**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 1 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 Used (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: Custom\_load (New Name)**

**Description (4.1):** The Custom\_load sub is done as soon as the custom mode 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 action.

**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 to represent the black checker boxes. 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 the can draw the normal checker board setup and will handle clicks during custom mode. This sub will later have methods to allow it to in cooperate save and load. 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 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()

Let’s 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 one of itself because of future 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 Form Close**

**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. 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 built in.

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 this setting the internal to 1second 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 is because the other modules is dependent of global variable states which allows this sub to not be dependent on 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.

**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) (New):**

Reset is now a simpler module and its responsibilities are to reset all the checker pieces when clicked in custom mode. This is implemented with the pseudo code below:

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:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable Use** | **Variable Name** | **Variable Type** | **Declaration Location** |
| Output | C\_Minutes | Integer | Local |
| Output | C\_Seconds | Integer | Local |

**Internal Implementation:**

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 specified, Clicks are gotten 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 so it is 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 and counts the pieces, imposes false when limit is reached and ceases the placement of pieces in 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 trigger 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 functionality in the game. Both of these requires a 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) (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) | Global |
| Input | M\_standard | Integer Array(32 | Global |

**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 is done, it is an individual form and forms a module by itself. This is the largest module that there is in this project and has many methods all specifically pertaining to setting up the board and logic for moving/jumping the pieces.

The group 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 move logic, it supports the modularity principles. This module minimal logic from other modules and has very high cohesion, making it an ideal choice for a well-defined component of this design.

**Interface Specification (4.2):**

Gamesetup()

Sets up the game as dictate by the user previously, either through loading, custom setup or standard setup

Movepiece()

Allows the user to make legal moves, including sideways moving, jumping and reverse moving for kings

CrownKing()

If the user moves a piece to the opposing end, the piece is immediately crowned king and gains the movability of a king

**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 | Private/ Local |
| Internal | C\_B1 | Integer | Private/ Local |
| Internal | C\_B2 | Integer | Private/ Local |
| Internal | C\_R1 | Integer | Private/ Local |
| Internal | C\_R2 | Integer | Private/ Local |
| Internal | C\_B3 | Picturebox | Private/ Local |
| Internal | C\_B4 | Picturebox | Private/ Local |
| Internal | C\_R3 | Picturebox | Private/ Local |
| Internal | C\_R4 | Picturebox | Private/ Local |
| Internal | BlackT | Boolean | Private/ Local |
| Internal | RedT | Boolean | Private/ Local |
| Internal | MoveCount | Integer | Private/Local |
| Output | M\_Infoarray | String Array (31) | Private/Local |

**Internal Implementation (4.5)-Board Setup**

Similarity to previous custom mode, this section is going to require 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

**Internal Implementation (4.5)-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 |

**Internal Implementation (4.5)-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 |

**Internal Implementation (4.5)-Jump Logic**

This logic follows the move piece logic, it simply checks if the moves will land on the pieces of another piece 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 |

**Internal Implementation (4.5)-King Logic**

Outputs: {King, No Change} Types: 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 |

**Preface:**

The programing language used is Visual Basics, under the visual studios 2013 VB.net. This programing language does not follow the same implementation as others and 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 and that is due to the high graphical nature of this assignment. The group will later use bottom-up logic in constructing and designing future code.