Tuesday, March 4th 2014

ASSIGNMENT I

Software Engineering

Software Design I 2AA4

Jadeesuan Anton	*1213386*
Connor Hallett	*1158083*
Spencer Lee	*1224941*
Nicolas Lelievre	*1203446*
Zhiting Qian	*1217485*

TABLE OF CONTENTS

PREFACE

The programing language we have chosen to use is Visual Basics (available within Visual Studios 2013 VB.net) since it allows an extensive customization of user interfaces. However, it does not follow the same implementation as other programming languages previously seen. Hence, Modules, classes and methods are virtually defined and are composed of functions and subs. The deliverables in this part are mostly leaves in the hierarchy tree due to the high graphical nature of this assignment. We will use bottom-up logic in constructing and designing future code.

CLASS AND MODULE DESCRIPTION, INTERFACE AND IMPLEMENTATION

Sub: Form1_Load

Description (4.1)

The Form1_Load sub is done as soon as the program is loaded as a result, this sub becomes necessary for declaration of initial variables and drawing graphic pieces. As a result, the group will create the checker board array using this sub. In addition to drawing graphics, this sub also linked an array to the picture boxes in order to handle future actions.

Interface Specifications (4.2)

- New PictureBox()
  - Draw the Checkers board
  - References: None

Variables (4.5)

Variable Use	Variables	Variable Type
Output	C_track	Integer
Output	InitialX	Integer
Output	C_trackarray	PictureBox Array(31)
Output	M_custom	Boolean
Output	M_standard	Boolean
Input	ErrorClick	PictureBox

Internal Implementation (4.5)

This code is meant to create picture boxes at given locations in a square matrix in order to represent the black checker tiles. The logic is to have a variable (i) increase by 50 each time and draw a new picture box there and every 4 i variables a j variable will be present to move to the next time. The implementation of the Logic can be found below

```
For I in range 0 to 7
     For j in range 0 to 3
          C_trackarray(c_track) = new PictureBox
          C_trackarray(c_track).position = (I*50 + InitialX,j*50)
          C_track +=1
          ErrorClick.sendtoback
     Next
Next
```

Protected Sub: MakePiece

Description (4.1)

This sub will handle all drawing functionalities within the requirements. It has internal methods that can draw the normal checker board setup and will handle clicks during custom mode. This sub will later have methods to allow it to incorporate save and load functionality. This sub will not handle movements which will be implemented in a future sub as a part of assignment 2. This sub was used to unify the custom and standard setups, both of which are triggered by clicks on valid checker board locations. The MakePiece sub is a leaf in the hierarchy and thus is defined as a user interface. In keeping with modularity and logic, this sub was made instead of having each other module draw its own pieces.

Interface Specifications (4.2)

- Trackarray.image()
  - Draws checker pieces upon clicks from the user
  - Deletes Checker Pieces if custom mode is not selected or if piece limit is reached (Very Quick Action)
- Custom Mode()
  - Lets the user setup custom pieces with mouse clicks and specified pieces
- Standard Mode()
  - Calls the standard setup for a game of checkers, displayed fully on the board

Variables (4.5)

Variable Use	Variable Name	Variable Type	Declaration Location
Input	M_custom	Boolean	Global
Input	M_standard	Boolean	Global
Internal	ThisPB	PictureBox	Local
Internal	Gamesetup	Integer Array(31)	Local
Output	Notice	String	Local
Output	Future	String	Local
Output	Trackarray	Picture box array (31)	Global

Internal Implementation (4.5)

Output is a set of actions and messages. Messages represent logic for states that appear in later assignments. Following the mode, the code matches up the trackarray to the setup mode integer array or gets the clicks and implements into an array. Since trackarray is linked to the public picture boxes, they will display the determined picture.

Logic: Mode		Output Action
M_custom	M_standard	Message Notice
Not M_custom	M_standard	ThisPB = Standard
M_custom	Not M_standard	ThisPB Is Custom Piece
Not M_custom	Not M_standard	Message Future

Sub: Quit Game Menu Strip

Description (4.1)

This is a quit option that is built into most games, it gives the user an option to end the game in progress. For future use, this will prompt a save response from the user.

This module is a module of itself since anticipations of a save function will be programed into this module.

Interface Specifications (4.2)

- QuitGameMenuStrip_Click()
    - Menu Strip that drops down and offers a “Quit” Option, it will promptly end the game after a confirmation message

Variables Used (4.5)

Variable Use	Variable Name	Variable Type	Declaration Location
Input	M_quit	Boolean	Local

Internal Implementation (4.5)

Visual Basics offers functions to end the currently running program with “End” and offers yes or no prompts with the built in message box function.

Output = {End, Nothing}

Logic: Mode	Output Action
Prompt Quit: VB Yes	End
Prompt Quit: VB No	Do Nothing

Sub: Quit Game Menu Strip

Description (4.1)

This module executes after the user has decided that it is time to close the program by clicking the "x" on the top right corner of the window. It merely prompts a Boolean response which can keep the form open if the user changes his/her mind and delivers a warning message.

Though this module does little for actual functionality and has a similar module for quitting, it was included for robustness of the program.

Interface Specifications (4.2)

- "X" option at the top right corner, can be used to close running program

Variables Used (4.5)

Variable Use	Variable Name	Variable Type	Declaration Location
Input	M_Close	Boolean	Local

Internal Implementation (4.5)

This is a very simple implementation as Visual Basics has the "End" Function and the message box function which easily implements the following. The "x" at the top of the form is automatically generated.

Output = {Form Close, Nothing}

Logic: Mode	Output Action
Prompt Quit: VB Yes	Form Close
Prompt Quit: VB No	Do Nothing

Sub: Game Timer

Description (4.1)

This function counts the time elapsed since a game has started and stores them inside a seconds variable that the user will see.

This function is merely a small part of the game experience and as it is not related with any other modules, it would be illogical to count it as such.

Interface Specification (4.2)

- TimerDisp.text
    - A Label will display a message of the time elapsed in minutes and seconds to the user, label information generated by timer.

Variables Used (4.5):

Variable Use	Variable Name	Variable Type	Declaration Location
Output	C_Minutes	Integer	Local
Output	C_Seconds	Integer	Local

Internal Implementation (4.5)

Implemented using the Tick operation provided in VB, the time runs every interval and sets the internal to 1 second. Updating the seconds local variable will provide a simple but functional timer. Updating the Minutes counter and resetting seconds is done at 60 seconds intervals.

```
Seconds +=1
If Seconds = 60
     Seconds = 0
     Minutes +=1
End If
```

Sub: Reset

Description (4.1)

This sub is meant to reset everything inside the checkers game to the state when the program was first opened. This is done with a menu strip drop down option.

This is its own module instead of using every other module since the other modules is dependent of global variable states which allow this sub to be independent of every other sub, which improves coupling.

Interface Specification (4.2)

- M_reset_Click()
    - The interface for this sub is a menu strip drop down and click. The click will promptly remove everything on the game board and return it to the state similar to when the program just started.
- makeInvis()
    - This function makes all visible elements in the game invisible

Variables (4.5)

Variable Use	Variable Name	Variable Type	Declaration Location
Input	M_custom	Boolean	Global
Input	M_standard	Boolean	Global
Output	Trackarray	Picture box array (31)	Global

Internal Implementation (4.5)

This has a Function call the functions stopTime() and makeInvis(). These combine to make everything that is unnecessary invisible and clears internal variables of their current values before returning them to when the form was loaded.

Internal Implmentation (4.5) MakeInvis Function

Makes the visible elements of the form invisible.

```
InfoLabel.visible = false
ExitMode.visible = false
CustomPiece.visible = false
```

Function: Stop Time

Description (4.1)

Internal operation that resets the time variables involved in the counting process.

Variables (4.5)

Variable Use	Variable Name	Variable Type	Declaration Location
Output	C_Minutes	Integer	Local
Output	C_Seconds	Integer	Local

Internal Implementation (4.5)

The implementation was just about setting c_minutes and c_seconds equal to 0.

```
Return C_minutes = 0
Return C_seconds = 0
```

Sub: Custom Mode

Description (4.1)

Custom Mode has the logics required to setup the game as the user specifies. Clicks are read and the required pieces are placed there as given by another logic.

This module already has a large set of related functions and is a well-defined area of code. It is therefore decomposed to itself.

Variables (4.5)

Variable Use	Variable Name	Variable Type	Declaration Location
Internal	BLKLimit	Boolean	Local
Internal	RedLimit	Boolean	Local
Internal	CustomPiece.image	Image	Form-Wide (Global)
Update	Trackarray	Picture box array (31)	Global
Update	C_BlackCount	Integer	Local
Update	C_RedCount	Integer	Local

Internal Implementation (4.5)

BLKLimit() and RedLimit()

Loops through the board, counts the pieces and imposes false when the limit is reached as well as ceases the placement of pieces in the make pieces module.

```
For I =0 to 31
If Black; Black = Black +1
If Red: Red = Red +1
End if
```

CustomPiece()

Implemented using Finite State Machine and If statements in code

Output {Red Piece, Red King, Black Piece, Black King, Nothing

Current State	Next State: Click
Red Piece	Red King
Red King	Black Piece
Black Piece	Black King
Black King	Nothing
Nothing	Red Piece

Sub: Standard Mode

Description (4.1)

Standard Mode has the information required to setup the game in the standard checkers mode. The standard game is triggered by a user click and updates the checker board with appropriate pieces.

The reasoning behind this module is to have it tie-in later with the save function. Both of these require logic from an array of integers for piece setting. Though not implemented in this specific deliverable, the additional parts will be implemented later.

Interface Specification (4.2)

None specifically, this sub is tied in with Make piece (similar to Methods)

Variables (4.5)

Variable Use	Variable Name	Variable Type	Declaration Location
Input	M_custom	Boolean	Global
Input	M_standard	Boolean	Global

Internal Implementation (4.5)

This sub can be explained using a tabular expression:

Logic: Input		Output Action
M_custom	M_standard	None
Not M_custom	M_standard	Standard
M_custom	Not M_standard	Custom
Not M_custom	Not M_standard	None

USES RELATIONSHIP

Below are the relationships we have created between methods, classes and modules in our implementation of Assignment 1.

QuitGameToolStripMenuItem_Click
Uses: None Yet

Form_FormClosing
Uses: None Yet

Form1_Load
Uses :MakePiece

MakePiece
Uses: LimitCheckRED LimitCheckBLK StandardToolStripMenuItem_Click CustomToolStripMenuItem_Click GameTimer_Tick

StandardToolStripMenuItem_Click
Uses: MakeInvis timestop

CustomToolStripMenuItem_Click
Uses: MakeInvis Timestop ExitMode_Click LimitCheckBLK LimitCheckRed

ExitMode_Click
Uses: gamebegin

ResetToolStripMenuItem_Click
Uses: MakeInvis

LimitCheckBLK
Uses: None Yet

LimitCheckRED
Uses: None Yet

CustomPiece_Click
Uses: CustomLoop

CustomLoop
Uses: None Yet

CustomInfo_Click
Uses: CustomLoop

ErrorClick_Click
Uses: None Yet

GameTimer_Tick
Uses: None Yet

MakeInvis
Uses: None Yet

timestop
Uses: None Yet

gamebegin
Uses: None Yet

REQUIREMENT TRACE BACKS

Generally speaking, “this assignment consists of generating the graphical representation of a checkers board, and being able to specify initial piece positions” (from Assignment 1 specifications). The following will describe how the implementation of our classes reflects and accomplishes the prescribed requirements.

Form1_Load

As is relatively apparent by the class’ name, Form1_Load loads Form1. This form displays the general user interface. This includes, in broad terms, the game board, a title, a score board for each player and alphanumeric place holders as well as a menu strip located at the top.

The game board consists of light and dark squares (8 on the height and 8 on the width) from which a light square may be found in the bottom right corner. The board itself (modeled to imitate a wooden surface) lies on a coloured background image. The different colored tiles on the board are easy to differentiate, making piece recognition and placing rather simple and intuitive.

The board is also traced by a sequence of letters and numbers used to identify individual piece locations. The letters (placed on the top and bottom) start with A on the left and end with H on the right-most tile. Similarly, the numbers (found on the right and left sides of the board) start with 1 at the bottom tile and incrementally increase up to 8 at the top tile.

Therefore, Form1_Load manages to accomplish the “initial set up” of “an 8-by-8 checkers board with dark and light squares” in which “a light square” is found “in the bottom right corner” of the playing surface. Place naming conventions have also been met.

StandardToolStripMenuItem

As was aforementioned, Form1_Load included a menu strip at the top of the window. This menu strip, amongst containing various functions, includes the StandardToolStripMenuItem class. This item may be accessed through the menu strip’s drop down menu under “Menu” and then “New Game”. This characterises the “standard” game setup for rapid game starts. Hence, “the user [is] able to set up an initial position of pieces on the board by specifying […] the standard opening position”.

Upon pressing “Standard”, a pop up window appears over Form1 informing the user to click any black tile in order to set the pieces and start the game timer. After agreeing to this statement and clicking on any black tile, the “standard opening position” places three rows of red pieces (white pieces on the Assignment specification diagram) on rows one to three, strictly positioned on dark tiles. Similarly, black pieces are placed on rows six to eight, once again, only on black tiles. A total of 12 pieces for each colour is placed on the board.

The initial game setup is therefore completely legal by default and obeys the initial setup requirements instilled by the assignment specifications.

CustomToolStripMenuItem

The CustomToolStripMenuItem behaves closely to that of the StandardToolStripMenuItem class. However, this class, as the name implies, allows the user to initiate a custom game in which pieces are placed at the user's will. This mode is accessed through the top menu strip under "Menu", "New Game".

After selecting this mode of gameplay, a single red game piece appears at the bottom of Form1 accompanied by a "Complete Setup" button as well as a label that reads "Click on the left picture to change pieces". At this moment, the red piece is selected and clicking on any black tile will place a red piece at that location. Clicking on the red piece outside of the board will circulate a red king piece. At this time, clicking on any black square will place a red king. A total of twelve combined king and standard red pieces may be placed. Clicking again on the selection piece will make a black piece appear followed by a black king piece, allowing the user to place black and black king pieces on the board respectively. Finally, pressing on the black king piece will show an empty space with the caption "Now removing. Click me to go to red". This allows the user to remove pieces originally placed on the board. Simply clicking on any dark tile housing any piece will remove it from game play.

Tiles may be overwritten in custom mode's set-up process. In other words, setting a piece of one color and then clicking the same space with a different piece will place the latter piece on the tile. Clicking multiple times on the same tile while in the same piece mode has no effect other than placing a piece on the first click. Clicking on any white space on the board during custom setup in any piece mode will display a pop-up window which reads "You cannot place a piece here", notifying the user that he/she has attempted an illegal procedure. Accepting this dialog box returns the player to the custom setup.

Once the player is content with the placement of all the game pieces, he/she may press the "Complete Setup" button at the bottom of the form. This button launches another dialog window reading "Game has begun and the time will start now". Upon pressing "OK", the piece mode selector, as well as the "Complete Setup" button, is removed from the form. A timer takes they're places and begins counting up from zero.

Therefore, the player uses a graphical interface method of placing pieces as opposed to the much less intuitive and timely alternative method offered alongside in the Assignment specifications. Requirements such as "users shall be warned if the position is illegal" and "pieces must not be placed on illegal squares (white/light square)" as well as "a maximum of 12 [red] pieces and 12 black pieces may be placed on the board" are all met within this class' interface. There is also "a way for the user to indicate that set up is complete" by using the "Complete Setup" button and commencing the game.

Note that all requirements specified in the Assignment have been met at this point and that the following modules are intended to offer supplementary functions in order to improve the user interface. They do not reflect requirements established in the Assignment documentation.

ResetToolStripMenuItem

This class is offered to accompany the above two standard and custom gameplay modes. Selecting "Settings" on the menu strip of Form1 displays the "Reset" option. This restores the board to the initial blank state of Form1_Load. The timer is reset and removed along with all pieces previously on the board.

GameTimer_Tick

This class is simply the timer used during games. It is intended to allow the user to keep track his/her game play length. The timer restarts with new games and is not visible when a game mode is not selected.

QuitGameToolStripMenuItem_Click

This class, activated by the user through the menu strip, can be found under "Menu". Its purpose is to allow the user to quit the game without necessarily using the "close" button on the window bar. Selecting it activates a dialog box that reads "Are you sure you would like to quit?". Clicking "Yes" will terminate the entire game, closing all windows, whereas selecting "No" will only close the message box and return the user to Form1 and its previous state.

Form_FormClosing

Lastly, this class resembles the above QuitGameToolStripMenuItem in that it allows the user to quit the current game and close all windows. By default, clicking the "close" button on the window bar of Form1 would simply close the form without any notice. This class' purpose is to notify the user that the game is about to be terminated. Hence, a message box appears upon pressing the red "x" that reads "Are you sure you would like to quit? Clicking 'Yes' will lose any current game progress". As the message states, selecting "Yes" will close all forms whereas selecting "No" will only close the quit message box and return the user to Form1 in its previous state (as does QuitGameToolStripMenuItem).

INTERNAL REVIEW AND EVALUATION OF DESIGN

An advantage to our design is the way the game actually plays out. Selecting the standard game mode brings up a screen that notifies the user that the game is about to begin and the timer will start on their signal, instead of throwing the user into the game immediately upon launching the application. Selecting the custom game mode lets the user select where they want their pieces to be placed by clicking on the black game tiles they prefer. There is a limit of 12 pieces that can be placed for each colour, but the order in which the users place their pieces is indiscriminate (for example, two players can simply alternate between placing red and black pieces). The way it is structured makes the Ultimate Checkers very user-friendly and easily customizable.

Furthermore, choosing to create our Ultimate Checkers game using Visual Basic made the application mostly GUI based; the user can easily navigate through different options and screens using the menu at the top of the application instead of entering inputs using a keyboard. Advancing from screen to screen requires a click of a single button or a menu option which makes the application more concise and facilitates function navigation.

TESTING

We preformed testing on our program by checking if it met each of the requirements. Requirements specific to the user interface were tested by simulating user gameplay through an extensive array of user-defined actions.

The first requirement was to set up the checkers board. The requirements specified an 8-by-8 board of dark and light squares (a light square in the bottom right corner), labels running from A-H along the columns (left to right) and labels 1-8 along the rows (from top to bottom). Testing that these requirements were met was straightforward; the program was executed and the appearance of the board on the interface was compared to what was outlined in the requirements specification.

The next set of requirements concerns the user's input on setting up the board. The user is given the option to do a standard or custom board setup. This option to the user was tested by running the program and clicking on both the standard setup and custom setup options, ensuring each button put the program into the appropriate state awaiting the next user input. In the state of custom setup, the user must be able to specify custom opening positions for the pieces. This option was tested by selecting the custom setup option and placing the pieces in various arrangements. Arrangements included using only red pieces, only black pieces, mixes of regular pieces and king pieces and varying numbers of pieces on the board at once. The requirements also specify the user be warned when attempting to place a piece on an illegal (light) tile. This requirement was met by having an alert pop up telling the user he/she cannot place a piece on the square that was selected. This was tested by running the program as the user and attempting to place a piece on an illegal square. A limit on how many pieces of each colour is also specified. There may never be more than 12 of each colour on the board at once. The requirement was met by implementing a counter (hidden to the user) of how many of each coloured piece is on the board. When the user tries to put a 13th piece of one colour on the board, an alert is shown notifying the user he/she has reached the piece limit. This implementation was tested by running the program as a user and attempting to place a 13th black piece and a 13th red piece, ensuring the desired result was reached in both cases.

The final requirement was to implement a way for the user to indicate that the custom board setup is complete when outside of the standard setup mode. This requirement was met by implementing a button on the user interface labelled "Complete Setup" that when pressed would begin the game in whatever state the board was placed in by the user. This button's functionality was tested by running the program as a user and pressing the button after various piece setups as well as pressing it after a custom setup and prior to a standard setup being executed.

ANNEX

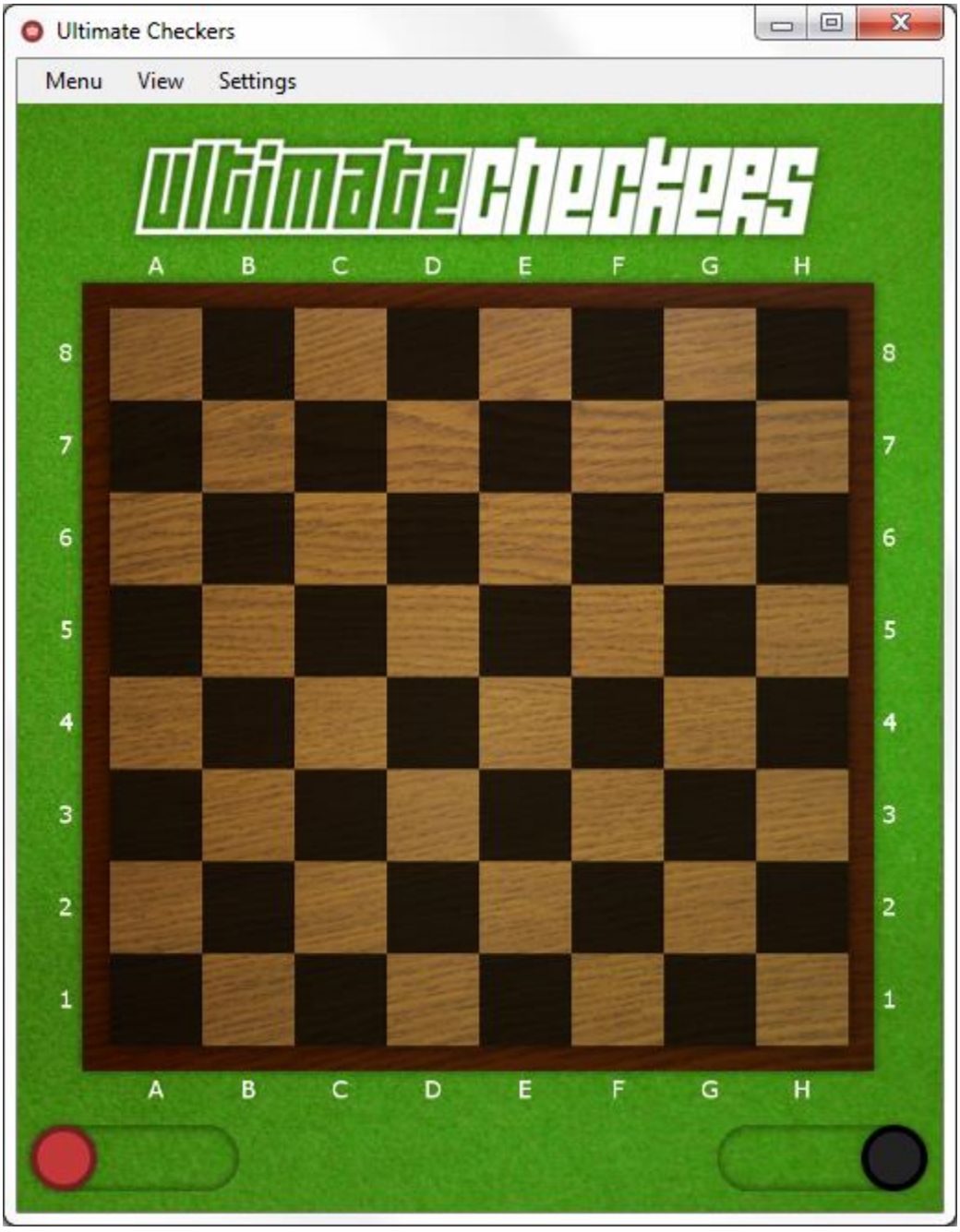

Figure 1
Default loaded checkers board.

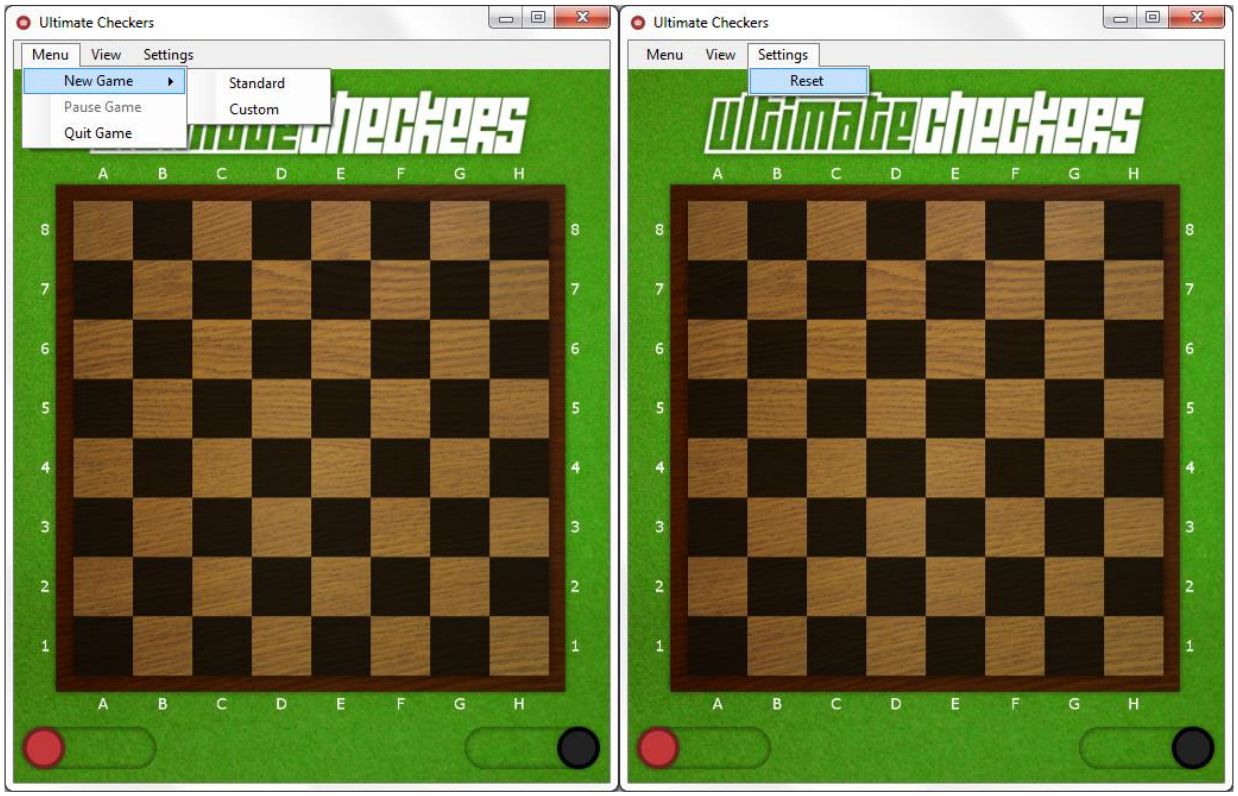

Figure 2

Examining the strip menu options at the top of the window. The “Pause Game” function is currently disabled and will be implemented at a later date.

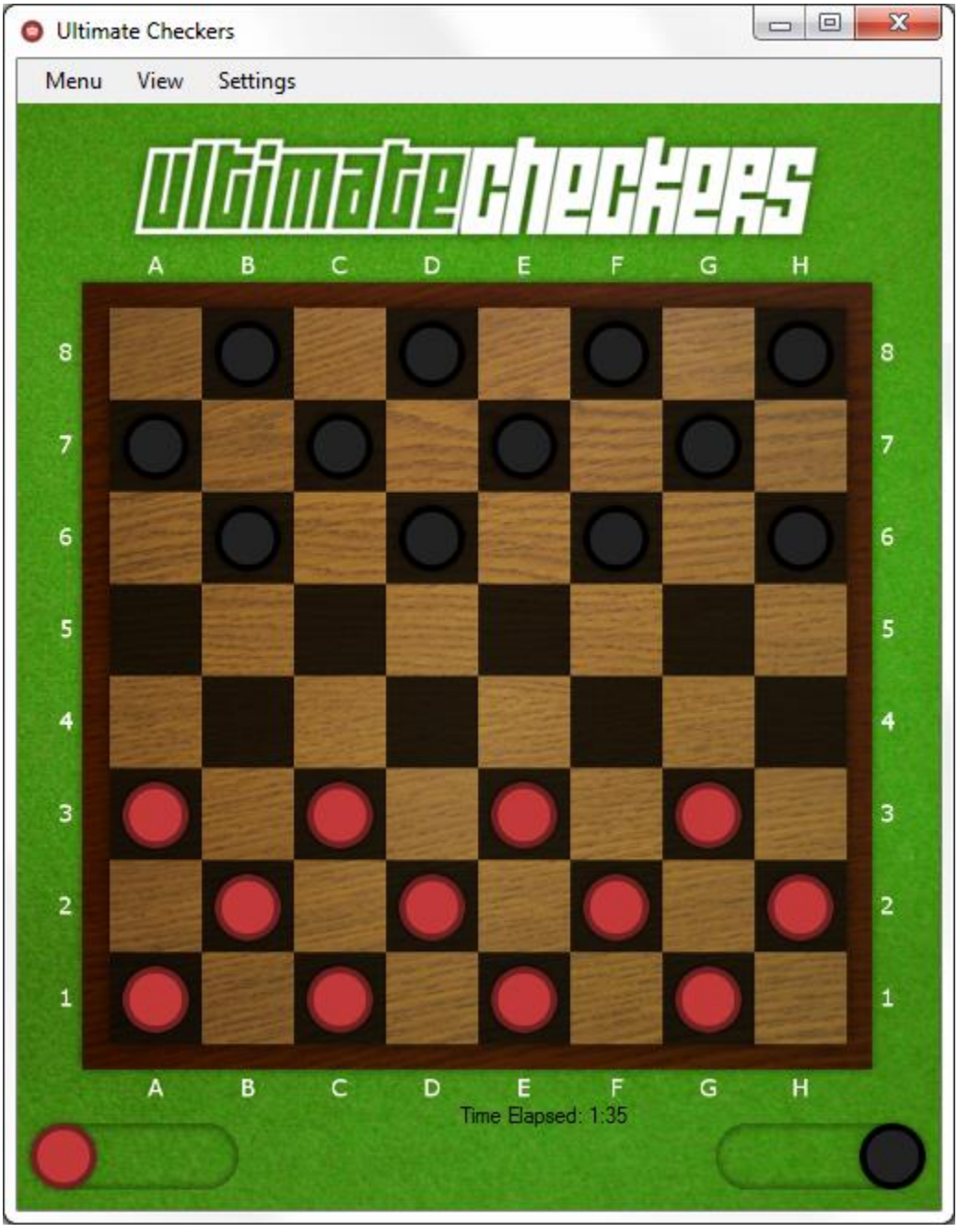

Figure 3
Board set up after selecting standard game mode.

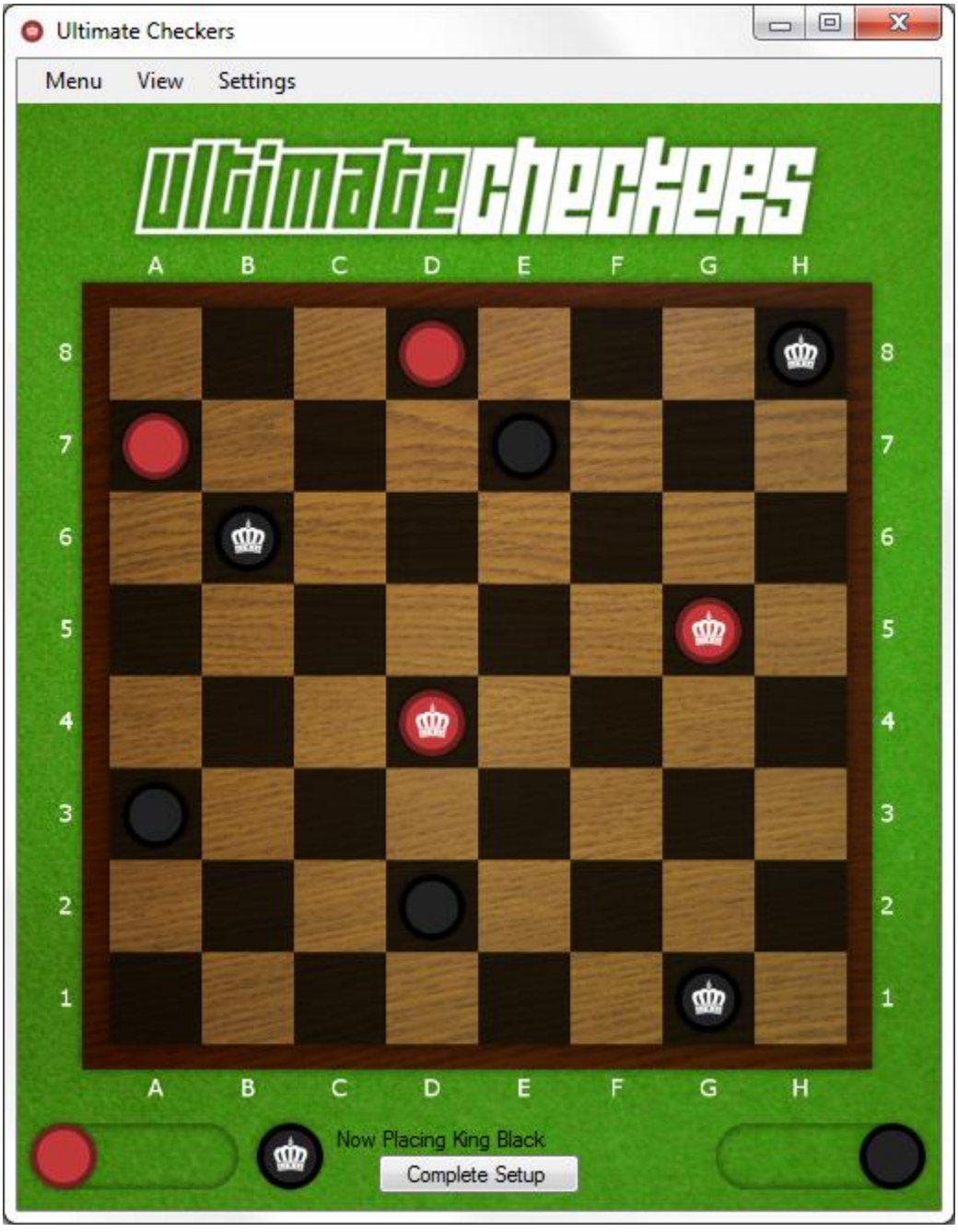

Figure 4

Board set up after selecting custom game and placing a few pieces.

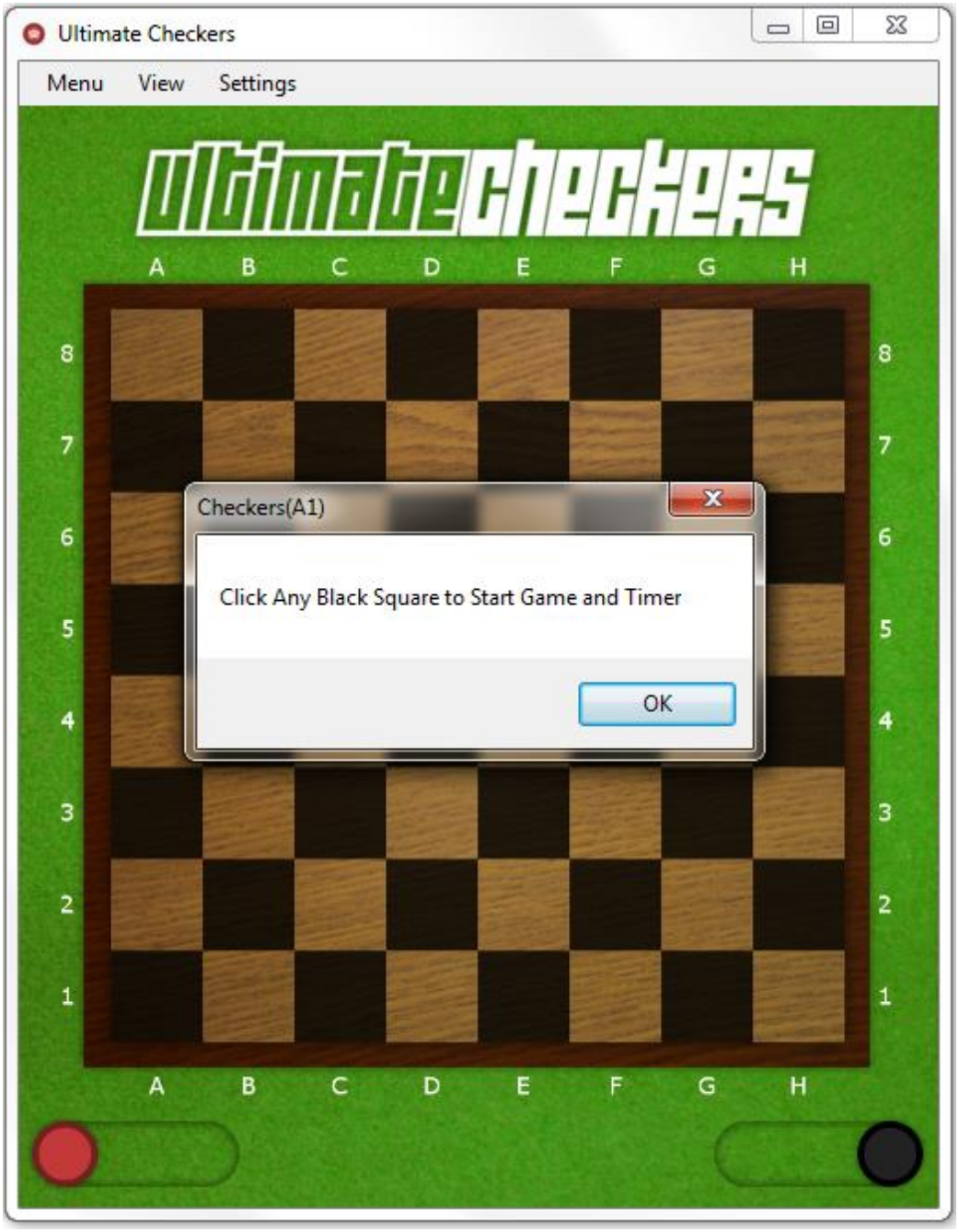

Figure 5
Message box upon selecting the standard game mode.

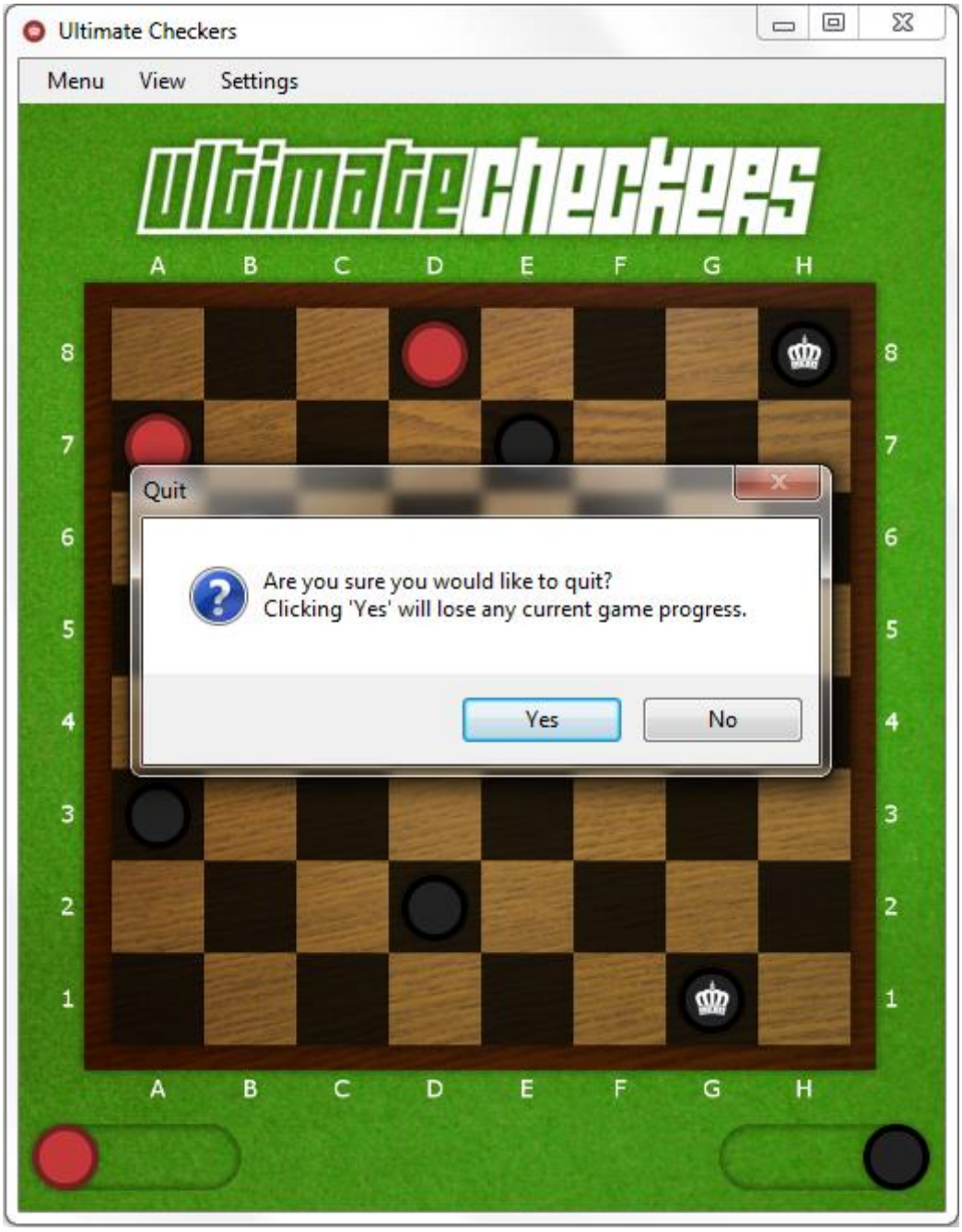

Figure 6

Message box upon attempting to exit the game by selecting the close button on the window bar.

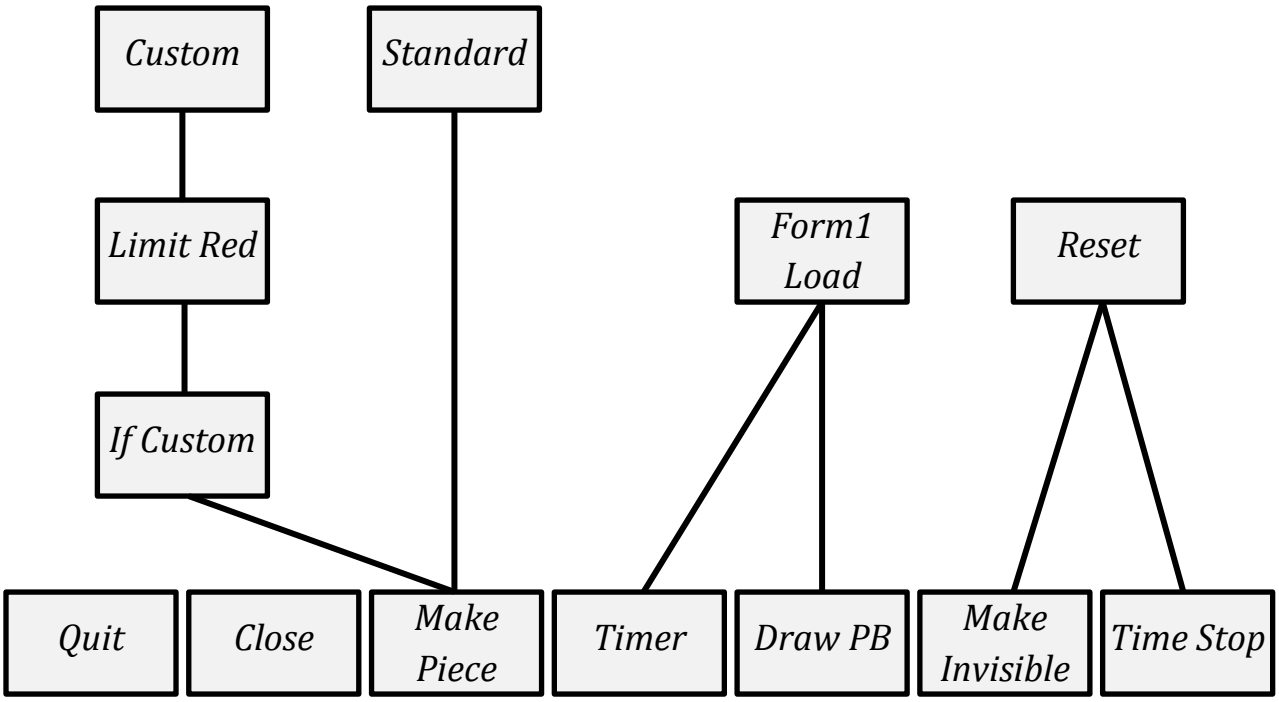

Figure 7

Current hierarchy. Note that hierarchies will increase in complexity as deliverables are added throughout further iterations of this assignment.