Project Proposal

Ammar Eltigani, Priya Rajbhandary

Q1.

Islang Island

Q3.

Introduction

Our proposed language is a sketching tool for **2D fantasy maps**. The language user can draw from a list of pre-defined shapes/structures or make their own and place them onto a 2D fantasy world canvas. The language supports specifying the magnitude, orientation, and scale of a shape/structure relative to others.
This tool would make the process of sketching fantasy maps for fiction or creative projects very accessible as it requires little knowledge of traditional coding or digital design tools. The ultimate goal is to provide the new user with a plethora of pre-defined shapes/structures they can immediately adopt and use. We envision (and hope that) this library of pre-defined resources will slowly grow through the contributions of the community of power users and developers.

Design Principles

Aesthetically, the principal goal of this language is to enable users to design beautiful maps of fantasy lands, acting as a creative outlet for their imaginations. Technically, this language achieves this through a modular component-based design where smaller and simpler drawings can be combined to achieve more complex ones. A key technical design feature is that the user can draw components with specified orientation, scale, and position that are *relative* to other components.

Examples

```
//E.g.1
TwoCircles 50x50 is:
    circle  point=(5,5) radius=5
    circle  point=(5,5) radius=3
ThreeCircles 100x100 is:
    TwoCircles
    circle point=(10,10) radius=4

//E.g.3
Three_clouds 50x100 is:
    Cloud 1 unit at the center
    Cloud 1 unit at the top-right
    Cloud 1 unit at top-left
island_clouds of size 200x120 is:
    Three_clouds 1 unit at the top
    island

//E.g.3
cloud_castle 50x200 is:
    Cloud 1 unit at the center
    Castle 1 unit at the top
newland 400x400 is:
    cloud_castle, 1 unit at the top-right
    mountains 4 units at the left rotated 270
```

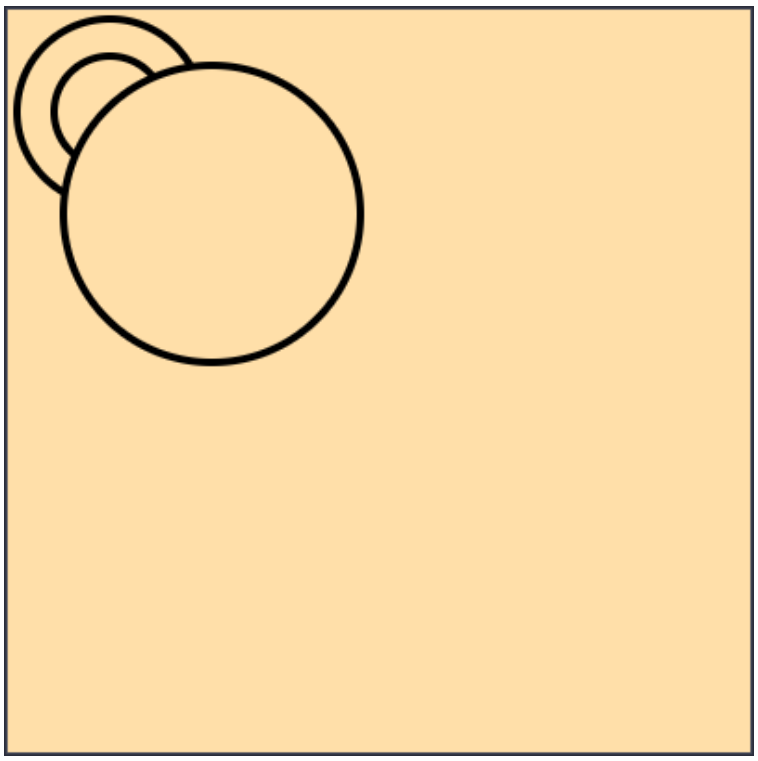

Figure 1: Circles

Figure 2: Example of one mountain scaled

Language Concepts

The core concepts of our language are the components that contain other components and/or primitives. The user builds components by nesting them together alongside (i) primitives at the bottom level, or (ii) pre-defined components. The primitives we supply are

- point
- circle

which we implement using the SVG library. We also hope to provide some very simple build-in components (combining forms) such as landmasses, islands, mountains, buildings, clouds, trees, shrubs, rods, bridges, etc. Users are encouraged to use these basic combining forms and/or primitives to create their own combining forms and build upon these components.

Syntax

We define a partial BNF syntax for our language below:

```
<point> ::= (<int>, <int>)

<circle> ::= circle <point> <int>

<name> ::= <str>

<dims> ::= <int> x <int>

<direction> ::= top | bottom | left | right | top-right
             | top-left | bottom-left | bottom-right

<position> ::= <int> unit to the <direction>

<orientation> ::= rotated <int>

<placement> ::= <position> | <orientation> | <position> <orientation> | ϵ

<definition> ::= <name> <dims> <component>+

<component> ::= <name> <orientation> <position>
          | <name> <placement>
          | <point>
          | <circle> <placement>
          | island <placement>
          | mountain <placement>
          | castle <placement>
          | cloud <placement>

<canvas> ::= <definition>+
```

Semantics

(i) We think of the primitives in our language as the set of geometric primitives we provide the user like points, circles (here we are using the terminology of the SVG package). We wish to enable the user to draw primitives using absolute coordinates on a standard-sized canvas if they wish to, which will then be made relative once an instance of this object is contextualized after it is placed inside another component. This feature is not yet reflected in our language syntax of semantics for all the primitives (only for circles) for brevity. But we will

add this at some point. Another extension to this would be to enable coordinate selection through I/O (clicking points on a blank canvas using the cursor, for example).

(ii) As stated previously, our *values* are components and they are combined by being placed into each other with relativistic spatial specifications. The *evaluation* of our program involves recursively unpacking this tree of nested components are drawing the primitives in the leaf components within the dimension bounds defined by the parent components.

(ii) Some of the F# Algebraic Data Types we will use are the following:

```
type Point = {x: int; y: int}

type Dims = {w: int; h: int}

type Direction =
| Top
| Right
| Bottom
| Left
| TopRight
| TopLeft
| BottomRight
| BottomLeft

type Position = Direction * int

type Rotation = int

type Placement = Position * Rotation

type Component =
| Name of string
| Circle of Point * int * Placement
| Island of Placement
| Mountain of Placement
| Castle of Placement
| Cloud of Placement

type Definition = {name: string; dims: Dims; components: Component list}

type Canvas = Canvas of Definition list

let CANVAS_SZ = 400
let canvas_color = "navajowhite"
```

In summary, every definition of a component will correspond to an SVG text wrapper function that is edited at each level in the inclusion of components as the evaluator makes its way to the outmost component, which we call the `canvas`. This wrapper function is first created at the lowest level where we have our primitives and is edited based on relativistic parameters at each level going up in the AST. Once we evaluator reaches the last definition string then we simply output the current versions of SVG for all our definitions to a file.

Syntax	Abstract Syntax	Type	Prec. & Assoc	Meaning
point=(x,y)	Point = x: int; y: int	Record: int;int	n/a	Represents the coordinates of a point on a grid(canvas) using the primitive int
circle point=(5,5) radius=5	Circle of Point * int	Point*int	1/left	Circle contains its origin (point) and radius which is then drawn in SVG. Both the radius and the elements of Point must evaluate to an integer.
name	Name of string	string	n/a	Represents the name of the variable used in defining a new definition
circle point=(5,5) radius=5	Circle of Point * int	Point*int	1/left	Circle contains its origin (point) and radius which is then drawn in SVG. Both the radius and the elements of Point must evaluate to an integer.
Top	Direction = Top Right Bottom Left TopRight TopLeft BottomRight BottomLeft	Direction	n/a	Evaluates which direction the component will eventually be placed within its bounding box.
1 unit at the top	Position of Direction*int	Direction * int	n/a	Evaluates the position of the current component in relation to the definition it is under. 1 unit is evaluated as an integer and the ”top” string is evaluated as a Direction.
cloud rotated 90	Rotated integer	Rotation	n/a	Evaluates the orientation of the current component.
cloud 1 unit to the top, rotated 90	Placement of Position * Rotation	Placement	n/a	Evaluates the and places the current component in relation to its definition using its rotation and position
intxint	Dims = w:int; h:int	record of integers	n/a	Dims for eg. 50x50, evaluates two integers as a record of integers as width and height. Dims falls on the definition line for a new variable that scales the shape(i.e component) or canvas.

Syntax	Abstract Syntax	Type	Prec. & Assoc	Meaning
definition dims is: cloud 1 unit at the top rotated 90	Component = name circle castle mountain cloud island	Component	n/a	Component either evaluates a name (a newly defined definition) as string of a new form or the primitive pre-built type circle, castle, mountain, island, or island.
TwoCircles 50x50 is: components	Definition of name: string; dims: Dims; components: Component list	Definition	n/a	This type defines a new variable in this case TwoCircles, that evaluates the variable name(combining form) as a string, the scale of the dimensions Dims as a record of two integers and the list of components with the primitive shapes like circle, castle etc.
definition: components definition: components	Canvas	Definition List	top to bottom	The definitions get parsed top to bottom, and using a dictionary, it builds SVG strings, concatenating each component, to each definition. At the end of the definition list, it then draws the last SVG string which is a compilation of every definition's SVG string

Remaining Work:

Evaluator:

1. Add and implement the features of direction, position, orientation, and placement, for the newly added primitive forms.
2. Add comments

Project Presentation:

Q4: