True Color CP437

A True Color Console for Unity3D

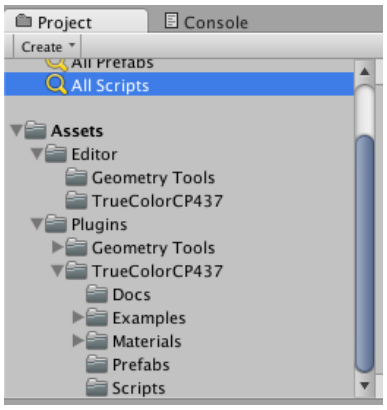
Contents

1 Creating and Configuring the Console

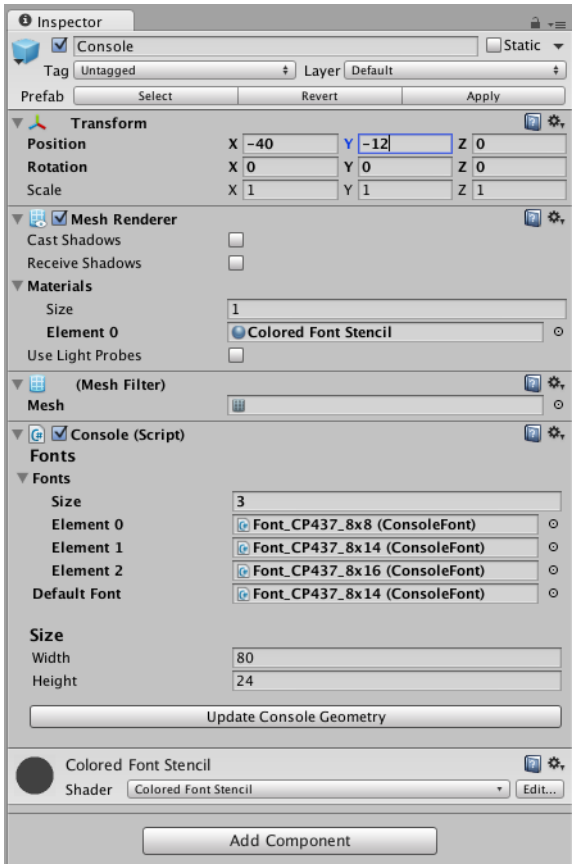
1.1 Importing the Package and Creating a fresh Console

Choose the **Assets -> Import Package -> Custom Package…** menu item and select the directory where you saved the TrueColorCP437 package. Make sure that all boxes in front of each file in the **Items to Import** window is checked and click Import.

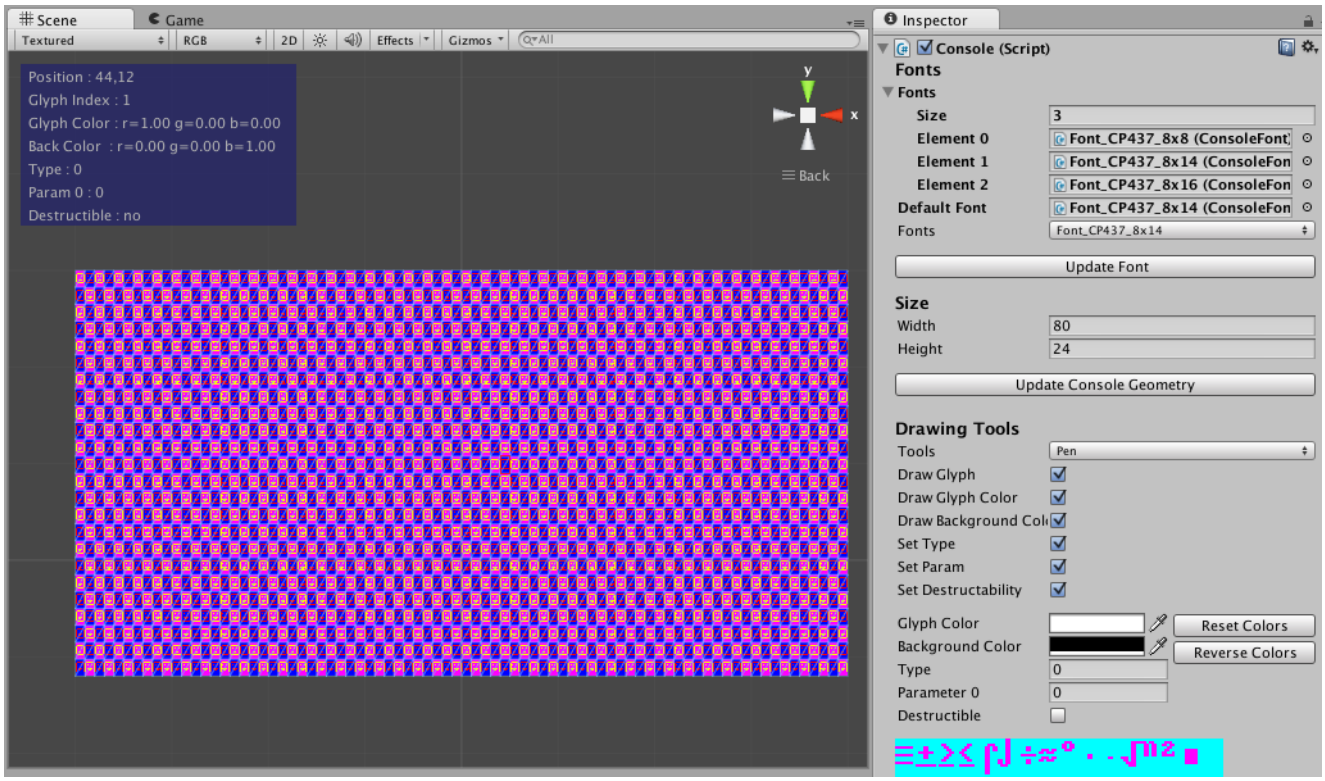
Just after importing the package in a fresh project, your Project view should look like this :



Unfold folders **Plugins -> TrueColorCP437 -> Prefabs** and drag an drop the **Console** prefab to the Scene or the Hierarchy view. The console is not yet visible and need to be configured. Select the console in the Hierarchy view and fill its Inspector as follow (all the fonts are in the **Prefabs** folder):



Push the **Update Console Geometry** button. The following figure shows what your Unity editor should look like.

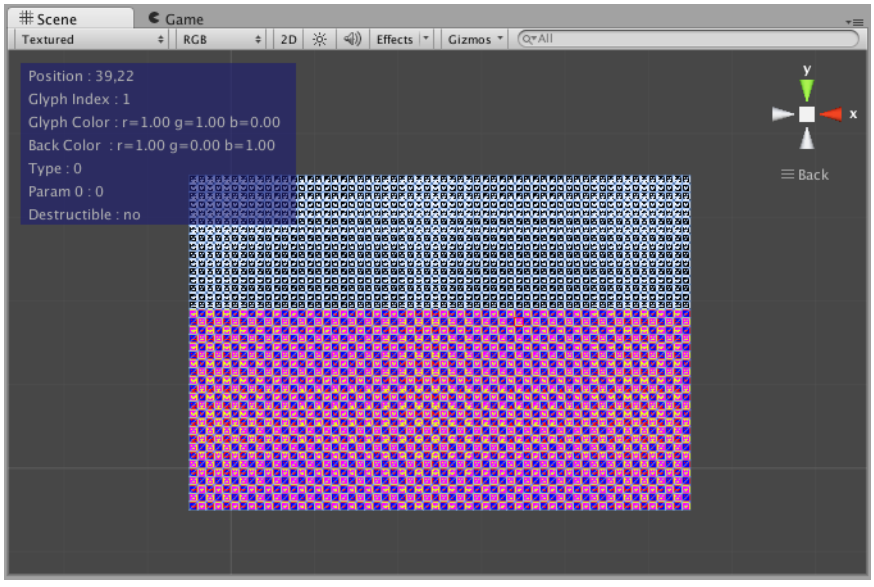


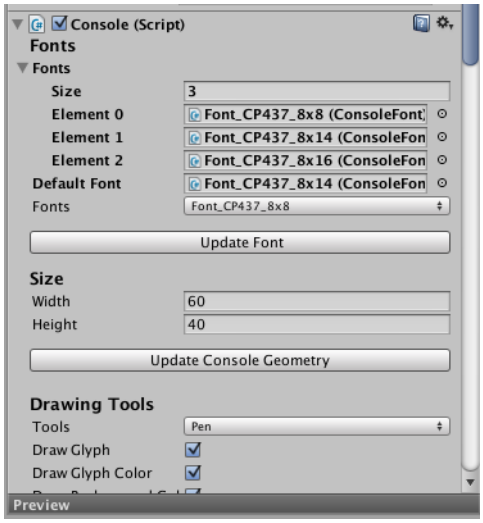
The console is now fully ready. It is a flat mesh that you can integrate in your scene in any way you like.

1.2 Fullscreen and Pixel Perfect Console

To make the console full screen, you have to adjust the Player Settings and the camera the proper way. We will take the example of a 60 by 40 characters console, using a 8 by 8 pixels font.

First, resize the console. Pick it in the Hierarchy tab. Move the console to position (-30, -20, 0) to ensure that the console will be properly centered camera-wise. If the Console component is folded, unfold it. Select the Font_CP437_8x8 font in the **Font** drop down menu. Enter 60 and 40 for the width and height in the **Size** section. Push the **Update Console Geometry** button. This will update both the current font and the console size. Your Scene view and Inspector tab should now look like this :

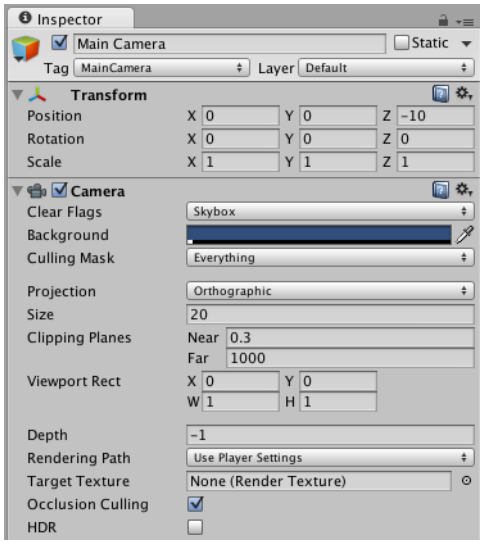




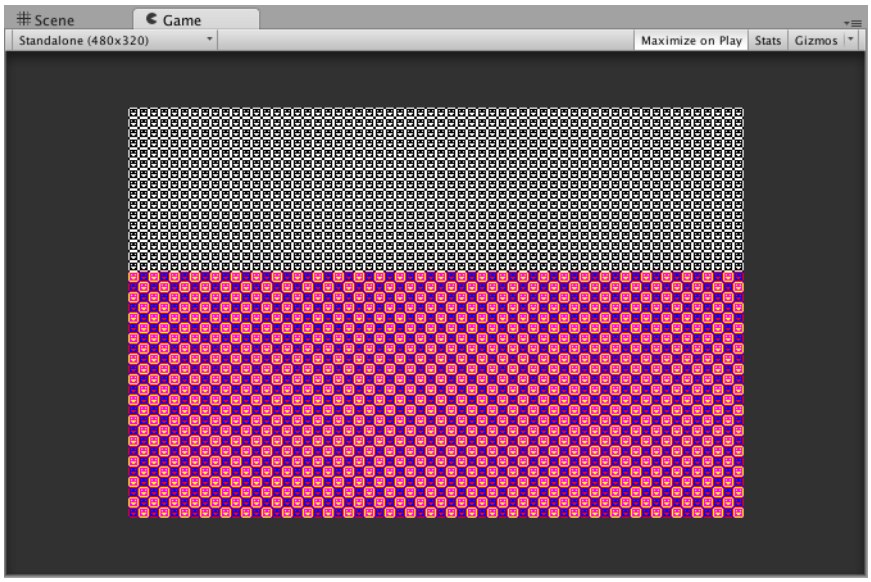
Adjust the **Player Settings**. In the **File** menu, choose the **Build Settings...** item. When the **Build Settings Window** appears click the **Player Settings** button. The Inspector tab should now display **Per-Platform Settings**. Choose the web player settings (the leftmost button showing a wireframe sphere). In the **Default Screen Width** and **Default Screen Height** fields respectively, enter 480 and 320. Quite logically the screen width is 60 characters * 8 pixels = 480 pixels. Likewise, the height is 40 * 8 = 320 pixels.

Adjust the camera settings. In the Hierarchy tab, pick the **Main Camera**. Move the camera to position (0, 0, -10). Set the **Clear Flags** to **Solid Color** and the **Projection** to **Orthographic**. In the Size field that just appeared, enter 20.
Why 20 ? In orthographic mode, the camera size represents half of the screen view height in Unity units. When the console is unscaled or pixel perfect, each 8 by 8 pixels character is 1 by 1 Unity units in size (a 8 by 14 pixels character is therefore 1 by 1.75 units, a 8 by 16 is 1 by 2 units, etc…) The console is 40 characters high and fullscreen therefore the half height of the screen is 20.

Here is what your main camera's Inspector tab should look like :



After setup is done the Game view should look like this :



You can now start using the console, either in the editor by using the tools available in the Console component in the Inspector tab or programmatically using the simple API documented later on.

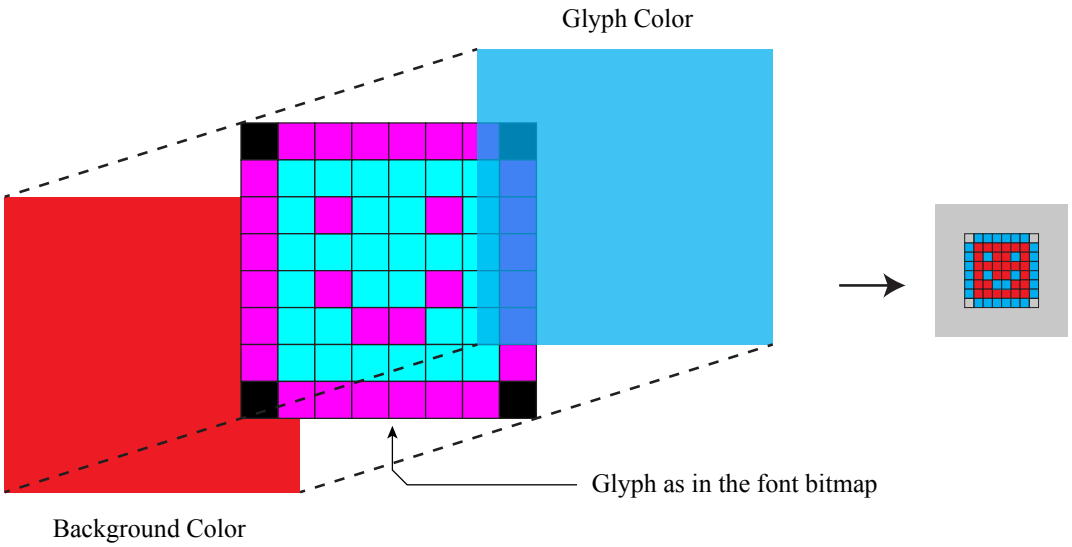
2 Drawing in the Editor

2.1 Tiles and Characters

Each character displayed by the console is actually a tile. Each tile has 6 different attributes :

- a glyph
- a glyph color
- a background color
- an integer type
- a float parameter
- a boolean destructability flag

The 3 graphical attributes work as follow: magenta pixels of the tile glyph will be displayed using the tile's glyph color; cyan pixels will be displayed using the tile's background color; black pixels will be transparent.



When using any of the drawing tools, you can enable or disable the drawing and setting of any of those 6 attributes using the toggles under the Tools drop down menu.
When retouching a drawing, it is sometimes convenient to disable some attributes. For example when only changing colors, uncheck the Draw Glyph toggle to only draw colors but not glyphs avoiding the painful process of re-picking each glyph everytime you want to change its colors.

2.2 Tools :

Drawing on the console within the Unity editor is very easy and is similar to drawing with most drawing applications.

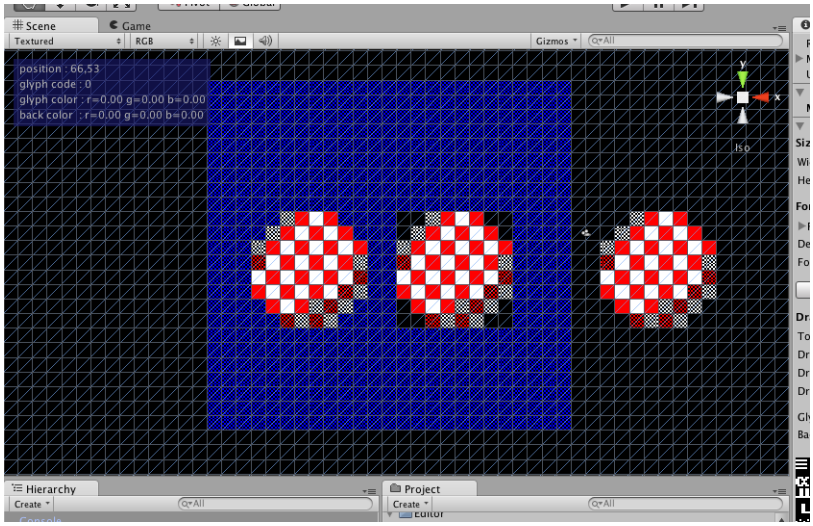
Drawing tools are selected with the **Tools** section drop down menu of the Console component in the inspector. Most tools are straigthforward to use. Only the **Pen** and **Stamp** tools need a little more in depth explanation.

The **Pen** primary function sets the clicked tile attributes to the current attributes displayed in the inspector. However if the *alt* key is pressed while clicking in the console, the pen works as a picker: the current attributes will be set to the ones from the clicked tile. Only enabled attributes are picked.

The **Stamp** tool allows you to copy and paste a rectangular portion of the console. To select a portion of the console, choose **Stamp** in the tools drop down menu then *alt*-click in the console to set the first corner of the selection. The selection frame turns green. Click a second time to set the second corner. The selection frame turns yellow, indicating that a selection was made. You can then paste the selection anywhere you want in the console.
The selection is not affected by attribute toggles but pasting is.

When the Stamp tool is selected, some additional toggles and buttons are added to the Tools section. The **Rotate**, **Flip**, **Load** and **Save** buttons are straightforward. Saved stamps can also be used as sprites by some of the **API** functions if saved somewhere in the current project's Asset folder.

When making a selection with the Stamp tool, the selection is checked for tiles with both a black glyph color and a black background color (black meaning red, green and blue set to 0 and alpha to 1). Wether tiles are all black or not defines a sprite mask: when pasted, if the **Use Sprite Mask** toggle is set, parts of the selection that are all black will not be drawn. In the following example, the original rightmost sphere was outlined with characters of black glyph color and black background color. The leftmost sphere was pasted with **Use Sprite Mask** enabled, the center one with **Use Sprite Mask** disabled.



3 Drawing Programmatically

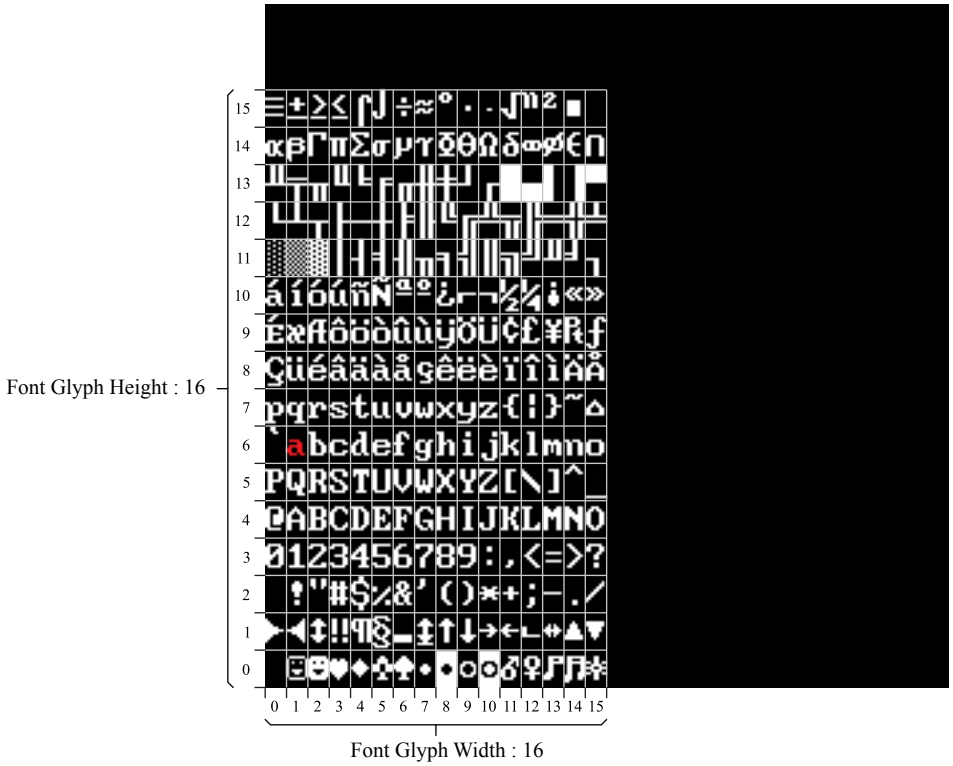
3.1 The Graphic Context and the Glyph Property

To operate properly most of the drawing functions require the graphic context to be properly set up. The graphic context consists in a series of properties the console API makes available. All of those properties are explained in the next section and understanding their use should be straightfoward. Nevertheless, the Glyph property probably needs more explanations.

The Glyph property stores the current glyph index. The glyph index represents the linear position of the glyph in the font array and is calculated like so :

*glyph index = glyph X + font array width * glyph Y*

The following figure shows the default 8 by 14 pixels font texture. Here, the glyph index for the letter *a* is 1 + 16 * 6 = 97.



Alternatively, when the mouse hovers above a character in the Scene view, its glyph index is displayed in the top left blue frame.

For the 3 default fonts provided with the package, all characters within the decimal 32 to decimal 127 ASCII range have their glyph index identical to their ASCII code. This is something you should keep in mind when designing your own fonts.

3.2 Accessing the Console Script

The following code samples show you the basic structure of any script accessing the console. **It is absolutely necessary for any drawing code in the console to be enclosed between calls to the** `PrepareUpdate` **and** `EndUpdate` **methods!!! If using sprites it is strongly advised to use the** `ClearUnusedTopSprites` **and** `ClearUnusedBottomSprites` **methods!!!**

In Javascript :

```
#pragma strict

// Interface :
private var console : Console;

function Start () {

	// Getting a reference on the Console component :
	console = GetComponent ( “Console” ) as Console;

	// ... and for example :
	console.ClearGlyph 		= 1;		// smiley is now ...
							// ... the clear glyph.
	console.ClearGlyphColor 	= Color.white;
	console.ClearBackgroundColor 	= Color.black;
	console.GraphicsOnly 		= true;		// draw methods will ...
							// ... only draw glyph and colors.

}

function Update () {

	// Drawing in the console begins here :
	console.PrepareUpdate ();

	console.Clear ();				//clearing the console

	console.Glyph 		= 97;
	console.GlyphColor 	= Color.blue;
	console.BackgroundColor = Color.red;
	console.SetCharacter ( 10, 10 );		// writes a in blue ...
							// ... over red at ( 10, 10 )

	//Drawing in the console ends here :
	console.EndUpdate ();

}
```

In C# :

```
using UnityEngine;
using System.Collections;

public class MyClassUsingAConsole : MonoBehavour {

	// Interface :
	private Console console;

	void Start () {

		// Getting a reference on the Console component :
		console = GetComponent<Console> ();

		// ... and for example :
		console.ClearGlyph 		= 1;		// smiley is now ...
							// ... the clear glyph.
		console.ClearGlyphColor 	= Color.white;
		console.ClearBackgroundColor 	= Color.black;
		console.GraphicsOnly 		= true;		// draw methods will ...
							// ... only draw glyph and colors.

	}
```

```
	void Update () {

		// Drawing in the console begins here :
		console.PrepareUpdate ();

		console.Clear ();                                   // clearing the console

		console.Glyph              = 97;
		console.GlyphColor         = Color.blue;
		console.BackgroundColor    = Color.red;
		console.SetCharacter ( 10, 10 );                    // writes “a” in blue ...
                                                            // ... over red at ( 10, 10 )

		// Drawing in the console ends here :
		console.EndUpdate ();

	}

}
```

4 Console Sprites and Console Animations

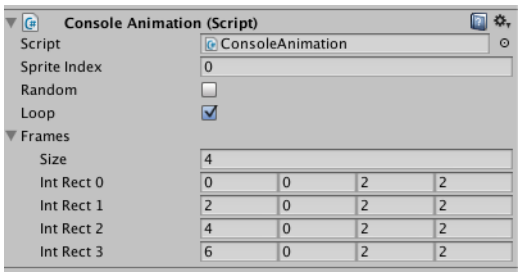
4.1 Console Sprites

Console Sprites are assets that store a square portion of the console that can later be drawn using some of the API methods.

Console Sprite assets are generated with the **stamp** tool in the console's inpector: select the part of the console that defines the sprite by *alt*-clicking on one corner of the selection and then framing it; when the selection frame turns yellow push the **Save Stamp** button in the inspector; save the asset anywhere in the project's Assets folder.

4.2 Console Animations

Console Animation assets use registered Console Sprite assets as sprite sheets. The **Sprite Index** field refers to the sprite sheet index in the console's list of registered Console Sprite assets (the list is explained in the next section). The **Frames** array stores each frame of the animation. Each field represents respectively the x, y, width and height properties of one particular frame in the sprite sheet.



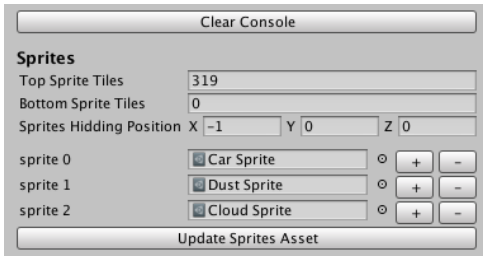
The **Random** and **Loop** toggles are self explanatory.

4.3 Drawing Sprites and Animations

Console Sprites are drawn in 2 differents ways. The `DrawConsoleSprite` method pastes a console sprite at the desired location in the console. It simply replaces the consoles tiles attributes' values with the ones stored in the sprite's asset.

The `DrawSprite` method works in a different way entirely. It draws sprites into an extra set of tiles that can be moved around freely in the Unity space. Using such sprites requires 2 steps.
First, all Console Sprite assets used in the project have to be registered in the list found at the bottom of the console's inspector. The list can be modified and updated at any time by pushing the **Update Sprite Assets** button. Note that all the API methods and Console Animation assets identify Console Sprite assets by their index in this list so when the list is updated, code and animations must be updated accordingly.
The extra tiles required by the free moving sprites must be allocated in the console's inspector. These extra tiles are separated in 2 groups: tiles that will be drawn under the console and tiles that will be drawn on top of the console. These numbers allocate the maximum number of tiles your application requires to properly display all the sprites it needs.

Let's take the car game example that is provided with the TrueColorCP437 package. Here is what the console's inspector sprite section looks like:



At the bottom is the list of all the Console Sprite assets required by the game: the car sprite, the dust cloud sprite and the cloud sprite. The cloud's Console Sprite asset is labeled as sprite 2. If we look at the code that displays the cloud, indeed, the asset is also refered to by the number 2 (notice the `Console.TOP` constant that tells the `DrawSprite` method

to draw this sprite on top of the console):

```
public void DoCloudDisplayUpdate () {
	...
	map.DrawSprite ( 2, cloud[i].x, cloud[i].y, Console.TOP );
}
```

There is no sprite under the console, so the **Bottom Sprite Tiles** input field is set to 0. The car's Console Sprite asset is 2x2, the dust cloud's Console Sprite asset is 1x1 and the cloud's Console Sprite asset is 15x10 tiles. There is only one car, the car generates about 60 dust clouds a second, each one lasting about 1/4th of a second and their can be a maximum of 2 clouds displayed at the same time. The maximum tile count needeed for all those sprite is therefore 1*2*2+1*1*60/4+2*15*10 or 319 tiles in the **Top Sprite Tiles** input field.

If your game displays a variable number of sprites in each frame (for example because bullets or a dead enemies disapear from screen), it is necessary to clear the unused sprite tiles by moving them back to the **Sprite Hidding Position**. This is achieved by calling the `ClearUnusedTopSprites` and `ClearUnusedBottomSprites` methods.

Animations are updated by calling their `Play` or `Reset` methods. They also expose a `Status` property that reflects the state of the animation. There are only 2 states : `ConsoleAnimation.PLAYING` or `ConsoleAnimation.FINISHED`.
Animations are displayed using the `DrawAnimation` and `DrawRotatedAnimation` methods. These 2 methods fundamentaly work the same as the `DrawSprite` method and all information mentioned earlier for Console Sprite assets apply.

4.4 Z-Sorting

As mentioned earlier, it is possible to draw sprites and animations behind or in front of the console. This is achieved by allocating the proper number of tiles in the Bottom and Top Sprite Tiles groups and then passing the appropriate `Console.BOTTOM` or `Console.TOP` flag to the sprite and animations drawing methods.

Within those 2 groups sprites are z-sorted in the reverse order their drawing methods are called. In the bottom and top tiles groups, the sprite that is drawn last in the drawing code is the sprite that will be drawn on top. Both groups are independant so it is not necessary to draw sprites in the bottom group before drawing sprites in the top group.

5 Strings

As explained in the `DrawString` method's descritpion, **when using the default fonts**, characters within the decimal 32 to decimal 127 range (meaning the alphabet both capital and non-capital, digits and most of the useful poncutation) are properly matched by their glyph index. Such characters passed in a string to the `DrawString` method will print properly. However, if you want the `DrawString` method to print characters outside of that range you have to use their glyph index padded with 0 and preceded by a slash (/).

Here are some examples :

```
console.DrawString ( 10, 20, “Hello” );
console.DrawString ( 10, 19, “Hello !!!” );
console.DrawString ( 10, 18, “Hello /001” );
console.DrawString ( 10, 17, “/017 Hello /016” );
console.DrawString ( 10, 16, “/176/177/178 Hello /178/177/176” );
```

will print :

6 Creating Fonts

6.1 Font Texture

The texture must be a rgba file. It can be of any size provided that size is supported by the device your building your game for. Note that powers of 2 dimensions offer better performances.

In the font texture, arrange your glyphs in a rectangular array. The console displays the glyph pixels set to magenta (r:255 g:0 b:255) with the tile's glyph color. Pixels set to cyan (r:0 g:255 b:255) are displayed with the background color. Pixels set to black are not displayed at all.

Once the texture is imported in Unity, set the following parameters :
- *Texture Type* : Texture
- *Alpha from Grayscale* : unchecked
- *Wrap mode* : Repeat
- *Filter Mode* : Point

Other parameters can be left untouched.

6.2 Font Object

In the editor create an empty game object (**GameObject menu -> Create Empty**). Drag and drop a ConsoleFont script component on the new empty game object in the Hierarchy tab.
The ConsoleFont script can be found in the **Plugins -> TrueColorCP437 -> Scripts** folder.

Select the new font game object in the Hierarchy tab and in the Inspector tab, enter the proper parameters for your font :
- *Font Name* : the name of the font
- *Font Glyph Width* : the width of the glyph array
- *Font Glyph Height* : the height of the glyph array
- *Glyph Width* : the width of one glyph in Unity units
- *Glyph Height* : the height of one glyph in Unity units
- *Glyph Pixel Width* : the width of one glyph in pixels
- *Glyph Pixel Height* : the height of one glyph in pixels
- *Font Texture* : the font's texture
- *Glyph Picker Tex Width* : the width in texture units of the glyph array
- *Glyph Picker Tex Height* : the height in texture units of the glyph array

Sometimes the glyph array doesn't fill the whole texture and is surrounded by black strips. These black strips needlessly clutter the glyph picker in the Inspector tab. The Glyph Picker Tex Width and Glyph Picker Tex Height parameters allow the glyph picker to properly frame the glyph array.

7 API Reference

7.1 Properties

Property Name	***Property Description***
`Width`	***type*** : int ***description*** : the width of the console measured in characters. ***access*** : read only
`Height`	***type*** : int ***description*** : the height of the console measured in characters. ***access*** : read only
`Glyph`	***type*** : int ***description*** : the current glyph used by drawing functions. ***access*** : read / write
`GlyphColor`	***type*** : Color ***description*** : the current glyph color used by drawing functions. ***access*** : read / write
`BackgroundColor`	***type*** : Color ***description*** : the current background color used by drawing functions. ***access*** : read / write
`Type`	**type** : int **description** : the current tile type set by drawing functions. **access** : read / write
`Param`	**type** : float array **description** : the current array of floats set by drawing functions. **access** : read / write
`Destructible`	**type** : bool **description** : the current destructability flag set by frawing functions. **access** : read / write
`ClearGlyph`	***type*** : int ***description*** : the glyph used by the Clear () method. ***access*** : read / write
`ClearGlyphColor`	***type*** : Color ***description*** : the glyph color used by the Clear () method. ***access*** : read / write
`ClearBackgroundColor`	***type*** : Color ***description*** : the background color used by the Clear () method. ***access*** : read / write
`DrawGlyph`	***type*** : bool ***description*** : enables / disables drawing glyphs if set to `true` or `false`. ***access*** : read / write
`DrawGlyphColor`	***type*** : bool ***description*** : enables / disables drawing glyph colors if set to `true` or `false`. ***access*** : read / write
`DrawBackgroundColor`	***type*** : bool ***description*** : enables / disables drawing background color if set to `true` or `false`. ***access*** : read / write
`SetType`	***type*** : bool ***description*** : enables / disables setting type if set to `true` or `false`. ***acces*** : read / write
`SetParam`	***type*** : bool ***description*** : enables / disables setting param if set to `true` or `false`. ***acces*** : read / write

Property Name	***Property Description***
`SetDestructability`	***type*** : bool ***description*** : enables / disables setting destructability if set to `true` or `false`. ***acces*** : read / write
`UseSpriteMask`	***type*** : bool ***description*** : enables / disables drawing sprite characters with a black glyph color and background color if set to `true` or `false`. ***access*** : read / write
`GraphicsOnly`	***type*** : bool ***description*** : there are 2 sets of drawing methods : methods that will draw and set all current attributes and faster methods that will only draw the glyph, glyph color and glyph background color. If set to true, the console uses the faster graphics only set of methods. If set to false the slower but exhaustive set is used. ***acces*** : read / write

7.2 Console Methods

Method Name	*Method Description*
`void Clear ()`	***return type*** : `void` ***arguments*** : none ***descritpion*** : clears the console using the current `ClearGlyph`, `ClearGlyphColor`, `ClearBackground-Color`, `Type`, `Param` and `Destructible` attributes. *Instance method*
`void SetCharacter ( int x, int y )`	***return type*** : `void` ***arguments*** : the `x` and `y` position of the character. ***descritpion*** : sets the character at the given coordinates using the current attributes. *Instance method*
`void StrokeHorizontalLine ( int x1, int x2, int y )`	***return type*** : `void` ***arguments*** : `x1` and `x2` are the horizontal boundaries of the horizontal line; `y` is its vertical position in the console. ***descritpion*** : strokes a horizontal line using the current attributes. *Instance method*
`void StrokeVerticalLine ( int x, int y1, int y2 )`	***return type*** : `void` ***arguments*** : `x` is the horizontal position of the vertical line; `y1` and `y2` are its vertical boundaries. ***descritpion*** : strokes a vertical line using the current attributes. *Instance method*
`void StrokeLine ( int x1, int y1, int x2, int y2, bool smooth )`	***return type*** : `void` ***arguments*** : the `x` and `y` coordinates represents both extremities of the line; the `smooth` flag defines wether the line function should smooth the line or not. ***descritpion*** : strokes a line between 2 points. If the smooth flag is set to `false`, the line is drawn using the current attributes. If the smooth flag is set to `true`, the method uses all current attributes but `Glyph`. It instead draws the line using a combination of `/`, `\`, and `_` glyphs to give the impression of a smoother line. *Instance method*
`void StrokeRectangle ( int x, int y, int w, int h )`	***return type*** : `void` ***arguments*** : `x` and `y` are the coordinates of the lower left corner of the rectangle; `w` and `h` are its width and height. ***descritpion*** : strokes a rectangle using the current attributes. *Instance method*
`void FillRectangle ( int x, int y, int w, int h )`	***return type*** : `void` ***arguments*** : `x` and `y` are the coordinates of the lower left corner of the rectangle; `w` and `h` are its width and height. ***descritpion*** : fills a rectangle using the current attributes. *Instance method*

Method Name	***Method Description***
`void StrokePolygon ( Vector2[] vertices, bool smooth )`	***return type*** : `void` ***arguments*** : `vertices` is an array containing the polygon's vertices; the `smooth` flag defines wether the polygon edges should be smoothed or not. ***descritpion*** : strokes a polygon. If the smooth flag is set to `false`, the edges are drawn using the current attributes. If the smooth flag is set to `true`, the method uses all current attributes but `Glyph`. It instead draws the line using a combination of `/`, `\`, and `_` glyphs to make them smoother. *Instance method*
`void FillPolygon ( Vector2[] vertices, bool smooth )`	***return type*** : `void` ***arguments*** : `vertices` is an array containing the polygon's vertices; the smooth flag defines wether the polygon edges should be smoothed or not. ***descritpion*** : fills a polygon using the current attributes. *Instance method*
`void DrawString ( int x, int y, string s )`	***return type*** : `void` ***arguments*** : `x` and `y` define where the string will be drawn (note : if the string contains carriage return characters, `x` and `y` are the coodinates of the top line); `string` is the string to draw. ***descritpion*** : draws a string at the given coordinates. When using the default fonts provided with the package, all characters within the decimal 32 to decimal 127 ASCII range are properly mapped. Characters outside this range are accessed with their glyph index preceded by a /. *Instance method*
`void DrawSprite ( ConsoleSprite sprite, int x, int y )`	***return type*** : `void` ***arguments*** : `sprite` is a ConsoleSprite object priviously serialized with the Stamp tool; `x` and `y` define where to draw the sprite. ***descritpion*** : draws a ConsoleSprite object at the given coordinates. If the `UseSpriteMask` propety is set to `true`, all characters in the sprite with black glyph color and blackback-ground color are ignored. If it is set to `false`, they aren't. *Instance method*
`static ConsoleSprite RotateSprite90DegreesLeft ( ConsoleSprite sprite )`	***return type*** : `ConsoleSprite` ***arguments*** : `sprite` is a ConsoleSprite asset previously serialized with the Stamp tool ***descritpion*** : returns a new ConsoleSprite object containing the original sprite data rotated 90° left. *Class method*

Method Name	*Method Description*
`static ConsoleSprite RotateSprite90Degrees-Right ( ConsoleSprite sprite )`	***return type*** : `ConsoleSprite` ***arguments*** : `sprite` is a ConsoleSprite asset previously serialized with the Stamp tool ***descritpion*** : returns a new ConsoleSprite object containing the original sprite data rotate 90° rigth. *Class method*
`static ConsoleSprite FlipSpriteHorizontal ( ConsoleSprite sprite )`	***return type*** : `ConsoleSprite` ***arguments*** : `sprite` is a ConsoleSprite asset previously serialized with the Stamp tool ***descritpion*** : returns a new ConsoleSprite object containing the original sprite data flipped horizontally. *Class method*
`static ConsoleSprite FlipSpriteVertical ( ConsoleSprite sprite )`	***return type*** : `ConsoleSprite` ***arguments*** : `sprite` is a ConsoleSprite asset previously serialized with the Stamp tool ***descritpion*** : returns a new ConsoleSprite object containing the original sprite data flipped vertically. *Class method*
`void DrawSprite ( int sprite, float x, float y, bool topOrBottom )`	***return type*** : `void` ***arguments*** : `sprite` is the index in the console Console Sprite assets list, `x` and `y` the sprite's drawing position in Unity units and `topOrBottom` a flag that defines wether the sprite is drawn on top or under the console. ***description*** : draws a sprite in Unity space at the desired position. *Instance Method*
`void DrawAnimation ( ConsoleAnimation animation, float x, float y, bool topOrBottom )`	***return type*** : `void` ***arguments*** : `animation` is a Console Animation asset, `x` and `y` define the animation's drawing position in Unity units and `topOrBottom` is a flag that defines wether the animation is drawn on top or under the console. ***description*** : draws an animation in Unity space at the desired position. *Instance Method*
`void DrawRotatedAnimation ( ConsoleAnimation animation, float x, float y, float angle, bool topOrBottom )`	***return type*** : `void` ***arguments*** : `animation` is a Console Animation asset, `x` and `y` define the animation's drawing position in Unity units, `angle` defines the orientation and `topOrBottom` is a flag that defines wether the animation is drawn on top or under the console. ***description*** : draws an animation in Unity space at the desired position. *Instance Method*
`void ClearUnusedTopSprites ()`	***return type*** : `void` ***arguments*** : none ***description*** : moves all unused tiles in the top group back to the sprite hidding position. *Instance Method*

Method Name	*Method Description*
`void ClearUnusedBottomSprites ()`	***return type*** : `void` ***arguments*** : none ***description*** : moves all unused tiles in the bottom group back to the sprite hidding position. *Instance Method*

7.3 Console Animation Properties

Property Name	*Property Description*
`Frame`	***type*** : IntRect ***description*** : the current frame. ***acces*** : read only
`FrameIndex`	***type*** : int ***description*** : the current frame index ***acces*** : read / write
`Status`	***type*** : int ***description*** : the current status of the animation. It can be either `ConsoleAnimation.PLAYING` or `ConsoleAnimation.FINISHED`. ***acces*** : read only

7.4 Console Animation Methods

Method Name	*Method Description*
`void Reset ()`	***return type*** : `void` ***arguments*** : none ***description*** : resets the animation to frame 0 and sets the animation's `Status` property to `ConsoleAnimation.PLAYING`. *Instance Method*
`void Play ()`	***return type*** : `void` ***arguments*** : none ***description*** : sets the animation's current frame to the next frame. If the animation is random, the next frame is randomly chosen. If the animation is a loop, the last frame loops to the first frame. If the animation is neither random or looped, reaching the last frame sets the `Status` property to `ConsoleAnimation.FINISHED` and subsequent calls to Play will have no effect. *Instance Method*

8 Troubleshooting

Q: I get a Null Reference exception immediately after I start running my game.
A: Verify that you are properly enclosing all your calls to drawing methods within `PrepareUpdate` and `EndUpdate` calls and have allocated enough tiles for your sprites.

Q: My console looks fine in the editor but when I run the game, what I see doesn't make any sense.
A: Before running your game, verify that you have re-enabled all the drawing attributes toggles in the console's inspector. You can also force-set them to `true` in the `Start` method of your game code.

Q: Some sprites stay on screen even if my code doesn't draw them anymore.
A: Verify that you properly call the `ClearUnusedTopSprites` and `ClearUnusedBottomSprites` methods at the end of your frame update code. Also verify that your Sprite Hidding Position is set outside of your camera field of view.

Q: I get seemingly random Null Reference exceptions when I run my game.
A: Verify that you have allocated enough tiles for your sprites.

Q: Glyphs are slightly verticaly offset in the web player under Windows.
A: These artifacts appear in the web player under Windows only. Playing with **Use Direct3D 11**, **GPU Skinning** and **Optimize Mesh Data** in the Player Settings often solve this problem.