Friday, March 21st 2014

# ASSIGNMENT II

## *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 ..... 4

CHANGE LOG ..... 5

Interface Change Log ..... 5

Variables Change Log ..... 5

Internal Implementation Change Log ..... 5

CLASS AND MODULE DESCRIPTION, INTERFACE AND IMPLEMENTATION ..... 6

Form: Start_Menu *New* ..... 6

*Description (4.1)* ..... 6

*Interface Specification (4.2)* ..... 6

*Variables (4.5)* ..... 6

*Internal Implementation (4.5) – Selection Logic* ..... 6

*Internal Implementation (4.5) – Parameter setup* ..... 7

Sub: Custommode_load *New Name* ..... 8

*Description (4.1)* ..... 8

*Interface Specifications (4.2)* ..... 8

*Variables (4.5)* ..... 8

*Internal Implementation (4.5)* ..... 8

Sub: Game_Timer *Revised* ..... 9

*Description (4.1)* ..... 9

*Interface Specification (4.2) New* ..... 9

*Variables Used (4.5)* ..... 9

*Internal Implementation (4.5)* ..... 9

Sub: Reset *Revised* ..... 10

*Description (4.1)* ..... 10

*Interface Specification (4.2)* ..... 10

*Variables (4.5) (2 Variables No Longer being used here)* ..... 10

*Internal Implementation (4.5)* ..... 10

Function: Stop_Time *Removed* ..... 11

*Description (4.1)* ..... 11

*Variables (4.5)* ..... 11

*Internal Implementation (4.5)* ..... 11

Sub: Standard_Mode *Revised* ..... 12

*Description (4.1)* ..... 12
*Interface Specification (4.2)* *New* ..... 12
*Variables (4.5)* ..... 12
*Internal Implementation (4.5)* ..... 12
`Form: Play` *New* ..... 13
*Description (4.1)* ..... 13
*Interface Specification (4.2)* ..... 13
*Variables (4.5)* ..... 13
*Internal Implementation (4.5)* ..... 14
`Sub: Save_Game` *New* ..... 16
*Description (4.1)* ..... 16
*Interface Specification (4.2)* ..... 16
*Variables (4.5)* ..... 16
*Internal Implementation (4.5)* ..... 16
`Sub: Load_Game` *New* ..... 17
*Description (4.1)* ..... 17
*Interface Specification (4.2)* ..... 17
*Variables (4.5)* ..... 17
*Internal Implementation (4.5)* ..... 17
USES RELATIONSHIP ..... 18
REQUIREMENT TRACE BACKS ..... 19
`Start_Menu` *New* ..... 19
`Play` *New* ..... 19
`Standard_Mode` *New name & Revised* ..... 20
`Custommode_load` *New name* ..... 20
`Save_Game` *New* ..... 21
`Load_Game` *New* ..... 22
`Reset` *Revised* ..... 22
`GameTimer` *Revised* ..... 22
`QuitGameToolStripMenuItem_Click` *Revised* ..... 22
INTERNAL REVIEW AND EVALUATION OF DESIGN ..... 23
TESTING ..... 24
ANNEX ..... 25

# 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.

# CHANGE LOG

## Interface Change Log

(March 12, 2014) Added **Start Menu** – Allows the user to select game mode and load games.
(March 12, 2014) Added **Play Form** – Supports legal movement of game pieces.
(March 13, 2014) Modified **Custom Mode** – No longer shares form with standard Mode
(March 13, 2014) Removed **Standard Mode** – Interface no longer visible, standard mode is now private that passes parameters to the Form: Play
(March 17, 2014) Removed **Game Timer** – The game play no longer has a time

## Variables Change Log

| Variable Name | Declaration Location | Date Changed | Change Type |
|---|---|---|---|
| GamesetupL | Global Integer | (March 12, 2014) | Added |
| GamesetupL | Global Integer | (March 12, 2014) | Added |
| GamesetupS | Global Integer | (March 12, 2014) | Added |
| GamesetupC | Global Integer | (March 12, 2014) | Added |
| M_Badnu, | Local PictureBox | (March 12, 2014) | Added |
| C_B1,C_B2,C_R1.C_R1 | Local, Integer | (March 12, 2014) | Added |
| C_B3,B_B4,C_R3,C_R4 | Local PictureBox | (March 12, 2014) | Added |
| MoveCount | Local Integer | (March 17, 2014) | Added |
| M_Infoarray | Local String Array(31) | (March 17, 2014) | Added |
| Gamesetup | Local Integer | (March 17, 2014) | Modified |
| C_Seconds | Local Integer | (March 13, 2014) | Removed |
| C_Minutes | Local Integer | (March 13, 2014) | Removed |

## Internal Implementation Change Log

(March 13, 2014) Added Logics to **Start Menu – Custom Mode, Standard Mode, Load Game,** also included a call to **End_Game** module.
(March 17, 2014) Added Logics for Moving Pieces – **Jump**, **CrownKing**, **GetClick,** all of which are methods under the new Form: Play.
(March 13, 2014) Added Private **Standard Mode** – Functionally equal to standard mode, no longer generates a graphical interface until Form: Play loads.
(March 12, 2014) Modified **Reset –** Reset functionally works the same, some state code removed to improve modularity.
(March 12, 2014) Removed **Standard Mode** – Removed all internal coding involved with Standard Mode.
(March 12, 2014) Removed Functions no longer being used – **Stoptime**(), **MakeInvis**() (MakeInvis still a part of the code, no calls being made at the moment).

# CLASS AND MODULE DESCRIPTION, INTERFACE AND IMPLEMENTATION

## Form: Start_Menu *New*

*Description (4.1)*

This succeeded the original form_load (now named custom_load). This is what the user initially sees when the game is loaded. It presents several options for gameplay including “Start Game”, “Load Game”, “Custom Game” and “Standard Game”, all of which are available to the user.

The form’s origin is largely due to the principles of modularity as this is a place that initializes one of the possible games modes and having them done in a single location rather than making states inside the actual play module would drastically increase maintainability and reduce the possibility of information leaking through internal states.

*Interface Specification (4.2)*

- StartGame()
  - Makes Custom and Standard visible
- Custom()
  - Enters Custom Setup mode
- Standard()
  - Enters Standard game
- Load()
  - Loads saved games
- HighScore()
  - A list of the player’s achievements, not yet implemented

*Variables (4.5)*

| Variable Use | Variable Name | Variable Type | Declaration Location |
|---|---|---|---|
| Input | GamesetupC | Integer Array(32) | Global |
| Input | GamesetupS | Integer Array(32) | Global |
| Input | GamesetupL | Integer Array(32) | Global |

*Internal Implementation (4.5) – Selection Logic*

Output: Gamesetup(32) Type: Integer Array(32)

| Logic: Input | Output Action |
|---|---|
| Custom | Initialize GameSetupC |
| Standard | Initialize GameSetupS |
| Load | Initialize GameSetupL |

*Internal Implementation (4.5) – Parameter setup*

Output: Gamesetup(32) Type: Integer Array(32)

| Logic: Input | Output Action |
|---|---|
| 1 | Black |
| 2 | Red |
| 3 | Black King |
| 4 | Red King |

## Sub: Custommode_load *New Name*

*Description (4.1)*

The Custom_load sub is done as soon as the custom mode is loaded. Therefore, this sub becomes necessary for declaration of initial variables and drawing graphic pieces. As a result, we 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 | Declaration Location |
|---|---|---|---|
| Output | C_track | Integer | Local |
| Output | InitialX | Integer | Local |
| Output | C_trackarray | PictureBox Array(31) | Global |
| Output | M_custom | Boolean | Local |
| Output | M_standard | Boolean | Local |
| Input | ErrorClick | PictureBox | Local |

*Internal Implementation (4.5)*

This code is meant to create picture boxes at given locations in a square matrix in order to display the black tiles. The logic is to have a variable (i) increase by 50 pixels each time and draw a new picture box at that location. At every 4 i variables, a j variable will be present to move 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
```

## Sub: Game_Timer *Revised*

*Description (4.1)*

This function counts the time elapsed since a game has started and stores them inside a seconds variable available to the user.

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)* *New*

The time works internally and will not display a time at this deliverable.

*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 (1 second in length) and updates the seconds local variable, providing a simple yet 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 *Revised*

*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 a unique module since modules are dependent of global variable states, which allow this sub to be independent, improving 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.

*Variables (4.5) (2 Variables No Longer being used here)*

| Variable Use | Variable Name | Variable Type | Declaration Location |
|---|---|---|---|
| Output | Trackarray | Picture box array (31) | Local |

*Internal Implementation (4.5)*

Reset is now a simpler module and its responsibilities are to reset all the checker pieces when clicked in custom mode.

```
For i in range of trackarray
    Trackarray(I).image = nothing
Next i
```

## Function: Stop_Time *Removed*

*Description (4.1)*

Completely 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: Standard_Mode *Revised*

*Description (4.1)*

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

The reasoning behind this module is to have it tie in with the save functionality in the game. Both of these require a logic from an array of integers for piece setting.

*Interface Specification (4.2)* *New*

This sub is no longer a part of the interface, now it is completely private and only updates the game setup for other modules.

*Variables (4.5)*

| Variable Use | Variable Name | Variable Type | Declaration Location |
|---|---|---|---|
| Input | M_custom | Integer Array(32) | Local |
| Input | M_standard | Integer Array(32 | Local |

*Internal Implementation (4.5)*

Output: Gamesetup(32), Type: Integer Array(32)

| Logic: Input | Output Action |
|---|---|
| 1 | Black |
| 2 | Red |
| 3 | Black King |
| 4 | Red King |

## Form: Play *New*

*Description (4.1)*

This is the form where the entire playing experience occurs. It is an individual form and module. This is also the largest module present and has many methods pertaining to setting up the board and logic for moving/jumping the pieces.

We decided that a logical convergence for all previous calculations would lead here for an actual playing experience. Due to the fact that this module takes setups and contains movement logic, it supports the modularity principles. Also note that this module shares minimal logic from others and has very high rate of cohesion, making it an ideal choice for a well-defined component of this design.

*Interface Specification (4.2)*

- Gamesetup()
  - Sets up the game as dictated by the user previously (either through loading, custom setup or standard setup)
- Movepiece()
  - Allows the user to make legal moves including sideways diagonally, jumping and reverse movement for king pieces
- CrownKing()
  - If the user moves a piece to the opposing end, the piece is immediately crowned king and gains its inherent system of movement

*Variables (4.5)*

| Variable Use | Variable Name | Variable Type | Declaration Location |
|---|---|---|---|
| Input | GamesetupC | Integer Array(32) | Global |
| Input | GamesetupS | Integer Array(32) | Global |
| Input | GamesetupL | Integer Array(32) | Global |
| Input | M_Badnum | Picturebox | Local |
| Internal | C_B1 | Integer | Local |
| Internal | C_B2 | Integer | Local |
| Internal | C_R1 | Integer | Local |
| Internal | C_R2 | Integer | Local |
| Internal | C_B3 | Picturebox | Local |
| Internal | C_B4 | Picturebox | Local |
| Internal | C_R3 | Picturebox | Local |
| Internal | C_R4 | Picturebox | Local |
| Internal | BlackT | Boolean | Local |
| Internal | RedT | Boolean | Local |
| Internal | MoveCount | Integer | Local |
| Output | M_Infoarray | String Array (31) | Local |

*Internal Implementation (4.5)*

### *Board Setup*

Similarly to the previous custom mode, this section requires a game board, as setup by the following pseudo code:

```
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
```

### *Move Logic*

Output: {C_B1, C_B2, C_R1, C_R2}, Types: Integers

| Logic: Row Number | Output Action: C_R1,C_R2 | Output Action: C_B1,C_B2 |
|---|---|---|
| Odd | 3,4 | 4,5 |
| Even | 4,5 | 3,4 |

### *Detect Piece Logic*

Updates: MoveCount, Types: Integers

| Logic: Click | Update: Movecount (Current Value: 0) | Update: Movecount (Current Value: 1) |
|---|---|---|
| Object: Empty | 0 | 1 |
| Object: Not Empty | 1 | 0 |

### *Jump Logic*

This logic follows the move piece logic and simply checks if the moves will land on other pieces or not. If a jump can be made, it removes the occupied space from possible moves and adds the jumped location.

Output: {C_B1, C_B2, C_R1, C_R2}, Types: Integers

| Logic: Next Piece | Update: Same Color | Update: Different Color |
|---|---|---|
| Top Right Corner: Empty | C_B1, C_B2, C_R1, C_R2 | N/A |
| Top Right Corner: Not Empty | C_B1, C_B2, C_R1, 0 | C_B1, C_B2, C_R1, C_R2+4 |
| Top Left Corner: Empty | C_B1, C_B2, C_R1, C_R2 | N/A |
| Top Left Corner: Not Empty | C_B1, C_B2, 0 C_R2 | C_B1, C_B2, C_R1+4, C_R2 |
| Bottom Left Corner: Empty | C_B1, C_B2, C_R1, C_R2 | N/A |
| Bottom Left Corner: Not Empty | 0, C_B2, C_R1, C_R2 | C_B1+4, C_B2, C_R1, C_R2 |
| Bottom Right Corner: Empty | C_B1, C_B2, C_R1, C_R2 | N/A |
| Bottom Right Corner: Not Empty | C_B1, 0, C_R1, C_R2 | C_B1, C_B2+4, C_R1, C_R2 |

### *King Logic*

Outputs: {King, No Change}, Type: Picturebox

| Piece Location/ Color | Output |
|---|---|
| 28-31/ Red | Red King |
| 28-31/ Black | No Change |
| 0-3/ Red | No Change |
| 0-3/ Black | Black King |
| Else | No Change |

## Sub: Save_Game *New*

*Description (4.1)*

This is a drop down menu option in the play game form that allows the user to save the current progress. Once the save is complete, the user is free to continue playing or close the game and load their progress next time in order to resume gameplay.

The save functionality is put into one module as it is necessary to maintain modularity, this area of code links to a form object and is well defined by itself.

*Interface Specification (4.2)*

- Save_Game()
  - Following the click of this button on the checkers interface, the game will be saved

*Variables (4.5)*

| Variable Use | Variable Name | Variable Type | Declaration Location |
|---|---|---|---|
| Output | Gamesetup | Integer Array(32) | Local |

*Internal Implementation (4.5)*

The implementation of this module is a simple write to file operation as modelled by the following pseudo code:

```
Dim writer as print writer
Write(../SavedGame.txt)
Writer(gamesetup)
```

## Sub: Load_Game *New*

*Description (4.1)*

This is a button in the start menu that allows the user to load their previously saved games, if no game was previously saved, the interface returns an error message notifying the user.

This sub was decomposed into a module by itself as it is an object on a form and contains a well-defined area of code.

*Interface Specification (4.2)*

- Load_Game()
    - Following the click of this button, the previously saved game will be loaded or an error message will be shown indicating that no saved game exists

*Variables (4.5)*

| Variable Use | Variable Name | Variable Type | Declaration Location |
|---|---|---|---|
| Input | GamesetupC | Integer Array(32) | Global |
| Input | GamesetupS | Integer Array(32) | Global |
| Input | GamesetupL | Integer Array(32) | Global |

*Internal Implementation (4.5)*

The implementation of this module is a simple read file operation as modelled by the following pseudo code:

```
Dim reader as print writer
read(../SavedGame.txt)
if reader.empty = true
     msgbox (“No Saved Game”)
end if
gamesetupC(0 to 32) = 0
gamesetupS(0 to 32) = 0
gamesetupL(0 to 32) = reader
```

## USES RELATIONSHIP

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

| **Start_Menu** |
| --- |
| **Uses**:<br>Custommode_Load<br>Load_Game<br>Standard_Mode |

| **QuitGameToolStripMenuItem_Click** |
| --- |
| **Uses**: None Yet |

| **Form_FormClosing** |
| --- |
| **Uses**: None Yet |

| **Form1_Load_Game** |
| --- |
| **Uses** :Nothing |

| **MakePiece** |
| --- |
| **Uses**: LimitCheckRED<br>LimitCheckBLK<br>StandardToolStripMenuItem_Click<br>CustomToolStripMenuItem_Click<br>GameTimer_Tick |

| **Standard_Mode** |
| --- |
| **Uses**: Nothing |

| **Custommode_load** |
| --- |
| **Uses**:<br>ExitMode_Click<br>LimitCheckBLK<br>LimitCheckRed |

| **Play** |
| --- |
| **Uses**:<br>Standard_Mode<br>Custommode_load<br>Load_Game |

| **ExitMode_Click** |
| --- |
| **Uses**: Nothing |

| **Reset** |
| --- |
| **Uses**: Nothing |

| **LimitCheckBLK** |
| --- |
| **Uses**: Nothing |

| **LimitCheckRED** |
| --- |
| **Uses**: Nothing |

| **CustomPiece_Click** |
| --- |
| **Uses**: CustomLoop |

| **CustomLoop** |
| --- |
| **Uses**: None Yet |

| **CustomInfo_Click** |
| --- |
| **Uses**: CustomLoop |

| **ErrorClick_Click** |
| --- |
| **Uses**: Nothing |

| **Game_Timer** |
| --- |
| **Uses**: Nothing |

| **Save_Game** |
| --- |
| **Uses**: Play |

# 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.

Note that changes to the design have been indicated with *New*, meaning we have implemented a new class previously inexistent, *New name*, meaning we have simply renamed the class for increased coherency, and *Revised*, meaning we have revised the class to manage its efficiency and tasks handled. Note that classes missing from this version have been removed since the last and are therefore irrelevant to the systems current traceability.

### `Start_Menu` *New*

The `Start_Menu` window is now the first one seen when launching the application as, originally, a player would simply be thrown into a standard game. However, with the option to save and load games, a more adequate solution was necessary to allow the user to choose which game method he/she prefers upon start-up. Usability is thus greatly enhanced. The page itself contains buttons including “Start Game” and “Load Game”. “Load Game” will launch a game previously saved (launches `Load_Game`) whereas “Start Game” will give two game option: “Standard” or “Custom”. As is obvious by the names chosen, “Standard” will load a checkers board arranged with the pieces in a standard manner (ie. three rows of red and black pieces on opposite sides, launches `Standard_Mode`) and “Custom” will give the player the opportunity to design a custom game, placing up to 12 pieces of each colour on any black tile (launches `Custommode_Load`).

Even though `Start_Menu` does not accomplish any of the requirements described in Assignment 2, it is imperative to the applications ease of use as it offers a simple user interface (UI) from which a player may select from many game modes.

### `Play` *New*

This form, adapted from the previous `Form1_Load`, 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, `Play` 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.

Originally, `Play` (or `Form1_Load`) managed two independent modules which allowed their own playing experience. This, however, necessitated an enormous amount of code duplication. It was therefore decided that the `Play` form would be the one on which all gameplay would be controlled, eliminating otherwise necessary repetition and, hence, avoiding possible pitfalls such as duplicated bugs.

This module therefore imports game setups from `Standard_Mode`, `Custommode_load` and `Load_Game` as well as containing all movement logic. The `Play` module allows a user to "start a game from a previously stored state", "make moves" that include "mov[ing] pieces from one position to another" that include "mov[ing] a piece to another square; jump[ing] the opponent's piece (so that piece is removed from the board); convert[ing] a piece to a "king" [and] mov[ing] kings in both directions (forwards and backwards)" (Assignment 2). This is all accomplished using a graphical user interface in which the user must simply click the piece he/she desires to move and then select the empty square they would like to move the piece to. The action will be completed providing the move is legal.

### `Standard_Mode` *New name* & *Revised*

As was aforementioned, `Start_Menu` included a button "Standard" under "Start Game". Selecting this game option pushes the locations of the game pieces to the `Play` window, displaying a standard game piece arrangement. 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" in the start menu, the play window opens, replacing the initially viewed menu. It then 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.

### `Custommode_load` *New name*

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 start menu after selecting "Start Game" followed by "Custom".

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. At this point, the `Custommode_load` window disappears, giving way to the `Play` window which has imported the piece locations designated by the user while in custom set-up. The game is then ready to start.

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.

### `Save_Game` *New*

The `Save_Game` module allows a user to save a game in progress. At this point, any legal game play mode may be saved which includes standard and empty boards.

Saving a game may be accomplished through the menu strip located on the `Play` form under "Menu". Selecting "Save Game" from these options will record piece locations in a file which may be retrieved whenever "Load Game" is executed. The state of the game in progress is therefore "saved within a file [which may] be resumed later" (Assignment 2).

Requirements are therefore met in order to save game progress at any moment during gameplay, allowing the user to record a games state to be retrieved and reloaded at a later date. Also note, since there is only one save slot at this time, that saving multiple times overwrites any older game state between the current game and past games.

### Load_Game *New*

As opposed to an option within the Play menu strip, Load_Game can be found as a game mode in the Start_Menu form. Simply put, this allows Play to load piece locations saved previously using Save_Game. Since there is, at the moment, only one storing state, Load_Game loads the last-saved (most currently saved) game. This allows a user to play a game saved at an earlier date.

*Note that all requirements specified in the Assignments 1 and 2 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.* 

### Reset *Revised*

This class is offered to accompany the above custom gameplay mode only. Selecting "Settings" on the menu strip of `Custommode_load` displays the "Reset" option. This restores the board to the initial blank state of `Custommode_load`, allowing the user to restart setting the piece locations from a "blank state".

### GameTimer *Revised*

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. Note that the game timer was originally visible, but is now hidden upon further review.

### QuitGameToolStripMenuItem_Click *Revised*

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 `Play` and its previous state. This feature has been reassigned exclusively to `Custommode_load` for the time being.

# INTERNAL REVIEW AND EVALUATION OF DESIGN

In comparison to the previous version of Ultimate Checkers (UC), the user interface and the game appearance have been modified while the ability to save and load games has also been added. The application's functionality was also drastically improved.

Upon start-up of the application, the user has a window consisting of 3 buttons to select from: "Start Game", "Load Game" and "High Scores". Selecting "Start Game" shows two more buttons – "Standard" and "Custom" – both of which function the same way the menu strip options in the previous version of the game did. Selecting "Load Game" functions as its name would suggest: UC loads the **most recent** user-saved game. Selecting "High Scores" would not do anything at this point in time but is geared towards displaying player achievements.

After the set-up process is complete on either game mode, the game reverts to the original style it had in the previous version of UC. However, there are minor adjustments and new additions to the game window. Because the ability to choose game modes was in the initial start-up window of the game, that option has been removed from the main menu. Instead, two options are now introduced: "Save Game" and "Exit Application". Selecting "Save Game" saves the game to the game directory, and **overwrites** any previously saved game while selecting "Exit Application" quits UC.

The improvement in functionality of the application is evident when the user starts playing the game. Selecting a piece shows the location of the piece at the bottom of the checkerboard and allows the user to perform valid moves. This includes jumping opponent pieces, becoming a king once the piece reaches the opposite side it started from, and moving backwards and forwards once becoming a king.

Changes brought to this iteration of Ultimate Checker's design are pertinent to the efficiency of the application as well as to maintainability and ease of use.

# TESTING 

Testing at this stage of the program development was conducted in the manner as before. We ran the program acting as the user and checked to ensure each of the requirements were met flawlessly.

It is required that the game be started from either a standard or custom setup. These two requirements were tested earlier as a part of assignment 1's deliverables. Changes were made to the start-up menu as the UI now has two different buttons before the standard and custom setup options are given. These two buttons provides the user with the option to start a new game or load a previous game. Functionality of the "Start Game" button was tested by clicking the button and ensuring it put the program into the next desired state (Standard and Custom setup buttons appear). The "Load Game" button was implemented to meet the requirement of being able to load a game that was previously saved. Testing this feature will be covered below.

During gameplay, it is required that the pieces be able to make all legal moves. This includes red pieces being able to move forwards and jump black pieces, black pieces to advance downwards and jump red pieces as well as king pieces being able to move and jump in all directions. We checked this requirement by running the code from the state in which the board is set up and moving the pieces around manually. We checked to ensure that red pieces could only move and jump forward, black pieces could only move and jump downward and kings could move and jump in any direction (all strictly within legal black spaces). It is also required that when a piece is jumped that it be removed from the board. This was tested by jumping over each type of piece with every other possible one (ie. ded piece jumped by black normal, black king, black piece jumped by red normal, red king…) and ensuring the jumped piece was removed from the board accordingly. The requirements also specify that a piece be changed from a normal piece to a king piece when advanced all the way across the board. Testing was done on this function by running the code as a user and moving a piece from its second furthest row from starting position to its far end of the board and ensuring it was changed to a king piece. All of these piece movement possibilities were tested from both in standard and custom setup.

As a final requirement, the game must be able to be saved and resumed at a later date. This option was implemented by putting a save game option on the "Menu" drop down menu. This option saves the current layout of the pieces. Once the application is exited, the "Load game" option on the start-up menu would allow the last saved game to be resumed. This was tested by running the code and attempting to save and load from both a standard and custom setup (from various layouts of pieces within each state).

An additional feature we added was the ability for the user to clear the entire board and start fresh while doing a custom setup. This user ability was implemented by putting a "Reset" option under the "Settings" drop down menu in custom setup mode. Functionality of this option was tested by doing various piece layouts and clicking the reset button, ensuring it cleared the board properly.

# ANNEX

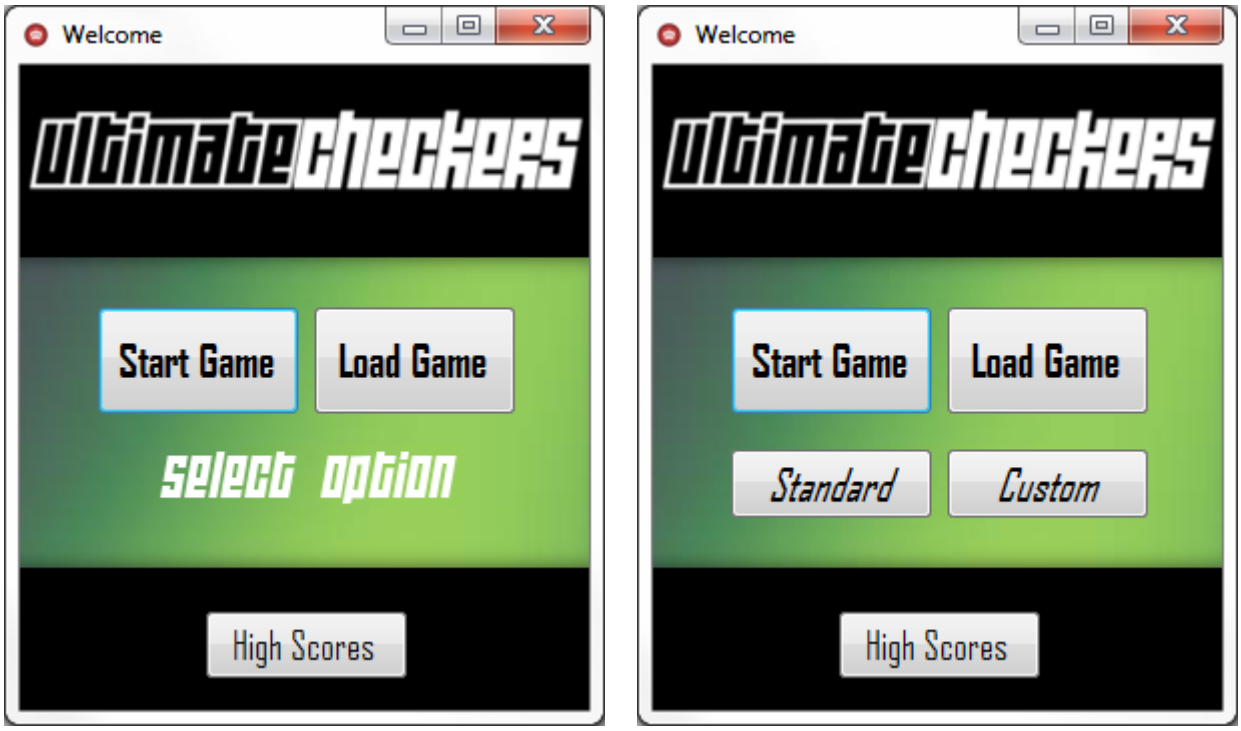


*Figure 1*

Initially loaded start menu. Used for selecting game modes before starting a game or potentially viewing high scores.

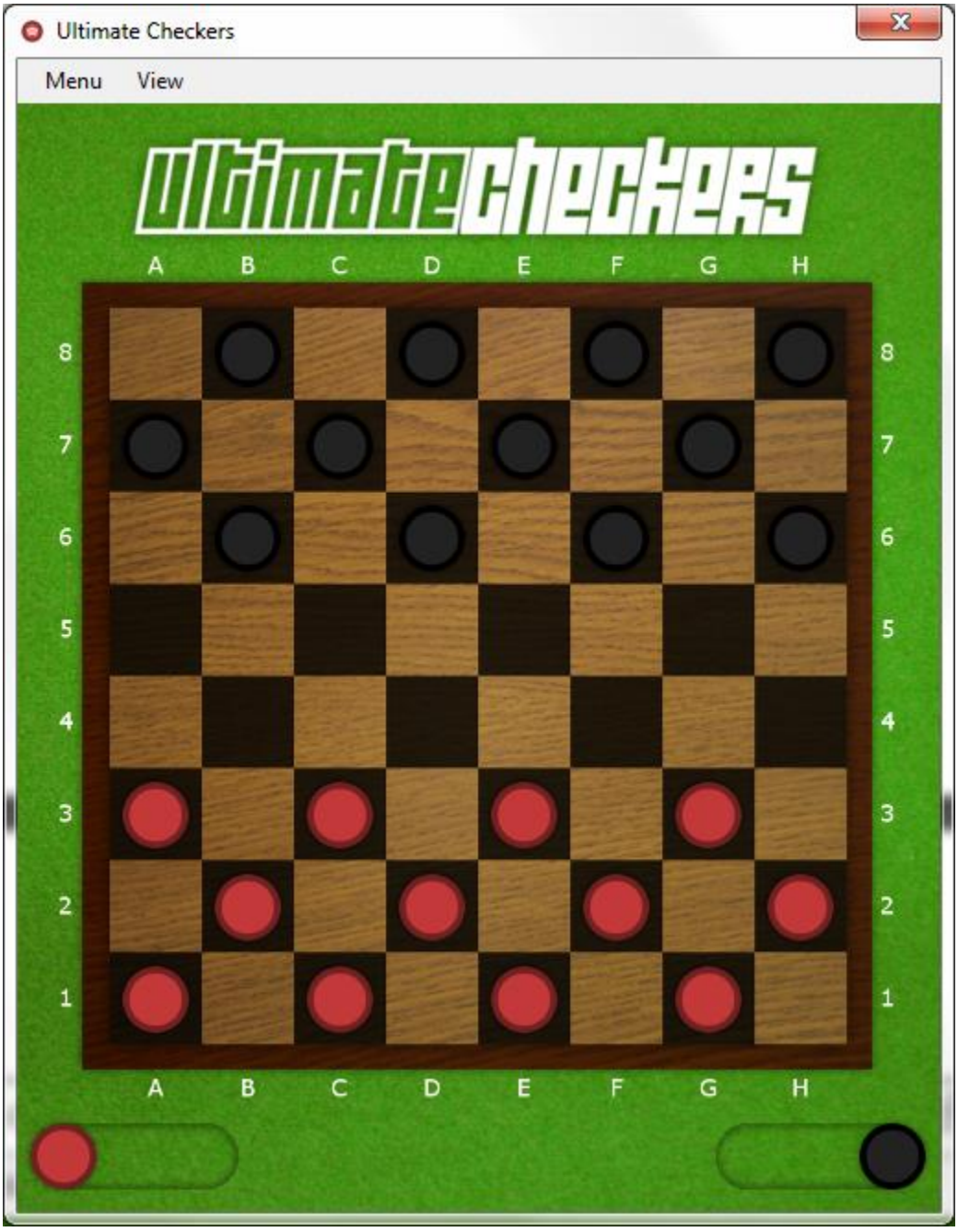


*Figure 2*
Standard game board configuration.

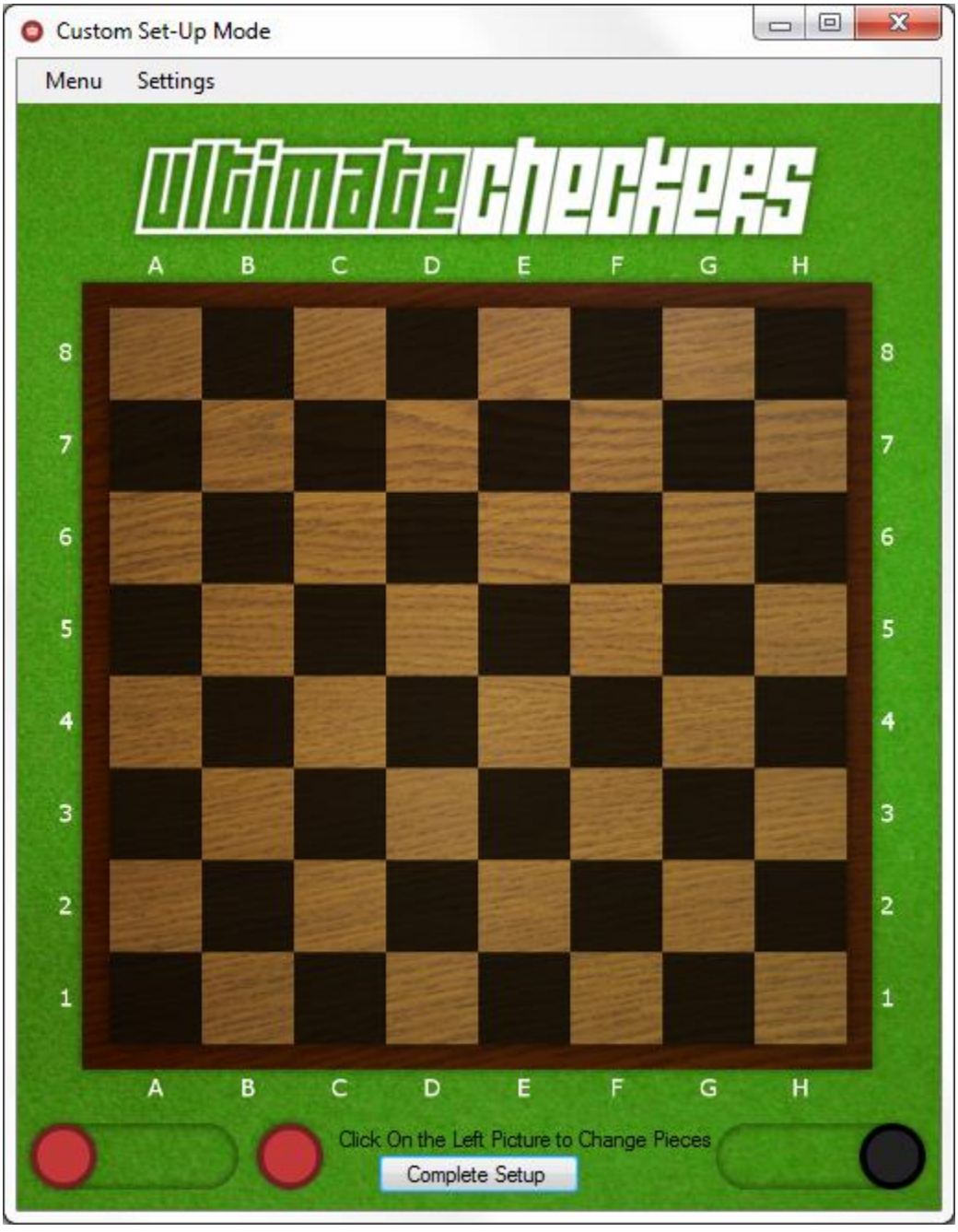


*Figure 3*
Empty (Reset) custom set-up configuration.

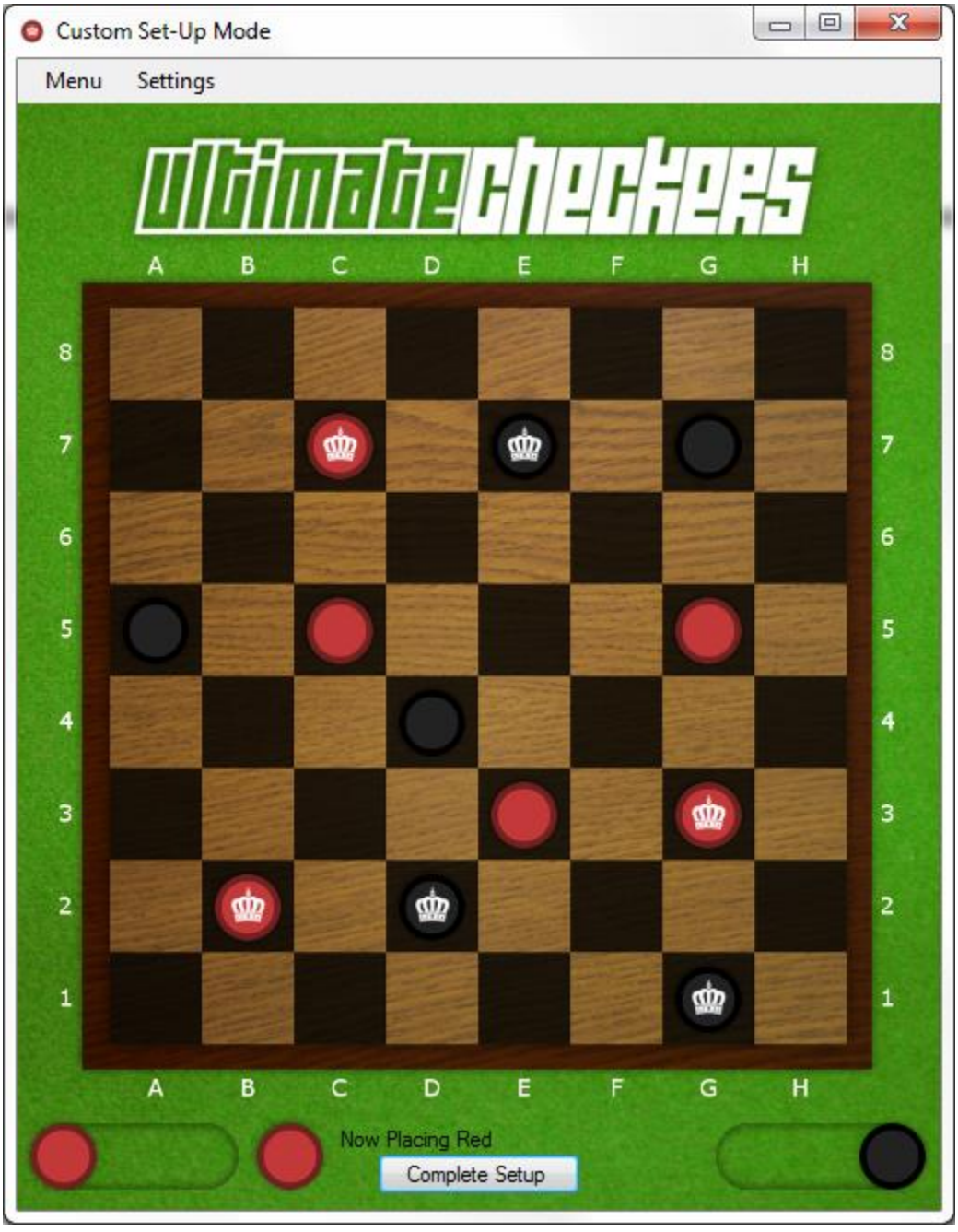


*Figure 4*
Custom game mode set-up.

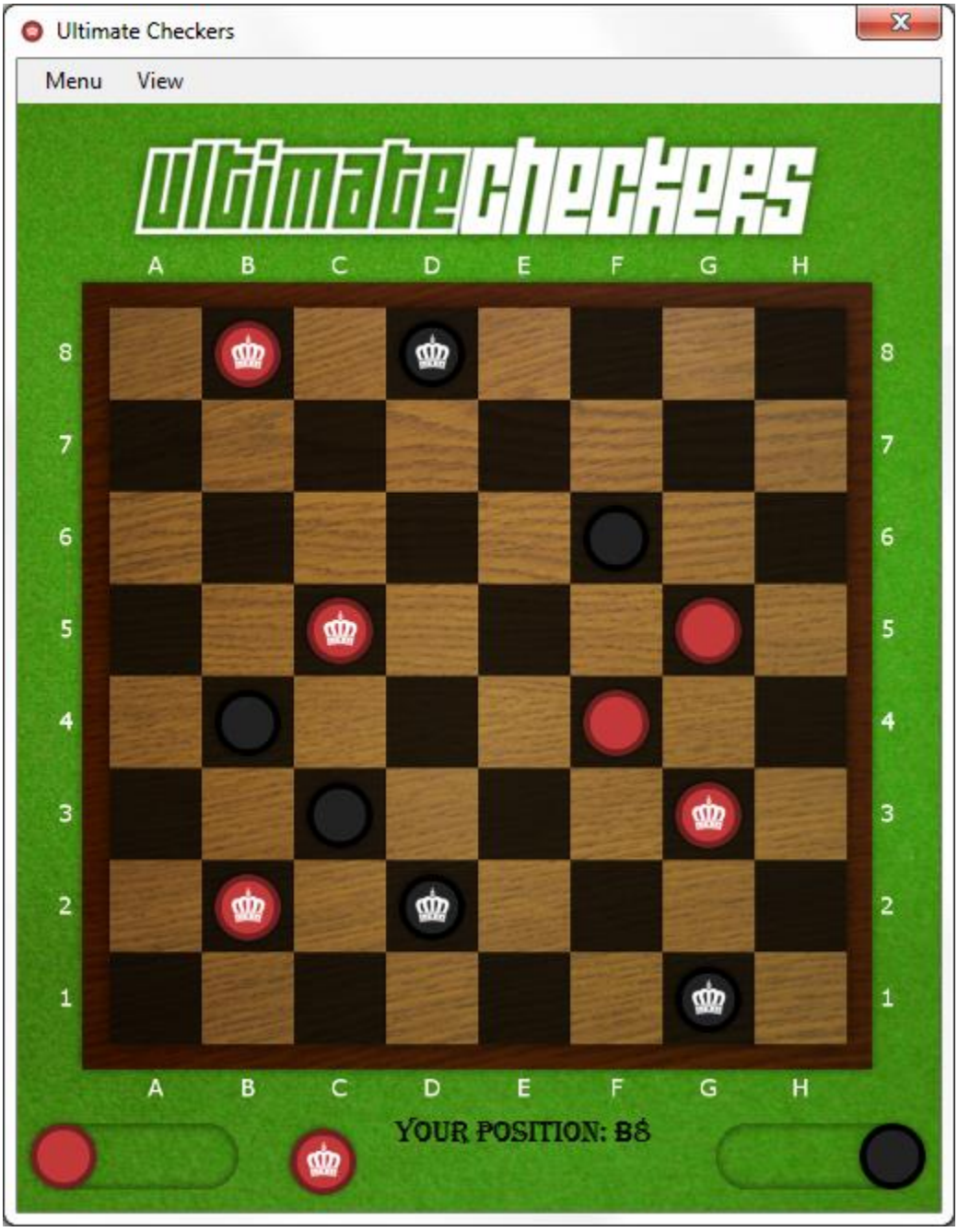


*Figure 5*

In-game mode after completing set-up of a custom game and moving a few pieces.

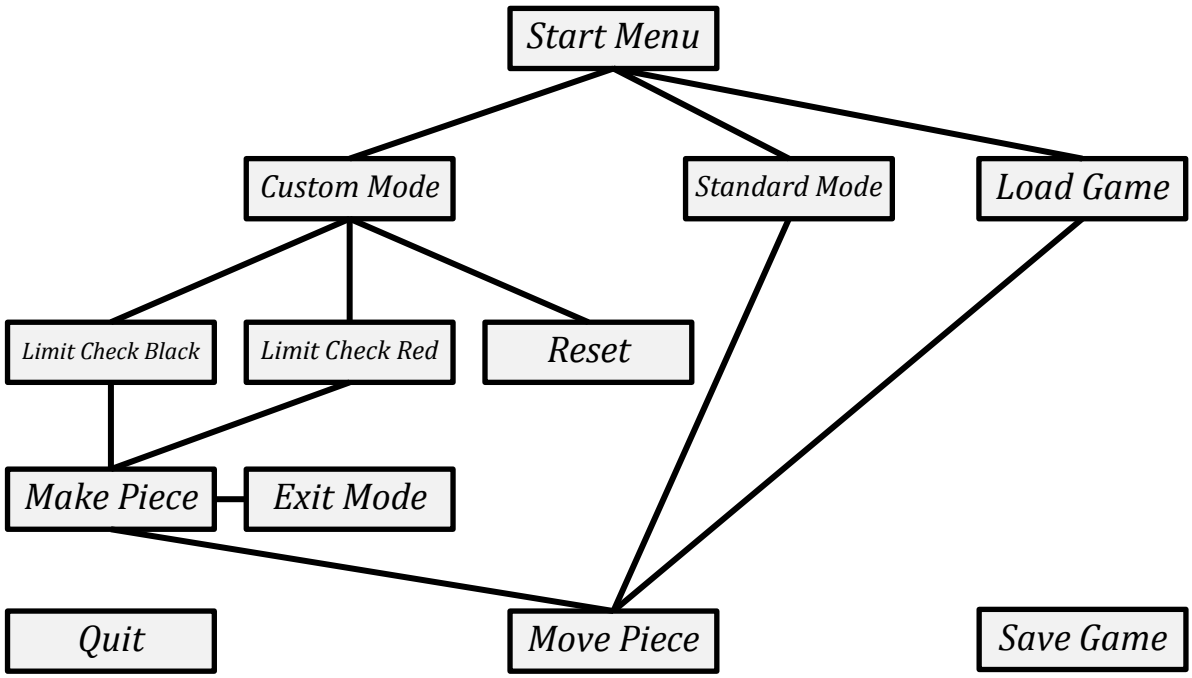


*Figure 6*

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