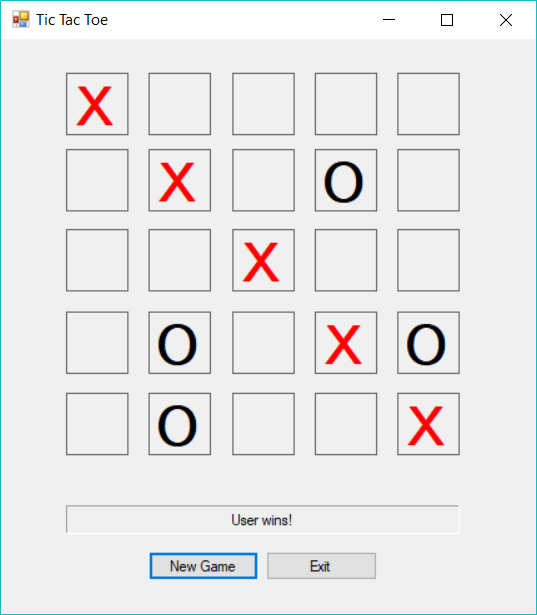
**Peer Evaluation for Lab 0 – TTT 5 by 5 And Memory**

|  |  |
| --- | --- |
| Your name: (Your lab is the one being evaluated) | Eva Schaeffer |
| Name(s) of peer evaluator(s) | Tim Allen |
| Date: |  |

Instructions  
You should have already completed Lab 0. After you and a peer have evaluated your work, you will submit this evaluation along with screen shots and source code indicated in Moodle. You may make corrections to your work as a result of the evaluation.

|  |  |
| --- | --- |
| ***TTT Problem*** | |
| Variable scope   * Does the application use any form level variables? What are they? Is it reasonable to use an instance variable for each of these? – No, there are no form level variables. Yes, it is reasonable to use instance variables. |  |
| Methods   * Application includes definitions for appropriate set of methods? yes * Does each pass parameters and return values appropriately? yes * Source code for methods that you wrote is included? yes |  |
| Event handlers   * Application includes definitions for appropriate set of event handlers? yes * Algorithm for each event handler is included? Yes * Source code for each event handler is included? Yes |  |
| Does the application function appropriately? Can you interact with the application in a way that causes it to “degrade” and not gracefully? Yes. |  |
| One thing that you learned from writing the application:  I learned that I should have coded over the break 😉 I also learned that I need to practice a lot more. | |
| Something that you’d like to continue working on:  I would like more practice writing algorithms and figuring out how all this works. I really struggled with just getting started, it was hard to do it without a jumping point. | |
| Time you spent completing the application:  Approximately 3 hours once I got passed the initial confusion and spent some time researching. | |

Screen shots and source code:



Methods:

//\* TODO: finish all of these that return true

private bool IsAnyRowWinner()

{

for (int row = 1; row < SIZE; row++)

if (IsRowWinner(row))

{

return true;

}

return false;

}

private bool IsColumnWinner(int col)

{

Label square = GetSquare(0, col);

string symbol = square.Text;

for (int row = 1; row < SIZE; row++)

{

square = GetSquare(row, col);

if (symbol == EMPTY || square.Text != symbol)

return false;

}

return true;

}

private bool IsAnyColumnWinner()

{

for (int col = 1; col < SIZE; col++)

{

if (IsColumnWinner(col))

{

return true;

}

}

return false;

}

private bool IsDiagonal1Winner()

{

Label square = GetSquare(0, 0);

string symbol = square.Text;

for (int diag = 1; diag < SIZE; diag ++)

{

square = GetSquare(diag, diag);

if (symbol == EMPTY || square.Text != symbol)

return false;

}

return true;

}

private bool IsDiagonal2Winner()

{

Label square = GetSquare(0, (SIZE - 1));

string symbol = square.Text;

for (int row = 1, col = SIZE - 2; row < SIZE; row++, col--)

{

square = GetSquare(row, col);

if (symbol == EMPTY || square.Text != symbol)

return false;

}

return true;

}

private bool IsAnyDiagonalWinner()

{

Label square = GetSquare(0, (SIZE - 1));

string symbol = square.Text;

for (int row = 1, col = SIZE-2; row < SIZE; row++, col--)

{

square = GetSquare(row, col);

if (symbol == EMPTY || square.Text != symbol)

return false;

}

return true;

}

private bool IsFull()

{

Label square = GetSquare(0, 0);

string symbol = square.Text;

for (int row = 0; row < SIZE; row++)

{

for (int col = 0; col < SIZE; col++)

{

square = GetSquare(row, col);

if (square.Text == EMPTY)

return false;

}

}

return true;

}

//\* TODO: finish these 2

private void HighlightRow(int row)

{

for (int col = 0; col < SIZE; col++)

{

Label square = GetSquare(row, col);

square.Enabled = true;

square.ForeColor = Color.Red;

}

}

private void HighlightDiagonal1()

{

for (int row = 0, col = 0; row < SIZE; row++, col++)

{

Label l = GetSquare(row, col);

l.Enabled = true;

l.ForeColor = Color.Red;

}

}

//\* TODO: finish this

private void HighlightWinner(string player, int winningDimension, int winningValue)

{

switch (winningDimension)

{

case ROW:

HighlightRow(winningValue);

resultLabel.Text = (player + " wins!");

break;

case COLUMN:

HighlightColumn(winningValue);

resultLabel.Text = (player + " wins!");

break;

case DIAGONAL:

HighlightDiagonal(winningValue);

resultLabel.Text = (player + " wins!");

break;

}

}

//\* TODO: finish these 2

private void ResetSquares()

{

for (int row = 0, col = 0; row < SIZE; row++, col++)

{

Label l = GetSquare(row, col);

l.Text = EMPTY;

l.Enabled = true;

}

}

private void MakeComputerMove()

{

Random randomNumberGenerator = new Random();

Label square;

do

{

int row = randomNumberGenerator.Next(0, SIZE);

int col = randomNumberGenerator.Next(0, SIZE);

square = GetSquare(row, col);

} while (square.Text != EMPTY);

square.Text = COMPUTER\_SYMBOL;

DisableSquare(square);

}

Event Handlers

//\* TODO: finish the event handlers

private void label\_Click(object sender, EventArgs e)

{

int winningDimension = NONE;

int winningValue = NONE;

Label clickedLabel = (Label)sender;

clickedLabel.Text = USER\_SYMBOL;

DisableSquare(clickedLabel);

if (IsWinner(out winningDimension, out winningValue) == false)

{

if (!IsFull())

{

MakeComputerMove();

if (IsWinner(out winningDimension, out winningValue))

{

HighlightWinner("Computer", winningDimension, winningValue);

DisableAllSquares();

}

}

else

{

MessageBox.Show("It's a Tie!");

}

}

else

{

HighlightWinner("User", winningDimension, winningValue);

DisableAllSquares();

}

}

private void newGameButton\_Click(object sender, EventArgs e)

{

Application.Restart();

}

private void exitButton\_Click(object sender, EventArgs e)

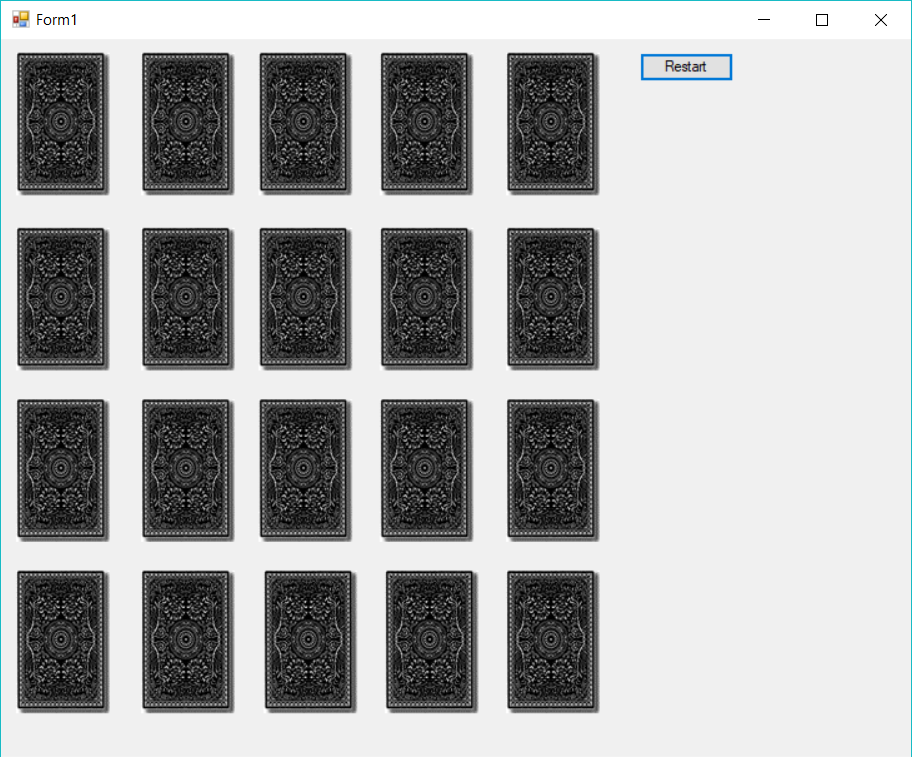
{

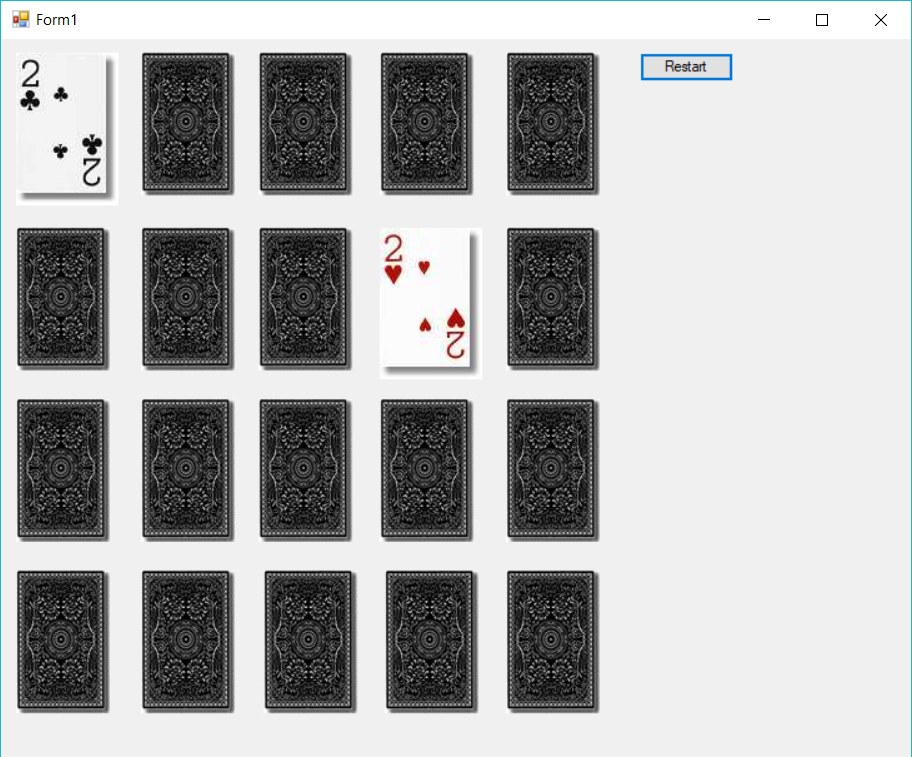
this.Close();

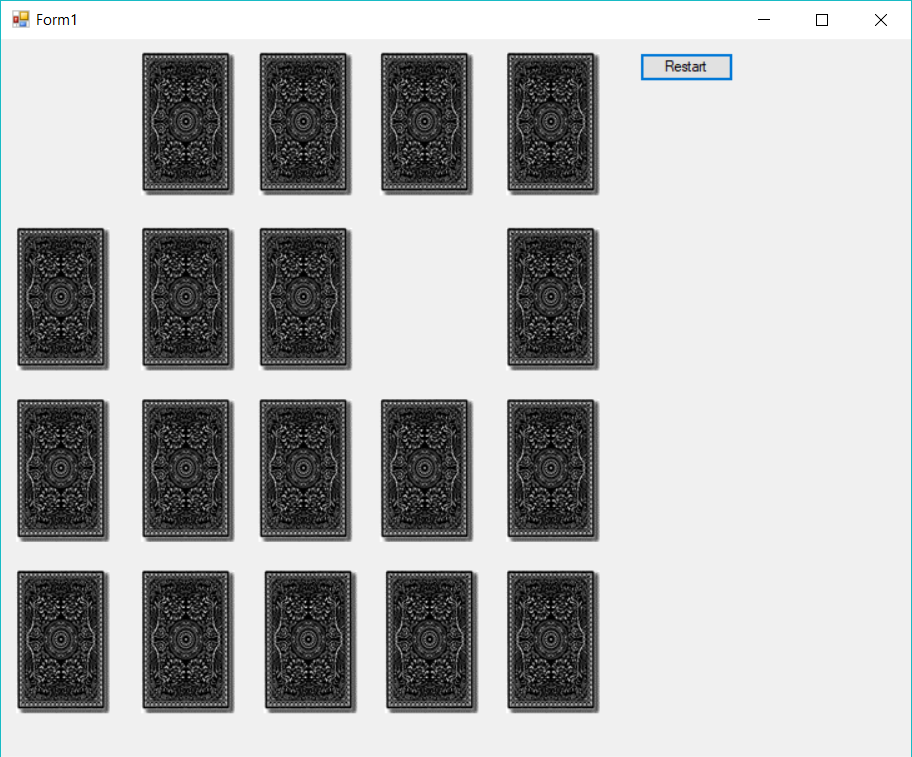
}

|  |  |
| --- | --- |
| ***Concentration Problem*** | |
| Variable scope   * Does the application use any form level variables? What are they? Yes, Timer. |  |
| Methods   * Application includes definitions for appropriate set of methods? yes * Does each pass parameters and return values appropriately? yes * Source code for methods that you wrote is included? yes |  |
| Event handlers   * Application includes definitions for appropriate set of event handlers? yes * ACE chart is included? Included in peer eval. * Algorithm for each event handler is included? yes * Source code for each event handler is included? Yes |  |
| Does the application function appropriately? Can you interact with the application in a way that causes it to “degrade” and not gracefully? Yes. |  |
| One thing that you learned from writing the application:  I feel pretty comfortable writing loops at this point. | |
| Something that you’d like to continue working on:  I’d like to work on writing more methods. | |
| Time you spent completing the application:  Approximately 2 ½ hours | |

Screen shots and source code:







public boardForm()

{

InitializeComponent();

timer.Interval = 1500;

timer.Tick += new EventHandler(flipTimer\_Tick);

}

// TODO: students should write this one

private bool IsMatch(int index1, int index2)

{

if (GetCardValue(index1) == GetCardValue(index2))

return true;

else

return false;

}

// TODO: students should write this one

private void ShuffleCards()

{

Random generator = new Random();

for (int cardNum = 1; cardNum <= 20; cardNum++)

{

int randomNum = generator.Next(1, 21);

string temp = GetCardFilename(cardNum);

string tempRandom = GetCardFilename(randomNum);

SetCardFilename(cardNum, tempRandom);

SetCardFilename(randomNum, temp);

}

}

private void LoadAllCards()

{

for (int i = 1; i <= 20; i++)

{

LoadCard(i);

}

}

// TODO: students should write all of these

// shows (loads) the backs of all of the cards

private void LoadAllCardBacks()

{

// loop to run load card back for each of the cards

for (int i = 1; i <= 20; i++)

{

LoadCardBack(i);

}

}

// Hides a picture box

private void HideCard(int i)

{

GetCard(i).Visible = false;

}

private void HideAllCards()

{

for (int i = 1; i <= 20; i++)

{

HideCard(i);

}

}

// shows a picture box

private void ShowCard(int i)

{

PictureBox card = GetCard(i);

card.Visible = true;

}

private void ShowAllCards()

{

for (int i = 1; i <= 20; i++)

{

ShowCard(i);

}

}

// disables a picture box

private void DisableCard(int i)

{

PictureBox card = GetCard(i);

card.Enabled = false;

}

private void DisableAllCards()

{

for (int i = 1; i <= 20; i++)

{

DisableCard(i);

}

}

private void EnableCard(int i)

{

PictureBox card = GetCard(i);

card.Enabled = true;

}

private void EnableAllCards()

{

for (int i = 1; i <= 20; i++)

{

EnableCard(i);

}

}

private void EnableAllVisibleCards()

{

for (int i = 1; i <= 20; i++)

{

PictureBox card = GetCard(i);

if (GetCard(i).Visible)

EnableCard(i);

}

}

#region EventHandlers

private void boardForm\_Load(object sender, EventArgs e)

{

/\*

\* Fill the picture boxes with filenames

\* Shuffle the cards

\* Load all of the card backs -

\* While you're testing you might want to load all of card faces

\* to make sure that the cards are loaded successfully and that

\* they're shuffled. If you get all 2s, something is wrong.

\*/

FillCardFilenames();

ShuffleCards();

LoadAllCards();

//int i = 1;

//LoadCardBack(i);

LoadAllCardBacks();

}

private void card\_Click(object sender, EventArgs e)

{

PictureBox card = (PictureBox)sender;

int cardNumber = int.Parse(card.Name.Substring(4));

/\*

\* if the first card isn't picked yet

\* save the first card index

\* load the card

\* disable the card

\* else (the user just picked the second card)

\* save the second card index

\* load the card

\* disable all of the cards

\* start the flip timer

\* end if

\*/

if (firstCardNumber == NOT\_PICKED\_YET)

{

firstCardNumber = cardNumber;

LoadCard(cardNumber);

DisableCard(cardNumber);

}

else if (secondCardNumber == NOT\_PICKED\_YET)

{

secondCardNumber = cardNumber;

LoadCard(cardNumber);

DisableAllCards();

timer.Start();

}

}

private void flipTimer\_Tick(object sender, EventArgs e)

{

/\*

\* stop the flip timer

\* if the first card and second card are a match

\* increment the number of matches

\* hide the first card

\* hide the second card

\* reset the first card number

\* reset the second card number

\* if the number of matches is 10

\* show a message box

\* else

\* enable all of the cards left on the board

\* end if

\* else

\* flip the first card back over

\* flip the second card back over

\* reset the first card number

\* reset the second card number

\* enable all of the cards left on the board

\* end if

\*/

timer.Stop();

if (IsMatch(firstCardNumber, secondCardNumber))

{

matches++;

HideCard(firstCardNumber);

HideCard(secondCardNumber);

DisableCard(firstCardNumber);

DisableCard(secondCardNumber);

firstCardNumber = NOT\_PICKED\_YET;

secondCardNumber = NOT\_PICKED\_YET;

if (matches == 10)

MessageBox.Show("You win!");

else

EnableAllCards();

}

else

{

LoadCardBack(firstCardNumber);

LoadCardBack(secondCardNumber);

firstCardNumber = NOT\_PICKED\_YET;

secondCardNumber = NOT\_PICKED\_YET;

EnableAllVisibleCards();

}

}

#endregion

private void restartButton\_Click(object sender, EventArgs e)

{

HideAllCards();

ShuffleCards();

LoadAllCardBacks();

ShowAllCards();

EnableAllCards();

}

}

}

**Lab 0-ACE Chart**

|  |  |  |
| --- | --- | --- |
| A | C | E |
|  | Exit button | click |
| Fill with the list of cards (so each card has a match)  Shuffle cards  Deal out 20 cards  Show cards face down | Form | Load |
| Flip the card  if it’s the first card store the card number in firstCardNumber  load the card, then disable it  if it’s the second card, store the card number in secondCardNumber  load the card, then disable the card.  compare the two card numbers | Card | click |
| Check for matches . . . | Timer | timer |

|  |  |
| --- | --- |
| ***Programming style for all programs*** | |
| Is proper indentation used? Is each event handler indented properly? | Yes |
| Are comments used appropriately? | Yes |
| Do variable names use camel case? (camelCase for example) | Yes |
| Do method names use Title Case (or Pascal Case?) | Yes |

General comments and notes from the evaluator:

Add more comments to my file.

One thing that you learned from completing the evaluation: