Unicorn HD Synth Eyes

Updated 2022/01/24

For this project you will need the following:

2x Raspberry Pi Zero **without the headers installed**
2x Pimoroni Unicorn Hat HD display panels (make sure it's the HD version)
2x MicroSD cards 2GB or larger
1x 2-pin jumper cap
...and a computer with an SD card reader of some kind.

The Unicorn HD panel is designed to plug directly into a Raspberry Pi. Pretty much any Pi will work, but to save space I recommend using the Pi Zero 2W boards. Development was originally done on the older 1.3 boards and these will work, but the Pi Zero 2 boots a lot faster.

If, down the line I can figure out a way to drive multiple panels off a single board things will be a lot simpler. For now, we need to use two Pi Zeroes, and in order to link the two machines together (and provide inputs for changing expressions) we need to customise the 40-pin headers.

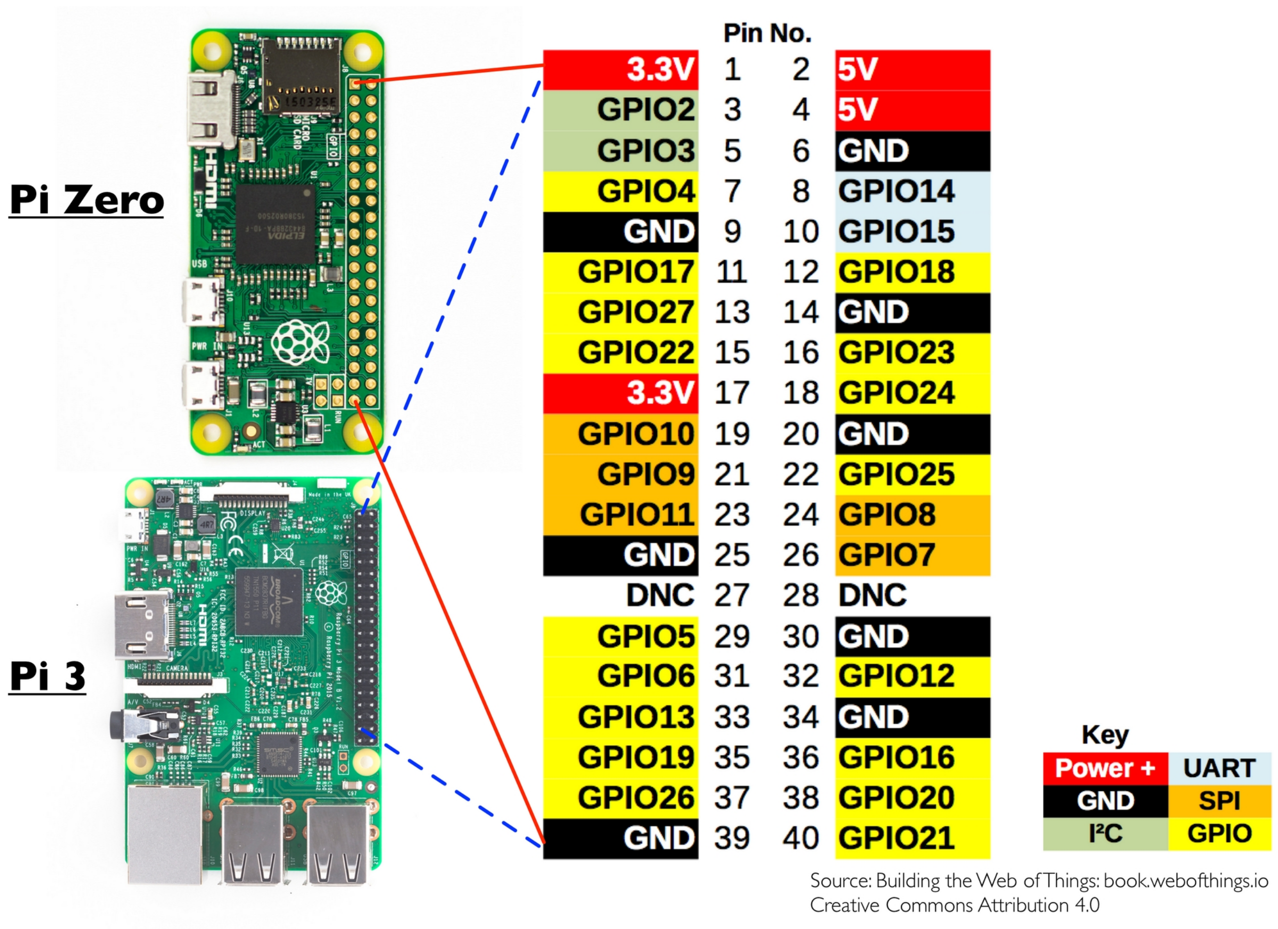
For this reason it is highly recommended that you purchase the boards with the headers provided, but *not* pre-installed (they are usually cheaper like that anyway). Otherwise, you will have to desolder some of the header pins to rearrange them and this is likely to damage the Pi unless you are very careful.

Because we have two separate devices, one of them will need to be the transmitter, and the other one must be configured to listen for commands.

With the W model of Pi Zero it might be possible to do this wirelessly, but I would expect problems if you have two such Synths next to each other – also it is likely to break down in case of heavy interference such as a furry convention full of bluetooth devices.

Wiring the headers

Here is the pinout for the Raspberry Pi Zero:



	Pin No.		
3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7
DNC	27	28	DNC
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

Key

Power +	UART
GND	SPI
I²C	GPIO

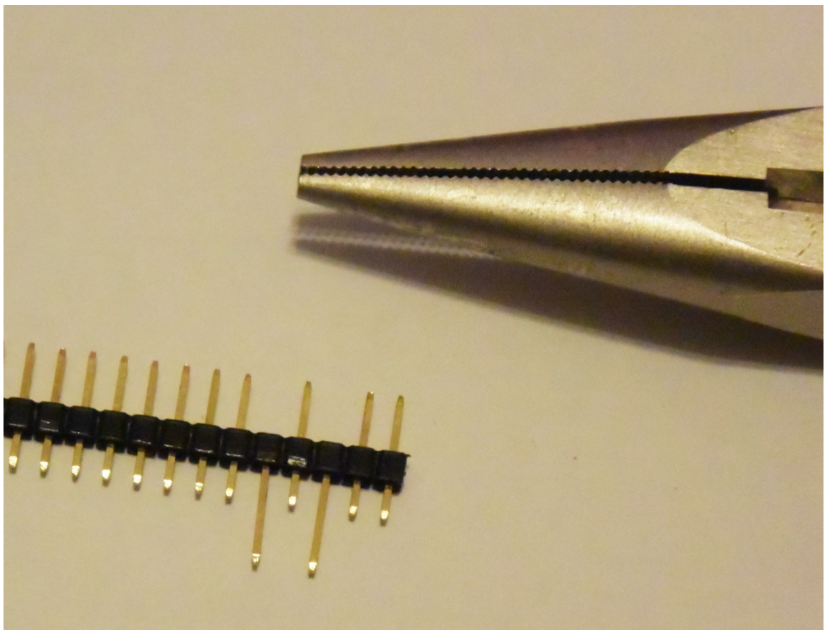
Source: Building the Web of Things: book.webofthings.io
Creative Commons Attribution 4.0

We need to wire both devices slightly differently.

Wiring the Receiver

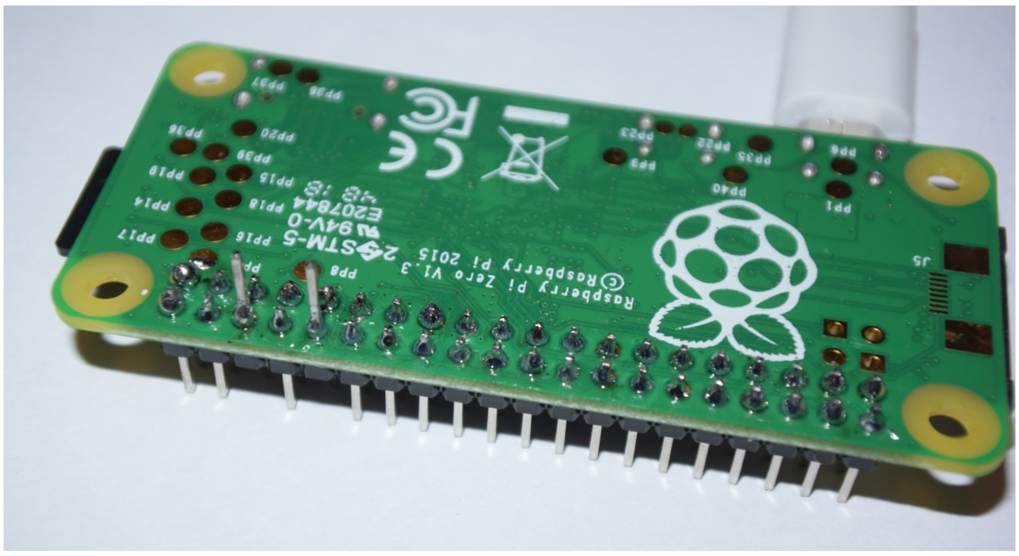
The Receiver is easiest so we'll look at that first. The above picture shows the top of the device – the header is inserted this side so the pins are sticking out the top. However, to provide a link to the other device, we will need to have two pins sticking out the bottom as well.

Use needle-nosed pliers or some other tool to push pins 6 and 10 into the black frame, e.g.



Make absolutely sure that it is pin 6 (GND) and not pin 4 (5V)... If you connect 5V to the other Pi, you will most likely have destroyed that I/O pin and will need to buy a replacement Pi.

Then, solder all 40 header pins onto the board. When it's done, it should look like this:



You may find it helpful to tack an LED to pins 39 and 40 on the back of the Receiver board. This will flash briefly to indicate that an expression change command has been sent, since it may not be obvious to the wearer if they managed to trigger it successfully. Wire negative to pin 39 (GND).

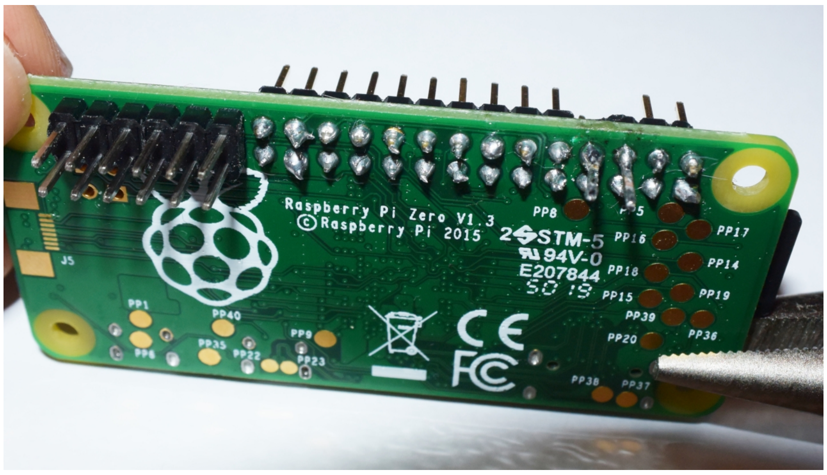
Wiring the Transmitter

Next, the Transmitter. This is more complicated. As before, you will need to push a number of pins through the connector so they come out of the bottom of the device.

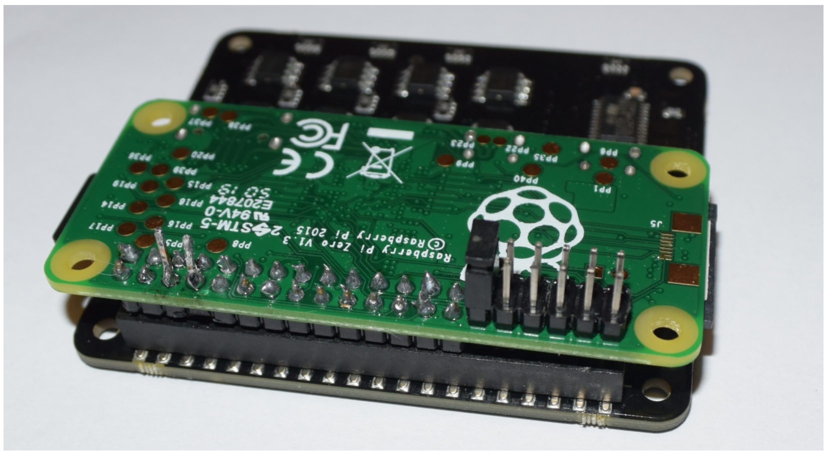
This time we need pins 6 and 8 – which are adjacent to each other. Again, be sure it's neither of the first two 5V pins since the Pi can't handle 5V logic levels.

1. In addition, we also need access to pins 29-40 – the 6 pairs of pins furthest from the SD card reader. We need these pins to be poking out the bottom of the circuit board, but because they are all grouped together it is best to cut the header at that point with a pair of sidecutters so that these 12 pins can be soldered on the rear of the board.

Again, solder all 40 pins. When it's done it should look like this:



At this point you can plug the Unicorn Hat boards into the Pi Zeros, e.g:



Now. Both Pi Zeros are running the exact same software. To tell the Transmitter unit that it's going to be the transmitter, we need to bridge pins 29 and 30. You could solder these two pins together, but I use a jumper cap of the kind often found in Desktop PCs – as shown in the above picture. This will be more future-proof in case we need those pins for some other purpose later on.

Expression Control Pins

Pins 34 and 39 are ground. If you temporarily bridge one of these pins to one of the surrounding pins it will trigger an expression change, as with the Arduino software.

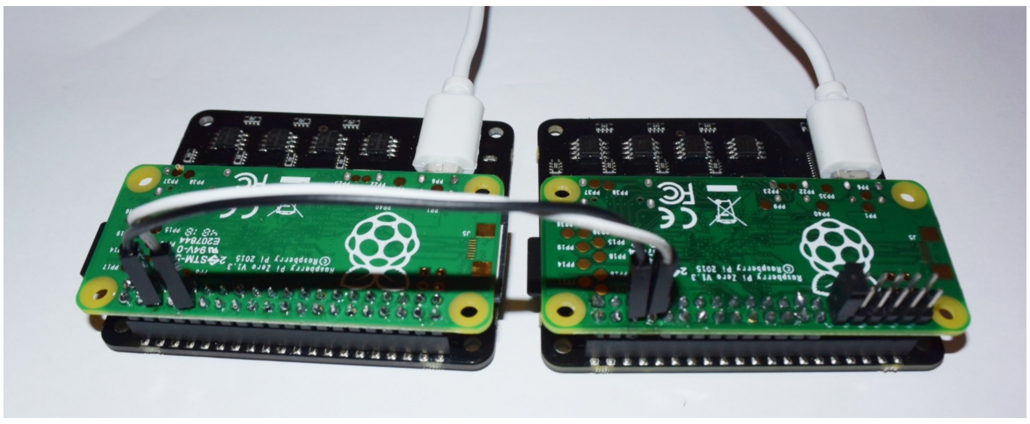
As of this writing, (with v3.00 of the software) the following pins do the following expressions by default, depending on the config file used:

Pin	**Xerian**	**Raptor's Den**
29	-	-
31	OwO	OwO
32	Annoyed	Annoyed
33	Eyeroll	Annoyed (no eyeroll yet)
36	Fault (X eyes)	Fault (X eyes)
37	Blushing	Blushing
38	Happy	Happy
40	Startled	Startled

Note that pin 35 is configured for *output* – do not short it to ground! It can be used to change the colours of the status lights in 'Fault' mode to make them flash red (if the Arduino driving the Synth's status lights is programmed to do this).

Linking the two devices

The last step hardware-wise is to connect the two Pi Zeroes together. This is fairly straightforward: Connect pin 6 on one Pi to pin 6 on the other. Then connect pin 8 on the transmitter to pin 10 on the receiver. Unless you get them crossed over this should be pretty foolproof.



OS installation

Now we need to install the OS onto the SD card. Copying the file won't work, you will need to use something like Raspberry Pi Imager to write the system directly to the SD card.

Download my custom image from the following location:

```
https://it-he.org/down/pi-eyes-sdcard-v3.7z
```

...it's about 300MB. Unzip it with 7zip and it will expand to about 2GB.

Inside Raspberry Pi Imager, select 'custom image' (near the end of the list) and choose the 'pi-eyes-minimal.img' file. Write this to both micro SD cards.

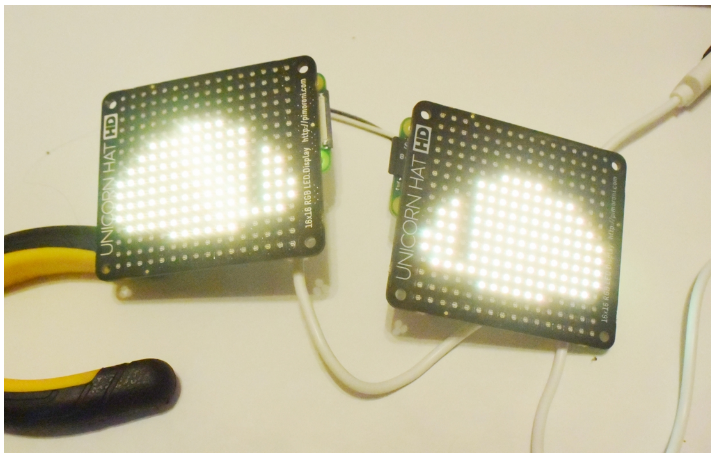
Once the system image has been written to the SD card there will be a small boot partition which will be visible to Windows (the main OS partition isn't because Windows is still stuck in 1989). If you need updates or customisations, I can provide an updated 'eyes' executable, which you can then copy on to the boot partition to update the software. You must overwrite the existing 'eyes' file – copying the file with another name won't work. (Remember to copy the file to both SD cards!)

Powering up

Now, insert the SD cards into the Pi Zeroes, and give them power via the MicroUSB port marked 'Power' (not the other one, which is for peripherals).

A green light should flash during boot. As of this writing the OS boot takes about 30 seconds on a an original Pi Zero – the Zero 2 takes more like 13 seconds, a significant improvement.

Once it's finished, the Unicorn display should light up, showing the version number of the synth software for about 2 seconds, and then displaying the eye which should blink on both devices.



During the boot screen one of them should have the word 'rcv' at the bottom, and the other should say 'xmit'. If they both display the same message, double check that pins 29 and 30 are joined (and only on the Transmitter!). If the receiver displays the eye but never ever blinks, make sure that the link is OK (pin 6 is connected to pin 6, and pin 8 to pin 10). The receiver will also be stuck displaying a stationary eye if the receiver is powered on and the transmitter isn't.

Customisation

The V3 software will look for the file eyeconfig3.txt in the boot partition. It will not read the old eyeconfig.txt file as it works differently. The new config file describes the animations and assigns them to GPIO pins on the hardware.

The actual graphics are loaded from animated GIF files (or static .GIF files for that matter). The exact location is set in the config file, but by default this will be `/boot/videos/raptors_den` . Note that there are currently three sets of video files, for Xerian, Raptor's Den and some miscellaneous sprites for demonstration purposes.

Because of this, it is now possible to customise the display yourself without needing to recompile the software as was the case with SynthEyes2.

The drawback is that the configuration is more complex, and the V3 software is unable to start without it. To assist in setup, the software will display a scrolling error message if there is a major problem with the config file.

There is also a desktop version of the software which can be used for testing. However, as of this writing it is only for Linux – a Windows port may appear in future, otherwise you might be able to run the Linux version through the Windows Subsystem for Linux.

Lines of the config file starting with # will be treated as a comment and ignored. Commented-out examples are provided for most of the supported commands – remove the leading # to enable them.

The next part of the documentation will describe the config file commands. You can probably get started by editing the examples in `/boot/examples/` on the SD card.

Config File Reference – General Setup

GIF directory

```
gifdir: ./videos/raptors_den
gifdir: /boot/videos/raptors_den
```

'gifdir' specifies the folder where the GIF files are stored. Since the SynthEye software runs on Linux, you must use the forward slash separators (e.g. `/boot/myfile.txt`) rather than the backslash separators used in Windows (e.g. `\boot\myfile.txt`).

You can have multiple gifdir: commands in the same config file – the last one that exists will be take precedence. This can be useful when testing the animations on the desktop software, since you probably won't have write access to /boot/ on a PC.

The above example does this: ./videos/raptors_den will work on the PC but fail on the Raspberry Pi SD card, likewise /boot/videos/raptors_den will typically only work on the Pi. As a result, the software ignores the one that doesn't work and uses the one that does.

If there there is no valid path, the software will halt with an error message once it tries to load in the animations.
If the gifdir: command is omitted entirely it will default to /boot/

Eye colour

```
eyecolour: #001122
```

Eyecolour will set the default colour of the eye animations, as with V2.

Note that because V3 uses animated .GIFs, this will only apply to animations which have the 'drawmode: monochrome' command dicussed later. Otherwise it will use the built-in colours from the GIF file.

Colours are specified as a hexadecimal red-green-blue triplet, the same as HTML. Xerian's default warm yellow colour is, e.g.

```
eyecolour: #ff8f00
```

(full red, about half green, no blue). #0080ff will provide a cyan eye.

Most art software such as Krita or GIMP provides a colour picker in this format and there will be colour charts available online as well.
Note, however, that the display's colour rendering is likely to differ wildly from your monitor, so some experimentation is likely to be needed.

Remember to copy the same config file to both SD cards, unless you want the Synth to have heterochromic eyes.

Cooldown

If the user holds down the expression change switch too long it would immediately retrigger the expression animation, which can look a little strange. To avoid this a cooldown timer has been implemented to help avoid running the same animation twice by accident. (Running a different expression animation should happen immediately – the lockout is for the same animation happening twice).

The duration of the cooldown timer can be changed with the cooldown: command and is measured in seconds. It defaults to 5 seconds. Setting it to 0 should disable the cooldown timer.

e.g.

```
cooldown: 5
```

Effects

By default, the eye will tend to be a fixed colour, either using the colours inside the GIF or by overriding them with a single colour.
However it can also be made to cycle through different colours. At present this is primarily intended as a demo mode, though it can be enabled just for certain expressions.

The effect: command determines which effect will be used when the eye is set to do this. Currently there are three effects available and they cannot be combined. The last one will take precedence.

These effects are rainbow_v, rainbow_h and rainbow_o, but more may be added in future.
The ‘Rainbow’ effects will stripe the eye with a rainbow that flows through different colours. It can be striped vertically, horizontally or in a circular effect.

The speed of the colour cycling is an update every 10 milliseconds by default, but can be adjusted with the rainbowspeed: command.

e.g.

```
#effect: rainbow_v
#effect: rainbow_h
effect: rainbow_o

rainbowspeed: 8
```

By default, the gradient will be a rainbow. If you want to change the colours, these can be altered by the setrainbow: command. The gradient has 16 colours in it, so you will need 16 setrainbow: commands to full reprogram it.

The first parameter is the slot from 1-16, the second parameter is the colour in hex as per HTML.
e.g.

```
setrainbow: 1 #ff00ff
   :
setrainbow: 16 #0f000f
```

Transmitter

Most people will not need to touch this but it’s included for completeness.

By default the software will boot in Receiver mode unless pin 29 is set to ground via the jumper (see page 4). For testing purposes you may wish to force the Pi into Transmitter mode even if the jumper is not bridged. As mentioned, most people will never need to do this. However, if you do for some reason, the transmitter: command will override the default.

e.g.

```
transmitter: true
```

Serial Port

Most people will not need to touch this but it’s included for completeness.

By default the two Pis will be linked via the serial port on /dev/ttyAMA0. If this should need to be changed to some other serial device, you can use the serial: command to do so.

e.g.

```
# To override serial port
serial: /dev/ttyS0
```

The serial link speed can also be changed via the baud: command, with the parameter in bits/second. By default this is 19200. Make sure both machines use the same baud rate!

e.g.

```
serial: /dev/ttyS0
baud: 9600
```

ACK Light

When an expression change is triggered on one of the GPIO pins it can be programmed to light up an LED on the Receiver board to signal the user that they’ve successfully changed the expression. On Xerian’s hardware I have added the LED between pins 40 and 39 (GND).

The duration of the acknowledgement LED can be changed with the acktime: command, with the duration in milliseconds. It defaults to 750 milliseconds.

e.g.

```
ackpin: 40
acktime: 650
```

Note: It is also possible to disable the ACK light for specific expressions, e.g. blinking, by adding the ‘ack: off’ command to the expression definition. See below.

Random expression chance

The system will display the IDLE expression by default, but you can define random expressions as well, e.g. to simulate blinking.

To prevent the face blinking twice in succession, it is possible to configure a percentage chance that it will look for a random expression to play, a random chance on top of a random chance to space things out more.

This defaults to 75%, but can be configured using the randomchance: command.

e.g.

```
randomchance: 60
```

...will reduce the chance to 60%, resulting in longer durations of the idle face.

Config File Reference – Animation Setup

While the previous commands are all single-line commands which can be run anywhere, setting up an animation is a more complex task and will require a block of related commands.

The simplest working expression can be set up like this:

```
idle: my_idle_animation
    gif: idle.gif
```

This will create an idle animation called ‘my_idle_animation’, which will display the animated GIF file on the second line.

The exact name given doesn’t really matter, but it’s probably best to give it something appropriate, especially if the GIF files are numbered or are otherwise unclear.
The indentation on the gif: line isn’t necessary (unlike YML or Python) but makes it easier to read.

The *order* of the commands is important, however. Doing ‘gif:’ first and then ‘idle:’ will confuse the program and probably display an error message, as ‘gif:’ is relying on ‘idle:’ to have set things up for it.
Likewise, later commands such as ‘drawmode:’ will only work once both ‘idle:’ and ‘gif:’ have been processed. More on that in due course.

Note that the names are unique... you can’t have two expressions called ‘my_idle_animation’.

Declaring a new expression

Currently the software has supports two types of expression – GIF files, and scrolling messages.
It also has four triggers which can cause them to be run – when idle, randomly, triggered by the GPIO pins, and called by a script.

As per the example, we tell it when the expression is going to run first, and then what it will be.

```
idle: my_idle_animation
    gif: idle.gif
```

..but we could also have a random scrolling message, e.g.

```
random: scrolly
    scroll: Hello World
    chance: 50
```

...which will randomly display the message “Hello World” on the display, with a roughly 50% chance.
Or, we could have a rolling-eye animation that is triggered by shorting pin 33 to ground:

```
gpio: eyeroll
       gif: eyeroll.gif
       pin: 33
```

Idle expression

As we have seen in the examples, you can (and must!) set up an idle expression for the program to work correctly. If this is not done, the program will halt with an error message.

With the current version of the software, there can only be one idle expression. If you declare two of them, it will always pick the first one. This may change in future.

The idle expression is the default expression and will always be displayed when nothing else takes precedence. If a GPIO event triggers a new expression, the idle animation will instantly stop.

The idle: command takes one parameter, the expressions's name. You will also need to specify an GIF or scrolling message command in the following line, but otherwise it's pretty self-contained.

```
idle: my_idle_animation
    gif: idle.gif
```

Random expressions

To allow for blinking and other random events, you can set up randomly-triggered expressions. These are not mandatory but will make the face a lot livelier. If a GPIO event happens, the triggered expression will not play until the random animation has ended.

As with idle, you will need to specify a name, and then a GIF or scrolling message on the following line. It is also strongly recommended to set the random chance via the chance: command, otherwise it will default to 0% and the expression will never be displayed.

```
random: blinking
    gif: blink.gif
    chance: 50
```

GPIO expressions

So far the face can display a default expression and blink randomly. However, if you want to trigger an expression manually, the easiest way is to use the GPIO pins on the Transmitter board. By briefly shorting one of these pins to ground via a switch, you can tell the software to run a particular expression such as an eye-roll or angry face. If two are triggered in quick succession, the first one will not be interrupted and the second one will only play after it has finished.

As with idle and random expressions, you will need to specify a name, and then a GIF or scrolling message in the following line. It is also strongly recommended to set the GPIO pin via the pin: command, otherwise it will default to nothing and the expression cannot be triggered.

```
gpio: eyeroll
       gif: eyeroll.gif
       pin: 33
```

The Raspberry Pi has 40 GPIO pins. Of these, only 30-40 are currently supported (others are used by the Pi Hat display). Not all of these can actually be used, since pins 30, 34 and 39 are hardwired to ground and cannot be used for I/O.
Pin 29 is reserved by the software to tell the Transmitter and Receiver boards apart and its use is not recommended.   The parameter for pin: is the actual hardware pin number.

Scripted expressions

Finally, it is possible to declare an expression that will be played under script control.

At the moment the only scripting support is to allow for actions to be run before or after an expression runs, but this does allow you to chain expressions together so they run in sequence. And this is where the scripted expressions become useful.

As with the other expressions, you will need to specify a name, and then a GIF or scrolling message on the following line. Nothing else is required.

As with the Random and GPIO expressions, the animation will not be interrupted by events such as a GPIO signal, and the next animation will not run until the current one has finished.

Here's an example from the demo files. It has a random expression displaying the eye tinted red, after which it calls the green eye expression, which in turn calls the blue eye expression:

```
random: redeye
	gif: idle.gif
	colour: #ff0000
	chance: 20
	after: chain greeneye

scripted: greeneye
	gif: idle.gif
	colour: #00ff00
	after: chain blueeye

scripted: blueeye
	gif: idle.gif
	colour: #0000ff
```

GIF Animations

The most common type of expression will be a .GIF file. This can be animated or a single frame, though since the expression will end when the animation is finished, still pictures are best used for the idle expression or other similar situations. Currently there is no support for looping animations.

The Unicorn Hat HD panel is 16x16, which conveniently means that 256 colours are not a limitation since there are only 256 pixels in total. That said, the SynthEye software was really intended to display cartoon graphics rather than full-colour video anyway.

16x16 GIF files are recommended for the Unicorn Hat HD display. Smaller images will be displayed in the top-left corner – larger images will show just the top-left corner of the image.

Note that since the software is displaying two distinct eyes, the Transmitter board will display the eye in its correct orientation. The Receiver board will mirror the eye on the other display so they will both be looking in the same direction. This mirroring can be disabled, e.g. if the animation contains text.

The bundled animations were all created using GIMP 2.10, treating each frame as a distinct image. I have not done extensive testing with files generated by other packages so I can't guarantee that the image loader is perfect, but so far it has handled everything I need it to.

As mentioned, the gif: command must directly follow one of the expression types (idle:, random:, gpio: or scripted:). It takes a filename as its parameter, and this will be loaded from the directory specified by gifdir: (or /boot if none is specified).

e.g.

```
random: blinking
    gif: blink.gif
    chance: 50
```

If the file cannot be found, or is corrupt in some way and cannot be loaded, the program will halt with an error message.

As mentioned, GIF files can have up to 256 colours including a transparent background. By default it will use the colours baked into the GIF file, e.g. if the eye is blue in the GIF file, it will be drawn as blue.
However it is possible to override this, turning the GIF into a monochrome image and rendering it with the colour of your choosing. Transparent sections of the GIF, and colours which map to pitch black (#000000) will be ignored.

This and other effects are documented in the Drawing Effects section below.

Scrolling Text

While .GIF animations are the main use case for the software, it is also possible to make it display scrolling messages. This was originally intended for error messages but it might be useful to do for other reasons, e.g. secret messages that display randomly or under GPIO control.

There is currently a maximum text length of around 1000 characters, and the alphabet is limited to basic A-Z characters, numbers and a few items of punctuation.

As mentioned, the scroll: command must directly follow one of the expression types (idle:, random:, gpio: or scripted:). The rest of the line will be treated as static text to be displayed in the message.

e.g.

```
random: scrolly
    scroll: Hello World
    chance: 50
```

As of this writing there are no parameters available to change the speed or Y offset of the scrolling text, this may be added in a future release.

Unless overridden, it will use the default eye colour (yellow-orange) for the text.

Drawing Effects

By default, a .GIF animation will be displayed as-is, using the colours from the file, and scrolling text will be displayed in orange.

These can, of course, be overridden, as hinted at earlier.
While .GIF animations are the main use case for the software, it is also possible to make it display scrolling messages. This was originally intended for error messages but it might be useful to do for other reasons, e.g. secret messages that display randomly or under GPIO control.

These commands should follow the 'gif:' or 'scroll:' lines and will not work correctly otherwise.

Draw Mode

The software currently supports the following drawing modes: colour, monochrome, gradient and flashing.

Colour (or color) is the default drawing mode for GIF files – it will use the colours baked into the GIF. It has no effect on scrolling messages.

Monochrome (or mono) will render a GIF file in a single colour, discarding the colour information from the GIF file entirely. By default this will be the eye colour. Monochrome is the default for scrolling text.

Gradient mode will render the GIF or scrolling message as a colour gradient, typically a rainbow effect. The orientation of the gradient is set with the effect: command (See 'General Setup').

Finally, Flashing mode works like the old HTML blink tag. It will flash on and off periodically. The speed of this is tied to the rainbowspeed: parameter (See 'General Setup') and be about 16 times slower. A separate timer may be implemented in future.

Examples:

```
gpio: citadel
       gif: citadel_guy.gif
       pin: 38
       drawmode: colour
       after: chain callback

gpio: eyeroll
       gif: eyeroll.gif
       pin: 33
       drawmode: monochrome

idle: idle1
       gif: idle.gif
       drawmode: gradient

gpio: error
       gif: error.gif
       colour: #ff0000
       drawmode: flash
       pin: 36
```

Colour

If you set the drawmode to ‘monochrome’, the GIF will be drawn in the default eye colour instead of using the colours baked into the GIF file. As mentioned this is also the default for scrolling text.

However, you may wish to override the default eye colour, e.g. to make the eye turn red temporarily if the character is angry.

This can be accomplished by the colour: (alias color:) command. It takes a colour as its parameter, as usual in the 6-digit HTML format.

If the drawmode was previously set to ‘colour’ (i.e. use the built-in colours), it will changed to ‘monochome’ on the assumption that you probably want to actually see the colour you’ve specified.

Example:

```
random: redeye
       gif: idle.gif
       colour: #ff0000
       chance: 20
       after: chain greeneye

scripted: greeneye
       gif: idle.gif
       colour: #00ff00
       after: chain blueeye

scripted: blueeye
       gif: idle.gif
       colour: #0000ff
```

Events

It is possible to make an action happen immediately before or after an expression is displayed.

The design goal was to allow GPIO ports to be programmed, so that an Arduino device driving lights elsewhere on the costume could be instructed to change the colour for the duration of the animation and back again afterwards.
This was done in an extensible manner so that other actions could be performed as well. At present, the only other action available is to run another expression.
(Note that it is possible to set up an infinite loop by doing this, so be careful)

Events are set up by using the ‘before:’ and ‘after:’ commands. The first parameter will be the command to run, e.g. ‘set_gpio’, ‘clear_gpio’ or ‘chain’ to chain expressions together.
With ‘set_gpio’ and ‘clear_gpio’ the last parameter will be the GPIO pin to affect.
With ‘chain’ the final parameter will be the name of the expression to be played – if it is not found, nothing will happen.

Note that the Arduino uses pull-up resistors, so if nothing is connected, the GPIO port defaults to logic *high* (3-5V). Because of this, we need to force the logic to be low (0V) to signal it. For this reason, ‘set_gpio’ is actually setting the pin to 0v instead of 3V. Likewise, ‘clear_gpio’ will set the pin to +3V, signalling logic high.

It is also important to note that we may wish to affect the GPIO pins on either the Transmitter board or the Receiver board. Hence, it is possible to select the device by prefixing the GPIO pin with R or T. If just the port number is specified, it will assume we mean *both* devices.

e.g.

```
       before: set_gpio R35     # Trigger GPIO on pin 35 of Receiver

       before: set_gpio T35     # Trigger GPIO on pin 35 of Transmitter

       before: set_gpio 35      # Trigger GPIO on pin 35 on both devices
```

It is strongly recommended to specify ‘R’ or ‘T’ for event GPIO, since if the pin is already being used for input, trying to write to it may damage the hardware. The software will attempt to detect this and display an error during startup.

I have included example Arduino software that will flash neopixel status lights and set them red in response to a GPIO event.

Examples:

```
gpio: error
       gif: error.gif
       pin: 36
       colour: #ff0000
       drawmode: flash
       before: set_gpio R35     # Set horns red (on Xerian's hardware)
       after: clear_gpio R35    # Set horns back to normal (Xerian's hardware)

random: redeye
       gif: idle.gif
       colour: #ff0000
       chance: 20
       after: chain greeneye

scripted: greeneye
       gif: idle.gif
       colour: #00ff00
       after: chain blueeye

scripted: blueeye
       gif: idle.gif
       colour: #0000ff
```

Mirror

As mentioned previously, the software will display expressions as-is on the Transmitter device, and mirror them on the Receiver to avoid giving you a boss-eyed costume.

If the expression contains text or something else which you do not want to be mirrored, you can disable this for a specific expression.

This is done with the command ‘mirror: false’ (or ‘mirror: off’).

Scrolling messages are not mirrored – in future I may add ‘mirror: true’ to force them to be mirrored anyway for completeness.

e.g.

```
random: blinking
    gif: blink.gif
    chance: 50
    mirror: false
```

ACK disable

As mentioned in ‘General Setup’ it is possible to flash an LED on the Receiver when an expression change is triggered to acknowledge to the wearer that they’ve successfully changed expression.

By default, this will also light up for expressions such as blinking. Since this may be a problem, you can disable it for certain expressions.

This is done with the command ‘ack: false’ (or ‘ack: off’).

e.g.

```
random: blinking
    gif: blink.gif
    chance: 50
    ack: false
```

Background

You may wish to overlay one expression on top of another. The example that prompted this requirement was blushing – since you may wish to customise the colour of the eye, using the internal colours from the GIF is a problem. Hence, it is better to have the blushing cheek effect on its own, and draw this on top of the default eye.

The ‘background:’ command is used to do this. It refers to another expression to take the image from, typically the Idle expression. (If you want to use a custom background that is never displayed elsewhere, set up a Scripted expression)

It is intended that the background expression should be a GIF - only the first frame will be displayed as a static backdrop. If you specify a scrolling message instead, nothing will happen.

Example:

```
gpio: blush
       gif: blush.gif
       pin: 37
       drawmode: colour                # Use GIF's own colours
       background: blush_background    # Draw blush effect on top of this

scripted: blush_background
       gif: blush_eye.gif
       drawmode: monochrome
```

...in this example, the main expression is the cheeks, and the eye is declared as the background in a scripted expression. This is because we need a custom image to raise the eye slightly and make room for the cheeks.

If the eye is not flush with the bottom of the screen, we could just use the Idle expression, e.g.

```
gpio: blush
       gif: blush.gif
       pin: 37
       drawmode: colour      # Use GIF's own colours
       background: idle      # Draw blush effect on top of this
```

Interruptible

With expressions such as the Idle display, we want them to immediately stop when something higher priority comes in, e.g. an expression change via GPIO.

By default, all other animations such as blinking, or the GPIO-triggered expressions themselves will play through completely before yielding, to avoid a sudden jump.

If you want to make an expression (or scrolling message) able to be interrupted, use the 'interruptible:' command.

This takes 'true' (or 'on'), or 'false' (or 'off') as its parameter. It can be used to make any animation or scroller interrupted, or alternatively to stop the Idle animation from accepting interruptions.

Note that making Idle uninterruptible is not recommended as it will introduce latency when switching expressions, but the option is there just in case.

Example:

```
random: blinking
    gif: blink.gif
    chance: 50
    interruptible: true
```

Source code

The source code and later versions of this document are available at:
`https://github.com/jpmorris33/syntheyes3`