Red Text = Changes/Additions to document for Assignment 3

Class: Checkers

This class has the set up of the checkers board game. It is the first thing made available to the user.

Uses: Tile

Checkers uses Tile to store and access states and information about the different tiles on the gameboard

Variables:

win: GraphWin
Creates the window that displays the game
isCustom: Boolean variable removed, replaced with 'state'
Determines whether or not the board is in custom setup mode
isStart: Boolean variable removed, replaced with 'state'
Determines if the game is in the 'playing' stage
state: String
Determines the state of the game (ie if in custom setup mode)
The default state is 'CustomSetup'
placeColour: String
Determines colour of the next piece to be placed
Default setting is 'White'
placeState: String variable removed, replaced with 'placeRank'
Determines if the tile to be placed is a 'Pawn' or a 'King'
Uses checker board
isKing: String variable removed, replaced with 'placeRank'
Determines if the next piece will be a King
placeRank: string
Determines whether the checker piece is a pawn or a king.
Default setting is 'Pawn'
gridLetters: Array of Strings previous name was 'letters'
Labels the columns of the checkers board
Exit_txt: Text Object (not string) variable removed
Label for the exit button
Place: Boolean variable removed, replaced with placeType
Determines whether the next click is going to place a piece
placeType: string
Determines whether a piece is to be placed or deleted
Default setting is 'Place'
State: string variable removed, replaced with placeRank

Determines whether a king or a pawn is being placed

tiles: array variable renamed to 'tiles,' previously called 'P_array'

Two-dimensional array that holds the tile objects

colour: string variable removed

represents the colour of a checker tile

is1P: Boolean

Determines whether the game is a one player game (player vs. computer) or a two player game (player vs. player). Default setting is False

pTurn: String

Determines whether it is the white players turn or black players turn.

Default setting is 'White'

selectedTile: string

Default setting is an empty string.

selectedTileAt: array

Determines the location of a tile the user has selected

Default setting is an empty array

hasMoved: Boolean

Determines whether a legal move has been made

Default setting is False

pieceCaptured: Boolean

Determines whether a checker piece has been captured or not

Default setting is False

BoardDimension: int

Stores the size of the game board

Default setting is 8

numPiecesAllowed: int

Determines the total number of checker pieces allowed on each side

Default setting is 12

compIsColour: string

Determines which colour the computer is playing as

Default is 'not playing'

moves: array

Stores the possible moves of the computer

Default is an empty array

badMoves: array

Stores the moves deemed as bad, to give the computer strategy

Default is an empty array

Access Programs:

SetPieces(s) No longer in code

Allows the user to set pieces on the board, in whatever valid setup they choose

Variables: (Temporary for Assignment 1)

Variable	Type	Description
w	GraphWin	*Placeholder for the game beginning*

Pseudo code rundown

```
def SetPieces(s):
while start has not been clicked
wait for user input through click

Close the window
Opens a temporary window to declare board setup has concluded
```

Click(s)

Has the user to perform a click, and depending on the location, calls another method to perform a functionality (ie if the "X" button is clicked, it will call the ExitGame method to close the game window)

Variables:

Variable	Type	Description
click	Object (holds value of mouse click	*Waits for mouse click and returns the location*
X, Y	Int	*Corresponds to location on the checkers window*
deleteTxt	Text Object	*Label for the delete button*

Pseudo code rundown

```
mouse click is performed
gets click coords and stores and variables X,Y
if the X button is clicked
   calls method ExitGame
elif the Save button is clicked
   calls method SaveSetupToFile
elif the state is in custom mode
```

```
        calls the method clickInCustom(X,Y)
    elif the state is in Play mode
        calls the method clickInPlay(X,Y)
```

CustomSetup(s) No longer in code

Enters custom setup mode, so the user can place the pieces manually

Variables:

Variable	**Type**	**Description**
WTxt	Text Object	*draws "W"*
BTxt	Text Object	*draws "B"*
KTxt	Text Object	*draws "K"*
deleteTxt	Text Object	*draws "Del"*

Pseudo code rundown

```
def CustomSetup(s):
    selects custom mode
    defaults the next colour to white
    defaults to not a king
    False if removing a piece

    draws and colours the buttons (W, B, K, and Del)

    while within custom mode
        perform clicks

    undraw and uncolour the buttons (W, B, K, Del)
```

StandardSetup(s)

Gives the Standard Setup button a functionality when clicked. Draws checker pieces on the tiles according to the default checker positions

Pseudo code rundown

```
def StandardSetup(s):
```

calls the ClearBoard method to clear the board
sets the variable ‘state’ to ‘CustomSetup’
places all the checker pieces in the default positions

ClearBoard(s)

Wipes the board clean of pieces, and resets some variables (state, pTurn) to default settings (CustomSetup and White respectively)

Pseudo code rundown

def ClearBoard(s):
creates the 2D list and initializes all 8x8 entries to an empty tile
recolours the tiles and resets the white tiles as buttons
resets state and pTurn to default settings
calls the method SetButtons

numWhite(s) No longer in code

returns the number of white checker pieces

Variable	Type	Description
c	int	acts as a counter

Psuedo code Rundown:

for each tile on the checker board
counts the number of white pieces and returns it as an int

numBlack(s) No longer in code

returns the number of black checker pieces

Variable	Type	Description
k	int	acts as a counter

for each tile on the checker board
counts the number of black pieces and returns it as an int

ClickedSquare(s, click)

determines which square of the game window has been clicked (returns bottom left corner). If the location is invalid, it does nothing.

Variables:

Variable	Type	Description
clickX	float (converted to int)	*x coordinates of the click location*
clickY	float (converted to int)	*y coordinates of the click location*

Pseudo code rundown

```
def ClickedSquare(s,click):
        rounds appropriately and returns the x and y coordinates of the clicked location
        if the clicked location is invalid:
            calls for a new click
```

SaveSetupToFile(s)

Saves the piece layout by writing the position and piece instances contained in the tiles variable to a text file called 'checkers.txt' in a condensed format.

Variables:

Variable	Type	Description
saveFile	file	*opens the file titled 'checkers.txt' for writing*
i_string, j_string	string (converted from int)	*assigned the value of the row and column of each piece*
has_been_changed (removed)	-	-

Pseudo code rundown

```
def SaveSetupToFile(s):
    (code dealing with the has_been_changed variable has been removed)
    opens the 'checkers.txt' file with the write permission enabled
    goes through loop and writes the row, column, pieceColour, and pieceRank for each
        verified piece on board to file
```

```
    saves whether it is black's or white's turn
    closes the file
```

LoadSetupFromFile(s)

Clears the board and adds the piece layout saved to the 'checkers.txt' file to the board

Variables:

Variable	Type	Description
loadFile	file	*opens the previously saved 'checkers.txt' file for reading*
piece_list	list of strings	*hold all the data associated with all the saved pieces*
tot_string	string	*holds just one saved piece*
x_var, y_var	int(converted from string)	*holds the row and column value for each saved piece*

Pseudo code rundown

```
def LoadSetupFromFile(s):
    opens the 'checkers.txt' file with the read permission enabled
    loads the data contained in the text file into the list of strings piece_list
    calls ClearsBoard method to clear the board(in case there are pieces placed)
    for i in range(length of the piece_list):
        if it is a piece: (this condition was removed due to less saved to file)
            if piece is white
               if piece is white King
                  draw white King piece
               else:
                  draw white pawn piece
            else: (piece is black)
               if piece is black King
                  draw black King piece
               else:
                  draw black pawn piece
    loads the last line of the text file which tells the program whose turn it was
    redraw the buttons so it shows whose turn it is before 'Start!' is clicked
    closes the file
```

clickInCustom(s,X,Y)

Determines where a click has occured in custom setup mode, and depending on the location, performs a specific action

Variables: There are several update variables, but no new variables are introduced

Pseudo Code rundown:

```
def clickInCustom(s,X,Y):
    if the Standard button is clicked
        calls the method StandardSetup
    elif the Start button is clicked
        checks to see if there are any pieces on the game board
        if not, displays an error message to the user
        else the state is set to Play, and the method SetButtons is called
    elif the Clear Board button is clicked
        calls the method ClearBoard
    elif the 1Player button is clicked
        sets the game mode to 1 player (is1P = True)
        calls the method SetButtons
    elif the 2Player button is clicked
        sets the game mode to 2 player (is1P = False)
        calls the method SetButtons
    elif the Load button is clicked
        calls the method LoadSetupFromFile
    elif the pTurn button is clicked during CustomSetup
        changes which colours goes first
        calls the method SetButtons
    elif the W button is clicked
        sets placeColour to White, and placeType to Place, allowing for placement of white
        coloured checker pieces
        calls the method SetButtons
    elif the B button is clicked
        sets placeColour to Black, and placeType to Place, allowing for placement of black
        coloured checker pieces
        calls the method SetButtons
    elif the K button is clicked
        switches the rank of the checkers pieces, allowing for placement of pieces of that rank
        calls the method SetButtons
    elif the Del button is clicked
        switches the placeType, allowing for checker pieces to be deleted
        calls the method SetButtons
    elif a tile is clicked in CustomSetup
        if clicked tile is White
            gives an error message
        elif the clicked tile would result in too many of colour being placed
```

```
        shows an error message
    else updates that tile in the array
        calls the method SetButtons
```

SetupBoard(s)

handles the setup/piece placement on the board, checking to see if the user is placing pieces in a custom board setup, and if they switch to place mode

Pseudo Code rundown:

```
def SetupBoard(s):
    while the state is in custom setup mode
        calls the method Click
    if the state is in play mode
        calls the method Play
```

Play(s)

handles the general play of the game, checking to see if the user remains in play mode, or if they switch to custom setup mode

Pseudo Code rundown:

```
def Play(s):
    while the state is in play mode
        calls the method Click
    if the user switches state to custom setup mode
        calls the method SetupBoard
```

ColourButton(s,colour,X,Y,width=1,height=1)

the purpose of this function is to create a rectangle with a given colour, size and location to represent the tiles of a checker board

Variables:

Variable	Type	Description
rect	instance (of rectangle object)	stores an instance of a rectangle to be drawn

Pseudo code rundown:

```
def ColourButton(s,colour,X,Y,width=1,height=1)
    creates and draws a rectangle given the specified data (from arguments)
```

TileColour(s,x,y)

method to determine what tiles should be what colours, returning the appropriate values

Pseudo code rundown:

```
def TileColour(s,x,y)
    sets every other square to red, and then every non red square to white
```

SetButtons(s)

A general purpose method that exists to call all the methods which draw buttons. This way instead of calling an entire list of methods, only this one needs to be called

Pseudo code rundown:

```
def SetButtons(s):
    if the state is in custom setup mode
        calls the method DrawStandard
        calls the method DrawStart
        calls the method DrawClear
        calls the method Draw1P
        calls the method Draw2P
        calls the method DrawLoad
        calls the method DrawSave
        calls the method DrawTurn
        calls the method DrawX
        calls the method DrawW
        calls the method DrawB
        calls the method DrawK
        calls the method DrawDel
        calls the method DrawScore (not a button)
    elif the state is in play mode
        calls the method DrawQuit
        calls the method DrawSave
        calls the method DrawTurn
        calls the method DrawX
        calls the method DrawScore (not a button)
```

DrawStandard(s)

draws a button so the user can use the standard setup of a checkers game

Pseudo code rundown:

```
def DrawStandard(s):
```

draws a button on the checkers window that displays the text ‘Standard Setup’

DrawCustom(s)

draws a button so the user can use customize the setup of the checker board

Pseudo code rundown:

```
def DrawCustom(s):
    draws a button on the checkers window that displays the text ‘Custom Setup’
```

DrawStart(s)

draws a button so the user can start playing the checkers game

Pseudo code rundown:

```
def DrawStart(s):
    draws a button on the checkers window that displays the text ‘Start!’
```

DrawClear(s)

draws a button so the user can clear the board of checker pieces

Pseudo code rundown:

```
def DrawClear(s):
    draws a button on the checkers window that displays the text ‘Clear Board’
```

Draw1P(s)

draws a button so the user can play a game versus the computer

Pseudo code rundown:

```
def Draw1P(s):
    draws a button on the checkers window that displays the text ‘1Player’
    changes colour to show if selected or not
```

Draw2P(s)

draws a button so the user can play a game versus another user

Pseudo code rundown:

```
def Draw2P(s):
    draws a button on the checkers window that displays the text ‘2Player’
    changes colour to show if selected or not
```

DrawLoad(s)

draws a button so the user can resume a saved game

Pseudo code rundown:

```
def DrawLoad(s):
    draws a button on the checkers window that displays the text ‘Load’
```

DrawSave(s)

draws a button so the user can save a game in progress to be continued later

Pseudo code rundown:

```
def DrawSave(s):
    draws a button on the checkers window that displays the text ‘Save’
```

DrawX(s)

draws a button so the user can close the window

Pseudo code rundown:

```
def DrawX(s):
    draws a button on the checkers window that displays the text ‘X’
```

DrawW(s)

draws a button so the user can place a white checker piece

Pseudo code rundown:

```
def DrawW(s):
    draws a button on the checkers window that displays the text ‘W’
    changes colour to show if selected or not
```

DrawB(s)

draws a button so the user can place a black checker piece

Pseudo code rundown:

```
def DrawB(s):
    draws a button on the checkers window that displays the text ‘B’
    changes colour to show if selected or not
```

DrawK(s)

draws a button so the user can change the rank of a piece to be placed

Pseudo code rundown:

```
def DrawK(s):
    draws a button on the checkers window that displays the text ‘K’
    changes colour to show if selected or not
```

DrawDel(s)

draws a button so the user can remove a checker piece from the board

Pseudo code rundown:

```
def DrawDel(s):
    draws a button on the checkers window that displays the text ‘Del’
    changes colour to show if selected or not
```

DrawQuit(s)

draws a button so the user can quit out of the game

Pseudo code rundown:

```
def DrawQuit(s):
    draws a button on the checkers window that displays the text ‘Quit’
```

DrawTurn(s)

draws a button so the user can change who goes first

Pseudo code rundown:

```
def DrawTurn(s):
    draws a button on the checkers window that displays the text ‘Turn’, along with the colou
    whos turn it is
```

DrawScore(s)

draws a box so the user can see how many pieces either side has lost

Pseudo code rundown:

```
def DrawScore(s):
    draws a box on the checkers window that displays text showing the number of pieces
    each colour has lost
```

clickInPlay(s,X,Y)

Gets the location of the mouse click in play mode, and depending on the location of the click determines what should happen next

Pseudo code rundown:

```
def clickInPlay(s,X,Y):
 if the Quit button is clicked
    sets the state to custom setup mode
    calls the method SetButtons
 elif a tile is clicked in play mode
    if the selected tile can move
       if the selected tile is selected again
          deselects that tile
       elif the player colour matches the selected piece colour and the piece has not been
captured and (the selected piece can capture a piece, or no pieces for that colour can capture
a piece):
          updates the tiles at the locations of the piece before and after the move
       elif the move is a valid move
          calls the method move to move the checker piece
          if the piece has not been captured, and cannot capture an opposing piece
             sets pieceCaptured to false
             deselects all tiles
             changes player turns
             calls the method SetButtons
       else:
          an error message displays notifying the user the attempted move is invalid
    else: #Select a Piece to move
       if player selects a piece from the opposing team
          an error message displays notifying them the piece is not theirs
       elif the player is able to capture a piece with a piece not selected
          an error message displays notifying them a move must be made to capture an
opposing piece
       else
          selects the tile
```

moveIsValid(s,x,y,X,Y)

determines whether or not an attempted move is legal/possible, and returns a boolean depending on the result

Pseudo code rundown:

```
def moveIsValid(s,x,y,X,Y)
   if the move can capture an opposing piece
      calls the method PieceCanCapturePiece
   elif the piece can jump to a valid location, return true
   elif the piece can travel to a valid location, without capturing a piece, return true
   else return false
```

move(s,x,y,X,Y)

Determines what moves can be made by the current player, such as if there is a move where a piece can be captured, or if there are no moves because someone has won the game

Pseudo code rundown:

```
def move(s,x,y,X,Y)
    updates the array with the coordinates of the move (given in parameters X, Y)

    if a piece makes it to the other side of the board, it updates it status to King
    updates the checker tiles with the new piece rank information, and the location

    if X-x == 2 or X-x == -2:
        if all the checker pieces of one colour are gone, displays a message of the winning
player
        updates the tiles array with the necessary information

        if the method PieceCanCapture returns true
            updates the tiles array with the necessary information

        updates which tile is currently selected
        sets pieceCaptured to true
    else
        sets so that no tiles are selected

        updates the tiles array with the necessary information
        sets pieceCaptured to false
```

PlayerCanCapture(s)

determines whether or not the player is able to capture an opposing checker piece

Pseudo code rundown:

```
def PlayerCanCapture(s)
    for each element on the board
        if the current piece belongs to the current player
            if the method PieceCanCapture returns true
                return true
    else return false
```

PieceCanCapture(s)

Determines whether or not the current piece is able to capture an opposing checker

piece

Pseudo code rundown:

```
def PieceCanCapture(s):
    for each element in range of the current piece
        if the method PieceCanCapturePiece returns true
            return true
    else return false
```

PieceCanCapturePiece(s)

Determines whether a piece on the checker board of the current player can capture a checker piece of the opposing player

Pseudo code rundown:

```
def PieceCanCapturePiece(s):
    if the method CanDoWalk returns true
        for each element in range of the current piece
            if the tile to be moved to is in bounds
                if there is not a piece on the tile to be moved to
                    return true
    else return false
```

PieceCanJumpTo(s,x,y,X,Y)

Determines whether a piece is able to make a jump to a tile by checking that that tile is in range, and there is an opposing piece to jump over and capture

Pseudo code rundown:

```
def PieceCanJumpTo(s,x,y,X,Y):
    for each element in range of the current piece
        if the tile to be moved to is within jumping range
            if PieceCanCapturePiece returns true
                return true
    else return false
```

CanDoWalk(s,x,y,X,Y,exception=False)

determines whether or not a piece is able to make a standard move (move diagonal without capturing a piece)

Pseudo code rundown:

```
def CanDoWalk(s,x,y,X,Y,exception=False)
    for each element in range of the current piece
```

```
        if the tile to be moved to is within range of the piece
            if the tile to be moved to is in bounds
                if the move is legal
                    if there is not already a piece on the tile
                        return true
    else return false
```

numColour(s,colour)

counts the number of pieces of a colour specified in the parameters, returning the number as an integer

Pseudo code rundown:

```
def numColour(s,colour)
    initiate a counter
        for each element on the board
            if the parameter for colour is white, and the piece on the current tile is white
                add 1 to the counter
            elif the parameter colour is black, and the piece on the current tile is black
                add 1 to the counter
    return the counter
```

opposite(s,opp)

returns the opposite setting of a variable, given in the parameter. This is so a general method can be called to switch from one setting to another, instead of changing each setting individually each time

Pseudo code rundown:

```
def opposite(s,opp)
    if opp is white, return black
    elif opp is black, return white
    elif opp is king, return pawn
    elif opp is pawn, return king
    elif opp is place, return delete
    elif opp is delete, return place
    else print an invalid message
```

ColourRect(win, Xmin, Ymin, Xmax, Ymax, colour) No longer in code, replaced with ColourButton

Creates a rectangle of a specific colour at the specified location

Variables:

Variable	Type	Description
rect	instance (of rectangle object)	stores an instance of a rectangle to be drawn

Pseudo code rundown:

```
def ColourRect(win,Xmin,Ymin,Xmax,Ymax,colour):
    creates and draws a rectangle given the specified data (from arguments)
```

ColourSquare(win, X, Y, colour) No longer in code

Creates a 1x1 square of a specific colour at the specified location

Variables:

Variable	Type	Description
sqr	instance (of rectangle object)	stores an instance of a rectangle to be drawn

Pseudo code rundown

```
def ColourSquare(win,X,Y,colour):
        creates and draws a square given the specified data (from arguments)
```

ExitGame(win)

Quits out of the game window

Pseudo code rundown

```
def ExitGame(win):
        closes the window and exits the program
```

CompTurn(s)

Simulates the computer's turn, creating a hierarchy of different moves that are 'better' than other moves, to determine which one is the best move the computer can make.

Variables:

Variable	Type	Description
moves	array	a list of all the possible moves that can be made during that turn
badMoves	array	contains all the 'bad' moves that can be made, to determine the best move the computer can make
lenM	int	integer storing the number of moves that can be made (updated regularly throughout method)
m	int	selects a random integer from 0 to lenM

Pseudo code rundown

```
def CompTurn(s)
    initialize variables
    for each element in moves
        if movesFromBack(move) returns true
            add move to badMoves
    calls method removeBadMoves()
    update lenM

    for each element in moves
        if a double jump move cannot be made
            add move to badMoves
    calls method removeBadMoves()
    update lenM

    for each element in move
        if a king cannot be made from a move
            add move to badMoves
     calls method removeBadMoves()
     update lenM

     for each element in move
         if a move cannot be made which takes an opposing king piece for free
             add move to badMoves
```

```
calls method removeBadMoves()
update lenM

for each element in move
    if a move cannot be made that trades a pawn for a king
        add move to badMoves
calls method removeBadMoves()
update lenM

for each element in move
    if a move cannot be made that trades pawn for pawn when computer has more pieces
        add move to badMoves
calls method removeBadMoves()
update lenM

for each element in move
    if a move is not safe (will allow opponent to take a piece)
        add move to badMoves
calls method removeBadMoves()

intializes m
using m, selects a remaining move from list move, and performs that move
```

hasMorePieces(s)

method which returns a boolean, outcome dependant on whether the current turn has more pieces than the opposing side

Pseudo code rundown

```
def hasMorePieces(s)
    return ((current turn pieces) > (opposing pieces))
```

isMoveSafe(s,move)

returns a boolean dependant on whether or not a proposed move will cause a piece to be taken immediately after the move

Variables:

Variable	Type	Description
X1	coords	contains the x coordinates of a given tile
Y1	coords	contains the y coordinates of

		a given tile

Pseudo code rundown

```
def isMoveSafe(s,move)
    gets the x and y coordinates of the tile where the move would end at
        for tiles in range of the ending tile
            if SpecialPCCP returns true
                return false
    return true
```

SpecialPCCP(s,piece2Colour,x,y,X,Y,initX,initY)

returns a boolean dependent on whether or not a piece can capture a piece, designed specifically to work with the method isMoveSafe

Variables:

Variable	Type	Description
X1	coord	contains the x coordinates of the tile 1 tile away from the current
Y1	coord	contains the y coordinates of the tile 1 tile away from the current
X2	coord	contains the x coordinates of the tile 2 tiles away from the current
Y2	coord	contains the y coordinates of the tile 2 tiles away from the current

Pseudo code rundown

```
def SpecialPCCP(s,piece2Colour,x,y,X,Y,initX,initY)
    initializes the variables
    if the coordinates to be moved to (x,y,X,Y) are valid (ie part of the board)
        if piece2Colour is the opposite colour of the tile to be jumped
            if CanDoWalk returns true
                for each tile within range
                    if the move is valid and the tile to be jumped to is empty
                        return true
```

```
    return false
```

removeBadMoves(s)

removes elements from the list of moves that can be made, which have been deemed as bad

Pseudo code rundown

```
def removeBadMoves(s)
    if the lists moves and badMoves are not the same
        for each element in badMoves
            remove that same element from moves
    updates badMoves to be an empty array
```

movesFromBack(s,move)

Returns a boolean, dependent on whether or not the move starts on the back row (row closest to the computer), using a 4 element array representing a move (parameter 'move')

Pseudo code rundown

```
def movesFromBack(s,move)
    if the computer is white and the piece is 0 moves from the back of the board, OR if the
        computer is black and the piece is 7 moves from the back of the board
        return true
    return false
```

moveEndsAt(s,move)

returns the location of where a piece ends it move at, using a 4 element array representing a move (parameter 'move')

Pseudo code rundown

```
def moveEndsAt(s,move)
    returns an array containing information of the tile that the piece ended on
```

movesAvailable(s)

determines all available moves and returns them as an array

Pseudo code rundown

```
def movesAvailable(s)
    initializes array 'moves'
    for each element on the board
        sets X1 and Y1 to the coordinates of the current element
            for each tile in range of the current coords
                if the tile is valid (ie on the game board)
                    if moveIsValid returns true
                        add the move to the array 'moves'
    return moves
```

validTileSelect(s,X,Y)

returns a boolean dependent on whether or not the tile attempting to be selected is valid

Pseudo code rundown

```
def validTileSelect(s,X,Y)
    if the tile clicked is in play
        if selectedTileAt is not an empty array
            if player turn colour matches the tile piece colour, and the piece has not been
                captured, and the piece selected can capture a piece or no pieces can be
                captured
                return true
            return false
        else
            if the player colour does not match the piece colour, and this be cannot capture a
                piece, but another one can
                return false
            return true
     return false
```

validTileMove(s,X,Y)

returns a boolean dependent on whether or not the tile attempting to be moved to is valid

Pseudo code rundown

```
def validTileMove(s,X,Y)
    if the tile clicked is in play
        if the array selectedTileAt is not empty
            if moveIsValid at the selected tile returns true
                return true
    return false
```

Class: Tile

Used for all the pieces in Board class, which included empty tiles (such as a null piece, such as an empty white tile)

Uses: None

init parameters:

win: GraphWin object
    contains the window from Board
X: int
    x *coordinates of the piece*
Y: int
    y *coordinates of the piece*
pieceColour: string renamed, previously 'colour'
    colour of the checker piece
pieceRank: string renamed, previously 'state;
    represents whether the piece is a king or a pawn or niether
isPiece: Boolean
    determines whether the piece is a 'real' piece
isSelected: Boolean
    determines whether the piece is currently selected or not (to have changes made)

Variables:

x: int
    x coordinate of the tile of the piece
y: int
    y coordinate of the tile of the piece
isPiece: Boolean
    represents whether the piece is a pawn or a king
isWhite: Boolean
    true if piece is white
isBlack: Boolean
    true if piece is black
isKing: Boolean
    true if piece is king
isPawn: Boolean
    true if piece is pawn
c: instance of point object
    is the centre point
circ: instance of circle object
    represents the circle of the checker piece
kingTxt: instance of text object

represents the "K" that appears on the checker piece

colour: string removed from code, replaced with pieceColour

represents the colour of the tile

Pseudo Code for init

```
if isWhite is true
    set pieceColour to white
elif isBlack is true
    set pieceColour to black
if isKing is true
    set pieceRank to king
elif isPawn is true
    set pieceRank to pawn

if isSelected is true
     outline the current piece in yellow
else remove the outline of the current piece

if isPiece is true (if the piece is 'real'
     call the method DrawPiece
```

ColourButton(s,colour,X,Y,width=1,height=1)

the purpose of this function is to create a rectangle with a given colour, size and location to represent the tiles of a checker board

Variables:

Variable	Type	Description
rect	instance (of rectangle object)	stores an instance of a rectangle to be drawn

Pseudo code rundown:

```
def ColourButton(s,colour,X,Y,width=1,height=1)
   creates and draws a rectangle given the specified data (from arguments)
```

TileColour(s,x,y)

method to determine what tiles should be what colours, returning the appropriate values

Pseudo code rundown:

```
def TileColour(s,x,y)
     sets every other square to red, and then every non red square to white
```

DrawPiece(s)

method to actually draw the checker pieces on the tiles, with appropriate colours and rank, depending on the values of each individual piece

Pseudo code rundown:

```
def DrawPiece(s)
    draw a circle
    if isWhite is true
        colour the inside white, with a black outline
    elif isBlack is true
        colour the inside black, with a white outline
    if isKing is true
        draw a 'K' on the piece
```

Explanation of Breakdown

We chose to have 2 classes, Board and Piece. Board contains all the relevant information of the game window, whereas Piece stores information about the individual game pieces. We decided to have these two classes, in this way we could keep different parts of the game separate from each other, making it easier to control each one. For Piece, since each one is its own object, it will be easier to move and modify each checker piece in the assignments to come.

Internal Review/Evaluation of our Design (As of Assignment #1)

Our design does exactly what has been asked of us to do, namely:
> Set up the Checkers Board
> Have a Standard Set up and a Custom Set up of the checkers tiles
> Have the 'Save' and 'Load' functionality working
> We have accomplished the task of enabling the user to start a game from original position/start a game from previously stored state.
> We have ensured that two users can play the game of checkers against one another.
> Allow a single user to play against the computer AI.
> Follow the rules as stated by the game '*American Checkers*'.
> A message is displayed once a game has been concluded, declaring the winner.
>We have ensured that no methods within the AI will requre use of any methods outside of the Checkers class that it is inside. However it does call many methods defined withing Checkers. This enforces high cohesion and very little coupling.
>Information hiding was handled by placing all the new methods in one group, all within the Checkers class.

Revision History Assignment #2

Functions in Class Checkers:

- Play(s) --> Handles general play of game
- ColourButton(s,colour,X,Y,width=1,height=1) --> function to create a rectangle with a given colour, size and location
- TileColour(s,x,y) --> Colours the tiles on the board
- SetButtons(s) --> Placeholder for the different buttons we will be drawing on the board.
- DrawStandard(s) --> Draws the 'Standard Setup' button
- DrawCuston(s) --> Draws the 'Custom Setup' button
- DrawStart(s) --> Draws the start button
- DrawClear(s) --> Makes a button to clear screen
- Darw1P(s) --> Draws a button for player 1
- Draw2P(s) --> A button to indicate if 2 players want to play
- DrawLoad(s) --> Load button

- DrawSave(s) --> Save button
- DrawX(s) --> Escape button
- DrawW(s) --> Draws a tile as 'white' in colour
- DrawB(s) --> Draws black tile
- DrawK(s) --> Draws a king
- DrawDel(s) --> Deletes a square for a tile that surrenders
- DrawQuit(s) --> Quit button for the game
- DRawTurn(s) --> Button to signify who's tuen it is
- DrawScore(s) --> Textbox to display the score

clickInCustom(s,X,Y) --> 'Click(s)'implementation of the custom setup. Click(s) has been broken down in Checkers_v16

- clickInPlay(s,X,Y) --> similar to clickInCutsom(s,X,Y) but implemented in-game.
- moveIsValid(s,x,y,X,Y) --> Checks if a particular move is valid or not.
- move(s,x,y,X,Y) --> Moves a piece
- PieceCanCapturePiece(s,x,y,X,Y) --> Enables a tile to capture and remove an opponent piece
- PieceCanJumpTo(s,x,y,X,Y) --> Enables a tile to jump to a new location
- CanDoWalk(s,x,y,X,Y,exception=False) --> Enables a 'king' to move front and back
- numColour(s,colour) --> counts the number of pieces of a given colour
- opposite(s,opp) --> Returns the opposite of a given parameter

Functions in Class Tile :

- ColourButton(s,colour,X,Y,width=1,height=1) --> function to create a rectangle with a given colour, size and location
- TileColour(s,x,y) --> Sets the colour of the tile while laying the board

Assignment #2 Requirements

Include choices that enable user to :

- Start game from original start position --> StandardSetup(s)
- Start a game from previously stored state --> LoadSetupFromFile(s)
- Make a move.
    - move piece to another square --> move(s,x,y,X,Y)
    - jump over opponent piece --> PieceCanJumpTo(s,x,y,X,Y)
        - remove opponent piece --> PieceCanCapturePiece(s,x,y,X,Y)
    - covert a piece to a 'king' --> isKing(s)
    - move kings in both direction --> CanDoWalk(s,x,y,X,Y,exception=False)
        - forward
        - backward
- Save a game to be resumed later --> SaveSetupToFile(s)

Revision History Assignment #3

New methods in Class Checkers:

- CompTurn(s) --> handles computer's turn
- hasMorePieces(s) --> returns true if whichever player's turn it is has more pieces, false otherwise
- isMoveSafe(s,move) --> determines whether a piece can be taken after it's move
- SpecialPCCP(s,piece2Colour,x,y,X,Y,initX,initY) -> allows PieceCanCapturePiece method to work with the isMoveSate method
- removeBadMoves(s) --> removes all the bad moves from the list of moves
- movesFromBack(s,move) --> determines whether the piece moved originated at the top/back row
- moveEndsAt(s,move) --> returns the location that the piece will end up at
- movesAvailable(s) --> calculates all the available valid moves
- DrawResign(s) --> changed from DrawQuit(s), doesn't change functionality
- ValidTileSelect(s,X,Y) --> determines the validity of the selected tile
- ValidTileMove(s,X,Y) --> determines the validity of the tile attempting to be moved to

Revised Methods in Class Checkers:

- __init__(s)
- Play(s)
- SetButtons(s)
- clickInCustom(s,X,Y)
- clickInPlay(s,X,Y)
- moveIsValid(s,x,y,X,Y)
- move(s,x,y,X,Y)
- PieceCanCapturePiece(s,x,y,X,Y)
- opposite(s,opp)