# CRASH

A practical graphical animation tool
05/13/03 Rev. A

By:
Mikhail Litvin
Vadim Belobrovka
Michael Anikin
Daniel Burdeinick

# Table of Contents

5. Crash's Architectural Design
    a. Block Diagram
    b. Interfaces
    c. Implementation Assignment

6. Test Plan

7. Conclusions
    a. Lessons learned
    b. Advice

8. Appendix

# Chapter 1 - Introduction

## a. Introduction

Since the dawn of humanity men were searching for a graphic and animation tool. For millennia there was nothing but dusk until God willed Macromedia Flash. But story of graphics and animation was far from being over, for All Mighty has played a joke on mankind and gave them a team of mediocre developers and willed them to create a fairly useless imitation of Flash. And it was called Crash. And it was good.

Crash language will be implemented using Java and platform independent graphic library, OpenGL. The idea of the language is to make it object oriented, where all the objects will have predefined methods with which to alter object properties. The following objects will be available: Scene, Animator, Layer, Path, and Element. The usefulness of this schema is that all the details of drawing, synchronization of the animation and other annoying nuances will be hidden behind this convenient abstraction. In the rest of the document we will be discussing the significant of these objects in the language and the concepts behind them.

## b. Main CRASH's objects:

### 1. Element

Element is one of the basic objects of the Crash language. Through this object we will be able to draw basic geometric figures like line, circle, rectangle and other. This object will have methods for changing the elements various properties like dimensions, color and other. The Element will be a complex object since one would be able to

combine several Elements into one compound Element; the only thing that the user will have to provide is the point of contact of the basic Elements. This feature will allow the programmer to create interesting geometrical figures and allow to conveniently treating them as one object. This object will also support such intrinsic methods as simple transformation, rotation and other.

## 2. Path

Another basic object of this language is a Path. Path is also a complex object that can combine in itself several simpler Paths; the programmer will only have to define the points of contact for these simple Paths. The path will have various properties that can be altered through intrinsic methods. It will also support basic transformation methods that will allow for greater variety of Paths. The concept of the Path allows the programmer to define a location on the screen where the Element objects will be situated at any point in time of the animation. For example, if the programmer will want to animate an Element from top left to the lower right of the screen, then he will define a Path to be a straight diagonal line. However, if the programmer wants a stationary Element throughout the animation he will define its Path as a point located anywhere on the screen.

## 3. Layer

For the convenience of the programmer, Elements and Paths can be separated into various Layers. A Layer objects carries a role of deciding which objects during animation will be on to and which on the bottom of the animation sequence. For example, if there would occur a graphical overlap of two Elements then which one would be culled and which will stay on top.

## 4. Animator

Animator object is present for the sake of animating a particular Element along a particular Path. The whole of the animation sequence will consist of the collection of

these minor Animators. The definition of the Animator requires that the programmer specifies several things about this single animation. First, a single Element and a singe Path along which this Element will be animated. Second, the start and the end points on this Path where this animation will begin and end for this Element. Third, the corresponding time frames for the beginning and end of this basic animation.

### 5. Scene

The Scene object is carries a role of defining the framework of the whole animation sequence. This main object allows the programmer to define the size of the screen that will hold the animation, the total duration in time frames, animation background and other initiations to the animation sequence.

## c. Concept of Animation

Now, let's discuss a simple example through which we can understand the interactions of these objects and their contribution to the whole animation sequence.

1. First we define a scene object. Here we specify that the active pane will be 200x200 pixels. The duration of the sequence will be 100 time frames. Time frame is predefined in its time duration, and is the basic unit of time in Crash. We set background to be while. It has begun!
2. Now we define an Element as a circle with a radius of 5 pixels and through and intrinsic method make its color to be red. We fill the circle with yellow.
3. Next we define 2 simple Paths which we will combine into a complex Path. Path 1 will be a straight horizontal line from point (10, 10) to point (120, 10). Path 2 will be a vertical line from point (120, 10) to point (120,120). These two Paths will be combined into a complex Path 3. Only paths with common end points can be combined into a complex Path. Here the common point is (120, 10).

4. Now we create a Layer object and assign this circle to this layer. (Layer in this example is not necessary for we will have only one Element, but more then one will call for layers to resolve the collision problem. If we would have another Element then we would create another layer and identify which layer would lie on top of which.)

5. Next we identify several Animator objects. With the first one we identify that the circle will be motionless at position (10, 10) for the duration of 20 time frames. This is done by identifying that at time frame 0 the circle will be attached by its center to point (10,10) on Path3, and at time frame 20 the circle will be attached by its center to point (10,10) on Path3. Now, to animate the circle along the Path3 curve for the rest of the duration of the sequence we define the following. At time frame 20 the circle will be attached by its center to point (10, 10) on Path3, and at time frame 100 the circle will be attached by its center to point (120,120) on Path3.

The above example did not nearly illustrate all the abilities of the Crash language, but it gave a sense of its possibilities. Enjoy debugging you own Crash scripts.

d. **Input/Output**

The Crash language will take files in ASCII format as input. The code can be indented for easier readability, but the indentation will have no effect on the input. The separator will be the end_of_line character. The intermediate computations will be done in C++, and then the program will be compiled in the executable machine code using C++ compiler.

e. **Variables**

We will implement the following variables in our language: floats -- to carry out the bulk of the computation, integers -- to be used in loops and for animations, and colors for different elements.

**f. Structures**

The input files for the Crash language will be sequentially executed line by line. The language will use if/else control flow structures as well as while loops. Curly braces will enclose these structures.

**g. Example of syntax**

```
/*The following line creates a new Scene object.  The instance is called ball:*/
Scene ball(5);
/*The next two lines show the usage of the intrinsic methods of the Scene object. Setpane(int width, int height) is used to change the dimensions of the active pane on which the animation will take place. Setduration(int number_of_time_frames) is used to change the duration of the animation sequence.:*/
ball.setpane(200,200);
ball.setduration(100);
```

**White Paper Summary**

The Crash language will be a simplified Flash-like object oriented language, easy to program, pleasant to use and to watch. Through short and simple scripts programmers will be able to come up with various pictures and animations.

The Crash language will have a lot of positive effects on the programming world. Most importantly it will make people appreciate Flash a little bit more. But it will also help the team of mediocre programmers to learn the basics of the language design, so that in the future the aforementioned team would be able to master the field and come up with something really useful.

# Chapter 2 - Getting Started

## Tutorial

With Crash's flexible object architecture, we are going to be able to build functional animations with a very reduced number of instructions, delegating most of the work to Crash's main engine and providing high-level graphics simulations. The following paragraphs will describe how to create, compile and run a simple but functional Crash program from scratch.

## Creating CRH Crash source files

The Crash language is a script file interpreter. Before you can do anything useful you must have at least one Crash file to interpret. Crash files must be formatted as plain ASCII text files. You can create these files using Emacs, VI, Windows Notepad or any text editor, nevertheless other word-processors will let you save scripts in plain text files.

The ".crh" filename extension is used in this manual for Crash file extensions, but you can use others just as well. Each line in a Crash file contains a statement, and a statement can be of a maximum of 255 characters long (Please refer to the Language Reference Manual for information on valid statements and commands). Indentation does not affect the behavior of the program but it is recommended for a more elegant and comprehensible source code.

## Our First Crash Program

Our first crash program will simply create a new Scene object with a yellow circle on the upper right corner. In order to do this, open a text editor such as Emacs and type the following code:

```
StartCRASH
Scene.setSize (600,600);
Scene.setFrameRate (20);
Color Blue (0.0,0.0,1.0,1.0);
Color Yellow (1.0,1.0,0.0,1.0);
Scene.setBackground (Blue);
Element Sun (Yellow,10);
Sun.addVertex(250,50);
Sun.addVertex(5,5);
EndCRASH
```

Please note that the first line of this file must be StartCRASH so that our compiler recognizes this file as a crash program. The following 2 lines set the properties of the default object Scene such as the canvas size and the Frame Rate. Following after setting the properties of the Scene object we can declare our variable and subsequent objects. This program alone as it is, draws a blue background canvas of size 600x600, a refresh frame rate of 20, with a static yellow ellipse element on the upper-right corner. There are no animations as of yet as we will add it in the following section to show you a step-by-step implementation.

Next we save the file with its proper extension, name it “world.crh” . and run the script with our interpretor.

## Using Variables

A variable is simply a placeholder for a value or object. In Crash a variable's value can be of int, float, or an object (Scene, Animation, Color, Element, Path, Layer). Note in our simple example we declare 2 variables of type Color, Blue and Yellow, and then we use them as parameters when methods for other objects are invoked.

# Chapter 3 – Language Reference Manual

**Introduction**

This Language Reference Manual describes the Crash programming language. Crash is an object-oriented language because thinking of objects is a comprehensive and natural way to deal with complex systems.

Crash is a language designed to help developers build functional graphical/text animations, facilitating the end user to generate animations in an intuitive way and simultaneously shielding the user from all the complexities that are related in building such animation. At the same time, Crash will be considered a high level language because it will be able to generate functional animations with very few instructions.

A Crash program is a sequence of commands or statements saved in a file with extension ".crh", this program will contain the desired instructions to generate the animation. Furthermore, the file must start with the Keyword: "StartCRASH" and end with the keyword "EndCRASH". Keyword definitions and characteristics will be explained in later chapters.

Each statement in a Crash program will be typed on a separate line; in other words, each instruction will be separated with the ASCII characters 10 and 13 (CRLF).

**Types**

The Crash language will have four types: int, float, char, and color. Color is a complex type, which consists of four float parts: red, green, blue (denoted as r, g and b)

and transparency (denoted as alpha). Each color component has range of 0.0 to 1.0. Colors can either be defined through constructor or by access methods. Strings are available for display during animation.

**Operations**

Crash supports the standard '+', '-', '*', and '/' operations for integers and floats. Addition and subtraction can be used to change colors. Since color has range from 0 to 1, arithmetical operations that result in a value outside this range will be set to either 0 or 1, depending on which value is closer to the result.

**Crash Program**

The code that will be processed by the compiler must start with the Keyword: "StartCRASH" and end with the keyword "EndCRASH.

Program -> 'StartCRASH' BlockOfCode 'EndCRASH';

**Crash Code**

The Crash language supports if statements and while loops. The while loop will have the expression, that is being evaluated, between the keywords 'While' and 'Do'. The end of the loop is to be marked by the keyword 'EndWhile'. The if statement is implemented the same way it is implemented in other programming languages – each 'else' matches the closest unmatched 'then'. The end of the if statement is marked by the 'EndIf' keyword.

BlockOfCode -> Statement BlockOfCode | ControlFlowBlock BlockOfCode | ;

ControlFlowBlock -> IfStatement | WhileLoop;
WhileLoop -> 'While' Expression 'Do' BlockOfCode 'EndWhile';
IfStatement -> MatchedStatement | UnmatchedStatement;
MatchedStatement -> 'If' Expression 'Then' MatchedStatement 'Else'
MatchedStatement ‘EndIf’;
UnmatchedStatement -> 'If' Expression 'Then' IfStatement ‘EndIf’ | 'If' Expression
'Then' Statement ‘EndIf’ | 'If' Expression ‘Then' MatchedStatement 'Else'
UnmatchedStatement ‘EndIf’;

**Statements, Declaration, Assignments**

Each statement is a declaration, any kind of assignment or a void function call. All variables must be declared before they are used. Several variables may be declared at once; in this case their names have to be separated by commas. There are two ways to declare an object – simply to declare it first and then set its parameters through access methods, or to declare the object using the constructor.

Statement -> Declaration | Assignment | VoidFuncCall;
Declaration -> Object Identifier ContinueDeclaration | ExtendedDeclaration;
ExtendedDeclaration -> Object Identifier Constructor ContinueDeclaration;
ContinueDeclaration -> ',' Identifier ContinueDeclaration | ;
Constructor -> '(' Parameter ContinueConstructor ')';
ContinueConstructor -> ',' Parameter | ;
Assignment -> Identifier '=' Expression;

**Expressions**

The following grammar is not ambiguous and is not left recursive. Atom is any object or type.

Expression -> Term '+' Expression | Term '-' Expression | Term;

Term -> Atom '*' Term | Atom '/' Term | Atom;

**Comments**

For multi-line comments in a Crash program, it should begins with the characters /* and ends with a matched */. Otherwise, for single-line comments, the comment should begin with the characters // and end with the first subsequent occurrence of a new line.Note: Comments enclosed by /* and */ can be nested.

**Casting**

Crash supports implicit casting – casting from integers to floats without mentioning. If an expression contains only integers, the result is an integer, but if there is at least one float, the result is a float.

**Constants**

The Crash language also supports the declaration of constants.

ConstantDeclaration -> ‘const’ Identifier ‘=’ Constant

## Crash Classes

# Scene

**Scene** is the top-level (global) object, which cannot be instantiated. The Scene object is carries a role of defining the framework of the whole animation sequence. This main object allows the programmer to define the size of the screen that will hold the animation, animation background and other initiations to the animation sequence.

- Methods

Start()

- setSize(**int** *width*, **int** *height*)
  Description: Sets the size of the window in pixels. Default is 300x300. Origin is in the middle of the screen.
  Return value: **void**
  Arguments:
  - *width* – width in pixels
  - *height* – height in pixels

- setBackground(**Color** *color*)
  Description: Sets the background **Color**
  Return value: **void**
  Arguments:
  - *color* – new background color. Default color is white.
- startAnimator(**Animator** *animator*)
  Description: Adds an **Animator** to the **Scene** and starts it.
  Return value: **void**
  Arguments:
  - *animator* – **Animator** to be added
  - *delay* – delay in seconds. Animation will start after the delay. Default 0.0.

- random()
  Description: Generates a random **float** number from 0.0 to 1.0
  Return value: **float**
- print(**float** number)
  Description: Outputs **float** to console.
  Return value: **void**
  Arguments:
  - *number* – number to output for debugging purposes.

- println(**float** number)
  Description: Outputs **float** to console with a new line character at the end.

Return value: **void**
Arguments:
- *number* – number to output for debugging purposes.

- exit()
  Description: Exits immediately
  Return value: void
  Arguments: none
- sleep(**float** *seconds*)
  Description: Pauses all code execution, **Animators** are still running
  Return value: **void**
  Arguments:
  - *seconds* – period in seconds

# Animator

**Animator** object is present for the sake of animating a particular **Element** along a Particular **Path**. The whole of the animation sequence will consist of the collection of these minor **Animator**s. The definition of the **Animator** requires that the programmer specifies several things about this single animation. Several **Elements** may be added to the animator, which will be treated as a single complex one. If multiple **Path**s are added to the animator, they are concatenated in a single **Path**. First vertex of the first **Element** starts from first vertex of the first **Path**.

## Declarations:

- **Animator** animator()

  Creates a new **Animator** object.

  Arguments: none

## Methods

- addPath(**Path** *path*)
  Description: Adds a **Path** to the **Animator**. Multiple **Paths** may be added. In this case they are concatenated in one single **Path**. Note that if Paths' endpoints do not agree a jump will occur
  Return value: **void**
  Arguments:

- *path* – **Path** to be added

- addElement(**Element** *element*)
  Description: Adds a **Element** to the **Animator**. Multiple **Elements** may be added. In this case they are treated in one single complex **Element**.
  Return value: **void**
  Arguments:
  - *element* – **Element** to be added
- **int** getStatus()
  Description: returns the status of the **Animator.** 0 – Running, 1- Finished, 2-Paused, 3-Did_not_start
  Return value: status of the **Animator**, type **int**
  Arguments: **void**
- setStatus(**int** status);

  Description: Sets the status property of the **Animator.** 0 – Running, 1-Finished, 2-Paused, 3-Did_not_start
  Return value: **void**
  Arguments: *status* – status value to set animator to.
- setLayer(**Layer** layer)
  Description: sets the **Element's Layer**.
  Return value: **void**
  Arguments: *layer* – **Elements's Layer**

# Path

Another basic object of this language is a Path. Path is also a complex object that can combine in itself several simpler Paths. Path supports basic transformation methods that will allow for greater variety of Paths. The concept of the Path allows the programmer to define a location on the screen where the Element objects will be situated at any point in time of the animation. For example, if the programmer will want to animate an Element from top left to the lower right of the screen, then he will define a Path to be a straight diagonal line. However, if the programmer wants a stationary Element throughout the animation he will define its Path as a point located anywhere on the screen. Path trajectory is defined by an Element.

## ➢ Declarations:

- **Path** path(**Element** *trajectory,***float** *duration*)
  Creates a new **Layer** object.
  Arguments:

- *trajectory* – trajectory specified by an **Element** object
- *duration* – how much time should Element travel across the **Path**

➢ Methods:

- setRotation(**float** *degrees,x,y*)
  Description: Specifies the rotation amount of the **Path**.
  Return value: **void**
  Arguments:
  - *radians* – rotation amount in radians. If positive clockwise, negative – counter clockwise. Default value is 0.0.
- setScale(**float** *scalex*, **float** *scaley*)

  Description: Specifies the scaling amount of the **Path**.

  Return value: **void**

  Arguments:
  - *scalex* – scaling along x axis. Value must be positive. Default value is 1.0(no scaling).
  - *Scaley* – same but along y axis
- setTrajectory(**Element** *trajectory*)

  Description: Specifies the trajectory of the **Path**.

  Return value: **void**

  Arguments:
  - *trajectory* – trajectory specified by an **Element** object
- **Element** getTrajectory()

  Description: Returns the trajectory of the **Path**.

  Return value: the trajectory of the **Path**, type **Element**

  Arguments: **void**
- setDuration (**float** *duration*)

  Description: Specifies how long should **Elements** 'travel' along the **Path**.

By default is 10 secongs.

  Return value: **void**

  Arguments:
  - *duration* – duration in seconds

# Layer

For the convenience of the programmer, **Elements** and **Paths** can be separated into various **Layers**. A **Layer** objects carries a role of deciding which objects during **Animation** will be on to and which on the bottom of the animation sequence. For example, if there would occur a graphical overlap of two **Elements** then which one would be culled and which will stay on top. Lower value of the **Layer** means higher priority.

- Declarations:
  - **Layer** layer( **int** *priority*)
    Creates a new **Layer** object.
    Arguments:
      - *priority* – priority of the **Layer,** lower value means higher priority
- Methods:
  - **int** getPriority()
    Description: Returns the priority of the **Layer**.
    Return value: current priority of the **Layer**, type **int**
    Arguments: **void**
  - setPriority(**int** priority)
    Description: Sets priority of the **Layer**.
    Return value: **void**
    Arguments:
      - *priority* – new priority

# Element

**Element** is one of the basic classes of the Crash language. Through this class we will be able to draw basic geometric figures like line, rectangle and

other. Different types of the **Element** will allow the programmer to create interesting geometrical figures and allow to conveniently treating them as one object. The **Element** class closely relate to OpenGL graphical primitives. Each **Element** consists of a set of vertices (x and y coordinate), which define the shape of an **Element** depending on its type. Vertices are added in order via **Element's** addVertex method.

Types of the Element class:

Note: n – number of vertices.

- Point

  Draws a point at each of the n vertices

- Line

  Draws a series of unconnected line segments. If n is odd, the last vertex is ignored

- Line strip

  Draws a line segment from vertex0 to vertex1 to vertex2 and so on. n must be greater than 1.

- Line loop

  Same as line strips, except that a final line segment is drawn between the first and the last vertex to create a closed loop.

- Triangle

  Triples of vertices interpreted as triangles. Extra vertices are ignored if they do not form a triplet.

- Triangle Strip

  Linked strip of triangles.

- Triangle Fan

  Linked fan of triangles.

- Quad

  Quadruples of vertices interpreted as four-sided polygons

- Quad Strip

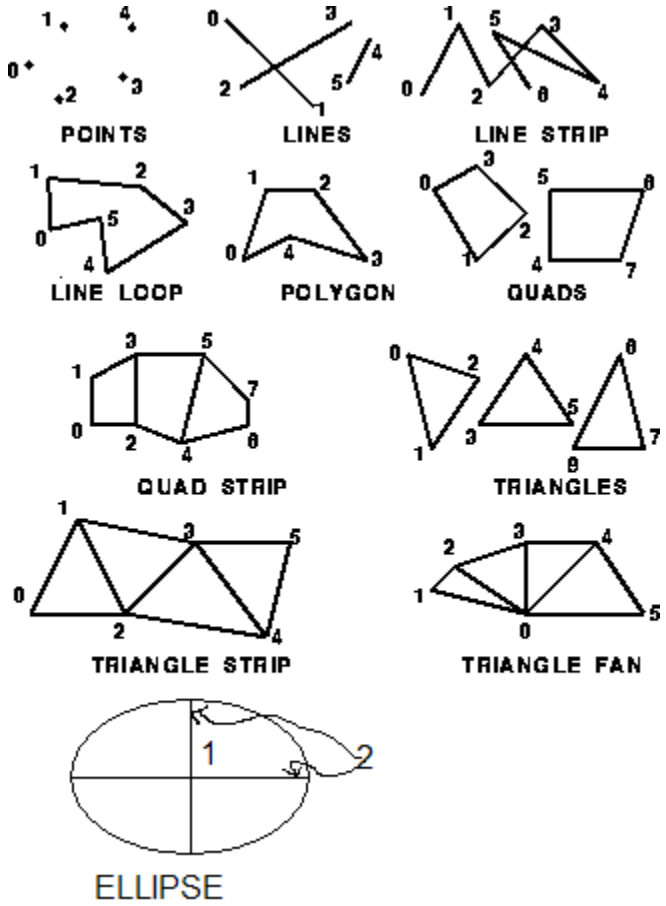
Linked strips of quadrilaterals

- Polygon

  Boundary of a simple, convex polygon

- Ellipse

  Draws an ellipse. First vertex is the center. Second is a pair of radiuses (horizontal and vertical)




There are no curved primitives except ellipse, however any curve can be approximated by a series of small lines, if those lines are small enough (less than a pixel), on pixel display it will look exactly as is supposed to look (no approximations, edges, etc)

- Declarations:

- **Element** element (***Color*** *color,* **int** *type*)

  Creates a new **Element** object.

  Arguments:

  - *color* – color of the **Element**
  - *type* – specifies the type of the **Element**
    - Point – 0
    - Line –1
    - Line strip – 2
    - Line loop – 3
    - Triangle – 4
    - Triangle Strip – 5
    - Triangle Fan – 6
    - Quad – 7
    - Quad Strip – 8
    - Polygon – 9
    - Ellipse – 10

## Methods:

- addVertex(**int** *x*,**int** *y*)

  Description: adds a vertex to the end of the **Element**'s vertex list.
  Return value: **void**
  Arguments:
  - *x* – horizontal coordinate in pixels
  - *y* – vertical coordinate in pixels

- setType(**int** *type*)

  Description: Sets the type of the Element.
  Return value: **void**
  Arguments:
  - *type* – on of the types specified above
- setColor(**Color** *color*)

  Description: Sets the color of the **Element**.
  Return value: **void**
  Arguments:
  - *color* – the color the **Element** will be set to

- fill(**Color** *color*)
  Description: filles the element with a specified color.
  Return value: **void**
  Arguments:
  - *color* – the color that the **Element** will be filled with

**Example, snippets of code**:

```
/* Start statement */
StartCRASH
Layer l(4.0); /* initalization of a layer and it's priority */
l.setPriority(5.0);
int ggg(5);

ggg=l.getPriority();
Animator a; /* create an animator, put it on layer l */
a.setLayer(l);
color cc(0.1,0.1,0.1);
Element e(cc,2); /* create a line strip, set its color to white */
e.addVertex(4,4);
e.setType(4);
e.setColor(cc);
e.fill(cc);
Path p(e,5.0);
p.setTrajectory(e);
p.setScale(1.0,1.0);
p.setRotation(1.0,56,45);
p.setDuration(1.0);
Scene.setBackgroundColor(cc);
Scene.setSize(1000,100);
Scene.start();
```

```
a.addPath(p);
a.addElement(e);
a.setStatus(2);
int f;
f=a.getStatus();

float nef(5345);
Scene.print(nef);
Scene.println(Scene.random());

Scene.startAnimator(a);
Scene.sleep(3);
Scene.exit();
EndCRASH
```

# Chapter 4 – Project Development Plan

**Planning, specification, development and testing**

Crash's development team spent numerous days in the careful planning, specification, development and testing phase of the project. This language development was done cooperatively with all the members of the team, involving architecture, grammar, parsing, error handling and testing.

The main theme in our language development planning consisted of the listing and ordering various activities so as to achieve the goals that were not precisely defined during the origination of Crash. It was only after we wrote the first revision of the White Paper that we understood the cumbersome work that will be involved in the development of this language. Following the first revision of the White Paper there were 12 others until the final White Paper was released and submitted for review.

Our planning model was to keep a common and open line of communications between all members of our team to define the objectives of Crash. In order to achieve this, we kept a log file to keep track of every idea, strategy, difficulty, and changes to our original design. Our directory would include copy of the most recent source files and would track date, time and person making changes to the log.

The planning phase of the project involved several meetings with all team members to define the goals of the project and to assign each other the corresponding responsibilities in order to achieve these goals. Once the specifications of the software were defined, each one of us worked cooperatively to accomplish those objectives and make sure that the integration with the other pieces of the software was seamless. The last

phase of the project was testing and all of the team members were directly or indirectly involved to ensure that the design was made to specification.

## Programming Style Guide

For the code that actually implements the function calls invoked by Crash files, we use the coding style followed the Java coding standard by Sun Microsystems. In other words, we used java's coding conventions to improve the readability of the software to facilitate all people in our team and other programmers to understand new code quickly. This Java language-coding standards can be found in the Java Language Specification from Sun Microsystems website. http://java.sun.com/docs/books/jls/index.html.

In general the following programming guidelines were used for our development environment:

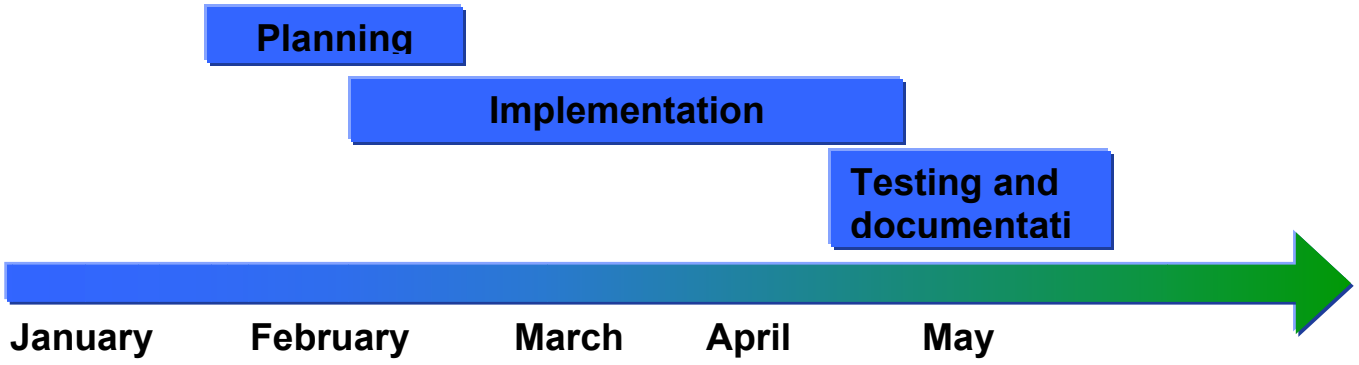
- For Java source files, we used the suffix ".java" in filenames, and Java bytecode we used the suffix ".class"
- The Java source files had a beginning comment that lists the class name, and general information of the class; package and import statements, and Class and Interface declarations.
- We used indentation in Classes, Methods, If Statements, Loops, and whenever an expression did not fit on a single line.
- We used comments in our code to provide implementation and documentation information about a particular expression or part of the code.
- We used get/set methods to access significant members of our Classes that were defined private or protected.
- Our parser and lexer generator used the ".g" file extension and use the coding guidelines defined in www.antlr.org

## Project Timeline

The final project line discussed during the planning phase was defined as following:

| | |
|---|---|
| 01-27-2003 | Project inception with an overview of functionality. |
| 01-31-2003 | First revision of White Paper with main goals defined. |
| 02-17-2003 | Final revision of White Paper with functionality and features defined. |
| 02-18-2003 | White paper due. |
| 02-24-2003 | Roles and responsibilities assigned. Development strategy and standards specified. Collaboration scheme established. |
| 03-21-2003 | First draft of Language Manual Reference. This included a basic architecture of each functional part of the project. |
| 03-26-2003 | Final draft of Language Reference Manual with Crash's functional specification, detailed methods and functional samples |
| 03-27-2003 | Language Reference Manual due. |
| 04-21-2003 | First trials of Lexer and Parser, as well as some functional code generated. |
| 05-01-2003 | Full code integration and testing. |
| 05-05-2003 | Final tests and system architecture refinement |
| 05-12-2003 | Final report and LRM refinements with changes made |
| 05-13-2003 | Presentation and final report due. |

## Estimated Timeline



## Team member's roles

We divided the workload in several parts, and decided that the best way to achieve the project's goals was to assign each part for two team members, so that every member would be responsible for 2 parts at the same time. We decided to take this approach because it seemed that it would facilitate the integration of the project.

| | |
|---|---|
| Vadim Belobrovka /Daniel Burdeinick | Compiler/Software integration |
| Mikhail Litvin/Vadim Belobrovka | Back-End software generation |
| Michael Anikin/Mikhail Litvin | Lexer and Parser |
| Daniel Burdeinick/Michael Anikin | Testing and documentation. |
| Whole Team | Code review and approval |

## Software Development Environment and Tools

All code was first produced in a separate development environment, where internal debugging and testing took place independently. After tested, all pieces were transferred to the CLIC lab servers where integration and testing took place.

The servers that where we installed Crash for refinement and testing, use the Red Hat Linux 2.4.18-3 Operating System with Sun's Java Development Kit version 1.4.1. The parser generator, lexer generator and AST walker for our language were developed using the parser generator ANTLR 2.7.1 for creating lexical and semantic analysis.

The actual functional code was developed using the API offered by OpenGL in Java. We chose to use GNU EMACS for editing and viewing source code. For documentation purposes we used Microsoft Word 2000 in a Microsoft Windows environment.

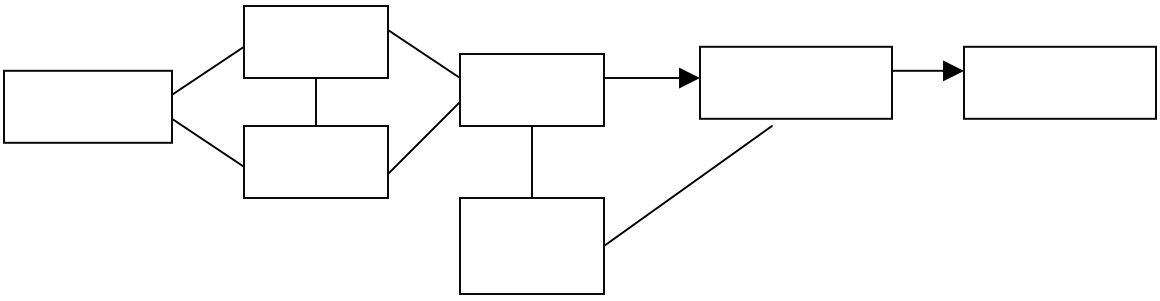
**Project Log**

- January 27: Team met to brainstorm ideas for our language. We decided on a Graphical Animation language and gave it the name "Crash" (Like Flash but inferior). We decided to implement this language using OpenGL API because some of the team members are taking the course Computer Graphics, which would help in the development of our language.
- January 27 to 31: More discussion as far as Crash's functionality and first draft of White Paper with a full introduction.
- February 1 to 17: Entire team reviewed and finalized the White Paper.
- February 18: White paper due in class.
- February 18 to 24: Roles and responsibilities assigned. Initial stages of planning and assessment.
- February 25 to March 21: After several meetings first draft of the Language Reference Manual is complete, with partial functionality defined in the draft.
- March 21 to 26: Final draft of LRM is complete. Team reviews and completes the LRM with detailed information about methods, members and structure information about the language.
- March 27: LRM due.
- March 27 to April 4: Lexer and parser generators built to support a basic functionality of our code using ANTLR as well as basic types with limited scope rules.
- April 3 to 10: First tests with Java code and the OpenGL framework
- April 4 to May1: Initial AST walker construction with while loops, precedence, subroutines, recursion, and procedure call are implemented.
- April 4 to May 1: First trials of integration with file input/output and interpreter.
- May 1 to 13: Final integration, testing and changes done to Crash in order to have a functional language. First draft of final report written.
- May 12: Final report finalized and reviewed by team members.
- May 13: Final report due

# Chapter 5 – Architectural Design

**Block Diagram**

The following block diagram shows Crash's implementation:



Our source file with extension "crh" is converted to a stream of tokens and passed to an Abstract Syntax Tree (AST) twice. The first time a Symbol table is constructed to print any type of semantic or lexical errors. The second time it is actually executed interfaced with the same symbol table. The output generated is a canvas with the animations as instructed in the source file.

**Interfaces**

The Lexer Class scans a given input source "crh" file and creates a token representation of it based on its definitions on the grammar.g file. These tokens are passed to the Parser Class to be parsed. After the Parser finishes parsing all tokens, it creates an AST based on the input.

The AST Walker Class uses this AST and does an initial traversal of the AST, evaluating at each node the proper action based on the node's children, comparing each node with the Symbol Table as reference. Once the AST Walker reaches the root node of

the tree it exits. The Symbol Table Class is used as reference to create variables, constants, members and procedure names as well as assigning them default values and types.

**Implementation assignment**

This language implementation required full cooperation from all of the team members, but specifically we tried to assigned small sections of the code to different people in our team. In particular, Michael Anikin and Michael Litvin worked on the syntax grammar of the language while Vadim and Daniel worked on the Engine side of Crash.

At the same time of development of these modules, testing was done and supervised by different team members at different times. Our main focus was to keep an open line of communications in every aspect of the project so that each one of use knew at some degree the overall status of the project.

# Chapter 6 – Test plan

The testing part of this language was developed in several phases. As in any other language development environment, there is no full testing plan that guarantees an error-free language. For such extensive language as Crash, this characteristic can only come after a long time of third party user input and countless days of careful development. Most of the testing done by our team was designed to find irregularities in our software and the way that our language processes our input file.

Initially each team member working in different parts of the project proposed a set of test scenarios. The test cases tested the parser/lexer, AST walker and the Java engine. These tests were run independently to test each module separately. The basic tests generated, examined variable declarations, iterations, code generation, and several other basic functionalities of Crash.

Closer to the finalization and problem resolution of initial tests, we proceeded to experiment with more complex tests that involved several different arrangements of code. These tests involved examining the final functionality with real examples from an input “crh” file to the expected animation as proposed. There were not automated test suites performed in this project, as time was a limiting factor in our development.

# Chapter 7 – Conclusions

## Lessons learned

This was a very interesting project and there were a number of common subjects that we learned in the development of this language.

Planning is very important and should not be overlooked, especially when all team members do not have a similar schedule to meet or work in the project. We learned how to do teamwork over long distances.

From the development point of view, we discover the real power of ANTLR that even though it took us countless hours to create the parser, lexer and walker, it would have taken 10 times as much generating these from scratch. It was very positive to learn about this software engineering tool and the fact that it can produce source code in different important languages.

In the testing aspect of this project we feel that we underestimated the project timelines and dedicated very little time to full integration testing. This led us to have a semi-incomplete language.

## Advice

The best advice that we can provide is to start early in the project, and allocate more time than thought to testing and implementation. Due to the scope of this project, you need to assign real limits in time to each of the parts that you are creating. A good way to achieve this is to start with a very small implementation of the language, the most basic features and then add functionality as the timeline progresses.

The testing phase is very important and should not be overlooked, because after tests are performed, you will realize which functionality is most advantageous and which is not. Therefore, we strongly suggest that you allocate a couple of weeks for the final testing, because this may cause you to go back and change parts of your original functional code. We also wish you the best of luck ;)

# APPENDIX

```
        /*
         * CrashWalker.java
         * @author  Mikhail Litvin
         */
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
import java.util.*;
import java.math.*;

public class CrashWalker implements CRASHTokenTypes {


   public static final  int INT_TYPE=0;
   public static final  int FLOAT_TYPE=1;
   public static final  int ANIMATOR_TYPE=2;
   public static final  int ELEMENT_TYPE=3;
   public static final  int PATH_TYPE=4;
   public static final  int SCENE_TYPE=5;
   public static final  int VERTEX_TYPE=6;
   public static final  int LAYER_TYPE=7;
   public static final  int COLOR_TYPE=8;


   CommonAST rootNode;
   SymbolTable symbolTable =new SymbolTable(100);
   Scene theScene=new Scene();
   Renderer render;
   /** Creates a new instance of CrashWalker */
   public CrashWalker(String filename) {

      try{
         BufferedReader input = new BufferedReader(new FileReader(filename));
         CRASH_L lexer = new CRASH_L(input);
         CRASH_P parser = new CRASH_P(lexer);
```

```
            //Parse the script
            parser.program();
            rootNode = (CommonAST) parser.getAST();
            ASTFrame frame = new ASTFrame("AST JTree", rootNode);
            frame.setVisible(true);

            //write the header for the java file

            symbolTable.insertCrashToken("Scene",SCENE_TYPE,theScene);
            for(AST
node=rootNode.getFirstChild();node!=null;node=node.getNextSibling()){
               processNode(node);
            }

         }

         catch(Exception e){
            e.printStackTrace();
         }
      }

      /**
       * @param args the command line arguments
       */
      public static void main(String[] args) {
         new CrashWalker("C:\\Documents and Settings\\Misha Litvin\\My
Documents\\download\\MishadaRussian\\test.crh");
      }

      private void processNode(AST currentNode){
         // for(;currentNode!=null;currentNode=currentNode.getNextSibling()){
         if (currentNode!=null){
            int nodetype=currentNode.getType();

            if (nodetype==ASSIGNMENT){
               processAssignment(currentNode);
            }
            else if (nodetype==DECLARATION){
               processDeclaration(currentNode);
            }
            else if (nodetype==IFSTATEMENT){
               processIfStatement(currentNode);
            }
            else if (nodetype==WHILELOOP){
               processWhileLoop(currentNode);
            }
```

```
        else if (nodetype==FUNCCALL){
            processFuncCall(currentNode.getFirstChild());
        }

        else{
            throw new  IllegalArgumentException("Not emplemented:" + nodetype );
        }

    }
    //}
}

private void processDeclaration(AST node){
    AST typeNode=node.getFirstChild();
    AST currentNode=typeNode.getNextSibling();

    String var_type_text=typeNode.getText();
    int var_type=-1; //illegal
    if (var_type_text.equals("int")){
        var_type=INT_TYPE;
    }
    else if (var_type_text.equals("float")){
        var_type=FLOAT_TYPE;
    }

    else if (var_type_text.equals("Animator")){
        var_type=ANIMATOR_TYPE;
    }
    else if (var_type_text.equals("Element")){
        var_type=ELEMENT_TYPE;
    }
    else if (var_type_text.equals("color")){
        var_type=COLOR_TYPE;
    }

    else if (var_type_text.equals("Path")){
        var_type=PATH_TYPE;
    }
    else if (var_type_text.equals("Layer")){
        var_type=LAYER_TYPE;
    }


    if (var_type==-1){
        throw new IllegalArgumentException( "Illegal Type:" +var_type_text );
    }
```

```
      for(;currentNode!=null;currentNode=currentNode.getNextSibling()){
         if (currentNode!=null){
            AST constructor=currentNode.getNextSibling();
            boolean skip=false;
            int arg_count=0;
            Token[] args=new Token[3];

            if (constructor!=null && constructor.getType()==CONSTRUCTOR){

               for(AST
curr_arg=constructor.getFirstChild();curr_arg!=null;curr_arg=curr_arg.getNextSibling())
{

                  if(curr_arg.getType()==FLOAT){
                     args[arg_count]=new Token("somestuff",FLOAT_TYPE,new
Double(Double.parseDouble(curr_arg.getText())));
                  }
                  else if(curr_arg.getType()==ID){
                     Token arg_token=symbolTable.getCrashToken(curr_arg.getText());
                     if (arg_token==null){
                        throw new IllegalArgumentException("Not Declared:" +
node.getText());
                     }
                     args[arg_count]=arg_token;

                  }
                  else{
                     throw new IllegalArgumentException("Unsupported argument type:" +
curr_arg.getType());
                  }
                  arg_count++;
               }
               skip=true;
            }

            String var_name=currentNode.getText();
            if (var_type==INT_TYPE){
               double value=0.0;
               if (skip && arg_count==1 && args[0].getType()==FLOAT_TYPE){
                  value=(int)((Double)args[0].getValue()).floatValue();

               }
```

```
            if                  (!(symbolTable.insertCrashToken(var_name,var_type,new
Double(value)))){
                throw new IllegalArgumentException(var_name + " is already declared");
            }
        }
        else if (var_type==FLOAT_TYPE){
            double value=0.0;
            if (skip && arg_count==1 && args[0].getType()==FLOAT_TYPE){
                value=((Double)args[0].getValue()).floatValue();
            }
            if                  (!(symbolTable.insertCrashToken(var_name,var_type,new
Double(value)))){
                throw new IllegalArgumentException(var_name + " is already declared");
            }
        }
        else if (var_type==LAYER_TYPE){
            double value=0.0;
            if (skip && arg_count==1 && args[0].getType()==FLOAT_TYPE){
                value=((Double)args[0].getValue()).floatValue();
            }
            if                  (!(symbolTable.insertCrashToken(var_name,var_type,new
Double(value)))){
                throw new IllegalArgumentException(var_name + " is already declared");
            }
        }


        else  if (var_type==ANIMATOR_TYPE){
            if (!(symbolTable.insertCrashToken(var_name,var_type,new Animator()))){
                throw new IllegalArgumentException(var_name + " is already declared");
            }
        }

        else  if (var_type==ELEMENT_TYPE){
            float[] colort = new float[]{0,0,0};
            String type=Element.POINTS;
            if (skip && arg_count==2){
                colort=((float[])args[0].getValue());
                int t=(int)((Double)args[1].getValue()).floatValue();

                if (t==1){
                    type= Element.LINES;
                }
                else if(t==2){
                    type= Element.LINE_STRIP;
                }
```

```
        else if(t==3){
            type= Element.LINE_LOOP;
        }
        else if(t==4){
            type= Element.TRIANGLES;
        }
        else if(t==5){
            type=Element.TRIANGLE_STRIP ;
        }
        else if(t==6){
            type= Element.TRIANGLE_FAN ;
        }
        else if(t==7){
            type= Element.QUADS;
        }
        else if(t==8){
            type=Element.QUAD_STRIP;
        }
        else if(t==9){
            type=Element.POLYGON;
        }
        else if(t==10){
            type=Element.ELLIPS ;
        }
    }

    Element  e=new Element(false,type);
    e.setColor(colort);
    if (!(symbolTable.insertCrashToken(var_name,var_type,e))){
        throw new IllegalArgumentException(var_name + " is already declared");
    }
}

else  if (var_type==PATH_TYPE){
    Element e=null;
    double dur=10.0;
    if (skip && arg_count==2){
        e=((Element)args[0].getValue());
        dur=((Double)args[1].getValue()).floatValue();

    }
    Path p=new Path(e);
    p.setDuration((float)dur);
    if (!(symbolTable.insertCrashToken(var_name,var_type,p))){
        throw new IllegalArgumentException(var_name + " is already declared");
    }
```

```
            }

            else  if (var_type==COLOR_TYPE){

               if (skip && arg_count==3){
                  float r=((Double)args[0].getValue()).floatValue();
                  float g=((Double)args[0].getValue()).floatValue();
                  float b=((Double)args[0].getValue()).floatValue();
                  float[] colort = new float[]{r,g,b};

                  if (!(symbolTable.insertCrashToken(var_name,var_type,colort))){
                     throw new IllegalArgumentException(var_name + " is already
declared");
                  }
               }
            }

            else{
               throw new IllegalArgumentException(var_name + " illegal declaration: " +
var_type);
            }

            if (skip){
               currentNode=currentNode.getNextSibling();
            }
         }
      }
   }

   private void processAssignment(AST node){
      AST variable=node.getFirstChild();
      AST expression=variable.getNextSibling();
      String variable_name=variable.getText();

      Token vartoken=symbolTable.getCrashToken(variable_name);
      if (vartoken==null){
         throw new IllegalArgumentException(variable_name + " is not declared");
      }
      /*if (vartoken.getType()!=INT_TYPE){
         throw new IllegalArgumentException("assignment is not supported for this
type");
      }*/

      int expr_type=expression.getType();
      Object expr_value=processExpression(expression);
      if (expr_value==null){
```

```
            throw new IllegalArgumentException("Evaluted to null");

        }

       //      System.out.println("evaluated      expression:"+      vartoken.getName()      +"="
+expr_value);

        if (expr_value instanceof Double ){
            Double d_expr_value=(Double)expr_value;
            if (vartoken.getType()==INT_TYPE){
                vartoken.setValue( new Double((double)(int)d_expr_value.longValue()));
            }
            else if (vartoken.getType()==FLOAT_TYPE){
                vartoken.setValue(d_expr_value);
            }
            else{
                throw new IllegalArgumentException("wrong variable type");
            }
        }
        else            if            (expr_value            instanceof            Animator            &&
vartoken.getType()==ANIMATOR_TYPE){
            vartoken.setValue(expr_value);
        }
        else if (expr_value instanceof Path && vartoken.getType()==PATH_TYPE){
            vartoken.setValue(expr_value);
        }
        else            if            (expr_value            instanceof            Element            &&
vartoken.getType()==ELEMENT_TYPE){
            vartoken.setValue(expr_value);
        }
        else{
            throw new IllegalArgumentException("Incompetible types");
        }
    }

    private Object processFuncCall(AST node){
        int node_type=node.getType();
        if(node_type==ID){
            //get token
            int arg_count=0;
            Token[] args=new Token[3];

            Token token=symbolTable.getCrashToken(node.getText());
            if (token==null){
                throw new IllegalArgumentException("Not Declared: " + node.getText());
            }
```

```
        //get argumnets;
        AST func_name_node=node.getFirstChild();

        AST curr_arg=func_name_node.getNextSibling();
        for(;curr_arg!=null;curr_arg=curr_arg.getNextSibling()){

          if(curr_arg.getType()==FLOAT){
            args[arg_count]=new Token("somestuff",FLOAT_TYPE,new
Double(Double.parseDouble(curr_arg.getText())));
          }
          else if(curr_arg.getType()==ID){
            Token arg_token=symbolTable.getCrashToken(curr_arg.getText());
            if (arg_token==null){
              throw new IllegalArgumentException("Not Declared: " + node.getText());
            }
            args[arg_count]=arg_token;

          }
          else if(curr_arg.getType()==FUNCCALL){
            args[arg_count]=new Token("somestuff",FLOAT_TYPE,
processFuncCall(curr_arg.getFirstChild()));
          }
          else{
            throw new IllegalArgumentException("Incorrect argument type: " +
curr_arg.getType());
          }
          arg_count++;
        }

        String func_name=func_name_node.getText();
        if (token.getType()==LAYER_TYPE){
          Double  layer=(Double)token.getValue();
          if (func_name.equals("setPriority") && arg_count==1){
            token.setValue(args[0].getValue());
            return new Double(1);
          }
          else if (func_name.equals("getPriority") && arg_count==0){
            return (Double)token.getValue() ;
          }

          else{
            throw new IllegalArgumentException("Invalid  function call");
          }

        }
          else if (token.getType()==ELEMENT_TYPE){
```

```
Element  el=(Element)token.getValue();
if (func_name.equals("addVertex") && arg_count==2){
   el.addVertex(new Vertex((int)((Double)(args[0].getValue()))).doubleValue(),
            (int)((Double)(args[0].getValue()))).doubleValue()));
   return new Double(1);
}

else if (func_name.equals("setType") && arg_count==1){

     String type =Element.POINTS;
    int t=(int)((Double)args[0].getValue()).floatValue();

       if (t==1){
          type= Element.LINES;
       }
       else if(t==2){
          type= Element.LINE_STRIP;
       }
       else if(t==3){
          type= Element.LINE_LOOP;
       }
       else if(t==4){
          type= Element.TRIANGLES;
       }
       else if(t==5){
          type=Element.TRIANGLE_STRIP ;
       }
       else if(t==6){
          type= Element.TRIANGLE_FAN ;
       }
       else if(t==7){
          type= Element.QUADS;
       }
       else if(t==8){
          type=Element.QUAD_STRIP;
       }
       else if(t==9){
          type=Element.POLYGON;
       }
       else if(t==10){
          type=Element.ELLIPS ;
       }

   el.setElementType(type);
   return new Double(1);
}
```

```
    else if (func_name.equals("setColor") && arg_count==1){
       el.setColor((float[])(args[0].getValue()));
       return new Double(1);
    }


    else if (func_name.equals("fill") && arg_count==1){
       el.setColor((float[])(args[0].getValue()));
       return new Double(1);
    }


    else{
       throw new IllegalArgumentException("Invalid function call:"+ func_name);
    }

 }
 else if (token.getType()==PATH_TYPE){
    Path  pth=(Path)token.getValue();
    if (func_name.equals("setRotation") && arg_count==3){
       pth.setRotation( (float)((Double)(args[0].getValue())).doubleValue());
       pth.setRotationPoint(

         new Vertex ( (int)((Double)(args[1].getValue())).doubleValue(),
                  (int)((Double)(args[2].getValue())).doubleValue())
         );

       return new Double(1);
    }
    else if (func_name.equals("setScale") && arg_count==2){
       pth.setScaleX( (float)((Double)(args[0].getValue())).doubleValue());
       pth.setScaleY( (float)((Double)(args[1].getValue())).doubleValue());
       return new Double(1);
    }
    else if (func_name.equals("setTrajectory") && arg_count==1){
       pth.setPath( (Element)(args[0].getValue()));
       return new Double(1);
    }
    else if (func_name.equals("setDuration") && arg_count==1){
       pth.setDuration( (float)((Double)(args[0].getValue())).doubleValue());
       return new Double(1);
    }
    else{
        throw new IllegalArgumentException("Invalid function call:"+ func_name);
    }
```

```
        }

        else if (token.getType()==ANIMATOR_TYPE){
           Animator anim=(Animator)token.getValue();

           if (func_name.equals("getStatus") && arg_count==0){
              return new Double(anim.getStatus());
           }
           else if (func_name.equals("setLayer") && arg_count==1){
              anim.setLayer((int)((Double)args[0].getValue()).floatValue());
              return new Double(1);
           }
           else if (func_name.equals("addPath") && arg_count==1){
              anim.addPath((Path)(args[0].getValue()));
              return new Double(1);
           }
           else if (func_name.equals("addElement") && arg_count==1){
              anim.addElement((Element)(args[0].getValue()));
              return new Double(1);
           }
           else if (func_name.equals("setStatus") && arg_count==1){
              anim.setStatus( (int)((Double)(args[0].getValue())).doubleValue());
              return new Double(1);
           }
           else if (func_name.equals("getStatus") && arg_count==0){
              return (new Double(anim.getStatus()));
           }
           else{
               throw new IllegalArgumentException("Invalid function call:"+
func_name);
            }
        }
        else if (token.getType()==SCENE_TYPE){
           if (func_name.equals("setBackgroundColor") && arg_count==1){

              theScene.setBackgroundColor((float[])(args[0].getValue()) );
              return new Double(1);
           }
           else if (func_name.equals("setSize") && arg_count==2){

              theScene.setWidth(  (int)((Double)(args[0].getValue())).doubleValue());
              theScene.setHeight(  (int)((Double)(args[1].getValue())).doubleValue());

              return new Double(1);
           }
           else if (func_name.equals("sleep") && arg_count==1){
```

```
            try{
               Thread.sleep( (int)((Double)(args[0].getValue())).doubleValue()*1000);
            }
            catch (InterruptedException ie){

            }
            return new Double(1);
         }
         else if (func_name.equals("start") && arg_count==0){

            render = (new Renderer(theScene));
            render.start();
            return new Double(1);
         }
         else if (func_name.equals("exit") && arg_count==0){
            render.stop();
            System.exit(0);
            return new Double(1);
         }
         else if ((func_name.equals("print") || func_name.equals("println")) &&
arg_count==1){
            String text="Crash:" +args[0].getValue().toString();
            if(func_name.equals("print")){
               System.out.print(text);
            }
            else{
               System.out.println(text);
            }
            return new Double(1);
         }
         else if (func_name.equals("startAnimator") && arg_count==1){
            Animator temp=  (Animator)(args[0].getValue());
            theScene.addAnimator(temp);
            return new Double(1);
         }

         else if (func_name.equals("random") && arg_count==0){
            return new Double(Math.random());
         }

         else{
             throw new IllegalArgumentException("Invalid function call:"+ func_name);
         }
      }
      else{
```

```
                throw new IllegalArgumentException("Invalid function call:"+ func_name);
            }
        }
        else{
            throw new IllegalArgumentException("Illegal Function Call");
        }
    }

    private Object processExpression(AST node){
        int node_type=node.getType();
        String node_name=node.getText();

        if (node_type==FLOAT){
            return new Double(Double.parseDouble(node.getText()));
        }
        else if (node_type==FUNCCALL){
            return processFuncCall(node.getFirstChild());
        }
        else if (node_type==ID){
            Object symbol=symbolTable.getCrashTokenValue(node_name);
            if(symbol==null){
                throw new IllegalArgumentException("Not Declared: " + node_name);
            }
            else{
                return symbol;
            }
        }
        else if (node_type==More || node_type==Less ||node_type==MoreEQ ||
node_type==LessEQ ||
        node_type==EQ || node_type==NEQ ||
        node_type==Plus || node_type==Star || node_type==Minus || node_type==Divide ||
        node_type==And || node_type==Or  ) {

            Object left=processExpression(node.getFirstChild());
            Object right=processExpression(node.getFirstChild().getNextSibling());


            if ((left instanceof Double) && (right instanceof Double)){
                double l=((Double)left).doubleValue();
                double r=((Double)right).doubleValue();

                if (node_type==More){
                    if (l>r){
                        return new Double(1.0);
                    }
                    else{
```

```
        return new Double(0.0);
    }
}
else if (node_type==Less){
    if (l<r){
        return new Double(1.0);
    }
    else{
        return new Double(0.0);
    }
}

else if (node_type==MoreEQ){
    if (l>=r){
        return new Double(1.0);
    }
    else{
        return new Double(0.0);
    }
}
else if (node_type==LessEQ){
    if (l<=r){
        return new Double(1.0);
    }
    else{
        return new Double(0.0);
    }
}

else if (node_type==EQ){
    if (l==r){
        return new Double(1.0);
    }
    else{
        return new Double(0.0);
    }
}
else if (node_type==NEQ){
    if (l!=r){
        return new Double(1.0);
    }
    else{
        return new Double(0.0);
    }
}
```

```
            else if (node_type==Plus){
               return new Double(l+r);
            }
            else if (node_type==Star){
               return new Double(l*r);
            }
            else if (node_type==Minus){
               return new Double(l-r);
            }
            else if (node_type==Divide){
               if (r==0.0){
                  throw new IllegalArgumentException("Devision by 0");
               }

               return new Double(l/r);
            }
            else if (node_type==And){
               if ( (l!=0.0) && (r!=0.0)){
                  return new Double(1.0);
               }
               else{
                  return new Double(0.0);
               }
            }
            else if (node_type==Or){
               if ( (l!=0.0) || (r!=0.0)){
                  return new Double(1.0);
               }
               else{
                  return new Double(0.0);
               }
            }

         }
         else{
            throw new IllegalArgumentException("Illegal value");
         }
      }

      else{
         throw new IllegalArgumentException("wrong expression:" + node_type);
      }
      return null;
   }

   private void processIfStatement(AST node){
```

```
        AST condition=node.getFirstChild();
        Object result=processExpression(condition);

        if (!(result instanceof Double)){
           throw new IllegalArgumentException("evaluation error");
        }
        double result_value=((Double)result).doubleValue();
        boolean do_stuff=(result_value!=0.0)?true:false;
        for(AST
anode=condition.getNextSibling();anode!=null;anode=anode.getNextSibling()){

           if (anode.getType()==LITERAL_Else){
              do_stuff=!do_stuff;
           }
           else if (do_stuff){

              processNode(anode);
           }
        }
     }

     private void processWhileLoop(AST node){
        AST condition=node.getFirstChild();
        Object result=processExpression(condition);

        if (!(result instanceof Double)){
           throw new IllegalArgumentException("evaluation error");
        }
        double result_value=((Double)result).doubleValue();
        boolean do_stuff=(result_value!=0.0)?true:false;

        if (do_stuff){
           for(AST
anode=condition.getNextSibling();anode!=null;anode=anode.getNextSibling()){
              processNode(anode);
           }
           processWhileLoop(node);
        }
     }
}
```

```
/* Grammar.g file by Michael Anikin*/
class CRASH_P extends Parser;
options {  k = 2;  buildAST = true;  exportVocab = CRASH;  defaultErrorHandler = false; }

tokens {
  PROGRAM;
  IFSTATEMENT;
  WHILELOOP;
//  EXPRESSION;
  DECLARATION;
  ASSIGNMENT;
  FUNCCALL;
  CONSTRUCTOR;
}


program : "StartCRASH"! blockOfCode "EndCRASH"!  { #program = #([PROGRAM, "program"], #program); } ;

blockOfCode : ( statement blockOfCode | controlFlowBlock blockOfCode )* ;

statement : (declaration Separator! | assignment Separator! | funcCall Separator!) ;

controlFlowBlock : ( ifStatement | whileLoop);

ifStatement :  "If"! expression "Then"! blockOfCode ( "Else" blockOfCode )? "EndIf"! { #ifStatement = #([IFSTATEMENT, "ifStatement"], #ifStatement); } ;

whileLoop : "While"! expression "Do"! blockOfCode "EndWhile"! { #whileLoop = #([WHILELOOP, "whileLoop"], #whileLoop); } ;

declaration :  objType ids { #declaration = #([DECLARATION, "declaration"], #declaration); } ;

ids : ID (constructor)? (Comma! ID (constructor)?)* ;
```

```
constructor : LeftP! parameters RightP! { #constructor = #([CONSTRUCTOR, "constructor"], #constructor); } ;

parameters : atom ((Comma!)(atom))* ;

assignment : (ID)(Equal!)(expression) { #assignment = #([ASSIGNMENT, "assignment"], #assignment); } ;

funcCall : ID^ Dot! ID LeftP! (parameters)? RightP! { #funcCall = #([FUNCCALL, "funcCall"], #funcCall); } ;



expression : (term) ((Plus^)(term) | (Minus^)(term))* ; //{ #expression = #([EXPRESSION, "expression"], #expression); } ;

term : (expr) ((Star^)(expr) | (Divide^)(expr))*  ;

expr : (test) ((And^)(test) | (Or^)(test))* ;

test : (exp) ( (EQ^)(exp) | (More^)(exp) | (Less^)(exp) | (NEQ^)(exp) | (MoreEQ^)(exp) | (LessEQ^)(exp))* ;

exp  : (atom)
    | (LeftP!)(expression)(RightP!)
    ;

atom : ( ID | FLOAT | funcCall) ;

objType : ( "int" | "float" | "string" | "color" | "Animator" | "Path" | "Layer" | "Element" | "Label" ) ;



class CRASH_L extends Lexer;
options {
        k = 2;   // set lookahead to 2
        charVocabulary = '\3'..'\377';   // Handle all 8-bit characters
        exportVocab = CRASH;   // Export these token types for tree walkers
        testLiterals = false;   // Disable checking every rule against keywords
      }



//StartCRASH : "StartCRASH" ;
```

```
//EndCRASH : "EndCRASH" ;

//While : "While" ;

//Do : "Do" ;

//EndWhile : "EndWhile" ;

//If : "If" ;

//Then : "Then" ;

//Else : "Else" ;

//EndIf: "EndIf" ;

Separator : ';' ;

EQ : "==" ;

NEQ : "!=" ;

Less : "<" ;

LessEQ : "<=" ;

More : ">" ;

MoreEQ : ">=" ;

Dot : '.' ;

Comma : ',' ;

Plus : '+' ;

Minus : '-' ;

Star : '*' ;

Divide : '/' ;

Equal : '=' ;

And : "&&" ;
```

```
Or : "||" ;

RightP : ')' ;

LeftP : '(' ;

UnderSc : '_' ;

WS :  ( ' ' | '\t' ) { $setType(Token.SKIP); newline();  } ;


EndOfLine : ("\n" | "\r\n" | "\r") { $setType(Token.SKIP); }  ;

Comment  : "/*" ( options {greedy=false;}: // Prevents .* from eating the whole file
        (
         ('\r' '\n') => '\r' '\n' { newline(); }
         | '\r' { newline(); }
         | '\n' { newline(); }
         | ~( '\n' | '\r' )
        )
       )*
        "*/"
       { $setType(Token.SKIP); }  ;

FLOAT:  ('0'..'9')+ (Dot ('0'..'9')+)? ;

ID options { testLiterals = true; } : ( 'a'..'z' | 'A'..'Z' ) ( 'a'..'z' | 'A'..'Z'  | UnderSc | '0'..'9' )* ;
```

```
// $ANTLR 2.7.2: "Crash.g" -> "CRASH_L.java"$

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

public class CRASH_L extends antlr.CharScanner implements CRASHTokenTypes,
TokenStream {
    public CRASH_L(InputStream in) {
        this(new ByteBuffer(in));
    }
    public CRASH_L(Reader in) {
        this(new CharBuffer(in));
    }
    public CRASH_L(InputBuffer ib) {
        this(new LexerSharedInputState(ib));
    }
    public CRASH_L(LexerSharedInputState state) {
```

```
    super(state);
    caseSensitiveLiterals = true;
    setCaseSensitive(true);
    literals = new Hashtable();
    literals.put(new ANTLRHashString("string", this), new Integer(42));
    literals.put(new ANTLRHashString("EndWhile", this), new Integer(20));
    literals.put(new ANTLRHashString("Layer", this), new Integer(46));
    literals.put(new ANTLRHashString("float", this), new Integer(41));
    literals.put(new ANTLRHashString("Then", this), new Integer(15));
    literals.put(new ANTLRHashString("Else", this), new Integer(16));
    literals.put(new ANTLRHashString("Do", this), new Integer(19));
    literals.put(new ANTLRHashString("StartCRASH", this), new Integer(11));
    literals.put(new ANTLRHashString("Animator", this), new Integer(44));
    literals.put(new ANTLRHashString("EndIf", this), new Integer(17));
    literals.put(new ANTLRHashString("EndCRASH", this), new Integer(12));
    literals.put(new ANTLRHashString("Label", this), new Integer(48));
    literals.put(new ANTLRHashString("color", this), new Integer(43));
    literals.put(new ANTLRHashString("Path", this), new Integer(45));
    literals.put(new ANTLRHashString("While", this), new Integer(18));
    literals.put(new ANTLRHashString("int", this), new Integer(40));
    literals.put(new ANTLRHashString("If", this), new Integer(14));
    literals.put(new ANTLRHashString("Element", this), new Integer(47));
}

public Token nextToken() throws TokenStreamException {
    Token theRetToken=null;
    tryAgain:
        for (;;) {
            Token _token = null;
            int _ttype = Token.INVALID_TYPE;
            resetText();
            try {   // for char stream error handling
                try {   // for lexical error handling
                    switch ( LA(1)) {
                        case ';': {
                            mSeparator(true);
                            theRetToken=_returnToken;
                            break;
                        }
                        case '!': {
                            mNEQ(true);
                            theRetToken=_returnToken;
                            break;
                        }
                        case '.': {
                            mDot(true);
```

```
    theRetToken=_returnToken;
    break;
}
case ',': {
    mComma(true);
    theRetToken=_returnToken;
    break;
}
case '+': {
    mPlus(true);
    theRetToken=_returnToken;
    break;
}
case '-': {
    mMinus(true);
    theRetToken=_returnToken;
    break;
}
case '*': {
    mStar(true);
    theRetToken=_returnToken;
    break;
}
case '&': {
    mAnd(true);
    theRetToken=_returnToken;
    break;
}
case '|': {
    mOr(true);
    theRetToken=_returnToken;
    break;
}
case ')': {
    mRightP(true);
    theRetToken=_returnToken;
    break;
}
case '(': {
    mLeftP(true);
    theRetToken=_returnToken;
    break;
}
case '_': {
    mUnderSc(true);
    theRetToken=_returnToken;
```

```
    break;
}
case '\t':  case ' ': {
    mWS(true);
    theRetToken=_returnToken;
    break;
}
case '\n':  case '\r': {
    mEndOfLine(true);
    theRetToken=_returnToken;
    break;
}
case '0':  case '1':  case '2':  case '3':
case '4':  case '5':  case '6':  case '7':
case '8':  case '9': {
    mFLOAT(true);
    theRetToken=_returnToken;
    break;
}
case 'A':  case 'B':  case 'C':  case 'D':
case 'E':  case 'F':  case 'G':  case 'H':
case 'I':  case 'J':  case 'K':  case 'L':
case 'M':  case 'N':  case 'O':  case 'P':
case 'Q':  case 'R':  case 'S':  case 'T':
case 'U':  case 'V':  case 'W':  case 'X':
case 'Y':  case 'Z':  case 'a':  case 'b':
case 'c':  case 'd':  case 'e':  case 'f':
case 'g':  case 'h':  case 'i':  case 'j':
case 'k':  case 'l':  case 'm':  case 'n':
case 'o':  case 'p':  case 'q':  case 'r':
case 's':  case 't':  case 'u':  case 'v':
case 'w':  case 'x':  case 'y':  case 'z': {
    mID(true);
    theRetToken=_returnToken;
    break;
}
default:
    if ((LA(1)=='=') && (LA(2)=='=')) {
        mEQ(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)=='<') && (LA(2)=='=')) {
        mLessEQ(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)=='>') && (LA(2)=='=')) {
```

```
                        mMoreEQ(true);
                        theRetToken=_returnToken;
                    }
                    else if ((LA(1)=='/') && (LA(2)=='*')) {
                        mComment(true);
                        theRetToken=_returnToken;
                    }
                    else if ((LA(1)=='<') && (true)) {
                        mLess(true);
                        theRetToken=_returnToken;
                    }
                    else if ((LA(1)=='>') && (true)) {
                        mMore(true);
                        theRetToken=_returnToken;
                    }
                    else if ((LA(1)=='/') && (true)) {
                        mDivide(true);
                        theRetToken=_returnToken;
                    }
                    else if ((LA(1)=='=') && (true)) {
                        mEqual(true);
                        theRetToken=_returnToken;
                    }
                    else {
                        if (LA(1)==EOF_CHAR) {uponEOF(); _returnToken =
makeToken(Token.EOF_TYPE);}
                        else {throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());}
                    }
                }
                if ( _returnToken==null ) continue tryAgain; // found SKIP token
                _ttype = _returnToken.getType();
                _returnToken.setType(_ttype);
                return _returnToken;
            }
            catch (RecognitionException e) {
                throw new TokenStreamRecognitionException(e);
            }
        }
        catch (CharStreamException cse) {
            if ( cse instanceof CharStreamIOException ) {
                throw new TokenStreamIOException(((CharStreamIOException)cse).io);
            }
            else {
                throw new TokenStreamException(cse.getMessage());
            }
```

```
            }
        }
    }

    public final void mSeparator(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = Separator;
        int _saveIndex;

        match(';');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mEQ(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = EQ;
        int _saveIndex;

        match("==");
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mNEQ(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = NEQ;
        int _saveIndex;

        match("!=");
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }
```

```
    public final void mLess(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = Less;
        int _saveIndex;

        match("<");
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mLessEQ(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = LessEQ;
        int _saveIndex;

        match("<=");
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mMore(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = More;
        int _saveIndex;

        match(">");
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mMoreEQ(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = MoreEQ;
```

```
        int _saveIndex;

        match(">=");
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mDot(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = Dot;
        int _saveIndex;

        match('.');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mComma(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = Comma;
        int _saveIndex;

        match(',');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mPlus(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = Plus;
        int _saveIndex;

        match('+');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
```

```
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
      }
      _returnToken = _token;
    }

    public final void mMinus(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
      int _ttype; Token _token=null; int _begin=text.length();
      _ttype = Minus;
      int _saveIndex;

      match('-');
      if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
      }
      _returnToken = _token;
    }

    public final void mStar(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
      int _ttype; Token _token=null; int _begin=text.length();
      _ttype = Star;
      int _saveIndex;

      match('*');
      if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
      }
      _returnToken = _token;
    }

    public final void mDivide(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
      int _ttype; Token _token=null; int _begin=text.length();
      _ttype = Divide;
      int _saveIndex;

      match('/');
      if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
      }
      _returnToken = _token;
```

```
    }

    public final void mEqual(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = Equal;
        int _saveIndex;

        match('=');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mAnd(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = And;
        int _saveIndex;

        match("&&");
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mOr(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = Or;
        int _saveIndex;

        match("||");
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mRightP(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
```

```
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = RightP;
        int _saveIndex;

        match(')');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mLeftP(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = LeftP;
        int _saveIndex;

        match('(');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mUnderSc(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = UnderSc;
        int _saveIndex;

        match('_');
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mWS(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = WS;
        int _saveIndex;
```

```
        {
            switch ( LA(1)) {
                case ' ': {
                    match(' ');
                    break;
                }
                case '\t': {
                    match('\t');
                    break;
                }
                default: {
                    throw new NoViableAltForCharException((char)LA(1), getFilename(),
getLine(), getColumn());
                }
            }
        }
        if ( inputState.guessing==0 ) {
            _ttype = Token.SKIP; newline();
        }
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mEndOfLine(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = EndOfLine;
        int _saveIndex;

        {
            if ((LA(1)=='\r') && (LA(2)=='\n')) {
                match("\r\n");
            }
            else if ((LA(1)=='\n')) {
                match("\n");
            }
            else if ((LA(1)=='\r') && (true)) {
                match("\r");
            }
            else {
                throw new NoViableAltForCharException((char)LA(1), getFilename(),
getLine(), getColumn());
            }
```

```
        }
        if ( inputState.guessing==0 ) {
            _ttype = Token.SKIP;
        }
        if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
            _token = makeToken(_ttype);
            _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
        }
        _returnToken = _token;
    }

    public final void mComment(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = Comment;
        int _saveIndex;

        match("/*"); {
            _loop108:
                do {
                    // nongreedy exit test
                    if ((LA(1)=='*') && (LA(2)=='/')) break _loop108;
                    if (((LA(1) >= '\u0003' && LA(1) <= '\u00ff')) && ((LA(2) >= '\u0003' &&
LA(2) <= '\u00ff'))) {
                        {
                            boolean synPredMatched106 = false;
                            if (((LA(1)=='\r') && (LA(2)=='\n'))) {
                                int _m106 = mark();
                                synPredMatched106 = true;
                                inputState.guessing++;
                                try {
                                    {
                                        match('\r');
                                        match('\n');
                                    }
                                }
                                catch (RecognitionException pe) {
                                    synPredMatched106 = false;
                                }
                                rewind(_m106);
                                inputState.guessing--;
                            }
                            if ( synPredMatched106 ) {
                                match('\r');
                                match('\n');
```

```
                    if ( inputState.guessing==0 ) {
                        newline();
                    }
                }
                else if ((LA(1)=='\r') && ((LA(2) >= '\u0003' && LA(2) <= '\u00ff'))) {
                    match('\r');
                    if ( inputState.guessing==0 ) {
                        newline();
                    }
                }
                else if ((LA(1)=='\n')) {
                    match('\n');
                    if ( inputState.guessing==0 ) {
                        newline();
                    }
                }
                else if ((_tokenSet_0.member(LA(1)))) {
                    {
                        match(_tokenSet_0);
                    }
                }
                else {
                    throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());
                }

            }
        }
        else {
            break _loop108;
        }

    } while (true);
}
match("*/");
if ( inputState.guessing==0 ) {
    _ttype = Token.SKIP;
}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
    _token = makeToken(_ttype);
    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}
```

```
   public final void mFLOAT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
      int _ttype; Token _token=null; int _begin=text.length();
      _ttype = FLOAT;
      int _saveIndex;

      {
         int _cnt111=0;
         _loop111:
            do {
               if (((LA(1) >= '0' && LA(1) <= '9'))) {
                  matchRange('0','9');
               }
               else {
                  if ( _cnt111>=1 ) { break _loop111; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
               }

               _cnt111++;
            } while (true);
      } {
         if ((LA(1)=='.')) {
            mDot(false); {
               int _cnt114=0;
               _loop114:
                  do {
                     if (((LA(1) >= '0' && LA(1) <= '9'))) {
                        matchRange('0','9');
                     }
                     else {
                        if ( _cnt114>=1 ) { break _loop114; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
                     }

                     _cnt114++;
                  } while (true);
            }
         }
         else {
         }

      }
      if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
         _token = makeToken(_ttype);
         _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
      }
```

```
        _returnToken = _token;
    }

    public final void mID(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
        int _ttype; Token _token=null; int _begin=text.length();
        _ttype = ID;
        int _saveIndex;

        {
            switch ( LA(1)) {
                case 'a':  case 'b':  case 'c':  case 'd':
                case 'e':  case 'f':  case 'g':  case 'h':
                case 'i':  case 'j':  case 'k':  case 'l':
                case 'm':  case 'n':  case 'o':  case 'p':
                case 'q':  case 'r':  case 's':  case 't':
                case 'u':  case 'v':  case 'w':  case 'x':
                case 'y':  case 'z': {
                    matchRange('a','z');
                    break;
                }
                case 'A':  case 'B':  case 'C':  case 'D':
                case 'E':  case 'F':  case 'G':  case 'H':
                case 'I':  case 'J':  case 'K':  case 'L':
                case 'M':  case 'N':  case 'O':  case 'P':
                case 'Q':  case 'R':  case 'S':  case 'T':
                case 'U':  case 'V':  case 'W':  case 'X':
                case 'Y':  case 'Z': {
                    matchRange('A','Z');
                    break;
                }
                default: {
                    throw new NoViableAltForCharException((char)LA(1), getFilename(),
getLine(), getColumn());
                }
            }
        } {
            _loop118:
                do {
                    switch ( LA(1)) {
                        case 'a':  case 'b':  case 'c':  case 'd':
                        case 'e':  case 'f':  case 'g':  case 'h':
                        case 'i':  case 'j':  case 'k':  case 'l':
                        case 'm':  case 'n':  case 'o':  case 'p':
                        case 'q':  case 'r':  case 's':  case 't':
                        case 'u':  case 'v':  case 'w':  case 'x':
```

```
                case 'y':  case 'z': {
                   matchRange('a','z');
                   break;
                }
                case 'A':  case 'B':  case 'C':  case 'D':
                case 'E':  case 'F':  case 'G':  case 'H':
                case 'I':  case 'J':  case 'K':  case 'L':
                case 'M':  case 'N':  case 'O':  case 'P':
                case 'Q':  case 'R':  case 'S':  case 'T':
                case 'U':  case 'V':  case 'W':  case 'X':
                case 'Y':  case 'Z': {
                   matchRange('A','Z');
                   break;
                }
                case '_': {
                   mUnderSc(false);
                   break;
                }
                case '0':  case '1':  case '2':  case '3':
                case '4':  case '5':  case '6':  case '7':
                case '8':  case '9': {
                   matchRange('0','9');
                   break;
                }
                default: {
                   break _loop118;
                }
             }
          } while (true);
   }
   _ttype = testLiteralsTable(_ttype);
   if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
      _token = makeToken(_ttype);
      _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
   }
   _returnToken = _token;
}


private static final long[] mk_tokenSet_0() {
   long[] data = new long[8];
   data[0]=-9224L;
   for (int i = 1; i<=3; i++) { data[i]=-1L; }
   return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
```

```
}
```

```
// $ANTLR 2.7.2: "Crash.g" -> "CRASH_P.java"$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import java.util.Hashtable;
import antlr.ASTFactory;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class CRASH_P extends antlr.LLkParser       implements CRASHTokenTypes {

   protected CRASH_P(TokenBuffer tokenBuf, int k) {
      super(tokenBuf,k);
      tokenNames = _tokenNames;
      buildTokenTypeASTClassMap();
      astFactory = new ASTFactory(getTokenTypeToASTClassMap());
   }

   public CRASH_P(TokenBuffer tokenBuf) {
      this(tokenBuf,2);
   }

   protected CRASH_P(TokenStream lexer, int k) {
      super(lexer,k);
      tokenNames = _tokenNames;
```

```
        buildTokenTypeASTClassMap();
        astFactory = new ASTFactory(getTokenTypeToASTClassMap());
    }

    public CRASH_P(TokenStream lexer) {
        this(lexer,2);
    }

    public CRASH_P(ParserSharedInputState state) {
        super(state,2);
        tokenNames = _tokenNames;
        buildTokenTypeASTClassMap();
        astFactory = new ASTFactory(getTokenTypeToASTClassMap());
    }

    public final void program() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST program_AST = null;

        match(LITERAL_StartCRASH);
        blockOfCode();
        astFactory.addASTChild(currentAST, returnAST);
        match(LITERAL_EndCRASH);
        program_AST = (AST)currentAST.root;
        program_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(PROGRAM,"program")).add(program_AST));
        currentAST.root = program_AST;
        currentAST.child = program_AST!=null &&program_AST.getFirstChild()!=null ?
        program_AST.getFirstChild() : program_AST;
        currentAST.advanceChildToEnd();
        program_AST = (AST)currentAST.root;
        returnAST = program_AST;
    }

    public final void blockOfCode() throws RecognitionException,
TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST blockOfCode_AST = null;

        {
            _loop4:
                do {
```

```
            if ((_tokenSet_0.member(LA(1))) &&
(LA(2)==ID||LA(2)==Equal||LA(2)==Dot)) {
               statement();
               astFactory.addASTChild(currentAST, returnAST);
               blockOfCode();
               astFactory.addASTChild(currentAST, returnAST);
            }
            else if ((LA(1)==LITERAL_If||LA(1)==LITERAL_While) &&
(LA(2)==ID||LA(2)==LeftP||LA(2)==FLOAT)) {
               controlFlowBlock();
               astFactory.addASTChild(currentAST, returnAST);
               blockOfCode();
               astFactory.addASTChild(currentAST, returnAST);
            }
            else {
               break _loop4;
            }

         } while (true);
      }
      blockOfCode_AST = (AST)currentAST.root;
      returnAST = blockOfCode_AST;
   }

   public final void statement() throws RecognitionException, TokenStreamException {

      returnAST = null;
      ASTPair currentAST = new ASTPair();
      AST statement_AST = null;

      {
         if (((LA(1) >= LITERAL_int && LA(1) <= LITERAL_Label))) {
            declaration();
            astFactory.addASTChild(currentAST, returnAST);
            match(Separator);
         }
         else if ((LA(1)==ID) && (LA(2)==Equal)) {
            assignment();
            astFactory.addASTChild(currentAST, returnAST);
            match(Separator);
         }
         else if ((LA(1)==ID) && (LA(2)==Dot)) {
            funcCall();
            astFactory.addASTChild(currentAST, returnAST);
            match(Separator);
         }
```

```
            else {
               throw new NoViableAltException(LT(1), getFilename());
            }

         }
         statement_AST = (AST)currentAST.root;
         returnAST = statement_AST;
      }

      public final void controlFlowBlock() throws RecognitionException,
   TokenStreamException {

         returnAST = null;
         ASTPair currentAST = new ASTPair();
         AST controlFlowBlock_AST = null;

         {
            switch ( LA(1)) {
               case LITERAL_If: {
                  ifStatement();
                  astFactory.addASTChild(currentAST, returnAST);
                  break;
               }
               case LITERAL_While: {
                  whileLoop();
                  astFactory.addASTChild(currentAST, returnAST);
                  break;
               }
               default: {
                  throw new NoViableAltException(LT(1), getFilename());
               }
            }
         }
         controlFlowBlock_AST = (AST)currentAST.root;
         returnAST = controlFlowBlock_AST;
      }

      public final void declaration() throws RecognitionException, TokenStreamException {

         returnAST = null;
         ASTPair currentAST = new ASTPair();
         AST declaration_AST = null;

         objType();
         astFactory.addASTChild(currentAST, returnAST);
         ids();
```

```
      astFactory.addASTChild(currentAST, returnAST);
      declaration_AST = (AST)currentAST.root;
      declaration_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(DECLARATION,"declaration")).add(declaration_A
ST));
      currentAST.root = declaration_AST;
      currentAST.child = declaration_AST!=null
&&declaration_AST.getFirstChild()!=null ?
      declaration_AST.getFirstChild() : declaration_AST;
      currentAST.advanceChildToEnd();
      declaration_AST = (AST)currentAST.root;
      returnAST = declaration_AST;
   }

   public final void assignment() throws RecognitionException, TokenStreamException {

      returnAST = null;
      ASTPair currentAST = new ASTPair();
      AST assignment_AST = null;

      {
         AST tmp6_AST = null;
         tmp6_AST = astFactory.create(LT(1));
         astFactory.addASTChild(currentAST, tmp6_AST);
         match(ID);
      } {
         match(Equal);
      } {
         expression();
         astFactory.addASTChild(currentAST, returnAST);
      }
      assignment_AST = (AST)currentAST.root;
      assignment_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(ASSIGNMENT,"assignment")).add(assignment_AS
T));
      currentAST.root = assignment_AST;
      currentAST.child = assignment_AST!=null
&&assignment_AST.getFirstChild()!=null ?
      assignment_AST.getFirstChild() : assignment_AST;
      currentAST.advanceChildToEnd();
      assignment_AST = (AST)currentAST.root;
      returnAST = assignment_AST;
   }

   public final void funcCall() throws RecognitionException, TokenStreamException {
```

```
      returnAST = null;
      ASTPair currentAST = new ASTPair();
      AST funcCall_AST = null;

      AST tmp8_AST = null;
      tmp8_AST = astFactory.create(LT(1));
      astFactory.makeASTRoot(currentAST, tmp8_AST);
      match(ID);
      match(Dot);
      AST tmp10_AST = null;
      tmp10_AST = astFactory.create(LT(1));
      astFactory.addASTChild(currentAST, tmp10_AST);
      match(ID);
      match(LeftP); {
         switch ( LA(1)) {
            case ID:
            case FLOAT: {
               parameters();
               astFactory.addASTChild(currentAST, returnAST);
               break;
            }
            case RightP: {
               break;
            }
            default: {
               throw new NoViableAltException(LT(1), getFilename());
            }
         }
      }
      match(RightP);
      funcCall_AST = (AST)currentAST.root;
      funcCall_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(FUNCCALL,"funcCall")).add(funcCall_AST));
      currentAST.root = funcCall_AST;
      currentAST.child = funcCall_AST!=null &&funcCall_AST.getFirstChild()!=null ?
      funcCall_AST.getFirstChild() : funcCall_AST;
      currentAST.advanceChildToEnd();
      funcCall_AST = (AST)currentAST.root;
      returnAST = funcCall_AST;
   }

   public final void ifStatement() throws RecognitionException, TokenStreamException {

      returnAST = null;
      ASTPair currentAST = new ASTPair();
      AST ifStatement_AST = null;
```

```
        match(LITERAL_If);
        expression();
        astFactory.addASTChild(currentAST, returnAST);
        match(LITERAL_Then);
        blockOfCode();
        astFactory.addASTChild(currentAST, returnAST); {
           switch ( LA(1)) {
              case LITERAL_Else: {
                 AST tmp15_AST = null;
                 tmp15_AST = astFactory.create(LT(1));
                 astFactory.addASTChild(currentAST, tmp15_AST);
                 match(LITERAL_Else);
                 blockOfCode();
                 astFactory.addASTChild(currentAST, returnAST);
                 break;
              }
              case LITERAL_EndIf: {
                 break;
              }
              default: {
                 throw new NoViableAltException(LT(1), getFilename());
              }
           }
        }
        match(LITERAL_EndIf);
        ifStatement_AST = (AST)currentAST.root;
        ifStatement_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(IFSTATEMENT,"ifStatement")).add(ifStatement_A
ST));
        currentAST.root = ifStatement_AST;
        currentAST.child = ifStatement_AST!=null
&&ifStatement_AST.getFirstChild()!=null ?
        ifStatement_AST.getFirstChild() : ifStatement_AST;
        currentAST.advanceChildToEnd();
        ifStatement_AST = (AST)currentAST.root;
        returnAST = ifStatement_AST;
     }

     public final void whileLoop() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST whileLoop_AST = null;

        match(LITERAL_While);
```

```
        expression();
        astFactory.addASTChild(currentAST, returnAST);
        match(LITERAL_Do);
        blockOfCode();
        astFactory.addASTChild(currentAST, returnAST);
        match(LITERAL_EndWhile);
        whileLoop_AST = (AST)currentAST.root;
        whileLoop_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(WHILELOOP,"whileLoop")).add(whileLoop_AST)
);
        currentAST.root = whileLoop_AST;
        currentAST.child = whileLoop_AST!=null
&&whileLoop_AST.getFirstChild()!=null ?
        whileLoop_AST.getFirstChild() : whileLoop_AST;
        currentAST.advanceChildToEnd();
        whileLoop_AST = (AST)currentAST.root;
        returnAST = whileLoop_AST;
    }

    public final void expression() throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST expression_AST = null;

        {
            term();
            astFactory.addASTChild(currentAST, returnAST);
        } {
            _loop37:
                do {
                    switch ( LA(1)) {
                        case Plus: {
                            {
                                AST tmp20_AST = null;
                                tmp20_AST = astFactory.create(LT(1));
                                astFactory.makeASTRoot(currentAST, tmp20_AST);
                                match(Plus);
                            } {
                                term();
                                astFactory.addASTChild(currentAST, returnAST);
                            }
                            break;
                        }
                        case Minus: {
                            {
```

```
                    AST tmp21_AST = null;
                    tmp21_AST = astFactory.create(LT(1));
                    astFactory.makeASTRoot(currentAST, tmp21_AST);
                    match(Minus);
                } {
                    term();
                    astFactory.addASTChild(currentAST, returnAST);
                }
                break;
            }
            default: {
                break _loop37;
            }
        }
    } while (true);
  }
  expression_AST = (AST)currentAST.root;
  returnAST = expression_AST;
}

public final void objType() throws RecognitionException, TokenStreamException {

  returnAST = null;
  ASTPair currentAST = new ASTPair();
  AST objType_AST = null;

  {
    switch ( LA(1)) {
      case LITERAL_int: {
        AST tmp22_AST = null;
        tmp22_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp22_AST);
        match(LITERAL_int);
        break;
      }
      case LITERAL_float: {
        AST tmp23_AST = null;
        tmp23_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp23_AST);
        match(LITERAL_float);
        break;
      }
      case LITERAL_string: {
        AST tmp24_AST = null;
        tmp24_AST = astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp24_AST);
```

```
    match(LITERAL_string);
    break;
}
case LITERAL_color: {
    AST tmp25_AST = null;
    tmp25_AST = astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp25_AST);
    match(LITERAL_color);
    break;
}
case LITERAL_Animator: {
    AST tmp26_AST = null;
    tmp26_AST = astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp26_AST);
    match(LITERAL_Animator);
    break;
}
case LITERAL_Path: {
    AST tmp27_AST = null;
    tmp27_AST = astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp27_AST);
    match(LITERAL_Path);
    break;
}
case LITERAL_Layer: {
    AST tmp28_AST = null;
    tmp28_AST = astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp28_AST);
    match(LITERAL_Layer);
    break;
}
case LITERAL_Element: {
    AST tmp29_AST = null;
    tmp29_AST = astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp29_AST);
    match(LITERAL_Element);
    break;
}
case LITERAL_Label: {
    AST tmp30_AST = null;
    tmp30_AST = astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp30_AST);
    match(LITERAL_Label);
    break;
}
default: {
```

```
            throw new NoViableAltException(LT(1), getFilename());
         }
      }
   }
   objType_AST = (AST)currentAST.root;
   returnAST = objType_AST;
}

public final void ids() throws RecognitionException, TokenStreamException {

   returnAST = null;
   ASTPair currentAST = new ASTPair();
   AST ids_AST = null;

   AST tmp31_AST = null;
   tmp31_AST = astFactory.create(LT(1));
   astFactory.addASTChild(currentAST, tmp31_AST);
   match(ID); {
      switch ( LA(1)) {
         case LeftP: {
            constructor();
            astFactory.addASTChild(currentAST, returnAST);
            break;
         }
         case Separator:
         case Comma: {
            break;
         }
         default: {
            throw new NoViableAltException(LT(1), getFilename());
         }
      }
   } {
      _loop17:
         do {
            if ((LA(1)==Comma)) {
               match(Comma);
               AST tmp33_AST = null;
               tmp33_AST = astFactory.create(LT(1));
               astFactory.addASTChild(currentAST, tmp33_AST);
               match(ID); {
                  switch ( LA(1)) {
                     case LeftP: {
                        constructor();
                        astFactory.addASTChild(currentAST, returnAST);
                        break;
```

```
                    }
                    case Separator:
                    case Comma: {
                       break;
                    }
                    default: {
                       throw new NoViableAltException(LT(1), getFilename());
                    }
                 }
              }
           }
           else {
              break _loop17;
           }

        } while (true);
     }
     ids_AST = (AST)currentAST.root;
     returnAST = ids_AST;
  }

  public final void constructor() throws RecognitionException, TokenStreamException {

     returnAST = null;
     ASTPair currentAST = new ASTPair();
     AST constructor_AST = null;

     match(LeftP);
     parameters();
     astFactory.addASTChild(currentAST, returnAST);
     match(RightP);
     constructor_AST = (AST)currentAST.root;
     constructor_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(CONSTRUCTOR,"constructor")).add(constructor_
AST));
     currentAST.root = constructor_AST;
     currentAST.child = constructor_AST!=null
&&constructor_AST.getFirstChild()!=null ?
     constructor_AST.getFirstChild() : constructor_AST;
     currentAST.advanceChildToEnd();
     constructor_AST = (AST)currentAST.root;
     returnAST = constructor_AST;
  }

  public final void parameters() throws RecognitionException, TokenStreamException {
```

```
    returnAST = null;
    ASTPair currentAST = new ASTPair();
    AST parameters_AST = null;

    atom();
    astFactory.addASTChild(currentAST, returnAST); {
       _loop23:
          do {
             if ((LA(1)==Comma)) {
                {
                   match(Comma);
                } {
                   atom();
                   astFactory.addASTChild(currentAST, returnAST);
                }
             }
             else {
                break _loop23;
             }

          } while (true);
    }
    parameters_AST = (AST)currentAST.root;
    returnAST = parameters_AST;
}

public final void atom() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    AST atom_AST = null;

    {
       if ((LA(1)==ID) && (_tokenSet_1.member(LA(2)))) {
          AST tmp37_AST = null;
          tmp37_AST = astFactory.create(LT(1));
          astFactory.addASTChild(currentAST, tmp37_AST);
          match(ID);
       }
       else if ((LA(1)==FLOAT)) {
          AST tmp38_AST = null;
          tmp38_AST = astFactory.create(LT(1));
          astFactory.addASTChild(currentAST, tmp38_AST);
          match(FLOAT);
       }
       else if ((LA(1)==ID) && (LA(2)==Dot)) {
```

```
            funcCall();
            astFactory.addASTChild(currentAST, returnAST);
        }
        else {
            throw new NoViableAltException(LT(1), getFilename());
        }

    }
    atom_AST = (AST)currentAST.root;
    returnAST = atom_AST;
}

public final void term() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    AST term_AST = null;

    {
        expr();
        astFactory.addASTChild(currentAST, returnAST);
    } {
        _loop45:
            do {
                switch ( LA(1)) {
                    case Star: {
                        {
                            AST tmp39_AST = null;
                            tmp39_AST = astFactory.create(LT(1));
                            astFactory.makeASTRoot(currentAST, tmp39_AST);
                            match(Star);
                        } {
                            expr();
                            astFactory.addASTChild(currentAST, returnAST);
                        }
                        break;
                    }
                    case Divide: {
                        {
                            AST tmp40_AST = null;
                            tmp40_AST = astFactory.create(LT(1));
                            astFactory.makeASTRoot(currentAST, tmp40_AST);
                            match(Divide);
                        } {
                            expr();
                            astFactory.addASTChild(currentAST, returnAST);
```

```
                }
                break;
            }
            default: {
                break _loop45;
            }
        }
    } while (true);
}
term_AST = (AST)currentAST.root;
returnAST = term_AST;
}

public final void expr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    AST expr_AST = null;

    {
        test();
        astFactory.addASTChild(currentAST, returnAST);
    } {
        _loop53:
            do {
                switch ( LA(1)) {
                    case And: {
                        {
                            AST tmp41_AST = null;
                            tmp41_AST = astFactory.create(LT(1));
                            astFactory.makeASTRoot(currentAST, tmp41_AST);
                            match(And);
                        } {
                            test();
                            astFactory.addASTChild(currentAST, returnAST);
                        }
                        break;
                    }
                    case Or: {
                        {
                            AST tmp42_AST = null;
                            tmp42_AST = astFactory.create(LT(1));
                            astFactory.makeASTRoot(currentAST, tmp42_AST);
                            match(Or);
                        } {
                            test();
```

```
                    astFactory.addASTChild(currentAST, returnAST);
                }
                break;
            }
            default: {
                break _loop53;
            }
        }
    } while (true);
}
expr_AST = (AST)currentAST.root;
returnAST = expr_AST;
}

public final void test() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    AST test_AST = null;

    {
        exp();
        astFactory.addASTChild(currentAST, returnAST);
    } {
        _loop69:
            do {
                switch ( LA(1)) {
                    case EQ: {
                        {
                            AST tmp43_AST = null;
                            tmp43_AST = astFactory.create(LT(1));
                            astFactory.makeASTRoot(currentAST, tmp43_AST);
                            match(EQ);
                        } {
                            exp();
                            astFactory.addASTChild(currentAST, returnAST);
                        }
                        break;
                    }
                    case More: {
                        {
                            AST tmp44_AST = null;
                            tmp44_AST = astFactory.create(LT(1));
                            astFactory.makeASTRoot(currentAST, tmp44_AST);
                            match(More);
                        } {
```

```
            exp();
            astFactory.addASTChild(currentAST, returnAST);
        }
        break;
    }
    case Less: {
        {
            AST tmp45_AST = null;
            tmp45_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp45_AST);
            match(Less);
        } {
            exp();
            astFactory.addASTChild(currentAST, returnAST);
        }
        break;
    }
    case NEQ: {
        {
            AST tmp46_AST = null;
            tmp46_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp46_AST);
            match(NEQ);
        } {
            exp();
            astFactory.addASTChild(currentAST, returnAST);
        }
        break;
    }
    case MoreEQ: {
        {
            AST tmp47_AST = null;
            tmp47_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp47_AST);
            match(MoreEQ);
        } {
            exp();
            astFactory.addASTChild(currentAST, returnAST);
        }
        break;
    }
    case LessEQ: {
        {
            AST tmp48_AST = null;
            tmp48_AST = astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp48_AST);
```

```
                    match(LessEQ);
                } {
                    exp();
                    astFactory.addASTChild(currentAST, returnAST);
                }
                break;
            }
            default: {
                break _loop69;
            }
        }
    } while (true);
}
test_AST = (AST)currentAST.root;
returnAST = test_AST;
}

public final void exp() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    AST exp_AST = null;

    switch ( LA(1)) {
        case ID:
        case FLOAT: {
            {
                atom();
                astFactory.addASTChild(currentAST, returnAST);
            }
            exp_AST = (AST)currentAST.root;
            break;
        }
        case LeftP: {
            {
                match(LeftP);
            } {
                expression();
                astFactory.addASTChild(currentAST, returnAST);
            } {
                match(RightP);
            }
            exp_AST = (AST)currentAST.root;
            break;
        }
        default: {
```

```
            throw new NoViableAltException(LT(1), getFilename());
        }
    }
    returnAST = exp_AST;
}


public static final String[] _tokenNames = {
    "<0>",
    "EOF",
    "<2>",
    "NULL_TREE_LOOKAHEAD",
    "PROGRAM",
    "IFSTATEMENT",
    "WHILELOOP",
    "DECLARATION",
    "ASSIGNMENT",
    "FUNCCALL",
    "CONSTRUCTOR",
    "\"StartCRASH\"",
    "\"EndCRASH\"",
    "Separator",
    "\"If\"",
    "\"Then\"",
    "\"Else\"",
    "\"EndIf\"",
    "\"While\"",
    "\"Do\"",
    "\"EndWhile\"",
    "ID",
    "Comma",
    "LeftP",
    "RightP",
    "Equal",
    "Dot",
    "Plus",
    "Minus",
    "Star",
    "Divide",
    "And",
    "Or",
    "EQ",
    "More",
    "Less",
    "NEQ",
    "MoreEQ",
```

```
        "LessEQ",
        "FLOAT",
        "\"int\"",
        "\"float\"",
        "\"string\"",
        "\"color\"",
        "\"Animator\"",
        "\"Path\"",
        "\"Layer\"",
        "\"Element\"",
        "\"Label\"",
        "UnderSc",
        "WS",
        "EndOfLine",
        "Comment"
    };

    protected void buildTokenTypeASTClassMap() {
        tokenTypeToASTClassMap=null;
    };

    private static final long[] mk_tokenSet_0() {
        long[] data = { 561850443890688L, 0L};
        return data;
    }
    public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
    private static final long[] mk_tokenSet_1() {
        long[] data = { 549643132928L, 0L};
        return data;
    }
    public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());

}









// $ANTLR 2.7.2: "Crash.g" -> "CRASH_L.java"$
```

```
public interface CRASHTokenTypes {
    int EOF = 1;
    int NULL_TREE_LOOKAHEAD = 3;
    int PROGRAM = 4;
    int IFSTATEMENT = 5;
    int WHILELOOP = 6;
    int DECLARATION = 7;
    int ASSIGNMENT = 8;
    int FUNCCALL = 9;
    int CONSTRUCTOR = 10;
    int LITERAL_StartCRASH = 11;
    int LITERAL_EndCRASH = 12;
    int Separator = 13;
    int LITERAL_If = 14;
    int LITERAL_Then = 15;
    int LITERAL_Else = 16;
    int LITERAL_EndIf = 17;
    int LITERAL_While = 18;
    int LITERAL_Do = 19;
    int LITERAL_EndWhile = 20;
    int ID = 21;
    int Comma = 22;
    int LeftP = 23;
    int RightP = 24;
    int Equal = 25;
    int Dot = 26;
    int Plus = 27;
    int Minus = 28;
    int Star = 29;
    int Divide = 30;
    int And = 31;
    int Or = 32;
    int EQ = 33;
    int More = 34;
    int Less = 35;
    int NEQ = 36;
    int MoreEQ = 37;
    int LessEQ = 38;
    int FLOAT = 39;
    int LITERAL_int = 40;
    int LITERAL_float = 41;
    int LITERAL_string = 42;
    int LITERAL_color = 43;
    int LITERAL_Animator = 44;
    int LITERAL_Path = 45;
    int LITERAL_Layer = 46;
```

```
    int LITERAL_Element = 47;
    int LITERAL_Label = 48;
    int UnderSc = 49;
    int WS = 50;
    int EndOfLine = 51;
    int Comment = 52;
}
```

```
package crash;
/**
 * <p>Title: Animation</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Vadim Belobrovka
 * @version 1.0
 */


import gl4java.awt.*;
import java.lang.Math.*;

public class Animation extends GLCanvas  // or GLAnimCanvas for Animation
{
  Scene scene;
        public Animation(Scene scene)
        {
                //This creates a Component with an initial size of 100 by 100.
                //The actual size may change as determined by the Layout Manager.
                super(scene.getWidth(),scene.getHeight());
           this.scene = scene;

        }

        //Any code that is required to execute BEFORE the GL Context is created
        //goes here in preInit().
        public void preInit()
```

```
        {
                //We want double buffering (true by default).
                //doubleBuffer = true;

                //But we dont want stereo view (false by default).
                //stereoView = false;


                //NOTE: Only calls related to GLAnimCanvas are discussed below.  You
cannot
                //use any of these calls with GLCanvas.

                //In some situations you may wish to limit the speed in which your
                //program runs to a certain number of Frames Per Second.

                //setAnimateFps(30.0);

                //Otherwise, turn that option off.

                //setUseFpsSleep(false);


                //Normally, we use the AWT repaint mechanism, however, it is possible to
squeeze out
                //a few more FPS if we paint directly.  Be fore warned however, that it
may be a long
                //time before any other parts of your window get repainted!

                //setUseRepaint(false);

                //If we want to draw as many frames as possible, we would call
setUseFpsSleep(false),
                //and setUseRepaint(false).  But this will still yield to other threads
because
                //normally we don't want to hog the cpu all of the time, just most of the
time ;-)
                //You can turn off this behaviour with setUseYield(false).

                //setUseYield(false);
        }

        //After the GL Context has been created, this function is called.  This is
        //where you set up your one-time ("initial") GL calls.  You can also call
        //start() to start your animation thread here if you extended GLAnimCanvas.
        public void init()
        {
```

```
        gl.glEnable(gl.GL_DEPTH_TEST);

                //This will clear the background color to Black
                gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

                //Reshape the drawing canvas
                reshape(getSize().width, getSize().height);

                //start()   //Start the animation thread (used with GLAnimCanvas).

        }

        //This function is called when the GL Context is being destroyed.  This is
        //where you release any resources you may have created, and/or stop() the
        //animation thread if you extended GLAnimCanvas.
        public void doCleanup()
        {
                //stop()   //Stop the animation thread (used with GLAnimCanvas).
        }

        //When the Windowing system has signaled that the GL Context needs to be
resized,
        //the Context is resized automatically for you, however you must still make
        //certain calls to OpenGL to inform OpenGL of the new size.
        public void reshape(int width, int height)
        {
      //Reset The Current Viewport And Perspective Transformation
                gl.glViewport(0,0,width,height);
                //Select The Projection Matrix
                gl.glMatrixMode(GL_PROJECTION);
                //Reset The Projection Matrix
                gl.glLoadIdentity();
                //Calculate The Aspect Ratio Of The Window
                //glu.gluPerspective(45.0f, (float)getSize().width / (float)getSize().height,
1.0f, 100.0f);
            gl.glOrtho(0,width,0,height,0,-100);
            //glu.gluLookAt(0, 0, 0,     0, 0, 100,     0, 1, 0);              // This determines
where the camera's position and view is
                //Select The Modelview Matrix
                gl.glMatrixMode(GL_MODELVIEW);
        }

        //The display() function is where all of your drawing is done. :-)
        public void display()
        {
       //Ensure GL is initialised correctly
```

```
        if (glj.gljMakeCurrent() == false) return;

        //Clear The Screen And The Depth Buffer
        gl.glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        ////////////////////////////////////////////////////////////////////////////////

gl.glClearColor(scene.getBackgroundColor()[0],scene.getBackgroundColor()[1],scene.ge
tBackgroundColor()[2],0.0f);

        showScene();

        ////////////////////////////////////////////////////////////////////////////////
        showIt();

        //Swap buffers, check for errors, and release the drawing context
        glj.gljSwap();
        glj.gljCheckGL();
        glj.gljFree();

          }

       private void showScene(){
        Animator currentAnimator;
        scene.resetIterator();
        while((currentAnimator = scene.nextAnimator())!= null){
          if(currentAnimator.getStatus() ==
Animator.RUNNING){dealWithRunningAnimator(currentAnimator);}
          if(currentAnimator.getStatus() ==
Animator.PAUSED){dealWithPausedAnimator(currentAnimator);}
        }
       }

       private void dealWithRunningAnimator(Animator currentAnim){
        currentAnim.resetElementIterator();
        firstPivot = currentAnim.nextElement().getThisVertex(0);
        Path currentPath;
        boolean searchThroughPaths=true;
        currentAnim.resetElementIterator();
        currentAnim.resetPathIterator();
        while(searchThroughPaths){
          currentPath = currentAnim.nextPath();
          currentPath.setCurrentTime();
          if(currentPath.getStatus() ==
Path.DID_NOT_START){dealPathDidNotStart(currentPath);}
          if(currentPath.getStatus() == Path.FINISHED){}
```

```
          if(currentPath.getStatus() ==
Path.RUNNING){dealPathRunning(currentPath,currentAnim);}
        }
       }

       Vertex firstPivot;

       private void dealPathRunning(Path currentPath, Animator currentAnim){
         currentPath.setCurrentTime();
         long timeDeltaX = currentPath.getCurrentTimeStampValue() -
currentPath.getTimeFromLastChangeX();;
         long timeDeltaY = currentPath.getCurrentTimeStampValue() -
currentPath.getTimeFromLastChangeY();
         long timeDeltaScaleX = currentPath.getCurrentTimeStampValue() -
currentPath.getTimeFromLastChangeScaleX();
         long timeDeltaScaleY = currentPath.getCurrentTimeStampValue() -
currentPath.getTimeFromLastChangeScaleX();
         long timeDeltaDegree = currentPath.getCurrentTimeStampValue() -
currentPath.getTimeFromLastChangeDegree();

         gl.glPushMatrix();
         gl.glLoadIdentity();


gl.glTranslatef(currentPath.getFromPoint().getX(),currentPath.getFromPoint().getY(),0);
//back to the point on the path
         if(currentPath.getTimePerX() != 0){ //translate X if need
           int xDelta=1;
           if(currentPath.getFromPoint().getX()>currentPath.getToVertex().getX()){xDelta
= -1;}
           if(currentPath.getTimePerX() < timeDeltaX){gl.glTranslatef(xDelta,0,0);}
         }
         if(currentPath.getTimePerY() != 0){ //translate Y if need
           int yDelta=1;
           if(currentPath.getFromPoint().getY()>currentPath.getToVertex().getY()){yDelta
= -1;}
           if(currentPath.getTimePerY() < timeDeltaY){gl.glTranslatef(yDelta,0,0);}
         }
         if(currentPath.getRotation() != 0){ //rotate if needed
           if(currentPath.getTimePerDegree() < timeDeltaDegree){
             currentPath.setRotatedSoFar(currentPath.getRotatedSoFar() + 1);
             gl.glRotatef(currentPath.getRotatedSoFar(),0,0,1);
           }
         }
         if(currentPath.getScaleX() > 1){ //scale x if needed
           if(currentPath.getTimePerScaleX() < timeDeltaScaleX){
```

```
            currentPath.setScaledXSoFar(currentPath.getScaledXSoFar() + 1);
            gl.glScalef(currentPath.getScaledXSoFar(),1,1);
          }
        }
        if(currentPath.getScaleY() > 1){ //scale y if needed
          if(currentPath.getTimePerScaleY() < timeDeltaScaleY){
            currentPath.setScaledYSoFar(currentPath.getScaledYSoFar() + 1);
            gl.glScalef(currentPath.getScaledYSoFar(),1,1);
          }
        }
        gl.glTranslatef(-1*currentPath.getRotationPoint().getX(),-
1*currentPath.getRotationPoint().getY(),0);


        //Draw the actual objects
        drawObjects(currentPath,currentAnim);

        gl.glPopMatrix();

      }

      private void drawObjects(Path currentPath, Animator currentAnim){
        Element currentElement;
        Vertex currentVertex;
        currentAnim.resetElementIterator();

        while((currentElement = currentAnim.nextElement()) != null){
          currentElement.resetIterator();
          gl.glBegin(currentElement.getElementType());
          while((currentVertex = currentElement.nextVertex())!=null){

gl.glVertex3f(currentVertex.getX(),currentVertex.getY(),currentAnim.getLayer());
          }
          gl.glEnd();
        }
      }

      private void dealPathDidNotStart(Path currentPath){
        //initiate to and from points
        currentPath.setFromPoint((currentPath.getPath()).getThisVertex(0));
        currentPath.setFromPoint((currentPath.getPath()).getThisVertex(1));

        //calculate speeds
        calculateTranslationSpeed(currentPath);

currentPath.setTimeFromLastChangeX(currentPath.getCurrentTimeStampValue());
```

```
currentPath.setTimeFromLastChangeY(currentPath.getCurrentTimeStampValue());

currentPath.setTimeFromLastChangeScaleX(currentPath.getCurrentTimeStampValue());

currentPath.setTimeFromLastChangeScaleY(currentPath.getCurrentTimeStampValue());

currentPath.setTimeFromLastChangeDegree(currentPath.getCurrentTimeStampValue());
        if(currentPath.getRotation() !=
0){currentPath.setTimePerDegree((long)((currentPath.getDuration()*1000)/currentPath.g
etRotation()));}
        if(currentPath.getTimePerScaleX() !=
1){currentPath.setTimePerScaleX((long)((currentPath.getDuration()*1000)/currentPath.g
etScaleX()));}
        if(currentPath.getTimePerScaleY() !=
1){currentPath.setTimePerScaleY((long)((currentPath.getDuration()*1000)/currentPath.g
etScaleY()));}

         //mark as Running
         currentPath.setStatus(Path.RUNNING);
       }

       private void calculateTranslationSpeed(Path currentPath){
         currentPath.setTimePerX(0);
         currentPath.setTimePerY(0);
         if(currentPath.getToVertex() == null){return;}
         currentPath.getPath().resetIterator();
         Vertex firstVertex = currentPath.getPath().nextVertex();
         Vertex secondVertex;
         while((secondVertex = currentPath.getPath().nextVertex())!= null){
           currentPath.setTimePerX(currentPath.getTimePerX() +
Math.abs(secondVertex.getX() - firstVertex.getX()));
           currentPath.setTimePerY(currentPath.getTimePerY() +
Math.abs(secondVertex.getY() - firstVertex.getY()));
         }
         if(currentPath.getTimePerX() != 0){

currentPath.setTimePerX((long)(currentPath.getDuration()*1000)/currentPath.getTimePe
rX());
         }
         if(currentPath.getTimePerY() != 0){

currentPath.setTimePerY((long)(currentPath.getDuration()*1000)/currentPath.getTimePe
rY());
         }
       }
```

```
    private void dealWithPausedAnimator(Animator currentAnim){

    }


    private void showIt(){
      //Reset The View
      gl.glLoadIdentity();
      gl.glColor3f(1.0f,0.0f,0.0f);
      gl.glBegin(GL_POLYGON);
            gl.glVertex3f(0.0f,  0.0f, 1.0f);     //Top
            gl.glVertex3f(500.0f, 0.0f, 1.0f);  //Bottom Right
            gl.glVertex3f(0.0f, 500.0f, 1.0f);  //Bottom Left
      gl.glEnd();

      gl.glPushMatrix();
      gl.glLoadIdentity();

      gl.glTranslatef(300.0f,400.0f,0.0f);
      gl.glColor3f(0.0f,1.0f,0.0f);

      gl.glBegin(GL_POLYGON);
            gl.glVertex3f(0.0f,  0.0f, 5.0f);     //Top
            gl.glVertex3f(500.0f, 0.0f, 5.0f);  //Bottom Right
            gl.glVertex3f(500.0f, 500.0f, 5.0f);          //Bottom Left
      gl.glEnd();

      gl.glPopMatrix();


      gl.glColor3f(0.0f,0.0f,1.0f);

      gl.glBegin(GL_POLYGON);
            gl.glVertex3f(0.0f,  0.0f, 10.0f);   //Top
            gl.glVertex3f(500.0f, 0.0f, 10.0f);           //Bottom Right
            gl.glVertex3f(500.0f, 500.0f, 10.0f);       //Bottom Left
            gl.glVertex3f(0.0f, 500.0f, 10.0f);           //Bottom Left
      gl.glEnd();
    }
/*
    public static void main(String[] args) {
      Animation test = new Animation();
      MyFrame frame=new MyFrame("Tutorial Part 1");
           frame.setSize(400,400);
           frame.add(test);
```

```
            frame.show();

        }
      */
}


package crash;

/**
 * <p>Title: Animator.java</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Daniel Burdeinick
 * @version 1.0
 */

import java.util.LinkedList;
import java.util.List;
import java.util.Iterator;

public class Animator {

  public static final int PAUSED = 0;
  public static final int RUNNING = 1;
  public static final int FINISHED = 2;
  public static final int DID_NOT_START = 3;

  private List elementList;
  private List pathList;
  private Iterator elementItr;
  private Iterator pathItr;
  private int status;
  private int layer;
  private long timeStarted;
  //private long timeLeft;
  //private long timePaused;
  //private boolean wasPaused;
  //private boolean didNotStart;

  public Animator() {
    status = DID_NOT_START;
    //didNotStart = true;
    elementList = new LinkedList();
    pathList = new LinkedList();
```

```
  elementItr = elementList.iterator();
  pathItr = pathList.iterator();
  layer = 0; //default value
 }

 //Getters
 public synchronized int getStatus(){return status;}
 public synchronized int getLayer(){return layer;}
 //public synchronized long getTimeStarted(){return timeStarted;}
 //public synchronized long getTimePaused(){return timePaused;}
 //public synchronized long getTimeLeft(){return timeLeft;}
 //public synchronized boolean wasPaused(){return wasPaused;}
 //public synchronized boolean didNotStart(){return didNotStart;}

 //Setters
 public synchronized void setStatus(int animatorStatus){status = animatorStatus;}
 public synchronized void setLayer(int animatorLayer){layer = animatorLayer;}
 public synchronized void setTimeStarted(long time){timeStarted = time;}
 //public synchronized void setTimePaused(long time){timePaused = time;}
 //public synchronized void setTimeLeft(long time){timeLeft = time;}
 //public synchronized void clearWasPaused(){wasPaused = false;}
 //public synchronized void clearDidNotStartFlag(){didNotStart = false;}


 //methods for the Element List
 public synchronized void addElement(Element element){elementList.add(element);}
 public synchronized void resetElementIterator(){elementItr = elementList.iterator();}
 public synchronized Element nextElement(){
  Element nextElement = null;
  if(elementItr.hasNext()){nextElement = (Element)elementItr.next();}
  return nextElement;
 }

 //methods for the Path List
 public synchronized void addPath(Path path){pathList.add(path);}
 public synchronized void resetPathIterator(){pathItr = pathList.iterator();}
 public synchronized Path nextPath(){
  Path nextPath = null;
  if(pathItr.hasNext()){nextPath = (Path)pathItr.next();}
  return nextPath;
 }



}
```

```
package crash;

/**
 * <p>Title: Element.java</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Vadim Belobrovka
 * @version 1.0
 */

import java.util.LinkedList;
import java.util.List;
import java.util.Iterator;

public class Element {

  public static final String POINTS = "GL_POINTS";
  public static final String LINES = "GL_LINES";
  public static final String LINE_STRIP = "GL_LINE_STRIP";
  public static final String LINE_LOOP = "GL_LINE_LOOP";
  public static final String TRIANGLES = "GL_TRIANGLES";
  public static final String TRIANGLE_STRIP = "GL_TRIANGLE_STRIP";
  public static final String TRIANGLE_FAN = "GL_TRIANGLE_FAN";
  public static final String QUADS = "GL_QUADS";
  public static final String QUAD_STRIP = "GL_QUAD_STRIP";
  public static final String POLYGON = "GL_POLYGON";
  public static final String ELLIPS = "ELLIPS";

  private String elementType;
  private List vertexList;
  private boolean isPath;
  private float[] color;
  private Iterator itr;

  public Element(boolean path,String type) {
    isPath = path;
    elementType = null;
    vertexList = new LinkedList();
    itr = vertexList.iterator();
    if(!isPath){elementType = type;}
    color = new float[]{0,0,0};
  }
```

```
  //Setters
  public synchronized void setIsPath(boolean path){isPath = path;}
  public synchronized void setColor(float[] elementColor){color = elementColor;}
  public synchronized void setElementType(String type){elementType = type;}

  //Getters
  public synchronized boolean isPath(){return isPath;}
  public synchronized float[] getColor(){return color;}
  public synchronized String getElementType(){return elementType;}

  //methods for the Vertex List
  public synchronized void addVertex(Vertex vertex){vertexList.add(vertex);}
  public synchronized void resetIterator(){itr = vertexList.iterator();}
  public synchronized Vertex getThisVertex(int index){
    if(index < vertexList.size()){return (Vertex)vertexList.get(index);}else{return null;}
  }
  public synchronized Vertex nextVertex(){
    Vertex nextVertex = null;
    if(itr.hasNext()){nextVertex = (Vertex)itr.next();}
    return nextVertex;
  }




}



package crash;

/**
 * <p>Title: MyFrame.java</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Daniel Burdeinick
 * @version 1.0
 */


import java.awt.*;
import java.awt.event.*;


public class MyFrame extends Frame implements WindowListener {
```

```
//        public int planet=4;
          public MyFrame(String title) {
                    super(title);
                    addWindowListener(this);
          }

          public void windowClosing(WindowEvent e) {
                    System.exit(0);
          }

          public void windowOpened(WindowEvent e) {}

          public void windowClosed(WindowEvent e) {}

          public void windowIconified(WindowEvent e) {}

          public void windowDeiconified(WindowEvent e) {}

          public void windowActivated(WindowEvent e) {}

          public void windowDeactivated(WindowEvent e) {}

   }



package crash;

/**
 * <p>Title: Path.java</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Vadim Belobrovka
 * @version 1.0
 */

import java.util.LinkedList;
import java.util.List;

public class Path {

  public static final int DID_NOT_START = 0;
  public static final int RUNNING = 1;
  public static final int FINISHED = 2;
```

```
private Element pathElement;
private float secondsOfDuration;
private float degreeOfRotation;
private float scaleX;
private float scaleY;
private int status;

private long currentTime;
private long timeFromLastChangeX;
private long timeFromLastChangeY;
private long timeFromLastChangeScaleX;
private long timeFromLastChangeScaleY;
private long timeFromLastChangeDegree;


private long timePerX;
private long timePerY;
private long timePerScaleX;
private long timePerScaleY;
private long timePerDegree;


private int degreesRotated;
private int scaleXScaled;
private int scaleYScaled;
private Vertex rotationPoint;
private Vertex fromPoint;
private Vertex toVertex;

public Path(Element path) {
  scaleX = 1;
  scaleY = 1;
  degreeOfRotation = 0;
  secondsOfDuration = 10;
  degreesRotated = 0;
  scaleXScaled = 1;
  scaleYScaled = 1;
  if(path == null){defaultElement();}
  status = DID_NOT_START;
  currentTime = System.currentTimeMillis();
  rotationPoint = null;
  fromPoint = null;
  toVertex = null;
}

private void defaultElement(){
```

```
    pathElement = new Element(true,null);
    pathElement.addVertex(new Vertex(0,0));
   }

   //Getters
   public synchronized Element getPath(){return pathElement;}
   public synchronized float getDuration(){return secondsOfDuration;}
   public synchronized float getRotation(){return degreeOfRotation;}
   public synchronized float getScaleX(){return scaleX;}
   public synchronized float getScaleY(){return scaleY;}
   public synchronized int getStatus(){return status;}
   public synchronized long getCurrentTimeStampValue(){return currentTime;}
   public synchronized long getTimeFromLastChangeX(){return timeFromLastChangeX;}
   public synchronized long getTimeFromLastChangeY(){return timeFromLastChangeY;}
   public synchronized long getTimeFromLastChangeScaleX(){return
 timeFromLastChangeScaleX;}
   public synchronized long getTimeFromLastChangeScaleY(){return
 timeFromLastChangeScaleY;}
   public synchronized long getTimeFromLastChangeDegree(){return
 timeFromLastChangeDegree;}
   public synchronized long getTimePerX(){return timePerX;}
   public synchronized long getTimePerY(){return timePerY;}
   public synchronized long getTimePerScaleX(){return timePerScaleX;}
   public synchronized long getTimePerScaleY(){return timePerScaleY;}
   public synchronized long getTimePerDegree(){return timePerDegree;}
   public synchronized Vertex getRotationPoint(){return rotationPoint;}
   public synchronized Vertex getFromPoint(){return fromPoint;}
   public synchronized Vertex getToVertex(){return toVertex;}
   public synchronized int getRotatedSoFar(){return degreesRotated;}
   public synchronized int getScaledXSoFar(){return scaleXScaled;}
   public synchronized int getScaledYSoFar(){return scaleYScaled;}

   //Setters
   public synchronized void setPath(Element path){pathElement = path;}
   public synchronized void setDuration(float seconds){secondsOfDuration = seconds;}
   public synchronized void setRotation(float degree){degreeOfRotation = degree;}
   public synchronized void setScaleX(float x){scaleX = x;}
   public synchronized void setScaleY(float y){scaleY = y;}
   public synchronized void setStatus(int animationStatus){status = animationStatus;}
   public synchronized void setCurrentTime(){currentTime =
 System.currentTimeMillis();}
   public synchronized void setTimeFromLastChangeX(long
 time){timeFromLastChangeX = time;}
   public synchronized void setTimeFromLastChangeY(long
 time){timeFromLastChangeY = time;}
```

```
  public synchronized void setTimeFromLastChangeScaleX(long
time){timeFromLastChangeScaleX = time;}
  public synchronized void setTimeFromLastChangeScaleY(long
time){timeFromLastChangeScaleY = time;}
  public synchronized void setTimeFromLastChangeDegree(long
time){timeFromLastChangeDegree = time;}
  public synchronized void setTimePerX(long time){timePerX = time;}
  public synchronized void setTimePerY(long time){timePerY = time;}
  public synchronized void setTimePerScaleX(long time){timePerScaleX = time;}
  public synchronized void setTimePerScaleY(long time){timePerScaleY = time;}
  public synchronized void setTimePerDegree(long time){timePerDegree = time;}
  public synchronized void setRotationPoint(Vertex point){rotationPoint = point;}
  public synchronized void setFromPoint(Vertex point){fromPoint = point;}
  public synchronized void setToVertex(Vertex point){toVertex = point;}
  public synchronized void setRotatedSoFar(int degrees){degreesRotated = degrees;}
  public synchronized void setScaledXSoFar(int scaleX){scaleXScaled = scaleX;}
  public synchronized void setScaledYSoFar(int scaleY){scaleYScaled = scaleY;}
}


package crash;

/**
 * <p>Title: Renderer.java</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Daniel Burdeinick
 * @version 1.0
 */

public class Renderer extends Thread {
  Scene sceneStructure;
  Animation animation;
  public Renderer(Scene scene){
    sceneStructure = scene;
  }

  public void run(){
    animation = new Animation(sceneStructure);
    MyFrame frame=new MyFrame("Crash");
              frame.setSize(sceneStructure.getWidth(),sceneStructure.getHeight());
              frame.add(animation);
              frame.show();

  }
```

```
}


package crash;

/**
 * <p>Title: Scene.java</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Vadim Belobrovka
 * @version 1.0
 */

import java.util.LinkedList;
import java.util.List;
import java.util.Iterator;

public class Scene {
  private int width;
  private int height;
  private float[] color;
  private List animatorList;
  private Iterator itr;
  private long startTime;

  public Scene(){
    width = 300;
    height = 300;
    color = new float[]{1,1,1};
    animatorList = new LinkedList();
    itr = animatorList.iterator();
    startTime = 0;
  }

  public Scene(int sceneWidth, int sceneHeight) {
    width = sceneWidth;
    height = sceneHeight;
    color = new float[]{1,1,1};
    animatorList = new LinkedList();
    itr = animatorList.iterator();
    startTime = 0;
  }

  //Scene starter
  public synchronized void sceneStarter(){startTime = System.currentTimeMillis();}
```

```
  //Setters
  public synchronized void setBackgroundColor(float[] bgColor){color = bgColor;}
  public synchronized void setWidth(int width){this.width = width;}
  public synchronized void setHeight(int height){this.height = height;}

  //Getters
  public synchronized float[] getBackgroundColor(){return color;}
  //public synchronized List getAnimatorList(){return animatorList;}
  public synchronized long getStartTime(){return startTime;}
  public synchronized int getWidth(){return width;}
  public synchronized int getHeight(){return height;}

  //methods for the Animator List
  public synchronized void addAnimator(Animator
animator){animatorList.add(animator);}
  public synchronized void resetIterator(){itr = animatorList.iterator();}
  public synchronized Animator nextAnimator(){
    Animator nextAnimator = null;
    if(itr.hasNext()){nextAnimator = (Animator)itr.next();}
    return nextAnimator;
  }



}


package crash;

/**
 * <p>Title: SymbolTable.java</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Michael Anikin
 * @version 1.0
 */
import java.util.Map;
import java.util.Hashtable;

public class SymbolTable {
  private Map table;

  public SymbolTable(int approximateSize) {
    table = new Hashtable(approximateSize);
```

```
  }

  public synchronized boolean insertCrashToken(String nameOfToken, int typeOfToken,
Object valueOfToken){
    if(table.containsKey(nameOfToken)){return false;}
    table.put(nameOfToken,new Token(nameOfToken,typeOfToken,valueOfToken));
    return true;
  }

  public synchronized boolean modifyCrshToken(String key, int newType, Object
newValue){
    if(!table.containsKey(key)){return false;}
    Token tempTokenHolder;
    tempTokenHolder = (Token) table.get(key);
    tempTokenHolder.setType(newType);
    tempTokenHolder.setValue(newValue);
    return true;
  }

  public synchronized boolean removeCrashToken(String key){
    if(!table.containsKey(key)){return false;}
    table.remove(key);
    return true;
  }

  public synchronized Token getCrashToken(String key){
    Token tempTokenHolder;
    tempTokenHolder = (Token) table.get(key);
    return tempTokenHolder;
  }

  public synchronized Object getCrashTokenValue(String key){
    Token tempTokenHolder;
    if((tempTokenHolder = (Token) table.get(key)) == null){
      return null;
    }else{
      return tempTokenHolder.getValue();
    }
  }

  public synchronized int getCrashTokenType(String key){
    Token tempTokenHolder;
    if((tempTokenHolder = (Token) table.get(key)) == null){
      return -1;
    }else{
      return tempTokenHolder.getType();
```

```
    }
  }


  public static void main(String[] args) {
    SymbolTable symbolTable1 = new SymbolTable(10);
    symbolTable1.insertCrashToken("key",1,"value");
    System.out.println("Token value:
"+(String)symbolTable1.getCrashTokenValue("key"));
    System.out.println("Token value: "+symbolTable1.getCrashTokenType("key"));
    symbolTable1.modifyCrshToken("key",2,"value");
    System.out.println("Token value:
"+(String)symbolTable1.getCrashTokenValue("key"));
    System.out.println("Token value: "+symbolTable1.getCrashTokenType("key"));
    if((symbolTable1.getCrashToken("other"))==null){
      System.out.println("No object");
    }
  }

}


package crash;

/**
 * <p>Title: Tester.java</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Michael Anikin
 * @version 1.0
 */

public class Tester {
  Renderer render;
  Scene scene = new Scene(800,800);
  public Tester() {
    Renderer render = new Renderer(scene);
    render.start();
    float[] color = new float[]{0,0,0};
    scene.setBackgroundColor(color);
  }
  public static void main(String[] args) {
    Tester tester1 = new Tester();
  }
```

```
}


package crash;

/**
 * <p>Title: Token.java</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Michael Anikin
 * @version 1.0
 */

public class Token {
  private String name;
  private int type;
  private Object value;

  public Token(String nameOfToken, int typeOfToken, Object valueOfToken) {
    setName(nameOfToken);
    setType(typeOfToken);
    setValue(valueOfToken);
  }

  //Setters
  public synchronized void setName(String nameOfToken){name = nameOfToken;}
  public synchronized void setType(int typeOfToken){type = typeOfToken;}
  public synchronized void setValue(Object valueOfToken){value = valueOfToken;}

  //Getters
  public synchronized String getName(){return name;}
  public synchronized int getType(){return type;}
  public synchronized Object getValue(){return value;}
}


package crash;

/**
 * <p>Title: Vertex.java</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Vadim Belobrovka
 * @version 1.0
```

```
 */

public class Vertex {

  private int xCoordinate;
  private int yCoordinate;

  public Vertex() {
    xCoordinate = 0;
    yCoordinate = 0;
  }

  public Vertex(int x, int y) {
    xCoordinate = x;
    yCoordinate = y;
  }

  //Getters
  public synchronized int getX(){return xCoordinate;}
  public synchronized int getY(){return yCoordinate;}

  //Setters
  public synchronized void setX(int x){xCoordinate = x;}
  public synchronized void setY(int y){yCoordinate = y;}
}
```