PIXCZAR

ANIMATION LANGUAGE

Language Reference Manual

Frank Aloia - fea2113@columbia.edu - Manager
Gary Chen - gc2676@columbia.edu - System Architect
Bryan Li - bl2557@columbia.edu - Language Guru
Matias Lirman - ml3707@columbia.edu - Testing

Contents

1. Introduction

PixCzar is an object-oriented, imperative programming language that provides users with convenient mechanisms for creating 2D animations. Programmers are able to define and manipulate the motion of created shapes (ellipses, rectangles, and triangles), textboxes, and uploaded images. Our language differs from existing animation-based languages because of its unique object-oriented paradigm. Conceptually, the workflow of the language can be likened to a flip-book animation. Programmers specify `Pix` objects, which are individually animated through a series of `Placements`, specifying attributes such as position at a moment in time. These objects are then written to the flip book itself, the `Frame` array, where each `Frame` contains any number of `Placement` objects.

The user will call the `render()` function that will take a Frame[] as an input and initiate a window to display the animation. The object hierarchy and strong references to Pix, Placement, and Frame objects allow for convenient data manipulation. For example, if a Pix is modified, all Frames with a Placement that references this Pix will contain the modified version.

This document details all aspects of language syntax and structure in an attempt to give users a resource for creating their PixCzar animations. In addition, it discusses the standard library which is an existing library of functions that aim to support programmers in PixCzar development.

2. Notation Used in This Manual

The PixCzar Language Reference Manual distinguishes prose, code, and grammar rules by the following font conventions:

- `code` - A monospaced font is used for both code in blocks and for code (e.g. keywords, types) in-line with prose.
- prose - A serif font is used for the content of the LRM itself.
- grammar rules - A sans serif font is used for specifying grammar rules.

3. Lexical Conventions

3.1. File Extension
PixCzar files must have file extension **.pxr**.

3.2. Entry Point

A single function `Void main()` is required. It is the desiginated entry point of the program.

3.3. Identifiers

An identifier is used for declaring variables, and is specified by a sequence of alphabetic characters, underscores, and digits. The first character must be alphabetic, and an identifier cannot be a keyword. There cannot be duplicate identifiers in a scope.

3.4. Variable Scope

Each variable only remains in scope in the declared block. If a variable is declared outside a block/function, it is global.

3.5. Comments

PixCzar supports both block comments opened by `/*` and closed by `*/`, and single-line comments opened by `//` and closed by a newline character.

3.6. Keywords

The following is a list of words reserved by PixCzar. Each of the keywords is further explained in its later relevant section.

category	keywords
basic types	Int, Boolean, Float, Array, String
advanced types	Pix, Placement, Frame, Struct
conditionals	if, else, else if
loops and branching	for, while, return, break, continue
built-in functions	print, length, render, getHash
literals	true, false, null
other	new, Void, import

3.7. Global Built-in Functions

- `Void print(Basic_Type output)` - will print any basic type to standard out
- `Void render(Frame[] frames, Int fps)` - will take a Frame[] as an input and initiate a window to display the animation at the specified fps rate. All Frames in `frames` must have the same dimensions in Pixels.

- `Int hashCode(Advanced_Type output)` - returns the hash code of an instance of an advanced data type (Section 4.2).

4. Types

All types are capitalized by convention.

4.1. Basic types

type name	description
Int	integer number
Float	floating-point number
Boolean	boolean value
String	ordered sequence of ASCII characters

4.1.1. Int

An integer number in base-10, consists of one or more digits between 0-9.

4.1.2. Float

A floating point number, consists of digits, a '.', and the decimal part in digits.

4.1.3. Boolean

A type that evaluates to either `true` or `false`.

4.1.4. String

An ordered sequence of ASCII characters. Must be enclosed by either single `'` or double `"` quotation marks.

4.2. Advanced types

Each instance of an advanced data type has a unique integer hash identifier.

type name	description
Pix	a shape, text box, or image
Placement	a collection of properties of a Pix
Frame	a single frame of animation
Array	a fixed-size collection of same-typed objects
Struct	a collection of basic types

4.2.1. Pix
Short for a Pixellation, a Pix is any object that can be animated, such as an image, rectangle, triangle, ellipse.

- Constructor: `Pix()`
- Built-In Functions (Modifiers for existing Objects)
  - `uploadImage(String path, Int width, Int height)`
    path to PNG image, width in pixels, height in pixels
  - `makeEllipse(Int width, Int height, Int[] rgb)`
    makes ellipse, `width` in pixels, `height` in pixels, `rgb` specifies color and is array `[r, g, b]` with each field between 0-255
  - `makeRectangle(Int width, Int height, Int[] rgb)`
    makes rectangle, `width` in pixels, `height` in pixels, `rgb` specifies color and is array `[r, g, b]` with each field between 0-255
  - `makeTriangle(Int length, Int[] rgb)`
    makes equilateral triangle, `length` in pixels, `rgb` specifies color and is array `[r, g, b]` with each field between 0-255
  - `makeTextBox(String text, Int fontSize)`
    makes textbox, `text` is content, `fontSize` is size of text
  - `remove()` - delete all pixels of Pix

4.2.2. Placement
A Placement describes the properties of a Pix at a moment in time.

- Constructor: `Placement(Pix ref, Int x, Int y, Int rank, Int group)`
  - `Pix ref` - reference to a Pix
  - `Int x`, `Int y` - position of Pix on a Frame in pixels
  - `Int rank` - denotes priority in which objects are layered on top of each other. Higher ranked objects will appear on top of lower ranked ones. Overlapping behavior of two objects with the same rank is undefined.
  - `Int group` - set the group for the object; specific functions can be called that will act on all Placements within the same group.

4.2.3. Frame

A single moment in animation, similar in concept to a page in a flip-book. Holds an array of Placements that define object positioning on a screen.

- Constructor: `Frame(Int width, Int height)` - width and height of the physical dimension of the frame, both in pixels
- Additional Attributes: `Placement[] placed` (Array of all Placements existing in the frame)
- Built-In Functions:
  - `addPlacement(Placement place)` - add Placement to `placed` Array of Frame
  - `removePlacement(Placement place)`

4.2.4. Array

A fixed-sized collection of same-typed objects.

- Built-In Functions:
  - `length()` - will return number of elements in the array

4.2.5. Struct

A collection of basic types.

5. Expressions

5.1. Precedence and Associativity Rules

Tokens (From High to Low Priority)	Associativity
() [] .	L-R
! ++ - -	R-L
* / %	L-R
+ -	L-R
> >= < <=	L-R
== !=	L-R
&&	L-R
||	L-R
=	R-L
,	L-R

5.2. Primary Expressions

Basic building block expressions:

- identifiers
- constants - integer number, floating point number, boolean, string, or null
- advanced types - Pix, Placement, Frame, Array, Struct
- `id( expr* )` - function call with an optional comma-separated list of arguments
- `id.id( expr* )` - built-in object function call with an optional comma-separated list of arguments
- `(expression)` - paranthesized expression
- `(identifier . member_of_data_type)` - accesing member of Struct, Placement, or Frame

5.3. Unary Operators

- `-(expression)` - evaluates a float or int to the negative value.
- `!(expression)` - logical negation for booleans or conditionals.
- `(expression)++; ++(expression)` - adds 1 to an int; if placed after the int, the value of the expression is the operand's original value; if placed after the int, the value of the expression is the operand's incremented value
- `(expression)--; --(expression)` - subtracts 1 from an int; if placed after the int, the value of the expression is the operand's original value; if placed after the int, the value of the expression is the operand's decremented value

5.4. Arithmetic Operators

For any (int) <operator>(float), the int will be cast to a float and the operator will return a float. In all other situations, operands must be of the same type.

Operation	Legal Types	Description
`expr1 + expr2`	Int/Float/String	Ints/Floats: returns sum
		Strings: returns concatenated string
`expr1 - expr2`	Int/Float	returns subtracted value
`expr1 * expr2`	Int/Float	returns multiplied value
`expr1 / expr2`	Int/Float	returns divided value
`expr1 % expr2`	Int	returns modulo of ints

5.5. Relational/Equality Operators

All return booleans.

Operation	Description
`expr1 > expr2`	greater than
`expr1 < expr2`	less than
`expr1 >= expr2`	greater than or equal to
`expr1 <= expr2`	less than or equal to
`expr1 == expr2`	equality
`expr1 != expr2`	inequality

5.6. Assignment Operators

`(id = expr)` will set the variable represented by the identifier to the expression. The variable and the expression must be of the same type. The assignment expression will return the value of the expression.

5.7. Array Operators

- `arr[ (idx) ]` - accessing the element of array `arr` referenced by the integer `idx`
- `arr[(inc) :  (exc)]` - returns a new sub-array of `arr` comprised of the elements referenced by the integer `inc` to integer `exc` - 1

6. Statements

6.1. Expression Statement

Simplest form of statement:

```
( expression );
```

6.2. Return Statement

Denotes a value to be returned from a function:
`return ( optional_expression );`
If no expression is provided, the return value is null.

6.3. Compound Statement

Allows several statements to be used where one is expected:
`{ statement_list; };`

6.4. If Statements

Evaluates condition(s) and executes the statement for the satisfied case.

- `if`: executes a statement if the condition evaluates to a positive (or non `false`, or non `null`) value.
  `if (`*`condition`*`) {`*`statement;`*`}`
- `else if`: any number of `else if` statements are associated with an `if`, and each specifies an alternative condition and statement.
  `if (condition) {statement;} else if (`*`condition`*`) {`*`statement;`*`}`
- `else`: executes a statement if no previous conditional is satisfied; will be connected to the last encountered elseless if
  `if (condition) {statement;} else {`*`statement;`*`}`

6.5. Loop Statements

PixCzar allows for both `for` and `while` loops, as well as branching statements `break` and `continue` for further loop control.

Loops:

- `while`: continually executes code block while condition is a positive (or non `false`, or non `null`) value.
  `while (`*`condition`*`) {`*`statement;`*`}`
- `for`: execute code block a specified number of times. The 3 expressions it takes are *initialization*, which initializes the loop, *termination*, which specifies an exit condition, and *increment*, which is invoked after each iteration of the loop.
  `for(`*`initialization, termination, increment`*`) {`*`statement;`*`}`

6.6. Branching Statements

- `break`: terminate the current (inner-most) loop. Note: loops consist of `for` and `while` iterations, and do not include `if` or `else` iterations.

- `continue`: skip the current iteration of a loop. Afterwards, the loop condition will be re-evaluated for the next iteration.

6.7. Declaration Statements

6.7.1. Default Type Values

Default values for each variable type:

Type	Default Value
Int	0
Float	0.0
String	””
Boolean	false
Pix/Placement/Frame	null
Array	null

6.7.2. Basic Type Declaration

To declare a variable id with a basic type (expression must be of type `typ`):

```
typ id = ( expression );
```

6.7.3. Pix/Placement/Frame Declaration

All variables of types Pix, Placements, or Frames are strong references to objects. To set a variable id to an existing object referenced by variable obj:

```
type id = obj;
```

To declare new objects, use the `new` keyword with the object constructor:

```
type id = new type( expr1, expr2, ...);
```

6.7.4. Array Declaration

To declare a new Array of type `typ` and size `size`:

```
typ id = new typ[size];
```

Values can also be initialized during declaration:

```
Int arr = [0, 1, 2, 3];
```

If not initialized, each element will be the default type value.

6.7.5. Struct Declaration

Struct types are declared as follows:

```
Struct Point{
Int x;
Int y = 0;};
```

Any number of variables can be declared (and optionally initialized) inside the brackets. The struct type name follows the same rules as normal identifiers.

Declaring a variable of a Struct type:
`Struct Point pt;` - Internal variables are set at default values if not initialized in the struct type declaration

6.7.6. Multiple Declarations in a Line

Multiple variables of the same type can be declared (and optionally initialized) in a single line. They are separated by commas:

```
Int x, y = 0, z;
```

6.8. Advanced-Type Function Calls

As described in Section 3, Pix, Placements, and Frames have built-in function calls that modify existing objects:

```
obj.func(expr*);
```

This will call a function func with the specified arguments on the object obj.

7. Additional Control Flow

7.1. Functions

Functions have any number of input arguments and a single return type. If the return type is Void, the function does not require a return statement. Function names follow the same rules as regular identifiers. There are no nested functions or high-order functions. They are declared as follows:

```
Type Function_Name(expr*){ statement_list; }
```

7.2. Linking Files

The contents of other .pxr files can be linked to the working file. There can only be one `Void main()` function among linked files. Files are linked using the `include` keyword: `include <pathname>`.

Pathname is a string denoting the relative path in the directory. The `include` keyword can be used in a statement at any point in the file and the contents of the linked file will always be global in scope.

8. Standard Library

The standard library will be a list of functions in file "stdlib.pxr".

- `Void adjustPlacements(Frame frame, Int dx, Int dy, Int group` - adjust the position by (`dx, dy`) pixels for all objects with the specified group number in a Frame
- `Void fillFrames(Frame[] frames, Placement placement, Int start, Int end` - add a Placement to every Frame in an Array from index `start` through `end`
- `Void addPlacementsFromFrame(Frame source, Frame destination, Int group)` - adds all Placements from `source` with the specified group number to `destination`; if `group` is -1, all Placements are added
- `Void keyFrame(Frames[] frames, Int start, Pix obj, Int[] from, Int[] to, Int duration)` - adds Placements to `frames` starting at index `start`, making `obj` move from point `from` to point `to` over a specified duration

9. Sample Code

9.1. Bouncing Ball

Simulates the animation of a ball bouncing up and down:

```
Void main() {
    Frame[] framesReel = new Frame[6];
    for(Int i = 0; i < 6; i++) {
        framesReel[i] = new Frame(10, 10);
        /* Creating 10X10 pixel Frames */
    }

    Pix ball = new Pix();
    ball.makeEllipse(2, 2, [255,255,255]);

    Placement p1 = new Placement(ball, 5, 5, 1, 1);
    Placement p2 = new Placement(ball, 5, 0, 1, 1);

    for(Int i = 0; i < framesReel.length() ; i++) {
        if (i % 2 == 0) {
            framesReel[i].addPlacement(p1);
        } else{
            framesReel[i].addPlacement(p2);
        }
    }
    render(framesReel, 30);

            }
```

9.2. Adjust Placements

Adjust Placements Library Function:

```
Void adjustPlacements(Frame frame, Int dx, Int dy, Int group)
{
    Placement[] placements = frame.placed;
    for(Int i = 0; i <= placements.length(); i++) {
        if (placements[i].group == group) {
            placements[i].x += dx;
            placements[i].y += dx;
        }
    }
}
```

9.3. Fill Frames

Fill Frames Library Function:

```
Void fillFrames(Frame[] frames, Placement placement, Int
start, Int end) {
    for(Int i = start; i <= end; i++) {
        frames.addPlacement(placement);
    }
}
```

9.4. Add Placements From Frame

Add Placements From Frame Library Function:

```
Void addPlacementsFromFrame(Frame source, Frame destina-
tion, Int group) {
    Placement[] placements = source.placed;
    for(Int i = 0; i <= placements.length(); i++) {
        if (placements[i].group == group || group == -1)
            destination.addPlacement(placements[i]);
    }
}
```

9.5. Key Frames

Key Frame Library Function:

```
Void keyFrame(Frames[] frames, Int start, Pix obj, Int[]
from, Int[] to, Int duration) {
    Float xTimeStep = (to[0]-from[0])/duration;
    Float yTimeSTep = (to[1]-from[1])/duration;

    for (Int i=0; i<frames.length(); i++) {
        frames[i].addPlacement(new Placement(
            obj, from[0] + i*xTimeStep,
            from[y] + i*yTimestep, 1, 1));
    }
}
```