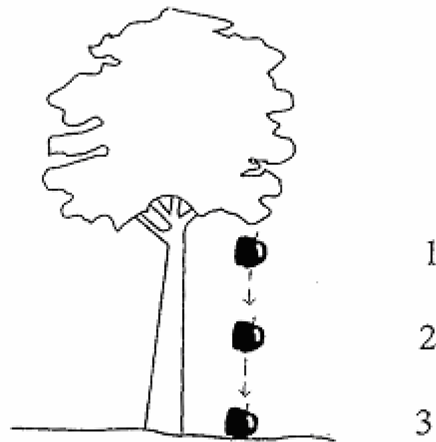
NPSL

Newtonian Physics Simulation Language
Version .5

Glenn Barney
gb2174@columbia.edu

COMS 4115 PLT



http://crede.berkeley.edu/images/rev_timss_gravity1tree.gif

Table Of Conents

1)Introduction 3

2)Tutorial and Example 4

3)LRM 8

4)Architechtual Design 20

5)Project Log and Personal Notes 24

6)Todos 25

7)Appendix A 26

NPSL-2D is a simulation modeling language for 2D Newtonian physics. It is built around the customization of forces that act on Shape (rectangle or circle) objects. A simulation is coded then run and translated to a visual display for the user

Note, that the compiler is essentially in a very early version where functions, type declarations and basic if statements are available. **Most everything else is not.** The framework for everything included in this documentation is in place, but the implementation is not. That is why I'm calling it "version .5". Anything in the document superscipted with an asterix (*) is not implemented yet. There are features of NPSL mentioned below that may not be implemented that are not marked with an asterix.

1) Introduction

1.1) Introduction
Physical simulations of basic Newtonian physics are relatively tedious to run in a typical programming language. Simple physical laws require modeling objects, collision rules and detection, basic geometric arithmetic, time monitoring, and linear equation processing. The goal of NPSL-2D is to provide the constructs to quickly and simply simulate basic 2D physical interactions. Students studying physics, artists wanted to create animations, and scientists/engineers wanting to run simulations for data purposes can use the language for a quick and easy way to model Newtonian mechanics.

2.1) Scope of Physics Model

Abilities and Limitations
NPSL-2D supports the modeling of basic forces represented through vectors acting on object of basic geometric shapes such as rectangles, and circles. That is velocity, acceleration, and force all will have a vector that includes a value and a direction. Each simulation has a timeline, starting at t=0 and simulating until a defined stop time. External forces affecting the entire world, called Forces can be modeled. Forces are totally defined by the suer and can be conditional, set as a global or object event force that is applied under certain rules. For example, air resistance is a global force proportional to an object's up/down (y direction) velocity. Another example: friction is a global force applied to all objects that is dependent on a predefined force (gravity) and a force from object interaction. It acts if there is a gravitational force on an object that is touching a second object and a component of force perpendicular to gravity acting on it from the second object.

Basic elastic and inelastic collisions can be modeled. Inelastic collisions can be modeled through a loss of energy due to kinetic energy transfer proportional to a basic coefficient of restitution based on a property set on each object. Complex forces such as buoyant forces of liquids lift of airplanes, and elastic or spring objects will not be supported . Every object will be treated as unbreakable and intractable.

In order to run a simulation beyond simple global force interaction (say dropping a ball under the force of gravity), note the programmer is able to introduce object event forces into

the system at a given set time or as a reactive response. For example, two identical blocks sitting on a table two lengths apart to collide after an object event force pushes one of them into the other with an initial momentum. Another use of an object event force could be a special "pushing ball" force defined on a unique object that activates on collision. When something comes in contact with the "pushing ball", it releases an additional force besides the natural equal and opposite force on collision.

Interactivity, Events, logging*:
The entire simulation will be non-interactive, the programmer defines the environment, including objects and forces, then runs the simulation. The simulation will display on the screen and also allow for logging. Logging can be set on every object defined, and is output in table format, outputting the time, each logged object's name, location, acting forces, velocity, momentum, and location . The user can set full or partial logging, and the logging interval in seconds, deepening on which of these she wishes to log.

2)Tutoral and Example

Unfortunately there isn't much of a tutorial because there still isn't much of what the user can do yet.

2.1)Hello World

To print out hello world this is what you would do :

```
        string y = "Hello World";
        print (y);
```

To build the code, make sure that you've unzipped everything in the correct directory. From the root, there is a file src, the two grammer files NPSL-2Dlexerparser.g and NPSLWalker.g reside here. Under src, all the java code goes in npsl. Under tests, the test code goes in test. The test data goes in test/test. There is also a .project and .classpath file included for eclipse. You should be able to load the code up from these files in eclipse after changing the root directory of your workspace to match these files.

2.2) Current Code example
Here's and example of what the code can do now :

```
int x = 4;
function int addOne (int x) { x = x+1 };

int y =20;
if (x == 4)
  addOne(y);

Circle  ball{
        mass = 5;
        free = true;
```

```
            radius = 2;
            xCoord = 400;
            yCoord = 300;
}
```

2.3)Future Code example*

Here's an example of how a dependent force hierarchy would be used, including coefficients. Note how much of the interaction is dependent on the user and how they define the force.

```
//assume user defines function to compute the line intersection of two Shapes,
//returning a Line. (Can be a line object with the same end and begin point (a
//point) if  Shape is touching at just a point function Line intersect(Shape shape1, Shape
//shape2) (){…};

//also assume user defines a function that returns an x component of a force
//given a magnitude and direction

world.setBounds(800,600);

Circle  ball{
            mass = 5;
            free = true;
            radius = 2;
            xCoord = 400;
            yCoord = 300;
}


Rect ground {
mass = 100;
free = false;
xCoord= 400;
YCoord = 50;
width = 200;
height = 100;
}

ball.coefFrictionGround =.35;

Force gravity {
            float direction = DOWN;
            float acceleration = 9.8;

            at_tick {
                        foreach_shape {
                        //from v = ½ at^2 + vi*t)
```

```
                shape.yCoord =  shape.yCoord 9.8; // apply gravity
                }
        }
}

Force normal {

        at_tick {
                //going to calculate the force normal, friction, and force down
                //the plane assuming that the touching shape  is fixed
                foreach_shape {
                    foreach_touching_shape {

                        //the normal force is gravity * cos (theta) where theta
                        //is the angle between the resting surface and the x  axis

                        float angle;  //angle from x axis is arctan slope
                        float forceNormalSize;
                        float forceNormalDirection;

                        float slope = intersect(shape, touching).slope;
                        angle = arctan(slope);
                        forceNormalSize = gravity.acceleration * cos(angle);
                        forceNormalDirection = gravity.direction +PI+ angle;

                        //store this value
                        shape.setValue (normal, "normSize",
                        touching, forceNormalSize);
                        shape.setValue (normal, "normDir",
                        touching, forceNormalDirection);

                        //apply the force;
                        shape.xCoord = shape.xCoord + getXPart(forceNormalSize,
                                                        forceNormalDirection);

                        shape.yCoord = shape.yCoord + getYPart(forceNormalSize,
                                                        forceNormalDirection);
                        }
                }
        }
}

Force down_slope {

        at_tick {
                //calculate the force force down assuming touching shape  is fixed
                foreach_shape {
                    foreach_touching_shape {
```

```
                //the down force is gravity * sin (theta) where theta
                //is the angle between the resting surface and the x  axis

                float angle;  //angle from x axis is arctan slope
                float forceDownSize;
                float forceDownDirection;

                float slope = intersect(shape, touching).slope;
                angle = arctan(slope);
                forceDownSize = gravity.acceleration * sin(angle);
                  forceDownDirection = gravity.direction +3*PI/2+ angle;

                //store this value
                shape.downSlopeSizeTouch =forceDownSize;
                shape.downSlopeirTouch = forceDownDirection;

                //apply the force;
                shape.xCoord = shape.xCoord +
                                forceDownSize.xforceDownDirection);

                shape.yCoord = shape.yCoord +
                                forceDownSize.xforceDownDirection);
                }
            }
        }
}

Force friction {

        at_tick {
            //calculate the force friction assuming the touching shape  is fixed
            foreach_shape {
               foreach_touching_shape {

                //the friction force is mu * force normal
                //is the angle between the resting surface and the x  axis

                float angle;  //angle from x axis is arctan slope
                float forceFrictionSize;
                float forceFrictionDirection;

                float slope = intersect(shape, touching).slope;
                angle = arctan(slope);
                forceFrictionSize = gravity.accelration * cos(angle) *
                shape.coefFrictionTouch;
                forceFrictionDirection = gravity.direction +PI/2+ angle;
```

```
                    //store this value
                    shape.frictionizeTouch = forceFrictionSize;
                    shape.frictionDirTouch = forceFrictionDirection;

                    //apply the force;
                    shape.xCoord = shape.xCoord +
                              forceDownSize.xforceDownDirection);

                    shape.yCoord = shape.yCoord +
                              forceDownSize.xforceDownDirection);
                    }
               }
          }
}

simulate;
```

Note with this example above, there is no kinetic energy used. The system could check the velocity of the shape object in the foreach_shape block and then apply a kinetic coefficient of friction vs a static one, it is all up to the language user.

3)LRM

.

3.1) Lexical Conventions

Tokens:
Tokens consist of Identifiers, Keywords, Constants, and Operators.

Whitespace : Spaces (or blanks), all tabs, ('\t') and line terminators (treated as '\r' '\n' for DOS or '\n' for UNIX machines) are considered white space. They exist solely as token separators.

Comments
Multi line comments are supported, and begin with "/*" and end with "*/". Comments can not be nested.

Identifiers
Identifier is a sequence of characters containing any letter or digit, but must start with a letter. Identifiers are case sensitive.

Constants
Constants are either integers (any series of digits), floats (syntax explained later), the boolean value true or false, or string constants (syntax also explained later).

Keywords (explained in the document, capital words are internal types or objects)

int	float	bool	string	false
true	Point	Line	Circle	Rect
Force	at_tick	interval	foreach_shape	foreach_touching_shape
shape	touching	function	World	Logger
void	if	else	return	for
while	break	simulation		

Expressions

Expressions evaluate to some value at the end of their execution. They are generally evaluated from operators but can be as simple as the value true. Here is a list of valid expressions:

lvalue
function-call
unary operator expression (see operator list)
binary operator expression (see operator list)
assignment expression (example : x = 5;)
type definition

Statements and Flow control

A statement controls logic flow and occur in order. A statement list is one or many statements, or nothing:

    statement-list :
        statement; | statement-list | e;

General statements.
-Type declaration and definition :

    type <identifier>; //declaration
    type <identifier> = rval;
        rval : constant
            | expression evaluating to type;

Braces {} are found in the follwing conditions :
- -to mark a begin end block for conditional statements (if, while, etc)
- -to mark a begin end block for a function
- -to mark a begin end block for a foreach section

Braces are not supported for general insertion arbitrarily into code.

Conditional (can be nested) :

```
if (expression) {
        statement-list
}
else { statement-list}
```

While and For statements are not implemented yet, they will have to come in the next version.
While statement*:
```
while ( expression ) {
        statement-list
}
```

For statement*:
```
for ( expression1; expression2; expression3) {

        statement-list;
}
```

A for statement executes first executes expression1. Then it repeatedly executes the statement list followed by expression3 as long as expression2 is true.

while and for also support the break statement, which exits the while or for loop :
```
        break;
```

The return statement returns from a function call :
```
        return;  or return (expression);
```

3.2) Memory Management, Scope, Constructors and Functions

Any constructor C can access any object if it was declared and defined before C's constructor starts. Objects created in NPSL-2D are all of global scope and are only destroyed at the end of the program. The language has no support for multiple threading and therefore objects in the main memory of the program are only accessible by the thread running the simulation. Functions can take objects of both basic and complex types as parameters; parameters will be passed by value. A new function enters a new scope with static scoping.

All points and shapes are treated in units on a Cartesian coordinate system starting with point 0,0 in the bottom right. x direction increases to the right and y direction increases up[1].

Functions are defined as follows:

```
function <return type> <identifier>(<ParamType> <param identifier>, <ParamType>
                                        <param identifier>…. ) {

        statement 1;
        statement 2;
        ….
```

[1] This is in contrast to the original proposal document. I will use a AffineTransform in Java in the translation layer to display pixels with the bottom left being the origin.

```
        statement n;
        return <type>;

}
```

More formally :
function return-type identifier (arg-list) body
arg-list :
    type identifier
    | type identifier, ag-list

body :
    statement-list

Return types can be one of any basic or complex built in type, or void. Parameter types can be one of any basic or complex built in type, including the type Shape. Functions may have side effects as parameters are passed by reference, but no function can pass another function as a parameter; there are no function closures in NPSL-2D.

3.3) Types

Basic types

<data type> :
    int | bool | float | string

Basic types all initialize to default values if declared but not defined. Integers, floats, initialize to 0. Booleans initialize to false, and strings initialize to the empty string "";

Integers – declared with keyword int.
Consist of a series of Digits

Booleans – declared with keyword bool
Consist of either the keyword "false" or "true".

Floats – declared with keyword float
Floats are floating point numbers. They are an optional integer number followed by either a fractional or exponential part, or both, and are computed base 10. Exponent part is "E" or "e" followed by an optionally signed integer. So these are valid floats (in order):

- -Integer, decimal point, fraction, exponent
- -Integer, decimal, fraction
- -Integer, decimal, exponent
- -Integer, decimal
- -Integer, exponent
- -Decimal, fractional
- -Decimal, exponent.

Strings – declared with keyword string
The String type is used to store character strings, and is primarily used for logging purposes and for use as keys when defining shape instance properties (such as coefficients of friction). The default value of a string upon declaration is the empty string, "". A string can be initialized by setting it to a constant, and you can use the backslash character as an escape to include the newline character or quote character in your string.

```
String example; // example is the empty string
example = "test\n";
```

Declarations and Definitions
Variables can be declared with the type and then the identifier :
<type> identifier;
And can be both declared and defined at the same time:
<type> identifier = value;

Value must be a literal of the same type, or an expression that evaluates to that type.

For Primitive data types:
```
int identifier = (<int> | expression evaluating to <int>)  //int x = 4;
float identifier = (<float> | expression evaluating to <float>);  //float y = 3.452e10
boolean identifier = (<bool> | expression evaluating to <bool>); bool isTrue = true;
```

Other legal statements are :
```
    int z = 3 + 2;
    float k = 3 + 3.34;
    bool test = true & false;
```

Built in Complex Types

<complex type> :
Point | Line | Shape | Circle | Rect | Force

The general format for constructing a complex type is :
```
    <type> identifier {
        field1 = defined variable or constant;
        field2 = defined variable or constant;
        ….
        //for each type that is an optional or required for construction
    }
```
And the general form for accessing fields in complex objects is the dot notation:
```
    log(indentifier.field1);
```

Point
A Point is a pair of integers, declared with the keyword Point. A Point must be constructed with two arguments, x and y, both integers.

```
Point <identifier> {
        x = int;
        y = int;
}

pont myPont {
        x = 1;
        y = 2;
}
```

The default value for Points declared but not initialized is 0,0. To access x and y use myPoint.x and myPoint.y respectively.

Line

A Line is simply a set of two Points. To define a Line, create two Point objects and use these to create your Line. Both Points are required. Be careful when using undefined Points in your Line object as they will initialize to 0,0, creating a valid, but perhaps unexpected Line.

```
Line <identifier> {
        point1 = Point;
        point2 = Point;
}
Point firstPoint { x =1; y =2; };
Point secondPoint {x = 2; y =3; };

Line myLine { point1=firstPoint; point2=secondPoint };
```

A Line can represent a Point if both point1 and poin2 have the same x and y values. Line also has the member functions:

float length() which returns a float value of the length of the Line.
float slope() which returns a float value : rise over run. Returns 0 if Line is a point.

Shape Types

There are two basic shape types supported, Circles and Rectangles. The names of these types are Circle and Rect. They both derive from abstract type Shape which cannot be defined or declared. A Circle or Shape can be treated as a Shape through functions or through the automatic iterator keywords foreach_shape and foreach_touching_shape, which are only accessible inside a force, inside a at_tick block.

```
ex : function int comapreShapes(Shape first, Shape second);
```

Shape fields can also be accessed from the Shape object, as described in the “Shape fields” chart.

Shapes

Declare and define a shape of type Circle or Rect with the following syntax. Optional fields below are marked with default. Shapes must be declared and defined at the same time, there

is no declaration only syntax for shapes. If you must declare a shape without any detail, initialize all the parameters to zero and set them later through access. Like any other float to int access, an implicit int to float cast will occur if you set any of these parameters to integers. xCoord and yCoord are the center of the Shape.

Shapes can be accessed in

```
Circle <identifier>{
        mass = float;
        free = bool;  --optional; default to true
        radius = int;
        xCoord = int;
        yCoord = int;
        color = color type; --optional; default to grey
        velocity = float; --optional; default to 0
        direction = float; --optional; default to 0
}

Rect <identifier>{
        mass = float;
        free = bool;  --optional; default to true
        width =  int;
        height =  int;
        xCoord = int;
        yCoord = int;
        xTopLeft = int; -- optional set to xCoord – ½ width
        yTopLeft = int; --optional set to yCoord – ½ height
        xBottomRight = int; -- optional set to xCoord + ½ width
        yBottomRight = int; --optional set to yCoord + ½ height
        color = color type; --optional;default to grey
        velocity = float; --optional; default to 0
        direction = float; --optional; default to 0
}
```

Shape fields
Common to all Shape objects

Name	Type	Purpose	Required / default
mass	float	Weight in “units” (eg kg) of the Shape	required
free	bool	True if not tied to background	def : true
xCoord	int	x coord of center point	required
yCoord	int	y coord of center point	required
color	Color	color of Shape when drawn	def : grey (so sad)
velocity	float	current speed of Shape	def: 0
direction	float	current direction of velocity	def: 0

Specific to Circle

Name	Type	Purpose	Required / default
radius	int	radius of circle	required

Specific to Rect

Name	Type	Purpose	Required / default
width	int	width of rectangle	required
height	int	height of rectangle	reqired
xTopLeft	int	x coord of top left corner	def: xCoord – ½ width
yTopLeft	int	y coord of top left corner	def: yCoord - ½ height
xBottomRight	int	x coord of bottom right corner	def: xCoord + ½ width
yBottomRight	int	y coord of bottom right corner	def: xCoord + ½ height

Accessors for Circle's and Rect's member fields are used with the dot function. For example, assume a Circle name myCircle was declared and defined. Then you can say :

```
int width = myCircle.mass;
myCircle.height = 2.45;
```

Forces

Forces act on a shape at each tick. The foreach keyword is used here, and it is the only time it is used. There are two types of forces, global forces and object forces, but it's really up to the user to implement a force by having the force act at each tick. For example, we can create member variables unique to a Force before the at_tick statement. These variables can be referenced later in other forces. To define a force use the following syntax :

```
Force <identifier> {
        statement 1;  //variable declaration
        statement 2; //variable declaration

        at_tick <int when>? < interval>?[2]{
                statement 1;
                statement 2;
                ….
        }
}
```

There are two util functions for a "force" to break a force up into X and Y component. You can pass it a magnitude and a direction, and this beaks up the vector into a X and Y magnitude (positive if in the +x or +y direction, negative in the –x or –y direction) They return new forces and are :

```
function int getXPart(float magnitude, float direction);
function int getYPart(float magnitude, float direction);
```

[2] ? here means one or zero times, the value when and interval is optional

Note these functions cast the result to an int, they are like a dx and dy on a point.

For example you can call

```
        Force myForce { //initialized} ;
        Force xComponent = myForce.getXpart();
```

The at_tick <int when> < interval> section is the main area where forces are applied, and they can act on two sets of Shapes. The first set of Shapes is the entire set in the world, this set is maintained by the compiler and accessed as shape under the foreach_shape block. The second set is the set of Shapes that the current shape is touching, this is accessed as touching under the foreach_touching_shape block. Any statement that is legal can be executed in the foreach block, and statements can access other forces.

```
foreach_shape {
  shape.y = shape.y - 9.8; (* apply gravity *)
 }
```

Every shape has a list of Shapes it is touching at the current tick. This list can be accessed only through the foreach_touching keyword, which lets you iterate on the touching Shape list. Access to the individual Shape in the loop is accessed through the "touching" keyword. They are used as so

```
        foreach_touching_shape {
                touching.mass = 4;
                float x;
                float y;
                if  (touching instanceof Circle) {
                        print touching.radius;
                }
        }
```

If an integer number is provided after the "at_tick" phrase, then a force will only act at that specific tick number. So for example at_tick 40 will have the force act once and only once at tick 40 (time = 40 seconds). If no integer is provided, then the constant force acts on every tick. Gravity is an example of such a constant force.

If both the when and interval numbers are present then the force acts every "when" ticks. So at_tick 4 interval would mean the force acts every 4 ticks.

Note on internal representation of force application: The language directly supports the mechanism for changing the location of a Shape by having the programmer move the x and y components of the object inside the force action. NPSL-2D runtime will apply all calculations effecting xCoord and yCoord on each tick for every Shape. It will then attempt to move every Shape in the world from its original xCoord and yCoord position to its new xCoord and yCoord position at the same time. If there is a collision then this will be

calculated - the Shape can clearly end up in a different location then the pure forces intended after one tick has passed.

Shape Interaction*

Every Shape can keep track of user defined numeric variables that relate one object to another under a specific force. We call these variables values, but they can be used for any reason. To set a value on a Shape, call the Shape setValue() member function. Pass in the force it relates to, the unique name for this value relative to this Shape, the second object that it is relative to, and finally the float value to set.

```
void setValue (Force f, string name, Shape s, float value);
```

To get a value on a Shape, call getValue(), passing in the same parameters as getValue() for retrieval.

```
float getValue(Force f, string name, Shape s);
```

If you set a unique value (that is a force, name, and shape pair tuple) to a new value, it will overwrite the old value.

We can also store data from one force into a specific coefficient (named so after the coefficient of friction) in a Shape, so that shape can use that force for other force calculations. This function is called as a Shape member function named setCoefficient(). Pass the unique name for this coefficient relative to this Shape, the second object that it is relative to, and finally the float value to set.

```
void setCoefficient (string name, Shape s, float value);
```

To get a value on a Shape, call getCoefficient(), passing in the same parameters as setCoefficient for retrieval.

```
float setCoefficient(string name, Shape s);
```

If you set a unique coefficient (that is a name, and shape pair tuple) to a new coefficient, it will overwrite the old coefficient.

Compile time checking ensures that every Shape be constructed with it's basic elements, and every object force that is referenced in a setCoefficient() call exists. However it does NOT ensure that a coefficient exists and if a getCoefficient() call can not find your value then the code will throw a runtime exception.

Collisions and the Built in Elastic Coefficient.*

In order to support collisions, the runtime will automatically have objects in fully elastic collision by default. This means objects will lose no kinetic energy on collision. In order to have inelastic collisions, the user must make the call :

```
shape.setElasticityCoef(Shape s, float val).
```

This call is commutative; that is

shape1.getElasticityCoef(shape2) == shape2.getElasticityCoef(shape1).

When one call is made both elasticity coefficients are set. The runtime will use these coefficients to effect the dx and dy between ticks representing the loss of kinetic energy in a collision.

3.4) Operators

Mathematical numeric operators act on floats and ints. For binary operators, if a float and int are mixed as parameters, then an int is automatically cast into a float. If an int is stored to a float, it is also implicitly automatically cast to a float. However floats are not cast back into ints for loss of precision. For that reason the statement int x = 3 + 2.3 is illegal. The variable x must be declared as a float.

Unary Mathematical Numeric operators		
Operator	**Function**	**Associativity**
-	Negation	N/A
(	Open Parenthesis – Grouping operator	Left to right relative to close parenthesis
)	Close Parenthesis –grouping operator	Left to right relative to open parenthesis
Binary Mathematical NumericOoperators (operates on float and int)		
+	Addition	Left to right
-	Minus	Left to right
*	Multiplication	Left to right
/	Division	Left to right
Binary Mathematical Relational Operators		
<	Less than	Left to right
>	Greater than	Left to right
=	Assign	Right to left
==, !=	Equal to, not equal to (int only)	Left to right
Binary Boolean Logical Operator		
&	Logical And	Left to right
|	Logical OR	Left to right

Binary Point		
==, !=	Equal to, if both x and y are equal, not equal if either one or both are different	Left to right
Binary Line		
==, !=	Equal to, if both point1 and point2 are equal, not equal if one or both are different	Left to right

Unary String operators		
Operator	**Function**	**Associativity**
[x]	Character substring – returns a substring with the character at index x	N/A
[x, y]	Character substring – returns a substring with the characters starting at index x and ending at index y	N/A
String Comparison (operates on magnitude of force component only)		
<	Alphabetically Less than	Left to right
>	Alphabetically Greater than	Left to right
=	Assign	Right to left
==, !=	Equal to (compares on string contents), not equal on string constants	Left to right
String Access Operators		
+	Concatenation – returns a new string	Left to right
Object (Shape) operators		
instanceof	Run time type checking – returns true of left-hand side same type as right-hand side, otherwise false	Left to right

3.5) Constants

Built in constants:

Name	Type	Value
PI	float	3.14159265
LEFT	float	0
UP	float	PI / 2
RIGHT	float	PI
DOWN	float	3* PI / 2

Type Color, built in constant type.
Color.black - The color black.
Color.blue - The color blue.
Color.cyan - The color cyan.
Color.darkGray - The color dark gray.
Color.gray - The color gray.
Color.green - The color green.
Color.lightGray - The color light gray.
Color.magenta- The color magenta.
Color.orange - The color orange.
Color.pink - The color pink.
Color.red - The color red.

Color.white - The color white.
Color.yellow - The color yellow.

Built in functions
Math basic built in functions exist :
function float cos (float radian) ; //returns cosine of radian
function float sin (float radian) ; //returns sin of radian
function float tan Funciton (float radian); //returns tangent of radian

function float arcsin (float radian); //retunrs arcsin of radian
function float arccos (float radian); //retunrs arccos of radian
function float arctan (float radian); //retunrs arctan of radian

Special static world objects*:
World – Sets the view of the world for the user and manipulates the speed of the simulations. Functions available on it :

setBounds(int x, int y) : This function must be called sometime before the end of the program, and before the simulate function(). It set's the viewable screen size in units. The x and y size is dependent on the size of the objects you define. A good way to think of the parameters to this function is each unit is a meter, and objects under gravity accelerate at 9.8 meters per second. However this is just one use case, it is up to user to define his objects in units relative to the bounds.

simulate() : This function is optional, as it must be called at the last line of the program. If you don't write it the compiler will insert it for you.

speedUp(float speed) : Speeds up the simulation by the float factor speed. As a default, one tick happens every second. For example, passing 4 to speedUp causes four ticks to pass every second.

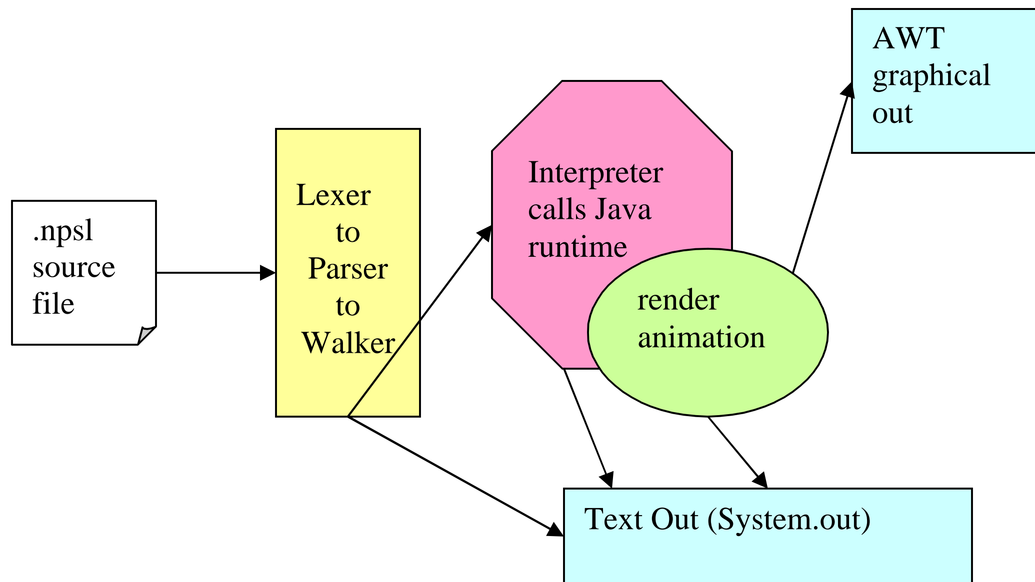
slowDown(float speed) : Slows the simulation by the float factor speed. As a default, one tick happens every second. For example, passing 4 to slowDown causes one tick to pass every four seconds.

logging – use the print or println functions to log

```
int test = 4;
print(test); //prints 4 to std out
```

4) Architectural Design

NPSL is an interpreted language, which generates Java code and executes it, rendering an animation in a JAVA awt screen



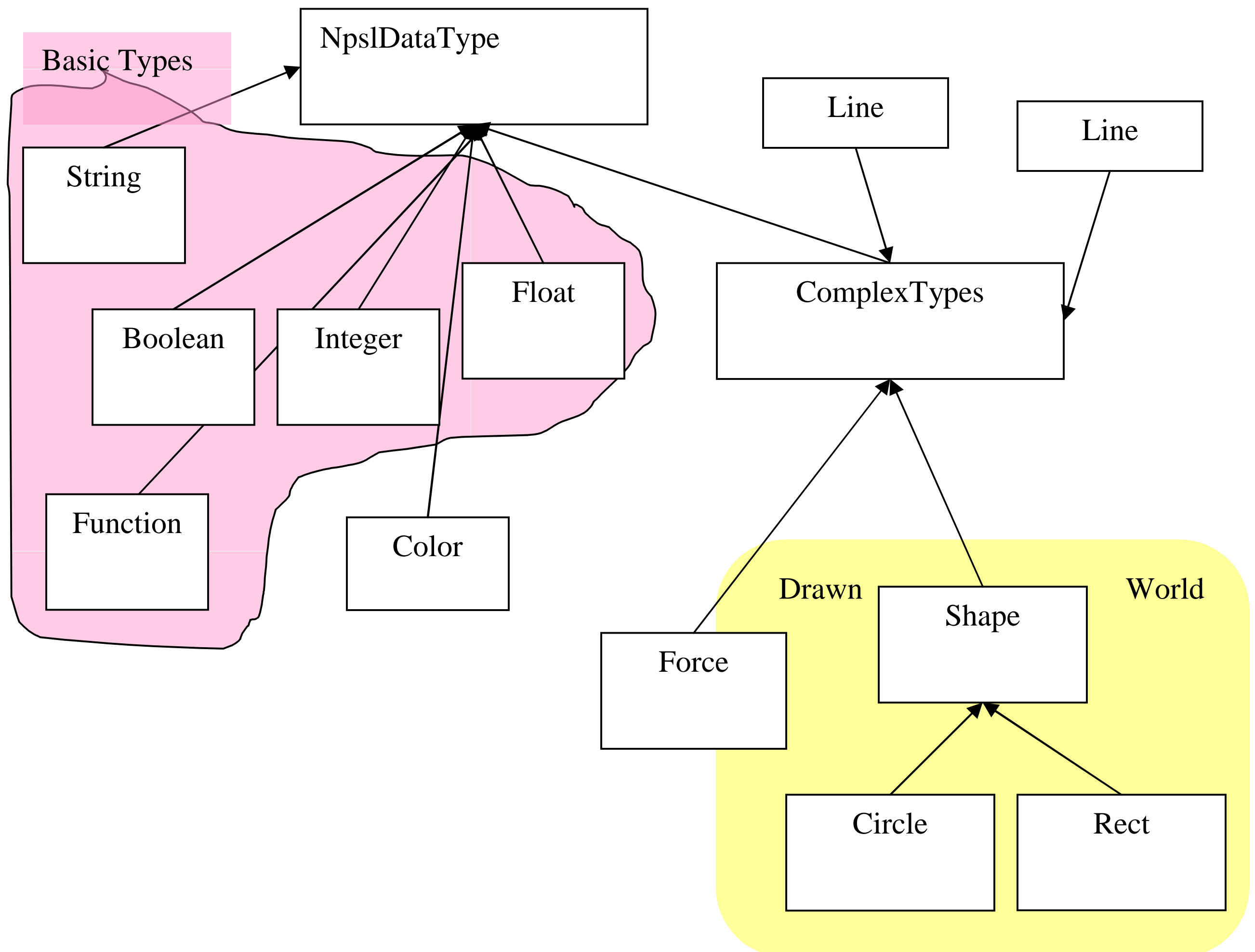
4.1)Main sections of the code

There are two main sections of NPSL, one is the drawing and simulating system, and one is the basic type and logic system. The basic type and logic system is mostly done as of this writing, lacking only a few logic constructs (while and for loops), but is full functioned arithmetically and boolean logically. These statements and expressions are used to create a world of complex objects, of visual and interactive shapes that are defined based on the more basic objects. This drawing and simulating section of NPSL is still in progress. Two intermediary types are the Point and the Line, they do not display on the screen but can be used mathematically to define the characteristics of the Shapes and Forces that control the animation. The Force type sets up the rules that control the simulation. Finally, there is a set of animation classes that interact with the user created Complex objects and render them on the screen. Currently, there is an outline of how this will be implemented, with a screen panel and a thread that will execute an animation based on objects coded by the user.

Actual changes to objects will be done with the ObjectManager, controlled by the logic set up by the user executed in the ForceManager*. Both objects will have a reference to the symbol table. For each tick, ForceManager will iterate through every force, both global and local as defined by the user, and call ObjectManager functions to both gain the information about objects at the current point in time, and to change their member variables (such as position) at any point in time. There is a Moveable interface that will be defined for moveable objects (as set with the free parameter on construction). The managers will use the isMoveable() function to apply Forces on certain objects.

Rendering is done with a single main panel called MainPanel which extends JPanel, a java class for displaying awt objects. Each shape has a BaseComponent object which will hook into the MainPanel for rendering. The changes to the internal NpslShape object will propagate down to the component for rendering.

4.2)Type System



NPSL is dynamically typed, but can use typed keywords to help write safer programs. If a variable is declared as one type, but then treated as another, the interpreter will display a warning message to the user. Sometimes, the program can continue with just a warning message, say if you wrote :

int foo = "hi";

The compiler will warn you, but then store "hi" into foo anyway and then change foo into a string. Sometimes a type def warning will precede an error, such as trying to pass a string variable into a Cirle's name parameter that requires a string. When a float is used where an integer is expected, NPSL will cast the float into an integer. When an integer is used where a float is expected, NPSL will cast the float into an int.

4.3) Object Idenity

For each of the types that the system has (complex and basic), a object has one unique name for the scope that it lives in. The name value is set on construction when creating an anonymous variable, as you name that variable. Literals such as 4 and "the" are constructed

and used in place and not stored anywhere, they are for all intents and purposed discarded after use. Functions are a basic type in NPSL, but NPSL is not functional and functions can not take themselves as arguments to other functions. Internal functions are defined in a special static symbol table outside of the main scope accessible to any layer of code. These functions are defined statically based on their signature so that they can be uniquely overloaded by parameter arguments. This feature is not supported in user defined functions; they are defined solely by the name of the function.

4.4)Data Storage

Objects in NPSL are passed by value. That is, when entering a new function scope or when calling an assignment operator on one variable to another, a new instance of the right hand side or parameter variable is created, and this value is stored in the left hand variable. For this reason, NPSL supports assignment operators on the base type only. That is the complex types, such as Points and Circles, exist on the 2 dimensional plane and can never be copied. They can, however, have each of their descriptive member variables (such as center point or color) changed using the "." operator* to access and modify their values. The only time that complex types are allowed to be constructed then is with the type declaration statement.

4.5)Error Handling

Errors are handled in NPSL by throwing a NpslException, which extends RuntimeException. It halts the program entirely and prints the error to the user. This is used for both illegal use runtime errors, such as attempting to use an assignment operator on a complex type, and an internal NPSL errors, such as the interpreter not finding a function that you attempt to call. Warnings in NPSL are simply handled with a println to the screen.

Predefined Functions

Currently there are only two predefined functions. There is a plan for more, such as mathematically functions like sine and cosine, power, etc.

-print (arg 1, arg2, …arg x). Prints out each argument in the list
-println(arg 1, arg2, …arg x). Prints out each argument in the list with a new line

-simulate; More a command then a function, this renders the simulation/animation that the user has setup. Running this command currently doesn't do anything.

4.6)Testing

Testing is done through three main types of junit tests. The first two use the antlr-testing package found at http://antlr-testing.sourceforge.net/, and are used to test the Lexer and the Parser. The third is a main test used for running a program, ie testing the Walker. Integration testing the functionality of drawing the objects after they have been fully defined and animated through force rules has not been done, although is planned for the future. The first type of tests, the lexer tests, take a string input and a grammer rule and attempt to tokenize the input according tot the grammar rule. If it can, then the assert passes, if not then it fails. The second type of test, the Parser test, takes a parser rule and a string, similar to the lexer tests. The preorder function generates an AST based on the start rule, and

outputs that AST in text format. I compare the expected text output of a preorder traversed AST, using parenthesis to demarcate between nodes. If they match then the assert passes.

The third test type uses NpslMain, and is an overall test that I use interactively to run a full .npsl program. It's basically a launcher for the main method in NpslMain to call a specific passed npsl program, output to the screen, and assert that the program did or did not have an error.

5) Project Log And Personal Notes

5.1)Log

Sept 25th	Proposal
Sep 29th	Install Eclipse, read Antlr tutorials
October 4	Install antlr-testing begin basic lexing
October 9	Lexing of basic types complete
October 13	antlr-testing of basic parsing
October 15	Design basic proof of concept for the high level animation
Ocbober 18	LRM complete
Ocbober 19	Parsing and lexing of basic types complete
Oct 22	Begin statements and expression
Oct 27-28th	Parsing and lexing of basic statements and expressions complete
Nov 3-4th	Study and test basic walker functionality
Nov 9-10th	Functions declaration
Nov 16-17th	Tweak typing system, function walking
Nov 23rd	Internal functions, Color constant class
Dec 2nd	If else control testing, Complex type declaration
Dec 14th-16th	Tweaking assignment functionality Code basic drawing functionality, try to tie it into a running thread.

5.2) Lessons Learned

Seeing as I did not complete the project, there are many lessons I have learned. First I will talk about technically and second about work schedule. Technically, I think I designed the type structure poorly. I took a lot of the type structure from Mx because of it's elegance of

dynamic and run time typing. However, this certainly did not translate to the strict typing that I had defined in my LRM, and the static nature of complex object types. At first I thought there would be no problem, like perl I would be able to define types with type names, but that would be mostly a "suggestion" to the compiler and user, where warnings would be the worst thing that would happen. What this resulted in was a lot of code checking at run time for the dynamic type of the variable being used. If (instanceof NpslComplexType) was true then I did one thing and if it was false I did another. Another bad design decision was while supporting dynamic types, every time an assign operator was used, a copy constructor is called to create a new object with the same value as the object on the right hand side. This is not a bad design decision by itself, but when I wanted to pass default types around from the walker to the interpreter functions, I ended up creating new objects with intial default values twice before finally having them overwritten with a new object that was yet another coy of the existing source object. Memory management in NPSL is therefore very inneficient

Secondly, I think I handled function definition a bit poorly. I tried to create a function signature for each function in order to uniquely identify it (like Java does) so that every defined function can be overloaded. This worked fine with the internal functions, but didn't so well with user defined functions as it would have taken a lot more parsing and interpreter logic to determine these signatures from the user input rather then the explicit coding of internal functions.

In terms of work schedule, I had less time to work on this class then I thought, and I severely underestimated the amount of time it would take to work on the project. There is a large learning curve figuring out how exactly the lexer and parser follow their rules. For example, just matching a constant string is a bit tricky at first, you have to realize how to define it in the parser, not the lexer to avoid conflicts with other tokens. Then you have to add this constant to a grammar rule that can be derived from the main code. Most of the hurdles were probably very similar to the ones other students had to face, but I underestimated the amount of time it took to get comfortable with defining lexing, parsing, and walker rules that actually do things like "pass the argument list to the function".

I can say that my first ideas for this project were very ambitious, but I had planned on at least getting a basic gravity animation supported. I feel that I have a lot of the framework in place, and I'm not too far away, perhaps a week's worth of work.

6) Todos

Here is a list of what I still need to do for this project :

-accessor dot operator (dot notation) for complex types for getting and setting
-varioouis internal functions (such as sine, power), no constants (UP, DOWN, PI…)
-Arrays of basic types (not so useful for my program, but more so for the building blocks users should be able to define forces)
-General shortcuts for the user:
  -Ability to declare more then one variable of a type on one line
  - Self assignment arithmetic operators such as +=, *=, etc
-Control flow : for loops and while loops
-The entire at_tick functionality, ie code to apply forces

-Animation and time code to have a tick pass and draw the screen
-Hit testing for shapes

7) Appendix A : Language Syntax

Syntax currently implemented as of this writing

A.1 Lexical Rules

letter -> 'a'..'z' | 'A'..'Z'
digit -> '0'..'9'
exp -> ('e' | 'E') ('+' | '-')? int
int_float -> (DIGIT)+ | ('.' (DIGIT)* (EXP)? | EXP)
                     | '.' (DIGIT)+ (EXP)?

```
So therefore :
INT -> (DIGIT)+
FLOAT ->    (('.' (DIGIT)* (EXP)? | EXP) | '.' (DIGIT)+ (EXP)?
```

id -> (letter | '_') (letter | digit | '_')*
esc -> '\\' ('n' | 't' | '"' | '\\')
string -> '"' (esc | ~('"' | '\\' | '\n' | '\r'))* '"'
newline -> '\r\n' | '\r' | '\n'
whitespace -> (' ' | '\t')+
comment -> '/*' (~'*/')* '*/' | '//' (~newline)* newline

A.2 Syntatic Rules

```
type -> base_type | complex_type

base_type -> int | float |  bool | void | string

complex_type -> Line | Circle | Rect | Point | Force

program -> ( stmt | func_def )* EOF

declaration -> base_type ID (= expression)? ;

declaration_compl -> complex_type ID {( ID = expression;)* }
```

Statements

```
//main statment list
stmt -> declaration | declaration_compl | if_stmt | assign_stmt
      | func_call_stmt | return_stmt | simulate

if_stmt -> if ( expression ) stmt ({greedy = true;}: else stmt)?

assign_stmt -> l_value = expression ;

func_call_stmt -> func_call ;

func_call -> ID( expr_list )

expr_list -> expression (, expression )*

func_def -> function type ID ( var_list ) func_body
```

```
var_list -> type ID (, type ID )*

func_body -> = expression ; ( ( stmt )* }

expr_stmt -> expression ;

return_stmt -> return (expression)? ;
```

Expressions

```
expression -> logic_term ( | logic_term )*

logic_term -> logic_factor (& logic_factor )*

logic_factor -> (!)? relat_expr

relat_expr -> arith_expr ((<= | <= | > | <= | == | !=) arith_expr )*

arith_expr -> arith_term ((+ | +) arith_term)*

arith_term -> arith_factor ((* | / |  % ) arith_factor)*

arith_factor + r_value  | - r_value | r_value

r_value -> l_value | func_call | INT | FLOAT | STRING | "true"
            | "false" | ( expression )

l_value -> ID
primary_expr -> ID
Simulate -> simulate;
```

Appendix B : Code Listing

NPSL-2Dlexerparser.g

```
/*
* NPSL-2Dgrammer.g : lexer and parser written in antlr
*
* by Glenn Barney
*/

header
{
package npsl;
}

class NPSLLexer extends Lexer;

options{
      k = 2;
      charVocabulary = '\3'..'\377';
      exportVocab = NPSL;
}
```

```
tokens {
	INT;
	FLOAT;
	NEGATE;

	//TODO test these
}

{
	//used for unit test or later error checking
	public int nr_error = 0;
	public void reportError( String s ) {
		super.reportError( s );
		nr_error++;
	}
	public void reportError( RecognitionException e ) {
		super.reportError( e );
		nr_error++;
	}
}

//OPERATORS
PLUS : '+' ;
MINUS : '-' ;
MULT : '*' ;
DIV : '/' ;
MOD : '%' ;
ASSIGN : '=' ;
NOT : '!' ;
LT : '<' ;
GT : '>' ;
LE : "<=" ;
GE : ">=" ;
EQ : "==" ;
NE : "!=";
AND : "&" ;
OR : "|" ;
INC : "++" ;
DEC : "--" ;
COMMA : ',' ;
SEMI : ';' ;
DQUOTE : '"' ;
LPAREN : '(' ;
RPAREN : ')' ;
LBRACE : '{' ;
RBRACE : '}' ;
LBRACK : '[';
RBRACK : ']';


protected LETTER : ('a'..'z' | 'A'..'Z');

protected DIGIT : '0'..'9';

protected EXP options { paraphrase = "an exponent part of a float"; }
		  : ('e' | 'E') ('+' | '-')? (DIGIT)+;
```

```
NEWLINE : ( '\n' | ('\r' '\n') => '\r' '\n' | '\r')
            {newline(); $setType(Token.SKIP);}
;

INT_FLOAT : (DIGIT)+ { $setType(INT); }
                  (('.' (DIGIT)* (EXP)? | EXP) { $setType(FLOAT); })?
                  | '.' (DIGIT)+ (EXP)? { $setType(FLOAT); }
;

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

COMMENT
  : "//" (~('\n'|'\r'))*
    { $setType(Token.SKIP); }
  ;

// multiple-line comments
ML_COMMENT
  : "/*"
    (
      options {
        generateAmbigWarnings=false;
      }
      :  { LA(2)!='/' }? '*'
      | '\r' '\n' {newline();}
      | '\r' {newline();}
      | '\n' {newline();}
      | ~('*'|'\n'|'\r')
    )*
    "*/"
    {$setType(Token.SKIP);}
;

//used for STRING
ESCAPE
      : '\\'
      ( 'n' { $setText("\n"); }
      | 't' { $setText("\t"); }
      | 'b' { $setText("\b"); }
      | 'f' { $setText("\f"); }
      | '\"' { $setText("\""); }
      | '\'' { $setText("\'"); }
      | '\\' { $setText("\\"); } )
;

STRING
      : '"'! ( ESCAPE | ~('\\'|'"') )* '"'!
;

//ID is an identifier : starts with a letter and can have digits,
letters, or _ follow
ID
options {
    testLiterals = true;
```

```
}
    : LETTER (LETTER | DIGIT | '_')* ;
/*
* NPSL-2Dparser.g : parser written in antlr
*
* by Glenn Barney
*/

class NPSLParser extends Parser;

options {

      classHeaderSuffix=org.norecess.antlr.TestableParser;
      k = 2;
      buildAST=true;
      exportVocab=NPSL;

}

tokens {

      //keywords
      UPLUS;
      UMINUS;
      STATEMENT;

      EXPR_LIST;
      VAR_LIST;
      FUNC_CALL;
      DECLARATION;
      DECLARATION_COMPL;
      SIMULATE;
}


{
  private boolean myFailure = false;
  private boolean myQuietErrors = false;

  //used for unit test or later error checking
  public int parse_error_count = 0;

  public void reportError(RecognitionException ex) {
    if (shouldPrintErrors())
      super.reportError(ex);
    parse_error_count++;
    myFailure = true;
  }

  public void reportError( String str ) {
      if (shouldPrintErrors())
                  super.reportError( str );
      parse_error_count++;
  }

  public boolean successfulParse() { return !myFailure; }
```

```
  public void atEOF() {
    try {
      match(EOF);
    } catch (Exception e) {
      myFailure = true;
    }
  }
  public void setQuietErrors(boolean quietErrors) {
    myQuietErrors = quietErrors;
  }
  private boolean shouldPrintErrors() {
    return !myQuietErrors;
  }
}


type
      : base_type | complex_type;

base_type
      :     "int"
      |     "float"
      |   "bool"
      |     "void"
      |   "string"
      ;

complex_type
      :     "Line"
      |     "Circle"
      |     "Rect"
      |   "Point"
      |   "Force"
      ;

program
      : ( stmt | func_def )* EOF!
//    : ( stmt )* EOF!
            {#program = #([STATEMENT,"PROG"], program); }
      ;

//TODO: add an = part
declaration
      : base_type ID (ASSIGN! expression)? SEMI!
      { #declaration = #([DECLARATION, "declaration"], declaration); }
      ;

declaration_compl
      : complex_type ID LBRACE! ( ID ASSIGN! expression SEMI!)* RBRACE!
      { #declaration_compl = #([DECLARATION_COMPL,
"declaration_compl"], declaration_compl); }
      ;


/**
* Statements
**/
```

```
//main statment list
stmt
      : declaration
      | declaration_compl
      | if_stmt
      | assign_stmt
      | func_call_stmt
      | return_stmt
      | simulate
      ;

if_stmt
       : "if"^ LPAREN! expression RPAREN! stmt
       (options {greedy = true;}: "else"! stmt )?
       ;

assign_stmt
      : l_value ASSIGN^ expression SEMI!
      ;

func_call_stmt : func_call SEMI! ;

func_call
      : ID LPAREN! expr_list RPAREN!
            { #func_call = #([FUNC_CALL,"FUNC_CALL"], func_call); }
      ;

expr_list

        : expression ( COMMA! expression )*
            {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list); }
        |   /*from class notes: although empty, we should have a node
*/
            {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list); }
        ;

func_def
        : "function"^ type ID LPAREN! var_list RPAREN! func_body
        ;

var_list
        : type ID ( COMMA! type ID )*
            {#var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
        |   /* add an node for empty list */
            {#var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
        ;

func_body
        : ASSIGN! a:expression SEMI!
            {#func_body = #a; }
        | LBRACE! ( stmt )* RBRACE!
            {#func_body = #([STATEMENT,"FUNC_BODY"], func_body); }
        ;

//this may be useless if I have no side effects
expr_stmt
```

```
      : expression SEMI!
      ;

return_stmt
      : "return"^ (expression)? SEMI!
      ;

//**************
// Expressions
//**************


expression
      :   logic_term ( OR^ logic_term )*
      ;

logic_term
        : logic_factor ( AND^ logic_factor )*
        ;

logic_factor
        : (NOT^)? relat_expr
        ;

relat_expr
        : arith_expr ((GE^ | LE^ | GT^ | LT^ | EQ^ | NE^) arith_expr )*
        ;

arith_expr
      :     arith_term ((PLUS^ | MINUS^) arith_term)*
      ;

arith_term
      :     arith_factor ((MULT^ | DIV^ | MOD^) arith_factor)*
      ;

arith_factor
      :     PLUS! r_value
            {#arith_factor = #([UPLUS,"UPLUS"], arith_factor); }
      |     MINUS! r_value
          {#arith_factor = #([UMINUS,"UMINUS"], arith_factor); }
      |     r_value
      ;

r_value
      :     l_value
      |     func_call
      |     INT
      |     FLOAT
      |   STRING
      |     "true"
      |     "false"
      |     LPAREN! expression RPAREN!
      ;

l_value
      :     ID
```

```
      ;

primary_expr
      :     ID
      ;

simulate
      : "simulate" SEMI!
      ;
```

NPSLwalker.g

```
/*
* NPSLWalker.g : the AST walker.
*
* @author gbarney gb2174@columbia.edu
*
*/

header
{
package npsl;
}

{
      import java.io.*;
      import java.util.*;
}

class NPSLWalker extends TreeParser;

options{
      importVocab = NPSL;
}

{
      static NpslDataType null_data = new NpslDataType( "<NULL>" );
      NpslInterpreter ipt = new NpslInterpreter();
}


expr returns [ NpslDataType r ]
{
      NpslDataType a, b;
      Vector v;
      NpslDataType[] x;
      String s = null;
      NpslDataType[] sx;
      r = null_data;
      String t;
}

      : #(OR a=expr right_or:.)
      {
            if ( a instanceof NpslBool )
                  r = ( ((NpslBool)a).var ? a : expr(#right_or) );
            else
```

```
                  r = a.or( expr(#right_or) );
      }

      | #(AND a=expr right_and:.)
      {
            if ( a instanceof NpslBool )
                  r = ( ((NpslBool)a).var ? expr(#right_and) : a );
            else
                  r = a.and( expr(#right_and) );
      }
      | #("not" a=expr)                   { r = a.not(); }
      | #(GE a=expr b=expr)         { r = a.ge( b ); }
      | #(LE a=expr b=expr)         { r = a.le( b ); }
      | #(GT a=expr b=expr)         { r = a.gt( b ); }
      | #(LT a=expr b=expr)         { r = a.lt( b ); }
      | #(EQ a=expr b=expr)         { r = a.eq( b ); }
      | #(NE a=expr b=expr)         { r = a.ne( b ); }
      | #(PLUS a=expr b=expr)       { r = a.plus( b ); }
      | #(MINUS a=expr b=expr)      { r = a.minus( b ); }
      | #(MULT a=expr b=expr)       { r = a.times( b ); }
      | #(DIV a=expr b=expr)        { r = a.divide( b ); }
      | #(MOD a=expr b=expr)        { r = a.modulus( b ); }
      | #(UPLUS a=expr)                   { r = a; }
      | #(UMINUS a=expr)                  { r = a.uminus(); }
      | decl:DECLARATION                  { declaration(#decl); }
      | #(ASSIGN a=expr b=expr)           { r = ipt.assign( a, b ); }
      | #(FUNC_CALL a=expr x=mexpr)
            { r = ipt.funcInvoke( this, a, x ); }
      | LITERAL_simulate            { ipt.simulate(); }
      | integer:INT                 { r = ipt.getNumber(
integer.getText() ); }
      | floatpt:FLOAT         { r = ipt.getNumber( floatpt.getText() );
}
      | str:STRING                  { r = new NpslString( str.getText()
); }
      | "true"                      { r = new NpslBool( true ); }
      | "false"                     { r = new NpslBool( false ); }
      | id:ID                       { r = ipt.getVariable( id.getText()
); }
      | decl1:LITERAL_Circle  { System.out.println("hit circle one"); }
      | decl2:DECLARATION_COMPL { declaration_compl(#decl2); }

//TODO Imlement for
//    | #("for" x=mexpr forbody:.)
//                      {
//                            }

      | #("if" a=expr thenp:. (elsep:.)?)
                  {
                        if ( !( a instanceof NpslBool ) )
                              return a.error( "if: expecting expression
bool" );
                        if ( ((NpslBool)a).var )
                              r = expr( #thenp );
                        else if ( null != elsep )
                              r = expr( #elsep );
                  }
```

```
	| #(STATEMENT (stmt:. { if ( ipt.canProceed() ) r = expr(#stmt);
} )*)
	//TODO implement loop
//	| #(LOOP loopbody:.
//			(loopid:ID { s = loopid.getText(); })?
//		)
//		{
//			while ( ipt.canProceed() )
//			{
//				r = expr( #loopbody );
//				ipt.loopNext( s );
//			}
//			ipt.loopEnd( s );
//		}
//	| #("break" (breakid:ID { s = breakid.getText(); })?
//		) { ipt.setBreak( s ); }
//	| #("continue" (contid:ID { s = contid.getText(); })?
//		) { ipt.setContinue( s ); }
	| #("return" ( a=expr { r = ipt.rvalue( a ); })?
		) { ipt.setReturn( null ); }
	| #("function"  t=getString fname:ID sx=vlist fbody:.)
		{ ipt.funcRegister( fname.getText(), sx, #fbody, t ); }
	;

declaration
	{
		NpslDataType t;
		NpslDataType a = null;
	}
	: #(DECLARATION t=base_type
		 vname:ID  (a=expr)?
		{ ipt.registerVariable(vname.getText(), t, a); a = null; }
	)
	;

declaration_compl
	{
		NpslDataType t;
		String paramName = null;
		HashMap<String,NpslDataType> fieldsAndValues =  new
HashMap<String,NpslDataType>();
		NpslDataType paramVal = null;
		String thisObjName= null;
		String thisObjType= null;
	}
	//note since these assignments are special I'm not going to
	//create the types yet, just var names as strings to initialize
	//the complex type object
	: #(DECLARATION_COMPL
		//the type
		 (  myType:LITERAL_Circle { thisObjType =
NpslTypeConsts.CIRCLE_TYPE_NAME; }
		       |LITERAL_Line   { thisObjType =
NpslTypeConsts.LINE_TYPE_NAME; }
		       |LITERAL_Force  { thisObjType =
NpslTypeConsts.FORCE_TYPE_NAME; }
```

```
                      |LITERAL_Point  { thisObjType =
NpslTypeConsts.POINT_TYPE_NAME; }
                      |LITERAL_Rect   { thisObjType =
NpslTypeConsts.RECT_TYPE_NAME; }
             )
             //the name
             (thisObjName=getString)
              (paramName=getString paramVal=expr {
fieldsAndValues.put(paramName, paramVal); })*
        ) { ipt.registerComplexVar ( thisObjName, fieldsAndValues,
thisObjType); }
      ;


//takes a string and returns a basic type
base_type returns [ NpslDataType t ]
      {
            t = null;
      }
      : "void" { t = new NpslVoid(); }
            //these are default initialize values;
            //this is a little bit sloppy as we are overwriting this
            //object with a new copy constructor if a is filled
          | "bool" { t = new NpslBool( false ); }
          | "int" { t = new NpslInt( 0 ); }
          | "float" { t = new NpslFloat( 0f ); }
          | "string" { t = new NpslString( "" ); }
;

mexpr returns [ NpslDataType[] rv ]
      {
            NpslDataType a;
            rv = null;
            Vector v;
      }
      : #(EXPR_LIST { v = new Vector(); }
            ( a=expr { v.add( a ); }
            )*
            ) { rv = ipt.convertExprList( v ); }
      | a=expr { rv = new NpslDataType[1]; rv[0] = a; }
      | #(FOR_CON { v = new Vector(); }
            ( s:ID a=expr { a.setName( s.getText() ); v.add(a); })+
            ) { rv = ipt.convertExprList( v ); }
;

vlist returns [ NpslDataType[] sv ]
{
      Vector v;
      NpslDataType myType = null;
      sv = null;
}
      : #(VAR_LIST { v = new Vector(); }

            (myType= base_type s:ID { myType.setName( s.getText() );
v.add(myType);  })*
            ) { sv = ipt.convertExprList( v ); }
```

```
;

getString returns [ String t ]
      {
            t = null;
      }
       : a:. { t= a.getText(); }
      ;
```

Internal Processing Classes

NpslInterpreter.java

```
package npsl;

import java.awt.Color;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Vector;

import javax.swing.JFrame;

import static npsl.NpslUtil.*;

import antlr.collections.AST;

public class NpslInterpreter implements NpslTypeConsts {

      final static int fc_none = 0;
      final static int fc_return = 4;

      String label;

      NpslSymbolTable symt;
      Hashtable funcs;

      private int control = fc_none;

      public NpslInterpreter() {
            symt = new NpslSymbolTable(null, null);
            funcs = new Hashtable<String, NpslFunction>();

            NpslInternalFunctions.register(funcs);
            initWorld();
      }

      // TODO implement assign
      public NpslDataType assign(NpslDataType a, NpslDataType b) {

            if (null != a.name) {
                  // only allow assignment on primitive types
                  if (!(b instanceof NpslComplexType)) {
                        if (!a.typename().equals(UNDEF_TYPE_NAME)
                                    &&
!(a.typename().equals(b.typename())))
```

```
				System.out.println("Warning, rhs  " +
a.name + " of type "
							+ a.typename() + " does not
match expected value "
							+ b.getName() + " of type " +
a.typename());
				NpslDataType x = rvalue(b);
				x.setName(a.name);
				symt.setValue(x.name, x, true, 0); // scope?
				return x;
			}
		}

		return a.error(b, "=");
	}

	// TODO Auto-generated method stub - tie in classes to load at
	// load time using sym table
	private void initWorld() {

	}

	public NpslDataType rvalue(NpslDataType a) {
		if (null == a.name)
			return a;

		return a.copy();
	}

	// TODO implement this
	public NpslDataType funcInvoke(NPSLWalker walker, NpslDataType
func,
			NpslDataType[] params) throws
antlr.RecognitionException {

		/*
		 * Check for internal function that takes no args. If
found, call it and
		 * return.
		 */
		String name = func.getName();
		NpslFunction function = (NpslFunction) funcs.get(name +
"(...)");
		if (null != function)
			return NpslInternalFunctions.call(function, params);

		// now check for the exact function signature
		String funcSig = NpslFunction.getSignature(name, params);
		function = (NpslFunction) funcs.get(funcSig);
		if (null != function)
			return NpslInternalFunctions.call(function, params);

		// Move on to check external functions
		// func must be an existing function
		if (func == null) {
			throw new NpslException("Function not found for ("
					+ (name != null ? name : "<?>") + " )");
```

```
		}

		if (!(func instanceof NpslFunction))
			return func.error("not a function");

		// check numbers of actual and formal arguments
		NpslDataType[] args = ((NpslFunction) func).getArgs();
		if (args.length != params.length)
			return func.error("unmatched length of parameters");

		// create a new symbol table
		symt = new NpslSymbolTable(
				((NpslFunction) func).getParentSymbolTable(),
symt);
		// assign actual parameters to formal arguments
		// check the types here to make sure that they match
		for (int i = 0; i < args.length; i++) {
			if (!params[i].typename().equals(args[i].typename()))
				System.out.println("Warning, on argument " + i
+ " parmeter  "
						+ params[i].name + " of type " +
params[i].typename()
						+ " does not match expected value "
						+ args[i].typename());
			NpslDataType d = rvalue(params[i]);
			d.setName(args[i].getName());
			symt.setValue(args[i].getName(), d, false, 0);
		}

		// call the function body
		NpslDataType r = walker.expr(((NpslFunction)
func).getBody());

		// no break or continue can go through the function
		// TODO implement this
		// if ( control == fc_break || control == fc_continue )
		// throw new MxException( "nowhere to break or continue" );
		// if a return was called
		if (control == fc_return)
			tryResetFlowControl(((NpslFunction) func).name);
		// remove this symbol table and return
		symt = symt.dynamicParent();
		if (!r.typename().equals(NpslTypeConsts.UNDEF_TYPE_NAME)
				&& !r.typename().equals(((NpslFunction)
func).getReturnType())) {
			System.out.println("Warning, function returning "
					+ ((NpslFunction) func).getReturnType()
					+ ", expecting type" + r.typename());
		}

		return r;
	}

	public static NpslDataType getNumber(String s) {

		if (s.indexOf('.') >= 0 || s.indexOf('e') >= 0 ||
s.indexOf('E') >= 0)
```

```
                        return new NpslFloat(Float.parseFloat(s));

                return new NpslInt(Integer.parseInt(s));
        }

        public void tryResetFlowControl(String loop_label) {
                if (null == label || label.equals(loop_label))
                        control = fc_none;
        }

        public boolean canProceed() {
                return (control == fc_none);
        }

        public void setReturn(String label) {
                this.label = label;
                control = fc_return;
        }

        public void registerVariable(String name, NpslDataType create,
                        NpslDataType val) {

                if (null != symt.getValue(name, true, 0))
                        throw new NpslException("duplicate variable: " +
name);
                if (null != val && !(val instanceof NpslComplexType)) {
                        NpslDataType x = val.copy();
                        x.setName(name);
                        symt.setValue(x.name, x, true, 0); // scope?
                } else {
                        create.setName(name);
                        symt.setValue(create.name, create, true, 0);
                }
        }

        public void registerComplexVar(String name,
                        HashMap<String, NpslDataType> fieldToVals, String
type) {

                if (null != symt.getValue(name, true, 0))
                        throw new NpslException("duplicate variable: " +
name);

                if (CIRCLE_TYPE_NAME.equals(type)) {
                        Float mass = getFloat(fieldToVals, MASS);
                        Float radius = getFloat(fieldToVals, RADIUS);
                        Integer xCoord = getInt(fieldToVals, X_COORD);
                        Integer yCoord = getInt(fieldToVals, Y_COORD);

                        if (mass == null || radius == null || xCoord == null
                                        || yCoord == null)
                                throw new NpslException(
                                                "Circle requires numeric values
defined for: " + MASS
                                                        + " and " + RADIUS + "
and " + X_COORD + "and"
                                                        + Y_COORD);
```

```
                else {
                      Boolean free = getBoolean(fieldToVals, FREE);
                      if (free == null)
                            free = true;
                      // todo, support color as base type
                      Color color = Color.RED;
                      Float velocity = getFloat(fieldToVals,
VELOCITY);
                      if (velocity == null)
                            velocity = 0f;
                      Float direction = getFloat(fieldToVals,
DIRECTION);
                      if (direction == null)
                            direction = 0f;

                      NpslCircle circ = new NpslCircle(name, xCoord,
yCoord, radius,
                                  mass, free, color, velocity,
direction);
                      symt.setValue(circ.name, circ, true, 0);
                }
          }

          else if (RECT_TYPE_NAME.equals(type)) {
                Float mass = getFloat(fieldToVals, MASS);
                Integer width = getInt(fieldToVals, WIDTH);
                Integer height = getInt(fieldToVals, HEIGHT);
                Integer xCoord = getInt(fieldToVals, X_COORD);
                Integer yCoord = getInt(fieldToVals, Y_COORD);

                if (mass == null || width == null || height == null
                            || xCoord == null || yCoord == null)
                      throw new NpslException(
                                  "Rect requires numeric values
defined for: " + MASS
                                              + " and " + RADIUS + "
and " + X_COORD + "and"
                                              + Y_COORD);
                else {
                      Boolean free = getBoolean(fieldToVals, FREE);
                      if (free == null)
                            free = true;
                      // todo, support color as base type
                      Color color = Color.RED;
                      Integer xTopLeft = getInt(fieldToVals,
X_COORD_2);
                      // if absent default to current pos x -
width/2, y -height/2
                      if (xTopLeft == null)
                            xTopLeft = xCoord - width / 2;
                      Integer yTopLeft = getInt(fieldToVals,
Y_COORD_2);
                      if (yTopLeft == null)
                            yTopLeft = yCoord - height / 2;
                      Integer xBottomRight = getInt(fieldToVals,
X_COORD_3);
```

```
                        // if absent default to current pos x +
width/2, y +height/2
                        if (xBottomRight == null)
                              xBottomRight = xCoord + width / 2;
                        Integer yBottomRight = getInt(fieldToVals,
Y_COORD_3);
                        if (yBottomRight == null)
                              yBottomRight = yCoord + height / 2;
                        Integer velocity = getInt(fieldToVals,
VELOCITY);
                        if (velocity == null)
                              velocity = 0;
                        Integer direction = getInt(fieldToVals,
DIRECTION);
                        if (direction == null)
                              direction = 0;

                        NpslRect rect = new NpslRect(name, xCoord,
yCoord, xTopLeft,
                                    yTopLeft, xBottomRight,
yBottomRight, width, height,
                                    mass, free, color, velocity,
direction);
                        symt.setValue(rect.name, rect, true, 0);
                  }
            }

            // TODO, store attick behavior
            // for now store the values, they can be manpulated in the
force code
            else if (FORCE_TYPE_NAME.equals(type)) {
                  NpslForce force = new NpslForce(name, fieldToVals);
                  symt.setValue(force.name, force, true, 0);
            }

            else if (LINE_TYPE_NAME.equals(type)) {
                  Integer x1_f = getInt(fieldToVals, X_COORD);
                  Integer y1_f = getInt(fieldToVals, Y_COORD);
                  Integer x2_f = getInt(fieldToVals, X_COORD_2);
                  Integer y2_f = getInt(fieldToVals, Y_COORD_2);
                  if (x1_f == null || y1_f == null || x2_f == null ||
y2_f == null)
                        throw new NpslException(
                                    "Line requires numeric values
defined for: " + X_COORD
                                                + " and " + Y_COORD + "
and " + X_COORD_2
                                                + "and" + Y_COORD_2);
                  else {

                        NpslLine line = new NpslLine(name, x1_f, y1_f,
x2_f, y2_f);
                        symt.setValue(line.name, line, true, 0);
                  }
            }

            else if (POINT_TYPE_NAME.equals(type)) {
```

```
                  Integer x1_f = getInt(fieldToVals, X_COORD);
                  Integer y1_f = getInt(fieldToVals, Y_COORD);
                  if (x1_f == null || y1_f == null)
                        throw new NpslException(
                                    "Point requires numeric values
defined for: " + X_COORD
                                                + " and " + Y_COORD);
                  else {
                        NpslPoint pt = new NpslPoint(name, x1_f, y1_f);
                        symt.setValue(pt.name, pt, true, 0);
                  }
            }

            else {
                  throw new NpslException("Didn't recognize complex
type for : "
                              + type);
            }

      }

      public NpslDataType getVariable(String s) {
            // default static scoping
            NpslDataType x = symt.getValue(s, true, 0);
            if (null == x)
                  return new NpslVariable(s);
            return x;
      }

      public NpslDataType[] convertExprList(Vector v) {
            /* Note: expr list can be empty */
            NpslDataType[] x = new NpslDataType[v.size()];
            for (int i = 0; i < x.length; i++)
                  x[i] = (NpslDataType) v.elementAt(i);
            return x;
      }

      public static String[] convertVarList(Vector v) {
            /* Note: var list can be empty */
            String[] sv = new String[v.size()];
            for (int i = 0; i < sv.length; i++)
                  sv[i] = (String) v.elementAt(i);
            return sv;
      }

      // used for user defined functions
      public void funcRegister(String name, NpslDataType[] args, AST
body,
                  String type) {
            symt.put(name, new NpslFunction(name, args, body, symt,
type));
      }

      // prints out the internal functions
      public void what(PrintWriter output) {
            for (Iterator it = funcs.values().iterator();
it.hasNext();) {
```

```
                NpslFunction d = ((NpslFunction) (it.next()));
                d.what(output);
            }
        }

        public void what() {
            what(new PrintWriter(System.out, true));
        }

        // for junit tests
        public NpslSymbolTable getSymT() {
            return symt;
        }

        // TODO: start the animation
        public void simulate() {
            // TODO get working graphics display
             GraphicRunner example = new GraphicRunner();
             example.start();

        }

}
```

NpslSymbolTable.java

```
package npsl;

import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Iterator;

public class NpslSymbolTable extends HashMap {

        NpslSymbolTable static_parent, dynamic_parent;
        boolean read_only;

        public NpslSymbolTable(NpslSymbolTable sparent, NpslSymbolTable
dparent) {
            static_parent = sparent;
            dynamic_parent = dparent;
            read_only = false;
        }

        public void setReadOnly() {
            read_only = true;
        }

        public final NpslSymbolTable staticParent() {
            return static_parent;
        }

        public final NpslSymbolTable dynamicParent() {
            return dynamic_parent;
        }

        public final NpslSymbolTable parent(boolean is_static) {
```

```
		return is_static ? static_parent : dynamic_parent;
	}

	public final boolean containsVar(String name) {
		return containsKey(name);
	}

	private final NpslSymbolTable gotoLevel(int level, boolean
is_static) {
		NpslSymbolTable st = this;
		if (level < 0) {
			// global variable
			while (null != st.static_parent)
				st = st.parent(is_static);
		} else {
			// local variable
			for (int i = level; i > 0; i--) {
				while (st.read_only) {
					st = st.parent(is_static);
					assert st != null;
				}
				if (null != st.parent(is_static))
					st = st.parent(is_static);
				else

					break;
			}
		}
		return st;
	}

	public final NpslDataType getValue(String name, boolean
is_static, int level) {
		NpslSymbolTable st = gotoLevel(level, is_static);
		Object x = st.get(name);
		while (null == x && null != st.parent(is_static)) {
			st = st.parent(is_static);
			x = st.get(name);
		}
		return (NpslDataType) x;
	}

	public final void setValue(String name, NpslDataType data,
			boolean is_static, int level) {
		NpslSymbolTable st = gotoLevel(level, is_static);
		while (st.read_only) {
			st = st.parent(is_static);
			assert st != null;
		}
		st.put(name, data);
	}

	public void what(PrintWriter output) {
		for (Iterator it = values().iterator(); it.hasNext();) {
			NpslDataType d = ((NpslDataType) (it.next()));
			if (!(d instanceof NpslFunction && ((NpslFunction)
d).isInternal()))
```

```
				d.what(output);
			}
		}

		public void what() {
			what(new PrintWriter(System.out, true));
		}

}
```

NpslInternalFunctions.java

```
package npsl;

import java.io.PrintWriter;
import java.util.Hashtable;
import java.util.Iterator;

//TODO: install math functions, print sin, cos etc
public class NpslInternalFunctions {

	// * Return types */

	/** Printing. */
	private static final int FN_PRINT = 0;
	private static final int FN_PRINTLN = 1;

	public static void register(Hashtable<String, NpslFunction>
funcs) {
		registerFunction(funcs, "print", FN_PRINT,
NpslTypeConsts.VOID_TYPE_NAME);
		registerFunction(funcs, "println", FN_PRINTLN,
NpslTypeConsts.VOID_TYPE_NAME);
	}

	private static void registerFunction(Hashtable<String,
NpslFunction> funcs,
			String name, int id, String retType) {

		NpslFunction func = new NpslFunction(name,
getArguments(id), null,
				retType, id);
		funcs.put(func.getSignature(), func);

	}

	public static NpslDataType call(NpslFunction func, NpslDataType[]
params) {

		checkParameters(func, params);
		String msg;
		switch (func.getInternalId()) {
		case FN_PRINT:
			// msg = func.getName() + "(): ";
			for (int i = 0; i < params.length; i++)
				params[i].print();
			return new NpslVoid();
		case FN_PRINTLN:
```

```
                  // msg = func.getName() + "(): ";
                  for (int i = 0; i < params.length; i++){
                        params[i].print();
                        System.out.println();
                  }
                  return new NpslVoid();
            default:
                  throw new RuntimeException("internal function"
                              + func.getSignature() + "not found.");
            }
      }

      private static NpslDataType[] getArguments(int id) {
            NpslDataType[] args;
            switch (id) {
            case FN_PRINT:
            case FN_PRINTLN:
                  /* (...). */
                  args = null;
                  break;
            default:
                  throw new RuntimeException("unknown internal
function");
            }
            return args;

      }

      private static void checkParameters(NpslFunction func,
NpslDataType[] params) {

            String error = null;

            int id = func.getInternalId();
            if (FN_PRINT == id) {
                  if (0 == params.length)
                        error = func.getName() + "() accepts 1 or more
parameters";
            }
            if (null != error)
                  throw new RuntimeException(error);
      }

}
```

NpslException.java

```
package npsl;

public class NpslException extends RuntimeException {
      NpslException( String msg ) {
            System.err.println( "Error: " + msg );
      }
}
```

NpslTypeConsts.java

```
package npsl;
```

```
public interface NpslTypeConsts {

	//Each type's unique name constant
	public static final String BOOL_TYPE_NAME = "bool";
	public static final String CIRCLE_TYPE_NAME = "Circle";
	public static final String COMPLEX_TYPE_NAME = "complex";
	public static final String BASIC_TYPE_NAME = "unknown";
	public static final String FLOAT_TYPE_NAME = "float";
	static public final String FORCE_TYPE_NAME = "force";
	public static final String FUNCTION_TYPE_NAME = "function";
	public static final String INT_TYPE_NAME = "int";
	public static final String LINE_TYPE_NAME = "Line";
	public static final String POINT_TYPE_NAME = "Point";
	public static final String RECT_TYPE_NAME = "Rect";
	public static final String SHAPE_TYPE_NAME = "Shape";

	public static final String STRING_TYPE_NAME = "string";
	public static final String UNDEF_TYPE_NAME = "undefined-
variable";
	public static final String VOID_TYPE_NAME = "void";
	public static final String COLOR_TYPE_NAME = "color";

	//used for declaring complex objects
	public static final String MASS = "mass";
	public static final String FREE = "free";
	public static final String WIDTH = "width";
	public static final String HEIGHT = "height";
	public static final String RADIUS = "radius";
	public static final String X_COORD = "xCoord";
	public static final String Y_COORD = "yCoord";
	public static final String X_COORD_2 = "xCoord2";
	public static final String Y_COORD_2 = "yCoord2";
	public static final String X_COORD_3 = "xCoord3";
	public static final String Y_COORD_3= "yCoord3";
	public static final String COLOR = "color";
	public static final String VELOCITY = "velocity";
	public static final String DIRECTION = "direction";

}
```

NpslUtil.java

```
package npsl;

import java.util.HashMap;

public class NpslUtil {

	public static Float getFloat( HashMap<String,NpslDataType>
fieldToVals, String toFind) {

		if ( toFind == null) {
			throw new NpslException ("Bad name to find");
		}

		NpslDataType val = fieldToVals.get(toFind);
		if (val == null)
			return null;
```

```
            else
            {
                  if ( val instanceof NpslFloat ) {
                        NpslFloat np = (NpslFloat) ( val ) ;
                        return np.var;
                  }
                  else if ( val instanceof NpslInt ) {
                        NpslInt np = (NpslInt) ( val );
                        return Float.valueOf(np.var);
                  }
                  else
                        throw new NpslException("illegal operation:
Type does" +
                                    "not match int or float, instead
<"+ val.typename() +" >");
            }
}

public static Integer getInt( HashMap<String,NpslDataType> fieldToVals,
String toFind) {

      if ( toFind == null) {
            throw new NpslException ("Bad name to find");
      }

      NpslDataType val = fieldToVals.get(toFind);
      if (val == null)
            return null;
      else
      {
            if ( val instanceof NpslInt ) {
                  NpslInt np = (NpslInt) ( val ) ;
                  return np.var;
            }
            else if ( val instanceof NpslFloat ) {
                  NpslFloat np = (NpslFloat) ( val );
                  return new Integer( (int) (np.var));
            }
            else
                  throw new NpslException("illegal operation: Type
does" +
                              "not match int or float, instead <"+
val.typename() +" >");
      }
}


public static String getString( HashMap<String,NpslDataType>
fieldToVals, String toFind) {

      if ( toFind == null) {
            throw new NpslException ("Bad name to find");
      }

      NpslDataType val = fieldToVals.get(toFind);
      if (val == null)
            return null;
```

```
	else
	{
		if ( val instanceof NpslString ) {
			NpslString np = (NpslString) ( val ) ;
			return np.var;
		}
		else
			throw new NpslException("illegal operation: Type 
does" +
					"not match string, instead <"+ 
val.typename() +" >");
	}
}


	public static Boolean getBoolean( HashMap<String,NpslDataType> 
fieldToVals , String toFind) {

	if ( toFind == null) {
			throw new NpslException ("Bad name to find");
		}

	NpslDataType val = fieldToVals.get(toFind);
	if (val == null)
		return null;
	else
	{
		if ( val instanceof NpslBool ) {
			NpslBool np = (NpslBool) ( val ) ;
			return np.var;
		}
		else
			throw new NpslException("illegal operation:  Type 
does" +
					"not match bool, instead <"+ 
val.typename() +" >");
	}
}

}
```

Type Classes

NpslDataType.java

```
package npsl;

import java.io.PrintWriter;

public class NpslDataType {

	String name; // used in hash table

	public NpslDataType() {
		name = null;
	}

	public NpslDataType(String name) {
```

```
		this.name = name;
	}

	public String typename() {
		return NpslTypeConsts.BASIC_TYPE_NAME;
	}

	public NpslDataType copy() {
		return new NpslDataType();
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public NpslDataType error(String msg) {
		throw new NpslException("illegal operation: " + msg + "( <"
				+ typename() + "> " + (name != null ? name :
"<?>") + " )");
	}

	public NpslDataType error(NpslDataType b, String msg) {
		if (null == b)
			return error(msg);
		throw new NpslException("illegal operation: " + msg + "( <"
				+ typename() + "> " + (name != null ? name :
"<?>") + " and "
				+ "<" + typename() + "> " + (name != null ?
name : "<?>")
				+ " )");
	}

	public void print(PrintWriter w) {
		if (name != null)
			w.print(name + " = ");
		w.println("<undefined>");
	}

	public void print() {
		print(new PrintWriter(System.out, true));
	}

	public void what(PrintWriter w) {
		w.print("<" + typename() + "> ");
		print(w);
	}

	public void what() {
		what(new PrintWriter(System.out, true));
	}

	public NpslDataType assign(NpslDataType b) {
		return error(b, "=");
```

```
}

public NpslDataType uminus() {
	return error("-");
}

public NpslDataType plus(NpslDataType b) {
	return error(b, "+");
}

public NpslDataType minus(NpslDataType b) {
	return error(b, "-");
}

public NpslDataType times(NpslDataType b) {
	return error(b, "*");
}

public NpslDataType divide(NpslDataType b) {
	return error(b, "/");
}

public NpslDataType modulus(NpslDataType b) {
	return error(b, "%");
}

public NpslDataType gt(NpslDataType b) {
	return error(b, ">");
}

public NpslDataType ge(NpslDataType b) {
	return error(b, ">=");
}

public NpslDataType lt(NpslDataType b) {
	return error(b, "<");
}

public NpslDataType le(NpslDataType b) {
	return error(b, "<=");
}

public NpslDataType eq(NpslDataType b) {
	return error(b, "==");
}

public NpslDataType ne(NpslDataType b) {
	return error(b, "!=");
}

public NpslDataType and(NpslDataType b) {
	return error(b, "and");
}

public NpslDataType or(NpslDataType b) {
	return error(b, "or");
}
```

```
	public NpslDataType not() {
		return error("not");
	}

}
```

NpslVariable.java

```
package npsl;

import java.io.PrintWriter;

/**
 * The wrapper class for unsigned variables
 *
 */

public class NpslVariable extends NpslDataType {

	public NpslVariable(String name) {
			super(name);
		}

		public String typename() {
			return NpslTypeConsts.UNDEF_TYPE_NAME;
		}

		public NpslDataType copy() {
			throw new NpslException("Variable " + name
					+ " has not been defined");
		}

		public void print(PrintWriter w) {
			w.println(name + " = <undefined>");
		}
}
```

NpslVoid.java

```
package npsl;

public class NpslVoid extends NpslDataType {


	public NpslVoid() {
		super();
	}

	public String typename() {
		return NpslTypeConsts.VOID_TYPE_NAME;
	}

}
```

NpslComplexType.java

```
package npsl;
```

```
abstract public class NpslComplexType extends NpslDataType {


	NpslComplexType (String name) {
		super (name);

	}
}
```

NpslBool.java

```
package npsl;

import java.io.PrintWriter;

public class NpslBool extends NpslDataType {

	boolean var;

	public NpslBool(boolean var) {
		this.var = var;
	}

	public String typename() {
		return NpslTypeConsts.BOOL_TYPE_NAME;
	}

	public NpslDataType copy() {
		return new NpslBool(var);
	}

	public void print(PrintWriter w) {
		if (name != null)
			w.print(name + " = ");
		w.println(var ? "true" : "false");
	}

	public NpslDataType and(NpslDataType b) {
		if (b instanceof NpslBool)
			return new NpslBool(var && ((NpslBool) b).var);
		return error(b, "and");
	}

	public NpslDataType or(NpslDataType b) {
		if (b instanceof NpslBool)
			return new NpslBool(var || ((NpslBool) b).var);
		return error(b, "or");
	}

	public NpslDataType not() {
		return new NpslBool(!var);
	}

	public NpslDataType eq(NpslDataType b) {
		// not exclusive or
		if (b instanceof NpslBool)
			return new NpslBool((var && ((NpslBool) b).var)
					|| (!var && !((NpslBool) b).var));
```

```
		return error(b, "==");
	}

	public NpslDataType ne(NpslDataType b) {
		// exclusive or
		if (b instanceof NpslBool)
			return new NpslBool((var && !((NpslBool) b).var)
					|| (!var && ((NpslBool) b).var));
		return error(b, "!=");
	}

}
```

NpslFloat.java

```
package npsl;

import java.io.PrintWriter;

public class NpslFloat extends NpslDataType {

	float var;

	public NpslFloat(Float x) {
		var = x;
	}

	public String typename() {
		return NpslTypeConsts.FLOAT_TYPE_NAME;
	}

	public NpslDataType copy() {
		return new NpslFloat(var);
	}

	public static Float FloatValue(NpslDataType b) {
		if (b instanceof NpslFloat)
			return ((NpslFloat) b).var;
		if (b instanceof NpslInt)
			return (Float.valueOf(((NpslInt) b).var));
		b.error("cast to Float");
		return 0f;
	}

	public void print(PrintWriter w) {
		if (name != null)
			w.print(name + " = ");
		w.println(Float.toString(var));
	}

	public NpslDataType uminus() {
		return new NpslFloat(-var);
	}

	public NpslDataType plus(NpslDataType b) {
		return new NpslFloat(var + FloatValue(b));
	}
```

```
	public NpslDataType minus(NpslDataType b) {
		return new NpslFloat(var - FloatValue(b));
	}

	public NpslDataType times(NpslDataType b) {

		return new NpslFloat(var * FloatValue(b));
	}

	public NpslDataType divide(NpslDataType b) {
		return new NpslFloat(var / FloatValue(b));
	}

	public NpslDataType modulus(NpslDataType b) {
		return new NpslFloat(var % FloatValue(b));
	}

	public NpslDataType gt(NpslDataType b) {
		return new NpslBool(var > FloatValue(b));
	}

	public NpslDataType ge(NpslDataType b) {
		return new NpslBool(var >= FloatValue(b));
	}

	public NpslDataType lt(NpslDataType b) {

		return new NpslBool(var < FloatValue(b));
	}

	public NpslDataType le(NpslDataType b) {

		return new NpslBool(var <= FloatValue(b));
	}

	public NpslDataType eq(NpslDataType b) {

		return new NpslBool(var == FloatValue(b));
	}

	public NpslDataType ne(NpslDataType b) {

		return new NpslBool(var != FloatValue(b));
	}

}
```

NpslInt.java

```
package npsl;

import java.io.PrintWriter;

public class NpslInt extends NpslDataType {

	int var;

	public NpslInt(int x) {
```

```
        var = x;
    }

    public String typename() {
        return NpslTypeConsts.INT_TYPE_NAME;
    }

    public NpslDataType copy() {
        return new NpslInt(var);
    }

    public static int intValue(NpslDataType b) {
        if (b instanceof NpslFloat)
            return (int) ((NpslFloat) b).var;
        if (b instanceof NpslInt)
            return ((NpslInt) b).var;
        b.error("cast to int");
        return 0;
    }

    public void print(PrintWriter w) {
        if (name != null)
            w.print(name + " = ");
        w.println(Integer.toString(var));
    }

    public NpslDataType uminus() {
        return new NpslInt(-var);
    }

    public NpslDataType plus(NpslDataType b) {
        if (b instanceof NpslInt)
            return new NpslInt(var + intValue(b));
        return new NpslFloat(var + NpslFloat.FloatValue(b));
    }

    public NpslDataType minus(NpslDataType b) {
        if (b instanceof NpslInt)
            return new NpslInt(var - intValue(b));
        return new NpslFloat(var - NpslFloat.FloatValue(b));
    }

    public NpslDataType times(NpslDataType b) {
        if (b instanceof NpslInt)
            return new NpslInt(var * intValue(b));
        return new NpslFloat(var * NpslFloat.FloatValue(b));
    }

    public NpslDataType divide(NpslDataType b) {
        if (b instanceof NpslInt)
            return new NpslInt(var / intValue(b));
        return new NpslFloat(var / NpslFloat.FloatValue(b));
    }

    public NpslDataType modulus(NpslDataType b) {
        if (b instanceof NpslInt)
            return new NpslInt(var % intValue(b));
```

```
		return new NpslFloat(var % NpslFloat.FloatValue(b));
	}

	public NpslDataType gt(NpslDataType b) {
		if (b instanceof NpslInt)
			return new NpslBool(var > intValue(b));
		return b.lt(this);
	}

	public NpslDataType ge(NpslDataType b) {
		if (b instanceof NpslInt)
			return new NpslBool(var >= intValue(b));
		return b.le(this);
	}

	public NpslDataType lt(NpslDataType b) {
		if (b instanceof NpslInt)
			return new NpslBool(var < intValue(b));
		return b.gt(this);
	}

	public NpslDataType le(NpslDataType b) {
		if (b instanceof NpslInt)
			return new NpslBool(var <= intValue(b));
		return b.ge(this);
	}

	public NpslDataType eq(NpslDataType b) {
		if (b instanceof NpslInt)
			return new NpslBool(var == intValue(b));
		return b.eq(this);
	}

	public NpslDataType ne(NpslDataType b) {
		if (b instanceof NpslInt)
			return new NpslBool(var != intValue(b));
		return b.ne(this);
	}

}
```

NpslString.java

```
package npsl;

import java.io.PrintWriter;

public class NpslString extends NpslDataType {

	String var;

	public NpslString(String str) {
		this.var = str;
	}

	public String typename() {
		return NpslTypeConsts.STRING_TYPE_NAME;
	}
```

```
	public NpslDataType copy() {
		return new NpslString(var);
	}

	public void print(PrintWriter w) {
		if (name != null)
			w.print(name + " = ");
		w.print(var);
		w.println();
	}

	public NpslDataType plus(NpslDataType b) {
		if (b instanceof NpslString)
			return new NpslString(var + ((NpslString) b).var);
		return error(b, "+");
	}

	public NpslDataType add(NpslDataType b) {
		if (b instanceof NpslString) {
			var = var + ((NpslString) b).var;
			return this;
		}
		return error(b, "+=");

	}
}
```

NpslFunction.java

```
package npsl;

import java.io.PrintWriter;

import antlr.collections.AST;

public class NpslFunction extends NpslDataType {

	// we need a reference to the AST for the function entry
	NpslDataType[] args;
	AST body;			// body = null means an internal
function.
	NpslSymbolTable pst;	// the symbol table of static parent
	int id;			// for internal functions only
	private String funcSig; /* main(), cos(float), print(...). */
	private String retType; //return type

	public NpslFunction(String name, NpslDataType[] args, AST body,
			NpslSymbolTable pst, String retType) {
		super(name);
		this.args = args;
		this.body = body;
		this.pst = pst;
		funcSig = getSignature(name, args);
		this.retType = retType;
	}
```

```
	public NpslFunction(String name, NpslDataType[] args, AST body ,
String retType) {
		super(name);
		this.args = args;
		this.body = body;
		funcSig = getSignature(name, args);
		this.retType = retType;
	}

	public NpslFunction(String name, NpslDataType[] args, AST body,
String retType, int id) {
		super(name);
		this.args = args;
		this.body = body;
		funcSig = getSignature(name, args);
		this.id = id;
		this.retType = retType;
	}

	boolean isInternal() {
		return body == null;
	}

	public final int getInternalId() {
		return id;
	}

	public final String getReturnType() {
		return retType;
	}

	public String typename() {
		return NpslTypeConsts.FUNCTION_TYPE_NAME;
	}

	public NpslDataType copy() {
		return new NpslFunction(name, args, body, pst, retType);
	}

	public void print(PrintWriter w) {
		if (body == null) {
			w.println( funcSig +" = <internal-function> #" + id);
		}
		else
			w.println ( funcSig +" = <defined-function> #" + id);
	}

	public NpslDataType[] getArgs() {
		return args;
	}

	public NpslSymbolTable getParentSymbolTable() {
		return pst;
	}

	public AST getBody() {
		return body;
```

```
	}

	public String getName() {
		return name;
	}

	//signature not based on return type
	public static String getSignature(String name, NpslDataType[]
args) {
		if (null == args || args.length == 0)
			return name + "(...)";
		String funcSig = name + "(";
		for (int i = 0; i < args.length; i++) {
			if (!(args[i] instanceof NpslDataType))
				throw new RuntimeException("unknown argument
type passed to "
						+ "function " + name + "()");
			funcSig += args[i].typename();
			if (i < args.length - 1)
				funcSig += ",";
		}
		funcSig += ")";
		return funcSig;
	}

	public String getSignature() {
		return funcSig;
	}
}
```

NpslShape.java

```
package npsl;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.geom.RectangularShape;
import java.util.List;

import javax.swing.JComponent;


abstract public class NpslShape extends NpslComplexType {

	public NpslShape (String name) {
		super (name);
		_pic = new BaseComponent();

	}

	//graphical representation of the object
	protected JComponent _pic;

	//shape of the object
	protected RectangularShape _shape;
```

```
      //color of the object
      protected Color _color = Color.blue;

      //list of all objects touching this object
      //at the current time
      protected List touching;

      //list of all objects attached to this object
      //at the current time.
      protected List attached;

      //at the current tick, calculates where the object should be
      //compared to the last tick. override, does nothing now
    protected void calcPosition () {};
    //given the internal position, set the position on the
    //shape object so it can be rendered
    protected abstract void setPosition ();

    //getter for the component
    protected JComponent getPic() {

      return _pic;
    }

      public class BaseComponent extends JComponent {

             /** Draws the shape on the screen. */
          @Override
          public void paintComponent(Graphics g) {
              super.paintComponent(g);
              Graphics2D g2 = (Graphics2D)g;
              // Antialiasing for smooth surfaces.
              g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_ON);

              // Determine position after this time increment
              calcPosition ();
              setPosition ();

              // Want a solid red ball
              g2.setColor (_color);
              g2.fill(_shape);

              // Now draw the ball.
              g2.draw (_shape);

          }


      }

}
```

NpslCircle.java

```
package npsl;

import java.awt.Color;
```

```
import java.awt.geom.Ellipse2D;
import java.io.PrintWriter;

public class NpslCircle extends NpslShape implements Movable {

      public String typename() {
            return NpslTypeConsts.CIRCLE_TYPE_NAME;
      }

      int _x;
      int _y;
      float _radius;
      float _weight;

      boolean _free; //true if free to move, false if fixed

      float _velocity; //velocity of the current object
      float _direction; //direction of the velocity in radians


      //required fields for the constructor are
      //xCoord: the x center position of the ball
      //yCoord: the y center position of the ball
      //radius : radius of the ball
      //weight : weight of the ball in kilograms
      //TODO: adjust _x,_y so they aren't on the top left
      public  NpslCircle (String name, int xCoord, int yCoord, float
radius,
                  float weight, boolean free, Color color, float
velocity, float dir) {
            super (name);
            _x = xCoord;
            _y = yCoord;
            _radius = radius;
            _weight = weight;
            _free = free;
            _color = color;
            _velocity = velocity;
            _direction = dir;

            _shape = new Ellipse2D.Float(_x, _y, _radius*2, _radius*2
);


      }


      public void setPosition() {
            // Move the object.
            _shape.setFrame(_x, _y,  _radius*2, _radius*2);
      }

      public void print(PrintWriter w) {

            w.println("Circle with values x="+_x+" _y="+_y+"
radius="+_radius+".");
      }
```

```
	public boolean isMovable() {
		return _free;
	}

}
```

NpslRect.java

```
package npsl;

import java.awt.Color;
import java.awt.geom.Rectangle2D;
import java.io.PrintWriter;

public class NpslRect extends NpslShape implements Movable{

	int _x;
	int _y;

	//needed for slanted shapes
	int _xTopLeft;
	int _yTopLeft;
	int _xBottomRight;
	int _yBottomRight;

	float _width;
	float _height;
	float _weight;

	boolean _free; //true if free to move, false if fixed

	float _velocity; //velocity of the current object
	float _direction; //direction of the velocity in radians


	//required fields for the constructor are
	//xCoord: the x center position of the rectangle
	//yCoord: the y center position of the rectange
	//width : width of the ball
	//height : height of the ball
	//weight : weight of the ball in kilograms
	//TODO: adjust _x,_y so they aren't on the top left
	//TODO: adjust _x,_y so they aren't on the top left
	public  NpslRect(String name, int xCoord, int yCoord, int 
xTopLeft, int yTopLeft,
			int xBottomRight, int yBottomRight, float width, 
float height,
			float weight, boolean free, Color color, float 
velocity, float dir) {
		super (name);
		_x = xCoord;
		_y = yCoord;
		_xTopLeft = xTopLeft;
		_yTopLeft = yTopLeft;
		_xBottomRight = xBottomRight;
		_yBottomRight = yBottomRight;
```

```
		_height = height;
		_width = width;
		_weight = weight;

		_free = free;
		_color = color;
		_velocity = velocity;
		_direction = dir;

		_shape = new Rectangle2D.Float( _x, _y, _width, _height );


	}

	//by default calcPosition doesn't do anything, must override
	//to move the object
	public void calcPosition () {
    // Move the object.

	}

	public void print(PrintWriter w) {
		w.println("Rect with values x="+_x+" _y="+_y+"
width="+_width+" height="+_height+".");
	}

	public void setPosition() {
		// Move the object.
		_shape.setFrame(_x, _y,  _width, _height);
	}

	//right now all rectangles are not movable because of the tourque
and spin
	//calculations that would be quite complex, this is a TODO for
the future
	public boolean isMovable() {

		return false;
	}

}
```

NpslLine.java

```
package npsl;

import java.awt.Point;
import java.io.PrintWriter;

public class NpslLine extends NpslComplexType {

	private Point p1;
	private Point p2;

	public NpslLine(String name, int x1, int y1, int x2, int y2) {
		super(name);
		p1 = new Point(x1, y1);
```

```
		p2 = new Point(x2, y2);
	}

	void setPoint1(Point x) {
		p1 = x;
	}

	Point getPoint1() {
		return p1;
	}

	void setPoint2(Point x) {
		p2 = x;
	}

	Point getPoint2() {
		return p2;
	}

	public String typename() {
		return NpslTypeConsts.LINE_TYPE_NAME;
	}


	public void print(PrintWriter w) {
		w.println("Point with values P1="+p1+", and P2="+p2);
	}
}
```

NpslPoint.java

```
package npsl;

import java.awt.Point;
import java.io.PrintWriter;

public class NpslPoint extends NpslComplexType {


	private Point p;

	NpslPoint(String name, int x1, int y1)
	{
		super(name);
		p = new Point (x1, y1);
	}

	void setPoint(Point x) {
		p = x;
	}

	Point getPoint() {
		return p;
	}

	public void print(PrintWriter w) {
		w.println("Point with values P="+p);
	}
```

```
}
```

NpslColor.java

```
package npsl;
import java.awt.Color;

//NPSL data type wraps the java color class
public class NpslColor extends NpslDataType {

      public String typename() {
            return NpslTypeConsts.COLOR_TYPE_NAME;
      }


    public enum NColor {
      BLACK (Color.black),
      BLUE (Color.blue),
      CYAN (Color.cyan),
      DARK_GRAY (Color.darkGray),
      LIGHT_GRAY (Color.lightGray),
      GRAY (Color.gray),
      GREEN (Color.green),
      MAGENTA (Color.magenta),
      ORANGE (Color.orange),
      PINK (Color.pink),
      RED (Color.red),
      WHITE (Color.white),
      YELLOW (Color.yellow);


      private final Color color;
      NColor (Color c) {
            color =c;
      }

    }
}
```

Internal Physics Classes

Movable.java

```
package npsl;

public interface Movable {

      boolean isMovable();

}
```

NpslForce.java

```
package npsl;

import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Iterator;

public class NpslForce extends NpslComplexType {
```

```
	//keeps track of all values that can be referenced later in the
force as passed in
	HashMap<String, NpslDataType> _memberVars;

	NpslForce (String name, HashMap<String,NpslDataType> memberVars)
	{
		super(name);
		_memberVars = memberVars;
	}


	public String typename() {
		return NpslTypeConsts.FORCE_TYPE_NAME;
	}

	//prints out the internal functions
	public void print(PrintWriter output) {
		for (Iterator it = _memberVars.values().iterator();
it.hasNext();) {
			NpslFunction d = ((NpslFunction) (it.next()));
				d.what(output);
		}
	}

}
```

ForceManager.java

```
package npsl;

public class ForceManager {

}
```

ObjectManager.java

```
package npsl;

//Will be called from the running thread to calculate
//hits and change velocity of shapes
public class ObjectManager {


	//TODO: implement hit test
	public static NpslPoint hitTest (NpslShape shape1, NpslShape
shape2) {

		return null;

	}



}
```

Rendering Classes

GraphicRunner.java

```
package npsl;

import javax.swing.JFrame;

public class GraphicRunner extends Thread {

      public void run() {
            JFrame window = new JFrame("Proof of Concept Demo");
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            window.setSize(500,500);
            MainPanel thePanel = new MainPanel();
            thePanel.buildTest();
        window.setContentPane(thePanel);
        //window.pack();
        //System.out.println(window.getContentPane().getSize());
        window.setLocationRelativeTo(null);
        window.setVisible(true);
      }

}
```

MainPanal.java

```
package npsl;

import java.util.*;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.geom.Ellipse2D;

import javax.swing.BorderFactory;
import javax.swing.JPanel;

public class MainPanel extends JPanel {

      public MainPanel () {
            //... Layout components

            this.setLayout(new BorderLayout(5, 5));
        this.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
      }

      private List<NpslShape> objectList = new ArrayList<NpslShape> ();

      public void addObj (NpslShape _add) {
            objectList.add(_add);
      }

      public void remObj (NpslShape _add) {
            objectList.add(_add);
      }

      public void buildTest() {

            NpslCircle fc1 =  new NpslCircle("test", 100,100, 5f, 10f,
```

```
                        true, Color.red, 0, 0);
            this.add(fc1.getPic());
      }

}
```

World.java

```
package npsl;

import javax.swing.*;


public class World extends JApplet {

    //================================================= applet
constructor
    /** Applet constructor requires putting the panel in applet.*/
    public World() {
        this.setContentPane(new World());
    }

    //========================================================= method
main
    /** Create JFrame and set content pane to a RollDicePanel. */
    public static void main(String[] args) {
        JFrame window = new JFrame("Proof of Concept Demo");
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            window.setSize(500,500);
            MainPanel thePanel = new MainPanel();
            thePanel.buildTest();
        window.setContentPane(thePanel);
        //window.pack();
        //System.out.println(window.getContentPane().getSize());
        window.setLocationRelativeTo(null);
        window.setVisible(true);
    }

}
```

Main program

NpslMain.java

```
package npsl;

import java.io.*;
import antlr.CommonAST;
import antlr.RecognitionException;
import antlr.TokenStreamException;

public class NpslMain {

      static boolean verbose = false;

      public static int runFile(String programFile, boolean verb) {
            verbose = verb;
            return runFile(programFile);
```

```
	}

	public static int runFile(String programFile) {

		try {
			InputStream input = (null != programFile) ?
(InputStream) new FileInputStream(
					programFile)
					: (InputStream) System.in;
			NPSLLexer lexer = new NPSLLexer(input);
			NPSLParser parser = new NPSLParser(lexer);
			// Parse the input program
			parser.program();
			if (lexer.nr_error > 0 || parser.parse_error_count >
0) {
				System.err.println("Parsing errors found.
Stop.");
				return -1;
			}
			CommonAST tree = (CommonAST) parser.getAST();
			if (verbose) {
				// Print the resulting tree out in LISP
notation
				System.out
						.println("========== Here's the
tree ================");
				System.out.println(tree.toStringList());
			}
			NPSLWalker walker = new NPSLWalker();
			// Traverse the tree created by the parser

			if (verbose)
				System.out
						.println("========== ogram output
==================");
			NpslDataType r = walker.expr(tree);
			if (verbose)
				System.out
						.println("========== program return
=================");
			if (null != r)
				r.print();
			if (verbose) {
				System.out
				.println("========== internal functions
================");
				walker.ipt.what();
				System.out
						.println("========== global
variables ================");
				walker.ipt.symt.what();
			}
		} catch (IOException e) {
			System.err.println("Error: I/O: " + e);
			return -1;
		} catch (RecognitionException e) {
			System.err.println("Error: Recognition: " + e);
```

```
                return -1;
            } catch (TokenStreamException e) {
                System.err.println("Error: Token stream: " + e);
                return -1;
            } catch (Exception e) {
                System.err.println("Error: " + e);
                e.printStackTrace();
                return -1;
            }
            return 0;
      }

      public static void main(String[] args) {
            verbose = args.length >= 1 && args[0].equals("-v");
            boolean batch = args.length >= 1 && args[0].equals("-b");
            if (args.length >= 1 && args[args.length - 1].charAt(0) !=
'-')
                  runFile(args[args.length - 1]);
            else if (batch)
                  runFile(null);
            System.exit(0);
      }
}
```

Build File

build.xml

```
<project name="NPSL-2D" default="all" basedir=".">

      <!-- set global properties for this build -->
      <property name="src" location="src"/>
      <property name="build" location="build"/>
      <property name="dist"  location="dist"/>
      <property name="tests"  location="tests"/>
      <property name="bin"  location="bin"/>
      <property name="tst-dir"  location="${tests}/test"/>
      <property name="lib"  location="lib"/>

      <property name="TALK" value="false" />

      <path id="classpath.test.jars">
          <pathelement location="${lib}/junit.jar" />
          <pathelement location="${lib}/antlrtesting-0.6.jar" />
          <pathelement location="${lib}/antlr.jar" />
      </path>

      <path id="classpath.build.test">
            <path refid="classpath.test.jars" />
        <pathelement location="${bin}" />
        </path>

      <path id="classpath.build.java">
          <pathelement location="${lib}/antlr.jar" />
          <pathelement location="${lib}/antlrtesting-0.6.jar" />
```

```
    </path>

      <path id="classpath.test">
            <path refid="classpath.test.jars" />
        <pathelement location="${bin}" />
          <pathelement location="${tests}" />
      </path>

      <target name="all" depends="base,compileJava" />

      <target name="base" depends="clean">
            <!--antlr target="${src}/NPSL-2Dgrammer.g"
outputdirectory="${src}/npsl/">
                  <classpath path="lib/antlr.jar" />
            </antlr>
            <antlr target="${src}/NPSL-2Dparser.g"
outputdirectory="${src}/npsl/">
                  <classpath path="lib/antlr.jar" />
            </antlr>
                  -->
            <antlr target="${src}/NPSL-2Dlexerparser.g"
outputdirectory="${src}/npsl/">
                  <classpath path="lib/antlr.jar" />
            </antlr>
            <antlr target="${src}/NPSLWalker.g"
outputdirectory="${src}/npsl/">
                  <classpath path="lib/antlr.jar" />
            </antlr>
      </target>

      <target name="compileJava" depends="base">
            <mkdir dir="bin" />
            <javac srcdir="src" destDir="bin" verbose="${TALK}">
                  <classpath refid="classpath.build.java"/>
            </javac>
      </target>

      <target name="runTests">
            <java classpath="{bin};lib/antlr.jar"
classname="com.javadude.xl2.XLRecognizer">
                  <arg value="tests"/>
            </java>
      </target>

      <target name="compile-test" depends="compileJava">
          <javac srcdir="${tst-dir}"
                 verbose="${TALK}">
            <classpath refid="classpath.build.test"/>
          </javac>
      </target>

      <target name="clean-compile-test">
          <delete verbose="${TALK}">
            <fileset dir="${bin}" includes="**/*.class" />
              <fileset dir="${bin}/npsl" includes="**/*.class" />
          </delete>
      </target>
```

```
      <target name="clean" depends="clean-compile-test">
          <delete verbose="${TALK}">
            <fileset dir="${tst-dir}" includes="**/*.class" />
              <fileset dir="${src}/npsl" includes="**/*.txt" />
            <fileset dir="${src}/npsl" includes="**/*.smap" />

              <fileset dir="${src}/npsl" includes="**/*Lexer*.java" />
              <fileset dir="${src}/npsl" includes="**/*Walker*.java" />

              <fileset dir="${src}/npsl" includes="**/*Parser*.java" />
              <fileset dir="${src}/npsl" includes="**/*Token*.java" />

          </delete>
      </target>

      <target name="test" depends="compile-test">
      <junit printsummary="yes">
            <classpath refid="classpath.test" />
            <batchtest fork="yes" todir="${tests}/reports">
                  <formatter type="plain"/>
                  <fileset dir="${tests}">
                        <include name="**/*.java" />
                  </fileset>
            </batchtest>
      </junit>
      </target>

</project>
```

Tests

Test Data Files

```
/*
 *
 * ArithmaticParseTest.java
 *
 */
package test;

import static npsl.NPSLTokenTypes.DIV;
import static npsl.NPSLTokenTypes.MOD;
import static npsl.NPSLTokenTypes.MULT;
import static npsl.NPSLTokenTypes.PLUS;

import java.io.StringReader;

import npsl.NPSLLexer;
import npsl.NPSLParser;

import org.norecess.antlr.ParserTestCase;
import org.norecess.antlr.TestableParser;

import antlr.TokenStream;
```

```
public class ArithmaticParseTest extends ParserTestCase{

      protected TokenStream makeLexer(String input) {
              return new NPSLLexer(new StringReader(input));
            }
            protected TestableParser makeParser(TokenStream lexer) {
              NPSLParser parser = new NPSLParser(lexer);

parser.getASTFactory().setASTNodeClass("org.norecess.antlr.LineNumberAS
T");
              return parser;
            }

            public void testArithIntegerPlus() {
                  assertPreorder("parse 3+2 to arith", PLUS,
"(+(3)(2))", produce("arith_expr", "3+2"));
            }

            public void testArithIntegerTimes() {
                  assertPreorder("parse 3*2 to arith", MULT,
"(*(3)(2))", produce("arith_expr", "3*2"));
            }

            public void testArithIntegerDivide() {
                  assertPreorder("parse 3/2 to arith", DIV,
"(/(3)(2))", produce("arith_expr", "3/2"));
            }

            public void testArithIntegerMod() {
                  assertPreorder("parse  3%2 to arith", MOD,
"(%(3)(2))", produce("arith_expr", "3%2"));
            }

            public void testArithFloatPlus() {
                  assertPreorder("parse 3.65E4+2.78E-2 to arith", PLUS,
"(+(3.65E4)(2.78E-2))", produce("arith_expr", "3.65E4+2.78E-2"));
            }

            public void testArithIntAndFloat() {
                  assertPreorder("parse 3.647+2 to arith", PLUS,
"(+(3.647)(2))", produce("arith_expr", "3.647+2"));
            }

            public void testOrderOfOp() {
                  assertPreorder("parse 1+(2*3)/4 order of Op", PLUS,
"(+(1)(/(*(2)(3))(4)))", produce("arith_expr", "1+2*3/4"));
            }

            public void testNestedParens() {
                  assertPreorder("parse 1+(2*3)/4 order of Op", PLUS,
"(+(1)(/(*(2)(3))(4)))", produce("arith_expr", "1+(2*3)/4"));
            }

            public void testNestedParens2() {
                  assertPreorder("parse (1+2)*3/4 order of Op", DIV,
"(/(*(+(1)(2))(3))(4))", produce("arith_expr", "(1+2)*3/4"));
            }
```

```
            public void testUnaryMinus() {
                  assertPreorder("parse 1+(2*-3)/4 order of Op", PLUS,
"(+(1)(*(2)(UMINUS(3))))", produce("arith_expr", "1+2*-3"));
            }

            public void testUnaryPlusOne() {
                  assertPreorder("parse 1+(2*-3)/4 order of Op", PLUS,
"(+(1)(*(2)(UPLUS(3))))", produce("arith_expr", "1+2*+3"));
            }

            public void testUnaryPlusWithParen() {
                  assertPreorder("parse 1+(2*-3)/4 order of Op", PLUS,
"(+(1)(*(UPLUS(2))(3)))", produce("arith_expr", "1+(+2)*3"));
            }


}


/*
 * ComplexTest.java
 */

package test;

import org.junit.Test;
import npsl.NpslMain;

import junit.framework.TestCase;

public class ComplexTest extends TestCase{


      @Test
      public void testAssignments()
      {
            System.out.println("*********Test assignemnt********");
            NpslMain exec = new NpslMain();
            assertTrue (exec.runFile("tests/data/complex.npsl", true)
>= 0);
      }
}


/*
 * ConstantParseTest.java
 */

package test;

import java.io.StringReader;
import org.norecess.antlr.ParserTestCase;
import org.norecess.antlr.TestableParser;
import antlr.TokenStream;
import antlr.TokenStreamException;
import antlr.collections.AST;
```

```
import npsl.NPSLLexer;
import npsl.NPSLParser;

//types to test
import static npsl.NPSLTokenTypes.*;


public class ConstantParseTest extends ParserTestCase{

      protected TokenStream makeLexer(String input) {
              return new NPSLLexer(new StringReader(input));
            }
            protected TestableParser makeParser(TokenStream lexer) {
              NPSLParser parser = new NPSLParser(lexer);

parser.getASTFactory().setASTNodeClass("org.norecess.antlr.LineNumberAS
T");
              return parser;
            }



      public void testInt() {
            assertPreorder("parse 3+2 to arith", INT, "(5)",
produce("r_value", "5"));
      }

      public void testFloat() {
            assertPreorder("parse 123.456", FLOAT, "(123.456)",
produce("r_value", "123.456"));
      }

      public void testString() {
            assertPreorder("parse testmestring", STRING,
"(testmestring)", produce("r_value", "\"testmestring\""));
      }

      public void testBoolean() {
            assertPreorder("parse true", LITERAL_true, "(true)",
produce("r_value", "true"));
      }
}

/*
 * ControlFlowLexTest.java
 */

package test;

import java.io.StringReader;

import org.junit.Ignore;
import org.norecess.antlr.LexerTestCase;
import npsl.NPSLLexer;
import antlr.TokenStream;
```

```
@Ignore
public class ControlFlowLexTest extends LexerTestCase {

      //TODO : implement for loops and whiles, then test
      protected TokenStream makeLexer(String input) {
              return new NPSLLexer(new StringReader(input));
            }


}

/*
 * ControlFlowParseTest.java
 */

package test;

import java.io.StringReader;

import org.junit.Ignore;
import org.norecess.antlr.ParserTestCase;
import org.norecess.antlr.TestableParser;

import antlr.TokenStream;
import npsl.NPSLLexer;
import npsl.NPSLParser;

public class ControlFlowParseTest extends ParserTestCase{

      protected TokenStream makeLexer(String input) {
              return new NPSLLexer(new StringReader(input));
            }
            protected TestableParser makeParser(TokenStream lexer) {
              NPSLParser parser = new NPSLParser(lexer);

parser.getASTFactory().setASTNodeClass("org.norecess.antlr.LineNumberAS
T");
              return parser;
            }

            @Ignore
            //TODO : implement for loops and whiles, then test
            public void test ()
            {

            }

}

/*
 * DeclarationAssignmentTest.java
 */

package test;

import org.junit.Test;
import npsl.NpslMain;
```

```
import junit.framework.TestCase;

public class DeclarationAssignmentTest extends TestCase{


	@Test
	public void testAssignments()
	{
		System.out.println("*********Test assignemnt********");
		NpslMain exec = new NpslMain();
		assertTrue (exec.runFile("tests/data/assignment.npsl",
true) >= 0);
	}


	@Test
	public void testDeclaration()
	{
		System.out.println("*********Test declaration********");
		NpslMain exec = new NpslMain();
		assertTrue (exec.runFile("tests/data/declaration.npsl",
true) >=0);
	}

	@Test
	public void testFunciton()
	{
		System.out.println("*********Test functions*******");
		NpslMain exec = new NpslMain();
		assertTrue(exec.runFile("tests/data/function.npsl", true)
>=0);

	}



}
/*
 * ExpressionLexTest.java
 */

package test;

import java.io.StringReader;

import npsl.NPSLLexer;

import org.junit.Ignore;
import org.norecess.antlr.LexerTestCase;

import antlr.TokenStream;

@Ignore
public class ExpressionLexTest extends LexerTestCase {

	protected TokenStream makeLexer(String input) {
```

```
                return new NPSLLexer(new StringReader(input));
            }


}

/*
 * ExpressionParseTest.java
 */

package test;

import java.io.StringReader;

import org.norecess.antlr.ParserTestCase;
import org.norecess.antlr.TestableParser;

import antlr.TokenStream;
import npsl.NPSLLexer;
import npsl.NPSLParser;

import static npsl.NPSLTokenTypes.*;

public class ExpressionParseTest extends ParserTestCase{

      protected TokenStream makeLexer(String input) {
                return new NPSLLexer(new StringReader(input));
            }

      protected TestableParser makeParser(TokenStream lexer) {
                NPSLParser parser = new NPSLParser(lexer);

parser.getASTFactory().setASTNodeClass("org.norecess.antlr.LineNumberAS
T");
                return parser;
      }

      public void testL_value() {
            //System.out.println("["+produce("relat_expr",
"1+2==3>4").toStringTree()+"]");
            assertPreorder("parse hi", ID, "(hi)", produce("l_value",
"hi"));
      }

      public void testL_value_True() {
            //System.out.println("["+produce("relat_expr",
"1+2==3>4").toStringTree()+"]");
            assertFailedParse("parse keyword true as id", "l_value"
,"true");
      }

      public void testR_value() {
            //System.out.println("["+produce("relat_expr",
"1+2==3>4").toStringTree()+"]");
            //assertPreorder("parse id", ID, "(hi)", produce("r_value",
"sopu"));
```

```
		assertPreorder("parse int", INT, "(123)",
produce("r_value", "123"));
		assertPreorder("parse float", FLOAT, "(11.34e10)",
produce("r_value", "11.34e10"));
		assertPreorder("parse expr", LITERAL_true, "(true)",
produce("r_value", "(true)"));
	}

	public void testArith_factor() {
		//assertPreorder("parse +id", UPLUS, "(UPLUS(hi))",
produce("arith_factor", "+hi"));
		assertPreorder("parse -id", UMINUS, "(UMINUS(43))",
produce("arith_factor", "-43"));
		assertPreorder("arithfact 12.3", FLOAT, "(12.3)",
produce("arith_factor", "12.3"));
	}

	public void testArith_term() {
		assertPreorder("parse 3*2", MULT, "(*(3)(2))",
produce("arith_term", "3*2"));
		assertPreorder("parse 3*2%4", MOD, "(%(*(3)(2))(4))",
produce("arith_term", "3*2%4"));
	}

	public void testArith_Expr() {
		assertPreorder("parse 3+2 to arith", PLUS, "(+(3)(2))",
produce("arith_expr", "3+2"));
	}

	public void testRelat_Expr() {
		assertPreorder("parse 1+2=3", EQ, "(==(+(1)(2))(3))",
produce("relat_expr", "1+2==3"));
	}

	public void testLogic_Factor() {
		assertPreorder("parse !4>6", NOT, "(!(>(4)(6)))",
produce("logic_factor", "!4>6"));
	}

	public void testLogic_Term() {
		assertPreorder("parse and", AND, "(&(hi)(==(2)(2.42)))",
produce("logic_term", "hi&2==2.42"));
		assertPreorder("parse and", AND, "(&(true)(==(2)(2.42)))",
produce("logic_term", "true&2==2.42"));
	}

	public void testExpression() {
		//System.out.println("["+produce("expression",
"true").toStringTree()+"]");
		assertPreorder("parse true!=true", NE, "(!=(true)(true))",
produce("expression", "true!=true"));

	}

	public void testType() {
		//System.out.println("["+produce("expression",
"true").toStringTree()+"]");
```

```
            assertPreorder("parse int", LITERAL_int, "(int)",
produce("type", "int"));
            assertPreorder("parse string", LITERAL_string, "(string)",
produce("type", "string"));
            assertPreorder("parse bool", LITERAL_bool, "(bool)",
produce("type", "bool"));
            assertPreorder("parse float", LITERAL_float, "(float)",
produce("type", "float"));
            assertPreorder("parse Line", LITERAL_Line, "(Line)",
produce("type", "Line"));
            assertPreorder("parse Circle", LITERAL_Circle, "(Circle)",
produce("type", "Circle"));
            assertPreorder("parse Rect", LITERAL_Rect, "(Rect)",
produce("type", "Rect"));
            assertPreorder("parse Force", LITERAL_Force, "(Force)",
produce("type", "Force"));
      }

      public void testBasicType() {
            //System.out.println("["+produce("expression",
"true").toStringTree()+"]");
            assertPreorder("parse int", DECLARATION,
"(declaration(int)(x)(4))", produce("declaration", "int x=4; "));
            assertPreorder("parse string", DECLARATION,
"(declaration(string)(x)(foo))", produce("declaration", "string
x=\"foo\"; "));
            assertPreorder("parse bool", DECLARATION,
"(declaration(bool)(y)(true))", produce("declaration", "bool y=true;
"));
            assertPreorder("parse float", DECLARATION,
"(declaration(float)(x)(4.3))", produce("declaration", "float x=4.3;
"));
      }

      public void testComplexType() {
            //System.out.println("["+produce("expression",
"true").toStringTree()+"]");
            assertPreorder("parse Line", DECLARATION_COMPL,
"(declaration_compl(Line)(foo))", produce("declaration_compl", "Line
foo{} "));
            assertPreorder("parse Circle", DECLARATION_COMPL,
"(declaration_compl(Force)(foo))", produce("declaration_compl", "Force
foo{}"));
            assertPreorder("parse Rect", DECLARATION_COMPL,
"(declaration_compl(Circle)(foo))", produce("declaration_compl",
"Circle foo{} "));
            assertPreorder("parse Force", DECLARATION_COMPL,
"(declaration_compl(Rect)(bar))", produce("declaration_compl", "Rect
bar{} "));
      }

      public void testComplexTypeParams() {
            //System.out.println("["+produce("expression",
"true").toStringTree()+"]");
            assertPreorder("parse Line", DECLARATION_COMPL,
"(declaration_compl(Line)(foo)(param)(4)(param2)(three)(orange)(3.42e10
```

```
))", produce("declaration_compl", "Line foo{ param=4; param2=\"three\";
orange=3.42e10;} "));
	}

}

/*
 * FunctionParseTest.java
 */

package test;

import static npsl.NPSLTokenTypes.*;

import java.io.StringReader;

import npsl.NPSLParser;
import npsl.NPSLLexer;
import org.norecess.antlr.ParserTestCase;
import org.norecess.antlr.TestableParser;

import antlr.TokenStream;
import antlr.TokenStreamException;


public class FunctionParseTest extends ParserTestCase {

	protected TokenStream makeLexer(String input) {
		return new NPSLLexer(new StringReader(input));
	}

	protected TestableParser makeParser(TokenStream lexer) {
		NPSLParser parser = new NPSLParser(lexer);

	parser.getASTFactory().setASTNodeClass("org.norecess.antlr.LineNu
mberAST");
		return parser;
	}

	public void testFunctionDef() {
		assertPreorder("empty funciton body", STATEMENT,
"(PROG(function(int)(test)(VAR_LIST)(FUNC_BODY)))", produce("program",
"function int test ( ) {}"));
		assertPreorder("with a arg list", STATEMENT,
"(PROG(function(int)(test)(VAR_LIST(int)(x)(float)(y))(FUNC_BODY)))",
produce("program", "function int test (int x, float y ) {}"));
		assertPreorder("with a arg list and body", STATEMENT,
"(PROG(function(int)(test)(VAR_LIST(int)(x)(float)(y))(FUNC_BODY(return
(+(x)(y))))))", produce("program", "function int test (int x, float y )
{return x+y;}"));
		assertPreorder("with a arg list and body", STATEMENT,
"(PROG(function(int)(addOneRef)(VAR_LIST(int)(x))(FUNC_BODY(=(x)(+(x)(1
)))))))", produce("program", "function int addOneRef (int x ) { x = x +
1; }"));
	}

	public void testFunctionCall() {
```

```
            assertPreorder("empty funciton body", STATEMENT,
"(PROG(FUNC_CALL(Z)(EXPR_LIST)))", produce("program", "Z();"));
            assertPreorder("with a arg list", STATEMENT,
"(PROG(FUNC_CALL(Y)(EXPR_LIST(5)(10))))", produce("program",
"Y(5,10);"));
      }

}

/*
 * IdenitfierLexTest.java
 */
package test;

import java.io.StringReader;

import org.junit.Test;
import org.norecess.antlr.LexerTestCase;

import antlr.TokenStream;
import antlr.TokenStreamException;
import antlr.TokenStreamRecognitionException;
import npsl.NPSLLexer;

import static npsl.NPSLTokenTypes.ID;

public class IdentifierLexTest extends LexerTestCase {

      protected TokenStream makeLexer(String input) {
              return new NPSLLexer(new StringReader(input));
            }

      public void testIdentifier() throws TokenStreamException {
            assertToken(ID, "ident", "ident");
      }

      public void testIdentifierWithUnderscore() throws
TokenStreamException {
            assertToken(ID, "ident_one", "ident_one");
      }

      public void testIdentifierWithNumber() throws
TokenStreamException {
            assertToken(ID, "ident1", "ident1");
      }

      public void testIdentifierStartsWithUnderscore() throws
TokenStreamException {
            assertToken(ID, "underScore_", "underScore_");
      }

      //todo need an assert fail?
      public void testIf() throws TokenStreamException {
            try {
            //    assertToken(ID, "if", "if");
                  }
            catch (     Exception e )
```

```
                {
                }
        }


        public void testBadID() throws TokenStreamException {
              try {
              assertFailedLex("@23423", "unexpected char: '@'");
              }
              catch (     TokenStreamRecognitionException e )
              {
              }
        }

}

/*
 * KeywordLexTest.java
 */

package test;

import static npsl.NPSLTokenTypes.*;

import java.io.StringReader;

import npsl.NPSLLexer;

import org.junit.Ignore;
import org.norecess.antlr.LexerTestCase;

import antlr.TokenStream;
import antlr.TokenStreamException;

public class KeywordLexTest extends LexerTestCase {

        protected TokenStream makeLexer(String input) {
                return new NPSLLexer(new StringReader(input));
              }

        public void testDefineKeywords() throws TokenStreamException {
              assertToken(LITERAL_function, "function", "function");
        }

        public void testBuiltInTypes() throws TokenStreamException {
              assertToken(LITERAL_Line, "Line", "Line");
              assertToken(LITERAL_Circle, "Circle", "Circle");
              assertToken(LITERAL_Rect, "Rect", "Rect");
              assertToken(LITERAL_Point, "Point", "Point");

              assertToken(LITERAL_Force, "Force", "Force");
        }

        public void testBasicTypes() throws TokenStreamException {
              assertToken(LITERAL_int, "int", "int");
              assertToken(LITERAL_float, "float", "float");
              assertToken(LITERAL_true, "true", "true");
```

```
            assertToken(LITERAL_false, "false", "false");
      }

      public void testFlowControl() throws TokenStreamException {
            assertToken(LITERAL_if, "if", "if");
            assertToken(LITERAL_else, "else", "else");
            assertToken(LITERAL_return, "return", "return");
      }

}

/*
 * RegisterComplexTest.java
 */

package test;
import java.util.HashMap;

import npsl.*;
import static npsl.NpslTypeConsts.*;
import static npsl.NpslUtil.*;

import org.junit.Before;
import org.junit.Test;

import junit.framework.TestCase;

public class RegisterComplexTest extends TestCase implements
NpslTypeConsts {

      @Before
      public void initialize() {


      }

      @Test
      public void testGetters() {

            HashMap<String, NpslDataType> nameToVals = new
HashMap<String, NpslDataType>();

            NpslInt first = new NpslInt(3); first.setName("firstInt");
            NpslBool second = new NpslBool(true);
second.setName("secondBool");
            NpslString third = new NpslString("hi");
third.setName("thirdString");
            NpslFloat fourth = new NpslFloat(4.23f);
third.setName("fourthFloat");

            String putString1 = "FIRST_INT";
            String putString2 = "SECOND_BOOL";
            String putString3 = "THIRD_STRING";
            String putString4 = "FOURTH_FLOAT";

            nameToVals.put(putString1, first);
            nameToVals.put(putString2, second);
```

```
            nameToVals.put(putString3, third);
            nameToVals.put(putString4, fourth);



            assertEquals( getInt( nameToVals, "FIRST_INT"), new
Integer(3) );
            assertEquals( getBoolean( nameToVals, "SECOND_BOOL"), new
Boolean(true) );
            assertEquals( getString( nameToVals, "THIRD_STRING"), "hi"
);
            assertEquals( getFloat( nameToVals, "FOURTH_FLOAT"), new
Float(4.23) );

            try  { getString( nameToVals, "FIRST_INT");fail();} catch
(NpslException e) {};
      }

      @Test
      public void testRegisterComplexCircle () {
            NpslInterpreter npsl = new NpslInterpreter();

            HashMap<String, NpslDataType> nameToVals = new
HashMap<String, NpslDataType>();
            NpslFloat mass = new NpslFloat(15f); mass.setName("mass");
            NpslInt radius = new NpslInt(10); radius.setName("radius");
            NpslInt xCoord = new NpslInt(15); radius.setName("xCoord");

            NpslInt yCoord = new NpslInt(20); radius.setName("yCoord");


            String putString1 = MASS;
            String putString2 = RADIUS;
            String putString3 = X_COORD;
            String putString4 = Y_COORD;

            nameToVals.put(putString1, mass);
            nameToVals.put(putString2, radius);
            nameToVals.put(putString3, xCoord);
            nameToVals.put(putString4, yCoord);

            npsl.registerComplexVar("mycircle", nameToVals,
CIRCLE_TYPE_NAME);
            npsl.what();
            npsl.getSymT().what();


      }


}

/*
 * StatementLexTest.java
 */
package test;
```

```
import static npsl.NPSLTokenTypes.*;

import java.io.StringReader;

import npsl.NPSLLexer;

import org.junit.Ignore;
import org.norecess.antlr.LexerTestCase;

import antlr.TokenStream;
import antlr.TokenStreamException;

public class StatementLexTest extends LexerTestCase {

	protected TokenStream makeLexer(String input) {
		return new NPSLLexer(new StringReader(input));
		}

	public void testDefineKeywords() throws TokenStreamException {
		assertToken(LITERAL_int, "int", "int foo;");
		assertToken(LITERAL_float, "float", "float foo;");
	}


}

/*
 * StatementParseTest.java
 */
package test;

import static npsl.NPSLTokenTypes.*;

import java.io.StringReader;

import org.norecess.antlr.ParserTestCase;
import org.norecess.antlr.TestableParser;
import npsl.NPSLLexer;
import npsl.NPSLParser;

import antlr.TokenStream;

public class StatementParseTest extends ParserTestCase{

	protected TokenStream makeLexer(String input) {
		return new NPSLLexer(new StringReader(input));
		}

	protected TestableParser makeParser(TokenStream lexer) {
		NPSLParser parser = new NPSLParser(lexer);

parser.getASTFactory().setASTNodeClass("org.norecess.antlr.LineNumberAS
T");
		return parser;
	}

	public void testAssign() {
```

```
            assertPreorder("parse foo=4", ASSIGN, "(=(moo)(4))",
produce("stmt", "moo=4;"));
            assertPreorder("parse if=4", ASSIGN,
"(=(orange)(+(5)(2)))", produce("assign_stmt", "orange=5+2;"));
      }

      public void testAssignFail() {
            assertFailedParse("without semi colon", "stmt" ,"cow=4");
            //assertFailedParse("using a keyword", "assign_stmt"
,"if=4;");
            assertFailedParse("using a paren", "stmt" ,"(=4;");
            assertFailedParse("store to a constant", "stmt"
,"true=4;");
      }

      public void testIf() {
            assertPreorder("normal if", LITERAL_if,
"(if(true)(=(oran)(true))(=(boo)(true)))", produce("stmt",
"if(true)oran=true;else boo=true;"));
            assertPreorder("if else ", LITERAL_if,
"(if(false)(=(man)(29))(=(bob)(3)))", produce("if_stmt", "if(false)
man=29; else bob=3;"));
            assertPreorder("nested if else ", LITERAL_if,
"(if(false)(if(true)(=(man)(29))(=(bob)(3))))", produce("if_stmt",
"if(false) if(true) man=29; else bob=3;"));
      }

      public void testDeclaration() {
            assertPreorder("parse foo=4", DECLARATION,
"(declaration(float)(moo))", produce("stmt", "float moo;"));
      }

      public void testComplexDeclaration() {
            assertPreorder("parse foo=4", DECLARATION_COMPL,
"(declaration_compl(Rect)(foo)(mass)(3.5)(free)(true)(radius)(3))",
produce("stmt", "Rect foo{ mass=3.5; free=true; radius=3; }"));
      }


}

/*
 * TestComment.java
 */

package test;

import java.io.StringReader;

import npsl.NPSLLexer;

import org.norecess.antlr.LexerTestCase;

import antlr.TokenStream;
import antlr.TokenStreamException;

//types to test
```

```
//this is a strange one, comments should just be skipped so EOF will be
//returned if a comment is processed correctly
import static npsl.NPSLTokenTypes.EOF;

public class TestComment extends LexerTestCase {

	protected TokenStream makeLexer(String input) {
		 return new NPSLLexer(new StringReader(input));
		}

	public void testOneLine() throws TokenStreamException {
		assertToken(EOF, null, "//comment");
	}

	public void testOneLineMulti() throws TokenStreamException {
		assertToken(EOF, null, "/* */");
	}

	public void testFewLineMulti() throws TokenStreamException {
		assertToken(EOF, null, "/* \n boo \n this \n is a comment
\n */");
	}

	public void testTricky() throws TokenStreamException {
		assertToken(EOF, null, "/***/");
		assertToken(EOF, null, " /* * */");
		assertToken(EOF, null, "/*/**/");
	}

}

/*
 * TestFloat.java
 */

package test;

//types to test
import static npsl.NPSLTokenTypes.FLOAT;

import java.io.StringReader;
import npsl.NPSLLexer;
import org.norecess.antlr.LexerTestCase;

import antlr.TokenStream;
import antlr.TokenStreamException;

public class TestFloat extends LexerTestCase {

	protected TokenStream makeLexer(String input) {
		 return new NPSLLexer(new StringReader(input));
		}

	public void testPoint() throws TokenStreamException {
		assertToken(FLOAT, "123.", "123.");
	}
```

```
	public void testFrac() throws TokenStreamException {
		assertToken(FLOAT, "123.45", "123.45");
	}

	public void testNegBoth() throws TokenStreamException {
		assertToken(FLOAT, "486.9845e-4", "486.9845e-4");
	}

	public void testFracOnly() throws TokenStreamException {
		assertToken(FLOAT, ".45", ".45");
	}

	public void testEPos() throws TokenStreamException {
		assertToken(FLOAT, "123.e10", "123.e10");
	}

	public void testENeg() throws TokenStreamException {
		assertToken(FLOAT, "123.e-10", "123.e-10");
	}

	public void testEOnly() throws TokenStreamException {
		assertToken(FLOAT, ".985e-10", ".985e-10");
	}

	public void testEWhiteSpace() throws TokenStreamException {
		assertToken(FLOAT, "123.e10",  "   123.e10   ");
	}


}

/*
 * TestInt.java
 */

package test;

import org.norecess.antlr.*;
import antlr.*;
import npsl.*;
import java.io.StringReader;

import static org.junit.Assert.*;

//types to test
import static npsl.NPSLTokenTypes.INT;

public class TestInt extends LexerTestCase {

	protected TokenStream makeLexer(String input) {
		  return new NPSLLexer(new StringReader(input));
		}

	public void testInteger() throws TokenStreamException {
		assertToken(INT, "123", "123");
	}
```

```
	public void testIntegerWhite() throws TokenStreamException {
		assertToken(INT, "9", "  9  ");
	}

}

/*
 * TestMainMath.java
 */

package test;

import java.io.StringReader;

import npsl.NPSLLexer;
import npsl.NPSLParser;
import npsl.NPSLWalker;
import npsl.NpslBool;
import npsl.NpslDataType;
import static npsl.NPSLWalkerTokenTypes.*;

import org.norecess.antlr.TreeParserTestCase;
import org.norecess.antlr.TestableParser;
import antlr.ASTFactory;

import antlr.CommonAST;
import antlr.TokenStream;
import antlr.TreeParser;
import antlr.collections.AST;

public class TestMainMath extends TreeParserTestCase {

	protected TokenStream makeLexer(String input) {
		return new NPSLLexer(new StringReader(input));
	}

	protected TestableParser makeParser(TokenStream lexer) {
		NPSLParser parser = new NPSLParser(lexer);
		parser.getASTFactory().setASTNodeClass(
				"org.norecess.antlr.LineNumberAST");
		return parser;
	}

	protected TreeParser makeTreeParser() {
		return new NPSLWalker();
	}

	public void testBasicProgram() {

		String program1 = "x=true;";
		String program2= "x = 5;\n" +
		 "if (x > 15)\n" +
		 "     y=6;\n" +
		 "else\n" +
		 "   y = 7;";

		AST atree = produce("program", program1);
```

```
            System.out.println(atree.toStringTree());
            System.out.println(atree.getType());
            System.out.println(atree.getNumberOfChildren());

            //AST ifTree = buildIfAST();
            //assertEquals(ifTree, build("expr", produce("program",
program1)));

            Object test = build("expr", produce("program", program1));

            NpslDataType dTest = (NpslDataType) test;
            assertTrue (dTest instanceof NpslBool);
            assertEquals( dTest.typename(), "bool");
            dTest.print();

            Object test2 = build("expr", produce("program", program2));
            NpslDataType dTest2 = (NpslDataType) test2;
            System.out.println(dTest2.toString());

      }

      //TODO add if and ast test
      public AST buildAssignAndIfAST () {
            ASTFactory factory = new ASTFactory();
              CommonAST r =
(CommonAST)factory.create(STATEMENT,"PROG");
              //CommonAST firstChild = (CommonAST) r.getFirstChild();
              //firstChild.setType(ASSIGN);

              r.addChild((CommonAST)factory.create(ASSIGN, "="));

              AST child = r.getFirstChild();
              child.addChild((CommonAST)factory.create(ID, "x"));
              child.addChild((CommonAST)factory.create(LITERAL_true,
"true"));
              System.out.println("Printing tomatch");
              System.out.println(r.toStringTree());
              System.out.println(r.getType());
              System.out.println(r.getNumberOfChildren());

              System.out.println("Now for the child");
              System.out.println(child.toStringTree());
              System.out.println(child.getType());
              System.out.println(child.getNumberOfChildren());

            return r;
      }

      public AST buildIfAST () {
            ASTFactory factory = new ASTFactory();
              CommonAST r =
(CommonAST)factory.create(STATEMENT,"PROG");
              //CommonAST firstChild = (CommonAST) r.getFirstChild();
              //firstChild.setType(ASSIGN);

              r.addChild((CommonAST)factory.create(ASSIGN, "="));
```

```
              AST child = r.getFirstChild();
              child.addChild((CommonAST)factory.create(ID, "x"));
              child.addChild((CommonAST)factory.create(LITERAL_true,
"true"));
              System.out.println("Printing tomatch");
              System.out.println(r.toStringTree());
              System.out.println(r.getType());
              System.out.println(r.getNumberOfChildren());

              System.out.println("Now for the child");
              System.out.println(child.toStringTree());
              System.out.println(child.getType());
              System.out.println(child.getNumberOfChildren());

            return r;
      }


}

/*
 * TestString.java
 */

package test;
import java.io.StringReader;

import npsl.NPSLLexer;

import org.norecess.antlr.LexerTestCase;

import antlr.TokenStream;
import antlr.TokenStreamException;

//types to test
import static npsl.NPSLTokenTypes.*;

public class TestString extends LexerTestCase {

      protected TokenStream makeLexer(String input) {
              return new NPSLLexer(new StringReader(input));
            }

      public void testBasicString() throws TokenStreamException {
             assertToken(STRING, "basic string", "\"basic string\"");
      }

      public void testStringWithNL() throws TokenStreamException {
             assertToken(STRING, "basic\n string", "\"basic\n
string\"");
      }

      public void testStringWithTab() throws TokenStreamException {
             assertToken(STRING, "basic\t string", "\"basic\t
string\"");
      }
```

```
	public void testStringWithB() throws TokenStreamException {
		assertToken(STRING, "basic\b string", "\"basic\b
string\"");
	}
	public void testStringWithF() throws TokenStreamException {
		assertToken(STRING, "basic\f string", "\"basic\f
string\"");
	}
	public void testStringWithQuote() throws TokenStreamException {
		assertToken(STRING, "basic\" string", "\"basic\\\"
string\"");
	}
	public void testStringWithQ() throws TokenStreamException {
		assertToken(STRING, "basic\' string", "\"basic\'
string\"");
	}
	public void testStringWithSlash() throws TokenStreamException {
		assertToken(STRING, "basic\\ string", "\"basic\\\\
string\"");
	}


}

/*
 * TypeFailuresTest.java
 */


package test;

import org.junit.Test;
import npsl.NpslMain;

import junit.framework.TestCase;

public class TypeFailuresTest extends TestCase{


	@Test
	public void testAssignments()
	{
		System.out.println("*********Test assignemnt********");
		NpslMain exec = new NpslMain();
		assertTrue (exec.runFile("tests/data/badAssign.npsl", true)
< 0);
	}

	/*
	 * This test is actually OK, as only a warning fires when you mix
types
	 * and the lhs gets overwritten with  the rhs type.  And floats
and ints
	 * are automatically cast.
	 */
	@Test
	public void testDeclaration()
```

```
	{
		System.out.println("*********Test declaration********");
		NpslMain exec = new NpslMain();
		assertTrue (exec.runFile("tests/data/badDeclare.npsl",
true) >=0);
	}

	@Test
	/*
	 * Undefined variables
	 */
	public void testFunciton()
	{
		System.out.println("*********Test functions*******");
		NpslMain exec = new NpslMain();
		assertTrue(exec.runFile("tests/data/badFunction.npsl",
true) <=0);
	}
}
```

Test Data files

assignment.npsl

```
x = 5;
y = 10.15 + 15;
z = x + y;
if ( x < y)
   cow = 1;
else
	cow = 2;
```

badAssign.npsl

```
y = "test";
x = 5;

z = y + x;
```

badDeclare.npsl

```
int x = 5;
string y;

y = 14;

int boo = 5.23e10;

float orange = 5;
```

badFunction.npls

```
function void addOneRef(int x) { return x*50; }

string y = addOneRef(z);
```

complex.npls

```
Circle mee {
	mass = 4.4;
```

```
      radius = 3;
      xCoord = 10;
      yCoord = 11;
      }

simulate;
```

declaration.npsl

```
int x;
float y = 11.412e10;
bool cow = true;
```

function.npsl

```
int x = 0;
function int addOneRef (int x ) { x = x + 1; }

print(x,y);
//test an internal function
println("applying function");
println(addOneRef);

//test a user defined function
x = addOneRef(x);
println(x);



//function
```