Autoplay Media Studio

Tutorial for making menu's the Skeraxe way.

To begin:

I'm going to use the games Bloodrayne and Bloodrayne2 for this tutorial.
The reasons I'm using these games is that they are easy to convert and I was playing it at the time this was written.

This will give an insight of the codes I'd use if I'd program my menu's in AMS (Autoplay Media Studio). You can use these codes for almost every game out there provided the conversion is properly done. This means that the menu could simply link to a setup program to install the game.

Contents of the tutorial:

Finding your resources:

Background:

Usually I simply Google for some nice wallpapers. I like to use 640x480 sized pictures, if I can't find any good pictures in that size, I take a larger size and resize it to fit my needs.

I found this background:

Logos:

Same as the background, simply Google for some nice game logos.

I found these logos:

Fonts:

A good font resource site is http://www.dafont.com, I usually search there or simply Google again. You don't always have to have a game font, sometimes the windows standard fonts will also look great.

If you don't know how to install a new font, you can find the answer either on the web or in the windows help manual.

Icon:

I always like to use the game's icon for the menu. Most of the games have the icon as a separate file either on the CD/DVD or when it's installed in the game's folder.
Otherwise I use a program to extract the icon from the game's executable file.

Registry Settings:

The registry holds a lot of information about everything your computer is and has. All the games store a lot of information here like where it's installed, what version it's at and uninstall settings.

For this menu we need to know where the game is installed but some users might have a different location then others. So the menu needs to look inside the registry to know where that user has installed the game.

Games always store the information at the same location inside the registry so we'll need to find where it is. Install the game and use either of these programs.

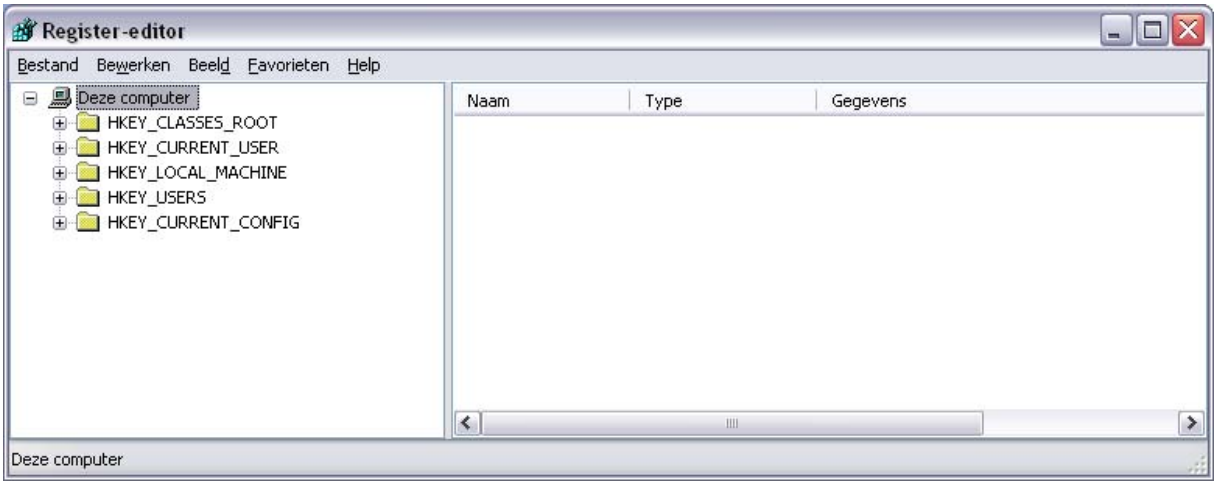
- **Regedit**: this program can also delete things from the registry so watch out!
- **Registry Searcher**: Available at http://www.cd-2-dvd.com in the tools section.

Both programs will be covert in this tutorial.

Regedit:

This program comes with windows. It can edit the entire registry so watch out, you don't want to accidentally delete any settings!

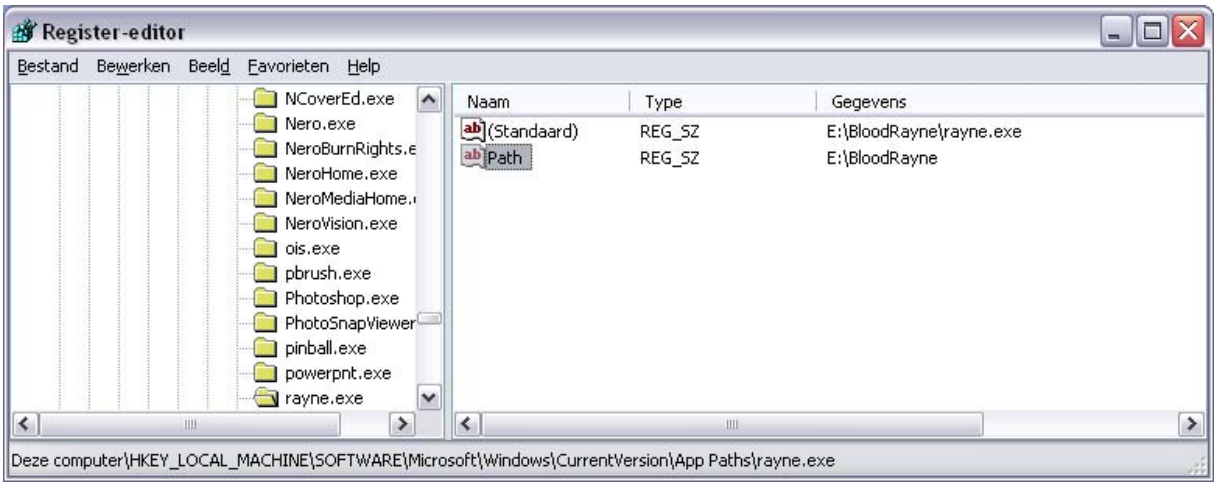
To start it go to the Start menu and select RUN. Type without the quotes "Regedit" and hit [Enter].



Regedit screenshot

It should look similar to this although I have the Dutch version so some menu's don't look the same. Either way, the registry folders do have the same names. In all language versions of windows, the registry's folder structure is the same.

Now I go to Edit ("Bewerken" for me) and find. Then I type the path where I've installed Bloodrayne. I've installed bloodrayne at "E:\Bloodrayne" and I've found this registry key:



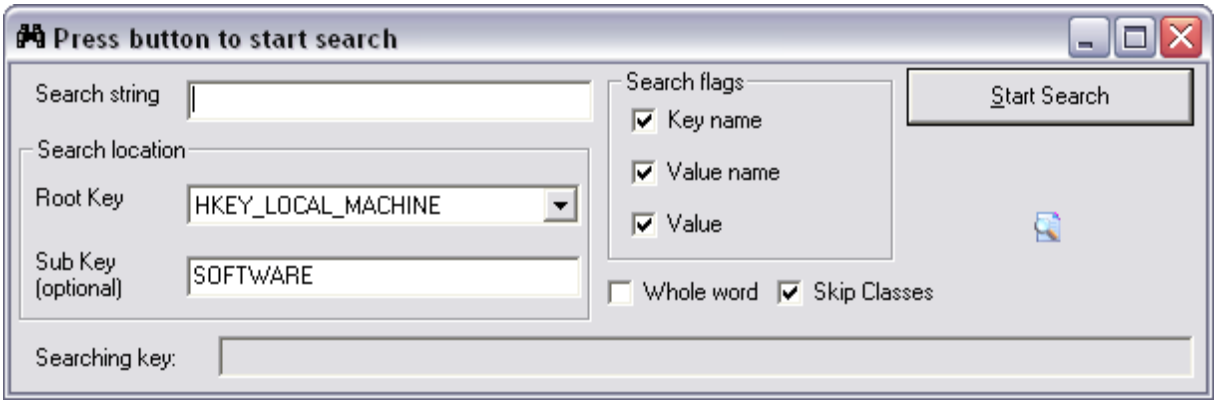
We need the registry path for the folder, not the executable file. So we need the "Path " and not the "(Standaard)" (default in English). I usually store the registry path in a text file along with the pictures.

The registry path is:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\rayne.exe\Path

Registry Searcher:

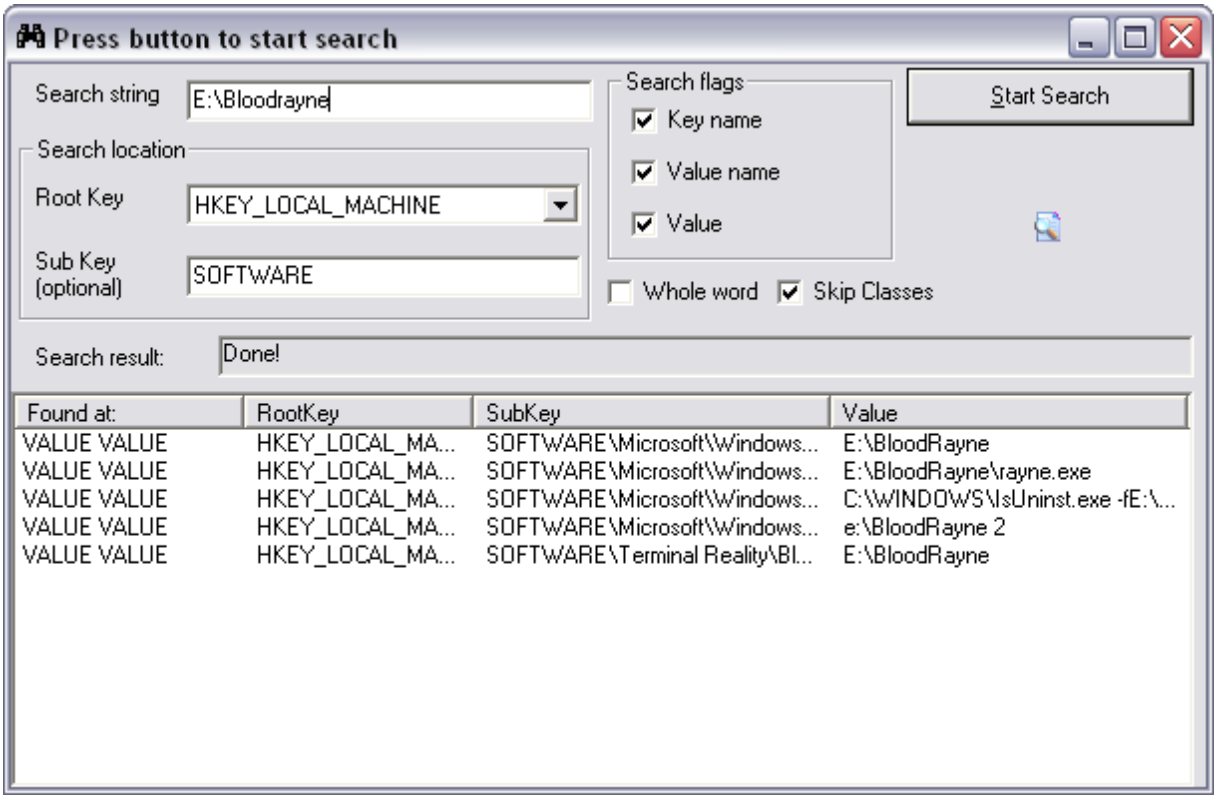
This program was build to search the registry, not to edit it. It's safe and easy to use.
You can find it in the tools section on http://www.cd-2-dvd.com.

This program was written by Arkadiy Olovyannikov and slightly modified by Skeraxe.
all credits should go to Arkadiy Olovyannikov for creating and sharing his source code.

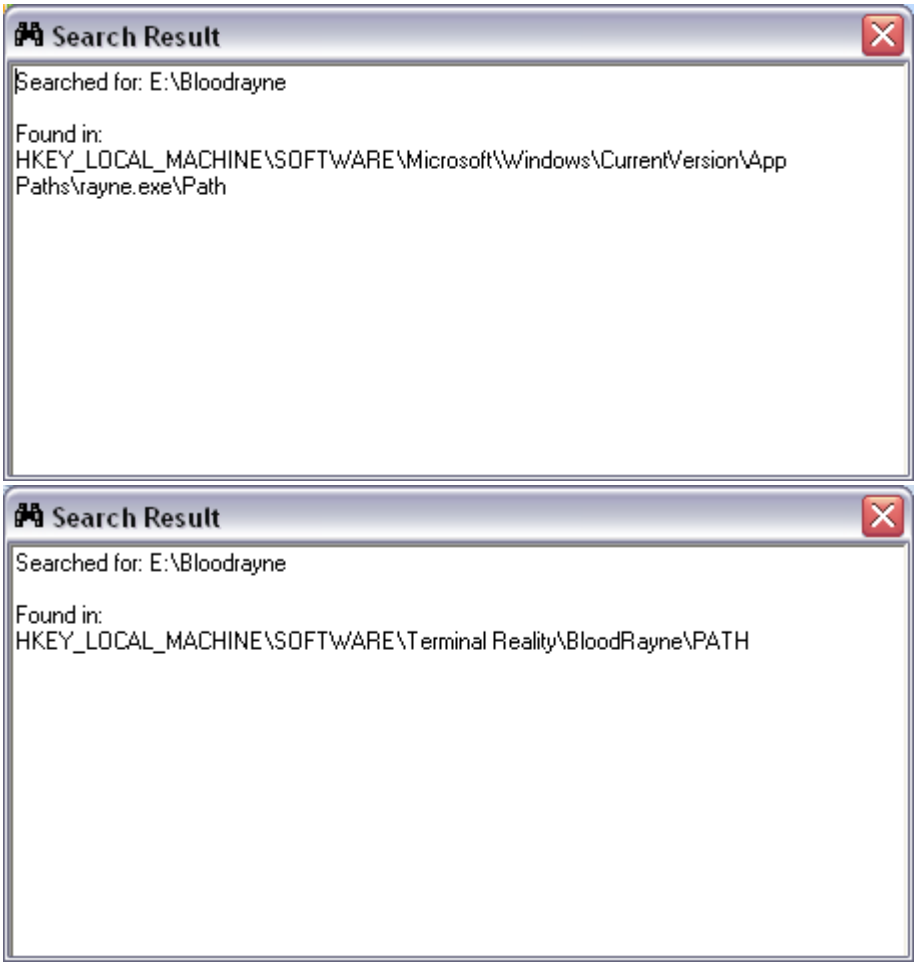


Most of these settings are already good, I've modified the program to make it easier for me to find the game paths. Included in the zip file is a detailed guide on how to use the program so I'll keep it short.

All I do is fill in where I've installed the game ("E:\Bloodrayne") and click "Start Search".



As you can see, there are 2 addresses I can use for this game. I double-click both of them to get these dialogs:



I try not to use the Microsoft\Windows\CurrentVersion path, not that it really matters but it's my personal policy.

So I simply copy and paste the other address to a text file I store with the pictures.
I could've found the second address with regedit as well simply by searching again. But I usually use this program so I get everything at once.

Editing your resources:

We've got everything we need, so now it's time to edit the pictures to form the menu.
I always use Adobe Photoshop although any image editing program will do.

I start with the background, I delete all the company logos and give it a standard Bevel.
Resized it to 640x480 and put a small signature on the bottom right.

Then I resize both logos so they're about the same size and position them on the background.

The first page is done, on to the second and most important page of the menu.

I reduce the size of both the logos and reposition them at the top one overlaying the other.
Then I type the text for the buttons, I don't use real buttons for this menu, only text.

I used the font: "Viner Hand ITC"
Gave it a 1 pixel stroke and a Bevel.

This is what the menu is going to look like when it's done. I don't use this picture as it is but it's more of a reference.

From all the buttons, I create 4 separate pictures:
Normal:

Highlighted:

Clicked:

and Disabled:

I also cut out the logos at the sizes I want to use, leaving me with:
1 Empty Background with the lady on the right
4 logos, 2 big and 2 small
28 button pictures, 4 pictures for every button.

Building the Menu:

Pages:

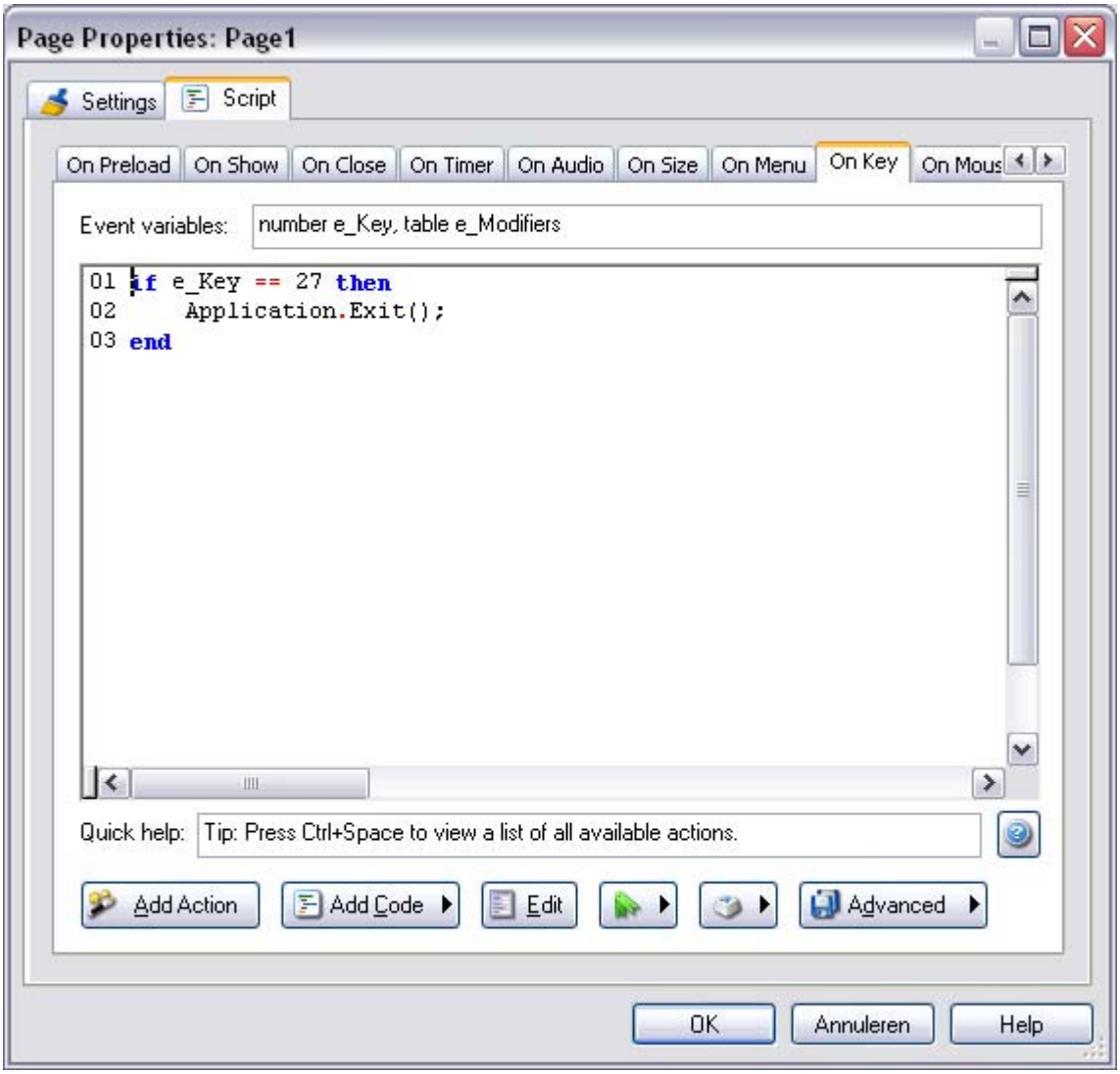
It's time to start AMS. I'm using AMS7.0 for this tutorial.
Create a new blank project with the name BloodrayneDVD.

In the "Project Settings" I change the size to 640x480 pixels to match the background picture.
Change the Style to "Flat" to remove the windows border and buttons.
In Options, I check the "Custom Icon" setting and browse for the icon file.

In the "Page Properties" for page1, tab "Settings", I change the background to Image and browse for the background picture. Setting the ImageStyle to "Actual Size".

The first thing I'm coding, is a way to exit the menu without the use of buttons. The first page doesn't have an exit button and I also removed the X button at the top right.

At the "Page Properties" for page1, tab "Script", sub tab "On Key", I put the following code.

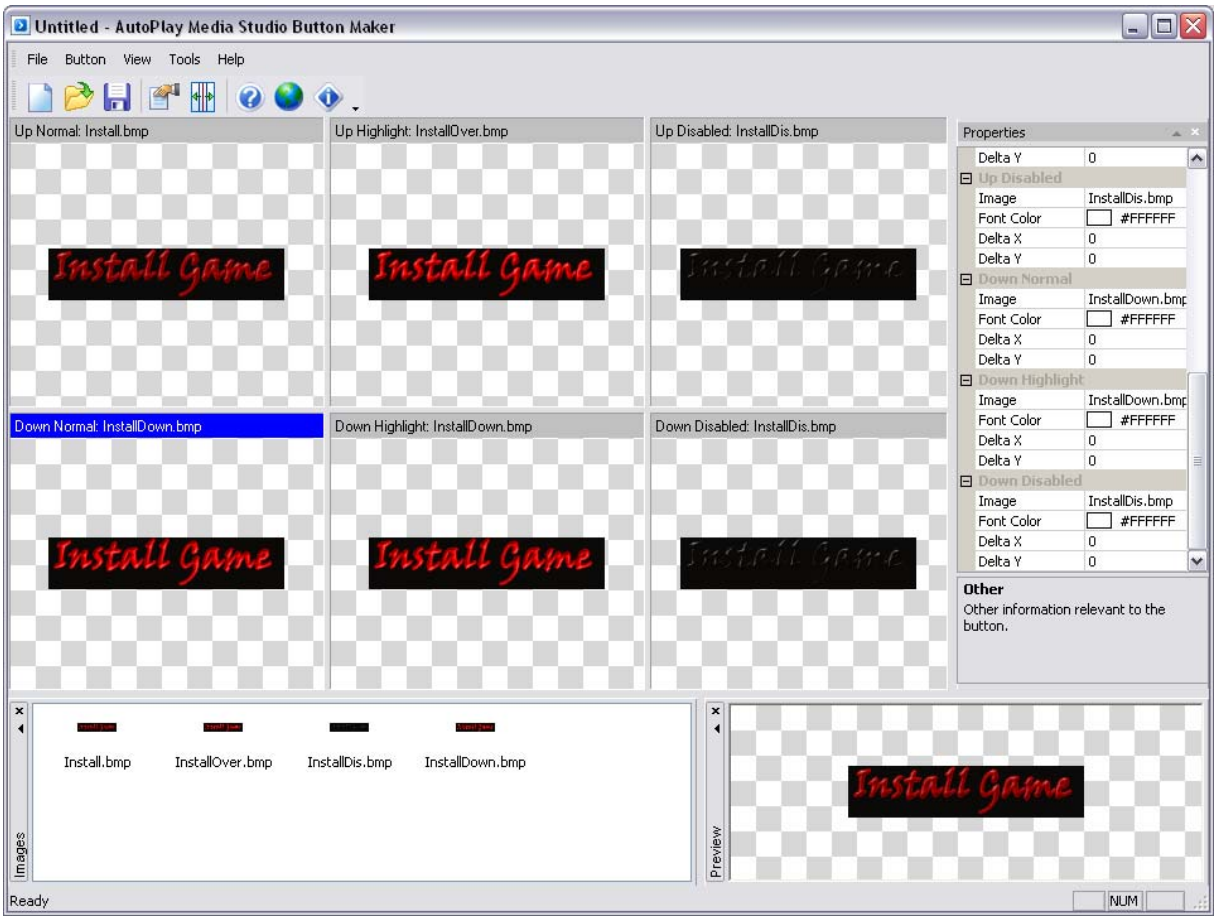


This code will exit the application any time the user presses the key [Escape] on the keyboard.
Now I simply add 2 image objects with the 2 big logos and page1 is done for now.

Now for the second page (page2) I set the page properties the same as with page1 and I also add the same code to exit the program. Again I add 2 image objects which contain the smaller logos, and put them on top of each other on the top of the page. Depending on which game the user choose, 1 logo will be shown while the other logo will be invisible. We'll code this later on…
I also rename the images to "imgLogo1" and "imgLogo2". This will help understand the code better.

Buttons:

I create custom buttons with the "Button Maker" tool provided with AMS.
It gives 6 options but I only need 4, the other options are simply filled with a similar picture.



Up Normal = Normal picture
Up Highlight = Highlight picture
Up Disabled = Disabled picture
Down Normal = Clicked picture
Down Highlight = Clicked picture
Down Disabled = Disabled picture

I also removed the text because the text is in the pictures.
So create all the buttons like this and position them on page2 any way you like it.
I always give the buttons the same name so I don't have to rewrite the code all the time.
btnInstall, btnPatch, btnCrack, btnChoose, btnBrowse, btnAbout and btnExit.
btn being a short code for button.

Coding Page1:

There's not much code to be written for this page. The only thing we need is to know which game the user choose and store that in a variable. Then show page2.

I use the Visual Basic naming standard but you can simply change the name if you want.
I'll save the game as a number in the variable "gintGameIndex"

g = global: because the entire application will use this variable
int = integer: the standard for saving small numbers
GameIndex: simply a name so I can recognize it in the code

So, go to the properties for the bloodrayne logo and to the scripting tab for the event: "On Click"
gintGameIndex = 1;
Page.Jump("Page2");

For the Bloodrayne 2 logo, write this script:
gintGameIndex = 2;
Page.Jump("Page2");

Repeat this for every game you want on the DVD, changing the number in gintGameIndex.

Coding Page 2:

Again, not much to code because I like to link the buttons to functions in a different place. This helps to keep things organized and easy to find. The functions will be discussed in the next chapter.

For each button, go to “properties”, “Scripts” tab and then “On Click” event sub tab.

Install Button:

```
InstallGame(gintGameIndex);
```

Both these function will be addressed in detail in the following chapter.
Note that the “InstallGame” function gets the same variable as we coded in page1. This is how the menu will know which game to install.

Patch Button:

```
PatchGame(gintGameIndex);
```

Crack Button:

```
CrackGame(gintGameIndex);
```

Choose Button:

```
gintGameIndex = 0;
Image.SetVisible("imgLogo1",false);
Image.SetVisible("imgLogo2",false);
Page.Jump(“Page1”);
```

This will reset the gameindex and logos. Then show the first page again.
Note that if you have more than 2 games, you should also put their visibility to false.

Browse DVD Button:

```
File.ExploreFolder( _SourceFolder ,SW_SHOWNORMAL);
```

This will open your windows explorer on the root of the DVD.

About Button:

```
File.Run(_SourceFolder.."\\AutoPlay\\Docs\\About.exe","",_SourceFolder.."\\AutoPlay\\Docs",SW_SHOWNORMAL,true);
```

I usually put an About button with this picture in the menu, but you can skip this button if you don’t want it in your menu.

Exit Button:

```
Application.Exit();
```

That's it for the buttons, but we still need to change the logo depending on the game the user choose. Go to the Page Properties for page2, to the "Scripts" tab and then the "On Preload" subtab.

```
if gintGameIndex == 0 then
	Dialog.Message(errNoGameIndex[1],errNoGameIndex[2]);
elseif gintGameIndex == 1 then
	Image.SetVisible("imgLogo1",true);
	Image.SetVisible("imgLogo2",false);
elseif gintGameIndex == 2 then
	Image.SetVisible("imgLogo1",false);
	Image.SetVisible("imgLogo2",true);
else
	Dialog.Message(errToManyGameIndex[1],errToManyGameIndex[2]);
end
```

It's all pretty easy to understand.
If gintGameIndex = 0 then give an error. (Explained in the next chapter)
If gintGameIndex = 1 then ensure that all logos are invisible except for the first.
If gintGameIndex = 2 then ensure that all logos are invisible except for the second.
If gintGameIndex = 3 or up, give an error (Explained in the next chapter)

Note that if you have more than 2 games, you should add the logo to "Image.SetVisible" list. And also write a new "elseif" structure.

I always like to build the menu to either disable or enable buttons depending on if the game is installed or not. The following code is needed to make this work. The function setButtons() is later explained in the next chapter.

Under the "change logo" code I put a blank line and add the following code. This will start a Timer which will call a function every 100 milliseconds.

```
Page.StartTimer(100);
```

On the "On Close" event tab, I add the following code. This will disable the timer when the user closes the menu.

```
Page.StopTimer();
```

On the "On Timer" event tab, I add the following code. This code will run every X milliseconds. For this example I've set the X to 100.

```
setButtons();
```

The pages are done. Now it's time for the actual functions.

Coding global functions:

Errors:

The errors will be shown in a dialog message box. I like to put the errors in a separate section to keep things organized. I will make a few small array's containing 2 strings. 1 for the dialog title and 1 for the actual message.

If you're building your menu in a different language, you can change all the texts the menu will show.

In the menu "Project" go to "Actions". Select the "On Startup" tab.

```
errNoSetup = {"No Setup found...", "The installation file couldn't be located"};
errNoPatch = {"No Patch found...", "There's no patch found on this DVD."};
errNoCrack = {"No Crack found...", "There's no crack found on this DVD."};
errNoGame = {"Game not Installed...", "The game isn't installed on this computer.\nPlease Install the game and try again."};
errToMany = {"To many files found...", "There are too many *.exe files in that folder.\nDo you want to Patch the game manually?"};
errDuringCrack = {"Error during cracking", "The game is NOT Cracked\nBecause of an unknown error."};
errCrackComplete = {"Cracking Complete" ,"The game has been cracked.\nHave fun playing!"};
errNoGameIndex = {"No GameIndex", "Their wasn't a game chosen on the first page."};
errToManyGameIndex = {"GameIndex to high", "There's not that many games programmed in the menu."};
```

Game Settings:

This part is very important. We'll make an array of all the games in the menu with all the settings the program will need. All the other functions which we'll get to later on in this chapter, will use this array. Because of this, we don't need to edit the functions for other games.

When you want to create another menu, you only have to build the GUI and edit the game settings. The functions should still work the way they will be when we're done.

I'm putting a blank line after the error section in the "On Startup" event. I've put some comments in the first array to tell you what everything does.

```
game_1 = {};  -- Don't change
game_1.Name = "Bloodrayne"; -- Enter the name of the game
game_1.Code = "BR1";    -- Short code for the game. Ensure all the folders needed are named like this
game_1.Index = 1;       -- Not used in the example but is useful if you put this variable array into another array
game_1.CrackPath = _SourceFolder .. "\\Cracks\\BR1";  -- The folder containing all the crack files
game_1.PatchPath = _SourceFolder .. "\\Patches\\BR1"; -- The folder containing the patch file(s)
game_1.RegistryKey = "SOFTWARE\\Terminal Reality\\Bloodrayne"; -- Registry path to the settings of the game
game_1.RegistryValue = "PATH";
-- This is where the setup stores the path to which the game is installed. Needed to check if a game is
```

```
installed and for cracking the game
-- If you only find a registry settings with a filename and path, the code will extract the path from the string
game_1.SetupPath = _SourceFolder .. "\\BR1\\Setup.exe"; -- Path to the setup which will install the game

game_2 = {};
game_2.Name = "Bloodrayne 2";
game_2.Code = "BR2";
game_2.Index = 2;
game_2.CrackPath = _SourceFolder .. "\\Cracks\\BR2";
game_2.PatchPath = _SourceFolder .. "\\Patches\\BR2";
game_2.RegistryKey = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\{04CB9967-A8BB-468C-ABA6-CE87328712BE}";
game_2.RegistryValue = "InstallLocation ";
game_2.SetupPath = _SourceFolder .. "\\BR2\\Setup.exe";

game_Table = {};       -- Don't change
game_Table[1] = game_1; -- Store the game_1 array with its settings inside the game_Table
game_Table[2] = game_2;         -- Store the game_2 array with its settings inside the game_Table
```

Please note the double slashes “\\” in the script. This is needed in AMS and the program wouldn’t work without them!

You only need to edit what’s behind the “=” symbol, if you change anything else then the program wouldn’t work correctly.

checkGame():

This function checks to see if a game is installed or not. I like to disable the patch and crack buttons if a game isn't installed. Or disable the install button if the game is installed.
This just gives a more professional look to the menu in my opinion.

From this point on, all the coding will be done in the "global functions" section.
In the menu "Project" go to "Global Functions".

```
function checkGame(Index)
-- Check the registry if a game is installed.
	Install_Path =
Registry.GetValue(HKEY_LOCAL_MACHINE,game_Table[Index].RegistryKey,game_Table[Index].Regist
ryValue,true);
	if Install_Path == "" then
		return false;
	else
		return true;
	end
end
```

This function is rather easy to understand. It checks the registry for a key.
If the key exists, the game is installed and the code will return TRUE.
If the key doesn't exist, the game isn't installed and the code will return FALSE.

getRelativePath():

This function is needed by the Cracking function. The menu will give it 2 paths to a file and returns the relative path. This is explained in more detail in the CrackGame() section of this chapter.

```
function getRelativePath(sPath, sRelativeTo)
-- Needed by the cracking code. Get's the relative path compared to another path.
	if String.Length(sPath) > String.Length(sRelativeTo) then
		if String.Right(sRelativeTo,1) ~= "\\" then sRelativeTo = sRelativeTo.."\\" end
		if String.Left(String.Lower(sPath),String.Length(sRelativeTo)) ==
String.Lower(sRelativeTo) then
			RelativePath = String.Right(sPath,String.Length(sPath) -
(String.Length(sRelativeTo)))
			return RelativePath;
		end
	end
end
```

setButtons():

This function will check if the game is installed and change the buttons to enabled or disabled accordingly. If the game isn't installed, it'll enable the install buttons and disable the patch and crack buttons. If the game is installed, it'll disable the install button and enable the patch and crack buttons.

```
function setButtons()
	if checkGame(gintGameIndex) == true then
		Button.SetEnabled("btnInstall",false);
		if Folder.DoesExist(game_Table[gintGameIndex].PatchPath) == true then
			Button.SetEnabled("btnPatch",true);
		else
			Button.SetEnabled("btnPatch",false);
		end
		if Folder.DoesExist(game_Table[gintGameIndex].CrackPath) == true then
			Button.SetEnabled("btnCrack",true);
		else
			Button.SetEnabled("btnCrack",false);
		end
		Page.StopTimer();
	else
		Button.SetEnabled("btnInstall",true);
		Button.SetEnabled("btnPatch",false);
		Button.SetEnabled("btnCrack",false);
	end
end
```

This function is called by the "On Timer" event we placed on Page2.
It uses the checkGame() function to check if a game is installed.
If the game is installed, it'll set the buttons accordingly and then stops the timer.

The reason I've put a timer on this function is this:
When I click the Install button and the setup begins to install the game, I can't check if it's done installing and if the install was a success. The checkGame() function checks the registry and if it finds a specific key it'll return TRUE. Meaning that the install is still busy or is already finished.
So when the setup is done installing the game, the buttons have changed and you can continue with either patching or cracking the game.

InstallGame():

This function will check if the setup file exists on the DVD, and run it if it is. It will also extract the "working folder" from the path. This option is needed by some setup files to work correctly.

```
function InstallGame(Index)
-- Simply runs the setup or gives an error if the setup file can't be found.
	strPath = game_Table[Index].SetupPath;
	if File.DoesExist(strPath) == true then
		working_Folder = String.SplitPath(strPath);
		strWorkingFolder = working_Folder.Drive..working_Folder.Folder;
		if String.Right(strWorkingFolder,1) == "\\" then
			strWorkingFolder = String.Left(strWorkingFolder,
(String.Length(strWorkingFolder) - 1));
		end
		File.Run(strPath,"",strWorkingFolder,SW_SHOWNORMAL,true);
	else
		Dialog.Message(errNoSetup[1], errNoSetup[2], MB_OK, MB_ICONEXCLAMATION,
MB_DEFBUTTON1);
	end
end
```

PatchGame():

This function looks in the Patch path for every *.exe file.
If it doesn't find any file, it'll give an error
If it finds 2 or more *.exe files, it'll ask if you want to run the patch manually
If it finds only 1 *.exe file, it'll run it automatically, patching the game.

```
function PatchGame(Index)
	strPath = game_Table[Index].PatchPath;
	if Folder.DoesExist(strPath) == true then
		found = File.Find( strPath, "*.exe", false, false);
		if found == nil then
			Dialog.Message(errNoPatch[1], errNoPatch[2], MB_OK,
MB_ICONEXCLAMATION, MB_DEFBUTTON1);
		elseif Table.Count(found) == 1 then
			File.Run(found[1], "", strPath, SW_SHOWNORMAL, false);
		else
			result = Dialog.Message(errToMany[1],errToMany[2], MB_YESNO,
MB_ICONINFORMATION, MB_DEFBUTTON1);
			if result == IDYES then
				File.ExploreFolder(strPath, SW_SHOWNORMAL);
			end
		end
	else
		Dialog.Message(errNoPatch[1], errNoPatch[2], MB_OK, MB_ICONEXCLAMATION,
MB_DEFBUTTON1);
	end
end
```

CrackGame():

This function will crack the game automatically. Meaning it'll copy all the cracked files to the games installation folder, creating a backup of the original.

Example how it works:
It checks the registry where the game is installed and stores it in "Install_Path".
If the registry returns a path with a filename, it'll strip the file and store only the path.
For this example, Install_Path = "C:\Bloodrayne 2".

Recursively search for all the files in the crack folder and process the files one-by-one.
For this example I will use this file: "D:\Cracks\BR2\rayne2.exe".
The crack folder is: "D:\Cracks\BR2"

The getRelativePath() functions gets both the file and the crack folder and returns the relative path of the file being "rayne2.exe". The cracking function adds this to the Install_Path getting: "C:\Bloodrayne 2\rayne2.exe".

Now it will copy D:\Cracks\BR2\rayne2.exe to C:\Bloodrayne 2\rayne2.exe, making a backup of the original file with the .bak extension and repeat this for any other file it found in the crack folder.

For the sake of this example, I'm going to pretend there is another file. Although this game doesn't have another cracked file, it'll give you an idea for other games.

It finds: "D:\Cracks\BR2\System\engine.dll"
The crack folder for this game is still: "D:\Cracks\BR2"

The getRelativePath() functions gets both the file and the crack folder and returns the relative path of the file being "System\engine.dll". The cracking function adds this to the Install_Path getting: "C:\Bloodrayne 2\System\engine.dll".

As you can see, if the crack needs to be in a subfolder of the game's install folder, you'll need to put that file in a subfolder with the same name in the crack folder…

All the other stuff in this code will give dialog message boxes.
Either if the crack succeeds or fails, it'll let the user know what happened.

```
function CrackGame(Index)
	strCrackPath = game_Table[Index].CrackPath;
	Install_Path =
Registry.GetValue(HKEY_LOCAL_MACHINE,game_Table[Index].RegistryKey,game_Table[Index].Regist
ryValue,true);
	if Install_Path ==  "" then
		Dialog.Message(errNoGame[1],errNoGame[2], MB_OK, MB_ICONEXCLAMATION,
MB_DEFBUTTON1);
	else
		Install_Path = String.SplitPath(Install_Path);
		if Install_Path.Extension == "" then
			strInstallPath = Install_Path.Drive.."\\"..Install_Path.Filename;
		else
			strInstallPath = Install_Path.Drive..Install_Path.Folder;
		end

		if String.Right(strInstallPath,1) == "\\" then
			strInstallPath = String.Left(strInstallPath, (String.Length(strInstallPath) - 1))
		end

		if Folder.DoesExist(strCrackPath) == true then
			found = File.Find(strCrackPath,"*.*",true,false);
			if found ~= nil then
				for index, path in found do
					RelativePath = getRelativePath(path, strCrackPath);
					bSuccess =
File.Install(path,strInstallPath.."\\"..RelativePath,FILE_INSTALL_ALWAYS,true,false,nil,nil);
				end

				if bSuccess then
					Dialog.Message(errCrackComplete[1], errCrackComplete[2]);
				else
					Dialog.Message(errDuringCrack[1], errDuringCrack[2],
MB_OK, MB_ICONEXCLAMATION, MB_DEFBUTTON1);
				end
			else
				Dialog.Message(errNoCrack[1],errNoCrack[2], MB_OK,
MB_ICONEXCLAMATION, MB_DEFBUTTON1);
			end
		else
			Dialog.Message(errNoCrack[1],errNoCrack[2], MB_OK,
MB_ICONEXCLAMATION, MB_DEFBUTTON1);
		end
	end
end
```

To end:

I've build two menu's using these codes and they work as expected. If you want to rewrite any of these codes, feel free to do so.

I would like to point out, that I never build my menu's (available on http://www.cd-2-dvd.com) in AMS. I build them in Visual Basic. So with my basic knowledge of AMS I've write these but maybe somebody with a better knowledge of the program could make them more efficient.

This tutorial is more of a guideline on how to build the menu's and I encourage people to fiddle around with the program and find their own way.