Tiny Shaders

Design Document

Overview

Introduction

ELEVATOR PITCH

Tiny Shaders is a coding puzzle game based around the challenges of shader development

DETAILS

USP: A puzzle game that evokes node-based shader development

Target Market: Puzzle gamers and programmers

Target PEGI Rating: 3

Platform: Unity 6

HIGH CONCEPT

Tiny Shaders is a 2D singleplayer puzzle game, where the player must solve puzzles by dragging blocks onto a board and connecting them with wires to build an algorithm.

The real fun comes once players have built a solution, as the game will prompt them to improve and optimise their solution by providing each solution with scores for their size, cost, and cycles.

As players progress through the game, they will unlock more puzzles!

Design Pillars

SIMPLE OPERATIONS

The constituent components of each player's solution should be relatively easy to comprehend, as should the requirements of each puzzle. This does not apply to the player's solution as a whole, however. The player should be tasked with making complexity arise from simplicity.

ROOM FOR IMPROVEMENT

The player should always be given the ability and motivation to improve upon their work, and their first solution should never be their last. By scoring each solution across multiple metrics and designing puzzles that give space for clever optimisations, we can ensure the player is given plenty room for improvement.

REMOVING TOOLS

Seemingly simple challenges will be made complex by removing familiar tools from the player. The player cannot assign variables, setting up loops is very difficult. Several simple operations are not given to the player. Removing these tools forces the player to think outside of the box and create novel solutions.

Milestones

PREPRODUCTION (W1-W2)

Research, pitch, and planning

GDD established

SPRINT 1 (W3-W4)

Circuit builder and interaction system developed

SPRINT 2 (W5-W7)

Code interpreter developed

Puzzle inputs and verification system developed

SPRINT 3 (W8-W9)

Menu navigation and 3 puzzles implemented

Playtesting

SPRINT 4 (W10-W11)

Art and sound pass

SPRINT 5 (W12-W14)

Saving and loading puzzle solutions

Tutorial puzzles

Polish

POST PRODUCTION (W15-W16)

Final Reflection

STRETCH GOALS

Additional puzzles

Further polish

Basics

Puzzles

COMPLETING PUZZLES

Tiny Shaders is composed of a series of open-ended puzzles that the player must solve. Each puzzle will have a unique set of inputs and outputs. The goal is to construct an algorithm that supplies he desired output by placing nodes and connecting them with wires.

The puzzle inputs and outputs are on a 16 by 16 grid of pixels, giving 256 input and output sets. The player's algorithm is run once for each pixel, and reset between computations. The player has completed the puzzle once their algorithm has run through each of the 256 pixels and supplied the desired output each time.

The player can see the desired outputs on the side of their screen.

INPUTS AND OUTPUTS

For each puzzle, the player is given a set of inputs and outputs. Each input and output is a single numeric value on a specific input or output channel. A player's algorithm can be made to read from the input channel at any time, either by using an input node or a sample node, and selecting the desired channel to read from. A player's algorithm can supply data to an output channel by providing the data to an output pin, and selecting the desired channel to write to. Once a value is written to a pixel in an output channel, it cannot be rewritten. Once all output channels are written to, the game will reset the player's algorithm and begin computing the next pixel. A player's algorithm can read the X and Y co-ordinates of the current pixel by using a co-ordinates node, and selecting either the X or Y axis.

COMPUTATION CYCLES

Computation of a player's algorithm is split into multiple cycles, where all nodes execute simultaneously. Each cycle is split into 2 phases: solve, and move.

In a solve phase, any nodes that are prepared to execute will do so, transforming their input values into output values. For example, multiplying the two inputs and supplying the result to the output.

In a move phase, any output pins that are blocked with data will supply that data to any connected, unblocked input pins.

Nodes

USING NODES

A node is the basic building block component of puzzle solutions. Each node occupies space on the board, and has a set of input and output pins, though not all nodes have both input and output pins (as can be seen with the discard node on the left).

MANIPULATING NODES

A new node can be placed by selecting the desired node from a panel on the side of the player's screen, and dragging it into an unoccupied position on the grid. Nodes can be rotated by pressing R while dragging them, and deleted by dragging them back to the panel.

There is no limit to the number of nodes allowed, except for the size of the board that they can be placed on.

NODES IN ACTION

Each node has a functionality, such as adding or multiplying their inputs. This function is almost always performed only when all input pins are blocked, and all output pins are unblocked, though there are exceptions.

Pins

USING PINS

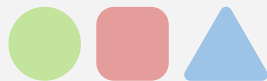
Pins exist on blocks, and come in two types: input and output. Input pins seek to receive data from connected output pins, and output pins seek to supply data to connected input pins.

A pin is considered “blocked” if it holds data during a computation cycle.

It can be thought of that input pins seek to become blocked by taking data from output pins, and output pins seek to unblock themselves by supplying data to input pins.

PIN TYPES

Unfilled pins mark input pins.



Filled pins mark output pins.

Circles, squares, and triangles are used in this document to refer to pins due to the previous existence of multiple data types. This is no longer the case, and all data consists of a single value from –999 to +999. These icons are now used interchangeably throughout the document, and there is no semantic difference between them.

#

USING WIRES

Wires are used to connect input pins to output pins. These connections can be of any length, and can connect any number of input pins with output pins.

Wires are placed by holding shift and left click, and dragging the mouse over area to paint.

Wires are removed in the same way by holding shift and right click.

During the move phase of a computation cycle, blocked output pins will seek to supply their data to connected, unblocked input pins

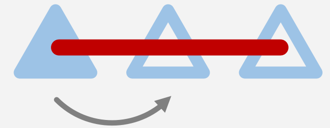
MULTIPLE WIRE CONNECTIONS

When wires connect multiple input pins and output pins, they must decide which input pin will receive the data from which output pin.

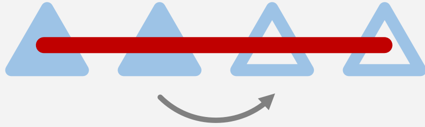
This is done by each unblocked input pin selecting a favoured blocked output pin, and each output pin selecting a favoured blocked input pin. If both pins favour each other, then the data is supplied from the output pin to the input pin.

Favourites are chosen by which block is closer along the path of the wire (not closest on the grid). When multiple pins are the same distance, output pins favour, in order: pins above, to the right, below, and then finally to the left. Input pins favour the reverse: first pins below, to the left, above, and then finally to the right.

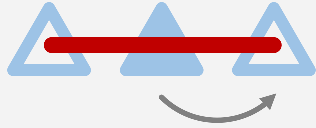
Here are some example cases:



Data will be transferred to the leftmost input pin, as it is closest.



Data will only be transferred between the middle two pins, as both pins on the end favour the other middle pin.



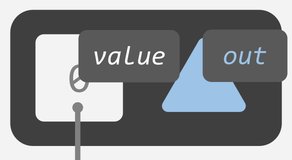
Data will be transferred to the rightmost input pin, as output pins favour supplying to the right over supplying to the left when distance is equal.

Nodes

Basic Nodes

ZX0000: INPUT

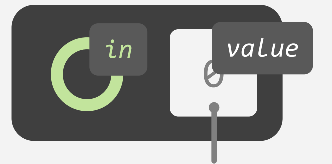
The input node contains 1 output pin, which becomes blocked at the start of the first cycle with the data of the supplied channel, after which, the node is rendered inert.



Scroll to cycle through input channels

IZ0001: OUTPUT

The output node contains 1 input pin. Upon receiving data, the node will push the data to the supplied output channel, but the pin will remain blocked.



Scroll to cycle through output channels

XZ0002: DISCARD

The input node contains 1 input pin. Each computation cycle, the node will discard the data in the input pin, and unblock the node.

Basic Nodes

XX0004: CONNECTOR

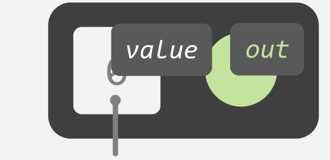
The connector node contains an input pin and an output pin. Each computation cycle, the node will transfer the data from the input pin to the output pin. The gap in the middle allows wires to cross over the node without connecting to the node.

XX0005: DUPLICATOR

The duplicator node contains an input pin and two output pins. Each computation cycle, the node will transfer the data from the input pins to the two output pins.

ZI0006: GENERATOR

The generator node contains 1 output pin, which will continuously output integers each cycle with values equal to the supplied number.

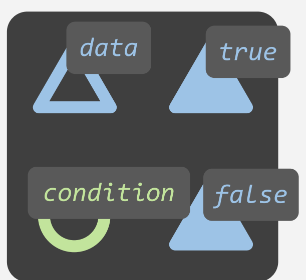


Scroll to cycle through values

Control Nodes

NX0100: IF

The if node has 2 input pins: data (wild), and the condition (int). It also has 2 output pins, with TRUE at the top, and FALSE at the bottom. If the condition input is not 0, the data will be passed to the TRUE output pin. Otherwise, it will be passed to the FALSE output pin.



######: FOR

The for node takes an integer input and output. Each cycle, the input will remain blocked, and the data will decrease by 1, outputting the current data on the output pin. If the output pin is blocked, the data will not decrease this cycle. Once the data reaches 0, the node will output 0, and unblock the input node.

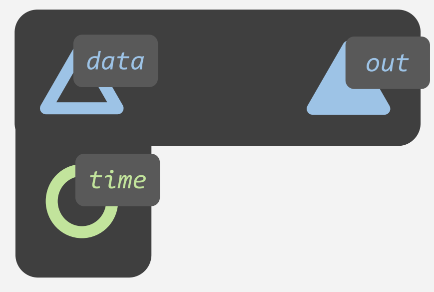
This can already be accomplished by setting up a loop. This node's uses are niche and potentially removes a fun puzzle for the player.

XX0101: REPEATER

The repeater node has an input and output pin. When data is received on the input, it will become unblocked, and the output pin will continuously output the data received each cycle. This cannot be stopped, but providing new data on the input pin will cause the node to output the new value.

XX0102: DELAY

The delay node has 2 input and 1 output pins. After being supplied with data, the input pin will become blocked. After the supplied number of cycles have elapsed, the data will be output on the output node.



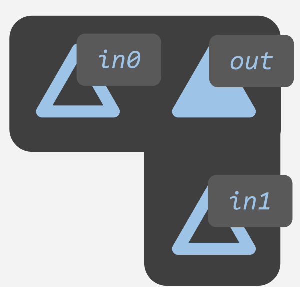
Logic Nodes

NX0200: ADDITION

Adds two integer values together. If supplied with a texture, adds the integer value to every cell in the texture.

XX0201: MULTIPLICATION

Multiplies two values together. If supplied with a texture and an integer, multiply every cell in the texture with the integer. If supplied with two textures, multiplies each cell in the first texture with the corresponding cell in the second texture. The resulting texture output is the minimum size of the two.



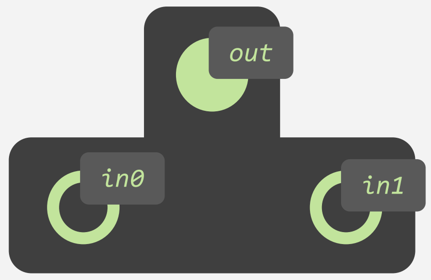
######: MODULO

Outputs the remainder of dividing the top integer by the bottom integer.

This can already be accomplished by setting up a loop. This node's uses are niche and potentially removes a fun puzzle for the player.

II0202: MAX

Returns the maximum integer of the two supplied.



Logic Nodes

II0203: BIT SHIFT

Shifts the binary representation of the first integer to the left, a number of times equal to the second integer. Supplying a negative value in the second position will shift the input to the right. This operation does not affect the sign of the integer, unless the integer wraps.

II0204: NOT

If the supplied value is zero, output a 1. Else, output 0.

######: COMPARATOR

If the supplied integers are equal, output a 1. Else, output a 0.

This can already be accomplished by multiplying one value by -1 and adding them together, then passing the value through a not node. Having the player figure this out is a fun challenge.

Texture Nodes

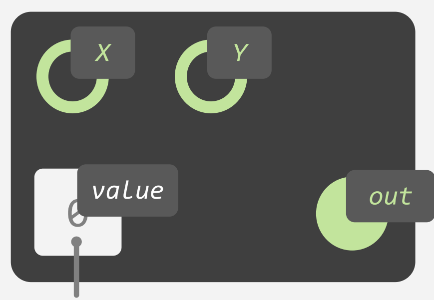
IT0000: COORDINATES

The coordinates node contains 1 output pin, which will continuously output the coordinates of the current pixel on the selected axis

Scroll to select X or Y axis

NI0301: SAMPLE

When supplied with X and Y values, samples the selected input channel at the supplied coordinates (mod 16), and outputs the result.



Scroll to cycle through input channels

Puzzles

Subtraction

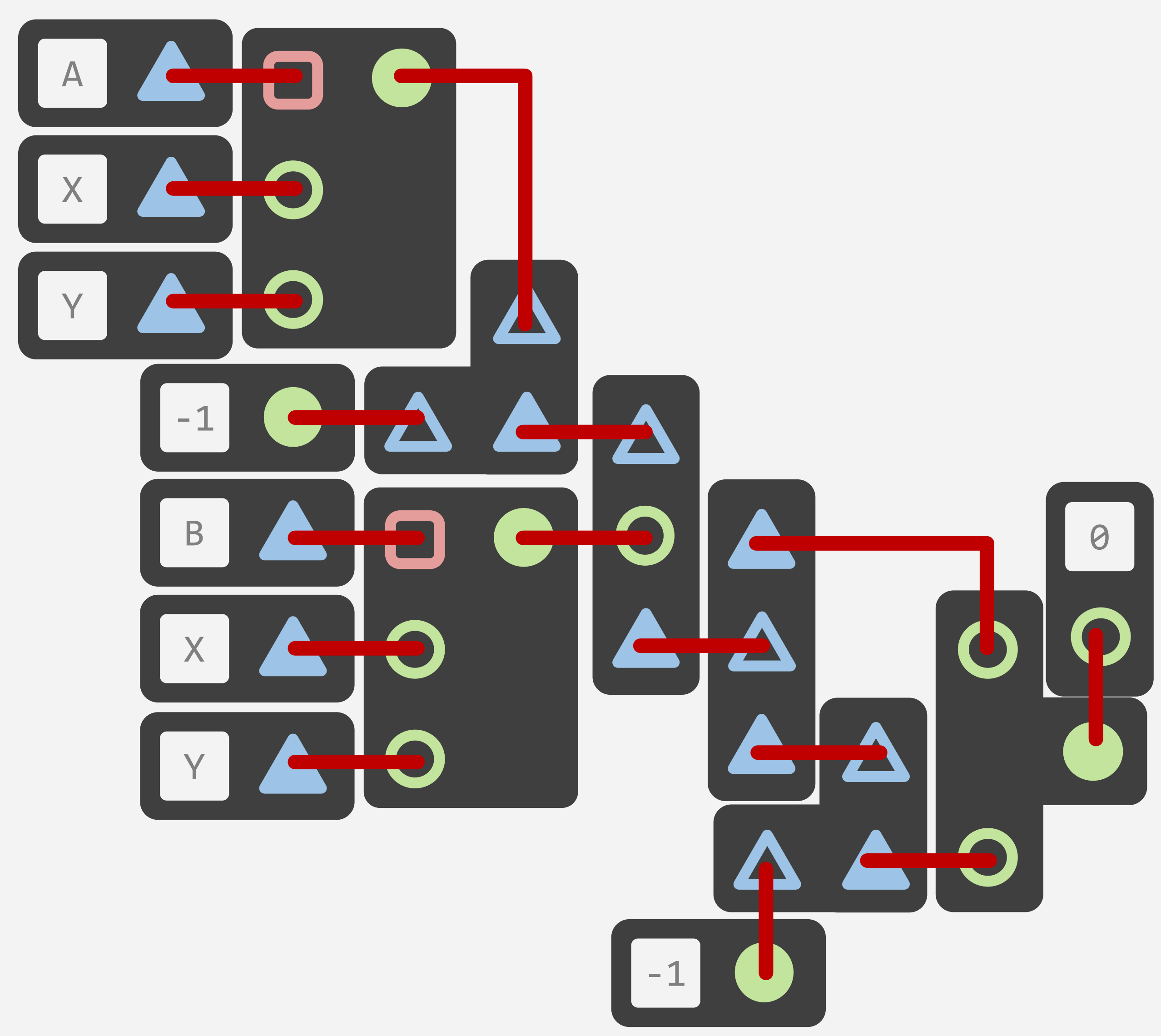
PROBLEM

Players are given two textures, and for each pixel in the texture, must subtract the lower-value texture from the higher value texture, outputting the result.

- The absence of subtraction, min, and abs nodes makes this slightly more puzzling than it may initially seem.
- Players must realise they can use multiplication by –1 to subtract one value from another, and that they can use a max node alongside multiplication by –1 to take an absolute value.

This puzzle is simple, and should serve as a good introduction to the game's mechanics.

EXAMPLE SOLOUTION



Division

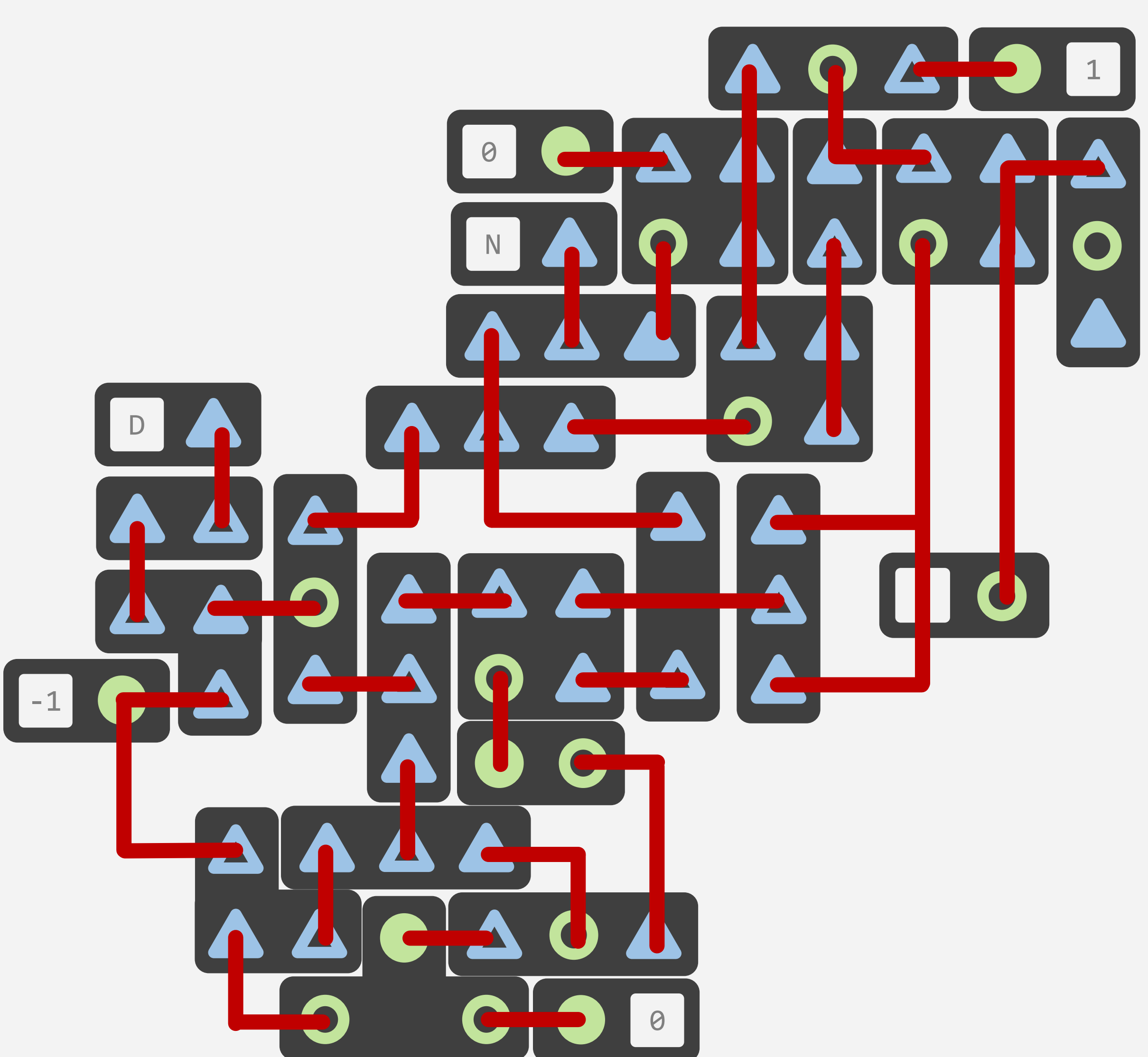
PROBLEM

Division is an essential operation, that with the removal of variables, becomes very difficult to perform. Players must develop an algorithm that:

- Loops over the numerator, repeatedly subtracting the denominator until left with 0 or less
- Counts how many loops occurred
- Outputs the count once the loop has finished

Each of these steps are non-trivial, and requires the player to utilise the provided nodes in unique ways.

EXAMPLE SOLOUTION



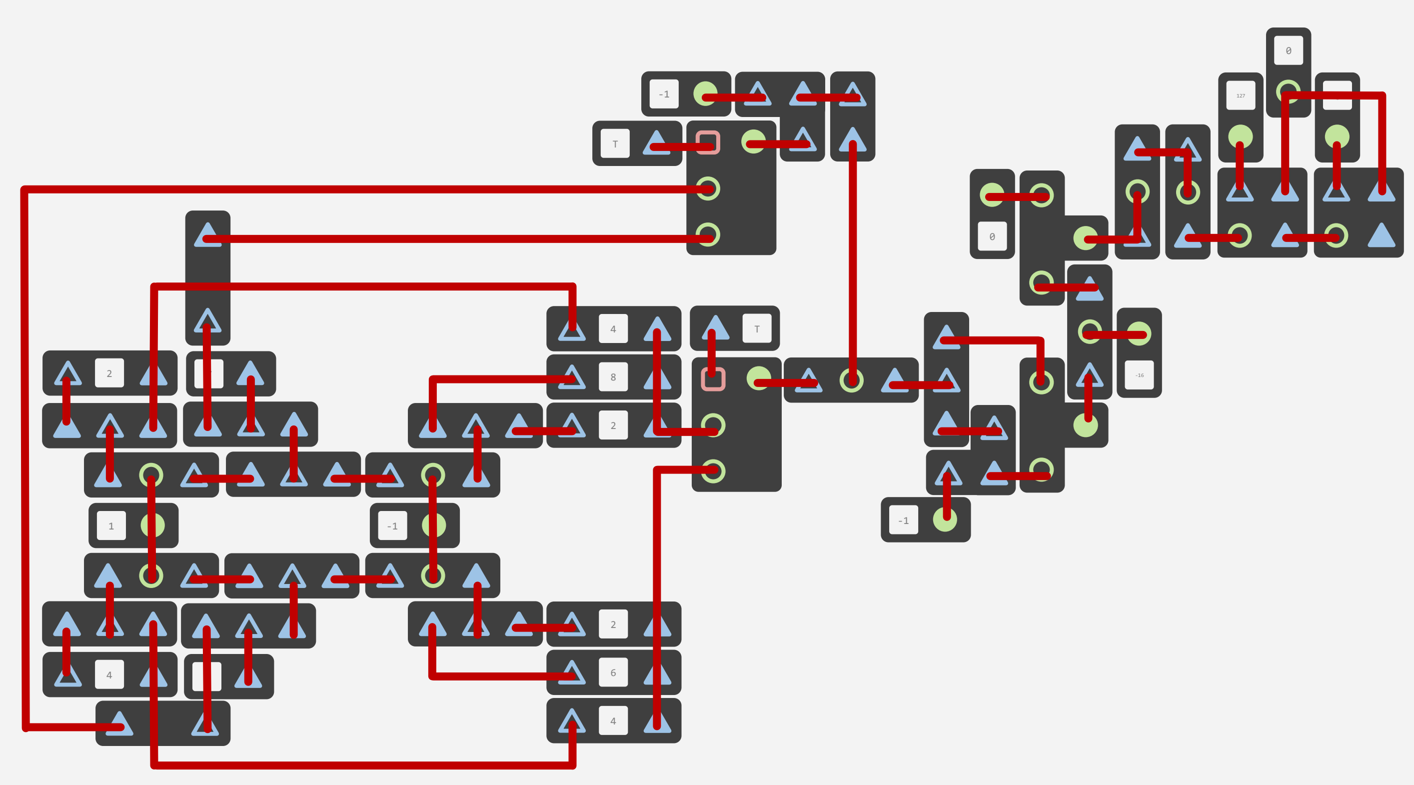
Edge Detection

PROBLEM

The player must build an algorithm that outputs 127 when a pixel in the texture is more than 16 values away from any of it’s 4 orthogonal neighbour pixels, and output 0 when it is not. The player must:

- Sample the four neighbour 4 values, either by setting up a loop or placing 4 different samplers.
- For each value, compare it to the current value and create a truthy signal if the difference is greater than 16.
- Output 127 when any of the 4 checks returns true.

EXAMPLE SOLOUTION



END OF DOCUMENT