ICFP Contest 2022

Specification V0

Alperen Keles

Introduction

The mighty wizards of Lambda land has seen all your poses from last year and they were absolutely fascinated by them. They were so inspired by you that they have been discovering the secret arts of painting for the rest of the year, waiting for you to join them. Your mission, if you choose to accept it; will be to develop algorithms for robo-painters of the future. After all, there are so many paintings to make, and so little of us functional programmers to make them. The winner will receive the honor medal of Leondardo Da Vinci, the RoboVinci Medal.

Timeline

Note that there will be updates to this specification, and more problems will be released during the contest. This will happen at these specific times:

- 4 hours into the contest (new problems only, no changes to specification)
- 8 hours into the contest (new problems only, no changes to specification)
- 12 hours into the contest (new problems, small changes to the specification)
- 24 hours into the contest (after the lightning division ends)
- 36 hours into the contest
- 48 hours into the contest

Changelog

Any changes will be published here.

Problem Specification

The task is to **paint** a given **canvas** with the least **cost** and highest **similarity**.

As part of the task, you will be given;

- a set of **moves** applicable over the **canvas**,
- an **instruction language** to express your set of **moves**,
- a **cost function** for calculating the **cost** of each **move**,
- a **similarity function** for calculating the **similarity** of your **painted canvas** to the **target painting**.

As part of individual problems, you will be given;

- an initial **canvas**,
- a target **painting**.

Canvas

Canvas is an abstract 2-dimensional pixel space of **RGBA** channels.

Each **move** transforms the canvas in a different sense.

After all moves are applied to a **canvas**, it can be rendered to a **painting**.

Canvas is made out of **blocks**.

Blocks

A block is either a frame that consists of a set of sub-blocks; or a simple structure with shape and color.

In the eyes of the functional programmer, one might imagine such a definition.

```
data SimpleBlock = SimpleBlock Shape Color
data ComplexBlock = ComplexBlock Shape ChildBlocks
type ChildBlocks = Set<SimpleBlocks>
data Block = Either SimpleBlock ComplexBlock
```

The **initial canvas** only holds exactly one block, colored with the color **RGBA(0, 0, 0, 0)**.

Blocks are uniquely defined by their **block_id**.

A global counter is held for block creations via merge, the initial canvas has one block with **block_id = 0**. Each new block created by a **merge move** increases this counter by 1. Each **cut move** generates sub-blocks of the block by simply adding **.0, .1, .2 or .3** to the end of the block id depending on the cut type.

Painting

Painting is a concrete 2-dimensional pixel space of **RGBA** channels. For individual problems, you will be provided with **PNG** files for the paintings.

Moves

We present you with **5** different moves you can use to paint your canvases.

Cut Moves

Cut moves take a block, and some cut instruction over that block. Create new sub-blocks; preserving the colors.

Line Cut Move A line cut move takes a block(defined by its block-id), an orientation(X or x for Vertical, Y or y for Horizontal), an offset(the 1-d coordinate for the cut operation) and creates 2 sub-blocks of the given block.

Sub-blocks of line cuts are numbered from *bottom to top, or left to right.*

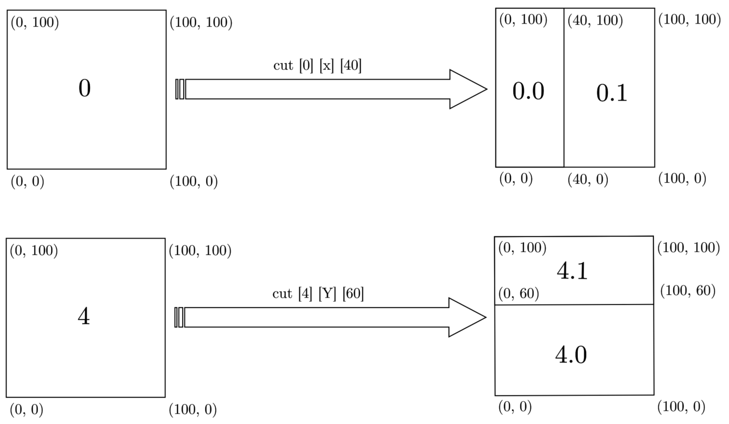

Figure 1: Line Cut Move Image

Point Cut Move A point cut move takes a block(defined by its block-id), an offset(the 2-d coordinate for the cut operation) and creates 4 sub-blocks of the given block.

Sub-blocks of point cuts are numbered from *bottom left using reverse clock-wise numbering.*

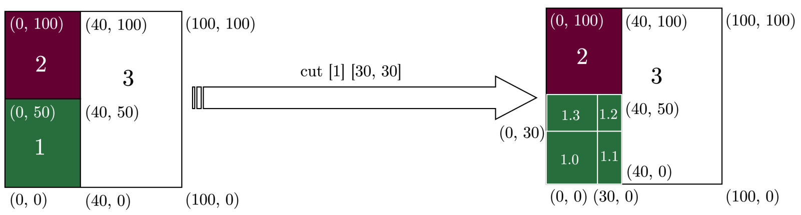

Figure 2: Point Cut Move Image

Color Move

Color move takes a block, and some color on **RGBA** space. It changes the color content of the given block to the given color.

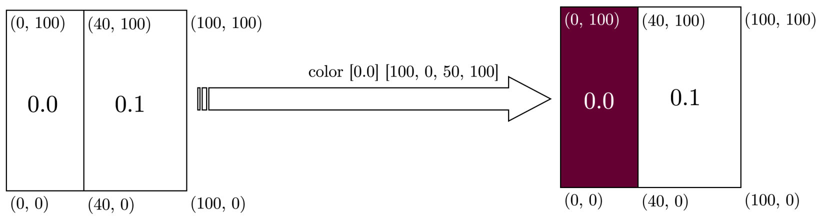

Figure 3: Color Move Image

Swap Move

Swap move takes two blocks. It swaps the contents of the given blocks.

Blocks must have the **same shape** to be swapped.

Merge Move

Merge move takes two blocks. It merges the blocks by creating a new block, adding these blocks to this new block as sub-blocks.

Blocks must be **compatible** to be merged. They must be adjoint, and their adjoint sides must have the same length. Informally; the merge of the blocks must create a new rectangle.

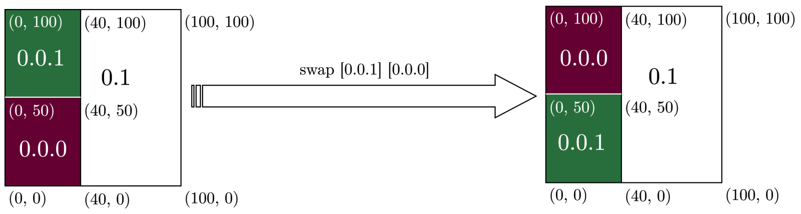

Figure 4: Swap Move Image

Each time a merge operation is performed, a new block is created. The newly created block has the **block id** according to the global counter. For example, when global counter is *n*, the block generated by this merge operation will have the block id *n+1*.

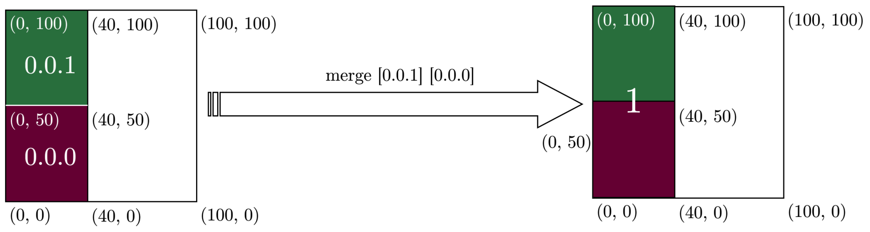

Figure 5: Merge Move Image

Instruction Language

Your task is to apply a set of moves to a canvas to similarize it to a given target painting. The way you will provide these set of moves is via submitting an **ISL(Instruction Set Language)** file for each problem. **ISL code** directly corresponds to the set of moves given above.

A BNF(Backus-Naur Form) form of the ISL grammar is given at the end of this specification.

Cost Function

Each move has a `base + dynamic` cost.

Below is the table for **base cost for each move**

Move Type	Base Cost
Line Cut	7
Point Cut	10
Color	5
Swap	3
Merge	1

The function for cost is;

```
cost(move, block, canvas) = round(base_cost(move) x size(canvas)/size(block))
```

For each submission, these score are calculated for each move and aggregated for the total cost calculation.

Similarity Function

After processing all moves of a submission, the system calculates the similarity of the result to the target painting.

This is done via calculating **pixel difference on RGBA Color Space** for each pixel and aggregating those results.

Pixel difference is calculated via **Euclidian Distance of RGBA Values of Pixels**. Calculation is given below;

$$pixeDistance = \sqrt{dist_R \times dist_R + dist_G \times dist_G + dist_B \times dist_B + dist_A \times dist_A}$$
$$imageDistance = sum(pixelDistance, \forall\ pixels) \times 0.05$$

We wil provide source code for these computations in order to avoid any confusion.

Scoring

Score for each individual **problem** is calculated by adding the cost of the `ISL code` and the result of the `Similarity Function`. . A higher cost will result in a lower position on the scoreboard.

For the contest scoreboard, we will first classify participants by the number of problems they submitted solutions to. For each class of participants, we will sum their costs, sort them accordingly. We will then sort participant classes by the number of submitted solutions.

As a more concrete example, for the contestant list given below:

P1: 2 problems, total cost 90
P2: 3 problems, total cost 100
P3: 2 problems, total cost 80
P4: 1 problems, total cost 50

Scoreboard would have the participants sorted as `P2, P3, P1, P4`.

Submission

You will submit **ISL code** over panel inside the portal. We will also provide a REST API for the submissions.

Deadlines

As traditional, the contest will have a Lightning Division spanning the first 24 hours. To qualify for the Lightning Division prize, submit your ISL codes by September 3, 2021, 12:00pm (noon) UTC.

To qualify for the Full Division prize, submit your ISL codes by September 5, 2021, 12:00pm (noon) UTC.

In order to qualify for any prizes, your source code must be submitted by the end of the contest as well. You can do this through the web portal.

Determining the Winner

We will use the same procedure to determine the winner in both the lightning and full divisions, ranking the teams by cumulative score, computed as the sum of scores for each task.

ISL Specification

ISL code is a set of *moves* over a canvas.

We start with a description of ISL grammar.

ISL Grammar

```
<program>                ::=     <program-line> | <program-line> <newline> <program>
<program-line>           ::=     <newline> | <comment> | <move>
<comment>                ::=     "#" <unicode-string>
<move>                   ::=     <pcut-move> | <lcut-move> | <color-move>
                                 | <swap-move> | <merge-move>
<pcut-move>              ::=     "cut" <block> <point>
<lcut-move>              ::=     "cut" <block> <orientation> <line-number>
<color-move>             ::=     "color" <block> <color>
<swap-move>              ::=     "swap" <block> <block>
<merge-move>             ::=     "merge" <block> <block>
<orientation>            ::=     "[" <orientation-type> "]"
<orientation-type>       ::=     <vertical> | <horizontal>
<vertical>               ::=     "X" | "x"
<horizontal>             ::=     "Y" | "y"
<line-number>            ::=     "[" <number> "]"
<block>                  ::=     "[" <block-id> "]"
<point>                  ::=     "[" <x> "," <y> "]"
<color>                  ::=     "[" <r> "," <g> "," <b> "," <a> "]"
<block-id>               ::=     <id> | <id> "." <block-id>
<x> | <y>                ::=     "0", "1", "2"...
<id> | <number>          ::=     "0", "1", "2"...
<r> | <g> | <b> | <a>    ::=     "0", "1", "2"..."255"
<newline>                ::=     "\n"
```