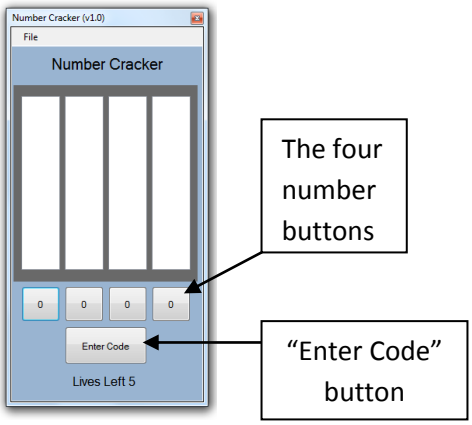
Number Cracker Creation Tutorial

Introduction

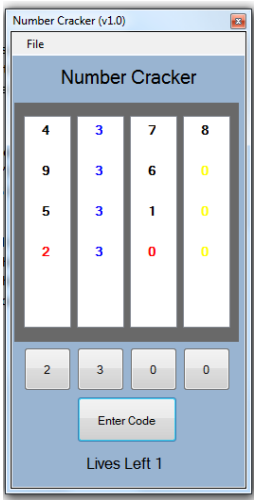
In this tutorial you will be creating a C# program written in visual studio 2008. The program is a very basic game/puzzle which requires the user to enter different numbers to try and work out the randomly generated number.

Fig 1.0



Puzzle Rules

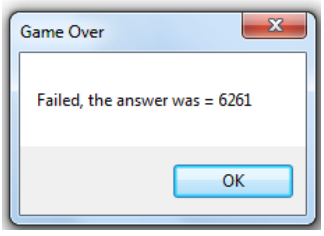
When the puzzle begins it will generate a random four digit number that is hidden from the user, it is the users job to work out that number. The user does this by picking a four digit number and entering it on the assigned buttons. When the four numbers are chosen the user will press the "Enter Code" button to submit the number and view the results on the list above. The user has 5 lives which are taken up each time the "Enter Code" button is pressed.

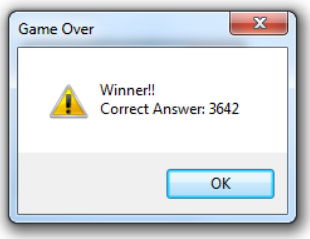


Guessing a 4 digit number in 5 attempts is rather difficult and not really a puzzle, so when the number is submitted the user will get feedback on the code they entered. Each button has its own column list that stores all the previously submitted numbers. These numbers will be colour coded depending on whether they are the correct numbers in the right place or not.

- Black – Number is not in the random hidden code.
- Blue - Number is correct and entered in the correct column.
- Red – Number is correct but in the wrong column.
- Yellow – Number is correct and in right place, but appears more than once.

With this information the user should be able to work out the hidden number relatively quickly. Once the correct number is entered, the game will end with a pop up message. The user then presses the "ok" button to reset the game with a new random number. Buttons reset to default 0 and the previous guesses are cleared.





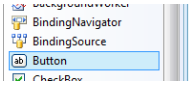
Creating the puzzle

First thing to do of course is to open Visual studio 2008 and create a new project in C#. This new project should have a suitable name, “Number Cracker” or something similar would be sensible. Next add a label for the title and resize the form to a similar shape as the image shows (Fig 1.0). You should always rename any added components to have a more related name, this will make coding it much easier to remember.

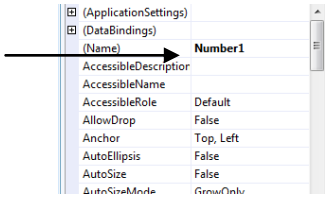
Right click the label and choose properties, the properties box appears bottom right corner on the screen, search for name and alter it to “titlelabel” or something similar.

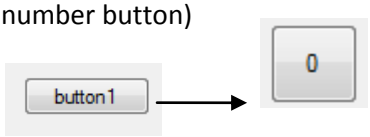
Creating the number buttons

Next create a button on the form using the tool bar

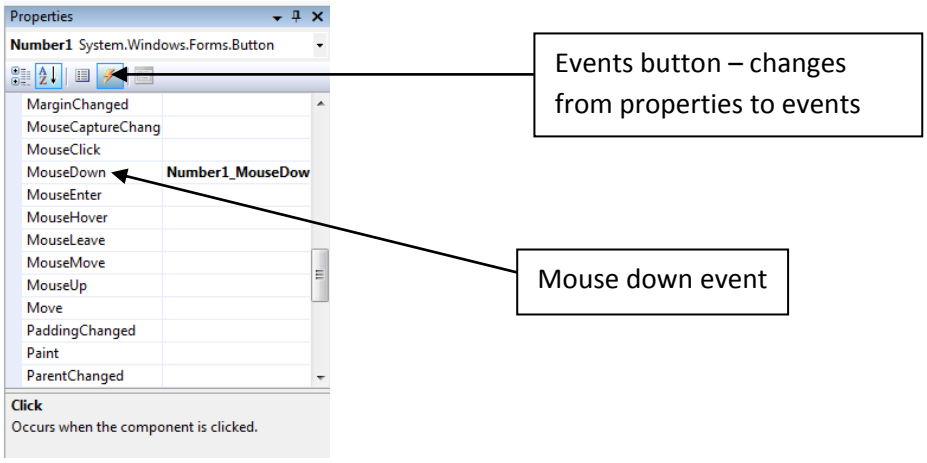


Using the buttons properties, change the text to read “0” and give the button a suitable name like Number1 (as the button will be the 1st number button)





From here you have two choices, you can either double click the button to open its code window, or you can add a “mouse down” event in the buttons events to create a button that can change depending on whether it’s a right mouse click or a left mouse click.



The next step is to create some code to make this button do something, the outcome should be every time the button is pressed, the number on the button should increase by 1, when the number reaches 10, it is reset back to 0. First thing needed is a global variable integer which will be the number displayed on the button, below is an image of an variable example.

```
//variable nu
int Num1 = 0
```

```
//if pressed increment number by 1
Num1++;
//if number reaches 10, reset it to 0
if (Num1 >= 10)
{
  Num1 = 0;
}
//change the string on the button to read this new number
Number1.Text = Num1.ToString();
```

Next part of the code should be located inside the button function. The recently added variable needs to be incremented each time it's pressed and updates the buttons string with the new number. We also need a way of checking what the number is and resetting it if it reaches the number 10.

If you used "the mouse" down event on the button then you can expand the code a little to check which mouse button has been pressed. A useful idea for this on this project is to make a right mouse button click decrement the button number. The code example to the right shows how this can be done. At this point you should be able to compile your work and test the button.

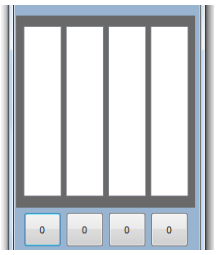
```
//Button 1, changes the number on it up/down depending on mouse click
private void Number1_MouseDown(object sender, MouseEventArgs e)
{
    if(e.Button == MouseButtons.Left)
    {
        Num1++;
        if (Num1 >= 10)
        {
            Num1 = 0;
        }
        Number1.Text = Num1.ToString();
    }
    else if (e.Button == MouseButtons.Right)
    {
        Num1--;
        if (Num1 < 0)
        {
            Num1 = 9;
        }
        Number1.Text = Num1.ToString();
    }
}
```

Creating the List

The list is not actually going to contain any code, it is simply there to copy the users guess results into, to record past number locations. So for this part simply add a list box above your button as the picture shows. The listbox should have its own unique name for later use so change its name in properties to match the button. (ListBox1 or SavedNumber1 for example)

Expanding

Now we need to add more buttons, create 3 more buttons, follow the last two stages so you end up with four buttons and four list boxes, each list box and button should have its own unique name.



Creating the random generated number

The randomly generated number needs to be created at game start and whenever the game is reset. The best way to do this is to create a new function that can be called from program launch and when the user finishes the current puzzle. Go into the code section of the form and look for "public Form1()". Inside this function create a new line to call the new function "NewGame()" and create this new function "public void NewGame()"

Next we need to create 4 global integer variables, these will be the hidden 4 digits in the code.

```
//Random numbers that are generated by code
int Rand1 = 0, Rand2 = 0, Rand3 = 0, Rand4 = 0;
```

Code below shows an example of the new game function and how to define a random number.

```
//Initialize Set up
public Form1()
{
    InitializeComponent();
    NewGame();
}

//Re-set everything to defaults for starting a new game
//or clearing exsisting game
public void NewGame()
{
    Random random1 = new Random();
    Rand1 = random1.Next(0, 9);
    Rand2 = random1.Next(0, 9);
    Rand3 = random1.Next(0, 9);
    Rand4 = random1.Next(0, 9);
```

Calls the "NewGame"

Creates the Random number generator.

Assigns a random number between 0 and 9 to the four different variables.

Creating the "Enter Code Button"

Now this is the button that will contain most of the code. The image below demonstrates the first part of the code. It takes the first button we created at the start and adds the buttons number to the list above it. It then compares it to the 1st random generated number to see if it matches, if it does it then changes that line on the list to blue. Then it checks to see if the button matches any of the other 3 random numbers. If so it changes the colour of that line to red. Finally if the button matches the random number then it checks to see if the random number matches any of the other random numbers. If it does then it will change it to yellow.

```
//Button "Enter Code", compares the 4 button numbers to the 4 random numbers, adds them to the correct
//list box for each and changes the colours when needed.
private void button1_Click(object sender, EventArgs e)
{
  // Add answer from button 1 to list 1
  SavedList1.Items.Add(Number1.Text);
  //1st List Colour Changes
  //compare button1 number with random number 1, if the same, make that line of text blue
  if (Number1.Text == Rand1.ToString())
  {
                  //onlinenumber is incremented each guess later on to keep it on right line,
    SavedList1.Items[OnLineNumber].ForeColor = System.Drawing.Color.Blue;
  }

  if (Number1.Text == Rand2.ToString() || Number1.Text == Rand3.ToString() || Number1.Text == Rand4.ToString())
  {
    // compare button 1 number to the other random numbers to see if it appears, if so make it red
    SavedList1.Items[OnLineNumber].ForeColor = System.Drawing.Color.Red;
  }
  if (Number1.Text == Rand1.ToString())
  {
    //finally check to see if the rand numner matches another rand number, make it yellow but only if button 1 number is
    //correct
    if (Rand1 == Rand2 || Rand1 == Rand3 || Rand1 == Rand4)
    {
      SavedList1.Items[OnLineNumber].ForeColor = System.Drawing.Color.Yellow;
    }
  }
```

Before you can compile the code a global integer is needed. This is needed to change the line number on the list box. Each time the "enter code" button is pressed it needs to be incremented by one.

```
//used to check to see what line the colours should be changed on
OnLineNumber++;
```

At this point you should be able to compile and test 1 button/list box. If it works repeat this process using the other 3 buttons. Compile again at the end to test and confirm it works.

Creating the life system and end game sequence

Still inside the enter code button function, extra code is needed to check if the answer is correct. At the end of the function add an if statement that checks if the answer is correct then to compare button 1 string to the 1st random number, then button 2 to 2nd random number etc, if all correct we can end the game with a message box.

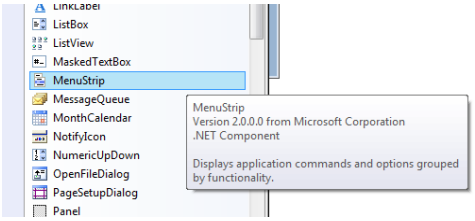
If the statement is incorrect then use an else statement to reduce the life counter and check to see if the user has run out of lives (The lives counter should be a global variable). If the lives reach 0 then the game is over, the program should create a message box saying "game over". This example also updates a textbox (labelled LivesLeftCounter) with the lives count in, so the user can see lives remaining. In both statements make sure the new game function is called after the message box to reset the game.

```
//Check to see if all 4 numbers are correct?
if (Number1.Text == Rand1.ToString() && Number2.Text == Rand2.ToString() && Number3.Text == Rand3.ToString()
                                    && Number4.Text == Rand4.ToString() )
{
  //if correct then we have a winner!!! Game over, create a message box!
  var result = MessageBox.Show("Winner!!" + "\n" + "Correct Answer: " + Rand1 + Rand2 + Rand3 + Rand4,
  "Game Over",  MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
  // Game over, time to restart when message box is closed.
  if (result == DialogResult.OK)
  {
    //calls the function that resets everything to defaults
    NewGame();
  }
}
else
{
  //if the answer was not correct! We minus 1 life
  Lives--;
  //update the life counter text
  LivesLeftCounter.Text = "Lives Left " + Lives;
  //check lives left to see if they can have another go?
  if (Lives == 0)
  {
    //game over, no lives, display message box and reset when box closed
    MessageBox.Show("Failed, the answer was = " + Rand1 + Rand2 + Rand3 + Rand4 + "", "Game Over");
    NewGame();
  }
}
}
```

Resetting the game

When we created the random numbers we created a new function called newgame, this function will be the perfect place to set everything to its default as it's also called at the start. All we have to do is add settings to clear the contents or to set numbers back to their defaults. Locate your newgame function and add your variables to set everything up.

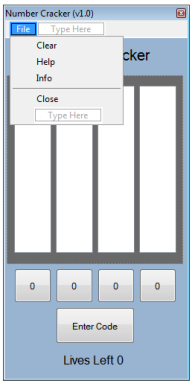
```
SavedList1.Items.Clear();
SavedList2.Items.Clear();
SavedList3.Items.Clear();
SavedList4.Items.Clear();
Num1 = Zero;
Num2 = Zero;
Num3 = Zero;
Num4 = Zero;
Number1.Text = Zero.ToString();
Number2.Text = Zero.ToString();
Number3.Text = Zero.ToString();
Number4.Text = Zero.ToString();
Rand1 = 0;
Rand2 = 0;
Rand3 = 0;
Rand4 = 0;
OnLineNumber = 0;
Lives = 5;
LivesLeftCounter.Text = "Lives Left " + Lives;
```



Creating the menu

Time to add some finishing touches. Add a Title bar to the program via the toolbar and include the headings "help", "clear" and "close".

The menu should look something like this. The code Application.Exit(); will close the program, and calling the NewGame(); function will reset the game, these need to be added to the correct menu options. .



The final program should look similar to this. Feel free to change colours and add different background styles. The yellow text can be hard to see on the white background. However be aware of the more appropriate colours to be used in case your user is colour blind.