ICFP Contest 2022

Specification V2

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 Leonardo Da Vinci, the RoboVinci Medal.

Full Division v.1

As we go through the contest, we must adapt to changes. RoboPainters will no longer work on empty canvases, but rather they will be given canvases already painted.

As part of the full division problems, you will be given an initial configuration and a target painting. Initial configurations will have the following json schema.

```
{
  "width": <number>,
  "height: <number>,
  "blocks": [
    {
      "blockId": <block-id>,
      "bottomLeft": <point>,
      "topRight": <point>,
      "color": <color>
    }
  ]
}
```

Where `number` is an integer, `block-id` is an integer block id enclosed in double quotes, bottomLeft and topRight are `[number, number]`, color is `[number, number, number, number]`. One example is given below. This is basically the initial configuration we used in lightning division.

```
{
  "width": 400,
  "height: 400,
  "blocks": [
    {
      "blockId": "0",
      "bottomLeft": [0, 0],
      "topRight": [400, 400],
      "color": [255, 255, 255, 255]
    }
  ]
}
```

Block id's will start from `0` and go to `n-1` for `n` blocks.

Full Division v.2

For the last 24 hours of the contest, we wanted to present you with a challenge worth of the time you have spent. In addition to having an initial configuration, now this initial configuration might contain a reference from a png. Your new task is to turn these pre-filled canvases to the target paintings.

New initial configurations will be in the form

```
{
  "width": 400,
  "height: 400,
  "sourcePng": "https://cdn.robovinci.xyz/initialpngs/36.json",
  "blocks": [
    {
      "blockId": "0",
      "bottomLeft": [0, 0],
      "topRight": [400, 200],
      "color": [255, 255, 255, 255]
    },
     {
      "blockId": "0",
      "bottomLeft": [0, 200],
      "topRight": [400, 400],
      "color": [200, 200]
    }
  ]
}
```

This means that, while the bottom part of the canvas is filled with (255, 255, 255, 255), upper part uses the png data given at "sourcePng" to fill itself. PNG data at "sourcePng" is the serialized RGBA data of a PNG the same shape/size/area as the canvas. Hence, each pixel coordinate corresponds to the same coordinate on the canvas.

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 - we decided no 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

- Fix(typo in question): Leondardo -> Leonardo
- Functional definition of blocks is removed as it caused confusion.
- Block definition is rewritten for clarity.
- Rest API Documentation will be announced soon.
- Size is the area of the rectangle. It can be calculated by multiplying width and height.
- Playground is aimed at learning. It has some bugs(probably ones we currently don't know too, so please act with caution)
- A shape is a rectangle
- All given canvases in the lightning round are 400x400, they are colored with RGBA(255, 255, 255, 255), sorry for the initial mistake in the specification.
- A minor bug in the implementation of the instruction validity checks is fixed. (cut [0] [200,200], swap [0.1] [0.3]) now works.
- A major bug in the implementation of swap is fixed. All submissions will be rejudged in regard to this change, system will be closed between 15.00-15.10 UTC.
- We decided to apply no changes to the specification at 12.
- A bug in the implementation of Point Cut is fixed. Please resubmit your codes in that regard.

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.

A move on a block either changes the block contents, or it destroys the old block and creates new ones.

Color and Swap changes the contents of the block.

Cut and Merge destroys(takes the block out of the scope so it cannot be referenced in the latter instructions) the old block, generates new blocks.

Merge does so by creating a new top level block where the id is created by holding a global counter incremented at each merge operation, destroying the old blocks and adding them as sub-blocks inside the newly created complex block. Sub-blocks generated via merge can never be addressed. They are only used to describe colors within a block. A simple block can only have pixels of the same color, but a complex block (consisting of multiple sub-blocks) can have multiple colors.

Cut destroys the old block, generates new blocks where the id is created by appending the old id with **.0, .1, .2 or .3**. Cut blocks are now independent from the block they originated from, even though their block-id is tied with that block.

The **initial canvas** is defined according to the initial configuration provided with the problem.

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

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.*

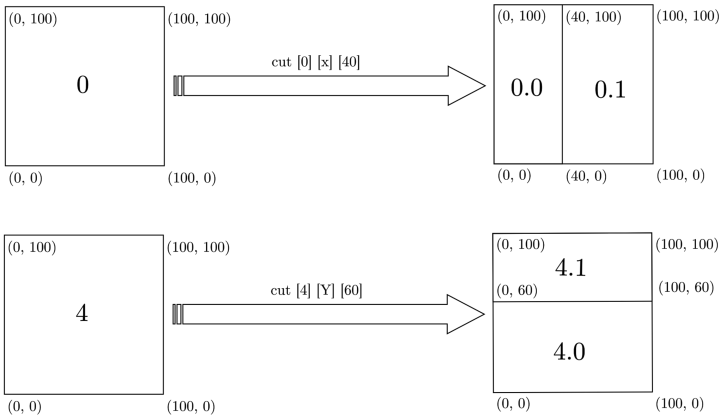

Figure 1: Line Cut Move Image

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.*

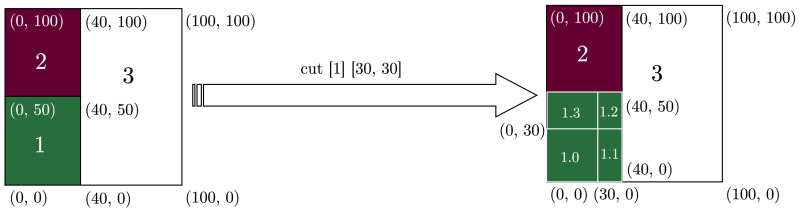

Figure 2: Point Cut Move Image

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.

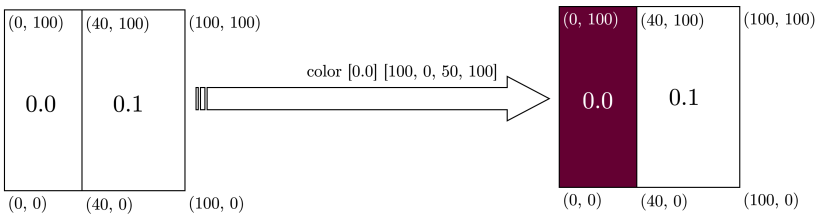

Figure 3: Color Move Image

Swap Move

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

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

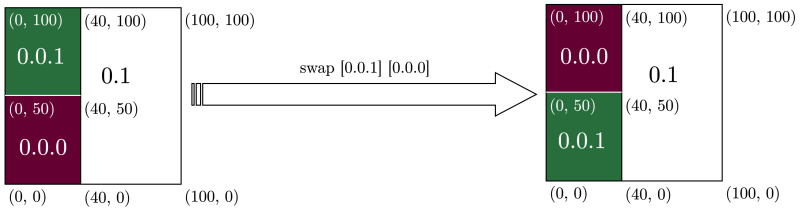

Figure 4: Swap Move Image

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.

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 $n+1$.

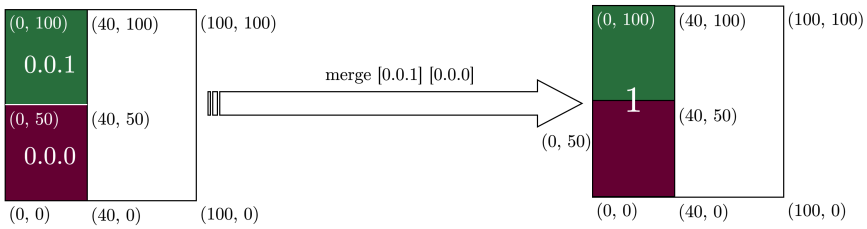

Figure 5: Merge Move Image

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 defined 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))
```

Where `size(rectangle) = rectangle.width * rectangle.height`, and round is the nearest integer. We are using Math.round function to compute the round.

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**. Typescript code for the calculation is given below;

```
 class RGBA {
    r: number;
    g: number;
    b: number;
    a: number;

    constructor(rgba: [number, number, number, number] = [0, 0, 0, 0]) {
      [this.r, this.g, this.b, this.a] = rgba;
    }
  }

class SimilarityChecker {
  static imageDiff(f1: RGBA[], f2: RGBA[]): number {
    let diff = 0;
    let alpha = 0.005;
    for (let index = 0; index < f1.length; index++) {
        const p1 = f1[index];
        const p2 = f2[index];
        diff += this.pixelDiff(p1, p2);
    }
    return Math.round(diff * alpha);
  }

  static pixelDiff(p1: RGBA, p2: RGBA): number {
    const rDist = (p1.r - p2.r) * (p1.r - p2.r);
    const gDist = (p1.g - p2.g) * (p1.g - p2.g);
    const bDist = (p1.b - p2.b) * (p1.b - p2.b);
    const aDist = (p1.a - p2.a) * (p1.a - p2.a);
    const distance = Math.sqrt(rDist + gDist + bDist + aDist);
    return distance;
  }
}
```

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.

REST API

You can use the given endpoints to do your submissions.

API Endpoints

GET /api/users Get information of your team. This can be used to verify that you can access API with your token.

POST /api/problems/$PROBLEM_ID Make a submission. The multipart/form-data body of the request should contain your isl file with the key 'file' . Returns a submission ID.

Example CURL Request :

```
curl --header "Authorization: Bearer YourAPIToken" -F
file=@your.isl https://robovinci.xyz/api/problems/1
```

GET /api/submissions/$SUBMISSION_ID Retrieves information about a submission. It returns a JSON object with the following attributes: • status: Either QUEUED, PROCESSING, SUCCEEDED, or FAILED • cost: The calculated cost, when submission is SUCCEEDED. • error: The error message, when submission is FAILED. • file_url: The link of the submitted file.

GET /api/results/user Retrieves result of all problems.

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"
```