Extracting Diablo II Animations

By Paul Siramy

Copyrights

Credits and Thanks (in no particularly order)

TeLAMoN	CV5, CV5 plugins, DCC thinking
Bilian Belchev	Doc on DCC format, helped me a LOT, and of course CVDCC.DLL
Alkalund & Myhrginoc	First researches about the DCC / COF / D2 files formats
Peter Hu (aka Isolde)	Various infos on .D2 file formats. AWESOME work on the patch 1.10, thanks you so much for all the new possibilities for us, mod-makers!
Sir_General & Jarulf	Infos about Animation Speed
Sloan Roy (SVR)	Great and very useful DRTester, I just love it
Phrozen Keep	Thanks to all the staff and members of that great community for their kindness and various helps

Special Greeting

Since this tutorial deals essentially with D2 animations, I would want to greet all the artists of Diablo II for their impressive amount of work on that game. Great job!

Table of Content

Overview

This tutorial will explain you the complete process that is required for making an animated GIF from an animation of the game **Diablo II**, and its expansion **Lord of Destruction**. With the current tools we have at our disposal today, this is the only way to transform a multi-component Monster into a torso-only animation, thus helping the creation of a 'new' Monster in the game with great ease.

It'll explain in very deep details how Animations are made in the game, so be prepared for lots of stuff that you didn't ask for ;) Therefore it'll makes the things more understandable if you want to make your own animations as they were originally made (anyone concerned?), or just if you want to enhanced them maybe. All this knowledge is not needed for the purpose of just making animated GIF, as you can achieve this result by trials and errors, but knowing how it is all working sure makes the things more accurate and easy.

I'll take several examples: Player Character, Monster, and Object, thus covering all major cases, if not all:

- I'll first take one of my Characters as an example, a dual-weapons **Barbarian**, which was equipped especially for this tutorial. This exercise will be the longest (yep, very long), since it'll explain all the concepts that are used for animations in D2, like components of animations, colormaps, how to find the correct informations in the TXT files, all the complex stuff about 1 hand / 2 hands / 2 weapons modes, why Tyrael and some other animations can't be converted easily … and the usage of some Tools.

- In the second exercise we'll extract a **Fallen**, because Monsters are a little different than Player Characters.

- Right after, we'll take the example of **the Countess**, to understand her color system in the patch 1.10.

- And finally, in the fourth exercise we'll take 1 Object, the **Arcane Teleport Pad**, for its alpha-blended layers problems.

Required knowledge:

I'll consider that you already know these bases of Mod Making:

- Extracting files from MPQ
- Reading TXT files in Microsoft Excel or D2Excel (we won't *edit* anything tough)

I'll also assume that you know how to use these programs, at least a minimum (no need to be an expert), as I won't repeat what's in their respective documentation (but sometimes I'll take the time to explain some things step-by-step, especially for DRTester):

- DRTester
- Merge_dcc
- Animdata_edit
- Paint Shop Pro - version 6.00
- Animation Shop 2 - version 2.00

1. GET THE TOOLS

1.1. DRTester

We'll first take an excellent Tool, that's DRTester by **Sloan Roy** (**SVR**), since this wonderful program allow you to browse an MPQ, test animations (COF, DCC) in real-time, test them with different colormaps, extract the file you're viewing and the related ones simultaneously. It also allows you to view DT1 and even DS1 (maps), among other good things! This program will be of an inestimable value for us. But the problem is that there are 2 versions of that program, each one having its good and bad points. I'll compare them later.

You can get the 1st version (28 Nov 2002) at:
http://files.d2mods.com/pafiledb.php?action=file&id=171 (49.9 kB)

And you can grab the 2nd version (01 Dec 2002) at:
http://files.d2mods.com/pafiledb.php?action=file&id=887 (50.3 kB)

None of these 2 versions require any additional files.

Note: you can get updated version of DRTester directly on the site of his creator (http://home.stx.rr.com/svr), but then this document will be slightly different of what you'll have. For instance the version available on this site can make a GIF from the animation settings you have set when viewing a COF. So, don't hesitate to upgrade 🙂

1.2. Merge_dcc

This is a program I have made that is especially designed to transform a multi-part animation into a torso-only animation, also allowing you to apply colormaps on each component. I'm taking this moment to thanks again **Bilian Belchev** for having help me a LOT to make my DCC decoder 🙂

You can get this Tool at:
http://files.d2mods.com/pafiledb.php?action=file&id=173 (113 kB)

And you also need the **Allegro DLL**, which can be retrieved at:
http://files.d2mods.com/pafiledb.php?action=file&id=161 (226 kB).

Just place that DLL in the directory of Merge_dcc and the program will run fine.

1.3. Animdata_edit

This is another program I have made. This one is for editing the file **Data\Global\Animdata.d2**. It extracts its datas in a .txt, allowing you to easily edit them with MS-Excel or D2Excel, and then it encodes them back.

You can get this Tool at:
http://files.d2mods.com/pafiledb.php?action=file&id=876 (19 kB)

It doesn't require any additional files.

2. EXERCISE 1: A PLAYER CHARACTER

2.1. Overview

Here's a quick glance of the Animation we'll recreate. It's not particularly beautiful, but it's using items that have some tints on them, so it'll be a very good exercise. I have chosen a Barbarian because the Dual-Weapon ability is an exception among the Player Characters animations, so this will make it also for a good exercise.

**My Barbarian, equipped
especially for us**

2.2. Equipments that modify the animations

As you have certainly noticed before, there are only 4 items that your Player is wearing that are affecting the animation of the player during the game. They are the Left and Right hands, the Armor, and the Helm:

The only 4 interesting items for us

The Amulet, the 2 Rings, the Belt, the Gaunt, the Boots, the Alternative weapon(s) and / or shield, and all the other items you have in your Inventory / Horadric Cube / Private Stash / Mercenary don't affect the animation of your player. These 4 items are the keys of the Player animation.

NOTE: The patch 1.10 has introduced the possibility for the Set Items to have overlays under the Player animation, but we won't take that effect into consideration in this document.

2.3. Finding the correct files

Now we need to know exactly which animation files we'll use. There can be 2 cases: either you know the items the Player have, or not. In our case we know, so it'll make things much more accurate. In the case of a 1.09 Monster (like Blood Raven), we'd have to find them by trials and errors, with several screenshots. But just let's see what my Barbarian have.

Socketed Magical Sword

By knowing the type of the item (here "Champion Sword") we'll be able to find the base graphic file, and by knowing the prefixes, suffixes, and socketed items, we'll be able to know the color effect to apply on the base image, if any.

Unique Helm

Unique items usually have a color effect, which is the case here. We'll learn how to know exactly which one.

Unique Weapon

Another interesting Unique item with a color effect, but this time it's a Weapon.

Set Item Armor

Set item, like the Unique items, usually have a color effect, which is the case here. We'll learn how to retrieve this one too.

2.3.1. Necessary items informations

So, what do we really need to know about these items? For Unique items we just need to know the name as we'll be able to retrieve all the other informations from there. That's the same for the Set items. For the other items, we need to know the name, the gems that are socketed into (if any), the prefixes and suffixes, and lastly the list of its properties, since items can get a color effect from there too. In our case it's simple:

- **Helm**
    - o Quality   = Unique Item
    - o Name   = "Vampire Gaze"

- **Left Hand**
    - o Quality   = Unique Item
    - o Name   = "Baranar's Star"

- **Armor**
    - o Quality   = Set Item
    - o Name   = "Sigon's Shelter"

- **Right Hand**
    - o Quality   = regular Item (not a Unique nor a Set item)
    - o Type   = Champion Sword
    - o Socketed gems   = Amn
    - o Affixes   = Cruel
    - o Properties   = Enhanced Damage, Life steal

2.3.2. Retrieving Unique items datas

Unique items are defined in **Data\Global\Excel\UniqueItems.txt**. As always, take the last version you have: check first in **patch_d2.mpq**, else in **d2exp.mpq**, else in **d2data.mpq**. I'll take the one in patch_d2.mpq, version <mark>1.09 D</mark>.

2.3.2.1. Vampire Gaze

It's the line 211 in Microsoft Excel, column **name** having *Vampiregaze* (without spaces). The interesting datas for us are:

- code   =   xh9
- type   =   Grin Helm (not really needed, but just for safety later)
- transform   = 1
- transformcolor   = 12

Now, let's take **Data\Global\Excel\Armor.txt**. A search for **xh9** into the **code** column give us the line 91 in Excel, **name** = *Grin Helm*. This is the type we have read in UniqueItems.txt so no errors, that's this one. The interesting datas for us are:

- alternategfx   = bhm
- Transform   = 2

The **Transform** value in Armor.txt set to 2 tell us the animation will use the colormap Data\Global\Items\Palette\grey2.dat found in d2data.mpq when a color effect needs to be made on this object. The **transform** value in UniqueItems.txt set to 1 tell us this animation will use it, and therefore the **transformcolor** 12 tell us it'll use the tint *Crystal Green*.

To understand exactly how these colors effects are made for Unique items, Set items, and Gems color effects, read a doc on that subject at: http://d2mods.com/colormaps.php

2.3.2.2. Baranar's Star

It's the line 259 in Microsoft Excel, **name** = *Baranar's Star*. The interesting datas are:

- code   =   7mt
- type   =   Devil Star
- transform   = 1
- transformcolor   = 7

Now, let's take **Data\Global\Excel\Weapons.txt**. A search for **7mt** into the **code** column give us the line 219 in Excel, **name** = *Devil Star*. The interesting datas are:

- alternategfx   = mac
- wclass =   1hs
- Transform   = 1

The **Transform** value in Weapons.txt set to 1 tells us the animation will use the colormap Data\Global\Items\Palette\grey.dat found in d2data.mpq. The **transform** value in UniqueItems.txt set to 1 tell us this animation will use it, and therefore the **transformcolor** 7 tell us it'll use the tint *Light Red*.

Just next to the **wclass** column stand the **2handedwclass** column. For the Mace the value is **1hs** for the 2 columns.

2.3.3. Retrieving Set item datas

Set items are defined in **Data\Global\Excel\SetItems.txt**. I'll take the one in patch_d2.mpq, version 1.09 D.

The **Sigon Set** collection is at the line 11 in Excel, column **name** = *Sigon's Complete Steel*. The **Sigon's Shelter** is at column AQ (**Item2 Suffix**). The interesting datas are:

- transform = 1
- transformcolor = 0
- Item 2 = gth

Now, let's take **Data\Global\Excel\Armor.txt**. A search for **gth** into the **code** column give us the line 20 in Excel, **name** = *Gothic Plate*. The interesting datas are:

- alternategfx = gth
- Transform = 2

The **Transform** value in Armor.txt set to 2 tell us the animation will use the colormap Data\Global\Items\Palette\grey2.dat found in d2data.mpq when a color effect needs to be made on this object. The **transform** value in SetItems.txt set to 1 tell us this animation will use it, and therefore the **transformcolor** 0 tell us it'll use the tint *White*.

```
Patch
1.10
```

This patch has brought modifications in **UniqueItems.txt** and **SetItems.txt**. Here are the modifications that concern both files:

➢ The column **transform** don't exists anymore

➢ The column **transformcolor** is split into 2 columns: **chrtransform** and **invtransform**: the first for the Player Character animation color, the other for the item inventory version.

➢ Both **chrtransform** and **chrtransform** are no longer index but color code, from **Data\Global\Excel\Colors.txt**.

➢ Colors of some Unique Items (and some Set Items as well) have changed from 1.09. In this tutorial, I'm referring to the colors in the version 1.09 D.

2.3.4. Retrieving other items datas

They're using both **Armor.txt** and **Weapons.txt**. The *Champion Sword* is at the line 235 with Excel in Weapons.txt. The interesting datas are:

- alternategfx   =   clm
- wclass   =   1hs
- Transform   =   1

The **Transform** = 1 tell us the animation will use the colormap Data\Global\Items\Palette\grey.dat found in d2data.mpq when a color effect needs to be made on this object (which is the case here, since the Sword have an **Amn** rune in it).

To known the tint that will be use, let's check **Data\Global\Excel\Gems.txt**. The Amn rune is at line 48 in Excel, and the **transform** column has an 18 there, so it'll be the tint *Dark Purple*…

Hey no, wait! The doc about the colors effects for Uniques, Set and Gems say:

> Runes can't *have* nor *make* any tint effect

So in fact, since it's a *rune* that's in the 1st socket, there is no colors effect on that Sword. If it was a **Perfect Saphire**, the line in Excel would have been the 11th, and therefore the tint would have been *Crystal Blue*. But since it's a rune, our Sword will sadly stay normal… Well, it'll make our job easier. But we'll still see later how to apply such a tint.

Note: of course I assume that we're working on a un-modified version of **Lod,** else as the doc on colors effects indicate, it's possible to have tints with Runes too, but it's not the case here.

But does the Claymore really doesn't have any color effect? 🤪 In fact it *does* have one in our case.

Not because of the Rune, but because of the **Cruel** Prefix, which makes it turn into a Black sword. That's why I said we needed the list of the Properties for the regular items.

There are 3 files that have the possibility to have properties that modify the color of an object. They are all in **Data\Global\Excel** and are **MagicPrefix.txt**, **MagicSuffix.txt** and **AutoMagic.txt**. I won't go into details tough, so I'll just say that they all have the **transform** and **transformcolor** columns, and that they works the same way as in weapons.txt, armor.txt and misc.txt.

The *Cruel* Prefix is in MagicPrefix.txt. It has the *transform* column set to 1, therefore when it is spawned on an item it changes its colors. The *transformcolor* set to *blac* tell us it'll be the tint Black (just check **Data\Global\Excel\colors.txt**) and therefore the index of this tint is **3** (we count the lines where the codes are, starting from 0, not 1).

Now, in Weapons.txt, just next to the **wclass** column, stand the **2handedwclass** column. For the *Champion Sword*, we can see that the 2 columns are different. It's **1hs** for the **wclass** column, while it's **2hs** for the **2handedwclass**. Let's open the file **Data\Global\Excel\WeaponClass.txt**:

Weapon Class	Code
None	
Hand To Hand	hth
Bow	bow
1 Hand Swing	1hs
1 Hand Thrust	1ht
Staff	stf
2 Hand Swing	2hs
2 Hand Thrust	2ht
Crossbow	xbw
Left Jab Right Swing	1js
Left Jab Right Thrust	1jt
Left Swing Right Swing	1ss
Left Swing Right Thrust	1st
One Hand-to-Hand	ht1
Two Hand-to-Hand	ht2

So, the Champion Sword has both the **1 Hand Swing** and the **2 Hand Swing** animations.

Finding what each code really does can be hard the first time you see them. So we'll first see how animations are made, and then only after will we understand what's the logic beyond these modes.

Now is the good time to let the deep analyze of the animations starts…

2.4. Components of Animations

As you may already know, the animations in Diablo II are split into several parts. When you are equipping another helm, just the head of the animation change. Same for when you're equipping another weapon, only the weapon part of the animation change. That's because the Player is divided into several parts. Let's open **Data\Global\Excel\Composit.txt**:

Name	Token
Head	HD
Torso	TR
Legs	LG
RightArm	RA
LeftArm	LA
RightHand	RH
LeftHand	LH
Shield	SH
Special1	S1
Special2	S2
Special3	S3
Special4	S4
Special5	S5
Special6	S6
Special7	S7
Special8	S8

Here you can see all the logical division of the animations in Diablo II, not only for the Players but also for Monsters and Objects. They are **Layers**: the game superposes them to recreate the Player animation. It's exactly like in the making of cartoons. By replacing the head layer by another head, the final animation is different, without having to change all.

These layers are ordered. When drawing an animation of a Player in a given direction, for a given frame, a table tells in what order to draw them. When the Player is facing West, you first see his Shield, and the rest of the body is behind. But when facing East, the Shield is now the one that is behind.

Here's an example. All of those layers are combined to make the final Player image:

S1   HD   S2

RH   TR   SH   Final result

RA   LG   LA
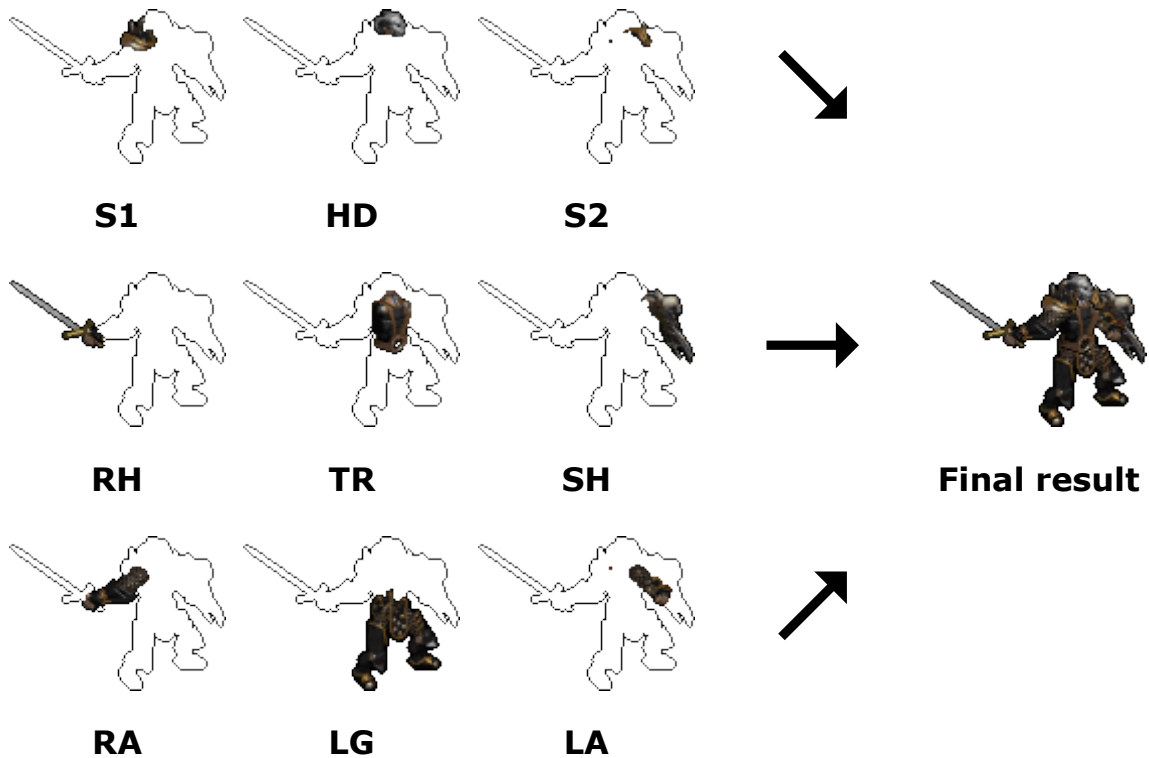
If you look closely, you'll see that it's not just a simple "crop" division: some layers intersect each other's, like the **RA** layer which is overlapping with the layers **S1**, **RH** and **TR**. That's why the order to draw them is important: you don't want the **RA** layer to be draw on top of the **S1**, do you?

2.5. Weapon's Class

Some weapons must be used with 2 hands, others can be wielded in both 1 and 2 hands, and another weapons in 1 hand only. In this last case the 2nd hand can be a shield or another weapon. The 1-hand weapons can be of the Swing type (sword) or Throw (javelin). That was just a *very* simple overview ;)

The key to understand the Weapon's Class is to think about the entire Body Position of the Player, and especially his **Arms**. The Body does not move the same way when fighting with a Bow than when fighting with a Spear for instance. There is one Weapon's Class for each possible case covered by the game. In the precedent example, the **RH** layer just had the Sword draw onto it, not the Arms. If the Player now equipped himself with an Axe, only the **RH** layer will change, the Arms and the rest of the Body will stay exactly the same:

RH   **Final result**

As you see, only the **RH** layer changed, and it doesn't have any Arms draw onto it, but the final image is still ok. That means that the precedent Sword, when equipped in 1 hand, and this Axe are using the same Weapon's Class: they are compatible Weapon's Alternative of the same Main Animation.

Here are examples and descriptions for each possible Weapon's Class of the game:

2HT	STF	2HS	BOW	XBW	HT1	HT2

1HT	1HS	HTH	1SS	1JT	1ST	1JS
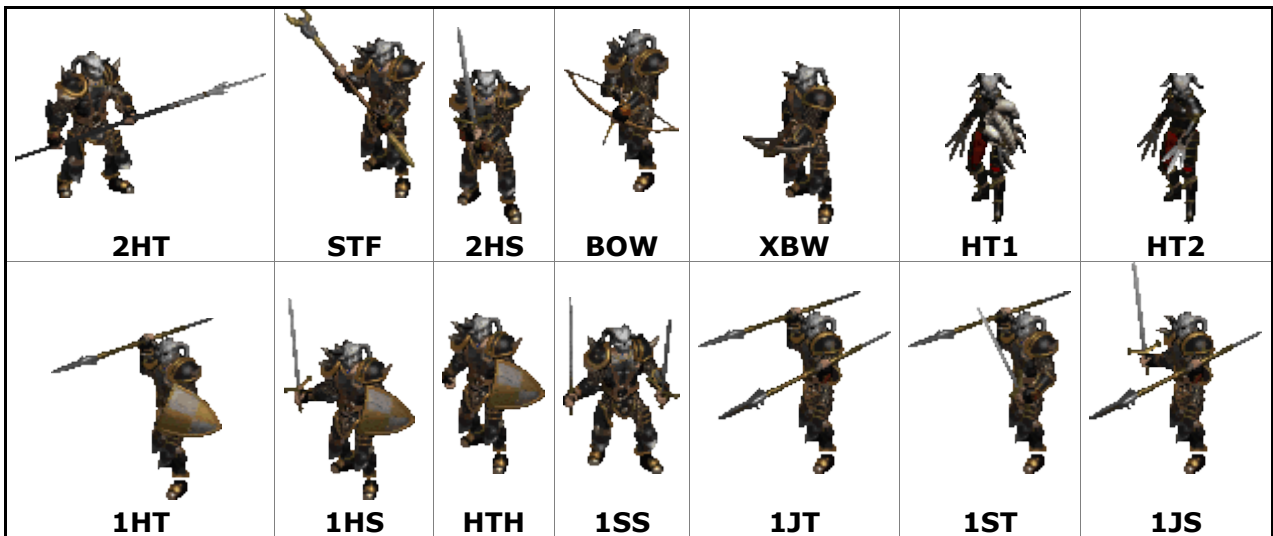
2HT	=	"2 Hand Thrust"	Spear
STF	=	"Staff"	Staff, Large Axe, Maul, Pole arm
2HS	=	"2 Hand Swing"	2-Handed Sword
BOW	=	"Bow"	Bow
XBW	=	"Crossbow"	Crossbow
HT1	=	"One Hand-to-Hand"	Shield + Claws
HT2	=	""Two Hand-to-Hand"	Claws + Claws
1HT	=	"1 Hand Thrust"	Shield + (Throwing potion, Knife, Throwing Knife, Javelin)
1HS	=	"1 Hand Swing"	Shield + (Axe, Wand, Club, Scepter, Mace, Hammer, Sword, Throwing Axe, Orb)
HTH	=	"Hand To Hand"	Shield + no weapon
1SS	=	"Left Swing Right Swing"	Left = **1HS**, Right = **1HS**
1JT	=	"Left Jab Right Thrust"	Left = **1HT**, Right = **1HT**
1ST	=	"Left Swing Right Thrust"	Left = **1HS**, Right = **1HT**
1JS	=	"Left Jab Right Swing"	Left = **1HT**, Right = **1HS**

2.6. Token and Path of the animations

Here are some Base Paths of different type of animations:

- **Data\Global\Chars**   for Player Characters (Amazon, Barbarian…)
- **Data\Global\Missiles**   for Missiles (Arrow, Fireball…)
- **Data\Global\Monsters**   for Monsters / NPC (Fallen, Duriel, Cain…)
- **Data\Global\Objects**   for Objects (Torch, Chest, Shrine…)
- **Data\Global\Overlays**   for Graphical Effects (Aura, Curse, Explosion…)

For the Player Characters, the animations are in **d2char.mpq** for the Classic game (Amazon, Sorceress, Necromancer, Paladin, and Barbarian). For the Expansion (Druid, Assassin) they're in **d2exp.mpq**. Let's open from d2exp.mpq the file **Data\Global\Excel\PlrType.txt** :

Name	Token
Amazon	AM
Sorceress	SO
Necromancer	NE
Paladin	PA
Barbarian	BA
Expansion	
Druid	DZ
Assassin	AI

Our Barbarian is the **BA** Token. Therefore the Base Path of all the animations of our Barbarian is **Data\Global\Chars\BA**. Be aware tough that despite you'll find of course in d2char.mpq all the classic Barbarian animations, there are more animations in d2exp.mpq, like these Class-Only Helms that they can wear in LoD now.

2.7. Player Character's Mode

Players have several Modes, and each Mode has its own animations. Monsters and Objects have other different Modes, but it's the same logic. So, what are they for our Barbarian ? Let's open the file **Data\Global\Excel\PlrMode.txt**:

Name	Token
Death	DT
Neutral	NU
Walk	WL
Run	RN
Get Hit	GH
Town Neutral	TN
Town Walk	TW
Attack1	A1
Attack2	A2
Block	BL
Cast	SC
Throw	TH
Kick	KK
Skill1	S1
Skill2	S2
Skill3	S3
Skill4	S4
Dead	DD
Sequence	GH
Knock back	GH

For the Barbarian animation that we'll recreate, the Character's Mode is TN (Town Neutral), and not the NU (Neutral). NU is the neutral mode when the Player is in the Wilderness, while the TN neutral mode is used while the Player is in Town.

The *Sequence* Mode is a special feature. There are some Players / Monsters that have an animation for some special cases which are coded as a sequence of frames from several other animations. This is a way to recycle images. But the problem for Mod Makers is that sequences are hard coded in DLL, so making a new Monster animation where such a sequence exists is hard (except for the patch 1.10 where here the monsters sequences are placed into **MonSeq.txt**).
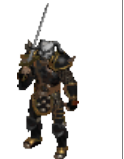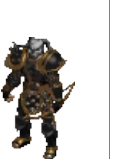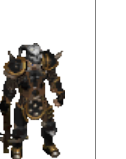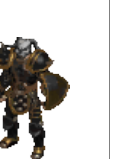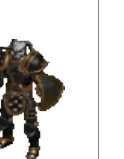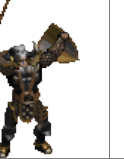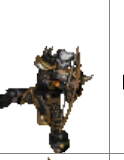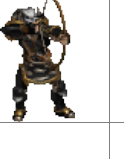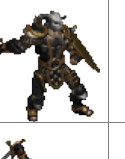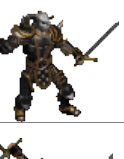
Here are all the Modes of the Barbarian. Note that a Player don't necessary have all the Modes that are defined in PlrMode.txt, which is the case here since the Barbarian don't have the **Skill 2** animation type. For another Player Character the actual animations can vary, for instance the Amazon have only **Skill 1**, and it's not a Jumping attack but a Dodging ability.

DT	DD	TN	NU	SC	KK
A1	A2	TH	S1	S3	S4
TW	WL	RN	BL	GH	
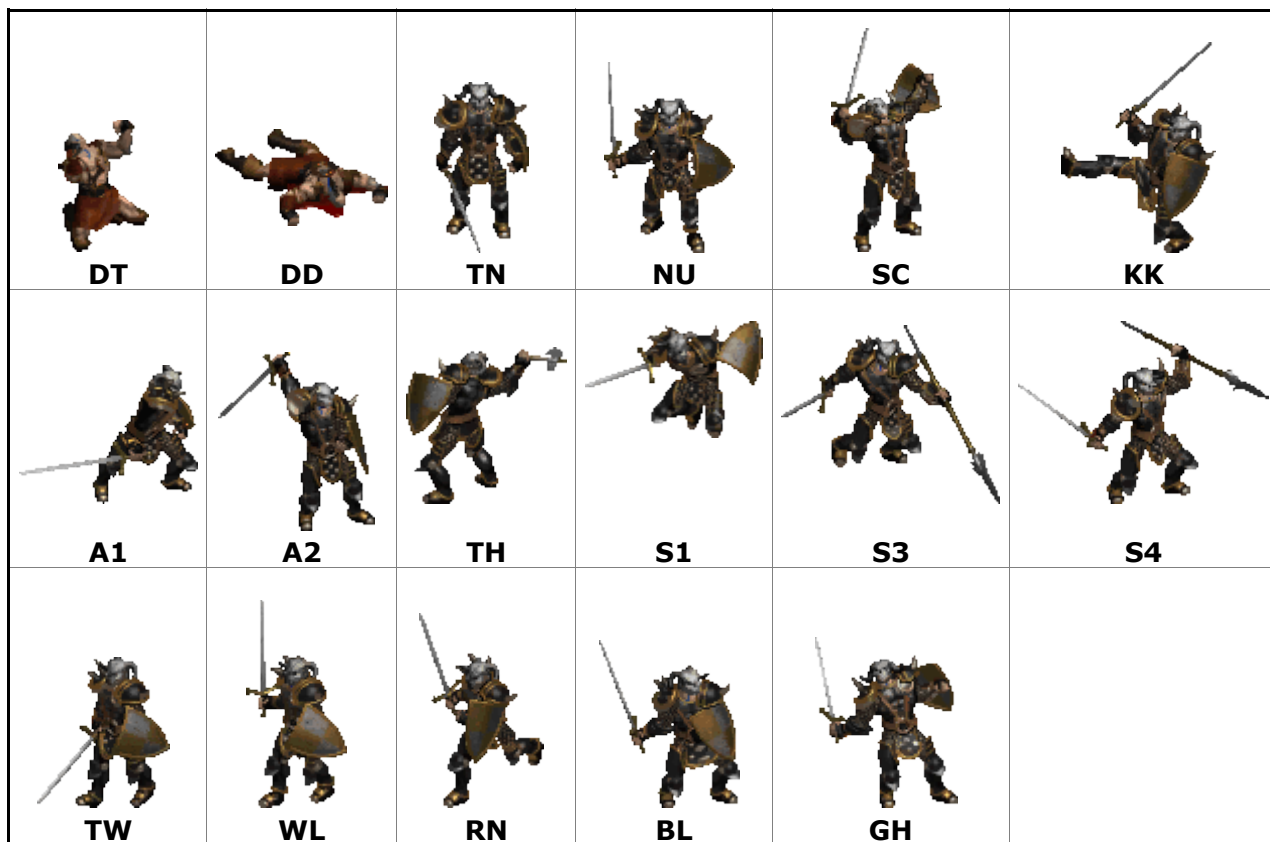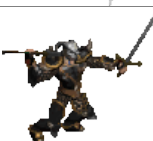
DT	=	Death	Dying animation.
DD	=	Dead	The corpse
TN	=	Town Neutral	Idle, lowering his guards in Town
NU	=	Neutral	Idle, still on his guards out of Town
SC	=	Cast	Casting a Skill
KK	=	Kick	Kicking a Barrel for instance
A1	=	Attack 1	Attack type 1, here Swinging by the side with a 1-hand sword
A2	=	Attack 2	Attack type 2, here Swinging by the top with a 1-hand sword
TH	=	Throw	Throwing axe, knife, javelin…
S1	=	Skill 1	Skill type 1, here the Jumping Attack
S3	=	Skill 3	Skill type 3, here a Swinging attack with the weapon in the Left hand
S4	=	Skill 4	Skill type 4, here a Throwing attack with the weapon in the Left hand
TW	=	Town Walk	Quiet walk in Town, lowering his guards
WL	=	Walk	Precautious walk out of Town, still on his guards
RN	=	Run	Running
BL	=	Block	Blocking an attack with his shield
GH	=	Get Hit	Getting hit by an attack, also used when taking a Knock back attack

Player's Mode and Weapon's Class are combined, so that's making a lot of different animation types. Almost all the Player's Mode just above are of a 1HS (1-hand swing + Shield) Weapon's Class, but there are (almost) all the same Player's mode with the other Weapon's Class.

Here's a table that present what are all the existing **Barbarian's animations type**, given a Weapon's Class and a Player's Mode:

	2HT	STF	2HS	BOW	XBW	1HT	1HS	HTH	1SS	1JT	1ST	1JS
DT	none	none	none	none	none	none	none		none	none	none	none
DD	none	none	none	none	none	none	none		None	none	none	none
TN												
NU												
SC												
KK												
A1												
A2				none	none			none				

	2HT	STF	2HS	BOW	XBW	1HT	1HS	HTH	1SS	1JT	1ST	1JS
TH	none	none	none	none	none							
S1												
S3	none	none	none	none	none	none	none	none				
S4	none	none	none	none	none	none	none	none				
TW												
WL												
RN												
BL	none	none	none	none	none				none	none	none	none
GH												

2.8. Types of Animation's files

There are 6 types of files relatives to animations: **DCC**, **COF**, **DAT**, **DC6**, **D2** and **TXT**.

- **DCC** are the most basic graphical part of an animation. For instance there's a DCC for the Claymore when used by the Barbarian while he's in Neutral mode, using this sword with the 2-handed Weapon's Class (because it's a sword that can be equipped by 1 or 2 hands by the Barbarian and in our example he don't have a Shield). If that Barbarian stay in his Neutral mode but now equipped himself with a Shield, the Claymore will be used in another Weapon's Class (1-handed), so it'll be another DCC that will be used to draw the Claymore. If now the Barbarian begins to walk, he'll be in another Mode, so it'll be again another DCC that will be used to draw that Claymore.

  Now, if he give that Claymore to a Paladin, even if that Paladin is in Neutral Mode, and is using the Sword in a 2-handed Weapon's Class (that Character don't have the choice anyway), it'll be again another DCC that will be loaded by the game to draw that Claymore, because it's another Player Character:

  > For a given item, there is a DCC for all possible combinations of Player / Mode / Weapon's Class where that item can be used in the game.

  Let's find for our Claymore how many DCC exists:
  - 7 Players
  - Around 12 Modes per Player where a Sword can be used
  - Between 1 and 5 Weapon's Class

  Well, we can't really make a formula, as they're so much special cases: Barbarian with Dual-Weapons ability, Swords that can be equipped in 1-hand OR in 2-hand, Barbarian's Skill 3 that other Players don't have… So, how many of them do *really* exist? If we check the MPQ we'll find **120** of them. That's many animations for just **1** Claymore, isn't it?

- **COF** are the files that controls how are assembled all Layers together to form the final animation. When the Barbarian is in Neutral Mode, with the 1HS (1-Hand Swing) Weapon's Class, this is 1 COF. Wetter he's using a 1-hand Claymore, or a 1-hand Axe, or a Club, or a Hammer… it's the same COF: the Barbarian' s Body movements are exactly the same for all of that weapons, because just that weapon *graphical* part change. The COF control layers drawing order, and drawing a Sword instead of an Axe when the moment to draw the weapon comes is not the purpose of the COF.

  The COF is basically a table that indicates for all directions and for all frames, the order of the layers to draw. In a given direction, for a given frame, the RH (Right Hand) Layer is draw at a particularly moment, and it is the same wetter the Barbarian is using a 1-hand axe or a 1-hand sword. COF is a sort of the model of the animation, and each one of the Layers of that COF usually have several DCC possibilities, thus making diversity in the animations of the game.

  For the Barbarian, remember all the images you have see in the 2 precedent pages. There were 149 images, so you'll find 149 COF in the MPQ for him. The 150th you can find in the MPQ is in fact not useable, as there's no DCC at all that are using it.

- **DAT** is a generic file extension that just means "DATA". For the purpose of Animations, the DAT files are colormaps, allowing an animation to be declined into several color set. For instance, The Fallen Monster is Red. A colormap associated to this animation makes it possible to have Blue, Green, and Brown Fallens. There are other DAT files, specifically made for all the Player Characters.

- **DC6** is the standard graphical file format used in Diablo II, not especially for animations but usually for Title Screen, Icons, Item's Graphics, Inventory Background, Skill Tab Background… The DC6 files are in a very simple encoding format. It's even simpler than the RLE encoding format used by the PCX files. That's because DC6 were made to only handle one thing: the transparency. All the solid pixels are in a raw format, no compression. There are just "jumps" of pixels encoded where transparency is found. That makes DC6 very quick to be used at run-time. On the opposite, DCC are very compressed, in a very complex format, and with a loss of quality (but this one is almost unnoticeable).

  An interesting thing to know about the DC6 and the DCC files, according to Bilian Belchev documentation on the DCC format (see below), is that DCC can be considered as compressed DC6, because the decoder of Diablo II when reading a DCC file, is not "simply" decompressing it, it's decoding it in the DC6 format. As a proof we can check the directory **Data\Global\Overlays**, we'll find 384 DCC (and 1 DC6, the usual Mephisto exception), while in the directory **Data\Global\uncompoverlays** we'll find 11 DC6 and no DCC at all ("uncomp" standing for "uncompressed").

  The DCC File Format Documentation by Bilian Belchev can be found at:
  http://files.d2mods.com/pafiledb.php?action=file&id=253

  Why talking about DC6 here? Because despite almost all of the animations files of Player Characters / Monster / Objects / Missiles are in DCC format, there are some exceptions, like Mephisto, Tyrael, Maggot Queen's Death, Mephisto's Hell Gate… It doesn't have any influence for DRTester, but it does have one for my Merge_dcc program: for now it doesn't handle DC6, just DCC, so the above exceptions can't be extracted… not easily at least.

- **D2** is a file format specifically used for animations datas. In fact, there are 2 type of D2 file in the MPQ despite they have the same extension. The first type is mainly used to retrieve the Animation Speed and the Number of Frames per Direction of any animation and is the source of common troubles when making a new animation. The second type of D2 format contains the copy of many COF. These last D2 files are preloaded at the start of the game, so when the game needs to read a COF, it first looks in that D2 files, and if the COF is not here, it'll looks for the regular one instead.

  D2 files of type 1 (animations datas), used by the game:

  - Data\Global\AnimData.d2

  D2 files of type 2 (copy of many COF), used by the game:

  - Data\Global\chars_cof.d2   Player Characters
  - Data\Global\cmncof_a1.d2 COF needed for Act 1
  - Data\Global\cmncof_a2.d2 for Act 2
  - Data\Global\cmncof_a3.d2 for Act 3
  - Data\Global\cmncof_a4.d2 for Act 4
  - Data\Global\cmncof_a6.d2 for Act 5 (that's really cmncof_a**6**.d2)

- **TXT** is the common Text file format found in Data\Global\Excel. So which file does we need from there? Objects.txt, because it have an Animation Speed data that override the one we can find in AnimData.d2. That's only for Objects of course, not the Player Characters, Monsters, Overlays or Missiles animations.

2.9. Naming convention of Animation's files

With so many different combinations of Players, Player's Mode, Weapon's Class, Components, Items… the need to order all of that that comes quickly. We'll see the naming convention used for COF and DCC, and the Directories used.

2.9.1. COF

The naming of a COF is simple. It has 7 letters for its name, then come the extension. The name is composed of 3 elements, the Token, the Mode, and the Weapon's Class. For our Barbarian, here's the name of the cof that we'll use: **BATN1SS.cof**

<Token> <Mode> <Weapon's Class> <Extension>

<Token>	=	**BA**
<Mode>	=	**TN**
<Weapon's Class>	=	**1SS**
<Extension>	=	**.cof**

2.9.2. DCC

The naming of the DCC follow the model of a COF, it just introduces 2 new elements: the Layer code, and the Item code for that Layer. That shows a relation between the DCC and the COF. The Helm animation file that our Barbarian will use is: **BAHDBHMTNHTH.dcc**

<Token> **<Layer> <Item>** <Mode> < Weapon's Class **in COF**> <Extension>

<Token>	=	**BA**
<Layer>	=	**HD**
<Item>	=	**BHM**
<Mode>	=	**TN**
<Weapon's Class>	=	**HTH**
<Extension>	=	**.dcc**

But why the <Weapon's Class> of the DCC is **HTH** while the COF that we'll use have the <Weapon's Class> **1SS?** Shouldn't it be **1SS** too? No, because there's something special in the COF: it doesn't have only tables for Layer drawing priority, it also has some datas for these Layers, and without going into details, one of them is the Weapon's Class to used for the DCC of that Layer.

Here's the example of our **BATN1SS.cof**. In that COF the Layers LA, RA, RH and LH are set to use DCC with a Weapon's Class **1SS** (that's seems normal for a **1SS** COF after all). But the other Layers (LG, TR, S1, S2 and HD) are set to only accept **HTH** DCC.

Why? Just to avoid to makes tons of useless animation. After all, should the Head Layer of the Barbarian be very different in Town Neutral Mode when he has no weapon than when he has 2 weapons? Do you really see a problem in the 2 images below for using the exact same Head for those 2 different animations? Obviously no.

2.9.3 Directories

As you know, all Unit Types (Player Character, Monster, Object and such) have their Base Path. In that Directory we find lots of other directories: one for each Token. For our Barbarian it's the **Data\Global\Chars\BA** directory.

Under that Token Directory there is always a **COF** directory, all the COF of that Unit are in there, like for our **BATN1SS.cof**. If our Unit has a **Palshift.dat**, we can find it there too. We won't find any for Player Characters, but we will for most of the Monsters.

Under the Token Directory, at the same level of the COF directory, we can find several Layer directories, like **HD**, **TR**, **LG**, **S1** … all the DCC of each particularly Layer are in there. For instance our DCC **BAHDBHMTNHTH.dcc** will be found in the **HD** Directory.

```
<Base Path>
    └── <Token>
            ├── <COF>
            │       ├── *.cof
            │       └── (Palshift.dat)
            └── <Layer>
                    └── *.dcc of that layer (there are *.dc6 in some cases)
```

2.10. Finding the good DCC of the Armor

Before we launch DRTester, there's a last thing we need to learn. Despite we have already found what are the DCC that will be used for some Layers (Right & Left hands, and the Head), there's still a problem with the Armor. In **Data\Global\Excel\Armor.txt** we can see that the armors are using more than 1 layer component in order to produce the final armor image. Let's take the example of our *Gothic Plate*:

Name	code	rArm	lArm	Torso	Legs	rSPad	lSPad
Gothic Plate	gth	2	2	1	2	2	2

The numbers are telling exactly which DCC for each Layer will be taking. That's why when we change the armor of the Player in the game, sometimes just 1 Shoulder change and not the Torso: the 2 armors were using the exact same 5 components (rArm, lArm, Torso, Legs, lSPad) but a different Right Shoulder (rSPad).

Now, let's open **Data\Global\Excel\ArmType.txt**:

Name	Token
Lite	lit
Medium	med
Heavy	hvy

We have 3 lines in **ArmType.txt**, and coincidently in **Armor.txt** we can only find values from 0 to 2 in the components. Obviously the numbers 0, 1 and 2 are the indexes of the line in **ArmType.txt** (starting the count from 0 and not 1).

So now we can deduce that our Gothic Plate will use these DCC for making the final Armor image:

Component	Layer	Value	Token	DCC
rArm	RA	2	hvy	BARAHVYTN1SS
lArm	LA	2	hvy	BALAHVYTN1SS
Torso	TR	1	med	BATRMEDTNHTH
Legs	LG	2	hvy	BALGHVYTNHTH
rSPad	S1	2	hvy	BAS1HVYTNHTH
lSPad	S2	2	hvy	BAS2HVYTNHTH

While we're at it, here are the DCC for the other layers, accordingly to the datas we found at the start of the exercise. That's the **alternategfx** column that was giving the DCC code:

Layer	alternategfx	DCC
HD	BHM	BAHDBHMTNHTH
LH	MAC	BALHMACTN1SS
RH	CLM	BARHCLMTN1SS

2.11. Comparing the 2 versions of DRTester

We're now almost ready to use DRTester, but since there are 2 versions in the File Center, let's see what their differences are.

	Version	
	28 Nov 2002	**01 Dec 2002**
Title Screen	🟡	✓
Zooming Animations	✓	🟡
DC6 Items	🟡	✓
DC6 Animations	🟡	✓
COF with DC6 layers	🔺	✓
Alpha-Blended Animations	🔺	✓
DS1 Maps orientation	🟡	✓
DS1 Maps Wall Layers	🟡	🟡

✓ **Good**
🟡 **More or less useable**
🔺 **Not useable**

Some more detailed explanations:

- **Title Screen**: In the 1st Version the Title Screen when you launch DRTest.exe is flipped vertically. In the 2nd Version the problem is not here. That's an easy way to distinguish the 2 Versions. The problem is that all DC6 in the 1st version are flipped.

- **Zooming Animations**: In the 1st version no particularly problems, but in the 2nd version some zoom values can be screwed, depending of your Operating System and / or Video Card, as show below:
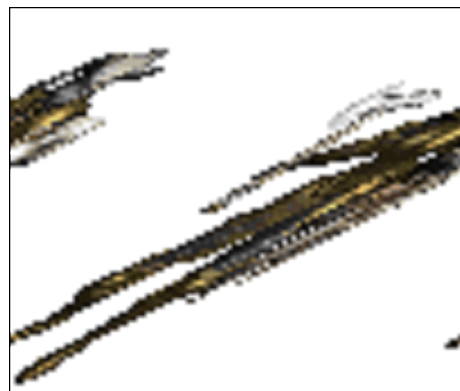
Zoom x2   **Screwed Zoom x3**

- **DC6 Items**: Like for the Title Screen, they're flipped vertically in the 1st version

- **DC6 Animations**: Idem

- **COF with DC6 layers**: In the 1st version, you simply can't use them. Therefore you can't see Mephisto, or the Wings of Tyrael for instance. But in the 2nd version no problems, as DRTester is now able to handle both DCC and DC6 layers indifferently.

- **Alpha-Blended Animations**: In the 1ˢᵗ version the Alpha Blending is not implemented, but <u>you can use it in the 2ⁿᵈ version</u> if you want (just choose a PL2).

Without Alpha-Blending   **With Alpha-Blending**

- **DS1 Maps orientation**: Like for the DC6, the <u>Tiles are flipped in the 1ˢᵗ version</u>, so you can hardly browse DS1. In the 2ⁿᵈ version the *orientation* problem is gone (but another one has appeared):

Flipped in version 1   **"Normal" in version 2**
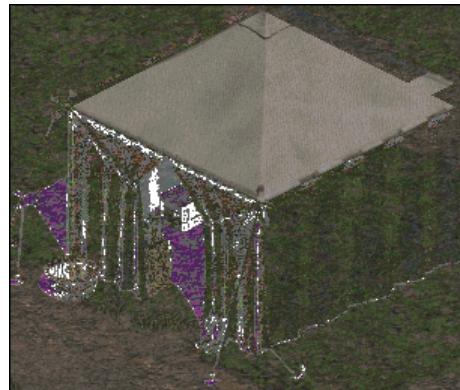
- **DS1 Maps Wall Layers**: <u>In the 2ⁿᵈ version the Wall Layers are using a colormap</u>, this was for a testing purpose. At first, they're looking weird, but by choosing the PL2 of the Act and the good Index (306 or 562) we can make them appear almost normal again:

By default

**With Act1\Pal.Pl2
and the Index 562**

<u>In all versions</u> the **Preview** button above the image allow you to view the map correctly, walls becoming normal (no flipping and no transparency problems in there).

2.12. Working with DRTester

Now the Theory is finish. You have learned:
- How to find in the .txt the informations you need (color effects, DCC code of all Layers)
- What are Player's Mode and Weapon's Class, therefore which COF to use
- Under which Path and Token are localized the Animations files
- That Animations have a variable speed

From now on, I'll use the last version of DRTester (01 Dec 2002, available on my site). We don't really need the Zoom, so the fact that it can be screwed is not a problem. And after all, this version is a lot better than the precedent 😊

We'll use DRTester to easily extract all the files we need, and to check if we have chosen all the correct Animations files.

Let's launch **DRTest.exe**. After a moment we'll have this window:
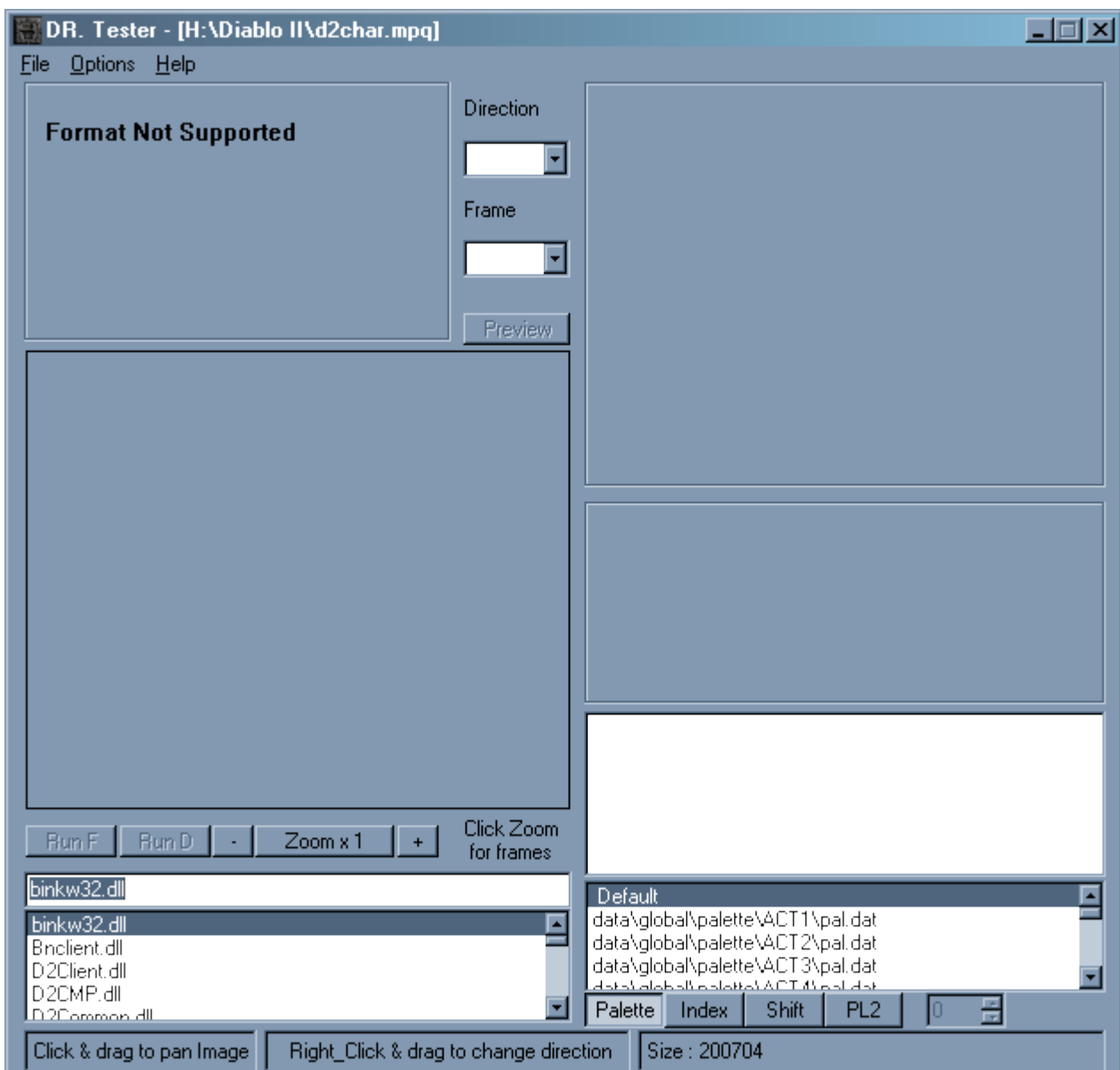
Here are all the commands available in the Menu bar:
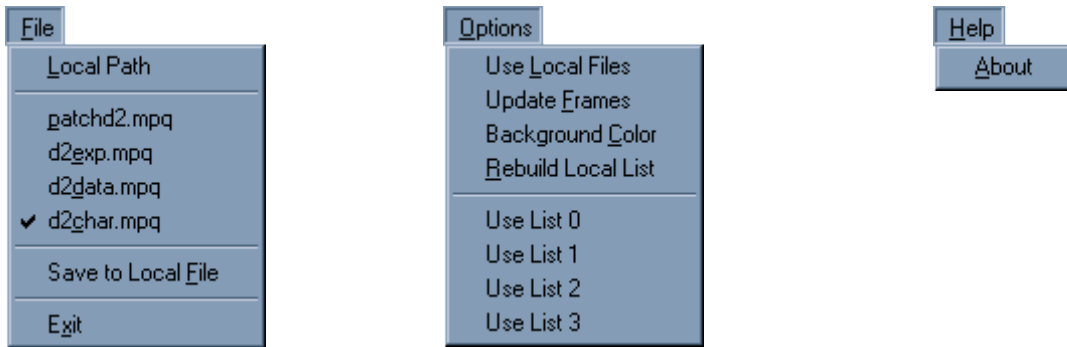
You can see a list of 4 MPQ files: **patchd2.mpq**, **d2exp.mpq**, **d2data.mpq** and **d2char.mpq**. DRTester can only browse one MPQ at a time, so this is the place where you can switch between all of them. Just choose another MPQ and you'll browse that new one, making the corresponding name checked in the Menu, instead of the precedent one. Since DRTester was primarily made to test the DCC format, that's the MPQ *d2char.mpq* which is checked by default. So each time you're opening DRTester you're browsing the Player Character's Animations. This is exactly what we need for our Barbarian.

When you're on a file, you can extract it with the menu **Save to Local File**. It can be any kind of file like DCC, COF, DT1, DS1, DC6 … but it can also be any kind of file present in the MPQ not supported by DRTester, like the TXT. Depending of the type of the file you're asking to save, you can be prompted to save not only the file you're currently viewing, but also the related files, and that's pretty useful:
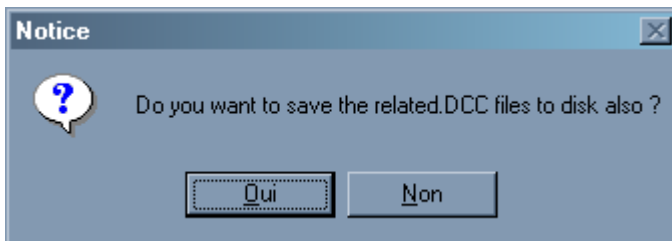
If you're viewing a COF you'll be prompted to save also **all** the DCC that this COF can use
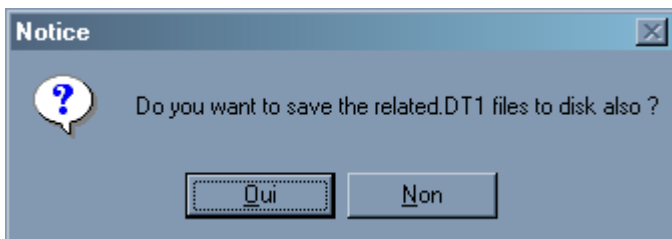
If you're viewing a DS1 you'll be prompted to save also the DT1 of that Map

When saving a file, DRTester always use its <u>Local Directory</u>. The first time, that directory doesn't exist. But right after you've save at least 1 file, you'll find a **data** directory in the directory of DRTester, and the structure in there is exactly the same as in the MPQ.

Here's an example. Here I was browsing the file **Data\Global\Chars\BA\cof\BATN1SS.cof** in DRTester, and then I saved it, with its related DCC. All the files were saved on my disk using their original structure in the MPQ, and not only the COF but also all the DCC that this COF can use were automatically saved:
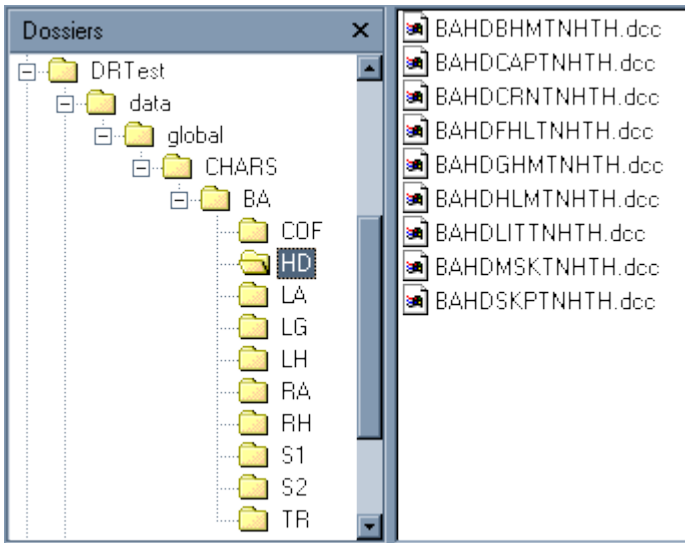
Here are the number of files that were created in all directories:

COF	:	only **1**, BATN1SS.COF
HD	:	**9**
LA	:	**3**
LG	:	**3**
LH	:	**17**
RA	:	**3**
RH	:	**17**
S1	:	**3**
S2	:	**3**
TR	:	**3**

So, in total, **62** files were created with just 1 click.

There's 1 file that is not save tough. If you're browsing the Animations of a Monster, and decide to save a COF and its related files, it will save the COF and DCC as expected, but it won't save the COF\Palshift.dat of that Monster. It's not really a problem, since you can manually select it, then save it, and it'll be at its right place in the local directory, but it's important to remember if you're planning to extract Monsters Animations: don't forget to save the Palshift.dat file too, if there's one with that Monster.

Now, I use the **File / d2data.mpq** command. Then I replace the current file name in the File Selection Box by the file **data\global\excel\armor.txt**, I select that file in the window just below, and finally I use the command **File / Save to Local File**.
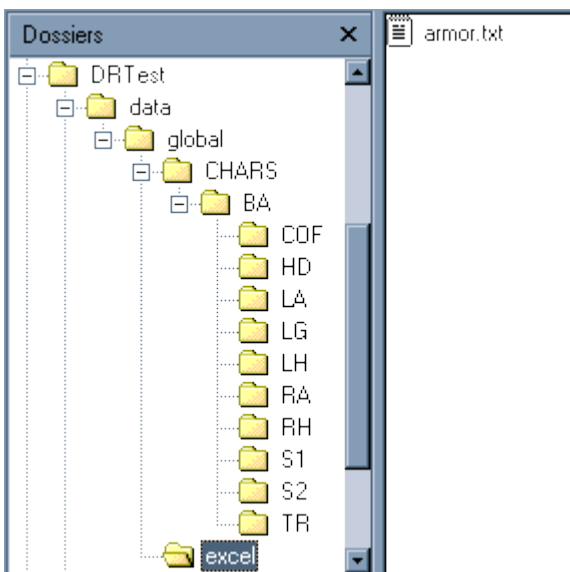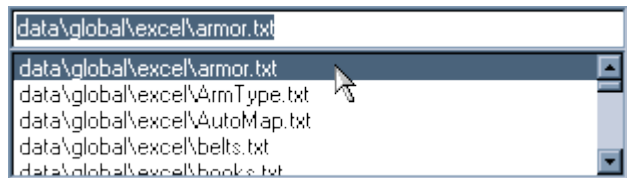
Now my Local Directory has not only the precedent COF and DCC, but it also has the file Armor.txt at the good place.

Then if I use the command **Options / Rebuild Local List**, I'll create a text file named **(listfile)** in the directory of DRTester, and it'll contain **all** the files and directories I have in my Local Directory, not only the usual Diablo II files but *any* files / directories that are under the **data** directory.

Now I'm able to use the command **File / Local Path**: it's telling DRTester to only works with the Local Directory, not a MPQ. It'll only display the files that are in that **(listfile)** text file, so you can edit it to filter the list if you want.

The commands **Options / Use List 0** to **Options / Use List 3** works the same way as the **File / Local Path** command, but while the command **File / Local Path** force DRTester to use a file named **(listfile)** to browse the **local directory**, the commands **Options / Use List n** force DRTester to use a file named **(listfile).n**, in order to see only the files you want in the corresponding MPQ: when these commands are checked, these **(listfile).n** files in your DRTester directory overrides the **(listfile)** found in the corresponding MPQ.

For instance, if you want to only see the file **Data\Global\Excel\Armor.txt** while browsing **d2exp.mpq**, you'll create a text file with just that line in it, you'll save it in the directory of DRTester under the name **(listfile).1** and you'll check the command **Options / Use List 1**. In DRTester, when browsing the **d2exp.mpq**, you'll now only see 1 file, **armor.txt**.

Note that it can also be useful to view *more* files in a MPQ: assuming there are such unknown files in a MPQ, and that you know exactly their paths & names, simply add their full paths in the **(listfile).n** files, check the corresponding command option, and you'll now be able to browse them in DRTester.

The extension numbers are following the order of the mpq in the File menu:

- patchd2.mpq   = (listfile).0
- d2exp.mpq   = (listfile).1
- d2data.mpq   = (listfile).2
- d2char.mpq   = (listfile).3

You may think the use of **(listfile).n** files are weird, but it's not. To understand the logic behind them, you must know that despite MPQ are something like ZIP archives, there's a big difference for the end-user: in a ZIP there's a list of the files, but this list is not necessary complete in the MPQ: when reading a MPQ, a program is assuming to know exactly where to find it. Some MPQ browsing programs are reading the internal **(listfile)** in the MPQ (like **CV5**), while others need an external list (like **MPQView**). So CV5 is easy to use since it don't require to have a file's list, but that means that it's depending of that internal file… and that's why you can't view all the sounds in d2sfx.mpq: all the .wav are missing from the internal file's list. On the opposite, as long as these file names are in the file's list of MPQView, you'll be able to see and extract them with this editor. DRTester is giving you both functionalities: by default it's using the internal file's list like CV5, but if you need, you can have the full control on this file's list by using external list like MPQView.

The command **Options / Update Frames** can be checked to force DRTester to update the Frame field when running an animation. On slow system, you should avoid using it, but if you decide to use it, then the Frame combo-box will automatically change its frame number when an animation is running. In the other case, this number will just stay at the last value you used (but the animation will still run). This is just an option to link the frame Number in the Frame field with the current frame that is displayed.

The command **Options / Background Color** allow you to change the… background color, hard to guess, isn't it? When viewing Characters animations, since some part can be dark, it's interesting in that case to set that color to gray, or white (or whatever else you prefer), but for some overlays that are using alpha-blending, it's sometimes more interesting to see them with a black background. Just do as you want.

2.13. Extracting the Barbarian animation files

I'll stop to explain in detail DRTester for now, to allow some practice. Let's extract all the files we'll need later to make the animation.

Launch DRTester. After some seconds, we're browsing the d2char.mpq (so no need to go into the File menu to choose another MPQ).

In the file selector, type **data\global\CHARS\BA\COF\BATN1SS.COF**, and click on that file name in the list below.

Now, just to make our job easy (but it's absolutely not necessary for the purpose of just extracting files), change the background color to White, click on the [+] button 1 time, click somewhere on the image, maintain the mouse button pressed, move the Barbarian in the middle of the image (this is know as the "drag' n drop" manipulation), and you should now have a window close to this one:
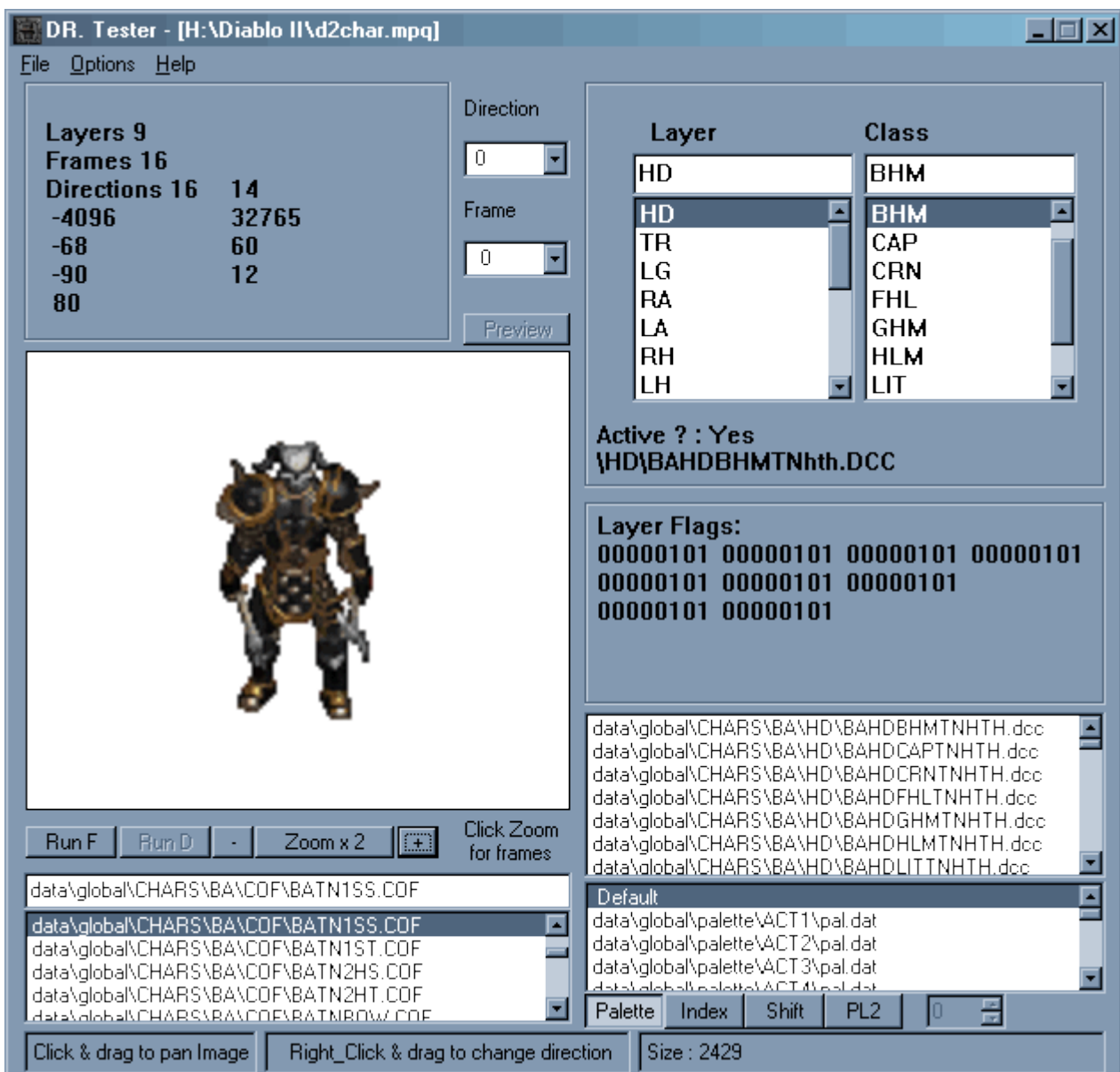
Not bad, but it's not exactly our Barbarian. So now, let's give him his good equipment.

On the upper / right corner you can see the **Layer** & **Class** windows. This is here that we'll change each layer to its own good item.

In the left window, the **HD** layer is already selected, and by chance it's the item we want (the **BHM**). So now let's proceed to the **TR** layer: click on **TR** in the left window, and chose **MED** in the right window. Immediately after having clicking on **MED**, the image is updated. Do the same for the other layers, in order to set all layers correctly:

Layer	Class
HD	BHM
TR	MED
LG	HVY
RA	HVY
LA	HVY
RH	CLM
LH	MAC
SH	None
S1	HVY
S2	HVY
S3	None
S4	None
S5	None
S6	None
S7	None
S8	None

This will give you this image:

Except for the colors, this is our Barbarian. Now, select the command **File / Save to Local File**, and you'll be prompted to save the related DCC files also. Choose YES.

We have finish with DRTester: we have extracted the COF, all the DCC of that COF, and we have check that this is the correct animation. Of course, it was a bit quick, so you can play with DRTester, and check how useful it is: play with the buttons [Zoom x 2] , [Frame = 0] , [Dir = 0] , [+] , [-] ; click on the image with the Right mouse button, maintain it pressed, and move left & right: the animation is facing other directions. Use the [Run F] button, try other items on several layers, try other COF… and when you'll be tired of it, just quit DRTester.

Note: the DCC that are saved when you're saving a COF are ALL the DCC this COF can use, not only the ones your current displayed animation is using. So you don't *have* to select all layers one by one to give them their correct item animation. But since it's a quick and easy way to find the good DCC, you should do it, at least to prepare your work with Merge_dcc, if not to be sure that the DCC you'll use are the good ones.

2.14. Configuring Merge_dcc for the Barbarian

In the directory of DRTester, go into the directory **Data\Global\CHARS**, and copy the **BA** directory into the **merge_dcc** directory:
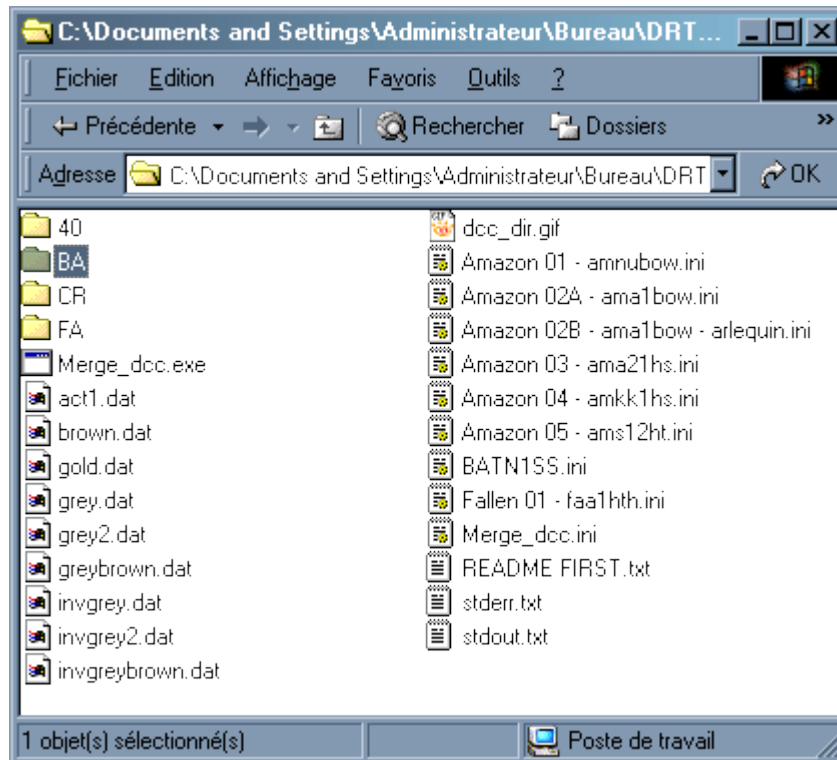
I won't go into detailed explanations (the documentation that comes with the program is there for that purpose after all), so just open the file **Merge_dcc.ini** and let's configure it for our Barbarian animation. The first 2 lines will be:

```
format=pcx
cof=BATN1SS
```

Now, let's configure each layer. First, the **HD** layer. We already know it's the **BHM** file.

```
HD=BHM
```

But this won't produce any color effect. So, let's remember what we found in the 2.3.2.1. section: colormap file is grey2.dat, and transformcolor is 12. So we complete the line as follow:

```
HD=BHM:grey2.dat:12
```

Proceed the same way for all the other layers, and the .ini will become something like this:

```
format=pcx
cof=BATN1SS
HD=BHM:grey2.dat:12
LA=HVY:grey2.dat:0
LG=HVY:grey2.dat:0
RA=HVY:grey2.dat:0
S1=HVY:grey2.dat:0
S2=HVY:grey2.dat:0
TR=MED:grey2.dat:0
RH=CLM:grey.dat:3
LH=MAC:grey.dat:7
```

Now, launch **merge_dcc.exe**, and after some seconds you'll have 256 PCX files, from **D00-(04)-F000.PCX** to **D15-(15)-F015.PCX**.

Here's an example. It's the frame 11 of the direction 2. You can easily see that the Helm has the correct color, and despite it's not as obvious, the weapons too have their correct colors.

If you wonder why on Earth the armor is not gray... it's just because the very first screenshot of that tutorial was taken while the game was in full-screen mode, and in this mode the settings of the Gamma and Contrast takes effect... and I have a dark screen, so I'm using bright settings... not mentioning that the screenshot was a JPEG, which don't helps to have an accurate image (JPEG is an image format with loss of quality)

D02-(00)-F011.PCX

But if we now compare a screenshot taken in windowed mode (with a third party program which can capture exactly the image), and the same frame made by merge_dcc... they're the same:

Accurate screenshot, in windowed mode

D00-(04)-F000.PCX, from Merge_dcc

2.15. Making the shadow of 1 frame

Now that we have all frames of the Barbarian, we can make an animated GIF. But right now I'll show you how the shadows are done in the game. To make it interesting, I'll consider that we'll make a simple Avatar for the Phrozen Keep forums. Such an Avatar is required to be at most 100 * 100 pixels, and less than 60 KB. I'll use Paint Shop Pro version 6.00, but the method is really simple to use with any other decent image editor:

| Take the frame | Fill its shape with shadows | Reduce height by 50 % | Skew it at 45° left | Done! |
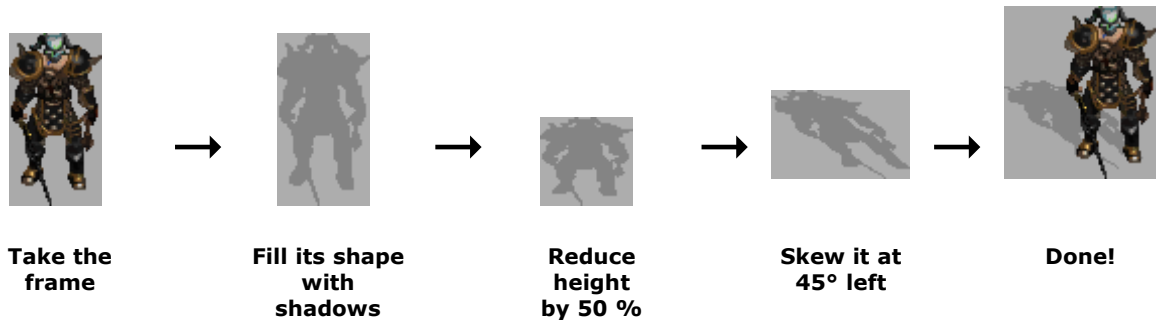
So, let's do it with Paint Shop Pro. First, open the frame you want (here the very first) in the editor. Click on the Dropper Tool icon, right-click somewhere on the background color of the image, and in the Color Palette menu left-click on the foreground color and choose the color of the shadow you like, here a mid-gray (color of index 29 in the D2 palette):

Dropper Tool icon   **Pick background color**   **Choose foreground color for shadow**
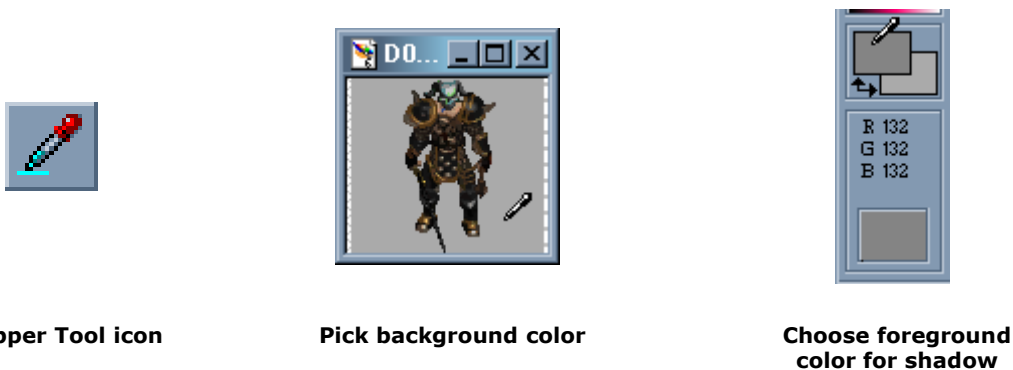
In the menu bar, choose **Window / Duplicate** (or use the **Shift + D** shortcut). This creates a new image, and starting from now we'll work only on this last one. Choose **Selection / Select All** (or use the **Ctrl + A** shortcut), this select all pixels of the image. Choose **Selection / Modify / Transparent Color** (or use the **Ctrl + T** shortcut), and choose the **Background color** in the combo box, with a tolerance of **0**, you'll now have all pixels selected except the ones in the background, therefore your selection is now the shape of the Character.

Transparent pixels are in the background color   **Shape selected**
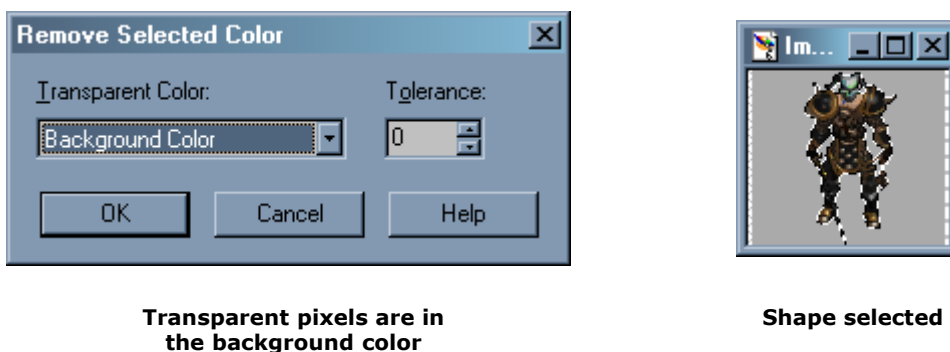
In the Color Palette menu, click on the icon with the 2 arrows, to switch the Foreground and the Background colors, press the **DEL** key, and switch back the colors. You have filled the shape of the Character with shadows.

Switch the colors   **Press DEL**   **Switch colors back**

Now, choose the menu command **Image / Resize** (or use the **Shift + S** shortcut), uncheck the Check Box "Maintain Aspect Ratio" if this one is selected, and then use a 100 % resize dimension for the width, and a 50 % resize dimension for the height.
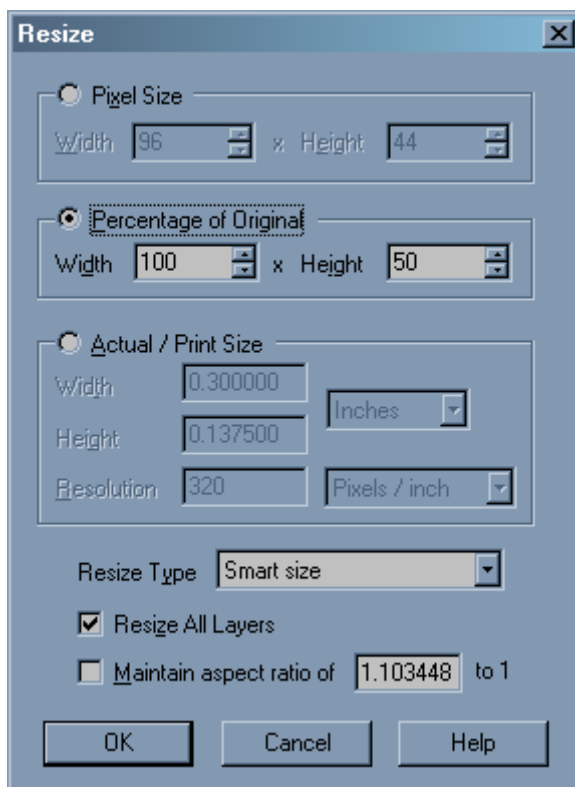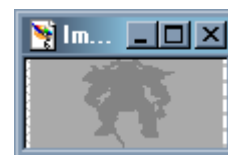
Resize 50 % height   **Shadow with good height**

We can now use the "Skew" deformation… except that this command only works in true color images, not 256 colors ones. So let's start by resampling the image to True Colors. Choose the menu command **Colors / Increase Color Depth / 16 Million Colors (24 bits)**. Nothing seems to change, but now in the bottom / right corner of Paint Shop Pro you can read that the image is in a 24-bpp mode:
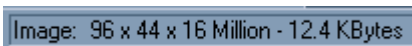
Now let's deform the image. Choose the menu command **Image / Deformations / Skew...** and in the window that appear, choose an horizontal skew of **–45** and a vertical skew of **0**.
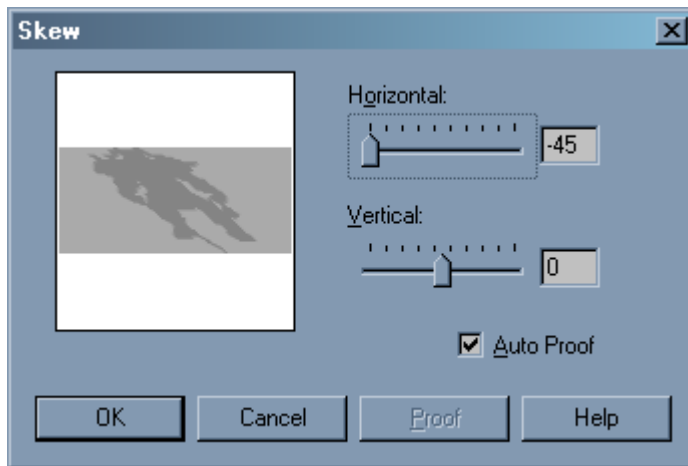
Horizontal skew of –45

Shadow like in the game

It's almost finished now. In the Background color, choose the Cyan color (Red=0, Green=255, Blue=255), create a new image of 100 * 100 pixels, in 16 Million Colors mode, with Background color, select the shape of the shadow and paste it into this image, do the same for the Body, downsample the image to 256 colors, set the Cyan color to be the transparency color, save it as a GIF, and it's done:

Pick Cyan as Background
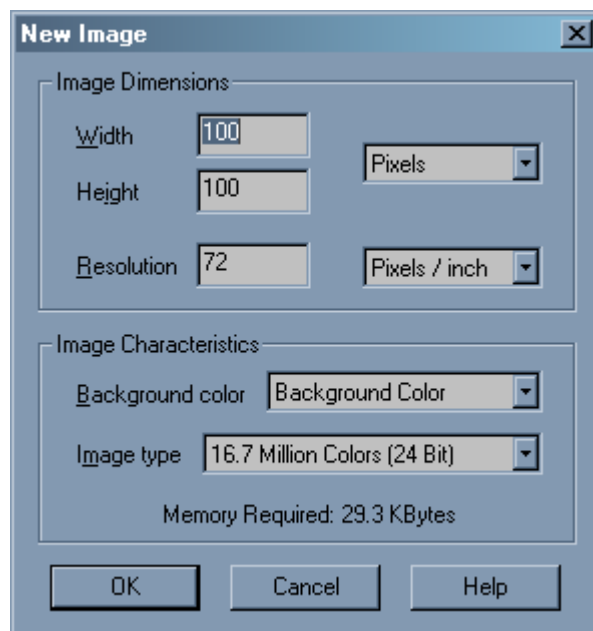
Create a new image with these settings
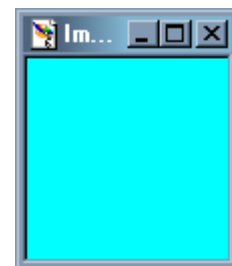
The new image, empty for now
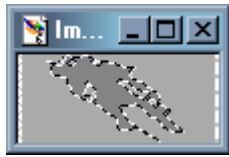
**Select the shape
of the Shadow**

**Copy / Paste as
New Selection**

**Select the shape
of the Body**
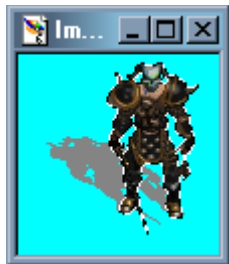
**Copy / Paste as
New Selection**
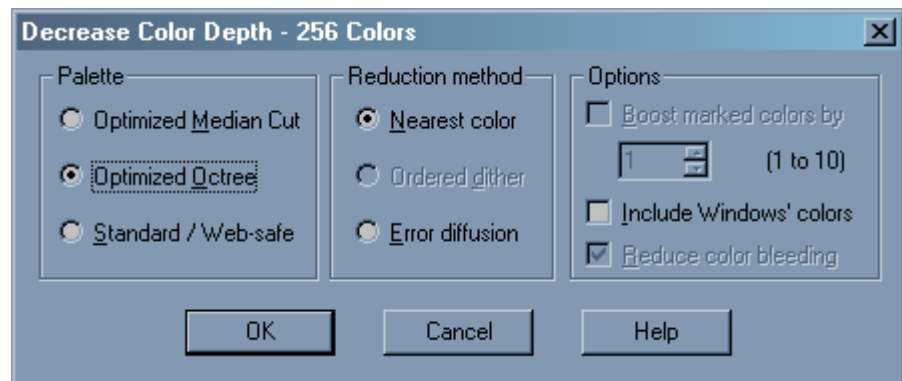
Resample the image to 256 colors
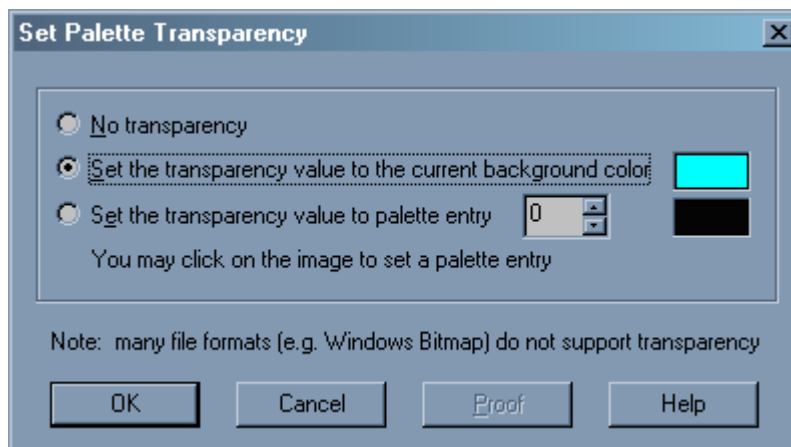
Set the Transparency to the Background color

**Save the image as a GIF, and
you have a new Avatar.**

Note: **Merge_dcc** should be able to draw the shadows for you in a future version, as it's not really difficult for this program, while it's a time-consuming operation for the end-user if he needs to make several of them, not mentioning that the shadows must be placed at the good offsets too…

For the purpose of just 1 frame, as for this Avatar example, it's ok, but I don't recommend you to plan to make an animated GIF and makes manually the shadows for EACH frames.

2.16. Finding the Barbarian Animation Speed

This time we won't make any shadow; we'll just take the frames of 1 direction and animate it at the good speed. I'll use **Animation Shop 2**, but you can use **Gamani Movie Gear** to make a GIF too (but don't ask me how, as I have never used it).

First, what will be the speed of the animation? The answer is in **Data\Global\AnimData.D2**. Open DRTester, choose the command **File / d2exp.mpq** (to have the most up-to-date AnimData.d2), select the AnimData.d2 file, and save it (**File / Save to Local File**). You can now quit DRTester.

Copy **AnimData.d2** from the **data\global** sub-directory of DRTester into the **animdata_edit** directory. Now, launch **Extract.bat** and the file **animdata.txt** is created. Open this TXT in Excel.

The first column in this tabulated TXT file is the name of a COF. The 2nd is not interesting for us, but the 3rd is the Animation Speed we want. So let's first find the COF **BATN1SS**: in Excel type **Ctrl + F**, and enter the string **BATN1SS**. It's at the **Row 3385**. Now check the 3rd column: you can read a value of **80**. But what does it means?

First, you must know that the game logic speed is hardcoded to 25 ticks per second. Now, this number in the "AnimationSpeed" column is a <u>relative</u> speed. 256 means that the animation will be displayed at 100% the rate of 25 fps, 128 will be at 50%, 64 will be at 25%, 512 will be at 200%, and so on… If you wonder why we find $1/256^{th}$ instead of %, that's because it's easier for computers to count with numbers of power of 2, but also to have an extra precision.

So our Barbarian animation has a relative $80/256^{th}$ animation speed of 25 fps (Frames Per Second). Just by curiosity, how many fps does it represents?

$$25 \text{ fps} * \mathbf{80} / 256 = 7.8125 \text{ fps}$$

To make it simple, you can consider that the game is running this animation at 8 fps. But in an Animated GIF, the speed info is not in fps. Instead, it's a delay that indicates how long a frame is displayed before the next one appears. This delay is a number of $1/100^{th}$ of a second: a delay of 100 for a frame will display it for 1 second, a number of 200 will display it for 2 seconds, 10 will display it for $1/10^{th}$ of a second, and so on… So, what will be our delay?

$$100 / (25 * \mathbf{80} * 256) = 100 / 7.8125 = 12.8$$

Our delay will then be **13** of $1/100^{th}$ of a second between each frame. As you see, our Barbarian animation will be almost the same as in the game. Not exactly, but we shouldn't notice it.

<u>Note</u>: It's important to know that this damn Microsoft Internet Explorer usually screw up the GIF animations, since it can't display animated GIF with a delay less than 10, so animated GIF quicker than 10 fps will be screwed (in AnimData.D2 such animations have an Animation Speed greater or equal to 108).

For our Barbarian it's not a problem (the delay being 13), but for other animations it can become one. I'll explain later how to counter that effect, but in short it'll require to set the GIF delay to 10, and to skip some frames here and there.

For reference, here's a table that gives the corresponding Delay between each frames and the global Animation's rate in frames per second, for all the different values of Animation Speed found in AnimData.D2:

Animation Speed	GIF Delay	Frames per second
24	42.67	2.34
32	32.00	3.13
40	25.60	3.91
48	21.33	4.69
56	18.29	5.47
64	16.00	6.25
72	14.22	7.03
80	12.80	7.81
88	11.64	8.59
96	10.67	9.38
104	9.85	10.16
112	9.14	10.94
120	8.53	11.72
128	8.00	12.50
136	7.53	13.28
144	7.11	14.06
152	6.74	14.84
160	6.40	15.63
168	6.10	16.41
176	5.82	17.19

Animation Speed	GIF Delay	Frames per second
184	5.57	17.97
188	5.45	18.36
192	5.33	18.75
200	5.12	19.53
208	4.92	20.31
216	4.74	21.09
224	4.57	21.88
232	4.41	22.66
240	4.27	23.44
248	4.13	24.22
255	4.02	24.90
256	4.00	25.00
260	3.94	25.39
272	3.76	26.56
288	3.56	28.13
336	3.05	32.81
344	2.98	33.59
376	2.72	36.72
488	2.10	47.66
512	2.00	50.00

Working with MS Internet Explorer

Now, here's how to handle the case of an animation with a speed greater than 10 fps in order to be displayed "correctly" in Microsoft Internet Explorer. Let's take the example of the **BATW1SS** animation. In **AnimData.d2** this animation has a speed of 256, meaning it'll be display at 25 fps, or if you prefer each frame should be displayed for $4/100^{th}$ of a second in an animated GIF. This COF has 8 frames. The problem is that we'll use a delay of 10 instead of 4 in the GIF, so we have to skip some frames.

First, let's make a table that tells when each frame is displayed normally, here at 25 fps with $4/100^{th}$ of a second between each frame. Then, let's take the frames that are displayed each $10/100^{th}$ of a second.

Frame	0	1	2	3	4	5	6	7
Displayed at tick	0	4	8	12	16	20	24	28

The frame **0** is always displayed. Then, $10/100^{th}$ of a second later, which frame is displayed? This is not the **3** because this one is displayed only at tick 12. The frame **2** is displayed from tick 8 to tick 11, so this is this one that we'll take for tick 10. At tick 20 that's the frame **5**. And then the animation loops.

So, if you want to make a *normal* animated GIF of the **BATW1SS** animation, you'll take frames 0 to 7 with a delay of $4/100^{th}$ of a second between each frame.

But if you want this animation to be displayed at the *apparent* good speed in MS Internet Explorer, you'll have to take only frames 0, 2 and 5, and choose a delay of $10/100^{th}$ of a second between each frame.

2.17. Creating new frames with transparency

This time we have all we need: the frames of the animation (the ones given by **Merge_dcc**), and the corresponding animation's speed. There's just a small problem if you're a perfectionist: with **Animation Shop 2** you can't simply set the transparency of the GIF to a given palette index. Since our frames are in .PCX format, no transparency is defined in the files (BMP and PCX are simple image formats which don't handle transparency at all). If we were creating the GIF right now, the animation would have a gray background. Not really a problem, but it's easy to solve, and it isn't that much works.

Note: If you're not using **Animation Shop 2** for making your animated GIF (for instance if you're using **Gamani Movie Gear**) then maybe you don't need to read that chapter, as it may be possible in your other editor to directly set the palette index 0 to be transparent. I don't know, so just do as you need.

Ok so, before we loads the frames in **Animation Shop 2**,follow these steps:

- Open Paint Shop Pro
- Drag'n Drop the 16 Barbarian frames into Paint Shop Pro (files **D00-(04)-Fnnn.PCX**)
- With the **Dropper Tool**, pick the Gray color of an image as the background color
- For each frames:
    - Set the Palette Transparency to be this gray color (**Shift + Ctrl + V** shortcut)
    - Save this modified frame as a **GIF** (**F12** shortcut)
- You can now delete the PCX if you want, as we'll only use the GIF frames
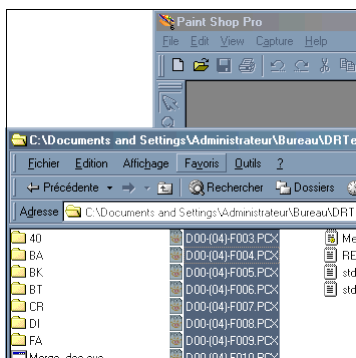
Select the PCX frames
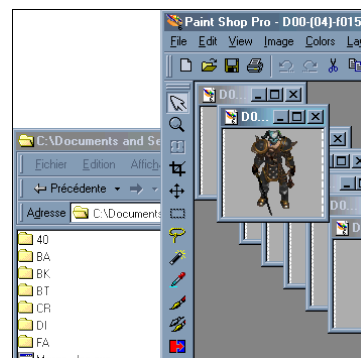
Drag'n drop them into PSP
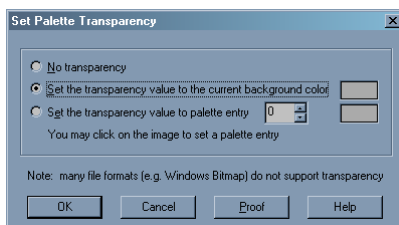
Set Transparency
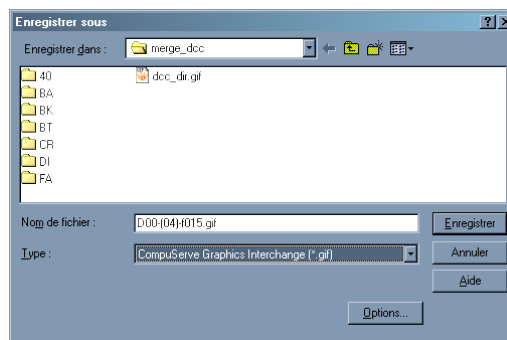
Save as... a GIF
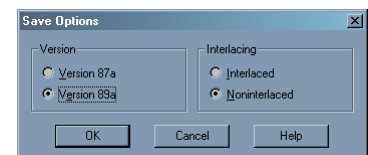
With these options

2.18. Creating the animated GIF of the Barbarian

This is really quick and easy with the help of the Wizard: open **Animation Shop 2**, click on the **Animation Wizard icon** and follow these steps:
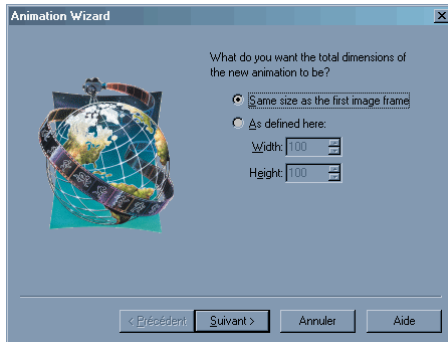
Same size as the first image
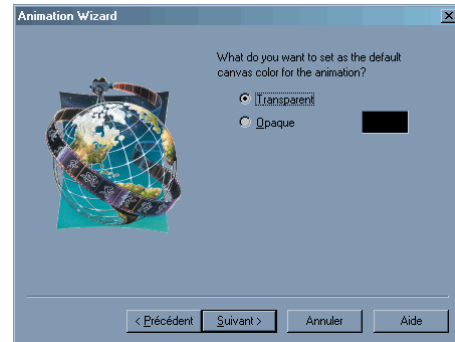
Set Background to transparent

This won't happen in our frames, so choose default options
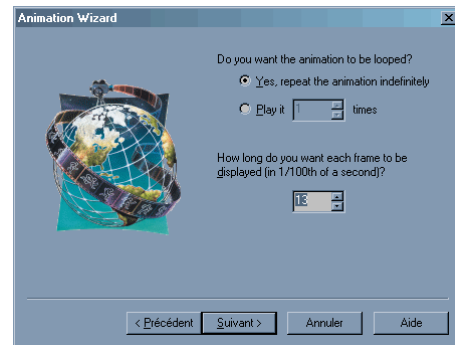
Play it indefinitely, with a delay of 13
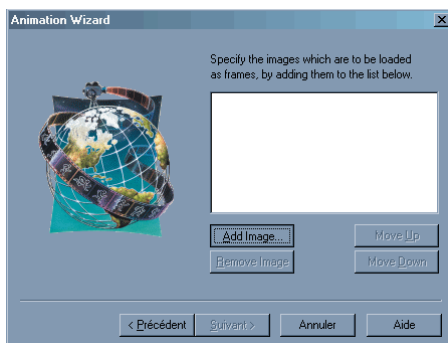
Click on the "Add images..." button
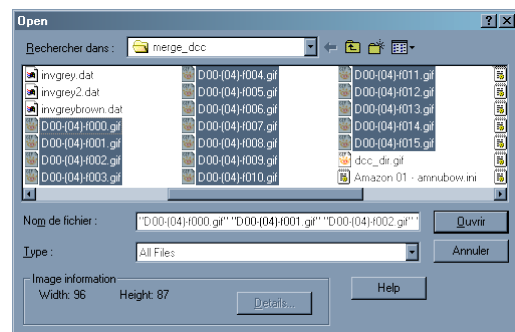
Select the GIF frames of the Barbarian

Hint: Select the files from last to first
(click on the last, then Shift + click on the first)
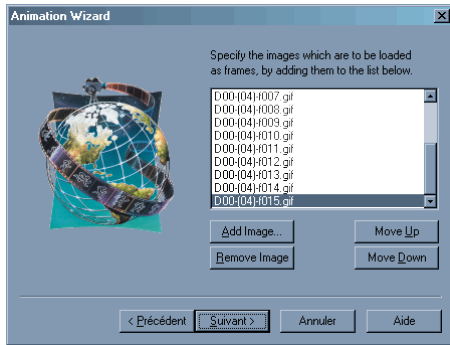
That way, no need to put back the first frame to the top here, as it sometimes happen
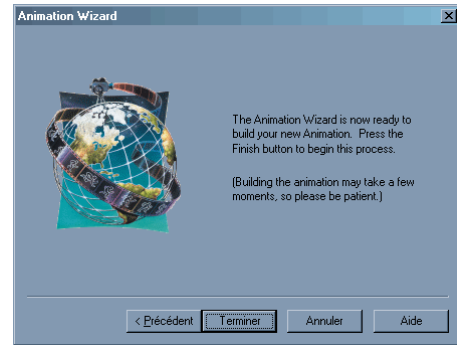
It's done

Now, click on the **View Animation icon** and you have a window looking like this:
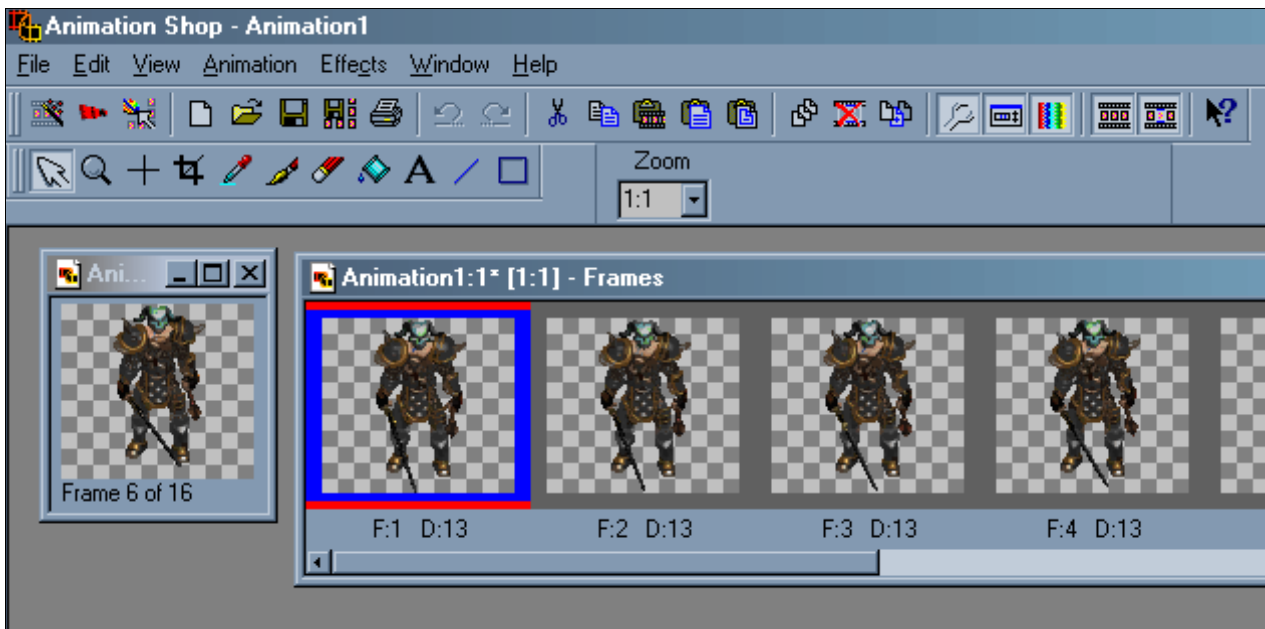
You shouldn't have any modifications to make, the background is transparent, and the delay for all frames is 13.

Since all is ok, just choose **Save** in the **File menu**, and follow these steps:
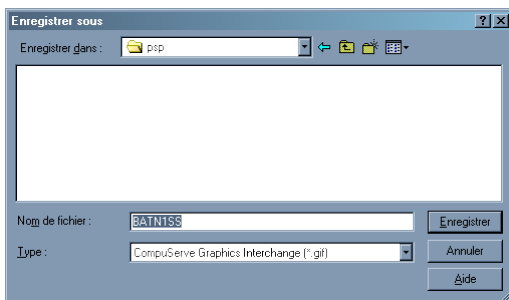
Choose a name
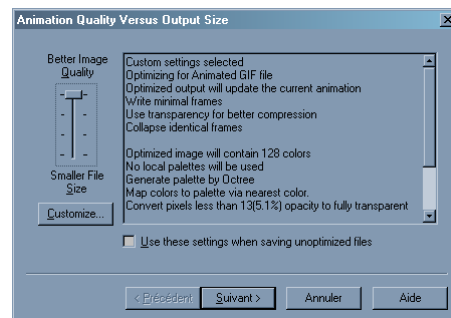
Click on the "Customize" button
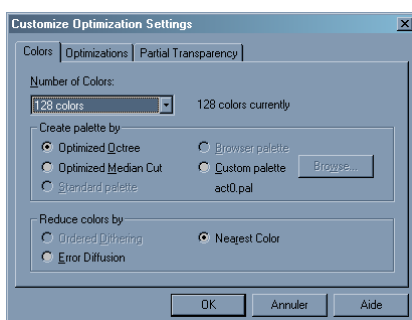
Choose your favorite color setting…
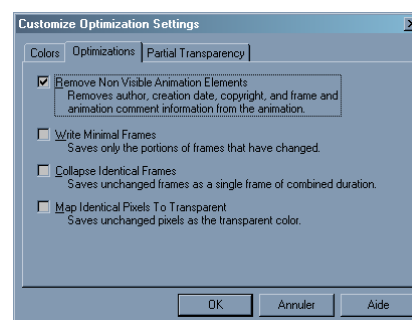
**… but if you're planning to use your GIF
with CV5, you'd better set these ones
as they're defined here**

And it's done; you finally have an animated GIF of a Barbarian exactly as he is in the B.Net channels. If you're one of that person that keep tracks of his inventory, I think you'll appreciate the possibility to add these animations. You can also make your own animated Avatar this way too, or maybe you'll import it into the Mod of another Game?

If you were planning to create an animated GIF for making new monsters animation (for instance, an Amazon with a Buriza as a replacement of a Corrupted Rogue), this is also the way you can do it, except that instead of making an animated GIF with only 1 direction you'll choose 8 directions (and you'll keep the gray background, don't make new GIF frames from the PCX with PSP, use directly the PCX). But making a new monster is another story. It's not the purpose of this tutorial as they're already others that teach how to make new monsters. I just hope that what you have learned here had helped you to understand better how the animations in Diablo II are done.

Next chapters will deals with special cases, but compare to the long way we took to make the Barbarian animation, they'll be very quick, because now you already know all that was needed to first understand the internal mechanism of D2 animations. These chapters will now just deals with some few things that weren't already explained, so they should go to the point quickly.

3. EXERCISE 2: A FALLEN

3.1. Overview

Extracting a Monster is not very different than for a Player Character, and it's even easier as Monsters have less colormaps possibilities. Like Player animations, Monsters animations are composed of the same layers, modes and weapon classes… except they just lack some.

The Fallen we'll recreate in animated GIF.
In the game, his name is "Carver"

3.2. Animation Token

In Diablo 2 version up to 1.09d, the **Token** data is stored into MonType.txt, but in version 1.10, due to the great rework of some internal systems, the Token is stored in MonStats.txt. In fact, there are so many differences that we'll study the 2 versions.

3.2.1. Animation Token in 1.09d

In this version we have to take care of a small but important thing: there's not 1 but 2 Token columns for Monsters, one in MonType.txt and the other in MonStats.txt. You have to know that the Token in MonType.txt is the one that the game uses as the base directory of all the Monster's animations (.COF, .DCC, .DC6), while the Token in MonStats.txt is used as the directory where the file PalShift.dat (color variants definitions for 1 monster) is retrieved. That's why in 1.09 Mods it's possible to easily have Fallen animations that are using Tomb Viper colormaps. This trick is not used here for our Fallens, but it was important to known.

So let's open MonType.txt in Excel. If we search the string **Fallen**, we'll find 5 of them, from "Enraged Fallen" to "Warped Fallen" (rows 21 to 25) and all of them are using a **FA** Token. **FA** is therefore our Animation Token. Note that the names in MonType.txt are not the real one used in the game.

3.2.2. Animation Token in 1.10

The 1.10 version of MonType.txt have nothing to do with its precedent structure. Here it's not a simple list of Tokens with number of direction for each mode, but rather a recursive list of type of monsters, something like in ItemTypes.txt. In this version the only thing we can know about the Fallens is that they're demons, there's nothing such as a Token column anymore.

But if we open the 1.10 version of MonStats.txt, we find again our Fallens at rows 21 to 25, and if we check the **Code** column we'll find the code **FA**, which is our Animation Token.

3.3. Finding the composition of the Carver

Now we have the Token of our Fallen, but which tint is he? If we play the game we'll see that the Carver is Blue, but how to reproduce the same effect with Merge_dcc?

First, we have to take the good PalShift.dat file. As mentioned before, the Token where to get that file is either the same as the Animations of the Monster (in 1.10), or it can be different if we're using the 1.09d version, and in this case we have to check MonStats.txt.

Whatever the version you're using, the result is still the same in the original game: the PalShift.dat to use with these Animations is in the FA Directory, the same place as where the other Animations files are (which is just all plain logic after all).

As always let's use DRTester to extract all the animations files of the Fallen we need:
- Launch DRTester
- File / d2data.mpq
- Select the file: **data\global\monsters\FA\COF\Faa1hth.cof**
- File / Save to Local File, and choose to save the .DCC as well.
- Select the file: **data\global\monsters\FA\COF\palshift.dat**
- File / Save to Local File

We now have the .COF, the .DCC and the PalShift.dat of the Fallen. Let's choose the layers we'll use. In 1.09d, the element of the monster were hardcoded, so you had to take screenshots to know what a specific Monster was using as Weapons, Shield, Head, Armor… In 1.10 not only we have the exact list of the components, but we can also change them (something we won't do here). Since the Fallens are the same in both 1.09d and 1.10, we'll use the files of that last version, to avoid the harassing work of taking screenshots, and checking all pixels.

In **1.10 MonStats.txt**, the "Carver" has *fallen2* in the column **MonStatsEx**. This means that this Fallen is using the line in **MonStats2.txt** that have the **Id** *fallen2*. Open the file MonStats2.txt in Excel and go to the row that have the **Id** set to *Fallen2* (row 22). We're interested by all the columns from **HDv** (Head Variants) to **S8v** (Special8 Variants). These columns indicate for each Layer what are all the possible choices of .DCC that the game can use when it's spawning a Monster. Here we have:

Column	Value
TRv	lit
RHv	ssd,clb,axe
SHv	nil,sml,buc,tch
S1v	lit,med
S2v	lit
S8v	lit

If we use DRTester, and test in real-time these parameters, we'll see that for the animated GIF that we'll create we'll use these layer parameters for the **FAA1HTH.COF**:

Layer	DCC
TR	LIT
RH	CLB
SH	SML
S1	MED
All others	None

As for the color variant of this Fallen, it's again easy to find in the version 1.10: it's the column **TransLvl**, which is set to *1* for our Carver. If you want to test it with DRTester, use the **TransLvl** value + 3, so with a PalShift index of 4 in our case. The difference is due to the fact that in a PalShift.dat there are 8 colormaps (from 1 to 8 in DRTester), but the game never use the 2 first, and start its numeration from the 3rd colormap, so **0** in **TransLvl** is **3** in DRTester.

3.4. Making the Carver frames with Merge_dcc

From the data directory of DRTester, copy the FA directory found under Data\Global\Monsters to the Merge_dcc Directory.

Now, open Merge_dcc.ini and edit it like this:

```
format=pcx
cof=FAA1HTH
TR=LIT:3
RH=CLB:3
SH=SML:3
S1=MED:3
```

Launch Merge_dcc.exe and you'll have all the frames of the Carver in 80 PCX images, from *D00-(04)-F000.PCX* to *D07-(03)-F009.PCX*. Just keep the 10 PCX of direction 2 (D02-*.pcx), as we just want to make a simple GIF with a Carver attacking in only 1 direction.

3.5. Making the animated GIF

Open animdata.txt (the .txt produced by Animdata_edit.exe) and search the row having FAA1HTH: you'll find an Animation Speed of 256 (25 fps, delay of 4 in a GIF).

If you want to make this animated GIF playing at the correct apparent speed in MS-Internet Explorer, then you must take only the frames 0 2 5 and 7 and use a delay of 10 (not 4) in the GIF:

Frame	0	1	2	3	4	5	6	7	8	9
Displayed at tick	0	4	8	12	16	20	24	28	32	36

You now have all the necessary informations to make our animated GIF:
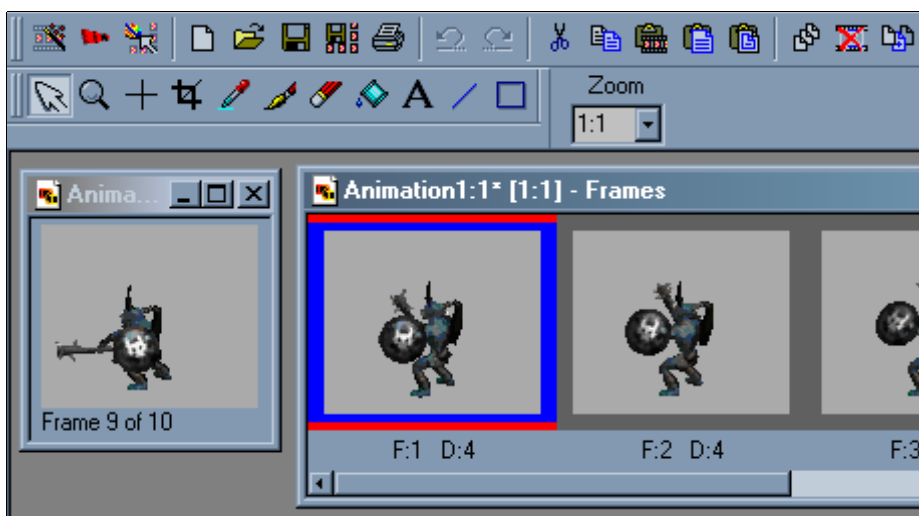
4. EXERCISE 3: THE COUNTESS

4.1. Overview

The Super unique Monsters in the patch 1.10 have a different color system than the one used in the patch 1.09D. Now they can use one of the 6 colormaps defined in the monster's Palshift.dat, or they can use one of the 22 first colormaps defined in Data\Global\Monsters\RandTransforms.dat (found in d2data.mpq). This is how the Countess appear in green in the game, while in her original palshift.dat there isn't any green colormap:

The Countess in-game	**Palshift.dat, block 3**	**Palshift.dat, block 4**	**Palshift.dat, block 5**
Palshift.dat, block 6	**Palshift.dat, block 7**	**Palshift.dat, block 8**	**RandTransforms.dat, block 9**

4.2. Finding the Countess color variations

The patch 1.10 has introduced 3 new columns in SuperUniques.txt and MonStats2.txt. They are **Utrans**, **Utrans(N)** and **Utrans(H)**. If you open SuperUniques.txt, you'll find that these 3 columns are set to **16** for the Countess line. This means that she'll use the same color variations in normal, nightmare and hell difficulties.

- ➢ Values **2 to 7** are using the Monster's Palshift.dat blocks **3 to 8**.
- ➢ Values **8 to 29** are using the RandTransforms.dat blocks **1 to 22**.
- ➢ The values **1** and **−1** means that no colormap is used, at all.
- ➢ The value **0** (or an **empty cell**) has different meanings depending of the text file:
    - o In MonStats2.txt, the game will use the Monster's Palshift.dat, block **5**.
    - o In SuperUniques.txt the game will use a random value, from **1 to 22**.
- ➢ With any other value the game will use the Monster's Palshift.dat, block **3**.

Since the value for the Countess is **16**, we'll use the RandTransforms.dat, block **9**.

Here's a screenshot of DRTester:
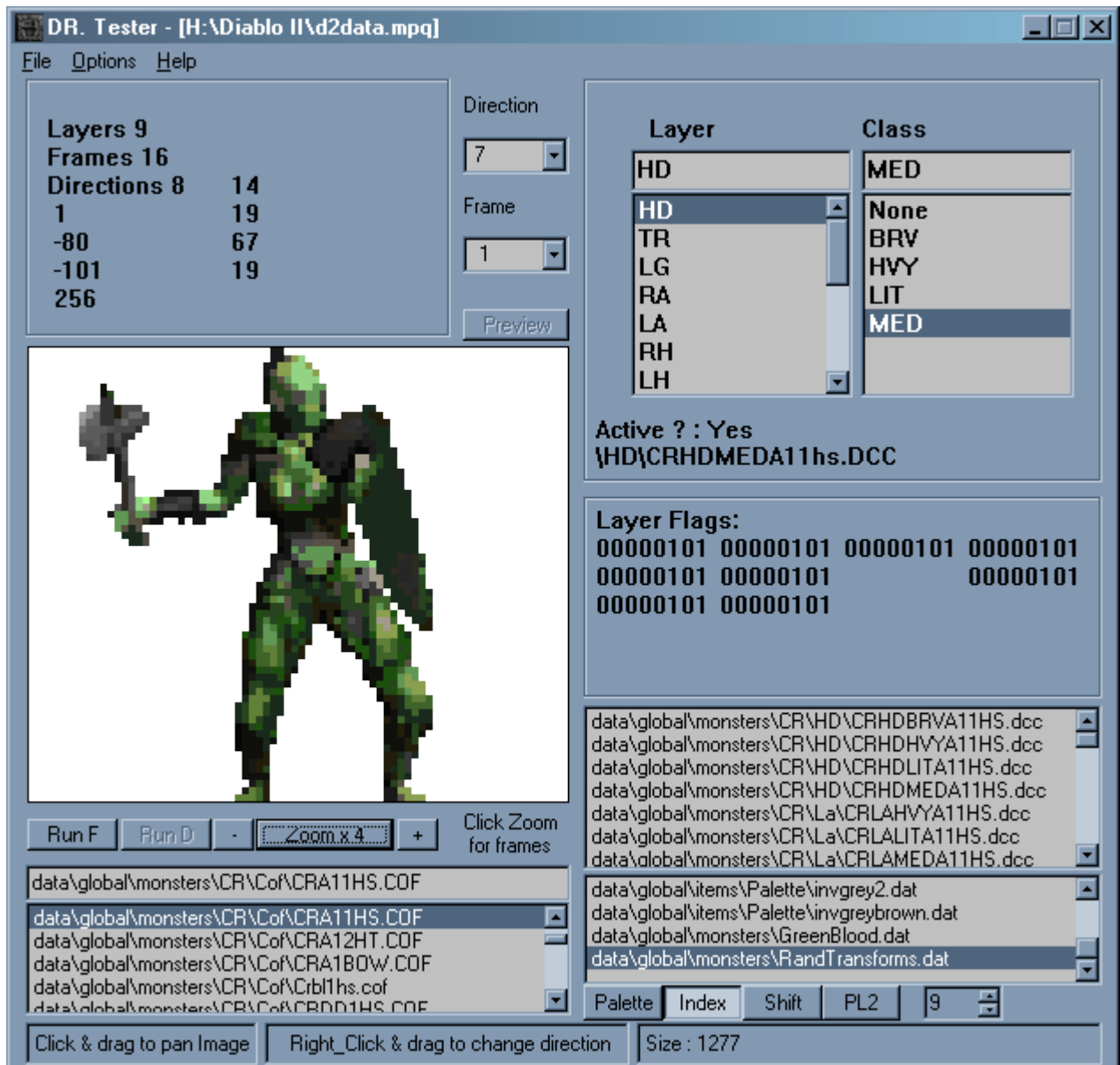
Here we have find the correct DCC to assigned to each layers, selected the RandTransforms.dat colormap file, and choose the block 9 from within. If we now compare the image we have here, and the screenshot of the Countess in-game, they're the same.

5. EXERCISE 4: ARCANE TELEPORT PAD

5.1. Overview

Objects are more or less like the other Animations. They too have layers, animation speed…
The main difference may be the fact they have their own modes. Just open **ObjMode.txt**, and
you'll find that they're using the modes Neutral, Opened, Operating, Special1… They also have
another big difference: they're not using animdata.d2! Instead, their animations informations
are in Objects.txt, but we'll see that later.

5.2. Get a necessary preview

Open DRTester. File / d2data.mpq. Select the file data\global\objects\7H\cof\7HNUHTH.COF.
File / Save to Local File, with DCC as well. You now have the COF and DCC of the Arcane
Teleport Pad. From a sub-directory of DRTester (data\global\objects of course), copy the
directory 7H, and paste it into the merge_dcc directory.

If you check in DRTester, you'll see that this Arcane Teleport Pad is composed of 3 layers (TR,
S1 and S2), all of them have only 1 possible DCC choice: LIT.

We're going to open all the different layers separately in Paint Shop Pro later, but to avoid
annoying offsets problems we'll proceed like this: open merge_dcc.ini and type this:

```
format=pcx
cof=7HNUHTH
TR=LIT
S1=LIT
S2=LIT
```

Then launch merge_dcc.exe. If you open the file D00-(00)-F000.PCX you'll have this image:
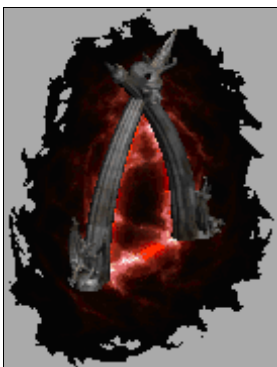
As you see that's our Arcane Teleport Pad, and it's looking
not too bad… except that it is surrounding by a black aura.
That black waves are in fact part of the red aura.

Why there's no such problem of black waves in the game?
That's because that aura layer is not draw "as-is" (which is
the case in merge_dcc), instead it use an **alpha-blending**
drawing mode. In this special mode, the luminance of the
pixel (its light level) defines its translucent level: the darker,
the more translucent; the brighter, the more visible.

We'll use Paint Shop Pro to draw that Arcane Red Portal like
it should be, on the background of our choice, by handling
the different layers blending mode like in the game.

5.3. Get all layers frames

We will now extract all layers individually. That's needed because all of them need to be draw with a different blending mode in Paint Shop Pro. But before anything else, open the file **stdout.txt** in the merge_dcc directory:

```
MERGE_DCC freeware 1.5, by Paul SIRAMY, 5 November 2002
DCC Decoder made with the precious help of Bilian BELCHEV
=========================================================
COF file               = 7HNUHTH
# directions           = 1
# frames per direction = 30
# layers               = 3
Layers list            = TR S1 S2
dcc TR to load = 7H\TR\7HTRLITNUHTH.DCC (colormap = none, block = 0)
dcc S1 to load = 7H\S1\7HS1LITNUHTH.DCC (colormap = none, block = 0)
dcc S2 to load = 7H\S2\7HS2LITNUHTH.DCC (colormap = none, block = 0)
box = (-76, -131) - (60, 49) = 137 * 181 pixels
```

What is important here is the BOX definition. Merge_dcc took all the 3 layers of the animation, placed them all on top of each others (respecting their own offsets), and kept the smallest box that enclose all pixels, without any border. We'll use the same BOX parameters for all the 3 different layers, that way we'll have images that are not only the exact same size, but that can also be place on top of each other without wondering about their offsets.

For the **TR layer**, open merge_dcc.ini and edit it like that:

```
format=pcx
cof=7HNUHTH
BOX=-76,-131,60,49
TR=LIT
```

Now launch Merge_dcc.exe. The files D00-(00)-F000.PCX to D00-(00)-F029.PCX are now all the stone layer, without any aura, like this one:

**TR layer is the Stone part of
the Arcane Teleport Pad**

Create a directory somewhere on your hard disk ("Arcane Teleport Pad" for instance), create a sub-directory inside ("TR") and place these 'stone' .pcx right there.

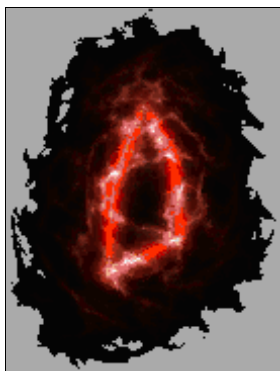Now edit merge_dcc.ini in order to extract the **S1 layer** the same way:

```
format=pcx
cof=7HNUHTH
BOX=-76,-131,60,49
S1=LIT
```

Create a "S1" directory in your "Arcane Teleport Pad" directory and place the 'red aura' .pcx there.

Do the same for the **S2 layer**:

```
format=pcx
cof=7HNUHTH
BOX=-76,-131,60,49
S2=LIT
```

Create a "S2" directory in your "Arcane Teleport Pad" directory and place the 'mask' .pcx there.
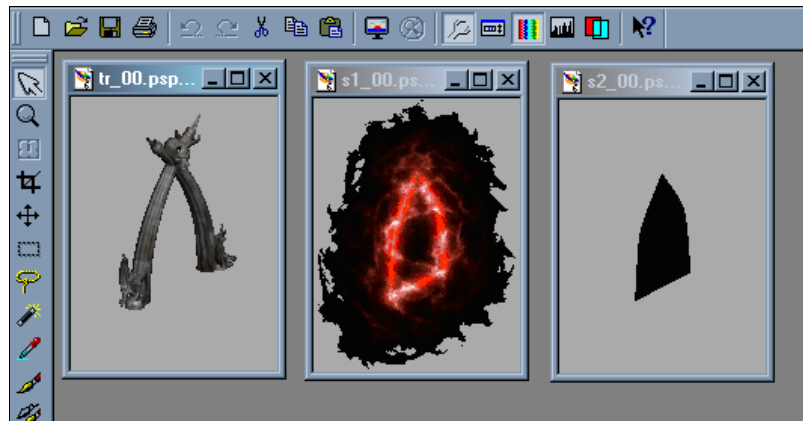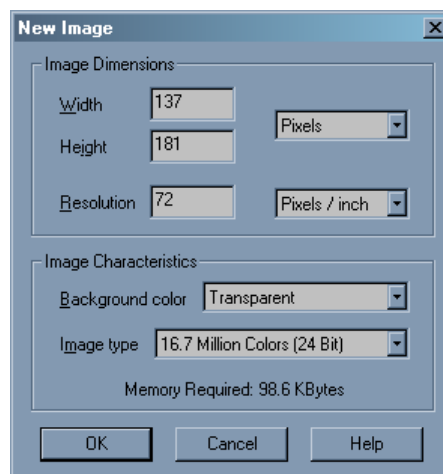

S1 layer is the Red Aura

S2 layer is a Mask

5.4. Handle layers in Paint Shop Pro

Open the first frames of all the 3 layers together in Paint Shop Pro:

If you check, you'll see that these 3 images are exactly the same size: 137 x 181 pixels, all in a 256 colors mode. We'll now create our own background. Choose the command **File / New**, and enter these parameters:

Then, in your Foreground color pick Blue, and for your Background color pick Black. Take the Flood Fill tool, Toggle Tools Options Window and set it like that:
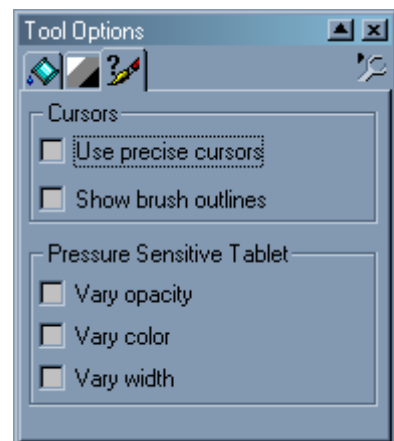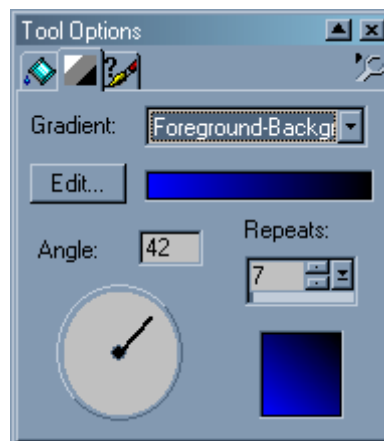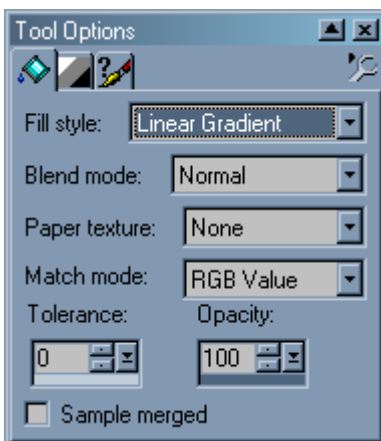
You can now fill the new image you have created with gradients of Blue and Black:

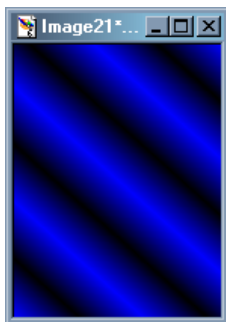This will be our own background, where we will draw an Arcane Teleport Pad that will looks like in the game. Of course you can choose any other background, like a portion of a web page where the animation will be placed, it's up to you, and the layer drawing manipulations will still be the same anyway.

Now, we'll place the 3 animation layers on top of this background, one after the other. Select that background image, then choose the menu **Layers / properties…** and in the window that appear, change the name of the background layer from **Layer1** to **Back, Blue-Black**:
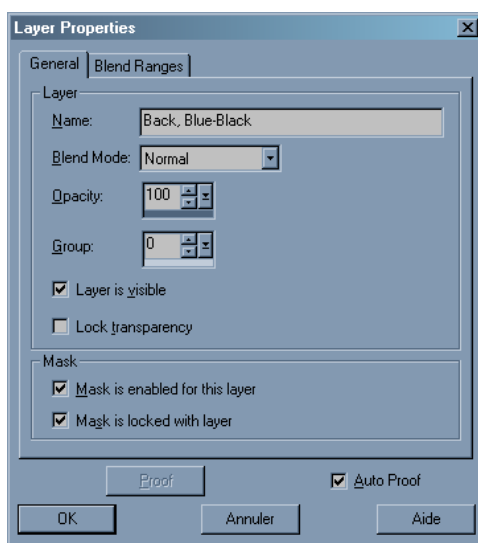
This is not necessary, but it's better to have descriptive name when working with several layers at once.

We'll now place the Red Aura layer on top of that blue-dark background. Select the red aura image, and choose the menu **Colors / Edit palette…** Then replace the Palette color index 0 from gray to black, and press OK: all the gray pixels are transformed into black. **Edit / Copy**, select the background window, and **Edit / Paste / As New Layer**: the blue-black background is now completely hidden by the red aura. Toggle the **Layer Palette**, and for the Layer1 (that is already selected), choose the **Screen** Blending mode. Rename that Layer to **Red Aura**, and we're done for this layer:
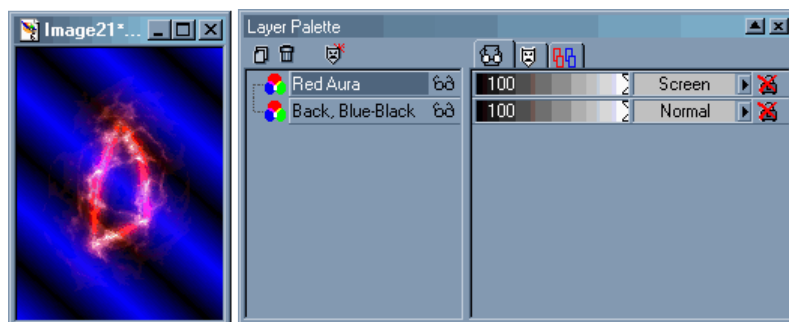
We'll now place the Stone layer. But we'll use a small trick to help us. Select the Stone layer window. Select a foreground color that is not the Palette index 0 (like white, index 255), zoom several time in the image, and place a white pixel at the exact upper / left corner, and do the same at the opposite (bottom / right corner). You can zoom back to 1:1.
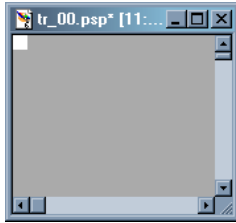
Upper / Left white dot
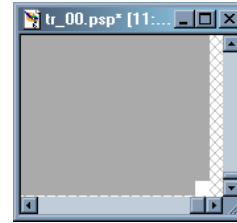
Bottom / Right white dot

Click on the Background color, choose the Palette index 0 (gray) and press OK. **Selection / Select All**, **Selections / Modify / Transparent Colors**, chooses *Background color* in the menu, *tolerance 0*, and press OK: all the pixels of the stone are selected including the 2 white dots (tough not obvious). **Edit / Copy**, select the background window, **Edit / Paste as New Layer**, and the stone pixels are placed exactly where expected (that's because of the 2 white dots). Change the name of this layer to be **Stone**, and it's done for that layer:
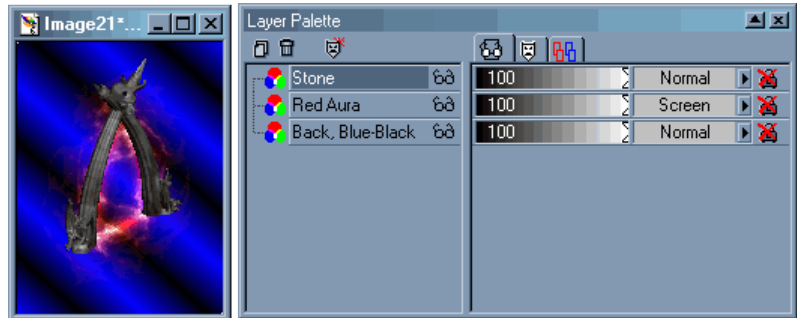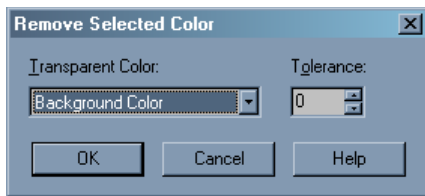
Proceed the same way for the Mask layer (2 white dots, select all but gray pixels, copy / paste as new layer), name this new layer **Mask**, make this layer having only a **25% visibility**, move the Mask layer from the top to the place between the **Red Aura** layer and the **Back, Blue-Black** layer (by drag'n drop the **Mask** layer field in the Layer Palette window), and it's done for that layer:
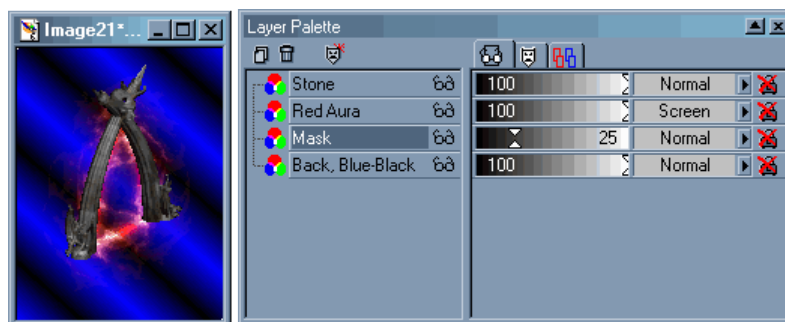

This will be enough for that Tutorial, but if you want to make an Animation, you have of course to proceed the same way for all the frames of the Teleport Pad Animation (yep, that's some work). If you want to convert a layered image to a normal one, choose the command **Layers / Merge / Merge All (Flatten)**. Ha, a last note: don't forget to remove the white dots 🙂

5.5. Finding Layers Blending Mode

Here we have set the Red Aura layer to use a 'screen' blending mode, and the Mask layer to use a 75% translucent blending mode… but how do we know how to draw each layer in Diablo 2?

In DRTester, pick the file **data\global\objects\7h\cof\7hNUHTH.COF**:
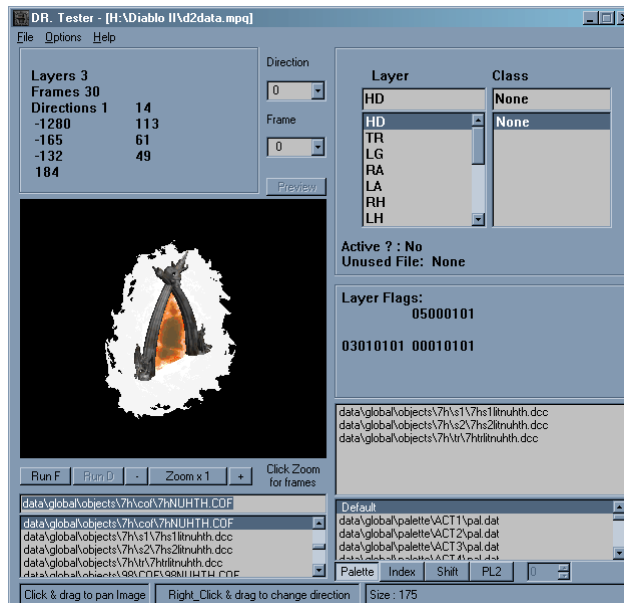
Well, it's not beautiful, and you may think it's even buggy, but not for long. Click on the Bottom / Right 'PL2' button, and choose an Act Palette (Act 1 will be fine).

Anyway, that's on this screen that we can learn how to draw each layer. On the middle / right border of the window, we have **Layer Flags**:

    TR = 05000101
    S1 = 03010101
    S2 = 00010101

The 2nd byte indicates if a blender mode is used. 00 means normal drawing, non-zero (01 usually) means the corresponding layer will be draw with a specific blender mode. In that case, that's the 1st byte that tells which effect will be use, following this table:

    00 = 75 % translucency
    01 = 50 % translucency
    02 = 25 % translucency
    03 = screen
    04 = luminance
    06 = not very well known, but it's some kind of bright screen.

So for the Teleport Pad we have:

    TR = 05 00 = normal drawing mode
    S1 = 03 01 = screen blending mode
    S2 = 00 01 = 75 % translucency blending mode

5.6. Animation Speed of Objects

Having the final frames of the Arcane Teleport Pad animation is good, but not enough. What is its animation speed? As said before, Objects are not using **animdata.d2**, but rather **Objects.txt**. In this file we find many columns, but for us, only the **FrameDelta0** to **FrameDelta7** are interesting.

If you open **ObjMode.txt**, you'll notice there are 8 modes:

Name	Token	Index
Neutral	NU	0
Operating	OP	1
Opened	ON	2
Special1	S1	3
Special2	S2	4
Special3	S3	5
Special4	S4	6
Special5	S5	7

If we count them starting from 0, the last mode is index 7. So it's now obvious that FrameDelta0 is used for the NU mode, and FrameDelta7 for the S5 mode. Our Teleport Pad we have made was using the NU mode, so we'll look the value that's in the FrameDelta0 column, which is 256. Exactly like for the Animation Speed data in Animdata.d2, this speed is a relative speed (in 256th) of 25 fps, so here the Teleport Pad is draw at a rate of 25 fps in the game.

Note that animations of Objects are a bit more complex than the Player Characters and Monsters ones, because in Objects.txt we can find the columns FrameCnt0-7, CycleAnim0-7 and Start0-7, which are used for some animations like Tree of Inifuss: despite the COFs of that Tree are made with 2 frames per direction, the DCC have only 1 frame. So if you try to play that 'animation' in DRTester, the Tree blinks. If you now check the FrameCnt columns for the Tree of Inifuss, you'll see that they have only 1 frame per direction there. Datas in Objects.txt take precedence against the COFs datas.

And that conclude the process of extracting Diablo II animations, with all its complexity of layers, modes, animation speeds, drawing modes… You should now have all the necessary knowledge to extract any animation from that great game. I hope it was interesting 😊

Paul SIRAMY