Sprite

COLLABORATORS

	TITLE : Sprite	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 23, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sprite

1.1 Sprite V1.00

```
PureBasic - Sprite Library V1.00

  'Sprites' sind allen Spieleprogrammierern wohlbekannt. Dies sind
  kleine Bilder, manchmal auch 'Brushes' (Pinsel) genannt, welche an
  jeder Position des Bildschirms angezeigt werden können. Die Sprites
  können über Grafiken bewegt werden, ohne diese zu zerstören. Nicht
  wie andere "zerstörerische" Grafikoperationen. PureBasic Sprites
  basieren komplett auf dem Blitter, einem sehr schnellen Hardware
  Chip, welcher schnell Unmengen von Daten bewegen kann. Diese Library
  ist stark optimiert und hat eine Besonderheit: sie arbeitet parallel
  mit dem Hauptprozessor, der 680x0 CPU. So können Sie einige Sprites
  anzeigen und die CPU für andere Aufgaben, wie künstliche Intelligenz,
  benutzen. Alle diese Funktionen sind 100% systemfreundlich und Sie
  können sie für jede Art von Programm benutzen.

Befehlsübersicht in alphabetischer Reihenfolge:

   AddBlockSprite
   AddBufferedSprite
   AddSprite
   CreateSpriteBuffer
   FreeSprite
   FreeSpriteBuffer
   InitSprite
   LoadSprite
   ResetSpriteServer
   RestoreBackground
   SpriteDepth
   SpriteHeight
   SpriteWidth
   StartSpriteServer
   StopSpriteServer
   UseSpriteBuffer
   WaitSpriteServer

Beispiel:
```

```
    Sprite example
```

1.2 addblocksprite

```
  SYNTAX
AddBlockSprite(#Sprite, x, y)

  COMMAND
Zeigt das Sprite an der angegebenen Position auf dem aktuellen Sprite-
Buffer an. Diese Funktion ist die schnellste Möglichkeit, ein Sprite
auf dem Bildschirm anzuzeigen, hat aber einige Einschränkungen:

  + Die Sprite-Breite muß ein Mehrfaches von 16 sein
    (z.B.: 16, 32, 48, 64...)
  + Die Position 'x' muß ein Mehrfaches von 16 sein.
  + Es gibt keine transparente Farbe für dieses Sprite.


Hinweis: Wird gerade ein anderes Sprite angezeigt, fügt diese Funktion
es zur Warteschlange hinzu und kehrt umgehend zurück. Sie können niemals
davon ausgehen, dass das Sprite tatsächlich angezeigt wird, wenn die
Funktion zurückkehrt! Benutzen Sie den Befehl 'WaitSpriteServer()',
wenn Sie sichergehen wollen, dass das Sprite wirklich angezeigt wird.
Sehen Sie sich die Beschreibung zu 'StartSpriteServer()' an, um weitere
Informationen über diesen asynchronen Prozeß zu erhalten.
```

1.3 addbufferedsprite

```
  SYNTAX
AddBufferedSprite(#Sprite, x, y)

  COMMAND
Zeigt das Sprite an der angegebenen Position auf dem aktuellen Sprite-
Buffer an. Bevor das Sprite angezeigt und damit der Hintergrund zerstört
wird, wird der hinter dem Sprite liegende Hintergrund im Sprite-Buffer
gesichert. Sie müssen etwas Speicher reservieren, um die Hintergrund-
Grafikdaten speichern zu können (dies erfolgt via 'CreateSpriteBuffer()').
Der gespeicherte Hintergrund kann später mittels 'RestoreBackground()'
wiederhergestellt werden. Die Farbe 0 des Sprites wird als transparent
betrachtet. Dieser Befehl wird nicht ge-'clipped'; seien Sie also sicher,
das Sprite innerhalb der BitMap darzustellen.

Hinweis: Wird gerade ein anderes Sprite angezeigt, fügt diese Funktion
es zur Warteschlange hinzu und kehrt umgehend zurück. Sie können niemals
davon ausgehen, dass das Sprite tatsächlich angezeigt wird, wenn die
Funktion zurückkehrt! Benutzen Sie den Befehl 'WaitSpriteServer()',
wenn Sie sichergehen wollen, dass das Sprite wirklich angezeigt wird.
Sehen Sie sich die Beschreibung zu 'StartSpriteServer()' an, um weitere
Informationen über diesen asynchronen Prozeß zu erhalten.
```

1.4 addsprite

```
  SYNTAX
AddSprite(#Sprite, x, y)

  COMMAND
```

Zeigt das Sprite an der angegebenen Position auf dem aktuellen Sprite-Buffer an. Der Hintergrund-Ausschnitt wird durch diese Funktion zerstört. Wenn Sie den Hintergrund erhalten möchten, benutzen Sie stattdessen 'AddBufferedSprite()'. Die Farbe 0 des Sprites wird als transparent betrachtet. Dieser Befehl wird nicht ge-'clipped'; seien Sie also sicher, das Sprite innerhalb der BitMap darzustellen.

Hinweis: Wird gerade ein anderes Sprite angezeigt, fügt diese Funktion es zur Warteschlange hinzu und kehrt umgehend zurück. Sie können niemals davon ausgehen, dass das Sprite tatsächlich angezeigt wird, wenn die Funktion zurückkehrt! Benutzen Sie den Befehl 'WaitSpriteServer()', wenn Sie sichergehen wollen, dass das Sprite wirklich angezeigt wird. Sehen Sie sich die Beschreibung zu 'StartSpriteServer()' an, um weitere Informationen über diesen asynchronen Prozeß zu erhalten.

1.5 createspritebuffer

```
  SYNTAX
CreateSpriteBuffer(#SpriteBuffer, Size, BitMapID)

  FUNCTION
```

Erstellt und initialisiert einen neuen Sprite-Buffer. 'BitMapID' ist die Nummer der BitMap, auf welcher Sie die Sprites darstellen wollen. 'Size' (Größe) ist nur sinnvoll, wenn Sie einige Sprites mit 'AddBufferedSprite()' darstellen möchten, wobei der Hintergrund gesichert wird. Die Größe errechnet sich aus der Anzahl der gleichzeitig dargestellten Sprites, ihren Größen und der BitMap-Tiefe. Um sie zu errechnen, können Sie folgende Regel benutzen:

```
  Size = (BitMapDepth * (SpriteWidth*SpriteHeight)*NumberOfSpriteDisplayed) / 8  ↩
     + 1000
```

Dieser neu erstellte Sprite-Buffer wird dann der aktuelle Sprite-Buffer.

1.6 freesprite

```
  SYNTAX
FreeSprite(#Sprite)

  FUNCTION
```

Entfernt das angegebene Sprite aus dem Speicher. Sie können es anschließend nicht mehr benutzen.

1.7 freespritebuffer

```
  SYNTAX
FreeSpriteBuffer(#SpriteBuffer)

  FUNCTION
Gibt den angegebenen Sprite-Buffer und allen hierfür reservierten Speicher
frei.
```

1.8 initsprite

```
  SYNTAX
Result = InitSpriteFile(#MaxDisplayedSprites, #MaxSpriteBuffers, #MaxSprites)

  FUNCTION
Initialisiert die gesamte Sprite-Umgebung zur späteren Benutzung. Sie müssen
diese Funktion am Anfang Ihres Sourcecodes einfügen, wenn Sie die Sprite-
Funktionen benutzen möchten. Sie können das Ergebnis ('result') testen,
um die korrekte Initialisierung der Sprite-Umgebung zu überprüfen. War diese
nicht erfolgreich, beenden Sie das Programm oder deaktivieren Sie alle
Aufrufe von Sprite-relevanten Befehlen.
```

1.9 loadsprite

```
  SYNTAX
Result = LoadSprite(#Sprite, FileName$)

  STATEMENT
Lädt das angegebene Sprite in den Speicher zur sofortigen Verwendung. Das
Sprite muß im IFF/ILBM Format (komprimiert oder nicht, beide Möglichkeiten
werden unterstützt) vorliegen. Ging beim Laden etwas schief, wird ein
negativer Wert zurückgegeben. Anderfalls, alles ist gut...
Hier die möglichen Werte für 'Result':

  -1 : Datei nicht gefunden oder konnte nicht geöffnet werden.
  -2 : Diese Datei ist keine IFF/ILBM Datei.
  -3 : Nicht genug Speicher.
  -4 : Korrupte IFF/ILBM Datei.
```

1.10 resetspriteserver

```
  SYNTAX
ResetSpriteServer()

  STATEMENT
Wenn Sie es einmal geschafft haben, alle benötigten Sprites anzuzeigen,
müssen Sie den Sprite-Server (Warteschlange) zurücksetzen, um dem System
mitzuteilen, dass alles in Ordnung ist. Intern wird die Warteschlange auf
0 zurückgesetzt, damit werden alle noch nicht dargestellten Sprites auch
```

```
nie mehr angezeigt. Benutzen Sie die 'WaitSpriteServer()' Funktion, um
sicherzugehen, dass der Sprite-Server fertig ist. Eine gute Lösung ist,
den Sprite-Server bei jedem Frame (Einzelbild) zurückzusetzen (natürlich
während einem Spiel).
```

1.11 restorebackground

```
  SYNTAX
RestoreBackground()

  STATEMENT
Stellt den zuvor zerstörten Hintergrund des aktuellen Sprite-Buffers wieder
her. Jeder Sprite-Buffer hat seinen eigenen Hintergrund-Ausschnitt. Der
Hintergrund wird mit der Funktion 'AddBufferedSprite()' gespeichert.
```

1.12 spritedepth

```
  SYNTAX
Result.w = SpriteDepth(#Sprite)

  STATEMENT
Gibt die Tiefe des angegebenen Sprites zurück.
```

1.13 spriteheight

```
  SYNTAX
Result.w = SpriteHeight(#Sprite)

  STATEMENT
Gibt die Höhe in Pixel des angegebenen Sprite zurück.
```

1.14 spritewidth

```
  SYNTAX
Result.w = SpriteWidth(#Sprite)

  STATEMENT
Gibt die Breite in Pixel des angegebenen Sprite zurück.
```

1.15 startspriteserver

```
  SYNTAX
StartSpriteServer()

  STATEMENT
```

Reserviert die 'Blitter'-Chip - Ressourcen auf einem systemfreundlichen Weg und initialisiert den Server zur späteren Benutzung. Wenn Sie diese Funktion einmal aufgerufen haben, können Sie beruhigt alle der AddSprite() Funktionen benutzen. Bitte beachten Sie aber, dass der Blitter komplett durch Ihr Programm übernommen wird und das ganze System ihn dann nicht benutzen kann (zum Zeichnen der Fenster, von Text...). Die Grafiken werden eingefroren, vergessen Sie also nicht, den Server zu stoppen, sobald Sie keine der Sprite-Funktionen mehr benötigen. Eine gute Idee ist, den Server zu jedem Frame zu starten/stoppen, damit sichern Sie ein Minimum an Rechenkapazität für das Weiterarbeiten vom OS. Diese Funktion ist natürlich sehr schnell.

Einige weitere Anmerkungen über den Sprite-Server (für fortgeschrittene User):

Der Server kann als eine Warteliste angesehen werden und wenn Sie ein Sprite zur Liste hinzufügen gibt es zwei Möglichkeiten: die Liste ist leer, das Sprite wird dann unverzüglich dargestellt; oder die Liste hat bereits einige Einträge, dann wird Ihr Sprite am Ende der Liste hinzugefügt. Alle Einträge werden einer nach dem anderen abgearbeitet, in der richtigen Reihenfolge. Der Vorteil dieses Systems ist, dass der Hauptprozessor (CPU) nicht darauf warten muß, dass der 'Blitter' seine Arbeit abgeschlossen hat. So gewinnen wir beträchtliche Rechen-Power gegenüber klassischen Lösungen. Um sicherzugehen, dass alle Ihre Sprites dargestellt wurden, benutzen Sie einfach den 'WaitSpriteServer()' Befehl.

1.16 stopspriteserver

SYNTAX
StopSpriteServer()

STATEMENT
Stoppt die Aktivität des Sprite-Server und gibt umgehend den 'Blitter' Chip an das AmigaOS frei. Wurden einige Sprites noch nicht dargestellt, sind diese verloren. Dieser Befehl ist sehr schnell.

1.17 usespritebuffer

SYNTAX
UseSpriteBuffer(#SpriteBuffer)

STATEMENT
Tauscht den aktuellen Sprite-Buffer durch den angegebenen aus.

1.18 waitspriteserver

SYNTAX
WaitSpriteServer()

STATEMENT
Wartet bis der Sprite-Server mit der Darstellung aller Sprites fertig geworden ist. Dieser Befehl wird benötigt, außer wenn Sie sicher sind,

dass alle Ihre Sprites bereits angezeigt wurden. Dieser Befehl sollte typischerweise am Ende der Hauptroutine aufgerufen werden, wenn alle anderen 'CPU'-only (nur den Hauptprozessor betreffenden) Befehle abgearbeitet wurden. Dies ist einer der Vorteile der parallelen Arbeitsweise: Sie können die CPU nutzen, während das Sprite dargestellt wird.