Team 4 Test Report

Geoffrey Blogref

Jonathan Parise

John Pistorius

Robert Sharp

Table of Contents

Introduction 3

Software Description 3

Software Quality Criteria 3

System Test Planning 3

Test Plan 3

Testing Philosophy 4

Test Metrics 4

Test Effort Metric 4

Behavior Specification 5

Mission Statement 5

Scope 5

Definitions, Acronyms and Abbreviations 5

Overview 5

Product Perspective 5

Product Functions 5

User Characteristics 5

Stakeholders 5

Constraints 6

Assumptions and Dependencies 6

General System Requirements 6

Undo Requirements 6

User Interface Requirements 6

Static Code Review 7

Test Cases/Results 14

Structural Tests 14

System Tests 30

Appendix A 72

Appendix B 100

Appendix C 102

References 116

# Introduction

## Software Description

The software under test is an open source chess program written in C# .Net. The chess program allows several different modes of gameplay including player vs. player, computer vs. player and computer VS computer. It also has many advanced features for setting up certain moves and has an undo and redo system.

For the purposes of this report, we are limiting the scope of our testing to only the player vs. player portions of the program. This is being done simply to keep the test system size reasonable for our project.

We chose to test a chess game because chess is a system with clearly defined rules. Most people know the rules of chess and these rules are testable. Additionally, chess is an interesting problem with a long history in computer science. The game often seems simple at first, but after investigation it is found to be very rich and complex. This should allow for many interesting and challenging testing opportunities that allow us to apply many techniques.

## Software Quality Criteria

Essentially for the software to meet out test criteria it must be playable. A game is meant to be fun and it isn’t very fun if the users can’t complete a game. However, even simply completing a game is not enough for the software to meet our standards.

Additionally, in order for the chess program to be of sufficient quality, users must be able to play a player vs. player game without encountering any significant errors. Significant errors include any error that causes the game to enter an unplayable state. They also include any error that allows the game to get into an illegal state. For example, allowing the player to make an illegal move is a significant error.

# System Test Planning

## Test Plan

1. Review and analyze requirements specification for completeness and correctness.
2. Perform a walk through using static code analysis
3. Develop test metrics and provide estimates to stakeholders.
4. Develop and execute structural unit tests.
5. Mitigate defects found
6. Develop and execute integration tests
7. Correct defects, re-execute structural and integration tests
8. Develop system functional tests, identify regression tests
9. Execute system testing, create deployment process
10. Correct defects determined from testing and conduct development team reviews.
11. Run regression tests
12. Execute customer acceptance tests
13. Prepare reports of test results and actual test metrics
14. Agreement from stake holders that system is ready for release
15. Release product to customer

## Testing Philosophy

Our testing philosophy is to apply many different techniques to the system as applicable. By their nature, certain aspects of the system can be more easily and better tested with certain techniques. We strive to find the best techniques for each section of the program and apply them.

Additionally, we leverage automated unit tests written in using the NUnit framework. These tests are quite helpful because they allow us to easily exercise test cases by setting up the correct program state. In some cases, a lot of preparation could be required for a single test case. It is often simpler to setup these cases programmatically. An example of this is testing for a stalemate. A stalemate only occurs if very few pieces are left on the board. Testing these conditions manually would require playing almost a full game to get the board cleared. Programmatically, it is very easy to clear the board and a unit test can be written for this case in only a few lines of code.

We used a mix of automated and manual testing in our approach to this system. These tests will be explained below.

## Test Metrics

See Appendix A for whole system metric.

## Test Effort Metric

|  |  |  |  |
| --- | --- | --- | --- |
| **Test (Sub) Groups** | **Estimated Number of Test Cases** | **Person-Day to create** | **Person-Day to Execute** |
| Structural-(automated unit tests) | 16 | 2 | 0.5 |
| Structural –(manual unit tests) | 64 | 3 | 7 |
| System – (functional) | 57 | 12 | 10 |
| Integration | 26 | 10 | 1 |
| **Total** | 163 | 27 | 18.5 |

# Behavior Specification

## Mission Statement

This “SrcChess” system will allow players to play chess either against other friends on the same computer or against an AI. The game allows players to learn and improve their chess playing skills by experimenting with new tactics. The intended audience for this document includes developers, testers and users

## Scope

The “SrcChess” system, herein referred to as “The System”, will be a software program for the Windows operating system that will be provided to players. The goal is to encourage more people to try chess and to help them learn about the game. Additionally, the system is designed to allow experimentation by players in the hope that they will learn new strategies.

## Definitions, Acronyms and Abbreviations

AI – Artificial Intelligence

## Overview

In this document, both functional and nonfunctional requirements have been divided into key features. The document is laid out in the IEEE 830 format, which presents the information in sections; the order of the sections and the requirements within them indicates their priority.

## Product Perspective

The product consists of a single desktop application used to setup and play chess games.. The product is expected to be used on a personal computer running the Windows operating system.

## Product Functions

Each product feature is explained in its own category. In all, there are six categories. Core system requirements are first, because they are the most important and include all of the base functionality. The second feature is undo, which aims to allow users to experiment with chess and try new tactics. Finally, the user interface is last, it controls everything the user sees and is important for allowing them to play.

## User Characteristics

The primary users of this system will all anyone who wants to play chess. As such, the expertise of each user will vary widely. Some users may be chess experts, while others may have little to no experience.

## Stakeholders

|  |  |  |
| --- | --- | --- |
| **Stakeholder** | **Rank** | **Rationale** |
| Players | High | They are the primary users of the system. They simply want to play chess. |
| Chess Aficionados | High | They will use the program to become better at chess. |

## Constraints

Below is a list of standards and regulations of the system.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Regulation / guideline** | **Rank** | **Rationale** |
| 2.4.1 | Laws of Chess | High | These are the official rules of chess from the Worldwide Chess Federation. |

## Assumptions and Dependencies

We assume users have a basic knowledge of how to use their computer.

## General System Requirements

1. The system shall comply with all laws, rules, and regulations referred to in section 2.4. This includes all of the rules for piece movement.
2. The system shall ensure that each legal move is accounted and played.
3. The system shall not allow illegal moves.
4. The system shall clearly identify the move being made by the player.
5. The system shall correctly end the game in the event of a checkmate.
6. The system shall end the game in the event of a stalemate.
7. The system shall keep a history of all moves made by each player.
8. The system shall track the playtime of each player, starting the clock when his or her turn begins and stopping it when it ends.
9. The system shall setup the chessboard in the exact way specified in the chess rules.
10. The system shall always allow the white player to move first.
11. The system shall display hints to help players learn the best move to make.

## Undo Requirements

1. The system shall allow the players to undo all moves.
2. The system shall allow the player to redo all their undone moves.

## User Interface Requirements

1. The system shall accurately display the chessboard for the game.
2. The system shall allow the user to choose the style of the chess pieces.
3. The system shall render all pieces correctly.
4. The system shall display the move history.
5. The system shall display the pieces captured by each player.
6. The system shall display the play clock for each player.

# Static Code Review

* 1. **Process Overview**. The Code Analysis feature of Visual Studio performs static code analysis to help developers identify potential design, globalization, interoperability, performance, security, and a host of other categories of potential problems. Code Analysis can be run manually at any time from within the Visual Studio IDE, or even setup to automatically run as part of a Team Build or check-in policy for Team Foundation Server.
  2. The testing team used the automated Code Analysis feature of Visual Studio to perform a static code analysis of the Chess application. This analysis was performed on the entire application code.

Once the code was loaded into Visual Studio, a code review rule set was selected from the library of rules available in the Integrated Developed Environment (IDE). The Basic Correctness Rules rule set focuses on logic errors and common mistakes in the usage of framework APIs. As this rule set was appropriate for the static code review, this set was chosen for the review. The complete set of rules is provided in Appendix B.

**Static Analysis Result Report.** The analysis tool identified 61 violations, where the actual code implementation did not meet the intent of one of the code review rule set items. These violations were to be corrected by the development team.

The code violations list is shown below:

1. CA1008 **Enums should have zero value**  
   Remove all members that have the value zero from 'ChessBoard.PieceE' except for one member that is named 'None'.  
   **SrcChess2** - ChessBoard.cs (Line 20)
2. CA1303 **Do not pass literals as localized parameters**  
   Method 'ChessBoardControl.BoardDesignMode.set(bool)' passes a literal string as parameter 'caption' of a call to 'MessageBox.Show(string, string, MessageBoxButton)'. Retrieve the following string(s) from a resource table instead: "SrcChess".
3. CA2202 **Do not dispose objects multiple times**  
   Object 'stream' can be disposed more than once in method 'ChessBoardControl.LoadFromFile()'. To avoid generating a System.ObjectDisposedException you should not call Dispose more than one time on an object.: Lines: 906  
   **SrcChess2** - ChessBoardControl.xaml.cs (Line 906)
4. CA1303 **Do not pass literals as localized parameters**  
   Method 'ChessBoardControl.SavePGNToFile()' passes a literal string as parameter 'caption' of a call to 'MessageBox.Show(string, string, MessageBoxButton)'. Retrieve the following string(s) from a resource table instead: "Saving to PGN File".  
   **SrcChess2** - ChessBoardControl.xaml.cs (Line 978)
5. CA2202 **Do not dispose objects multiple times**  
   Object 'stream' can be disposed more than once in method 'ChessBoardControl.SavePGNToFile()'. To avoid generating a System.ObjectDisposedException you should not call Dispose more than one time on an object.: Lines: 997  
   **SrcChess2** - ChessBoardControl.xaml.cs (Line 997)
6. CA1033 **Interface methods should be callable by child types**  
   Make 'ChessBoardControl' sealed (a breaking change if this class has previously shipped), implement the method non-explicitly, or implement a new method that exposes the functionality of 'SearchEngine.ITrace.TraceSearch(int, ChessBoard.PlayerColorE, ChessBoard.MovePosS, int)' and is visible to derived classes.  
   **SrcChess2** - ChessBoardControl.xaml.cs (Line 1251)
7. CA1009 **Declare event handlers correctly**  
   Declare the second parameter of 'Action<Color>' as an EventArgs, or an instance of a type that extends EventArgs, named 'e'.  
   **SrcChess2** - CustomColorPicker.xaml.cs (Line 16)
8. CA1009 **Declare event handlers correctly**  
   Declare the first parameter of 'Action<Color>' as an object named 'sender'.  
   **SrcChess2** - CustomColorPicker.xaml.cs (Line 16)
9. CA1303 **Do not pass literals as localized parameters**  
   Method 'ColorPicker.MakeColorFromHex(object)' passes a literal string as parameter 'value' of a call to 'TextBox.Text.set(string)'. Retrieve the following string(s) from a resource table instead: "#".  
   **SrcChess2** - CustomColorSelector.xaml.cs (Line 210)
10. CA1303 **Do not pass literals as localized parameters**  
    Method 'ColorPicker.txt\_TextChanged(object, RoutedEventArgs)' passes a literal string as parameter 'value' of a call to 'TextBox.Text.set(string)'. Retrieve the following string(s) from a resource table instead: "255".  
    **SrcChess2** - CustomColorSelector.xaml.cs (Line 225)
11. CA1303 **Do not pass literals as localized parameters**  
    Method 'ColorPicker.UpdatePreview()' passes a literal string as parameter 'value' of a call to 'TextBox.Text.set(string)'. Retrieve the following string(s) from a resource table instead: "#".  
    **SrcChess2** - CustomColorSelector.xaml.cs (Line 414)
12. CA1303 **Do not pass literals as localized parameters**  
    Method 'frmSearchMode.frmSearchMode(SearchEngine.SearchMode, BoardEvaluationUtil)' passes a literal string as parameter 'value' of a call to 'TextBox.Text.set(string)'. Retrieve the following string(s) from a resource table instead: "15".  
    **SrcChess2** - frmSearchMode.xaml.cs (Line 68)
13. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.NewGameCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 54)
14. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_New Game".  
    **SrcChess2** - MainWindow.xaml.cs (Line 54)
15. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.LoadGameCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 56)
16. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.CreateGameCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 58)
17. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Create Game".  
    **SrcChess2** - MainWindow.xaml.cs (Line 58)
18. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.SaveGameCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 60)
19. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Save Game".  
    **SrcChess2** - MainWindow.xaml.cs (Line 60)
20. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.SaveGameInPGNCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 62)
21. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "Save Game \_To PGN".  
    **SrcChess2** - MainWindow.xaml.cs (Line 62)
22. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.QuitCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 64)
23. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Quit".  
    **SrcChess2** - MainWindow.xaml.cs (Line 64)
24. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.HintCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 67)
25. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Hint".  
    **SrcChess2** - MainWindow.xaml.cs (Line 67)
26. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.UndoCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 69)
27. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Undo".  
    **SrcChess2** - MainWindow.xaml.cs (Line 69)
28. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.RedoCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 71)
29. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Redo".  
    **SrcChess2** - MainWindow.xaml.cs (Line 71)
30. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.RevertBoardCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 73)
31. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "Revert \_Board".  
    **SrcChess2** - MainWindow.xaml.cs (Line 73)
32. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.PlayerAgainstPlayerCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 75)
33. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Player Against Player".  
    **SrcChess2** - MainWindow.xaml.cs (Line 75)
34. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.AutomaticPlayCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 77)
35. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Automatic Play".  
    **SrcChess2** - MainWindow.xaml.cs (Line 77)
36. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.FastAutomaticPlayCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 79)
37. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Fast Automatic Play".  
    **SrcChess2** - MainWindow.xaml.cs (Line 79)
38. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.CancelPlayCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 81)
39. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Cancel Play".  
    **SrcChess2** - MainWindow.xaml.cs (Line 81)
40. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.DesignModeCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 83)
41. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Design Mode".  
    **SrcChess2** - MainWindow.xaml.cs (Line 83)
42. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.SearchModeCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 86)
43. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Search Mode...".  
    **SrcChess2** - MainWindow.xaml.cs (Line 86)
44. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.FlashPieceCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 88)
45. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Flash Piece".  
    **SrcChess2** - MainWindow.xaml.cs (Line 88)
46. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.PGNNotationCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 90)
47. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_PGN Notation".  
    **S**CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.BoardSettingCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 92)
48. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Board Settings...".  
    **SrcChess2** - MainWindow.xaml.cs (Line 92)
49. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.CreateBookCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 95)
50. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Create a Book...".  
    **SrcChess2** - MainWindow.xaml.cs (Line 95)
51. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.FilterPGNFileCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 97)
52. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Filter a PGN File...".  
    **SrcChess2** - MainWindow.xaml.cs (Line 97)
53. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.TestBoardEvaluationCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 99)
54. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_Test Board Evaluation...".  
    **SrcChess2** - MainWindow.xaml.cs (Line 99)
55. CA2104 **Do not declare read only mutable reference types**  
    Remove the read-only designation from 'MainWindow.AboutCommand' or change the field to one that is an immutable reference type. If the reference type 'RoutedUICommand' is, in fact, immutable, exclude this message.  
    **SrcChess2** - MainWindow.xaml.cs (Line 102)
56. CA1303 **Do not pass literals as localized parameters**  
    Method 'MainWindow.MainWindow()' passes a literal string as parameter 'text' of a call to 'RoutedUICommand.RoutedUICommand(string, string, Type)'. Retrieve the following string(s) from a resource table instead: "\_About...".  
    **SrcChess2** - MainWindow.xaml.cs (Line 102)
57. CA1033 **Interface methods should be callable by child types**  
    Make 'MainWindow' sealed (a breaking change if this class has previously shipped), implement the method non-explicitly, or implement a new method that exposes the functionality of 'ChessBoardControl.IMoveListUI.Reset(ChessBoard)' and is visible to derived classes.  
    **SrcChess2** - MainWindow.xaml.cs (Line 543)
58. CA1033 **Interface methods should be callable by child types**  
    Make 'MainWindow' sealed (a breaking change if this class has previously shipped), implement the method non-explicitly, or implement a new method that exposes the functionality of 'ChessBoardControl.IMoveListUI.NewMoveDone(int, int, int)' and is visible to derived classes.  
    **SrcChess2** - MainWindow.xaml.cs (Line 554)
59. CA1033 **Interface methods should be callable by child types**  
    Make 'MainWindow' sealed (a breaking change if this class has previously shipped), implement the method non-explicitly, or implement a new method that exposes the functionality of 'ChessBoardControl.IMoveListUI.RedoPosChanged()' and is visible to derived classes.  
    **SrcChess2** - MainWindow.xaml.cs (Line 567)
60. CA1820 **Test for empty strings using string length**  
    Replace the call to 'string.operator ==(string, string)' in 'PgnUtil.GetNextNonEmptyLine(TextReader)' with a call to 'String.IsNullOrEmpty'.  
    **SrcChess2** - PgnUtil.cs (Line 158)

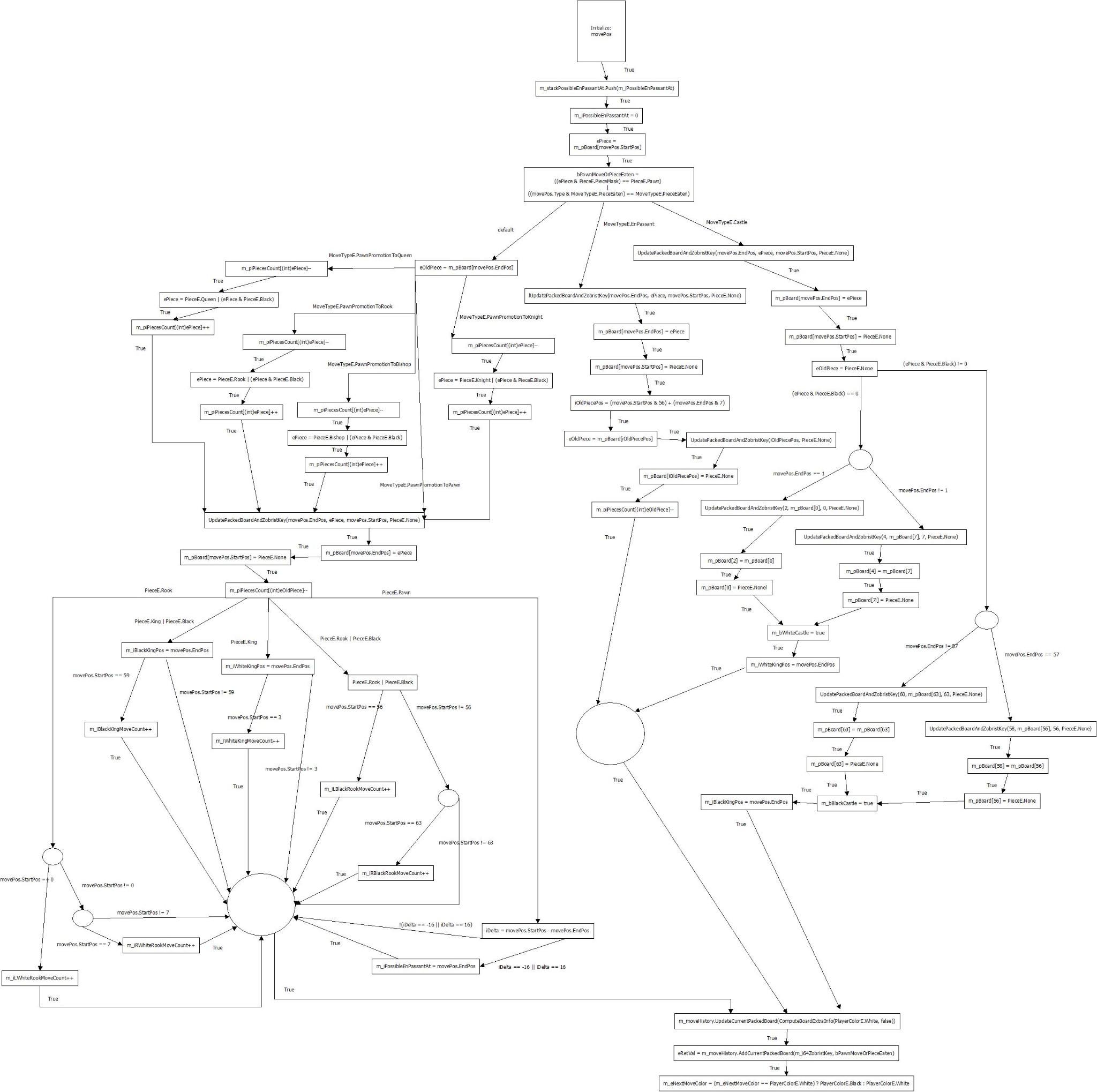
CA1008 **Enums should have zero value**  
In enum 'SearchEngine.SearchMode.OptionE', change the name of 'SearchEngine.SearchMode.OptionE.UseMinMax' to 'None'.  
**SrcChess2** - SearchEngine.cs (Line 47)

# Test Cases/Results

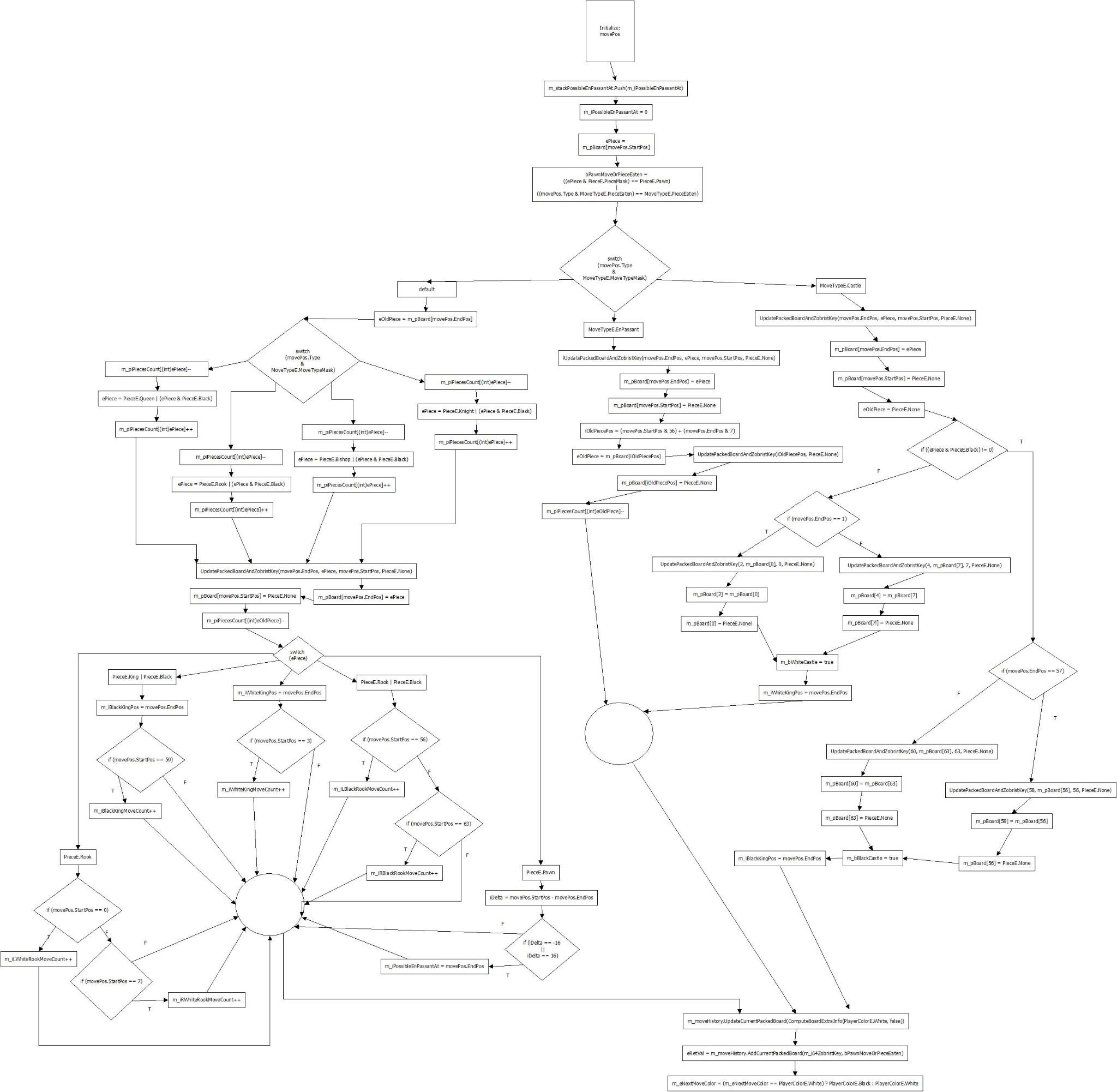
## Structural Tests

We created data and control flow diagrams for some key methods in the system. These diagrams were then used to derive test cases.

### Data Flow Diagram

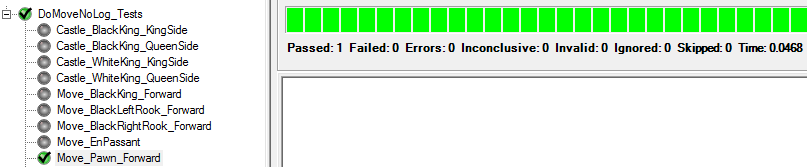


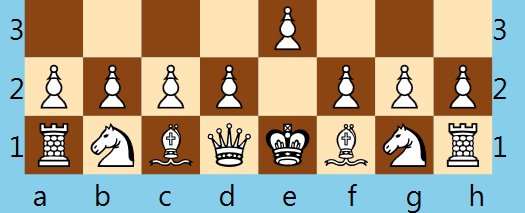
### Control Flow Diagram



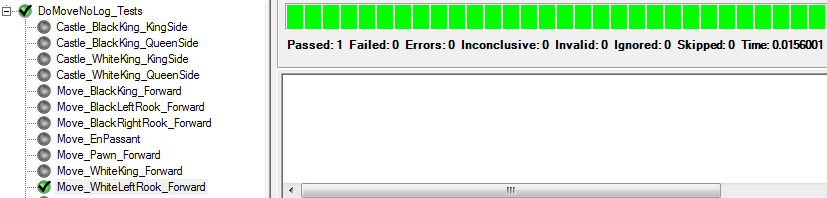
### Control Flow and Data Flow Tests

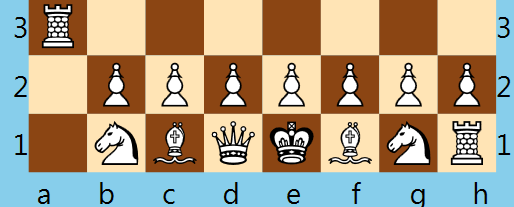
#### Moving Pawn Forward



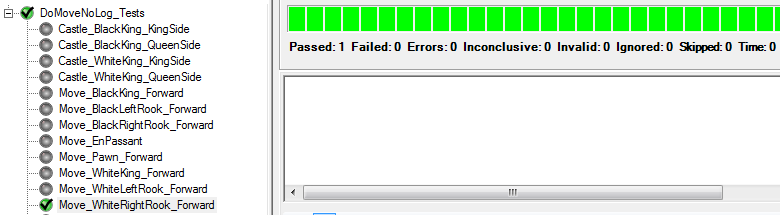


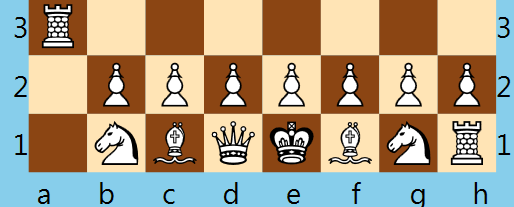
#### Moving Left Rook Forward



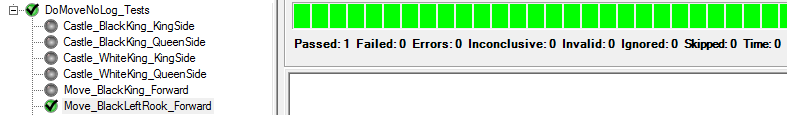


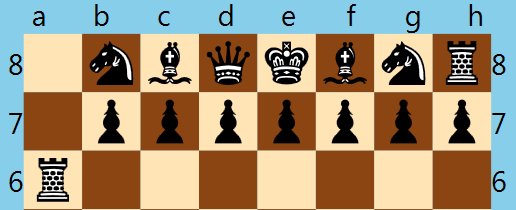
#### Moving Right Took Forward



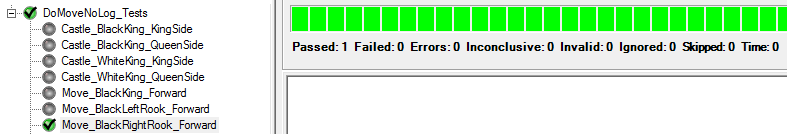


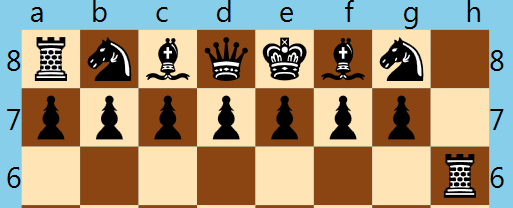
#### Moving Left Rook | Black Forward



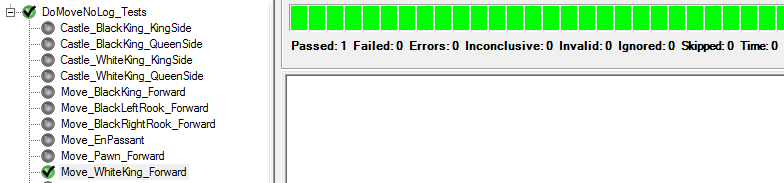


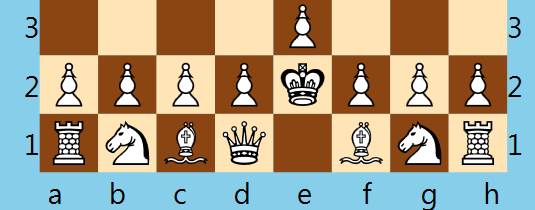
#### Moving Right Rook | Black Forward



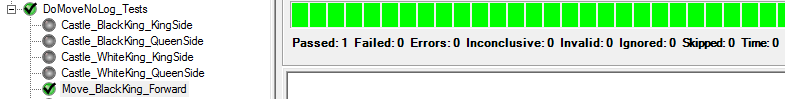


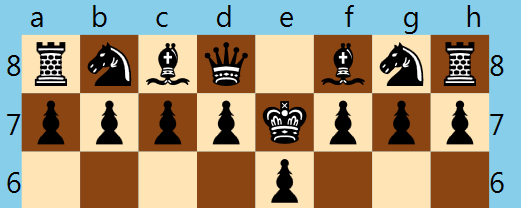
#### Moving White King Forward



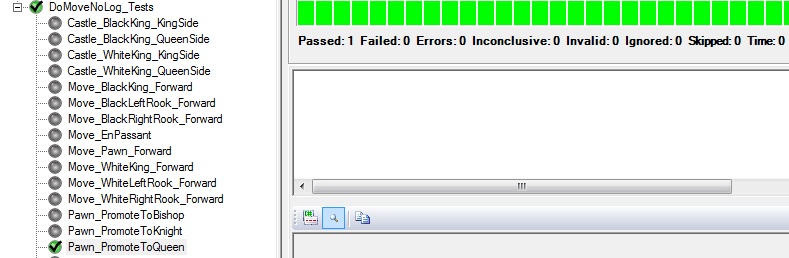


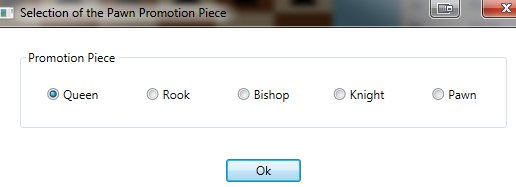
#### Moving Black King Forward

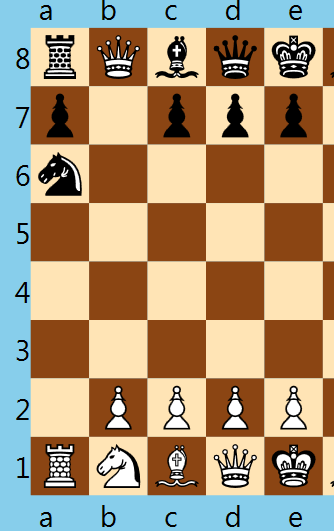




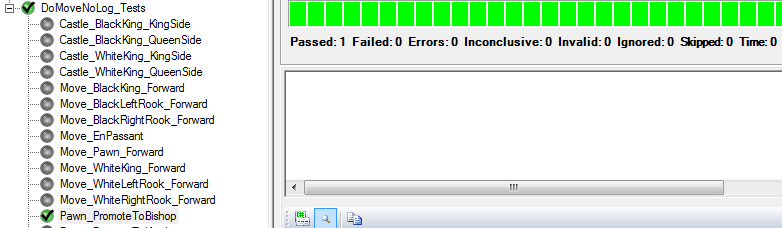
#### Pawn Promotion to Queen

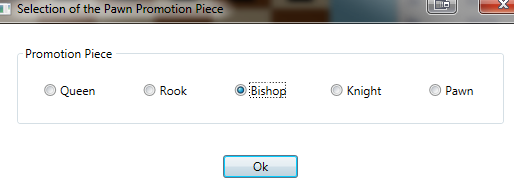


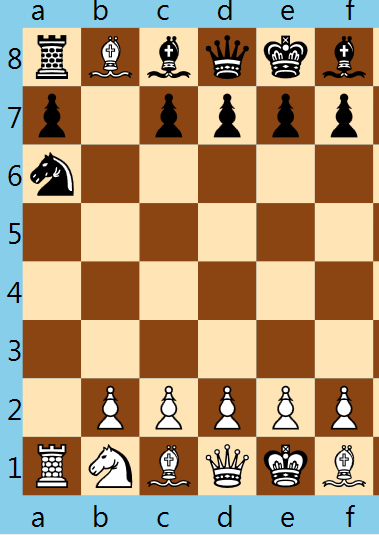




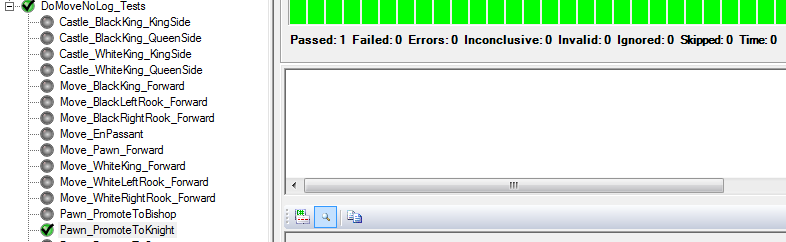
#### Pawn Promotion to Bishop

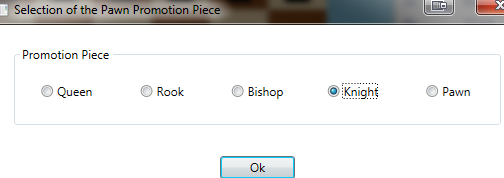






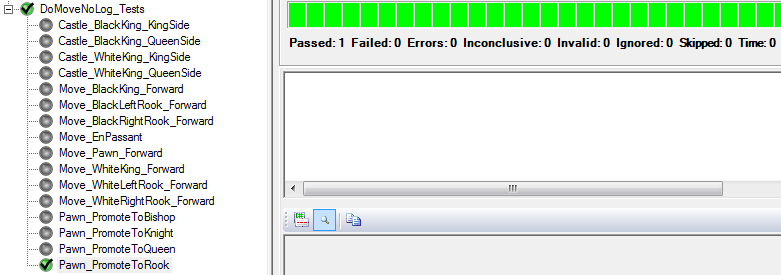
#### Pawn Promotion to Knight

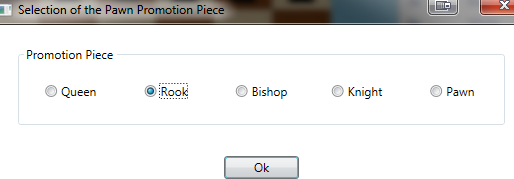


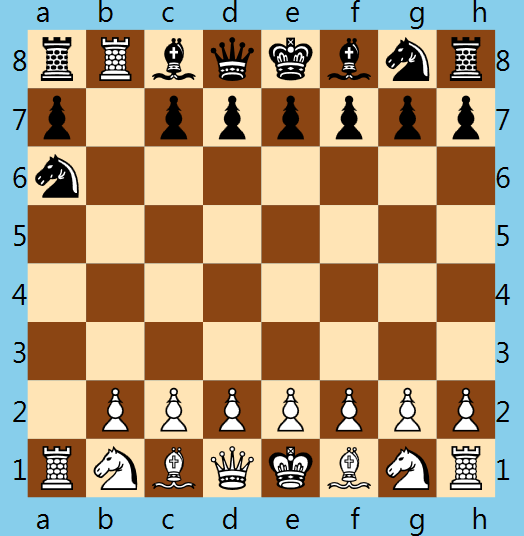




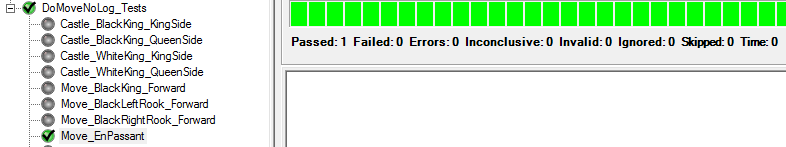
#### Pawn Promotion to Rook

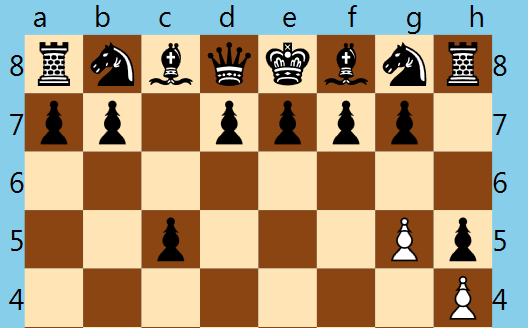


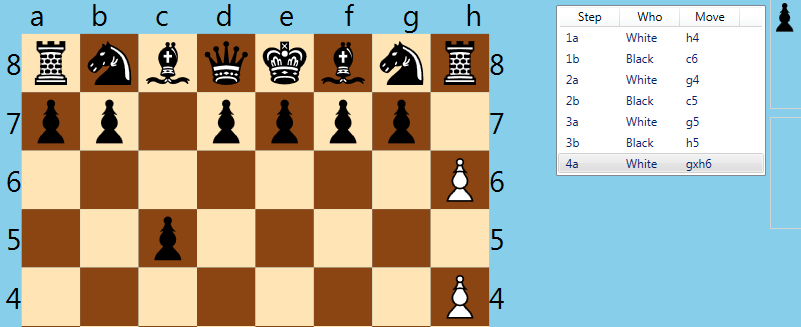




#### Moving EnPassant

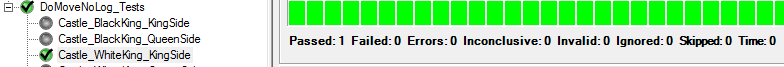


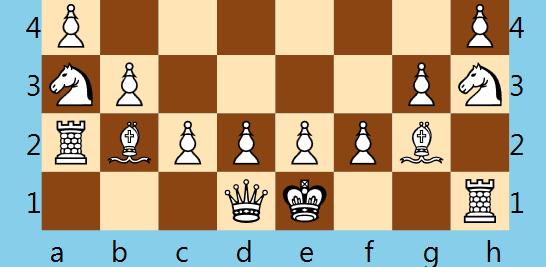


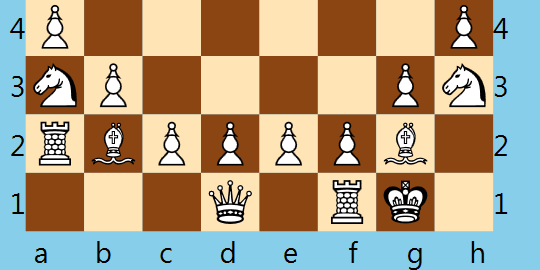


#### Move Castling

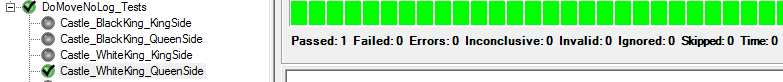
##### White King – King side

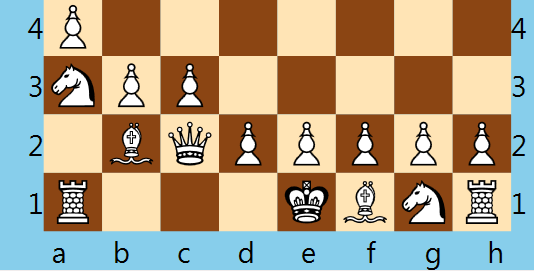


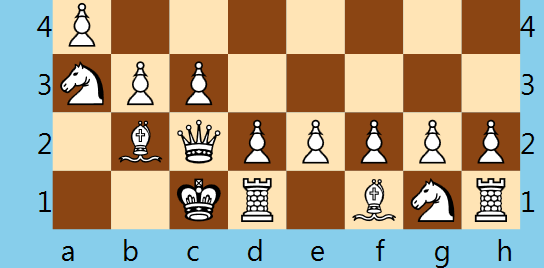




##### White King – Queen side

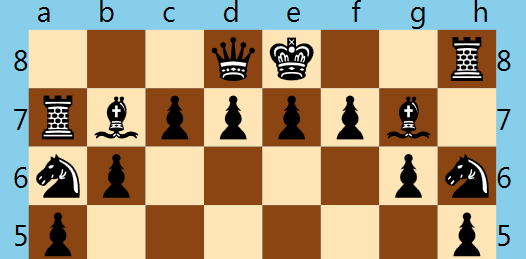


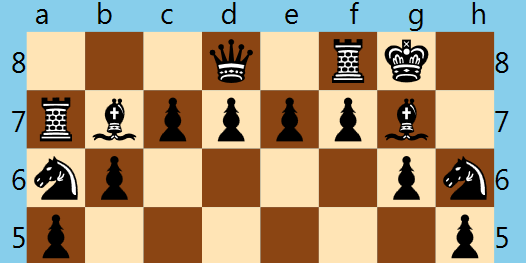




##### Black King – King side

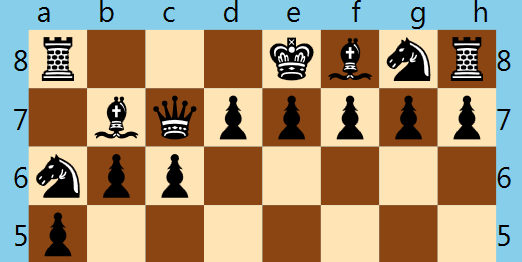


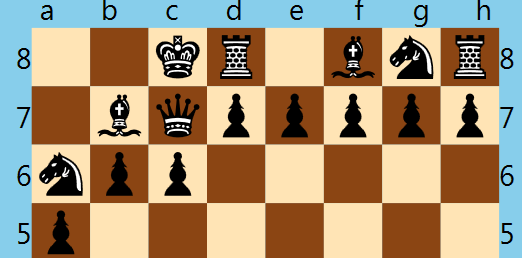




##### Black King – Queen side







## System Tests

### Functional Tests

#### Equivalence Class Partitioning

1. **Test:** Verification of Correct Chess Piece Movement (non-Capture)
2. **Specification:**

The non-capture movements of the various chess pieces were designated in the following shorthand manner:

|  |  |
| --- | --- |
| **Table 1. Movement Definitions** | |
| Short hand descriptor | Meaning |
| F | Forward – movement from the player toward opposing player |
| B | Backward – movement from the opposing player toward the player |
| L | Left – movement to the player’s left |
| R | Right – movement to the player’s right |
| H | Horizontal – straight-line movement left to right or right to left |
| V | Vertical – straight-line movement forward to back or back to forward |
| M | Multiple – the number of squares that can be moved in one turn = {1,2,3,4,5,6,7} |
| S | Single - limited to a single integer square that can be moved |
| 1 or 2 | 1 or 2 squares movement |
| O | Omni-direction – piece is capable of all movements: SL or D, except LS |
| SL | Straight-Line movement only – equal to VF or VB or HR or HL |
| D | Diagonal movement only – equal to FL or FR or BL or BR |
| LS | L-Shape movement – equal to B2LH1, B2RH1, B1LH2, B1RH2, F2LH1, F2RH1, F1LH2, or F1RH2 |

Examples: FS = forward, single square: the motion of a pawn or king (forward 1 space)

BVM = backward, vertical, multiple spaces allowed: the motion of a rook or queen

B2LH1 = backward 2 space, left horizontal 1 space: the motion of a knight

The chess pieces were classified by a movement class as shown in Table 2. For clarification, it was noted that although a Queen could be classified as any of the Movement Classes below except the Knight, the Movement Class is designated as “only” that type of mover. For example, the Queen is not a D+M only mover, such as a Bishop would be.

| **Table 2. Movement Classes by Chess Piece** | | | |
| --- | --- | --- | --- |
| Movement Class | Chess Piece | Number of Pieces | Class Alias |
| O+M-capable movers | Queen | 1-white, 1-black | Class Q |
| O+S-limited movers | King | 1-white, 1-black | Class KG |
| SL+M only movers | Rook | 2-white, 2-black | Class R |
| D+M only movers | Bishop | 2-white, 2-black | Class B |
| VF+S only movers | Pawn | 8-white, 8-black | Class P |
| LS only movers | Knight | 2-white, 2-black | Class K |

1. **Derivation of Test Case Input Sets:**
2. **Derived Equivalence Class Partitions**

|  |  |  |
| --- | --- | --- |
| **Table 3. Derived Equivalence Classes** | | |
| Equivalence Class | Equivalence Class Movement | Acceptable Movement? |
| EC1 | Class Q moving multi-spaces in omni-direction | Valid |
| EC2 | Non Class Q moving multi-spaces in omni-directions | Invalid |
| EC3 | Class KG moving a single space in omni-directions | Valid |
| EC4 | Class KG moving multi-spaces in omni-directions | Invalid |
| EC5 | Class R moving multi-spaces in a straight-line direction | Valid |
| EC6 | Class R moving multi spaces in a diagonal direction | Invalid |
| EC7 | Non Class R or Q moving multi-spaces in a straight line direction | Invalid |
| EC8 | Class B moving multi-spaces in a diagonal direction | Valid |
| EC9 | Class B moving multi spaces in a straight-line direction | Invalid |
| EC10 | Non Class B or Q moving multi-spaces in a diagonal direction | Invalid |
| EC11 | Class P moving one space forward | Valid |
| EC12 | Class P moving one space backward | Invalid |
| EC13 | Class P moving one space left | Invalid |
| EC14 | Class P moving one space right | Invalid |
| EC15 | Class K moving in L-shape | Valid |
| EC16 | Non Class K moving in L-shape | Invalid |

For simplification of the test, it was decided to rotate the class color to test the variation that could exist between white and black chess pieces.

1. **Black Box Test Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 4. Test Cases** | | | |
| **Test Case Id** | **Test Data** | **Expected Outcome** | **EC Classes Covered** |
| TC1 | white Q move M+D | Pass | EC1 |
| TC2 | black Q move M+SL | Pass | EC1 |
| TC3 | white B move M+SL | Fail | EC2, EC7, EC9 |
| TC4 | black B move M+D | Pass | EC8 |
| TC5 | white KG move S+D | Pass | EC3 |
| TC6 | black KG move S+SL | Pass | EC3 |
| TC7 | white KG move M+D | Fail | EC2, EC4, EC10 |
| TC8 | black KG move M+SL | Fail | EC2, EC4, EC7 |
| TC9 | white R move M+SL | Pass | EC5 |
| TC10 | black R move M +D | Fail | EC6, EC10 |
| TC11 | white P move S+SL+F | Pass | EC11 |
| TC12 | black P move S+SL+B | Fail | EC12 |
| TC13 | white P move S+H+L | Fail | EC13 |
| TC14 | white P move S+H+R | Fail | EC14 |
| TC15 | black K move LS | Pass | EC15 |
| TC16 | black Q move LS | Fail | EC16 |

1. Test Results

|  |  |  |
| --- | --- | --- |
| **Table 5.** | | |
| **Test Case Id** | **Expected Result** | **Result Obtained** |
| TC1 | Pass | Pass |
| TC2 | Pass | Pass |
| TC3 | Fail | Fail |
| TC4 | Pass | Pass |
| TC5 | Pass | Pass |
| TC6 | Pass | Pass |
| TC7 | Fail | Fail |
| TC8 | Fail | Fail |
| TC9 | Pass | Pass |
| TC10 | Fail | Fail |
| TC11 | Pass | Pass |
| TC12 | Fail | Fail |
| TC13 | Fail | Fail |
| TC14 | Fail | Fail |
| TC15 | Pass | Pass |
| TC16 | Fail | Fail |

1. **Conclusion**

Functional testing using an equivalence class partitioning is technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In this process, test cases were designed to cover each partition at least once. This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed. An advantage of this approach is reduction in the time required for testing a software due to a lesser required number of test cases. In this test, 16 test cases were constructed to test correct non-capture movement of all of the chess pieces. The test yielded successful results in all cases. The test also indicated that regardless of the chess piece color correct movement occurred.

#### Pairwise Tests

1. **Test:** Verification of Opposing Player Capture by Pieces That Can Capture in a Straight Line
2. **Specification:**

The chess pieces have been assigned a particular movement that is allowed to capture the opposing chess pieces. The movements have been designated in the following shorthand manner:

|  |  |
| --- | --- |
| Short hand descriptor | Meaning |
| F | Forward – movement from the player toward opposing player |
| B | Backward – movement from the opposing player toward the player |
| L | Left – movement to the player’s left |
| R | Right – movement to the player’s right |
| H | Horizontal – straight-line movement left to right or right to left |
| V | Vertical – straight-line movement forward to back or back to forward |
| M | Multiple – the number of spaces that can be moved in one turn = {1,2,3,4,5,6,7,8} |
| {integer value, e.g. 1 or 2} | limited to integer space that can be moved |

Examples: FL1 = forward, left, 1 space: the capture motion of a pawn (diagonal 1 space)

BVM = backward, vertical, multiple spaces allowed: the capture motion of a rook or queen

B2LH1 = backward 2 space, left horizontal 1 space: the capture motion of a knight

1. **Derivation of Test Case Input Sets:**

For this test, the specific capture motion that will be evaluated is specific to those pieces that can capture in a straight line, either horizontal or vertical.

1. **Various Values that Need to Be Tested In Combinations**

|  |  |
| --- | --- |
| Variables - Factors [ALIAS LETTER} | Values |
| Chess piece [A] | Rook [0], Queen [1], King [2] |
| Chess piece position on board [B] | Right [0], Left [1] |
| Chess piece color [C] | White [0], Black [1] |
| Acceptable capture motion [D] | FVM [0], BVM [1], RHM [2], LHM [3] |
|  |  |

For all-testing, this would require: 3 \* 2 \* 2 \* 4 = 48 test cases

1. **Orthogonal Array Model**

There are four independent variables (A, B, C, D). Each variable can take at most 4 values. From [ref] the following orthogonal array L16(44) was chosen.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 0 0 0 | 0 1 2 3 | 0 1 2 3 | 0 1 2 3 | 0 1 2 3 |
| 1 1 1 1 | 0 1 2 3 | 1 0 3 2 | 2 3 0 1 | 3 2 1 0 |
| 2 2 2 2 | 0 1 2 3 | 2 3 0 1 | 3 2 1 0 | 1 0 3 2 |
| 3 3 3 3 | 0 1 2 3 | 3 2 1 0 | 1 0 3 2 | 2 3 0 1 |

1. Mapping the Orthogonal Array

|  | **Factors** | | | |
| --- | --- | --- | --- | --- |
| **Test Case Id** | **A** | **B** | **C** | **D** |
| TC1 | Rook | Right | White | FVM |
| TC2 | Rook | Left | Black | BVM |
| TC3 | Rook | Left | White | RHM |
| TC4 | Rook | Right | Black | LHM |
| TC5 | Queen | Right | Black | RHM |
| TC6 | Queen | Left | White | LHM |
| TC7 | Queen | Left | White | FVM |
| TC8 | Queen | Right | Black | BVM |
| TC9 | King | Right | White | LHM |
| TC10 | King | Left | Black | RHM |
| TC11 | King | Right | White | BVM |
| TC12 | King | Left | Black | FVM |
| TC13 | Rook | Right | White | BVM |
| TC14 | Queen | Left | Black | FVM |
| TC15 | King | Left | Black | LHM |
| TC16 | Rook | Right | White | RHM |

Using Pair-wise testing based on an orthogonal array, 16 test cases will be used in place of 48

1. Test Results

| **Test Results** | | |
| --- | --- | --- |
| **Test Case Id** | **Expected Result** | **Result Obtained** |
| TC1 | Pass | Pass |
| TC2 | Pass | Pass |
| TC3 | Pass | Pass |
| TC4 | Pass | Pass |
| TC5 | Pass | Pass |
| TC6 | Pass | Pass |
| TC7 | Pass | Pass |
| TC8 | Pass | Pass |
| TC9 | Pass | Pass |
| TC10 | Pass | Pass |
| TC11 | Pass | Pass |
| TC12 | Pass | Pass |
| TC13 | Pass | Pass |
| TC14 | Pass | Pass |
| TC15 | Pass | Pass |
| TC16 | Pass | Pass |

#### Decision Table Based Testing

##### Stalemate detection

The chess program has a stalemate detection algorithm to detect when the game is winnable. This mechanism ensures that the game stops if there is a stalemate. We applied decision table based testing to test this algorithm.



Figure : The stalemate detection detected a stalemate.

Based on the rules of chess, the game can continue are:

1) Either player has at least 1 pawn.

2) There is at least 1 player with a queen or rook.

3) One player has at least two bishops and knights (combined).

4) Otherwise a stalemate is occurring.

For our purposes, we define key pieces as Bishops and Knights and big pieces as Queens and Rooks. Because some of these are combined in the logic the inputs really break down to the following:

Inputs:

1) Number of pawns

2) White knights & White Bishops -> White Key pieces

3) Black knights & Black Bishops -> White Key Pieces

4) Number of Big pieces (white & black rooks and queens)

Rules can be better defined as:

1) There is at least 1 pawn on the board.

2) There is at least 1 big piece on the board.

3) Black has 2 key pieces

4) White has two key pieces

Based on these rules we can create a decision table for testing.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Rules or Combinations | | | | | | | | | | | | | | | |
| Conditions | Values | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| C1: Number of pawns > 0 | T, F | T | T | T | T | T | T | T | T | F | F | F | F | F | F | F | F |
| C2: Number of white key pieces > 2 | T, F | T | T | T | T | F | F | F | F | T | T | T | T | F | F | F | F |
| C3: Number of black key pieces > 2 | T, F | T | T | F | F | T | T | F | F | T | T | F | F | T | T | F | F |
| C4: Number of big pieces > 1 | T, F | T | F | T | F | T | F | T | F | T | F | T | F | T | F | T | F |
| Effects |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E1: Checkmate is possible |  | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |  |
| E2: Checkmate impossible |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |

From this, we see there is only one case where checkmate is not possible. It is the

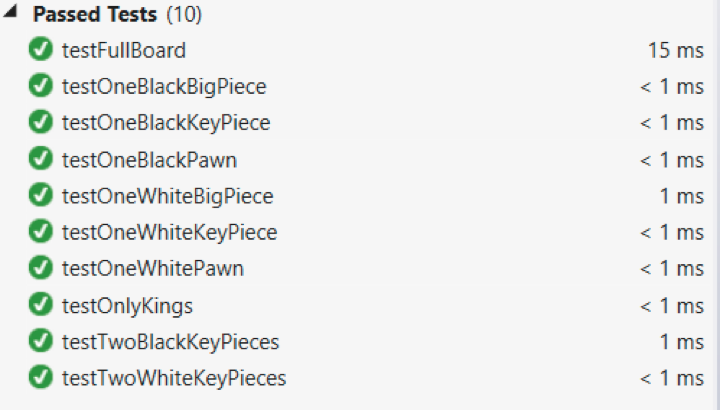
From this, we see there is only one case where checkmate is not possible. It is the case where all four conditions are false. Based on this, the table can be reduced using don’t care values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Rules or Combinations |  |  |  |  |
| Conditions | Values | 1 | 2 | 3 | 4 | 5 |
| C1: Number of pawns > 0 | T, F, - | T | - | - | - | F |
| C2: Number of white key pieces > 2 | T, F, - | - | T | - | - | F |
| C3: Number of black key pieces > 2 | T, F, - | - | - | T | - | F |
| C4: Number of big pieces > 1 | T, F, - | - | - | - | T | F |
| Effects |  |  |  |  |  |  |
| E1: Checkmate is possible |  | X | X | X | X |  |
| E2: Checkmate impossible |  |  |  |  |  | X |

To test the stalemate behavior based on the decision table above, 10 NUnit unit tests were developed. These 10 tests cover the entire decisions table. All ten tests passed.

| Test Case | Description | Result |
| --- | --- | --- |
| testFullBoard | Tests that there is no stalemate for a full chess board. | Passed |
| TestOneBlackKeyPiece | Tests that there is no stalemate if there is one black key piece. | Passed |
| TestOneWhiteKeyPiece | Tests that there is no stalemate if there is one white key piece. | Passed |
| testOneBlackKeyPiece | Tests that there is a stalemate if there is only 1 black key piece. | Passed |
| testOneWhieKeyPiece | Tests that there is a stalemate if there is only 1 white key piece. | Passed |
| testOneBlackPawn | Tests that there is no stalemate if there is one black pawn. | Passed |
| testOneWhitePawn | Tests that there is no stalemate if there is one white pawn. | Passed |
| testTwoBlackKeyPeices | Tests that there is no stalemate if there are two black key pieces. | Passed |
| testTwoWhiteKeyPeices | Tests that there is no stalemate if there are two white key pieces. | Passed |
| testOnlyKings | Tests that there is a stalemate if only a black king and white king remain. | Passed |

When run, we saw all ten of the tests pass. This allowed us to determine that the stalemate detection was work.



### Random Testing

1. **Test:** Verification that configuration of the chess piece font set changes correctly.
2. **Specification:** The chess game provides the capability to select from a large set of chess piece sets. In essence, the chess pieces are assigned a particular font from a library of standard chess pieces, i.e. chess piece styles that are commonly used among chess piece sets sold by vendors and identified as standard by the various chess organizations. This selection list is made available on the Options | Piece Set dropdown listbox. The domain for the chess piece sets is the set of chess pieces fonts made available in the dropdown listbox. The domain set consists of these fonts:

| **Table 1. Domain Set of Chess Piece Sets (Fonts)** | |
| --- | --- |
| **Set Number** | **Chess Piece Sets Made Available by the Application** |
| 1 | Adventurer |
| 2 | Alphonso-X |
| 3 | Alpha |
| 4 | Alpha-2 |
| 5 | Arial Unicode |
| 6 | Berlin |
| 7 | Cases |
| 8 | Chess-7 |
| 9 | Condal |
| 10 | Harlequin |
| 11 | Kingdom |
| 12 | Leipzig |
| 13 | Line |
| 14 | Lucena |
| 15 | Magnetic |
| 16 | Maya |
| 17 | Mediaeval |
| 18 | Merida |
| 19 | Millennia |
| 20 | Motif |
| 21 | MS Mincho |
| 22 | Plain |
| 23 | Segoe ui Symbol |
| 24 | Smart |
| 25 | Traveller Standard |
| 26 | Usual (aka Regular or Classic) |

1. **Derivation of Test Case Input Sets:** For this test, the test inputs were selected independently of the domain. To do so, the Set Numbers (above) were used to randomly select 10 chess piece set numbers. From the random selection process, the following were chosen as the Test Case Input Set:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table 2. Randomly Selected Chess Piece Sets (Fonts)** | | | | |
| 17 | 3 | 10 | 8 | 19 |
| 7 | 13 | 2 | 12 | 6 |

* 1. **Standard (or Specification) for the Chess Piece Font**

| **Table 3. Standard Chess Piece Set Fonts** | | |
| --- | --- | --- |
| Test Case Number | Chess Piece Font Style | Standard (or Specification) |
| 17 | Mediaeval | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessPiecesStyles\mediaeval.PNG |
| 3 | Alpha | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessPiecesStyles\alpha.PNG |
| 10 | Harlequin | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessPiecesStyles\harlequin.PNG |
| 8 | Chess-7 | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessPiecesStyles\chess7.PNG |
| 19 | Millennia | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessPiecesStyles\milennia.PNG |
| 7 | Cases | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessPiecesStyles\Cases.PNG |
| 13 | Line | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessPiecesStyles\line.PNG |
| 2 | Alphonso-X | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessPiecesStyles\alfonsoX.PNG |
| 12 | Leipzig | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessPiecesStyles\leipzig.PNG |
| 6 | Berlin | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessPiecesStyles\berlin.PNG |

\*Note: the Standard was taken from a web-based chess organization, the fonts were provided at URL: <http://www.enpassant.dk/chess/fonteng.htm>

* 1. **Test Method**

Each of the Test Cases designated in Table 3 was selected from the dropdown listbox provided by the application during execution. After selection, a screen capture was taken and imported into the Test Results table (below).

* 1. **Test Oracle**

The screen capture provides a depiction of the appearance of the chess pieces post piece set configuration. The appearance of the chess pieces were compared to standard appearance for the chess set pieces in Table 3. A member of Team 4 then assessed whether the appearance during execution of the game matched the standard appearance provided in Table 3.

1. **Test Results**

| **Table 4.** | | | |
| --- | --- | --- | --- |
| **Test Case Number** | **Chess Piece Set from application Listbox** | **Screen Depiction** | **Test Result** |
| 17 | Mediaeval | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessGameViews2\Meidaeval.PNG | Pass |
| 3 | Alpha | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessGameViews2\alpha.PNG | Pass |
| 10 | Harlequin | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessGameViews2\harlequin.PNG | Pass |
| 8 | Chess-7 | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessGameViews2\chess7.PNG | Pass |
| 19 | Millennia | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessGameViews2\millennia.PNG | Pass |
| 7 | Cases | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessGameViews2\Cases.PNG | Pass |
| 13 | Line | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessGameViews2\line.PNG | Pass |
| 2 | Alphonso-X | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessGameViews2\alfonsoX.PNG | Pass |
| 12 | Leipzig | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessGameViews2\leipzig.PNG | Pass |
| 6 | Berlin | G:\PSU\2013\SWENG581 SoftwareTesting\Week 4\chessGameViews2\berlin.PNG | Pass |

1. **Conclusion**

Functional testing using a random testing approach provides an estimate of the reliability of the software. In this test, ten randomly selected test cases yielded successful results in all cases. The test indicates that ten of 26 domain set members were configured and provided correctly by the application software. These test results provide confidence that all 26 chess piece fonts were implemented and could be correctly selected by the user during execution of the application.

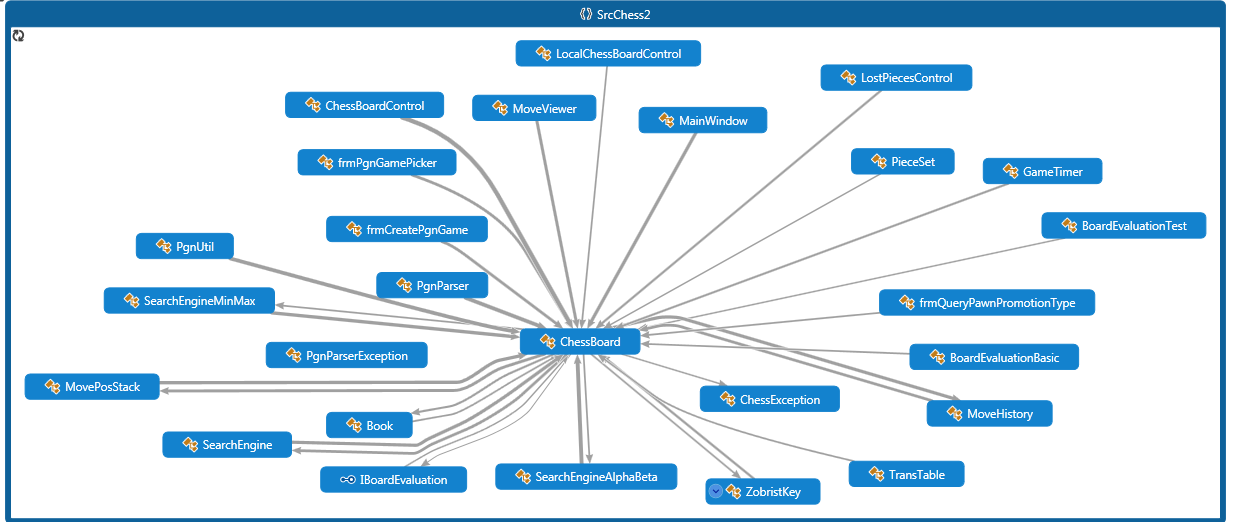
### Manual Unit Testing

We developed several unit tests for piece movement and ran them manually. These were some of the first tests we ran when we were simply trying to get a feel for the program.

See table in Appendix C.

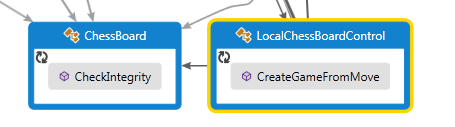
### Integration Testing

Overall Integration Tests between Chessboard Class:



**Integration Test 1:** Chessboard and LocalChessBoardControl Integration Test:

**Visual Representation:**



**Test:** Successfully pass CreateGameFromMove parameters through CheckIntegrity

**1:** Pass list of moves parameters to CheckIntegrity

ChessBoard chessBoardStarting,

List<ChessBoard.MovePosS>listMove,

ChessBoard.PlayerColorE eNextMoveColor,

string strWhitePlayerName,

string strBlackPlayerName,

PgnParser.PlayerTypeE eWhitePlayerType,

PgnParser.PlayerTypeE eBlackPlayerType,

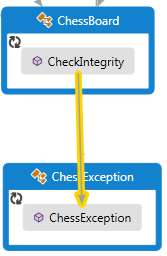
TimeSpan spanPlayerWhite,

TimeSpan spanPlayerBlack

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Integration Test 2:** Chessboard ChessExeception Integration Test:

Visual Representation:



**Test 2:** Successfully pass CheckIntegrity parameters through ChessException

Integrity issue is dispositioned

CheckIntegrity()

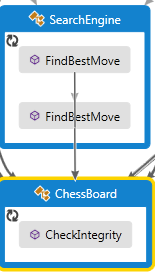
**Test 2a:** “Piece count mismatch” passed to ChessException

**Test 2b:** “King position mismatch” passed to ChessException

**Result:** Parameters successfully passed to ChessException and mismatch messages displayed

**Integration Test 3:** SearchEngine ChessBoard Integration Test:

Visual Representation:



**Test 3:** Successfully pass SearchEngine parameters through CheckIntegrity

Pass list of moves to CheckIntegrity

ChessBoard chessBoard,

SearchMode searchMode,

ChessBoard.PlayerColorE ePlayerColor,

out ChessBoard.MovePosS moveBest,

out int iPermCount,

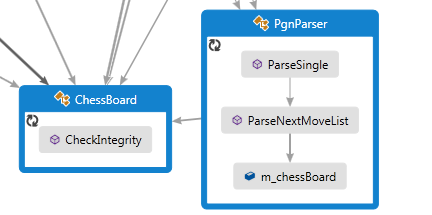
out int iCacheHit,

out int iMaxDepth

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Integration Test 4:** PgnParser ChessBoard Integration Test:

Visual Representation:



**Test:** Successfully pass ParseSingle, ParseNextMoveList, and m\_chessBoard parameters through CheckIntegrity

**Test 4a:** Pass associated ParseSingle PGN parameters to CheckIntegrity

string strText,

bool bIgnoreMoveListIfFEN,

List<ChessBoard.MovePosS> listMovePos,

out int iSkip,

out int iTruncated,

out ChessBoard chessBoardStarting,

out ChessBoard.PlayerColorE eStartingColor,

out string strWhitePlayerName,

out string strBlackPlayerName,

out PlayerTypeE eWhitePlayerType,

out PlayerTypeE eBlackPlayerType,

out TimeSpan spanWhitePlayer,

out TimeSpan spanBlackPlayer

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 4b:** Pass associated ParseNextMoveList parameters to CheckIntegrity

bIgnoreMoveListIfFEN,

out piMoveList,

out attrs,

listMovePos,

ref iSkip,

ref iTruncated,

out chessBoardStarting,

out eStartingColor,

out strWhitePlayerName,

out strBlackPlayerName,

out eWhitePlayerType,

out eBlackPlayerType,

out spanWhitePlayer,

out spanBlackPlayer

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 4c:** Pass associated m\_chessBoard parameters to CheckIntegrity and generate exception when a move cannot be resolved

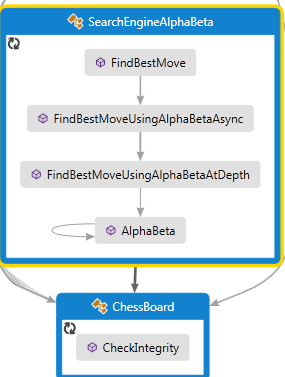
**Result:** Pass: Piece count mismatch identified

**Test 4d:** Pass associated m\_chessBoard parameters to CheckIntegrity and generate exception when a move cannot be resolved

**Result:** Pass: Kind position mismatch identified

**Integration Test:** SearchEngineAlphaBeta ChessBoard Integration Test:

Visual Representation:



**Test 5a:** Pass associated FindBestMove parameters to CheckIntegrity

ChessBoard chessBoard,

SearchEngine.SearchMode searchMode,

ChessBoard.PlayerColorE ePlayerColor,

List<ChessBoard.MovePosS> moveList,

int[] arrIndex,

ChessBoard.PosInfoS posInfo,

ref ChessBoard.MovePosS moveBest,

out int iPermCount,

out int iCacheHit,

out int iMaxDepth) {

bool bRetVal = false;

bool bMultipleThread;

bool bUseTransTable;

ChessBoard[] arrChessBoard;

Task<AlphaBetaResult>[] taskArray;

List<ChessBoard.MovePosS>[] arrMoveList;

AlphaBetaResult alphaBetaRes;

ChessBoard.PosInfoS posInfoWhite;

ChessBoard.PosInfoS posInfoBlack;

int iAlpha;

int iBeta;

int iThreadCount;

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 5b:** Pass associated FindBestMoveUsingAlphaBetaAsync parameters to CheckIntegrity

AlphaBetaResult resRetVal;

DateTime dtTimeOut;

int iDepth;

int iPermCountAtLevel;

int iPoint;

int iBestMoveIndex;

int iDepthLimit;

int[] arrPoints;

System.Threading.ThreadPriority eThreadPriority;

TransTable transTable;

bool bTimeOut;

bool bIterativeDepthFirst;

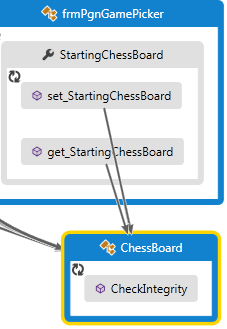
**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 5c:** Pass associated FindBestMoveUsingAlphaBetaAtDepth parameters to CheckIntegrity

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Integration Test:** frmpgnGamPicker ChessBoard Integration Test:

Visual Representation:

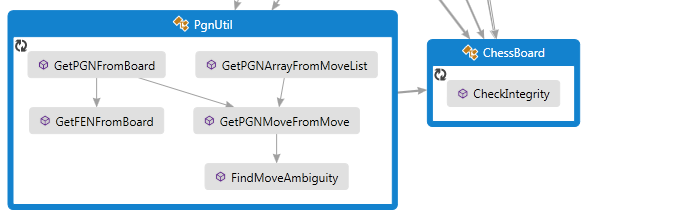


**Test 6a:** Pass associated get\_StartingChessBoard and set\_StartingChessBoard parameters to CheckIntegrity

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Integration Test:** PgnUtil ChessBoard Integration Test:

Visual Representation:



**Test 7a:** Pass associated GetPGNFromBoard parameters to CheckIntegrity

ChessBoard chessBoard,

bool bIncludeRedoMove,

string strEvent,

string strSite,

string strDate,

string strRound,

string strWhitePlayer,

string strBlackPlayer,

PgnParser.PlayerTypeE eWhitePlayerType,

PgnParser.PlayerTypeE eBlackPlayerType,

TimeSpan spanWhitePlayer,

TimeSpan spanBlackPlayer

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 7b:** Pass associated GetFENFromBoard parameters to CheckIntegrity

ChessBoard chessBoard) {

StringBuilder strBuilder;

int iEmptyCount;

ChessBoard.PieceE ePiece;

Char cPiece;

ChessBoard.PlayerColorE eNextMoveColor;

ChessBoard.BoardStateMaskE eBoardStateMask;

int iEnPassant;

bool bCastling;

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 7c:** Pass associated GetPGNArrayFromMoveList parameters to CheckIntegrity

ChessBoard chessBoard) {

string[] arrRetVal;

int iOriPos;

int iMoveIndex;

MovePosStack moveStack;

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 7d:** Pass associated GetPGNMoveFromList parameters to CheckIntegrity

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 7e:** Pass associated FindMoveAmbiguity parameters to CheckIntegrity

(ChessBoard chessBoard, ChessBoard.MovePosS move, ChessBoard.PlayerColorE eMovePlayer) {

PGNAmbiguity eRetVal = PGNAmbiguity.NotFound;

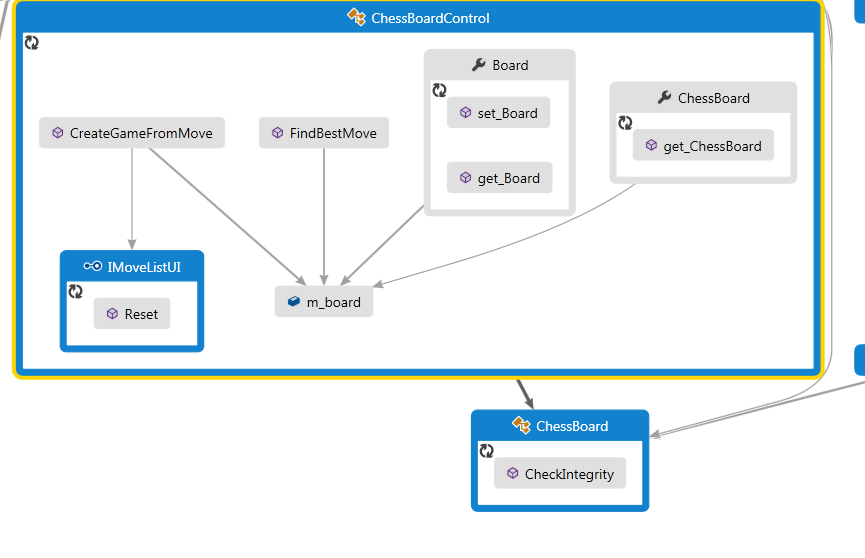
ChessBoard.PieceE ePieceMove;

List<ChessBoard.MovePosS> moveList;

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Integration Test:** ChessBoardControl ChessBoard Integration Test:

Visual Representation:



**Test 8a:** Pass associated Reset parameters to CheckIntegrity

Reset(ChessBoard chessBoard)

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 8b:** Pass associated CreateGameFromMove parameters to CheckIntegrity

frmPicker.StartingChessBoard,

frmPicker.MoveList,

frmPicker.StartingColor,

frmPicker.WhitePlayerName,

frmPicker.BlackPlayerName,

frmPicker.WhitePlayerType,

frmPicker.BlackPlayerType,

frmPicker.WhiteTimer,

frmPicker.BlackTimer

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 8c:** Pass associated FindBestMove parameters to CheckIntegrity

SearchEngine.SearchMode searchMode,

ChessBoard chessBoard,

out ChessBoard.MovePosS moveBest, out int iPermCount, out int iCacheHit, out int iMaxDepth)

{

bool bRetVal;

bool bUseBook;

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 8d:** Pass associated set\_board and get\_board parameters to CheckIntegrity

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 8e:** Pass associated get\_Chessboard parameters to CheckIntegrity

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 8f:** Pass associated m\_board parameters to CheckIntegrity

**Used In:**

public ChessBoardControl() {

InitializeComponent();

m\_board = new ChessBoard(this);

if (!m\_board.ReadBook("book.bin")) {

m\_board.ReadBookFromResource("SrcChess2.Book.bin");

}

m\_ptSelectedCell = new IntPoint(-1, -1);

m\_bAutoSelection = true;

m\_gameTimer = new GameTimer();

m\_gameTimer.Enabled = false;

m\_gameTimer.Reset(m\_board.NextMoveColor);

m\_strWhitePlayerName = "Player 1";

m\_strBlackPlayerName = "Player 2";

m\_eWhitePlayerType = PgnParser.PlayerTypeE.Human;

m\_eBlackPlayerType = PgnParser.PlayerTypeE.Human;

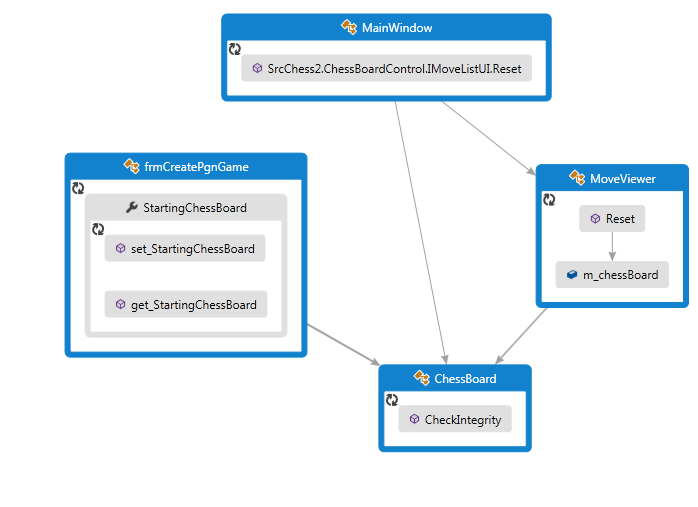
InitCell();

}

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Integration Test:** MainWindow, MoveViewer, frmCreatePgnGame ChessBoard Integration Test:

Visual Representation:



**Test 9A:** Pass associated set\_StartingChessboard and get\_StartingChessBoard parameters to CheckIntegrity

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 9B:** Pass associated JMoveListUS.Reset parameters to CheckIntegrity

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**Test 9C:** Pass associated Reset parameters to CheckIntegrity

(ChessBoard chessBoard) {

int iCurPos;

int iCount;

MoveList.Clear();

**Result:** Pass: Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception)

**m\_chessBoard tested in previous test**

### Regression Tests

1. Objective. To verify that a new defect has not been introduced into the unchanged portion of the Chess application due to changes made to the application.
2. Procedure. As a change is made to the Chess application software, subsequent to successful initial execution of all tests listed in the Regression Test, execute the Regression Test to ensure that new defects have not been introduced. Successful execution of the Regression Test is indicated if a Test Status of “pass” is obtained for each Test Case.
3. Regression Test.

| **Test Suite 4. Test Id: 4.1. Regression Test** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Suite Id** | **Test Case Id** | **Test Case Title** | **Test Objective** | **Expected Result(s)** | **Actual Result(s)** | **Test Status** |
| 1 | 1.1 | Moving Pawn Forward | Verify pawn chess piece moves forward | Pawn moves forward |  |  |
| 1 | 1.2 | Moving white left rook forward | Verify white left rook moves forward | White left rook moves forward |  |  |
| 1 | 1.3 | Moving white right rook forward | Verify white right rook moves forward | White right rook moves forward |  |  |
| 1 | 1.4 | Moving black left rook forward | Verify black left rook moves forward | Black left rook moves forward |  |  |
| 1 | 1.5 | Moving black right rook forward | Verify black right rook moves forward | Black right rook moves forward |  |  |
| 1 | 1.6 | Moving white king forward | Verify white king moves forward | White king moves forward |  |  |
| 1 | 1.7 | Moving black king forward | Verify black king moves forward | Black king moves forward |  |  |
| 1 | 1.8 | Pawn promotion to queen | Verify that a pawn right opposing sides last row allows for trading the pawn for a queen | Pawn promotion to a queen is successful |  |  |
| 1 | 1.9 | Pawn promotion to bishop | Verify that a pawn right opposing sides last row allows for trading the pawn for a bishop | Pawn promotion to a queen is successful |  |  |
| 1 | 1.10 | Pawn promotion to knight | Verify that a pawn right opposing sides last row allows for trading the pawn for a queen | Pawn promotion to a bishop is successful |  |  |
| 1 | 1.11 | Pawn promotion to rook | Verify that a pawn right opposing sides last row allows for trading the pawn for a rook | Pawn promotion to a rook is successful |  |  |
| 1 | 1.12 | Moving En Passant | Opposing pawn captures the moving pawn if it moves *en passant* | A pawn this is moved two spaces on initial move and can be in an ‘en passant’ circumstance is captured |  |  |
| 1 | 1.13.1 | Move Castling: white king – king side | Verify that castling moves the king and rook pieces correctly (two pieces at once and the king to the player’s right), and only when castling conditions exist | The white king and rook are castled correctly for a king side movement |  |  |
| 1 | 1.13.2 | Move Castling: white king – queen side | Verify that castling moves the king and rook pieces correctly (two pieces at once and the king to the player’s left, and only when castling conditions exist | The white king and rook are castled correctly for a queen side movement |  |  |
| 1 | 1.13.3 | Move Castling: black king – king side | Verify that castling moves the king and rook pieces correctly (two pieces at once and the king to the player’s right), and only when castling conditions exist | The black king and rook are castled correctly for a king side movement |  |  |
| 1 | 1.13.4 | Move Castling: black king – queen side | Verify that castling moves the king and rook pieces correctly (two pieces at once and the king to the player’s left), and only when castling conditions exist | The black king and rook are castled correctly for a queen side movement |  |  |
| 2 | 2.1 | Verification of Opposing Player Capture by Pieces that Can Capture in a Straight Line | Verify capture of opposing chess piece by chess pieces that capture in a straight line movement | Sixteen test cases (TC1 – TC16), each executed successfully |  |  |
| 2 | 2.2 | Stalemate Detection | Uses game’s stalemate detection algorithm to determine that game should stop in stalemate situation | Ten unit tests depict possible stalemate condition, if so, game stops execution |  |  |
| 2 | 2.4 | Verification of Correct Chess Piece Movement (non-capture) | Verify that each of the chess piece classes moves correctly | Sixteen test cases representing the chess piece class each move correctly on the board |  |  |
| 3 | 3.1 | Chessboard and LocalChessBoardControl Integration Test | Successfully pass CreateGameFromMove parameters through CheckIntegrity | Pass list of moves parameters to CheckIntegrity |  |  |
| 3 | 3.2.1 | Chessboard ChessException Integration Test | Successfully pass CheckIntegrity parameters through ChessException | ‘Piece count mismatch’ passed to ChessException |  |  |
| 3 | 3.2.2 | Chessboard ChessException Integration Test | Successfully pass CheckIntegrity parameters through ChessException | ‘King position mismatch’ passed to ChessException |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  | Regression Test Status: (all tests must have a status of ‘pass’ for RT status = ‘pass’) =====🡺 | Final Result: |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. Conclusion. If the Regression Test ‘passes’ (Final Result = ‘pass’), then there is assurance that the software change did not introduce a new defect into previously tested code. If the Regression Test ‘does not pass’, then the development team must identify and correct the defect before continuing further development work. Development shall only continue if the Regression Test status is ‘pass’.

### Migration Tests

The Migration testing we preformed was running our automated usability tests and automated functional tests on different computers with operating systems.  We tested on an XP machine with 4 gigs of RAM with a duel core processor, a Windows 7 Pro with 8 gigs of RAM and a Quad-core duel processor, a Windows 7 Ultimate with 4 gigs of Ram with a duel core processor, a Windows 7 Home Premium with 8 gigs of RAM Quad-core duel processor, and a Macintosh running a virtual machine with Windows 8 that has 8 gigs of RAM Quad-core processor. It work fine on all machines.

### Usability Tests

See table in Appendix C.

# Appendix A

| Type | Member | Maintainability Index | Cyclomatic Complexity | Depth of Inheritance | Class Coupling | Lines of Code |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | 84 | 1113 | 10 | 156 | 2788 |
|  |  | 84 | 1107 | 10 | 151 | 2779 |
| MoveHistory | AddCurrentPackedBoard(long, bool) : ChessBoard.RepeatResultE | 52 | 6 |  | 4 | 22 |
| GeneratedInternalTypeHelper | AddEventHandler(EventInfo, object, Delegate) : void | 94 | 1 |  | 2 | 1 |
| ChessBoard | AddIfNotCheck(ChessBoard.PlayerColorE, int, int, ChessBoard.MoveTypeE, List<ChessBoard.MovePosS>) : void | 54 | 6 |  | 6 | 17 |
| MovePosStack | AddMove(ChessBoard.MovePosS) : void | 69 | 2 |  | 2 | 6 |
| ChessBoard | AddMoveIfEnemyOrEmpty(ChessBoard.PlayerColorE, int, int, List<ChessBoard.MovePosS>) : bool | 65 | 3 |  | 5 | 8 |
| ChessBoard | AddPawnPromotionIfNotCheck(ChessBoard.PlayerColorE, int, int, List<ChessBoard.MovePosS>) : void | 71 | 1 |  | 3 | 5 |
| SearchEngineAlphaBeta | AlphaBeta(ChessBoard, ChessBoard.PlayerColorE, int, int, int, int, int, SearchEngineAlphaBeta.AlphaBetaInfo) : int | 37 | 22 |  | 15 | 56 |
| SearchEngineAlphaBeta.AlphaBetaInfo | AlphaBetaInfo() | 100 | 1 |  | 0 | 1 |
| ChessBoard | BlackPieceCount.get() : int | 72 | 2 |  | 0 | 5 |
| GameTimer | BlackPlayTime.get() : TimeSpan | 85 | 1 |  | 1 | 3 |
| BoardEvaluationBasic | BoardEvaluationBasic() | 100 | 1 |  | 0 | 1 |
| BoardEvaluationBasic | BoardEvaluationBasic() | 60 | 1 |  | 0 | 13 |
| BoardEvaluationTest | BoardEvaluationTest() | 100 | 1 |  | 1 | 1 |
| BoardEvaluationUtil | BoardEvaluationUtil() | 91 | 1 |  | 0 | 2 |
| BoardEvaluationUtil | BoardEvaluators.get() : List<IBoardEvaluation> | 91 | 1 |  | 2 | 2 |
| Book | Book() | 70 | 1 |  | 1 | 6 |
| BoardEvaluationUtil | BuildBoardEvaluationList() : void | 64 | 4 |  | 5 | 8 |
| ToolBarButton | BuildInnerButton() : void | 59 | 1 |  | 12 | 12 |
| TransTable | CacheHit.get() : int | 91 | 1 |  | 0 | 2 |
| TransTable | CacheHit.set(int) : void | 95 | 1 |  | 0 | 1 |
| ChessBoard | CancelSearch() : void | 88 | 1 |  | 3 | 2 |
| SearchEngine | CancelSearch() : void | 95 | 1 |  | 0 | 1 |
| ChessBoard | CheckIntegrity() : void | 57 | 8 |  | 2 | 14 |
| ChessBoard | CheckNextMove() : ChessBoard.MoveResultE | 61 | 5 |  | 4 | 11 |
| ChessBoard | ChessBoard() | 49 | 2 |  | 3 | 26 |
| ChessBoard | ChessBoard() | 98 | 1 |  | 0 | 1 |
| ChessBoard | ChessBoard(ChessBoard) | 85 | 1 |  | 2 | 2 |
| ChessBoard | ChessBoard(SearchEngine.ITrace) | 74 | 1 |  | 4 | 4 |
| ChessBoard | ChessBoard(SearchEngineAlphaBeta, SearchEngineMinMax) | 58 | 1 |  | 10 | 14 |
| ChessException | ChessException() | 100 | 1 |  | 1 | 1 |
| ChessException | ChessException(SerializationInfo, StreamingContext) | 95 | 1 |  | 3 | 1 |
| ChessException | ChessException(string, Exception) | 95 | 1 |  | 1 | 1 |
| ChessException | ChessException(string) | 98 | 1 |  | 1 | 1 |
| MovePosStack | Clear() : void | 86 | 1 |  | 2 | 2 |
| ChessBoard | Clone() : ChessBoard | 89 | 1 |  | 0 | 2 |
| MoveHistory | Clone() : MoveHistory | 89 | 1 |  | 0 | 2 |
| MovePosStack | Clone() : MovePosStack | 89 | 1 |  | 0 | 2 |
| ChessBoard | CloseDesignMode(ChessBoard.PlayerColorE, ChessBoard.BoardStateMaskE, int) : bool | 64 | 4 |  | 2 | 10 |
| PgnParser | CnvRawMoveToPosMove(ChessBoard.PlayerColorE, List<string>, out int[], List<ChessBoard.MovePosS>, ref int, ref int) : void | 50 | 12 |  | 7 | 23 |
| PgnParser | CnvRawMoveToPosMove(ChessBoard.PlayerColorE, string, out int, ref int, ref ChessBoard.MovePosS) : void | 40 | 19 |  | 4 | 48 |
| ColorToSolidColorBrushConverter | ColorToSolidColorBrushConverter() | 100 | 1 |  | 0 | 1 |
| GameTimer | Commit() : void | 65 | 3 |  | 3 | 8 |
| Book.CompareIntArray | Compare(int[], int[]) : int | 54 | 9 |  | 0 | 19 |
| Book.CompareIntArray | CompareIntArray() | 100 | 1 |  | 0 | 1 |
| Book | CompareKey(int[], List<int>) : bool | 66 | 4 |  | 1 | 8 |
| Book | CompareList(int[], int[], int) : bool | 66 | 4 |  | 0 | 8 |
| PgnUtil.PGNGameDescItem | CompareTo(PgnUtil.PGNGameDescItem) : int | 85 | 1 |  | 0 | 2 |
| SearchEngine.IndexPoint | CompareTo(SearchEngine.IndexPoint) : int | 67 | 4 |  | 0 | 8 |
| ChessBoard | ComputeBoardExtraInfo(ChessBoard.PlayerColorE, bool) : ChessBoard.BoardStateMaskE | 55 | 9 |  | 3 | 17 |
| ZobristKey | ComputeBoardZobristKey(ChessBoard.PieceE[]) : long | 71 | 2 |  | 1 | 5 |
| MoveHistory | ComputeCurrentPackedBoard(ChessBoard.PieceE[], ChessBoard.BoardStateMaskE) : void | 92 | 1 |  | 3 | 1 |
| MoveHistory | ComputePackedBoard(ChessBoard.PieceE[], ChessBoard.BoardStateMaskE) : MoveHistory.PackedBoard | 66 | 1 |  | 3 | 7 |
| MoveHistory | ComputePackedValue(ChessBoard.PieceE[], int) : long | 70 | 2 |  | 1 | 5 |
| ChessBoard | ComputePiecesCoverage(ChessBoard.PlayerColorE, out ChessBoard.PosInfoS) : void | 92 | 1 |  | 2 | 1 |
| Book | ComputeWeight() : void | 95 | 1 |  | 0 | 1 |
| Book | ComputeWeight(int) : int | 65 | 2 |  | 1 | 8 |
| ColorToSolidColorBrushConverter | Convert(object, Type, object, CultureInfo) : object | 87 | 1 |  | 4 | 2 |
| ColorToSolidColorBrushConverter | ConvertBack(object, Type, object, CultureInfo) : object | 98 | 1 |  | 3 | 1 |
| ChessBoard | CopyFrom(ChessBoard) : void | 52 | 1 |  | 9 | 22 |
| MovePosStack | Count.get() : int | 87 | 1 |  | 2 | 2 |
| Book | CreateBookList(List<int[]>, int, int) : int | 59 | 5 |  | 5 | 12 |
| Book | CreateBookList(List<int[]>, int) : Book.BookEntry[] | 60 | 1 |  | 2 | 11 |
| GeneratedInternalTypeHelper | CreateDelegate(Type, object, string) : Delegate | 81 | 1 |  | 2 | 2 |
| Book | CreateEntries(List<int[]>, List<Book.BookEntry>, List<int>, out int, int) : int | 48 | 7 |  | 4 | 27 |
| LocalChessBoardControl | CreateGameFromMove(ChessBoard, List<ChessBoard.MovePosS>, ChessBoard.PlayerColorE, string, string, PgnParser.PlayerTypeE, PgnParser.PlayerTypeE, TimeSpan, TimeSpan) : void | 59 | 4 |  | 8 | 13 |
| ChessBoard | CreateGameFromMove(ChessBoard, List<ChessBoard.MovePosS>, ChessBoard.PlayerColorE) : void | 65 | 3 |  | 6 | 8 |
| GeneratedInternalTypeHelper | CreateInstance(Type, CultureInfo) : object | 84 | 1 |  | 3 | 2 |
| PgnUtil | CreateOutFile(string) : Stream | 70 | 2 |  | 3 | 6 |
| PgnUtil | CreatePGNSubset() : void | 42 | 13 |  | 13 | 40 |
| MovePosStack | CurrentMove.get() : ChessBoard.MovePosS | 87 | 1 |  | 1 | 2 |
| ChessBoard | CurrentMoveColor.get() : ChessBoard.PlayerColorE | 87 | 2 |  | 1 | 2 |
| ChessBoard | CurrentZobristKey.get() : long | 91 | 1 |  | 0 | 2 |
| CustomColors | CustomColors() | 65 | 2 |  | 7 | 8 |
| PgnParser | DecodeMove(string, out int, out int, out int) : void | 45 | 24 |  | 1 | 26 |
| PgnUtil.PGNGameDescItem | Description.get() : string | 91 | 1 |  | 0 | 2 |
| ChessBoard | DesignMode.get() : bool | 91 | 1 |  | 0 | 2 |
| ToolBarButton | DisabledImage.get() : ImageSource | 87 | 1 |  | 3 | 2 |
| ToolBarButton | DisabledImage.set(ImageSource) : void | 95 | 1 |  | 3 | 1 |
| ToolBarButton | DisabledImageChanged(DependencyObject, DependencyPropertyChangedEventArgs) : void | 77 | 3 |  | 2 | 3 |
| ToolBarButton | DisplayStyle.get() : ToolBarButton.DisplayStyleE | 87 | 1 |  | 3 | 2 |
| ToolBarButton | DisplayStyle.set(ToolBarButton.DisplayStyleE) : void | 95 | 1 |  | 3 | 1 |
| ToolBarButton | DisplayStyleChanged(DependencyObject, DependencyPropertyChangedEventArgs) : void | 77 | 3 |  | 2 | 3 |
| ChessBoard | DoMove(ChessBoard.MovePosS) : ChessBoard.MoveResultE | 65 | 3 |  | 4 | 10 |
| ChessBoard | DoMoveNoLog(ChessBoard.MovePosS) : ChessBoard.RepeatResultE | 30 | 29 |  | 7 | 93 |
| GameTimer | Enabled.get() : bool | 91 | 1 |  | 0 | 2 |
| GameTimer | Enabled.set(bool) : void | 72 | 3 |  | 1 | 6 |
| ChessBoard | EnumAttackPos(ChessBoard.PlayerColorE, int, List<byte>) : int | 56 | 3 |  | 3 | 14 |
| ChessBoard | EnumCastleMove(ChessBoard.PlayerColorE, List<ChessBoard.MovePosS>) : void | 49 | 28 |  | 4 | 18 |
| ChessBoard | EnumEnPassant(ChessBoard.PlayerColorE, List<ChessBoard.MovePosS>) : void | 57 | 7 |  | 4 | 13 |
| ChessBoard | EnumFromArray(ChessBoard.PlayerColorE, int, int[], List<ChessBoard.MovePosS>) : void | 78 | 2 |  | 3 | 3 |
| ChessBoard | EnumFromArray(ChessBoard.PlayerColorE, int, int[][], List<ChessBoard.MovePosS>) : void | 70 | 4 |  | 3 | 6 |
| ChessBoard | EnumMoveList(ChessBoard.PlayerColorE, bool, out ChessBoard.PosInfoS) : List<ChessBoard.MovePosS> | 50 | 11 |  | 5 | 24 |
| ChessBoard | EnumMoveList(ChessBoard.PlayerColorE) : List<ChessBoard.MovePosS> | 85 | 1 |  | 4 | 2 |
| ChessBoard | EnumPawnMove(ChessBoard.PlayerColorE, int, List<ChessBoard.MovePosS>) : void | 43 | 22 |  | 5 | 33 |
| ChessBoard | EnumTheseAttackPos(List<byte>, int[], ChessBoard.PieceE) : int | 65 | 4 |  | 2 | 9 |
| ChessBoard | EnumTheseAttackPos(List<byte>, int[][], ChessBoard.PieceE, ChessBoard.PieceE) : int | 59 | 7 |  | 2 | 14 |
| PgnUtil | ExtractValue(string) : string | 65 | 3 |  | 0 | 8 |
| LocalChessBoardControl | Father.get() : MainWindow | 98 | 1 |  | 1 | 1 |
| LocalChessBoardControl | Father.set(MainWindow) : void | 95 | 1 |  | 1 | 1 |
| PgnUtil | FillListBoxWithDesc(Stream, ItemsControl) : int | 54 | 3 |  | 9 | 17 |
| ChessBoard | FillMoves(int, List<int[]>, int[], bool) : void | 47 | 14 |  | 1 | 27 |
| PgnUtil.FilterClause | FilterClause() | 100 | 1 |  | 0 | 1 |
| PgnUtil | FilterPGN(Stream, Stream, PgnUtil.FilterClause) : int | 49 | 10 |  | 9 | 25 |
| SearchEngineAlphaBeta | FindBestMove(ChessBoard, SearchEngine.SearchMode, ChessBoard.PlayerColorE, List<ChessBoard.MovePosS>, int[], ChessBoard.PosInfoS, ref ChessBoard.MovePosS, out int, out int, out int) : bool | 33 | 18 |  | 16 | 78 |
| SearchEngineMinMax | FindBestMove(ChessBoard, SearchEngine.SearchMode, ChessBoard.PlayerColorE, List<ChessBoard.MovePosS>, int[], ChessBoard.PosInfoS, ref ChessBoard.MovePosS, out int, out int, out int) : bool | 56 | 3 |  | 8 | 16 |
| SearchEngine | FindBestMove(ChessBoard, SearchEngine.SearchMode, ChessBoard.PlayerColorE, List<ChessBoard.MovePosS>, int[], ChessBoard.PosInfoS, ref ChessBoard.MovePosS, out int, out int, out int) : bool | 100 | 1 |  | 6 | 0 |
| SearchEngine | FindBestMove(ChessBoard, SearchEngine.SearchMode, ChessBoard.PlayerColorE, out ChessBoard.MovePosS, out int, out int, out int) : bool | 51 | 5 |  | 10 | 21 |
| ChessBoard | FindBestMove(SearchEngine.SearchMode, ChessBoard.PlayerColorE, out ChessBoard.MovePosS, out int, out int, out int) : bool | 70 | 2 |  | 7 | 5 |
| SearchEngineAlphaBeta | FindBestMoveUsingAlphaBetaAsync(ChessBoard, SearchEngine.SearchMode, ChessBoard.PlayerColorE, int, List<ChessBoard.MovePosS>, ChessBoard.PosInfoS, ChessBoard.PosInfoS, int, int, int) : SearchEngineAlphaBeta.AlphaBetaResult | 40 | 14 |  | 16 | 43 |
| SearchEngineAlphaBeta | FindBestMoveUsingAlphaBetaAtDepth(ChessBoard, SearchEngine.SearchMode, ChessBoard.PlayerColorE, List<ChessBoard.MovePosS>, ChessBoard.PosInfoS, ChessBoard.PosInfoS, int, int, int, int, TransTable, DateTime, out int, out int, out bool, out int[]) : int | 42 | 8 |  | 11 | 44 |
| SearchEngineMinMax | FindBestMoveUsingMinMaxAtDepth(ChessBoard, SearchEngine.SearchMode, ChessBoard.PlayerColorE, List<ChessBoard.MovePosS>, int[], int, ref ChessBoard.MovePosS, out int) : bool | 50 | 6 |  | 7 | 25 |
| BoardEvaluationUtil | FindBoardEvaluator(string) : IBoardEvaluation | 65 | 4 |  | 4 | 8 |
| ChessBoard | FindBookMove(SearchEngine.SearchMode, ChessBoard.PlayerColorE, ChessBoard.MovePosS[], out ChessBoard.MovePosS) : bool | 54 | 4 |  | 8 | 19 |
| PgnParser | FindCastling(ChessBoard.PlayerColorE, bool, ref int, string, ref ChessBoard.MovePosS) : int | 54 | 7 |  | 8 | 16 |
| ChessBoard | FindIfValid(ChessBoard.PlayerColorE, int, int) : ChessBoard.MovePosS | 60 | 4 |  | 6 | 11 |
| PgnUtil | FindMoveAmbiguity(ChessBoard, ChessBoard.MovePosS, ChessBoard.PlayerColorE) : PgnUtil.PGNAmbiguity | 55 | 7 |  | 9 | 16 |
| Book | FindMoveInBook(ChessBoard.MovePosS[], Random) : short | 44 | 13 |  | 3 | 35 |
| PgnParser | FindPieceMove(ChessBoard.PlayerColorE, ChessBoard.PieceE, int, int, int, ChessBoard.MoveTypeE, string, ref int, ref ChessBoard.MovePosS) : int | 50 | 14 |  | 9 | 20 |
| ChessBoard | FindValidPawnPromotion(ChessBoard.PlayerColorE, int, int) : ChessBoard.ValidPawnPromotionE | 54 | 10 |  | 7 | 18 |
| ToolBarButton | Flip.get() : bool | 87 | 1 |  | 2 | 2 |
| ToolBarButton | Flip.set(bool) : void | 95 | 1 |  | 2 | 1 |
| ToolBarButton | FlipChanged(DependencyObject, DependencyPropertyChangedEventArgs) : void | 77 | 3 |  | 2 | 3 |
| GameTimer | GameTimer() | 81 | 1 |  | 0 | 3 |
| GeneratedInternalTypeHelper | GeneratedInternalTypeHelper() | 100 | 1 |  | 1 | 1 |
| MoveHistory | GetBoardCount(MoveHistory.PackedBoard) : int | 68 | 3 |  | 1 | 6 |
| PieceSet | GetChessPieceFromPiece(ChessBoard.PieceE) : PieceSet.ChessPiece | 50 | 14 |  | 2 | 29 |
| PgnParser | GetChr() : char | 82 | 2 |  | 0 | 2 |
| PgnParser | GetCodeInError() : string | 83 | 1 |  | 0 | 2 |
| MoveHistory | GetCurrentBoardCount(long) : int | 68 | 2 |  | 2 | 7 |
| ToolBarButton | GetDisplayStyle(DependencyObject) : ToolBarButton.DisplayStyleE | 76 | 2 |  | 4 | 4 |
| ChessBoard | GetEatedPieceCount(ChessBoard.PieceE) : int | 60 | 5 |  | 1 | 13 |
| PgnUtil | GetELOValue(string) : int | 68 | 2 |  | 0 | 7 |
| PgnUtil | GetFENFromBoard(ChessBoard) : string | 36 | 21 |  | 5 | 62 |
| GameTimer | GetHumanElapse(TimeSpan) : string | 65 | 3 |  | 1 | 8 |
| ChessBoard | GetHumanPos(ChessBoard.MovePosS) : string | 53 | 9 |  | 2 | 19 |
| ChessBoard | GetHumanPos(int) : string | 70 | 1 |  | 0 | 5 |
| PgnUtil | GetNextNonEmptyLine(TextReader) : string | 62 | 4 |  | 1 | 11 |
| PgnUtil | GetNGame(Stream, int) : string | 50 | 8 |  | 7 | 22 |
| PgnParserException | GetObjectData(SerializationInfo, StreamingContext) : void | 79 | 1 |  | 3 | 3 |
| PgnUtil | GetPGNArrayFromMoveList(ChessBoard) : string[] | 60 | 2 |  | 6 | 11 |
| PgnUtil | GetPGNFromBoard(ChessBoard, bool, string, string, string, string, string, string, PgnParser.PlayerTypeE, PgnParser.PlayerTypeE, TimeSpan, TimeSpan) : string | 39 | 12 |  | 7 | 46 |
| PgnUtil | GetPGNGame(TextReader, out int, out int, out int) : List<string> | 48 | 12 |  | 2 | 26 |
| PgnUtil | GetPGNGame(TextReader, out int) : List<string> | 84 | 1 |  | 2 | 2 |
| PgnUtil | GetPGNGameDesc(List<string>, int) : string | 69 | 7 |  | 1 | 4 |
| PgnUtil | GetPGNGameInfo(List<string>, int, out int, out int) : void | 90 | 1 |  | 1 | 1 |
| PgnUtil | GetPGNGameInfo(List<string>, int, out string, out string, out string, out string, out int, out int) : void | 49 | 10 |  | 1 | 24 |
| PgnUtil | GetPGNMoveFromMove(ChessBoard, ChessBoard.MovePosS, bool) : string | 35 | 29 |  | 7 | 62 |
| PgnUtil | GetPGNSquareID(int) : string | 76 | 1 |  | 0 | 3 |
| GeneratedInternalTypeHelper | GetPropertyValue(PropertyInfo, object, CultureInfo) : object | 83 | 1 |  | 2 | 2 |
| PgnUtil | GetSquareIDFromPGN(string) : int | 60 | 6 |  | 0 | 11 |
| TransTable | GetTransTable(int) : TransTable | 75 | 2 |  | 0 | 4 |
| ToolBarButton | Image.get() : ImageSource | 87 | 1 |  | 3 | 2 |
| ToolBarButton | Image.set(ImageSource) : void | 95 | 1 |  | 3 | 1 |
| ToolBarButton | ImageChanged(DependencyObject, DependencyPropertyChangedEventArgs) : void | 77 | 3 |  | 2 | 3 |
| PgnUtil.PGNGameDescItem | Index.get() : int | 91 | 1 |  | 0 | 2 |
| ChessBoard | IsCheck(ChessBoard.PlayerColorE, int) : bool | 84 | 1 |  | 1 | 2 |
| ChessBoard | IsCheck(ChessBoard.PlayerColorE) : bool | 84 | 2 |  | 1 | 2 |
| ToolBarButton | IsEnabledChanged(DependencyObject, DependencyPropertyChangedEventArgs) : void | 77 | 3 |  | 2 | 3 |
| ChessBoard | IsEnoughPieceForCheckMate() : bool | 55 | 6 |  | 0 | 17 |
| PgnUtil | IsRetained(List<string>, int, int, PgnUtil.FilterClause) : bool | 49 | 14 |  | 3 | 25 |
| MoveHistory | IsTwoBoardEqual(MoveHistory.PackedBoard, MoveHistory.PackedBoard) : bool | 67 | 6 |  | 2 | 6 |
| MovePosStack | List.get() : List<ChessBoard.MovePosS> | 91 | 1 |  | 2 | 2 |
| ChessBoard | LoadBoard(BinaryReader) : bool | 51 | 5 |  | 9 | 22 |
| MovePosStack | LoadFromReader(BinaryReader) : void | 63 | 2 |  | 5 | 9 |
| MoveHistory | LoadFromStream(BinaryReader) : void | 50 | 5 |  | 6 | 24 |
| MoveHistory.PackedBoard | LoadFromStream(BinaryReader) : void | 71 | 1 |  | 2 | 5 |
| LocalChessBoardControl | LoadGame(BinaryReader) : bool | 61 | 3 |  | 4 | 12 |
| PieceSet | LoadPiece(PieceSet.ChessPiece) : UserControl | 100 | 1 |  | 2 | 0 |
| PieceSetStandard | LoadPiece(PieceSet.ChessPiece) : UserControl | 69 | 1 |  | 4 | 5 |
| PieceSetStandard | LoadPieceSetFromResource() : SortedList<string, PieceSet> | 48 | 10 |  | 14 | 25 |
| LocalChessBoardControl | LocalChessBoardControl() | 100 | 1 |  | 1 | 1 |
| SearchEngineMinMax | MinMax(ChessBoard, SearchEngine.SearchMode, ChessBoard.PlayerColorE, int, int, int, ref int) : int | 46 | 12 |  | 10 | 31 |
| MoveItem | Move.get() : string | 98 | 1 |  | 0 | 1 |
| MoveItem | Move.set(string) : void | 95 | 1 |  | 0 | 1 |
| MoveHistory | MoveHistory() | 69 | 1 |  | 1 | 7 |
| MoveHistory | MoveHistory(MoveHistory) | 65 | 1 |  | 2 | 8 |
| MoveItem | MoveItem(string, string, string) | 91 | 1 |  | 0 | 1 |
| MoveItemList | MoveItemList() | 100 | 1 |  | 2 | 1 |
| ChessBoard | MovePosStack.get() : MovePosStack | 91 | 1 |  | 1 | 2 |
| MovePosStack | MovePosStack() | 79 | 1 |  | 2 | 3 |
| MovePosStack | MovePosStack(MovePosStack) | 79 | 1 |  | 2 | 3 |
| MovePosStack | MoveToNext() : void | 78 | 2 |  | 0 | 3 |
| MovePosStack | MoveToPrevious() : void | 84 | 2 |  | 0 | 2 |
| BoardEvaluationBasic | Name.get() : string | 91 | 1 |  | 0 | 2 |
| BoardEvaluationTest | Name.get() : string | 91 | 1 |  | 0 | 2 |
| IBoardEvaluation | Name.get() : string | 100 | 1 |  | 0 | 0 |
| PieceSet | Name.get() : string | 98 | 1 |  | 0 | 1 |
| PieceSet | Name.set(string) : void | 95 | 1 |  | 0 | 1 |
| PieceSetStandard | NameFromChessPiece(PieceSet.ChessPiece) : string | 51 | 13 |  | 1 | 29 |
| MovePosStack | NextMove.get() : ChessBoard.MovePosS | 85 | 1 |  | 1 | 2 |
| ChessBoard | NextMoveColor.get() : ChessBoard.PlayerColorE | 91 | 1 |  | 1 | 2 |
| ChessBoard | OpenDesignMode() : void | 95 | 1 |  | 0 | 1 |
| PgnUtil | OpenInpFile(string) : Stream | 70 | 2 |  | 3 | 6 |
| PgnParser | Parse(string, List<int[]>, out int, out int) : void | 56 | 5 |  | 6 | 15 |
| PgnParser | ParseAttr(Dictionary<string, string>) : void | 52 | 9 |  | 3 | 20 |
| PgnParser | ParseFEN(string, out ChessBoard.PlayerColorE, out ChessBoard.BoardStateMaskE, out int) : bool | 30 | 40 |  | 5 | 87 |
| PgnParser | ParseNextMoveList(bool, out int[], out Dictionary<string, string>, List<ChessBoard.MovePosS>, ref int, ref int, out ChessBoard, out ChessBoard.PlayerColorE, out string, out string, out PgnParser.PlayerTypeE, out PgnParser.PlayerTypeE, out TimeSpan, out TimeSpan) : bool | 36 | 25 |  | 9 | 56 |
| PgnParser | ParseRawMove(int, out string) : bool | 47 | 11 |  | 1 | 29 |
| PgnParser | ParseSingle(string, bool, List<ChessBoard.MovePosS>, out int, out int, out ChessBoard, out ChessBoard.PlayerColorE, out string, out string, out PgnParser.PlayerTypeE, out PgnParser.PlayerTypeE, out TimeSpan, out TimeSpan) : bool | 53 | 2 |  | 7 | 21 |
| PgnParser | PeekChr() : char | 84 | 2 |  | 0 | 2 |
| PgnUtil.PGNGameDescItem | PGNGameDescItem(string, int) | 81 | 1 |  | 0 | 3 |
| PgnParser | PgnParser(bool) | 80 | 1 |  | 1 | 3 |
| PgnParserException | PgnParserException() | 94 | 1 |  | 0 | 1 |
| PgnParserException | PgnParserException(SerializationInfo, StreamingContext) | 77 | 1 |  | 4 | 3 |
| PgnParserException | PgnParserException(string, string, Exception) | 92 | 1 |  | 1 | 1 |
| PgnParserException | PgnParserException(string, string) | 94 | 1 |  | 0 | 1 |
| PgnParserException | PgnParserException(string) | 94 | 1 |  | 0 | 1 |
| PgnUtil | PgnUtil() | 84 | 1 |  | 0 | 2 |
| PieceSet | PieceSet(string) | 89 | 1 |  | 0 | 2 |
| PieceSetStandard | PieceSetStandard(string, string) | 86 | 1 |  | 1 | 2 |
| GameTimer | PlayerColor.get() : ChessBoard.PlayerColorE | 91 | 1 |  | 1 | 2 |
| GameTimer | PlayerColor.set(ChessBoard.PlayerColorE) : void | 95 | 1 |  | 1 | 1 |
| BoardEvaluationBasic | Points(ChessBoard.PieceE[], int[], ChessBoard.PosInfoS, int, int, bool, bool, int) : int | 56 | 6 |  | 2 | 15 |
| BoardEvaluationTest | Points(ChessBoard.PieceE[], int[], ChessBoard.PosInfoS, int, int, bool, bool, int) : int | 57 | 6 |  | 3 | 14 |
| IBoardEvaluation | Points(ChessBoard.PieceE[], int[], ChessBoard.PosInfoS, int, int, bool, bool, int) : int | 100 | 1 |  | 2 | 0 |
| ChessBoard | Points(SearchEngine.SearchMode, ChessBoard.PlayerColorE, int, int, ChessBoard.PosInfoS, ChessBoard.PosInfoS) : int | 62 | 2 |  | 5 | 10 |
| ChessBoard.PosInfoS | PosInfoS(int, int) | 92 | 1 |  | 0 | 1 |
| MovePosStack | PositionInList.get() : int | 91 | 1 |  | 0 | 2 |
| TransTable | ProbeEntry(long, ChessBoard.BoardStateMaskE, int, int, int) : int | 55 | 10 |  | 3 | 17 |
| ChessBoard | ReadBook(string) : bool | 74 | 2 |  | 1 | 5 |
| Book | ReadBookFromFile(string) : bool | 66 | 3 |  | 4 | 8 |
| Book | ReadBookFromReader(BinaryReader) : bool | 58 | 3 |  | 2 | 13 |
| Book | ReadBookFromResource(Assembly, string) : bool | 62 | 3 |  | 4 | 10 |
| ChessBoard | ReadBookFromResource(string) : bool | 74 | 2 |  | 1 | 5 |
| Book | ReadBookFromResource(string) : bool | 75 | 1 |  | 2 | 4 |
| TransTable | RecordEntry(long, ChessBoard.BoardStateMaskE, int, int, TransEntryTypeE) : void | 65 | 1 |  | 3 | 8 |
| ChessBoard | RedoMove() : ChessBoard.MoveResultE | 64 | 3 |  | 3 | 10 |
| MoveHistory | RemoveLastMove(long) : void | 67 | 2 |  | 2 | 7 |
| TransTable | Reset() : void | 84 | 1 |  | 0 | 2 |
| MoveHistory | Reset(ChessBoard.PieceE[], ChessBoard.BoardStateMaskE) : void | 73 | 1 |  | 3 | 5 |
| GameTimer | Reset(ChessBoard.PlayerColorE) : void | 94 | 1 |  | 1 | 1 |
| ChessBoard | ResetBoard() : void | 52 | 3 |  | 1 | 22 |
| ChessBoard | ResetInitialBoardInfo(ChessBoard.PlayerColorE, bool, ChessBoard.BoardStateMaskE, int) : void | 49 | 8 |  | 8 | 25 |
| GameTimer | ResetTo(ChessBoard.PlayerColorE, long, long) : void | 74 | 1 |  | 3 | 4 |
| ChessBoard | SaveBoard(BinaryWriter) : void | 58 | 2 |  | 6 | 14 |
| Book | SaveBookToFile(string) : void | 60 | 3 |  | 5 | 11 |
| LocalChessBoardControl | SaveGame(BinaryWriter) : void | 79 | 1 |  | 3 | 3 |
| MoveHistory | SaveToStream(BinaryWriter) : void | 66 | 3 |  | 2 | 7 |
| MoveHistory.PackedBoard | SaveToStream(BinaryWriter) : void | 73 | 1 |  | 2 | 5 |
| MovePosStack | SaveToWriter(BinaryWriter) : void | 66 | 2 |  | 7 | 7 |
| PgnUtil | ScanPGN(Stream, Dictionary<string, string>, ref int, ref int) : int | 48 | 12 |  | 7 | 26 |
| SearchEngine | SearchEngine(SearchEngine.ITrace, Random, Random) | 76 | 1 |  | 2 | 4 |
| SearchEngineAlphaBeta | SearchEngineAlphaBeta(SearchEngine.ITrace, Random, Random) | 94 | 1 |  | 3 | 1 |
| SearchEngineMinMax | SearchEngineMinMax(SearchEngine.ITrace, Random, Random) | 94 | 1 |  | 3 | 1 |
| SearchEngine.SearchMode | SearchMode(IBoardEvaluation, IBoardEvaluation, SearchEngine.SearchMode.OptionE, SearchEngine.SearchMode.ThreadingModeE, int, int, SearchEngine.SearchMode.RandomModeE) | 86 | 1 |  | 4 | 1 |
| CustomColors | SelectableColors.get() : List<Color> | 98 | 1 |  | 2 | 1 |
| CustomColors | SelectableColors.set(List<Color>) : void | 95 | 1 |  | 2 | 1 |
| ToolBarButton | SetDisplayStyle(DependencyObject, ToolBarButton.DisplayStyleE) : void | 79 | 2 |  | 4 | 3 |
| ToolBarButton | SetImage(bool) : void | 66 | 3 |  | 4 | 7 |
| GeneratedInternalTypeHelper | SetPropertyValue(PropertyInfo, object, object, CultureInfo) : void | 90 | 1 |  | 2 | 1 |
| ChessBoard | SetUndoRedoPosition(int) : void | 68 | 3 |  | 1 | 7 |
| PgnParser | SkipAltMoveAndRemark() : void | 63 | 6 |  | 0 | 9 |
| PgnParser | SkipSpace() : void | 77 | 3 |  | 0 | 3 |
| SearchEngine | SortMoveList(List<ChessBoard.MovePosS>, int[]) : List<ChessBoard.MovePosS> | 60 | 3 |  | 4 | 11 |
| ChessBoard | StandardInitialBoard.get() : bool | 91 | 1 |  | 0 | 2 |
| ToolBarButton | TextChanged(DependencyObject, DependencyPropertyChangedEventArgs) : void | 77 | 3 |  | 2 | 3 |
| ToolBarButton | ToolBarButton() | 78 | 1 |  | 5 | 3 |
| ToolBarButton | ToolBarButton() | 64 | 1 |  | 7 | 6 |
| PgnUtil.PGNGameDescItem | ToString() : string | 91 | 1 |  | 0 | 2 |
| SearchEngine | TraceSearch(int, ChessBoard.PlayerColorE, ChessBoard.MovePosS, int) : void | 84 | 2 |  | 3 | 2 |
| SearchEngine.ITrace | TraceSearch(int, ChessBoard.PlayerColorE, ChessBoard.MovePosS, int) : void | 100 | 1 |  | 2 | 0 |
| TransTable | TranslationTableSize.get() : int | 91 | 1 |  | 0 | 2 |
| TransTable | TranslationTableSize.set(int) : void | 80 | 2 |  | 1 | 3 |
| TransTable | TransTable() | 84 | 1 |  | 1 | 2 |
| TransTable | TransTable(int) | 77 | 1 |  | 1 | 4 |
| ChessBoard | UndoAllMoves() : void | 81 | 2 |  | 1 | 3 |
| ChessBoard | UndoMove() : void | 85 | 1 |  | 1 | 2 |
| ChessBoard | UndoMoveNoLog(ChessBoard.MovePosS) : void | 34 | 22 |  | 6 | 71 |
| MoveHistory | UnpackBoard(MoveHistory.PackedBoard, ChessBoard.PieceE[]) : void | 74 | 1 |  | 2 | 4 |
| MoveHistory | UnpackBoardValue(long, ChessBoard.PieceE[], int) : void | 79 | 2 |  | 1 | 2 |
| MoveHistory | UpdateCurrentPackedBoard(ChessBoard.BoardStateMaskE) : void | 92 | 1 |  | 2 | 1 |
| MoveHistory | UpdateCurrentPackedBoard(int, ChessBoard.PieceE) : void | 59 | 4 |  | 2 | 11 |
| ToolBarButton | UpdateInnerButton() : void | 55 | 7 |  | 8 | 15 |
| ChessBoard | UpdatePackedBoardAndZobristKey(int, ChessBoard.PieceE, int, ChessBoard.PieceE) : void | 76 | 1 |  | 3 | 3 |
| ChessBoard | UpdatePackedBoardAndZobristKey(int, ChessBoard.PieceE) : void | 82 | 1 |  | 3 | 2 |
| ZobristKey | UpdateZobristKey(long, int, ChessBoard.PieceE, ChessBoard.PieceE, int, ChessBoard.PieceE, ChessBoard.PieceE) : long | 70 | 1 |  | 1 | 5 |
| ZobristKey | UpdateZobristKey(long, int, ChessBoard.PieceE, ChessBoard.PieceE) : long | 74 | 1 |  | 1 | 4 |
| PgnUtil | WritePGN(TextWriter, List<string>, int) : void | 67 | 3 |  | 2 | 6 |
| ZobristKey | ZobristKey() | 63 | 2 |  | 1 | 8 |

# Appendix B

The following table describes all the rules in the Microsoft Basic Correctness Rules rule set.

| **Rule** | **Description** |
| --- | --- |
| CA1001 | Types that own disposable fields should be disposable |
| CA1009 | Declare event handlers correctly |
| CA1016 | Mark assemblies with AssemblyVersionAttribute |
| CA1033 | Interface methods should be callable by child types |
| CA1049 | Types that own native resources should be disposable |
| CA1060 | Move P/Invokes to NativeMethods class |
| CA1061 | Do not hide base class methods |
| CA1063 | Implement IDisposable correctly |
| CA1065 | Do not raise exceptions in unexpected locations |
| CA1301 | Avoid duplicate accelerators |
| CA1400 | P/Invoke entry points should exist |
| CA1401 | P/Invokes should not be visible |
| CA1403 | Auto layout types should not be COM visible |
| CA1404 | Call GetLastError immediately after P/Invoke |
| CA1405 | COM visible type base types should be COM vi |
| CA1410 | COM registration methods should be matched |
| CA1415 | Declare P/Invokes correctly |
| CA1821 | Remove empty finalizers |
| CA1900 | Value type fields should be portable |
| CA1901 | P/Invoke declarations should be portable |
| CA2002 | Do not lock on objects with weak identity |
| CA2100 | Review SQL queries for security vulnerabilities |
| CA2101 | Specify marshaling for P/Invoke string arguments |
| CA2108 | Review declarative security on value types |
| CA2111 | Pointers should not be visible |
| CA2112 | Secured types should not expose fields |
| CA2114 | Method security should be a superset of type |
| CA2116 | APTCA methods should only call APTCA methods |
| CA2117 | APTCA types should only extend APTCA base types |
| CA2122 | Do not indirectly expose methods with link demands |
| CA2123 | Override link demands should be identical to base |
| CA2124 | Wrap vulnerable finally clauses in outer try |
| CA2126 | Type link demands require inheritance demands |
| CA2131 | Security critical types may not participate in type equivalence |
| CA2132 | Default constructors must be at least as critical as base type default constructors |
| CA2133 | Delegates must bind to methods with consistent transparency |
| CA2134 | Methods must keep consistent transparency when overriding base methods |
| CA2137 | Transparent methods must contain only verifiable |
| CA2138 | Transparent methods must not call methods with the SuppressUnmanagedCodeSecurity attribute |
| CA2140 | Transparent code must not reference security critical items |
| CA2141 | Transparent methods must not satisfy LinkDemands |
| CA2146 | Types must be at least as critical as their base types and interfaces |
| CA2147 | Transparent methods may not use security asserts |
| CA2149 | Transparent methods must not call into native code |
| CA2200 | Rethrow to preserve stack details |
| CA2202 | Do not dispose objects multiple times |
| CA2207 | Initialize value type static fields inline |
| CA2212 | Do not mark serviced components with WebMethod |
| CA2213 | Disposable fields should be disposed |
| CA2214 | Do not call overridable methods in constructors |
| CA2216 | Disposable types should declare finalizer |
| CA2220 | Finalizers should call base class finalizer |
| CA2229 | Implement serialization constructors |
| CA2231 | Overload operator equals on overriding ValueType.Equals |
| CA2232 | Mark Windows Forms entry points with STAThread |
| CA2235 | Mark all non-serializable fields |
| CA2236 | Call base class methods on ISerializable types |
| CA2237 | Mark ISerializable types with SerializableAttribute |
| CA2238 | Implement serialization methods correctly |
| CA2240 | Implement ISerializable correctly |
| CA2241 | Provide correct arguments to formatting methods |
| CA2242 | Test for NaN correctly |
| CA1008 | Enums should have zero value |
| CA1013 | Overload operator equals on overloading add and subtract |
| CA1303 | Do not pass literals as localized parameters |
| CA1308 | Normalize strings to uppercase |
| CA1806 | Do not ignore method results |
| CA1816 | Call GC.SuppressFinalize correctly |
| CA1819 | Properties should not return arrays |
| CA1820 | Test for empty strings using string length |
| CA1903 | Use only API from targeted framework |
| CA2004 | Remove calls to GC.KeepAlive |
| CA2006 | Use SafeHandle to encapsulate native resources |
| CA2102 | Catch non-CLSCompliant exceptions in general handlers |
| CA2104 | Do not declare read only mutable reference types |
| CA2105 | Array fields should not be read only |
| CA2106 | Secure asserts |
| CA2115 | Call GC.KeepAlive when using native resources |
| CA2119 | Seal methods that satisfy private interfaces |
| CA2120 | Secure serialization constructors |
| CA2121 | Static constructors should be private |
| CA2130 | Security critical constants should be transparent |
| CA2205 | Use managed equivalents of Win32 API |
| CA2215 | Dispose methods should call base class dispose |
| CA2221 | Finalizers should be protected |
| CA2222 | Do not decrease inherited member visibility |
| CA2223 | Members should differ by more than return type |
| CA2224 | Override equals on overloading operator equals |
| CA2226 | Operators should have symmetrical overloads |
| CA2227 | Collection properties should be read only |
| CA2239 | Provide deserialization methods for optional fields |

# Appendix C

| **Test Suite Lists** | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Suite** | **Test Case Id** | **Test Case Title** | **Test Category** | **Test Objective** | **Expected Result(s)** | **Actual Result(s)** | **Test Status** |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.1 | Moving Pawn Forward | Structural – unit test – automated | verify pawn chess piece moves forward | pawn moves forward | pawn moves forward | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.2 | Moving white left rook forward | Structural – unit test – automated | verify white left rook moves forward | white left rook moves forward | white left rook moves forward | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.3 | Moving white right rook forward | Structural – unit test – automated | verify white right rook moves forward | White rook moves forward | White rook moves forward | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.4 | Moving black left rook forward | Structural – unit test – automated | Verify black left rook moves forward | Black left rook moves forward | Black left rook moves forward | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.5 | Moving black right rook forward | Structural – unit test – automated | Verify black right rook moves forward | Black right rook moves forward | Black left rook moves forward | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.6 | Moving white king forward | Structural – unit test – automated | Verify white king moves forward | White king moves forward | White king moves forward | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.7 | Moving black king forward | Structural – unit test – automated | Verify black king moves forward | Black king moves forward | Black king moves forward | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.8 | Pawn promotion to queen | Structural – unit test – automated | Verify pawn is promoted to a queen when moved to the last row. | Pawn becomes a queen. | Pawn becomes a queen. | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.9 | Pawn promotion to bishop | Structural – unit test – automated | Verify pawn is promoted to a bishop when moved to the last row. | Pawn becomes a bishop. | Pawn becomes a bishop. | pass |
| 1.Control Flow 1. Control Flow & Data Flow Test: DoMoveNoLog | 1.10 | Pawn promotion to knight | Structural – unit test – automated | Verify pawn is promoted to a bishop when moved to the last row. | Pawn becomes a knight. | Pawn becomes a knight. | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.11 | Pawn promotion to rook | Structural – unit test – automated | Verify pawn is promoted to a rook when moved to the last row. | Pawn becomes a rook. | Pawn becomes a rook. | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.12 | Moving EnPassant | Structural – unit test – automated | Opposing pawn captures the moving pawn if it moves *en passant* | A pawn this is moved two spaces on initial move and can be in an ‘en passant’ circumstance is captured | A pawn this is moved two spaces on initial move and can be in an ‘en passant’ circumstance is captured | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.13.1 | Move Castling: white king – king side | Structural – unit test – automated | Verify that castling moves the king and rook pieces correctly (two pieces at once and the king to the player’s right), and only when castling conditions exist | The white king and rook are castled correctly for a king side movement | The white king and rook are castled correctly for a king side movement | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.13.2 | Move Castling: white king – queen side | Structural – unit test – automated | Verify that castling moves the king and rook pieces correctly (two pieces at once and the king to the player’s left, and only when castling conditions exist | The white king and rook are castled correctly for a queen side movement | The white king and rook are castled correctly for a queen side movement | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.13.3 | Move Castling: black king – king side | Structural – unit test – automated | Verify that castling moves the king and rook pieces correctly (two pieces at once and the king to the player’s right), and only when castling conditions exist | The black king and rook are castled correctly for a king side movement | The black king and rook are castled correctly for a king side movement | pass |
| 1.Control Flow & Data Flow Test: DoMoveNoLog | 1.13.4 | Move Castling: black king – queen side | Structural – unit test – automated | Verify that castling moves the king and rook pieces correctly (two pieces at once and the king to the player’s left), and only when castling conditions exist | The black king and rook are castled correctly for a queen side movement | The black king and rook are castled correctly for a queen side movement | pass |
| 2.Functional Tests - | 2.1 | Verification of Opposing Player Capture by Pieces that Can Capture in a Straight Line | Functional – Pairwise Testing – manual test | verify capture of opposing chess piece by chess pieces that capture in a straight line movement | Sixteen test cases (TC1 – TC16), each successfully are executed | Sixteen test cases (TC1 – TC16), were each successfully executed | pass |
| 2.Functional Tests | 2.2 | Stalemate Detection | Functional – Decision Table Based Testing – unit tests | uses game’s stalemate detection algorithm to determine that game should stop in stalemate situation | 10 unit tests depict possible stalemate condition, game stops execution | Game stopped for all 10 unit tests | pass |
| 2.Functional Tests | 2.3 | Verification that configuration of the chess piece font set changes correctly | Functional – Random Testing – manual test | verify that selection of a specific chess piece font correctly sets the chess piece font | 10 randomly selected fonts result in the correct chess piece font being displayed | The correct chess piece font was displayed for each of the 10 font selections | pass |
| 2.Functional Tests | 2.4 | Verification of Correct Chess Piece Movement (non-capture) | Functional – Equivalence Class Partitioning Testing – manual test | verify that each of the chess piece classes moves correctly | Sixteen test cases representing the chess piece class each move correctly on the board | Each of the equivalence chess piece classes moved as expected | pass |
| 3.Integration Tests | 3.1 | Chessboard and LocalChessBoardControl Integration Test | Integration test - | Successfully pass CreateGameFromMove parameters through CheckIntegrity | pass list of moves parameters to CheckIntegrity | interaction confirmed, and no piece count mismatch or king position mismatch message displayed | pass |
| 3.Integration Tests | 3.2.1 | Chessboard ChessException Integration Test | Integration test | Successfully pass CheckIntegrity parameters through ChessException | ‘piece count mismatch’ passed to ChessException | parameters successfully passed to Chess Exception | pass |
| 3.Integration Tests | 3.2.2 | Chessboard ChessException Integration Test | Integration test | Successfully pass CheckIntegrity parameters through ChessException | ‘king position mismatch’ passed to ChessException | parameters mismatch messages displayed | pass |
| 3.Integration Tests | 3.3 | SearchEngine ChessBoard Integration Test | Integration test | Successfully pass SearchEngine parameters through CheckIntegrity | pass list of moves to CheckIntegrity | interaction confirmed and no piece count mismatch or king position mismatch message | pass |
| 3.Integration Tests | 3.4.1 | PgnParser ChessBoard Integration Test | Integration test | Successfully pass ParseSingle, ParseNextMoveList, and m\_chessBoard parameters through CheckIntegrity | pass associated ParseSingle PGN parameters to CheckIntegrity | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.4.2 | PgnParser ChessBoard Integration Test | Integration test | Successfully pass ParseSingle, ParseNextMoveList, and m\_chessBoard parameters through CheckIntegrity | pass associated ParseNextMoveList parameters to CheckIntegrity | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.4.3 | PgnParser ChessBoard Integration Test | Integration test | Successfully pass ParseSingle, ParseNextMoveList, and m\_chessBoard parameters through CheckIntegrity | Pass associated m\_chessBoard parameters to CheckIntegrity and generate exception when a move cannot be resolved | Piece count mismatch identified | pass |
| 3.Integration Tests | 3.4.4 | PgnParser ChessBoard Integration Test | Integration test | Successfully pass ParseSingle, ParseNextMoveList, and m\_chessBoard parameters through CheckIntegrity | Pass associated m\_chessBoard parameters to CheckIntegrity and generate exception when a move cannot be resolved | Kind position mismatch identified | pass |
| 3.Integration Tests | 3.5.1 | SearchEngineAlphaBeta ChessBoard Integration Test | Integration test | Pass associated FindBestMove parameters to CheckIntegrity | * successful interaction between components is confirmed * no piece count mismatch * no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.5.2 | SearchEngineAlphaBeta ChessBoard Integration Test | Integration test | Pass associated FindBestMoveUsingAlphaBetaAsync parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.5.3 | SearchEngineAlphaBeta ChessBoard Integration Test | Integration test | Pass associated FindBestMoveUsingAlphaBetaAtDepth parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.6 | frmpgnGamPicker ChessBoard Integration Test | Integration test | Pass associated get\_StartingChessBoard and set\_StartingChessBoard parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.7.1 | PgnUtil ChessBoard Integration Test | Integration test | Pass associated GetPGNFromBoard parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.7.2 | PgnUtil ChessBoard Integration Test | Integration test | Pass associated GetFENFromBoard parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.7.3 | PgnUtil ChessBoard Integration Test | Integration test | Pass associated GetPGNArrayFromMoveList parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.7.4 | PgnUtil ChessBoard Integration Test | Integration test | Pass associated GetPGNMoveFromList parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.7.5 | PgnUtil ChessBoard Integration Test | Integration test | Pass associated FindMoveAmbiguity parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.8.1 | ChessBoardControl ChessBoard Integration Test | Integration tests | Pass associated Reset parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.8.2 | ChessBoardControl ChessBoard Integration Test | Integration tests | Pass associated CreateGameFromMove parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.8.3 | ChessBoardControl ChessBoard Integration Test | Integration tests | Pass associated FindBestMove parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.8.4 | ChessBoardControl ChessBoard Integration Test | Integration tests | Pass associated set\_board and get\_board parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.8.5 | ChessBoardControl ChessBoard Integration Test | Integration tests | Pass associated get\_Chessboard parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.8.6 | ChessBoardControl ChessBoard Integration Test | Integration tests | Pass associated m\_board parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.9.1 | MainWindow, MoveViewer, frmCreatePgnGame ChessBoard Integration Test | Integration tests | Pass associated set\_StartingChessboard and get\_StartingChessBoard parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.9.2 | MainWindow, MoveViewer, frmCreatePgnGame ChessBoard Integration Test | Integration tests | Pass associated JMoveListUS.Reset parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |
| 3.Integration Tests | 3.9.3 | MainWindow, MoveViewer, frmCreatePgnGame ChessBoard Integration Test | Integration tests | Pass associated Reset parameters to CheckIntegrity | •successful interaction between components is confirmed  •no piece count mismatch  •no king position mismatch message displayed | Interaction confirmed, and no piece count mismatch or king position mismatch message displayed (through chess exception) | pass |  |  |  |  |
| 4. Unit Tests | 4.1.1 | Move pawn at start | Structural – unit test – manual | Verify pawn can move one or two spaces at the start. | Pawn moves the correct number of spaces. | Piece flashes  Piece moves to selected square  Piece flashes | pass |
| 4. Unit Tests | 4.1.2 | Attack diagonal with Pawn | Structural – unit test – manual | Verify attacking diagonal with a pawn captures the enemy piece. | Pawn moves, enemy piece captured. | Piece flashes  Piece moves to selected square  Piece flashes  Opposing piece moved to defeated area | pass |
| 4. Unit Tests | 4.1.3 | Attack forward with pawn | Structural – unit test – manual | Verify pawns can’t attack forward. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.1.4 | Move pawn backwards. | Structural – unit test – manual | Verify pawns can’t move backwards. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.1.5 | Move pawn side to side | Structural – unit test – manual | Verify pawns can’t move sideways. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.2.1 | Move bishop sideways | Structural – unit test – manual | Verify bishop can’t move sideways. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.2.2 | Move bishop up and down | Structural – unit test – manual | Verify bishop can’t move in a straight line. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.2.3 | Attack forward bishop bishop L pattern | Structural – unit test – manual | Verify bishop can’t attack forward. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.2.4 | Attack diagonal bishop | Structural – unit test – manual | Verify bishops can attack diagonally. | System allows move, bishop advances. Piece captured. | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.2.5 | Attack sideways | Structural – unit test – manual | Verify bishop can’t attack sideways. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.2.6 | Attack backwards | Structural – unit test – manual | Verify bishop can’t attack backwards in a straight backwards in a straight line. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.2.7 | Move bishop diagonal | Structural – unit test – manual | Verify bishops can move diagonally. | Bishop advances to square. | Piece flashes  Piece moves to selected square  Piece flashes | pass |
| 4. Unit Tests | 4.3.1 | King move it at start of game | Structural – unit test – manual | Verify king can’t move at the start | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.3.2 | King attack straight on | Structural – unit test – manual | Verify king can attack forward. | Move allowed, king advances piece captured. | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.3.3 | King attack diagonal | Structural – unit test – manual | Verify king can attack diagonally. | Move allowed, king advances piece captured. | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.3.4 | King attack side to side | Structural – unit test – manual | Verify king can attack sideways. | Move allowed, king advances piece captured. | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.4.4 | King move backwards | Structural – unit test – manual | Verify king can move backwards. | King advances | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.4.5 | King move side to side | Structural – unit test – manual | Verify king can attack sideways. | King advances | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.4.6 | King move diagonal | Structural – unit test – manual | Verify king can move diagonal. | King advances | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.5.1 | Queen move it at start of game | Structural – unit test – manual | Verify Queen can’t move at the start | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.5.2 | Queen attack straight on | Structural – unit test – manual | Verify Queen can attack forward. | Queen advances. | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.5.3 | Queen attack diagonal | Structural – unit test – manual | Verify Queen can attack diagonally. | Queen advances, piece captured. | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.5.4 | Queen attack side to side | Structural – unit test – manual | Verify Queen can attack sideways. | attack side to side | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.5.5 | Queen move backwards | Structural – unit test – manual | Verify Queen can move backwards. | Queen moves | Piece flashes  Piece moves to selected square  Piece flashes | pass |
| 4. Unit Tests | 4.5.6 | Queen move side to side | Structural – unit test – manual | Verify Queen can attack sideways. | Queen moves | Piece flashes  Piece moves to selected square  Piece flashes | pass |
| 4. Unit Tests | 4.5.7 | Queen move diagonal | Structural – unit test – manual | Verify Queen can move diagonal. | Queen moves | Piece flashes  Piece moves to selected square  Piece flashes | pass |
| 4. Unit Tests | 4.6.1 | Move knight at start of game. | Structural – unit test – manual | Verify Knight can’t move at the start | System doesn’t allow move. | Piece flashes  Piece moves to selected square  Piece flashes | pass |
| 4. Unit Tests | 4.6.2 | Knight attack straight on | Structural – unit test – manual | Verify Knight can attack forward. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.6.3 | Knight attack diagonal | Structural – unit test – manual | Verify Knight can attack diagonally. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.6.4 | Knight attack side to side | Structural – unit test – manual | Verify Knight can attack sideways. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.6.5 | Knight move backwards | Structural – unit test – manual | Verify Knight can move backwards. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.6.6 | Knight move side to side | Structural – unit test – manual | Verify Knight can attack sideways. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.6.7 | Knight move diagonal | Structural – unit test – manual | Verify Knight can move diagonal. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.7.1 | Rook move it at start of game | Structural – unit test – manual | Verify Knight can’t move at the start | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.7.2 | Rook attack straight on | Structural – unit test – manual | Verify Rook can attack forward. | Attack succeeds, piece captured, rook advances. | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.7.3 | Rook attack diagonal | Structural – unit test – manual | Verify Rook can attack diagonally. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.7.4 | Rook attack side to side | Structural – unit test – manual | Verify Rook can attack sideways. | Attack succeeds, piece captured, rook advances. | Piece flashes  Piece moves to selected targets square  Piece flashes | pass |
| 4. Unit Tests | 4.7.5 | Rook move backwards | Structural – unit test – manual | Verify Rook can move backwards. | Rook advances. | Piece flashes  Piece moves to selected square  Piece flashes | pass |
| 4. Unit Tests | 4.7.6 | Rook move side to side | Structural – unit test – manual | Verify Rook can attack sideways. | Rook advances. | Piece flashes  Piece moves to selected square  Piece flashes | pass |
| 4. Unit Tests | 4.7.7 | Rook move diagonal | Structural – unit test – manual | Verify Rook can move diagonal. | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.8.1 | Undo and then redo | Structural – unit test – manual | Undo an action and then redo it. | Move a pawn forward 1 space  Press undo  Press Redo | Piece flashes  Piece moves to selected square  Piece flashes  Undo pressed, pieces flashes and moves back to original square  Redo pressed, piece flashes moves to indicated square | pass |
| 4. Unit Tests | 4.9.1 | Test checkmate | Structural – unit test – manual | Verify that checkmate works. | A King in a state of check cannot move or block capture. Checkmate triggered | Game ends with a checkmate. | pass |
| 4. Unit Tests | 4.9.2 | Test check | Structural – unit test – manual | Test valid check state. Put a piece in a position to attack the king. Check is triggered. | Check is triggered | Check is triggered | pass |
| 4. Unit Tests | 4.9.3 | Test move out of check. | Structural – unit test – manual | Verify check is lifted when the king is no longer under attacker. | Player moves out of check state, check is lifted. | Check is no longer triggered. | pass |
| 4. Unit Tests | 4.10.1 | Test white moving on black’s turn. | Structural – unit test – manual | Try to move a white piece on black's turn | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.10.2 | Test black moving on white’s turn. | Structural – unit test – manual | Try to move a black piece on white's turn | System doesn’t allow move. | System beeps  Not moved | pass |
| 4. Unit Tests | 4.10.3 | Move pawn forward two spaced at start. | Structural – unit test – manual | Verify a pawn can move two spaces at the start of the game. | Pawn advances. | Piece flashes  Piece moves to selected square  Piece flashes | pass |

# References

"Chess Program in C#” *Code Project* Web. 19 Apr. 2013.

<<http://www.codeproject.com/Articles/36112/Chess-Program-in-C>>

“Laws of Chess” World Chess Federation 19 Apr. 2013

<<http://www.fide.com/component/handbook/?id=124&view=article>>