Font Setter / Font Packer user manual

Table of Contents

0. Introduction

Unity is capable of displaying text in a rendered scene, via text meshes, using GUIText, or using UnityGUI. By default, there is no simple way to create fully customised fonts. This editor extension gives the user the freedom to have any created art asset act as a font.

Videos.

For a series of videos that cover how to use the Font Setter, Font Packer, and animating sprite functions; check this Youtube playlist:

www.youtube.com/playlist?list=PL3Mj-QGBidI3EvVRJEaw0zLovfSX9kZdf

Limitations.

There are a few small limitations:

- **Fonts created this way cannot be resized in the conventional manner GetComponent(TextMesh).fontSize = 12; By changing the 'font size' parameter in the inspector).**
    - **TextMeshes have a property 'characterSize', which can be changed, 1.0 is full-size, less than 1.0 is smaller, greater than 1.0 is bigger, Alternatively, you can change the localScale of the TextMesh gameObject transform.**
- **Fonts created this way will not display fully textured when using UnityGUI (GUI.Label(Rect(0,0,40,20), "This is a GUI Label", "someGUIStyle").**
    - **Fonts assigned to UnityGUI via GUIStyles display as silhouette only, so you can still use Font Setter to create the shapes for UnityGUI fonts.**
    - **There are workarounds for this, see (Section 10).**
- **Entering PlayMode at any point while the editor is open will break some functionality, so the extension disables itself. You can, however, start the extension during PlayMode.**

1. Terminology

In Unity, fonts can either be displayed by pre-generated textures, or (from Unity 3.0) using the FreeType font rendering engine for 'dynamic' fonts. If you import a font, it will probably be set by default to dynamic. This means that fonts do not need a texture, and can be resized and displayed in large typefaces, without a loss of quality. However, using dynamic fonts, or relying on Unity to generate a texture prevents you from using full-color lettering in your Text Mesh assets.

It is possible to create your own font textures, of any shape, size and color, and this editor extension will help you define the character parameters.

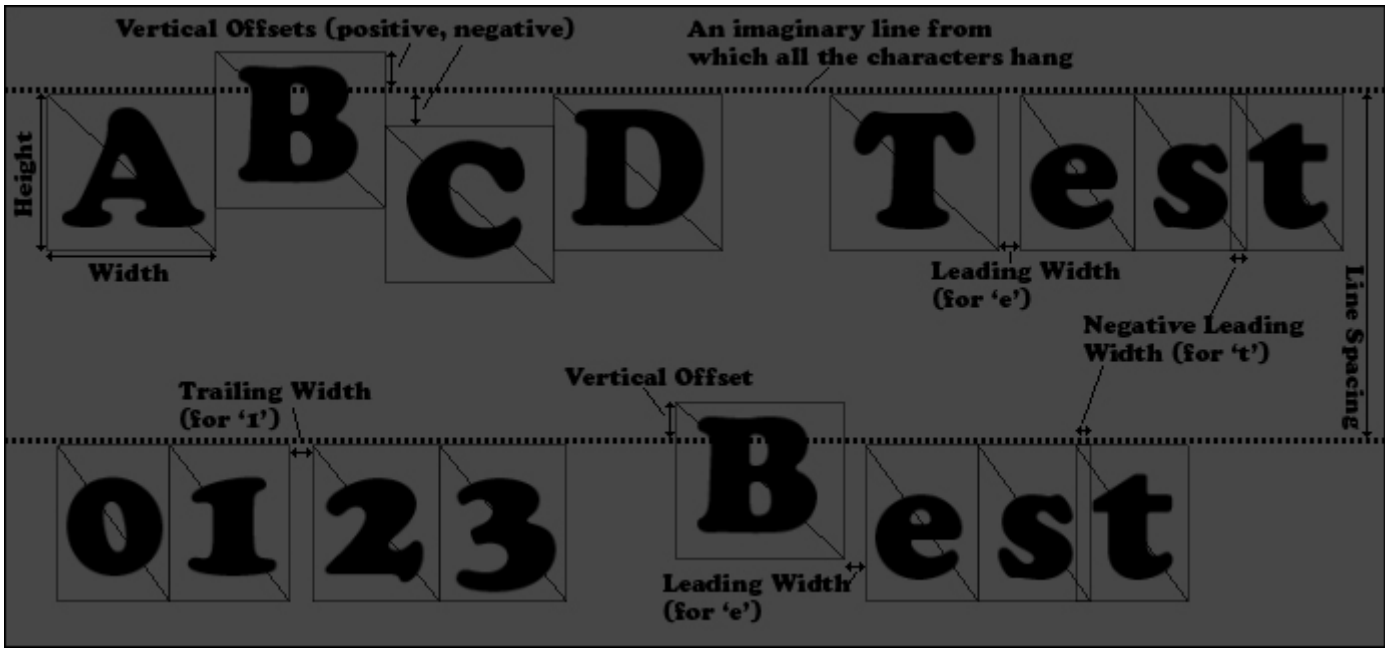


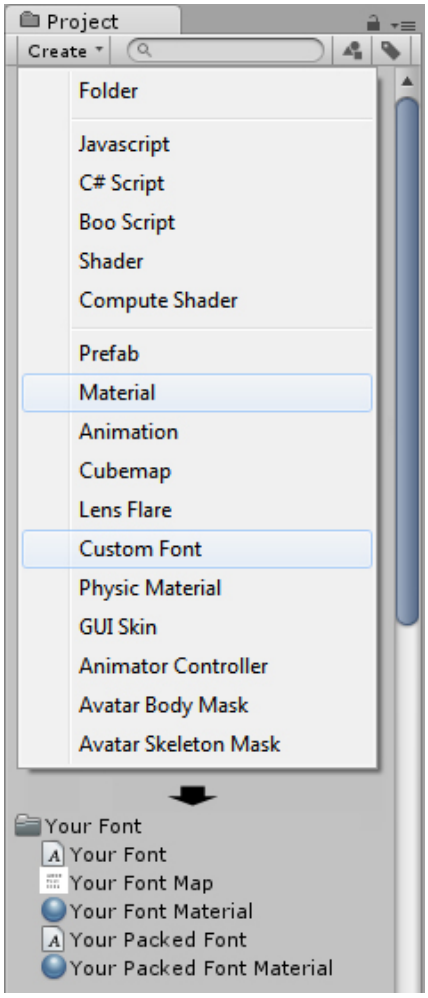
Fig1. Character properties and measurements

When Unity draws Text Mesh characters, there are 3 (4) key components in play; they are created according to a 'font settings' object (someFont.fontsettings), drawn using a 'material' (someFont.mat), which in turn references (via a shader, someFontShader.shader) a bitmap texture, or 'font map' (eg. SomeFont.png).

The font settings object defines the measurements shown in [Fig1.].

- **Every character definition has a height and a width.**
- **If you imagine a line, like a wire with flags hanging below it, the vertical offset defines how far above or below that line each character is drawn.**
- **The leading and trailing width controls the size of the gap before and after specific characters. This can be negative to allow for squashing individual characters together.**
- **Kerning squashes and separates ALL characters in a font.**
- **Line spacing determines the gap created when using newline characters in strings (pressing return).**
- **Packing a font map means reducing any gaps in the texture, to enable smaller texture file sizes – this can be important for distribution (download sizes).**

2. Initial Set-up.



Before using the Font Setter/Packer editor extension, you will need to create the following assets:

-A bitmap file (font map) containing your font characters. Create this in your preferred image editing software (eg. Photoshop). The file should be either a .png, or something that maintains transparency (eg .psd). In the import settings of the image, you will need to tick 'Read/Write enabled' in the Advanced texture type. You may also need to increase the 'Max Size' to be bigger than the size of the image, or quality will be reduced. Ideally the image should have sizes that are powers of 2 (512,1024,etc). Lastly, the image must be in the correct format: Automatic Compressed and Automatic Truecolor both work fine. 16Bits will not.

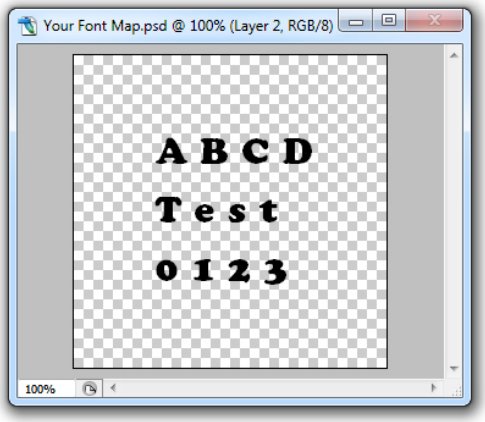
**-A material, with a font rendering shader.
Included with the Font Setter editor should be "Textured Text Shader.shader", taken from
http://wiki.unity3d.com/index.php?title=TexturedFont
Assign your Font Map as the texture.**

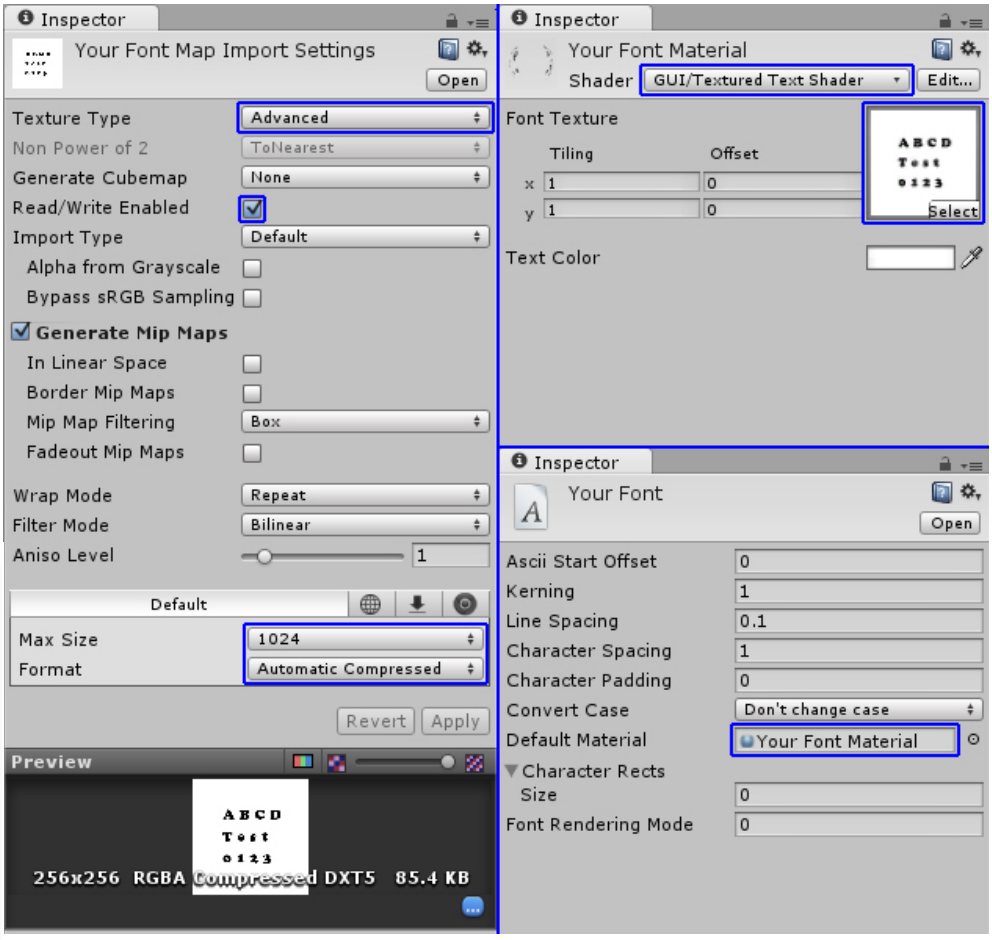
**-A font settings object. These are added from the Unity project browser > Create > Custom Font.
In the inspector, assign your Font Material to the 'Default Material' slot.**

For packing your font, you will also need:

-Another material with a font rendering shader. The texture for this will be generated by the program, and will need to be assigned in the inspector later.

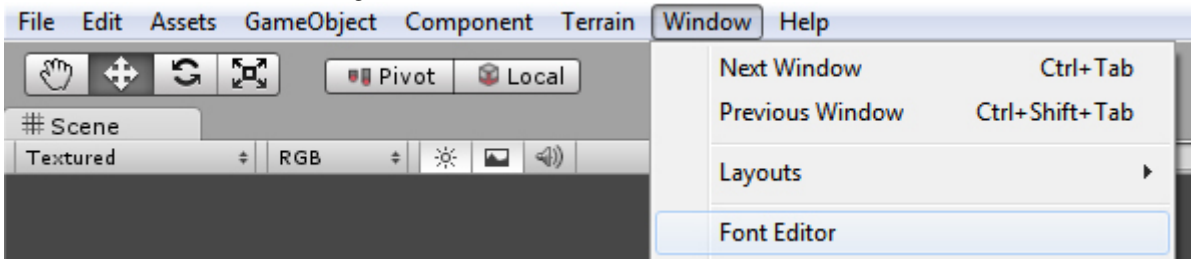
-Another font settings object, with the second material assigned to it.





3. Main Menu

To start the Font Setter/Packer editor extension, click on Window > Font Editor in the menu bar of the Unity3D editor.



This should open a new floating, dockable window as seen in [Fig2.].

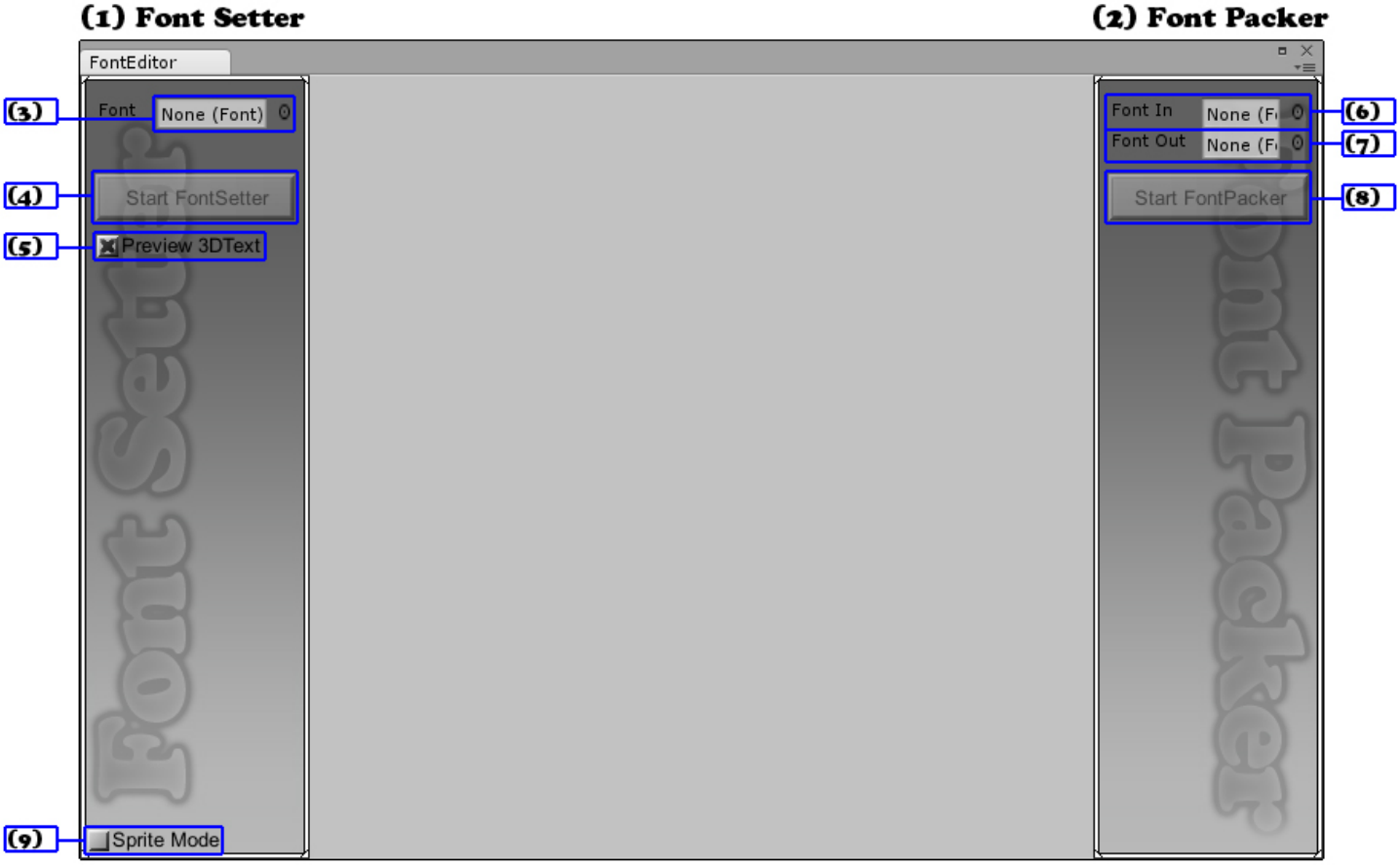


Fig2. The Main Menu

To use (1) Font Setter, use the controls on the left of the editor extension window.

Assign your font settings object in (3), and the button (4) will enable. Click to start the Font Setter.

If ticked, the option (5) will instruct the editor to create a 3DText object in the scene with a sample string, using the font settings and bitmap assigned, so you can see the progress of the new font as you edit.

To use (2) Font Packer (generally, after using Font Setter to create a functioning font), use the controls on the right of the editor window.

Assign the first font settings object to (6), and a second, different font settings object in (7). The button (8) to start the Font Packer is now enabled.

(9) See (Section 11).

4. Font Setter.

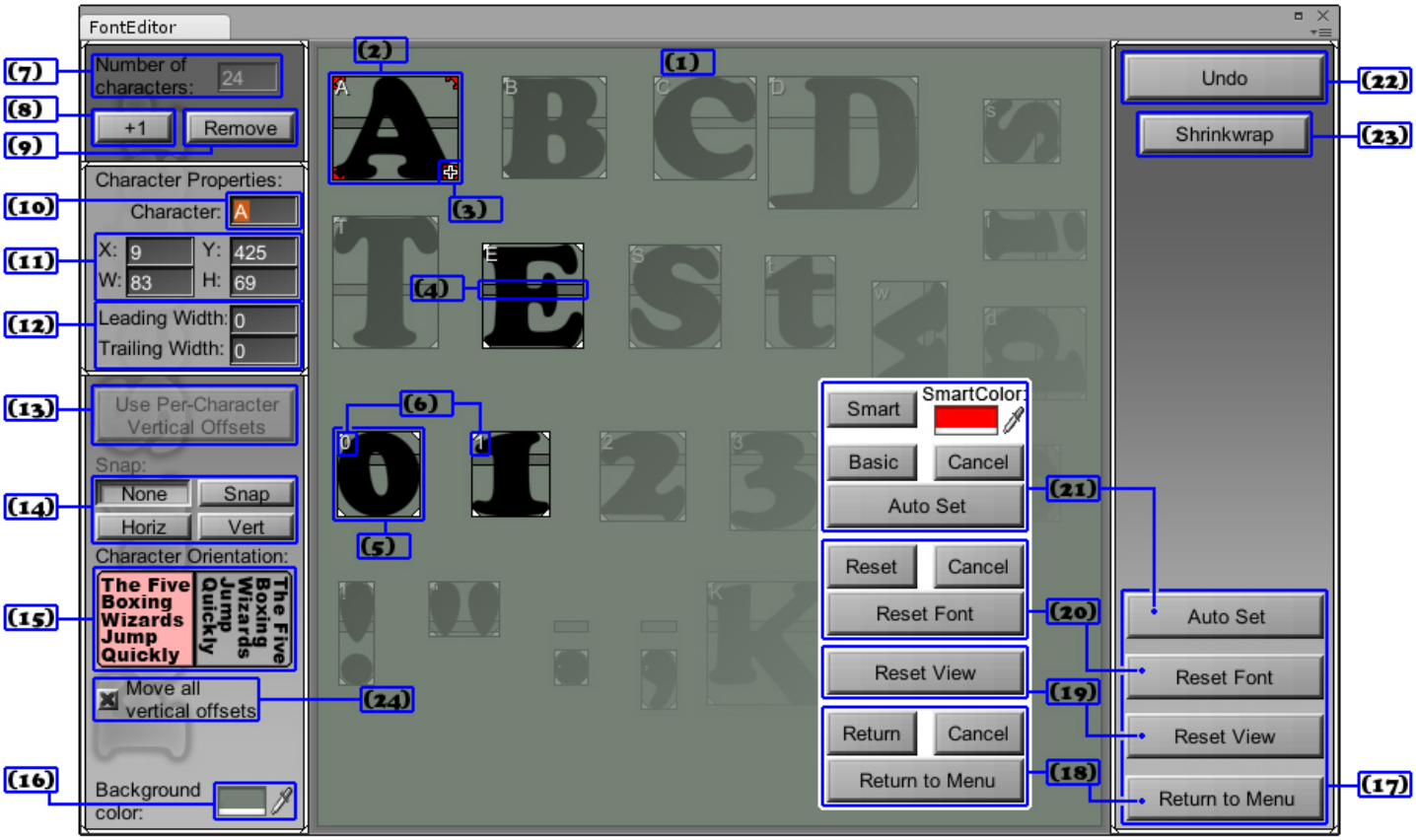


Fig3. The Font Setter Interface

Fonts are defined by rectangles (Rects), and this screen allows to you create and define the size and position of these Rects.

(1) Your font bitmap displays in the main portion of the window, at full resolution. Click and drag with the right (or middle) mouse button to pan, and use the scroll wheel to zoom. (Most characters are greyed out in this document purely for readability)
(2) This is the currently active/selected Rect (red corners). Click and drag with the left mouse button to change its position.
(3) Click on the resize handle to change the size of the currently active Rect.
(4) When enabled, this darker slider represents the characters' vertical offset; how far above or below the character is displayed, compared to the line on which you are typing (see Section 2).
(5) This is an unselected Rect (white corners).
(6) The currently assigned character is displayed inside the Rect.
(7) This field displayed how many character Rects are currently defined.
(8) Click this button to add a Rect. If no Rects are selected, a new one will appear towards the bottom left of your bitmap. If a Rect is selected, the new Rect will duplicate its size, and appear next to it.
(9) This button removes the currently active Rect, alternatively, press delete on the keyboard.
(10) Type into this field to define the character for the currently active Rect. You can paste into this field if you want special characters
(11) This group of fields allows you to type in the X,Y position of the current Rect, and the width and height. The units are measured in pixels, in relation to the font bitmap, with X=0, Y=0 at the bottom left corner.

(12) Here you can set the leading and trailing width of the selected character (see Section 2).
(13) Click this button to enable per-character vertical offsets, if they are not already active (shows all the [Fig3. (4)]'s).
(14) These buttons let you snap Rects together:

- **None: No snapping.**
- **Snap: Snaps the corners of Rects together, for both moving and resizing.**
- **Horiz: Stops you from moving or resizing vertically.**
- **Vert: Stops you from moving or resizing horizontally**
- **Snap also affects the vertical offset sliders [Fig3. (4)]. If you attempt to snap one characters' slider to another, it will position itself at the same height. Trying to snap the slider from an upright character to a sideways character (see [Fig3. (15)]) it will instead copy the vertical offset value directly.**
- **Note that when resizing the currently selected Rect [Fig3. (3)], if the Rect is already snapped on one corner, it will appear to try and snap to itself if you resize on a different axis.**

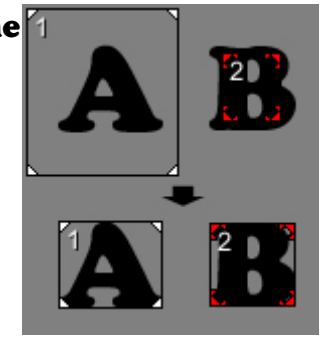
(15) If any of your characters are rotated 90 degrees clockwise, (ie. they are sideways) change the Rect orientation here. Note: Doing so changes the height property of the Rect to a negative number.
(16) Change the background color here. The default is grey-green, so if your font bitmap has characters of this color, you might want to change background.
(17) The left side buttons expand in place, as shown at (18) – (21).
(18) Click "Return to Menu" > "Return" to go back to the main menu.
(19) Click "Reset View" to re-center the image, if you have panned.
(20) Click "Reset Font" > "Reset" to remove all Rects from the font.
(21) Clicking "Auto Set" provides a few more options.

- **Basic: Start the 'Basic Auto Set' as described in (Section 6).**
- **Smart: Start the 'Smart Auto Set' as described in (Section 7).**
- **Smart Color: Choose a color for 'Smart Auto Set' (see Section 7).**

Pressing the "Cancel" button for any of these toolboxes retracts the additional buttons.
(22) Click to undo the most recent change.

- **Note: Undoing any other way (in the main unity windows, or Ctrl+z) will also undo changes made to the font settings object using the editor.**
- **Note: This button will also perform an undo for any action made outside the Font Editor window, be careful not to accidentally undo important changes made to other objects.**

(23) Click "Shrinkwrap" to automatically resize all Rects to the minimum size needed to encompass the bitmap shape they are covering. Rects that cover no shapes are not affected. This should only really be needed when manually settings Rects, if you have used an auto set mode, (Section 6,7) all Rects should already be the correct size.
(24) Toggle this on to enable moving all the vertical offset sliders [Fig3. (4)] at the same time.



Shrinkwrap

5. Procedure for manual setting.

If you choose to manually set the font Rects, this is roughly the method to follow.

- **Assign your font and bitmap objects into their respective slots in the main menu [Fig2. (3,4)]. Click “Start Font Setter” [Fig2. (5)].**
- **Click “+1” [Fig3. (8)] to add a Rect. It will appear towards the bottom left.**
- **Click and drag the Rect with the left mouse button to an appropriate position and size for a character, eg. over the 'a' character. Adjust using the type-in fields [Fig3. (11)] if needed.**
- **Type the character eg. 'a' into the field [Fig3. (10)].**
- **If the character is oriented vertically, indicate as such using [Fig3. (15)].**
- **Press “+1” again. The editor will add a new Rect of the same size adjacent to the one just set. The character displayed in the field [Fig3. (10)] will increment to 'b'.**
- **Move and resize the Rect for another character eg. 'b' (Note: change the character in [Fig3. (10)], it increments alphabetically for ease of use only).**
- **Repeat until all characters are defined by a Rect.**
- **If desired, click “Shrinkwrap” [Fig3. (23)] to automatically adjust all the Rects to the 'ideal' size.**
- **Don't forget to add a Rect for the ' ' (space) character. This can be made relatively small and the width of the space controlled using [Fig3. (12)].**
- **If the checkbox for “Preview 3DText” [Fig2. (6)] was ticked, you should find a 3DText object called “Font Setter Preview Text” in the scene, with the font displayed. Carefully check this to see if any of the Rects need adjusting, any of the letters are missing, etc.**
- **Don't forget to adjust the 'Line Spacing' in the font settings.**
- **You may also want to set the kerning up or down.**

6. Basic Auto Set.

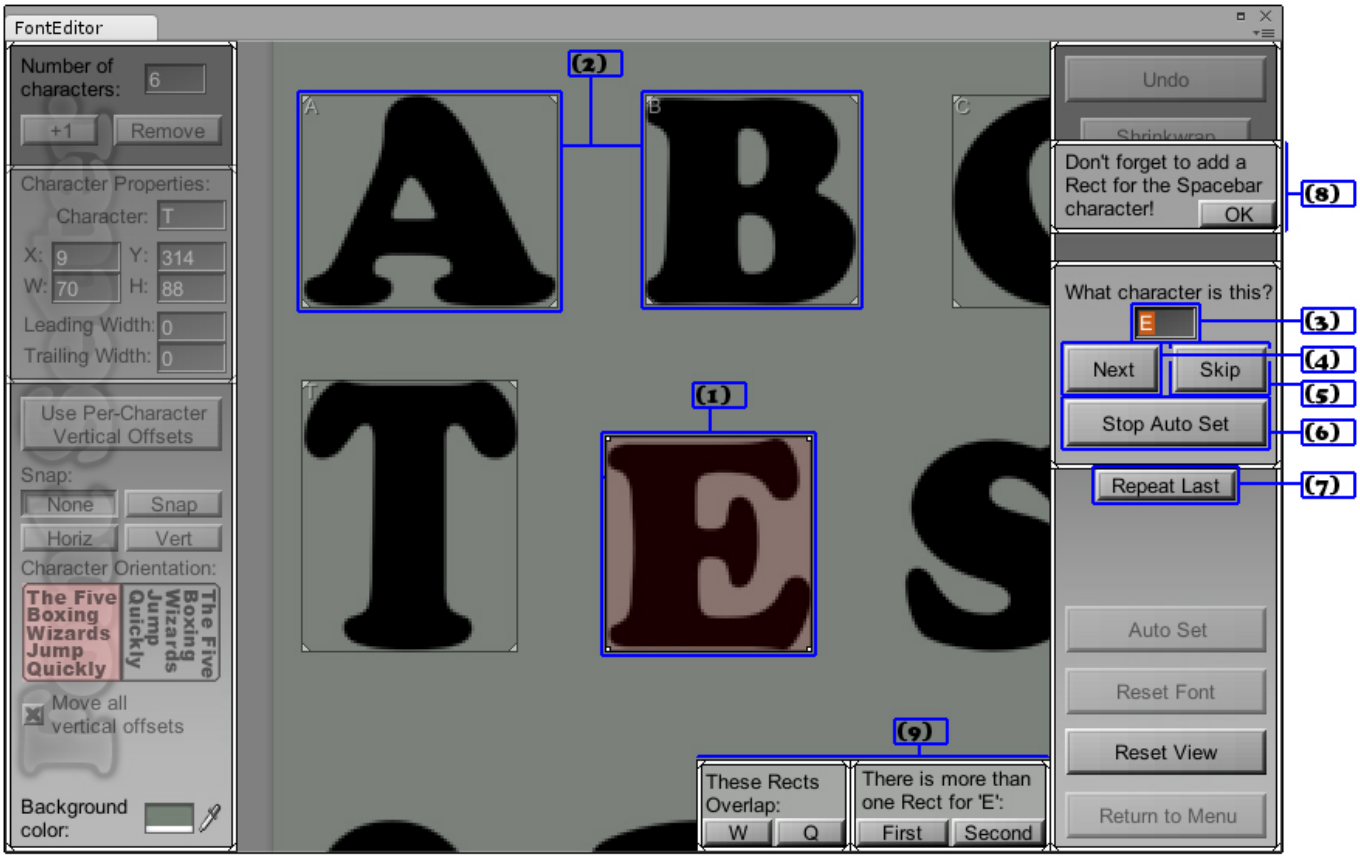


Fig4. Basic Auto Set

When you click "Basic" [Fig3. (21)], the font is first reset, removing all defined Rects – make sure you want to Auto Set first! The Font Setter then automatically identifies shapes in your font map, and brings up the dialog shown on the right, asking you to identify the shapes, one at a time.

(1) This is the currently queried shape (highlighted red).
(2) These are shapes already identified.
(3) Type or paste into this field which character the editor is querying.
(4) Click "Next" after typing a character to identify the current shape, and move to the next. Alternatively, press Enter.
(5) Click "Skip" to skip to the next shape without identifying the current one. Press Escape to skip, without the need to click.
(6) Clicking "Stop Auto Set" and confirming will end the Auto Set procedure.
(7) If you have previously attempted to Auto Set a font during this session, you can click "Repeat Last" to automatically identify all the characters in the same order as before. See (9) for usefulness.
(8) At the end of the Auto Set process, a prompt reminding you to add a space character appears. Dismiss it by clicking "OK".
(9) After the Auto Set is complete, some information may be shown in popups inside the editor window.

- **If you (by accident, or intention) identified 2 shapes with the same character, you will be told which character and where the Rects are located. If you have 2 characters the same in a font, only one will ever be rendered.**

- **If any shapes were so close that their bounding Rects overlapped, you will be told which characters. You should then go back into your image editing program, and separate them, or you will end up with characters being drawn with orphaned pixels. After adjusting the font map, run the Auto Set again; to save time you can simply press [Fig4. (7)] and it will identify the shapes in the same order as before. [1]**
- **Clicking the buttons inside these prompts will select and move the view to the respective Rects.**

After the Auto Set has finished, your font is now ready to roll. If you had [Fig2. (6)] checked, you should find a 3DText object called "Font Setter Preview Text" in the scene, with the font displayed. Carefully check this to see if any of the Rects need adjusting, any of the letters are missing, etc.

However, to get best results from the Auto Setter, your font map is probably quite space inefficient (see Section 9), so it should be Packed (see Section 8).

While this Auto Set mode is probably sufficient for simple fonts, like pixel fonts, or fonts with a consistent character height...

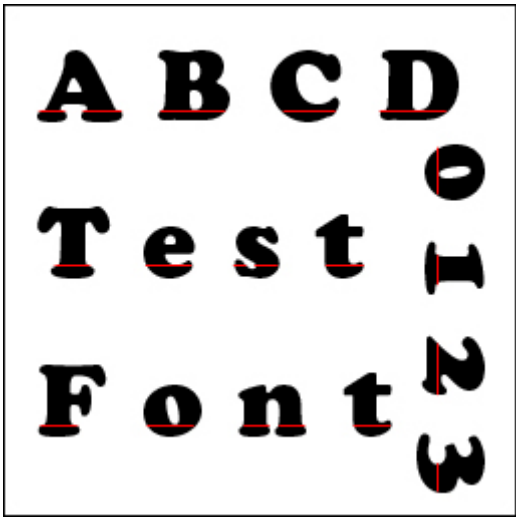
...it does not set vertical offsets for each character (see Section 1). When your font is more complicated with tall and short characters, you will probably want to use the Smart set (see Section 7).

---

1 **The Auto Set process identifies shapes roughly from the top down, left to right. If you change your font map such that one shape is now higher than before, the order in which it identifies the shapes will change. For making small adjustments, you should only move a character left and right, and should preserve their order.**

7. Smart Set.

For Smart Set to work, you need to make an extra edit to your font map, draw a single pixel line over each character, such that the line is the same vertical height on every letter and number. Set the Smart Color [Fig3. (21)] to this same color, and the Auto Setter will detect those pixels, and automatically assign the character vertical offset. Additionally, it will detect if any of the characters are rotated, and set the relevant properties.



Ensure that this line does not extend past the bounding rectangle of the character, or the Auto Setter will assume the character is bigger than intended. If you mask the line with the shapes of the characters, ensure that the masks are sufficiently opaque that the color is 100% itself, not semi-transparent, or the Auto Setter will not treat it as special. For the same reason, make sure there are no layers above the line layer that may change its color (eg. transparent effect layers).

When you click the Smart button [Fig3. (21)], the program goes straight to the Auto Set procedure shown in (Section 6), it has no additional steps.

Once the Auto Set is complete, and you are happy with the Rects (check the preview 3DText [Fig2. (6)]), you should then go back to your image editing software and remove the line layer(s).

8. Font Packer.

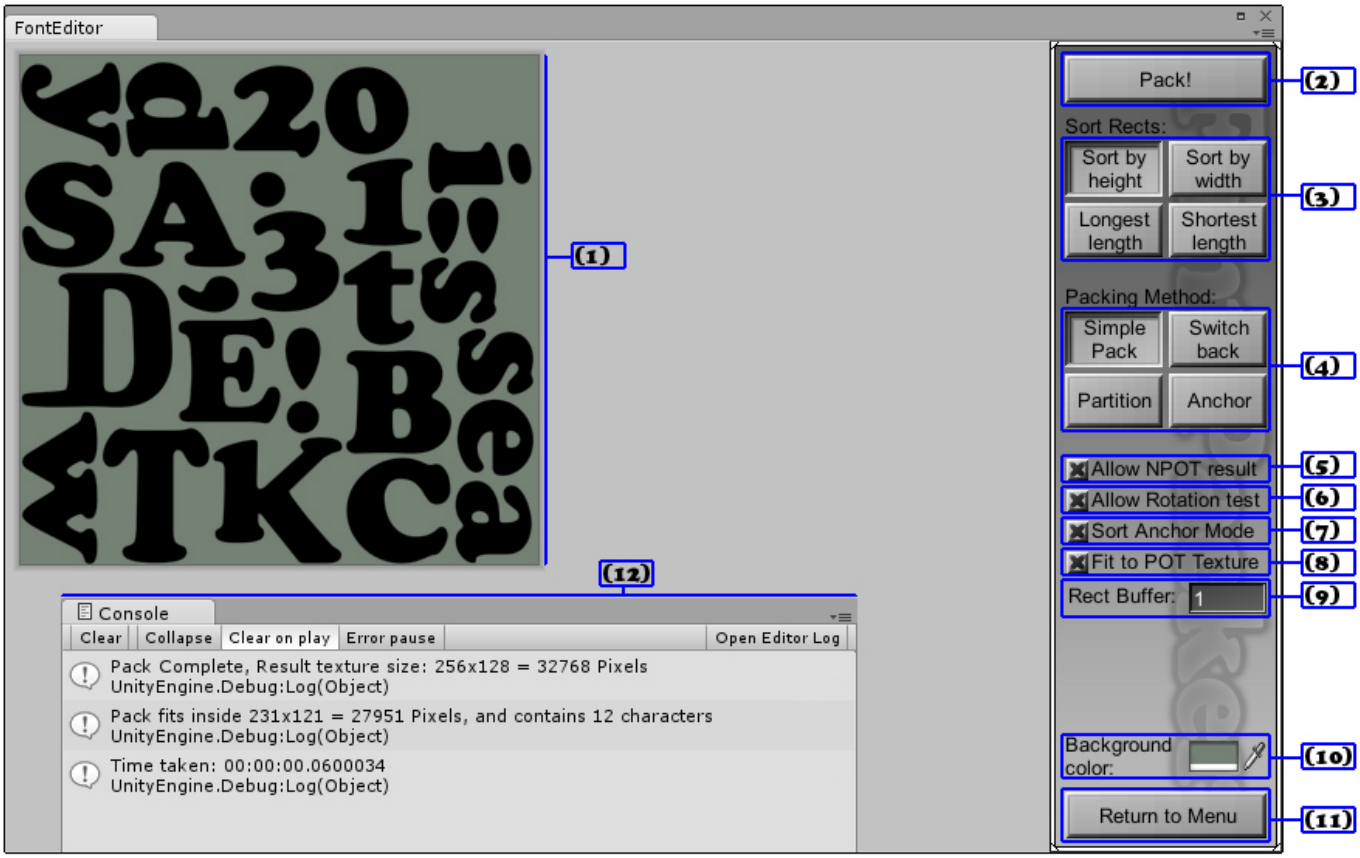


Fig5. The Font Packer Interface

Note:

Back in the main menu, you must assign your font bitmap to [Fig2. (7)], the font settings object that you modified in the Font Setter to [Fig2. (8)] and a second, different font settings object to [Fig2. (9)].

The Font Packer does not modify the first font settings object, but does reset and clear the second, so please ensure you assign the the correct one to each slot. This is important, in case you wish to go back into the Font Setter to make adjustments.

(1) The finished packed font map is displayed here (after running the packer). Pan with the right (or middle) mouse button.
(2) Press "Pack!" to run the packer according to the settings defined below.
(3) The packer inserts the Rects in a sorted order. Choose here which property to sort by (see below for recommendations).
(4) The packer has 4 different packing algorithms, each suited to different needs, choose the method here (see below).
(5) Tick this box to dis/allow Non-Power-Of-Two results. If unchecked, the resulting bitmap will always be POT; if checked, the resulting bitmap size is determined by the packer, or forced to POT by [Fig5. (8)].
(6) Tick this box to dis/allow the packer rotating characters after it has sorted them (different sort modes will rotate characters before any packing is done).
(7) Tick this box to change how Anchor Mode packs Rects (no effect on the other modes).
(8) Tick this box to force the resulting bitmap to be POT. If [Fig5. (5)] is ticked, this can result in blank areas (in some cases this may be beneficial, see [Fig6.]).
(9) Set a value here to space the Rects apart. This can prevent texture bleeding, but will result in a larger bitmap.

(10) Change the background of the bitmap display. If your font has a lot of grey in it, changing the background color can help visibility.
(11) Clicking "Return to Menu" and confirming will return you to the main menu.
(12) When the pack is complete, it writes a few lines to the console, to inform you how large the texture is, how large the actual pack is, how many characters were drawn, and how long the pack took (larger, complicated packs with many characters could take longer).

Once completed, you should find, in the same directory as the input font bitmap, a newly packed bitmap. You may need to Refresh the project window (Ctrl+R), and you should refresh after subsequent uses of Font Packer, to make sure it re-imports the new bitmap.

Recommendations.
Depending on your input font map, different sort and pack modes will give varying results. For a font with many different sized characters, Anchor pack seems to give the most efficient and aesthetically pleasing results. For a font with mostly similar sized Rects, Simple pack will probably give a better result. Sorting by shortest length generally seems better. While allowing NPOT results will produce a smaller image, Unity will automatically resample NPOT images up to the nearest power-of-two if you use the texture on a non-GUI object (ie. A TextMesh), regardless of the texture import settings.

http://answers.unity3d.com/questions/386490/how-can-i-get-unity-to-stop-resampling-my-sprites.html

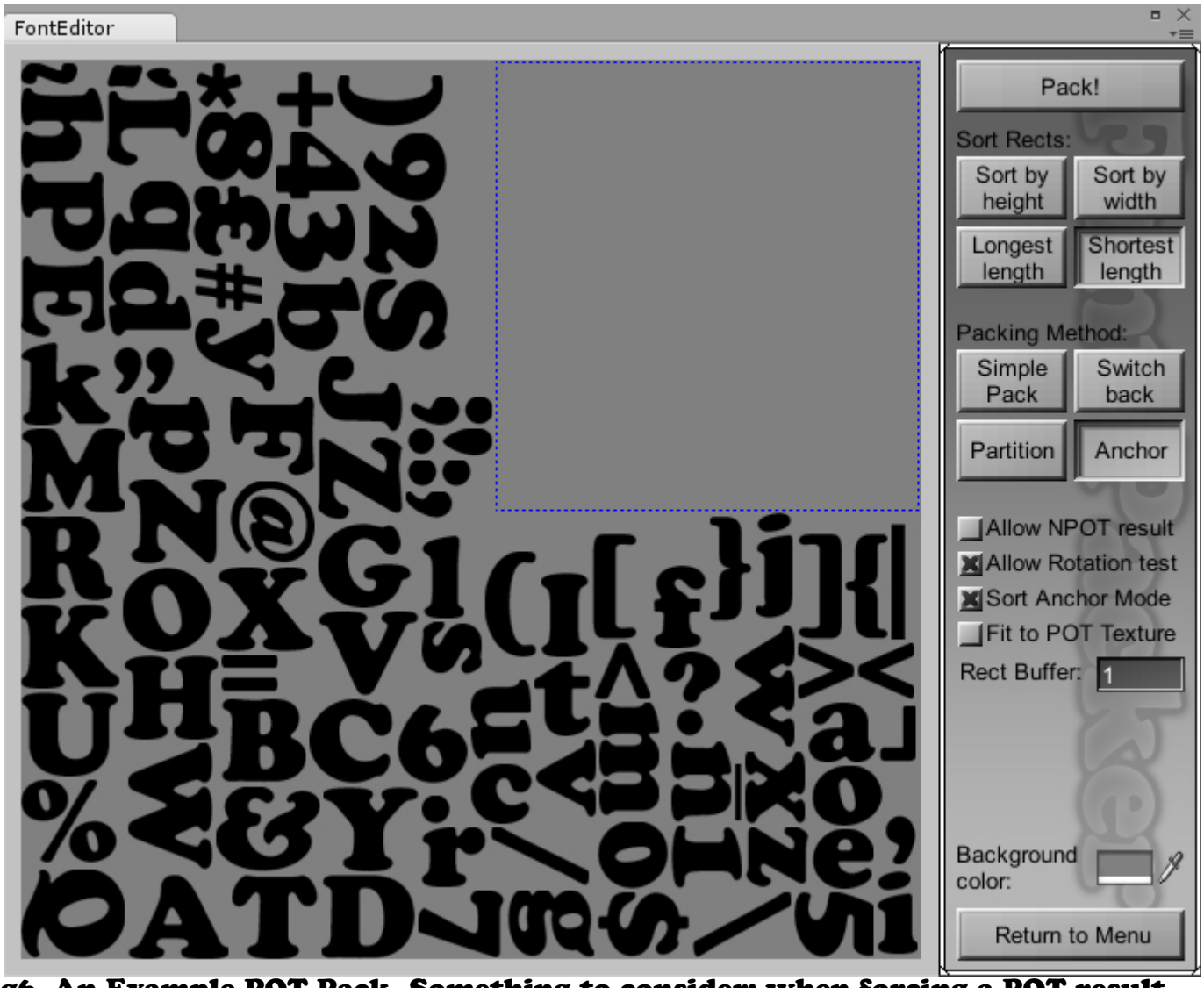


Fig6. An Example POT Pack. Something to consider: when forcing a POT result, like this, that's a big square space which could possibly be atlas'ed with another bitmap for use elsewhere in your project.

9. Font Creation Procedure Overview.

From start to finish, this is an outline of the process you can follow to create a full-color font ready for use in your project.

For a video tutorial, visit:
http://www.youtube.com/watch?v=K_E2gElbqr8

- **Using Photoshop, start with an image with a very large width, Unity can only take images up to 4096.**
- **If using an existing font as a starting point, type in all the letters, numbers, and symbols you need in the font. If making the font entirely from scratch, this step will take a bit longer.**
- **Now you have the shape of your font, you can use it as a mask for other layers.**
- **Once artistically complete, add a layer to the top, and draw a single pixel line horizontally over the whole image, so that it is at the same vertical position on every character. Mask this line to the same shape as the characters.**
- **Save in a format that respects transparency.**
- **Ideally the images should respect power-of-two sizes (256, 512, 1024, 2048).**
- **In Unity, adjust the import settings of the image so that Read/Write is enabled, and no compression is applied.**
- **Create the required Font and Material assets according to (Section 2.).**
- **Start the editor extension, assign [Fig2. (3)] and start Font Setter.**
- **Start the Smart set, with the correct color assigned in [Fig3. (21)].**
- **Identify every character as prompted.**
- **Fix any issues as directed by the prompts.**
- **Add a new Rect for the 'space' character.**
- **Return to Photoshop and turn off the Smart line layer(s), and re-save.**
- **Switch to the Font Packer, via the editor extension main menu, assigning [Fig2. (6 & 7)].**
- **Try different settings to achieve the most efficient pack.**
- **Refresh the project browser to get at the packed bitmap.**
- **Adjust its import settings for Read/Write and NPOT as needed.**
- **Close the editor extension.**
- **In the font settings object, adjust the line spacing.**

The font object can now be used in any object that requires a font, 3DText TextMeshes, GUIText objects, or used via GUIStyles for OnGUI Labels, etc.

10. Bonus: Textured GUI Label

It is possible to emulate UnityGUI bitmap font text, by sequentially calling GUI.DrawTextureWithTexCoords. Included with the Font Setter/Packer package is a small example script to do just this.

With the 'TexturedGUILabel' script in your project, anywhere inside OnGUI you want a bitmap font label, call...

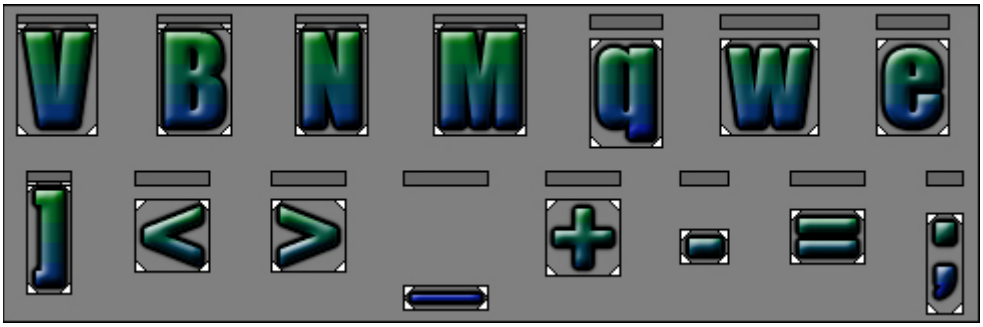
```
new TexturedGUILabel(new Rect(x, y, width, height), "text here", someFont);
```

...to draw your font using UnityGUI.

However, as this is just an example script, this method has a few drawbacks and stipulations.

- **One drawcall per character, a 20-character string will result in 20 drawcalls. This is compared to a single draw call using normal OnGUI Labels, GUITexts, or TextMeshes.**
- **In order to draw characters which are rotated (Character orientation in [Fig3. (15)] is changed), this TexturedGUILabel method uses GUIUtility.RotateAroundPivot to re-orient the character. If the font bitmap uses bilinear or trilinear filtering, the rendering quality suffers for characters that need to be rotated, and there will be an obvious difference in rotated vs non-rotated characters. Set the bitmap filter mode to point to correct this. (In this image, the font has all characters rotated except the 'i' and 'r'. The top line uses bilinear filtering, and the chracters except the 'i' and 'r' appear blurry; the bottom uses point filtering, all characters appear crisp.)**

- **No multi-line labels, currently there is no way to access line spacing programatically to increment vertical positioning.**
- **For optimal GUI positioning, the vertical offset sliders [Fig3. (4)] for each character should be position towards the top of the character, so that UnityGUI can 'hang' the characters from the line (see Section 1).**



11. Bonus: Sprites

Since the Custom Font object in Unity is really just a container for uv rectangle definitions, the Font Setter can also be used to manage animated sprites. Subsequently, you can use a 3DText GameObject to draw the sprites, and utilise multiple fonts to hold different frame sequences (animations).

You will need to set up your assets as shown in [Fig7.].

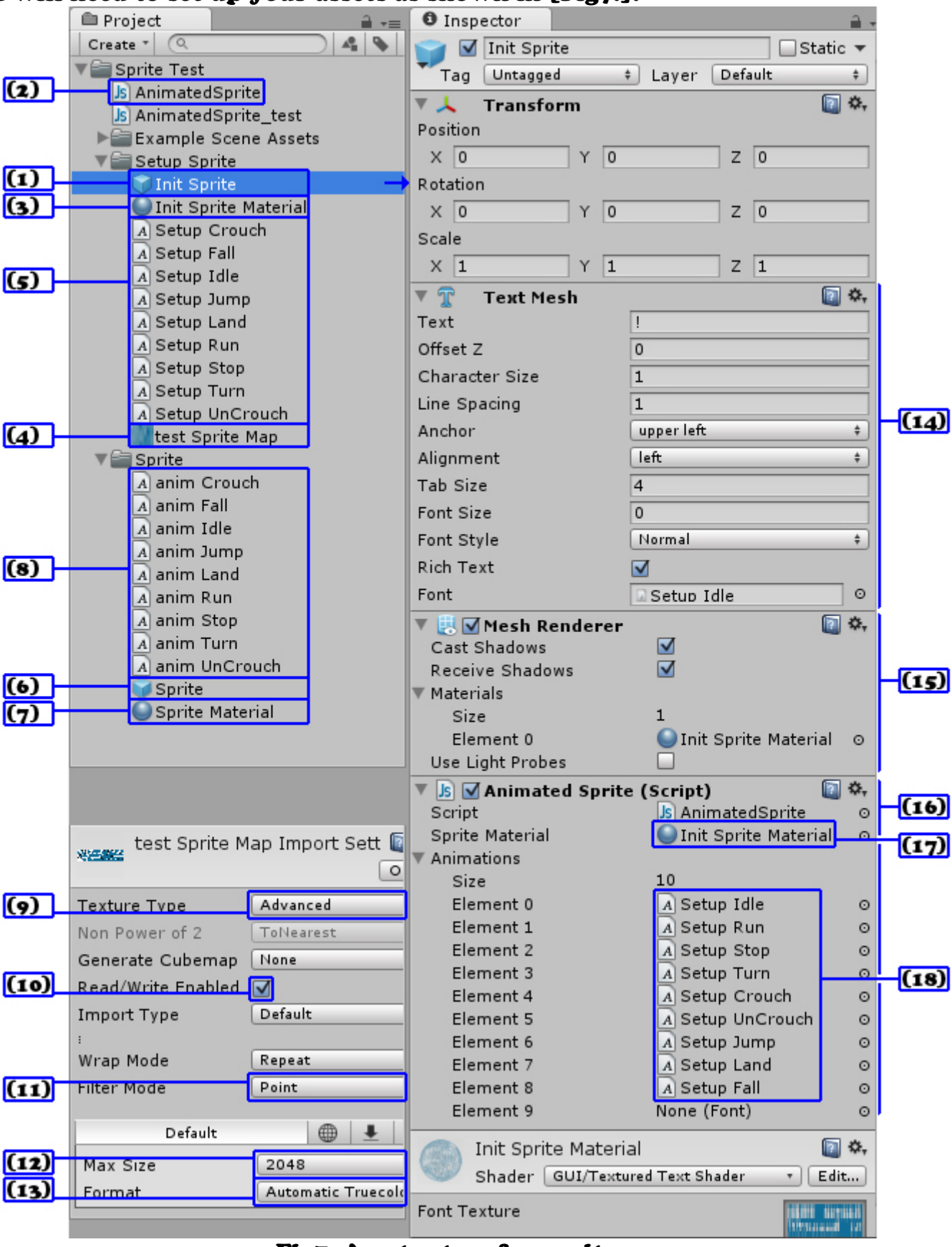


Fig7. Asset set-up for sprite use.

You will need

- **(1) a prefab GameObject, with (14) TextMesh, (15) Mesh Renderer and script (2, 16) Components. The project includes "Animated Sprite" as an option.**
- **(3) a Material, with a (4) sprite map assigned, using the Textured Text Shader. The sprite map should have import settings as shown**
  - **(9) Type will need to be set to advanced to show the other options.**
  - **(10) Tick Read/Write Enabled.**
  - **(11) Sprites probably want to be pixelated, so use the Point filter.**
  - **(12) Increase the Max Size if needed. Note: If the bitmap is not sized to powers-of-two, Unity will automatically re-sample it, and you will get artifacts on your crisp, low-resolution sprites.**
  - **(13) To preserve colors, use Truecolor or another RGBA32 format.**
- **Assign the material into (17) the Animated Sprite component material field.**
- **(5) Several fonts to act as animations. You will need one font per animation. Assign these into (18) the "Animations" array in the Animated Sprite component (this is a Font array).**
- **If you wish to pack the sprites into a smaller bitmap, you will need a second set of the above, (6) (TextMesh, Renderer, Animated Sprite)Prefab, (7) Material and (8) font collection.**

With that done, In the FontSetter Main Menu (Section 3) toggle on "Sprite Mode" [Fig2. (9)], and drop the sprite prefab into [Fig2. (3)] and click start to begin sprite editing.

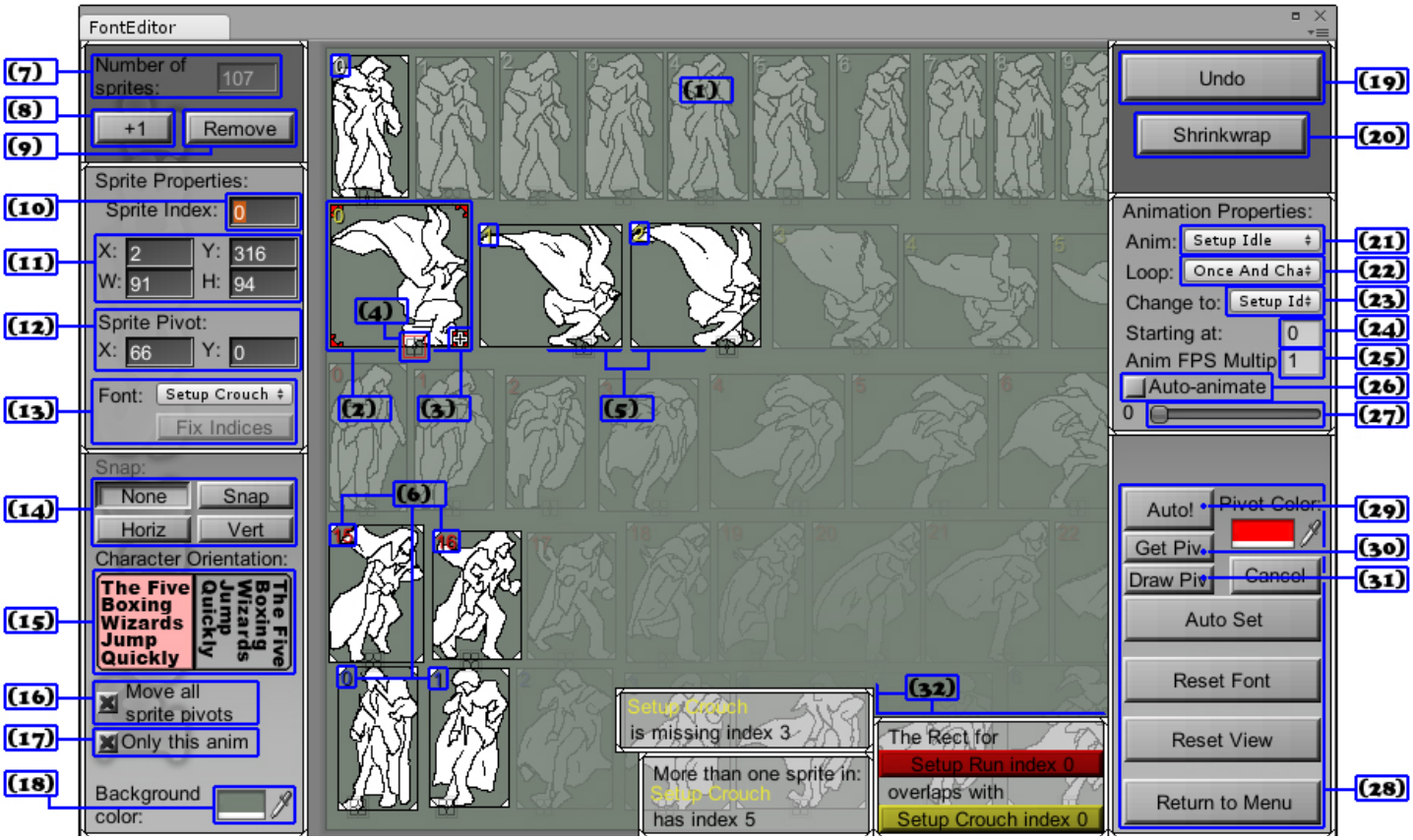


Fig8. The Font Setter interface, with sprite specific controls.

(1) Your sprite map displays in the main portion of the window, at full resolution. Click and drag with the right mouse button to pan, and use the scroll wheel to zoom. (Most sprites are greyed out in this document purely for readability)
(2) This is the currently active/selected Rect (red corners). Click and drag with the left mouse button to change its position.
(3) Click on the resize handle to change the size of the currently active Rect.
(4) This target marker represents the sprites' pivot; where it is drawn in relation to the GameObject transform.
(5) This is an unselected Rect (white corners).
(6) Every Rect shows which animation frame it is (zero-based). These are color-coded by animation. The error prompts (32) use the same color scheme.
(7) This field displayed how many sprite Rects are currently defined.
(8) Click this button to add a Rect. If no Rects are selected, a new one will appear towards the bottom left of your bitmap. If a Rect is selected, the new Rect will duplicate its size, and appear next to it, be in the same animation, and increment its' frame index.
(9) This button removes the currently active Rect, alternatively, press delete on the keyboard.
(10) Type into this field to define the frame index for the currently active Rect.
(11) This group of fields allows you to type in the X,Y position of the current Rect, and the width and height. The units are measured in pixels, in relation to the sprite bitmap, with X=0, Y=0 at the bottom left corner.
(12) Here you can see and set the sprite pivot. Measured in pixels from the bottom left of the sprite Rect.
(13) The drop-down list lets you select which animation the currently select Rect belongs to. If the selected animation has index errors (missing index or duplicated index), the "Fix Indices" button is enabled, and clicking it will trim the indices into a zero-based sequence.
(14) These buttons let you snap Rects together:

- **None: No snapping.**
- **Snap: Snaps the corners of Rects together, for both moving and resizing.**
- **Horiz: Stops you from moving or resizing vertically.**
- **Vert: Stops you from moving or resizing horizontally**
- **Snap also affects the sprite pivots. In snap mode, the pivot marker will snap to the corners and centers of it's Rect.**
- **Note that when resizing the currently selected Rect [Fig3. (3)], if the Rect is already snapped on one corner, it will appear to try and snap to itself if you resize on a different axis.**

(15) If any of your sprites are rotated 90 degrees clockwise, (ie. they are sideways) change the Rect orientation here. Note: Doing so changes the height property of the Rect to a negative number.
(16) Toggle this on to enable moving all sprite pivots at the same time. This has some extra interaction with the snap buttons (14):

- **None: Pivots can be moved off the pixel grid (decimal values for the pivot).**
- **Snap: Snaps all pivots to the corners or centers of their own rects, not where you are snapping the currently selected one.**
- **Horiz and Vert: Moves all pivots, but preserves the pixel grid spacing.**

(17) If (16) is on, this toggle will let you move only pivots in the same animation as the currently selected Rect.
(18) Change the background color here. The default is grey-green, so if your sprite bitmap has a lot of this color, you might want to change the background.

(19) Click to undo the most recent change.

- **Note: Undoing any other way (in the main unity windows, or Ctrl+z) will also undo changes made to the sprites font objects using the editor.**
- **Note: This button will also perform an undo for any action made outside the Font Editor window, be careful not to accidentally undo important changes made to other objects.**

(20) Click "Shrinkwrap" to automatically resize all Rects to the minimum size needed to encompass the bitmap shape they are covering. Rects that cover no shapes are not affected. This should only really be needed when manually settings Rects, if you have used an auto set mode, (see below) all Rects should already be the correct size.

The panel on the right (21-27) contains controls that affect the entire animation, not the currently selected frame (Rect).

(21) Use this drop-down list to select an animation.

(22) Choose a loop style from this list:

- **Loop: When the animation reaches the end, start playing again from the frame set in (24).**
- **Ping Pong: Play forwards, then change to playing backwards.**
- **Once and Hold: Play the animation once, then freeze on the final frame in the sequence.**
- **Once and Change: Play the animation once, then switch into playing a different animation (choose below).**

(23) When (22) is set to "Once and Change", this list appears, allowing to select which animation follows next.

(24) When (22) is set to "Loop", "Ping Pong" or "Once and Change", this field allow to you select which frame is considered the 'start' of the loop or animtion.

(25) This field allows to you set a multiplier for the fps playback speed. Values less than one will result in slower playback, and greater than one will cause the animation to play faster (eg, if the sprite has an FPS property, and it is set to 10 (frames per second), and "Anim FPS Multip" is set to 1.5, the result for this animation will be 15 fps).

(26) If [Fig2. (5)] was enabled, in the scene should be an object called "Sprite Setter Preview Sprite". With (26) toggle on, this object will play the animtion selected in (21) according to the properties defined by (22-25).

(27) Drag this slider to change which frame is shown by the preview object (if "Auto-animate" is not on). You can also use the . and , buttons (< and >) to move forwards and backwards in the animation.

(28) The left side buttons expand in place, and function as described in (Section4), with exception of the contents of the Auto Set menu.

(29) Clicking this will start the auto set for sprites (See Section 12.).

(30) Clicking this will move all sprite pivots to the first found pixel with the color defined by "Pivot Color".

(31) Clicking this will create a bitmap in the same directory as the sprite bitmap, with the pivot locations marked with a pixel of the color defined by "Pivot Color".

(32) Various error prompts can appear in the window. These are color-coded with the same color used by (6).

- **Overlapping Rects can result in unwanted pixels being drawn.**
- **Animations with missing frame indices will result in blanks being drawn when animating.**
- **Animations with duplicated frame indices will only ever drawn one of the frames, the other(s) with the same index will never be played.**
- **Clicking the buttons in the overlap prompt will select the relevant Rects.**

12. Bonus: Advanced sprite functions

The auto set for sprites is slightly different to when in font mode.

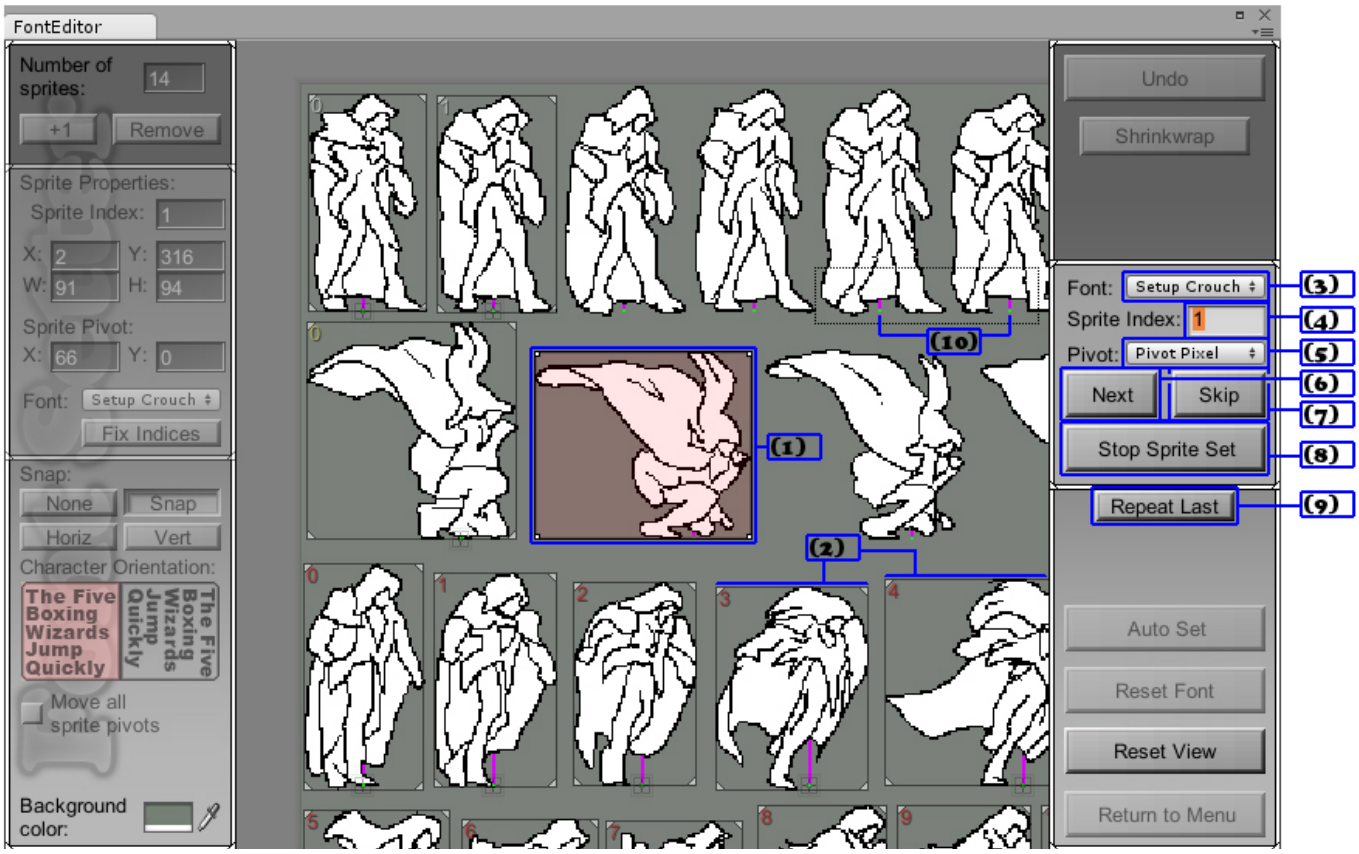
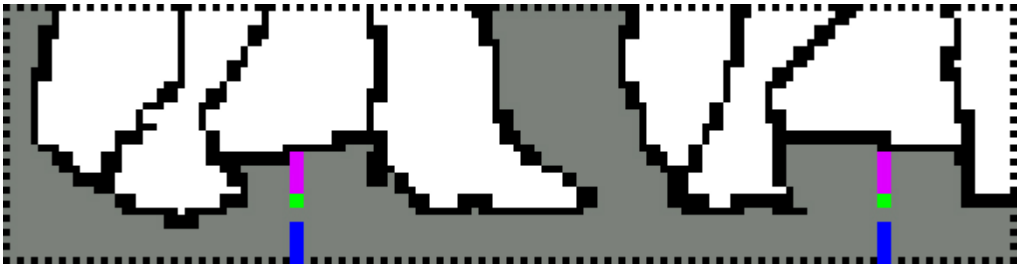
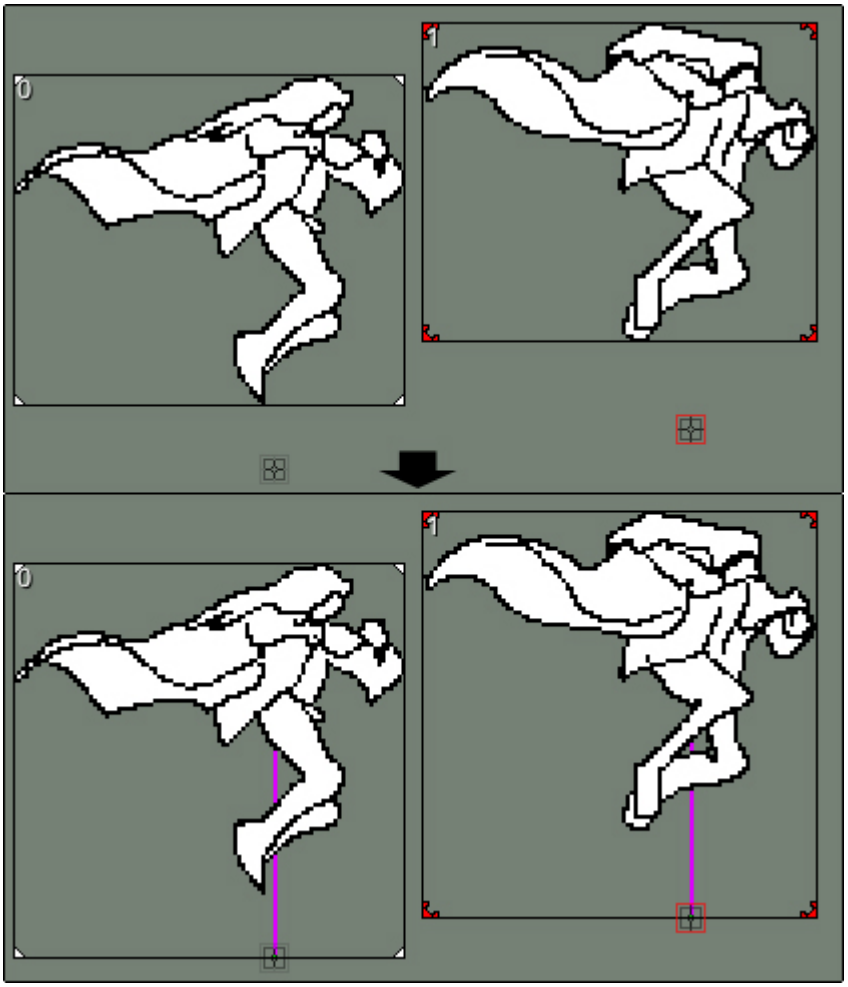


Fig9. The Sprite mode Auto Setter.

(1) This is the currently queried shape (highlighted red).
(2) These are shapes already identified, with their pivot markers in the correct position as indicated by the pivot pixels (see (10)).
(3) Choose from this list which animation the highlighted sprite belongs to.
(4) Use this field to indicate which frame index the editor is querying. This field auto-increments as you click "Next", and auto-zeros when you change animations.
(5) Use this drop-down list to choose where the sprite pivot should be. In addition to "Pivot Pixel" (shown) there are the standard placement options (top-left, middle-center, bottom-right, etc).
(6) Click "Next" to identify the current shape, and move to the next. Alternatively, press Enter.
(7) Click "Skip" to skip to the next shape without identifying the current one. Press Escape to skip, without the need to click.
(8) Clicking "Stop Auto Set" and confirming will end the Auto Set procedure.
(9) If you have previously attempted to auto set the sprite sheet during this session, you can click "Repeat Last" to automatically identify all the sprites in the same order as before.



(10) The green pixels pictured here (zoomed in to [Fig9.]) are the "Pivot Pixels". If you have "Pivot Pixels" chosen during the auto set, or are using "Get Piv" [Fig8. (30)], this is where the sprite pivot is moved to. It should be only a single pixel on the sprite sheet, and in cases where they are not directly joined to the pixels of the sprite, you will need to draw in a few extra (the purple pixels in this image), just to make them a part of the same 'shape'. After using auto set, or "Get Piv", you should return to Photoshop and hide these pixels. You may also then ned to shrinkwrap [Fig8. (20)] to reduce the Rect size, if the pivot is somewhere outside the normal boundary of the sprite.



With all the sprites defined by a Rect, and all animation properties set as desired, you can now use the packer to pack the sprites into a smaller bitmap. The procedure for this is the same as described in (Section 8.), except you must assign the sprite prefab objects as created at the start of (Section 11) in the Main Menu.

Lastly, to turn the animated sprite into a controllable player character it's simply a case of adding a character controller component, and enhancing the script. Included with the Font Setter project is AnimatedSprite_test, this script has a simple set of instructions that takes user input, converts it into movement and calls to a function which changes the font used by the TextMesh component. If you wish to create your own alternative script to handle sprite animation, but also want it to work with the Font Setter editor extension, check the AnimatedSprite_test script for information on how to achieve this.