blur (.blr)

Project Proposal

Dexter Callender | dec2148
Tim Goodwin | tlg2132
Daniel Hong | sh3266
Melissa Kaufman-Gomez| mhk2149

Motivation

In this modern age of minimalism and simplicity, Blur presents a medium to easily visualize data in an ASCII character representation. Blur can be used to gracefully produce casual, qualitative representations of image data at minimal filesize, enabling a new channel for the communication of visual data in extremely bandwidth-limited scenarios, which are frequently encountered in the emerging Internet of Things or in the growing DIY drone hobbyist community. Looking to ASCII, Blur embraces the universality of this encoding scheme to minimize incompatibilities in its data manipulation functionality. Blur provides a semantic image manipulation tool for programmers, artists, and hackers alike.

Description

Blur draws inspiration from traditional ASCII art and pixel manipulation. Blur is a lightweight programming language that focuses on the manipulation and presentation of data in Euclidean, matrix-like representations. It provides the building blocks to allow the programmer to edit this ASCII art, for example, use only certain characters, limit the number of different characters used, control the density, etc.

The language is centered around the use of the "Canvas" datatype, which is a two-dimensional grid of characters that can be drawn on using a "Point" data type, which represents a character at a specific location on the canvas. From these basic datatypes, various geometries and patterns can be coded into Blur’s standard library to more easily paint more complex objects onto the canvas using less code.

Features

- Create a canvas. Use a coordinate system to print characters and strings to the canvas.
- Upload images and convert to ASCII art.
- Add and subtract canvases to create new images.
- Upload ASCII art and make modifications.

Potential Applications

Using the building blocks defined by the language, a potential application would be allowing the user to upload any file to the canvas, and converting the data in that file to ASCII densities. This could provide alternative/unconventional ways of encoding and visualizing data.

This could also be used for QR code generation: a quick, lightweight way to produce a URI -> QR Code hash that could then be added to a lookup table.

Additionally, Blur could allow for quick unicode visualizations of small files in command-line based environments. E.G. qualitatively verifying the features of a low-resolution image on stdout without leaving the command line.
Sending highly compressed sketches of image or video data in extremely bandwidth-limited situations (E.G. potential future DIY space-probe or deep-sea rover projects).

Syntax

Basic Types

Data Type	**Example/Description**
char	ASCII character, ‘*’
boolean	true or false
int	Standard integer 30
string	Standard string “hello”

null	Standard null (no data)

Complex Types

Data Type	**Example/Description**
Array	Array a = Array.build[10];

Canvas	Implemented as a 2D array onto which the user “prints” his/her ASCII art.
	Canvases can be added or subtracted.
	
	`Canvas c = { 10, 10, ‘-’ };`
	`Canvas d = { 5, 5, ‘*’ };`
	`char b = c.getPoint(x, y);`
	`Canvas added = c + d;`

Point	Any character that we put on the canvas is of data type Point. This will be a building block for drawing shapes.
	`Point( canvasName, x, y, char );`
	`Point( c, 10, 10, ‘@’ );`

Keywords

Keyword	**Declaration**
for	`for i in range(0,100){ Point(c,i,i)  }`
if	`if( ){ }`
else	`else{ }`
else if	`else if( ) { }`
void	Standard void
CENTER	CENTER centers the shape at the given coordinate `Rectangle( canvas, 0, 0, 10, 10, CENTER )`
TOP_RIGHT	TOP_RIGHT positions the shape at the top right of the given coordinate

	`Rectangle( canvas, 0, 0, 10, 10, TOP_RIGHT )`
TOP_LEFT	TOP_LEFT positions the shape at the top left of the given coordinate `Rectangle( canvas, 0, 0, 10, 10, TOP_LEFT )`
BOTTOM_RIGHT	BOTTOM_RIGHT positions the shape at the bottom right of the given coordinate `Rectangle( canvas, 0, 0, 10, 10, BOTTOM_RIGHT )`
BOTTOM_LEFT	BOTTOM_LEFT positions the shape at the bottom left of the given coordinate `Rectangle( canvas, 0, 0, 10, 10, BOTTOM_LEFT )`

Sample Code

Basic Syntax - Function Declaration and Comments

```
// This is a single-line comment.
/* This is a multiline comment.
   Still part of the multiline comment. */
void move(Point p, int vertical, int horizontal) {
       p = p.vertical + vertical;
       p = p.horizontal + horizontal;
}
```

Canvas Drawing and Manipulation

```
Canvas c = { 100, 100, “-” };     // Canvas of size 100x100 with “-” as backdrop.
/* Loads an image, converts into ASCII, and places it onto canvas c at argument
position.
Default to placing the image at the top left, unless position (i.e: CENTER) is
specified. */
Load( c, “ascii_image.jpg”, 0, 0, CENTER );
// Draw an ellipse on the canvas with height 5 and width 3 centered at position (0,0).
Ellipse( c, 0, 0, 5, 3, CENTER );
Rectangle( c, 0, 0, 10, 10, CENTER);
Point( c, 10, 10, ‘@’ );
Line( c, 0, 5, 10, 15 );
```

```
Text( c, 0, 0, “text you want” );
paint( c, “filename.txt” );      // Paint the canvas to a file.
print(“for debugging”); // Print to stdout.

// Canvas Operators and Aggregators
Canvas d = { 100, 100, “-” };    // Create a canvas 100X100 with background “-”.
Canvas e = c + d;   // Canvas addition, first canvas (c) on the bottom.
Canvas f = c - d;   // Canvas subtraction.

// Canvas iteration
Canvas c = { 100, 100, “-” };
// Loop over the width, then the height of the canvas.
for p in width then height on c {
       // Change to a blank character if the current character is ‘0’.
       if (p == ‘0‘) {
              c = ‘ ‘;
       }
}
```

Sample Image to ASCII Art