55 Stonehaven Crescent Deakin

2014

SSD-Stage IV

Chess Game

26510562

Contents

[Evaluation 3](#_Toc391318686)

[Screen design 4](#_Toc391318687)

[Original design – game screen 4](#_Toc391318688)

[Final design – game screen 5](#_Toc391318689)

[Original design – main menu 6](#_Toc391318690)

[Final design – main menu 7](#_Toc391318691)

[Original features implementation 8](#_Toc391318692)

[Movement 8](#_Toc391318693)

[Scoring 8](#_Toc391318694)

[Game Initialisation 9](#_Toc391318695)

[Overall game 9](#_Toc391318696)

[Added features 10](#_Toc391318697)

[Promotion 10](#_Toc391318698)

[Castling 10](#_Toc391318699)

[Computer assisted movement 10](#_Toc391318700)

[Dynamically creating the game 10](#_Toc391318701)

[Saving and loading games 10](#_Toc391318702)

[Testing 11](#_Toc391318703)

[Testing During Development 11](#_Toc391318704)

[Changes made because of testing 11](#_Toc391318705)

[Thorough testing of three functions 12](#_Toc391318706)

[Validate Integer 12](#_Toc391318707)

[Function code 12](#_Toc391318708)

[Test data 12](#_Toc391318709)

[Driver code 13](#_Toc391318710)

[Driver output 13](#_Toc391318711)

[get\_increment 14](#_Toc391318712)

[Function code 14](#_Toc391318713)

[Test data 14](#_Toc391318714)

[Driver code 15](#_Toc391318715)

[Driver output 15](#_Toc391318716)

[Get\_background 16](#_Toc391318717)

[Function code 16](#_Toc391318718)

[Test data 16](#_Toc391318719)

[Driver code 17](#_Toc391318720)

[Driver output 17](#_Toc391318721)

[Testing user input 18](#_Toc391318722)

[New game screen 18](#_Toc391318723)

[Test data 18](#_Toc391318724)

[Testing In different environments 19](#_Toc391318725)

[My laptop 19](#_Toc391318726)

[My mother’s laptop 20](#_Toc391318727)

[Survey results 21](#_Toc391318728)

[Help material 21](#_Toc391318729)

[Question 1 21](#_Toc391318730)

[Question 2 21](#_Toc391318731)

[Question 3 22](#_Toc391318732)

[Question 4 22](#_Toc391318733)

[Saving games 23](#_Toc391318734)

[Question 5 23](#_Toc391318735)

[Question 6 23](#_Toc391318736)

[Navigation 24](#_Toc391318737)

[Question 7 24](#_Toc391318738)

[Question 8 24](#_Toc391318739)

[Playing the game 25](#_Toc391318740)

[Question 9 25](#_Toc391318741)

[Question 10 25](#_Toc391318742)

[Question 11 26](#_Toc391318743)

[Survey summary 26](#_Toc391318744)

# Evaluation

The final product contained a number of new features designed during development. These were not originally planned but add to the final products value. These additions include:

* Saving and loading saved games
* Pawn Promotion
* Castling
* Computer assisted moves
* Dynamically creating the whole game

These changes are reflected in the final screen design and the final code, however, the features outlined in the original design specifications are all present in the final product and the final screen design is fairly similar using the same basic layout of the original screen design. The data structures used in my final product are very similar to those used in the original design specifications.

The data gained from my survey reflected the fact that the game was fairly well polished but that chess as a game is not particularly fun to play on a computer without an Artificial Intelligence.

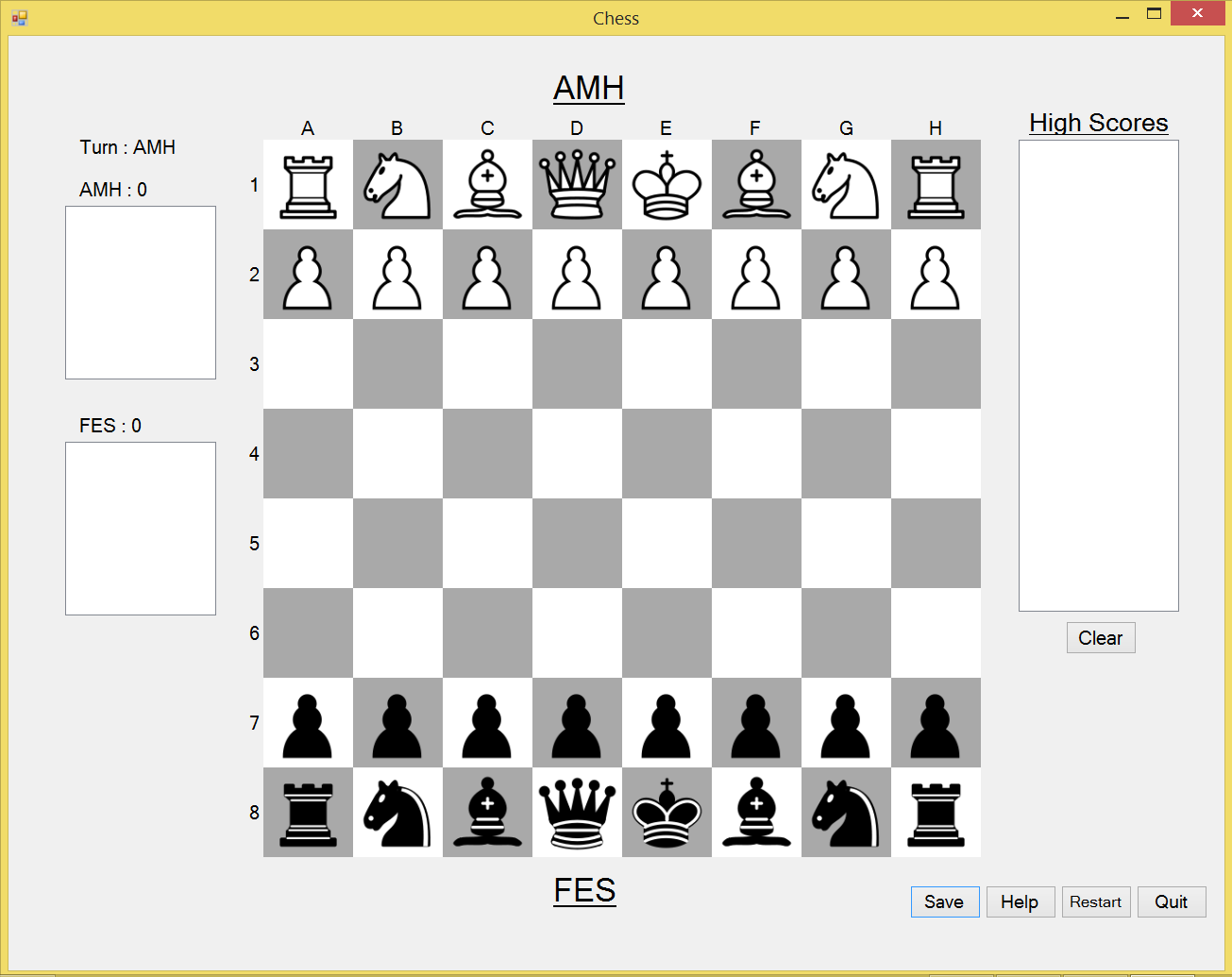
Several problems were outlined but overall the feedback was positive.

# Screen design

## Screen Design gamescreen.pngOriginal design – game screen

This screen design contains all the basic elements contained in the final screen designs, is does however use slightly different graphics to represent the various elements. Some noticeable differences are the colour of the “black” tiles which were changed to grey so as to be able to see the black pieces; the final design also has a panel of buttons in the bottom right hand corner to facilitate navigation, restarting and help. Another change is the lack of the “Submit Move” buttons on the final screen, this is because I found out that picture boxes have a clicked event handler in Visual Studio 2010.

## Final design – game screen

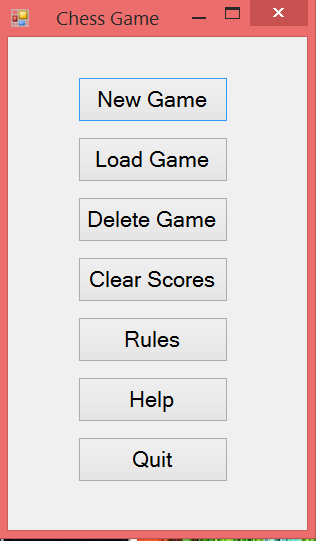
This final screen design clearly draws a lot from the original design but it does also represent a full utilisation of the tools that Visual Studio 2010 has to offer. The graphics are very smooth and nice, the colour scheme is softer and the screen elements are placed more ergonomically. The final game did no use full names for security and file handling reasons and this is represented in this screen with the two sets of initials.

## Original design – main menu



The original design of the main menu was that it would accommodate navigation to and from the game, access to the rules and help and the ability to exit the game. In these respects, the final design is very similar to the original, however, the final design does contain several more buttons to allow for the extra functionality that the game includes.

## Final design – main menu

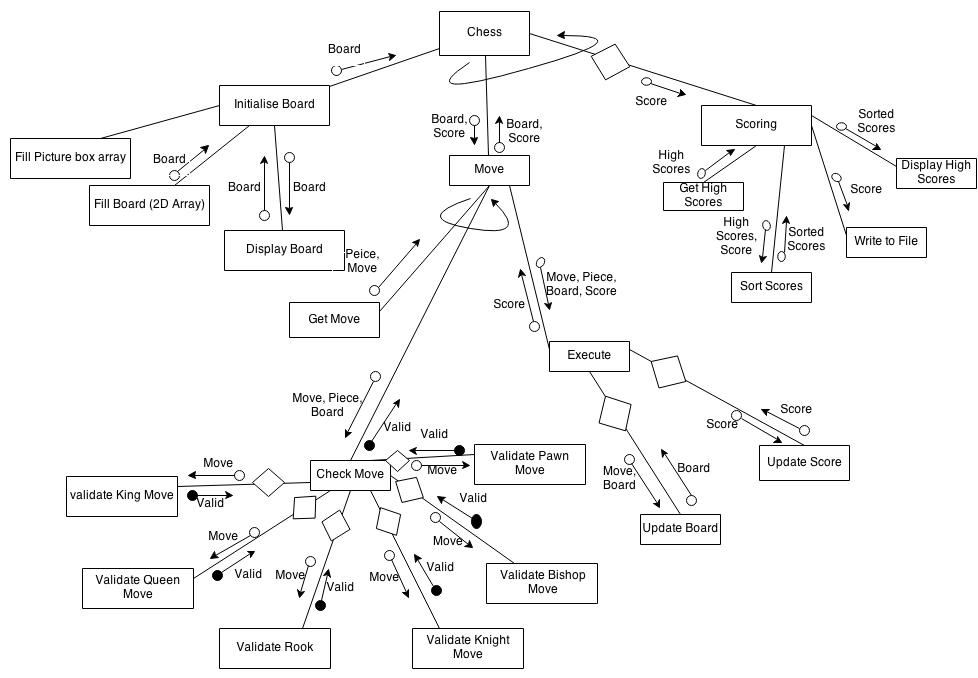


The final design of the main menu is very reminiscent of the original design in its simplistic style although it does feature the added buttons and does not have a large heading as seen in the original design.

# Original features implementation

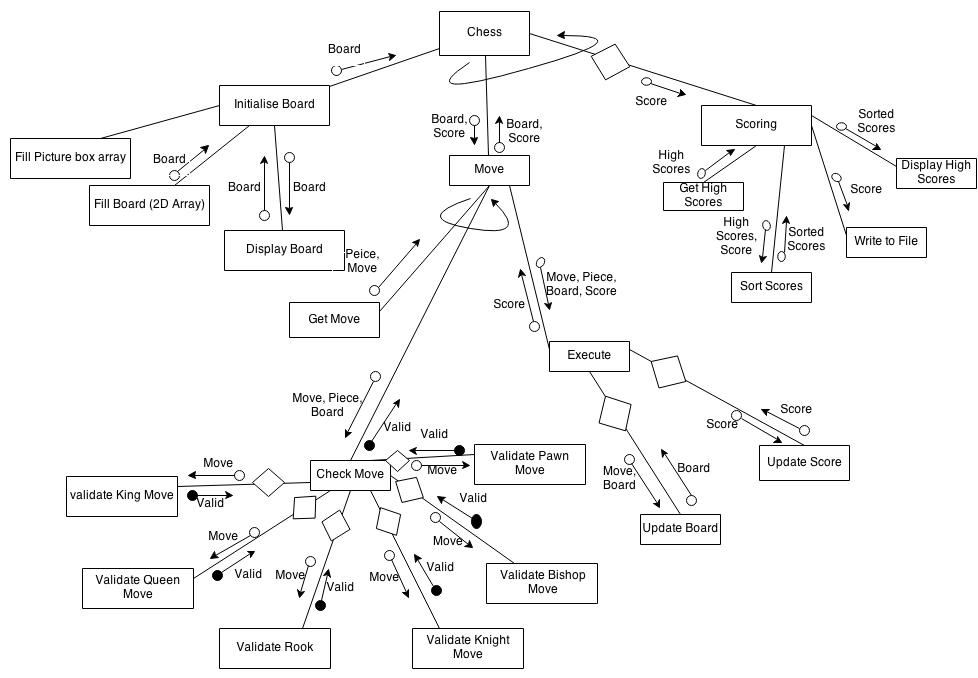
## Movement

This is an essential part of a chess game and was designed to be implemented using a set of functions to check a move, these functions would all be called from a module that would select which one to use. (Fig 1) This routine would be called from a click event that handled every tile on the board. This design was carried over through to my final program and has barely changed. A few additions have been made to accommodate extra functionality but the structure is almost identical to the original designs.

Fig 1

## Scoring

This feature was a requirement of the game and was designed to be implemented using one main subroutine that would call subs to get the high scores, sort the scores, write them to a file and finally display them. (Fig 2) This structure was carried over into my final game which used a selection sort to sort the scores and displayed them in a list box.

Fig 2

## Game Initialisation

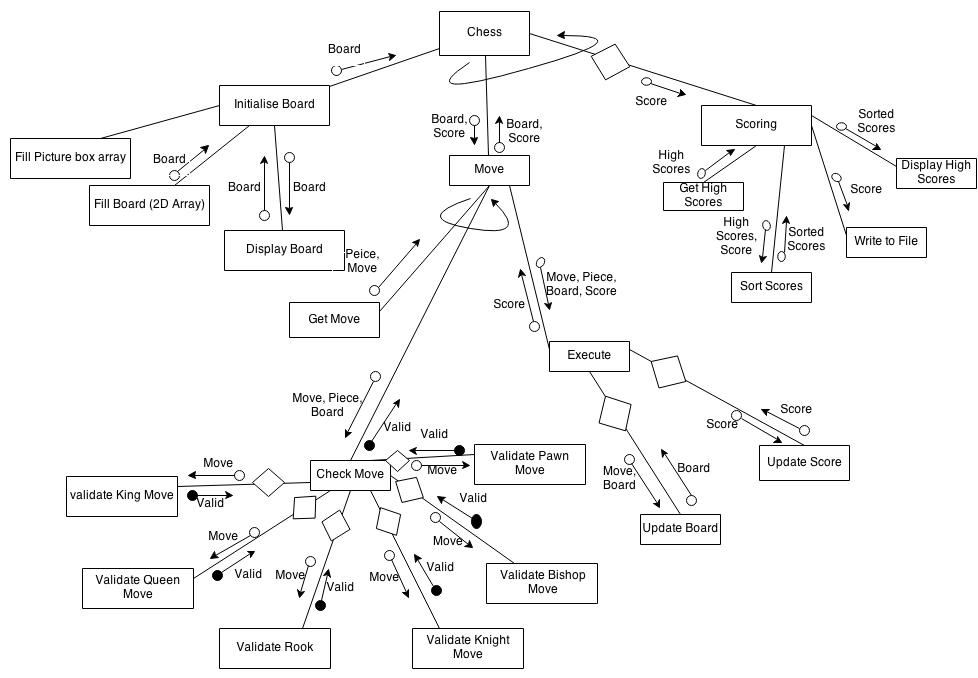
The initialisation of a new game of chess was implemented in a way that was fairly similar to the way outlined in the design specifications (Fig 3), however, there were several additions to allow for the dynamic nature of the board. The final code combined the original “Fill picturebox array” and “Fill Board” subroutines into one but split up the display board into several routines that created the board and then filled it.

Fig 3

## Overall game

Overall, I managed to achieve the game that I set out to create. This means that the final product was a working two player chess game that used the rules of chess to effect game play. The final product also utilised the designed scoring mechanism. There have been many additions to my game since it’s design and this has of course effected the code and screen designs but the basic structure of the game is the same.

# Added features

## Promotion

The ability to promote pawns was not originally in the design specifications but was added due to advice from early bug testers. This allows the game to flow more freely because pawns can now be used once they have reached the end of the board. This feature was very necessary and adds a layer of complexity to the strategy behind the chess game.

## Castling

The ability to castle was added due to advice from early alpha testers who were complaining that the game was not realistic and representative of a real game of chess. It took several weeks to work out how to easily add this ability but it finally worked and beta testers agreed that it was an improvement.

## Computer assisted movement

This feature involves the computer highlighting all the possible moves that a piece can make. It was added due to advice from AMG as well as a suggestion from an alpha tester. Due to the modular structure of my code and the use of functions to check a moves validity, this feature only took an hour to code and was very successful. The code had to be later adjusted to highlight castle moves red as well. This feature is a standout and adds a layer of sophistication to the game that may otherwise be lacking.

## Dynamically creating the game

This feature was added not so much for the users as for source code portability reasons. The fact that the code can run in debug mode without having to manually create the required board objects is a huge bonus for moving the code between computers and systems. This feature came with an unforseen problem that it would originally only work on 1920x1080 screens. This problem was luckily detected and resolved after a bit of thinking.

## Saving and loading games

This feature was not originally in the specifications due to its perceived complexity and my initial inability to design a working system of saving and loading games. However, soon into development, I realised that this ability would be invaluable not only to the user, but for bug testing by saving a game at a certain point to re-create errors in the game. This feature was added in one bulk session of coding and worked seamlessly thereafter. It gives users to start from where they left off and has been praised by beta testers.

# Testing

## Testing During Development

During development, the pseudocode written in stage II was used to generate equivalent Visual Basic code, this process was usually fairly painless however, in almost every sub, there was an unforseen error. These errors were usually easily fixed but this did mean that a lot of white box testing was needed to ensure that each subroutine and function performed as expected on all code paths. For this, I used the test data from stage II as well as a simple driver that would output the results into a temporary list box.

While I was developing my approach to dynamic object creation, I used a stub in which a message box would display the name of the object that had been clicked. This ensured that the objects were being displayed in the intended order and that the objects were running code when they were clicked.

For added features where I had not created any pseudocode, I generally, started by writing the headers for each subroutine that I thought would be required. I then wrote a comment or two about how I wanted each sub to work and exactly what the input would be. This allowed me to develop subs individually and then connect them all to a main subroutine. This allowed for easy isolation of bugs and was conforming to the modular approach outlined in the task specifications.

Another form of testing was a constant stream of bug testers. Once the basic game play was working, I asked lots of people to play my game and give me suggestions about its improvement and to notify me of any bugs found. This was extremely useful because they revealed several bugs and generally gave a lot of useful advice that I tried to incorporate into the final game. Ideas provided by the bug testers included Pawn Promotion, Castling and Move Highlighting.

## Changes made because of testing

As mentioned above, there were hundreds of small changes made to the original pseudo code because of bugs, however, there were also larger changes made because of bugs found during testing. When I tested my game saving feature, I found that although new games could be loaded and saved, the grave was not being saved and loaded as well. This problem was fixed with the addition of a subroutine that counted the pieces not on the board and put them in the grave.

When I tested my game on a computer with a lower resolution, I found that the game screen was taking up too much space on the screen, I realised that the dynamic nature of my code meant that I would have to shrink the size of my game manually in the code so that it could run on different screens. This problem was fixed by multiplying all sizes in the code by a common ratio.

When I tested my first implementation of the castle move, I found that although it worked for the white pieces, it did not work for the black side because the pieces were reversed. This problem was easily rectified but may not have been found without thorough testing.

# Thorough testing of three functions

As per the project specifications, I have completed “Thorough Testing” of three nominated functions. This was done using Visual Basic to write simple drivers that would test all code paths and several other key values. The output was a list box with the input, result and expected result.

## Validate Integer

### Function code

Private Function validate\_integer(number As String) As Boolean

'This Takes a candidate and returns a boolean indicating if each and every character in the candidate is an integer

If number = "" Or IsNothing(number) Then 'this checks for null string values

Return False

End If

For Each i In number 'This increments through each character in the string

If Asc(i) < 48 Or Asc(i) > 57 Then 'These are the ascii characters that represent the digits

Return False

End If

Next

Return True 'return true in all other cases

End Function

### Test data

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Test Data | Expected Output | Reason for inclusion |
| number | “-1” | False | Test negative input |
| number | “0” | True | Test boundary number |
| number | “1” | True | Test small number |
| number | “9” | True | Test boundary number |
| number | “10” | True | Test two digit number |
| number | “5.9” | False | Test decimal number |
| number | “a” | False | Test character |
| number | “/” | False | Test ascii lower boundary case |
| number | “:” | False | Test ascii upper boundary case |
| number | “” | False | Test empty string boundary case |
| number | “Nothing” | False | Test nothing boundary case |

### Driver code

Sub test\_validate\_integer()

'a dictionary of input data : expected output

Dim test\_data As New Dictionary(Of String, Boolean) From {

{"-1", False},

{"0", True},

{"1", True},

{"9", True},

{"10", True},

{"5.9", False},

{"a", False},

{"/", False},

{":", False},

{"", False},

{"Nothing", False}}

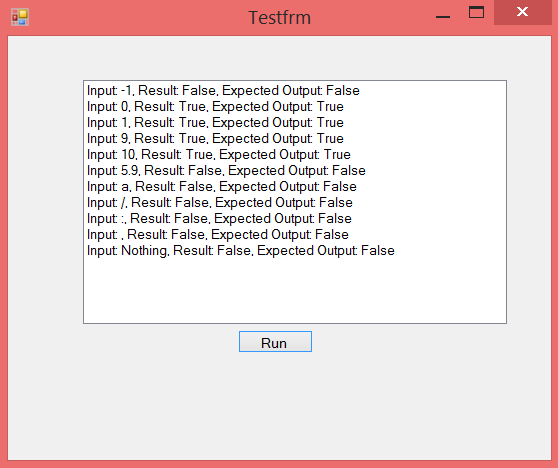
For Each i In test\_data.Keys

lstTest.Items.Add("Input: " & i & ", Result: " & validate\_integer(i) & ", Expected Output: " & test\_data(i))

Next i

End Sub

### Driver output



## get\_increment

### Function code

Private Function get\_increment(change\_x As Integer, change\_y As Integer) As String

'used with the check\_bishop function

'this checks wether to increment or decrement x and y and Returns a string

If change\_x > 0 Then 'if change\_x > 0, then the piece is moving diagonally to the Right

If change\_y > 0 Then 'if change\_y > 0 then the piece is moving diagonally forward

Return "++"

Else

Return "+-"

End If

Else

If change\_y > 0 Then

Return "-+"

Else

Return "--"

End If

End If

End Function

### Test data

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Test Data | Expected Output | Reason for Inclusion |
| change\_x | -1 | -- | test two negative values |
| change\_y | -1 |
| change\_x | -1 | -- | test negative value and boundary case 0 |
| change\_y | 0 |
| change\_x | -1 | -+ | test negative value and positive boundary case |
| change\_y | 1 |
| change\_x | 0 | -- | test boundary case 0 and negative value |
| change\_y | -1 |
| change\_x | 0 | -- | test boundary case 0 and boundary case 0 |
| change\_y | 0 |
| change\_x | 0 | -+ | test boundary case 0 and boundary case 1 |
| change\_y | 1 |
| change\_x | 1 | +- | test boundary case 1 and negative value |
| change\_y | -1 |
| change\_x | 1 | +- | test boundary case 1 and boundary case 0 |
| change\_y | 0 |
| change\_x | 1 | ++ | test boundary case 1 and boundary case 1 |
| change\_y | 1 |

### Driver code

Sub test\_get\_increment()

Dim x, y As Integer

Dim test\_data As New Dictionary(Of Integer(), String) From

{{{-1, -1}, "--"}, {{-1, 0}, "--"}, {{-1, 1}, "-+"}, {{0, -1}, "--"}, {{0, 0}, "--"}, {{0, 1}, "-+"}, {{1, -1}, "+-"}, {{1, 0}, "+-"}, {{1, 1}, "++"}}

For Each i In test\_data.Keys

x = i(0)

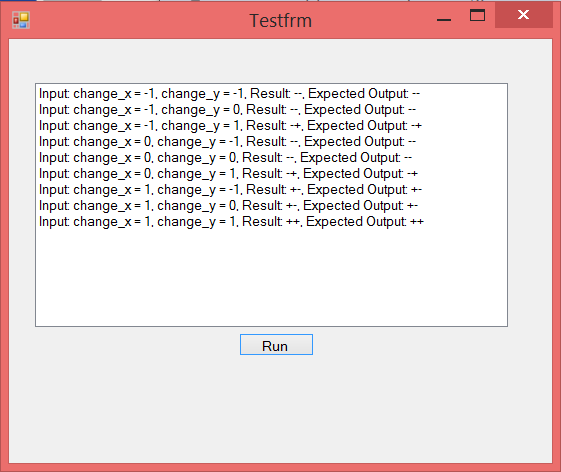
y = i(1)

lstTest.Items.Add("Input: " & "x = " & x & ", y = " & y & ", " & "Result: " & get\_increment(x, y) & ", Expected Output: " & test\_data(i))

Next i

End Sub

### Driver output



## Get\_background

### Function code

Private Function get\_background(x As Integer, y As Integer) As Color

'These If statements display the checked background that chess boards have

If x Mod 2 = 0 And y Mod 2 = 0 Then

Return Color.White

End If

If x Mod 2 <> 0 And y Mod 2 = 0 Then

Return Color.DarkGray

End If

If x Mod 2 = 0 And y Mod 2 <> 0 Then

Return Color.DarkGray

End If

If x Mod 2 <> 0 And y Mod 2 <> 0 Then

Return Color.White

End If

End Function

### Test data

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Test Data | Expected Output | Reason for Inclusion |
| x | 4 | Color.White | test two even numbers |
| y | 4 |
| x | 4 | Color.DarkGray | test x even, y odd |
| y | 3 |
| x | 3 | Color.DarkGray | test x odd, y even |
| y | 4 |
| x | 3 | Color.White | test two odd numbers |
| y | 3 |

### Driver code

Sub test\_get\_background()

Dim x, y As Integer

Dim result, expected As String

Dim test\_data As New Dictionary(Of Integer(), Color) From

{{{4, 4}, Color.White}, {{4, 3}, Color.DarkGray}, {{3, 4}, Color.DarkGray}, {{3, 3}, Color.White}}

For Each i In test\_data.Keys

x = i(0)

y = i(1)

result = get\_background(x, y).ToString

expected = test\_data(i).ToString

lstTest.Items.Add("Input: " & "x = " & x & ", y = " & y & ", " & "Result: " & result & ", Expected Output: " & expected)

Next i

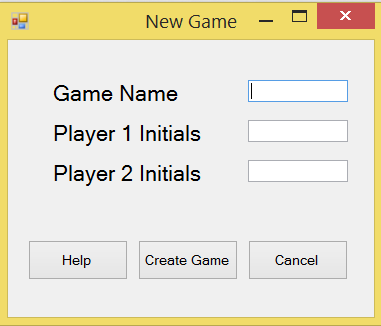
End Sub

### Driver output

# Testing user input

User input has been tested for the “New Game” screen and it was found to be working as expected. This is the only screen that takes variable user input and so it is the only one that has been tested in this way.

## New game screen



### Test data

|  |  |  |  |
| --- | --- | --- | --- |
| Game Name | Player 1 Initials | Player 2 Initials | Output |
| "Test" | "AMH" | "TIM" | game created |
| "Test" | "amh" | "tim" | error message |
| "Test" | "amh" | "TIM" | error message |
| "Test" | "AMH" | "tim" | error message |
| "Test" | "" | "TIM" | error message |
| "Test" | "AMH" | "" | error message |
| "Test" | "ABCDE" | "TIM" | game created |
| "Test" | "AMH" | "ABCDE" | game created |
| "Test" | "ABCDEF" | "TIM" | error message |
| "Test" | "AMH" | "ABCDEF" | error message |
| "Test#" | "AMH" | "TIM" | error message |
| "Test" | "AMH#" | "TIM" | error message |
| "Test" | "AMH" | "TIM#" | error message |
| "Test&#" | "42" | "&$" | error message |

# Testing In different environments

Several tests of my project have been done in different environments. I have run my game from two locations “C:\Hall Andrew (11) Stage 3\Chess Game\bin\Debug\”, “C:\Users\Andrew\Desktop\ Hall Andrew (11) Stage 3\Chess Game\bin\Debug\” and from an external USB “ANDY\ Hall Andrew (11) Stage 3\Chess Game\bin\Debug\”. The game performed as expected in all three cases although it was observed to be slightly slower to load and run from the USB. I have also performed tests of my game on two different computers described below.

## My laptop

Operating System: Windows 8.1

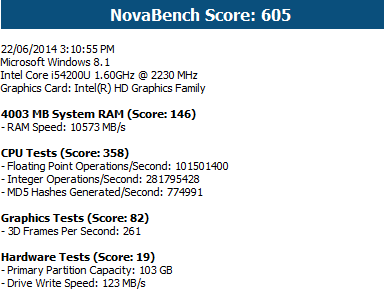
Processor: Intel® Core™ i5-4200U CPU @ 1.60GHz

RAM: 4.00 GB

Memory: 100GB SSD

Screen Resolution: 1920x1080

Nova Bench Results:



The game performed perfectly on this computer, no lag effects were seen and the screen resolution was perfect for the game.

## My mother’s laptop

Operating System: Windows Vista

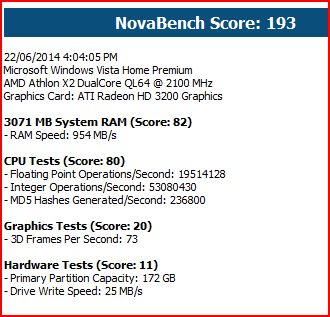
Processor: AMD Athlon X2 DualCore QL64 @ 2100 MHz

RAM: 4GB

Memory: 170 GB HDD

Resolution: 1280 × 720

Nova Bench Results:



The game was slightly slower to load but this was expected. There were no portability issues and the different screen resolution was accommodated beautifully.

# Survey results

## Help material

### Question 1

If you read the rules PDF, on a scale of 1 to 5, how useful did you find it?

The rules PDF seems to have been well received with no people scoring it less than 3. It was designed to give the player the rules that the game would implement and to that end is was a success.

### Question 2

If you used the Tutorial (main menu > Help), on a scale of 1 to 5, how informative was it?

The tutorial was reviewed very well with the same scores as the Rules PDF, this again indicates that it achieved its purpose by providing the user with a good introduction into the game.

### Question 3

If you used the Help File (press F1), on a scale of 1 to 5, how helpful was it to your experience?

The help file seems to have been received very poorly, this is due to a number of factors.

1. It did not work on the school computers where almost half of the people surveyed played the game
2. It was poorly formatted at best and was created on a very out dated help file creator
3. It restated the user manual which was provided for testers to read before starting the survey.

## Question 4

Overall, how would you describe the usefulness of the help material that the game provided?

The responses to this question were generally positive with an average response of 3.8 for all questions. The help file was reviewed fairly poorly and this is probably due to its poor formatting and the fact that it does not work on the school computers where it was reviewed by several people.

The tutorial was generally met with appreciation but many comments noted that the help file did not work.

Responses included:

* “Clear concise, easy to follow”
* “The help material was good but the Help file doesn’t work”
* “Useful, though slightly tedious in length.”
* “Help file did not work thought other forms of help were useful”

## Saving games

### Question 5

On a scale of 1 to 5, how useful were the save and load features?

This feature was reviewed well for its implementation but testers were sceptical as to how often it would be used given that chess games are rarely saved in real life.

### Question 6

If you have any other comments about the saving feature, please write them below.

This section did provide some useful feedback. People generally noted that although a nice feature, it was not generally likely to be used because a game is rarely saved anyway.

It was generally thought to be well implemented and the average response for this section was 4.1 which is a remarkable result.

Responses included:

* “Well implemented but unlikely to be used often.”
* “No confirmation on “Delete” of saved game before game disappears”
* “Buttons not greyed out when not available”

## Navigation

### Question 7

On a scale of 1 to 5, how easy was the game to navigate?

Exactly half of the testers gave this element a 5/5 rating but there were several lower marks. This is an interesting result which seems to indicate that it was well implemented but not artistically brilliant.

### Question 8

If you have any other comments about the navigation, please write them below.

This section was generally very well reviewed. People praised the clear screen design and the use of a main menu to easily navigate from however, several people noted that some buttons could have been named more appropriately and that the load and delete screens could have been the same screen.

Responses included:

* “Buttons need to be more descriptive”
* “Wasn’t obvious I could click a button, need hand cursor.”
* “Straight forward”

## Playing the game

### Question 9

On a scale of 1 to 5, how fun would you say your overall experience was?

The results to this question show that although the game was not terrible and boring, it was not very fun nor was it amazingly engaging. This probably stems from the fact that it is a two player game and that chess is not a particularly fun game at the best of times.

### Question 10

On a scale of 1 to 5, how would you rate the game interface?

This rating is a very positive review of the game interface with over half of the testers giving it a 5/5 review. The interface was clean and neat but not particularly stimulating and I think that this accounts for the lower reviews.

### Question 11

If you have any other comments about the game, please write them below.

This section allowed people to express a whole list of grievances that they had found with the game. This included interface issues, game related issues, and many others. Among the problems highlighted were:

* Only two player chess allowed
* Quite boring to play against one’s self
* Some technical features of chess were not present. E.g. Check
* Bland game screen design

The game play was otherwise reviewed quite well with an average response for this section of 4. People praised the computer assisted movement as well as the ability to castle and to promote pawns.

Responses Included:

* “A well implemented game”
* “Playing against oneself limits enjoyment. Also disorientating always viewing pieces from black side”
* “Easy and enjoyable”

## Survey summary

Those who filled out the survey had generally good things to say but also highlighted many issues that could be fixed in a version 2 of the game. Most of the respondents came from my Software Design class and generally these respondents were the most interesting with their critique. However, as a game for the public, it should be tested with the public and of those that tested my game, the response was generally positive and indicated that this was a successful implementation of Chess.