could make it before the original creator and corner the market. It’s believed by many that software does not require such protection as it does not take anything physical to build it. Where-as real-world objects require machines to manufacture it and then other things to get it out there and to help along such as employees and a distribution network. Getting access to all that can be difficult to someone who has just started out and on their own. On the other hand with software all that is required is a compiler and an internet connection to implement and distribute it, so there is less need to protect as not much time is required [3].

**How Important are Algorithms?**

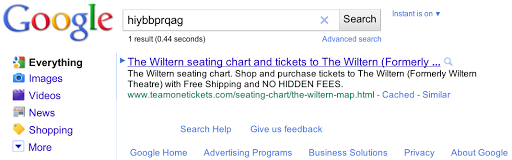
To some companies algorithms could be the difference to being an unknown company to being one of the biggest companies in the world such as Google. Google are known for their search engine which has been around since 1998 [4] which uses an algorithm to search for pages required by the user. At the start of 2011 Microsoft’s Bing was claimed to have been using Google’s search results which caused blasting from Google and Microsoft. In this google blog it shows tests being carried out on both Google and Bing where a random string of characters were entered and the exact same results popped up on both search engines. See next page for examples.

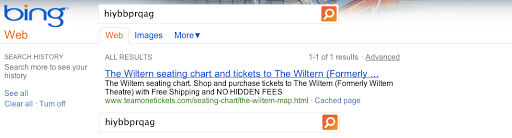
[delhipublicschool40 chdjob] was entered and the same result appeared:





[hiybbprqag] was also inputted and the same happened:



[5]

Many other queries were inputted like the examples and the same happened each time.

Having this happen can lead to a company gaining a bad name and could possibly lose users.

Section 3 – LSEPI Issue 1 Conclusions

From the research gathered the following statements can be made that will be relevant when implementing Bellum:

* Algorithms can be patented
* Using something that has been patented is breaking the law
* Using algorithms made by a rival company can give a bad name.

When algorithms get researched for Bellum extra care will be taken. If an algorithm is desired and will be useful to the design and implementation for Bellums algorithms measures will be taken to ensure no law is broken. This includes:

* First checking to see if the algorithm is open sourced and available for use by the public
* If this is not the case or there’s no official statement or evidence that leads to the knowing of the algorithm being open sourced then contact may be made with the creator/developer/company to request whether it can be used and referenced

Section 4 – LSEPI issue 2 – Ethics – Video Game Violence

The biggest controversy in video games is the violence with them and their influences on people. For this section material will be gathered and compiled on the worlds mixed view on the violence in video games, with certain headlines about crimes that have been committed and medias blast on violent games being the influence. Reasoning for discussing this LSEPI issue is due to the project contain mild violence in which as mentioned in the project synopsis pieces will carry out an attacking animation to take/destroy/kill a piece.

**Sandy Hook Elemental Shooting**

The best place to start is the most recent tabloid headline: The Sandy Hook Shootings. On Friday 14th December 2012 Adam Lanza aged 20 first killed his mother at their home in Newtown then moved onto Sandy Hook Elementary School where he killed twenty children, six staff members and injured a further two [6]. The shootings were blamed on the perpetrator’s interest in video games. The author of the article explains he is a video game violence researcher and also someone who has done scholarship on mass homicides has stated that “*Yesterday, Senator Jay Rockefeller*[*introduced*](http://www.huffingtonpost.com/2012/12/19/video-games-sandy-hook_n_2330741.html?utm_hp_ref=technology&utm_hp_ref=technology)*a bill calling on the National Academy of Sciences to “study” video game violence on children.”*[7]. It’s understandable that when such a horrific and emotional incident happens people are going to want to grieve and find something to blame for it. People that grieved over the Sandy Hook massacre found Adam Lanza’s Facebook page and seen that he had a big interest in the video game Mass Effect so they are going to jump at the throat of violent video games [8]. Mass Effect is far from the bloodiest game but still the roaring mob took to the game’s Facebook page and branded the developers ‘child killers’ [8]. Even GTA5 got blamed for this incident, “US Senator Joe Manchin has decided to place the blame on the popular “Grand Theft Auto” franchise and insists that the upcoming “GTA 5” be banned”. A game that is not even out yet. From this article let alone any other incident along the same lines, if there’s a mention that the murderer had any interest in video games then the media will use it as a scapegoat as it provides a good headline.[9]

**Stefan Pakeerah Shooting (Manhunt)**

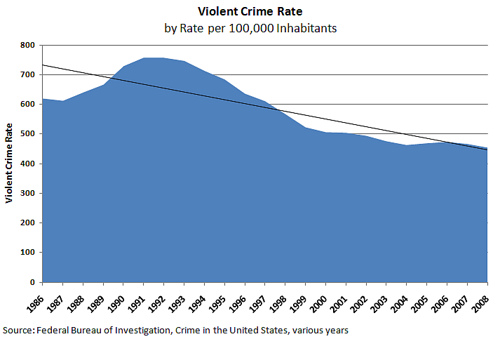
Back in 2004 on 26 February Stefan Pakeerah aged 14 was brutally murdered with a claw hammer by Warren Leblanc aged 17. The game to blame this time was Manhunt which both were obsessed with. Bear in mind Manhunt is an 18 rated game, for a reason. Manhunt was developed by Rockstar North based in Edinburgh. The aim of the game is to gain points from killing, the more violent the kill the more points. Manhunt is known well for the amount of countries that it has been banned in including, New Zealand, Germany etc. In one article the victim’s father claimed they played Manhunt and that the way Warren was killed was in a similar way used on the game, followed by “There is some connection between the game and what he has done” [10]. Within the same article a spokesman for the Entertainment and Leisure Software Publishers’ Association sympathised for their loss but rejected any suggestion or association between the incident and the game and stated “The game in question is classified 18 by the British Board of Film Classification and therefore should not be in the possession of a juvenile. Simply being in someone's possession does not and should not lead to the conclusion that a game is responsible for these tragic events” [10]. In a different article it explains how this murder was committed to steal from the victim to repay a £75 debt that Leblanc owed to a gang. Roderick Price QC, defending Leblanc said “The reason he killed Stefan is rooted not in video games but in fear – in desperation born of fear” [11].

**Mature Rating on Games**

From the two stories we can gather that multiple things could have triggered the murders to do what they did. Leblanc owed a gang money and was too young to be playing the game. Most incidents that you read about within the media that are linked to video game violence are committed by minors, under the age of 18. The Stefan murder was just one of a few murders committed by a teen. People of a younger age are more influenced by what they see, as they are still learner and not fully sure what is right and what is wrong. It’s a reason why these games are given a 18+ rating so that it should only be played by people over the age of 18. Under that age the responsibility is down to the parents or guardians, therefore in some context they should take the blame for wrongly bringing up their children. From statistics at Pew Research Center show that 50% of Teens prefer Mature rating games, given their rating due to the extensive violence, blood, gore and/or bad language. Within the USA mature rating is 17+. The tests were done on between 12-17 year olds who should know what games rated 17+ are like, as the game is not intended for their entertainment [12].

**Violent Crime Rate**

Take a look at this chart which shows the violent crime rate across a 22 year period from 1986 to 2008, to which you can see that the crime rate has dropped a fair amount in that time. Yet the media constantly blast violent videos which have only really become noticeably gorier in the past 5 years due to the graphical limitations in previous years, but within them 5 years that’s where the biggest drop in crime rate has occurred [13].



Section 5 – LSEPI Issue 2 Conclusions

From the information gathered on Violence in Video Games it’s clear to say that the media seems to make it out that that is all there is to blame. No other factors are considered such as pressure to pay back a debt to a gang with the Stefan murder.

There’s two possible factors to blame for these incidents; either the convict has mental health issues or if it is due to the game then the parents/guardian should be to blame. There are millions of people who play games yet there’s only been a few cases where an avid gamer has committed a crime.

From studying the research Bellum will still contain the attacking animation but as planned it will not contain gore or blood. As the targeted audience is people of all ages this minimal amount of violence will be suitable.

Section 6 – Bibliography/Evidence of Research

[1] – WIPO, N/A – *What is Intellectual Property?.* [online] Available at: <http://www.wipo.int/about-ip/en/> [Accessed on: 01 December 2012].

[2] – The UK Copyright Service, N/A – *Summary of intellectual property rights.* [online] Available at: <http://www.copyrightservice.co.uk/copyright/intellectual_property> [Accessed on: 01 December 2012].

[3] – Programmers, 2010. *Can an algorithm be patented?*. [online] Available at: <http://programmers.stackexchange.com/questions/32482/can-an-algorithm-be-patented> [Accessed on: 01 December 2012].

[4] – Google Company, 2012. *Googles mission is to organize the world’s information and make it universally accessible and useful*. [online] Available at: <http://www.google.com/intl/en/about/company/> [Accessed on 01 December 2012].

[5] – Google Official Blog, 2011. *Microsoft’s Bing uses Google search results – and denies it*. [online] <http://googleblog.blogspot.co.uk/2011/02/microsofts-bing-uses-google-search.html> [Accessed on 01 December 2012].

[6] – CBCNews, 2012. *20 children among dead in Connecticut school massacre*. [online] Available at: <http://www.cbc.ca/news/world/story/2012/12/14/connecticut-school-shooting.html> [Accessed on: 16 December 2012].

[7] – Time Ideas, 2012. *Sandy Hook Shooting: Video Games Blamed, Again*. [online] Available at: <http://ideas.time.com/2012/12/20/sandy-hook-shooting-video-games-blamed-again/> [Accessed on: 22 December 2012].

[8] – Msn Now, 2012. *Internet mob blames Mass Effect video game for Sandy Hook shooting*. [online] Available at: <http://now.msn.com/sandy-hook-shooting-blamed-on-mass-effect-video-game> [Accessed on: 22 December 2012]

[9] – examiner.com, 2012. *‘GTA5 gets blamed for Sandy Hook Elementary Shooting by US Senator(Photos).* [online] Available at: <http://www.examiner.com/article/gta-5-gets-blamed-for-sandy-hook-elementary-shooting-by-us-senator> [Accessed on 22 December 2012]

[10] – BBC News, 2004. *Game blamed for hammer murder*. [online] Available at: <http://news.bbc.co.uk/1/hi/england/leicestershire/3934277.stm> [Accessed on: 23 December 2012]

[11] – The Telegraph, 2004. *Life for teenage who battered friend to death*. [online] Available at: <http://www.telegraph.co.uk/news/1470892/Life-for-teenager-who-battered-friend-to-death.html> [Accessed on 23 December 2012]

[12] – Pew Internet, 2008. *New Pew Internet/MacArthur Report on Teens, Video Games and Civics.* [online] Available at: <http://www.pewinternet.org/Commentary/2008/September/New-Pew-InternetMacArthur-Report-on-Teens-Video-Games-and-Civics.aspx> [Accessed on 23 December 2012]

[13] – The Technology Liberation Front, 2010. *Violent Video Games &Youth Violence: What Does Real-World Evidence Suggest?* By Adam Thierer. [online] Available at: <http://techliberation.com/2010/02/09/violent-video-games-youth-violence-what-does-real-world-evidence-suggest/> [Accessed on 23 December 2012]

## 8.3 Appendix 3

FINAL YEAR DEGREE PROJECT

**Objective Settings Proforma**

(to be completed and submitted to your supervisor by Friday 19th October, 2012)

Student’s Name: James Southall

First Assessor: Prof Andrew Ware

Second Assessor: Dr Alex Lohfink

Project Title:

Design and Implement an Original Functioning Turn Based Strategy Game Based on Chess

Project Objectives & Deliverables

Research

Design

Implementation

Evaluation

*Please tick this box to indicate your awareness of the university’s policy on ethical issues*

*The deliverables and objectives can often change due to unforeseen circumstances, or through the student’s research causing the project to follow a different path. If this is the case, and the project objectives change significantly, then the first assessor should make a note of the date and fill in a new objectives proforma, which should also be included as an appendix to the project report. The project organiser is to be consulted at this stage.*

## 8.4 Appendix 4

**Degree Scheme in Computing**

**FINAL Project Assessment and Comment Form 2012-13**

**Student James Southall**

**Project Title:** Design and Implement an Original Functioning Turn Based Strategy Game Based on Chess

**Supervisor: (1 or 2) …………………………………………**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mark**  **Category** | **Proposed Weighting Ranges** | **Agreed**  **Weighting** | **Mark**  **Allocated** |
| **Project Management**  ***(Only set by Supervisor 1)*** | 50 – 80 | **70** |  |
| **Originality & Self-Direction** | 40 – 80 | **80** |  |
| **Technical Complexity** | 20 – 80 | **60** |  |
| **Solutions, Evaluation & Conclusions** | 80 – 120 | **80** |  |
| **Final & Sub-Report Quality** | 50 |  |  |
| **Prototype / System Demo**  **Or Project Deliverable** | 50 – 100 | **80** |  |
| **Sponsor Mark** | 00 – 60 | **0** |  |
| **Sub-Total Marks** | ------- |  |  |
| **Sub-Total Percentage** | ------- | **60%** | % |
| **Milestone 1 (initial)**  **Research**  **Research Applied to Design**  **Prototype Development** | 20% | **20%** | % |
| **Final Presentation (including poster production)** | ------- | **20%** | % |
| **TOTAL PERCENTAGE** | ------- | **100%** | % |

*The weightings for the various aspects of the project are to be set at the initial objectives setting stage. The ranges shown are guidelines and the actual weighting set may be outside of these ranges if deemed appropriate.*

## 8.5 Appendix 5

**STATEMENT OF ORIGINALITY**

**SCHOOL OF COMPUTING**

**DEGREE SCHEME IN COMPUTING**

**LEVEL SIX PROJECT**

This is to certify that, except where specific reference is made, the work described within this project is the result of the investigation carried out by myself, and that neither this project, nor any part of it, has been submitted in candidature for any other award other than this being presently studied.

Any material taken from published texts or computerised sources have been fully referenced, and I fully realise the consequences of plagiarising any of these sources.

Student Name (Printed) ……………………………….

Student Signature ……………………………….

Registered Scheme of Study ……………………………….

Date of Signing ……………………………….