**Memory Game**

In this game, sixteen squares are used to hide eight different pairs of pictures. The player chooses two squares on the board and the pictures behind them are revealed. If the pictures match, those squares are removed from the board. If there is no match, the pictures are recovered and the player tries again. The play continues until all eight pairs are matched

.

# C:\Users\e20720\AppData\Local\Temp\SNAGHTML100d627.PNGPlace Controls on Form

Start a new project and place the memoryBackground picture in the form’s BACKGROUND image property and set the background layout property to centre.

Place sixteen picture box controls on the form and name them picHidden0 to picHidden15. Place nine smaller picture box controls on the form and change their visible property to false. Name one picBack and the remaining eight, picChoice0 to picChoice7. Change ALL the Picturebox BackColor properties to Transparent. Place a timer in your project for the flip delay and insert the appropriate images like below.

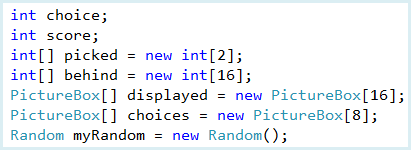


**Write Event methods**

When the game starts, pick one box, then another. The game stops when all matching picture pairs have been found. A delay is used to display the pictures for one second before deciding whether or not there is a match.

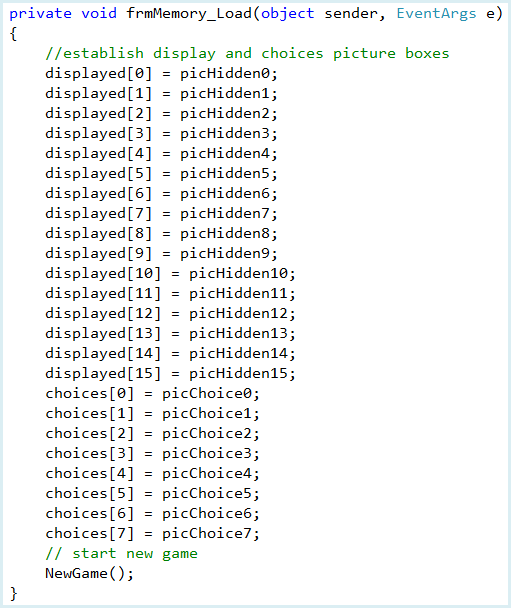
To make our programming job easier, we will establish a control array (**displayed**) for the 16 picture box controls used to display the photos on the form. We will also use a control array (**choices**) to hold the 8 photos that are used in the game. These arrays are defined in the form level declarations area of the code window.

Add this code to the **general** **declarations** area:

****

The assignments are made in the **frmMemory Load** event method.

The **Form1\_Load** event establishes the pictureboxes and runs the NewGame() method.



**Code Design – Integer Shuffling**

We will be using code for a **shuffle** process, so called because it can be used to shuffle a deck of cards. Here, we’ll just be shuffling 16 “cards,” though the code can be generalized to any number of “cards.”

Randomly shuffle the integers from 0 to 15. After the shuffle, for any value greater than 7, we will subtract 8. This will give us a list of 16 random integers. This list will have two of each index from 0 to 7, telling us which photo is behind which picture box control.

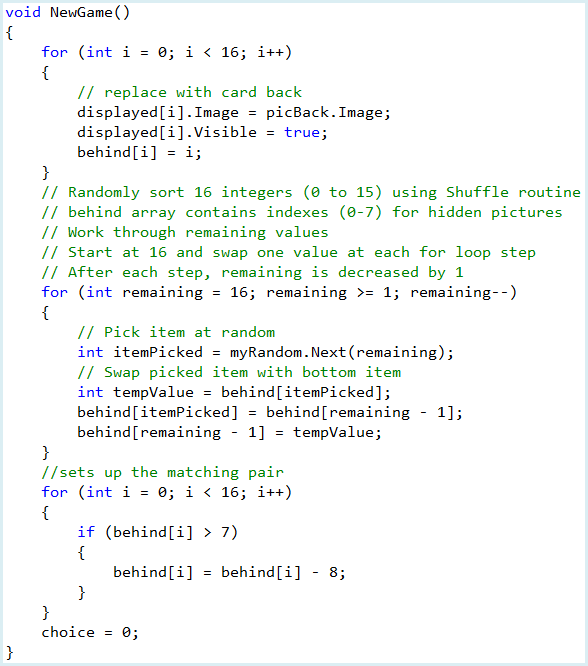
We will perform what is called a “one card shuffle.” In a one card shuffle, you pull a single card (at random) out of the deck and lay it aside on a pile. Repeat this 52 times and the cards are shuffled. Try it! I think you see this idea is simple, but doing a one card shuffle with a real deck of cards would be awfully time consuming.

We’ll use the idea of a one card shuffle here, with a slight twist. Rather than lay the selected card on a pile, we will swap it with the bottom card in the stack of cards remaining to be shuffled. This takes the selected card out of the deck and replaces it with the remaining bottom card. The result is the same as if we lay it aside.

Here’s how the shuffle works with n numbers:

* Start with a list of **n** consecutive integers.
* Randomly pick one item from the list. Swap that item with the last item.
* You now have one fewer items in the list to be sorted (called the remaining list), or **n** is now **n - 1**.
* Randomly pick one item from the remaining list. Swap it with the item on the bottom of the remaining list. Again, your remaining list now has one fewer items.
* Repeatedly remove one item from the remaining list and swap it with the item on the bottom of the remaining list until you have run out of items.
* When done, the list will have been replaced with the original list in random order.

The **NewGame** method sets up the hidden pictures:

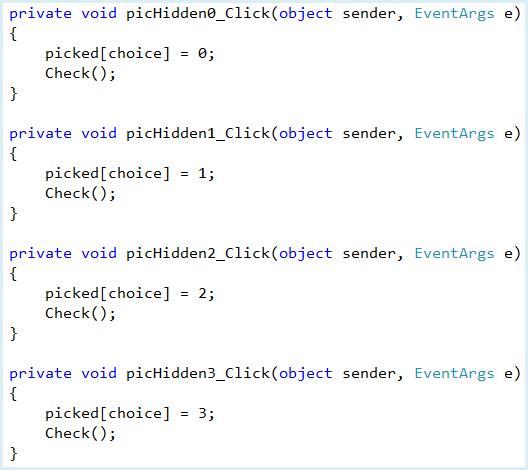


**Q1:** What is the role of the *behind* array?

**Q2:** Explain, in your own words, how the deck shuffling works.

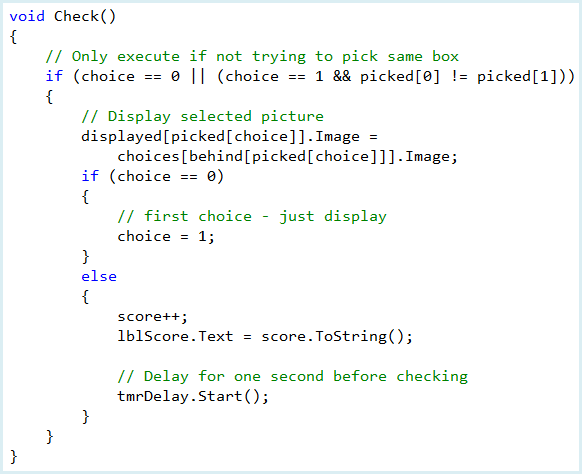
**Q3:** In our final “for loop”, why do we subtract 8 from any behind[i] value greater than 7?

The **Click** event methods for the 16 picture boxes:

****

You must continue with the remaining picture boxes.

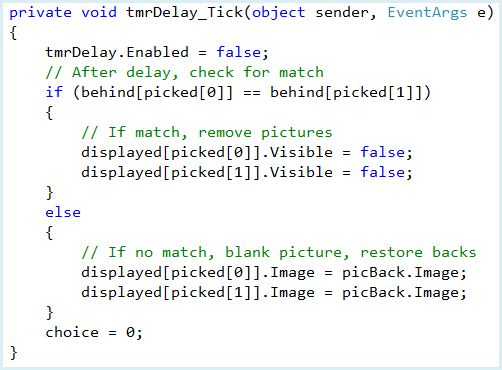
The **Check** method that displays the choices for a match and updates the score:



**Q4:** Describe what the condition is checking in the first “if statement”, in your own words.

**Q5:** *choices[behind[picked[choice]]].Image* displays the image that is selected. Explain how this works.

The **tmrDelay\_Tick** method that checks for matches after a delay:

****

**Run the Project**

Save your work. Run the project. Sixteen boxes appear. Click on one and view the picture. Click on another. If there is a match, the two pictures are removed (after a delay). If there is no match, the boxes are restored (also after a delay).

**Level 4+ Achievements**

* Have a way to replay / start a new game.
* Record high (low) scores
* The game works fine if the user waits for the timer to flip the cards back over before selecting new cards. However, the user can still click on tiles while the timer is running thus breaking the game. Fix this…