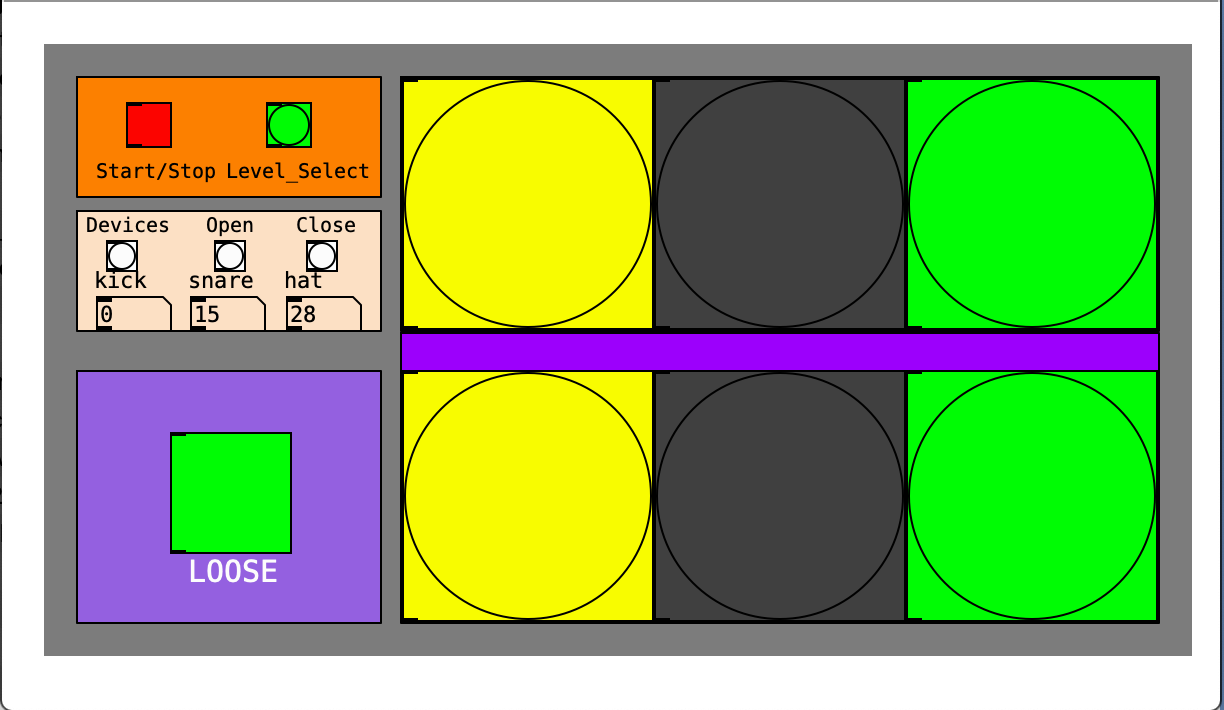
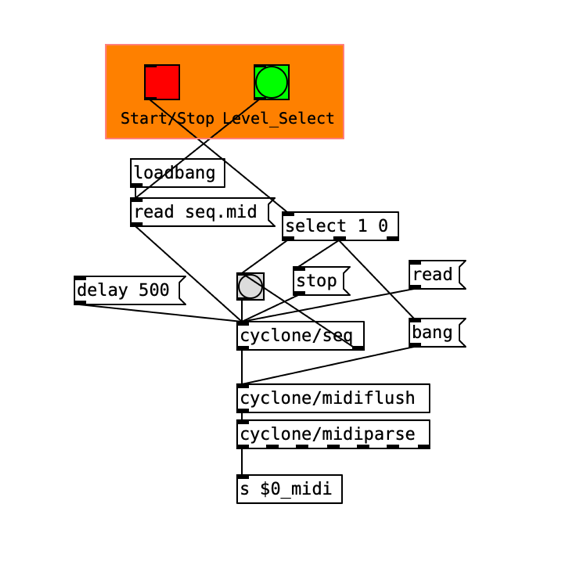
# How the Patch works

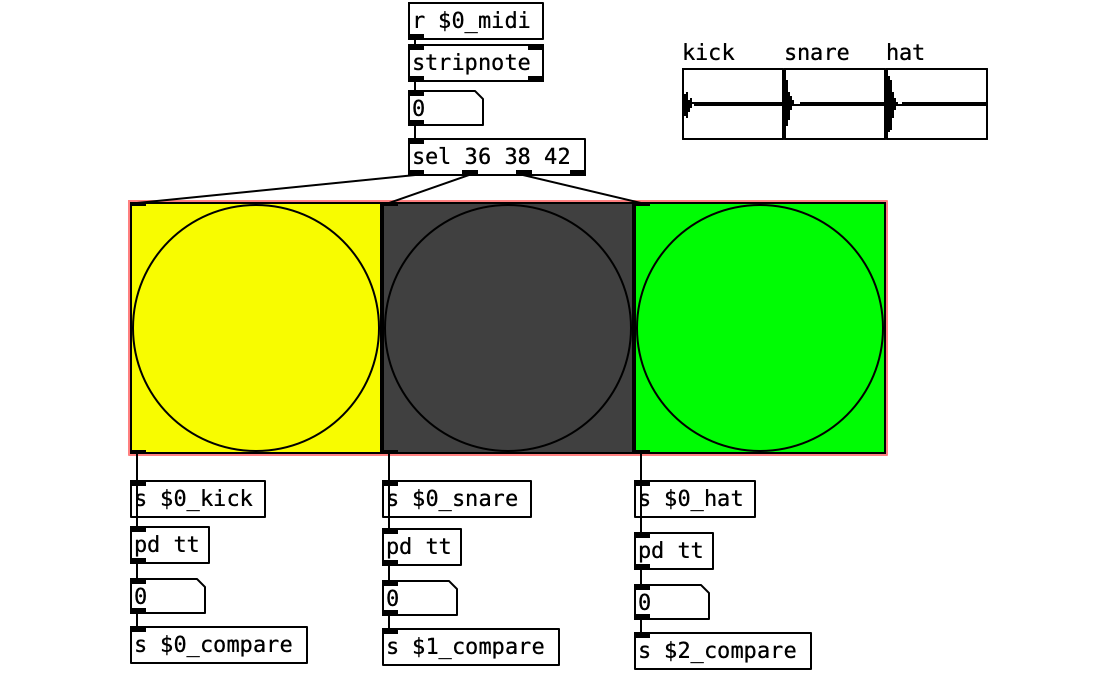
The main graphic user interface is designed with the particular user in mind which is ages 4-8. Its visually pleasing for this purpose and must conveys as much information visually. The colours of the buttons correlate with the physical buttons. The user can see what’s being played by themselves and the computer. As the game is focused around timing and mimicking the drum sequences, this is important for the user to interpret on a visual level and in audio. The yellow box represents the Kick, the black box represents the snare and the green representing the hi-hat. The player must maintain rhythm and timing as close in sync for as long as possible with the computer. Failing to do so for an extended period of time will indicate the player has lost an the sequence will stop. There are multiple sequences the player can slect from wich increase in complexity from level 1 – 3.



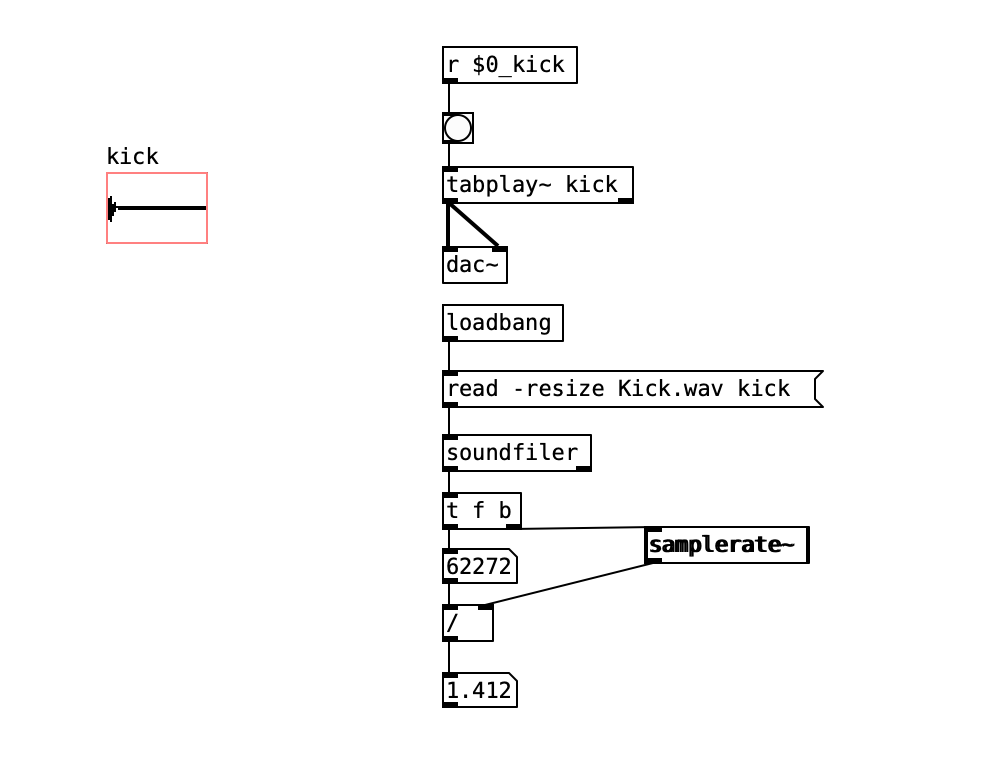
From the ‘st/stop – level select’ section of the patch, the user can select the midi file with the level of complexity. There are 9 midi files divided into 3 levels each level with 3 different sequenced combinations of the kick, snare and hat. The user can also start and stop the sequence at their own pleasing. When the midi files are loaded, they are read via the *cyclone/seq* object.



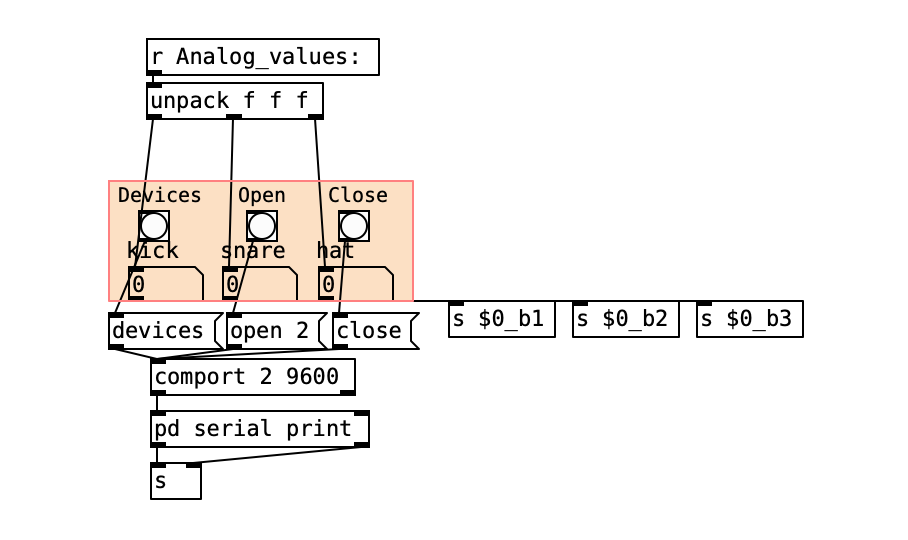
The midi drum sequences are read as midi notes C1 (kick), D1 (snare), & F#1 (hi-hat). They are then outputted as numbers 36, 38 & 42. The strip note function removed the note off part of the midi note information. Without this the sounds would be triggered twice when the midi note on/off information is received. The ‘*bang’* object is used to convey the visual information for the user. They are at a good size and coloured to match the physical buttons. As the sequence is played the ‘*bang’* object flashes according to the activity of kick snare and hat. The player can see this as well as hear the drums sequence and therefor can interpret the timing that is required. The output from each ‘*bang’* object is sent to the *‘pd tap tempo’* object, which then calculates the difference in timing between each note in milliseconds



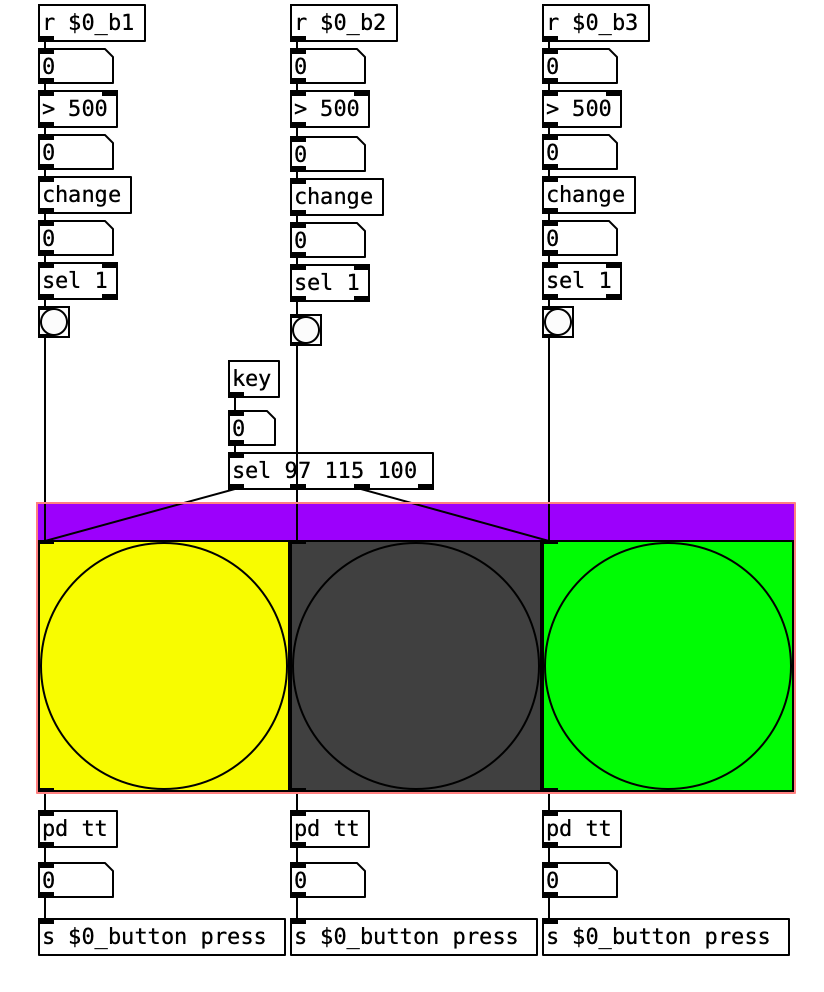
The audio samples are triggered when they receive a bang from their corresponding colour. Using the *‘array’* object, a sample can be loaded into it using the *‘soundfiler’* object. The samples are read from the root folder from which the PD project is saved. When the project is opened, they are loaded by default. After the samples are loaded, they are played back using the *‘tablay~’* object.



The output of the Arduino from the analogue inputs is read via the ‘*pd serial print’* abstraction. This abstraction will have to be copied into the project so it remains there, otherwise the abstraction wont work. The *‘comport’* object is able to open and close the connection between the Arduino board and PD, it can also select the port in which you are using. This will be whichever USB socket is being used at the time. The signals received from the user’s manual trigger via the physical push buttons is then sent remotely to the manual trigger visual section.



When the user presses the physical button, that information triggers the corresponding bangs, it can only be seen by the user, there is no audio. The frequency of the button presses is calculated again with the ‘*pd tap tempo’* abstraction. They can also be triggered with the letters A,S,D on the qwerty keyboard with the ‘*key’* object.



The timing information of the user’s manual trigger, is compared to the midi sequence of the computer with the *‘expression’* object. Using the *‘greater than’* object, the *‘metro 1000’* can be start or stopped to count down for 20 secs and then display a ‘X’ in the loose box. The sequence will stop when the ‘X’ is displayed. At this point the user can re-attempt, or select another sequence more or less complicated. When the sequence is started, the lose box is reset removing the ‘X’. The user can reverse the 20 second countdown timer if their timing becomes within range again, rendering the countdown timer to stop.

